Software Manual

Renesas Flexible Software Package (FSP) v1.0.0

Software Manual

Renesas RX Family

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

Renesas Electronics
www.renesas.com

v1.0.0 Sept.29.2026

Table of Contents

Chapter 1 Introduction . . . 8
1.1 Overview . . . 8
1.2 Using this Manual . . . 8
1.3 Documentation Standard . . . 8
1.4 Introduction to FSP . . . 8
1.4.1 Purpose . . . 8
1.4.2 Quality . . . 9
1.4.3 Ease of Use . . . 9
1.4.4 Scalability . . . 9
1.4.5 Build Time Configurations . . . 9
1.4.6 e2 studio IDE . . . 9
Chapter 2 Starting Development . . . 10
2.1 Starting Development Introduction . . . 10
2.2 e2 studio User Guide . . . 11
2.2.1 What is e2 studio? . . . 11
2.2.2 e2 studio Prerequisites . . . 12
2.2.2.1 Obtaining an RX MCU Kit . . . 13
2.2.2.2 PC Requirements . . . 13
2.2.2.3 Installing e2 studio, platform installer and the FSP package . . . 13
2.2.2.4 Choosing a Toolchain . . . 13
2.2.2.5 Licensing . . . 13
2.2.3 What is a Project? . . . 13
2.2.4 Creating a Project . . . 15
2.2.4.1 Creating a New Project . . . 15
2.2.4.2 Selecting a Board and Toolchain . . . 16
2.2.4.3 Selecting a Project Template . . . 17
2.2.5 Configuring a Project . . . 20
2.2.5.1 Summary Tab . . . 20
2.2.5.2 Configuring the BSP . . . 21
2.2.5.3 Configuring Clocks . . . 22
2.2.5.4 Configuring Pins . . . 22
2.2.5.5 Configuring Interrupts from the Stacks Tab . . . 25
2.2.5.6 Viewing Event Links . . . 27
2.2.6 Adding Threads and Drivers . . . 27
2.2.6.1 Adding and Configuring HAL Drivers . . . 28
2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers . . . 30
2.2.6.3 Configuring Threads . . . 32
2.2.7 Reviewing and Adding Components . . . 34
2.2.8 Writing the Application . . . 35
2.2.8.1 Coding Features . . . 35
2.2.8.2 HAL Modules in FSP: A Practical Description . . . 41
2.2.8.3 RTOS-Independent Applications . . . 42
2.2.8.4 RTOS Applications . . . 43
2.2.9 Debugging the Project . . . 45
2.2.10 Modifying Toolchain Settings . . . 46
2.2.11 Importing an Existing Project into e2 studio . . . 47
2.3 Tutorial: Your First RX MCU Project - Blinky . . . 51
2.3.1 Tutorial Blinky . . . 51
2.3.2 What Does Blinky Do? . . . 52
2.3.3 Prerequisites . . . 52
2.3.4 Create a New Project for Blinky . . . 52

2.3.4.1 Details about the Blinky Configuration . . . 57
2.3.4.2 Configuring the Blinky Clocks . . . 57
2.3.4.3 Configuring the Blinky Pins . . . 57
2.3.4.4 Configuring the Parameters for Blinky Components . . . 57
2.3.4.5 Where is main()? . . . 57
2.3.4.6 Blinky Example Code . . . 57
2.3.5 Build the Blinky Project . . . 57
2.3.6 Debug the Blinky Project . . . 58
2.3.6.1 Debug prerequisites . . . 58
2.3.6.2 Debug steps . . . 59
2.3.6.3 Details about the Debug Process . . . 60
2.3.7 Run the Blinky Project . . . 61
2.4 Tutorial: Using HAL Drivers - Programming the WDT . . . 61
2.4.1 Application WDT . . . 61
2.4.2 Creating a WDT Application Using the RX MCU FSP and e² studio . . . 61
2.4.2.1 Using FSP and e² studio . . . 61
2.4.2.2 The WDT Application . . . 61
2.4.2.3 WDT Application flow . . . 62
2.4.3 Creating the Project with e² studio . . . 62
2.4.4 Configuring the Project with e² studio . . . 67
2.4.4.1 BSP Tab . . . 69
2.4.4.2 Clocks Tab . . . 69
2.4.4.3 Interrupts Tab . . . 70
2.4.4.4 Event Links Tab . . . 70
2.4.4.5 Pins Tab . . . 70
2.4.4.6 Stacks Tab . . . 70
2.4.4.7 Components Tab . . . 73
2.4.5 WDT Generated Project Files . . . 74
2.4.5.1 WDT hal_data.h . . . 76
2.4.5.2 WDT hal_data.c . . . 77
2.4.5.3 WDT main.c . . . 78
2.4.5.4 WDT hal_entry.c . . . 78
2.4.6 Building and Testing the Project . . . 81
2.5 RXSC User Guide for IAR . . . 83
2.5.1 What is RXSC? . . . 83
2.5.2 Using RX Smart Configurator with IAR EWRX . . . 83
2.5.2.1 Prerequisites . . . 83
2.5.2.2 Create new RX FSP project . . . 83
Chapter 3 FSP Architecture . . . 86
3.1 FSP Architecture Overview . . . 86
3.1.1 C99 Use . . . 86
3.1.2 Doxygen . . . 86
3.1.3 Weak Symbols . . . 86
3.1.4 Memory Allocation . . . 86
Chapter 4 Reference Materials . . . 87
4.1 RX Caches . . . 87
4.1.1 RX Cache Features . . . 87
4.1.2 Cache Functions and Macros . . . 87
4.1.3 References . . . 88
4.2 Linker Script Generation Guide . . . 89
4.2.1 FSP Linker Scripts . . . 89
4.2.1.1 Section Model (high level) . . . 89
4.2.1.2 Interacting with the Flow . . . 90
4.2.1.3 **Output** Section Translation (for GCC for RX, CC-RX, IAR) . . . 92
4.2.1.4 Behavior differences worth noting . . . 93

4.2.1.5 Examples . . . 93
4.2.2 FAQ . . . 93
4.2.3 Summary . . . 94
4.2.4 Appendix: Linker Input Sections . . . 94
4.2.4.1 Memory Class Dictionaries . . . 94
4.2.4.2 RAM-Resident Sections . . . 94
4.2.4.3 ROM-Resident Sections . . . 95
4.2.4.4 Summary . . . 96
Chapter 5 API Reference . . . 97
5.1 BSP . . . 98
5.1.1 Common Error Codes . . . 100
5.1.2 MCU Board Support Package . . . 112
5.1.2.1 RX74M . . . 167
5.1.2.2 RX74N . . . 176
5.1.3 BSP I/O access . . . 185
5.1.4 BSP SDRAM support . . . 200
5.2 Modules . . . 202
5.2.1 Analog . . . 204
5.2.1.1 ADC (r_adc_b) . . . 205
5.2.1.2 DAC_B (r_dac_b) . . . 243
5.2.1.3 Delta-Sigma Modulator Interface (r_dsmif) . . . 249
5.2.2 AI . . . 281
5.2.3 Audio . . . 282
5.2.3.1 Pulse Density Modulation Interface (r_pdm) . . . 282
5.2.4 Bootloader . . . 303
5.2.4.1 MCUboot Port (rm_mcuboot_port) . . . 304
5.2.5 CapTouch . . . 314
5.2.6 Connectivity . . . 314
5.2.6.1 CAN FD (r_canfd) . . . 316
5.2.6.2 CAN FDG (r_canfdg) . . . 343
5.2.6.3 I3C (r_i3c) . . . 381
5.2.6.4 I2C Master (r_iic_b_master) . . . 422
5.2.6.5 I2C Slave (r_iic_b_slave) . . . 435
5.2.6.6 I2C Master (r_sci_b_i2c) . . . 447
5.2.6.7 SPI (r_sci_b_spi) . . . 459
5.2.6.8 UART (r_sci_b_uart) . . . 471
5.2.6.9 SPI (r_spi_b) . . . 491
5.2.6.10 I2S (r_ssi) . . . 509
5.2.6.11 USB (r_usb_basic) . . . 524
5.2.6.12 USB Composite (r_usb_composite) . . . 557
5.2.6.13 USB Host Communications Device Class (r_usb_hcdc) . . . 586
5.2.6.14 USB Host Human Interface Device Class Driver (r_usb_hhid) . . . 598
5.2.6.15 USB Host Mass Storage Class Driver (r_usb_hmsc) . . . 609
5.2.6.16 USB Host Vendor class (r_usb_hvnd) . . . 616
5.2.6.17 USB PCDC (r_usb_pcdc) . . . 629
5.2.6.18 USB PHID (r_usb_phid) . . . 639
5.2.6.19 USB PMSC (r_usb_pmsc) . . . 651
5.2.6.20 USB Peripheral Printer (r_usb_pprn) . . . 657
5.2.6.21 USB Peripheral Vendor class (r_usb_pvnd) . . . 665
5.2.6.22 USB_PCDC Communication Device (rm_comms_usb_pcdc) . . . 677
5.2.6.23 Azure RTOS USBX Port (rm_usbx_port) . . . 685
5.2.7 DSP . . . 701
5.2.8 Graphics . . . 701
5.2.8.1 Parallel Data Capture (r_pdc) . . . 702
5.2.8.2 SEGGER emWin RX Serial Port (rm_emwin_rx_port) . . . 712

5.2.9 Input . . . 721
5.2.9.1 External IRQ (r_icu) . . . 722
5.2.10 Monitoring . . . 729
5.2.10.1 Clock Accuracy Circuit (r_cac) . . . 730
5.2.10.2 CRC (r_crc) . . . 737
5.2.10.3 Data Operation Circuit (r_doc) . . . 743
5.2.10.4 Independent Watchdog (r_iwdt) . . . 749
5.2.10.5 Programmable Voltage Detection (r_pvd) . . . 759
5.2.10.6 Watchdog (r_wdt) . . . 766
5.2.11 Motor . . . 777
5.2.11.1 Phase Counting Interface (r_pcif) . . . 778
5.2.11.2 Motor Current Controller (rm_motor_current) . . . 802
5.2.11.3 ADC and PWM Modulation (rm_motor_driver) . . . 813
5.2.11.4 Motor Encoder Vector Control (rm_motor_encoder) . . . 823
5.2.11.5 Motor Angle (rm_motor_estimate) . . . 838
5.2.11.6 Motor Inertia estimate (rm_motor_inertia_estimate) . . . 848
5.2.11.7 Motor Position Controller (rm_motor_position) . . . 857
5.2.11.8 Motor return origin (rm_motor_return_origin) . . . 868
5.2.11.9 Motor Angle (rm_motor_sense_encoder) . . . 877
5.2.11.10 Motor Sensorless Vector Control (rm_motor_sensorless) . . . 889
5.2.11.11 Motor Speed Controller (rm_motor_speed) . . . 900
5.2.12 Networking . . . 912
5.2.12.1 Ethercat (r_ethercat_phy) . . . 914
5.2.12.2 Ethernet Generic PTP Timer (r_gptp) . . . 924
5.2.12.3 Layer3 Ethernet Switch (r_layer3_switch) . . . 934
5.2.12.4 Ethernet (r_rmac) . . . 1012
5.2.12.5 Ethernet (r_rmac_phy) . . . 1028
5.2.12.6 AWS PKCS11 PAL on LittleFS (rm_aws_pkcs11_pal_littlefs) . . . 1044
5.2.12.7 EtherCAT SSC Port (rm_ethercat_ssc_port) . . . 1044
5.2.12.8 AWS coreHTTP . . . 1051
5.2.12.9 AWS MQTT . . . 1056
5.2.12.10 FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp) . . . 1063
5.2.12.11 gPTP Ethernet Switch Module(rm_gptp_eswm) . . . 1071
5.2.12.12 gPTP Baremetal Porting Layer (rm_gptp_sys_baremetal) . . . 1077
5.2.12.13 gPTP FreeRTOS Porting Layer (rm_gptp_sys_freertos) . . . 1077
5.2.12.14 lwIP HTTP Server . . . 1077
5.2.12.15 lwIP Iperf . . . 1087
5.2.12.16 lwIP MQTT . . . 1091
5.2.12.17 lwIP NETBIOSNS . . . 1098
5.2.12.18 lwIP SMTP . . . 1100
5.2.12.19 lwIP SNTP . . . 1106
5.2.12.20 lwIP Ethernet Driver (rm_lwip_ether) . . . 1110
5.2.12.21 lwIP Baremetal Porting Layer (rm_lwip_sys_baremetal) . . . 1114
5.2.12.22 lwIP FreeRTOS Porting Layer (rm_lwip_sys_freertos) . . . 1115
5.2.12.23 NetX Duo Ethernet Driver (rm_netxduo_ether) . . . 1115
5.2.13 Power . . . 1119
5.2.13.1 Low Power Modes (r_lpm) . . . 1119
5.2.14 RTOS . . . 1130
5.2.14.1 FreeRTOS Port (rm_freertos_port) . . . 1131
5.2.14.2 Azure RTOS ThreadX Port (rm_threadx_port) . . . 1157
5.2.15 Security . . . 1165
5.2.15.1 Secure Key Injection (r_rsip_key_injection) . . . 1166
5.2.15.2 Renesas Secure IP (r_rsip_protected) . . . 1199
5.2.15.3 Azure RTOS NetX Crypto HW Acceleration (rm_netx_secure_crypto) . . . 1359
5.2.15.4 Mbed Crypto H/W Acceleration (rm_psa_crypto) . . . 1398

5.2.16 Sensor . . . 1483
5.2.17 Storage . . . 1483
5.2.17.1 MRAM (r_mram) . . . 1485
5.2.17.2 SD/MMC (r_sdhi) . . . 1499
5.2.17.3 XSPI (r_xspi_controller) . . . 1513
5.2.17.4 XSPI (r_xspi_target) . . . 1563
5.2.17.5 Block Media RAM (rm_block_media_ram) . . . 1577
5.2.17.6 Block Media SD/MMC (rm_block_media_sdmmc) . . . 1584
5.2.17.7 Block Media SPI Flash (rm_block_media_spi) . . . 1591
5.2.17.8 Block Media USB (rm_block_media_usb) . . . 1601
5.2.17.9 Block Media Custom Implementation (rm_block_media_user) . . . 1608
5.2.17.10 FreeRTOS+FAT Port for RX (rm_freertos_plus_fat) . . . 1610
5.2.17.11 LittleFS on Flash (rm_littlefs_flash) . . . 1623
5.2.17.12 LittleFS on SPI Flash (rm_littlefs_spi_flash) . . . 1630
5.2.18 System . . . 1638
5.2.18.1 Clock Generation Circuit (r_cgc) . . . 1638
5.2.18.2 Event Link Controller (r_elc) . . . 1662
5.2.18.3 I/O Ports (r_ioport) . . . 1673
5.2.19 Timers . . . 1707
5.2.19.1 Compare Match Timer (r_cmt) . . . 1708
5.2.19.2 Compare Match Timer W (r_cmtw) . . . 1720
5.2.19.3 Timer, General PWM (r_gpt) . . . 1740
5.2.19.4 Three-Phase PWM (r_gpt_three_phase) . . . 1790
5.2.19.5 Port Output Enable for GPT (r_poeg) . . . 1799
5.2.19.6 Realtime Clock (r_rtc) . . . 1805
5.2.19.7 Timer, 8-Bit Timer (r_tmr) . . . 1822
5.2.20 Transfer . . . 1841
5.2.20.1 DMAC Transfer (r_dmac) . . . 1841
5.2.20.2 Transfer (r_dtc) . . . 1856
5.2.20.3 EXDMAC Transfer (r_exdmac) . . . 1871
5.2.21 ProtectZone . . . 1886
5.3 Interfaces . . . 1886
5.3.1 Analog . . . 1888
5.3.1.1 ADC Interface . . . 1888
5.3.1.2 DAC Interface . . . 1913
5.3.2 AI . . . 1917
5.3.3 Audio . . . 1917
5.3.3.1 PDM Interface . . . 1917
5.3.4 CapTouch . . . 1927
5.3.5 Connectivity . . . 1928
5.3.5.1 CAN Interface . . . 1929
5.3.5.2 I2C Master Interface . . . 1939
5.3.5.3 I2C Slave Interface . . . 1947
5.3.5.4 I2S Interface . . . 1955
5.3.5.5 I3C Interface . . . 1966
5.3.5.6 PCIF Interface . . . 1983
5.3.5.7 SPI Interface . . . 1995
5.3.5.8 UART Interface . . . 2006
5.3.5.9 USB Interface . . . 2017
5.3.5.10 USB HCDC Interface . . . 2045
5.3.5.11 USB HHID Interface . . . 2051
5.3.5.12 USB HMSC Interface . . . 2053
5.3.5.13 USB PCDC Interface . . . 2058
5.3.5.14 USB PHID Interface . . . 2060
5.3.5.15 USB PMSC Interface . . . 2060

5.3.5.16 USB PPRN Interface . . . 2061
5.3.5.17 Communications Middleware Interface . . . 2062
5.3.6 DSP . . . 2066
5.3.7 Graphics . . . 2066
5.3.7.1 CAPTURE Interface . . . 2066
5.3.8 Input . . . 2072
5.3.8.1 External IRQ Interface . . . 2072
5.3.9 Monitoring . . . 2079
5.3.9.1 CAC Interface . . . 2080
5.3.9.2 CRC Interface . . . 2090
5.3.9.3 DOC Interface . . . 2095
5.3.9.4 Voltage Detection Interface . . . 2102
5.3.9.5 WDT Interface . . . 2114
5.3.10 Motor . . . 2123
5.3.10.1 Motor angle Interface . . . 2124
5.3.10.2 Motor Interface . . . 2132
5.3.10.3 Motor current Interface . . . 2140
5.3.10.4 Motor driver Interface . . . 2148
5.3.10.5 Motor Inertia Estimate Interface . . . 2154
5.3.10.6 Motor position Interface . . . 2159
5.3.10.7 Motor Return Origin Function Interface . . . 2165
5.3.10.8 Motor speed Interface . . . 2170
5.3.11 Networking . . . 2176
5.3.11.1 Ethernet Interface . . . 2177
5.3.11.2 Ethernet PHY Interface . . . 2187
5.3.11.3 Ethernet Switch Interface . . . 2196
5.3.11.4 GPTP Interface . . . 2200
5.3.11.5 EtherCAT SSC Interface . . . 2205
5.3.12 Power . . . 2209
5.3.12.1 Low Power Modes Interface . . . 2210
5.3.13 Security . . . 2230
5.3.13.1 RSIP key injection Interface . . . 2231
5.3.13.2 RSIP Interface . . . 2259
5.3.14 Sensor . . . 2316
5.3.15 Storage . . . 2316
5.3.15.1 Flash Interface . . . 2317
5.3.15.2 SD/MMC Interface . . . 2333
5.3.15.3 SPI Flash Interface . . . 2349
5.3.15.4 Block Media Interface . . . 2362
5.3.15.5 FreeRTOS+FAT Port Interface . . . 2370
5.3.15.6 LittleFS Interface . . . 2376
5.3.16 System . . . 2378
5.3.16.1 CGC Interface . . . 2379
5.3.16.2 ELC Interface . . . 2391
5.3.16.3 I/O Port Interface . . . 2395
5.3.17 Timers . . . 2402
5.3.17.1 POEG Interface . . . 2403
5.3.17.2 RTC Interface . . . 2412
5.3.17.3 Three-Phase Interface . . . 2427
5.3.17.4 Timer Interface . . . 2432
5.3.18 Transfer . . . 2445
5.3.18.1 Transfer Interface . . . 2446
Chapter 6 Copyright . . . 2462

Chapter 1 Introduction

1.1 Overview

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

1.2 Using this Manual

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

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

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

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

1.3 Documentation Standard

Each Modules section user guide outlines the following:

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

Each Interfaces section user guide outlines the following:

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

1.4 Introduction to FSP

1.4.1 Purpose

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

1.4.2 Quality

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

1.4.3 Ease of Use

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

1.4.4 Scalability

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

1.4.5 Build Time Configurations

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

1.4.6 e2 studio IDE

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

Chapter 2 Starting Development

2.1 Starting Development Introduction

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

1. Read the section What is e2 studio?, up to but not including e2 studio Prerequisites. This will provide a description of the various windows and views to use e2 studio to create a project, add modules and threads, configure module properties, add code, and debug a project. It also describes how to use key coding 'accelerators' like Developer Assist (to drag and drop parameter populated API function calls right into your code), a context aware Autocomplete (to easily find and select from suggested enumerations, functions, types, and many other coding elements), and many other similar productivity enhancers.
2. Read the FSP Architecture, fsp-modules and fsp-stacks sections. These provide the basic background on how FSP modules and stacks are used to construct your application. Understanding their definitions and the theory behind how they combine will make it easier to develop with FSP.
3. Read a few Modules sections to see how to use API function calls, structures, enumerations, types and callbacks. These module guides provide the information you will use to implement your project code.
4. After you have a Kit and you have downloaded and installed e2 studio and FSP, you can build and debug a simple project to test your installation, tool flow, and the kit. (If you do not have a Kit or have not yet installed the development software, use the links included in the e2 studio Prerequisites for more information.) The simple Tutorial: Your First RX MCU Project - Blinky will Blink an LED on and off. Follow the instructions for importing and running this project in section Create a New Project for Blinky. It will use some of the key steps for managing projects within e2 studio and is a good way to learn the basics.
5. Once you have successfully run Blinky you have a good starting point for using FSP for more complex projects. The Using HAL Drivers Tutorial, available at Tutorial: Using HAL Drivers - Programming the WDT, shows how to create a project from scratch, using FSP API functions. Do this next.
6. The balance of the FSP Architecture sections (that is, those not called out in step 2 above) contain additional reference material that may be helpful in the future. Scan them so you know what they contain, in case you need them.
7. The balance of the e2 studio User Guide, starting with the What is a Project? section up to, but not including, Writing the Application section, provides a detailed description of each of the key steps, windows, and entries used to create, manage, configure, build and debug a project. Much of this may be familiar after running through the tutorials and Quick Labs. However, it is important to have a good grasp of what each of the configuration tabs are used for as that is where the bulk of the project preparation work takes place prior to writing code. Skim over this section as it may help with any questions in the future.
8. Read the Writing the Application section to get a short introduction to the steps used when creating application code with FSP. It covers both RTOS-independent and RTOS-dependent applications. It also includes a short description for several of the code accelerators you

should be familiar with by now.
9. Scan the Debugging the Project section to see the steps required to download and start a debug session.
10. Explore the additional material available on the following web pages and bookmark the resources that look most valuable to you:
    a. RX Landing Page: https://www.renesas.com/rx
    b. Online Technical Support: https://www.renesas.com/us/en/support/contact.html

2.2 e2 studio User Guide

2.2.1 What is e2 studio?

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

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

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

Figure 1: e2 studio Splash Screen

e2 studio organizes project work based on Perspectives, Views, Windows, Panes, and Pages (sometimes called Tabs). A window is a section of the e2 studio GUI that presents information on a key topic. Windows often use tabs to select sub-topics. For example, an editor window might have a tab available for each open file, so it is easy to switch back and forth between them. A window Pane

is a section of a window. Within a window, multiple Panes can be opened and viewed simultaneously, as opposed to a tabbed window, where only individual content is displayed. A memory-display Window, for example, might have multiple Panes that allow the data to be displayed in different formats, simultaneously. A Perspective is a collection of Views and Windows typical for a specific stage of development. The default perspectives are a C/C++ Perspective, an FSP Configuration Perspective and a Debug Perspective. These provide specific Views, Windows, Tabs, and Panes tailored for the common tasks needed during the specific development stage.

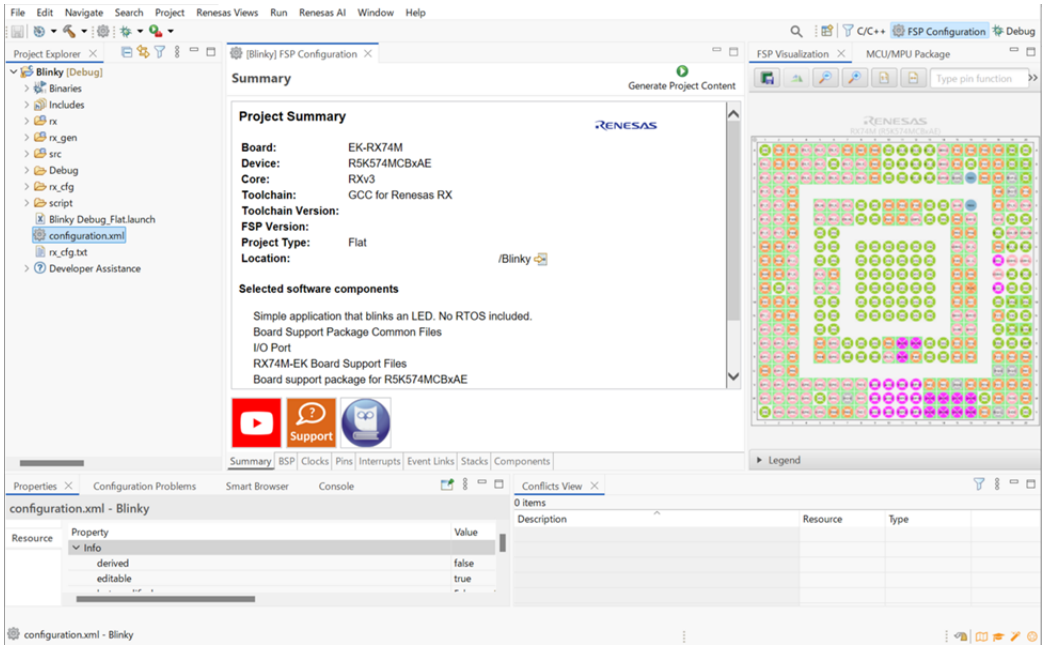


Figure 2: Default Perspective

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

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

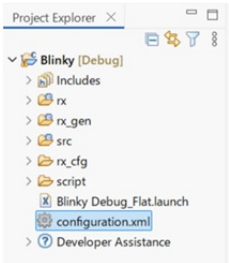


Figure 3: Project Explorer Window showing generated folders and configuration.xml file

2.2.2 e2 studio Prerequisites

2.2.2.1 Obtaining an RX MCU Kit

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

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

2.2.2.2 PC Requirements

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

- System: x64 based processor, 2 GHz or faster (with multicore CPUs)
    - Windows 11
    - Windows 10 (64-bit version)
- Memory: Recommend 8 GB or more. At least 4 GB.
- Capacity of hard disk: At least 2 GB of free space.

2.2.2.3 Installing e2 studio, platform installer and the FSP package

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

2.2.2.4 Choosing a Toolchain

e2 studio can work with several toolchains and toolchain versions

such as the GCC for Renesas RX compiler and Renesas CCRX. A version of the Renesas compiler is included in the e2 studio installer and has been verified to run with the FSP version.

2.2.2.5 Licensing

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

2.2.3 What is a Project?

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

1. Creating a Project
2. Configuring a Project

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

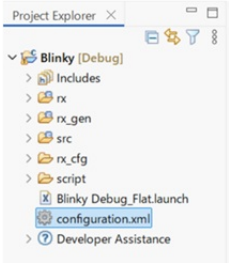


Figure 4: e2 studio Project Configuration file

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

Figure 5: e2 studio FSP Configuration Perspective

Note

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

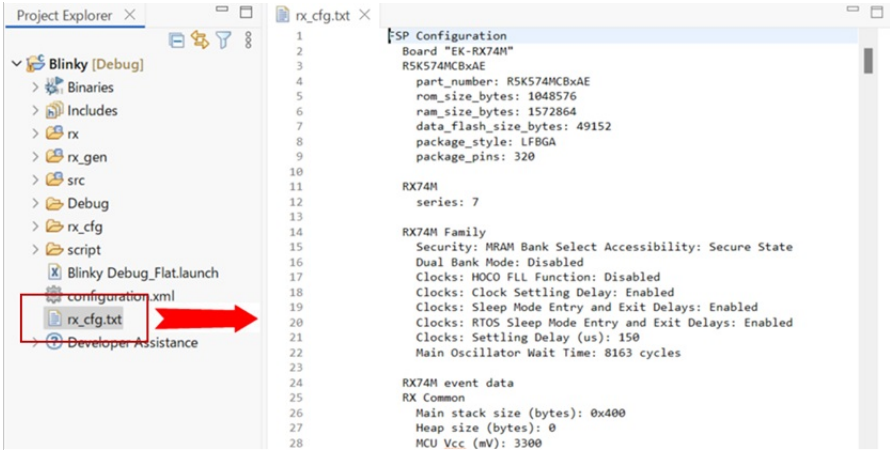


Figure 6: RX Project Report

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

Note

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

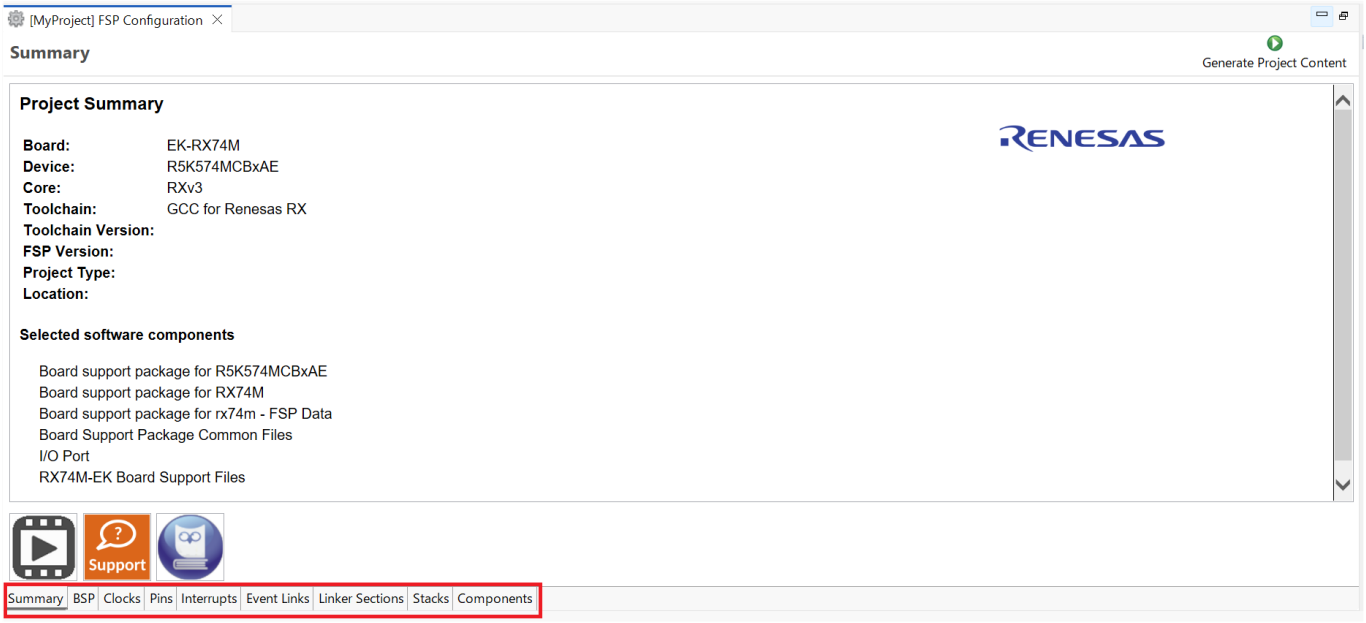


Figure 7: RX Project Editor tabs

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

2.2.4 Creating a Project

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

2.2.4.1 Creating a New Project

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

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

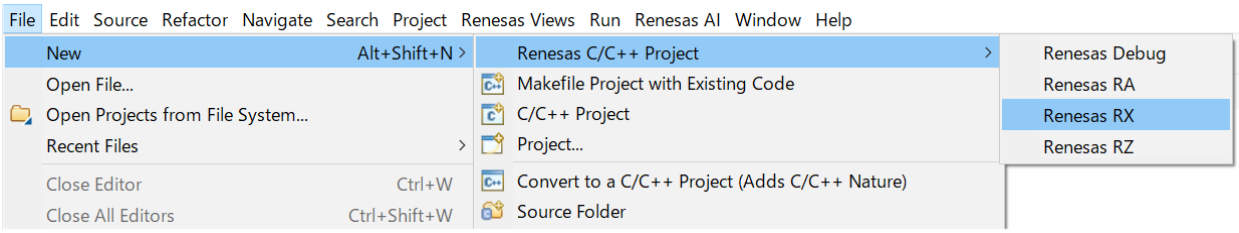


Figure 8: New RX MCU Project

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

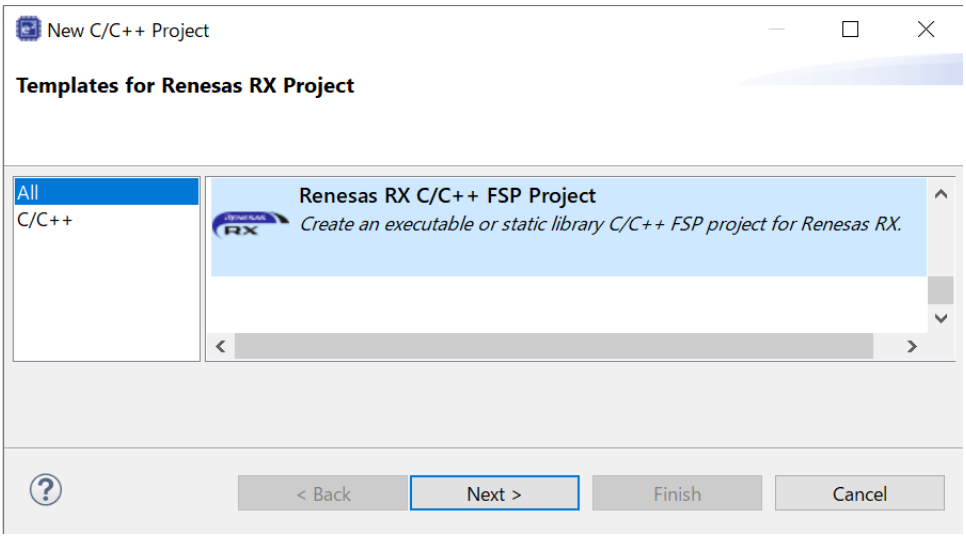


Figure 9: New Project Templates

2. Select a project name and location.

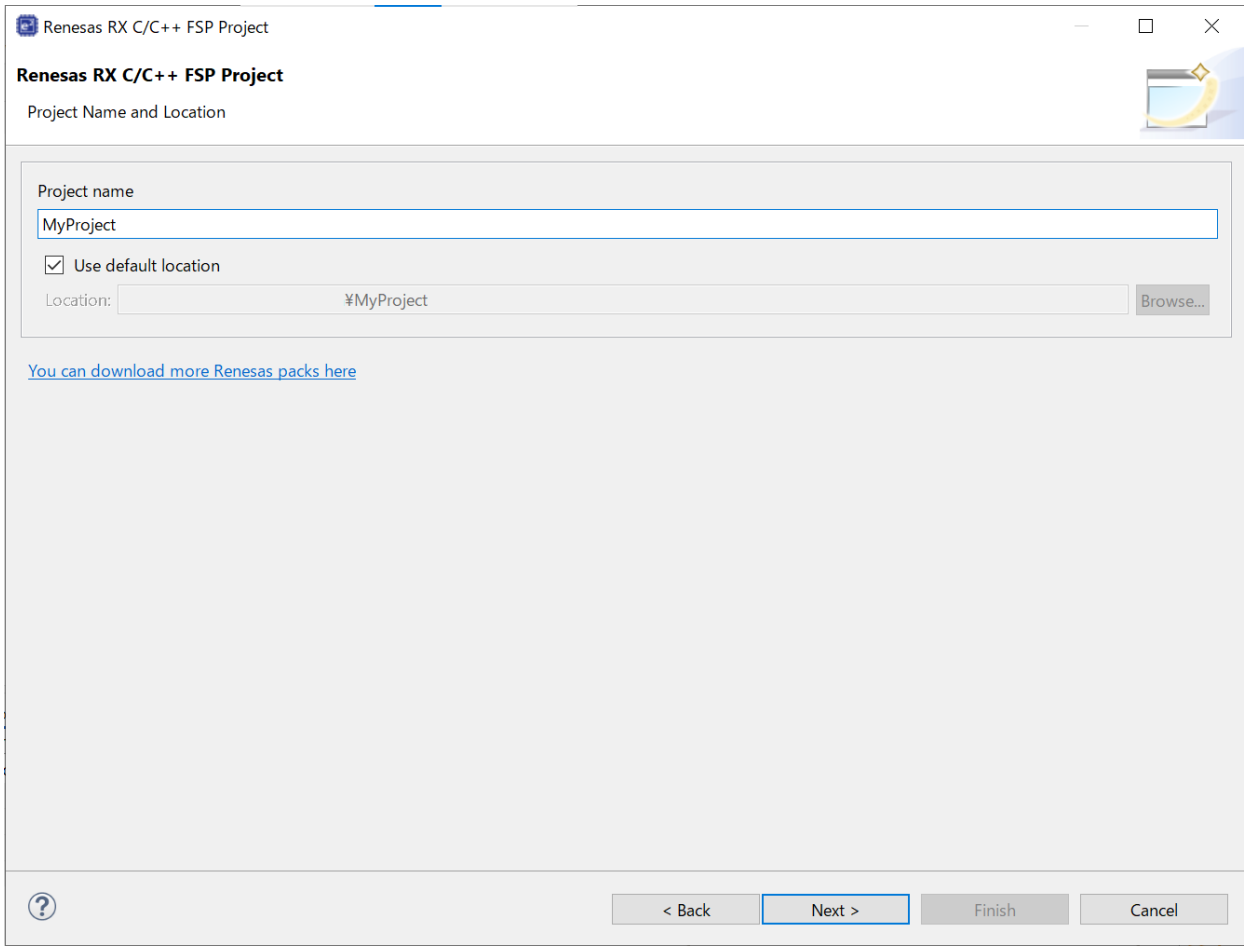


Figure 10: RX MCU Project Generator (Screen 1)

3. Click **Next**.

2.2.4.2 Selecting a Board and Toolchain

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

1. Select the **FSP version**.
2. Select the **Board** for your application. You can select an existing RX MCU Evaluation Kit or select **Custom User Board** for any of the RX MCU devices with your own BSP definition.
3. Select the **Device**. The **Device** is automatically populated based on the **Board** selection. Only change the **Device** when using the **Custom User Board (Any Device)** board selection.
4. To add threads, select **RTOS**, or **No RTOS** if an RTOS is not being used.
5. The **Toolchain** selection defaults to **GCC for Renesas RX**.
6. Select the **Toolchain version**. This should default to the installed toolchain version.
7. Select the **Debugger**. The E2OB Debugger is preselected.

8. Click **Next**.

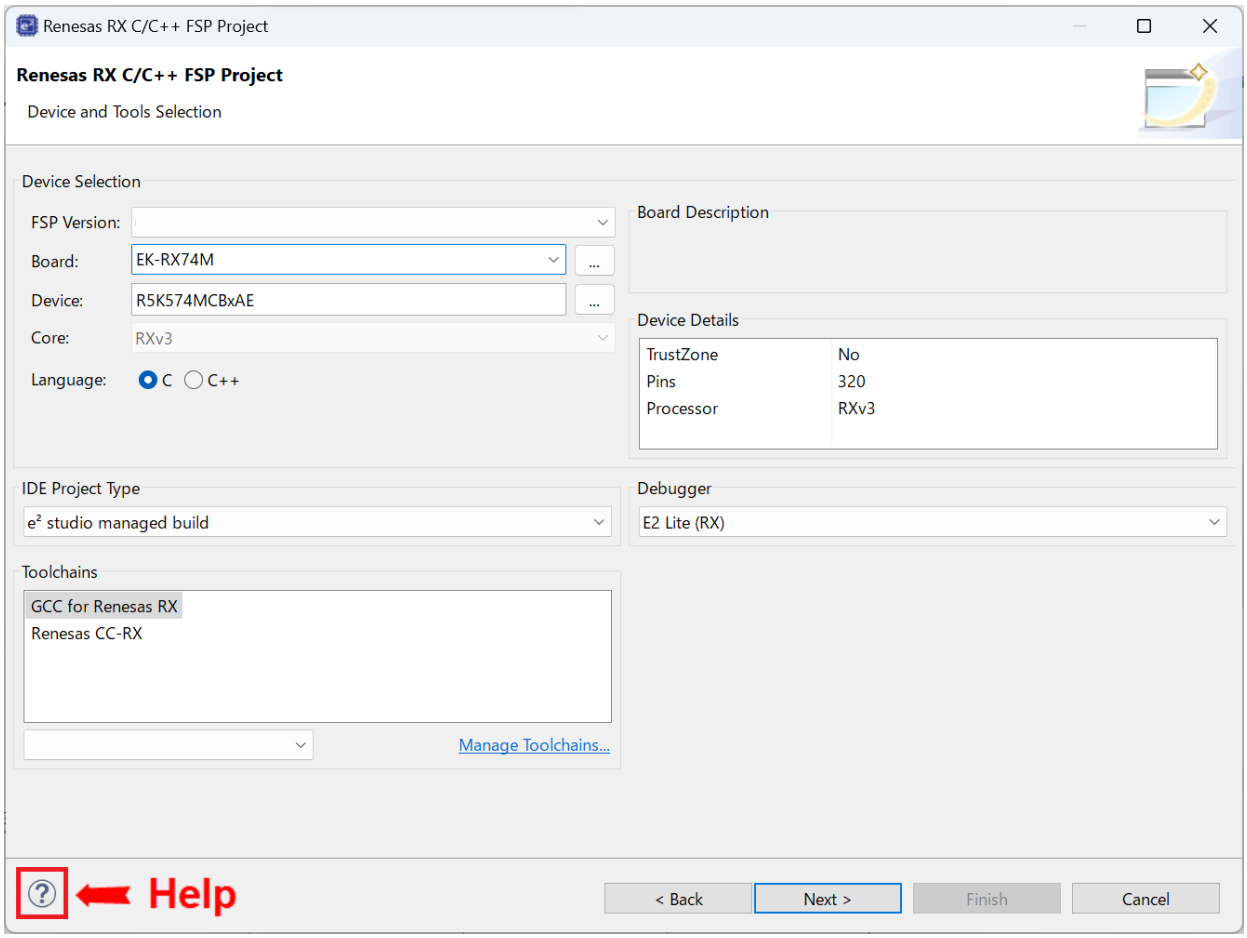


Figure 11: RX MCU Project Generator (Screen 2)

Note

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

2.2.4.3 Selecting a Project Template

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

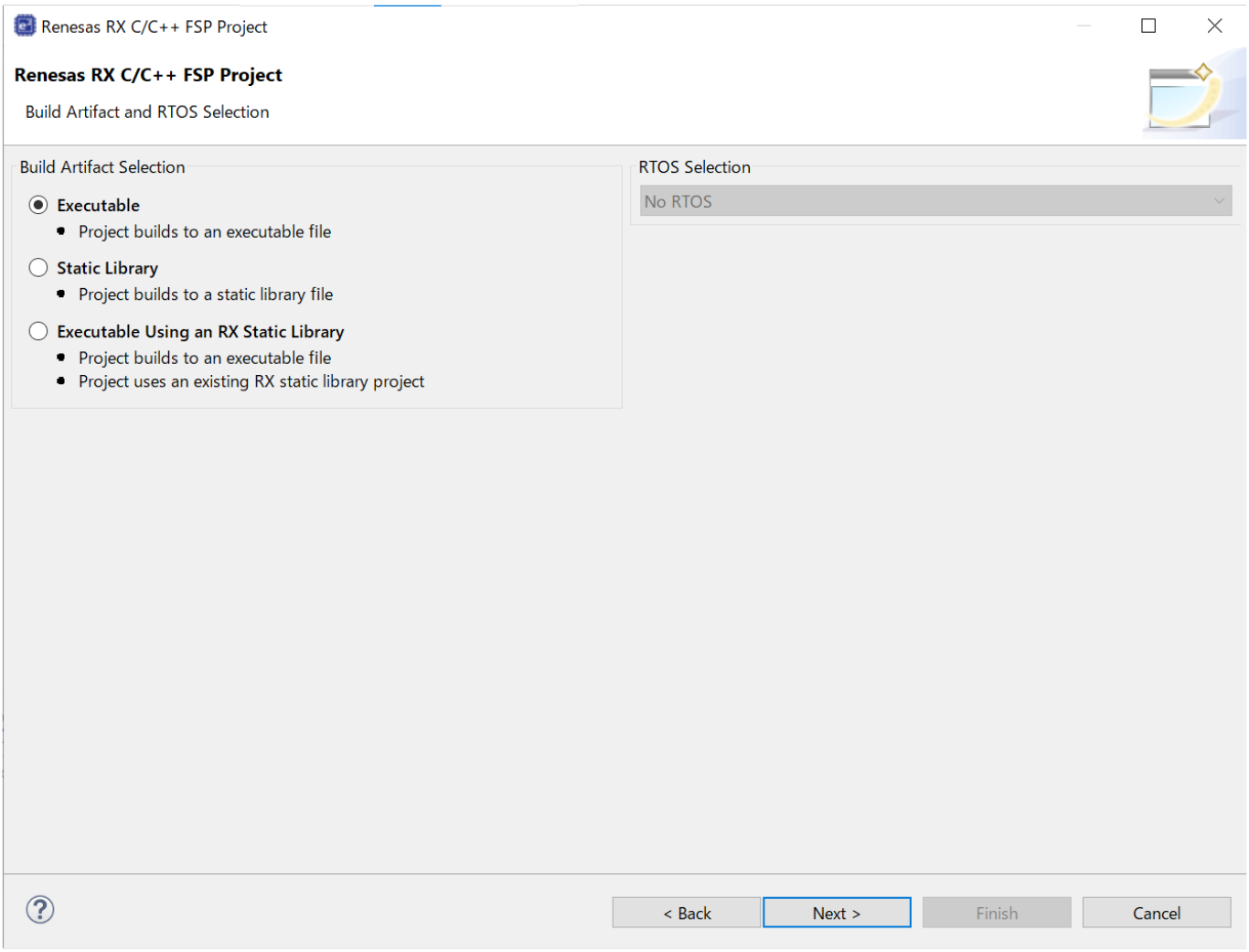


Figure 12: RX MCU Project Generator (Screen 3)

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

Note

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

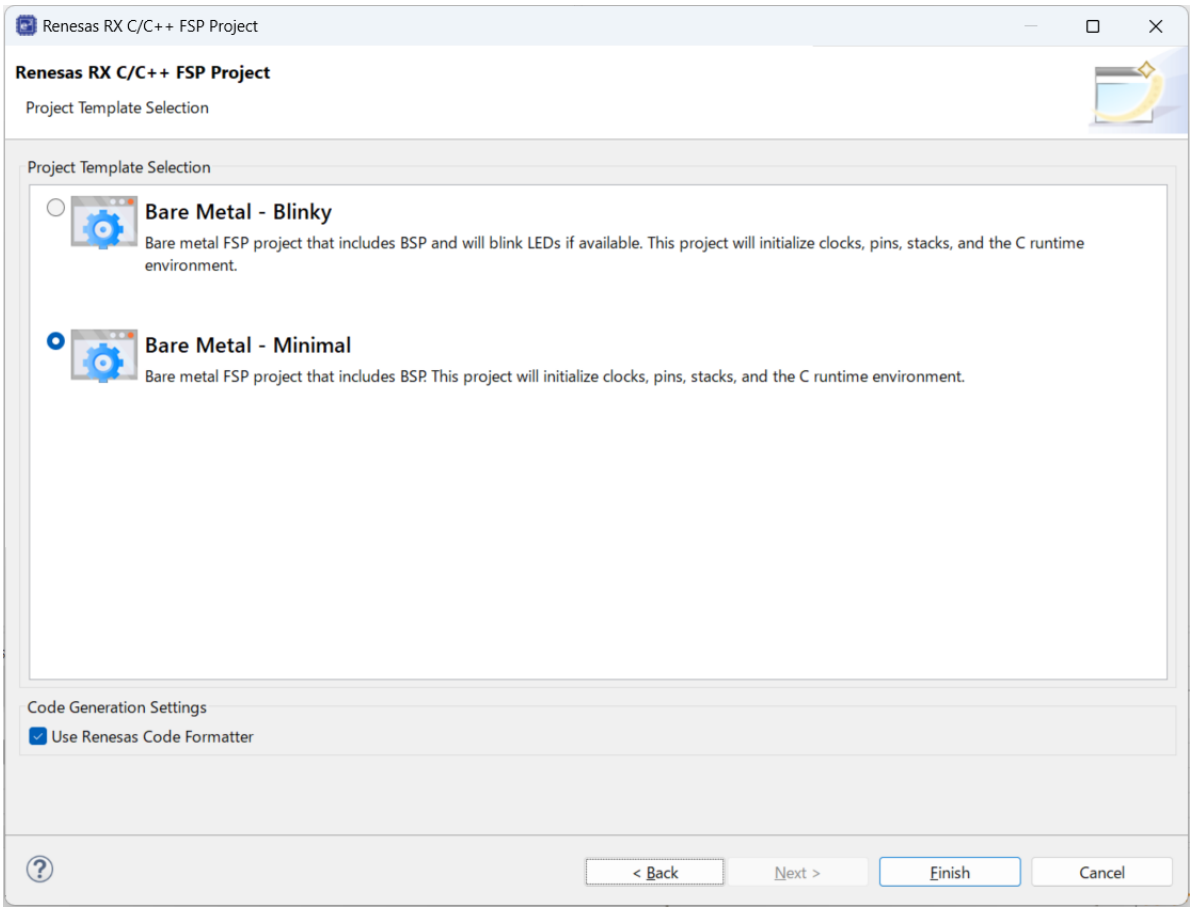


Figure 13: RX MCU Project Generator (Screen 4)

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

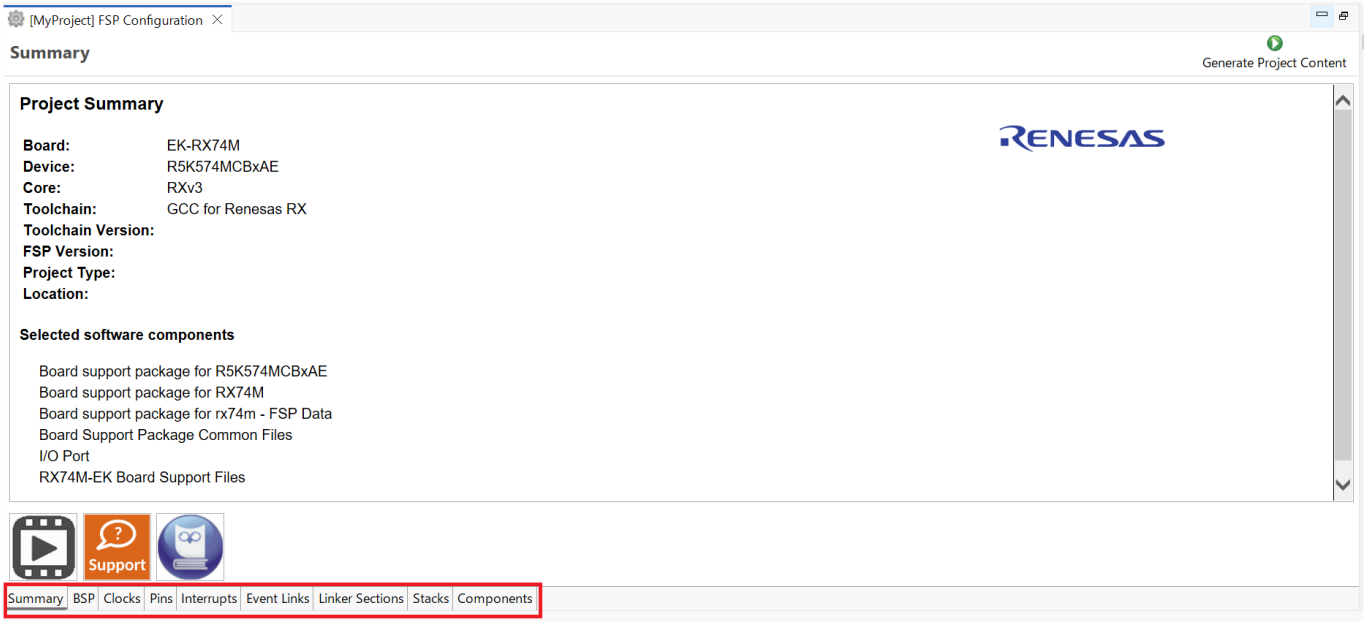


Figure 14: RX MCU Project Editor and available editor tabs

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

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

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

2.2.5 Configuring a Project

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

2.2.5.1 Summary Tab

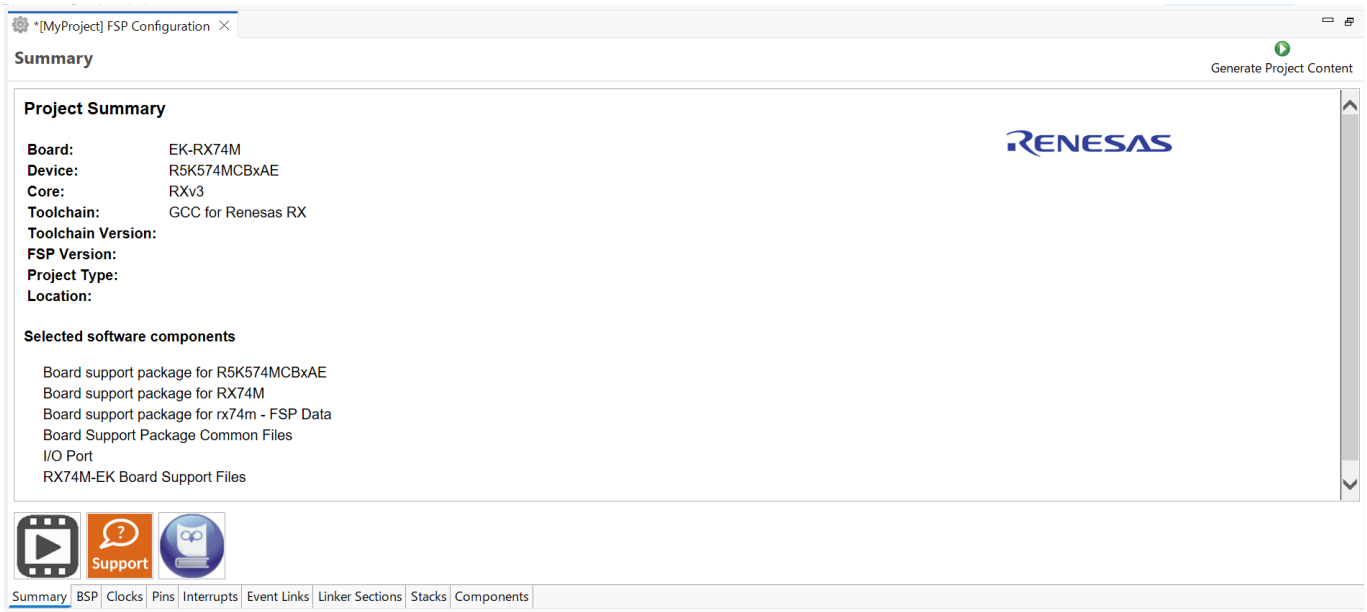


Figure 15: Configuration Summary tab

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

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

2.2.5.2 Configuring the BSP

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

Note

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

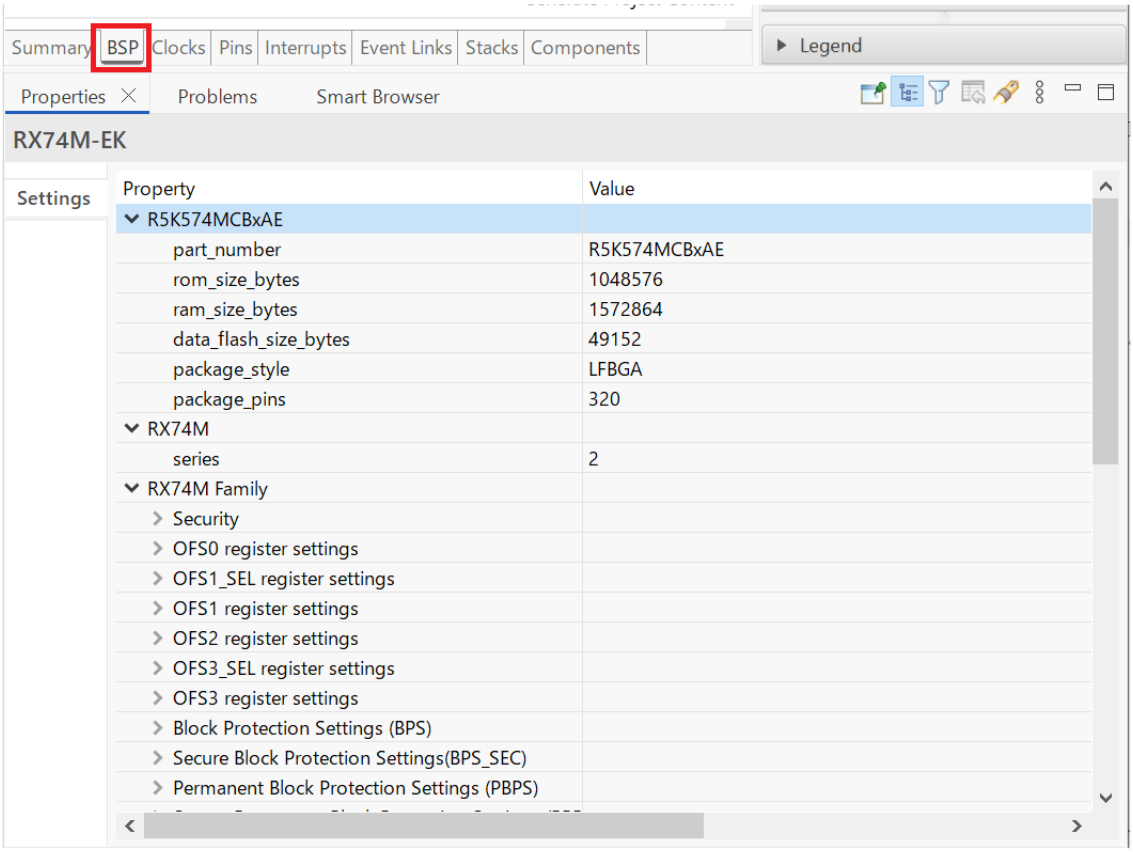


Figure 16: Configuration BSP tab

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

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

This file is created if it does not already exist.

Warning

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

2.2.5.3 Configuring Clocks

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

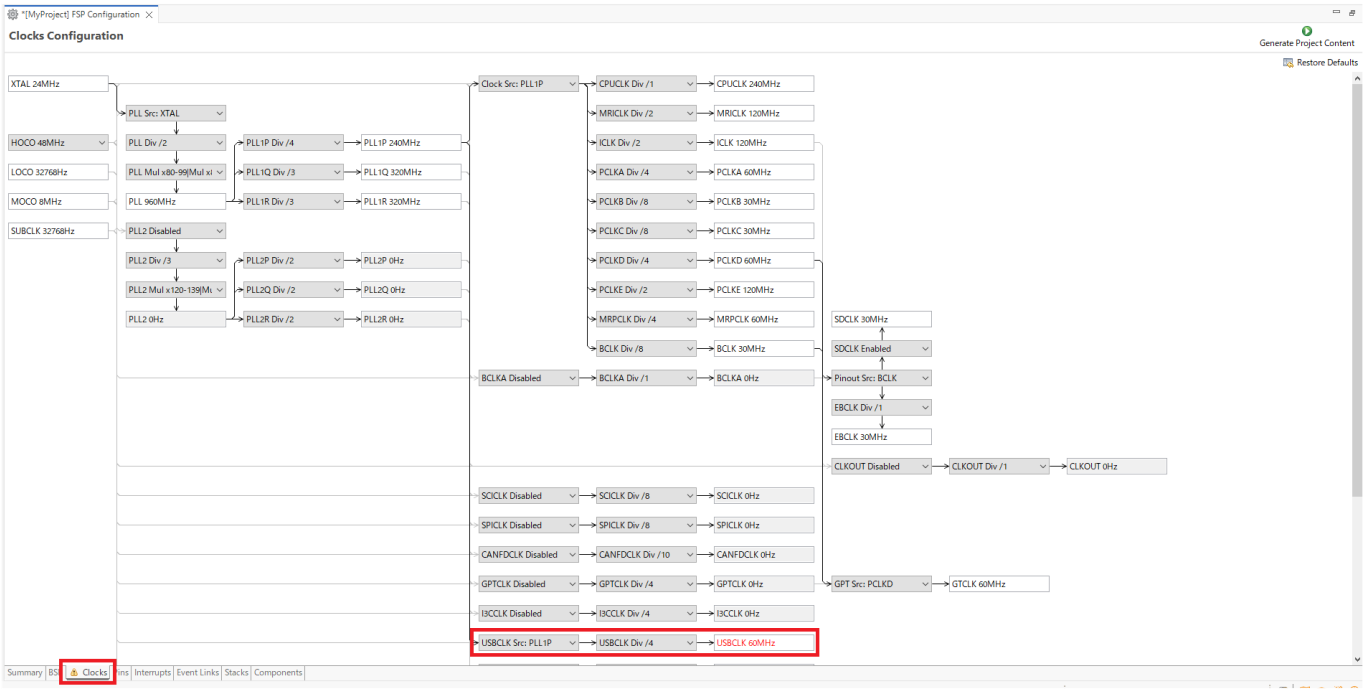


Figure 17: Configuration Clocks tab

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

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

Warning

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

2.2.5.4 Configuring Pins

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

Note

*If the **Package** view window is not open in e2 studio, select **Window** > **Show View** > **Pin Configurator** > **Package***

from the top menu bar to open it.

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

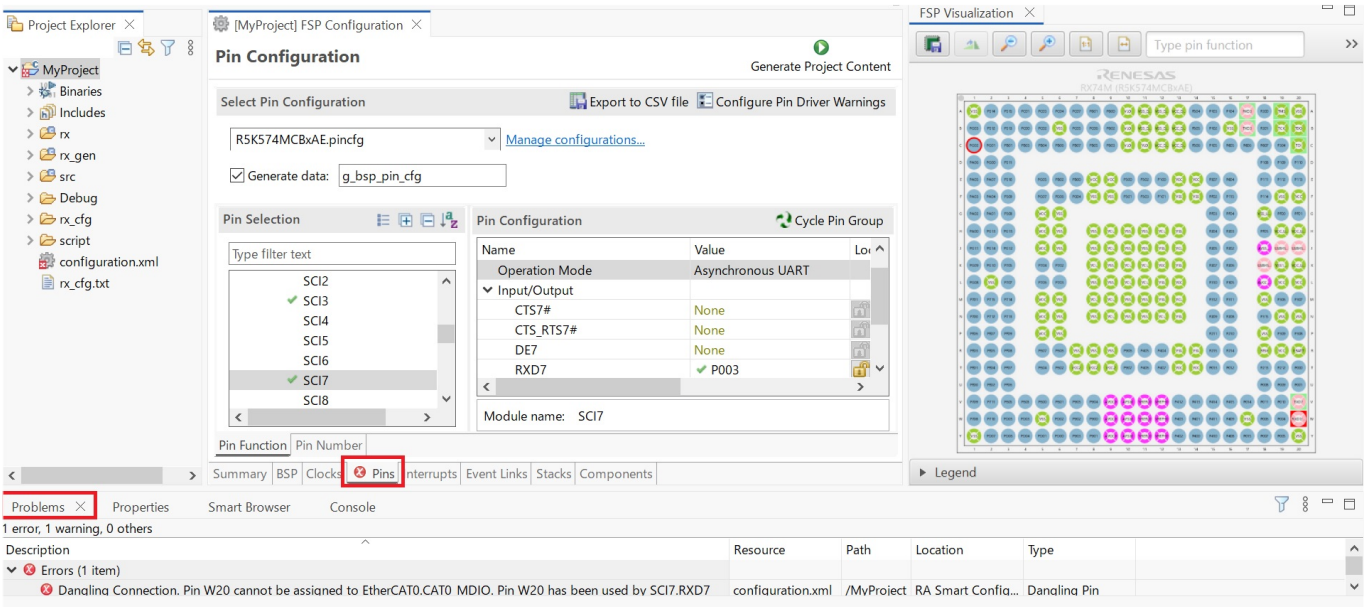


Figure 18: Pins Configuration

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

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

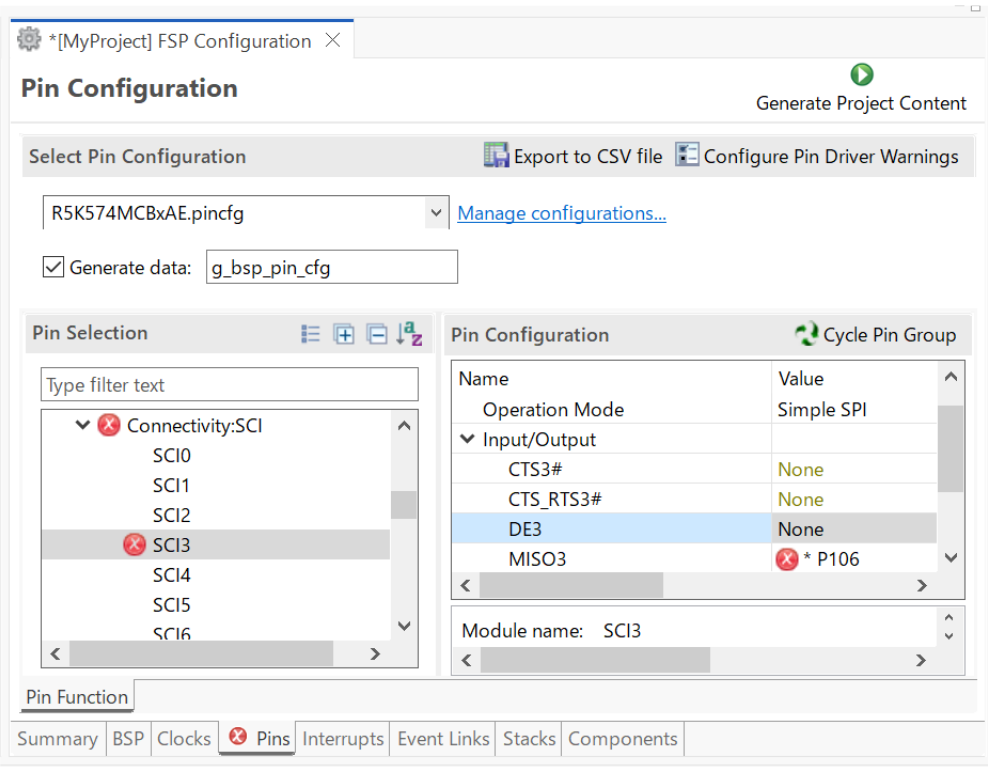


Figure 19: e2 studio Pin configurator

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

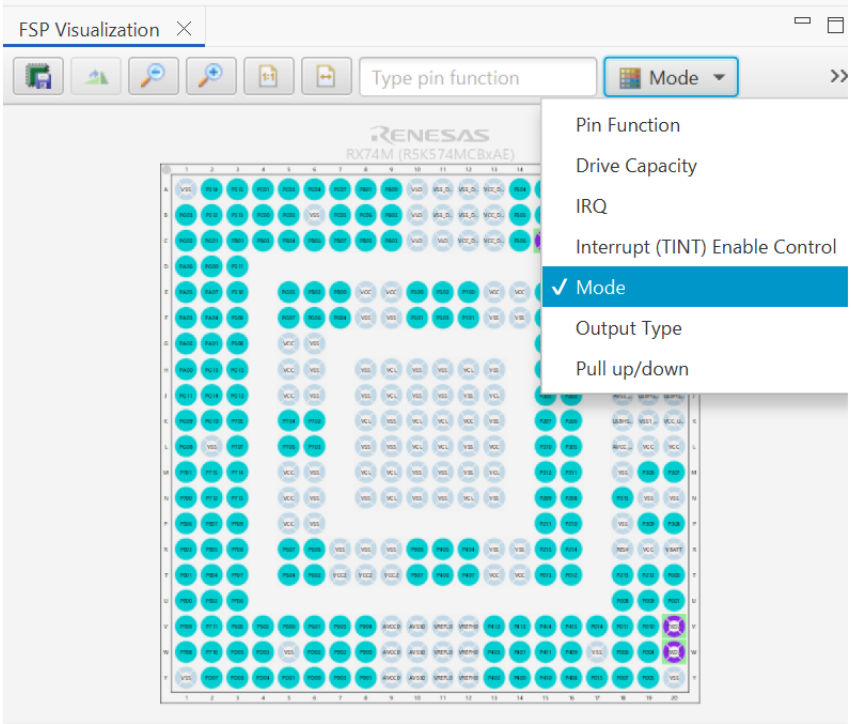


Figure 20: e2 studio Pin configurator package view

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

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

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

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

2.2.5.5 Configuring Interrupts from the Stacks Tab

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

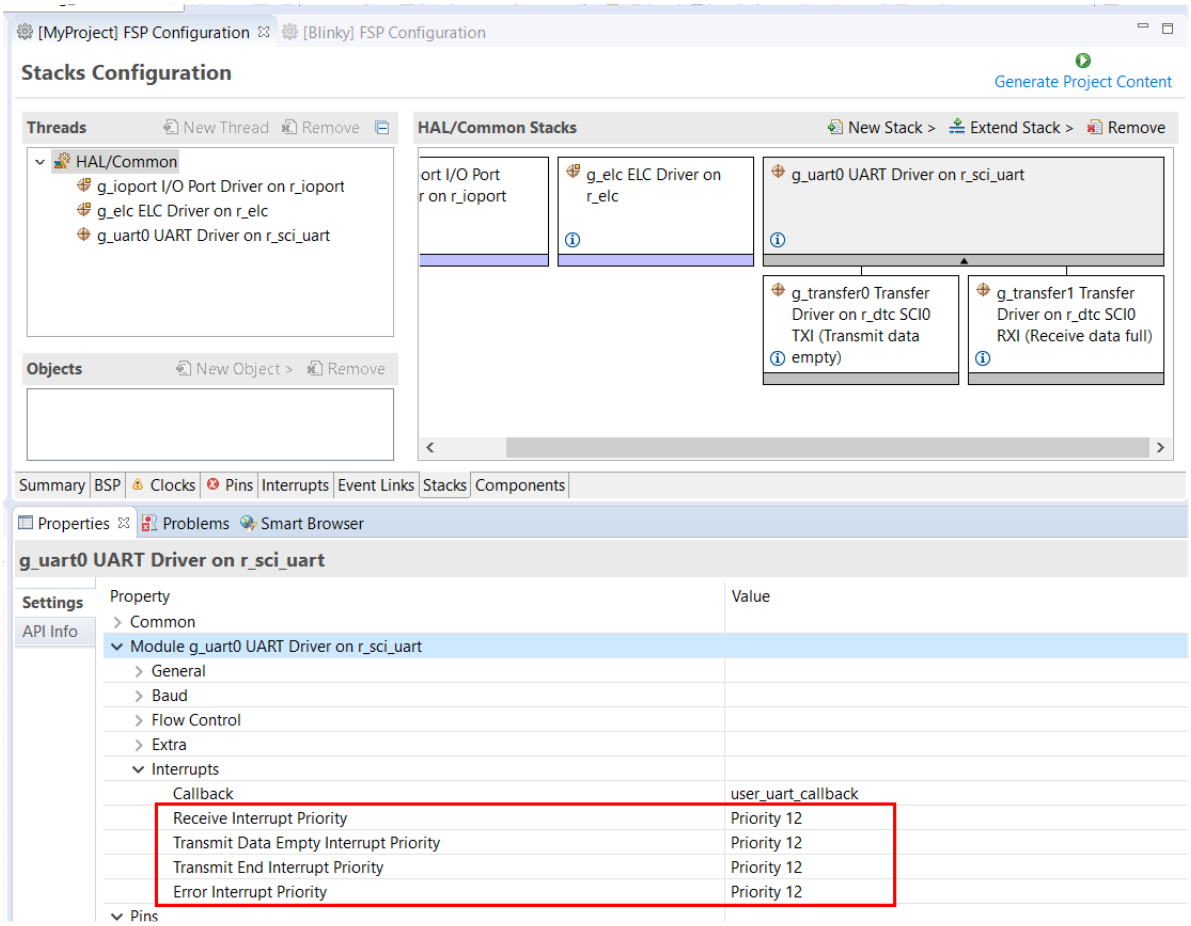


Figure 21: Configuring Interrupts in the Stacks tab

Creating Interrupts from the Interrupts Tab

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

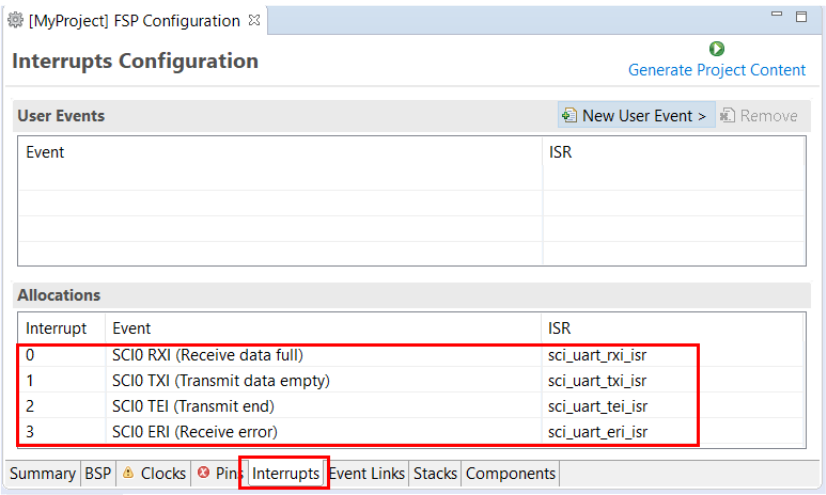


Figure 22: Configuring interrupt in Interrupt Tab

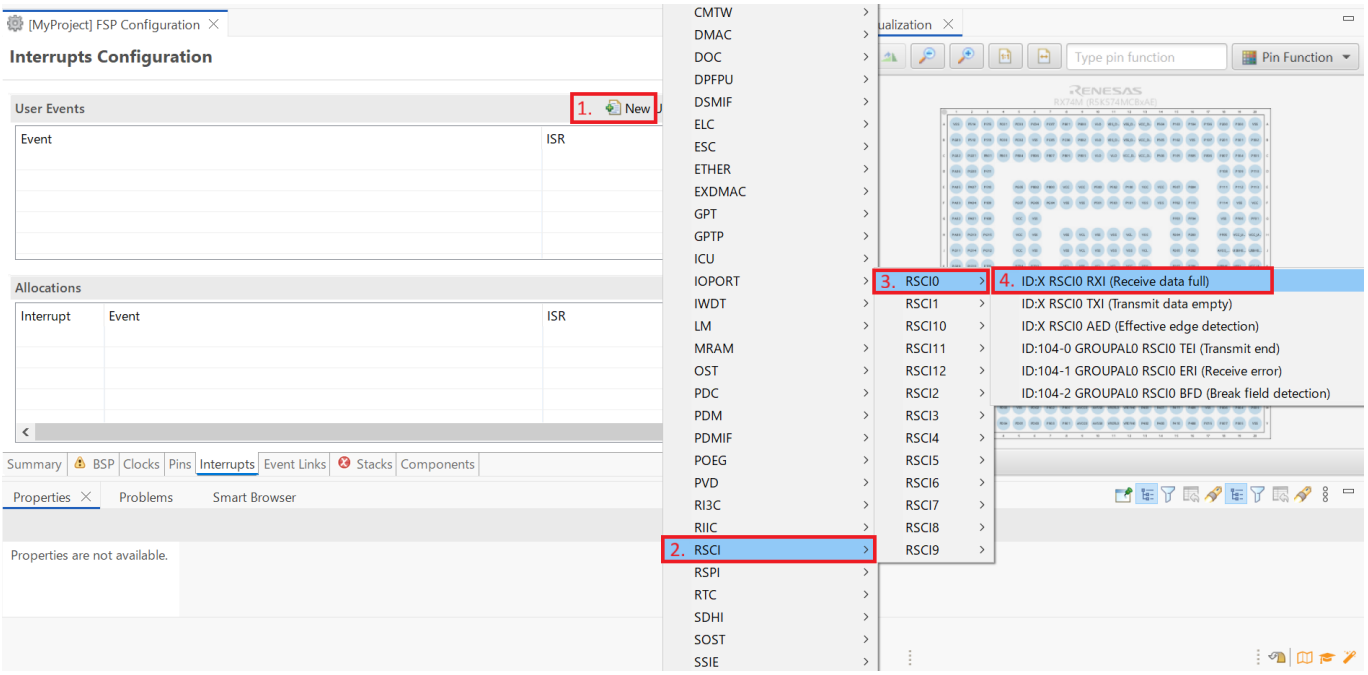


Figure 23: Adding user-defined event

Enter the name of ISR for the new user event.

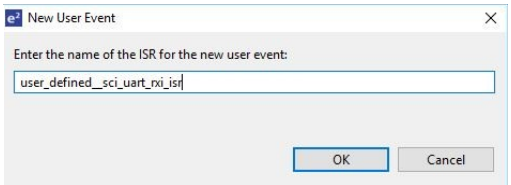


Figure 24: User-defined event ISR

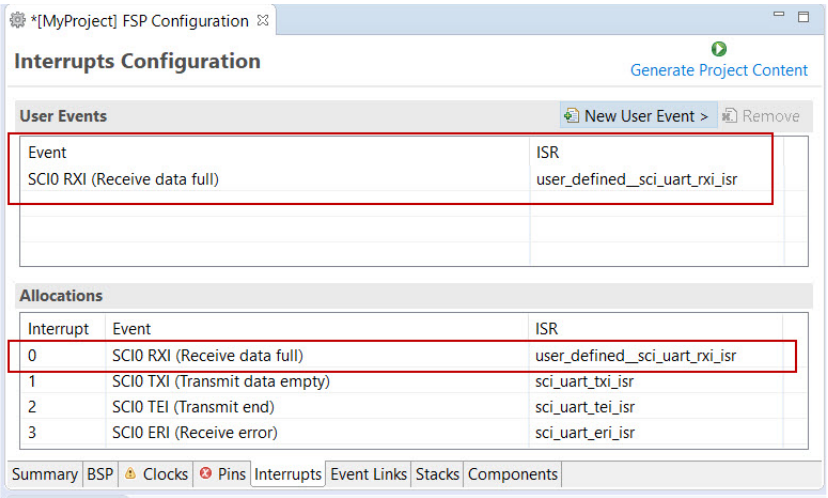


Figure 25: Using a user-defined event

2.2.5.6 Viewing Event Links

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

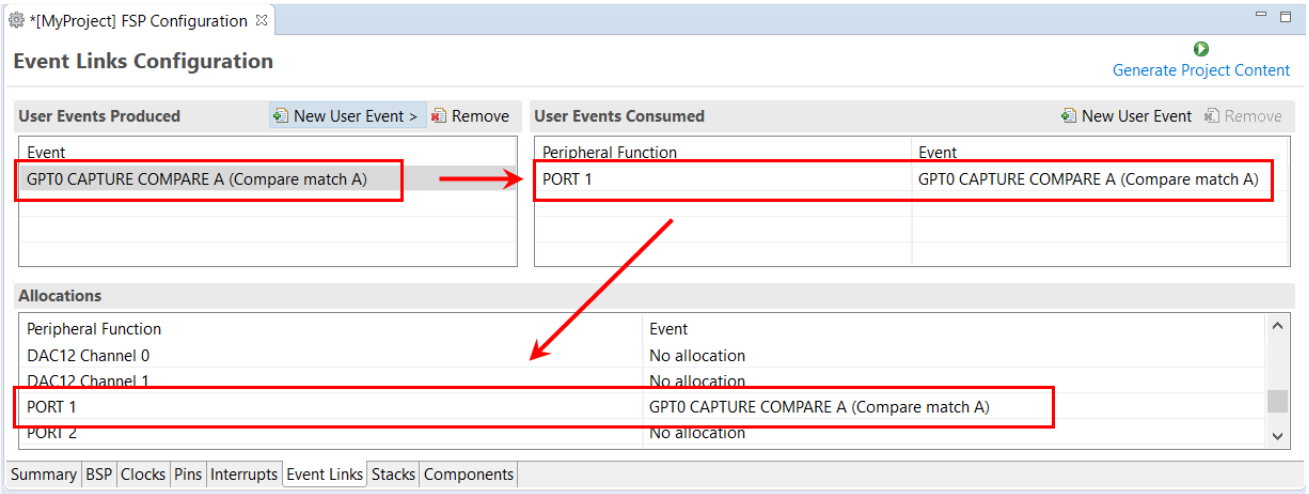


Figure 26: Viewing Event Links

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

Note

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

2.2.6 Adding Threads and Drivers

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

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

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

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

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

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

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

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

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

Note

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

2.2.6.1 Adding and Configuring HAL Drivers

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

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

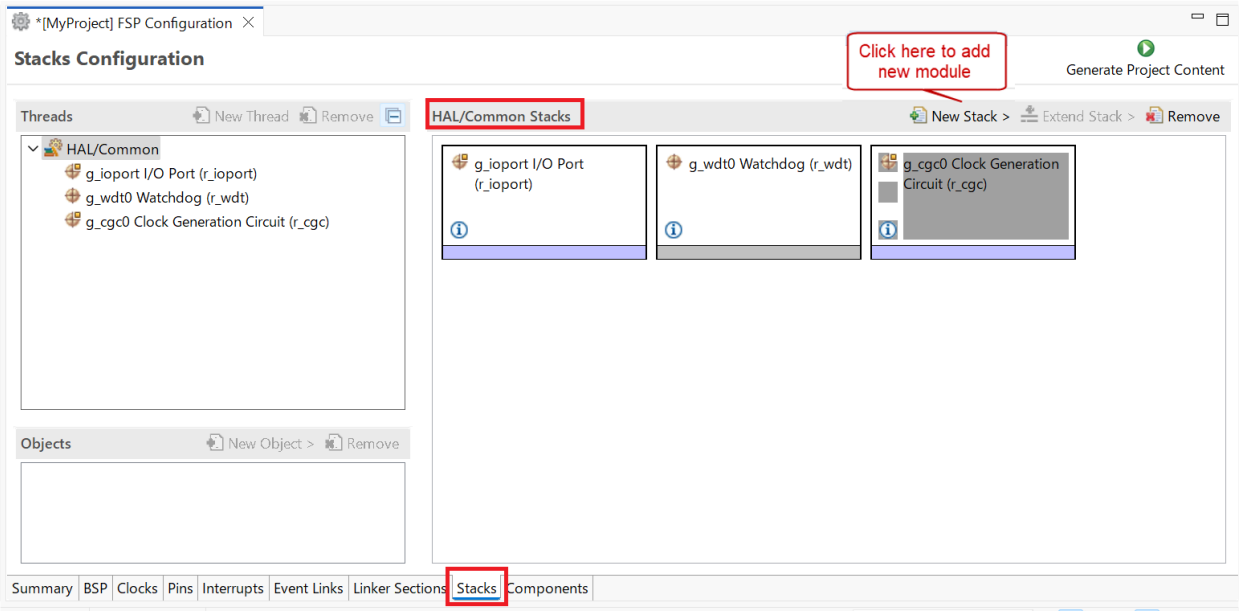


Figure 27: e2 studio Project configurator - Adding drivers

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

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

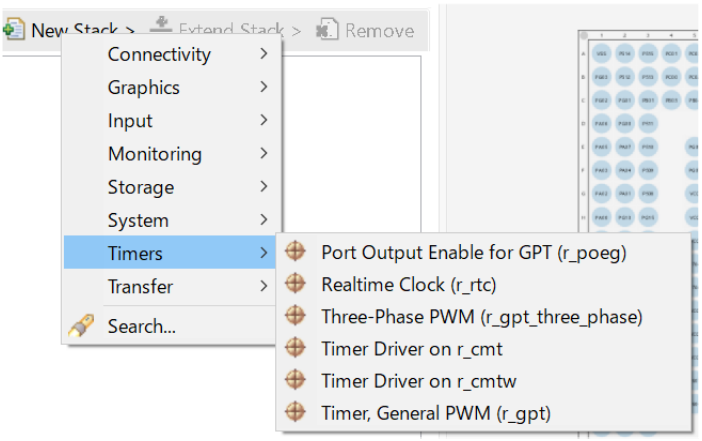


Figure 28: Select a driver

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

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

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

File	Contents	Overwritten by Generate Project Content?

rx_gen/main.c	Contains main() calling generated and user code. When called, the BSP already has Initialized the MCU.	Yes
rx_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
rx_gen/hal_data.h	Header file for HAL driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

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

2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers

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

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

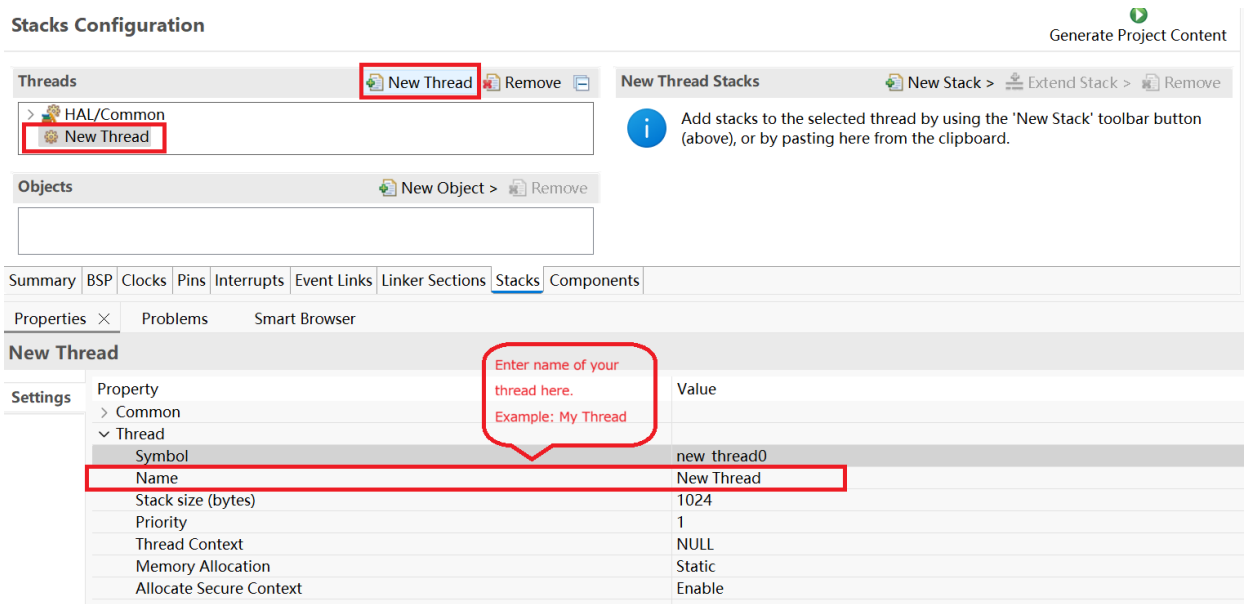


Figure 29: Adding a new RTOS Thread on the Stacks tab

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

Note

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

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

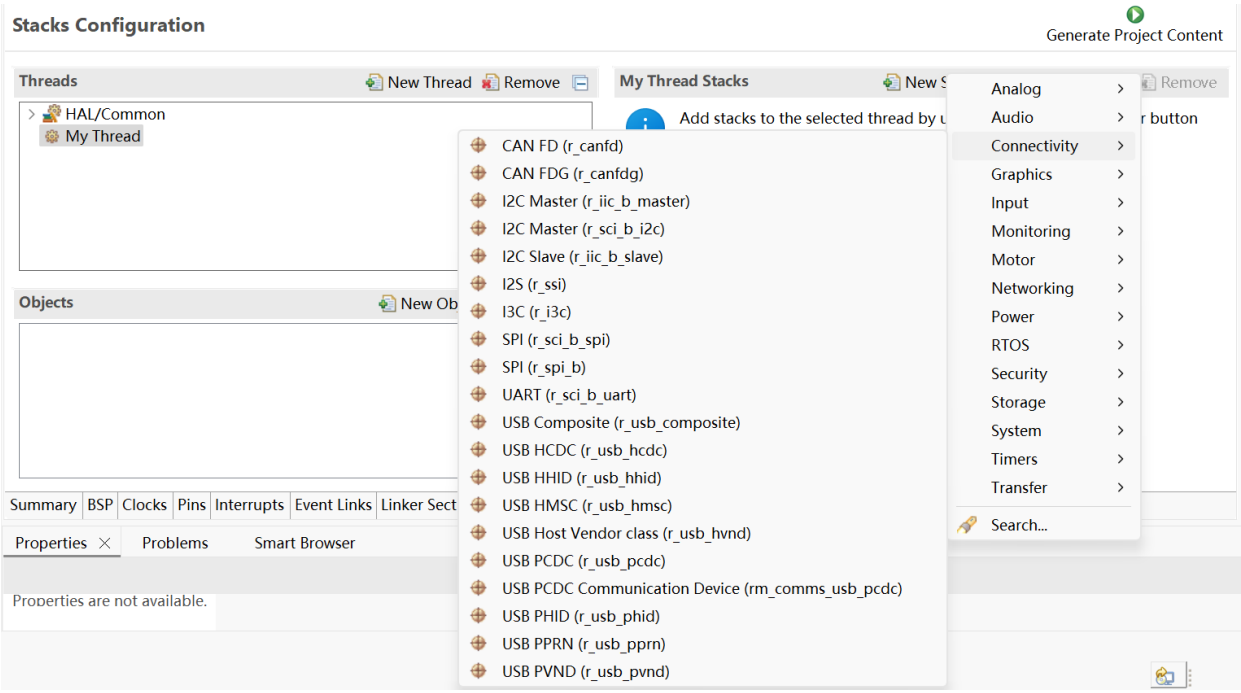


Figure 30: Adding Modules and Drivers to a thread

4. Select a module or driver from the list.

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

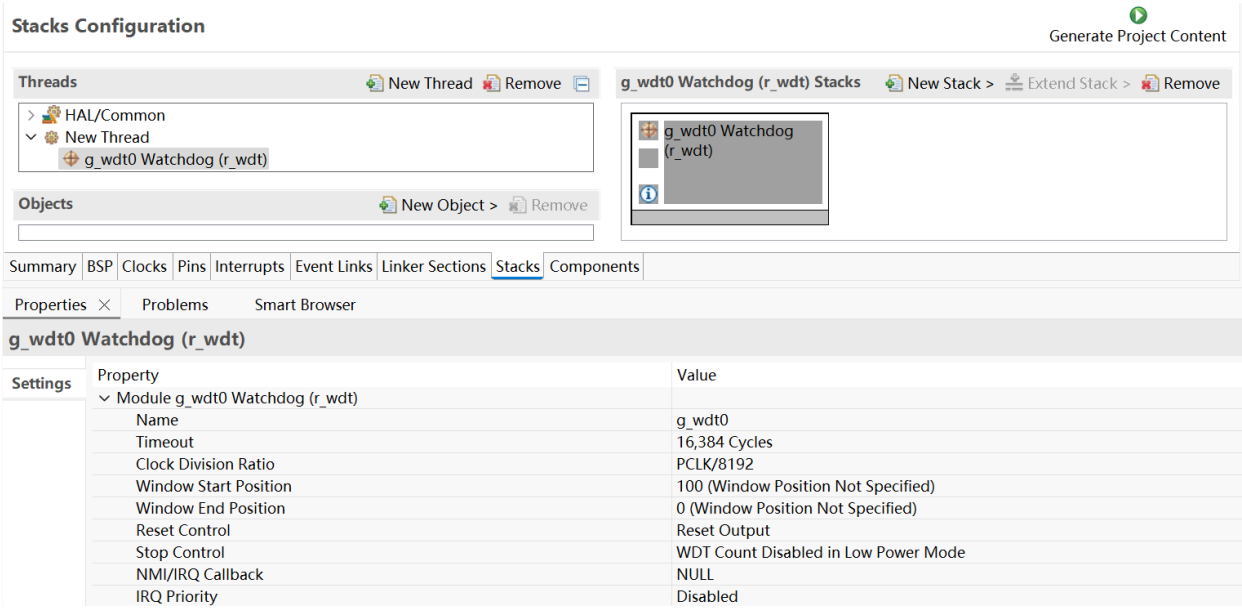


Figure 31: Configuring Module or Driver properties

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

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

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

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

2.2.6.3 Configuring Threads

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

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

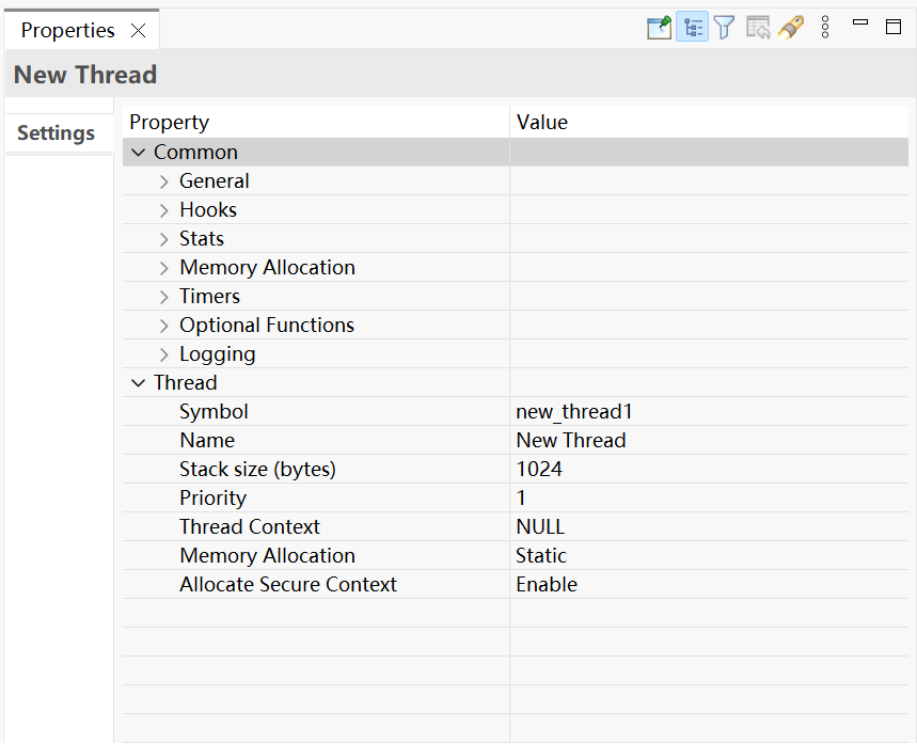


Figure 32: New Thread Properties

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

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

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

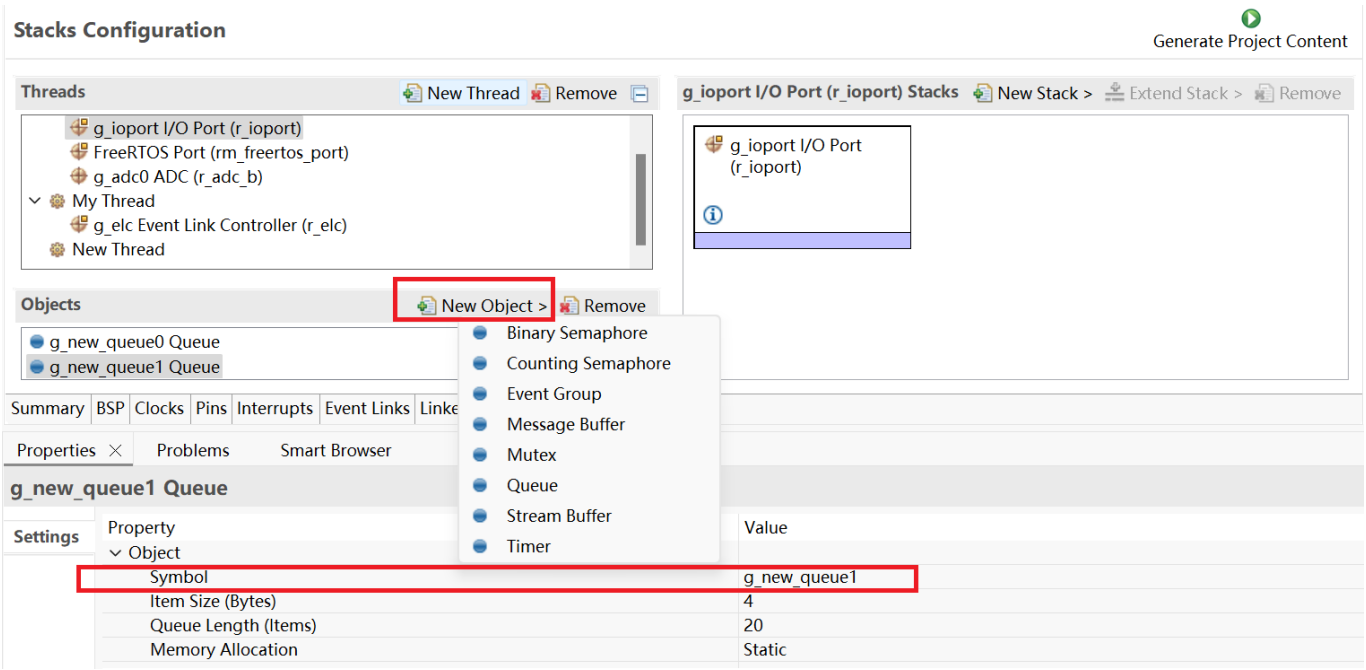


Figure 33: Configuring Thread Object Properties

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

2.2.7 Reviewing and Adding Components

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

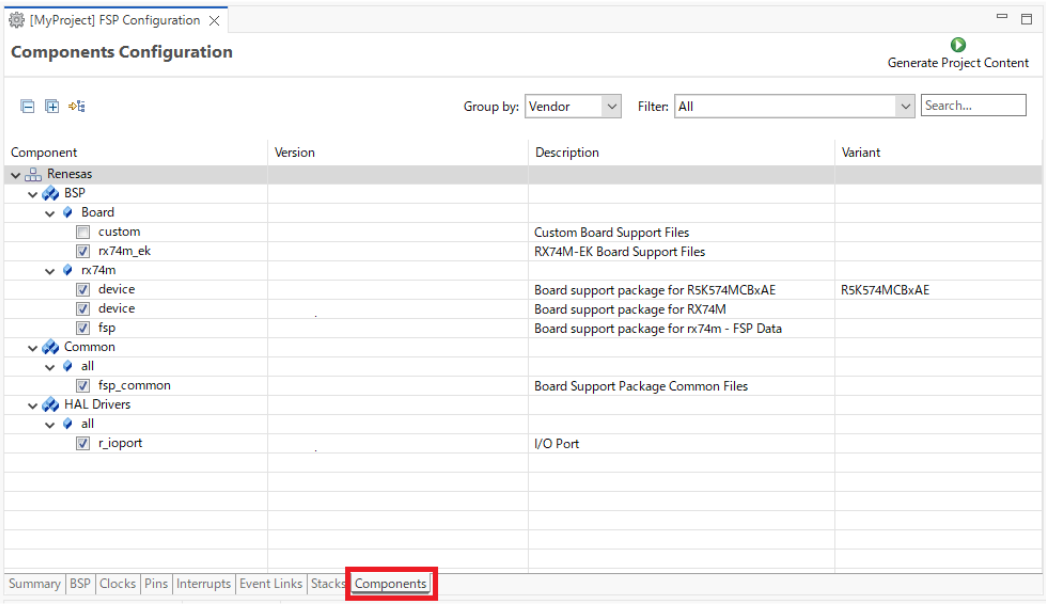


Figure 34: Components Tab

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

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

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

2.2.8 Writing the Application

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

Note

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

2.2.8.1 Coding Features

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

Autocomplete

Autocomplete is a context aware coding accelerator that suggests possible completions for partially typed-in code elements. If you can 'guess' the first part of a macro, for example, the Autocomplete function can suggest options for completing the rest of the macro.

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

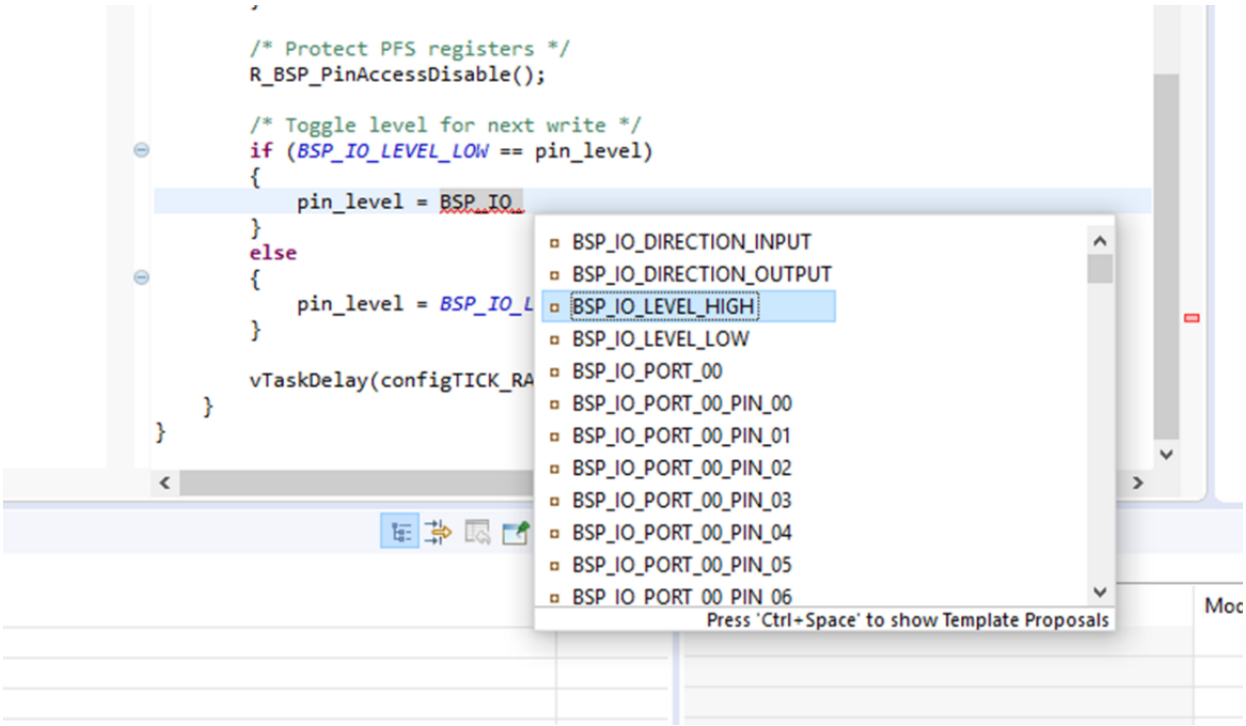


Figure 35: Autocomplete example

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

Welcome Window

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

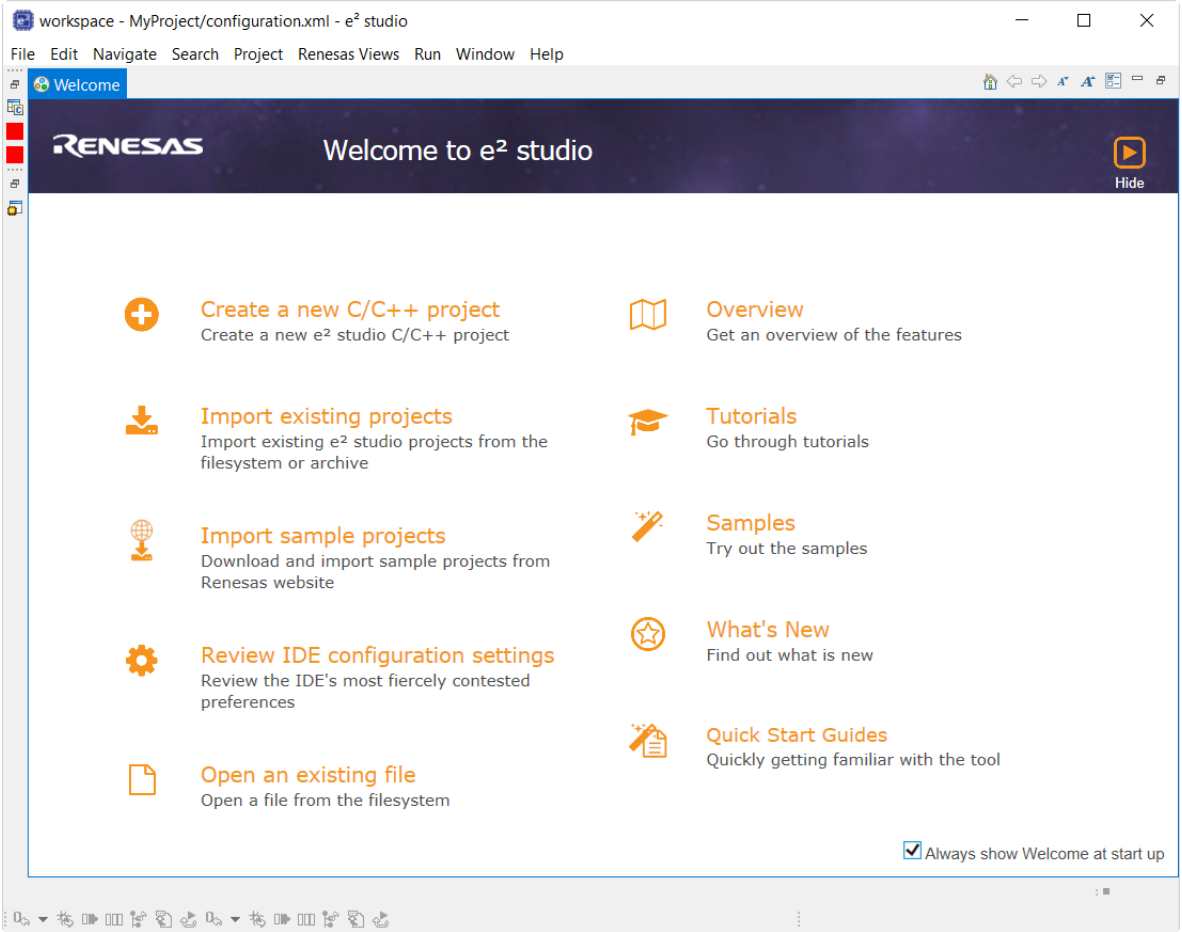


Figure 36: Welcome window

Cheat Sheets

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

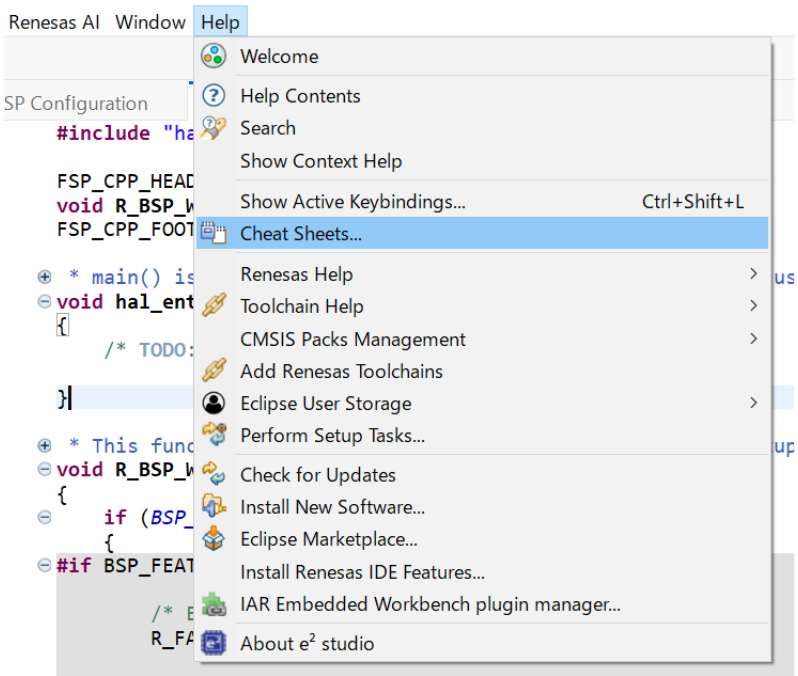


Figure 37: Cheat Sheets

Developer Assistance

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

1. Expand the project explorer to view Developer Assistance

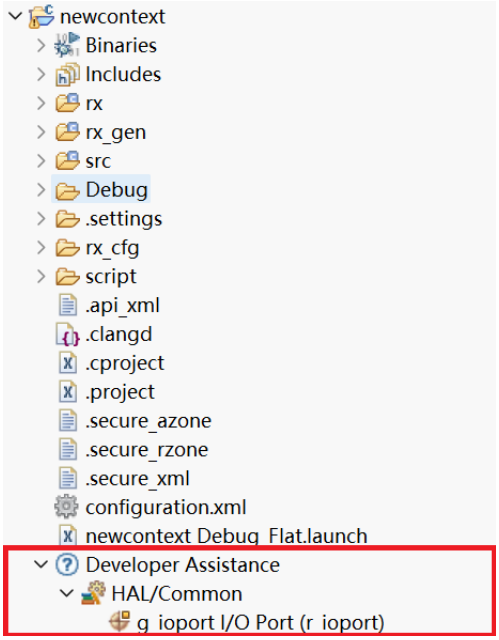


Figure 38: Developer Assistance

2. Expand a stack module to show its APIs

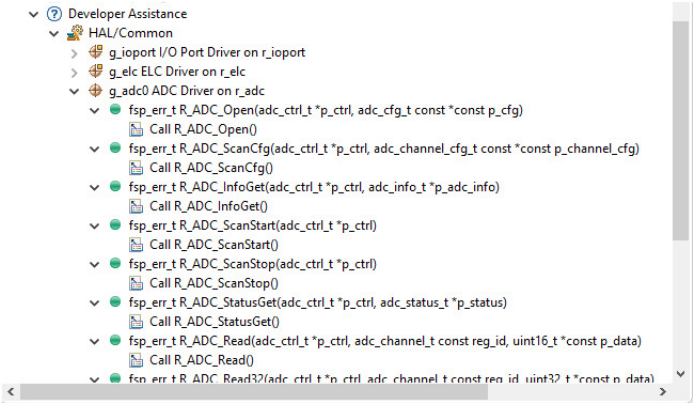


Figure 39: Developer Assistance APIs

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

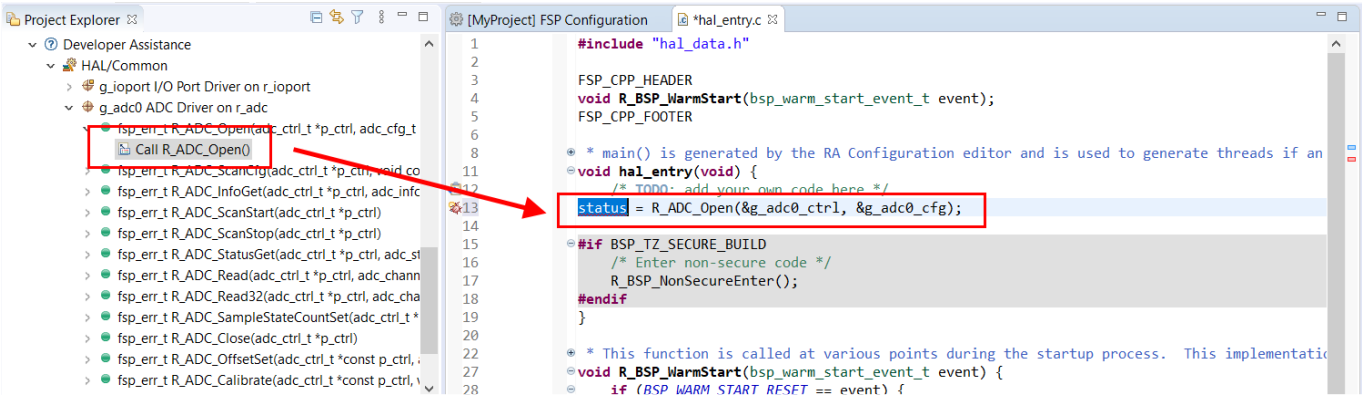


Figure 40: Dragging and Dropping an API in Developer Assistance

Information Icon

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

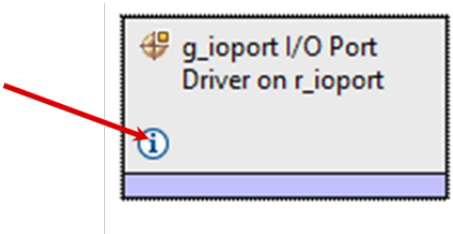


Figure 41: Information icon

IDE Help

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

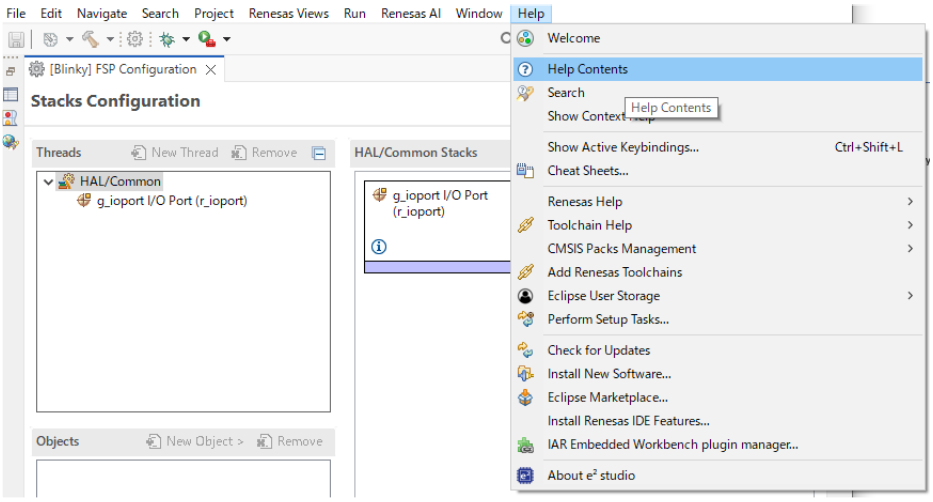


Figure 42: Opening the Help System

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

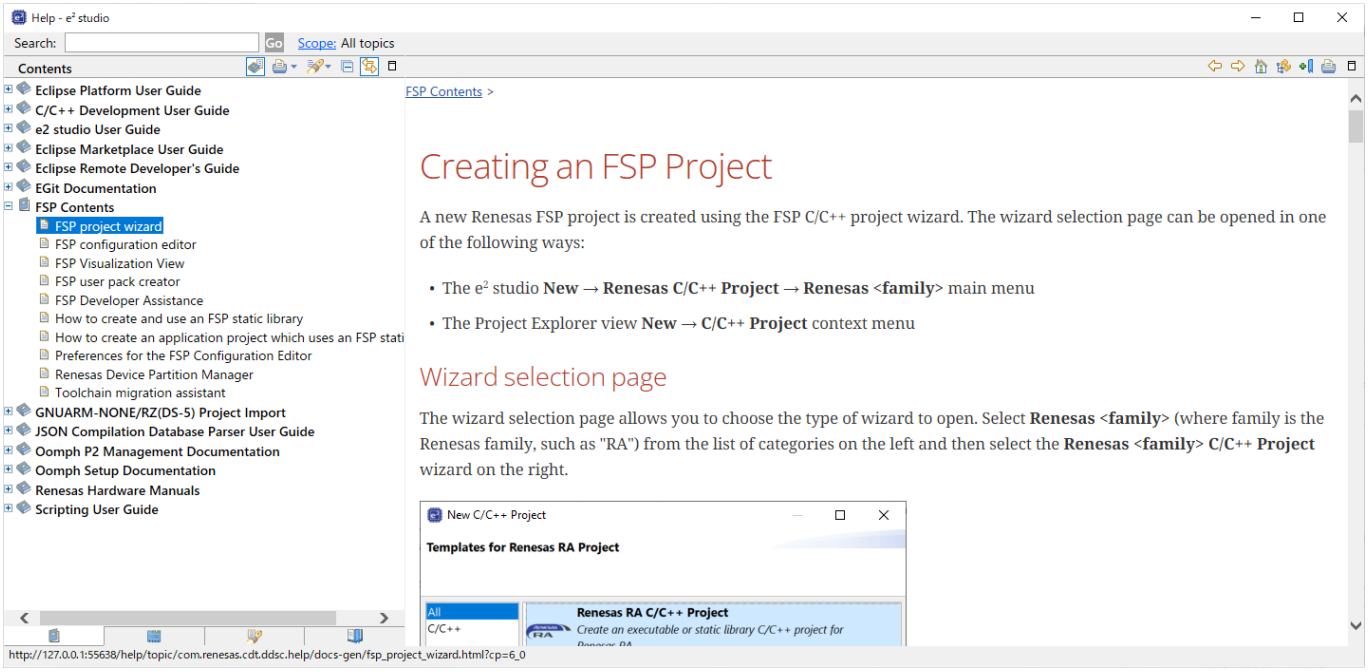


Figure 43: FSP Content Help

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

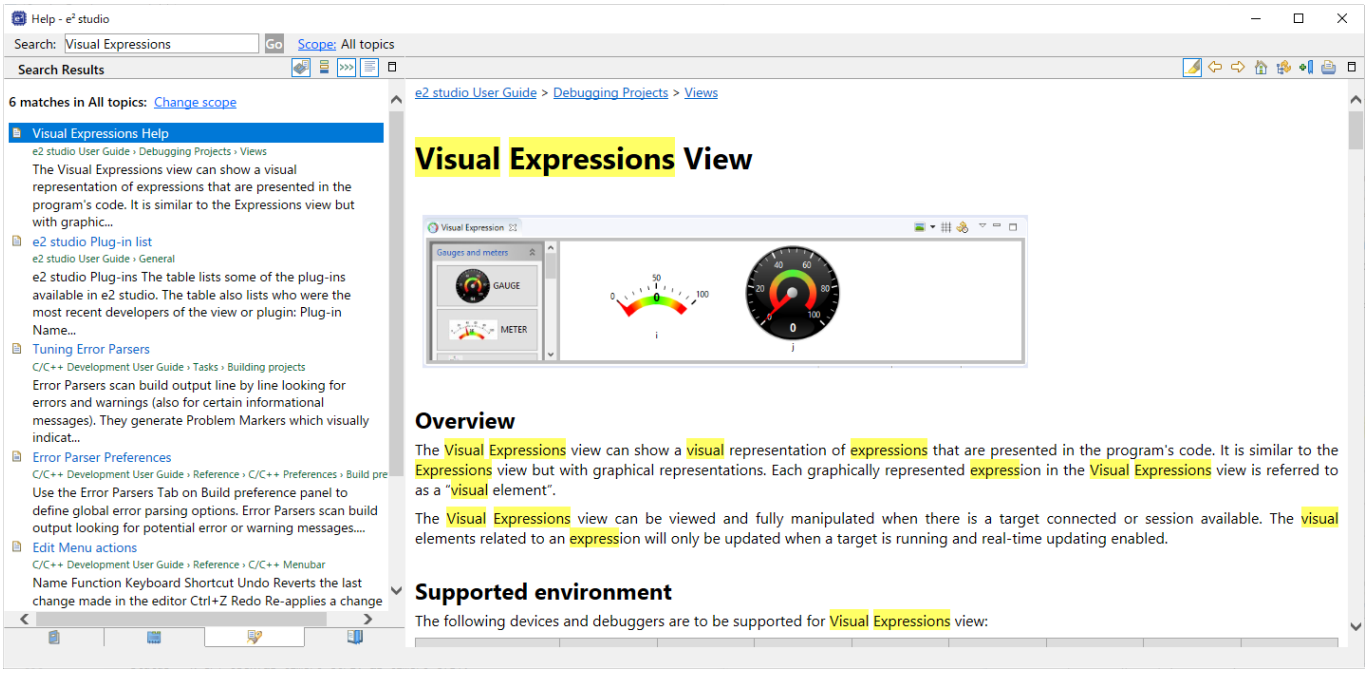


Figure 44: e2 studio Help from the Search Bar

2.2.8.2 HAL Modules in FSP: A Practical Description

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

Introduction to HAL Modules

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

HAL Driver API Function Call Formats

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

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

Here are some examples:

```
status = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);

status = R_GPT_Start(&g_timer0_ctrl);
```

```
status = R_GPT_PeriodSet(&g_timer0_ctrl, period);

status = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);

status = R_ADC_InfoGet(&g_adc0_ctrl, &adc_info);
```

HAL Driver API Call Reference Information

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

◆ R_GPT_PeriodSet()

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

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

Warning

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

Example:

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

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

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

Return values

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

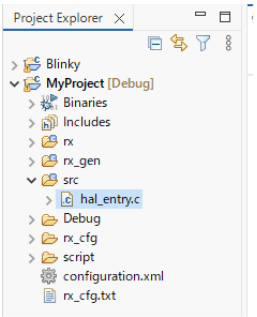
Figure 45: Module Api Reference Section Example

2.2.8.3 RTOS-Independent Applications

To write application code:

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

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



Note

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

Warning

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

5. Add your application code here:

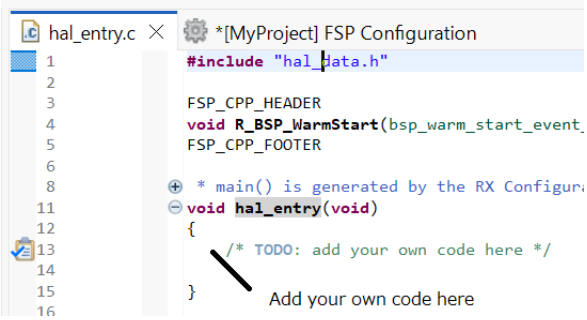


Figure 46: Adding user code to hal_entry.c

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

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

The WDT example is a HAL level application which does not use an RTOS.

The user guides for each module also include basic application code that you can add to hal_entry.c.

2.2.8.4 RTOS Applications

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

1. Add a thread using the **Stacks** tab.
2. Provide a unique name for the thread in the **Properties** view for this thread.
3. Configure all drivers and resources for this thread and resolve all dependencies flagged by

e2 studio such as missing interrupts or drivers.

4. Configure the thread objects.
5. Provide unique names for each thread object in the **Properties** view for each object.
6. Add more threads if needed and repeat steps 1 to 5.
7. In the **RX Project Editor**, click the **Generate Project Content** button.

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

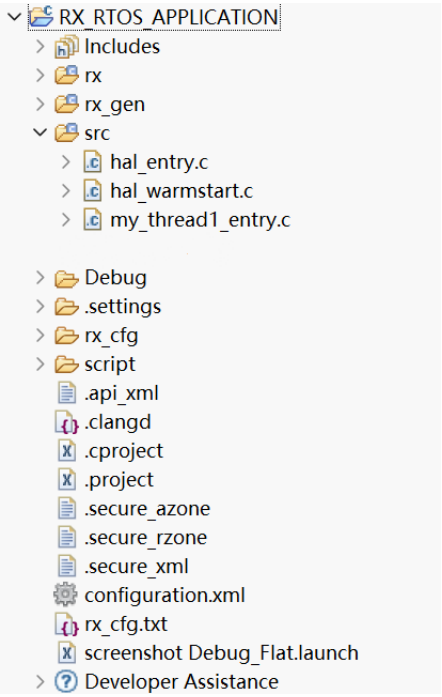


Figure 47: Generated files for an RTOS application

Note

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

Warning

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

9. Add your application code here:

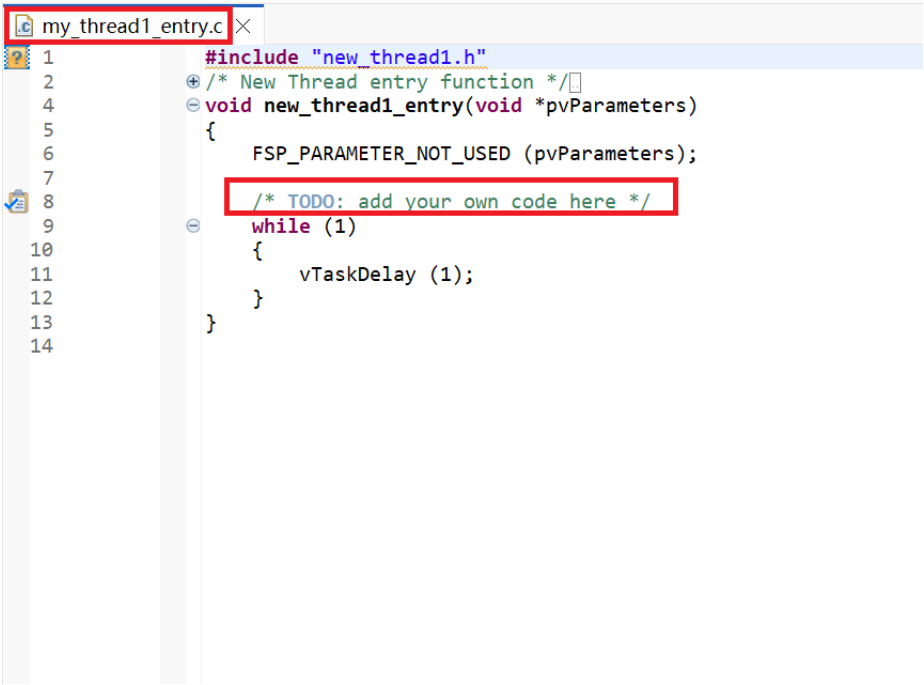


Figure 48: Adding user code to my_thread_1.entry

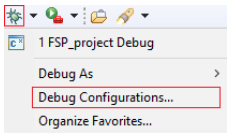
10. Repeat steps 1 to 9 for the next thread.
11. Build your project without errors by clicking on **Project > Build Project**.

2.2.9 Debugging the Project

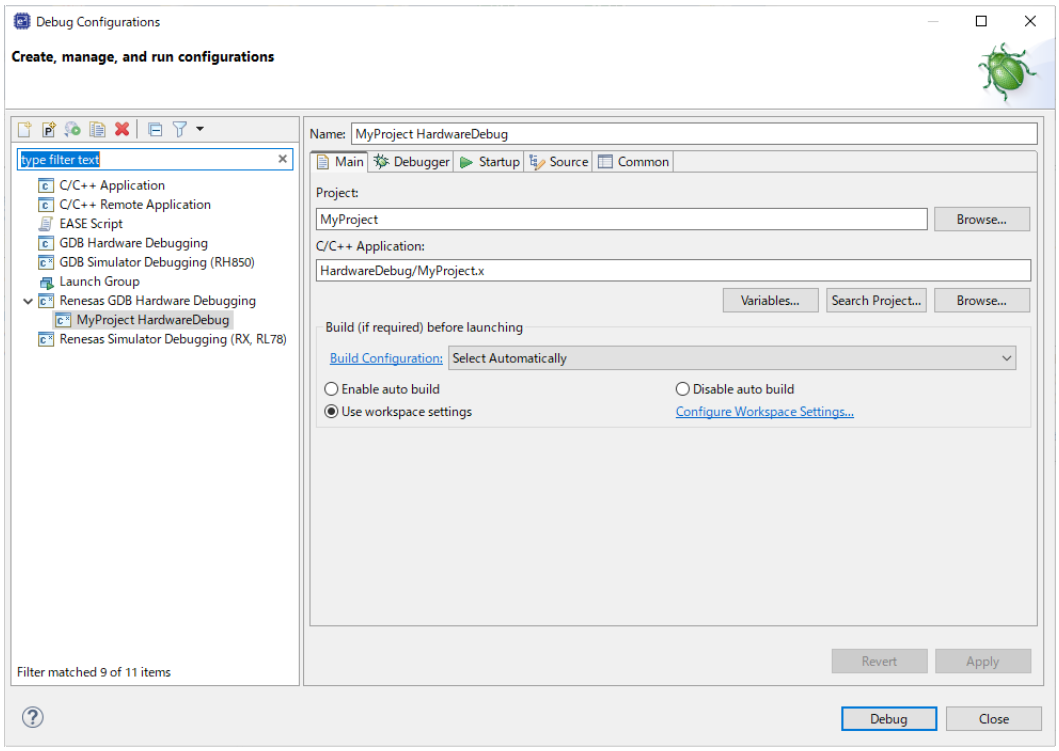
Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

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



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



3. Connect the board to your PC via either an E2OB On-Board debugger, or an E2 or E2 Lite and click **Debug**.

Note

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

2.2.10 Modifying Toolchain Settings

There are instances where it may be necessary to make changes to the toolchain being used (for example, to change optimization level of the compiler or add a library to the linker). Such modifications can be made from within e2 studio through the menu **Project > Properties > C/C++ Build > Settings** when the project is selected. The following screenshot shows the settings dialog for the GCC for Renesas RX toolchain. This dialog will look slightly different depending upon the toolchain being used.

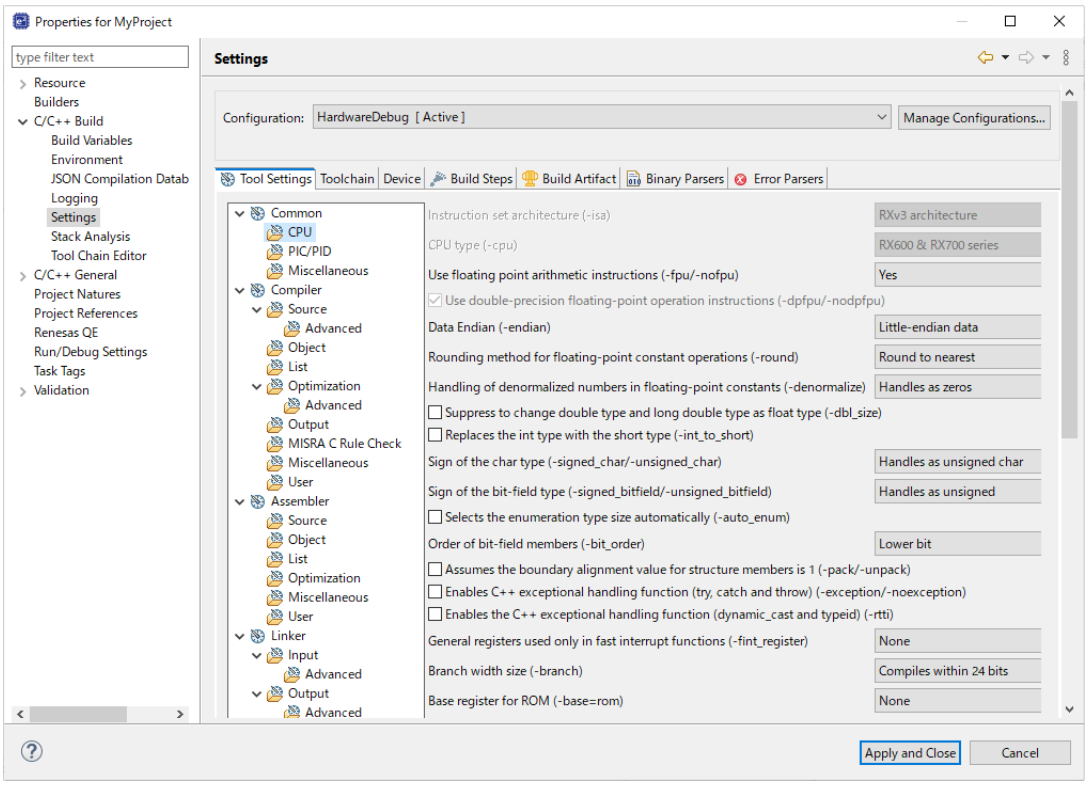


Figure 49: e2 studio Project toolchain settings

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

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

2.2.11 Importing an Existing Project into e2 studio

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

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

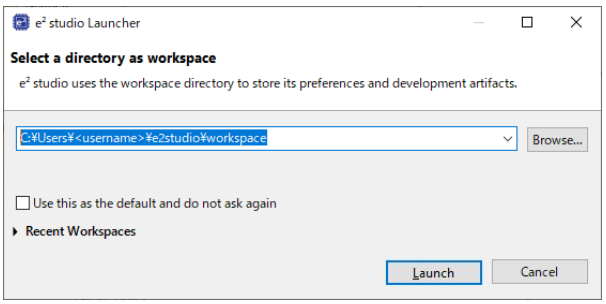


Figure 50: Workspace Launcher dialog

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

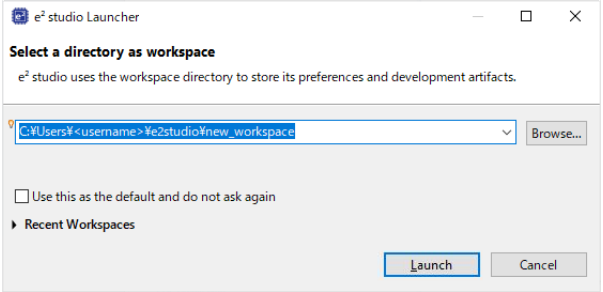


Figure 51: Workspace Launcher dialog - Select Workspace

c. Click **Launch**.

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

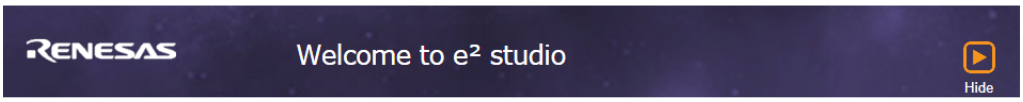


Figure 52: Workbench arrow button

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

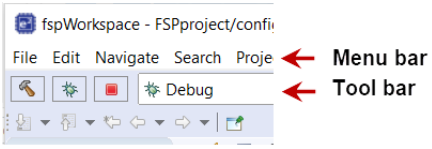


Figure 53: Menu and tool bar

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

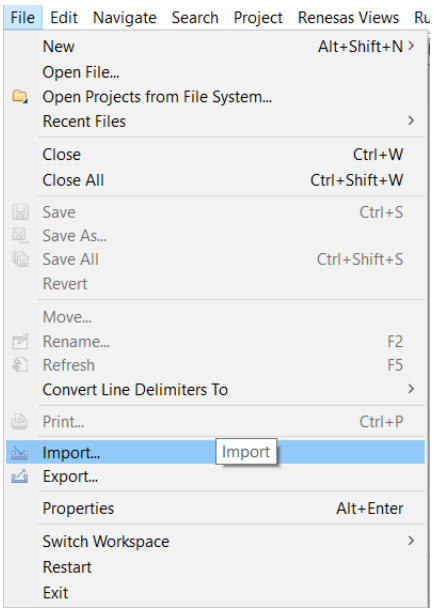


Figure 54: File drop-down menu

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

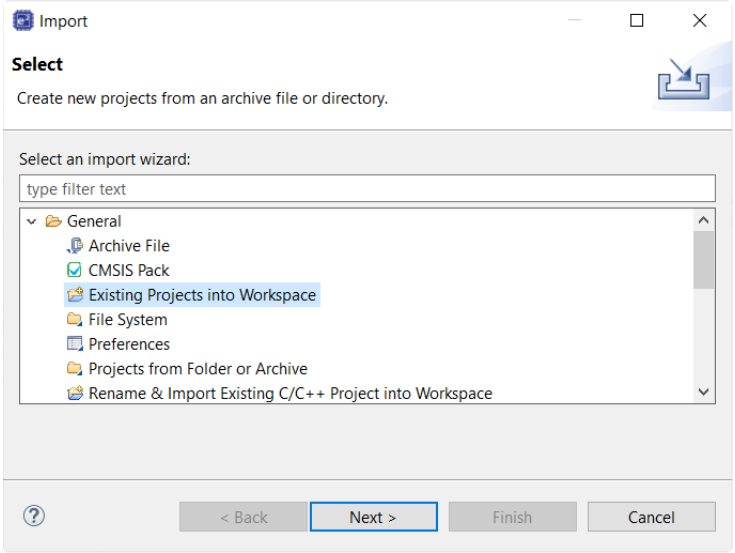


Figure 55: Project Import dialog with "Existing Projects into Workspace" option selected

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

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

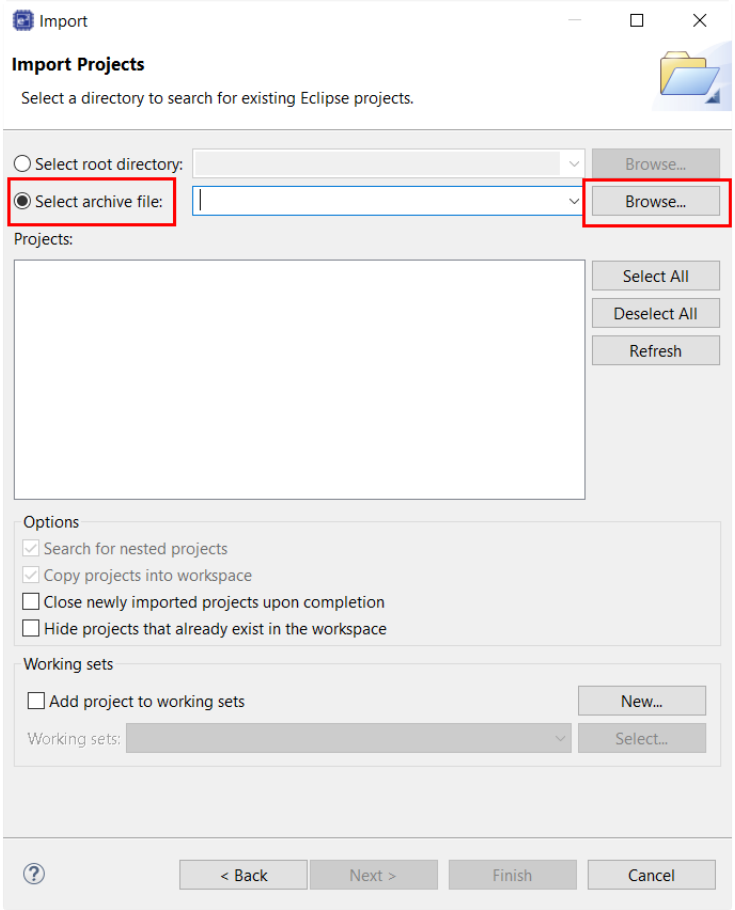


Figure 56: Import Existing Project dialog 1 - Select archive file

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

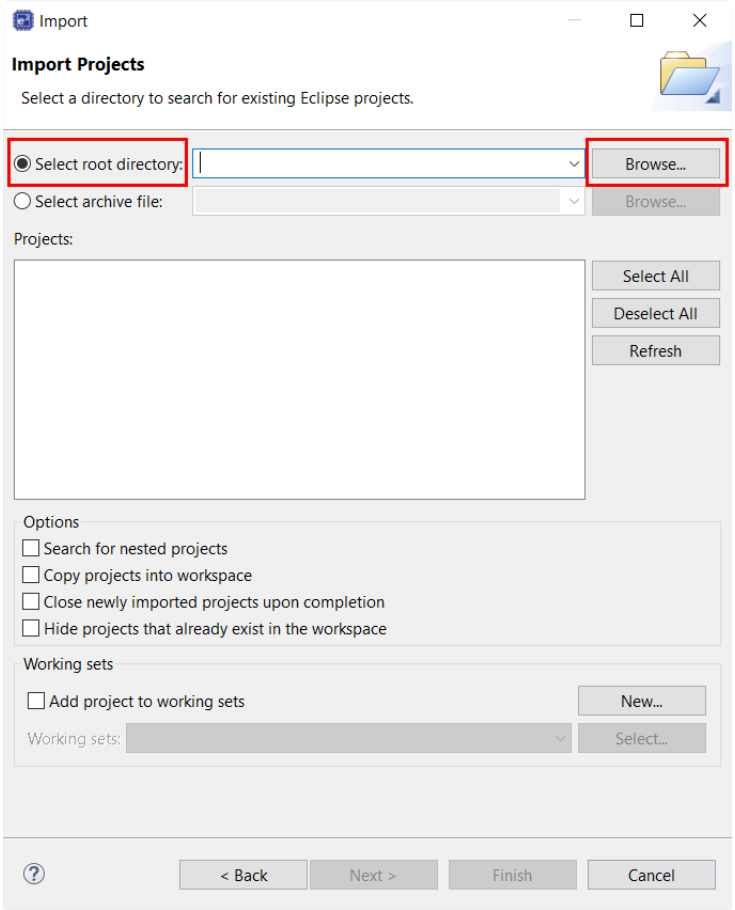


Figure 57: Import Existing Project dialog 1 - Select root directory

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

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

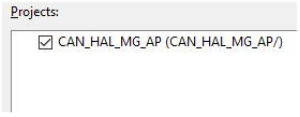


Figure 58: Import Existing Project dialog 2

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

2.3 Tutorial: Your First RX MCU Project - Blinky

2.3.1 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Flexible Platform by moving through the

steps of creating a simple application using e² studio and running that application on an RX MCU board.

2.3.2 What Does Blinky Do?

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

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

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

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

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

2.3.3 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e² studio
- Flexible Software Package
- A RX MCU board kit

2.3.4 Create a New Project for Blinky

The creation and configuration of a RX MCU project is the first step in the creation of an application. The base RX MCU pack includes a pre-written Blinky example application that is simple and works on all Renesas RX MCU boards.

Follow these steps to create an RX MCU project:

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

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

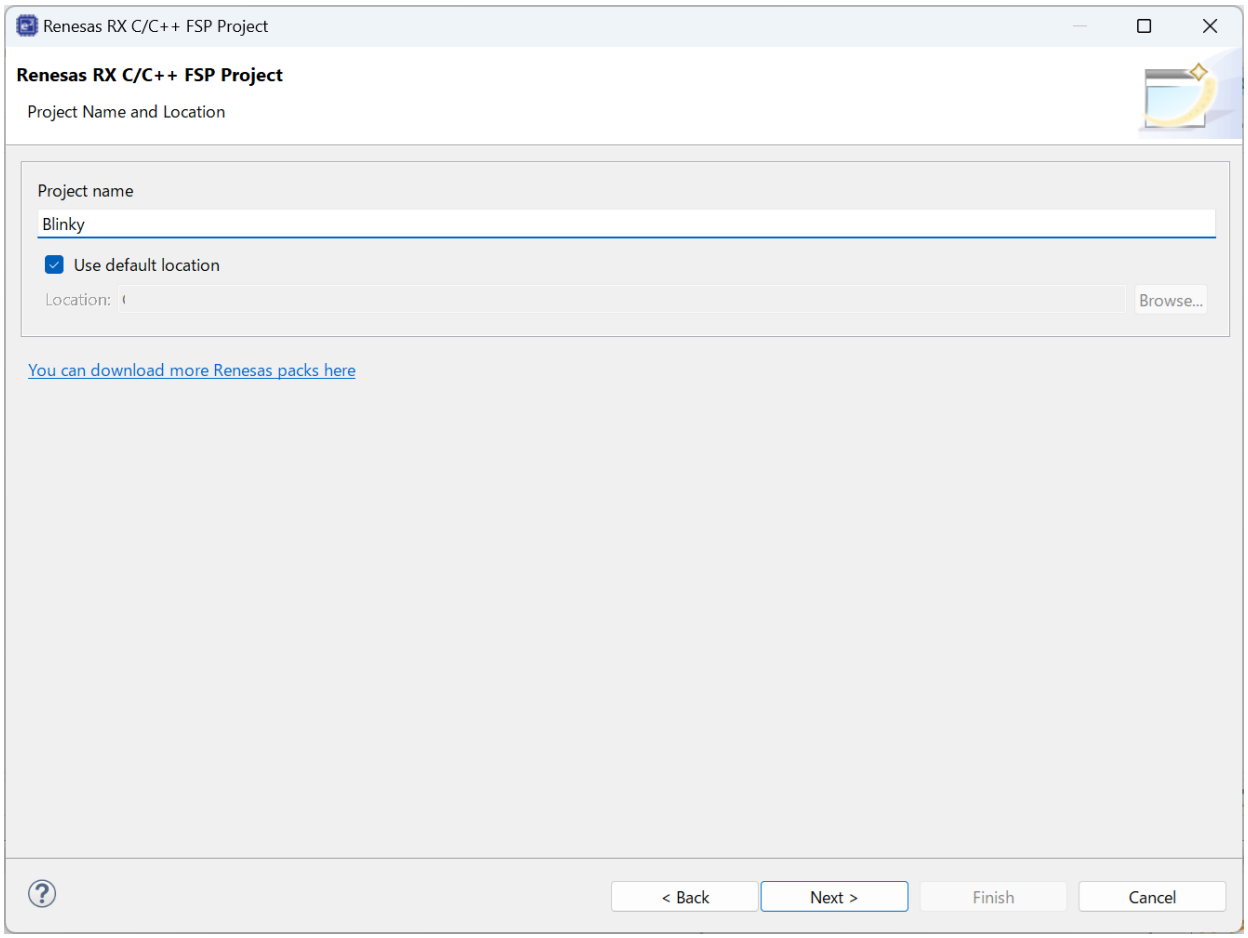


Figure 59: e² studio Project Configuration window (part 1)

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

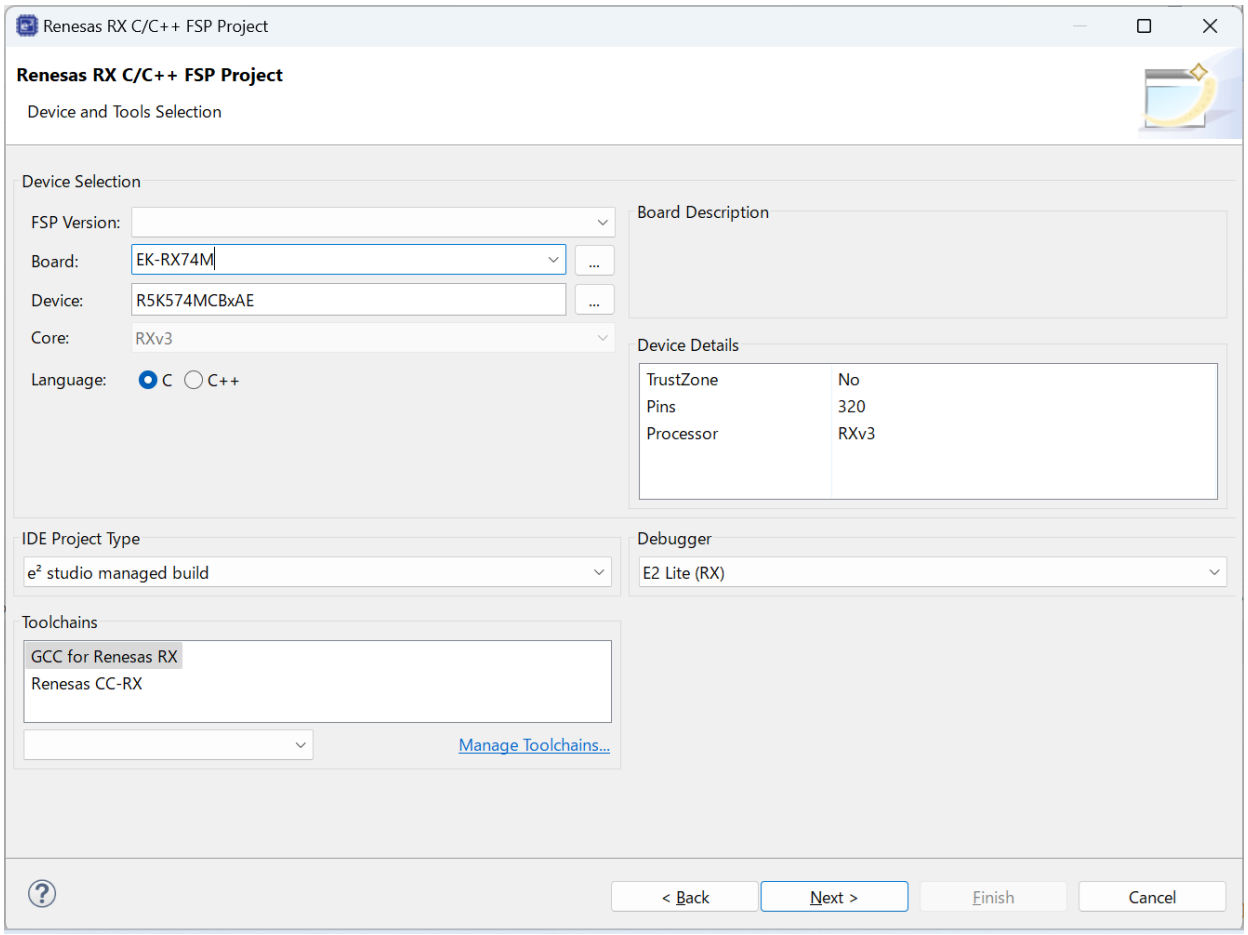


Figure 60: e² studio Project Configuration window (part 2)

5. Select the build artifact and RTOS.

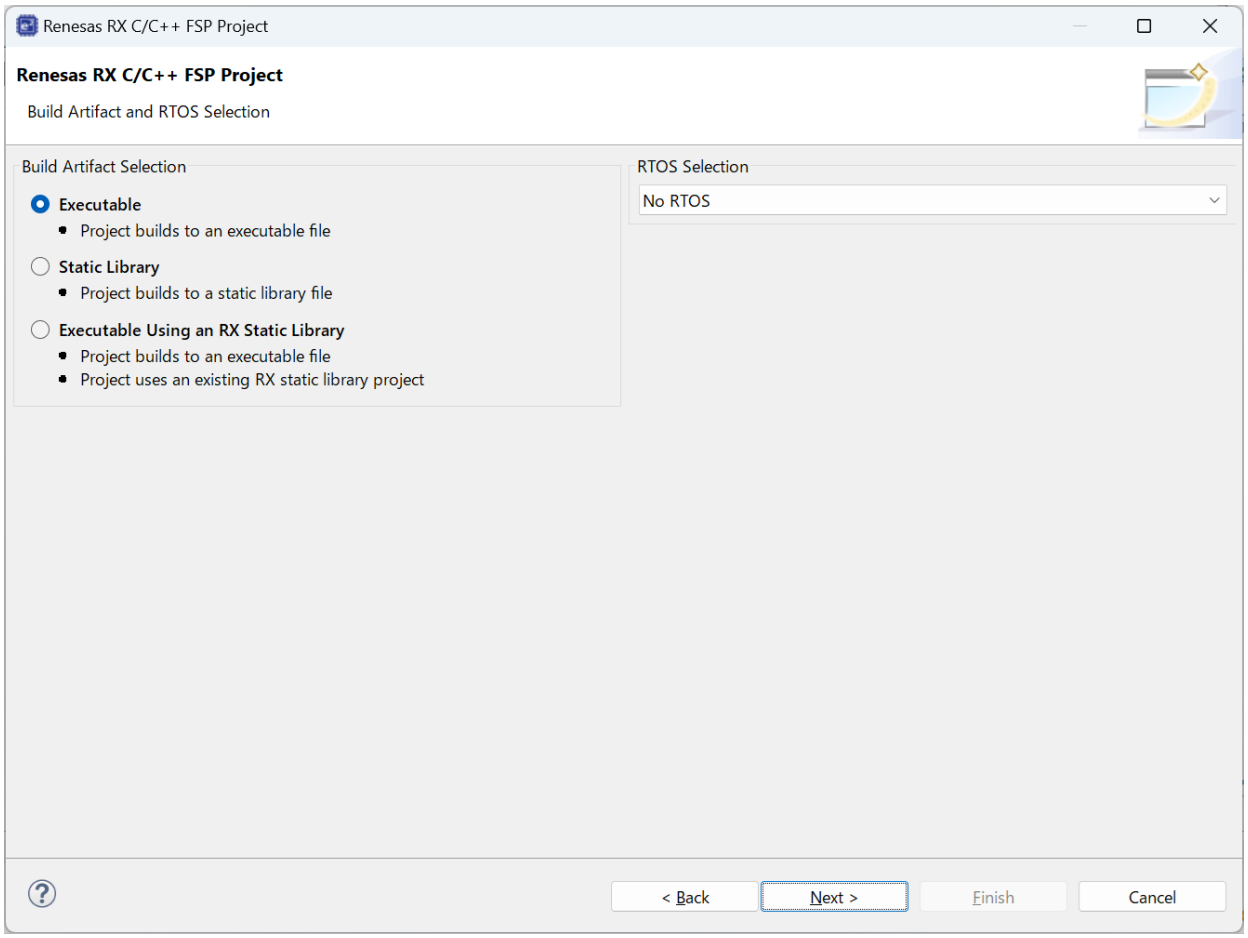


Figure 61: e² studio Project Configuration window (part 3)

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

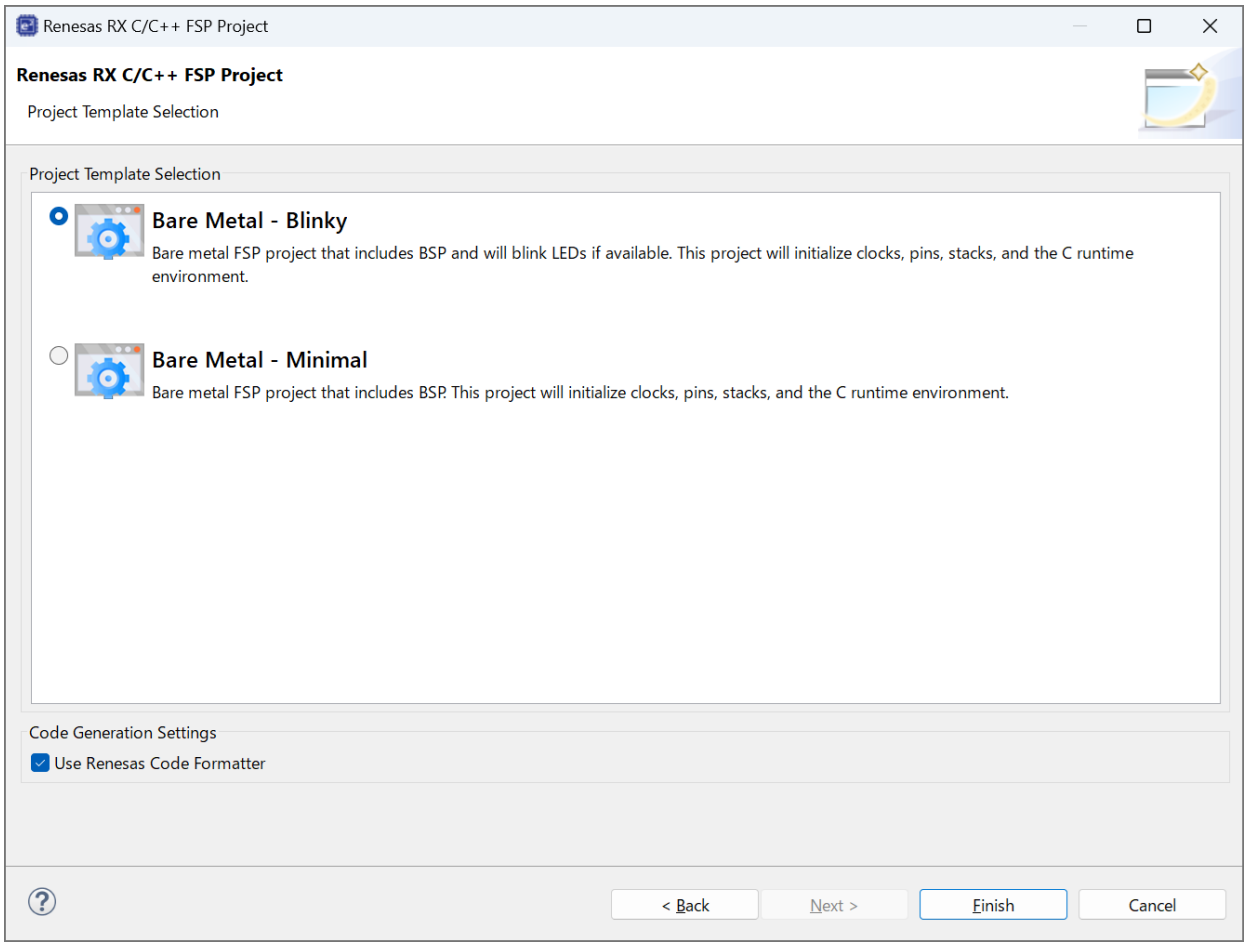


Figure 62: e² studio Project Configuration window (part 4)

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

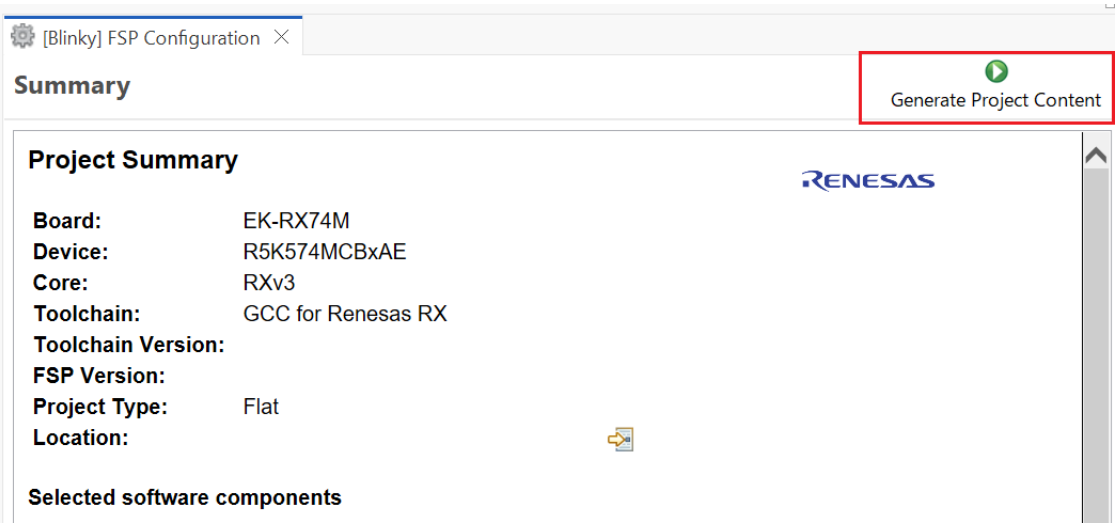


Figure 63: e² studio Project Configuration tab

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

2.3.4.1 Details about the Blinky Configuration

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

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

2.3.4.2 Configuring the Blinky Clocks

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

2.3.4.3 Configuring the Blinky Pins

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

2.3.4.4 Configuring the Parameters for Blinky Components

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

- r_ioport

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

2.3.4.5 Where is main()?

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

2.3.4.6 Blinky Example Code

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

The application performs the following steps:

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

2.3.5 Build the Blinky Project

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

There are three ways to build a project:

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

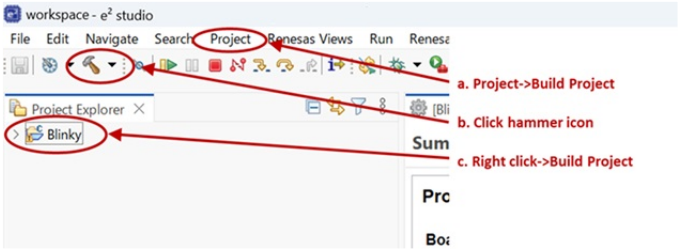


Figure 64: e² studio Project Explorer window

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

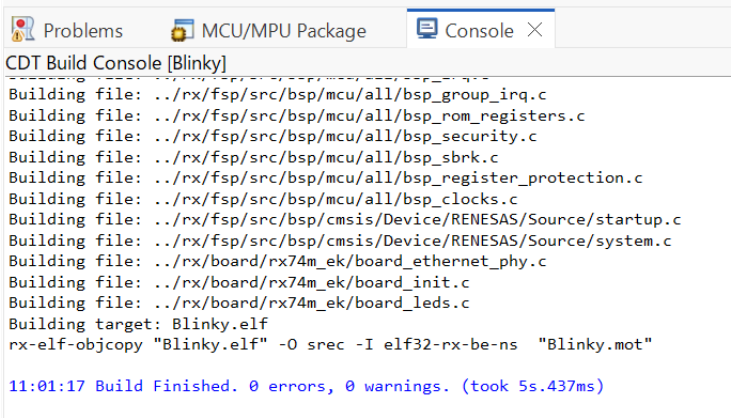


Figure 65: e² studio Project Build console

2.3.6 Debug the Blinky Project

2.3.6.1 Debug prerequisites

To debug the project on a board, you need

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

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

- JTAG debugger

- Built-in boot-loader via UART or USB

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

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

2.3.6.2 Debug steps

To debug the Blinky application, follow these steps:

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

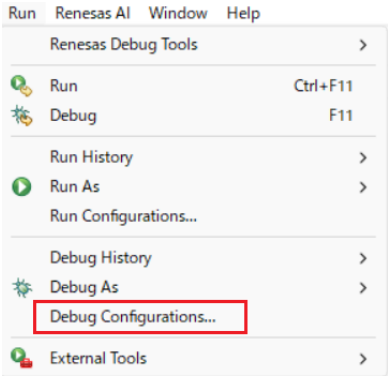


Figure 66: e² studio Debug icon

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

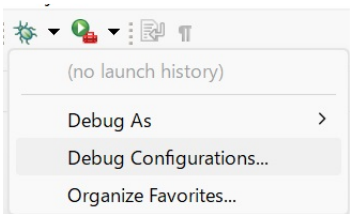


Figure 67: e² studio Debugger Configurations selection option

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

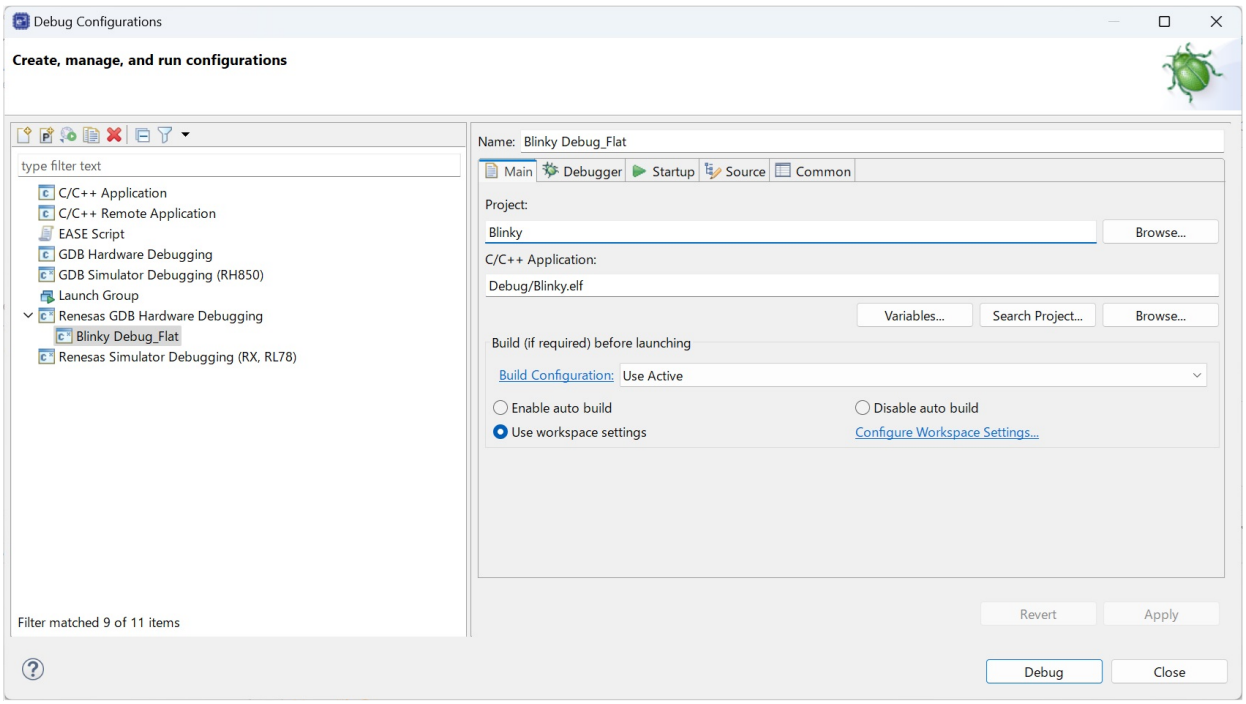
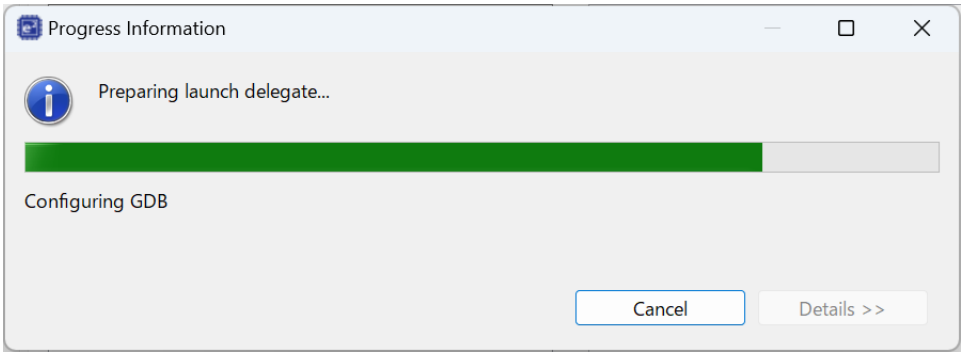


Figure 68: e² studio Debugger Configurations window with Blinky project

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

4. Extracting RX Debug.



2.3.6.3 Details about the Debug Process

In debug mode, e² studio executes the following tasks:

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

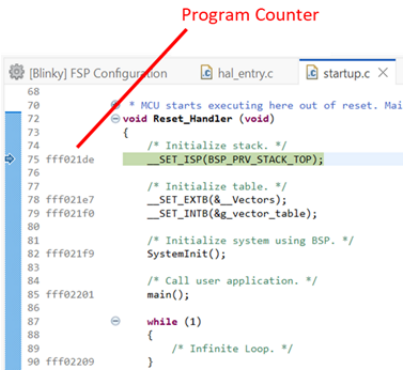


Figure 69: e² studio Debugger memory window

2.3.7 Run the Blinky Project

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

Figure 70: e² studio Debugger Play icon

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

2.4 Tutorial: Using HAL Drivers - Programming the WDT

2.4.1 Application WDT

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

This application makes use of the following FSP modules:

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

2.4.2 Creating a WDT Application Using the RX MCU FSP and e² studio

2.4.2.1 Using FSP and e² studio

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

2.4.2.2 The WDT Application

The flowchart for the WDT application is shown below.

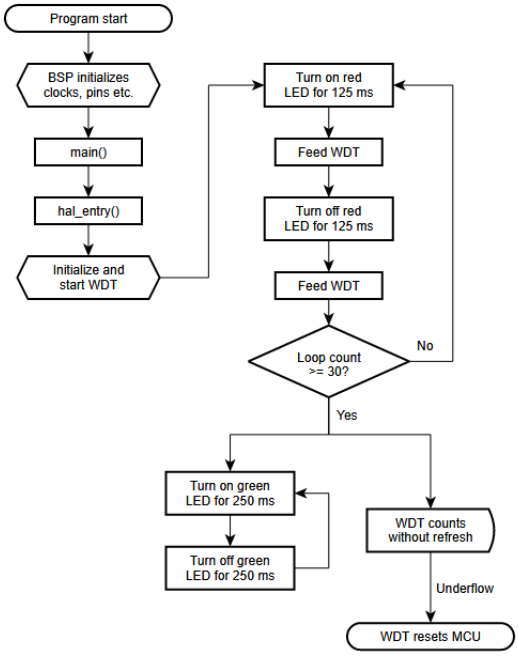


Figure 71: WDT Application flow diagram

2.4.2.3 WDT Application flow

The main sections of the WDT application are:

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

2.4.3 Creating the Project with e² studio

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

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

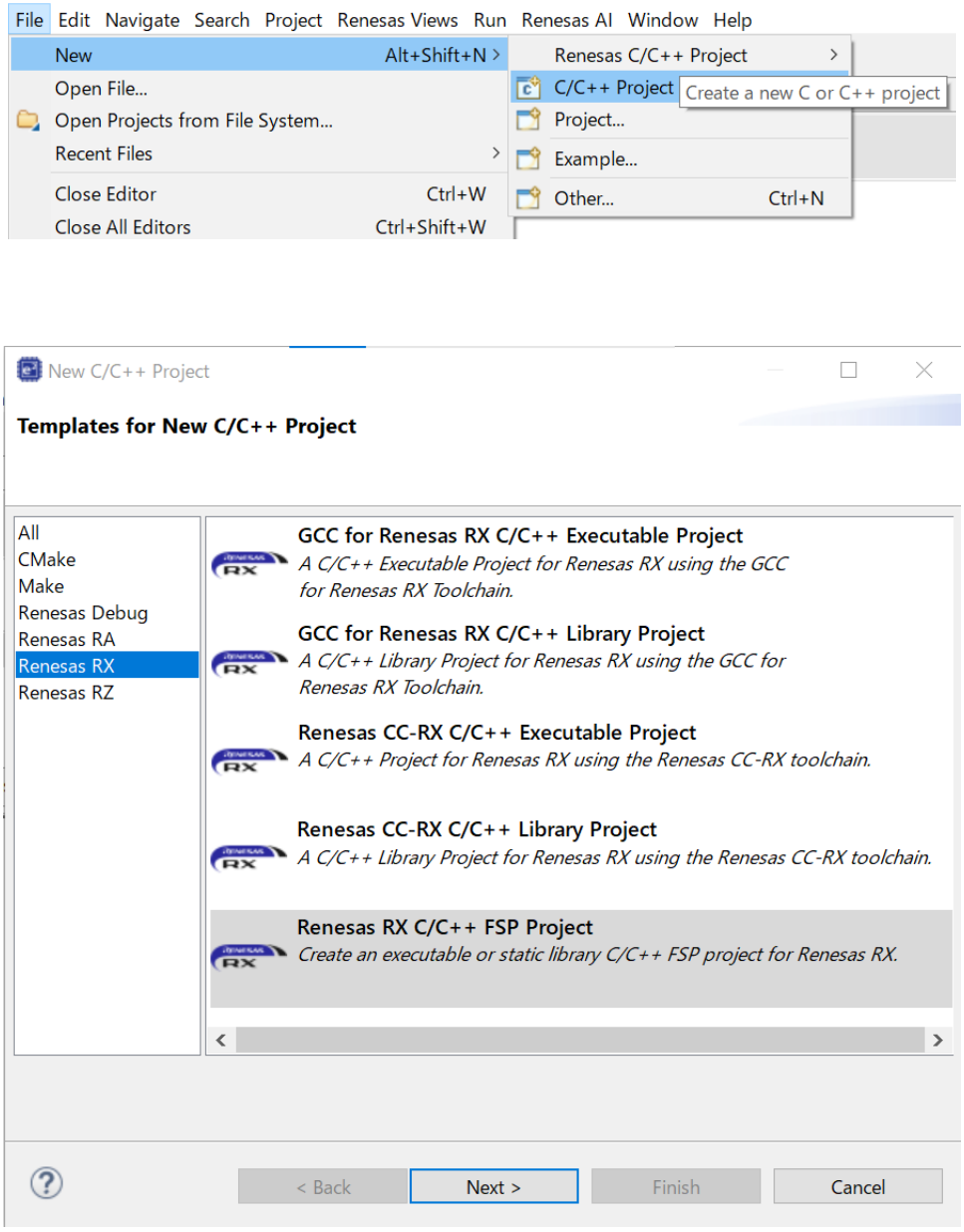


Figure 72: Creating a new project

2. In the e² studio Project **Configuration (RX Project)** window enter a project name, for example, WDT_Application. In addition, select the toolchain. If you want to choose new locations for the project unselect **Use default location**. Click **Next**.

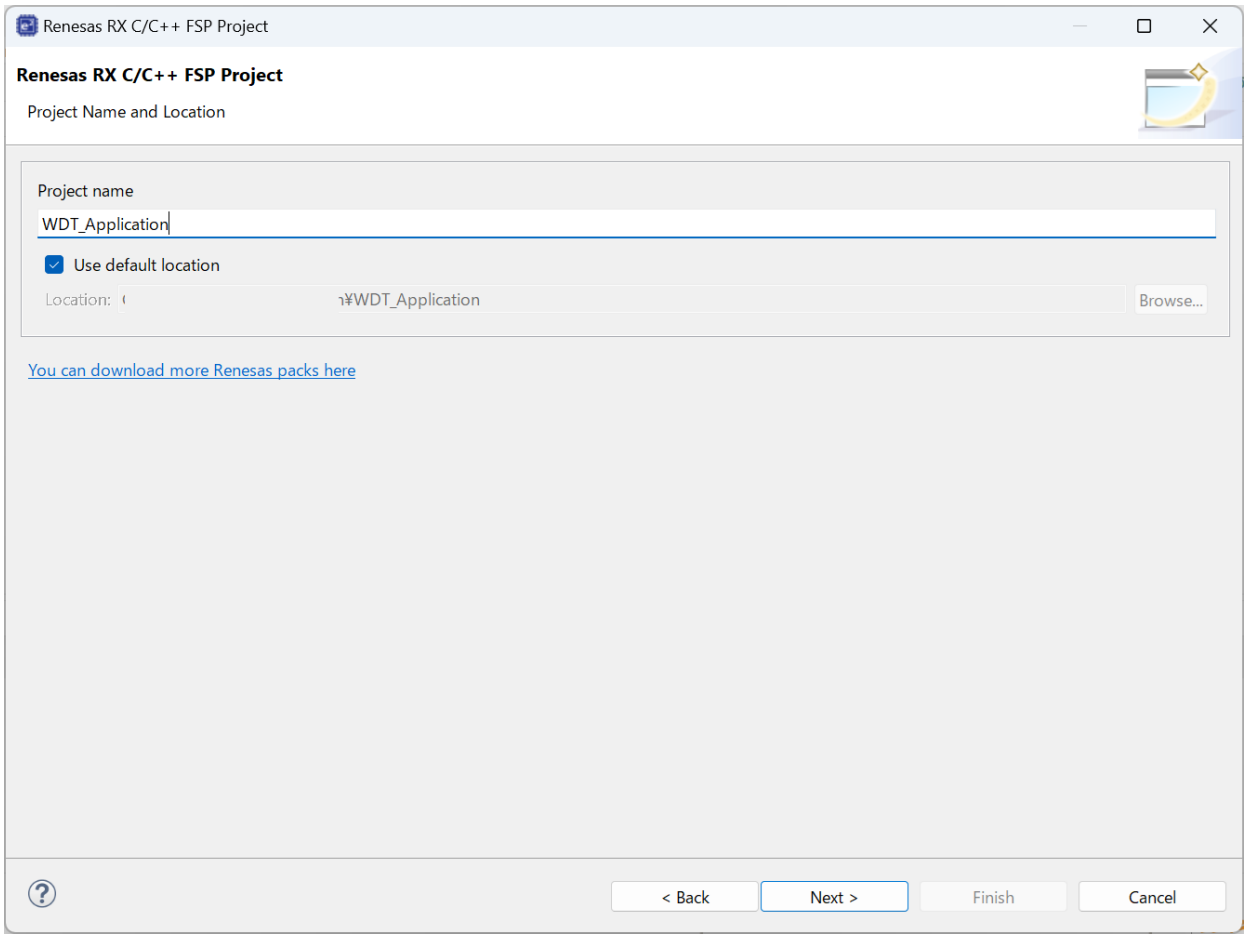


Figure 73: Project configuration (part 1)

3. This application runs on the EK-RX74M board. So, for the **Board** select **EK-RX74M**.

   This will automatically populate the **Device** drop-down with the correct device used on this board. Select the **Toolchain** version. Select **E2 Lite** as the **Debugger**. Click **Next** to configure the project.

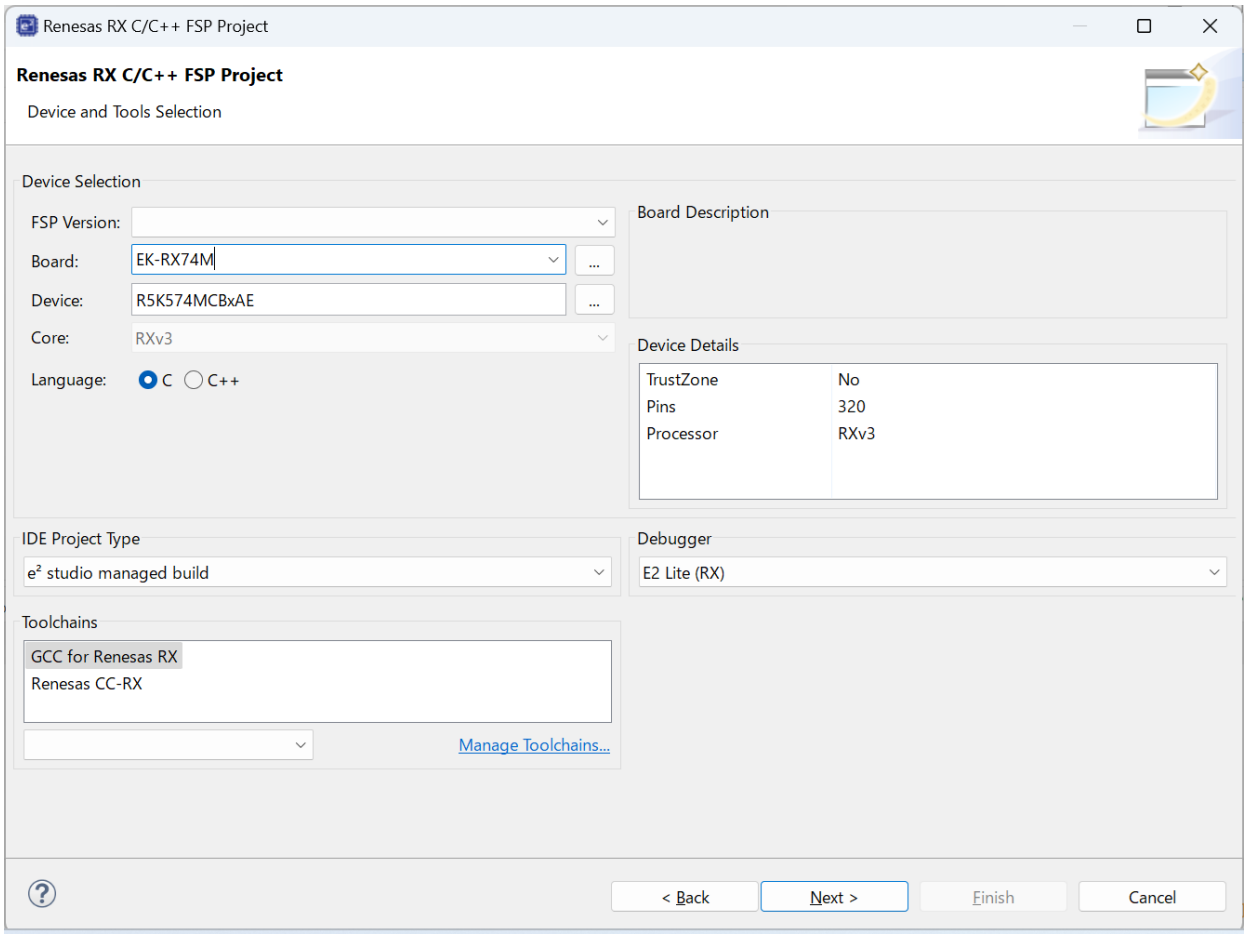


Figure 74: Project configuration (part 2-1)

4. This applicaiton runs without RTOS. So, for the **RTOS selection** select **No RTOS**. For the **Build Artifact Selection** select **Excutable**

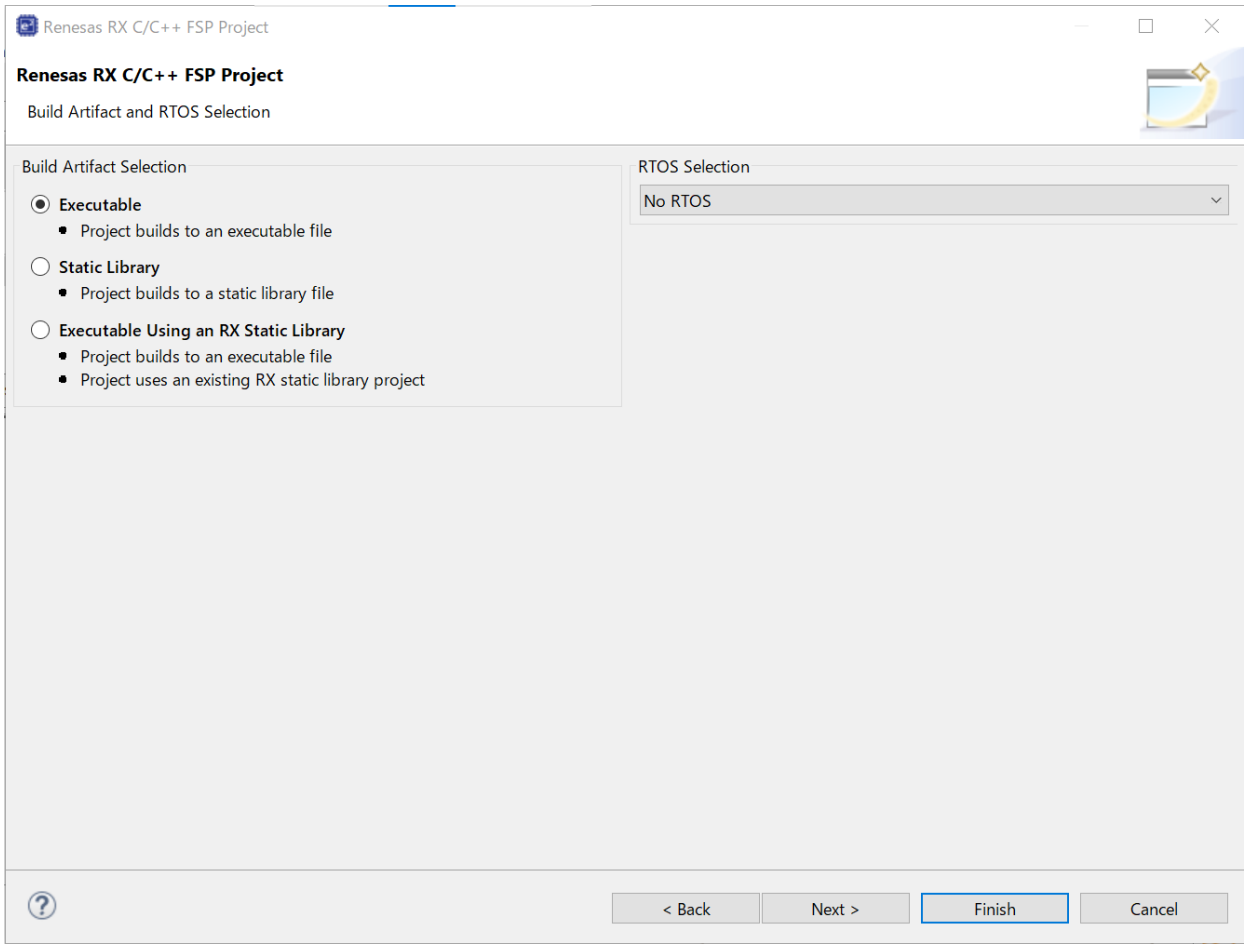


Figure 75: Project configuration (part 2-2)

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

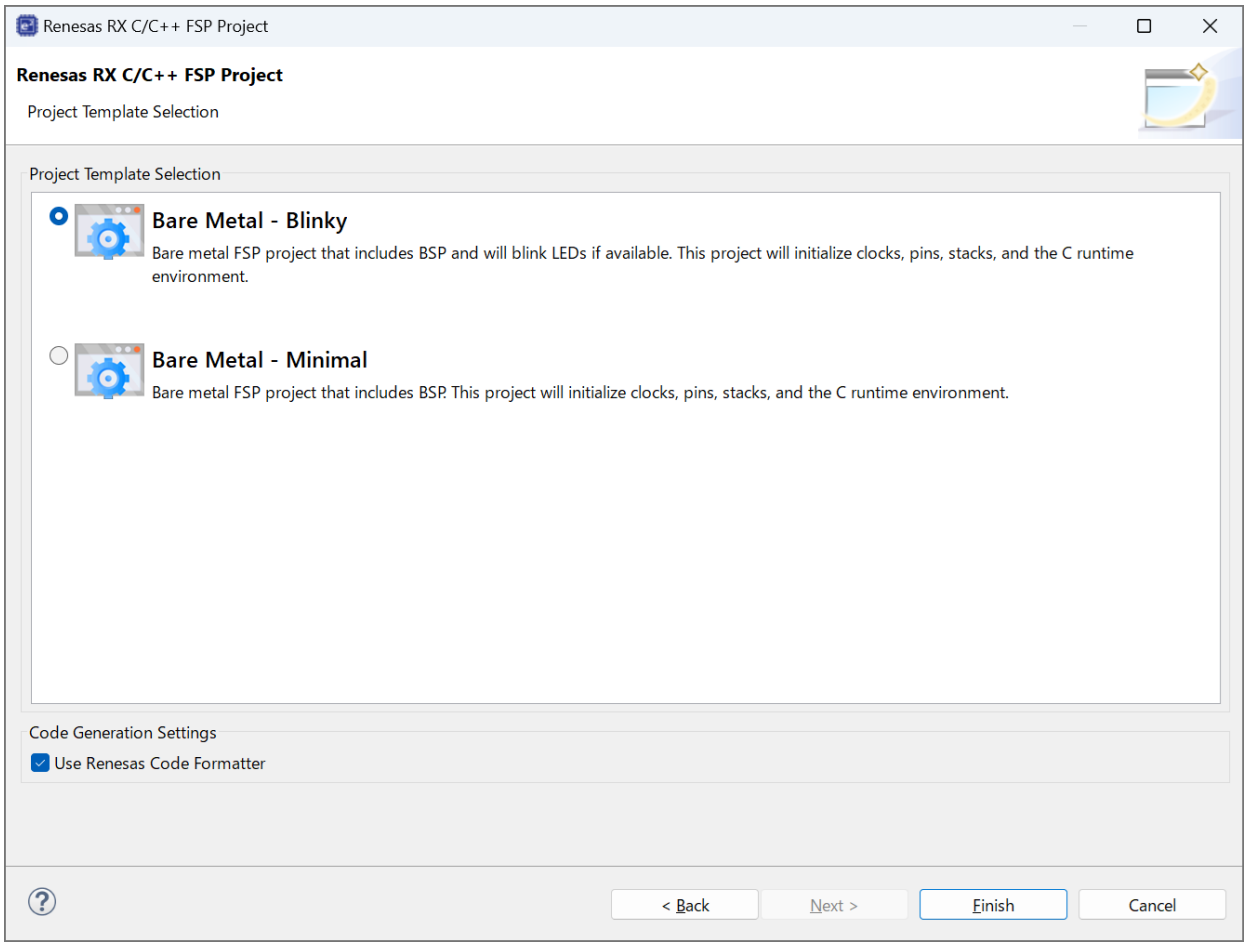


Figure 76: Project configuration (part 3)

5. Click **Finish**.

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

2.4.4 Configuring the Project with e² studio

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

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

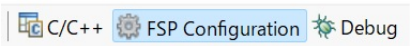


Figure 77: Selecting a perspective

The **C/C++** perspective provides a layout selected for code editing. The **FSP Configuration**

perspective provides elements for configuring a RX MCU project, and the **Debug** perspective provides a view suited for debugging.

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

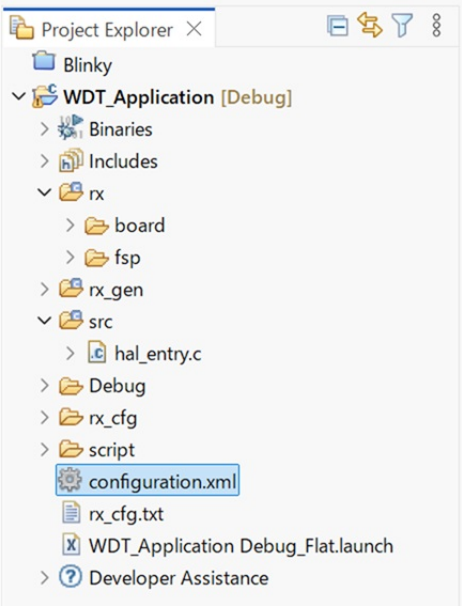


Figure 78: RX MCU Project Configuration Settings

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

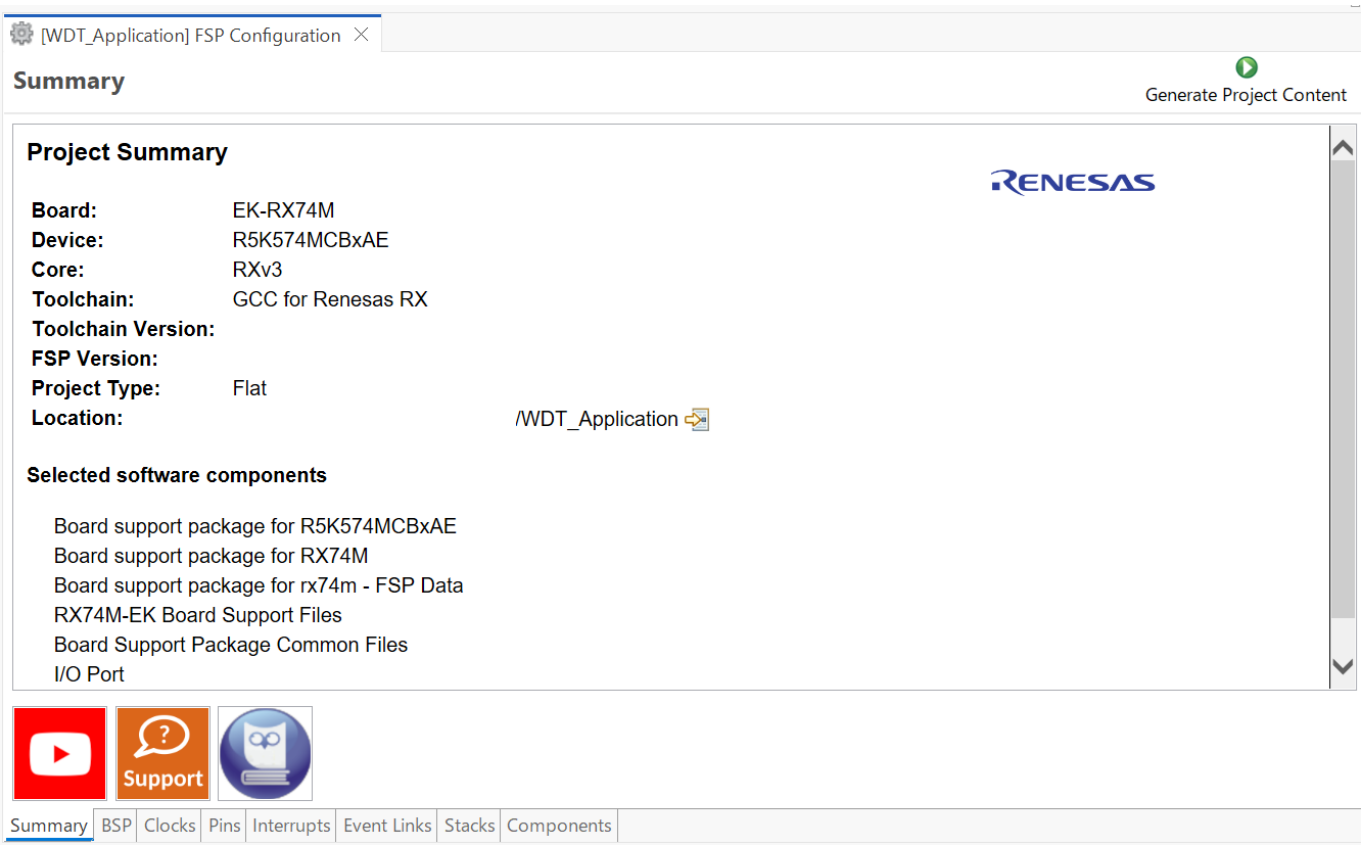


Figure 79: Project Configuration Tabs

2.4.4.1 BSP Tab

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

2.4.4.2 Clocks Tab

The **Clocks** tab presents a graphical view of the clock tree of the device. The drop-down boxes in the GUI enables configuration of the various clocks. The WDT uses PCLKB. The default output frequency for this clock is 75 MHz. Ensure this clock is outputting this value.

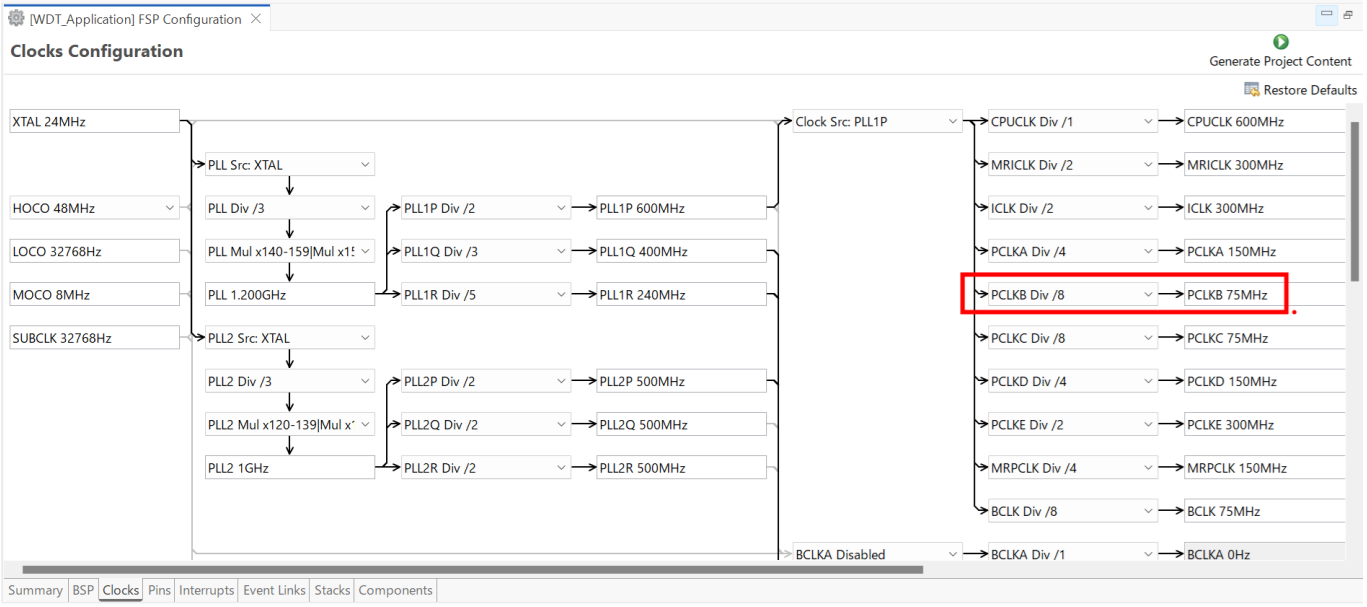


Figure 80: Clock configuration

2.4.4.3 Interrupts Tab

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

2.4.4.4 Event Links Tab

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

2.4.4.5 Pins Tab

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

2.4.4.6 Stacks Tab

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

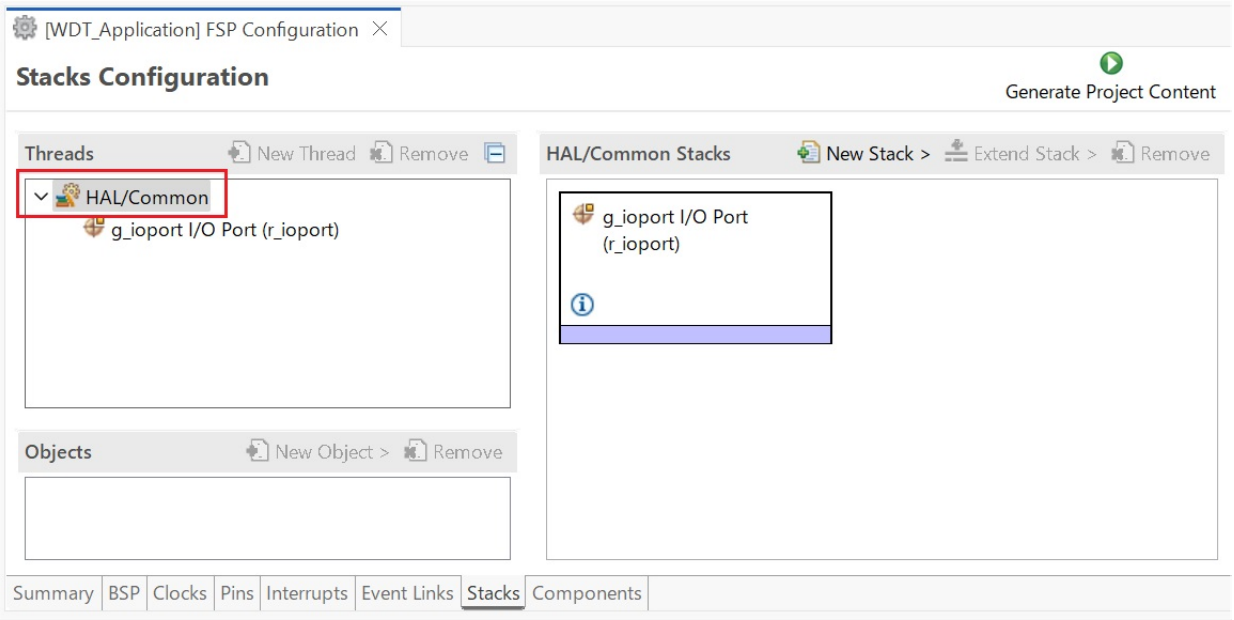


Figure 81: Stacks tab

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

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

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

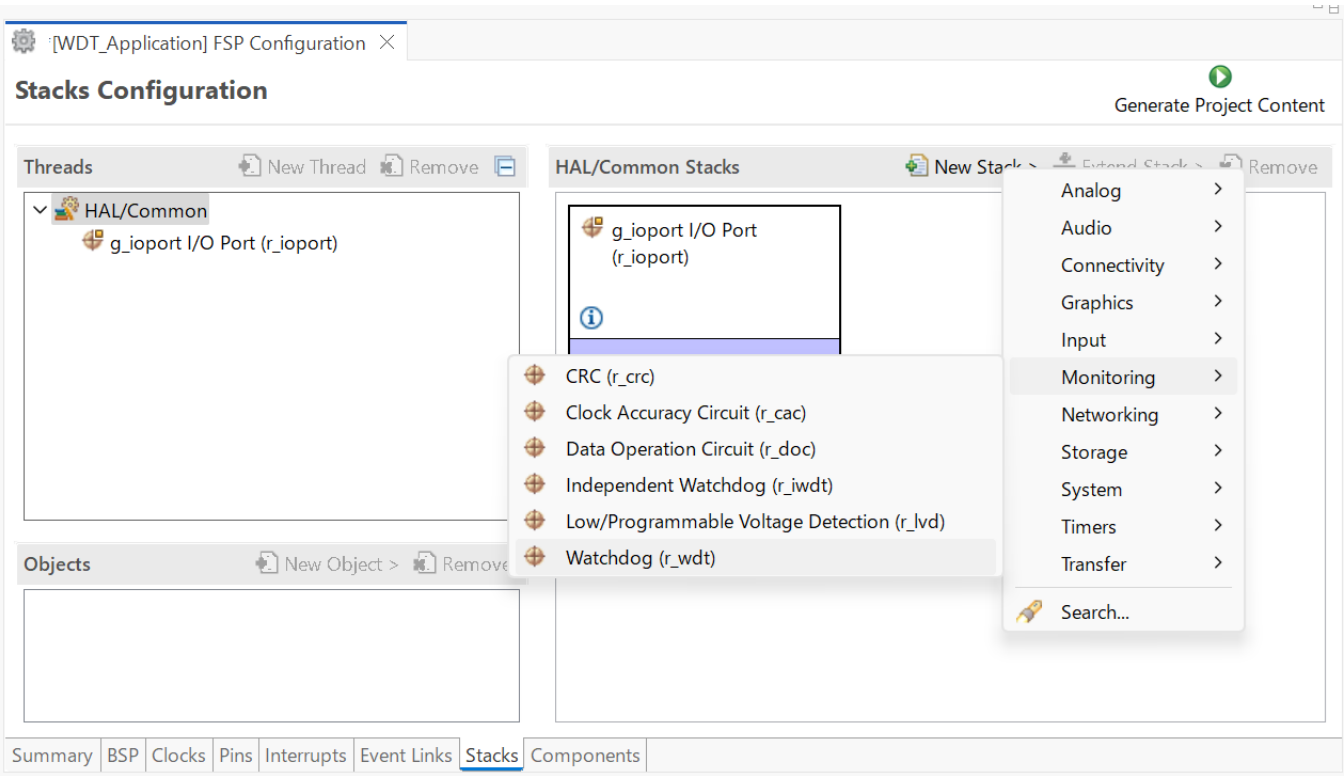


Figure 82: Module Selection

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

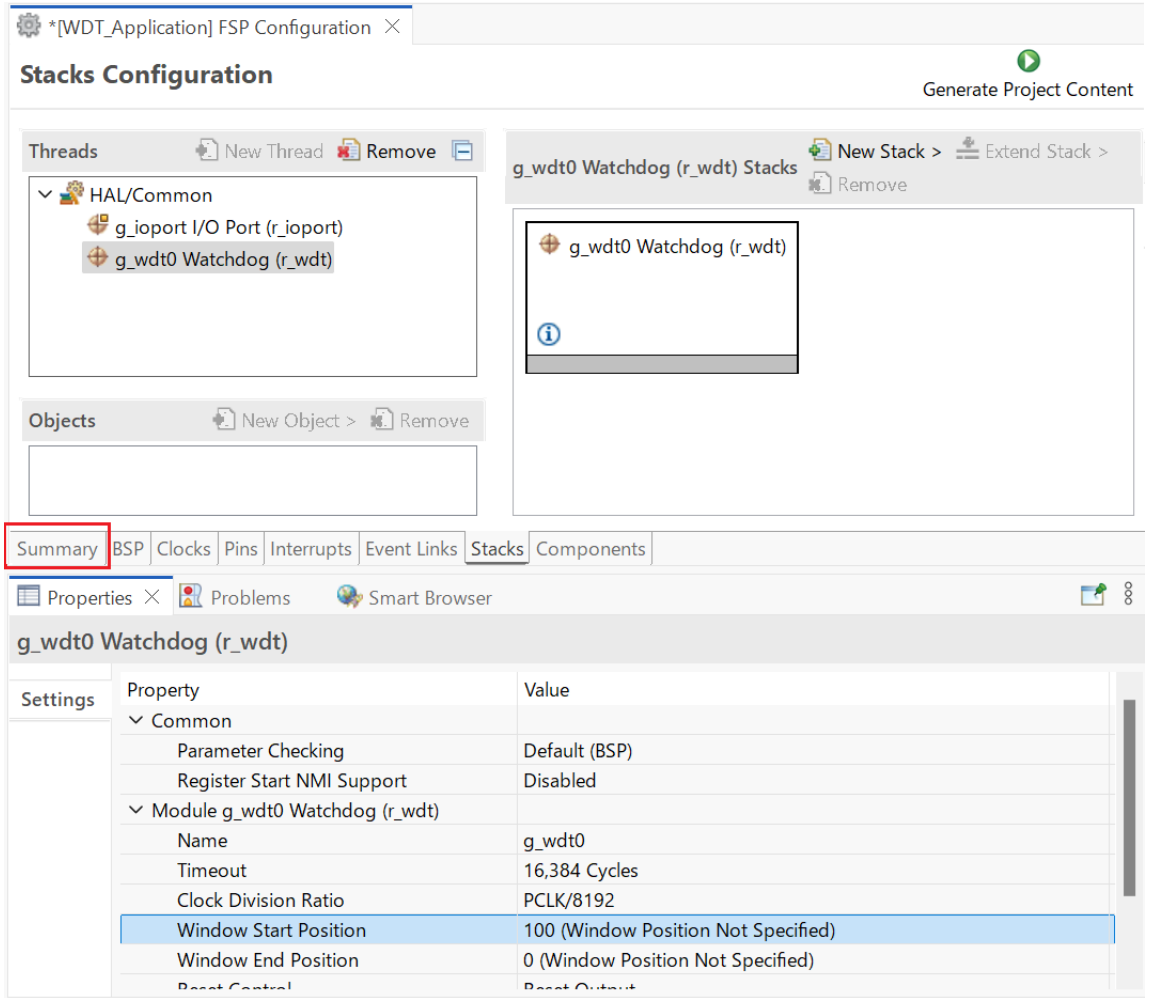


Figure 83: Module Properties

All parameters can be left with their default values.

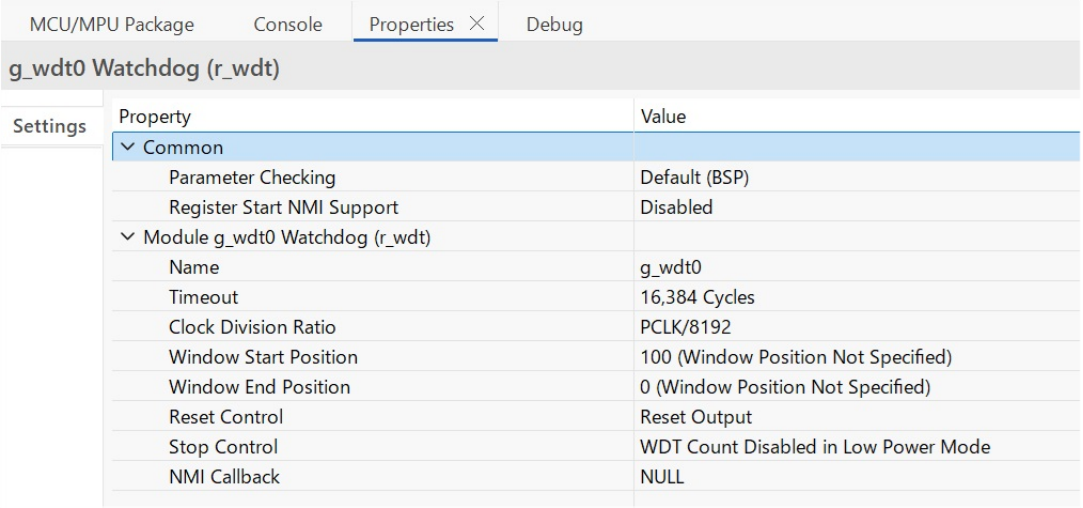


Figure 84: g_wdt WATCHDOG Driver on WDT properties

With PCLKB running at 75 MHz the WDT will reset the device 1.79 seconds after the last refresh.

WDT clock = 75 MHz / 8192 = 9.155 kHz

Cycle time = 1 / 9.155 kHz = 109.23 us

Timeout = 109.23 us x 16384 = 1.79 seconds

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

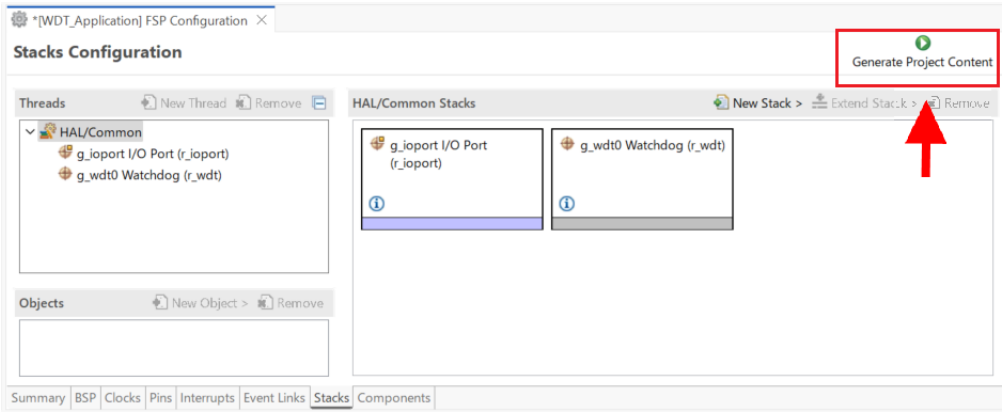


Figure 85: Generate Project Content button

e² studio generates the project files.

2.4.4.7 Components Tab

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

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

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

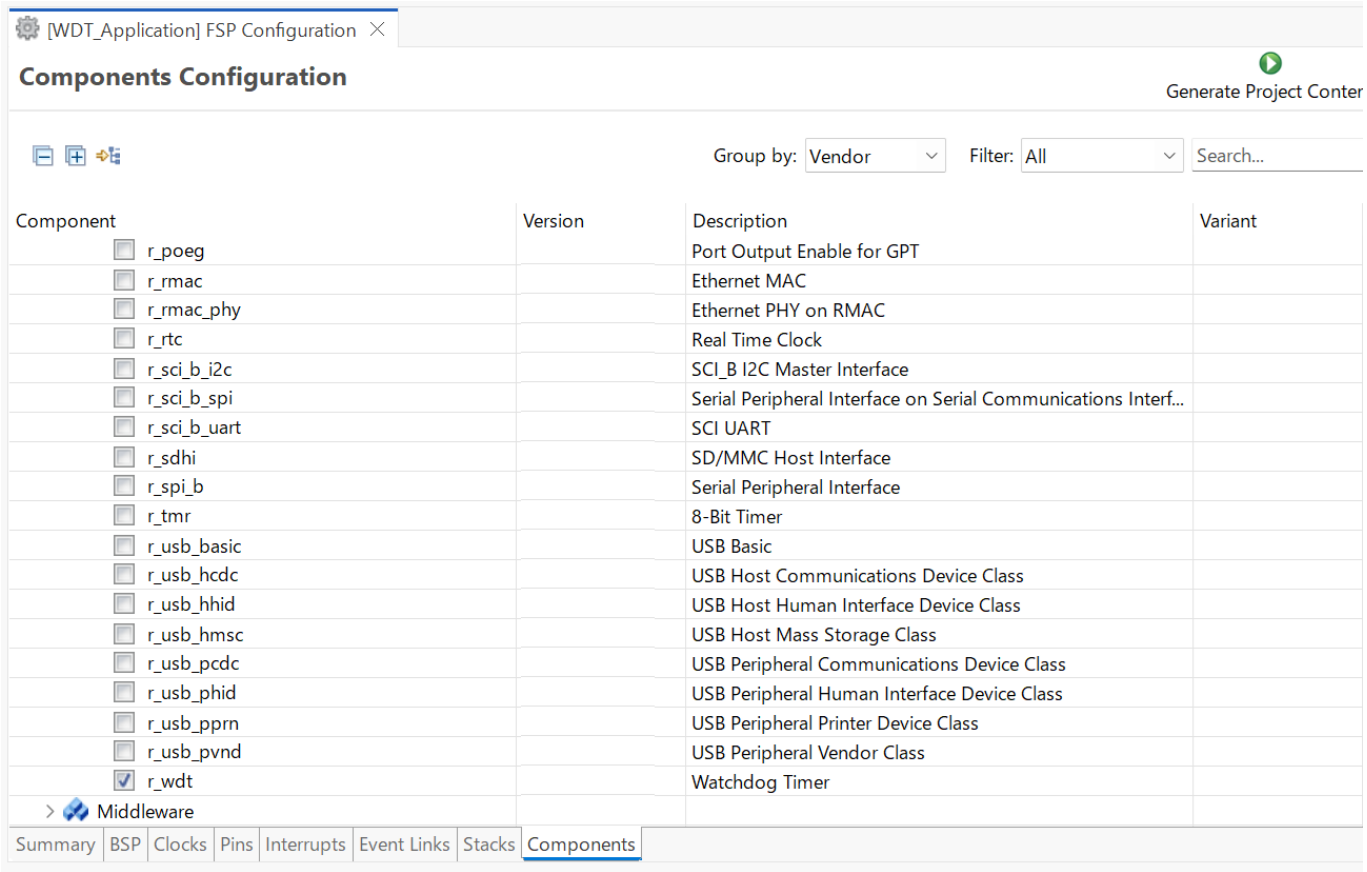


Figure 86: Component Selection

Note

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

2.4.5 WDT Generated Project Files

Clicking the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:

  rx/fsp/src

- r_wdt_api.h created in:

  rx/fsp/inc/api

- r_wdt.h created in:

  rx/fsp/inc/instances

The above files are the standard files for the WDT HAL module. They contain no specific project

contents. They are the driver files for the WDT. Further information on the contents of these files can be found in the documentation for the WDT HAL module.

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

rx_cfg/fsp_cfg/r_wdt_cfg.h

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

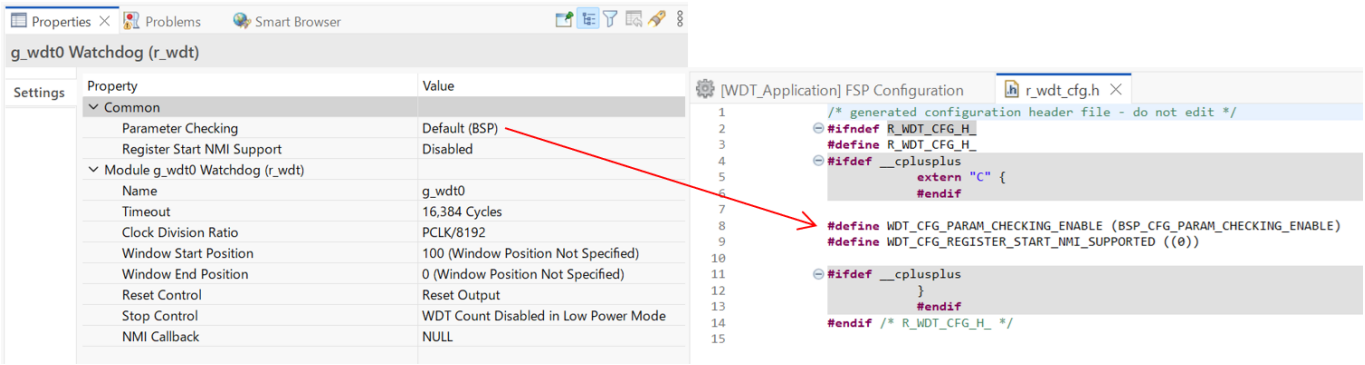


Figure 87: r_wdt_cfg.h contents

Warning

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

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

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

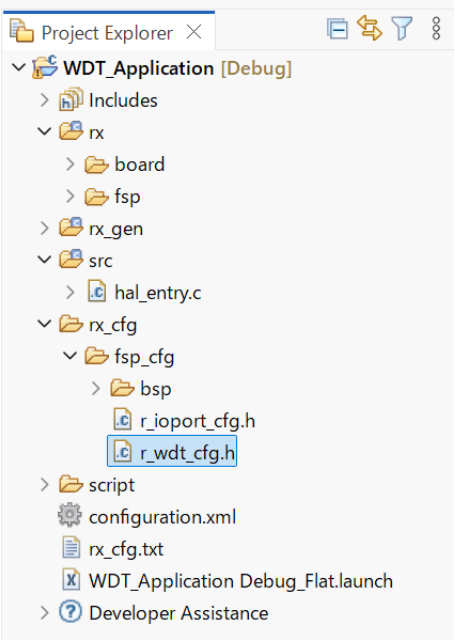


Figure 88: WDT project files

2.4.5.1 WDT hal_data.h

The contents of hal_data.h are shown below.

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

```
FSP_FOOTER

#endif  /* HAL_DATA_H_ */
```

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

Warning

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

2.4.5.2 WDT hal_data.c

The contents of hal_data.c are shown below.

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

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

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

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

2.4.5.3 WDT main.c

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

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

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

2.4.5.4 WDT hal_entry.c

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

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

```
#include "hal_data.h"
#include "bsp_pin_cfg.h"
#include "r_ioport.h"
#define RED_LED_NO_OF_FLASHES 30
#define RED_LED_PIN BSP_IO_PORT_06_PIN_04
#define GREEN_LED_PIN BSP_IO_PORT_08_PIN_07
#define RED_LED_DELAY_MS 125
#define GREEN_LED_DELAY_MS 250
volatile uint32_t delay_counter;
```

```
volatile uint16_t loop_counter;
void R_BSP_WarmStart(bsp_warm_start_event_t event);
/*******************************************************************************
******************************/
void hal_entry (void)
{
 /* TODO: add your own code here */
 /* Open the WDT */
 R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Start the WDT by refreshing it */
 R_WDT_Refresh(&g_wdt0_ctrl);
 /* Flash the red LED and feed the WDT for a few seconds */
 for (loop_counter = 0; loop_counter < RED_LED_NO_OF_FLASHES; loop_counter++)
    {
 /* Turn red LED on */
 R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_LOW);
 /* Delay */
 R_BSP_SoftwareDelay(RED_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Refresh WDT */
 R_WDT_Refresh(&g_wdt0_ctrl);
 R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_HIGH);
 /* Delay */
 R_BSP_SoftwareDelay(RED_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Refresh WDT */
 R_WDT_Refresh(&g_wdt0_ctrl);
    }
 /* Flash green LED but STOP feeding the WDT. WDT should reset the
  * device */
 while (1)
    {
 /* Turn green LED on */
 R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_LOW);
 /* Delay */
 R_BSP_SoftwareDelay(GREEN_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
```

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

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

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

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

The WDT is started and refreshed through the API call:

```
/* Start the WDT by refreshing it */

R_WDT_Refresh(&g_wdt0_ctrl);
```

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

2.4.6 Building and Testing the Project

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

To debug the project

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

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

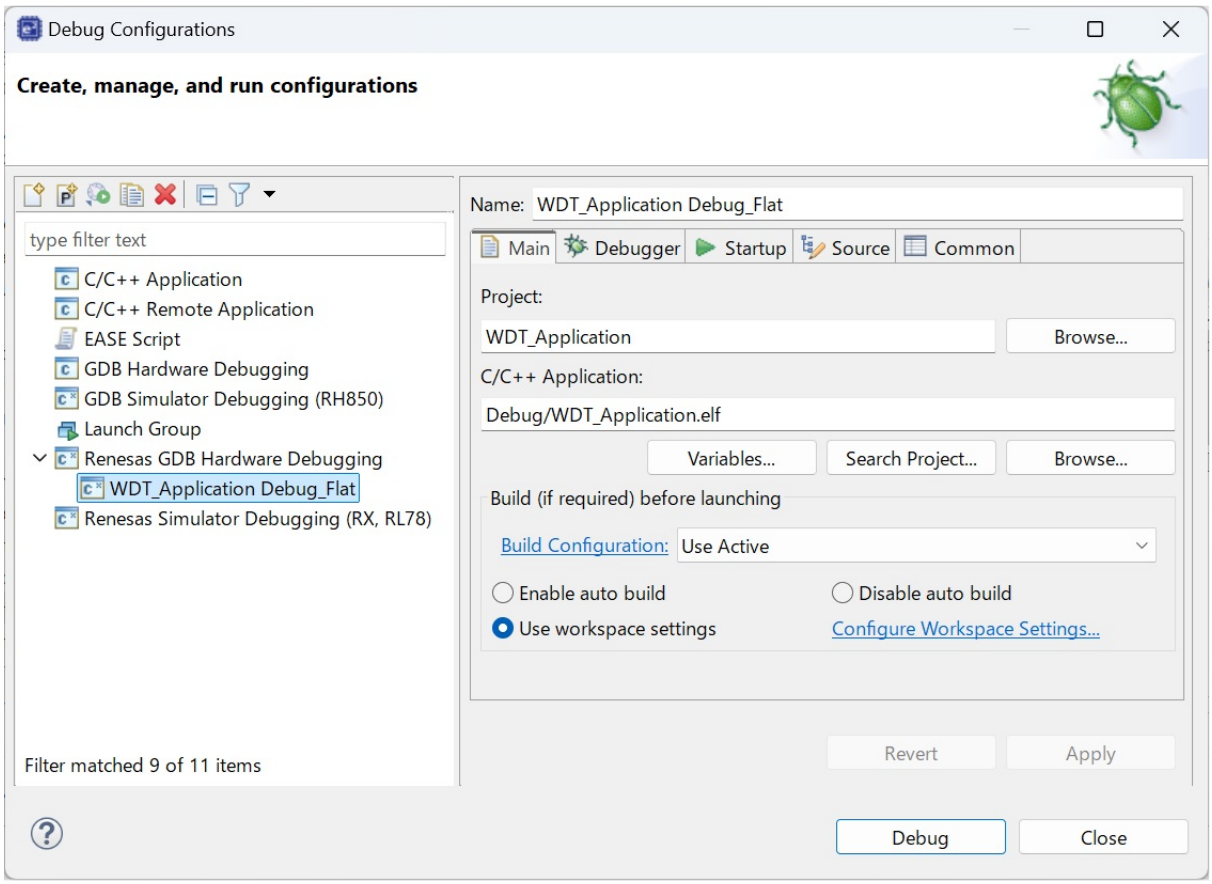
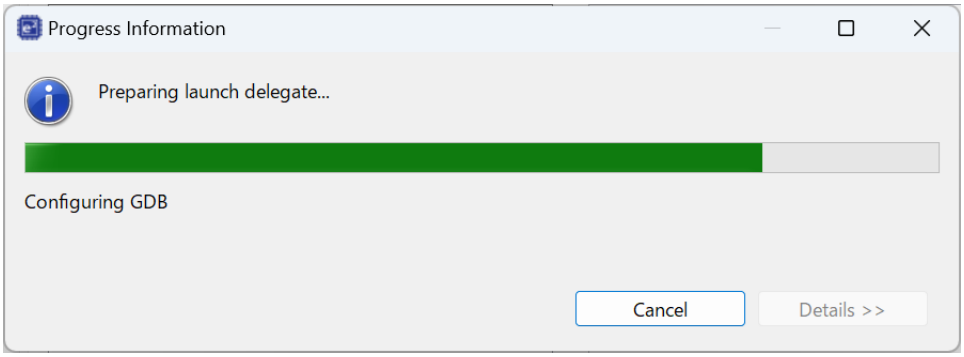


Figure 89: Debug configuration

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



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

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

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

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

2.5 RXSC User Guide for IAR

2.5.1 What is RXSC?

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

The RX Smart Configurator can currently be used with

1. IAR EWRX with IAR toolchain for RX

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

2.5.2 Using RX Smart Configurator with IAR EWRX

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

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

2.5.2.1 Prerequisites

- IAR EWRX installed and licensed. Pleae refer to the Release notes for the version to be installed.
- RXSC and RX FSP Installed

2.5.2.2 Create new RX FSP project

The following steps are required to create an RX FSP project using IAR EWRX, RXSC and RX FSP:

1. Start the RX Smart Configurator.

2. Select the target Device family.

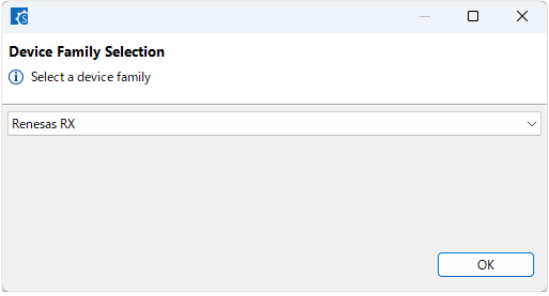


Figure 90: RXSC Device Family Selection

3. Enter a project folder and project name.

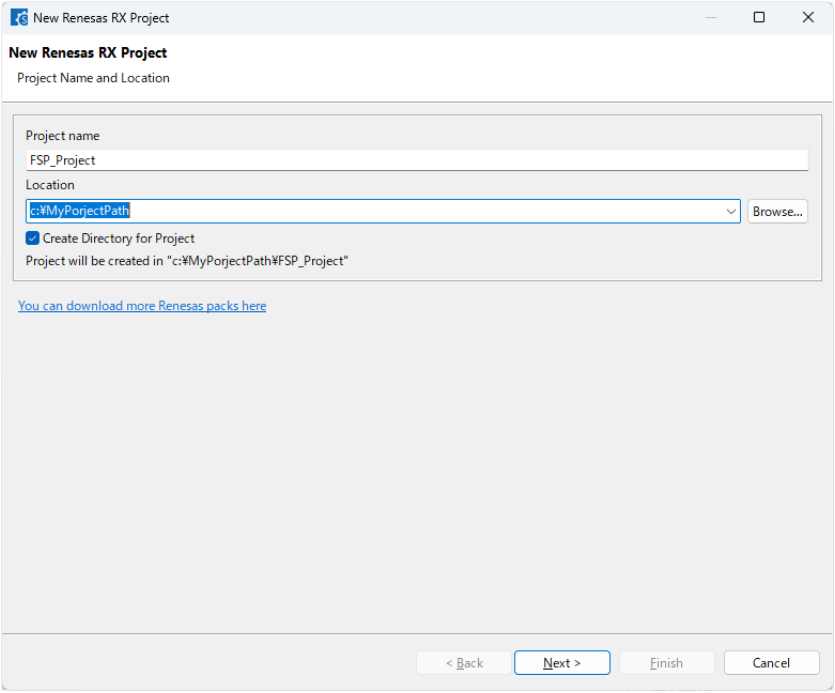


Figure 91: RXSC project settings

4. Select the target device and IDE.

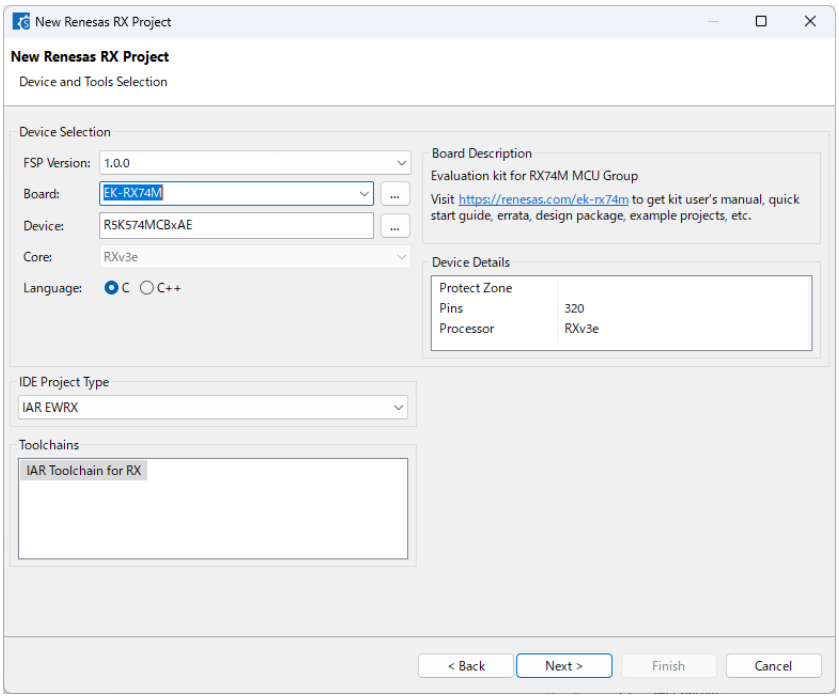


Figure 92: Target device and IDE selection

5. The rest of the project generator and FSP configuration operates the same as e² studio. Refer to the previous sections for details.
6. On completion of FSP configuration, press **Generate Project Content**.
7. A new IAR EWRX project file will be generated in the project path. Double click this file to open IAR EWRX and continue development as usual.
8. To Use RXSC with EWRX, RXSC needs to configured as a tool in EWRX by selecting the menu item **Tools > Configure Tools...**. Select **New** to create a new tool in the dialog shown and add the following information:
    - Menu Text: **RX Smart Configurator**
        a. Command: Select **Browse...** and navigate to rasc.exe in the installed RXSC
        b. Argument: configuration.xml
        c. Initial Directory: $PROJ_DIR$
        d. Tool Available: Always
9. RXSC can now be re-launched from EWRX using the menu item **Tools > RX Smart Configurator**.
10. A Project connection needs to be set up in EWRX to build the project. Select **Project > Add Project Connection** in EWRX and select **IAR Project Connection**. Navigate to the project folder and select buildinfo.ipcf and click **Open**. The project can now build in EWRX.

Chapter 3 FSP Architecture

3.1 FSP Architecture Overview

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

3.1.1 C99 Use

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

3.1.2 Doxygen

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

3.1.3 Weak Symbols

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

3.1.4 Memory Allocation

Dynamic memory allocation through use of the malloc() and free() functions are not used in FSP modules; all memory required by FSP modules is allocated in the application and passed to the module in a pointer. Exceptions are considered only for ports of 3rd party code that require dynamic memory.

Chapter 4 Reference Materials

This section contains miscellaneous reference materials.

Document	Summary
RX Caches	Overview of the RX caches present in RX74M devices and when to perform cache maintenance.
Linker Script Generation Guide	Linker section naming guide

4.1 RX Caches

Note

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

Overview

When using any type of caches in a system, coherency must be considered. A cache may contain data that is different from the backing memory (e.g. SRAM, Flash, etc.), or contain data that is different from another cache, which would make the cache incoherent. Coherency can be maintained through hardware and/or software support.

4.1.1 RX Cache Features

Please refer to the RX Hardware User Manuals for details on the caching feature.

4.1.2 Cache Functions and Macros

Renesas BSP Configuration Macros

Macro	Purpose	Notes

BSP_CFG_DCACHE_ENABLED	Defaults to 0 (disabled). If defined and set to a non-zero value, the FSP startup code in system.c enables the MPU and configures the D-Cache for the XSPI0 CS0/1, XSPI1 CS0/1, SDRAM, Code MRAM, and Internal RAM regions.	This is normally configured in e2 studio under the BSP->Cache settings->Data cache properties panel for the project.
BSP_CFG_ICACHE_ENABLED	Defaults to 0 (disabled). If defined and set to a non-zero value, the FSP startup code in system.c enables the MPU and configures the I-Cache for the XSPI0 CS0/1, XSPI1 CS0/1, SDRAM, Code MRAM, and Internal RAM regions.	This is normally configured in e2 studio under the BSP->Cache settings->Instruction cache properties panel for the project.

Note on Cache Coherency

Software is required to guarantee coherency between the caches and external memory or on-chip SRAM. When multiple bus masters (for example, the CPU and DMAC) share an area in the scope of a cache, flush and write back the cache contents as required.

4.1.3 References

Note

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

Renesas

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

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

RX74M

1. RX74M Product Page
2. RX74M Datasheet
3. RX74M User's Manual: Hardware

BSP Usage Notes

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

4.2 Linker Script Generation Guide

4.2.1 FSP Linker Scripts

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

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

What the System Delivers

- **One policy for three toolchains**: scripts are **generated** from a common policy so behavior is equivalent across GCC for RX, CC-RX and IAR.
- **Per-device awareness**: OFS register locations and available memories (SDRAM/XSPI) are baked in at generation time.
- **Non-invasive placement**: place variables/functions/sections into arbitrary memories **without touching source** via the DDSC **Linker Section Mapping** dialog.
- **Multi-RAM initialization**: support for multiple RAM families (On-chip RAM, LRAM and SDRAM) with correct copy/zero at startup.
- **Fine-grained OFS Registers**: separate, explicit sections for each register region instead of one all-or-nothing block.

Layout & File Model

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

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

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

4.2.1.1 Section Model (high level)

The scripts split into **semantically named outputs** per memory and init behavior. These names are **consistent across toolchains** and expose **Base/Limit symbols** per region.

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

- **FLASH**
  - __flash_vectors$$ – Vector table / Reset handler
  - __flash_readonly$$ – Code & const data (also GOT/EXIDX/ctor arrays: __flash_.got$$, __flash_arm.exidx$$, __flash_{preinit,init,fini}_array$$)
  - __flash_noinit$$ – Reserved/no-load content in flash
- **On-chip RAM**
  - __ram_dtc_vector$$ - The vector table for the dtc peripheral (located at start of ram for alignment restrictions)
  - __ram_from_{ROM REGION}$$ – Initialized data copied from each available persistent source
  - __ram_zero$$ – Zero-initialized bss/tbss
  - __ram_noinit$$ – Non-initialized (e.g., heaps, main stack)
  - Thread helpers (used by threading libraries when present): __ram_thread_stack$$
  - Optional **no-cache** variants (device-dependent): __ram_zero_nocache$$, __ram_noinit_nocache$$
- **Secondary ROM Memories** (such as data flash, xspi)
  - __{ROM Region}_readonly$$ - Initialized space in this region (constants, code)
  - __{ROM Region}_noinit$$ - Uninitialized space in this region that application code can write at runtime
- **Secondary RAM Memories** (such as lram, sdram)
  - __{RAM Region}_from_{ROM Region}$$ - Initialized data copied from each available persistent source
  - __{RAM Region}_zero$$ – Zero-initialized data in this region
  - __{RAM Region}_noinit$$ – Non-initialized data in this region
- **OFS Registers** (fine-grained)
  - __option_setting_{OFS Register}_reg$$ - Fine grained output section for each OFS register

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

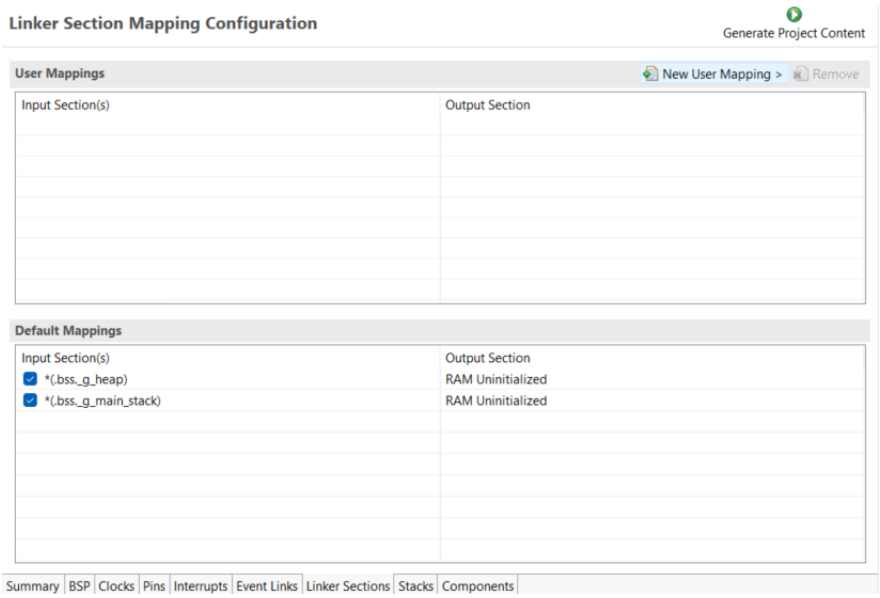
4.2.1.2 Interacting with the Flow

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

Use **DDSC → Linker Section Mapping**

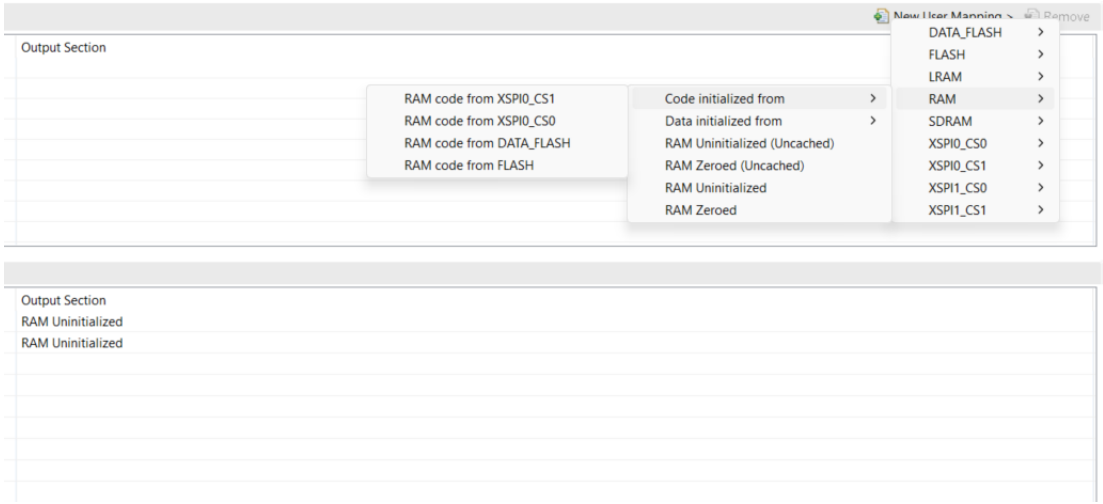
Here you can see typical default mappings for a device.

- The heap and stack are mapped to uninitialized RAM (otherwise they would be placed in zeroed RAM memory)
  - Unchecking the default selection will remove the mapping and allow standard linker placement.
  - These allocations can be overridden to other regions. For example the stack could be placed in LRAM or SDRAM by unchecking the selection and creating a new input section mapping.

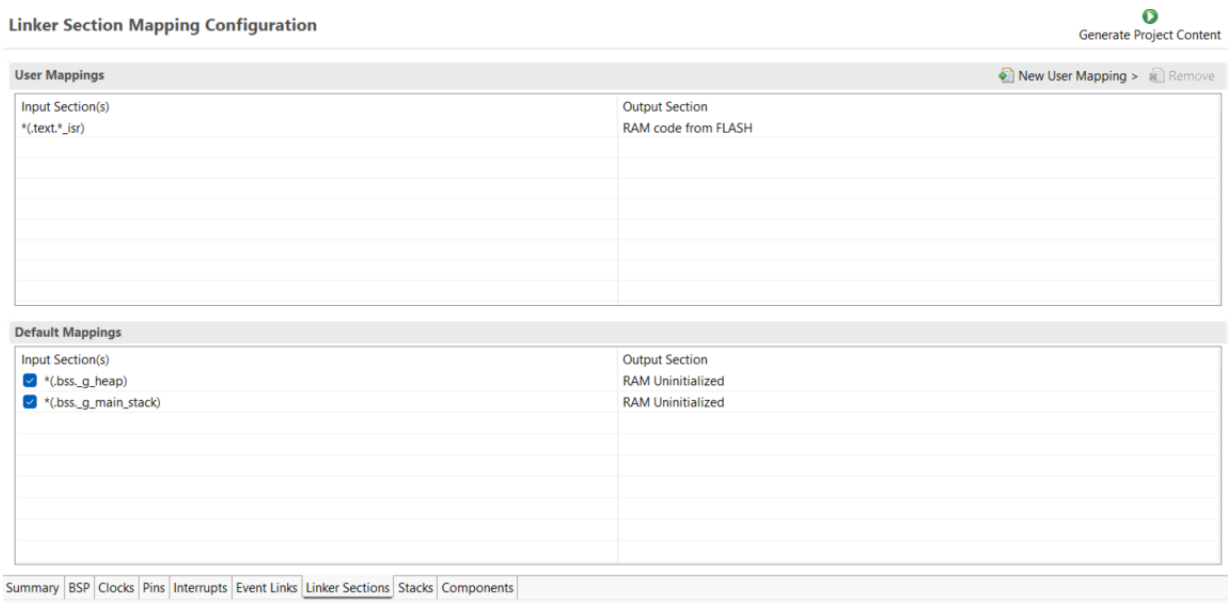


To create a new user mapping, select the desired destination (and the initialization source, if appropriate)

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



This means any code placed into an input section matching *(.text.*_isr) will be persistently stored in FLASH space (in the output elf and srecord) and loaded to RAM by the fsp startup code. This can be useful for time-critical interrupt service routines that benefit from executing from RAM instead of FLASH.



B) Source-level placement

BSP_PLACE_IN_SECTION("<name>") is available. If you use the macro, use the **input names** as defined in the appendix.

4.2.1.3 **Output** Section Translation (for GCC for RX, CC-RX, IAR)

```
output_sections:
  .text: [__flash_vectors$$ (Reset handler & vectors), __flash_readonly$$
(code/rodata)]
  .data: __ram_from_flash$$
  .bss: __ram_zero$$
  .noinit: __ram_noinit$$
  .data_flash: [__data_flash_readonly$$, __data_flash_noinit$$]
  .XSPI_DEVICE_0/.XSPI_DEVICE_1: XSPI family outputs #(e.g.,
__xspi0_cs0_from_flash$$, __xspi0_cs1_readonly$$) – pick readonly/from/zero/noinit as
needed
  .got: __flash_.got$$
  .option_setting_*: fine?grained __option_setting_*_reg$$
```

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

Note: The CC-RX C compiler does not use section names that contain $$.
Therefore, in the linker script for CC-RX, $$ in section names is replaced with _.
When this document shows section names that contain $$, read $$ as _ in the CC-RX environment.

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

4.2.1.4 Behavior differences worth noting

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

4.2.1.5 Examples

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

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

B) Put firmware asset in XSPI1 CS1 ReadOnly

- **Mapping path:** XSPI_CS1 Constant Data
- **Input pattern:** assets.o(.rodata*)
- **Input section placed in:** .xspi1_cs1
- **Output section placed in:** __xspi1_cs1_readonly$$

4.2.2 FAQ

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

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

Q: What is the bsp_linker_info.h file located in the <configuration> folder? A: To support

the seamless support of the new linker process, we must pass information to the source code (such as the source and destination of memory initializers). Unfortunately, the current DDSC tooling has limitations that only allow us to generate a header file to this location. Future releases will likely change the location of this information.

4.2.3 Summary

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

4.2.4 Appendix: Linker Input Sections

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

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

RAM Target Capability Matrix

RAM Class	ROM Init		Zeroed		NoInit	
	Code	Data		NoCache		NoCache
RAM	□	□	□	□	□	□
LRAM	□	□	□		□	
SDRAM	□	□	□	□	□	□

ROM Target Capability Matrix

ROM Class	Code	ReadOnly	NoInit
ROM	□	□	□

4.2.4.1 Memory Class Dictionaries

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

```
ram_class ::= "ram" | "lram" | "sdram"
rom_class ::= "flash" | "data_flash" | "xspi0_cs0" | "xspi0_cs1" | "xspi1_cs0" |
"xspi1_cs1"
```

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

4.2.4.2 RAM-Resident Sections

Zero-Initialized RAM

```
.<ram_class>
```

Used for memory cleared to zero at startup.

Examples:

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

Uninitialized RAM

```
.<ram_class>_noinit

.<ram_class>_noinit_nocache
```

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

Examples:

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

RAM Initialized from ROM

```
.<ram_class>_code_from_<rom_class>

.<ram_class>_from_<rom_class>
```

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

Examples:

- *(.ram_code_from_flash)
- *(.sdram_from_data_flash)
- *(.lram_from_xspi0_cs1)

4.2.4.3 ROM-Resident Sections

Standard ROM Sections

```
.<rom_class>

.<rom_class>_code

.<rom_class>_noinit
```

- .rom_class_code: Executable code

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

Examples:

- *(.flash_code)
- *(.data_flash)
- *(.xspi0_cs0_noinit)

4.2.4.4 Summary

- linker_section names follow consistent, structured patterns.
- RAM and ROM classes reflect physical memory categories and device-specific capabilities.
- Use _nocache suffixes only for memory that must not be cached (e.g., DMA regions).

Chapter 5 API Reference

This section includes the FSP API Reference for the Module and Interface level functions.

▼BSP	Common code shared by FSP drivers
Common Error Codes	
►MCU Board Support Package	The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor
BSP I/O access	This module provides basic read/write access to port pins
BSP SDRAM support	Code that initializes the SDRAMC and SDR SDRAM device memory
▼Modules	Modules are the smallest unit of software available in the FSP. Each module implements one interface
►Analog	Analog Modules
AI	Artificial Intelligence Modules
►Audio	Audio Modules
►Bootloader	Bootloader Modules
CapTouch	CapTouch Modules
►Connectivity	Connectivity Modules
DSP	DSP Modules
►Graphics	Graphics Modules
►Input	Input Modules
►Monitoring	Monitoring Modules
►Motor	Motor Modules
►Networking	Networking Modules
►Power	Power Modules
►RTOS	RTOS Modules
►Security	Security Modules
Sensor	Sensor Modules
►Storage	Storage Modules
►System	System Modules

►Timers	Timers Modules
►Transfer	Transfer Modules
ProtectZone	ProtectZone Modules
▼Interfaces	The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer
►Analog	Analog Interfaces
AI	AI Interfaces
►Audio	Audio Interfaces
CapTouch	CapTouch Interfaces
►Connectivity	Connectivity Interfaces
DSP	DSP Interfaces
►Graphics	Graphics Interfaces
►Input	Input Interfaces
►Monitoring	Monitoring Interfaces
►Motor	Motor Interfaces
►Networking	Networking Interfaces
►Power	Power Interfaces
►Security	Security Interfaces
Sensor	Sensor Interfaces
►Storage	Storage Interfaces
►System	System Interfaces
►Timers	Timers Interfaces
►Transfer	Transfer Interfaces

5.1 BSP

Detailed Description

Common code shared by FSP drivers.

Modules

Common Error Codes

	MCU Board Support Package The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.
	BSP I/O access This module provides basic read/write access to port pins.
	BSP SDRAM support Code that initializes the SDRAMC and SDR SDRAM device memory.

Data Structures

union	fsp_pack_version_t

Macros

#define	FSP_VERSION_MAJOR
#define	FSP_VERSION_MINOR
#define	FSP_VERSION_PATCH
#define	FSP_VERSION_BUILD
#define	FSP_VERSION_STRING
#define	FSP_VERSION_BUILD_STRING

Data Structure Documentation

◆ fsp_pack_version_t

union fsp_pack_version_t		
FSP Pack version structure		
Data Fields		
uint32_t	version_id	Version id
struct version_id_b_s	version_id_b	

Macro Definition Documentation

◆ FSP_VERSION_MAJOR

#define FSP_VERSION_MAJOR
FSP pack major version.

◆ FSP_VERSION_MINOR

#define FSP_VERSION_MINOR
FSP pack minor version.

◆ FSP_VERSION_PATCH

#define FSP_VERSION_PATCH
FSP pack patch version.

◆ FSP_VERSION_BUILD

#define FSP_VERSION_BUILD
FSP pack version build number (currently unused).

◆ FSP_VERSION_STRING

#define FSP_VERSION_STRING
Public FSP version name.

◆ FSP_VERSION_BUILD_STRING

#define FSP_VERSION_BUILD_STRING
Unique FSP version ID.

5.1.1 Common Error Codes

BSP

Detailed Description

All FSP modules share these common error codes.

Macros

#define	FSP_PARAMETER_NOT_USED(p)
#define	FSP_CPP_HEADER
#define	FSP_HEADER
#define	FSP_SECURE_ARGUMENT

Enumerations

enum	fsp_err_t

Macro Definition Documentation

◆ FSP_PARAMETER_NOT_USED

#define FSP_PARAMETER_NOT_USED (p)
This macro is used to suppress compiler messages about a parameter not being used in a function. The nice thing about using this implementation is that it does not take any extra RAM or ROM.

◆ FSP_CPP_HEADER

#define FSP_CPP_HEADER
Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information.

◆ FSP_HEADER

#define FSP_HEADER
FSP Header and Footer definitions

◆ FSP_SECURE_ARGUMENT

#define FSP_SECURE_ARGUMENT
Macro to be used when argument to function is ignored since function call is NSC and the parameter is statically defined on the Secure side.

Enumeration Type Documentation

◆ fsp_err_t

enum fsp_err_t	
Common error codes	
Enumerator	
FSP_ERR_ASSERTION	A critical assertion has failed.
FSP_ERR_INVALID_POINTER	Pointer points to invalid memory location.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.
FSP_ERR_INVALID_CHANNEL	Selected channel does not exist.
FSP_ERR_INVALID_MODE	Unsupported or incorrect mode.
FSP_ERR_UNSUPPORTED	Selected mode not supported by this API.
FSP_ERR_NOT_OPEN	Requested channel is not configured or API not open.
FSP_ERR_IN_USE	Channel/peripheral is running/busy.
FSP_ERR_OUT_OF_MEMORY	Allocate more memory in the driver's cfg.h.
FSP_ERR_HW_LOCKED	Hardware is locked.
FSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
FSP_ERR_OVERFLOW	Hardware overflow.
FSP_ERR_UNDERFLOW	Hardware underflow.
FSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
FSP_ERR_APPROXIMATION	Could not set value to exact result.
FSP_ERR_CLAMPED	Value had to be limited for some reason.
FSP_ERR_INVALID_RATE	Selected rate could not be met.
FSP_ERR_ABORTED	An operation was aborted.
FSP_ERR_NOT_ENABLED	Requested operation is not enabled.
FSP_ERR_TIMEOUT	Timeout error.

FSP_ERR_INVALID_BLOCKS	Invalid number of blocks supplied.
FSP_ERR_INVALID_ADDRESS	Invalid address supplied.
FSP_ERR_INVALID_SIZE	Invalid size/length supplied for operation.
FSP_ERR_WRITE_FAILED	Write operation failed.
FSP_ERR_ERASE_FAILED	Erase operation failed.
FSP_ERR_INVALID_CALL	Invalid function call is made.
FSP_ERR_INVALID_HW_CONDITION	Detected hardware is in invalid condition.
FSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not available on this MCU.
FSP_ERR_INVALID_STATE	API or command not valid in the current state.
FSP_ERR_NOT_ERASED	Erase verification failed.
FSP_ERR_SECTOR_RELEASE_FAILED	Sector release failed.
FSP_ERR_NOT_INITIALIZED	Required initialization not complete.
FSP_ERR_NOT_FOUND	The requested item could not be found.
FSP_ERR_NO_CALLBACK_MEMORY	Non-secure callback memory not provided for non-secure callback.
FSP_ERR_BUFFER_EMPTY	No data available in buffer.
FSP_ERR_INVALID_DATA	Accuracy of data is not guaranteed.
FSP_ERR_INTERNAL	Internal error.
FSP_ERR_WAIT_ABORTED	Wait aborted.
FSP_ERR_FRAMING	Framing error occurs.
FSP_ERR_BREAK_DETECT	Break signal detects.
FSP_ERR_PARITY	Parity error occurs.
FSP_ERR_RXBUF_OVERFLOW	Receive queue overflow.
FSP_ERR_QUEUE_UNAVAILABLE	Can't open s/w queue.
FSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular

	buffer.
FSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
FSP_ERR_TRANSFER_ABORTED	The data transfer was aborted.
FSP_ERR_MODE_FAULT	Mode fault error.
FSP_ERR_READ_OVERFLOW	Read overflow.
FSP_ERR_SPI_PARITY	Parity error.
FSP_ERR_OVERRUN	Overrun error.
FSP_ERR_CLOCK_INACTIVE	Inactive clock specified as system clock.
FSP_ERR_CLOCK_ACTIVE	Active clock source cannot be modified without stopping first.
FSP_ERR_NOT_STABILIZED	Clock has not stabilized after its been turned on/off.
FSP_ERR_PLL_SRC_INACTIVE	PLL initialization attempted when PLL source is turned off.
FSP_ERR_OSC_STOP_DET_ENABLED	Illegal attempt to stop LOCO when Oscillation stop is enabled.
FSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detection status flag is set.
FSP_ERR_OSC_STOP_CLOCK_ACTIVE	Attempt to clear Oscillation Stop Detect Status with PLL/MAIN_OSC active.
FSP_ERR_CLKOUT_EXCEEDED	Output on target output clock pin exceeds maximum supported limit.
FSP_ERR_USB_MODULE_ENABLED	USB clock configure request with USB Module enabled.
FSP_ERR_HARDWARE_TIMEOUT	A register read or write timed out.
FSP_ERR_LOW_VOLTAGE_MODE	Invalid clock setting attempted in low voltage mode.
FSP_ERR_PE_FAILURE	Unable to enter Programming mode.
FSP_ERR_CMD_LOCKED	Peripheral in command locked state.
FSP_ERR_FCLK	

	FCLK must be >= 4 MHz.
FSP_ERR_INVALID_LINKED_ADDRESS	Function or data are linked at an invalid region of memory.
FSP_ERR_BLANK_CHECK_FAILED	Blank check operation failed.
FSP_ERR_HUK_ZEROIZATION	W-HUK zeroization is in progress.
FSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate < reference clock rate.
FSP_ERR_INVALID_RESULT	The result of one or more calculations was +/- infinity.
FSP_ERR_CLOCK_GENERATION	Clock cannot be specified as system clock.
FSP_ERR_INVALID_TIMING_SETTING	Invalid timing parameter.
FSP_ERR_INVALID_LAYER_SETTING	Invalid layer parameter.
FSP_ERR_INVALID_ALIGNMENT	Invalid memory alignment found.
FSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction parameter.
FSP_ERR_INVALID_LAYER_FORMAT	Invalid color format in layer.
FSP_ERR_INVALID_UPDATE_TIMING	Invalid timing for register update.
FSP_ERR_INVALID_CLUT_ACCESS	Invalid access to CLUT entry.
FSP_ERR_INVALID_FADE_SETTING	Invalid fade-in/fade-out setting.
FSP_ERR_INVALID_BRIGHTNESS_SETTING	Invalid gamma correction parameter.
FSP_ERR_JPEG_ERR	JPEG error.
FSP_ERR_JPEG_SOI_NOT_DETECTED	SOI not detected until EOI detected.
FSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED	SOF1 to SOFF detected.
FSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT	Unprovided pixel format detected.
FSP_ERR_JPEG_SOF_ACCURACY_ERROR	SOF accuracy error: other than 8 detected.
FSP_ERR_JPEG_DQT_ACCURACY_ERROR	DQT accuracy error: other than 0 detected.
FSP_ERR_JPEG_COMPONENT_ERROR1	Component error 1: the number of SOF0 header components detected is other than 1, 3, or 4.

FSP_ERR_JPEG_COMPONENT_ERROR2	Component error 2: the number of components differs between SOF0 header and SOS.
FSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED	SOF0, DQT, and DHT not detected when SOS detected.
FSP_ERR_JPEG_SOS_NOT_DETECTED	SOS not detected: SOS not detected until EOI detected.
FSP_ERR_JPEG_EOI_NOT_DETECTED	EOI not detected (default)
FSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR	Restart interval data number error detected.
FSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size error detected.
FSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR	Last MCU data number error detected.
FSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR	Block data number error detected.
FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	User provided buffer size not enough.
FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE	JPEG Image size is not aligned with MCU.
FSP_ERR_CALIBRATE_FAILED	Calibration failed.
FSP_ERR_IIRFA_ECC_1BIT	1-bit ECC error detected
FSP_ERR_IIRFA_ECC_2BIT	2-bit ECC error detected
FSP_ERR_IP_HARDWARE_NOT_PRESENT	Requested IP does not exist on this device.
FSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this device.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this device.
FSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool.
FSP_ERR_ILLEGAL_BUFFER_ADDRESS	Buffer address is out of block memory pool.
FSP_ERR_INVALID_WORKBUFFER_SIZE	Work buffer size is invalid.
FSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
FSP_ERR_TOO_MANY_BUFFERS	Number of buffer is too many.
FSP_ERR_NO_SUBSCRIBER_FOUND	No message subscriber found.

FSP_ERR_MESSAGE_QUEUE_EMPTY	No message found in the message queue.
FSP_ERR_MESSAGE_QUEUE_FULL	No room for new message in the message queue.
FSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.
FSP_ERR_BUFFER_RELEASED	Buffer has been released.
FSP_ERR_D2D_ERROR_INIT	D/AVE 2D has an error in the initialization.
FSP_ERR_D2D_ERROR_DEINIT	D/AVE 2D has an error in the initialization.
FSP_ERR_D2D_ERROR_RENDERING	D/AVE 2D has an error in the rendering.
FSP_ERR_D2D_ERROR_SIZE	D/AVE 2D has an error in the rendering.
FSP_ERR_ETHER_ERROR_NO_DATA	No Data in Receive buffer.
FSP_ERR_ETHER_ERROR_LINK	ETHERC/EDMAC has an error in the Auto-negotiation.
FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE	As a Magic Packet is being detected, and transmission/reception is not enabled.
FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL	Transmit buffer is not empty.
FSP_ERR_ETHER_ERROR_FILTERING	Detect multicast frame when multicast frame filtering enable.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	ETHERC/EDMAC has an error in the phy communication.
FSP_ERR_ETHER_RECEIVE_BUFFER_ACTIVE	Receive buffer is active.
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY is not link up.
FSP_ERR_ETHER_PHY_NOT_READY	PHY has an error in the Auto-negotiation.
FSP_ERR_QUEUE_FULL	Queue is full, cannot queue another data.
FSP_ERR_QUEUE_EMPTY	Queue is empty, no data to dequeue.
FSP_ERR_CTSU_SCANNING	Scanning.
FSP_ERR_CTSU_NOT_GET_DATA	Not processed previous scan data.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.

FSP_ERR_CTSU_DIAG_NOT_YET	Diagnosis of data collected no yet.
FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE	Diagnosis of LDO over voltage failed.
FSP_ERR_CTSU_DIAG_CCO_HIGH	Diagnosis of CCO into 19.2uA failed.
FSP_ERR_CTSU_DIAG_CCO_LOW	Diagnosis of CCO into 2.4uA failed.
FSP_ERR_CTSU_DIAG_SSCG	Diagnosis of SSCG frequency failed.
FSP_ERR_CTSU_DIAG_DAC	Diagnosis of non-touch count value failed.
FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE	Diagnosis of LDO output voltage failed.
FSP_ERR_CTSU_DIAG_OVER_VOLTAGE	Diagnosis of over voltage detection circuit failed.
FSP_ERR_CTSU_DIAG_OVER_CURRENT	Diagnosis of over current detection circuit failed.
FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE	Diagnosis of LDO internal resistance value failed.
FSP_ERR_CTSU_DIAG_CURRENT_SOURCE	Diagnosis of Current source value failed.
FSP_ERR_CTSU_DIAG_SENSCLK_GAIN	Diagnosis of SENSCLK frequency gain failed.
FSP_ERR_CTSU_DIAG_SUCLK_GAIN	Diagnosis of SUCLK frequency gain failed.
FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY	Diagnosis of SUCLK clock recovery function failed.
FSP_ERR_CTSU_DIAG_CFC_GAIN	Diagnosis of CFC oscillator gain failed.
FSP_ERR_CARD_INIT_FAILED	SD card or eMMC device failed to initialize.
FSP_ERR_CARD_NOT_INSERTED	SD card not installed.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low or another operation is ongoing.
FSP_ERR_CARD_NOT_INITIALIZED	SD card was removed.
FSP_ERR_CARD_WRITE_PROTECTED	Media is write protected.
FSP_ERR_TRANSFER_BUSY	Transfer in progress.
FSP_ERR_RESPONSE	Card did not respond or responded with an error.

FSP_ERR_MEDIA_FORMAT_FAILED	Media format failed.
FSP_ERR_MEDIA_OPEN_FAILED	Media open failed.
FSP_ERR_CAN_DATA_UNAVAILABLE	No data available.
FSP_ERR_CAN_MODE_SWITCH_FAILED	Switching operation modes failed.
FSP_ERR_CAN_INIT_FAILED	Hardware initialization failed.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress.
FSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup as a receive mailbox.
FSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is setup as a transmit mailbox.
FSP_ERR_CAN_MESSAGE_LOST	Receive message has been overwritten or overrun.
FSP_ERR_CAN_TRANSMIT_FIFO_FULL	Transmit FIFO is full.
FSP_ERR_WIFI_CONFIG_FAILED	WiFi module Configuration failed.
FSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed.
FSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed.
FSP_ERR_WIFI_INVALID_MODE	API called when provisioned in client mode.
FSP_ERR_WIFI_FAILED	WiFi Failed.
FSP_ERR_WIFI_SCAN_COMPLETE	Wifi scan has completed.
FSP_ERR_WIFI_AP_NOT_CONNECTED	WiFi module is not connected to access point.
FSP_ERR_WIFI_UNKNOWN_AT_CMD	DA16XXX Unknown AT command Error.
FSP_ERR_WIFI_INSUF_PARAM	DA16XXX Insufficient parameter.
FSP_ERR_WIFI_TOO_MANY_PARAMS	DA16XXX Too many parameters.
FSP_ERR_WIFI_INV_PARAM_VAL	DA16XXX Wrong parameter value.
FSP_ERR_WIFI_NO_RESULT	DA16XXX No result.
FSP_ERR_WIFI_RSP_BUF_OVFLW	DA16XXX Response buffer overflow.
FSP_ERR_WIFI_FUNC_NOT_CONFIG	DA16XXX Function is not configured.

FSP_ERR_WIFI_NVRAM_WR_FAIL	DA16XXX NVRAM write failure.
FSP_ERR_WIFI_RET_MEM_WR_FAIL	DA16XXX Retention memory write failure.
FSP_ERR_WIFI_UNKNOWN_ERR	DA16XXX unknown error.
FSP_ERR_CELLULAR_CONFIG_FAILED	Cellular module Configuration failed.
FSP_ERR_CELLULAR_INIT_FAILED	Cellular module initialization failed.
FSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmission failed.
FSP_ERR_CELLULAR_FW_UPTODATE	Firmware is uptodate.
FSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Firmware upgrade failed.
FSP_ERR_CELLULAR_FAILED	Cellular Failed.
FSP_ERR_CELLULAR_INVALID_STATE	API Called in invalid state.
FSP_ERR_CELLULAR_REGISTRATION_FAILED	Cellular Network registration failed.
FSP_ERR_BLE_FAILED	BLE operation failed.
FSP_ERR_BLE_INIT_FAILED	BLE device initialization failed.
FSP_ERR_BLE_CONFIG_FAILED	BLE device configuration failed.
FSP_ERR_BLE_PRF_ALREADY_ENABLED	BLE device Profile already enabled.
FSP_ERR_BLE_PRF_NOT_ENABLED	BLE device not enabled.
FSP_ERR_BLE_ABS_INVALID_OPERATION	Invalid operation is executed.
FSP_ERR_BLE_ABS_NOT_FOUND	Valid data or free space is not found.
FSP_ERR_CRYPTO_CONTINUE	Continue executing function.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware resource busy.
FSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
FSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid index.
FSP_ERR_CRYPTO_SCE_RETRY	Retry.
FSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verify is failed.

FSP_ERR_CRYPTO_SCE_ALREADY_OPEN	HW SCE module is already opened.
FSP_ERR_CRYPTO_NOT_OPEN	Hardware module is not initialized.
FSP_ERR_CRYPTO_UNKNOWN	Some unknown error occurred.
FSP_ERR_CRYPTO_NULL_POINTER	Null pointer input as a parameter.
FSP_ERR_CRYPTO_NOT_IMPLEMENTED	Algorithm/size not implemented.
FSP_ERR_CRYPTO_RNG_INVALID_PARAM	An invalid parameter is specified.
FSP_ERR_CRYPTO_RNG_FATAL_ERROR	A fatal error occurred.
FSP_ERR_CRYPTO_INVALID_SIZE	Size specified is invalid.
FSP_ERR_CRYPTO_INVALID_STATE	Function used in an valid state.
FSP_ERR_CRYPTO_ALREADY_OPEN	control block is already opened
FSP_ERR_CRYPTO_INSTALL_KEY_FAILED	Specified input key is invalid.
FSP_ERR_CRYPTO_AUTHENTICATION_FAILED	Authentication failed.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Failure to Init Cipher.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input date is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function call occurred.
FSP_ERR_CRYPTO_SCE_PASS_1	Private SCE return code.
FSP_ERR_CRYPTO_SCE_PASS_2	Private SCE return code.
FSP_ERR_CRYPTO_SCE_LBIST_CHECK_BUSY	LBIST Check BUSY.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Hardware resource is busy.
FSP_ERR_CRYPTO_RSIP_FATAL	Hardware fatal error or unexpected return.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key type is illegal.
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	Authentication failed.

FSP_ERR_CRYPTO_RSIP_DLMS_ERROR	An error occurred during DLMS processing.
FSP_ERR_CRYPTO_RSIP_LBIST_CHECK_BUSY	LBIST Check BUSY.
FSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common is not opened.
FSP_ERR_CRYPTO_HAL_ERROR	Cryoto HAL module returned an error.
FSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH	Key buffer size is not enough to generate a key.
FSP_ERR_CRYPTO_BUF_OVERFLOW	Attempt to write data larger than what the buffer can hold.
FSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode.
FSP_ERR_MESSAGE_TOO_LONG	Message for RSA encryption is too long.
FSP_ERR_RSA_DECRYPTION_ERROR	RSA Decryption error.
FSP_ERR_SENSOR_INVALID_DATA	Data is invalid.
FSP_ERR_SENSOR_IN_STABILIZATION	Sensor is stabilizing.
FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED	Measurement is not finished.
FSP_ERR_COMMS_BUS_NOT_OPEN	Bus is not open.

5.1.2 MCU Board Support Package

BSP

Functions

fsp_err_t	R_FSP_VersionGet (fsp_pack_version_t *const p_version)
void	Reset_Handler (void)
void	Default_Handler (void)
BSP_WEAK_REFERENCE void	R_BSP_WarmStart (bsp_warm_start_event_t event) __attribute__((noinline))
void	SystemInit (void)
void	R_BSP_CacheEnable (void)

	Enable caches. More...
void	R_BSP_CacheDisable (void) Disable caches. More...
void	R_BSP_CacheClean (void) Clean caches. More...
void	R_BSP_CacheInvalidate (void) Flush caches. More...
void	R_BSP_CacheCleanInvalidate (void) Clean and flush caches. More...
__STATIC_INLINE void	bsp_cache_operate_all (uint8_t operate)
__STATIC_INLINE void	R_BSP_CacheEnableData (void)
__STATIC_INLINE void	R_BSP_CacheDisableData (void)
__STATIC_INLINE void	R_BSP_CacheCleanAllData (void)
__STATIC_INLINE void	R_BSP_CacheInvalidateAllData (void)
__STATIC_INLINE void	R_BSP_CacheCleanInvalidateAllData (void)
uint32_t	R_BSP_SourceClockHzGet (fsp_priv_source_clock_t clock)
__STATIC_INLINE IRQn_Type	R_FSP_CurrentIrqGet (void)
__STATIC_INLINE uint32_t	R_FSP_SystemClockHzGet (fsp_priv_clock_t clock)
__STATIC_INLINE uint32_t	R_FSP_ClockDividerGet (uint32_t ckdivcr)
__STATIC_INLINE bsp_unique_id_t const *	R_BSP_UniqueIdGet (void)
__STATIC_INLINE fsp_err_t	R_BSP_PartNumberGet (bsp_part_number_t *const p_part_number)
void	R_BSP_SoftwareDelay (uint32_t delay, bsp_delay_units_t units)
fsp_err_t	R_BSP_GroupIrqWrite (bsp_grp_irq_nmi_t irq,

	void(*p_callback)(bsp_grp_irq_nmi_t irq))
void	bsp_nmi_handler (void)
bool	R_BSP_GroupIrqCheck (uint8_t irq)
uint32_t	R_BSP_GetCurrentIFlagState (void) Provides the current I flag state on PSW register. More...
void	R_BSP_SetIFlagState (uint32_t interrupt_mask) Set I flag state on PSW register. If this flag is 0, all interrupts are disabled. More...
uint32_t	R_BSP_GetCurrentIpl (void) Get current interrupt priority level from PSW register. More...
void	R_BSP_SetIpl (uint32_t interrupt_mask_level) Set IPL state on PSW register. If the passed interrupt priority level (IPL) is maximum, all interrupts are disabled. If the passed IPL is less than the maximum, interrupts which IPL is above the set value will be processed. More...
uint32_t	R_BSP_IrqIsEnabled (IRQn_Type const irq)
void	R_BSP_IrqStatusClear (IRQn_Type irq)
void	R_BSP_IrqClearPending (IRQn_Type irq)
void	R_BSP_IrqCfg (IRQn_Type const irq, uint32_t priority, void *p_context)
void	R_BSP_IrqEnableNoClear (IRQn_Type const irq)
void	R_BSP_IrqEnable (IRQn_Type const irq)
void	R_BSP_IrqDisable (IRQn_Type const irq)
void	R_BSP_IrqCfgEnable (IRQn_Type const irq, uint32_t priority, void *p_context)
void	R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect)
void	R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect)

Detailed Description

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.

- BSP Features
- BSP Clock Configuration
- System Interrupts
- Group Interrupts
- External and Peripheral Interrupts
- Error Logging
- BSP Weak Symbols
- Warm Start Callbacks
- Sub Clock Stabilization Wait Callback
- SDRAM Initialization
- C Runtime Initialization
- Register Protection
- Software Delay
- Endian
- Limited D-Cache and I-Cache Support
- Cacheable Buffer Placement Example
- Non-Cacheable Buffer Placement Example
- Configuration
- Limitation

Overview

BSP Features

BSP Clock Configuration

All system clocks are set up during BSP initialization based on the settings in bsp_clock_cfg.h. These settings are derived from clock configuration information provided from the FSP Configuration editor **Clocks** tab.

- Clock configuration is performed prior to initializing the C runtime environment to speed up the startup process, as it is possible to start up on a relatively slow (that is, 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.
- The BSP will configure the CMSIS SystemCoreClock variable after clock initialization with the current system clock frequency.

System Interrupts

As RX MCUs are based on the RXv3 architecture, the Interrupt Controller Unit (ICU) handles exceptions and interrupt configuration, prioritization and interrupt masking using the interrupt vector table (INTB) in CPU. In the exception vector table, the individual vectors are allocated to in a 124-byte area where the value indicated by the exception table register (EXTB) is used as the starting address (ExtBase). The reset vector is always allocated to FFFFFFFCh, regardless of the value of the exception vector table. The exceptions except nomal non-maskable interrupt are shown below:

- Undefined instruction exception
- Privileged instruction exception

- Access exception
- Secure exception
- Address exception
- Single-precision floating-point exception
- Reset
- Non-maskable interrupt
- Interrupt
- Unconditional trap

Group Interrupts

Group interrupt is the term used to describe the 13 sources that can trigger the Non-Maskable Interrupt (NMI). When an NMI occurs the NMI Handler examines the NMISR (status register) to determine the source of the interrupt. NMI interrupts take precedence over all interrupts, are usable only as CPU interrupts, and cannot activate the RX peripherals Data Transfer Controller (DTC) or Direct Memory Access Controller (DMAC).

Possible group interrupt sources include:

- IWDT Underflow/Refresh Error
- WDT Underflow/Refresh Error
- Voltage-Monitoring 1 Interrupt
- Voltage-Monitoring 2 Interrupt
- Main Oscillation Stop Detection
- NMI pin Interrupt
- Common Memory Error Interrupt
- Bus Error Interrupt
- Sub Oscillation Stop Detection
- Local Memory Error Interrupt
- Double-precision Floating-point Exception
- Code MRAM Read Error Interrupt
- Extra MRAM Read Error Interrupt

A user may enable notification for one or NMI interrupts by registering a callback using the BSP API function R_BSP_GroupIrqWrite(). When an NMI interrupt occurs, the NMI handler checks to see if there is a callback registered for the cause of the interrupt and if so calls the registered callback function.

External and Peripheral Interrupts

Maskable interrupts are Peripheral interrupts, Software interrupts, Group interrupts, External pin interrupts. User unconfigurable interrupts (software interrupts, group interrupts, external interrupts, peripheral-generated interrupts) are assigned to vector numbers 0 to 127, and user configurable interrupts (software configurable interrupts) are assigned to vector numbers 128 to 255.

By allowing the user to select only the software configurable interrupts they are interested in as interrupt sources, we are able to provide an interrupt service routine that is fast and software configurable interrupt specific.

For example, on other microcontrollers an interrupt vector table might contain a single vector entry for the SCI0 (Serial Communications Interface) peripheral. The interrupt service routine for this would have to check a status register for the 'real' source of the interrupt. In the RX implementation there is a vector entry for each of the SCI0 events that we are interested in.

Nested interrupts are explicitly enabled by setting the PSW.I bit when entering a maskable interrupt

handler.

Note: RX MCUs disable nested interrupts by default. RX FSP enables nested interrupts in software.`

BSP Weak Symbols

You might wonder how the BSP is able to place ISR addresses in the interrupt table without the user having explicitly defined one. All that is required by the BSP is that the interrupt event be given a priority.

This is accomplished through the use of the 'weak' attribute. The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. A weak symbol is one that can be overridden by an accompanying strong reference with the same name. When the BSP declares a function as weak, user code can define the same function and it will be used in place of the BSP function. By defining all possible interrupt sources as weak, the vector table can be built at compile time and any user declarations (strong references) will be used at runtime.

Weak symbols are supported for ELF targets and also for a.out targets when using the GNU assembler and linker.

Note that in CMSIS system.c, there is also a weak definition (and a function body) for the Warm Start callback function R_BSP_WarmStart(). Because this function is defined in the same file as the weak declaration, it will be called as the 'default' implementation. The function may be overridden by the user by copying the body into their user application and modifying it as necessary. The linker identifies this as the 'strong' reference and uses it.

Warm Start Callbacks

As the BSP is in the process of bringing up the board out of reset, there are three points where the user can request a callback. These are defined as the 'Pre Clock Init', 'Post Clock Init' and 'Post C' warm start callbacks.

As described above, this function is already weakly defined as R_BSP_WarmStart(), so it is a simple matter of redefining the function or copying the existing body from CMSIS system.c into the application code to get a callback. R_BSP_WarmStart() takes an event parameter of type bsp_warm_start_event_t which describes the type of warm start callback being made.

This function is not enabled/disabled and is always called for both events as part of the BSP startup. Therefore it needs a function body, which will not be called if the user is overriding it. The function body is located in system.c. To use this function just copy this function into your own code and modify it to meet your needs.

Sub Clock Stabilization Wait Callback

When Sub-Clock oscillator is populated in the application, the BSP startup code waits for some time(sub-clock stabilization time) to allow Sub-clock to stabilize. Enabling the watchdog (IWDT or WDT) timer with Auto start mode in an application using Sub-clock may cause system to generate Reset or NMI interrupt before reaching the application code if watchdog refresh register is not updated in the configured refresh Window. To overcome this problem a weakly defined callback R_BSP_SubClockStabilizeWait() can be overridden. Redefine the callback function in the application code and add code to update the watchdog refresh register. R_BSP_SubClockStabilizeWait() takes a parameter delay of type uint32_t which describes the time in milliseconds required to stabilize the sub-clock.

Sub Clock Stabilization Wait After Reset callback

After Power-On-Reset, the BSP startup code may have to wait for some time(sub-clock stabilization time) to allow Sub-clock to stabilize. This can cause problem to RTC in case device is to be reset frequently. If Sub-Clock registers are not initialized during a reset, BSP actually does not have to wait for Sub-clock to stabilize. To overcome this problem, a weakly defined callback R_BSP_SubClockStabilizeWaitAfterReset() is provided. Reimplement the callback function in the application code to determine whether BSP has to wait for stabilization time based on the current reset type. R_BSP_SubClockStabilizeWaitAfterReset() takes a parameter delay of type uint32_t which describes the time in milliseconds required to stabilize the sub-clock.

SDRAM Initialization

The BSP provides support for usage of external SDRAM modules on MCUs with SDRAM support. SDRAM is enabled and configured in the BSP tab of the FSP configuration editor. The default location for initialization is in the 'Post C' warm start callback. If required, the call to R_BSP_SdramInit() can be moved anywhere after clock and pin initialization, but it must only be called once after reset.

Before entering Software Standby or Deep Software Standby, the user must call R_BSP_SdramSelfRefreshEnable() to change from Auto-Refresh to Self-Refresh in order to preserve data during the low power state. No SDRAM access is allowed after this function is called. The user must not place FSP code or data or their wakeup interrupt handling functions into SDRAM to ensure there are no issues around transitions into and out of Software Standby since SDRAM access will be disabled then and trigger a fault if access is requested.

When resuming from Software Standby, the user must call R_BSP_SdramSelfRefreshDisable() to change from Self-Refresh to Auto-Refresh and restore SDRAM access.

When resuming operation after Deep Software Standby or another situation where there is already data present in the SDRAM modules that must be preserved, the user must call R_BSP_SdramInit(false) before pin initialization and then call R_BSP_SdramSelfRefreshDisable() after pins have been configured in order to resume operations with the SDRAM.

C Runtime Initialization

This BSP configuration allows the user to skip the FSP C runtime initialization code by setting the "C Runtime Initialization" to "Disabled" on the BSP tab of the FSP Configuration editor. Disabling this option is useful in cases where a non-standard linker script is being used or other modifications to the runtime initialization are desired. If this macro is disabled, the user must use the 'Post Clock Init' event from the warm start (described above) to run their own runtime initialization code.

Heap Allocation

The relatively low amount of on-chip SRAM available and lack of memory protection in an MCU means that heap use must be very carefully controlled to avoid memory leaks, overruns and attempted overallocation. Further, many RTOSes provide their own dynamic memory allocation system. For these reasons the default heap size is set at 0 bytes, effectively disabling dynamic memory. If it is required for an application setting a positive value to the "Heap size (bytes)" option in the RX Common configurations on the **BSP** tab will allocate a heap.

Note

When using printf/sprintf (and other variants) to output floating point numbers a heap is required. A minimum size of 0x1000 (4096) bytes is recommended when starting development in this case.

Error Logging

When error logging is enabled, the error logging function can be redefined on the command line by

defining FSP_ERROR_LOG(err) to the desired function call. The default function implementation is FSP_ERROR_LOG(err)=fsp_error_log(err, **FILE**, **LINE**). This implementation uses the predefined macros **FILE** and **LINE** to help identify the location where the error occurred. Removing the line from the function call can reduce code size when error logging is enabled. Some compilers may support other predefined macros like **FUNCTION**, which could be helpful for customizing the error logger.

Register Protection

The BSP register protection functions utilize reference counters to ensure that an application which has specified a certain register and subsequently calls another function doesn't have its register protection settings inadvertently modified.

Each time R_BSP_RegisterProtectDisable() is called, the respective reference counter is incremented.

Each time R_BSP_RegisterProtectEnable() is called, the respective reference counter is decremented.

Both functions will only modify the protection state if their reference counter is zero.

```
/* Enable writing to protected CGC registers */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* Insert code to modify protected CGC registers. */
/* Disable writing to protected CGC registers */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

Option-Setting Memory

Option-setting memory includes OFS registers (e.g. OFS1), block protections settings (BPS/PBPS), and other device-dependent non-volatile options. Option-setting memory is MCU specific, and not all MCUs implement all option-setting registers. Option-setting configurations available on the selected device are configurable in the BSP properties. These configurations are placed in sections to be loaded at the required flash address by the linker.

OFS registers can be individually enabled or disabled. When an OFS register is disabled, its flash address will not be erased or programmed. The existing settings on the device will not be changed. For safety purpose, OFS registers that could limit or disable debug or ROM write access (like BANKSEL, BPS, PBPS etc.) are disabled by default. Please note:

- Secure OFS registers can only be set by Secure/Flat projects. Secure OFS register settings are ignored in Non-Secure projects.
- For MCUBoot use case, OFS registers will only be set by bootloader projects.

All *_SEL registers default to allowing both Secure and Non-Secure access unless otherwise noted here. If block protection is configured in a Secure project, the BSP sets the corresponding configuration to Secure access only by updating the corresponding *_SEL register. Similarly, the PVD related settings in the OFSn_SEL registers are automatically set to Secure if the corresponding PVD monitor is used in the Secure project.

Software Delay

Implements a blocking software delay. A delay can be specified in microseconds, milliseconds or

seconds. The delay is implemented based on the system clock rate.

```
 /* Delay at least 1 second. Depending on the number of wait states required for the region of memory
  * that the software_delay_loop has been linked in this could take longer. The default is 4 cycles per loop.
  * This can be modified by redefining DELAY_LOOP_CYCLES. BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS,
  * and BSP_DELAY_UNITS_MICROSECONDS can all be used with R_BSP_SoftwareDelay. */
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

Trignometric Function

To utilize the TFU hardware for trigonometric math functions, please use the functions provided by each compiler. The CCRX, GCC, and IAR compilers offer options related to TFU. These compilers provide an option to replace calls to the following mathematical library functions with TFU-enabled code. By modifying the compiler options, the trigonometric library functions sinf(), cosf(), atan2f(), hypotf() and more can be mapped to their respective TFU functions. Extended functions sincosf() and atan2hypotf() are also available when the compiler option is enabled in project settings.

Note

Refer to the MCU hardware user's manual or datasheet to determine if it has TFU support.

Critical Section Macros

Implements a critical section. Some MCUs (MCUs with the BASEPRI register) support allowing high priority interrupts to execute during critical sections. On these MCUs, interrupts with priority less than or equal to BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION are not serviced in critical sections. Interrupts with higher priority than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION still execute in critical sections.

```
   FSP_CRITICAL_SECTION_DEFINE;
 /* Store the current interrupt posture. */
 FSP_CRITICAL_SECTION_ENTER;
 /* Interrupts cannot run in this section unless their priority is less than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. */
 /* Restore saved interrupt posture. */
 FSP_CRITICAL_SECTION_EXIT;
```

XSPIClock Update

Supports changing the XSPI Clock (XSPICLK) during runtime if supported by the MCU. The XSPICLK source and clock divisor can be updated. It is user's responsibility to ensure the selected clock

source is running before attempting to update XSPICLK.

Endian

The BSP supports both big-endian and little-endian. Users can configure the MCU endian through the compiler settings.

Devices with LVOCR

By default, FSP disables I/O port low voltage mode for devices with the LVOCR register. Presently, RX74M and RX74N devices configure I/O ports based on VCC and VCC2 voltage levels. For cases where VCC and/or VCC2 will be less than 2.7 V for these devices, the I/O port low voltage mode should be enabled for VCC.

Note

If the I/O port low voltage mode is not enabled for VCC when less than 2.7 V, device electrical characteristics cannot be guaranteed. Do not enable I/O port low voltage mode for VCC when they are greater than or equal to 2.7 V.

Limited D-Cache and I-Cache Support

For MCUs with D-Cache and I-Cache, limited support is available for enabling the caches and automatically configuring predefined non-cacheable and cacheable regions via the MPU during BSP initialization.

For these MCUs, D-Cache is disabled by default because certain existing drivers do not support data coherency with D-Cache enabled. Enabling the D-Cache requires that data coherency be considered in any circumstance where a core interacts with other bus members.

For these MCUs, I-Cache is also disabled by default to maintain compatibility with existing applications. Users can enable I-Cache separately if the performance benefits are desired and the application has been validated accordingly.

Cacheable Buffer Placement Example

The predefined cacheable regions configured by the MPU when D-Cache is enabled can be used to contain data that should be cached. To use the predefined cacheable regions, place the data into the corresponding cacheable section defined by the linker script for the chosen toolchain.

Below is an example of placing buffers in cacheable regions:

```
uint8_t cached_uninitialized_buffer_sram[1024] BSP_PLACE_IN_SECTION(".ram_noinit");
```

Note

The .ram_noinit and .sdram_noinit sections were treated as uninitialized sections.

Non-Cacheable Buffer Placement Example

The predefined non-cacheable regions configured by the MPU when D-Cache is enabled can be used to contain data that should not be cached, ensuring data coherency for that data. To use the predefined non-cacheable regions, place the data into the corresponding non-cacheable section defined by the linker script for the chosen toolchain. The predefined non-cacheable regions are not initialized by the BSP.

Below is an example of placing buffers in non-cacheable regions:

```
uint8_t uncached_uninitialized_buffer_sram[1024] BSP_PLACE_IN_SECTION(".ram_noinit_no
cache");
```

Note

The .ram_noinit_nocache and .sdram_noinit_nocache sections were treated as uninitialized sections.

Configuration

The BSP is heavily data driven with most features and functionality being configured based on the content from configuration files. Configuration files represent the settings specified by the user and are generated when the project is built and/or when the Generate Project Content button is clicked in the FSP Configuration editor.

Build Time Configurations for fsp_common

The following build time configurations are defined in fsp_cfg/bsp/bsp_cfg.h:

Configuration	Options	Default	Description
Main stack size (bytes)	Value must be an integer multiple of 8 and between 8 and 0xFFFFFFFF	0x400	Set the size of the main program stack. NOTE: This entry is for the main stack. When using an RTOS, thread stacks can be configured in the properties for each thread.
Heap size (bytes)	Value must be 0 or an integer multiple of 8 between 8 and 0xFFFFFFFF.	0	The main heap is disabled by default. Set the heap size to a positive integer divisible by 8 to enable it. A minimum of 4K (0x1000) is recommended if standard library functions are to be used.
Protect zone root stack size (bytes)	Value must be 0 or an integer multiple of 8 between 8 and 0xFFFFFFFF.	0x800	The number of root interrupt stacks used by the ProtectZone hypervisor.
MCU Vcc (mV)	Value must between 0 and 5500 (5.5V)	3300	Some peripherals require different settings based on the supplied voltage.

			Entering Vcc here (in mV) allows the relevant driver modules to configure the associated peripherals accordingly.
Parameter checking	• Enabled • Disabled	Disabled	When enabled, parameter checking for the BSP is turned on. In addition, any modules whose parameter checking configuration is set to 'Default (BSP)' will perform parameter checking as well.
Assert Failures	• Return FSP_ERR_ASSERTION • Call fsp_error_log then Return FSP_ERR_ASSERTION • Use assert() to Halt Execution • Disable checks that would return FSP_ERR_ASSERTION	Return FSP_ERR_ASSERTION	Define the behavior of the FSP_ASSERT() macro.
Clock Registers not Reset Values during Startup	• Disabled • Enabled	Disabled	If enabled, registers are assumed to be set to their reset value during startup. Enable this if another application such as a bootloader or Secure project has already configured the clocks before the startup code runs.
Main Oscillator Populated	• Populated • Not Populated	Populated	Select whether or not there is a main oscillator (XTAL) on the board. This setting can be overridden in board_cfg.h.
PFS Protect	• Disabled • Enabled	Enabled	Keep the PFS registers locked when they are not being modified. If disabled they will be unlocked during startup.

C Runtime Initialization	• Enabled • Disabled	Enabled	Select if the C runtime initialization in the BSP is to be used. If disabled, use the BSP_WARM_START_POST_CLOCK event to run user defined equivalent.
Early BSP Initialization	• Enabled • Disabled	Disabled	Enable this option to use BSP functions before C runtime initialization (BSP_WARM_START_RESET or BSP_WARM_START_POST_CLOCK).
Main Oscillator Clock Source	• External Oscillator • Crystal or Resonator	Crystal or Resonator	Select the main oscillator clock source. This setting can be overridden in board_cfg.h
Subclock Populated	• Populated • Not Populated	Populated	Select whether or not there is a subclock crystal on the board. This setting can be overridden in board_cfg.h.
Subclock Drive (Drive capacitance availability varies by MCU)	• Standard/Normal mode • Low/Low power mode 1 • Low power mode 2 • Low power mode 3	Standard/Normal mode	Select the subclock oscillator drive capacitance. This setting can be overridden in board_cfg.h
Subclock Stabilization Time (ms)	Value must between 0 and 10000	1000	Select the subclock oscillator stabilization time. This is only used in the startup code if the subclock is selected as the system clock on the Clocks tab or if the HOCO FLL function is enabled. This setting can be overridden in board_cfg.h

Usage Notes

Limitation

The BSP will not initialize any memory sections in the SDRAM. The user is responsible for initializing

any code or data stored in SDRAM.

FSP C++ projects are not supported with the CC-RX compiler because it does not support the C++ 11 features, which is required for C++ project. Use a C project when building with CC-RX.

ProtectZone

Secure World Project Preparation

Secure Service Call

Secure Service Call (SSC) to retrieve clock information from the Non-Secure world. Users can add their corresponding processing as required.

ProtectZone Limitations

Secure Execution Blocks Non-Secure Execution

Non-Secure interrupts that occur during Secure execution are pended.

This BSP does not support hypervisor calls that invoke Non-Secure functions from the Secure world.

Secure Interrupt Priority

In ProtectZone Secure projects, Secure interrupt priorities must be higher than 15 to prevent Non-Secure interrupts from preempting Secure execution.

Secure Floating-Point Registers Not Cleared

Secure floating-point registers are not cleared when transitioning from the Secure to the Non-Secure world.

CCRX Secure Project Setting

When building a secure application with the CC-RX compiler, define the _RX_PZ_SECURE assembler symbol as follows:

1. From the Project menu, select C/C++ Project Settings.

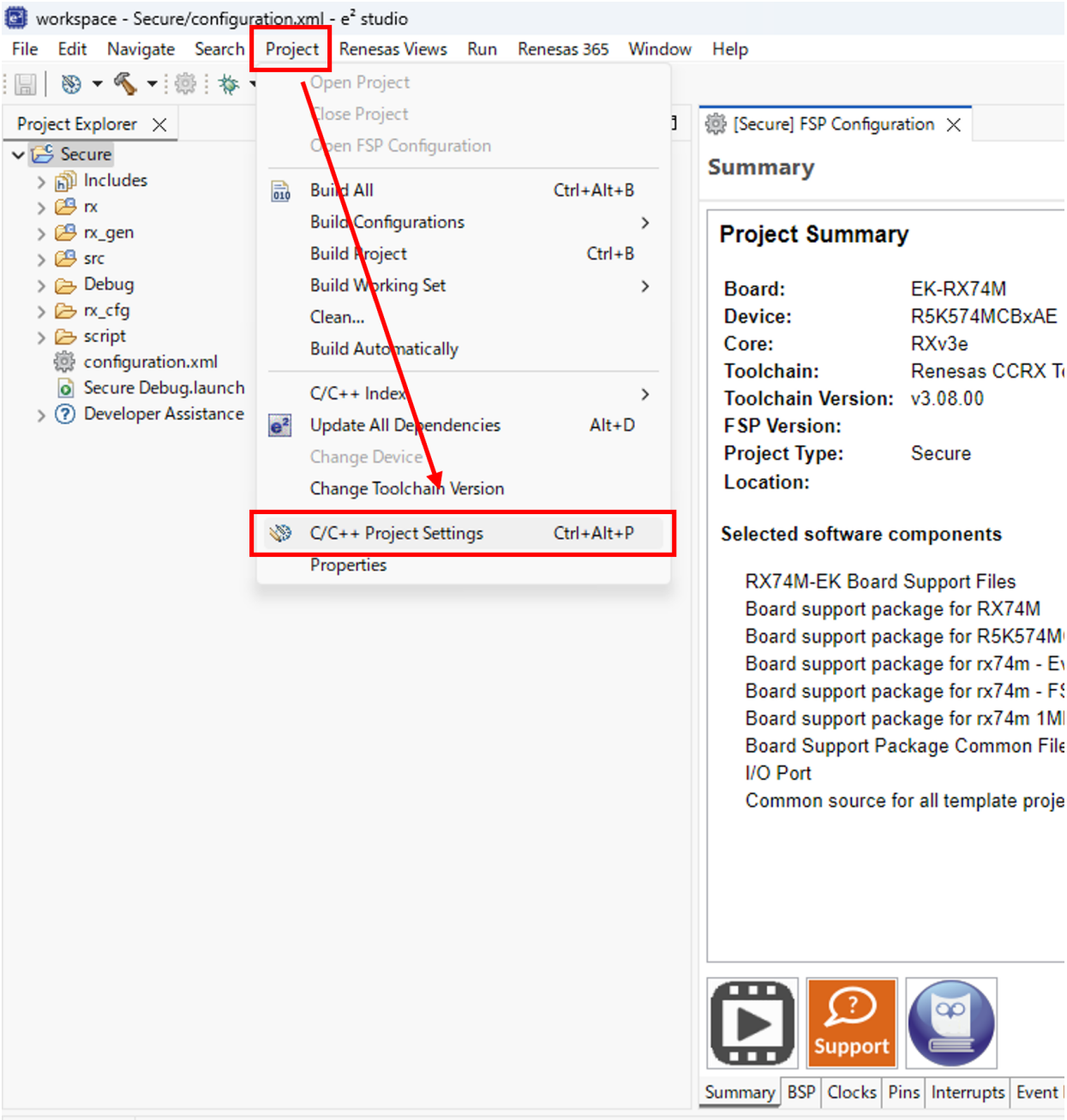


Figure 93: Project Settings

2. Select Paths and Symbols, then open the Symbols tab. Under Languages, select Assembly and click Add.

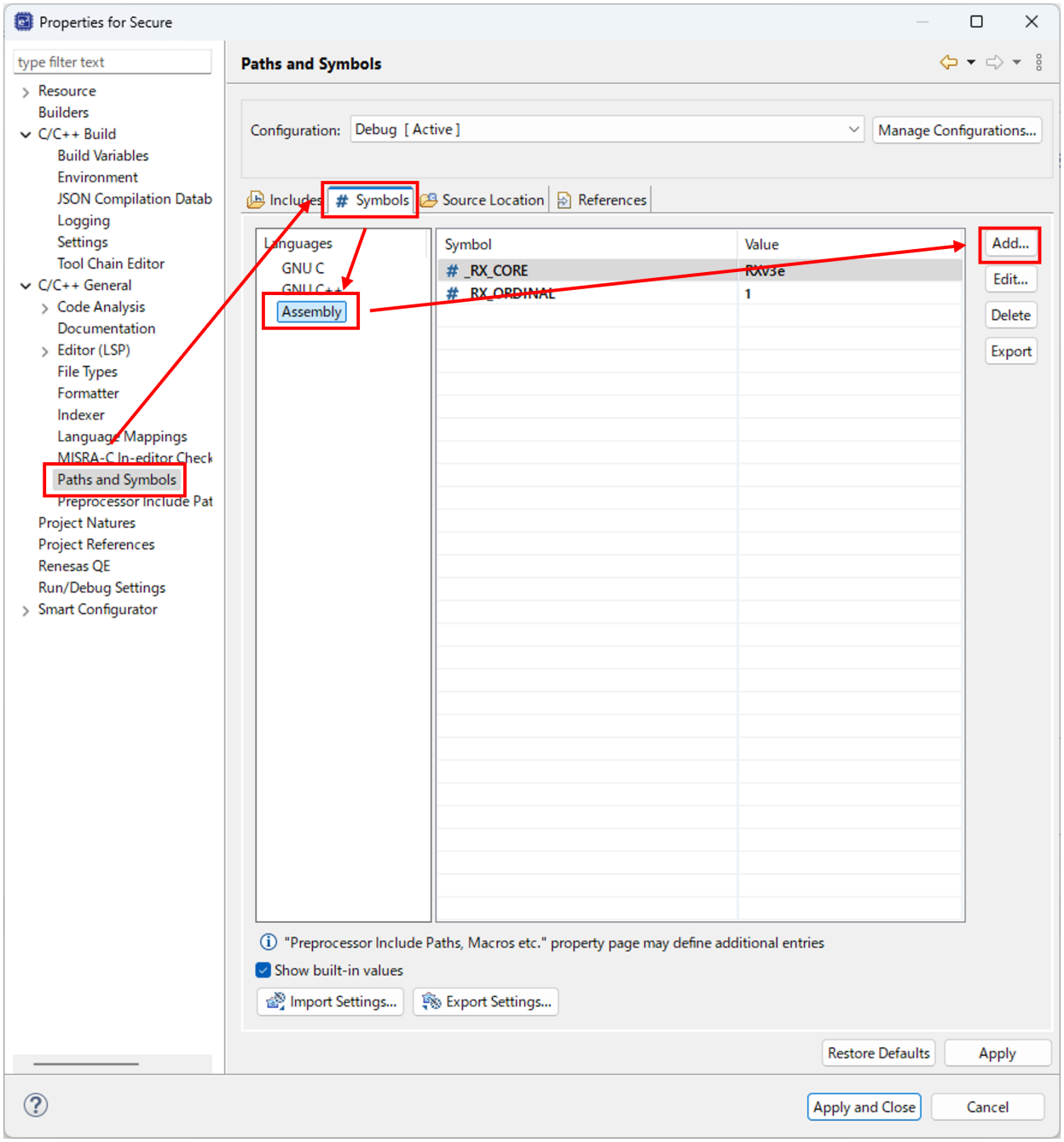


Figure 94: Project Settings path_and_symbols

3. Enter _RX_PZ_SECURE in the Name field and 1 in the Value field, then click OK.

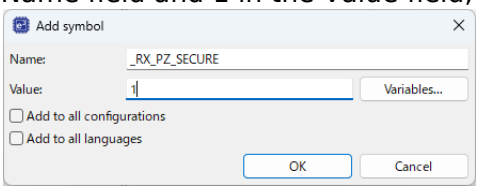


Figure 95: Project Settings asm symbol Settings

Modules

	RX74M
	RX74N

Data Structures

struct	transfer_info_t

Macros

#define	BSP_IRQ_DISABLED
#define	FSP_LOG_PRINT(X)
#define	FSP_RETURN(err)
#define	FSP_ERROR_LOG(err)
#define	FSP_ASSERT(a)
#define	FSP_ERROR_RETURN(a, err)
#define	FSP_CRITICAL_SECTION_ENTER
#define	FSP_CRITICAL_SECTION_EXIT
#define	FSP_INVALID_VECTOR
#define	BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(x)
#define	BSP_STACK_ALIGNMENT
#define	R_BSP_MODULE_START(ip, channel)
#define	R_BSP_MODULE_STOP(ip, channel)

Enumerations

enum	fsp_ip_t
enum	fsp_signal_t
enum	bsp_warm_start_event_t
enum	fsp_priv_clock_t
enum	fsp_priv_source_clock_t
enum	bsp_delay_units_t

enum	bsp_grp_irq_nmi_t
enum	bsp_reg_protect_t
enum	transfer_mode_t
enum	transfer_size_t
enum	transfer_addr_mode_t
enum	transfer_repeat_area_t
enum	transfer_chain_mode_t
enum	transfer_irq_t

Variables

uint32_t SystemCoreClock	BSP_SECTION_EARLY_INIT

Data Structure Documentation

◆ transfer_info_t

struct transfer_info_t

This structure specifies the properties of the transfer.

Warning

When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: p_src, p_dest, num_blocks, and length.
When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique transfer_info_t.
When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed.

Note

When using DTC, consider placing instances of this structure in a protected section of memory.

Data Fields

union transfer_info_t	transfer_settings_u	
void const *volatile	p_src	Source pointer.
void *volatile	p_dest	Destination pointer.
volatile uint16_t	num_blocks	Number of blocks to transfer when using TRANSFER_MODE_BLOCK (DTC, DMAC and EXDMAC) or TRANSFER_MODE_REPEAT (DMAC and EXDMAC only) or TRANSFER_MODE_REPEAT_BLO

		CK (DMAC and EXDMAC only), unused in other modes. Number of blocks to transfer when using TRANSFER_MODE_BLOCK (DTC, DMAC or EXDMAC) or TRANSFER_MODE_REPEAT (DMAC and EXDMAC only) or TRANSFER_MODE_REPEAT_BLOCK (DMAC and EXDMAC only), unused in other modes.
volatile uint16_t	length	Length of each transfer. Range limited for TRANSFER_MODE_BLOCK, TRANSFER_MODE_REPEAT, and TRANSFER_MODE_REPEAT_BLOCK see HAL driver for details.
union transfer_info_t	transfer_settings_u	

Macro Definition Documentation

◆ BSP_IRQ_DISABLED

#define BSP_IRQ_DISABLED
Used to configure the interrupt priority level into level 0 for disabling the interrupt.

◆ FSP_LOG_PRINT

#define FSP_LOG_PRINT (X)
Macro that can be defined in order to enable logging in FSP modules.

◆ FSP_RETURN

#define FSP_RETURN (err)
Macro to log and return error without an assertion.

◆ FSP_ERROR_LOG

#define FSP_ERROR_LOG (err)
This function is called before returning an error code. To stop on a runtime error, define fsp_error_log in user code and do required debugging (breakpoints, stack dump, etc) in this function.

◆ FSP_ASSERT

#define FSP_ASSERT (a)
Default assertion calls FSP_ERROR_RETURN if condition "a" is false. Used to identify incorrect use of API's in FSP functions.

◆ FSP_ERROR_RETURN

#define FSP_ERROR_RETURN (a, err)
All FSP error codes are returned using this macro. Calls FSP_ERROR_LOG function if condition "a" is false. Used to identify runtime errors in FSP functions.

◆ FSP_CRITICAL_SECTION_ENTER

#define FSP_CRITICAL_SECTION_ENTER
This macro temporarily saves the current interrupt state and disables interrupts.

◆ FSP_CRITICAL_SECTION_EXIT

#define FSP_CRITICAL_SECTION_EXIT
This macro restores the previously saved interrupt state, reenabling interrupts.

◆ FSP_INVALID_VECTOR

#define FSP_INVALID_VECTOR
Used to signify that the requested IRQ vector is not defined in this system.

◆ BSP_CFG_HANDLE_UNRECOVERABLE_ERROR

#define BSP_CFG_HANDLE_UNRECOVERABLE_ERROR (x)
In the event of an unrecoverable error the BSP will by default call the __BKPT() intrinsic function which will alert the user of the error. The user can override this default behavior by defining their own BSP_CFG_HANDLE_UNRECOVERABLE_ERROR macro.

◆ BSP_STACK_ALIGNMENT

#define BSP_STACK_ALIGNMENT
Stacks (and heap) must be sized and aligned to an integer multiple of this number.

◆ R_BSP_MODULE_START

#define R_BSP_MODULE_START (ip, channel)

Cancels the module stop state.

Parameters

ip	fsp_ip_t enum value for the module to be stopped
channel	The channel. Use channel 0 for modules without channels.

◆ R_BSP_MODULE_STOP

#define R_BSP_MODULE_STOP (ip, channel)

Enables the module stop state.

Parameters

ip	fsp_ip_t enum value for the module to be stopped
channel	The channel. Use channel 0 for modules without channels.

Enumeration Type Documentation

◆ fsp_ip_t

enum fsp_ip_t	
Available modules.	
Enumerator	
FSP_IP_CFLASH	Code Flash.
FSP_IP_DFLASH	Data Flash.
FSP_IP_RAM	RAM.
FSP_IP_LVD	Low Voltage Detection.
FSP_IP_CGC	Clock Generation Circuit.
FSP_IP_LPM	Low Power Modes.
FSP_IP_FCU	Flash Control Unit.
FSP_IP_ICU	Interrupt Control Unit.
FSP_IP_DMAC	DMA Controller.
FSP_IP_DTC	Data Transfer Controller.
FSP_IP_IOPORT	I/O Ports.
FSP_IP_PFS	Pin Function Select.
FSP_IP_ELC	Event Link Controller.
FSP_IP_MPU	Memory Protection Unit.
FSP_IP_MSTP	Module Stop.
FSP_IP_MMF	Memory Mirror Function.
FSP_IP_KEY	Key Interrupt Function.
FSP_IP_CAC	Clock Frequency Accuracy Measurement Circuit.
FSP_IP_DOC	Data Operation Circuit.
FSP_IP_CRC	Cyclic Redundancy Check Calculator.
FSP_IP_SCI	Serial Communications Interface.

FSP_IP_IIC	I2C Bus Interface.
FSP_IP_SPI	Serial Peripheral Interface.
FSP_IP_CTSU	Capacitive Touch Sensing Unit.
FSP_IP_SCE	Secure Cryptographic Engine.
FSP_IP_SLCDC	Segment LCD Controller.
FSP_IP_AES	Advanced Encryption Standard.
FSP_IP_TRNG	True Random Number Generator.
FSP_IP_FCACHE	Flash Cache.
FSP_IP_SRAM	SRAM.
FSP_IP_ADC	A/D Converter.
FSP_IP_DAC	12-Bit D/A Converter
FSP_IP_TSN	Temperature Sensor.
FSP_IP_DAAD	D/A A/D Synchronous Unit.
FSP_IP_ACMPHS	High Speed Analog Comparator.
FSP_IP_ACMPLP	Low Power Analog Comparator.
FSP_IP_OPAMP	Operational Amplifier.
FSP_IP_SDADC	Sigma Delta A/D Converter.
FSP_IP_RTC	Real Time Clock.
FSP_IP_WDT	Watch Dog Timer.
FSP_IP_IWDT	Independent Watch Dog Timer.
FSP_IP_GPT	General PWM Timer.
FSP_IP_POEG	Port Output Enable for GPT.
FSP_IP_OPS	Output Phase Switch.
FSP_IP_AGT	Asynchronous General-Purpose Timer.

FSP_IP_CAN	Controller Area Network.
FSP_IP_IRDA	Infrared Data Association.
FSP_IP_QSPI	Quad Serial Peripheral Interface.
FSP_IP_USBFS	USB Full Speed.
FSP_IP_SDHI	SD Host Interface.
FSP_IP_SRC	Sampling Rate Converter.
FSP_IP_SSI	Serial Sound Interface.
FSP_IP_DALI	Digital Addressable Lighting Interface.
FSP_IP_ETHER	Ethernet MAC Controller.
FSP_IP_EDMAC	Ethernet DMA Controller.
FSP_IP_EPTPC	Ethernet PTP Controller.
FSP_IP_PDC	Parallel Data Capture Unit.
FSP_IP_GLCDC	Graphics LCD Controller.
FSP_IP_DRW	2D Drawing Engine
FSP_IP_JPEG	JPEG.
FSP_IP_DAC8	8-Bit D/A Converter
FSP_IP_USBHS	USB High Speed.
FSP_IP_OSPI	Octa Serial Peripheral Interface.
FSP_IP_CEC	HDMI CEC.
FSP_IP_TFU	Trigonometric Function Unit.
FSP_IP_IIRFA	IIR Filter Accelerator.
FSP_IP_CANFD	CAN-FD.
FSP_IP_EXDMAC	EXDMA Controller.
FSP_IP_ACSE	All-Module Clock Stop Mode Enable.

FSP_IP_EXPANSIONRAM	Expansion RAM.
FSP_IP_ECCRAM	ECCRAM.
FSP_IP_STANDBYRAM	Standby RAM.
FSP_IP_MMCIF	MMC Host Interface.
FSP_IP_TSIP	Trusted Secure IP.
FSP_IP_TPU	16-Bit Timer Pulse Unit
FSP_IP_CMT	Compare Match Timer.
FSP_IP_CMTW	Compare Match Timer W.
FSP_IP_TMR	8-Bit Timer
FSP_IP_MTU	Multifunction Timer Pulse Unit.
FSP_IP_PPG	Programmable Pulse Generator.
FSP_IP_HRPWM	High Resolution PWM Waveform Generation Circuit.
FSP_IP_CMPC	Comparator C.
FSP_IP_SCI_B	Serial Communications Interface.
FSP_IP_I3C	I3C Bus Interface.
FSP_IP_SPI_B	Serial Peripheral Interface.
FSP_IP_ENDAT	EnDat 2.2.
FSP_IP_ENDAT3	EnDat 3.
FSP_IP_BISS	BiSS-C.
FSP_IP_BISSL	BiSS-Line.
FSP_IP_HDSL	HIPERFACE DSL.
FSP_IP_AFMT	A-format.
FSP_IP_FFT	FFT Accelerator.

◆ fsp_signal_t

enum fsp_signal_t	
Signals that can be mapped to an interrupt.	
Enumerator	
FSP_SIGNAL_ADC_COMPARE_MATCH	ADC COMPARE MATCH.
FSP_SIGNAL_ADC_COMPARE_MISMATCH	ADC COMPARE MISMATCH.
FSP_SIGNAL_ADC_SCAN_END	ADC SCAN END.
FSP_SIGNAL_ADC_SCAN_END_B	ADC SCAN END B.
FSP_SIGNAL_ADC_WINDOW_A	ADC WINDOW A.
FSP_SIGNAL_ADC_WINDOW_B	ADC WINDOW B.
FSP_SIGNAL_AES_RDREQ	AES RDREQ.
FSP_SIGNAL_AES_WRREQ	AES WRREQ.
FSP_SIGNAL_AGT_COMPARE_A	AGT COMPARE A.
FSP_SIGNAL_AGT_COMPARE_B	AGT COMPARE B.
FSP_SIGNAL_AGT_INT	AGT INT.
FSP_SIGNAL_CAC_FREQUENCY_ERROR	CAC FREQUENCY ERROR.
FSP_SIGNAL_CAC_MEASUREMENT_END	CAC MEASUREMENT END.
FSP_SIGNAL_CAC_OVERFLOW	CAC OVERFLOW.
FSP_SIGNAL_CAN_ERROR	CAN ERROR.
FSP_SIGNAL_CAN_FIFO_RX	CAN FIFO RX.
FSP_SIGNAL_CAN_FIFO_TX	CAN FIFO TX.
FSP_SIGNAL_CAN_MAILBOX_RX	CAN MAILBOX RX.
FSP_SIGNAL_CAN_MAILBOX_TX	CAN MAILBOX TX.
FSP_SIGNAL_CGC_MOSC_STOP	CGC MOSC STOP.
FSP_SIGNAL_LPM_SNOOZE_REQUEST	LPM SNOOZE REQUEST.

FSP_SIGNAL_LVD_LVD1	LVD LVD1.
FSP_SIGNAL_LVD_LVD2	LVD LVD2.
FSP_SIGNAL_VBATT_LVD	VBATT LVD.
FSP_SIGNAL_LVD_VBATT	LVD VBATT.
FSP_SIGNAL_ACMPHS_INT	ACMPHS INT.
FSP_SIGNAL_ACMPLP_INT	ACMPLP INT.
FSP_SIGNAL_CTSU_END	CTSU END.
FSP_SIGNAL_CTSU_READ	CTSU READ.
FSP_SIGNAL_CTSU_WRITE	CTSU WRITE.
FSP_SIGNAL_DALI_DEI	DALI DEI.
FSP_SIGNAL_DALI_CLI	DALI CLI.
FSP_SIGNAL_DALI_SDI	DALI SDI.
FSP_SIGNAL_DALI_BPI	DALI BPI.
FSP_SIGNAL_DALI_FEI	DALI FEI.
FSP_SIGNAL_DALI_SDI_OR_BPI	DALI SDI OR BPI.
FSP_SIGNAL_DMAC_INT	DMAC INT.
FSP_SIGNAL_DOC_INT	DOC INT.
FSP_SIGNAL_DRW_INT	DRW INT.
FSP_SIGNAL_DTC_COMPLETE	DTC COMPLETE.
FSP_SIGNAL_DTC_END	DTC END.
FSP_SIGNAL_EDMAC_EINT	EDMAC EINT.
FSP_SIGNAL_ELC_SOFTWARE_EVENT_0	ELC SOFTWARE EVENT 0.
FSP_SIGNAL_ELC_SOFTWARE_EVENT_1	ELC SOFTWARE EVENT 1.
FSP_SIGNAL_EPTPC_IPLS	EPTPC IPLS.

FSP_SIGNAL_EPTPC_MINT	EPTPC MINT.
FSP_SIGNAL_EPTPC_PINT	EPTPC PINT.
FSP_SIGNAL_EPTPC_TIMER0_FALL	EPTPC TIMER0 FALL.
FSP_SIGNAL_EPTPC_TIMER0_RISE	EPTPC TIMER0 RISE.
FSP_SIGNAL_EPTPC_TIMER1_FALL	EPTPC TIMER1 FALL.
FSP_SIGNAL_EPTPC_TIMER1_RISE	EPTPC TIMER1 RISE.
FSP_SIGNAL_EPTPC_TIMER2_FALL	EPTPC TIMER2 FALL.
FSP_SIGNAL_EPTPC_TIMER2_RISE	EPTPC TIMER2 RISE.
FSP_SIGNAL_EPTPC_TIMER3_FALL	EPTPC TIMER3 FALL.
FSP_SIGNAL_EPTPC_TIMER3_RISE	EPTPC TIMER3 RISE.
FSP_SIGNAL_EPTPC_TIMER4_FALL	EPTPC TIMER4 FALL.
FSP_SIGNAL_EPTPC_TIMER4_RISE	EPTPC TIMER4 RISE.
FSP_SIGNAL_EPTPC_TIMER5_FALL	EPTPC TIMER5 FALL.
FSP_SIGNAL_EPTPC_TIMER5_RISE	EPTPC TIMER5 RISE.
FSP_SIGNAL_FCU_FIFERR	FCU FIFERR.
FSP_SIGNAL_FCU_FRDYI	FCU FRDYI.
FSP_SIGNAL_GLCDC_LINE_DETECT	GLCDC LINE DETECT.
FSP_SIGNAL_GLCDC_UNDERFLOW_1	GLCDC UNDERFLOW 1.
FSP_SIGNAL_GLCDC_UNDERFLOW_2	GLCDC UNDERFLOW 2.
FSP_SIGNAL_GPT_CAPTURE_COMPARE_A	GPT CAPTURE COMPARE A.
FSP_SIGNAL_GPT_CAPTURE_COMPARE_B	GPT CAPTURE COMPARE B.
FSP_SIGNAL_GPT_COMPARE_C	GPT COMPARE C.
FSP_SIGNAL_GPT_COMPARE_D	GPT COMPARE D.
FSP_SIGNAL_GPT_COMPARE_E	GPT COMPARE E.

FSP_SIGNAL_GPT_COMPARE_F	GPT COMPARE F.
FSP_SIGNAL_GPT_COUNTER_OVERFLOW	GPT COUNTER OVERFLOW.
FSP_SIGNAL_GPT_COUNTER_UNDERFLOW	GPT COUNTER UNDERFLOW.
FSP_SIGNAL_GPT_AD_TRIG_A	GPT AD TRIG A.
FSP_SIGNAL_GPT_AD_TRIG_B	GPT AD TRIG B.
FSP_SIGNAL_OPS_UVW_EDGE	OPS UVW EDGE.
FSP_SIGNAL_ICU_IRQ0	ICU IRQ0.
FSP_SIGNAL_ICU_IRQ1	ICU IRQ1.
FSP_SIGNAL_ICU_IRQ2	ICU IRQ2.
FSP_SIGNAL_ICU_IRQ3	ICU IRQ3.
FSP_SIGNAL_ICU_IRQ4	ICU IRQ4.
FSP_SIGNAL_ICU_IRQ5	ICU IRQ5.
FSP_SIGNAL_ICU_IRQ6	ICU IRQ6.
FSP_SIGNAL_ICU_IRQ7	ICU IRQ7.
FSP_SIGNAL_ICU_IRQ8	ICU IRQ8.
FSP_SIGNAL_ICU_IRQ9	ICU IRQ9.
FSP_SIGNAL_ICU_IRQ10	ICU IRQ10.
FSP_SIGNAL_ICU_IRQ11	ICU IRQ11.
FSP_SIGNAL_ICU_IRQ12	ICU IRQ12.
FSP_SIGNAL_ICU_IRQ13	ICU IRQ13.
FSP_SIGNAL_ICU_IRQ14	ICU IRQ14.
FSP_SIGNAL_ICU_IRQ15	ICU IRQ15.
FSP_SIGNAL_ICU_SNOOZE_CANCEL	ICU SNOOZE CANCEL.
FSP_SIGNAL_IIC_ERI	IIC ERI.

FSP_SIGNAL_IIC_RXI	IIC RXI.
FSP_SIGNAL_IIC_TEI	IIC TEI.
FSP_SIGNAL_IIC_TXI	IIC TXI.
FSP_SIGNAL_IIC_WUI	IIC WUI.
FSP_SIGNAL_IOPORT_EVENT_1	IOPORT EVENT 1.
FSP_SIGNAL_IOPORT_EVENT_2	IOPORT EVENT 2.
FSP_SIGNAL_IOPORT_EVENT_3	IOPORT EVENT 3.
FSP_SIGNAL_IOPORT_EVENT_4	IOPORT EVENT 4.
FSP_SIGNAL_IOPORT_EVENT_B	IOPORT EVENT B.
FSP_SIGNAL_IOPORT_EVENT_C	IOPORT EVENT C.
FSP_SIGNAL_IOPORT_EVENT_D	IOPORT EVENT D.
FSP_SIGNAL_IOPORT_EVENT_E	IOPORT EVENT E.
FSP_SIGNAL_IWDT_UNDERFLOW	IWDT UNDERFLOW.
FSP_SIGNAL_JPEG_JDTI	JPEG JDTI.
FSP_SIGNAL_JPEG_JEDI	JPEG JEDI.
FSP_SIGNAL_KEY_INT	KEY INT.
FSP_SIGNAL_PDC_FRAME_END	PDC FRAME END.
FSP_SIGNAL_PDC_INT	PDC INT.
FSP_SIGNAL_PDC_RECEIVE_DATA_READY	PDC RECEIVE DATA READY.
FSP_SIGNAL_POEG_EVENT	POEG EVENT.
FSP_SIGNAL_QSPI_INT	QSPI INT.
FSP_SIGNAL_RTC_ALARM	RTC ALARM.
FSP_SIGNAL_RTC_PERIOD	RTC PERIOD.
FSP_SIGNAL_RTC_CARRY	RTC CARRY.

FSP_SIGNAL_SCE_INTEGRATE_RDRDY	SCE INTEGRATE RDRDY.
FSP_SIGNAL_SCE_INTEGRATE_WRRDY	SCE INTEGRATE WRRDY.
FSP_SIGNAL_SCE_LONG_PLG	SCE LONG PLG.
FSP_SIGNAL_SCE_PROC_BUSY	SCE PROC BUSY.
FSP_SIGNAL_SCE_RDRDY_0	SCE RDRDY 0.
FSP_SIGNAL_SCE_RDRDY_1	SCE RDRDY 1.
FSP_SIGNAL_SCE_ROMOK	SCE ROMOK.
FSP_SIGNAL_SCE_TEST_BUSY	SCE TEST BUSY.
FSP_SIGNAL_SCE_WRRDY_0	SCE WRRDY 0.
FSP_SIGNAL_SCE_WRRDY_1	SCE WRRDY 1.
FSP_SIGNAL_SCE_WRRDY_4	SCE WRRDY 4.
FSP_SIGNAL_SCI_AM	SCI AM.
FSP_SIGNAL_SCI_ERI	SCI ERI.
FSP_SIGNAL_SCI_RXI	SCI RXI.
FSP_SIGNAL_SCI_RXI_OR_ERI	SCI RXI OR ERI.
FSP_SIGNAL_SCI_TEI	SCI TEI.
FSP_SIGNAL_SCI_TXI	SCI TXI.
FSP_SIGNAL_SDADC_ADI	SDADC ADI.
FSP_SIGNAL_SDADC_SCANEND	SDADC SCANEND.
FSP_SIGNAL_SDADC_CALIEND	SDADC CALIEND.
FSP_SIGNAL_SDHIMMC_ACCS	SDHIMMC ACCS.
FSP_SIGNAL_SDHIMMC_CARD	SDHIMMC CARD.
FSP_SIGNAL_SDHIMMC_DMA_REQ	SDHIMMC DMA REQ.
FSP_SIGNAL_SDHIMMC_SDIO	SDHIMMC SDIO.

FSP_SIGNAL_SPI_ERI	SPI ERI.
FSP_SIGNAL_SPI_IDLE	SPI IDLE.
FSP_SIGNAL_SPI_RXI	SPI RXI.
FSP_SIGNAL_SPI_TEI	SPI TEI.
FSP_SIGNAL_SPI_TXI	SPI TXI.
FSP_SIGNAL_SRC_CONVERSION_END	SRC CONVERSION END.
FSP_SIGNAL_SRC_INPUT_FIFO_EMPTY	SRC INPUT FIFO EMPTY.
FSP_SIGNAL_SRC_OUTPUT_FIFO_FULL	SRC OUTPUT FIFO FULL.
FSP_SIGNAL_SRC_OUTPUT_FIFO_OVERFLOW	SRC OUTPUT FIFO OVERFLOW.
FSP_SIGNAL_SRC_OUTPUT_FIFO_UNDERFLOW	SRC OUTPUT FIFO UNDERFLOW.
FSP_SIGNAL_SSI_INT	SSI INT.
FSP_SIGNAL_SSI_RXI	SSI RXI.
FSP_SIGNAL_SSI_TXI	SSI TXI.
FSP_SIGNAL_SSI_TXI_RXI	SSI TXI RXI.
FSP_SIGNAL_TRNG_RDREQ	TRNG RDREQ.
FSP_SIGNAL_USB_FIFO_0	USB FIFO 0.
FSP_SIGNAL_USB_FIFO_1	USB FIFO 1.
FSP_SIGNAL_USB_INT	USB INT.
FSP_SIGNAL_USB_RESUME	USB RESUME.
FSP_SIGNAL_USB_USB_INT_RESUME	USB USB INT RESUME.
FSP_SIGNAL_WDT_UNDERFLOW	WDT UNDERFLOW.
FSP_SIGNAL_ULPT_COMPARE_A	ULPT COMPARE A.
FSP_SIGNAL_ULPT_COMPARE_B	ULPT COMPARE B.
FSP_SIGNAL_ULPT_INT	ULPT INT.

◆ bsp_warm_start_event_t

enum bsp_warm_start_event_t	
Different warm start entry locations in the BSP.	
Enumerator	
BSP_WARM_START_RESET	Called almost immediately after reset. No C runtime environment, clocks, or IRQs.
BSP_WARM_START_POST_CLOCK	Called after clock initialization. No C runtime environment or IRQs.
BSP_WARM_START_POST_C	Called after clocks and C runtime environment have been set up.

◆ fsp_priv_clock_t

enum fsp_priv_clock_t	
Enumerator	
FSP_PRIV_CLOCK_UNUSED	Sentinel value for unused clock.

◆ fsp_priv_source_clock_t

enum fsp_priv_source_clock_t	
Enumerator	
FSP_PRIV_CLOCK_HOCO	The high speed on chip oscillator.
FSP_PRIV_CLOCK_MOCO	The middle speed on chip oscillator.
FSP_PRIV_CLOCK_LOCO	The low speed on chip oscillator.
FSP_PRIV_CLOCK_MAIN_OSC	The main oscillator.
FSP_PRIV_CLOCK_SUBCLOCK	The subclock oscillator.
FSP_PRIV_CLOCK_PLL	The PLL output.
FSP_PRIV_CLOCK_PLL1P	The PLL1P output.
FSP_PRIV_CLOCK_PLL2	The PLL2 output.
FSP_PRIV_CLOCK_PLL2P	The PLL2P output.
FSP_PRIV_CLOCK_PLL1Q	The PLL1Q output.
FSP_PRIV_CLOCK_PLL1R	The PLL1R output.
FSP_PRIV_CLOCK_PLL2Q	The PLL2Q output.
FSP_PRIV_CLOCK_PLL2R	The PLL2R output.

◆ bsp_delay_units_t

enum bsp_delay_units_t	
Available delay units for R_BSP_SoftwareDelay(). These are ultimately used to calculate a total # of microseconds	
Enumerator	
BSP_DELAY_UNITS_SECONDS	Requested delay amount is in seconds.
BSP_DELAY_UNITS_MILLISECONDS	Requested delay amount is in milliseconds.
BSP_DELAY_UNITS_MICROSECONDS	Requested delay amount is in microseconds.

◆ bsp_grp_irq_nmi_t

enum bsp_grp_irq_nmi_t	
Which interrupts can have callbacks registered.	
Enumerator	
BSP_GRP_IRQ_IWDT_ERROR	IWDT underflow/refresh error has occurred.
BSP_GRP_IRQ_WDT_ERROR	WDT underflow/refresh error has occurred.
BSP_GRP_IRQ_PVD1	Voltage monitoring 1 interrupt.
BSP_GRP_IRQ_PVD2	Voltage monitoring 2 interrupt.
BSP_GRP_IRQ_SOSC_STOP_DETECT	Sub-clock oscillation stop detection interrupt.
BSP_GRP_IRQ_OSC_STOP_DETECT	Main clock oscillation stop detection interrupt.
BSP_GRP_IRQ_NMI_PIN	NMI pin interrupt.
BSP_GRP_IRQ_BUS	Bus error interrupt.
BSP_GRP_IRQ_CM_ERROR	Common memory error interrupt.
BSP_GRP_IRQ_LM_ERROR	Local memory error interrupt.
BSP_GRP_IRQ_DPFPU	Double-precision floating-point exception.
BSP_GRP_IRQ_MRC_READ_DETECT	Code MRAM read access error interrupt.
BSP_GRP_IRQ_MRE_READ_DETECT	Extra MRAM read access error interrupt.

◆ bsp_reg_protect_t

enum bsp_reg_protect_t	
The different types of registers that can be protected.	
Enumerator	
BSP_REG_PROTECT_CGC	Enables writing to the registers related to the clock generation circuit.
BSP_REG_PROTECT_OM_LPC_BATT_SWR	Enables writing to the registers related to operating modes, low power consumption, battery backup function, and software reset function.
BSP_REG_PROTECT_PVD	Enables writing to the registers related to the PVD.
BSP_REG_PROTECT_SAR	Enables writing to the registers related to the security function.
BSP_REG_PROTECT_RST	Enables writing to the registers related to the Reset.

◆ transfer_mode_t

enum transfer_mode_t	
Transfer mode describes what will happen when a transfer request occurs.	
Enumerator	
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC or EXDMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT_BLOCK	In addition to block mode features, repeat-block mode supports a ring buffer of blocks and offsets within a block (to split blocks into arrays of their first data, second data, etc.)
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.

TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC or EXDMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT_BLOCK	In addition to block mode features, repeat-block mode supports a ring buffer of blocks and offsets within a block (to split blocks into arrays of their first data, second data, etc.)

◆ transfer_size_t

enum transfer_size_t	
Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t	
Enumerator	
TRANSFER_SIZE_1_BYTE	Each transfer transfers a 8-bit value.
TRANSFER_SIZE_2_BYTE	Each transfer transfers a 16-bit value.
TRANSFER_SIZE_4_BYTE	Each transfer transfers a 32-bit value.
TRANSFER_SIZE_8_BYTE	Each transfer transfers a 64-bit value.
TRANSFER_SIZE_1_BYTE	Each transfer transfers a 8-bit value.
TRANSFER_SIZE_2_BYTE	Each transfer transfers a 16-bit value.
TRANSFER_SIZE_4_BYTE	Each transfer transfers a 32-bit value.
TRANSFER_SIZE_8_BYTE	Each transfer transfers a 64-bit value.

◆ transfer_addr_mode_t

enum transfer_addr_mode_t	
Address mode specifies whether to modify (increment or decrement) pointer after each transfer.	
Enumerator	
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Offset is added to the address pointer after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Offset is added to the address pointer after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.

◆ transfer_repeat_area_t

enum transfer_repeat_area_t	
Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT_BLOCK, the selected pointer returns to its original value after each transfer.	
Enumerator	
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.

◆ transfer_chain_mode_t

enum transfer_chain_mode_t	
Chain transfer mode options. ***Note*** *Only applies for DTC.*	
Enumerator	
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.

◆ transfer_irq_t

enum transfer_irq_t	
Interrupt options.	
Enumerator	
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. *Note* *Not available in all HAL drivers. See HAL driver for details.*
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. *Note* *Not available in all HAL drivers. See HAL driver for details.*

Function Documentation

◆ R_FSP_VersionGet()

fsp_err_t R_FSP_VersionGet (fsp_pack_version_t *const *p_version*)

Get the FSP version based on compile time macros.

Parameters

[out]	p_version	Memory address to return version information to.

Return values

FSP_SUCCESS	Version information stored.
FSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ Reset_Handler()

void Reset_Handler (void)

MCU starts executing here out of reset. Main stack pointer is set up already.

◆ Default_Handler()

void Default_Handler (void)

Default exception handler.

◆ R_BSP_WarmStart()

void R_BSP_WarmStart (bsp_warm_start_event_t *event*)

This function is called at various points during the startup process. This function is declared as a weak symbol higher up in this file because it is meant to be overridden by a user implemented version. One of the main uses for this function is to call functional safety code during the startup process. To use this function just copy this function into your own code and modify it to meet your needs.

Parameters

[in]	event	Where the code currently is in the start up process

◆ SystemInit()

void SystemInit (void)

Initialize the MCU and the runtime environment.

◆ R_BSP_CacheEnable()

void R_BSP_CacheEnable (void)
Enable caches. Enables all configured cache regions.

◆ R_BSP_CacheDisable()

void R_BSP_CacheDisable (void)
Disable caches. Disable all configured cache regions.

◆ R_BSP_CacheClean()

void R_BSP_CacheClean (void)
Clean caches. Writes back any modified cache contents to memory.

◆ R_BSP_CacheInvalidate()

void R_BSP_CacheInvalidate (void)
Flush caches. Invalidates all cache entries without writing back cached data to memory. Any modified cache contents that have not been cleaned may be discarded.

◆ R_BSP_CacheCleanInvalidate()

void R_BSP_CacheCleanInvalidate (void)
Clean and flush caches. Writes back any modified cache contents to memory and invalidates all cache entries.

◆ bsp_cache_operate_all()

__STATIC_INLINE void bsp_cache_operate_all (uint8_t *operate*)

Cache operation to all area.

Parameters

[in]	operate	Cache operation. BSP_PRV_DCACHE_INV(0) - Cache invalidate. BSP_PRV_DCACHE_CLEAN(1) - Cache clean. BSP_PRV_DCACHE_CLEAN_INV(2) - Cache clean and invalidate.

◆ R_BSP_CacheEnableData()

__STATIC_INLINE void R_BSP_CacheEnableData (void)

Enable Data cache.

◆ R_BSP_CacheDisableData()

__STATIC_INLINE void R_BSP_CacheDisableData (void)

Disable Data cache.

◆ R_BSP_CacheCleanAllData()

__STATIC_INLINE void R_BSP_CacheCleanAllData (void)

Clean whole of the data cache.

◆ R_BSP_CacheInvalidateAllData()

__STATIC_INLINE void R_BSP_CacheInvalidateAllData (void)

Invalidate whole of the data cache.

◆ R_BSP_CacheCleanInvalidateAllData()

__STATIC_INLINE void R_BSP_CacheCleanInvalidateAllData (void)

Clean and invalidate the whole of data cache.

◆ R_BSP_SourceClockHzGet()

uint32_t R_BSP_SourceClockHzGet (fsp_priv_source_clock_t *clock*)

Gets the frequency of a source clock.

Parameters

[in]	clock	Element of clock souce enum which provides clock source.

Returns
Frequency of requested clock in Hertz.

◆ R_FSP_CurrentIrqGet()

__STATIC_INLINE IRQn_Type R_FSP_CurrentIrqGet (void)

Return active interrupt vector number value

Returns
Active interrupt vector number value

◆ R_FSP_SystemClockHzGet()

__STATIC_INLINE uint32_t R_FSP_SystemClockHzGet (fsp_priv_clock_t *clock*)

Gets the frequency of a system clock.

Returns
Frequency of requested clock in Hertz.

◆ R_FSP_ClockDividerGet()

__STATIC_INLINE uint32_t R_FSP_ClockDividerGet (uint32_t *ckdivcr*)

Converts a clock's CKDIVCR register value to a clock divider (Eg: SPICKDIVCR).

Returns
Clock Divider

◆ R_BSP_UniqueIdGet()

__STATIC_INLINE bsp_unique_id_t const* R_BSP_UniqueIdGet (void)

Get unique ID for this device.

Returns

A pointer to the unique identifier structure

◆ R_BSP_PartNumberGet()

__STATIC_INLINE fsp_err_t R_BSP_PartNumberGet (bsp_part_number_t *const *p_part_number*)

Get part number for this device.

Parameters

[out]	p_part_number	Memory address to return MCU's part number to.

Return values

FSP_SUCCESS	Part number information stored.
FSP_ERR_ASSERTION	The parameter p_part_number is NULL.
FSP_ERR_NOT_FOUND	An error occurred when retrieving data from part number register.

◆ R_BSP_SoftwareDelay()

void R_BSP_SoftwareDelay (uint32_t *delay*, bsp_delay_units_t *units*)

Delay for at least the specified duration in units and return.

Parameters

[in]	delay	The number of 'units' to delay.
[in]	units	The 'base' (bsp_delay_units_t) for the units specified. Valid values are: BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS, BSP_DELAY_UNITS_MICROSECONDS. For example: At 1 MHz one cycle takes 1 microsecond (.000001 seconds). At 12 MHz one cycle takes

		1/12 microsecond or 83 nanoseconds. Therefore one run through bsp_prv_software_delay_loop() takes: ~ (83 * BSP_DELAY_LOOP_CYCLES) or 332 ns. A delay of 2 us therefore requires 2000ns/332ns or 6 loops.

The 'theoretical' maximum delay that may be obtained is determined by a full 32 bit loop count and the system clock rate. @120MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 120000000) = 143 seconds. @32MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 32000000) = 536 seconds

Note that requests for very large delays will be affected by rounding in the calculations and the actual delay achieved may be slightly longer. @32 MHz, for example, a request for 532 seconds will be closer to 536 seconds.

Note also that if the calculations result in a loop_cnt of zero, the bsp_prv_software_delay_loop() function is not called at all. In this case the requested delay is too small (nanoseconds) to be carried out by the loop itself, and the overhead associated with executing the code to just get to this point has certainly satisfied the requested delay.

Note

This function calls bsp_cpu_clock_get() which ultimately calls R_CGC_SystemClockFreqGet() and therefore requires that the BSP has already initialized the CGC (which it does as part of the Sysinit). Care should be taken to ensure this remains the case if in the future this function were to be called as part of the BSP initialization.
*This function will delay for **at least** the specified duration. Due to overhead in calculating the correct number of loops to delay, very small delay values (generally 1-5 microseconds) may be significantly longer than specified. Approximate overhead for this function is as follows:*

- *RXv3: 35 - 43 cycles*

If more accurate microsecond timing must be performed in software it is recommended to use bsp_prv_software_delay_loop() directly. In this case, use BSP_DELAY_LOOP_CYCLES or BSP_DELAY_LOOPS_CALCULATE() to convert a calculated delay cycle count to a number of software delay loops.
Delays may be longer than expected when compiler optimization is turned off.

◆ R_BSP_GroupIrqWrite()

fsp_err_t R_BSP_GroupIrqWrite (bsp_grp_irq_nmi_t *irq*, void(*)(bsp_grp_irq_nmi_t irq) *p_callback*)

Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.

Parameters

[in]	irq	Interrupt for which to register a callback.
[in]	p_callback	Pointer to function to call when interrupt occurs.

Return values

FSP_SUCCESS	Callback registered
FSP_ERR_ASSERTION	Callback pointer is NULL

◆ bsp_nmi_handler()

void bsp_nmi_handler (void)

Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because there are many sources that map to the NMI exception.

◆ R_BSP_GroupIrqCheck()

bool R_BSP_GroupIrqCheck (uint8_t *irq*)

Non-maskable interrupt handler.

◆ R_BSP_GetCurrentIFlagState()

uint32_t R_BSP_GetCurrentIFlagState (void)

Provides the current I flag state on PSW register.

Return values

0	Disabled interrupts.
1	Enabled interrupts.

◆ R_BSP_SetIFlagState()

void R_BSP_SetIFlagState (uint32_t *interrupt_mask*)

Set I flag state on PSW register. If this flag is 0, all interrupts are disabled.

Parameters

[in]	interrupt_mask	Value to set to the I-flag.

◆ R_BSP_GetCurrentIpl()

uint32_t R_BSP_GetCurrentIpl (void)

Get current interrupt priority level from PSW register.

Returns

Returns the value of interrupt priority level (IPL) currently registered in PSW.

◆ R_BSP_SetIpl()

void R_BSP_SetIpl (uint32_t *interrupt_mask_level*)

Set IPL state on PSW register. If the passed interrupt priority level (IPL) is maximum, all interrupts are disabled. If the passed IPL is less than the maximum, interrupts which IPL is above the set value will be processed.

Parameters

[in]	interrupt_mask_level	Interrupt priority to be masked.

◆ R_BSP_IrqIsEnabled()

uint32_t R_BSP_IrqIsEnabled (IRQn_Type const *irq*)

Get interrupt request enable status.

Parameters

[in]	irq	The IRQ to get interrupt request enable. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers.

Return values

0	Interrupt request is disabled.
1	Interrupt request is enabled.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqStatusClear()

void R_BSP_IrqStatusClear (IRQn_Type *irq*)

Clear the interrupt status flag (IR) for a given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately.

Parameters

[in]	irq	Interrupt for which to clear the IR bit. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqClearPending()

void R_BSP_IrqClearPending (IRQn_Type *irq*)

Clear the interrupt status flag (IR) for a given interrupt and clear the ICU pending interrupt.

Parameters

[in]	irq	Interrupt for which to clear the IR bit. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqCfg()

void R_BSP_IrqCfg (IRQn_Type const *irq*, uint32_t *priority*, void * *p_context*)

Sets the interrupt priority and context.

Parameters

[in]	irq	The IRQ to configure.
[in]	priority	ICU priority of the interrupt
[in]	p_context	The interrupt context is a pointer to data required in the ISR.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqEnableNoClear()

void R_BSP_IrqEnableNoClear (IRQn_Type const *irq*)

Enable the IRQ in the ICU (Without clearing the pending bit).

Parameters

[in]	irq	The IRQ to enable. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqEnable()

void R_BSP_IrqEnable (IRQn_Type const *irq*)

Clears pending interrupts in the ICU, then enables the interrupt.

Parameters

[in]	irq	Interrupt for which to clear the IR bit and enable in the ICU. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqDisable()

void R_BSP_IrqDisable (IRQn_Type const *irq*)

Disables interrupts in the ICU.

Parameters

[in]	irq	The IRQ to disable in the ICU. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_IrqCfgEnable()

void R_BSP_IrqCfgEnable (IRQn_Type const *irq*, uint32_t *priority*, void * *p_context*)

Sets the interrupt priority and context, clears pending interrupts, then enables the interrupt.

Parameters

[in]	irq	Interrupt number.
[in]	priority	ICU priority of the interrupt
[in]	p_context	The interrupt context is a pointer to data required in the ISR.

Warning

Do not call this function for system exceptions where the IRQn_Type value is < 0.

◆ R_BSP_RegisterProtectEnable()

void R_BSP_RegisterProtectEnable (bsp_reg_protect_t *regs_to_protect*)

Enable register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

Parameters

[in]	regs_to_protect	Registers which have write protection enabled.

◆ R_BSP_RegisterProtectDisable()

void R_BSP_RegisterProtectDisable (bsp_reg_protect_t *regs_to_unprotect*)

Disable register protection. Registers that are protected cannot be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

Parameters

[in]	regs_to_unprotect	Registers which have write protection disabled.

Variable Documentation

◆ BSP_SECTION_EARLY_INIT

uint32_t SystemCoreClock BSP_SECTION_EARLY_INIT
System Clock Frequency (Core Clock)

5.1.2.1 RX74M

BSP » MCU Board Support Package

Functions

void	bsp_selectable_irq_cfg (void)
__STATIC_INLINE void	bsp_set_selectable_irq_cfg (uint16_t *const p_base_addr, bsp_interrupt_event_t const *const p_selectable_irq_table, uint32_t size)
__STATIC_INLINE uint8_t	bsp_get_interrupt_priority (uint8_t irq) Get the interrupt priority register number from the IRQ. More...

Detailed Description

Build Time Configurations for rx74m_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
SDRAM > Timings > tRAS (cycles)	• 1 cycles • 2 cycles • 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles	6 cycles	Row Active Interval
SDRAM > Timings > tRCD (cycles)	• 1 cycles • 2 cycles • 3 cycles • 4 cycles	2 cycles	Row Column Latency
SDRAM > Timings > tRP (cycles)	• 1 cycles • 2 cycles • 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles • 8 cycles	2 cycles	Row Precharge Interval

SDRAM > Timings > tWR (cycles)	• 1 cycles • 2 cycles	2 cycles	Write Recovery Interval
SDRAM > Timings > tCL (cycles)	• 1 cycles • 2 cycles • 3 cycles	3 cycles	Column Latency
SDRAM > Timings > tRFC (cycles)	tRFC must be between 2 and 4096 cycles	1171	Auto-Refresh Request Interval Setting
SDRAM > Timings > tREFW (cycles)	Refer to the RX Configuration tool for available options.	5 cycles	Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting.
SDRAM > Initialization > Auto-Refresh Interval (ARFI)	Refer to the RX Configuration tool for available options.	5 cycles	Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence.
SDRAM > Initialization > Auto-Refresh Count (ARFC)	Refer to the RX Configuration tool for available options.	2 times	Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence.
SDRAM > Initialization > Precharge Cycle Count (PRC)	• 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles • 8 cycles • 9 cycles • 10 cycles	3 cycles	Specifies the number of precharged cycles in the SDRAM initialization sequence.
SDRAM > SDRAM Support	• Enabled • Disabled	Disabled	If enabled, SDRAM will be initialized and configured by the BSP with the provided settings.
SDRAM > Address Multiplex Shift	• 8-bit shift • 9-bit shift • 10-bit shift • 11-bit shift	8-bit shift	Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing.
SDRAM > Endian Mode	• Endian of SDRAM address space is the same as the endian of operating mode. • Endian of SDRAM address space is not the endian of	Endian of SDRAM address space is the same as the endian of operating mode.	Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian.

	operating mode.		
SDRAM > Continuous Access Mode	• Enabled • Disabled	Enabled	If enabled, SDRAM continuous access mode will be enabled.
SDRAM > Bus Width	• 8-bit • 16-bit • 32-bit	16-bit	Specifies the data bus width for SDRAM.
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Clocks > Clock Settling Delay	• Enabled • Disabled	Enabled	Setting this option to Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RX74M HWM (R01UH1055) section 9.11.1 for details.
Clocks > Sleep Mode Entry and Exit Delays	• Enabled • Disabled	Enabled	Setting this option to Enabled will insert delays before and after entering sleep modes to ensure voltage supply stability.
Clocks > RTOS Sleep	• Enabled	Enabled	Setting this option to

Mode Entry and Exit Delays	• Disabled		Enabled will insert delays before and after WFI calls in the RTOS to ensure voltage supply stability.
Clocks > Settling Delay (us)	Unit must be a non-negative integer	150	Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays.
Clocks > Main Clock Oscillator Drive Capability	• Auto • 8MHz • 8MHz to 24MHz • 8MHz to 48MHz	Auto	Main oscillator drive setting. Auto uses FSP default based on XTAL frequency.
Cache settings > Data cache	• Enabled • Disabled	Disabled	Enable limited D-Cache support. See BSP usage notes for limitations.
Cache settings > Instruction cache	• Enabled • Disabled	Disabled	Enable limited I-Cache support. See BSP usage notes for limitations.
Security > Exceptions > Exception Response	• Interrupt • Reset	Interrupt	Specifies the behavior when an access violation is detected. This exception is generated when an access to a Secure area from a Non-Secure master is detected. This setting is only valid when building projects with ProtectZone.
Security > Exceptions > NMI Target	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the NMI exception. This setting is only valid when building projects with ProtectZone.
Security > BUS Accessibility > Bus Security Attribution Register A	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes for the arbitration and throughput control registers. This setting is only valid when building projects with ProtectZone.
Security > BUS	• Secure State	Non-Secure State	Defines the security

Accessibility > Bus Security Attribution Register B	• Non-Secure State		attributes of registers related to bus errors. This setting is only valid when building projects with ProtectZone.
Security > BUS Accessibility > Bus Security Attribution Register C	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes for the SDRAM/CSC control registers. This setting is only valid when building projects with ProtectZone.
Security > SRAM Accessibility > SRAM0 Protection	• Secure State • Non-Secure State	Non-Secure State	Define the security attributes for SRAMCR0, SRAMECCRGN0, SRAMESCLR.CLR00, and SRAMESCLR.CLR01. This setting is only valid when building projects with ProtectZone.
Security > SRAM Accessibility > SRAM1 Protection	• Secure State • Non-Secure State	Non-Secure State	Define the security attributes for SRAMCR1, SRAMECCRGN1, SRAMESCLR.CLR10, and SRAMESCLR.CLR11. This setting is only valid when building projects with ProtectZone.
Security > LRAM Accessibility > LRAM Protection	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of LRMCRx and LRMESCLR. This setting is only valid when building projects with ProtectZone.
Security > Number of SSC Table Entries	1	1	The number of SSC Table Entries used by the ProtectZone hypervisor.
Security > Cache Accessibility	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the cash registers. This setting is only

			valid when building projects with ProtectZone.
Security > Branch Instruction Processing Accessibility	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the Branch Instruction Processing Accelerator registers. This setting is only valid when building projects with ProtectZone.
Security > System Reset Status Accessibility	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes for the Reset Status Registers (RSTSRn) and Reset Output Status Registers (ROUTSRn). This setting is only valid when building projects with ProtectZone.
Security > Battery Backup Accessibility	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the battery backup register. This setting is only valid when building projects with ProtectZone.
Security > Uninitialized Non-Secure Application Fallback	• Enable Uninitialized Non-Secure Application Fallback • Disable Uninitialized Non-Secure Application Fallback	Enable Uninitialized Non-Secure Application Fallback	If enabled, the secure application checks if the non-secure application has been programmed in non-secure Code MRAM before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM.
I/O Ports > Voltage Mode > VCC	• Low (VCC < 2.7 V) • High (VCC >= 2.7 V)	High (VCC >= 2.7 V)	When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low.
Enable inline BSP IRQ functions	• Enabled • Disabled	Enabled	Using static inline functions will slightly increase code size, but

			will slightly decrease cycles taken in ISRs in return.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.
Prefetch Buffer	• Enabled • Disabled	Disabled	Enabling this option enables Prefetch buffer function in the BSP.

Macros

#define	BSP_ELC_PERIPHERAL_MASK

Enumerations

enum	elc_peripheral_t
enum	elc_software_event_t
enum	elc_interrupt_t
enum	icu_event_t
enum	bsp_grp_irq_t

Macro Definition Documentation

◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK
Positions of event link set registers (ELSRs) available on this MCU

Enumeration Type Documentation

◆ elc_peripheral_t

enum elc_peripheral_t
Possible peripherals to be linked to event signals ***Note*** *This list is device specific.*

◆ elc_software_event_t

enum elc_software_event_t	
Possible software events ***Note*** *This list is device specific.*	
Enumerator	
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_2	Software event 2.
ELC_SOFTWARE_EVENT_3	Software event 3.
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_2	Software event 2.
ELC_SOFTWARE_EVENT_3	Software event 3.

◆ elc_interrupt_t

enum elc_interrupt_t	
Event link possible interrupts	
Enumerator	
ELC_INTERRUPT_1	Interrupt 1.
ELC_INTERRUPT_2	Interrupt 2.
ELC_INTERRUPT_3	Interrupt 3.
ELC_INTERRUPT_4	Interrupt 4.

◆ icu_event_t

enum icu_event_t

Events to be used with the IELSR register to link interrupt events to the NVIC

Note
This list is device specific.

◆ bsp_grp_irq_t

enum bsp_grp_irq_t

Which interrupts can have callbacks registered.

Function Documentation

◆ bsp_selectable_irq_cfg()

void bsp_selectable_irq_cfg (void)

Configure selectable irq depend on RX MCU. RX74M has three selectable irq register(SLIXR, SLIXR).

◆ bsp_set_selectable_irq_cfg()

__STATIC_INLINE void bsp_set_selectable_irq_cfg (uint16_t *const *p_base_addr*, bsp_interrupt_event_t const *const *p_selectable_irq_table*, uint32_t *size*)

Set selectable irq table value that user setting to selectable irq register.

Parameters

[out]	p_base_addr	The p_base_addr is write base address of selectable irq register.
[in]	p_selectable_irq_table	The p_selectable_irq_table is uesr setting selectable irq table.
[in]	size	The size is read range of p_selectable_irq_table array.

Warning
Do not call this function with values where size over p_selectable_irq_table allowd range.

◆ bsp_get_interrupt_priority()

__STATIC_INLINE uint8_t bsp_get_interrupt_priority (uint8_t *irq*)

Get the interrupt priority register number from the IRQ.

Parameters

[in]	irq	The IRQ to get the the interrupt priority register number

Returns
interrupt priority register number

5.1.2.2 RX74N

BSP » MCU Board Support Package

Functions

void	bsp_selectable_irq_cfg (void)
__STATIC_INLINE void	bsp_set_selectable_irq_cfg (uint16_t *const p_base_addr, bsp_interrupt_event_t const *const p_selectable_irq_table, uint32_t size)
__STATIC_INLINE uint8_t	bsp_get_interrupt_priority (uint8_t irq) Get the interrupt priority register number from the IRQ. More...

Detailed Description

Build Time Configurations for rx74n_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
SDRAM > Timings > tRAS (cycles)	• 1 cycles • 2 cycles • 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles	6 cycles	Row Active Interval
SDRAM > Timings > tRCD (cycles)	• 1 cycles • 2 cycles • 3 cycles	2 cycles	Row Column Latency

	• 4 cycles		
SDRAM > Timings > tRP (cycles)	• 1 cycles • 2 cycles • 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles • 8 cycles	2 cycles	Row Precharge Interval
SDRAM > Timings > tWR (cycles)	• 1 cycles • 2 cycles	2 cycles	Write Recovery Interval
SDRAM > Timings > tCL (cycles)	• 1 cycles • 2 cycles • 3 cycles	3 cycles	Column Latency
SDRAM > Timings > tRFC (cycles)	tRFC must be between 2 and 4096 cycles	1171	Auto-Refresh Request Interval Setting
SDRAM > Timings > tREFW (cycles)	Refer to the RX Configuration tool for available options.	5 cycles	Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting.
SDRAM > Initialization > Auto-Refresh Interval (ARFI)	Refer to the RX Configuration tool for available options.	5 cycles	Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence.
SDRAM > Initialization > Auto-Refresh Count (ARFC)	Refer to the RX Configuration tool for available options.	2 times	Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence.
SDRAM > Initialization > Precharge Cycle Count (PRC)	• 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles • 8 cycles • 9 cycles • 10 cycles	3 cycles	Specifies the number of precharged cycles in the SDRAM initialization sequence.
SDRAM > SDRAM Support	• Enabled • Disabled	Disabled	If enabled, SDRAM will be initialized and configured by the BSP with the provided settings.
SDRAM > Address Multiplex Shift	• 8-bit shift • 9-bit shift • 10-bit shift • 11-bit shift	8-bit shift	Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing.
SDRAM > Endian Mode	• Endian of	Endian of SDRAM	Specifies the

	SDRAM address space is the same as the endian of operating mode. • Endian of SDRAM address space is not the endian of operating mode.	address space is the same as the endian of operating mode.	endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian.
SDRAM > Continuous Access Mode	• Enabled • Disabled	Enabled	If enabled, SDRAM continuous access mode will be enabled.
SDRAM > Bus Width	• 8-bit • 16-bit • 32-bit	16-bit	Specifies the data bus width for SDRAM.
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Clocks > Clock Settling Delay	• Enabled • Disabled	Enabled	Setting this option to Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RX74N HWM

			(R01UH1056) section 9.11.1 for details.
Clocks > Sleep Mode Entry and Exit Delays	• Enabled • Disabled	Enabled	Setting this option to Enabled will insert delays before and after entering sleep modes to ensure voltage supply stability.
Clocks > RTOS Sleep Mode Entry and Exit Delays	• Enabled • Disabled	Enabled	Setting this option to Enabled will insert delays before and after WFI calls in the RTOS to ensure voltage supply stability.
Clocks > Settling Delay (us)	Unit must be a non-negative integer	150	Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays.
Clocks > Main Clock Oscillator Drive Capability	• Auto • 8MHz • 8MHz to 24MHz • 8MHz to 48MHz	Auto	Main oscillator drive setting. Auto uses FSP default based on XTAL frequency.
Cache settings > Data cache	• Enabled • Disabled	Disabled	Enable limited D-Cache support. See BSP usage notes for limitations.
Cache settings > Instruction cache	• Enabled • Disabled	Disabled	Enable limited I-Cache support. See BSP usage notes for limitations.
Security > Exceptions > Exception Response	• Interrupt • Reset	Interrupt	Specifies the behavior when an access violation is detected. This exception is generated when an access to a Secure area from a Non-Secure master is detected. This setting is only valid when building projects with ProtectZone.
Security > Exceptions > NMI Target	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the NMI exception. This setting is only valid when building projects with ProtectZone.

Security > BUS Accessibility > Bus Security Attribution Register A	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes for the arbitration and throughput control registers. This setting is only valid when building projects with ProtectZone.
Security > BUS Accessibility > Bus Security Attribution Register B	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of registers related to bus errors. This setting is only valid when building projects with ProtectZone.
Security > BUS Accessibility > Bus Security Attribution Register C	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes for the SDRAM/CSC control registers. This setting is only valid when building projects with ProtectZone.
Security > SRAM Accessibility > SRAM0 Protection	• Secure State • Non-Secure State	Non-Secure State	Define the security attributes for SRAMCR0, SRAMECCRGN0, SRAMESCLR.CLR00, and SRAMESCLR.CLR01. This setting is only valid when building projects with ProtectZone.
Security > SRAM Accessibility > SRAM1 Protection	• Secure State • Non-Secure State	Non-Secure State	Define the security attributes for SRAMCR1, SRAMECCRGN1, SRAMESCLR.CLR10, and SRAMESCLR.CLR11. This setting is only valid when building projects with ProtectZone.
Security > LRAM Accessibility > LRAM Protection	• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of LRMCRx and LRMESCLR. This setting is only valid when building projects with

				ProtectZone.
Security > Number of SSC Table Entries	1		1	The number of SSC Table Entries used by the ProtectZone hypervisor.
Security > Cache Accessibility		• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the cash registers. This setting is only valid when building projects with ProtectZone.
Security > Branch Instruction Processing Accessibility		• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the Branch Instruction Processing Accelerator registers. This setting is only valid when building projects with ProtectZone.
Security > System Reset Status Accessibility		• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes for the Reset Status Registers (RSTSRn) and Reset Output Status Registers (ROUTSRn). This setting is only valid when building projects with ProtectZone.
Security > Battery Backup Accessibility		• Secure State • Non-Secure State	Non-Secure State	Defines the security attributes of the battery backup register. This setting is only valid when building projects with ProtectZone.
Security > Uninitialized Non-Secure Application Fallback		• Enable Uninitialized Non-Secure Application Fallback • Disable Uninitialized Non-Secure Application Fallback	Enable Uninitialized Non-Secure Application Fallback	If enabled, the secure application checks if the non-secure application has been programmed in non-secure Code MRAM before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite

			loop in non-secure RAM.
I/O Ports > Voltage Mode > VCC	• Low (VCC < 2.7 V) • High (VCC >= 2.7 V)	High (VCC >= 2.7 V)	When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low.
Enable inline BSP IRQ functions	• Enabled • Disabled	Enabled	Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.
Prefetch Buffer	• Enabled • Disabled	Disabled	Enabling this option enables Prefetch buffer function in the BSP.

Macros

#define	BSP_ELC_PERIPHERAL_MASK

Enumerations

enum	elc_peripheral_t
enum	elc_software_event_t
enum	elc_interrupt_t
enum	icu_event_t
enum	bsp_grp_irq_t

Macro Definition Documentation

◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK
Positions of event link set registers (ELSRs) available on this MCU

Enumeration Type Documentation

◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

Note
This list is device specific.

◆ elc_software_event_t

enum elc_software_event_t

Possible software events

Note
This list is device specific.

Enumerator	
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_2	Software event 2.
ELC_SOFTWARE_EVENT_3	Software event 3.
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_2	Software event 2.
ELC_SOFTWARE_EVENT_3	Software event 3.

◆ elc_interrupt_t

enum elc_interrupt_t	
Enumerator	
ELC_INTERRUPT_1	Interrupt 1.
ELC_INTERRUPT_2	Interrupt 2.
ELC_INTERRUPT_3	Interrupt 3.
ELC_INTERRUPT_4	Interrupt 4.

◆ icu_event_t

enum icu_event_t
Events to be used with the IELSR register to link interrupt events to the NVIC ***Note*** *This list is device specific.*

◆ bsp_grp_irq_t

enum bsp_grp_irq_t
Which interrupts can have callbacks registered.

Function Documentation

◆ bsp_selectable_irq_cfg()

void bsp_selectable_irq_cfg (void)
Configure selectable irq depend on RX MCU. RX74M has three selectable irq register(SLIXR, SLIXR).

◆ bsp_set_selectable_irq_cfg()

__STATIC_INLINE void bsp_set_selectable_irq_cfg (uint16_t *const *p_base_addr*, bsp_interrupt_event_t const *const *p_selectable_irq_table*, uint32_t *size*)

Set selectable irq table value that user setting to selectable irq register.

Parameters

[out]	p_base_addr	The p_base_addr is write base address of selectable irq register.
[in]	p_selectable_irq_table	The p_selectable_irq_table is uesr setting selectable irq table.
[in]	size	The size is read range of p_selectable_irq_table array.

Warning

Do not call this function with values where size over p_selectable_irq_table allowd range.

◆ bsp_get_interrupt_priority()

__STATIC_INLINE uint8_t bsp_get_interrupt_priority (uint8_t *irq*)

Get the interrupt priority register number from the IRQ.

Parameters

[in]	irq	The IRQ to get the the interrupt priority register number

Returns

interrupt priority register number

5.1.3 BSP I/O access

BSP

Functions

__STATIC_INLINE uint32_t	R_BSP_PinRead (bsp_io_port_pin_t pin)
__STATIC_INLINE void	R_BSP_PinWrite (bsp_io_port_pin_t pin, bsp_io_level_t level)
__STATIC_INLINE void	R_BSP_PinCfg (bsp_io_port_pin_t pin, uint32_t cfg)

__STATIC_INLINE void	R_BSP_PinAccessEnable (void)
__STATIC_INLINE void	R_BSP_PinAccessDisable (void)

Detailed Description

This module provides basic read/write access to port pins.

Enumerations

enum	bsp_io_level_t
enum	bsp_io_direction_t
enum	bsp_io_port_t
enum	bsp_io_port_pin_t

Enumeration Type Documentation

◆ bsp_io_level_t

enum bsp_io_level_t	
Levels that can be set and read for individual pins	
Enumerator	
BSP_IO_LEVEL_LOW	Low.
BSP_IO_LEVEL_HIGH	High.

◆ bsp_io_direction_t

enum bsp_io_direction_t	
Direction of individual pins	
Enumerator	
BSP_IO_DIRECTION_INPUT	Input.
BSP_IO_DIRECTION_OUTPUT	Output.

◆ bsp_io_port_t

enum bsp_io_port_t	
Superset list of all possible IO ports.	
Enumerator	
BSP_IO_PORT_00	IO port 0.
BSP_IO_PORT_01	IO port 1.
BSP_IO_PORT_02	IO port 2.
BSP_IO_PORT_03	IO port 3.
BSP_IO_PORT_04	IO port 4.
BSP_IO_PORT_05	IO port 5.
BSP_IO_PORT_06	IO port 6.
BSP_IO_PORT_07	IO port 7.
BSP_IO_PORT_08	IO port 8.
BSP_IO_PORT_09	IO port 9.
BSP_IO_PORT_10	IO port 10.
BSP_IO_PORT_11	IO port 11.
BSP_IO_PORT_12	IO port 12.
BSP_IO_PORT_13	IO port 13.
BSP_IO_PORT_14	IO port 14.
BSP_IO_PORT_15	IO port 15.
BSP_IO_PORT_16	IO port 16.

◆ bsp_io_port_pin_t

enum bsp_io_port_pin_t	
Superset list of all possible IO port pins.	
Enumerator	
BSP_IO_PORT_00_PIN_00	IO port 0 pin 0.
BSP_IO_PORT_00_PIN_01	IO port 0 pin 1.
BSP_IO_PORT_00_PIN_02	IO port 0 pin 2.
BSP_IO_PORT_00_PIN_03	IO port 0 pin 3.
BSP_IO_PORT_00_PIN_04	IO port 0 pin 4.
BSP_IO_PORT_00_PIN_05	IO port 0 pin 5.
BSP_IO_PORT_00_PIN_06	IO port 0 pin 6.
BSP_IO_PORT_00_PIN_07	IO port 0 pin 7.
BSP_IO_PORT_00_PIN_08	IO port 0 pin 8.
BSP_IO_PORT_00_PIN_09	IO port 0 pin 9.
BSP_IO_PORT_00_PIN_10	IO port 0 pin 10.
BSP_IO_PORT_00_PIN_11	IO port 0 pin 11.
BSP_IO_PORT_00_PIN_12	IO port 0 pin 12.
BSP_IO_PORT_00_PIN_13	IO port 0 pin 13.
BSP_IO_PORT_00_PIN_14	IO port 0 pin 14.
BSP_IO_PORT_00_PIN_15	IO port 0 pin 15.
BSP_IO_PORT_01_PIN_00	IO port 1 pin 0.
BSP_IO_PORT_01_PIN_01	IO port 1 pin 1.
BSP_IO_PORT_01_PIN_02	IO port 1 pin 2.
BSP_IO_PORT_01_PIN_03	IO port 1 pin 3.
BSP_IO_PORT_01_PIN_04	IO port 1 pin 4.

BSP_IO_PORT_01_PIN_05	IO port 1 pin 5.
BSP_IO_PORT_01_PIN_06	IO port 1 pin 6.
BSP_IO_PORT_01_PIN_07	IO port 1 pin 7.
BSP_IO_PORT_01_PIN_08	IO port 1 pin 8.
BSP_IO_PORT_01_PIN_09	IO port 1 pin 9.
BSP_IO_PORT_01_PIN_10	IO port 1 pin 10.
BSP_IO_PORT_01_PIN_11	IO port 1 pin 11.
BSP_IO_PORT_01_PIN_12	IO port 1 pin 12.
BSP_IO_PORT_01_PIN_13	IO port 1 pin 13.
BSP_IO_PORT_01_PIN_14	IO port 1 pin 14.
BSP_IO_PORT_01_PIN_15	IO port 1 pin 15.
BSP_IO_PORT_02_PIN_00	IO port 2 pin 0.
BSP_IO_PORT_02_PIN_01	IO port 2 pin 1.
BSP_IO_PORT_02_PIN_02	IO port 2 pin 2.
BSP_IO_PORT_02_PIN_03	IO port 2 pin 3.
BSP_IO_PORT_02_PIN_04	IO port 2 pin 4.
BSP_IO_PORT_02_PIN_05	IO port 2 pin 5.
BSP_IO_PORT_02_PIN_06	IO port 2 pin 6.
BSP_IO_PORT_02_PIN_07	IO port 2 pin 7.
BSP_IO_PORT_02_PIN_08	IO port 2 pin 8.
BSP_IO_PORT_02_PIN_09	IO port 2 pin 9.
BSP_IO_PORT_02_PIN_10	IO port 2 pin 10.
BSP_IO_PORT_02_PIN_11	IO port 2 pin 11.
BSP_IO_PORT_02_PIN_12	IO port 2 pin 12.

BSP_IO_PORT_02_PIN_13	IO port 2 pin 13.
BSP_IO_PORT_02_PIN_14	IO port 2 pin 14.
BSP_IO_PORT_02_PIN_15	IO port 2 pin 15.
BSP_IO_PORT_03_PIN_00	IO port 3 pin 0.
BSP_IO_PORT_03_PIN_01	IO port 3 pin 1.
BSP_IO_PORT_03_PIN_02	IO port 3 pin 2.
BSP_IO_PORT_03_PIN_03	IO port 3 pin 3.
BSP_IO_PORT_03_PIN_04	IO port 3 pin 4.
BSP_IO_PORT_03_PIN_05	IO port 3 pin 5.
BSP_IO_PORT_03_PIN_06	IO port 3 pin 6.
BSP_IO_PORT_03_PIN_07	IO port 3 pin 7.
BSP_IO_PORT_03_PIN_08	IO port 3 pin 8.
BSP_IO_PORT_03_PIN_09	IO port 3 pin 9.
BSP_IO_PORT_03_PIN_10	IO port 3 pin 10.
BSP_IO_PORT_03_PIN_11	IO port 3 pin 11.
BSP_IO_PORT_03_PIN_12	IO port 3 pin 12.
BSP_IO_PORT_03_PIN_13	IO port 3 pin 13.
BSP_IO_PORT_03_PIN_14	IO port 3 pin 14.
BSP_IO_PORT_03_PIN_15	IO port 3 pin 15.
BSP_IO_PORT_04_PIN_00	IO port 4 pin 0.
BSP_IO_PORT_04_PIN_01	IO port 4 pin 1.
BSP_IO_PORT_04_PIN_02	IO port 4 pin 2.
BSP_IO_PORT_04_PIN_03	IO port 4 pin 3.
BSP_IO_PORT_04_PIN_04	IO port 4 pin 4.

BSP_IO_PORT_04_PIN_05	IO port 4 pin 5.
BSP_IO_PORT_04_PIN_06	IO port 4 pin 6.
BSP_IO_PORT_04_PIN_07	IO port 4 pin 7.
BSP_IO_PORT_04_PIN_08	IO port 4 pin 8.
BSP_IO_PORT_04_PIN_09	IO port 4 pin 9.
BSP_IO_PORT_04_PIN_10	IO port 4 pin 10.
BSP_IO_PORT_04_PIN_11	IO port 4 pin 11.
BSP_IO_PORT_04_PIN_12	IO port 4 pin 12.
BSP_IO_PORT_04_PIN_13	IO port 4 pin 13.
BSP_IO_PORT_04_PIN_14	IO port 4 pin 14.
BSP_IO_PORT_04_PIN_15	IO port 4 pin 15.
BSP_IO_PORT_05_PIN_00	IO port 5 pin 0.
BSP_IO_PORT_05_PIN_01	IO port 5 pin 1.
BSP_IO_PORT_05_PIN_02	IO port 5 pin 2.
BSP_IO_PORT_05_PIN_03	IO port 5 pin 3.
BSP_IO_PORT_05_PIN_04	IO port 5 pin 4.
BSP_IO_PORT_05_PIN_05	IO port 5 pin 5.
BSP_IO_PORT_05_PIN_06	IO port 5 pin 6.
BSP_IO_PORT_05_PIN_07	IO port 5 pin 7.
BSP_IO_PORT_05_PIN_08	IO port 5 pin 8.
BSP_IO_PORT_05_PIN_09	IO port 5 pin 9.
BSP_IO_PORT_05_PIN_10	IO port 5 pin 10.
BSP_IO_PORT_05_PIN_11	IO port 5 pin 11.
BSP_IO_PORT_05_PIN_12	IO port 5 pin 12.

BSP_IO_PORT_05_PIN_13	IO port 5 pin 13.
BSP_IO_PORT_05_PIN_14	IO port 5 pin 14.
BSP_IO_PORT_05_PIN_15	IO port 5 pin 15.
BSP_IO_PORT_06_PIN_00	IO port 6 pin 0.
BSP_IO_PORT_06_PIN_01	IO port 6 pin 1.
BSP_IO_PORT_06_PIN_02	IO port 6 pin 2.
BSP_IO_PORT_06_PIN_03	IO port 6 pin 3.
BSP_IO_PORT_06_PIN_04	IO port 6 pin 4.
BSP_IO_PORT_06_PIN_05	IO port 6 pin 5.
BSP_IO_PORT_06_PIN_06	IO port 6 pin 6.
BSP_IO_PORT_06_PIN_07	IO port 6 pin 7.
BSP_IO_PORT_06_PIN_08	IO port 6 pin 8.
BSP_IO_PORT_06_PIN_09	IO port 6 pin 9.
BSP_IO_PORT_06_PIN_10	IO port 6 pin 10.
BSP_IO_PORT_06_PIN_11	IO port 6 pin 11.
BSP_IO_PORT_06_PIN_12	IO port 6 pin 12.
BSP_IO_PORT_06_PIN_13	IO port 6 pin 13.
BSP_IO_PORT_06_PIN_14	IO port 6 pin 14.
BSP_IO_PORT_06_PIN_15	IO port 6 pin 15.
BSP_IO_PORT_07_PIN_00	IO port 7 pin 0.
BSP_IO_PORT_07_PIN_01	IO port 7 pin 1.
BSP_IO_PORT_07_PIN_02	IO port 7 pin 2.
BSP_IO_PORT_07_PIN_03	IO port 7 pin 3.
BSP_IO_PORT_07_PIN_04	IO port 7 pin 4.

BSP_IO_PORT_07_PIN_05	IO port 7 pin 5.
BSP_IO_PORT_07_PIN_06	IO port 7 pin 6.
BSP_IO_PORT_07_PIN_07	IO port 7 pin 7.
BSP_IO_PORT_07_PIN_08	IO port 7 pin 8.
BSP_IO_PORT_07_PIN_09	IO port 7 pin 9.
BSP_IO_PORT_07_PIN_10	IO port 7 pin 10.
BSP_IO_PORT_07_PIN_11	IO port 7 pin 11.
BSP_IO_PORT_07_PIN_12	IO port 7 pin 12.
BSP_IO_PORT_07_PIN_13	IO port 7 pin 13.
BSP_IO_PORT_07_PIN_14	IO port 7 pin 14.
BSP_IO_PORT_07_PIN_15	IO port 7 pin 15.
BSP_IO_PORT_08_PIN_00	IO port 8 pin 0.
BSP_IO_PORT_08_PIN_01	IO port 8 pin 1.
BSP_IO_PORT_08_PIN_02	IO port 8 pin 2.
BSP_IO_PORT_08_PIN_03	IO port 8 pin 3.
BSP_IO_PORT_08_PIN_04	IO port 8 pin 4.
BSP_IO_PORT_08_PIN_05	IO port 8 pin 5.
BSP_IO_PORT_08_PIN_06	IO port 8 pin 6.
BSP_IO_PORT_08_PIN_07	IO port 8 pin 7.
BSP_IO_PORT_08_PIN_08	IO port 8 pin 8.
BSP_IO_PORT_08_PIN_09	IO port 8 pin 9.
BSP_IO_PORT_08_PIN_10	IO port 8 pin 10.
BSP_IO_PORT_08_PIN_11	IO port 8 pin 11.
BSP_IO_PORT_08_PIN_12	IO port 8 pin 12.

BSP_IO_PORT_08_PIN_13	IO port 8 pin 13.
BSP_IO_PORT_08_PIN_14	IO port 8 pin 14.
BSP_IO_PORT_08_PIN_15	IO port 8 pin 15.
BSP_IO_PORT_09_PIN_00	IO port 9 pin 0.
BSP_IO_PORT_09_PIN_01	IO port 9 pin 1.
BSP_IO_PORT_09_PIN_02	IO port 9 pin 2.
BSP_IO_PORT_09_PIN_03	IO port 9 pin 3.
BSP_IO_PORT_09_PIN_04	IO port 9 pin 4.
BSP_IO_PORT_09_PIN_05	IO port 9 pin 5.
BSP_IO_PORT_09_PIN_06	IO port 9 pin 6.
BSP_IO_PORT_09_PIN_07	IO port 9 pin 7.
BSP_IO_PORT_09_PIN_08	IO port 9 pin 8.
BSP_IO_PORT_09_PIN_09	IO port 9 pin 9.
BSP_IO_PORT_09_PIN_10	IO port 9 pin 10.
BSP_IO_PORT_09_PIN_11	IO port 9 pin 11.
BSP_IO_PORT_09_PIN_12	IO port 9 pin 12.
BSP_IO_PORT_09_PIN_13	IO port 9 pin 13.
BSP_IO_PORT_09_PIN_14	IO port 9 pin 14.
BSP_IO_PORT_09_PIN_15	IO port 9 pin 15.
BSP_IO_PORT_10_PIN_00	IO port 10 pin 0.
BSP_IO_PORT_10_PIN_01	IO port 10 pin 1.
BSP_IO_PORT_10_PIN_02	IO port 10 pin 2.
BSP_IO_PORT_10_PIN_03	IO port 10 pin 3.
BSP_IO_PORT_10_PIN_04	IO port 10 pin 4.

BSP_IO_PORT_10_PIN_05	IO port 10 pin 5.
BSP_IO_PORT_10_PIN_06	IO port 10 pin 6.
BSP_IO_PORT_10_PIN_07	IO port 10 pin 7.
BSP_IO_PORT_10_PIN_08	IO port 10 pin 8.
BSP_IO_PORT_10_PIN_09	IO port 10 pin 9.
BSP_IO_PORT_10_PIN_10	IO port 10 pin 10.
BSP_IO_PORT_10_PIN_11	IO port 10 pin 11.
BSP_IO_PORT_10_PIN_12	IO port 10 pin 12.
BSP_IO_PORT_10_PIN_13	IO port 10 pin 13.
BSP_IO_PORT_10_PIN_14	IO port 10 pin 14.
BSP_IO_PORT_10_PIN_15	IO port 10 pin 15.
BSP_IO_PORT_11_PIN_00	IO port 11 pin 0.
BSP_IO_PORT_11_PIN_01	IO port 11 pin 1.
BSP_IO_PORT_11_PIN_02	IO port 11 pin 2.
BSP_IO_PORT_11_PIN_03	IO port 11 pin 3.
BSP_IO_PORT_11_PIN_04	IO port 11 pin 4.
BSP_IO_PORT_11_PIN_05	IO port 11 pin 5.
BSP_IO_PORT_11_PIN_06	IO port 11 pin 6.
BSP_IO_PORT_11_PIN_07	IO port 11 pin 7.
BSP_IO_PORT_11_PIN_08	IO port 11 pin 8.
BSP_IO_PORT_11_PIN_09	IO port 11 pin 9.
BSP_IO_PORT_11_PIN_10	IO port 11 pin 10.
BSP_IO_PORT_11_PIN_11	IO port 11 pin 11.
BSP_IO_PORT_11_PIN_12	IO port 11 pin 12.

BSP_IO_PORT_11_PIN_13	IO port 11 pin 13.
BSP_IO_PORT_11_PIN_14	IO port 11 pin 14.
BSP_IO_PORT_11_PIN_15	IO port 11 pin 15.
BSP_IO_PORT_12_PIN_00	IO port 12 pin 0.
BSP_IO_PORT_12_PIN_01	IO port 12 pin 1.
BSP_IO_PORT_12_PIN_02	IO port 12 pin 2.
BSP_IO_PORT_12_PIN_03	IO port 12 pin 3.
BSP_IO_PORT_12_PIN_04	IO port 12 pin 4.
BSP_IO_PORT_12_PIN_05	IO port 12 pin 5.
BSP_IO_PORT_12_PIN_06	IO port 12 pin 6.
BSP_IO_PORT_12_PIN_07	IO port 12 pin 7.
BSP_IO_PORT_12_PIN_08	IO port 12 pin 8.
BSP_IO_PORT_12_PIN_09	IO port 12 pin 9.
BSP_IO_PORT_12_PIN_10	IO port 12 pin 10.
BSP_IO_PORT_12_PIN_11	IO port 12 pin 11.
BSP_IO_PORT_12_PIN_12	IO port 12 pin 12.
BSP_IO_PORT_12_PIN_13	IO port 12 pin 13.
BSP_IO_PORT_12_PIN_14	IO port 12 pin 14.
BSP_IO_PORT_12_PIN_15	IO port 12 pin 15.
BSP_IO_PORT_13_PIN_00	IO port 13 pin 0.
BSP_IO_PORT_13_PIN_01	IO port 13 pin 1.
BSP_IO_PORT_13_PIN_02	IO port 13 pin 2.
BSP_IO_PORT_13_PIN_03	IO port 13 pin 3.
BSP_IO_PORT_13_PIN_04	IO port 13 pin 4.

BSP_IO_PORT_13_PIN_05	IO port 13 pin 5.
BSP_IO_PORT_13_PIN_06	IO port 13 pin 6.
BSP_IO_PORT_13_PIN_07	IO port 13 pin 7.
BSP_IO_PORT_13_PIN_08	IO port 13 pin 8.
BSP_IO_PORT_13_PIN_09	IO port 13 pin 9.
BSP_IO_PORT_13_PIN_10	IO port 13 pin 10.
BSP_IO_PORT_13_PIN_11	IO port 13 pin 11.
BSP_IO_PORT_13_PIN_12	IO port 13 pin 12.
BSP_IO_PORT_13_PIN_13	IO port 13 pin 13.
BSP_IO_PORT_13_PIN_14	IO port 13 pin 14.
BSP_IO_PORT_13_PIN_15	IO port 13 pin 15.
BSP_IO_PORT_14_PIN_00	IO port 14 pin 0.
BSP_IO_PORT_14_PIN_01	IO port 14 pin 1.
BSP_IO_PORT_14_PIN_02	IO port 14 pin 2.
BSP_IO_PORT_14_PIN_03	IO port 14 pin 3.
BSP_IO_PORT_14_PIN_04	IO port 14 pin 4.
BSP_IO_PORT_14_PIN_05	IO port 14 pin 5.
BSP_IO_PORT_14_PIN_06	IO port 14 pin 6.
BSP_IO_PORT_14_PIN_07	IO port 14 pin 7.
BSP_IO_PORT_14_PIN_08	IO port 14 pin 8.
BSP_IO_PORT_14_PIN_09	IO port 14 pin 9.
BSP_IO_PORT_14_PIN_10	IO port 14 pin 10.
BSP_IO_PORT_14_PIN_11	IO port 14 pin 11.
BSP_IO_PORT_14_PIN_12	IO port 14 pin 12.

BSP_IO_PORT_14_PIN_13	IO port 14 pin 13.
BSP_IO_PORT_14_PIN_14	IO port 14 pin 14.
BSP_IO_PORT_14_PIN_15	IO port 14 pin 15.
BSP_IO_PORT_15_PIN_00	IO port 15 pin 0.
BSP_IO_PORT_15_PIN_01	IO port 15 pin 1.
BSP_IO_PORT_15_PIN_02	IO port 15 pin 2.
BSP_IO_PORT_15_PIN_03	IO port 15 pin 3.
BSP_IO_PORT_15_PIN_04	IO port 15 pin 4.
BSP_IO_PORT_15_PIN_05	IO port 15 pin 5.
BSP_IO_PORT_15_PIN_06	IO port 15 pin 6.
BSP_IO_PORT_15_PIN_07	IO port 15 pin 7.
BSP_IO_PORT_15_PIN_08	IO port 15 pin 8.
BSP_IO_PORT_15_PIN_09	IO port 15 pin 9.
BSP_IO_PORT_15_PIN_10	IO port 15 pin 10.
BSP_IO_PORT_15_PIN_11	IO port 15 pin 11.
BSP_IO_PORT_15_PIN_12	IO port 15 pin 12.
BSP_IO_PORT_15_PIN_13	IO port 15 pin 13.
BSP_IO_PORT_15_PIN_14	IO port 15 pin 14.
BSP_IO_PORT_15_PIN_15	IO port 15 pin 15.
BSP_IO_PORT_16_PIN_00	IO port 16 pin 0.
BSP_IO_PORT_16_PIN_01	IO port 16 pin 1.
BSP_IO_PORT_16_PIN_02	IO port 16 pin 2.
BSP_IO_PORT_16_PIN_03	IO port 16 pin 3.
BSP_IO_PORT_16_PIN_04	IO port 16 pin 4.

BSP_IO_PORT_16_PIN_05	IO port 16 pin 5.
BSP_IO_PORT_16_PIN_06	IO port 16 pin 6.
BSP_IO_PORT_16_PIN_07	IO port 16 pin 7.
BSP_IO_PORT_16_PIN_08	IO port 16 pin 8.
BSP_IO_PORT_16_PIN_09	IO port 16 pin 9.
BSP_IO_PORT_16_PIN_10	IO port 16 pin 10.
BSP_IO_PORT_16_PIN_11	IO port 16 pin 11.
BSP_IO_PORT_16_PIN_12	IO port 16 pin 12.
BSP_IO_PORT_16_PIN_13	IO port 16 pin 13.
BSP_IO_PORT_16_PIN_14	IO port 16 pin 14.
BSP_IO_PORT_16_PIN_15	IO port 16 pin 15.
BSP_IO_PORT_FF_PIN_FF	Invalid IO port.

Function Documentation

◆ R_BSP_PinRead()

__STATIC_INLINE uint32_t R_BSP_PinRead (bsp_io_port_pin_t *pin*)

Read the current input level of the pin.

Parameters

[in]	pin	The pin

Return values

Current	input level

◆ R_BSP_PinWrite()

__STATIC_INLINE void R_BSP_PinWrite (bsp_io_port_pin_t *pin*, bsp_io_level_t *level*)

Set a pin to output and set the output level to the level provided. If PFS protection is enabled, disable PFS protection using R_BSP_PinAccessEnable() before calling this function.

Parameters

[in]	pin	The pin
[in]	level	The level

◆ R_BSP_PinCfg()

__STATIC_INLINE void R_BSP_PinCfg (bsp_io_port_pin_t *pin*, uint32_t *cfg*)

Configure a pin. If PFS protection is enabled, disable PFS protection using R_BSP_PinAccessEnable() before calling this function.

Parameters

[in]	pin	The pin
[in]	cfg	Configuration for the pin (PmnPFS register setting)

◆ R_BSP_PinAccessEnable()

__STATIC_INLINE void R_BSP_PinAccessEnable (void)

Enable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

◆ R_BSP_PinAccessDisable()

__STATIC_INLINE void R_BSP_PinAccessDisable (void)

Disable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

5.1.4 BSP SDRAM support

BSP

Functions

void	R_BSP_SdramInit (bool init_memory) Initializes SDRAM. More...
void	R_BSP_SdramSelfRefreshEnable (void) Changes SDRAM from Auto-refresh to Self-refresh. More...
void	R_BSP_SdramSelfRefreshDisable (void) Changes SDRAM from Self-refresh to Auto-refresh. More...

Detailed Description

Code that initializes the SDRAMC and SDR SDRAM device memory.

Function Documentation

◆ R_BSP_SdramInit()

void R_BSP_SdramInit (bool *init_memory*)

Initializes SDRAM.

Parameters

init_memory	If true, this function will execute the initialization of the external modules. Otherwise, it will only initialize the SDRAMC and leave the memory in self-refresh mode.

This function initializes SDRAMC and SDR SDRAM device.

Note

This function must only be called once after reset.

◆ R_BSP_SdramSelfRefreshEnable()

void R_BSP_SdramSelfRefreshEnable (void)

Changes SDRAM from Auto-refresh to Self-refresh.

This function allows Software Standby and Deep Software Standby modes to be entered without data loss.

Note

SDRAM cannot be accessed after calling this function. Use R_BSP_SdramSelfRefreshDisable to resume normal SDRAM operation.

◆ R_BSP_SdramSelfRefreshDisable()

void R_BSP_SdramSelfRefreshDisable (void)
Changes SDRAM from Self-refresh to Auto-refresh. This function changes back to Auto-refresh and allows normal SDRAM operation to resume.

5.2 Modules

Detailed Description

Modules are the smallest unit of software available in the FSP. Each module implements one interface.

For more information on FSP Modules and Interfaces review fsp-modules, fsp-stacks and fsp-interfaces in the FSP Architecutre section of this manual.

Note

Not all modules are available for all MCUs. For more information, see the User's Manual for the specific MCU.

Organization of Module Sections

Each module within FSP has a detailed Users' Guide listed below. Each guide typically includes the following content:

- Functions: A list of all the API functions associated with the module
- Detailed Description: A short description of the module and the peripherals used
- Overview: An operational summary and a list of high level features provided by the module
- Configuration: A description of module specific settings available in the configuration tool including clock and pin configurations
- Usage Notes: Module specific documentation and limitations
- Examples: Illustrative code snippets that help the user better understand API use and operation
- Data Structure and Enumeration: Definitions for data structures, enumerations and similar elements used by the module API
- Function Documentation: Details on each API function, including the function prototype, a function summary, a simple use example, list of return values and links to documentation for any needed parameter definitions

Modules

	Analog Analog Modules.
	AI Artificial Intelligence Modules.

	Audio Audio Modules.
	Bootloader Bootloader Modules.
	CapTouch CapTouch Modules.
	Connectivity Connectivity Modules.
	DSP DSP Modules.
	Graphics Graphics Modules.
	Input Input Modules.
	Monitoring Monitoring Modules.
	Motor Motor Modules.
	Networking Networking Modules.
	Power Power Modules.

	RTOS RTOS Modules.
	Security Security Modules.
	Sensor Sensor Modules.
	Storage Storage Modules.
	System System Modules.
	Timers Timers Modules.
	Transfer Transfer Modules.
	ProtectZone ProtectZone Modules.

5.2.1 Analog
Modules

Detailed Description

Analog Modules.

Modules

	ADC (r_adc_b)

	Driver for the ADC_B peripheral on RX MCUs. This module implements the ADC Interface.
	DAC_B (r_dac_b) Driver for the DAC12 version B peripheral on RX MCUs. This module implements the DAC Interface.
	Delta-Sigma Modulator Interface (r_dsmif) Driver for the DSMIF peripheral on RX MCUs. This module implements the ADC Interface.

5.2.1.1 ADC (r_adc_b)

Modules » Analog

Functions

fsp_err_t	R_ADC_B_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t	R_ADC_B_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_scan_cfg)
fsp_err_t	R_ADC_B_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info)
fsp_err_t	R_ADC_B_ScanStart (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_B_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)
fsp_err_t	R_ADC_B_ScanStop (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_B_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status)
fsp_err_t	R_ADC_B_Read (adc_ctrl_t *p_ctrl, adc_channel_t const channel_id, uint16_t *const p_data)
fsp_err_t	R_ADC_B_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const channel_id, uint32_t *const p_data)
fsp_err_t	R_ADC_B_FifoRead (adc_ctrl_t *p_ctrl, adc_group_mask_t const group_mask, adc_b_fifo_read_t *const p_data)
fsp_err_t	R_ADC_B_Close (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_B_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset)

fsp_err_t	R_ADC_B_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend)
fsp_err_t	R_ADC_B_CallbackSet (adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the ADC_B peripheral on RX MCUs. This module implements the ADC Interface.

Overview

Features

The ADC_B module supports the following features:

- 16 bit resolution
- Selectable data format (16, 14, 12, and 10-bit)
- Configurable high-speed and high-accuracy conversion methods
- Configure scans to include:
  - Multiple analog channels
  - Temperature sensor channel
  - Reference Voltage sensor channel
  - Self-Diagnostic channel
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
  - External scan triggers from the ADTRGn port pins
- Configurable scan modes:
  - Single scan mode, where each trigger starts a single scan
  - Continuous scan mode, where all channels are scanned continuously
  - Synchronous scan mode, where A/D converters operate synchronously
- Variable sampling time
- Self-calibration
- Channel-dedicated sample-and-hold circuits
- Supports adding and averaging converted samples
- Limiter clip function
- User offset adjustment function
- User gain adjustment function
- Built-in FIFO
- Optional callback when scan completes, FIFO data is ready, an error occurs, or other conditions are triggered.

Configuration

Virtual Channels and Scan Groups

A virtual channel is a group of registers that stores the A/D conversion configuration for a single analog pin. Each virtual channel has a number of options including the channel for conversion, settings for conversion, data processing method and so on.

To perform A/D conversion of an analog pin, the channel associated with the pin must first be assigned to a virtual channel. That virtual channel is then assigned to a scan group, which brings together one or more virtual channels to be converted in sequence with a specified conversion unit.

Note

Analog channels may be assigned to more than one virtual channel. However, a virtual channel can be assigned to only one scan group. When performing A/D conversion on one analog channel in different scan groups or when converting a channel several times within the same scan group, assign several virtual channels to one analog channel.
To avoid data being overwritten, when converting a channel multiple times within the same scan group use R_ADC_B_FifoRead() instead of R_ADC_B_Read().

Configuring a Scan

To perform A/D conversion of a scan group the following should be configured:

- Assign the analog channel for conversion to a virtual channel.
- Assign the virtual channel to a scan group.
- Assign the scan group to an A/D Converter.

Note

Up to 8 virtual channels can be assigned to a scan group. If more than 8 channels are assigned to a group, only the lowest 8 will be targeted for A/D conversion.

Build Time Configurations for r_adc_b

The following build time configurations are defined in fsp_cfg/r_adc_b_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	bsp	If selected, code for parameter checking is included in the build.

Note

The instance configurations available in this driver are too numerous to list here. Please refer to the RA Configuration editor in e² studio for further details.

Clock Configuration

The ADC_B conversion clock source may be configured to use ADCCLK, PCLKA, or GPT with a selectable division ratio. The ADC_B clock may operate between 25 MHz at a minimum and 60 MHz at a maximum.

Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC_B.

ADTRG0,ADTRG1 and ADTRG2 can be used to start scans with an external trigger. When external triggers are used, ADC_B scans begin on the falling edge of the ADTRG pin.

Usage Notes

Limitations

Developers should be aware of the following limitations when using R_ADC_B:

- Except for Group Priority Operation, if ADC0 or ADC1 are currently performing an A/D conversion operation, attempting to start another scan group that uses the same A/D converter will be ignored. This also applies to starting multiple groups at one time. When Group Priority Operation is not enabled, only the lowest numbered group will be started (for each ADC converter), other groups will be ignored.

Self-Calibration

Calibration is required to use this peripheral; call R_ADC_B_Calibrate() after R_ADC_B_ScanCfg() and prior to any other function. Self-Calibration should be performed any time ADC operating characteristics are modified, including after reset, releasing module-stop, when returning from software standby or deep software standby mode, each time the ADC ScanCfg function is called, and any time ADC related clocks are updated.

Note

Self-calibration is a non-blocking operation. The application should wait for an ADC_EVENT_CALIBRATION_COMPLETE callback before using other ADC_B functionality. The self-calibration process will disable hardware triggers that were previously enabled.

ADC_B Operation Modes

The driver supports two primary operational modes, single-scan and continuous-scan. In single scan mode, one or more groups are scanned once per trigger. In continuous scan mode, one or more groups are started with a single trigger. Scans continue until R_ADC_B_ScanStop() is called. In each mode, analog groups and virtual channels are converted in ascending order.

Single Scan Mode

- Assign any selected analog input or analog channel to any scan group, and convert the selected analog input only once per scan group for each start condition.
- By selecting the scan start conditions for each scan group individually, A/D conversion for each scan group can be started at different times.

Continuous Scan Mode

- Assign any selected analog input or analog channel to any scan group and repeat A/D conversion in scan group units after the first start condition. Conversion continues until the driver is closed.

Background Continuous Scan Mode

- Assign any selected analog input or analog channel of the extended analog function to any scan group and repeat A/D conversion in scan group units after the first start condition. The A/D conversion is performed in the background until the driver is closed. In Background Continuous Scan Mode, if the A/D conversion data will be acquired at the point a start condition is entered.

Note

Background Continuous Scan Mode is only available for Hybrid Mode

Synchronous-Scan Mode

When synchronous operation is enabled all A/D conversions are guaranteed to begin and end based on a user-configured period. When both conversion units are selected for synchronous scan they run from the same period, allowing for consistent timing of simultaneous conversions. Consult section "Synchronous Operation" in User's Manual for details.

ADC_B Conversion Methods

Note

The ADC peripheral supports specific high-speed, high-precision, and normal-precision channels. See Table "A/D conversion characteristics" in the User's Manual for details about what conversion methods are supported by specific physical channels.

Successive Approximation Register (SAR) Mode

- A/D Converter samples the signal source once, and convert by Successive Approximation Register method.
- Fast A/D conversion
- Up to 8 channels per 1 scan group.
- Support only single-ended input (excluding Self-diagnosis function)
- Requires Digital Filter Selection to be disabled

Oversampling Mode

- A/D Converter oversamples the signal source, and converts analog to digital by Noise Shaping Successive Approximation Register method.
- High-accuracy A/D conversion
- Support up to 8 channels per 1 scan group
- Support single-ended input and differential input
- Requires Digital Filter Selection to be enabled

Hybrid Mode

- A/D Converter oversamples the signal source, and converts analog to digital by Noise Shaping Successive Approximation Register method.
- Background Continuous Scan Mode operation enables both high-precision A/D conversion and fast conversion.
- Support up to 4 channels per 1 scan group
- Support single-ended input and differential input
- Requires Digital Filter Selection to be enabled

Digital Filter Selection

The following characteristics of the digital filter for the A/D converter unit may be selected:

- Sinc3 Filter (Over Sampling = 8)
- Minimum Phase Filter (Group Delay < 5)

Note

The digital filter must be disabled for ADC units configured to use Successive Approximation Register (SAR) Mode
The digital filter must be enabled for ADC units configured to use Oversampling Mode or Hybrid Mode

Sample-and-Hold

Enabling sample-and-hold on one or more channels instructs the ADC to perform sampling on all channels as soon as a group scan is started. Internal circuitry holds the sampled voltages until the conversion unit is ready.

Note

Each sample-and-hold unit is connected to two analog channels (0/1, 2/3 etc). When this function is enabled on both members of a pair only one of the two may be scanned at a time.
Enabling sample-and-hold functionality reduces maximum scan frequency because the sample hold time is added to each scan. Refer to the hardware manual for details on the sample-and-hold time. Consult section "ADC Characteristics" in the User's Manual for details.
If you use the channel-dedicated sample-and-hold circuits in Hybrid mode, the virtual channels and the scan groups are constrained. A dummy channel must be configured as last within the group. The A/D conversion data of the channel used as the dummy conversion channel is not guaranteed. See Operation in Hybrid Mode with Channel-dedicated Sample-and-hold Circuit section in the User Manual for more information.

Self-Diagnosis

ADC_B has a built-in self-diagnosis function that can be used to confirm the unit is working correctly. One of three self-diagnosis voltages can be converted and compared to reference values. A self-diagnosis conversion produces a signed data value indicating the ideal A/D converter result. Reference values for 16-bit data format are shown below.

Self-diagnosis mode	Expected reference data	Accuracy Error
Self-diagnosis mode 1	0x0000	A/D conversion result becomes +1 or more when a positive accuracy error occurs and -1 or less when a negative accuracy error occurs.
Self-diagnosis mode 2	0x8000	A/D conversion result becomes greater than or equal to reference data when an accuracy error occurs.
Self-diagnosis mode 3	0x7FFF	A/D conversion result becomes less than or equal to reference data when an accuracy error occurs.

Add/Average Function

The ADC can be configured to automatically add or average a number of conversions into a single result. When enabled only the result of the operation is returned.

Note

When the A/D-converted value addition/average function is used, overflow of conversion data may occur. However, in certain conditions overflow may not be detected. See the A/D Conversion Overflow section below for details.

Data Format

The A/D converter in this peripheral has a resolution of 16 bits. When 14-bit or 12-bit data format is selected, the lower 2 or 4 bits (respectively) of the A/D conversion result are extended for data processing, error calibration (Self-Calibration), gain/offset adjustment and the averaging function

before rounding is applied.

Limiter Clip

The limit clipping function allows for setting upper and lower bounds on converted data. When the A/D conversion data exceeds the specified upper limit value, it is clipped to the upper limit value. If the A/D conversion data falls below the specified lower limit, it is clipped to the lower limit value.

The upper and lower limits are set in one of eight table entries. To perform limit clip functionality, each virtual channel may (optionally) have one of these entries assigned. Interrupts may be enabled for when limiter clip conditions are triggered.

Note

When 14-/12-/10-bit is selected as the data length of the A/D conversion data, the lower bits are cut based on the data-format selection. When 16-bit format is selected, the data length is not rounded.

Using the Temperature Sensor with the ADC_B

The ADC_B HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula:

T = (Vs - V1)/slope + T1

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- slope: Temperature gradient of the temperature sensor (V/degrees C), given as (V2 - V1) / (T2 - T1)
    - T2: Temperature at the experimental measurement of another point (degrees C)
    - V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)

Note

The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.

User Offset and Gain

The user offset adjustment function adds or subtracts a constant value to or from the A/D conversion data. Virtual channels select an offset from a table of values specified by the user.

The user gain adjustment function multiplies the A/D conversion data by an arbitrary coefficient value. As with offset adjustment, virtual channels may select a gain value from a table specified by the user.

Note

When the offset or gain adjusting functions are used overflow of A/D conversion data may occur. See the A/D Conversion Overflow section below for details.
When 14-/12-/10-bit is selected as the data length of converted data the lower bits of offset values are cut based on the data-format selection.

FIFO

The FIFOs consist of 8 stages and can hold up to 8 A/D conversion data. One FIFO is implemented for each scan group. Each FIFO acts as a ring buffer and data will be lost if the FIFO is not read as needed. Interrupts may be enabled for a specific data storage threshold and on overrun.

When Interrupts Are Not Enabled

Interrupts are enabled by default. If scan-complete interrupts are disabled,@ ref R_ADC_B_StatusGet() can be used to poll the ADC_B driver to determine when the scan has completed. R_ADC_B_Read() is used to access the converted ADC_B result.

A/D Conversion Overflow

A/D conversion overflow is detected when converted data exceeds the range that can be handled in the specified data format. When overflow occurs, data is restricted to the upper or lower limit value of the specified data format. Overflow is detected in the following cases:

- When the input to the A/D converter exceeds VREFH0 or falls below VREFL0
- When overflow occurs by the internal processing (calculation) for the A/D conversion data due to the following: – Gain Error and Offset Error Calibration – User Gain/Offset adjustment function – When using A/D-Converted Value Addition/Averaging Function – Data Formatting Process

Examples

Basic Example

This is a basic example of minimal use of the ADC_B in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
options selected. If additional
 * configurations are desired additional adc_channel_cfg_t elements can be defined
and passed to R_ADC_B_ScanCfg. */
extern const adc_b_scan_cfg_t g_adc_b0_scan_cfg;
void adc_b_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_B_Open(&g_adc_b0_ctrl, &g_adc_b0_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable channels. */
    err = R_ADC_B_ScanCfg(&g_adc_b0_ctrl, &g_adc_b0_scan_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_B_Calibrate(&g_adc_b0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
```

```
/* Wait for calibration to complete */
adc_status_t status = {.state = ADC_STATE_CALIBRATION_IN_PROGRESS};
while ((ADC_STATE_IDLE != status.state) &&
         (FSP_SUCCESS == err))
   {
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
      err = R_ADC_B_StatusGet(&g_adc_b0_ctrl, &status);
   }
   assert(FSP_SUCCESS == err);
/* Start one or more scan groups by calling R_ADC_B_ScanGroupStart(). Alternatively,
all scan groups may be started
 * by calling R_ADC_B_ScanCfg(). */
   (void) R_ADC_B_ScanGroupStart(&g_adc_b0_ctrl, ADC_GROUP_MASK_0);
/* Wait for conversion to complete. */
   status.state = ADC_STATE_SCAN_IN_PROGRESS;
while (ADC_STATE_IDLE != status.state)
   {
      (void) R_ADC_B_StatusGet(&g_adc_b0_ctrl, &status);
   }
/* Read converted data. */
   uint16_t channel_0_conversion_result;
   err = R_ADC_B_Read(&g_adc_b0_ctrl, ADC_CHANNEL_0, &channel_0_conversion_result);
   assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	adc_b_fifo_data_t
struct	adc_b_fifo_read_t
struct	adc_b_group_cfg_t
struct	adc_b_scan_cfg_t
struct	adc_b_isr_cfg_t
struct	adc_b_extended_cfg_t

struct	adc_b_instance_ctrl_t

Enumerations

enum	adc_b_clock_divider_t
enum	adc_b_converter_mode_t
enum	adc_b_conversion_method_t
enum	adc_b_data_format_t
enum	adc_b_limit_clip_table_id_t
enum	adc_b_unit_id_t
enum	adc_b_unit_mask_t
enum	adc_b_add_avg_mode_t
enum	adc_b_add_avg_count_t
enum	adc_b_gpt_trigger_t
enum	adc_b_elc_trigger_t
enum	adc_b_external_trigger_t
enum	adc_b_self_diagnosis_mode_t
enum	adc_b_sample_and_hold_mask_t
enum	adc_b_pga_gain_t
enum	adc_b_digital_filter_selection_t
enum	adc_b_sampling_state_table_id_t
enum	adc_b_user_gain_table_id_t
enum	adc_b_user_offset_table_selection_id_t
enum	adc_b_converter_state_t

Data Structure Documentation

◆ adc_b_fifo_data_t

struct adc_b_fifo_data_t

ADC FIFO data type		
Data Fields		
uint32_t	data: 16	Conversion Data.
uint32_t	__pad0__: 8	
adc_channel_t	physical_channel: 7	Channel number for data.
uint32_t	err: 1	Error bit.

◆ adc_b_fifo_read_t

struct adc_b_fifo_read_t		
ADC FIFO Read data structure		
Data Fields		
uint8_t	count	Number of valid FIFO data read.
adc_b_fifo_data_t	fifo_data[12]	FIFO data.

◆ adc_b_group_cfg_t

struct adc_b_group_cfg_t		
ADC Group configuration data		
Data Fields		
adc_group_id_t	scan_group_id	Scan Group ID.
adc_b_unit_id_t	converter_selection	Converter selection.
bool	scan_group_enable	Scan Group enable state.
uint8_t	virtual_channel_count	Virtual Channel count.
bool	scan_end_interrupt_enable	Scan End Interrupt enable.
adc_b_external_trigger_t	external_trigger_enable_mask	External Trigger mask.
adc_b_elc_trigger_t	elc_trigger_enable_mask	ELC Trigger mask.
adc_b_gpt_trigger_t	gpt_trigger_enable_mask	GPT Trigger mask.
uint8_t	conversion_start_delay	Conversion start delay.
uint32_t	self_diagnosis_mask	Self-Diagnosis register data.
bool	limit_clip_interrupt_enable	Limiter Clip interrupt enable.
adc_b_virtual_channel_cfg_t **	p_virtual_channels	Pointer to virtual channel configuration array of size virtual_channel_count.

◆ adc_b_scan_cfg_t

struct adc_b_scan_cfg_t
ADC Scan Group configuration
Data Fields

uint8_t	group_count	Group Count.
adc_b_group_cfg_t **	p_adc_groups	Pointer to ADC group configuration data.

◆ adc_b_isr_cfg_t

struct adc_b_isr_cfg_t		
ADC ISR configuration structure		
Data Fields		
uint8_t	calibration_end_ipl_adc_0	Calibration end IPL for A/D converter unit 0.
uint8_t	calibration_end_ipl_adc_1	Calibration end IPL for A/D converter unit 1.
uint8_t	calibration_end_ipl_adc_2	Calibration end IPL for A/D converter unit 2.
uint8_t	conversion_error_ipl_adc_0	Conversion error IPL for A/D converter unit 0.
uint8_t	conversion_error_ipl_adc_1	Conversion error IPL for A/D converter unit 1.
uint8_t	conversion_error_ipl_adc_2	Conversion error IPL for A/D converter unit 2.
uint8_t	fifo_overflow_ipl	FIFO Overflow IPL.
uint8_t	fifo_read_ipl_group_0	FIFO Read threshold request IPL for Group 0.
uint8_t	fifo_read_ipl_group_1	FIFO Read threshold request IPL for Group 1.
uint8_t	fifo_read_ipl_group_2	FIFO Read threshold request IPL for Group 2.
uint8_t	fifo_read_ipl_group_3	FIFO Read threshold request IPL for Group 3.
uint8_t	fifo_read_ipl_group_4	FIFO Read threshold request IPL for Group 4.
uint8_t	fifo_read_ipl_group_5	FIFO Read threshold request IPL for Group 5.
uint8_t	fifo_read_ipl_group_6	FIFO Read threshold request IPL for Group 6.
uint8_t	fifo_read_ipl_group_7	FIFO Read threshold request IPL for Group 7.
uint8_t	fifo_read_ipl_group_8	FIFO Read threshold request IPL for Group 8.
uint8_t	fifo_read_ipl_group_9	FIFO Read threshold request IPL for Group 9.

uint8_t	fifo_read_ipl_group_10	FIFO Read threshold request IPL for Group 10.
uint8_t	fifo_read_ipl_group_11	FIFO Read threshold request IPL for Group 11.
uint8_t	fifo_read_ipl_group_5678	FIFO Read threshold request IPL for Groups 5, 6, 7, and 8.
uint8_t	limit_clip_ipl	Limiter Clip IPL.
uint8_t	overflow_error_ipl_adc_0	Overflow error IPL for A/D converter unit 0.
uint8_t	overflow_error_ipl_adc_1	Overflow error IPL for A/D converter unit 1.
uint8_t	overflow_error_ipl_adc_2	Overflow error IPL for A/D converter unit 2.
uint8_t	scan_end_ipl_group_0	Scan End IPL for A/D Group 0.
uint8_t	scan_end_ipl_group_1	Scan End IPL for A/D Group 1.
uint8_t	scan_end_ipl_group_2	Scan End IPL for A/D Group 2.
uint8_t	scan_end_ipl_group_3	Scan End IPL for A/D Group 3.
uint8_t	scan_end_ipl_group_4	Scan End IPL for A/D Group 4.
uint8_t	scan_end_ipl_group_5	Scan End IPL for A/D Group 5.
uint8_t	scan_end_ipl_group_6	Scan End IPL for A/D Group 6.
uint8_t	scan_end_ipl_group_7	Scan End IPL for A/D Group 7.
uint8_t	scan_end_ipl_group_8	Scan End IPL for A/D Group 8.
uint8_t	scan_end_ipl_group_9	Scan End IPL for A/D Group 9.
uint8_t	scan_end_ipl_group_10	Scan End IPL for A/D Group 10.
uint8_t	scan_end_ipl_group_11	Scan End IPL for A/D Group 11.
uint8_t	scan_end_ipl_group_5678	Scan End IRQ for A/D Groups 5, 6, 7, and 8.
IRQn_Type	calibration_end_irq_adc_0	Calibration end IRQ for A/D converter unit 0.
IRQn_Type	calibration_end_irq_adc_1	Calibration end IRQ for A/D converter unit 1.
IRQn_Type	calibration_end_irq_adc_2	Calibration end IRQ for A/D converter unit 2.
IRQn_Type	conversion_error_irq_adc_0	Conversion error IRQ for A/D converter unit 0.
IRQn_Type	conversion_error_irq_adc_1	Conversion error IRQ for A/D converter unit 1.
IRQn_Type	conversion_error_irq_adc_2	Conversion error IRQ for A/D converter unit 2.

IRQn_Type	fifo_overflow_irq	FIFO Overflow IRQ.
IRQn_Type	fifo_read_irq_group_0	FIFO Read threshold request IRQ for Group 0.
IRQn_Type	fifo_read_irq_group_1	FIFO Read threshold request IRQ for Group 1.
IRQn_Type	fifo_read_irq_group_2	FIFO Read threshold request IRQ for Group 2.
IRQn_Type	fifo_read_irq_group_3	FIFO Read threshold request IRQ for Group 3.
IRQn_Type	fifo_read_irq_group_4	FIFO Read threshold request IRQ for Group 4.
IRQn_Type	fifo_read_irq_group_5	FIFO Read threshold request IRQ for Group 5.
IRQn_Type	fifo_read_irq_group_6	FIFO Read threshold request IRQ for Group 6.
IRQn_Type	fifo_read_irq_group_7	FIFO Read threshold request IRQ for Group 7.
IRQn_Type	fifo_read_irq_group_8	FIFO Read threshold request IRQ for Group 8.
IRQn_Type	fifo_read_irq_group_9	FIFO Read threshold request IRQ for Group 9.
IRQn_Type	fifo_read_irq_group_10	FIFO Read threshold request IRQ for Group 10.
IRQn_Type	fifo_read_irq_group_11	FIFO Read threshold request IRQ for Group 11.
IRQn_Type	fifo_read_irq_group_5678	FIFO Read threshold request IRQ for Groups 5, 6, 7, and 8.
IRQn_Type	limit_clip_irq	Limiter Clip IRQ.
IRQn_Type	overflow_error_irq_adc_0	Overflow error IRQ for A/D converter unit 0.
IRQn_Type	overflow_error_irq_adc_1	Overflow error IRQ for A/D converter unit 1.
IRQn_Type	overflow_error_irq_adc_2	Overflow error IRQ for A/D converter unit 2.
IRQn_Type	scan_end_irq_group_0	Scan End IRQ for A/D Group 0.
IRQn_Type	scan_end_irq_group_1	Scan End IRQ for A/D Group 1.
IRQn_Type	scan_end_irq_group_2	Scan End IRQ for A/D Group 2.
IRQn_Type	scan_end_irq_group_3	Scan End IRQ for A/D Group 3.
IRQn_Type	scan_end_irq_group_4	Scan End IRQ for A/D Group 4.
IRQn_Type	scan_end_irq_group_5	Scan End IRQ for A/D Group 5.

IRQn_Type	scan_end_irq_group_6	Scan End IRQ for A/D Group 6.
IRQn_Type	scan_end_irq_group_7	Scan End IRQ for A/D Group 7.
IRQn_Type	scan_end_irq_group_8	Scan End IRQ for A/D Group 8.
IRQn_Type	scan_end_irq_group_9	Scan End IRQ for A/D Group 9.
IRQn_Type	scan_end_irq_group_10	Scan End IRQ for A/D Group 10.
IRQn_Type	scan_end_irq_group_11	Scan End IRQ for A/D Group 11.
IRQn_Type	scan_end_irq_group_5678	Scan End IRQ for A/D Groups 5, 6, 7, and 8.

◆ adc_b_extended_cfg_t

struct adc_b_extended_cfg_t		
ADC extended configuration data		
Data Fields		
adc_b_pga_gain_t	pga_gain[4]	PGA Gain selection.
union adc_b_extended_cfg_t	__unnamed__	
union adc_b_extended_cfg_t	adc_filter_selection[3]	
union adc_b_extended_cfg_t	clock_control_u	
union adc_b_extended_cfg_t	sync_operation_control_u	
	__PACKED_STRUCT	ADC_B converter mode.
adc_b_conversion_method_t	method: 2	ADC_B converter method.
uint8_t	__pad0__: 4	
	adc_b_converter_mode[3]	
uint32_t	scan_group_enable	Scan Group enable register data.
uint8_t	converter_selection[12]	Converter selection for groups 0-11.
union adc_b_extended_cfg_t	fifo_enable_mask_u	
uint16_t	fifo_interrupt_enable_mask	FIFO interrupt enable register data.
uint16_t	fifo_interrupt_level[12]	FIFO data threshold interrupt level for groups 0-11.
uint16_t	start_trigger_delay_table[12]	Start trigger delay for groups 0-11.
uint32_t	calibration_adc_state	Calibration State register data.
uint32_t	calibration_sample_and_hold	Calibration Sample and Hold register data.
const adc_b_isr_cfg_t *	p_isr_cfg	Pointer to ISR configuration.

uint16_t	sampling_state_table[16]	Sampling State Tables.
uint8_t	sample_and_hold_enable_mask	Sample and Hold enable register data.
uint32_t	sample_and_hold_config_012	Sample and Hold configuration register data.
uint32_t	sample_and_hold_config_456	Sample and Hold configuration register data.
uint32_t	conversion_state	ADC 0/1/2 Successive Approximation Time Configuration.
int32_t	user_offset_tables[8]	User Offset Table register data.
uint32_t	user_gain_tables[8]	User Gain Table register data.
uint32_t	limiter_clip_interrupt_enable_mask	Limiter clip interrupt enable register data.
uint32_t	limiter_clip_tables[8]	Limiter clip Table register data.

◆ adc_b_instance_ctrl_t

struct adc_b_instance_ctrl_t	
ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open().	
Data Fields	
adc_b_converter_state_t	adc_state
	ADC 0 converter State.
uint32_t	cached_adtrgenr
	Cached conversion peripheral trigger bits, used when starting and stopping scans.
uint32_t	cached_adsystr
	Cached conversion software start bits, used when starting and stopping scans.
uint32_t	trigger_disable_wait_cycles
	ADC clock cycles required to wait after disabling trigger input.
adc_cfg_t const *	p_cfg

	Boolean to verify that the Unit has been initialized.
void(*	p_callback)(adc_callback_args_t *)
	Pointer to callback that is called when an adc_b_event_t occurs.
adc_callback_args_t *	p_callback_memory
	Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory.
void *	p_context
	User defined context passed into callback function.
uint32_t	initialized
	Initialized status of ADC_B.
uint32_t	opened
	Open status of ADC_B.

Enumeration Type Documentation

◆ adc_b_clock_divider_t

enum adc_b_clock_divider_t	
ADC clock divider selection	
Enumerator	
ADC_B_CLOCK_DIV_1	ADC Clock Division 1/1.
ADC_B_CLOCK_DIV_2	ADC Clock Division 1/2.
ADC_B_CLOCK_DIV_3	ADC Clock Division 1/3.
ADC_B_CLOCK_DIV_4	ADC Clock Division 1/4.
ADC_B_CLOCK_DIV_5	ADC Clock Division 1/5.
ADC_B_CLOCK_DIV_6	ADC Clock Division 1/6.
ADC_B_CLOCK_DIV_7	ADC Clock Division 1/7.
ADC_B_CLOCK_DIV_8	ADC Clock Division 1/8.

◆ adc_b_converter_mode_t

enum adc_b_converter_mode_t	
ADC_B Conversion Mode	
Enumerator	
ADC_B_CONVERTER_MODE_SINGLE_SCAN	Single scan mode.
ADC_B_CONVERTER_MODE_CONTINUOUS_SCAN	Continuous scan mode.
ADC_B_CONVERTER_MODE_BACKGROUND_SCAN	Background continuous scan mode (Valid for Hybrid mode only)

◆ adc_b_conversion_method_t

enum adc_b_conversion_method_t	
ADC_B Conversion Method	
Enumerator	
ADC_B_CONVERSION_METHOD_SAR	SAR conversion method.
ADC_B_CONVERSION_METHOD_OVERSAMPLE	Oversampling conversion method.
ADC_B_CONVERSION_METHOD_HYBRID	Hybrid conversion method.

◆ adc_b_data_format_t

enum adc_b_data_format_t	
ADC_B data data format definitions	
Enumerator	
ADC_B_DATA_FORMAT_16_BIT	16 bit adc_b data format
ADC_B_DATA_FORMAT_14_BIT	14 bit adc_b data format
ADC_B_DATA_FORMAT_12_BIT	12 bit adc_b data format
ADC_B_DATA_FORMAT_10_BIT	10 bit adc_b data format

◆ adc_b_limit_clip_table_id_t

enum adc_b_limit_clip_table_id_t	
ADC limiter clipping table id selection options	
Enumerator	
ADC_B_LIMIT_CLIP_TABLE_SELECTION_NONE	Limiter Clip Disabled.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_0	Limiter Clip Table 0.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_1	Limiter Clip Table 1.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_2	Limiter Clip Table 2.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_3	Limiter Clip Table 3.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_4	Limiter Clip Table 4.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_5	Limiter Clip Table 5.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_6	Limiter Clip Table 6.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_7	Limiter Clip Table 7.

◆ adc_b_unit_id_t

enum adc_b_unit_id_t	
ADC unit selection options	
Enumerator	
ADC_B_UNIT_ID_0	ADC Unit ID 0.
ADC_B_UNIT_ID_1	ADC Unit ID 1.
ADC_B_UNIT_ID_2	ADC Unit ID 2.

◆ adc_b_unit_mask_t

enum adc_b_unit_mask_t	
ADC unit selection options	
Enumerator	
ADC_B_UNIT_MASK_0	ADC Unit Mask 0.
ADC_B_UNIT_MASK_1	ADC Unit Mask 1.
ADC_B_UNIT_MASK_2	ADC Unit Mask 2.
ADC_B_UNIT_MASK_UNDEFINED	ADC Unit Mask Unknown.

◆ adc_b_add_avg_mode_t

enum adc_b_add_avg_mode_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_B_ADD_AVERAGE_OFF	Add/Average turned off for channels/sensors.
ADC_B_ADD_AVERAGE_ADDITION_ENABLE	Addition Mode Enabled.
ADC_B_ADD_AVERAGE_AVERAGE_ENABLE	Average Mode Enabled.

◆ adc_b_add_avg_count_t

enum adc_b_add_avg_count_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_B_ADD_AVERAGE_1	Addition turned off for channels/sensors.
ADC_B_ADD_AVERAGE_2	Add/Average 2 samples.
ADC_B_ADD_AVERAGE_4	Add/Average 4 samples.
ADC_B_ADD_AVERAGE_8	Add/Average 8 samples.
ADC_B_ADD_AVERAGE_16	Add/Average 16 samples.
ADC_B_ADD_AVERAGE_32	Add/Average 32 samples.
ADC_B_ADD_AVERAGE_64	Add/Average 64 samples.
ADC_B_ADD_AVERAGE_128	Add/Average 128 samples.
ADC_B_ADD_AVERAGE_256	Add/Average 256 samples.
ADC_B_ADD_AVERAGE_512	Add/Average 512 samples.
ADC_B_ADD_AVERAGE_1024	Add/Average 1024 samples.

◆ adc_b_gpt_trigger_t

enum adc_b_gpt_trigger_t	
ADC GPT Trigger options	
Enumerator	
ADC_B_GPT_TRIGGER_NONE	GPT Trigger Disabled.
ADC_B_GPT_TRIGGER_A0	GPT Trigger A0.
ADC_B_GPT_TRIGGER_A1	GPT Trigger A1.
ADC_B_GPT_TRIGGER_A2	GPT Trigger A2.
ADC_B_GPT_TRIGGER_A3	GPT Trigger A3.
ADC_B_GPT_TRIGGER_A4	GPT Trigger A4.
ADC_B_GPT_TRIGGER_A5	GPT Trigger A5.
ADC_B_GPT_TRIGGER_A6	GPT Trigger A6.
ADC_B_GPT_TRIGGER_A7	GPT Trigger A7.
ADC_B_GPT_TRIGGER_A8	GPT Trigger A8.
ADC_B_GPT_TRIGGER_A9	GPT Trigger A9.
ADC_B_GPT_TRIGGER_A10	GPT Trigger A10.
ADC_B_GPT_TRIGGER_A11	GPT Trigger A11.
ADC_B_GPT_TRIGGER_A12	GPT Trigger A12.
ADC_B_GPT_TRIGGER_A13	GPT Trigger A13.
ADC_B_GPT_TRIGGER_A14	GPT Trigger A14.
ADC_B_GPT_TRIGGER_A15	GPT Trigger A15.
ADC_B_GPT_TRIGGER_B0	GPT Trigger B0.
ADC_B_GPT_TRIGGER_B1	GPT Trigger B1.
ADC_B_GPT_TRIGGER_B2	GPT Trigger B2.
ADC_B_GPT_TRIGGER_B3	GPT Trigger B3.

ADC_B_GPT_TRIGGER_B4	GPT Trigger B4.
ADC_B_GPT_TRIGGER_B5	GPT Trigger B5.
ADC_B_GPT_TRIGGER_B6	GPT Trigger B6.
ADC_B_GPT_TRIGGER_B7	GPT Trigger B7.
ADC_B_GPT_TRIGGER_B8	GPT Trigger B8.
ADC_B_GPT_TRIGGER_B9	GPT Trigger B9.
ADC_B_GPT_TRIGGER_B10	GPT Trigger B10.
ADC_B_GPT_TRIGGER_B11	GPT Trigger B11.
ADC_B_GPT_TRIGGER_B12	GPT Trigger B12.
ADC_B_GPT_TRIGGER_B13	GPT Trigger B13.
ADC_B_GPT_TRIGGER_B14	GPT Trigger B14.
ADC_B_GPT_TRIGGER_B15	GPT Trigger B15.

◆ adc_b_elc_trigger_t

enum adc_b_elc_trigger_t	
ADC ELC Trigger options	
Enumerator	
ADC_B_ELC_TRIGGER_NONE	ELC Trigger Disabled.
ADC_B_ELC_TRIGGER_0	ELC Trigger 0.
ADC_B_ELC_TRIGGER_1	ELC Trigger 1.
ADC_B_ELC_TRIGGER_2	ELC Trigger 2.
ADC_B_ELC_TRIGGER_3	ELC Trigger 3.
ADC_B_ELC_TRIGGER_4	ELC Trigger 4.
ADC_B_ELC_TRIGGER_5	ELC Trigger 5.
ADC_B_ELC_TRIGGER_6	ELC Trigger 6.
ADC_B_ELC_TRIGGER_7	ELC Trigger 7.
ADC_B_ELC_TRIGGER_8	ELC Trigger 8.

◆ adc_b_external_trigger_t

enum adc_b_external_trigger_t	
ADC External Trigger options	
Enumerator	
ADC_B_EXTERNAL_TRIGGER_NONE	External Trigger Disabled.
ADC_B_EXTERNAL_TRIGGER_ADTRG0	External Trigger ADTRG0 Selection.
ADC_B_EXTERNAL_TRIGGER_ADTRG1	External Trigger ADTRG1 Selection.
ADC_B_EXTERNAL_TRIGGER_ADTRG2	External Trigger ADTRG2 Selection.

◆ adc_b_self_diagnosis_mode_t

enum adc_b_self_diagnosis_mode_t	
ADC Self-Diagnosis mode options	
Enumerator	
ADC_B_SELF_DIAGNOSIS_DISABLED	Self-Diagnosis Disabled.
ADC_B_SELF_DIAGNOSIS_MODE_1	Self-Diagnosis Mode 1.
ADC_B_SELF_DIAGNOSIS_MODE_2	Self-Diagnosis Mode 2.
ADC_B_SELF_DIAGNOSIS_MODE_3	Self-Diagnosis Mode 3.

◆ adc_b_sample_and_hold_mask_t

enum adc_b_sample_and_hold_mask_t	
ADC Sample-and-Hold unit enable mask	
Enumerator	
ADC_B_SAMPLE_AND_HOLD_MASK_NONE	Sample-and-Hold Disabled.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_0	Sample-and-Hold Unit 0.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_1	Sample-and-Hold Unit 1.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_2	Sample-and-Hold Unit 2.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_4	Sample-and-Hold Unit 3.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_5	Sample-and-Hold Unit 4.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_6	Sample-and-Hold Unit 5.

◆ adc_b_pga_gain_t

enum adc_b_pga_gain_t	
ADC PGA Gain	
Enumerator	
ADC_B_PGA_GAIN_DISABLED	PGA Gain Disabled.
ADC_B_PGA_GAIN_DIFFERENTIAL_1_500	PGA Gain Setting 1.500.
ADC_B_PGA_GAIN_DIFFERENTIAL_2_333	PGA Gain Setting 2.333.
ADC_B_PGA_GAIN_DIFFERENTIAL_4_000	PGA Gain Setting 4.000.
ADC_B_PGA_GAIN_DIFFERENTIAL_5_667	PGA Gain Setting 5.667.
ADC_B_PGA_GAIN_SINGLE_ENDED_2_500	PGA Gain Setting 2.500.
ADC_B_PGA_GAIN_SINGLE_ENDED_2_667	PGA Gain Setting 2.667.
ADC_B_PGA_GAIN_SINGLE_ENDED_2_857	PGA Gain Setting 2.857.
ADC_B_PGA_GAIN_SINGLE_ENDED_3_077	PGA Gain Setting 3.077.
ADC_B_PGA_GAIN_SINGLE_ENDED_3_333	PGA Gain Setting 3.333.
ADC_B_PGA_GAIN_SINGLE_ENDED_3_636	PGA Gain Setting 3.636.
ADC_B_PGA_GAIN_SINGLE_ENDED_4_000	PGA Gain Setting 4.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_4_444	PGA Gain Setting 4.444.
ADC_B_PGA_GAIN_SINGLE_ENDED_5_000	PGA Gain Setting 5.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_5_714	PGA Gain Setting 5.714.
ADC_B_PGA_GAIN_SINGLE_ENDED_6_667	PGA Gain Setting 6.667.
ADC_B_PGA_GAIN_SINGLE_ENDED_8_000	PGA Gain Setting 8.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_10_000	PGA Gain Setting 10.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_13_333	PGA Gain Setting 13.333.

◆ adc_b_digital_filter_selection_t

enum adc_b_digital_filter_selection_t	
ADC Digital Filter Selection	
Enumerator	
ADC_B_DIGITAL_FILTER_MODE_SINC3	Digital filter Sinc3 filter (Oversampling Rate = 8)
ADC_B_DIGITAL_FILTER_MODE_PHASE	Digital filter Minimum phase filter (Group delay < 2)

◆ adc_b_sampling_state_table_id_t

enum adc_b_sampling_state_table_id_t	
ADC Sampling State table selection options	
Enumerator	
ADC_B_SAMPLING_STATE_TABLE_0	Sampling State Table 0.
ADC_B_SAMPLING_STATE_TABLE_1	Sampling State Table 1.
ADC_B_SAMPLING_STATE_TABLE_2	Sampling State Table 2.
ADC_B_SAMPLING_STATE_TABLE_3	Sampling State Table 3.
ADC_B_SAMPLING_STATE_TABLE_4	Sampling State Table 4.
ADC_B_SAMPLING_STATE_TABLE_5	Sampling State Table 5.
ADC_B_SAMPLING_STATE_TABLE_6	Sampling State Table 6.
ADC_B_SAMPLING_STATE_TABLE_7	Sampling State Table 7.
ADC_B_SAMPLING_STATE_TABLE_8	Sampling State Table 8.
ADC_B_SAMPLING_STATE_TABLE_9	Sampling State Table 9.
ADC_B_SAMPLING_STATE_TABLE_10	Sampling State Table 10.
ADC_B_SAMPLING_STATE_TABLE_11	Sampling State Table 12.
ADC_B_SAMPLING_STATE_TABLE_12	Sampling State Table 13.
ADC_B_SAMPLING_STATE_TABLE_13	Sampling State Table 14.
ADC_B_SAMPLING_STATE_TABLE_14	Sampling State Table 15.
ADC_B_SAMPLING_STATE_TABLE_15	Sampling State Table 16.

◆ adc_b_user_gain_table_id_t

enum adc_b_user_gain_table_id_t	
ADC User Gain table options	
Enumerator	
ADC_B_USER_GAIN_TABLE_SELECTION_DISABLED	User Gain disabled.
ADC_B_USER_GAIN_TABLE_SELECTION_0	User Gain table 0.
ADC_B_USER_GAIN_TABLE_SELECTION_1	User Gain table 1.
ADC_B_USER_GAIN_TABLE_SELECTION_2	User Gain table 2.
ADC_B_USER_GAIN_TABLE_SELECTION_3	User Gain table 3.
ADC_B_USER_GAIN_TABLE_SELECTION_4	User Gain table 4.
ADC_B_USER_GAIN_TABLE_SELECTION_5	User Gain table 5.
ADC_B_USER_GAIN_TABLE_SELECTION_6	User Gain table 6.
ADC_B_USER_GAIN_TABLE_SELECTION_7	User Gain table 7.

◆ adc_b_user_offset_table_selection_id_t

enum adc_b_user_offset_table_selection_id_t	
ADC User Offset table options	
Enumerator	
ADC_B_USER_OFFSET_TABLE_SELECTION_DISABLED	User Offset disabled.
ADC_B_USER_OFFSET_TABLE_SELECTION_0	User Offset table 0.
ADC_B_USER_OFFSET_TABLE_SELECTION_1	User Offset table 1.
ADC_B_USER_OFFSET_TABLE_SELECTION_2	User Offset table 2.
ADC_B_USER_OFFSET_TABLE_SELECTION_3	User Offset table 3.
ADC_B_USER_OFFSET_TABLE_SELECTION_4	User Offset table 4.
ADC_B_USER_OFFSET_TABLE_SELECTION_5	User Offset table 5.
ADC_B_USER_OFFSET_TABLE_SELECTION_6	User Offset table 6.
ADC_B_USER_OFFSET_TABLE_SELECTION_7	User Offset table 7.

◆ adc_b_converter_state_t

enum adc_b_converter_state_t	
Enumerator	
ADC_B_CONVERTER_STATE_NONE	Converter state uninitialized.
ADC_B_CONVERTER_STATE_ADC_0_CALIBRATING	ADC 0 Internal Circuit and Gain/Offset Calibration.
ADC_B_CONVERTER_STATE_ADC_1_CALIBRATING	ADC 1 Internal Circuit and Gain/Offset Calibration.
ADC_B_CONVERTER_STATE_ADC_2_CALIBRATING	ADC 2 Internal Circuit and Gain/Offset Calibration.
ADC_B_CONVERTER_STATE_SH_0_2_CALIBRATING	ADC 0 Sample-and-Hold Calibration.
ADC_B_CONVERTER_STATE_SH_4_6_CALIBRATING	ADC 1 Sample-and-Hold Calibration.
ADC_B_CONVERTER_STATE_READY	Calibration is complete / ADC 0,1,2 are ready.
ADC_B_CONVERTER_STATE_CALIBRATION_FAIL	Calibration has failed.

Function Documentation

◆ R_ADC_B_Open()

fsp_err_t R_ADC_B_Open (adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg*)

Sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If provided, the function registers a callback function pointer for notifying the user whenever a scan has completed, error has occurred, FIFO read request is generated, or other ADC interrupt event occurrs. Implements adc_api_t::open.

Return values

FSP_SUCCESS	Module is ready for use.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_ALREADY_OPEN	The instance control structure has already been opened.

◆ R_ADC_B_ScanCfg()

fsp_err_t R_ADC_B_ScanCfg (adc_ctrl_t * *p_ctrl*, void const *const *p_scan_cfg*)

Configures the ADC_B scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_b_scan_cfg_t to p_channel_cfg. Implements adc_api_t::scanCfg.

Note

This starts group B scans if adc_b_scan_cfg_t::priority_group_a is set to ADC_B_GROUP_A_GROUP_B_CONTINUOUS_SCAN.

Return values

FSP_SUCCESS	Channel specific settings applied.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_INVALID_STATE	Invalid Scan Configuration.
FSP_ERR_INVALID_CHANNEL	Invalid configured channel for group converter id.

◆ R_ADC_B_InfoGet()

fsp_err_t R_ADC_B_InfoGet (adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info*)

Provides the temperature sensor slope and the calibration data for the sensor if available on this MCU. Otherwise, invalid calibration data of 0xFFFFFFFF will be returned.

Return values

FSP_SUCCESS	Info is read into p_adc_info.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.

◆ R_ADC_B_ScanStart()

fsp_err_t R_ADC_B_ScanStart (adc_ctrl_t * *p_ctrl*)

Enables the hardware trigger for a scan depending on how the triggers were configured in the R_ADC_B_ScanCfg call. If the unit was configured for ELC, GPT, or external hardware triggering, then this function allows the trigger signal to get to the ADC unit. The function is not able to control the generation of the trigger itself. If the unit was configured for software triggering, This function was added to this ADC version for compatibility with r_adc driver. For additional flexibility, it is recommended to use R_ADC_B_ScanGroupStart.

Precondition

Call R_ADC_B_ScanCfg after R_ADC_B_Open before starting a scan.
Call R_ADC_B_Calibrate and wait for calibration to complete before starting a scan.

Return values

FSP_SUCCESS	Scan started (software trigger) or hardware triggers enabled.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_ARGUMENT	No hardware triggers configured for groups.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_STATE	Calibration required.

◆ R_ADC_B_ScanGroupStart()

fsp_err_t R_ADC_B_ScanGroupStart (adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_mask*)

Starts a software scan or enables the hardware trigger for a scan depending on how triggers were configured. If the group was configured for ELC, GPT, or external hardware triggering then this function allows the trigger signal to get to the ADC unit. The function itself is not able to control the generation of peripheral triggers. If the unit was configured for software triggering, then this function starts the software triggered scan.

Note

Except for Group Priority Operation, if ADC0 or ADC1 are currently performing an A/D conversion operation, attempting to start another scan group that uses the same A/D converter will be ignored. This also applies to starting multiple groups at one time. When Group Priority Operation is not enabled, only the lowest numbered group will be started (for each ADC converter), other groups will be ignored.

Precondition

Call R_ADC_B_ScanCfg after R_ADC_B_Open before starting a scan.
Call R_ADC_B_Calibrate and wait for calibration to complete before starting a scan.

Return values

FSP_SUCCESS	Scan started (software trigger) or hardware triggers enabled.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_ARGUMENT	An invalid group has been provided.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_STATE	Calibration required.

◆ R_ADC_B_ScanStop()

fsp_err_t R_ADC_B_ScanStop (adc_ctrl_t * *p_ctrl*)

Disables the hardware trigger for a scan and immediately stops all active converters. This function will abortall active conversions.

Return values

FSP_SUCCESS	All scans stopped.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_ARGUMENT	No hardware triggers configured for groups.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.

◆ R_ADC_B_StatusGet()

fsp_err_t R_ADC_B_StatusGet (adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status*)

Provides the status of any scan process that was started, including scans started by ELC or external triggers and calibration scans on MCUs that support calibration.

Return values

FSP_SUCCESS	Module status stored in the provided pointer p_status
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_B_Read()

fsp_err_t R_ADC_B_Read (adc_ctrl_t * *p_ctrl*, adc_channel_t const *channel_id*, uint16_t *const *p_data*)

Reads conversion results from a single channel or sensor.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_INVALID_DATA	Accuracy of data cannot be guaranteed. ADC requires calibration or SAR timing settings are irregular.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_CHANNEL	Invalid channel provided.

◆ R_ADC_B_Read32()

fsp_err_t R_ADC_B_Read32 (adc_ctrl_t * *p_ctrl*, adc_channel_t const *channel_id*, uint32_t *const *p_data*)

Reads conversion results from a single channel or sensor register into a 32-bit result.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_INVALID_DATA	Accuracy of data cannot be guaranteed. ADC requires calibration or SAR timing settings are irregular.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_CHANNEL	Invalid channel provided.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- adc_api_t::read

◆ R_ADC_B_FifoRead()

fsp_err_t R_ADC_B_FifoRead (adc_ctrl_t * *p_ctrl*, adc_group_mask_t const *group_mask*, adc_b_fifo_read_t *const *p_data*)

Reads conversion results from FIFO for the given group mask.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_ARGUMENT	Invalid group provided.
FSP_ERR_UNDERFLOW	FIFO empty.

◆ R_ADC_B_Close()

fsp_err_t R_ADC_B_Close (adc_ctrl_t * *p_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

Return values

FSP_SUCCESS	Module closed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_B_OffsetSet()

fsp_err_t R_ADC_B_OffsetSet (adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset*)

adc_api_t::offsetSet is not supported on the ADC_B.

Return values

FSP_ERR_UNSUPPORTED	Function not supported in this implementation.

◆ R_ADC_B_Calibrate()

fsp_err_t R_ADC_B_Calibrate (adc_ctrl_t *const *p_ctrl*, void const * *p_extend*)

Initiates calibration of the ADC_B. This function must be called before starting a scan and again whenever ADC_B configuration or state is changed.

Note

Self-calibration is a non-blocking operation. The application must wait for calibration to complete before using other ADC_B functionality.
The self-calibration process will disable hardware triggers that were previously enabled.

Parameters

[in]	p_ctrl	Pointer to the instance control structure
[in]	p_extend	Unused argument.

Return values

FSP_SUCCESS	Calibration successfully initiated.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_B_CallbackSet()

fsp_err_t R_ADC_B_CallbackSet (adc_ctrl_t *const *p_api_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void *const *p_context*, adc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements adc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.1.2 DAC_B (r_dac_b)

Modules » Analog

Functions

fsp_err_t	R_DAC_B_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg)
fsp_err_t	R_DAC_B_Write (dac_ctrl_t *p_api_ctrl, uint16_t value)

fsp_err_t	R_DAC_B_Start (dac_ctrl_t *p_api_ctrl)
fsp_err_t	R_DAC_B_Stop (dac_ctrl_t *p_api_ctrl)
fsp_err_t	R_DAC_B_Close (dac_ctrl_t *p_api_ctrl)

Detailed Description

Driver for the DAC12 version B peripheral on RX MCUs. This module implements the DAC Interface.

Overview

Features

The DAC_B module outputs one of 4096 voltage levels between the positive and negative reference voltages.

- Supports setting left-justified or right-justified 12-bit value format for the 16-bit input data registers

Build Time Configurations for r_dac_b

The following build time configurations are defined in fsp_cfg/r_dac_b_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > DAC (r_dac_b)

This module can be added to the Stacks tab via New Stack > Analog > DAC (r_dac_b).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_dac_b0	Module name.
Channel	Value must be an integer greater than or equal to 0	0	Specify the hardware channel.
DAC Operating Voltage Mode	• Normal voltage mode (VREFH >= 2.7v) • Low voltage mode (VREFH < 2.7v)	Normal voltage mode (VREFH >= 2.7v)	Select VREFH voltage for optimization of DAC output.
Data Format	• Right Justified • Left Justified	Right Justified	Specify the DAC data format.

Internal Output	MCU Specific Options	Enable DAC output to internal modules.
Output Amplifier	MCU Specific Options	Enable the DAC output amplifier.
ELC Trigger Source	MCU Specific Options	ELC event source that will trigger the DAC to start a conversion.

Clock Configuration

The DAC peripheral module uses PCLKB as its clock source.

Pin Configuration

The DAn pins are used as analog outputs. Each DAC channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply pins for the DAC and ADC.

The VREFH and VREFL pins are top and ground voltage reference pins for the DAC and ADC.

Usage Notes

Output to Internal Modules

The DAC output can be used as an analog input to other peripherals on the MCU (eg. ACMPHS, ADC) without outputting the voltage to an external pin.

This functionality must be enabled during configuration using dac_b_extended_cfg_t::internal_output_enabled. When internal output is enabled, the DAC output will be routed to internal modules. If the internal output is disabled, the output will be routed to the DAC output pin (DAn).

Examples

Basic Example

This is a basic example of minimal use of the R_DAC_B in an application. This example shows how this driver can be used for basic Digital to Analog Conversion operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA0 as Analog. */
 /* Initialize the DAC channel */
    err = R_DAC_B_Open(&g_dac_b_ctrl, &g_dac_b_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
    value = (uint16_t) DAC_EXAMPLE_VALUE;
    err = R_DAC_B_Write(&g_dac_b_ctrl, value);
    assert(FSP_SUCCESS == err);
    err = R_DAC_B_Start(&g_dac_b_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	dac_b_instance_ctrl_t
struct	dac_b_extended_cfg_t

Enumerations

enum	dac_b_vrefh_t
enum	dac_b_output_t

Data Structure Documentation

◆ dac_b_instance_ctrl_t

struct dac_b_instance_ctrl_t
DAC instance control block.

◆ dac_b_extended_cfg_t

struct dac_b_extended_cfg_t		
DAC extended configuration		
Data Fields		
bool	output_amplifier_enabled	Output amplifier enabled/disabled.
dac_b_output_t	internal_output_enabled	Internal output enable available on selected MCUs.
dac_data_format_t	data_format	Data format.
dac_b_vrefh_t	vrefh	High voltage reference range.

Enumeration Type Documentation

◆ dac_b_vrefh_t

enum dac_b_vrefh_t	
DAC Vref Type	
Enumerator	
DAC_B_VREFH_NORMAL	VREFH >= 2.7v.
DAC_B_VREFH_LOW	VREFH < 2.7v.

◆ dac_b_output_t

enum dac_b_output_t	
Enumerator	
DAC_B_OUTPUT_EXTERNAL	DAOUTDIS = 0.
DAC_B_OUTPUT_INTERNAL	DAOUTDIS = 1.
DAC_B_OUTPUT_EXTERNAL	DAOUTDIS = 0, DAOUTEN = 0.
DAC_B_OUTPUT_INTERNAL	DAOUTDIS = 1, DAOUTEN = 0.
DAC_B_OUTPUT_BOTH	DAOUTDIS = 1, DAOUTEN = 1.
DAC_B_OUTPUT_EXTERNAL	DAOUTDIS = 0, DAOUTEN = 0.
DAC_B_OUTPUT_INTERNAL	DAOUTDIS = 1, DAOUTEN = 0.
DAC_B_OUTPUT_BOTH	DAOUTDIS = 1, DAOUTEN = 1.

Function Documentation

◆ R_DAC_B_Open()

fsp_err_t R_DAC_B_Open (dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg*)

Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

Return values

FSP_SUCCESS	The channel was successfully opened.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. One or both of the following parameters may be NULL: p_api_ctrl or p_cfg 2. data_format value in p_cfg is out of range. 3. Extended configuration structure is set to NULL
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel ID requested in p_cfg may not available on the devices.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.

◆ R_DAC_B_Write()

fsp_err_t R_DAC_B_Write (dac_ctrl_t * *p_api_ctrl*, uint16_t *value*)

Write data to the D/A converter and enable the output if it has not been enabled.

Return values

FSP_SUCCESS	Data is successfully written to the D/A Converter.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

◆ R_DAC_B_Start()

fsp_err_t R_DAC_B_Start (dac_ctrl_t * *p_api_ctrl*)
Start the D/A conversion output if it has not been started.

Return values

FSP_SUCCESS	The channel is started successfully.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_IN_USE	Attempt to re-start a channel.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

◆ R_DAC_B_Stop()

fsp_err_t R_DAC_B_Stop (dac_ctrl_t * *p_api_ctrl*)
Stop the D/A conversion and disable the output signal.

Return values

FSP_SUCCESS	The control is successfully stopped.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

◆ R_DAC_B_Close()

fsp_err_t R_DAC_B_Close (dac_ctrl_t * *p_api_ctrl*)
Stop the D/A conversion, stop output, and close the DAC channel.

Return values

FSP_SUCCESS	The channel is successfully closed.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

5.2.1.3 Delta-Sigma Modulator Interface (r_dsmif)

Modules » Analog

Functions

fsp_err_t	R_DSMIF_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t	R_DSMIF_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend)
fsp_err_t	R_DSMIF_ScanStart (adc_ctrl_t *p_ctrl)
fsp_err_t	R_DSMIF_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)
fsp_err_t	R_DSMIF_ScanStop (adc_ctrl_t *p_ctrl)
fsp_err_t	R_DSMIF_ScanStatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status)
fsp_err_t	R_DSMIF_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)
fsp_err_t	R_DSMIF_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)
fsp_err_t	R_DSMIF_Calibrate (adc_ctrl_t *p_ctrl, void const *p_extend)
fsp_err_t	R_DSMIF_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)
fsp_err_t	R_DSMIF_CallbackSet (adc_ctrl_t *const p_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory)
fsp_err_t	R_DSMIF_Close (adc_ctrl_t *p_ctrl)
fsp_err_t	R_DSMIF_InfoGet (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
fsp_err_t	R_DSMIF_ErrorDetectionEnable (adc_ctrl_t *p_ctrl)
fsp_err_t	R_DSMIF_ErrorStatusGet (adc_ctrl_t *p_ctrl, dsmif_error_status_t *p_error_status)

Detailed Description

Driver for the DSMIF peripheral on RX MCUs. This module implements the ADC Interface.

Overview

The Delta-Sigma Modulator interface (DSMIF) functions similarly to a normal ADC, however, instead

of sampling an analog signal directly, DSMIF samples a PDM signal that is generated by an externally connected 1-bit Delta-Sigma Modulator.

dsmif-overview-light.drawio.svg
Figure 96: DSMIF Overview

Every instance of the DSMIF peripheral has 3 separate channels that can sample independent PDM signals. Each channel has a pair of sinc filters. One sinc filter is used for normal measurements, and the other sinc filter is used for detecting error conditions. The sinc filter configuration will determine the sample resolution, and the final sample rate for each channel. There are three primary configuration settings for the sinc filters:

- Sample Rate - The frequency that the PDM signal is sampled.
- Decimation Count - Number of samples that are removed between the integration stage and the differentiation stage of the sinc filter.
- Filter order - The filter order of the sinc filter: [1st, 2nd, 3rd] order.

Features

The DSMIF module supports the following features:

- Two independent units (3 channels per unit for a total of 6 channels)
- Configurable sinc filter (1st, 2nd, 3rd order)
- Configurable decimation count in the range (4,256)
- Support for master and slave modes
- Channel synchronization
- Trigger data capture via ELC events
- Trigger reset of decimation counter via ELC events
- Configurable error detection
  - Overcurrent error detection (Can trigger common error interrupt)
  - Overcurrent sum error detection (Can trigger common error interrupt)
  - Short-circuit detection (Can trigger common error interrupt)
  - Window notification error detection (Can trigger Port Output Enable for GPT (r_poeg))

Configuration

Build Time Configurations for r_dsmif

The following build time configurations are defined in fsp_cfg/r_dsmif_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Error Detection Support	• Enabled • Disabled	Enabled	If selected code for parameter checking is included in the build.

Configurations for Analog > DSMIF (r_dsmif)

This module can be added to the Stacks tab via New Stack > Analog > DSMIF (r_dsmif).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_dsmif0	Module name
General > Clock Selection	• PCLKA • DSMIFCLK • GPTCLK	PCLKA	Select the DSMIF Core Clock.
General > Unit	• Unit 0 • Unit 1	Unit 0	DSMIF unit number.
General > Alignment	• Left Justified • Right Justified	Right Justified	Select if the conversion result is aligned to the left or right of the channel data registers.
General > Callback	Name must be a valid C symbol	NULL	Callback for handling DSMIF interrupts.
Overcurrent Sum Error > Upper Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent sum error upper limit detection.
Overcurrent Sum Error > Upper Limit > Threshold	Must be an integer in the range [0,0x3FFFF].	0	Overcurrent sum error detection upper limit.
Overcurrent Sum Error > Lower Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent sum error lower limit detection.
Overcurrent Sum Error > Lower Limit > Threshold	Must be an integer in the range [0,0x3FFFF].	0	Overcurrent sum error detection lower limit.
Overcurrent Sum Error > Channel Selection	• Channels 0, 1 and 2 • Channels 0 and 1 • Channel 0 • Channel 1 • Channel 2	Channels 0, 1 and 2	Specify the channel(s) used for detecting overcurrent sum errors.
Interrupts > Channel Synchronization	• Channels are not synchronized • Channels 0 and 1 are synchronized • Channels 0, 1 and 2 are synchronized	Channels are not synchronized	Select the channel synchronization mode. Two or three channels that share the same configuration may optionaly configure a common interrupts instead of channel specific interrupts.
Interrupts > Common	MCU Specific Options		The Common Data

Data Update Interrupt Priority (CDUPD_COM)			Update Interrupt occurs when a conversion is completed on all synchronized channels.
Interrupts > Common Capture A Data Update Interrupt Priority (CDACUPD_COM)	MCU Specific Options		The Common Capture A Interrupt occurs when a Capture A conversion is completed on all synchronized channels.
Interrupts > Common Capture B Data Update Interrupt Priority (CDBCUPD_COM)	MCU Specific Options		The Common Capture B Interrupt occurs when a Capture B conversion is completed on all synchronized channels.
Interrupts > Error Interrupt Priority	MCU Specific Options		The Error Interrupt occurs when a short-cicuit, overcurrent, overcurrent sum, or window notification error occurs on any channel.

Configurations for Analog > DSMIF Channel Configuration (r_dsmif)

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_dsmif_channel0	Module name
Clock > Clock Direction	• Slave Mode • Master Mode	Slave Mode	Select the conversion clock mode for the DSMIF channel. In master mode the clock is output from the DSMIF channel. In slave mode the clock is input to the DSMIF channel.
Clock > Clock Edge	• Negative Edge • Positive Edge	Negative Edge	Select the sampling edge of DSMCLKn for this channel.
Clock > CKDIV Setting (Master Mode Only)	Must be an integer in the range [2,63].	2	Set the A/D conversion clock division ratio. The clock division ratio is equal to 2 x (CKDIV + 1). This setting is only used in master mode.
Clock > DSMCLK Frequency (Slave Mode Only)	Must be an integer in the range [0,25000000].	25000000	If slave mode is selected, then manually enter the DSMCLK frequency.

			This setting is used to calculate the settling time of the DSMIF sinc filters.
Filter > Filter Order	• First Order • Second Order • Third Order	Third Order	Select 1st, 2nd, or 3rd order sinc filter for current data.
Filter > Decimation Count	Must be an integer in the range [4,256].	256	Decimation ratio of the current data sinc filter.
Trigger > Filter Initialization > Trigger	• No Trigger • Trigger 0 (ELC_DSMIFm_CNT0) • Trigger 1 (ELC_DSMIFm_CNT1) • Trigger 2 (ELC_DSMIFm_CNT2)	No Trigger	Select filter initialization trigger. The filter initialization trigger resets the sinc filter's decimation clock via ELC event.
Trigger > Filter Initialization > Edge	• Negative • Positive	Positive	Select either a positive or negative edge for the filter initialization trigger.
Trigger > Capture Trigger A	• No Trigger • Trigger 0 (ELC_DSMIFm_CAP0) • Trigger 1 (ELC_DSMIFm_CAP1) • Trigger 2 (ELC_DSMIFm_CAP2) • Trigger 3 (ELC_DSMIFm_CAP3) • Trigger 4 (ELC_DSMIFm_CAP4) • Trigger 5 (ELC_DSMIFm_CAP5)	No Trigger	Current capture trigger A selection.
Trigger > Capture Trigger B	• No Trigger • Trigger 0 (ELC_DSMIFm_CAP0) • Trigger 1 (ELC_DSMIFm_CAP1) • Trigger 2 (ELC_DSMIFm_CAP2) • Trigger 3 (ELC_DSMIFm_CAP3) • Trigger 4 (ELC_DSMIFm_CAP4) • Trigger 5 (ELC_DSMIFm_CAP5)	No Trigger	Current capture trigger B selection.
Error Detection > Filter > Filter Order	• First Order • Second Order • Third Order	Third Order	Select 1st, 2nd, or 3rd order sinc filter for overcurrent detection.
Error Detection > Filter	Must be an integer in	256	Decimation ratio of the

> Decimation Count	the range [4,256].		overcurrent detection filter.
Error Detection > Overcurrent Detection 0 > Upper Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent 0 upper limit.
Error Detection > Overcurrent Detection 0 > Upper Limit > Threshold	Must be an integer in the range [0,65535].	0xFFFF	Configure the overcurrent 0 upper limit. Valid settings are in the range [0,65535].
Error Detection > Overcurrent Detection 0 > Lower Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent 0 lower limit.
Error Detection > Overcurrent Detection 0 > Lower Limit > Threshold	Must be an integer in the range [0,65535].	0	Configure the overcurrent 0 lower limit. Valid settings are in the range [0,65535].
Error Detection > Overcurrent Detection 1 > Upper Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent 1 upper limit.
Error Detection > Overcurrent Detection 1 > Upper Limit > Threshold	Must be an integer in the range [0,65535].	0xFFFF	Configure the overcurrent 1 upper limit. Valid settings are in the range [0,65535].
Error Detection > Overcurrent Detection 1 > Lower Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent 1 lower limit.
Error Detection > Overcurrent Detection 1 > Lower Limit > Threshold	Must be an integer in the range [0,65535].	0	Configure the overcurrent 1 lower limit. Valid settings are in the range [0,65535].
Error Detection > Overcurrent Detection 2 > Upper Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent 2 upper limit.
Error Detection > Overcurrent Detection 2 > Upper Limit > Threshold	Must be an integer in the range [0,65535].	0xFFFF	Configure the overcurrent 2 upper limit. Valid settings are in the range [0,65535].
Error Detection > Overcurrent Detection 2 > Lower Limit > Detection	• Disabled • Enabled	Disabled	Enable or disable overcurrent 2 lower limit.
Error Detection >	Must be an integer in	0	Configure the

Overcurrent Detection 2 > Lower Limit > Threshold	the range [0,65535].		overcurrent 2 lower limit. Valid settings are in the range [0,65535].
Error Detection > Overcurrent Window Notifcation 0 > Notification	• Disabled • Enabled	Disabled	Enable or disable overcurrent window notification 0.
Error Detection > Overcurrent Window Notifcation 0 > Mode	• Overcurrent Data < Lower Limit 0 or Upper Limit 0 < Overcurrent Data • Lower Limit 0 <= Overcurrent Data <= Upper Limit 0	Overcurrent Data < Lower Limit 0 or Upper Limit 0 < Overcurrent Data	Select overcurrent window notification 0 mode.
Error Detection > Overcurrent Window Notifcation 1 > Notification	• Disabled • Enabled	Disabled	Enable or disable overcurrent window notification 1.
Error Detection > Overcurrent Window Notifcation 1 > Mode	• Overcurrent Data < Lower Limit 1 or Upper Limit 1 < Overcurrent Data • Lower Limit 1 <= Overcurrent Data <= Upper Limit 1	Overcurrent Data < Lower Limit 1 or Upper Limit 1 < Overcurrent Data	Select overcurrent window notification 1 mode.
Error Detection > Overcurrent Window Notifcation 2 > Notification	• Disabled • Enabled	Disabled	Enable or disable overcurrent window notification 2.
Error Detection > Overcurrent Window Notifcation 2 > Mode	• Overcurrent Data < Lower Limit 2 or Upper Limit 2 < Overcurrent Data • Lower Limit 2 <= Overcurrent Data <= Upper Limit 2	Overcurrent Data < Lower Limit 2 or Upper Limit 2 < Overcurrent Data	Select overcurrent window notification 2 mode.
Error Detection > Overcurrent Window Notifcation 3 > Notification	• Disabled • Enabled	Disabled	Enable or disable overcurrent window notification 3.
Error Detection >	Refer to the RX	Window notification 0	Select overcurrent

Overcurrent Window Notifcation 3 > Mode	Configuration tool for available options.		window notification 3 mode.
Error Detection > Short Circuit > Detection	• Disabled • Enabled	Disabled	Enable or disable short circuit detection.
Error Detection > Short Circuit > Low Count	Must be an integer in the range [0,8191].	0	Short circuit detection low count.
Error Detection > Short Circuit > High Count	Must be an integer in the range [0,8191].	0	Short circuit detection high count.
Interrupts > Current Data Update Interrupt Priority (CDUPD)	MCU Specific Options		The Current Data Update Interrupt occurs when a conversion is completed.
Interrupts > Capture A Data Update Interrupt Priority (CDAUPD)	MCU Specific Options		The Capture A Data Update Interrupt occurs when a capture A conversion is completed.
Interrupts > Capture B Data Update Interrupt Priority (CDBUPD)	MCU Specific Options		The Capture B Data Update Interrupt occurs when a capture B conversion is completed.

Clock Configuration

The DSMIF conversion rate is determined by the **DSMIF Core Clock**, **A/D Conversion Division Ratio (CKDIV)**, and the configured sinc filter's **decimation count**.

The **DSMIF Core Clock** is common to all channels and may be configured to the following settings:

- PCLKA - Peripheral Clock A
- DSMIFCLK - Dedicated DSMIF Clock
- GPTCLK - Dedicated GPT Clock

The **A/D Conversion Division Ratio (CKDIV)** is a channel specific setting that further divides the core clock by the following formula:

```
A/D Conversion Division Ratio = (2 x (CKDIV + 1))
```

Note

1. *CKDIV is set in the range [2,63].*
2. *CKDIV is only valid in master mode.*

The **decimation count** is a channel specific sinc filter setting that will determine the final conversion rate of a DSMIF channel.

Note

1. *The decimation count is set in the range [4,256].*

DSMIF Recommended Clock Configuration:

DSMIF Core Clock (Mhz)	CKDIV	DSMCLKn (Mhz)
300	0x05	25
	0x08	16.67
	0x09	15
	0x0E	10
	0x1D	5
200	0x03	25
	0x04	20
	0x07	12.5
	0x09	10
	0x0F	6.25
	0x13	5

Note

1. If settings other than the recommended are used, there is an additional clock constraint: DSMIF Core Clock/5 ? PCLKA

Pin Configuration

Each DSMIF channel has two pins:

- DSMmCLKn - DSMIF Clock Input/Output. In Master mode the clock pin is an output, in slave mode, the clock is an input.
- DSMmDATn - DSMIF Data Input.

Usage Notes

Channel Synchronization

Multiple channels may be optionally synchronized in order to generate common interrupts in response to the following events:

- ADC_EVENT_CONVERSION_COMPLETE
- ADC_EVENT_CAPTURE_A
- ADC_EVENT_CAPTURE_B

The following sets of channels may be synchronized:

- Channel 0 and Channel 1
- Channel 0, Channel 1, and Channel 2

When synchronizing channels, each channel must use the same 'Clock', 'Filter', and 'Trigger' configuration settings. For convenience, channel configurations can be reused between channels of a DSMIF instance instead of creating new channel configurations for each channel.

Note

1. Channel configurations cannot be reused between different DSMIF instances.
2. Channel configurations can only be reused between channels that are synchronized.

Filter Settling Time

The filter settling time is the time required for the filter results to be valid after the DSMIF channels have been started. The settling time is determined by the DSMCLK period, Decimation Ratio, Filter Order, DSMIF Core Clock period, and Peripheral Clock period.

The following macros are generated by the tooling in order to assist with determining the required settling time:

```
/* Calculated settling time required for DSMIF filters (See "Settling Time" section
in the MCU user manual). */
#define DSMIF0_FILTER_SETTLING_TIME_SYNCHRONOUS_US (...)
#define DSMIF0_FILTER_SETTLING_TIME_ASYNCHRONOUS_US (...)
#define DSMIF0_FILTER_SETTLING_TIME_GPT_SYNCHRONOUS_US (...)
#define DSMIF0_FILTER_SETTLING_TIME_GPT_ASYNCHRONOUS_US (...)
```

Note

1. For more information on how to calculate the settling time refer to the MCU user manual.

Error Detection

There are four different types of error interrupts generated by the DSMIF peripheral:

- **Overcurrent error detection** - Triggered when a conversion result exceeds a configured upper/lower limit on a particular channel.
- **Overcurrent sum error detection** - Triggered when the sum of conversion results from a set of channels exceeds a configured upper/lower limit.
- **Window Notification error detection** - Triggered when a conversion result falls within a set of configurable ranges.
- **Short-circuit detection** - Triggered when a configured number of consecutive '1's or '0's are input on a DSMmDATn pin.

In order to enable error detection, the application must call R_DSMIF_ErrorDetectionEnable after waiting for all DSMIF channel filters to stabilize. The amount of time required for a filter to stabilize is dependent on the filter parameters, and clock settings. Refer to the MCU user manual for determining the filter stabilization time.

Once error detection is enabled, R_DSMIF_ErrorStatusGet can be called to read the error status.

Note

1. Error detection is disabled when channel operation is stopped via R_DSMIF_ScanStop. When starting channel operation again, the same settling time is required before reenabling error detection.

Error Interrupt

If the error interrupt is enabled, the following configured error flags will generate an interrupt request:

- Overcurrent error detection
- Overcurrent sum error detection
- Short-circuit detection

Note

1. Since the error interrupt is a level interrupt, R_DSMIF_ErrorStatusGet must be called from the user callback in order to clear the associated error status flags.
2. Window Notification error detection does not generate an interrupt request.

Limitations

Developers should be aware of the following limitations when using DSMIF:

- The DSMIF peripheral must be stopped prior to entering software standby mode.

Examples

Basic Example

This is a basic example of minimal use of the ADC in an application.

```
void dsmif_basic_example (void)
{
 fsp_err_t err = R_DSMIF_Open(&g_dsmif_ctrl, &g_dsmif_cfg);
    assert(FSP_SUCCESS == err);
    err = R_DSMIF_ScanStart(&g_dsmif_ctrl);
    assert(FSP_SUCCESS == err);
 /* The application must wait for the configured filters to stabilize. */
    app_wait_for_filter_stabilization(DSMIF0_FILTER_SETTLING_TIME_SYNCHRONOUS_US);
    err = R_DSMIF_ErrorDetectionEnable(&g_dsmif_ctrl);
    assert(FSP_SUCCESS == err);
}
void dsmif_callback (adc_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case ADC_EVENT_CONVERSION_COMPLETE:
      {
            uint32_t conversion;
 fsp_err_t err = R_DSMIF_Read32(&g_dsmif_ctrl, p_args->channel, &conversion);
```

```
if (FSP_SUCCESS == err)
     {
/* Application processes the converted data. */
                app_dsmif_data_process(p_args->channel, conversion);
           }
break;
      }
case ADC_EVENT_CAPTURE_A:
     {
           uint32_t conversion;
fsp_err_t err = R_DSMIF_Read32(&g_dsmif_ctrl, p_args->channel, &conversion);
if (FSP_SUCCESS == err)
     {
/* Application processes the converted capture A data. */
                app_dsmif_capture_a_data_process(p_args->channel, conversion);
           }
break;
      }
case ADC_EVENT_CAPTURE_B:
     {
           uint32_t conversion;
fsp_err_t err = R_DSMIF_Read32(&g_dsmif_ctrl, p_args->channel, &conversion);
if (FSP_SUCCESS == err)
     {
/* Application processes the converted capture B data. */
                app_dsmif_capture_b_data_process(p_args->channel, conversion);
           }
break;
      }
case ADC_EVENT_CONVERSION_ERROR:
     {
dsmif_error_status_t error_status;
fsp_err_t            err = R_DSMIF_ErrorStatusGet(&g_dsmif_ctrl, &error_status);
if (FSP_SUCCESS == err)
```

```
     {
 /* Application processes the error status. */
                app_dsmif_error_process(&error_status);
           }
 break;
      }
 default:
     {
 /* Invalid event. */
 break;
      }
   }
}
```

Reading Current/Overcurrent/Capture data

This is a basic example of how to read the following types of data by OR'ing the appropriate channel mask:

- Current Data
- Capture A/B Data
- Overcurrent Data
- Overcurrent Capture Data
- Overcurrent Sum Capture Data

```
void dsmif_channel_data_masks_example (void)
{
 fsp_err_t err;
    uint16_t conversion;
 /* Read normal current data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0, &conversion);
    assert(FSP_SUCCESS == err);
 /* Read capture current data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_CAPTURE_A, &conversion);
    assert(FSP_SUCCESS == err);
 /* Read overcurrent data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_OVERCURRENT, &conversion);
    assert(FSP_SUCCESS == err);
```

```
 /* Read overcurrent capture data 0 from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_OVERCURRENT0_CAPTURE,
&conversion);
    assert(FSP_SUCCESS == err);
 /* Read overcurrent sum capture data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_OVERCURRENT_SUM_CAPTURE,
&conversion);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	dsmif_clock_cfg_t
struct	dsmif_filter_cfg_t
struct	dsmif_trigger_cfg_t
struct	dsmif_interrupt_cfg_t
struct	dsmif_error_detection_cfg_t
struct	dsmif_error_status_t
struct	dsmif_channel_cfg_t
struct	dsmif_extended_cfg_t
struct	dsmif_instance_ctrl_t

Enumerations

enum	dsmif_clock_selection_t
enum	dsmif_overcurrent_sum_error_channel_t
enum	dsmif_channel_synchronization_mode_t
enum	dsmif_clock_direction_t
enum	dsmif_clock_edge_t
enum	dsmif_filter_order_t
enum	dsmif_data_shift_t

enum	dsmif_capture_trigger_t
enum	dsmif_filter_initialization_trigger_t
enum	dsmif_overcurrent_window_notification_mode_t
enum	dsmif_overcurrent_window_notification_mode3_t
enum	dsmif_overcurrent_status_t
enum	dsmif_short_circuit_status_t
enum	dsmif_overcurrent_sum_status_t
enum	dsmif_overcurrent_window_notification_status_t

Data Structure Documentation

◆ dsmif_clock_cfg_t

struct dsmif_clock_cfg_t		
Channel clock configuration		
Data Fields		
dsmif_clock_direction_t	direction	A/D conversion clock master/slave switching.
dsmif_clock_edge_t	edge	Sampling edge selection.
uint8_t	divider	A/D conversion clock division ratio.

◆ dsmif_filter_cfg_t

struct dsmif_filter_cfg_t		
Filter configuration		
Data Fields		
dsmif_filter_order_t	filter_order	Sinc filter order (1st, 2nd, or 3rd)
dsmif_data_shift_t	data_shift	Data shift setting.
uint8_t	decimation_count	Decimation count [0,255].

◆ dsmif_trigger_cfg_t

struct dsmif_trigger_cfg_t		
Event trigger configuration.		
Data Fields		
dsmif_capture_trigger_t	capture_trigger_a	Current capture trigger A

		selection.
dsmif_capture_trigger_t	capture_trigger_b	Current capture trigger B selection.
dsmif_filter_initialization_trigger_t	filter_initialization_trigger	Current measurement filter initialization trigger selection.
dsmif_clock_edge_t	filter_initialization_edge	Current measurement filter initialization trigger edge selection.

◆ dsmif_interrupt_cfg_t

struct dsmif_interrupt_cfg_t		
Channel interrupt configuration.		
Data Fields		
IRQn_Type	current_data_update_irq	Channel current data update interrupt request.
uint8_t	current_data_update_ipl	Channel current data update interrupt priority.
IRQn_Type	capture_data_a_irq	Channel capture data a interrupt request.
uint8_t	capture_data_a_ipl	Channel capture data a interrupt priority.
IRQn_Type	capture_data_b_irq	Channel capture data b interrupt request.
uint8_t	capture_data_b_ipl	Channel capture data b interrupt priority.

◆ dsmif_error_detection_cfg_t

struct dsmif_error_detection_cfg_t		
Channel error configuration: • Overcurrent upper/lower limit error detection 0,1,2 • Overcurrent window notification 0,1,2,3 • Short circuit low/high detection		
Data Fields		
union dsmif_error_detection_cfg_t	overcurrent_detection_u	
union dsmif_error_detection_cfg_t	overcurrent_window_notification_enable_u	
union dsmif_error_detection_cfg_t	overcurrent_window_notification_mode_u	
bool	short_circuit_detection_enable	Enable short circuit detection.
uint32_t	short_circuit_detection_low_cou	Short circuit low count.

	nt	
uint32_t	short_circuit_detection_high_count	Short circuit high count.
uint32_t	overcurrent_lower_limit[3]	Overcurrent lower limit settings.
uint32_t	overcurrent_upper_limit[3]	Overcurrent upper limit settings.
dsmif_filter_cfg_t	overcurrent_filter_cfg	Sinc filter configuration for overcurrent detection.

◆ dsmif_error_status_t

struct dsmif_error_status_t		
Error status.		
Data Fields		
dsmif_overcurrent_status_t	channel_overcurrent_status	Channel Overcurrent state.
dsmif_short_circuit_status_t	channel_short_circuit_status	Channel Short circuit state.
dsmif_overcurrent_sum_status_t	overcurrent_sum_status	Overcurrent Sum state.
dsmif_overcurrent_window_notification_status_t	channel_overcurrent_window_status	Channel Overcurrent Window State.

◆ dsmif_channel_cfg_t

struct dsmif_channel_cfg_t		
Channel configuration.		
Data Fields		
dsmif_clock_cfg_t	clock_cfg	Channel clock configuration.
dsmif_filter_cfg_t	filter_cfg	Channel sinc filter configuration.
dsmif_trigger_cfg_t	trigger_cfg	Channel event trigger configuration.
dsmif_interrupt_cfg_t	irq_cfg	Channel interrupt configuration.

◆ dsmif_extended_cfg_t

struct dsmif_extended_cfg_t		
DSMIF configuration extension. This extension is required and must be provided in dsmif_cfg_t::p_extend.		
Data Fields		
dsmif_clock_selection_t	clock_selection	DSMIF core clock selection.
uint32_t	channel_mask	Channel bitmask.
dsmif_channel_cfg_t const *	p_channel_cfgs[DSMIF_MAX_NU	Configuration for each channel,

	M_CHANNELS]	set to NULL if unused.
dsmif_channel_synchronization_mode_t	synchronization_mode	Channel synchronization mode.
IRQn_Type	common_current_data_irq	Common current data interrupt request.
uint8_t	common_current_data_ipl	Common current data interrupt priority.
IRQn_Type	common_capture_data_a_irq	Channel capture data a interrupt request.
uint8_t	common_capture_data_a_ipl	Channel capture data a interrupt priority.
IRQn_Type	common_capture_data_b_irq	Channel capture data b interrupt request.
uint8_t	common_capture_data_b_ipl	Channel capture data b interrupt priority.

◆ dsmif_instance_ctrl_t

struct dsmif_instance_ctrl_t
DSMIF instance control block. DO NOT INITIALIZE.

Enumeration Type Documentation

◆ dsmif_clock_selection_t

enum dsmif_clock_selection_t	
DSMIF clock selection.	
Enumerator	
DSMIF_CLOCK_SELECTION_PCLKA	Select PCLKA as the DSMIF clock.
DSMIF_CLOCK_SELECTION_DSMIFCLK	Select DSMIFCLK as the DSMIF clock.
DSMIF_CLOCK_SELECTION_GPTCLK	Select GPTCLK as the DSMIF clock.

◆ dsmif_overcurrent_sum_error_channel_t

enum dsmif_overcurrent_sum_error_channel_t	
Select the channel(s) for overcurrent sum error detection.	
Enumerator	
DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_0_2	Detects error of sum value of overcurrent data of CH0, CH1, CH2.
DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_0_1	Detects error of sum value of overcurrent data of CH0, CH1.
DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_0	Detects error of overcurrent data of CH0.
DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_1	Detects error of overcurrent data of CH1.
DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_2	Detects error of overcurrent data of CH2.

◆ dsmif_channel_synchronization_mode_t

enum dsmif_channel_synchronization_mode_t	
Select the channel(s) that are synchronized.	
Enumerator	
DSMIF_CHANNEL_SYNCHRONIZATION_MODE_DISABLED	Channels are not synchronized.
DSMIF_CHANNEL_SYNCHRONIZATION_MODE_CHANNEL_0_2	Channel 0 and Channel 1 and Channel 2 are synchronized.
DSMIF_CHANNEL_SYNCHRONIZATION_MODE_CHANNEL_0_1	Channel 0 and Channel 1 are synchronized.

◆ dsmif_clock_direction_t

enum dsmif_clock_direction_t	
Enumerator	
DSMIF_CLOCK_DIRECTION_SLAVE	Clock signal is input (slave operation)
DSMIF_CLOCK_DIRECTION_MASTER	Clock signal is output (master operation)

◆ dsmif_clock_edge_t

enum dsmif_clock_edge_t	
Select the clock edge for sampling DSMIF data.	
Enumerator	
DSMIF_CLOCK_EDGE_NEGATIVE	Data is sampled on the negative edge of clock signal.
DSMIF_CLOCK_EDGE_POSITIVE	Data is sampled on the positive edge of clock signal.

◆ dsmif_filter_order_t

enum dsmif_filter_order_t	
Select the DSMIF filter order.	
Enumerator	
DSMIF_FILTER_ORDER_3RD	Filter order setting 3rd order.
DSMIF_FILTER_ORDER_1ST	Filter order setting 1st order.
DSMIF_FILTER_ORDER_2ND	Filter order setting 2nd order.

◆ dsmif_data_shift_t

enum dsmif_data_shift_t	
Select the [MSB,LSB] that are copied from the filter output into the data register.	
Enumerator	
DSMIF_DATA_SHIFT_23_8	Data shift setting [23:8].
DSMIF_DATA_SHIFT_22_7	Data shift setting [22:7].
DSMIF_DATA_SHIFT_21_6	Data shift setting [21:6].
DSMIF_DATA_SHIFT_20_5	Data shift setting [20:5].
DSMIF_DATA_SHIFT_19_4	Data shift setting [19:4].
DSMIF_DATA_SHIFT_18_3	Data shift setting [18:3].
DSMIF_DATA_SHIFT_17_2	Data shift setting [17:2].
DSMIF_DATA_SHIFT_16_1	Data shift setting [16:1].
DSMIF_DATA_SHIFT_15_0	Data shift setting [15:0].
DSMIF_DATA_SHIFT_14_0	Data shift setting [14:0].
DSMIF_DATA_SHIFT_13_0	Data shift setting [13:0].
DSMIF_DATA_SHIFT_12_0	Data shift setting [12:0].
DSMIF_DATA_SHIFT_11_0	Data shift setting [11:0].
DSMIF_DATA_SHIFT_10_0	Data shift setting [10:0].
DSMIF_DATA_SHIFT_9_0	Data shift setting [9:0].
DSMIF_DATA_SHIFT_8_0	Data shift setting [8:0].
DSMIF_DATA_SHIFT_7_0	Data shift setting [7:0].
DSMIF_DATA_SHIFT_6_0	Data shift setting [6:0].
DSMIF_DATA_SHIFT_5_0	Data shift setting [5:0].
DSMIF_DATA_SHIFT_4_0	Data shift setting [4:0].

◆ dsmif_capture_trigger_t

enum dsmif_capture_trigger_t	
Enumerator	
DSMIF_CAPTURE_TRIGGER_NONE	Do not capture.
DSMIF_CAPTURE_TRIGGER_0	Select current data capture trigger 0.
DSMIF_CAPTURE_TRIGGER_1	Select current data capture trigger 1.
DSMIF_CAPTURE_TRIGGER_2	Select current data capture trigger 2.
DSMIF_CAPTURE_TRIGGER_3	Select current data capture trigger 3.
DSMIF_CAPTURE_TRIGGER_4	Select current data capture trigger 4.
DSMIF_CAPTURE_TRIGGER_5	Select current data capture trigger 5.

◆ dsmif_filter_initialization_trigger_t

enum dsmif_filter_initialization_trigger_t	
Enumerator	
DSMIF_FILTER_INITIALIZATION_TRIGGER_NONE	Do not initialize.
DSMIF_FILTER_INITIALIZATION_TRIGGER_0	Select decimation dividing counter initialization trigger 0.
DSMIF_FILTER_INITIALIZATION_TRIGGER_1	Select decimation dividing counter initialization trigger 1.
DSMIF_FILTER_INITIALIZATION_TRIGGER_2	Select decimation dividing counter initialization trigger 2.

◆ dsmif_overcurrent_window_notification_mode_t

enum dsmif_overcurrent_window_notification_mode_t	
Overcurrent window notification window mode setting [0,2].	
Enumerator	
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE_OUTSIDE	Overcurrent data <= lower limit, upper limit <= overcurrent data.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE_INSIDE	lower limit < overcurrent data < upper limit.

◆ dsmif_overcurrent_window_notification_mode3_t

enum dsmif_overcurrent_window_notification_mode3_t	
Overcurrent window detection window mode setting for window notification 3.	
Enumerator	
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0	Overcurrent detection window notification 0.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_1	Overcurrent detection window notification 1.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_2	Overcurrent detection window notification 2.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_OR_1	Overcurrent detection window notification 0 OR 1.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_AND_1	Overcurrent detection window notification 0 AND 1.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_OR_2	Overcurrent detection window notification 0 OR 2.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_AND_2	Overcurrent detection window notification 0 AND 2.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_1_OR_2	Overcurrent detection window notification 1 OR 2.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_1_AND_2	Overcurrent detection window notification 1 AND 2.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_OR_1_OR_2	Overcurrent detection window notification 0 OR 1 OR 2.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_AND_1_AND_2	Overcurrent detection window notification 0 AND 1 AND 2.

◆ dsmif_overcurrent_status_t

enum dsmif_overcurrent_status_t	
DSMIF Channel Overcurrent detect status.	
Enumerator	
DSMIF_OVERCURRENT_STATUS_CH0_LOWER_LIMIT_0	Channel 0 overcurrent lower limit detection 0 flag.

DSMIF_OVERCURRENT_STATUS_CH1_LOWER_LIMIT_0	Channel 1 overcurrent lower limit detection 0 flag.
DSMIF_OVERCURRENT_STATUS_CH2_LOWER_LIMIT_0	Channel 2 overcurrent lower limit detection 0 flag.
DSMIF_OVERCURRENT_STATUS_CH0_UPPER_LIMIT_0	Channel 0 overcurrent upper limit exceeded 0 flag.
DSMIF_OVERCURRENT_STATUS_CH1_UPPER_LIMIT_0	Channel 1 overcurrent upper limit exceeded 0 flag.
DSMIF_OVERCURRENT_STATUS_CH2_UPPER_LIMIT_0	Channel 2 overcurrent upper limit exceeded 0 flag.
DSMIF_OVERCURRENT_STATUS_CH0_LOWER_LIMIT_1	Channel 0 overcurrent lower limit detection 1 flag.
DSMIF_OVERCURRENT_STATUS_CH1_LOWER_LIMIT_1	Channel 1 overcurrent lower limit detection 1 flag.
DSMIF_OVERCURRENT_STATUS_CH2_LOWER_LIMIT_1	Channel 2 overcurrent lower limit detection 1 flag.
DSMIF_OVERCURRENT_STATUS_CH0_UPPER_LIMIT_1	Channel 0 overcurrent upper limit exceeded 1 flag.
DSMIF_OVERCURRENT_STATUS_CH1_UPPER_LIMIT_1	Channel 1 overcurrent upper limit exceeded 1 flag.
DSMIF_OVERCURRENT_STATUS_CH2_UPPER_LIMIT_1	Channel 2 overcurrent upper limit exceeded 1 flag.
DSMIF_OVERCURRENT_STATUS_CH0_LOWER_LIMIT_2	Channel 0 overcurrent lower limit detection 2 flag.
DSMIF_OVERCURRENT_STATUS_CH1_LOWER_LIMIT_2	Channel 1 overcurrent lower limit detection 2 flag.
DSMIF_OVERCURRENT_STATUS_CH2_LOWER_LIMIT_2	Channel 2 overcurrent lower limit detection 2 flag.
DSMIF_OVERCURRENT_STATUS_CH0_UPPER_LIMIT_2	Channel 0 overcurrent upper limit exceeded 2 flag.
DSMIF_OVERCURRENT_STATUS_CH1_UPPER_LIMIT_2	Channel 1 overcurrent upper limit exceeded 2 flag.
DSMIF_OVERCURRENT_STATUS_CH2_UPPER_LIMIT_2	Channel 2 overcurrent upper limit exceeded 2 flag.

◆ dsmif_short_circuit_status_t

enum dsmif_short_circuit_status_t	
DSMIF Channel Short circuit detect status.	
Enumerator	
DSMIF_SHORT_CIRCUIT_STATUS_CH0	Channel 0 short circuit detection flag.
DSMIF_SHORT_CIRCUIT_STATUS_CH1	Channel 1 short circuit detection flag.
DSMIF_SHORT_CIRCUIT_STATUS_CH2	Channel 2 short circuit detection flag.

◆ dsmif_overcurrent_sum_status_t

enum dsmif_overcurrent_sum_status_t	
DSMIF Overcurrent Sum detect status.	
Enumerator	
DSMIF_OVERCURRENT_SUM_STATUS_LOWER_LIMIT	Overcurrent sum error lower limit detection flag.
DSMIF_OVERCURRENT_SUM_STATUS_UPPER_LIMIT	Overcurrent sum error upper limit detection flag.

◆ dsmif_overcurrent_window_notification_status_t

enum dsmif_overcurrent_window_notification_status_t	
DSMIF Overcurrent Window detect status.	
Enumerator	
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_0	Channel 0 overcurrent detection window notification 0 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_0	Channel 1 overcurrent detection window notification 0 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_0	Channel 2 overcurrent detection window notification 0 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_1	Channel 0 overcurrent detection window notification 1 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_1	Channel 1 overcurrent detection window notification 1 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_1	Channel 2 overcurrent detection window notification 1 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_2	Channel 0 overcurrent detection window notification 2 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_2	Channel 1 overcurrent detection window notification 2 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_2	Channel 2 overcurrent detection window notification 2 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_3	Channel 0 overcurrent detection window notification 3 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_3	Channel 1 overcurrent detection window notification 3 flag.
DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_3	Channel 2 overcurrent detection window notification 3 flag.

Function Documentation

◆ R_DSMIF_Open()

fsp_err_t R_DSMIF_Open (adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg*)

Applies power to the DSMIF and initializes the hardware based on the user configuration. As part of this initialization, set interrupts, set DSMIF error interrupt registers, etc.

Return values

FSP_SUCCESS	Configuration successful.
FSP_ERR_ASSERTION	An input pointer is NULL or an input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Control block is already open.
FSP_ERR_IP_UNIT_NOT_PRESENT	The Unit requested in the p_cfg parameter is not available on this device.

◆ R_DSMIF_ScanCfg()

fsp_err_t R_DSMIF_ScanCfg (adc_ctrl_t * *p_ctrl*, void const *const *p_extend*)

adc_api_t::scanCfg is not supported on DSMIF.

Return values

FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_ScanStart()

fsp_err_t R_DSMIF_ScanStart (adc_ctrl_t * *p_ctrl*)

Starts all configured DSMIF channels

Note

After executing R_DSMIF_ScanStart, it is necessary to wait until the filter result is output stably.

Return values

FSP_SUCCESS	Scan started or hardware triggers enabled successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_TIMEOUT	Timeout occurred while waiting for the channel to start running.

◆ R_DSMIF_ScanGroupStart()

fsp_err_t R_DSMIF_ScanGroupStart (adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_mask*)

adc_api_t::scanGroupStart is not supported on DSMIF.

Return values

FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_ScanStop()

fsp_err_t R_DSMIF_ScanStop (adc_ctrl_t * *p_ctrl*)

Stops any in-progress scan started by software.

Return values

FSP_SUCCESS	Scan stopped or hardware triggers disabled successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_TIMEOUT	Timeout occurred while waiting for the channel to stop.

◆ R_DSMIF_ScanStatusGet()

fsp_err_t R_DSMIF_ScanStatusGet (adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status*)

Returns the status of a scan started by software.

Return values

FSP_SUCCESS	No software scan is in progress.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_DSMIF_Read()

fsp_err_t R_DSMIF_Read (adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data*)

Reads the most recent result from a channel.

Return values

FSP_SUCCESS	Result in p_data.
FSP_ERR_ASSERTION	An input pointer was NULL or an input parameter was invalid.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_Read32()

fsp_err_t R_DSMIF_Read32 (adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data*)

Reads the most recent conversion result from a single channel into a 32-bit result.

Return values

FSP_SUCCESS	Result in p_data.
FSP_ERR_ASSERTION	An input pointer was NULL or an input parameter was invalid.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_Calibrate()

fsp_err_t R_DSMIF_Calibrate (adc_ctrl_t * *p_ctrl*, void const * *p_extend*)

adc_api_t::calibrate is not supported on DSMIF.

Return values

FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_OffsetSet()

fsp_err_t R_DSMIF_OffsetSet (adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t const *offset*)

adc_api_t::offsetSet is not supported on DSMIF.

Return values

FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_CallbackSet()

fsp_err_t R_DSMIF_CallbackSet (adc_ctrl_t *const *p_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void *const *p_context*, adc_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_DSMIF_Close()

fsp_err_t R_DSMIF_Close (adc_ctrl_t * *p_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the DSMIF peripheral.

Note

This function is delayed at least until the outage is complete, as required by the DSMIF outage procedure.

Return values

FSP_SUCCESS	Instance control block closed successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_DSMIF_InfoGet()

fsp_err_t R_DSMIF_InfoGet (adc_ctrl_t *const *p_ctrl*, adc_info_t *const *p_adc_info*)

adc_api_t::infoGet is not supported on DSMIF.

Return values

FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_ErrorDetectionEnable()

fsp_err_t R_DSMIF_ErrorDetectionEnable (adc_ctrl_t * *p_ctrl*)

Enable error detection.

Return values

FSP_SUCCESS	Configuration setting successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

◆ R_DSMIF_ErrorStatusGet()

fsp_err_t R_DSMIF_ErrorStatusGet (adc_ctrl_t * *p_ctrl*, dsmif_error_status_t * *p_error_status*)

Returns the error status of a scan started by software.

Return values

FSP_SUCCESS	No software scan is in progress.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_UNSUPPORTED	This function is not supported by the DSMIF instance.

5.2.2 AI

Modules

Artificial Intelligence Modules.

5.2.3 Audio

Modules

Detailed Description

Audio Modules.

Modules

	Pulse Density Modulation Interface (r_pdm) Driver for the PDM peripheral on RX MCUs. This module implements the PDM Interface.

5.2.3.1 Pulse Density Modulation Interface (r_pdm)

Modules » Audio

Functions

fsp_err_t	R_PDM_Open (pdm_ctrl_t *const p_ctrl, pdm_cfg_t const *const p_cfg)
fsp_err_t	R_PDM_Start (pdm_ctrl_t *const p_ctrl, void *const p_buffer, size_t const buffer_size, uint32_t const number_of_data_to_callback)
fsp_err_t	R_PDM_Stop (pdm_ctrl_t *const p_ctrl)
fsp_err_t	R_PDM_SoundDetectionEnable (pdm_ctrl_t *const p_ctrl, pdm_sound_detection_setting_t sound_detection_setting)
fsp_err_t	R_PDM_SoundDetectionDisable (pdm_ctrl_t *const p_ctrl)
fsp_err_t	R_PDM_Read (pdm_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)
fsp_err_t	R_PDM_StatusGet (pdm_ctrl_t *const p_ctrl, pdm_status_t *const p_status)
fsp_err_t	R_PDM_Close (pdm_ctrl_t *const p_ctrl)

fsp_err_t	R_PDM_CallbackSet (pdm_ctrl_t *const p_ctrl, void(*p_callback)(pdm_callback_args_t *), void *const p_context, pdm_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the PDM peripheral on RX MCUs. This module implements the PDM Interface.

Overview

The Pulse Density Modulation Interface (PDM-IF) has channels that are connectable with external microphone which outputs the pulse density modulated signal. PDM-IF can filter and convert 1-bit digital data streams that were pulse density modulated at a high sampling rate into 20-bit or 16-bit digital data at a lower sampling rate.

Features

- Capable of filtering 1-bit digital input data and converting them into 20-bit or 16-bit digital data. The bit order is little-endian.
- Sound detection can be enabled for each channel. A moving average filter can be enabled for sound detection. Sound detection can be used to wake up the MCU from software standby mode.
- Each channel of IP includes programmable filters: 4th order sinc filter, high-pass filter (for suppression of DC bias), correction filter (for sinc passband distortion), and half-band decimation filter (for aliasing distortion). – IP supports programmable and flexible decimation ratios. – The sinc filter is selectable as first-, second-, third-, fourth-order.
- IP supports DMA operation through APB. – Each channel has a 32 stage FIFO for receiving converted PDM data.
- Configurable callback for each channel which notifies of received data based on a configurable threshold as well as sound detection and error events.

Configuration

Build Time Configurations for r_pdm

The following build time configurations are defined in fsp_cfg/r_pdm_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DMAC Support	• Enabled • Disabled	Disabled	If enabled code for DMAC transfer support is included in the build.

Configurations for Audio > PDM (r_pdm)

This module can be added to the Stacks tab via New Stack > Audio > PDM (r_pdm).

Configuration	Options	Default	Description
Input > Edge	• Rising • Falling	Rising	Select the edge of input audio data.
Input > PDM_CLK Frequency	Refer to the RX Configuration tool for available options.	module.driver.pdm.clock_div.clock_div_2	Select PDM_CLK Frequency (Hz).
Output > Bit Format	• 20 Bits mode {1 Sign bit and [18:0] bits of the buffer} • 20 Bits mode {2 Sign bits and [18:1] bits of the buffer} • 20 Bits mode {3 Sign bits and [18:2] bits of the buffer} • 20 Bits mode {4 Sign bits and [18:3] bits of the buffer} • 16 Bits mode {1 Sign bit and [18:4] bits of the buffer} • 16 Bits mode {1 Sign bit and [17:3] bits of the buffer} • 16 Bits mode {1 Sign bit and [16:2] bits of the buffer} • 16 Bits mode {1 Sign bit and [15:1] bits of the buffer} • 16 Bits mode {1 Sign bit and [14:0] bits of the buffer}	module.driver.pdm.pcm_width.pcm_width_20bits_0_18	Select the bit format of the PCM data stored into the data FIFO.
Output > Target Sampling Frequency (Hz)	Manual Entry	32000	Select Target sampling Frequency (Hz).
Filter > Low-pass Filter > Low-pass Filter Input Shift	• No shift • 1-bit right shift • 2-bits right shift • 3-bits right shift	No shift	Select the low-pass (half-band decimation) filter input shift setting.
Filter > Low-pass Filter > LPF Coefficient	Value must be an integer between	0x0400	Specify the Low-Pass Filter Coefficient h0(10)

h0(10)	0x0000 and 0x1FFF		
Filter > Low-pass Filter > LPF Coefficient h1(0)	Value must be an integer between 0x0000 and 0x1FFF	0x1FF8	Specify the Low-Pass Filter Coefficient h1(0)
Filter > Low-pass Filter > LPF Coefficient h1(1)	Value must be an integer between 0x0000 and 0x1FFF	0x000A	Specify the Low-Pass Filter Coefficient h1(1)
Filter > Low-pass Filter > LPF Coefficient h1(2)	Value must be an integer between 0x0000 and 0x1FFF	0x1FF0	Specify the Low-Pass Filter Coefficient h1(2)
Filter > Low-pass Filter > LPF Coefficient h1(3)	Value must be an integer between 0x0000 and 0x1FFF	0x0018	Specify the Low-Pass Filter Coefficient h1(3)
Filter > Low-pass Filter > LPF Coefficient h1(4)	Value must be an integer between 0x0000 and 0x1FFF	0x1FDC	Specify the Low-Pass Filter Coefficient h1(4)
Filter > Low-pass Filter > LPF Coefficient h1(5)	Value must be an integer between 0x0000 and 0x1FFF	0x0034	Specify the Low-Pass Filter Coefficient h1(5)
Filter > Low-pass Filter > LPF Coefficient h1(6)	Value must be an integer between 0x0000 and 0x1FFF	0x1FB3	Specify the Low-Pass Filter Coefficient h1(6)
Filter > Low-pass Filter > LPF Coefficient h1(7)	Value must be an integer between 0x0000 and 0x1FFF	0x0076	Specify the Low-Pass Filter Coefficient h1(7)
Filter > Low-pass Filter > LPF Coefficient h1(8)	Value must be an integer between 0x0000 and 0x1FFF	0x1F2E	Specify the Low-Pass Filter Coefficient h1(8)
Filter > Low-pass Filter > LPF Coefficient h1(9)	Value must be an integer between 0x0000 and 0x1FFF	0x0289	Specify the Low-Pass Filter Coefficient h1(9)
Filter > Low-pass Filter > LPF Coefficient h1(10)	Value must be an integer between 0x0000 and 0x1FFF	0x0289	Specify the Low-Pass Filter Coefficient h1(10)
Filter > Low-pass Filter > LPF Coefficient h1(11)	Value must be an integer between 0x0000 and 0x1FFF	0x1F2E	Specify the Low-Pass Filter Coefficient h1(11)
Filter > Low-pass Filter > LPF Coefficient h1(12)	Value must be an integer between 0x0000 and 0x1FFF	0x0076	Specify the Low-Pass Filter Coefficient h1(12)
Filter > Low-pass Filter > LPF Coefficient h1(13)	Value must be an integer between 0x0000 and 0x1FFF	0x1FB3	Specify the Low-Pass Filter Coefficient h1(13)
Filter > Low-pass Filter > LPF Coefficient	Value must be an integer between	0x0034	Specify the Low-Pass Filter Coefficient h1(14)

h1(14)	0x0000 and 0x1FFF		
Filter > Low-pass Filter > LPF Coefficient h1(15)	Value must be an integer between 0x0000 and 0x1FFF	0x1FDC	Specify the Low-Pass Filter Coefficient h1(15)
Filter > Low-pass Filter > LPF Coefficient h1(16)	Value must be an integer between 0x0000 and 0x1FFF	0x0018	Specify the Low-Pass Filter Coefficient h1(16)
Filter > Low-pass Filter > LPF Coefficient h1(17)	Value must be an integer between 0x0000 and 0x1FFF	0x1FF0	Specify the Low-Pass Filter Coefficient h1(17)
Filter > Low-pass Filter > LPF Coefficient h1(18)	Value must be an integer between 0x0000 and 0x1FFF	0x000A	Specify the Low-Pass Filter Coefficient h1(18)
Filter > Low-pass Filter > LPF Coefficient h1(19)	Value must be an integer between 0x0000 and 0x1FFF	0x1FF8	Specify the Low-Pass Filter Coefficient h1(19)
Filter > Compensation Filter > Compensation Filter Input Shift	• No shift • 1-bit right shift • 2-bits right shift • 3-bits right shift	No shift	Select the compensation filter input shift setting.
Filter > Compensation Filter > Compensation Filter Coefficient h(0)	Value must be an integer between 0x0000 and 0x1FFF	0x1FE8	Specify the Compensation Filter Coefficient h(0)
Filter > Compensation Filter > Compensation Filter Coefficient h(1)	Value must be an integer between 0x0000 and 0x1FFF	0x0039	Specify the Compensation Filter Coefficient h(1)
Filter > Compensation Filter > Compensation Filter Coefficient h(2)	Value must be an integer between 0x0000 and 0x1FFF	0x003C	Specify the Compensation Filter Coefficient h(2)
Filter > Compensation Filter > Compensation Filter Coefficient h(3)	Value must be an integer between 0x0000 and 0x1FFF	0x1E56	Specify the Compensation Filter Coefficient h(3)
Filter > Compensation Filter > Compensation Filter Coefficient h(4)	Value must be an integer between 0x0000 and 0x1FFF	0x01DC	Specify the Compensation Filter Coefficient h(4)
Filter > Compensation Filter > Compensation Filter Coefficient h(5)	Value must be an integer between 0x0000 and 0x1FFF	0x06E1	Specify the Compensation Filter Coefficient h(5)
Filter > Compensation Filter > Compensation Filter Coefficient h(6)	Value must be an integer between 0x0000 and 0x1FFF	0x01DC	Specify the Compensation Filter Coefficient h(6)
Filter > Compensation Filter > Compensation Filter Coefficient h(7)	Value must be an integer between 0x0000 and 0x1FFF	0x1E56	Specify the Compensation Filter Coefficient h(7)
Filter > Compensation	Value must be an	0x003C	Specify the

Filter > Compensation Filter Coefficient h(8)	integer between 0x0000 and 0x1FFF		Compensation Filter Coefficient h(8)
Filter > Compensation Filter > Compensation Filter Coefficient h(9)	Value must be an integer between 0x0000 and 0x1FFF	0x0039	Specify the Compensation Filter Coefficient h(9)
Filter > Compensation Filter > Compensation Filter Coefficient h(10)	Value must be an integer between 0x0000 and 0x1FFF	0x1FE8	Specify the Compensation Filter Coefficient h(10)
Filter > High-pass Filter > High-pass Filter Input Shift	• No shift • 1-bit right shift • 2-bits right shift • 3-bits right shift	No shift	Select the high-pass filter input shift setting.
Filter > High-pass Filter > HPF Coefficient s(0)	Value must be an integer between 0x0000 and 0xFFFF	0x3F61	Specify the High-Pass Filter Coefficient s(0)
Filter > High-pass Filter > HPF Coefficient k(1)	Value must be an integer between 0x0000 and 0xFFFF	0x3EC1	Specify the High-Pass Filter Coefficient k(1)
Filter > High-pass Filter > HPF Coefficient h(0)	Value must be an integer between 0x0000 and 0xFFFF	0x4000	Specify the High-Pass Filter Coefficient h(0)
Filter > High-pass Filter > HPF Coefficient h(1)	Value must be an integer between 0x0000 and 0xFFFF	0xC000	Specify the High-Pass Filter Coefficient h(1)
Filter > Moving Average Mode	• 1-order (filter is skipped) • 2-order • 4-order	1-order (filter is skipped)	Select the moving average mode for sound detection data.
Filter > Sinc Filter Mode	• 1-order • 2-order • 3-order • 4-order	4-order	Select the sinc filter mode setting.
Data Reception > Buffer Overwrite Detection Enable	• Enabled • Disabled	Enabled	Select buffer overwrite detection.
Data Reception > Data Reception Interrupt Threshold	• 1 or more data • 2 or more data • 4 or more data • 8 or more data • 16 or more data	16 or more data	Select Data reception interrupt threshold.
Short Circuit Detection > Short Circuit Detection Enable	• Enabled • Disabled	Disabled	Select short circuit detection.
Short Circuit Detection > Short Circuit Detection High Continuous Detection	Value must be an integer between 0x0000 and 0x1FFF	0x0	Specify the Short circuit detection High Continuous detection count.

Count			
Short Circuit Detection > Short Circuit Detection Low Continuous Detection Count	Value must be an integer between 0x0000 and 0x1FFF	0x0	Short circuit detection Low Continuous detection count.
Overvoltage Detection > Overvoltage Detection Lower Limit Enable	• Enabled • Disabled	Disabled	Select overvoltage detection lower limit.
Overvoltage Detection > Overvoltage Detection Lower Limit	Value must be an integer between 0x0000 and 0xFFFFF	0x0	Specify the Overvoltage detection lower limit
Overvoltage Detection > Overvoltage Detection Upper Limit Enable	• Enabled • Disabled	Disabled	Select overvoltage detection upper limit.
Overvoltage Detection > Overvoltage Detection Upper Limit	Value must be an integer between 0x0000 and 0xFFFFF	0x0	Specify the Overvoltage detection upper limit
Interrupt > Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from all three interrupt service routines (ISR).
Interrupt > Sound Detection Interrupt Priority	MCU Specific Options		Select the sound detection interrupt priority.
Interrupt > Data Reception Interrupt Priority	MCU Specific Options		Select the data reception interrupt priority.
Interrupt > Error Detection Interrupt Priority	MCU Specific Options		Select the error detection interrupt priority.
Name	Name must be a valid C symbol	g_pdm0	Module name.
Channel	Value must be an non-negative integer	0	Specify the PDM channel.

Usage Notes

Startup and Settling Time

Upon calling R_PDM_Open the user must wait for the filters to settle and any connected microphones

to startup, otherwise PDM data may not be correct upon calling R_PDM_Start.

A macro PDMn_FILTER_SETTLING_TIME_US (where n is the channel number) is generated for the settling time. The user can use this macro to delay for the settling time via their preferred method of doing a delay.

In addition to the filter settling time any connected microphone startup time should be accounted for. This can be found in the connected microphone's datasheet.

Audio Data

PDM hardware converts PDM data to PCM data.

Data arrays have the following size, alignment, and length based on the "Bit Depth" setting:

Bit Depth	Array Data Type	Required Alignment
16 Bits	32-bit integer, right justified	4 byte alignment
20 Bits	32-bit integer, right justified	4 byte alignment

Data Format for 16 Bits

Bit	Description
31-20	Fixed to 0
19-16	Sign bits
15-0	Fixed-point data

Data Format for 20 Bits

Bit	Description
31-20	Fixed to 0
19-0	Fixed-point data

Modes of Operation

In addition to calling R_PDM_Open, other APIs need to be called to activate certain features of PDM. The following table lists what each API controls:

API	Features Enabled by API
R_PDM_Open	Over/Undervoltage detection, Short circuit detection
R_PDM_Start	Data reception via interrupts or DMAC depending configuration, error interrupts
R_PDM_SoundDetectionEnable	Sound detection, Sound detection thresholds, Error interrupts

Error Interrupts

Error interrupts are only enabled when data reception is enabled (R_PDM_Start has been called) OR

sound detection is enabled (R_PDM_SoundDetectionEnable has been called).

The following configured error flags will generate an interrupt request:

- Short-circuit Detection
- Overvoltage Detection
- Data buffer overwrite detection

R_PDM_StatusGet can be called to get error statuses if error interrupts are not enabled (e.g. if only R_PDM_Open has been called).

Low Power Mode

Note

In order to use PDM during low power mode MOCO must be enabled for software standby mode (see MOCOSCR register).

Some microphones support going into a low power mode by changing the PDM clock to a low frequency. In order to do this with r_pdm the module instance should be closed and reopened with a separate configuration that uses a lower PDM clock.

Limitations

- R_PDM_Read is not support, only read data when the interrupt occurs by ISR function or DMAC.
- When calling R_PDM_Start, any data in the FIFO before DATRE is enabled will be discarded.

Examples

Basic Example

This is a basic example of minimal use of the PDM in an application.

```
#define PDM_BUFFER_NUM_SAMPLES 1024 // Number of samples to store in data buffer
#define PDM_CALLBACK_NUM_SAMPLES PDM_BUFFER_NUM_SAMPLES / 2 // Number of samples to
store in data buffer before getting a callback
#define PDM_MIC_STARTUP_TIME_US 35000 // PDM microphone startup time, RA8P1_EK mic
startup time is 35ms
#define PDM_SDE_UPPER_LIMIT 5000 // Detect sound above this value
#define PDM_SDE_LOWER_LIMIT 0xFFF80000 // Detect sound below this value
uint32_t g_pdm0_buffer[PDM_BUFFER_NUM_SAMPLES];
void r_pdm_basic_messaging_core0_example (void)
{
 /* Open PDM instance. */
 fsp_err_t err = R_PDM_Open(&g_pdm0_ctrl, &g_pdm0_cfg);
    assert(FSP_SUCCESS == err);
```

```
 /* Wait for PDM filters to settle. Use settling time calculated by e2 studio.
  * Also wait for PDM mic to startup. */
 R_BSP_SoftwareDelay(PDM0_FILTER_SETTLING_TIME_US + PDM_MIC_STARTUP_TIME_US,
BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable sound detection (if desired). */
 pdm_sound_detection_setting_t sound_detection_setting =
    {
       .sound_detection_lower_limit = PDM_SDE_LOWER_LIMIT,
       .sound_detection_upper_limit = PDM_SDE_UPPER_LIMIT
    };
    err = R_PDM_SoundDetectionEnable(&g_pdm0_ctrl, sound_detection_setting);
    assert(FSP_SUCCESS == err);
 /* Start receiving PDM data. */
    err = R_PDM_Start(&g_pdm0_ctrl, g_pdm0_buffer, sizeof(g_pdm0_buffer),
PDM_CALLBACK_NUM_SAMPLES);
    assert(FSP_SUCCESS == err);
 /* DATA PROCESSING HERE */
 /* Stop receiving PDM data. */
    err = R_PDM_Stop(&g_pdm0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Close PDM instance. */
    err = R_PDM_Close(&g_pdm0_ctrl);
    assert(FSP_SUCCESS == err);
}
void pdm_example_callback (pdm_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case PDM_EVENT_SOUND_DETECTION:
      {
 /* Sound has been detected based on configured thresholds. */
 break;
       }
 case PDM_EVENT_DATA:
```

```
     {
/* PDM_CALLBACK_NUM_SAMPLES has been stored in g_pdm0_buffer. */
break;
      }
case PDM_EVENT_ERROR:
     {
if (p_args->error & PDM_ERROR_SHORT_CIRCUIT)
     {
/* Short circuit has been detected based on configured counts */
        }
if (p_args->error & PDM_ERROR_OVERVOLTAGE_LOWER)
     {
/* Overvoltage has been detected based on configured lower limit */
        }
if (p_args->error & PDM_ERROR_OVERVOLTAGE_UPPER)
     {
/* Overvoltage has been detected based on configured upper limit */
        }
if (p_args->error & PDM_ERROR_BUFFER_OVERWRITE)
     {
/* Buffer overwrite has occurred */
        }
break;
      }
default:
     {
break;
      }
   }
}
```

Data Structures

struct	pdm_instance_ctrl_t
struct	pdm_extended_cfg_t

Macros

#define	PDM_NUM_HPF_COEFFICIENT_H Number of high pass filter h coefficients.
#define	PDM_NUM_COMPENSATION_FILTER_COEFFICIENT_H Number of compensation filter h coefficients.
#define	PDM_NUM_LPF_FILTER_COEFFICIENT_H1 Number of low pass filter h1 coefficients.

Enumerations

enum	pdm_clk_div_t
enum	pdm_moving_average_mode_t
enum	pdm_low_pass_filter_shift_t
enum	pdm_compensation_filter_shift_t
enum	pdm_high_pass_filter_shift_t
enum	pdm_sinc_filter_mode_t
enum	pdm_interrupt_threshold_t
enum	pdm_short_circuit_enable_t
enum	pdm_overvoltage_lower_limit_enable_t
enum	pdm_overvoltage_upper_limit_enable_t
enum	pdm_buffer_overwrite_detection_enable_t

Data Structure Documentation

◆ pdm_instance_ctrl_t

struct pdm_instance_ctrl_t
Channel instance control block. DO NOT INITIALIZE. Initialization occurs when pdm_api_t::open is called.

◆ pdm_extended_cfg_t

struct pdm_extended_cfg_t		
PDM configuration extension. This extension is optional.		
Data Fields		
pdm_clk_div_t	clock_div	PDM_CLKn's division ratio to core clock.
pdm_short_circuit_enable_t	short_circuit_detection_enable	Short circuit detection enable. Function Settings.
pdm_overvoltage_lower_limit_enable_t	over_voltage_lower_limit_detection_enable	Overvoltage detection lower limit enable.
pdm_overvoltage_upper_limit_enable_t	over_voltage_upper_limit_detection_enable	Overvoltage detection upper limit enable.
pdm_buffer_overwrite_detection_enable_t	buffer_overwrite_detection_enable	Buffer overwrite detection enable.
pdm_moving_average_mode_t	moving_average_mode	Moving average mode f or sound detection data. Filter Settings.
pdm_low_pass_filter_shift_t	low_pass_filter_shift	Low-pass (half -band decimation) filter input shift setting.
pdm_compensation_filter_shift_t	compensation_filter_shift	Compensation filter input shift setting.
pdm_high_pass_filter_shift_t	high_pass_filter_shift	High-pass filter input shift setting.
pdm_sinc_filter_mode_t	sinc_filter_mode	Sinc filter mode setting.
uint8_t	sincrng	Sinc filter output valid range.
uint8_t	sincdec	Sinc filter decimation ratio.
uint16_t	hpf_coefficient_s0	High-pass filter coefficient s0 (16-bit signed fixed point data)
uint16_t	hpf_coefficient_k1	High-pass filter coefficient k1 (16-bit signed fixed point data)
uint16_t	hpf_coefficient_h[PDM_NUM_HPF_COEFFICIENT_H]	High-pass filter coefficient h(p) (16-bit signed fixed point data)
uint16_t	compensation_filter_coefficient_h[PDM_NUM_COMPENSATION_FILTER_COEFFICIENT_H]	High-pass filter coefficient h(p) (13-bit signed fixed point data)
uint16_t	lpf_coefficient_h0	Low-pass (half-band decimation) filter coefficient h0 (13-bit signed fixed point data)

uint16_t	lpf_coefficient_h1[PDM_NUM_LPF_FILTER_COEFFICIENT_H1]	Low-pass (half-band decimation) filter coefficient h1 (13-bit signed fixed point data)
pdm_interrupt_threshold_t	interrupt_threshold	Data reception interrupt threshold. Data Reception.
uint16_t	short_circuit_count_h	Short circuit detection High Continuous detection count. Short-Circuit Detection.
uint16_t	short_circuit_count_l	Continuous detection count.
uint32_t	overvoltage_detection_lower_limit	Overvoltage detection lower limit (20-bit signed fixed point data) Overvoltage Detection.
uint32_t	overvoltage_detection_upper_limit	Overvoltage detection upper limit (20-bit signed fixed point data)

Enumeration Type Documentation

◆ pdm_clk_div_t

enum pdm_clk_div_t	
PDM_CLKn's division ratio to core clock.	
Enumerator	
PDM_CLOCK_DIV_2	0000b: 1/2
PDM_CLOCK_DIV_4	0001b: 1/4
PDM_CLOCK_DIV_6	0010b: 1/6
PDM_CLOCK_DIV_8	0011b: 1/8
PDM_CLOCK_DIV_10	0100b: 1/10
PDM_CLOCK_DIV_12	0101b: 1/12
PDM_CLOCK_DIV_14	0110b: 1/14
PDM_CLOCK_DIV_16	0111b: 1/16
PDM_CLOCK_DIV_18	1000b: 1/18
PDM_CLOCK_DIV_20	1001b: 1/20
PDM_CLOCK_DIV_22	1010b: 1/22
PDM_CLOCK_DIV_24	1011b: 1/24
PDM_CLOCK_DIV_26	1100b: 1/26
PDM_CLOCK_DIV_28	1101b: 1/28
PDM_CLOCK_DIV_30	1110b: 1/30
PDM_CLOCK_DIV_32	1111b: 1/32

◆ pdm_moving_average_mode_t

enum pdm_moving_average_mode_t	
Moving average mode for sound detection data.	
Enumerator	
PDM_MOVING_AVERAGE_MODE_1_ORDER	1-order
PDM_MOVING_AVERAGE_MODE_2_ORDER	2-order
PDM_MOVING_AVERAGE_MODE_4_ORDER	4-order

◆ pdm_low_pass_filter_shift_t

enum pdm_low_pass_filter_shift_t	
Low-pass (half-band decimation) filter input shift setting.	
Enumerator	
PDM_LPF_RIGHT_SHIFT_0	No shift.
PDM_LPF_RIGHT_SHIFT_1	1-bit right shift
PDM_LPF_RIGHT_SHIFT_2	2-bit right shift
PDM_LPF_RIGHT_SHIFT_3	3-bit right shift

◆ pdm_compensation_filter_shift_t

enum pdm_compensation_filter_shift_t	
Compensation filter input shift setting.	
Enumerator	
PDM_COMPENSATION_FILTER_RIGHT_SHIFT_0	No shift.
PDM_COMPENSATION_FILTER_RIGHT_SHIFT_1	1-bit right shift
PDM_COMPENSATION_FILTER_RIGHT_SHIFT_2	2-bit right shift
PDM_COMPENSATION_FILTER_RIGHT_SHIFT_3	3-bit right shift

◆ pdm_high_pass_filter_shift_t

enum pdm_high_pass_filter_shift_t	
High-pass filter input shift setting.	
Enumerator	
PDM_HPF_RIGHT_SHIFT_0	No shift.
PDM_HPF_RIGHT_SHIFT_1	1-bit right shift
PDM_HPF_RIGHT_SHIFT_2	2-bit right shift
PDM_HPF_RIGHT_SHIFT_3	3-bit right shift

◆ pdm_sinc_filter_mode_t

enum pdm_sinc_filter_mode_t	
Sinc filter mode setting.	
Enumerator	
PDM_SINC_FILTER_MODE_1	1-order
PDM_SINC_FILTER_MODE_2	2-order
PDM_SINC_FILTER_MODE_3	3-order
PDM_SINC_FILTER_MODE_4	4-order

◆ pdm_interrupt_threshold_t

enum pdm_interrupt_threshold_t	
Data reception interrupt threshold.	
Enumerator	
PDM_INTERRUPT_THRESHOLD_1	000b: Output interrupt when receiving 1 or more data
PDM_INTERRUPT_THRESHOLD_2	001b: Output interrupt when receiving 2 or more data
PDM_INTERRUPT_THRESHOLD_4	010b: Output interrupt when receiving 4 or more data
PDM_INTERRUPT_THRESHOLD_8	011b: Output interrupt when receiving 8 or more data
PDM_INTERRUPT_THRESHOLD_16	100b: Output interrupt when receiving 16 or more data

◆ pdm_short_circuit_enable_t

enum pdm_short_circuit_enable_t	
Short circuit detection enable setting.	
Enumerator	
PDM_SHORT_CIRCUIT_DISABLED	Short circuit disabled.
PDM_SHORT_CIRCUIT_ENABLED	Short circuit enabled.

◆ pdm_overvoltage_lower_limit_enable_t

enum pdm_overvoltage_lower_limit_enable_t	
Overvoltage lower limit enable setting.	
Enumerator	
PDM_OVERVOLTAGE_LOWER_LIMIT_DISABLED	Overvoltage lower limit disabled.
PDM_OVERVOLTAGE_LOWER_LIMIT_ENABLED	Overvoltage lower limit enabled.

◆ pdm_overvoltage_upper_limit_enable_t

enum pdm_overvoltage_upper_limit_enable_t	
Overvoltage upper limit enable setting.	
Enumerator	
PDM_OVERVOLTAGE_UPPER_LIMIT_DISABLED	Overvoltage upper limit disabled.
PDM_OVERVOLTAGE_UPPER_LIMIT_ENABLED	Overvoltage upper limit enabled.

◆ pdm_buffer_overwrite_detection_enable_t

enum pdm_buffer_overwrite_detection_enable_t	
Buffer overwrite detection enable setting.	
Enumerator	
PDM_BUFFER_OVERWRITE_DETECTION_DISABLED	Buffer overwrite detection disabled.
PDM_BUFFER_OVERWRITE_DETECTION_ENABLED	Buffer overwrite detection enabled.

Function Documentation

◆ R_PDM_Open()

fsp_err_t R_PDM_Open (pdm_ctrl_t *const *p_ctrl*, pdm_cfg_t const *const *p_cfg*)

Opens the PDM. Implements pdm_api_t::open.

This function sets this clock divisor and the configurations specified in pdm_cfg_t.

Return values

FSP_SUCCESS	Ready for PDM communication.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_cfg is null.
FSP_ERR_ALREADY_OPEN	The control block has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel number is not available on this device.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ R_PDM_Start()

fsp_err_t R_PDM_Start (pdm_ctrl_t *const *p_ctrl*, void *const *p_buffer*, size_t const *buffer_size*, uint32_t const *number_of_data_to_callback*)

Starts PDM. Implements pdm_api_t::start.

This function enables reception, and enables any transfer instances used.

The PDM will start on the next frame boundary.

Return values

FSP_SUCCESS	PDM communication stop request issued.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_INVALID_SIZE	The size of buffer is not a multiple of pdm_interrupt_threshold_t.
FSP_ERR_INVALID_ALIGNMENT	The alignment of argument buffer_size is not matched with hardware specification.
FSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_PDM_Stop()

fsp_err_t R_PDM_Stop (pdm_ctrl_t *const *p_ctrl*)

Stops PDM. Implements pdm_api_t::stop.

This function disables reception, and disables any transfer instances used.

The PDM will stop on the next frame boundary.

Return values

FSP_SUCCESS	PDM communication stop request issued.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_PDM_SoundDetectionEnable()

fsp_err_t R_PDM_SoundDetectionEnable (pdm_ctrl_t *const *p_ctrl*, pdm_sound_detection_setting_t *sound_detection_setting*)

Configures and enables sound detection. Implements pdm_api_t::soundDetectionEnable.

This function configures sound detection based on the passed in settings. It then enables sound detection.

Return values

FSP_SUCCESS	Sound detection configured and enabled.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_PDM_SoundDetectionDisable()

fsp_err_t R_PDM_SoundDetectionDisable (pdm_ctrl_t *const *p_ctrl*)

Disables sound detection. Implements pdm_api_t::soundDetectionDisable.

This function disables sound detection.

Return values

FSP_SUCCESS	Sound detection disabled.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_PDM_Read()

fsp_err_t R_PDM_Read (pdm_ctrl_t *const *p_ctrl*, void *const *p_dest*, uint32_t const *bytes*)

This function is not supported. Implements pdm_api_t::read.

Return values

FSP_ERR_UNSUPPORTED	This function is not supported.

◆ R_PDM_StatusGet()

fsp_err_t R_PDM_StatusGet (pdm_ctrl_t *const *p_ctrl*, pdm_status_t *const *p_status*)

Gets PDM status and stores it in provided pointer p_status. Implements pdm_api_t::statusGet.

Calling this function will clear any set error flags in the hardware.

Return values

FSP_SUCCESS	Information stored successfully.
FSP_ERR_ASSERTION	The p_ctrl or p_status parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_PDM_Close()

fsp_err_t R_PDM_Close (pdm_ctrl_t *const *p_ctrl*)

Closes PDM. Implements pdm_api_t::close.

This function powers down the PDM and closes the lower level timer and transfer drivers if they are used.

Return values

FSP_SUCCESS	Device closed successfully.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_PDM_CallbackSet()

fsp_err_t R_PDM_CallbackSet (pdm_ctrl_t *const *p_ctrl*, void(*)(pdm_callback_args_t *) *p_callback*, void *const *p_context*, pdm_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements pdm_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.4 Bootloader

Modules

Detailed Description

Bootloader Modules.

Modules

MCUboot Port (rm_mcuboot_port)	MCUboot Port for RX MCUs.

5.2.4.1 MCUboot Port (rm_mcuboot_port)

Modules » Bootloader

MCUboot Port for RX MCUs.

Overview

Note

The MCUboot Port does not provide any interfaces to the user. Consult the MCUboot documentation at https://mcu-tools.github.io/mcuboot/ for further information.

Configuration

Build Time Configurations for MCUboot

The following build time configurations are defined in mcu-tools/include/mcuboot_config/mcuboot_config.h:

Configuration	Options	Default	Description
General > Custom mcuboot_config.h	Manual Entry		Add a path to your custom mcuboot_config.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
General > Upgrade Mode	• Swap • Overwrite Only	Overwrite Only	Swap supports A/B image swapping with

	• Overwrite Only Fast • Direct XIP		rollback. Other modes with simpler code path, which only supports overwriting the existing image with the update image (Overwrite Only) or running the newest image directly from its flash partition (Direct XIP), are also available.
General > Validate Primary Image	• Enabled • Disabled	Enabled	Always check the signature of the image in the primary slot before booting, even if no upgrade was performed. This is recommended if the boot time penalty is acceptable.
General > Downgrade Prevention (Overwrite Only)	• Enabled • Disabled	Disabled	Prevent downgrades by enforcing incrementing version numbers. When this option is set, any upgrade must have greater major version or greater minor version with equal major version. This mechanism only protects against some attacks against version downgrades (for example, a JTAG could be used to write an older version).
General > Watchdog Feed	Manual Entry		This function might be implemented if the OS / HW watchdog is enabled while doing a swap upgrade and the time it takes for a swapping is long enough to cause an unwanted reset. If implementing this, the OS main.c must also enable the watchdog (if required)!
General > Measured Boot	• Enabled • Disabled	Disabled	Copies the boot data into the secure RAM, intended to be used by the secure App.

General > Data Sharing	• Enabled • Disabled	Disabled	Copies the user data into the secure RAM, intended to be used by the secure App.
Signing and Encryption Options > TrustZone > Boot Record (Image 2)	String length must be 12 characters or less.		Create CBOR encoded boot record TLV for Image 2. Represents the role of the software component (e.g. CoFM for coprocessor firmware). [max. 12 characters]
Signing and Encryption Options > TrustZone > Custom (Image 2)	Manual Entry	--confirm	Add any custom options to pass to imgtool.py here. --pad places a trailer on the image that indicates that the image should be considered an upgrade. Writing this image in the secondary slot will then cause the bootloader to upgrade to it. --confirm marks the image as confirmed, which causes the upgrade to be permanent.
Signing and Encryption Options > Signature Type	• None • ECDSA P-256 • ECDSA P-384 • RSA 2048 • RSA 3072 • ML-DSA 44 • ML-DSA 65	ECDSA P-256	Configure the signature type.
Signing and Encryption Options > Boot Record	String length must be 12 characters or less.		Create CBOR encoded boot record TLV. Represents the role of the software component (e.g. CoFM for coprocessor firmware). [max. 12 characters]
Signing and Encryption Options > Custom	Manual Entry	--confirm	Add any custom options to pass to imgtool.py here. --pad places a trailer on the image that indicates that the image should be considered an upgrade. Writing this image in the secondary

			slot will then cause the bootloader to upgrade to it. --confirm marks the image as confirmed, which causes the upgrade to be permanent.
Signing and Encryption Options > Python	Manual Entry	python	Name of the python command to use. Default is python, but can be updated to python3 for Linux or an absolute path if needed.
Signing and Encryption Options > Encryption Scheme	• ECIES-P256 • RSA-OAEP (RSA 2048 only) • Encryption Disabled	Encryption Disabled	Choose the encryption scheme.
Flash Configuration > Dual Bank Mode	Disabled	0	Enable dual bank mode for Direct XIP, Not support on the RX74M/N.
Flash Configuration > Memory Mirror Function (MMF) support	Disabled	0	Use MMF for MCUBoot dxip mode, Not support on the RX MCU.
Data Sharing > Maximum Measured Boot Record Size (Bytes)	Value must be an integer.	0x64	Maximum size of the boot record.
Data Sharing > Shared Data Size (Bytes)	Value must be an integer.	0x380	Size of the shared RAM area. Required for Measured Boot.
Data Sharing > Shared Data Address	Value must be an integer	0xFFFFF000	Shared RAM start address. Required for Measured Boot.

Clock Configuration

- This module does not use peripheral clocks.
- For best performance it is recommended to use the fastest clock settings supported by the device in order to reduce boot times.
- **The bootloader must not disable the MOCO prior to calling the application. If the bootloader disables the MOCO, then operation cannot be guaranteed.**

Pin Configuration

This module does not use I/O pins.

Usage Notes

Getting Started: Creating an MCUboot Project

Start by creating a new project in e² studio or RXSC. Select 'Flat Project' template for single-image projects.

Add either the example keys or generate your own key. The MCUboot key generation tool is provided at rx/mcu-tools/MCUboot/scripts/imgtool.py and documented at https://github.com/mcu-tools/mcuboot/blob/master/docs/imgtool.md. Install the following required python packages to use imgtool.py: https://github.com/mcu-tools/mcuboot/blob/master/scripts/requirements.txt.

In src/hal_entry.c, drag in Developer Assistance > HAL/Common > MCUboot > Quick Setup > Call Quick Setup. Add a call to mcuboot_quick_setup() in the application and make any desired updates.

Note

MCUboot will contain either the verification public key or its hash. During production it is necessary to permanently lock the flash region where MCUboot is programmed to prevent the keys or the code from being modified.

MCUboot Layout

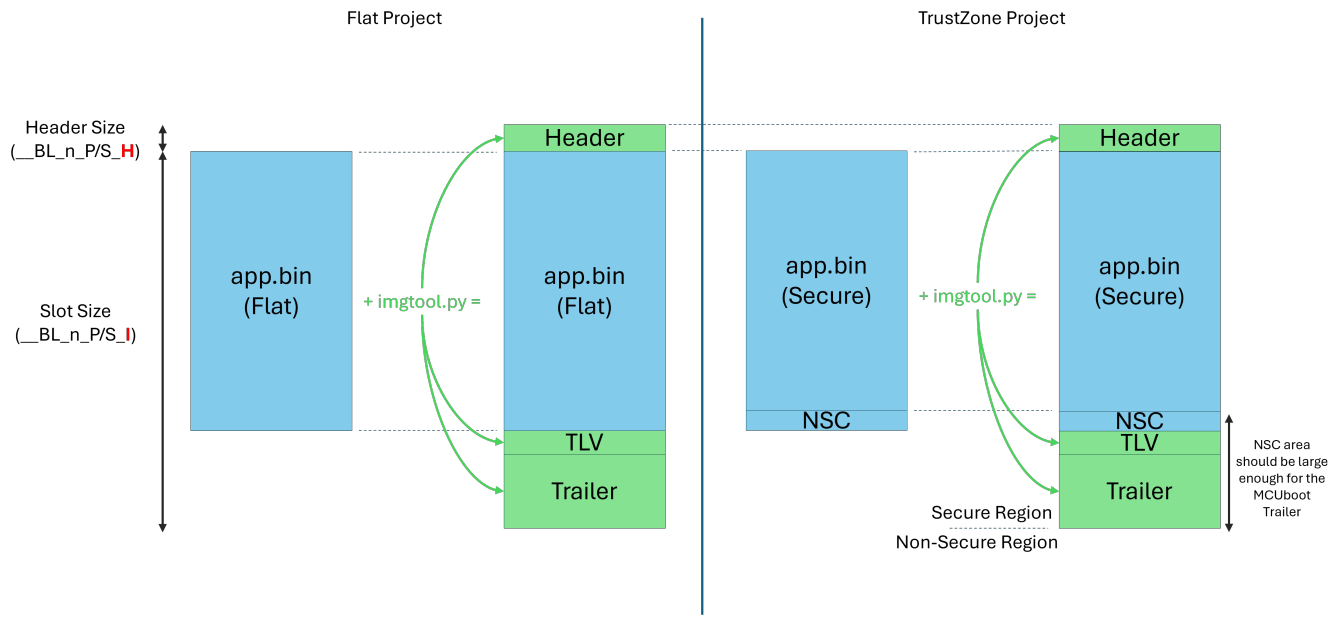


Figure 97: imgtool.py siging script adds header and trailer to user binary

Getting Started: Signing Tool Prerequisite

To use the MCUboot signing tool, ensure you have Python 3.x installed on your system. Then install the Python packages required for the signing tool with the following command (modifying the path as needed depending on current directory):

pip3 install --user -r rx/mcu-tools/MCUboot/scripts/requirements.txt

Getting Started: Converting an Application Project to an MCUboot Image

MCUboot application images must execute from the image slot defined by the MCUboot project. They are also limited to a single downloadable flash region. All of this is configured in the MCUboot module and the information is added to the generated SecureBundle (.sbd) file in the MCUboot

project.

Any existing project can be converted to an MCUboot image.

1. Right click the project to convert in e² studio or RXSC and select Properties.
2. Open C/C++ Build and select Build Variables.
3. Click Add...
4. For Variable Name, enter SmartBundle. For Type, select File. Browse to the *.sbd file created alongside the *.elf file for the associated MCUboot project. If the MCUboot project was specified as the Preceding Project during project creation, this step is done automatically.
5. Click OK, then Apply and Close.

MCUboot application images must also be signed to work with MCUboot. At a minimum, this involves adding a SHA and MCUboot specific constant data called boot magic in the image trailer. The size of the image trailer is (5 * BOOT_MAX_ALIGN) + (BOOT_MAX_IMG_SECTORS * min-write-size * 3).

Signing can be done on the as a post-build step in e² studio. To sign the image as a post-build step:

1. If Linux is used to develop the application image, change the MCUboot property Signing > Python to python3.
2. Build the bootloader project to generate the *.bld file. Make sure to build the bootloader project on the same computer as the application image to ensure the path to the signing script is correct.
3. Define environment variables in the Properties of the application image project in e² studio.
    a. Right click the application image project, and select **Properties**.
    b. Select **C/C++ Build > Environment** on the left.
    c. Click **Add...**
    d. Define the following environment variables one at a time:
        - MCUBOOT_IMAGE_VERSION: Set to the version of the application image.
        - MCUBOOT_IMAGE_SIGNING_KEY: Set the path to the key used for signing. If signing is not required, do not set this variable. If example keys are used, set MCUBOOT_IMAGE_SIGNING_KEY as follows (replace <boot_project> with the bootloader project path):
            - ECC 256: <boot_project>/rx/mcu-tools/MCUboot/root-ec-p256.pem
            - ECC 384: <boot_project>/rx/mcu-tools/MCUboot/root-ec-p384.pem
            - RSA 2K: <boot_project>/rx/mcu-tools/MCUboot/root-rsa-2048.pem
            - RSA 3K: <boot_project>/rx/mcu-tools/MCUboot/root-rsa-3072.pem
        - MCUBOOT_IMAGE_ENC_KEY: Set the path to the key used for encryption. If encryption is not required, do not set this variable. If example keys are used, set MCUBOOT_IMAGE_ENC_KEY as follows (replace <boot_project> with the bootloader project path):
            - ECIES: <boot_project>/rx/mcu-tools/MCUboot/enc-ec256-pub.pem
            - RSA 2K: <boot_project>/rx/mcu-tools/MCUboot/enc-rsa2048-pub.pem
        - MCUBOOT_APP_BIN_CONVERTER: Optional. Set to path to objcopy, arm-none-eabi-objcopy, fromelf, or ielftool. Not required if one of these tools is on the path.
4. Build the project.
5. The signed image is output next to the application <project>.elf file with the name

<project>bin.signed.

Bootloader region encoding

The header and image regions have to be named in the manner described below in the Solution project:

- FLASH_<core> - This indicates the internal flash memory where the boot loader area or primary area is located. These are created by default and cannot be renamed.
- Examples:
    - FLASH_RXv3e_B: RXv3e Bootloader area.
- __BL: Indicates this is a bootloader symbol.
- _: delimiter between fields.
- ([0-9])|S: Image definitions field.
    - S indicates the Scratch area.
    - 0-9 correspond to traditional image IDs (only using 0/1 in legacy).
- _: delimiter between fields.
- (P|S): Indicates Primary or Secondary area.
    - S Secondary areas must have both Header (H) and Image (I) component.
    - P Primary areas must not have an I component (as the Image component is defined by the solution application region).
    - Secondary header must be the same size as Primary header.
    - Secondary images must be the same size as the area allocated for the project (Primary Image).
    - Headers must immediately precede the Image/application space.
- _: delimiter between fields
- (H|I): Indicates bootloader Header or Image component (not used with Scratch).
    - H indicates the MCUboot Header for Primary or Secondary (P/S) areas (use 0x200 for compatibility).
    - I indicates the MCUboot Secondary (S) Image allocation space (not used with Primary images).
- Bootloader flash area size must be an integer multiple of the largest erase size on the mcu.
- Primary or Secondary area size (including header and trailer) must be an integer multiple of the largest erase size on the mcu.
- Examples:
    - __BL_0_P_H: Image 0 Primary Header.
    - __BL_0_S_H: Image 0 Secondary Header.
    - __BL_1_S_I: Image 1 Secondary Image.
    - __BL_S : Scratch area.

Using Bootloader Solution Templates

MCUBoot templates can be used to generate a pre-configured Solution and required projects. RX FSP recommends using this to generate solution projects. The template can be generated from the New C/C++ Project wizard -> Renesas RX FSP Solution. The "Single core" -> "Bootloader" -> "Single Image Bootloader" option in the Renesas RX FSP Solution is the solution template for MCUBoot. In the template, regions are defined in the following order: MCUboot Image -> Primary Image -> Secondary Image -> Scratch Area. Please do not change this order in the current version.

Getting Started: Download and Debug

After all images have finished building, debug the MCUboot project using the debug configuration for the xxx_app2 project. Before running, load the signed primary image to the address indicated by __BL_0_P_H. This can be done using the "Load Auxiliary File" button while debugging in e²studio. The secondary image can be loaded into the secondary image slot in the same way.

1. In the Debug Configurations for your project, on the Startup tab, click Add... to add the MCUboot project *.elf file (Image and Symbols), and optionally the Non-Secure project *.elf file.
2. For the Secure and Non-Secure project *.elf file, load Symbols Only.
3. After starting to debug, load the signed primary image to the address indicated by __BL_0_P_H. This can be done using the "Load Auxiliary File" button while debugging in e²studio. The secondary image can be loaded into the secondary image slot in the same way.

Note

Before writing a new boot image to MRAM or Flash memory, ensure that the secondary image slot and scratch area are erased. If any data other than the initial values remain in these areas, MCUBoot may mistakenly consider it valid data, potentially causing unexpected behavior.

Confirming Upgrade in Swap Mode

In Swap Mode operation, if the upgrade image is signed with the –pad option, MCUboot will install that image as a temporary update where if nothing else is done, a reboot will cause MCUboot to revert to the image version that was swapped out during the upgrade. In order for the updated image to prevent this reversion and make the update permanent, the boot_set_confirmed() must be called from the application.

To avail this capability in the application image, from the Stacks tab, add New > Bootloader > MCUboot Image Utilities (Swap Mode). Resolve any constraint errors and edit configurations as desired.

In src/hal_entry.c, drag in Developer Assistance > HAL/Common > MCUboot Image Utilities > Quick Setup > Confirm Primary Image. Add a call to boot_set_confirmed() in the application and confirm the image in the primary slot.

XIP Mode operation

XIP mode is enabled selecting "Direct XIP" as the Upgrade Mode option in the configurator or by defining "MCUBOOT_DIRECT_XIP" in the mcuboot config file. When running a secondary image in XIP mode, you must modify the linker script of the secondary image project by following these steps.

- Copy <secondary_image_project>\Debug\memory_regions.ld and rename it to memory_regions_custom.ld.
- Open memory_regions_custom.ld and change the value of "FLASH_START" to the Start address of "__BL_0_S_I".
- Open <secondary_image_project>\script\fsp.ld and change "INCLUDE memory_regions.ld" to "INCLUDE memory_regions_custom.ld".
- Rebuild the secondary image project.

External Memory Support

XSPI

The current version does not support XSPI for secondary image storage.

MCUboot Memory Map

For single-image projects that do not support external memory, as provided by the Single Image Bootloer template, the default memory map is as follows:

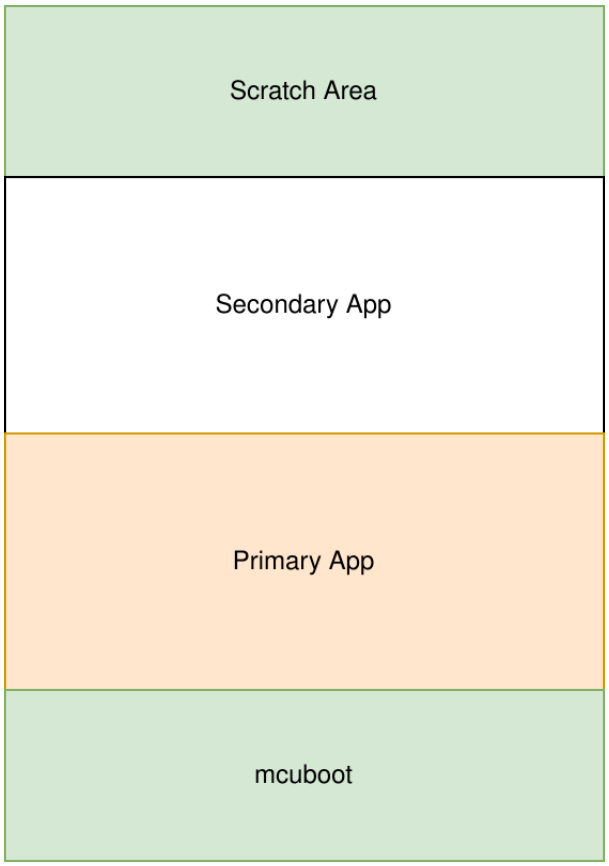


Figure 98: MCUboot Memory Map

MCUboot verification options

MCUboot in FSP supports the following secure image verification options

1. Hash verification only (SHA256).
2. Hash and signature verification (ECDSA-P256, RSA-2048).
3. Hash, signature verification and image encryption (ECIES-P256 and RSA-OAEP-2048 with AES-128).
4. Hash and image encryption only.

MCUboot also supports signature verification using EdDSA-25519 and image encryption using AES-KW-128 and AES-KW-256 but those are currently not supported in FSP.

Limitations

- XSPI external storage functionality for secondary images is not supported.
- RSIP protected mode is not supported.
- Some signature verification methods are not supported (ECDSA-P384, RSA-3072, ML-DSA-44, ML-DSA-65).

Notes

When encryption is enabled, MCUboot requires the image in the primary slot to be unencrypted. Only the image loaded in the secondary slot can be encrypted.

MCUboot Crypto Stack Options

The following crypto stacks can be used with MCUboot in FSP:

- MbedTLS, which is hardware accelerated on all RX devices.
- RSIP Protected Mode on devices that have the RSIP module.(This feature is not currently supported.)

MCUboot boot time

The time from Reset to executing the application will depend on how quickly MCUboot finishes execution. This is dependent on a variety of factors including

1. The crypto algorithms chosen for image verification and whether hardware acceleration is enabled. Hardware accelerated SHA256 will be the fastest while encryption enabled modes will be the slowest.
2. The operating clock speeds.
3. Whether flash programming was required.

Examples

Basic Example

This is an example of using MCUboot in an application.

```
void rm_mcuboot_port_example (void)
{
#ifdef MCUBOOT_USE_MBED_TLS
 /* Initialize mbedtls. */
 mbedtls_platform_context ctx = {0};
    assert(0 == mbedtls_platform_setup(&ctx));
#else
 /* Initialize RSIP Protected Mode driver. */
    assert(FSP_SUCCESS == R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg));
#endif
 /* (Optional) To check for updates, call boot_set_pending. */
 bool update = 0;
 if (update)
    {
       boot_set_pending(0);
    }
 /* Verify the boot image and get its location. */
 struct boot_rsp rsp;
    assert(0 == boot_go(&rsp));
```

```
 /* Enter the application. */
    RM_MCUBOOT_PORT_BootApp(&rsp);
}
```

5.2.5 CapTouch

Modules

CapTouch Modules.

5.2.6 Connectivity

Modules

Detailed Description

Connectivity Modules.

Modules

	CAN FD (r_canfd) Driver for the CANFD peripheral on RX MCUs. This module implements the CAN Interface.
	CAN FDG (r_canfdg) Driver for the CANFDG peripheral on RX MCUs. This module implements the CAN Interface.
	I3C (r_i3c) Driver for the I3C peripheral on RX MCUs. This module implements the I3C Interface.
	I2C Master (r_iic_b_master) I2C Driver for the IIC/I3C peripheral on RX MCUs. This module implements the I2C Master Interface.
	I2C Slave (r_iic_b_slave)

	Driver for the IIC/I3C peripheral on RX MCUs. This module implements the I2C Slave Interface.
	I2C Master (r_sci_b_i2c) Driver for the SCI_B peripheral on RX MCUs. This module implements the I2C Master Interface.
	SPI (r_sci_b_spi) Driver for the SCI peripheral on RX MCUs. This module implements the SPI Interface.
	UART (r_sci_b_uart) Driver for the SCI peripheral on RX MCUs. This module implements the UART Interface.
	SPI (r_spi_b) Driver for the SPI peripheral on RX MCUs. This module implements the SPI Interface.
	I2S (r_ssi) Driver for the SSIE peripheral on RX MCUs. This module implements the I2S Interface.
	USB (r_usb_basic) Driver for the USB peripheral on RX MCUs. This module implements the USB Interface.
	USB Composite (r_usb_composite)
	USB Host Communications Device Class (r_usb_hcdc) This module provides a USB Host Communications Device Class (HCDC) driver. It implements the USB HCDC Interface.
	USB Host Human Interface Device Class Driver (r_usb_hhid) This module provides a USB Host Human Interface Device Class Driver (HHID). It implements the USB HHID Interface.
	USB Host Mass Storage Class Driver (r_usb_hmsc)

	This module provides a USB Host Mass Storage Class (HMSC) driver. It implements the USB HMSC Interface.
	USB Host Vendor class (r_usb_hvnd)
	USB PCDC (r_usb_pcdc) This module provides a USB Peripheral Communications Device Class Driver (PCDC). It implements the USB PCDC Interface.
	USB PHID (r_usb_phid) This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.
	USB PMSC (r_usb_pmsc) This module provides a USB Peripheral Mass Storage Class (PMSC) driver. It implements the USB PMSC Interface.
	USB Peripheral Printer (r_usb_pprn) This module is USB Peripheral Printer Device Class Driver (PPRN). It implements the USB PPRN Interface.
	USB Peripheral Vendor class (r_usb_pvnd)
	USB_PCDC Communication Device (rm_comms_usb_pcdc) Middleware to implement a generic communications interface over USB_PCDC. This module implements the Communications Middleware Interface.
	Azure RTOS USBX Port (rm_usbx_port)

5.2.6.1 CAN FD (r_canfd)

Modules » Connectivity

Functions

fsp_err_t	R_CANFD_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg)
fsp_err_t	R_CANFD_Close (can_ctrl_t *const p_api_ctrl)

fsp_err_t	R_CANFD_Write (can_ctrl_t *const p_api_ctrl, uint32_t const buffer, can_frame_t *const p_frame)
fsp_err_t	R_CANFD_Read (can_ctrl_t *const p_api_ctrl, uint32_t const buffer, can_frame_t *const p_frame)
fsp_err_t	R_CANFD_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)
fsp_err_t	R_CANFD_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info)
fsp_err_t	R_CANFD_CallbackSet (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CANFD peripheral on RX MCUs. This module implements the CAN Interface.

Overview

The CANFD module can be used to communicate over CAN networks, optionally using Flexible Data (CAN-FD) to accelerate the data phase. A variety of message filtering and buffer options are available.

Features

Common Features

- Compatibility
  - Send and receive CAN 2.0 and CAN-FD frames on the same channel
  - Bitrate up to 1 Mbps with FD data phase speeds up to 8 Mbps
  - ISO 11898-1 Support
- Buffers
  - 32 global receive Message Buffers (RX MBs)
  - 2 global receive FIFOs (RX FIFOs)
  - 4 transmit Message Buffers (TX MBs)
- Filtering
  - Each filter rule can be individually configured to accept messages based on:
    - ID
    - Standard or Extended ID (IDE bit)
    - Data or Remote Frame (RTR bit)
    - ID/IDE/RTR mask
    - Minimum DLC (data length) value
- Interrupts
  - Configurable Global RX FIFO Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message
  - Configurable Common FIFO RX Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message

- Channel TX Interrupt
  - Interrupt on every transmitted message or when a Common FIFO is empty
- Global Error
  - DLC Check
  - Message Lost
  - FD Payload Overflow
- Channel Error
  - Bus Error
  - Error Warning
  - Error Passive
  - Bus-Off Entry
  - Bus-Off Recovery
  - Overload
  - Bus Lock
  - Arbitration Loss
  - Transmission Aborted

Per-MCU Specifications

	RX74M	RX74N
Channels	1	1
Max nominal bitrate	1 Mbps	1 Mbps
Max data bitrate	8 Mbps	8 Mbps
Filter rules	32	32
TX message buffers	4	4
RX message buffers	32	32
RX FIFOs	2	2
RX Buffer RAM	1216 bytes	1216 bytes
Common FIFOs	1	1

Note

Each message buffer comprises 12 header bytes plus data length (8-64 bytes). The above buffer RAM values therefore correspond to the following capacities:

	RX74M	*RX74N*
Max 64-byte storage	*16 messages*	*16 messages*
Max 8-byte storage	*60 messages*	*60 messages*

Each Common FIFO buffer can support the following message capacities:

	RX74M	*RX74N*
Maximum payload size	*64 bytes*	*64 bytes*
Maximum FIFO depth	*48 messages*	*48 messages*

Configuration

Build Time Configurations for r_canfd

The following build time configurations are defined in fsp_cfg/r_canfd_cfg.h:

Configuration	Options	Default	Description
Global Error Interrupt > Callback Channel	MCU Specific Options		Specify which channel callback should be called to handle global errors. When starting the driver this channel must be opened first.
Global Error Interrupt > Priority	MCU Specific Options		This interrupt is fired for each of the error sources selected below.
Reception > FIFOs > Interrupt Priority	MCU Specific Options		This priority level will apply to all FIFO interrupts globally.
Reception > Acceptance Filtering > Channel 0 or 2 Rule Count	The number of AFL rules must be a positive integer.	16	Number of acceptance filter list rules dedicated to Channel 0 or 2.
Reception > Acceptance Filtering > Channel 1 Rule Count	The number of AFL rules must be a positive integer.	16	Number of acceptance filter list rules dedicated to Channel 1.
Flexible Data (FD) > Protocol Exceptions	• Enabled (ISO 11898-1) • Disabled	Enabled (ISO 11898-1)	Select whether to enter the protocol exception handling state when a RES bit is sampled recessive as defined in ISO 11898-1.
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Connectivity > CAN FD (r_canfd)

This module can be added to the Stacks tab via New Stack > Connectivity > CAN FD (r_canfd).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_canfd0	Module name.
General > Channel	MCU Specific Options		Specify the CAN channel to use.
Bitrate > Automatic > Nominal Rate (bps)	Must be a valid integer with a maximum of 1MHz.	500000	Specify nominal bitrate in bits per second.

Bitrate > Automatic > FD Data Rate (bps)	Must be a valid integer	2000000	Specify data bitrate in bits per second. This value is not used when sending messages without the FD and BRS bits enabled and should be left at the default setting if only Classical CAN will be used.
Bitrate > Automatic > Sample Point (%)	Must be a valid integer between 60 and 99.	75	Specify desired sample point.
Bitrate > Manual > Nominal > Prescaler (divisor)	Value must be a non-negative integer between 1 and 1024.	1	Specify clock divisor for nominal bitrate.
Bitrate > Manual > Nominal > Time Segment 1 (Tq)	Value must be a non-negative integer between 2 and 256.	29	Select the Time Segment 1 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Nominal > Time Segment 2 (Tq)	Value must be a non-negative integer between 2 and 128.	10	Select the Time Segment 2 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Nominal > Sync Jump Width (Tq)	Value must be a non-negative integer between 1 and 128.	4	Select the Synchronization Jump Width value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Prescaler (divisor)	Value must be a non-negative integer between 1 and 256.	1	Specify clock divisor for data bitrate.
Bitrate > Manual > Data > Time Segment 1 (Tq)	Value must be a non-negative integer between 2 and 32.	5	Select the Time Segment 1 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Time Segment 2 (Tq)	Value must be a non-negative integer between 2 and 16.	2	Select the Time Segment 2 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Sync Jump Width (Tq)	Value must be a non-negative integer between 1 and 16.	1	Select the Synchronization Jump Width value. Check module usage notes for how to calculate this

			value.
Bitrate > Manual > Use manual settings	• Yes • No	No	Select whether or not to override automatic baudrate generation and instead use the values specified here.
Bitrate > Delay Compensation	• Enable • Disable	Enable	When enabled the CANFD module will automatically compensate for any transceiver or bus delay between transmitted and received bits. When manually supplying bit timing values with delay compensation enabled be sure the data prescaler is 2 or smaller for correct operation.
Interrupts > Callback	Name must be a valid C symbol	canfd0_callback	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs.
Interrupts > Channel Interrupt Priority Level	MCU Specific Options		Channel Error/Transmit interrupt priority.
Global Error Interrupt > Sources	• DLC Check • Message Lost • FD Payload Overflow	0U	Select which errors should trigger an interrupt.
Flexible Data (FD) > Payload Overflow	• Reject • Truncate	Reject	Configure whether received messages larger than the destination buffer should be truncated or rejected.
Reception > Message Buffers > Number of Buffers	RX Message Buffer number must be an integer between 0 and 32.	0	Number of message buffers available for reception. As there is no interrupt for message buffer reception it is recommended to use RX FIFOs instead. Set this value to 0 to disable RX Message

			Buffers.
Reception > Message Buffers > Payload Size	• 8 bytes • 12 bytes • 16 bytes • 20 bytes • 24 bytes • 32 bytes • 48 bytes • 64 bytes	8 bytes	Payload size for all RX Message Buffers.
Reception > FIFOs > FIFO 0 > Enable	• Enabled • Disabled	Enabled	Enable or disable RX FIFO 0.
Reception > FIFOs > FIFO 0 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 0 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 0 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 0.
Reception > FIFOs > FIFO 0 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 0.
Reception > FIFOs > FIFO 1 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 1.
Reception > FIFOs > FIFO 1 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 1. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 1 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 1. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs >	MCU Specific Options		Select the message

FIFO 1 > Payload Size			payload size for RX FIFO 1.
Reception > FIFOs > FIFO 1 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 1.
Common FIFOs > FIFO 0 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 0.
Common FIFOs > FIFO 0 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 0.
Common FIFOs > FIFO 0 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 0.
Common FIFOs > FIFO 0 > TX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message transmission for Common FIFO 0.
Common FIFOs > FIFO 0 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 0 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 0 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 0.
Common FIFOs > FIFO 0 > Depth	MCU Specific Options		Select the number of stages for Common FIFO 0.
Common FIFOs > FIFO 0 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 0.
Common FIFOs > FIFO 0 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for Common FIFO 0 in

			units of the Interval Timer Clock.
Common FIFOs > FIFO 0 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 0. The reference clock can be scaled by x10 below.
Common FIFOs > FIFO 0 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 0.
Transmit Interrupts	MCU Specific Options		Select which TX Message Buffers should trigger an interrupt when transmission is complete.
Channel Error Interrupts	• Error Warning • Error Passive • Bus-Off Entry • Bus-Off Recovery • Overload	0U	Select which channel error interrupt sources to enable.
Filter List Array	Name must be a valid C symbol	p_canfd0_afl	Acceptance Filter List (AFL) rule array symbol name.
Transmission Priority	• Message ID • Buffer Number	Buffer Number	Select how messages should be prioritized for transmission. In either case, lower numbers indicate higher priority.
DLC Check	• Disabled • Enabled • Enabled w/truncate	module.driver.canfd.dlc_check.disabled	When enabled received messages will be rejected if their DLC field is less than the value configured in the associated AFL rule. If 'Enabled w/truncate' is selected and a message passes the DLC check the DLC field is set to the value in the associated AFL rule and any excess data is discarded.
Interval Timer Prescaler	Timer prescaler must be an integer between 0 and 65535.	0	FIFO interval timer prescaler, required for Interval TX Delay.

Clock Configuration

The CANFD peripheral uses PLL as its clock source. The RX Configuration editor will attempt to get as close as possible to the supplied bitrate with the configured clock source. To achieve an exact bitrate the CANFD source clock or divisor may need to be adjusted to meet the criteria in the formula below:

```
bitrate = canfd_clock_hz / ((time_segment_1 + time_segment_2 + 1) * prescaler)
```

For CANFD, the possible values for each element are as follows:

Element	Min	Max (Nominal)	Max (Data)
Bitrate	-	1 Mbps	8 Mbps
Time Segment 1	2 Tq	256 Tq	32 Tq
Time Segment 2	2 Tq	128 Tq	16 Tq
Sync Jump Width	1 Tq	Time Segment 2	Time Segment 2
Prescaler	1	1024	256

Use the **Clocks** tab of the RX Configuration editor to configure the CANFD clock source/divisor as well as to set the frequency of PLL. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

Pin Configuration

CANFD channels each control two pins: CRX (receive) and CTX (transmit).

Usage Notes

Buffers

The CANFD driver provides four types of buffers: Transmit Message Buffers (TX MBs), Receive Message Buffers (RX MBs), Receive FIFOs (RX FIFOs), and Common FIFOs (RX/TX Common FIFOs).

TX Message Buffers

TX MBs are used for transmission only. Refer to the hardware manual for your device for information on which TX MBs are available.

Note

The CANFD peripheral continually scans TX MBs for new data. Depending on the provided clock it may be possible to write to multiple TX MBs before transmission begins. In this case, messages will be sent in the priority specified by the Transmission Priority option in the RX Configuration editor.

RX Message Buffers

RX MBs are for reception only and may only hold one message at a time. The number of available RX MBs varies per device.

RX FIFOs

RX FIFOs provide interrupt-driven queue functionality for receiving messages. All FIFOs have the following capabilities:

- Up to 64 byte payloads
- Up to 48 messages for CANFD.
- Interrupt events:
    - On every received frame OR when filled to a specified fraction of its capacity
    - When a message is overwritten (message received on full FIFO)

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFD_InfoGet and R_CANFD_Read, respectively.

RX Buffer Pool

The CANFD peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. The RX Configuration editor will provide a warning when the limit is exceeded.

The number of bytes used by RX MBs and individual FIFOs can be calculated as follows:

```
Total RX MB bytes used = (number of RX MBs enabled) * (RX MB payload size + 12 header bytes)
RX FIFO bytes used = (number of FIFO stages) * (FIFO payload size + 12 header bytes)
```

Common FIFOs

Common FIFOs provide interrupt-driven queue functionality for receiving or transmitting messages. Unlike RX FIFOs, Common FIFOs are individual to each channel. Refer to the hardware manual for your device for information on how many Common FIFOs are available. Common FIFOs support the following capabilities:

- Either TX or RX behavior can be configured.
    - Each Common FIFO can only be configured as one or the other. It cannot operate in both TX and RX modes at the same time.
- Up to 64 byte payloads
- Up to 48 messages on CANFD
- Interrupt events:
    - On every message received or when filled to a specified fraction of its capacity
    - On every message transmitted or when all messages have been transmitted and the FIFO is empty

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFD_InfoGet and R_CANFD_Read, respectively.

Users should be aware of the total memory allocated in the internal CAN-FD RAM area for Common FIFOs. Allocating too many FIFOs or entries can lead to unexpected behavior.

Message Filtering (Acceptance Filter List)

To filter messages to the desired message buffer or FIFO the CANFD peripheral uses an Acceptance Filter List (AFL). Each entry in the AFL provides a rule to check a message against along with destination and other filtering information. When a message is received the CANFD peripheral internally checks against every configured AFL rule for the channel. If a match is found the message is transferred to the destination(s) specified in the rule. The default template with one entry is shown below:

```
static const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH0_RULE_NUM] =
{
    {
 /* Specify the ID, ID type and frame type to accept. */
            .id_u.id =
      {
             .id         = 0x00000000,
             .frame_type = CAN_FRAME_TYPE_DATA,
             .id_mode    = CAN_ID_MODE_EXTENDED,
            },
 /* These values mask which ID/mode bits to compare when filtering messages. */
            .mask_u.mask =
      {
             .mask_id         = 0x1FFFFFFF,
             .mask_frame_type = 1,
             .mask_id_mode    = 1,
            },
            .destination_u.destination =
      {
 /* If DLC checking is enabled any messages shorter than the below setting will be
rejected. */
            .minimum_dlc = CANFD_MINIMUM_DLC_0,
 /* Optionally specify a Receive Message Buffer (RX MB) to store accepted frames. RX
MBs do not have an
  * interrupt or overwrite protection and must be checked with R_CANFD_InfoGet and
R_CANFD_Read. */
            .rx_buffer   = CANFD_RX_MB_NONE,
 /* Specify which FIFO(s) to send filtered messages to. Multiple FIFOs can be OR'd
together. */
```

```
            .fifo_select_flags = CANFD_RX_FIFO_0,
            }
    }
};
```

AFL templates can be easily added to a project using the Developer Assistance feature in e² studio. Once the CANFD module is added to a project, drag and drop the elements circled below to build a filter list:

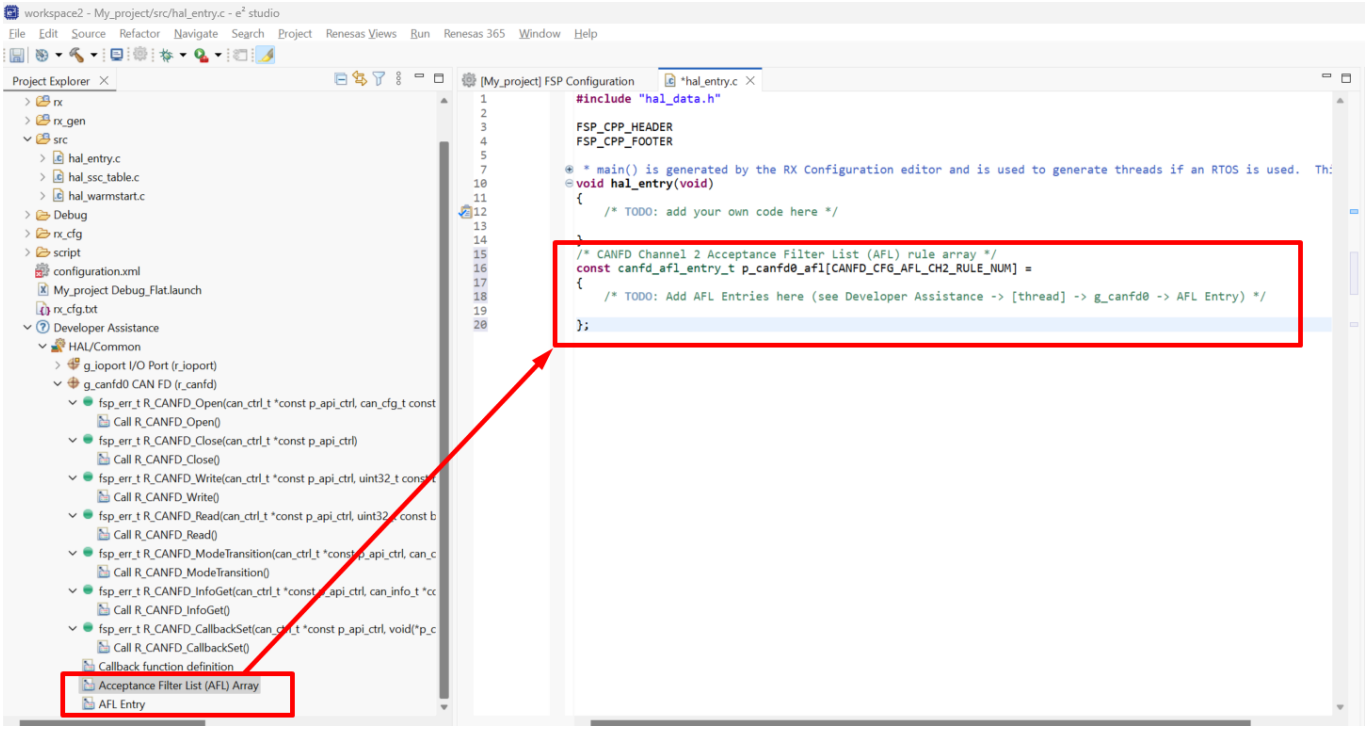


Figure 99: CANFD Developer Assistance AFL Templates

For an example configuration refer to the AFL Example below.

Flexible Data (FD)

Flexible Data is an extension of the CAN protocol allowing for messages up to 64 bytes and higher data bitrates, among other features. The CANFD driver supports the following:

- Sending and receiving FD messages
- Bitrate switching for data phase
- Manual and automatic setting of the error state (ESI) bit

To specify one or more of these options when transmitting set can_frame_t::options with combined values from canfd_frame_options_t. Received messages will automatically have this field filled, if applicable.

```
/* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
   g_can_tx_frame.id               = CAN_EXAMPLE_ID;
   g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
   g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
   g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
   g_can_tx_frame.options          = CANFD_FRAME_OPTION_FD | CANFD_FRAME_OPTION_BRS;
```

Note

When using bitrate switching be sure to configure the Data Bitrate as desired in the RX Configuration editor.

Bit Rate Calculation

For convenience, the baudrate of the CANFD peripheral is automatically set through the RX Configuration editor using a best effort approach.

Enabling Delay Compensation instructs the CANFD peripheral to measure TX to RX tranceiver delay and automatically adjust for it, improving the reliability of high-speed FD messages. This option may severely limit available bitrate settings depending on the source clock; it is highly recommended to check the generated values when enabled.

If the auto-generated baud settings cause deviation that is not tolerable by the application the user can override the auto-generated settings and put in manually calculated values through the RX Configuration editor. For more details on how the bitrate is calculated refer to the Clock Configuration section above.

Sync Jump Width

The Sync Jump Width option specifies the maximum number of time quanta that the sample point may be delayed by to account for differences in oscillators on the bus. It should be set to a value between 1 and the configured Time Segment 2 value depending on the maximum permissible clock error.

Error Handling

The CANFD peripheral provides two types of error interrupts: Channel and Global. As the names imply, each channel has its own Channel Error interrupt but there is only one Global Error interrupt. Only the configured channel will receive callbacks for Global Errors.

Error interrupt callbacks will pass either CAN_EVENT_ERR_CHANNEL or CAN_EVENT_ERR_GLOBAL in the can_callback_args_t::event field. A second field, can_callback_args_t::error, provides the actual error code as canfd_error_t. Cast to this enum to retrieve the error condition. See the callback in the Basic Example below for a demonstration.

DLC Checking

When DLC Checking is enabled messages are checked against the destination.minimum_dlc value of each AFL rule. If the data length of a message is less than this value the message will be rejected. When DLC checking is set to "Enabled w/truncate" in the RX Configuration editor any data in excess of the minimum DLC setting will be truncated and the DLC value for the frame will be set to match.

FD Payload Overflow

When an FD message is received with a DLC larger than the destination buffer an FD Payload Overflow interrupt is thrown (if configured). When Payload Overflow is set to "Truncate" the message will still be accepted but only data up to the buffer capacity will be preserved. The DLC value is unchanged in this case; any data beyond this value in the can_frame_t::data array should not be used.

Test Modes

The CANFD peripheral provides three basic test modes: Listen Only, Internal Loopback and External Loopback. Use R_CANFD_ModeTransition to switch to a test mode.

Existing Low Power Consumption modes

For RX74M/RX74N, the message buffer RAM state is undefined after exiting Software Standby Mode 1.

Since the CANFD module is not reset during Software Standby Mode 1, the message buffer RAM is not automatically reinitialized after wake-up. Therefore, we must manually initialize the message buffer RAM before resuming CANFD operation.

Follow the software flow shown below to enter **RAM Test Mode** and clear all message buffer RAM locations by writing 0 to each location.

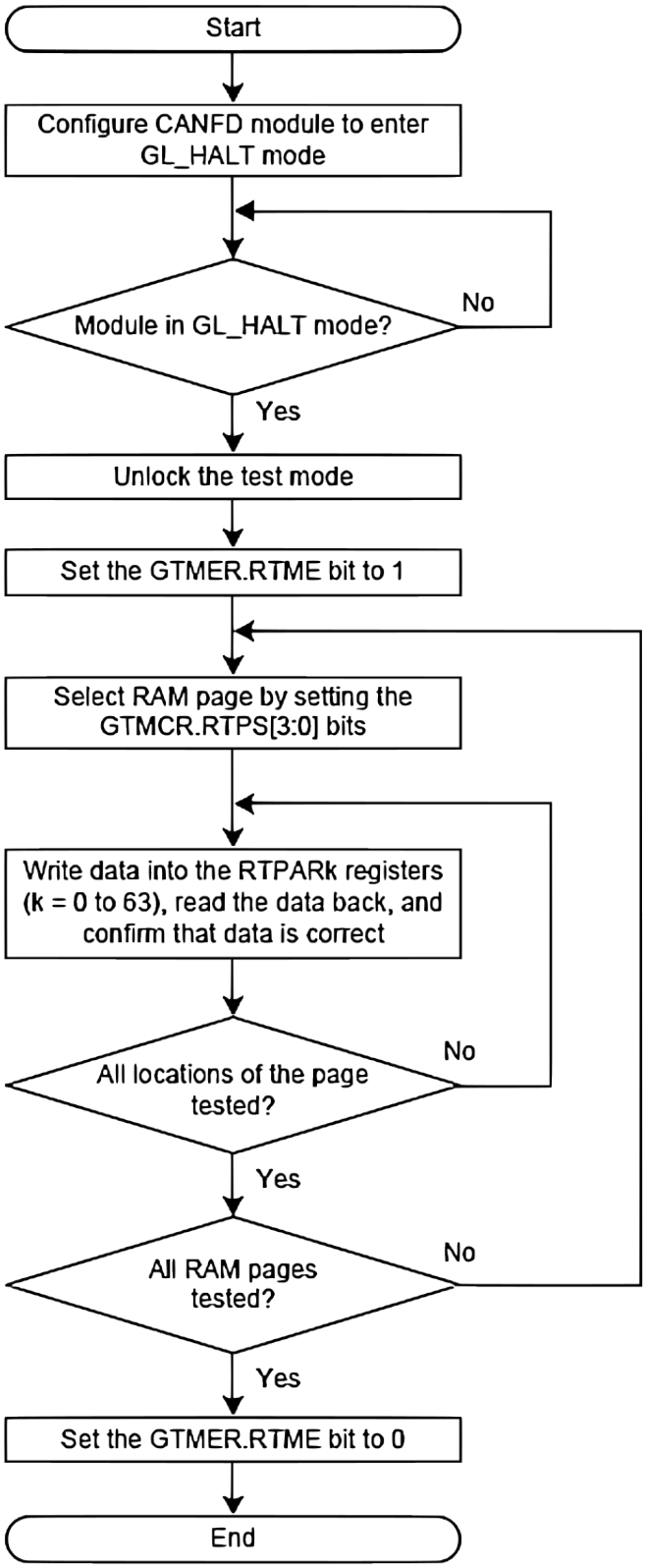


Figure 100: RAM Test Mode Software Flow

After completing the RAM initialization, perform the **CANFD Module Configuration after a Reset**

procedure.

Limitations

Developers should be aware of the following limitations when using CANFD:

- The CANFD peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. See the RX Buffer Pool section above for more information.
- When switching modes with R_CANFD_ModeTransition a delay of up to several CAN frames may be incurred. Consult "Timing of Channel Mode Change" in the CANFD section of the relevant hardware manual for details.
- Only one channel will receive callbacks for Global Errors. If a different channel is opened first these error interrupts will be suppressed until the specified handler channel is opened.

Examples

AFL Example

The below is an example Acceptance Filter List (AFL) declaration with two rules.

```
const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH2_RULE_NUM] =
{
 /* Store all data frames with at least 4 bytes from Standard IDs 0x40-0x4F in RX
FIFO 0 and RX FIFO 1 */
    {
       .id_u.id =
      {
            .id         = 0x40,
            .frame_type = CAN_FRAME_TYPE_DATA,
            .id_mode    = CAN_ID_MODE_STANDARD
      },
       .mask_u.mask =
      {
            .mask_id         = 0x7F0,
            .mask_frame_type = 1,
            .mask_id_mode    = 1,
      },
       .destination_u.destination =
      {
            .minimum_dlc       = CANFD_MINIMUM_DLC_4,
            .rx_buffer         = CANFD_RX_MB_NONE,
```

```
            .fifo_select_flags = (canfd_rx_fifo_t) (CANFD_RX_FIFO_0 |
CANFD_RX_FIFO_1),
      }
    },
 /* Store all frames from Extended ID 0x1100 in RX FIFO 1 and RX MB 0 */
    {
       .id_u.id =
      {
            .id         = 0x1100,
            .frame_type = CAN_FRAME_TYPE_DATA, // This setting is ignored by the mask
            .id_mode    = CAN_ID_MODE_EXTENDED
      },
       .mask_u.mask =
      {
            .mask_id         = 0x1FFFFFFF,
            .mask_frame_type = 0,
            .mask_id_mode    = 1,
      },
       .destination_u.destination =
      {
            .minimum_dlc       = CANFD_MINIMUM_DLC_0,
            .rx_buffer         = CANFD_RX_MB_0,
            .fifo_select_flags = CANFD_RX_FIFO_1,
      }
    }
};
```

Basic Example

This is a basic example of minimal use of the CANFD module in an application.

```
#define CAN_EXAMPLE_ID (0x20)
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile canfd_error_t g_err_status = (canfd_error_t) 0;
```

```
void canfd_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
      {
 /* Read received frame */
            memcpy(&g_can_rx_frame, &p_args->frame, sizeof(can_frame_t));
 /* Handle event */
 break;
      }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
      {
 /* Handle event */
 break;
      }
 case CAN_EVENT_ERR_GLOBAL:                        /* Global error. */
 case CAN_EVENT_ERR_CHANNEL:                      /* Channel error. */
      {
 /* Get error status */
            g_err_status = (canfd_error_t) p_args->error; /* Check error code with
canfd_error_t. */
 /* Handle event */
 break;
      }
 default:
      {
 break;
      }
    }
}
void canfd_basic_example (void)
{
 fsp_err_t err;
```

```
/* Initialize the CAN module */
   err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Setup frame to write to CAN ID 0x20 */
   g_can_tx_frame.id               = CAN_EXAMPLE_ID;
   g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
   g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
   g_can_tx_frame.data_length_code = 8;
   g_can_tx_frame.options          = 0;
/* Write some data to the transmit frame */
for (uint32_t i = 0; i < 8; i++)
   {
      g_can_tx_frame.data[i] = (uint8_t) i;
   }
/* Send data on the bus */
   err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
   assert(FSP_SUCCESS == err);
/* Wait for a transmit callback event */
}
```

Flexible Data

This example demonstrates sending an FD message with bitrate switching over external loopback. The CTX and CRX pins must be connected when using external loopback, though if a CAN tranciever is onboard a 120 Ohm resistor should be connected across CANH and CANL instead.

Data Structures

struct	canfd_afl_entry_t
struct	canfd_global_cfg_t
struct	canfd_extended_cfg_t

Enumerations

enum	canfd_status_t
enum	canfd_error_t

enum	canfd_tx_buffer_t
enum	canfd_tx_mb_t
enum	canfd_rx_buffer_t
enum	canfd_rx_mb_t
enum	canfd_rx_fifo_t
enum	canfd_minimum_dlc_t
enum	canfd_frame_options_t

Data Structure Documentation

◆ canfd_afl_entry_t

struct canfd_afl_entry_t
AFL Entry (based on R_CANFD_AFL_Type in r_canfd_reg.h)

◆ canfd_global_cfg_t

struct canfd_global_cfg_t		
CANFD Global Configuration		
Data Fields		
uint32_t	global_interrupts	Global control options (GCR register setting)
uint32_t	global_config	Global configuration options (GCFG register setting)
uint32_t	rx_fifo_config[2]	RX FIFO configuration (RFCRn register settings)
uint32_t	rx_mb_config	Number and size of RX Message Buffers (RMCR register setting)
uint8_t	global_err_ipl	Global Error interrupt priority.
uint8_t	rx_fifo_ipl	RX FIFO interrupt priority.
uint32_t	common_fifo_config[1]	Common FIFO configurations.

◆ canfd_extended_cfg_t

struct canfd_extended_cfg_t		
CANFD Extended Configuration		
Data Fields		
canfd_afl_entry_t const *	p_afl	AFL rules list.

uint64_t	txmb_txi_enable	Array of TX Message Buffer enable bits.
uint32_t	error_interrupts	Error interrupt enable bits.
can_bit_timing_cfg_t *	p_data_timing	FD Data Rate (when bitrate switching is used)
uint8_t	delay_compensation	FD Transceiver Delay Compensation (enable or disable)
canfd_global_cfg_t *	p_global_cfg	Global configuration (global error callback channel only)

Enumeration Type Documentation

◆ canfd_status_t

enum canfd_status_t	
CANFD Status	
Enumerator	
CANFD_STATUS_RESET_MODE	Channel in Reset mode.
CANFD_STATUS_HALT_MODE	Channel in Halt mode.
CANFD_STATUS_SLEEP_MODE	Channel in Sleep mode.
CANFD_STATUS_ERROR_PASSIVE	Channel in error-passive state.
CANFD_STATUS_BUS_OFF	Channel in bus-off state.
CANFD_STATUS_TRANSMITTING	Channel is transmitting.
CANFD_STATUS_RECEIVING	Channel is receiving.
CANFD_STATUS_READY	Channel is ready for communication.
CANFD_STATUS_ESI	At least one CAN-FD message was received with the ESI flag set.

◆ canfd_error_t

enum canfd_error_t	
CANFD Error Code	
Enumerator	
CANFD_ERROR_CHANNEL_BUS	Bus Error.
CANFD_ERROR_CHANNEL_WARNING	Error Warning (TX/RX error count over 0x5F)
CANFD_ERROR_CHANNEL_PASSIVE	Error Passive (TX/RX error count over 0x7F)
CANFD_ERROR_CHANNEL_BUS_OFF_ENTRY	Bus-Off State Entry.
CANFD_ERROR_CHANNEL_BUS_OFF_RECOVERY	Recovery from Bus-Off State.
CANFD_ERROR_CHANNEL_OVERLOAD	Overload.
CANFD_ERROR_CHANNEL_BUS_LOCK	Bus Locked.
CANFD_ERROR_CHANNEL_ARBITRATION_LOSS	Arbitration Lost.
CANFD_ERROR_CHANNEL_STUFF	Stuff Error.
CANFD_ERROR_CHANNEL_FORM	Form Error.
CANFD_ERROR_CHANNEL_ACK	ACK Error.
CANFD_ERROR_CHANNEL_CRC	CRC Error.
CANFD_ERROR_CHANNEL_BIT_RECESSIVE	Bit Error (recessive) Error.
CANFD_ERROR_CHANNEL_BIT_DOMINANT	Bit Error (dominant) Error.
CANFD_ERROR_CHANNEL_ACK_DELIMITER	ACK Delimiter Error.
CANFD_ERROR_GLOBAL_DLC	DLC Error.
CANFD_ERROR_GLOBAL_MESSAGE_LOST	Message Lost.
CANFD_ERROR_GLOBAL_PAYLOAD_OVERFLOW	FD Payload Overflow.
CANFD_ERROR_GLOBAL_CH02_SCAN_FAIL	Channel 0 or 2 RX Scan Failure.
CANFD_ERROR_GLOBAL_CH02_ECC	Channel 0 or 2 ECC Error.

◆ canfd_tx_buffer_t

enum canfd_tx_buffer_t
CANFD Transmit Buffer (MB + CFIFO)

◆ canfd_tx_mb_t

enum canfd_tx_mb_t
CANFD Transmit Message Buffer (TX MB)

◆ canfd_rx_buffer_t

enum canfd_rx_buffer_t
CANFD Receive Buffer (MB + FIFO + CFIFO)

◆ canfd_rx_mb_t

enum canfd_rx_mb_t
CANFD Receive Message Buffer (RX MB)

◆ canfd_rx_fifo_t

enum canfd_rx_fifo_t
CANFD Receive FIFO (RX FIFO)

◆ canfd_minimum_dlc_t

enum canfd_minimum_dlc_t
CANFD AFL Minimum DLC settings

◆ canfd_frame_options_t

enum canfd_frame_options_t	
CANFD Frame Options	
Enumerator	
CANFD_FRAME_OPTION_ERROR	Error state set (ESI).
CANFD_FRAME_OPTION_BRS	Bit Rate Switching (BRS) enabled.
CANFD_FRAME_OPTION_FD	Flexible Data frame (FDF).

Function Documentation

◆ R_CANFD_Open()

fsp_err_t R_CANFD_Open (can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg*)

Open and configure the CANFD channel for operation.

Example:

```
/* Initialize the CAN module */
   err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
```

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ALREADY_OPEN	Driver already open.
FSP_ERR_IN_USE	Channel is already in use.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel does not exist on this MCU.
FSP_ERR_ASSERTION	A required pointer was NULL.
FSP_ERR_CAN_INIT_FAILED	The provided nominal or data bitrate is invalid.
FSP_ERR_CLOCK_INACTIVE	CANFD source clock is disabled (PLL or PLL2).

◆ R_CANFD_Close()

fsp_err_t R_CANFD_Close (can_ctrl_t *const *p_api_ctrl*)

Close the CANFD channel.

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented.

◆ R_CANFD_Write()

fsp_err_t R_CANFD_Write (can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame*)

Write data to the CANFD channel.

Example:

```
/* Send data on the bus */
   err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
FSP_ERR_INVALID_ARGUMENT	Data length or buffer number invalid.
FSP_ERR_INVALID_MODE	An FD option was set on a non-FD frame.
FSP_ERR_ASSERTION	One or more pointer arguments is NULL.
FSP_ERR_UNSUPPORTED	FD is not supported on this MCU.

◆ R_CANFD_Read()

fsp_err_t R_CANFD_Read (can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame*)

Read data from a CANFD Message Buffer or FIFO.

Example: snippet r_canfd_example.c R_CANFD_Read

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_ARGUMENT	Buffer number invalid.
FSP_ERR_ASSERTION	p_api_ctrl or p_frame is NULL.
FSP_ERR_BUFFER_EMPTY	Buffer or FIFO is empty.

◆ R_CANFD_ModeTransition()

fsp_err_t R_CANFD_ModeTransition (can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode*)

Switch to a different channel, global or test mode.

Example:

```
/* Switch to external loopback mode */
   err = R_CANFD_ModeTransition(&g_canfd0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented
FSP_ERR_INVALID_MODE	Cannot change to the requested mode from the current global mode.

◆ R_CANFD_InfoGet()

fsp_err_t R_CANFD_InfoGet (can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info*)

Get CANFD state and status information for the channel.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CANFD_CallbackSet()

fsp_err_t R_CANFD_CallbackSet (can_ctrl_t *const *p_api_ctrl*, void(*)(can_callback_args_t *) *p_callback*, void *const *p_context*, can_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements can_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.6.2 CAN FDG (r_canfdg)

Modules » Connectivity

Functions

fsp_err_t	R_CANFDG_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg)
fsp_err_t	R_CANFDG_Close (can_ctrl_t *const p_api_ctrl)
fsp_err_t	R_CANFDG_Write (can_ctrl_t *const p_api_ctrl, uint32_t const buffer, can_frame_t *const p_frame)
fsp_err_t	R_CANFDG_Read (can_ctrl_t *const p_api_ctrl, uint32_t const buffer, can_frame_t *const p_frame)
fsp_err_t	R_CANFDG_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)

fsp_err_t	R_CANFDG_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info)
fsp_err_t	R_CANFDG_CallbackSet (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CANFDG peripheral on RX MCUs. This module implements the CAN Interface.

Overview

The CANFDG module can be used to communicate over CAN networks, optionally using Flexible Data (CAN-FD) to accelerate the data phase. A variety of message filtering and buffer options are available. This document covers the CANFDG (r_canfdg) module as they are based on the same hardware.

Features

Common Features

- Compatibility
  - Send and receive CAN 2.0 and CAN-FD frames on the same channel
  - Bitrate up to 1 Mbps with FD data phase speeds up to 8 Mbps
  - ISO 11898-1 Support
- Buffers
  - 32 global receive Message Buffers (RX MBs)
  - 8 global receive FIFOs (RX FIFOs)
  - 16 transmit Message Buffers (TX MBs) per channel
- Filtering
  - Each filter rule can be individually configured to accept messages based on:
    - ID
    - Standard or Extended ID (IDE bit)
    - Data or Remote Frame (RTR bit)
    - ID/IDE/RTR mask
    - Minimum DLC (data length) value
- Interrupts
  - Configurable Global RX FIFO Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message
  - Configurable Common FIFO RX Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message
  - Channel TX Interrupt
    - Interrupt on every transmitted message or when a Common FIFO is empty
  - Global Error
    - DLC Check
    - Message Lost
    - FD Payload Overflow
  - Channel Error

- Bus Error
- Error Warning
- Error Passive
- Bus-Off Entry
- Bus-Off Recovery
- Overload
- Bus Lock
- Arbitration Loss
- Transmission Aborted

Per-MCU Specifications

	RX74M	RX74N
Channels	2	2
Max nominal bitrate	1 Mbps	1 Mbps
Max data bitrate	8 Mbps	8 Mbps
Filter rules	128	128
TX message buffers	16/ch	16/ch
RX message buffers	32	32
RX FIFOs	8	8
RX Buffer RAM	4864 bytes	4864 bytes
Common FIFOs	3/ch	3/ch

Note

Each message buffer comprises 12 header bytes plus data length (8-64 bytes). The above buffer RAM values therefore correspond to the following capacities:

	RX74M	*RX74N*
Max 64-byte storage	*64 messages*	*64 messages*
Max 8-byte storage	*243 messages*	*243 messages*

Each Common FIFO buffer can support the following message capacities:

	RX74M	*RX74N*
Maximum payload size	*64 bytes*	*64 bytes*
Maximum FIFO depth	*128 messages*	*128 messages*

Configuration

Build Time Configurations for r_canfdg

The following build time configurations are defined in fsp_cfg/r_canfdg_cfg.h:

Configuration	Options	Default	Description
Global Error Interrupt >	MCU Specific Options		Specify which channel

Callback Channel			callback should be called to handle global errors. When starting the driver this channel must be opened first.
Global Error Interrupt > Priority	MCU Specific Options		This interrupt is fired for each of the error sources selected below.
Global Error Interrupt > Sources	• DLC Check • Message Lost • FD Payload Overflow	0U	Select which errors should trigger an interrupt.
Reception > FIFOs > FIFO 0 > Enable	• Enabled • Disabled	Enabled	Enable or disable RX FIFO 0.
Reception > FIFOs > FIFO 0 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 0 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 0 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 0.
Reception > FIFOs > FIFO 0 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 0.
Reception > FIFOs > FIFO 1 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 1.
Reception > FIFOs > FIFO 1 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 1. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 1 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 1. This setting is only applicable when

			the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 1 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 1.
Reception > FIFOs > FIFO 1 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 1.
Reception > FIFOs > FIFO 2 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 2.
Reception > FIFOs > FIFO 2 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 2. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 2 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 2. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 2 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 2.
Reception > FIFOs > FIFO 2 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 2.
Reception > FIFOs > FIFO 3 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 3.
Reception > FIFOs > FIFO 3 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 3. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 3 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 3. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs >	MCU Specific Options		Select the message

FIFO 3 > Payload Size			payload size for RX FIFO 3.
Reception > FIFOs > FIFO 3 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 3.
Reception > FIFOs > FIFO 4 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 4.
Reception > FIFOs > FIFO 4 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 4. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 4 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 4. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 4 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 4.
Reception > FIFOs > FIFO 4 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 4.
Reception > FIFOs > FIFO 5 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 5.
Reception > FIFOs > FIFO 5 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 5. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 5 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 5. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 5 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 5.
Reception > FIFOs > FIFO 5 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 5.

Reception > FIFOs > FIFO 6 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 6.
Reception > FIFOs > FIFO 6 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 6. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 6 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 6. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 6 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 6.
Reception > FIFOs > FIFO 6 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 6.
Reception > FIFOs > FIFO 7 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 7.
Reception > FIFOs > FIFO 7 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 7. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 7 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 7. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 7 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 7.
Reception > FIFOs > FIFO 7 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 7.
Reception > FIFOs > Interrupt Priority	MCU Specific Options		This priority level will apply to all FIFO interrupts globally.
Reception > Message	RX Message Buffer	0	Number of message

Buffers > Number of Buffers	number must be an integer between 0 and 32.		buffers available for reception. As there is no interrupt for message buffer reception it is recommended to use RX FIFOs instead. Set this value to 0 to disable RX Message Buffers.
Reception > Message Buffers > Payload Size	• 8 bytes • 12 bytes • 16 bytes • 20 bytes • 24 bytes • 32 bytes • 48 bytes • 64 bytes	8 bytes	Payload size for all RX Message Buffers.
Reception > Acceptance Filtering > Channel 0 Rule Count	The number of AFL rules must be a positive integer.	32	Number of acceptance filter list rules dedicated to Channel 0.
Reception > Acceptance Filtering > Channel 1 Rule Count	The number of AFL rules must be a positive integer.	0	Number of acceptance filter list rules dedicated to Channel 1.
Flexible Data (FD) > Protocol Exceptions	• Enabled (ISO 11898-1) • Disabled	Enabled (ISO 11898-1)	Select whether to enter the protocol exception handling state when a RES bit is sampled recessive as defined in ISO 11898-1.
Flexible Data (FD) > Payload Overflow	• Reject • Truncate	Reject	Configure whether received messages larger than the destination buffer should be truncated or rejected.
Common FIFOs > FIFO 0 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 0.
Common FIFOs > FIFO 0 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 0.
Common FIFOs > FIFO 0 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 0.
Common FIFOs > FIFO	• Enabled	Disabled	Enable to allow

0 > TX Interrupt Enable	• Disabled		interrupts on message transmission for Common FIFO 0.
Common FIFOs > FIFO 0 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 0 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 0 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 0.
Common FIFOs > FIFO 0 > Depth	MCU Specific Options		Select the number of stages for Common FIFO 0.
Common FIFOs > FIFO 0 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 0.
Common FIFOs > FIFO 0 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for Common FIFO 0 in units of the Interval Timer Clock.
Common FIFOs > FIFO 0 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 0. The reference clock can be scaled by x10 below.
Common FIFOs > FIFO 0 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 0.
Common FIFOs > FIFO 1 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 1.

Common FIFOs > FIFO 1 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 1.
Common FIFOs > FIFO 1 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 1.
Common FIFOs > FIFO 1 > TX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message transmission for Common FIFO 1.
Common FIFOs > FIFO 1 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 1. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 1 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 1. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 1 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 1.
Common FIFOs > FIFO 1 > Depth	MCU Specific Options		Select the number of stages for Common FIFO 1.
Common FIFOs > FIFO 1 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 1.
Common FIFOs > FIFO 1 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for Common FIFO 1 in units of the Interval Timer Clock.
Common FIFOs > FIFO 1 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 1. The reference clock can be scaled by x10 below.

Common FIFOs > FIFO 1 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 1.
Common FIFOs > FIFO 2 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 2.
Common FIFOs > FIFO 2 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 2.
Common FIFOs > FIFO 2 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 2.
Common FIFOs > FIFO 2 > TX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message transmission for Common FIFO 2.
Common FIFOs > FIFO 2 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 2. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 2 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 2. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 2 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 2.
Common FIFOs > FIFO 2 > Depth	MCU Specific Options		Select the number of stages for Common FIFO 2.
Common FIFOs > FIFO 2 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 2.
Common FIFOs > FIFO 2 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for

			Common FIFO 2 in units of the Interval Timer Clock.
Common FIFOs > FIFO 2 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 2. The reference clock can be scaled by x10 below.
Common FIFOs > FIFO 2 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 2.
Common FIFOs > FIFO 3 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 3.
Common FIFOs > FIFO 3 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 3.
Common FIFOs > FIFO 3 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 3.
Common FIFOs > FIFO 3 > TX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message transmission for Common FIFO 3.
Common FIFOs > FIFO 3 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 3. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 3 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 3. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 3 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 3.
Common FIFOs > FIFO 3 > Depth	MCU Specific Options		Select the number of stages for Common

			FIFO 3.
Common FIFOs > FIFO 3 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 3.
Common FIFOs > FIFO 3 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for Common FIFO 3 in units of the Interval Timer Clock.
Common FIFOs > FIFO 3 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 3. The reference clock can be scaled by x10 below.
Common FIFOs > FIFO 3 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 3.
Common FIFOs > FIFO 4 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 4.
Common FIFOs > FIFO 4 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 4.
Common FIFOs > FIFO 4 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 4.
Common FIFOs > FIFO 4 > TX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message transmission for Common FIFO 4.
Common FIFOs > FIFO 4 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 4. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 4 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 4. This setting is only

			applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 4 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 4.
Common FIFOs > FIFO 4 > Depth	MCU Specific Options		Select the number of stages for Common FIFO 4.
Common FIFOs > FIFO 4 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 4.
Common FIFOs > FIFO 4 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for Common FIFO 4 in units of the Interval Timer Clock.
Common FIFOs > FIFO 4 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 4. The reference clock can be scaled by x10 below.
Common FIFOs > FIFO 4 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 4.
Common FIFOs > FIFO 5 > Enable	• Enabled • Disabled	Disabled	Enable or disable Common FIFO 5.
Common FIFOs > FIFO 5 > Mode	• Receive • Transmit	Receive	Select the operation mode for Common FIFO 5.
Common FIFOs > FIFO 5 > RX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message receiption for Common FIFO 5.
Common FIFOs > FIFO 5 > TX Interrupt Enable	• Enabled • Disabled	Disabled	Enable to allow interrupts on message transmission for Common FIFO 5.
Common FIFOs > FIFO 5 > Interrupt Mode	• Threshold • Every frame	Threshold	Set the interrupt mode for Common FIFO 5. Threshold mode will only fire an interrupt each time an incoming

			message crosses the threshold value set below or when all messages have been transmitted.
Common FIFOs > FIFO 5 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for Common FIFO 5. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Common FIFOs > FIFO 5 > Payload Size	MCU Specific Options		Select the message payload size for Common FIFO 5.
Common FIFOs > FIFO 5 > Depth	MCU Specific Options		Select the number of stages for Common FIFO 5.
Common FIFOs > FIFO 5 > TX Message Buffer	MCU Specific Options		Select the TX message buffer that participates in TX scans related to Common FIFO 5.
Common FIFOs > FIFO 5 > Interval TX Delay	TX delay must be an integer between 0 and 255.	0	Selects the delay between successive transmission for Common FIFO 5 in units of the Interval Timer Clock.
Common FIFOs > FIFO 5 > Interval Timer	• Reference clock • Bit time clock	Reference clock	Select the source of the interval timer clock for Common FIFO 5. The reference clock can be scaled by x10 below.
Common FIFOs > FIFO 5 > Interval Timer Reference Clock Resolution	• x1 • x10	x1	Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 5.
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Transmission Priority	• Message ID • Buffer Number	Buffer Number	Select how messages should be prioritized for transmission. In either case, lower numbers indicate higher priority.
DLC Check	• Disabled	config.driver.canfdg.dlc	When enabled received

	• Enabled • Enabled w/truncate	_check.disabled	messages will be rejected if their DLC field is less than the value configured in the associated AFL rule. If 'Enabled w/truncate' is selected and a message passes the DLC check the DLC field is set to the value in the associated AFL rule and any excess data is discarded.
Interval Timer Prescaler	Timer prescaler must be an integer between 0 and 65535.	0	FIFO interval timer prescaler, required for Interval TX Delay.

Configurations for Connectivity > CAN FDG (r_canfdg)

This module can be added to the Stacks tab via New Stack > Connectivity > CAN FDG (r_canfdg).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_canfdg0	Module name.
General > Channel	Channel should be 0 or 1	0	Specify the CAN channel to use.
Bitrate > Automatic > Nominal Rate (bps)	Must be a valid integer with a maximum of 1MHz.	500000	Specify nominal bitrate in bits per second.
Bitrate > Automatic > FD Data Rate (bps)	Must be a valid integer	2000000	Specify data bitrate in bits per second. This value is not used when sending messages without the FD and BRS bits enabled and should be left at the default setting if only Classical CAN will be used.
Bitrate > Automatic > Sample Point (%)	Must be a valid integer between 60 and 99.	75	Specify desired sample point.
Bitrate > Manual > Nominal > Prescaler (divisor)	Value must be a non-negative integer between 1 and 1024.	1	Specify clock divisor for nominal bitrate.
Bitrate > Manual > Nominal > Time Segment 1 (Tq)	Value must be a non-negative integer between 2 and 256.	29	Select the Time Segment 1 value. Check module usage notes for how to calculate this value.

Bitrate > Manual > Nominal > Time Segment 2 (Tq)	Value must be a non-negative integer between 2 and 128.	10	Select the Time Segment 2 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Nominal > Sync Jump Width (Tq)	Value must be a non-negative integer between 1 and 128.	4	Select the Synchronization Jump Width value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Prescaler (divisor)	Value must be a non-negative integer between 1 and 256.	1	Specify clock divisor for data bitrate.
Bitrate > Manual > Data > Time Segment 1 (Tq)	Value must be a non-negative integer between 2 and 32.	5	Select the Time Segment 1 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Time Segment 2 (Tq)	Value must be a non-negative integer between 2 and 16.	2	Select the Time Segment 2 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Sync Jump Width (Tq)	Value must be a non-negative integer between 1 and 16.	1	Select the Synchronization Jump Width value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Use manual settings	• Yes • No	No	Select whether or not to override automatic baudrate generation and instead use the values specified here.
Bitrate > Delay Compensation	• Enable • Disable	Enable	When enabled the CANFD module will automatically compensate for any transceiver or bus delay between transmitted and received bits. When manually supplying bit timing values with delay compensation enabled be sure the data prescaler is 2 or smaller for correct operation.

Interrupts > Callback	Name must be a valid C symbol	canfdg0_callback	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs.
Interrupts > Channel Interrupt Priority Level	MCU Specific Options		Channel Error/Transmit interrupt priority.
Transmit Interrupts	MCU Specific Options		Select which TX Message Buffers should trigger an interrupt when transmission is complete.
Channel Error Interrupts	• Error Warning • Error Passive • Bus-Off Entry • Bus-Off Recovery • Overload	0U	Select which channel error interrupt sources to enable.
Filter List Array	Name must be a valid C symbol	p_canfd0_afl	Acceptance Filter List (AFL) rule array symbol name.

Clock Configuration

The CANFDG peripheral uses PLL or PLL2 as its clock source. The RX Configuration editor will attempt to get as close as possible to the supplied bitrate with the configured clock source. To achieve an exact bitrate the CANFDG source clock or divisor may need to be adjusted to meet the criteria in the formula below:

```
bitrate = canfdg_clock_hz / ((time_segment_1 + time_segment_2 + 1) * prescaler)
```

For CANFD, the possible values for each element are as follows:

Element	Min	Max (Nominal)	Max (Data)
Bitrate	-	1 Mbps	8 Mbps
Time Segment 1	2 Tq	256 Tq	32 Tq
Time Segment 2	2 Tq	128 Tq	16 Tq
Sync Jump Width	1 Tq	Time Segment 2	Time Segment 2
Prescaler	1	1024	256

Use the **Clocks** tab of the RX Configuration editor to configure the CANFD clock source/divisor as well as to set the frequency of PLL or PLL2. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

Pin Configuration

CANFDG channels each control two pins: CRX (receive) and CTX (transmit).

Usage Notes

Buffers

The CANFDG driver provides four types of buffers: Transmit Message Buffers (TX MBs), Receive Message Buffers (RX MBs), Receive FIFOs (RX FIFOs), and Common FIFOs (RX/TX Common FIFOs).

TX Message Buffers

TX MBs are used for transmission only. Refer to the hardware manual for your device for information on which TX MBs are available.

Note

The CANFDG peripheral continually scans TX MBs for new data. Depending on the provided clock it may be possible to write to multiple TX MBs before transmission begins. In this case, messages will be sent in the priority specified by the Transmission Priority option in the RX Configuration editor.

RX Message Buffers

RX MBs are for reception only and may only hold one message at a time. The number of available RX MBs varies per device.

On CANFDG, RX MBs are shared between channels and no interrupts are provided. Use R_CANFDG_InfoGet and R_CANFDG_Read to poll and read them, respectively.

RX FIFOs

RX FIFOs provide interrupt-driven queue functionality for receiving messages. All FIFOs have the following capabilities:

- Up to 64 byte payloads
- Up to 128 messages for CANFDG module.
- Interrupt events:
    - On every received frame OR when filled to a specified fraction of its capacity
    - When a message is overwritten (message received on full FIFO)

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFDG_InfoGet and R_CANFDG_Read, respectively.

Note

On the CANFDG module, FIFOs are shared across all channels.

RX Buffer Pool

The CANFDG peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. The RX Configuration editor will provide a warning when the limit is exceeded.

The number of bytes used by RX MBs and individual FIFOs can be calculated as follows:

```
Total RX MB bytes used = (number of RX MBs enabled) * (RX MB payload size + 12 header bytes)
RX FIFO bytes used = (number of FIFO stages) * (FIFO payload size + 12 header bytes)
```

Common FIFOs

Common FIFOs provide interrupt-driven queue functionality for receiving or transmitting messages. Unlike RX FIFOs, Common FIFOs are individual to each channel. Refer to the hardware manual for your device for information on how many Common FIFOs are available. Common FIFOs support the following capabilities:

- Either TX or RX behavior can be configured.
  - Each Common FIFO can only be configured as one or the other. It cannot operate in both TX and RX modes at the same time.
- Up to 64 byte payloads
- Up to 128 messages on CANFDG module.
- Interrupt events:
  - On every message received or when filled to a specified fraction of its capacity
  - On every message transmitted or when all messages have been transmitted and the FIFO is empty

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFDG_InfoGet and R_CANFDG_Read, respectively.

Users should be aware of the total memory allocated in the internal CAN-FD RAM area for Common FIFOs. Allocating too many FIFOs or entries can lead to unexpected behavior.

Message Filtering (Acceptance Filter List)

To filter messages to the desired message buffer or FIFO the CANFDG peripheral uses an Acceptance Filter List (AFL). Each entry in the AFL provides a rule to check a message against along with destination and other filtering information. When a message is received the CANFDG peripheral internally checks against every configured AFL rule for the channel. If a match is found the message is transferred to the destination(s) specified in the rule. The default template with one entry is shown below:

```
static const canfdg_afl_entry_t p_canfdg0_afl[CANFDG_CFG_AFL_CH0_RULE_NUM] =
{
    {
 /* Specify the ID, ID type and frame type to accept. */
      .id_u.id =
      {
       .id         = 0x00000000,
       .frame_type = CAN_FRAME_TYPE_DATA,
```

```
        .id_mode    = CAN_ID_MODE_EXTENDED,
      },
 /* These values mask which ID/mode bits to compare when filtering messages. */
      .mask_u.mask =
      {
        .mask_id         = 0x1FFFFFFF,
        .mask_frame_type = 1,
        .mask_id_mode    = 1,
      }
      .destination_u.destination =
      {
 /* If DLC checking is enabled any messages shorter than the below setting will be
rejected. */
        .minimum_dlc = CANFDG_MINIMUM_DLC_0,
 /* Optionally specify a Receive Message Buffer (RX MB) to store accepted frames. RX
MBs do not have an
  * interrupt or overwrite protection and must be checked with R_CANFDG_InfoGet and
R_CANFDG_Read. */
        .rx_buffer   = CANFDG_RX_MB_NONE,
 /* Specify which FIFO(s) to send filtered messages to. Multiple FIFOs can be OR'd
together. */
        .fifo_select_flags = CANFDG_RX_FIFO_0,
      }
    }
};
```

AFL templates can be easily added to a project using the Developer Assistance feature in e² studio. Once the CANFDG module is added to a project, drag and drop the elements circled below to build a filter list:

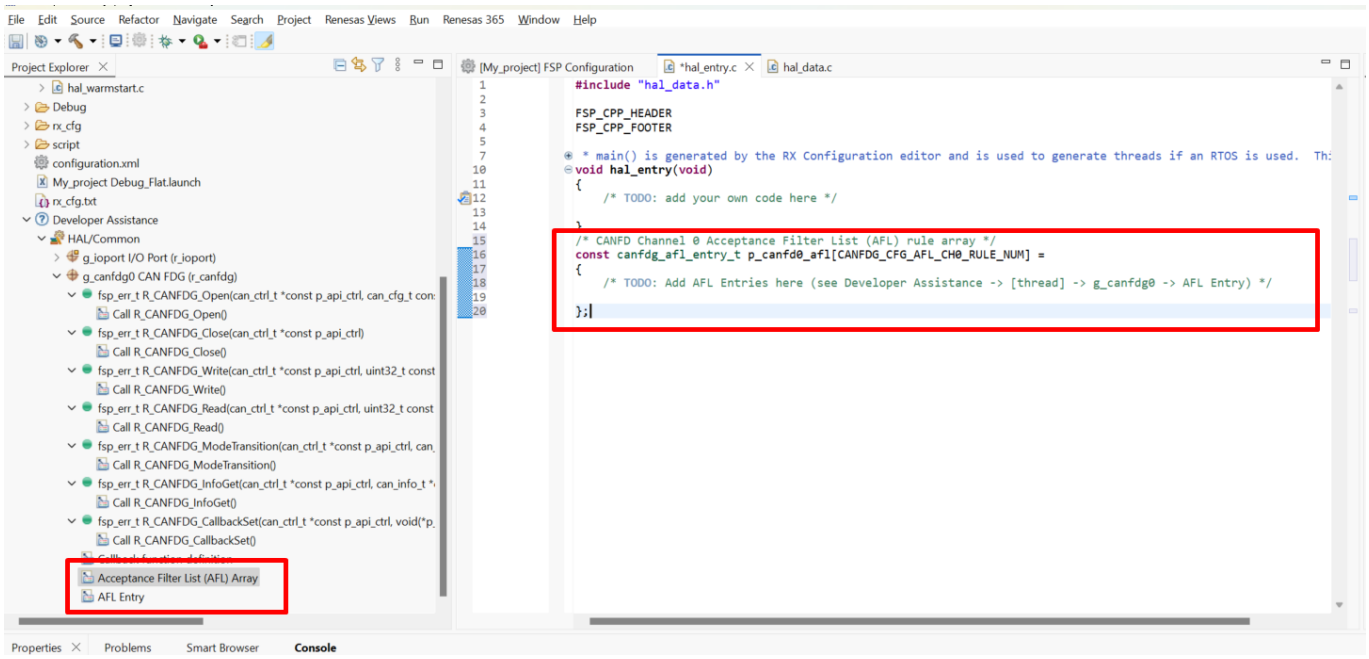


Figure 101: CANFDG Developer Assistance AFL Templates

For an example configuration refer to the AFL Example below.

Flexible Data (FD)

Flexible Data is an extension of the CAN protocol allowing for messages up to 64 bytes and higher data bitrates, among other features. The CANFDG driver supports the following:

- Sending and receiving FD messages
- Bitrate switching for data phase
- Manual and automatic setting of the error state (ESI) bit

To specify one or more of these options when transmitting set can_frame_t::options with combined values from canfdg_frame_options_t. Received messages will automatically have this field filled, if applicable.

```
/* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
   g_can_tx_frame.id               = CAN_EXAMPLE_ID;
   g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
   g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
   g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
   g_can_tx_frame.options          = CANFDG_FRAME_OPTION_FD |
CANFDG_FRAME_OPTION_BRS;
```

Note

When using bitrate switching be sure to configure the Data Bitrate as desired in the RX Configuration editor.

Bit Rate Calculation

For convenience, the baudrate of the CANFDG peripheral is automatically set through the RX Configuration editor using a best effort approach.

Enabling Delay Compensation instructs the CANFDG peripheral to measure TX to RX transceiver delay and automatically adjust for it, improving the reliability of high-speed FD messages. This option may severely limit available bitrate settings depending on the source clock; it is highly recommended to check the generated values when enabled.

If the auto-generated baud settings cause deviation that is not tolerable by the application the user can override the auto-generated settings and put in manually calculated values through the RX Configuration editor. For more details on how the bitrate is calculated refer to the Clock Configuration section above.

Sync Jump Width

The Sync Jump Width option specifies the maximum number of time quanta that the sample point may be delayed by to account for differences in oscillators on the bus. It should be set to a value between 1 and the configured Time Segment 2 value depending on the maximum permissible clock error.

Error Handling

The CANFDG peripheral provides two types of error interrupts: Channel and Global. As the names imply, each channel has its own Channel Error interrupt but there is only one Global Error interrupt. Only the configured channel will receive callbacks for Global Errors.

Error interrupt callbacks will pass either CAN_EVENT_ERR_CHANNEL or CAN_EVENT_ERR_GLOBAL in the can_callback_args_t::event field. A second field, can_callback_args_t::error, provides the actual error code as canfdg_error_t. Cast to this enum to retrieve the error condition. See the callback in the Basic Example below for a demonstration.

DLC Checking

When DLC Checking is enabled messages are checked against the destination.minimum_dlc value of each AFL rule. If the data length of a message is less than this value the message will be rejected. When DLC checking is set to "Enabled w/truncate" in the RX Configuration editor any data in excess of the minimum DLC setting will be truncated and the DLC value for the frame will be set to match.

FD Payload Overflow

When an FD message is received with a DLC larger than the destination buffer an FD Payload Overflow interrupt is thrown (if configured). When Payload Overflow is set to "Truncate" the message will still be accepted but only data up to the buffer capacity will be preserved. The DLC value is unchanged in this case; any data beyond this value in the can_frame_t::data array should not be used.

Test Modes

The CANFDG peripheral provides three basic test modes: Listen Only, Internal Loopback and External Loopback. Use R_CANFDG_ModeTransition to switch to a test mode.

On some MCUs an additional "Internal Bus" test mode is available that allows connecting both CANFDG channels together on an internal bus, effectively creating an internal CAN network. See the

Internal Bus example below for details.

Existing Low Power Consumption modes

For RX74M/RX74N, the message buffer RAM state is undefined after exiting Software Standby Mode 1.

Since the CANFDG module is not reset during Software Standby Mode 1, the message buffer RAM and the acceptance filter list RAM are not automatically reinitialized after wake-up. Therefore, we must manually initialize the message buffer RAM before resuming CANFDG operation.

Follow the software flow shown below to enter **RAM Test Mode** and clear all message buffer RAM and the acceptance filter list RAM locations by writing 0 to each location.

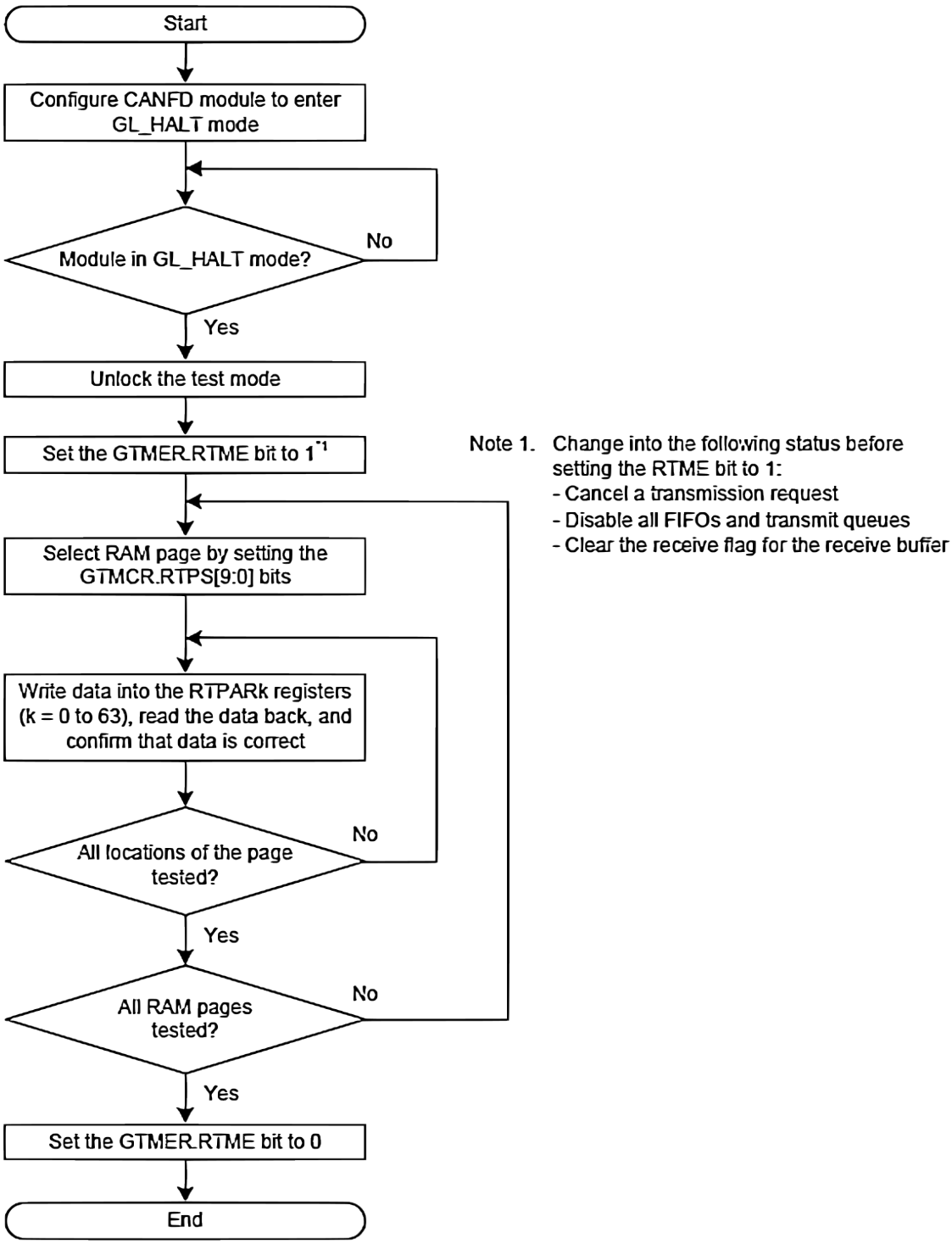


Figure 102: RAM Test Mode Software Flow

After completing the RAM initialization, perform the **CANFD Module Configuration after a Reset** procedure.

Limitations

Developers should be aware of the following limitations when using CANFD:

- On CANFDG module, RX Message Buffers do not have an associated interrupt. To use them in an application one of the following is recommended:
  - Use R_CANFDG_InfoGet to determine if any RX MBs have received data, then use R_CANFDG_Read to obtain it
  - Select an RX FIFO as an additional destination for the relevant filter rules and configure the FIFO interrupt/callback as desired
- The CANFDG peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. See the RX Buffer Pool section above for more information.
- When switching modes with R_CANFDG_ModeTransition a delay of up to several CAN frames may be incurred. Consult Section "Timing of Channel Mode Change" in the CANFDG section of the relevant hardware manual for details.
- Only one channel will receive callbacks for Global Errors. If a different channel is opened first these error interrupts will be suppressed until the specified handler channel is opened.
- The CANFDG driver does not currently support updating AFL entries in Operation mode (communication active). Please refer to the "Updating AFL entry during communication" section in your device's manual on how to use this feature (if available).

Examples

AFL Example

The below is an example Acceptance Filter List (AFL) declaration with two rules.

```
const canfdg_afl_entry_t p_canfdg0_afl[CANFDG_CFG_AFL_CH1_RULE_NUM] =
{
 /* Store all data frames with at least 4 bytes from Standard IDs 0x40-0x4F in RX
FIFO 0 and RX FIFO 1 */
    {
        .id_u.id =
      {
            .id         = 0x40,
         .frame_type = CAN_FRAME_TYPE_DATA,
            .id_mode    = CAN_ID_MODE_STANDARD
        },
        .mask_u.mask =
      {
            .mask_id         = 0x7F0,
         .mask_frame_type = 1,
         .mask_id_mode    = 1,
        },
        .destination_u.destination =
```

```
      {
            .minimum_dlc       = CANFDG_MINIMUM_DLC_4,
            .rx_buffer         = CANFDG_RX_MB_NONE,
            .fifo_select_flags = (canfdg_rx_fifo_t) (CANFDG_RX_FIFO_0 |
CANFDG_RX_FIFO_1),
        }
    },
 /* Store all frames from Extended ID 0x1100 in RX FIFO 2 and RX MB 0 */
    {
        .id_u.id =
      {
            .id         = 0x1100,
         .frame_type = CAN_FRAME_TYPE_DATA, // This setting is ignored by the mask
            .id_mode    = CAN_ID_MODE_EXTENDED
        },
        .mask_u.mask =
      {
         .mask_id         = 0x1FFFFFFF,
            .mask_frame_type = 0,
         .mask_id_mode    = 1,
        },
        .destination_u.destination =
      {
            .minimum_dlc       = CANFDG_MINIMUM_DLC_0,
            .rx_buffer         = CANFDG_RX_MB_0,
            .fifo_select_flags = CANFDG_RX_FIFO_2,
        }
    }
};
```

Basic Example

This is a basic example of minimal use of the CANFDG module in an application.

Note

On CANFDG module, it is recommended to use RX FIFOs for reception as there are no interrupts for RX message

buffers.

```
#define CAN_EXAMPLE_ID (0x20)
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile canfdg_error_t g_err_status = (canfdg_error_t) 0;
void canfdg_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
      {
 /* Read received frame */
            memcpy(&g_can_rx_frame, &p_args->frame, sizeof(can_frame_t));
 /* Handle event */
 break;
        }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
      {
 /* Handle event */
 break;
        }
 case CAN_EVENT_ERR_GLOBAL:                        /* Global error. */
 case CAN_EVENT_ERR_CHANNEL:                       /* Channel error. */
      {
 /* Get error status */
            g_err_status = (canfdg_error_t) p_args->error; /* Check error code with
canfdg_error_t. */
 /* Handle event */
 break;
        }
 default:
      {
 break;
        }
```

```
    }
}
void canfdg_basic_example (void)
{
 fsp_err_t err;
 /* Initialize the CAN module */
    err = R_CANFDG_Open(&g_canfdg0_ctrl, &g_canfd0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Setup frame to write to CAN ID 0x20 */
    g_can_tx_frame.id               = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = 8;
    g_can_tx_frame.options          = 0;
 /* Write some data to the transmit frame */
 for (uint32_t i = 0; i < 8; i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    err = R_CANFDG_Write(&g_canfdg0_ctrl, CANFDG_TX_MB_0, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
 /* Wait for a transmit callback event */
}
```

Flexible Data

This example demonstrates sending an FD message with bitrate switching over external loopback. The CTX and CRX pins must be connected when using external loopback, though if a CAN transceiver is onboard a 120 Ohm resistor should be connected across CANH and CANL instead.

```
#define CAN_EXAMPLE_64_BYTES 64
void canfdg_fd_loopback_example (void)
{
 fsp_err_t err;
```

```
    err = R_CANFDG_Open(&g_canfdg0_ctrl, &g_canfd0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Switch to external loopback mode */
    err = R_CANFDG_ModeTransition(&g_canfdg0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
    assert(FSP_SUCCESS == err);
 /* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
    g_can_tx_frame.id               = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
    g_can_tx_frame.options          = CANFDG_FRAME_OPTION_FD |
CANFDG_FRAME_OPTION_BRS;
 /* Write some data to the transmit frame */
 for (uint32_t i = 0; i < CAN_DATA_BUFFER_LENGTH; i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    err = R_CANFDG_Write(&g_canfdg0_ctrl, CANFDG_TX_MB_0, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
 /* Wait for a transmit and/or receive callback event */
}
```

Internal Bus

In this example two CANFDG channels are connected to the Internal Bus test mode. API error checking has been omitted for clarity.

Note

Internal Bus mode is only available on MCUs with more than one CANFDG channel. In addition, use of Global Modes for any other purpose is not recommended without consulting the device User's Manual.

Data Structures

struct	canfdg_afl_entry_t
struct	canfdg_global_cfg_t

struct	canfdg_extended_cfg_t

Enumerations

enum	canfdg_status_t
enum	canfdg_error_t
enum	canfdg_tx_buffer_t
enum	canfdg_tx_mb_t
enum	canfdg_rx_buffer_t
enum	canfdg_rx_mb_t
enum	canfdg_rx_fifo_t
enum	canfdg_minimum_dlc_t
enum	canfdg_frame_options_t

Data Structure Documentation

◆ canfdg_afl_entry_t

struct canfdg_afl_entry_t
AFL Entry (based on R_CANFDG_AFL_Type in r_canfdg_reg.h)

◆ canfdg_global_cfg_t

struct canfdg_global_cfg_t		
CANFDG Global Configuration		
Data Fields		
uint32_t	global_interrupts	Global control options (GCR register setting)
uint32_t	global_config	Global configuration options (GCFG register setting)
uint32_t	rx_fifo_config[8]	RX FIFO configuration (RFCRn register settings)
uint32_t	rx_mb_config	Number and size of RX Message Buffers (RMCR register setting)
uint8_t	global_err_ipl	Global Error interrupt priority.
uint8_t	rx_fifo_ipl	RX FIFO interrupt priority.

uint32_t	common_fifo_config[R_CANFDG_NUM_COMMON_FIFOS]	Common FIFO configurations.

◆ canfdg_extended_cfg_t

struct canfdg_extended_cfg_t		
CANFD Extended Configuration		
Data Fields		
canfdg_afl_entry_t const *	p_afl	AFL rules list.
uint64_t	txmb_txi_enable	Array of TX Message Buffer enable bits.
uint32_t	error_interrupts	Error interrupt enable bits.
can_bit_timing_cfg_t *	p_data_timing	FD Data Rate (when bitrate switching is used)
uint8_t	delay_compensation	FD Transceiver Delay Compensation (enable or disable)
canfdg_global_cfg_t *	p_global_cfg	Global configuration (global error callback channel only)

Enumeration Type Documentation

◆ canfdg_status_t

enum canfdg_status_t	
CANFDG Status	
Enumerator	
CANFDG_STATUS_RESET_MODE	Channel in Reset mode.
CANFDG_STATUS_HALT_MODE	Channel in Halt mode.
CANFDG_STATUS_SLEEP_MODE	Channel in Sleep mode.
CANFDG_STATUS_ERROR_PASSIVE	Channel in error-passive state.
CANFDG_STATUS_BUS_OFF	Channel in bus-off state.
CANFDG_STATUS_TRANSMITTING	Channel is transmitting.
CANFDG_STATUS_RECEIVING	Channel is receiving.
CANFDG_STATUS_READY	Channel is ready for communication.
CANFDG_STATUS_ESI	At least one CAN-FD message was received with the ESI flag set.

◆ canfdg_error_t

enum canfdg_error_t	
CANFDG Error Code	
Enumerator	
CANFDG_ERROR_CHANNEL_BUS	Bus Error.
CANFDG_ERROR_CHANNEL_WARNING	Error Warning (TX/RX error count over 0x5F)
CANFDG_ERROR_CHANNEL_PASSIVE	Error Passive (TX/RX error count over 0x7F)
CANFDG_ERROR_CHANNEL_BUS_OFF_ENTRY	Bus-Off State Entry.
CANFDG_ERROR_CHANNEL_BUS_OFF_RECOVERY	Recovery from Bus-Off State.
CANFDG_ERROR_CHANNEL_OVERLOAD	Overload.
CANFDG_ERROR_CHANNEL_BUS_LOCK	Bus Locked.
CANFDG_ERROR_CHANNEL_ARBITRATION_LOSS	Arbitration Lost.

CANFDG_ERROR_CHANNEL_STUFF	Stuff Error.
CANFDG_ERROR_CHANNEL_FORM	Form Error.
CANFDG_ERROR_CHANNEL_ACK	ACK Error.
CANFDG_ERROR_CHANNEL_CRC	CRC Error.
CANFDG_ERROR_CHANNEL_BIT_RECESSIVE	Bit Error (recessive) Error.
CANFDG_ERROR_CHANNEL_BIT_DOMINANT	Bit Error (dominant) Error.
CANFDG_ERROR_CHANNEL_ACK_DELIMITER	ACK Delimiter Error.
CANFDG_ERROR_GLOBAL_DLC	DLC Error.
CANFDG_ERROR_GLOBAL_MESSAGE_LOST	Message Lost.
CANFDG_ERROR_GLOBAL_PAYLOAD_OVERFLOW	FD Payload Overflow.
CANFDG_ERROR_GLOBAL_TXQ_OVERWRITE	TX Queue Message Overwrite.
CANFDG_ERROR_GLOBAL_TXQ_MESSAGE_LOST	TX Queue Message Lost.
CANFDG_ERROR_GLOBAL_CH0_SCAN_FAIL	Channel 0 RX Scan Failure.
CANFDG_ERROR_GLOBAL_CH1_SCAN_FAIL	Channel 1 RX Scan Failure.
CANFDG_ERROR_GLOBAL_CH0_ECC	Channel 0 ECC Error.
CANFDG_ERROR_GLOBAL_CH1_ECC	Channel 1 ECC Error.

◆ canfdg_tx_buffer_t

enum canfdg_tx_buffer_t
CANFDG Transmit Buffer (MB + CFIFO)

◆ canfdg_tx_mb_t

enum canfdg_tx_mb_t
CANFDG Transmit Message Buffer (TX MB)

◆ canfdg_rx_buffer_t

enum canfdg_rx_buffer_t
CANFDG Receive Buffer (MB + FIFO + CFIFO)

◆ canfdg_rx_mb_t

enum canfdg_rx_mb_t
CANFD Receive Message Buffer (RX MB)

◆ canfdg_rx_fifo_t

enum canfdg_rx_fifo_t
CANFDG Receive FIFO (RX FIFO)

◆ canfdg_minimum_dlc_t

enum canfdg_minimum_dlc_t
CANFDG AFL Minimum DLC settings

◆ canfdg_frame_options_t

enum canfdg_frame_options_t	
CANFDG Frame Options	
Enumerator	
CANFDG_FRAME_OPTION_ERROR	Error state set (ESI).
CANFDG_FRAME_OPTION_BRS	Bit Rate Switching (BRS) enabled.
CANFDG_FRAME_OPTION_FD	Flexible Data frame (FDF).

Function Documentation

◆ R_CANFDG_Open()

fsp_err_t R_CANFDG_Open (can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg*)

Open and configure the CANFDG channel for operation.

Example:

```
/* Initialize the CAN module */
   err = R_CANFDG_Open(&g_canfdg0_ctrl, &g_canfd0_cfg);
```

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ALREADY_OPEN	Driver already open.
FSP_ERR_IN_USE	Channel is already in use.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel does not exist on this MCU.
FSP_ERR_ASSERTION	A required pointer was NULL.
FSP_ERR_CAN_INIT_FAILED	The provided nominal or data bitrate is invalid.
FSP_ERR_CLOCK_INACTIVE	CANFD source clock is disabled (PLL or PLL2).

◆ R_CANFDG_Close()

fsp_err_t R_CANFDG_Close (can_ctrl_t *const *p_api_ctrl*)

Close the CANFDG channel.

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented.

◆ R_CANFDG_Write()

fsp_err_t R_CANFDG_Write (can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame*)

Write data to the CANFDG channel.

Example:

```
/* Send data on the bus */
   err = R_CANFDG_Write(&g_canfdg0_ctrl, CANFDG_TX_MB_0, &g_can_tx_frame);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
FSP_ERR_INVALID_ARGUMENT	Data length or buffer number invalid.
FSP_ERR_INVALID_MODE	An FD option was set on a non-FD frame.
FSP_ERR_ASSERTION	One or more pointer arguments is NULL.
FSP_ERR_UNSUPPORTED	FD is not supported on this MCU.

◆ R_CANFDG_Read()

fsp_err_t R_CANFDG_Read (can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame*)

Read data from a CANFDG Message Buffer or FIFO.

Example: snippet r_canfdg_example.c R_CANFDG_Read

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_ARGUMENT	Buffer number invalid.
FSP_ERR_ASSERTION	p_api_ctrl or p_frame is NULL.
FSP_ERR_BUFFER_EMPTY	Buffer or FIFO is empty.

◆ R_CANFDG_ModeTransition()

fsp_err_t R_CANFDG_ModeTransition (can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode*)

Switch to a different channel, global or test mode.

Example:

```
/* Switch to external loopback mode */
   err = R_CANFDG_ModeTransition(&g_canfdg0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented
FSP_ERR_INVALID_MODE	Cannot change to the requested mode from the current global mode.

◆ R_CANFDG_InfoGet()

fsp_err_t R_CANFDG_InfoGet (can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info*)

Get CANFDG state and status information for the channel.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CANFDG_CallbackSet()

fsp_err_t R_CANFDG_CallbackSet (can_ctrl_t *const *p_api_ctrl*, void(*)(can_callback_args_t *) *p_callback*, void *const *p_context*, can_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements can_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.6.3 I3C (r_i3c)

Modules » Connectivity

Functions

fsp_err_t	R_I3C_Open (i3c_ctrl_t *const p_api_ctrl, i3c_cfg_t const *const p_cfg)
fsp_err_t	R_I3C_Enable (i3c_ctrl_t *const p_api_ctrl)
fsp_err_t	R_I3C_DeviceCfgSet (i3c_ctrl_t *const p_api_ctrl, i3c_device_cfg_t const *const p_device_cfg)
fsp_err_t	R_I3C_MasterDeviceTableSet (i3c_ctrl_t *const p_api_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)
fsp_err_t	R_I3C_SlaveStatusSet (i3c_ctrl_t *const p_api_ctrl, i3c_device_status_t status)
fsp_err_t	R_I3C_DeviceSelect (i3c_ctrl_t *const p_api_ctrl, uint32_t device_index, uint32_t bitrate_mode)
fsp_err_t	R_I3C_DynamicAddressAssignmentStart (i3c_ctrl_t *const p_api_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)
fsp_err_t	R_I3C_CommandSend (i3c_ctrl_t *const p_api_ctrl, i3c_command_descriptor_t const *const p_command_descriptor)
fsp_err_t	R_I3C_Write (i3c_ctrl_t *const p_api_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)

fsp_err_t	R_I3C_Read (i3c_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t length, bool restart)
fsp_err_t	R_I3C_IbiWrite (i3c_ctrl_t *const p_api_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)
fsp_err_t	R_I3C_IbiRead (i3c_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t length)
fsp_err_t	R_I3C_Close (i3c_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the I3C peripheral on RX MCUs. This module implements the I3C Interface.

Overview

I3C is a communication protocol defined by MIPI that aims to improve on I2C by increasing the maximum transfer rate, as well as providing other features like "In-band Interrupts", "Dynamic Address Assignment", and a set of standard "Common Command Codes".

Features

- I3C Master Mode
- I3C Slave Mode
- Dynamic Address Assignment (ENTDAA/SETDASA)
- SDR Read/Write transfers
- HDR-DDR Read/Write Commands
- I2C Legacy Read/Write transfers
- In-Band Interrupts (Interrupt Requests, Hot-Join Requests)
- Common Command Codes
- Clock Stalling
- Timeout Detection

Master Mode

On an I3C bus, only one device may operate in master mode at a time. The current master is responsible for initiating I2C Legacy transfers, SDR transfers, Common Command Codes, and handling IBIs (Interrupt Requests, Hot-Join Requests). In order to perform these operations, the driver has an internal device table that is used for storing configuration information for each device on the bus (See i3c_device_table_cfg_t). Each entry in the device table contains the static or dynamic address of the device, and IBI permissions for accepting or rejecting IBI requests from the device. The device table has a limited number of entries as well as one extended device entry that only contains the static or dynamic address of a device (See below).

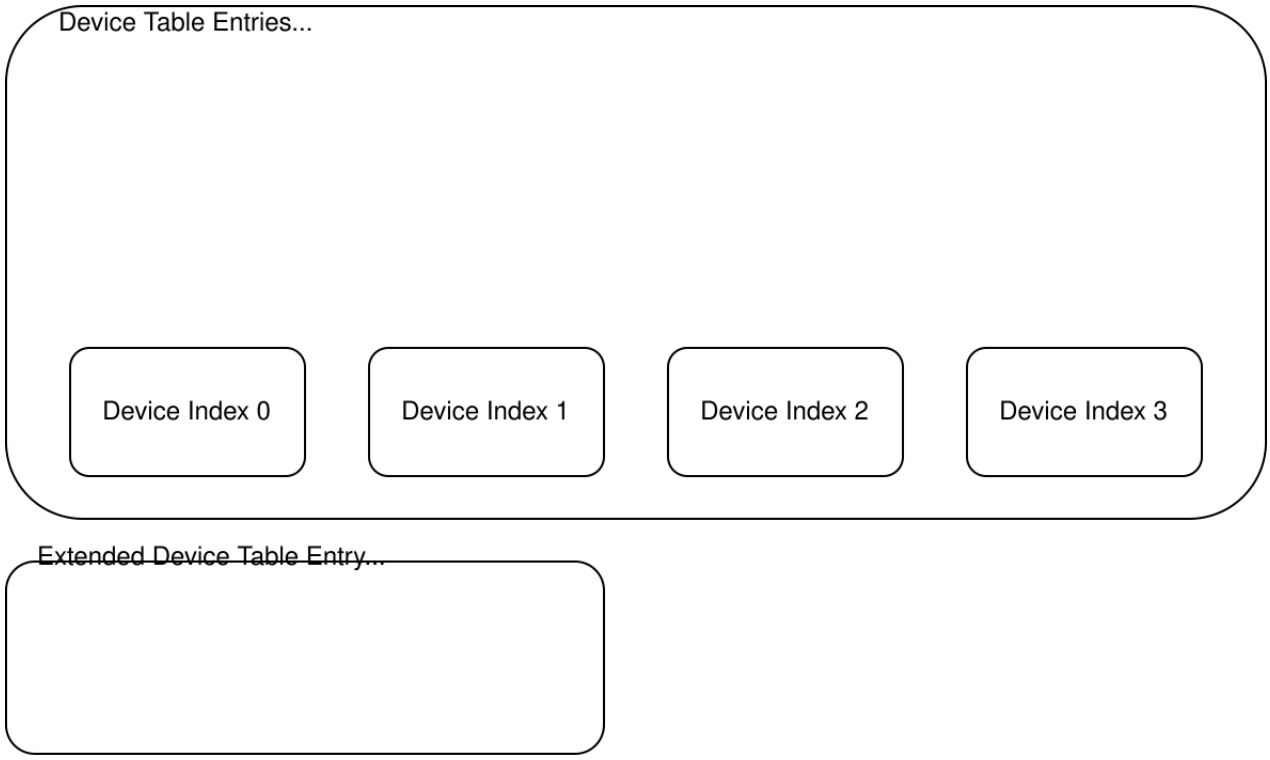


Figure 103: Master Device Table

In order to initiate I2C Legacy transfers, SDR transfers, or Common Command Codes, the master must select a device entry from the device table using i3c_api_t::deviceSelect. Once a device has been selected, all subsequent operations will be directed to the selected device until a new device is selected using i3c_api_t::deviceSelect.

The master may also receive IBI requests that are initiated by slave devices on the bus. If there is a payload, then the driver will write the data into a buffer that is provided by the application by calling i3c_api_t::ibiRead. If the application has not provided an IBI buffer prior to receiving an IBI, then the it will get a callback requesting an IBI buffer. Once the IBI is completed, the application will be notified by a callback.

Note

1. Even though there are only a limited number of device table entries and one extended device table entry, the application can operate on more devices by maintaining its own list of devices and updating the extended device entry as needed. Note however that devices defined in the extended device table entry will not be able to initiate IBI requests.

Main Master

The main master is responsible for configuring the dynamic address of all devices on the bus. The driver initiates this procedure by calling i3c_api_t::dynamicAddressAssignmentStart. Before starting address assignment, the application must configure the device table using i3c_api_t::masterDeviceTableSet.

Enter Dynamic Address Assignment (ENTDAA):

The application initiates the ENTDAA operation by calling i3c_api_t::dynamicAddressAssignmentStart with a starting index into the master device table and a count specifying the number of devices to configure. The master starts by sending the ENTDAA command. Every I3C device on the bus that has not already been initialized will acknowledge the command and attempt to write its Provisional ID, DCR, and BCR registers. The device with the smallest value in these registers will win arbitration and be assigned with the first dynamic address defined in the master device table. The master will then increment the index and repeat the process by assigning the dynamic address to the next device. The process continues until the specified number of devices have been initialized or until there are no more devices to configure.

Note

1. The IBI payload setting will automatically be updated in the master device table based on the BCR setting that was read during ENTDAA.
2. After each device successfully writes its Provisional ID, DCR, and BCR registers, the application will get a callback that will provide the value of the registers.
3. If the starting index is set to the extended device entry, then the device count must be set to 1.
4. The main master assigns its own dynamic address with i3c_api_t::deviceCfgSet.

Set Dynamic Address from Static Address (SETDASA):

The application initiates the SETDASA operation by calling i3c_api_t::dynamicAddressAssignmentStart with an index into the master device table. The master sends the SETDASA command to the static address defined in the given device table entry, and then assigns the associated dynamic address.

Note

1. Set the count to 0 when using SETDASA.

Slave Mode

In slave mode, the device configures its static address, Provisional ID, BCR, and DCR registers using i3c_api_t::deviceCfgSet, and then waits for the master to initiate communication. Prior to being assigned a dynamic address, the slave will operate as an I2C device using its static address. The application will receive a callback when the master assigns it a dynamic address, after which point, the slave will operate as an I3C device until it receives the RSTDAA command.

Depending on the capabilities defined in its BCR register, the slave may also initiate IBI Interrupt Requests, and Hot-Join Requests using i3c_api_t::ibiWrite

Configuration

Build Time Configurations for r_i3c

The following build time configurations are defined in fsp_cfg/r_i3c_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Unaligned Buffer Support	• Enabled • Disabled	Enabled	Unaligned buffer support may be

			optionally disabled for improved performance.
Master Support	• Enabled • Disabled	Enabled	If only slave mode is required, disable master support to decrease code size.
Slave Support	• Enabled • Disabled	Enabled	If only master mode is required, disable slave support to decrease code size.
Error Recovery Procedure	MCU Specific Options		There are two different chip versions of Error Recovery. Each version requires a different procedure for recovering from errors.

Configurations for Connectivity > I3C (r_i3c)

This module can be added to the Stacks tab via New Stack > Connectivity > I3C (r_i3c).

Configuration	Options	Default	Description
Bitrate Settings > Standard Mode > Open-Drain > Logic High Period (ns)	Must be an integer greater than 0.	167	The Logic High period of SCL during Standard Mode Open Drain transfers.
Bitrate Settings > Standard Mode > Open-Drain > Frequency	Must be an integer greater than 0.	1000000	The Frequency of SCL during Standard Mode Open Drain transfers.
Bitrate Settings > Standard Mode > Push-Pull > Logic High Period (ns)	The Logic High Period must be greater than or equal to 24 Nanoseconds.	167	The Logic High period of SCL during Standard Mode Push Pull transfers.
Bitrate Settings > Standard Mode > Push-Pull > Frequency	Push-Pull frequency must be greater than or equal to 10000 Hz.	3400000	The Frequency of SCL during Standard Mode Push-Pull transfers.
Bitrate Settings > Extended Mode > Open-Drain > Logic High Period (ns)	Must be an integer greater than 0.	167	The Logic High period of SCL during Extended Mode Open Drain transfers.
Bitrate Settings > Extended Mode > Open-Drain > Frequency	Must be an integer greater than 0.	1000000	The Frequency of SCL during Extended Mode Open Drain transfers.
Bitrate Settings > Extended Mode > Push-Pull > Logic High Period (ns)	The Logic High Period must be greater than or equal to 24 Nanoseconds.	167	The Logic High period of SCL during Extended Mode Push Pull transfers.
Bitrate Settings >	Push-Pull frequency	3400000	The Frequency of SCL

Extended Mode > Push-Pull > Frequency	must be greater than or equal to 10000 Hz.		during Extended Mode Push-Pull transfers.
Bitrate Settings > Bus Timing > Open Drain Rising Time (ns)	Rising time must be greater than or equal to 0 nanoseconds.	0	The Open Drain rising time in nanoseconds.
Bitrate Settings > Bus Timing > Open Drain Falling Time (ns)	Falling time must be greater than or equal to 0 nanoseconds.	0	The Open Drain falling time in nanoseconds.
Bitrate Settings > Bus Timing > Push-Pull Rising Time (ns)	Rising time must be greater than or equal to 0 nanoseconds.	0	The Push-Pull rising time in nanoseconds.
Bitrate Settings > Bus Timing > Push-Pull Falling Time (ns)	Falling time must be greater than or equal to 0 nanoseconds.	0	The Push-Pull rising time in nanoseconds.
Bitrate Settings > Clock Stalling > Address Assignment Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the Address Assignment Phase of ENTDAA.
Bitrate Settings > Clock Stalling > Transition Phase	MCU Specific Options		Enable clock stalling during the Transition Bit of a read transfer.
Bitrate Settings > Clock Stalling > Parity Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the Parity Bit of a write transfer.
Bitrate Settings > Clock Stalling > Ack Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the ACK phase of a transfer.
Bitrate Settings > Clock Stalling > Time (us)	Must be greater than or equal to 0.	0	The amount of time to stall the clock during the Address Assignment Phase, Transition Phase, Parity Phase, and ACK Phase.
Master Mode > ACK Hot-Join Requests	• Enabled • Disabled	Disabled	If enabled, the I3C instance will ACK Hot-Join Requests and notify the application.
Master Mode > Notify Rejected Hot-Join Requests.	• Enabled • Disabled	Disabled	If enabled, the application will get a callback when an IBI Hot-Join Request is rejected.
Master Mode > Notify Rejected Mastership Requests.	• Enabled • Disabled	Disabled	If enabled, the application will get a callback when an IBI Mastership Request is rejected.

Master Mode > Notify Rejected Interrupt Requests.	• Enabled • Disabled	Disabled	If enabled, the application will get a callback when an IBI Interrupt Request is rejected.
Slave Mode > Command Response Info > ENEC/DISEC > In-Band Interrupts	• Enabled • Disabled	Disabled	Configure whether the slave can issue IBI requests.
Slave Mode > Command Response Info > ENEC/DISEC > Hot-Join Requests	• Enabled • Disabled	Disabled	Configure whether the slave can issue Hot-Join requests.
Slave Mode > Command Response Info > ENTASn > Activity State	• Activity State 0 • Activity State 1 • Activity State 2 • Activity State 3	Activity State 0	Configure the starting activity state of the slave.
Slave Mode > Command Response Info > SETMWL/GETMWL > Max Write Length	Write length must be in the range of [8, 65535].	65535	Set the max write length.
Slave Mode > Command Response Info > SETMRL/GETMRL > Max Read Length	Read length must be in the range of [16, 65535].	65535	Set the max read length.
Slave Mode > Command Response Info > SETMRL/GETMRL > Max IBI Payload Length	Read length must be in the range of [0, 255].	0	Set the max IBI payload length, or set it to 0 for unlimited.
Slave Mode > Command Response Info > GETMXDS > Write Data Rate	• FSCL_MAX • 8Mhz • 6Mhz • 4Mhz • 2Mhz	2Mhz	Set the max write data rate.
Slave Mode > Command Response Info > GETMXDS > Read Data Rate	• FSCL_MAX • 8Mhz • 6Mhz • 4Mhz • 2Mhz	2Mhz	Set the max read data rate.
Slave Mode > Command Response Info > GETMXDS > Clock to Data Turnaround Time	• 8 Nanoseconds • 9 Nanoseconds • 10 Nanoseconds • 11 Nanoseconds • 12 Nanoseconds	8 Nanoseconds	Set the clock to data turnaround time.

	• Greater than 12 Nanoseconds		
Slave Mode > Command Response Info > GETMXDS > Include Max Read Turnaround Time	• Enabled • Disabled	Disabled	Configure whether the Max Read Turnaround time will be transmitted.
Slave Mode > Command Response Info > GETMXDS > Max Read Turnaround Time	Value must be in the range [0, 255].	0	Set max read turnaround time.
Slave Mode > Command Response Info > GETXTIME > Frequency Byte	Value must be in the range [0, 255].	0	Set the internal oscillator frequency in increments of 0.5 Mhz.
Slave Mode > Command Response Info > GETXTIME > Inaccuracy Byte	Value must be in the range [0, 255].	0	Set the oscillator inaccuracy byte in increments of 0.5%
Slave Mode > Command Response Info > GETCAP/GETHDRCAP > HDR-DDR (Mode 0)	MCU Specific Options		Slave supports HDR-DDR (Mode 0)
Slave Mode > Command Response Info > GETCAP/GETHDRCAP > HDR-TSP (Mode 1)	MCU Specific Options		Slave supports HDR-TSP (Mode 1)
Slave Mode > Command Response Info > GETCAP/GETHDRCAP > HDR-TSL (Mode 2)	MCU Specific Options		Slave supports HDR-TSL (Mode 2)
Interrupts > Interrupt Priority	MCU Specific Options		The interrupt priority of the RX, TX, RESPONSE, RCV_STATUS, and IBI ISRs.
Interrupts > Error and Event Interrupt Priority	MCU Specific Options		The interrupt priority of the EEI ISR which is used to notify the application when an Internal Error, HDR Exit Pattern, or Timeout is detected.
Name	Name must be a valid C symbol	g_i3c0	Module name.
Callback	Name must be a valid	g_i3c0_callback	A user callback

	C symbol		function must be provided. This will be called in order to notify the application of I3C events and provide status information.
Device Type	• Main Master • Slave	Slave	The role that the I3C instance will take on the I3C bus.
Bus Free Condition Detection Time (ns)	Must be greater than or equal to 38.4 nanoseconds.	38.4	The minimum period occurring after a STOP and before a START.
Bus Available Condition Detection Time (us)	Must be greater than or equal to 1 microsecond.	1	The minimum period occurring after the Bus Free Condition when Slaves can initiate IBI requests.
Bus Idle Condition Detection Time (us)	Must be greater than or equal to 1000 microseconds.	1000	The minimum period occurring after the Bus Available Condition when Slaves can initiate Hot-Join requests.
Timeout Detection	• Enabled • Disabled	Disabled	If enabled, the application will get a callback if SCL is stuck at a logic high or logic low level for more than 65535 cycles of the I3C source clock.

Clock Configuration

The following settings are used to configure the timing of SCL.

- Frequency of TCLK
- Standard Mode
  - Open Drain High Period (T_{HIGH})
  - Open Drain Frequency
  - Push-Pull High Period (T_{HIGH})
  - Push-Pull Frequency
- Extended Mode
  - Open Drain High Period (T_{HIGH})
  - Open Drain Frequency
  - Push-Pull High Period (T_{HIGH})
  - Push-Pull Frequency
- i3c_bitrate_mode_t (Set during i3c_api_t::deviceSelect)

The Standard and Extended Mode settings define two separate SCL configurations that can be selected at run-time using i3c_api_t::deviceSelect.

In addition to selecting between the Standard and Extended Mode settings, the base SCL period can

also be multiplied using the following options:

- I3C_BITRATE_MODE_I3C_SDR2_STDBR_X2: Multiple the base Standard Open Drain and Push-Pull period by 2.
- I3C_BITRATE_MODE_I3C_SDR3_EXTBR_X2: Multiple the base Extended Open Drain and Push-Pull period by 2.
- I3C_BITRATE_MODE_I3C_SDR4_EXTBR_X4: Multiple the base Extended Open Drain and Push-Pull period by 4.

In order to get accurate frequency calculations, the Rising and Falling edges must be input into the calculation. These values will depend on the topology the I3C bus that will be different for every application.

Note

1. The Standard and Extended Open Drain period settings define the period to use during legacy I2C transfers (Only use the following i3c_bitrate_mode_t settings with I2C transfers: I3C_BITRATE_MODE_I2C_STDBR, I3C_BITRATE_MODE_I2C_EXTBR).
2. T_{HIGH} is defined in Figure 31 in the MIPI I3C Specification v1.0 and describes the Logic High period.
3. Depending on the MCU, TCLK is either derived from PCLKD or from a dedicated I3C Clock (I3CCLK).

Pin Configuration

The I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I3C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Read and Write in Slave Mode

In slave mode, calling read or write does not start a transfer. Instead, calling read or write will configure the driver to perform the next read or write transfer using the user provided buffer.

Provided that a transfer is not already in progress, i3c_api_t::read and i3c_api_t::write can be called to update the internal buffers even if the transfer has not been completed yet. Both the read and write buffers can be configured at the same time in order to prepare the driver for when the master initiates a read or a write transfer.

If there is no space remaining in a user configured read buffer, the application will get a I3C_EVENT_READ_BUFFER_FULL callback requesting for a new read buffer to be provided.

Event Status

When a write, read, ibiWrite, ibiRead or commandSend, operation is completed, the i3c_callback_args_t::event_status should be checked. The event_status will provide information about the success or failure of the operation.

The following are possible statuses:

- I3C_EVENT_STATUS_SUCCESS
- I3C_EVENT_STATUS_PARITY
- I3C_EVENT_STATUS_FRAME
- I3C_EVENT_STATUS_ADDRESS_HEADER
- I3C_EVENT_STATUS_NACK

- I3C_EVENT_STATUS_OVERFLOW
- I3C_EVENT_STATUS_ABORTED_TO_MASTER
- I3C_EVENT_STATUS_ABORTED

Note

In master mode, if the master issues a stop condition before the slave ends the transfer via the 'T' bit, the status will be I3C_EVENT_STATUS_ABORTED.
In slave mode, if the master issues a stop condition before the slave ends the transfer via the 'T' bit, the status will be I3C_EVENT_STATUS_ABORTED_TO_MASTER.

Direct Get Common Command Codes in Slave Mode

When an I3C slave receives a Command Code of type Direct Get, the response is automatically sent from the device Special Function Registers (SFR). The SFR contains information for each command code and can be configured during open (See i3c_extended_cfg_t::slave_command_response_info). This allows the slave to respond to Direct Get Command Codes much faster, and removes the burden of responding to these commands from the application.

The response to the GETSTATUS command can be configured at run-time using i3c_api_t::slaveStatusSet.

Disabling Unaligned Buffer Support

Support for performing read and write operations on unaligned buffers can be disabled in order to improve performance. When unaligned buffer support is disabled, all buffers passed to read, ibiRead, write, ibiWrite, and commandSend must be aligned to 4 bytes and the size of the buffers must be a multiple of 4 bytes.

In master mode, the value of the length passed to i3c_api_t::read and i3c_api_t::write sets the total length of the operation in bytes. During the read or write operation, the driver may read or write to the last word of memory during the operation. This means that the allocated memory for the buffer passed to read and write needs to be a multiple of 4 bytes even though the transfer length is not a multiple of 4 bytes.

In slave mode, the length passed to i3c_api_t::read must be a multiple of 4 bytes. The length passed to i3c_api_t::write sets the number of bytes that the slave will write. The size of the buffer passed to write still needs to be a multiple of 4 bytes.

Mixed Fast Bus

The MIPI I3C Specification v1.0 defines a Mixed Fast Bus as a bus that has legacy I2C devices that all have a 50ns Spike Filter.

On Mixed Fast Buses, SCL has the following constraints during I3C SDR transfers:

- SCL High Period: $t_{DIG_H_MIXED(MIN)}$ to $t_{DIG_H_MIXED(MAX)}$
- SCL Low Period: Up to $t_{DIG_L(MAX)}$

In this case, configure the Extended Bitrate settings for I3C SDR transfers, and use the Standard Bitrate settings for I2C transfers.

Note

1. See section 5.1.2.4 in the MIPI I3C Specification v1.0.

Mixed Slow Bus

The MIPI I3C Specification v1.0 defines a Mixed Slow Bus as a bus that has legacy I2C devices that do not have a 50ns Spike Filter.

In this case, the SCL frequency is limited to I2C Fast Mode or I2C Fast Mode Plus.

Note

1. See section 5.1.2.4 in the MIPI I3C Specification v1.0.

Limitations

Developers should be aware of the following limitations when using the I3C:

- The MIPI Reserved area and Vendor Extension area of Command Codes are not supported.
- Secondary Master device role is not currently supported.

Examples

I3C Master Basic Example

This is a basic example of minimal use of the I3C Master in an application.

```
void i3c_master_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t master_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_MASTER_STATIC_ADDRESS,
 /* If the device is a main master, it must configure its own dynamic address. */
       .dynamic_address = EXAMPLE_MASTER_DYNAMIC_ADDRESS,
    };
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &master_device_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_table_cfg_t device_table_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address       = EXAMPLE_STATIC_ADDRESS,
```

```
 /* Dynamic address is not used in I2C. */
       .dynamic_address       = EXAMPLE_DYNAMIC_ADDRESS,
 /* This is the type of device. It may be either an I2C device or an I3C device. */
       .device_protocol       = I3C_DEVICE_PROTOCOL_I3C,
       .ibi_accept            = false,
 /* Depending on the device the IBI requests may have a data payload.
  * Note that this field will be automatically updated if the device is configured
using ENTDAA.
  */
       .ibi_payload          = false,
 /* Master requests cannot be accepted because Secondary Master is not supported. */
       .master_request_accept = false,
    };
 /* Set the device configuration in the master device table. */
    status = R_I3C_MasterDeviceTableSet(&g_i3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable the I3C device. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 /* Start assigning dynamic addresses to devices on the bus using the ENTDAA command.
*/
    status = R_I3C_DynamicAddressAssignmentStart(&g_i3c_ctrl,
I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA, 0, 1);
    assert(FSP_SUCCESS == status);
 /* Wait for dynamic address assignment to complete. */
    i3c_app_event_wait(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 /* Select the configured device and bitrate mode for the following operations. */
    status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0, I3C_BITRATE_MODE_I3C_SDR0_STDBR);
    assert(FSP_SUCCESS == status);
 /* Start a write transfer. */
 static uint8_t p_write_buffer[] = {1, 2, 3, 4, 5};
    status = R_I3C_Write(&g_i3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the write transfer to complete. */
```

```
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
 /* Start a read transfer. */
 static uint8_t p_read_buffer[16];
    status = R_I3C_Read(&g_i3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the read transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
}
/* This function is called by the I3C driver from ISRs in order to notify the
application of I3C events. */
void i3c_master_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_ENTDAA_ADDRESS_PHASE:
      {
 /* The device PID, DCR, and BCR registers will be available in
i3c_callback_args_t::p_slave_info. */
 break;
       }
 case I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 break;
       }
 case I3C_EVENT_WRITE_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read from the slave will be available in
i3c_callback_args_t::transfer_size. */
```

```
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Slave Basic Example

This is a basic example of minimal use of the I3C Slave in an application.

```
void i3c_slave_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t slave_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_STATIC_ADDRESS,
 /* The dynamic address will be automatically updated when the master configures this
device using ENTDAA. */
       .dynamic_address = 0,
 /* Device Registers that are read by the master. */
       .slave_info      =
      {
            .bcr_u.bcr = EXAMPLE_BCR_SETTING,
            .dcr       = EXAMPLE_DCR_SETTING,
            .pid       =
      {
                0, 1, 2, 3, 4, 5
```

```
        }
      }
   };
 /* Set the device configuration for this device. */
   status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &slave_device_cfg);
   assert(FSP_SUCCESS == status);
 /* Enable Slave Mode. */
   status = R_I3C_Enable(&g_i3c_ctrl);
   assert(FSP_SUCCESS == status);
 static uint8_t p_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 static uint8_t p_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE];
 /* Set the buffer for storing data received during a read transfer. */
   status = R_I3C_Read(&g_i3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
   assert(FSP_SUCCESS == status);
 /* Wait for the master to complete a read transfer. */
   i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
 /* Set the write buffer that will be transmitted during a write transfer. */
   status = R_I3C_Write(&g_i3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
   assert(FSP_SUCCESS == status);
 /* Wait for the master to complete a write transfer. */
   i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
}
void i3c_slave_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
   {
 case I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
     {
           i3c_app_event_notify(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 break;
      }
 case I3C_EVENT_READ_BUFFER_FULL:
     {
 /* If there is no user provided read buffer, or if the user provided read buffer has
```

```
been filled,
  * the driver will notify the application that the buffer is full. The application
may provide
  * a new read buffer by calling i3c_api_t::read. If no read buffer is provided, then
any remaining bytes
  * in the transfer will be dropped. */
            uint8_t * p_read_buffer = i3c_app_next_read_buffer_get();
 R_I3C_Read(&g_i3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE, false);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read by the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 /* Note that the application may also call i3c_api_t::read or i3c_api_t::write from
this event
  * In order to set the transfer buffers for the next transfer. */
 break;
       }
 case I3C_EVENT_WRITE_COMPLETE:
      {
 /* The number of bytes written by the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 /* Note that the application may also call i3c_api_t::read or i3c_api_t::write from
this event
  * In order to set the transfer buffers for the next transfer. */
 break;
       }
 default:
      {
 break;
       }
```

```
    }
}
```

I2C Legacy Basic Example

This is a basic example of minimal use of I2C Legacy transfers in an application.

```
void i2c_legacy_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t master_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_MASTER_STATIC_ADDRESS,
 /* If the device is a main master, it must configure its own dynamic address. */
       .dynamic_address = EXAMPLE_MASTER_DYNAMIC_ADDRESS,
    };
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &master_device_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_table_cfg_t device_table_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address       = EXAMPLE_STATIC_ADDRESS,
 /* Dynamic address is not used in I2C. */
       .dynamic_address       = 0,
 /* This is the type of device. It may be either an I2C device or an I3C device. */
       .device_protocol       = I3C_DEVICE_PROTOCOL_I2C,
 /* These options are not used in I2C. */
       .ibi_accept            = false,
 /* Depending on the device the IBI requests may have a data payload.
  * Note that this field will be automatically updated if the device is configured
```

```
using ENTDAA.
  */
       .ibi_payload          = false,
 /* Master requests cannot be accepted because Secondary Master is not supported. */
       .master_request_accept = false,
    };
 /* Set the device configuration in the master device table. */
    status = R_I3C_MasterDeviceTableSet(&g_i3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable the I3C device. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 /* Select the configured device for the following operations. */
    status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0, I3C_BITRATE_MODE_I2C_STDBR);
    assert(FSP_SUCCESS == status);
 /* Start a write transfer. */
 static uint8_t p_write_data[] = {1, 2, 3, 4, 5};
    status = R_I3C_Write(&g_i3c_ctrl, p_write_data, sizeof(p_write_data), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the write transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
 /* Start a read transfer. */
 static uint8_t p_read_data[16];
    status = R_I3C_Read(&g_i3c_ctrl, p_read_data, sizeof(p_read_data), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the read transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
}
void i2c_legacy_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_WRITE_COMPLETE:
      {
```

```
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read from the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Master In-band Interrupts Example

This is a basic example of reading In-band Interrupts in I3C Master mode.

```
void i3c_master_ibi_basic_example (void)
{
 static uint8_t p_ibi_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Set the buffer for storing IBI data that is read from the slave. */
 fsp_err_t status = R_I3C_IbiRead(&g_i3c_ctrl, p_ibi_read_buffer, sizeof
(p_ibi_read_buffer));
    assert(FSP_SUCCESS == status);
 /* Wait for the ibiRead transfer to complete.
  * Note that the master does not need to wait for the IBI, and can start other
operations. */
    i3c_app_event_wait(I3C_EVENT_IBI_READ_COMPLETE);
}
void i3c_master_ibi_basic_example_callback (i3c_callback_args_t const * const p_args)
{
```

```
 switch (p_args->event)
    {
 case I3C_EVENT_IBI_READ_BUFFER_FULL:
      {
 /* If there is no user provided ibiRead buffer, or if the user provided ibiRead
buffer has been filled,
  * the driver will notify the application that the buffer is full. The application
may provide
  * a new read buffer by calling i3c_api_t::ibiRead. If no read buffer is provided,
then any remaining bytes
  * in the transfer will be dropped. */
            uint8_t * p_read_buffer = i3c_app_next_read_buffer_get();
 R_I3C_IbiRead(&g_i3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE);
 break;
       }
 case I3C_EVENT_IBI_READ_COMPLETE:
      {
 /* When an IBI is completed, the transfer_size, ibi_type, and ibi_address will be
available in p_args. */
 switch (p_args->ibi_type)
      {
 case I3C_IBI_TYPE_INTERRUPT:
      {
 /* Notify the application that an IBI was read. */
                   i3c_app_event_notify(I3C_EVENT_IBI_READ_COMPLETE);
 break;
         }
 case I3C_IBI_TYPE_HOT_JOIN:
      {
 /* If a Hot-Join event is received, then the master can initiate the dynamic address
assignment procedure. */
 R_I3C_DynamicAddressAssignmentStart(&g_i3c_ctrl, I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA,
0, 1);
 break;
```

```
        }
 default:
     {
 break;
        }
        }
      }
 default:
     {
 break;
      }
   }
}
```

I3C Slave In-band Interrupts Example

This is a basic example of writing In-band Interrupts in I3C Slave mode.

```
void i3c_slave_ibi_write_basic_example (void)
{
    uint8_t ibi_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE] = {0};
 /* Initiate an In-band interrupt in slave mode.
  * Note: If the slave does not have an IBI payload or if it is a Hot-Join request,
the write buffer should be set
  * to NULL and the write length should be set to 0. */
 fsp_err_t status = R_I3C_IbiWrite(&g_i3c_ctrl, I3C_IBI_TYPE_INTERRUPT,
ibi_write_buffer, sizeof(ibi_write_buffer));
    assert(FSP_SUCCESS == status);
 /* Wait for the ibiWrite transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_IBI_WRITE_COMPLETE);
}
void i3c_slave_ibi_write_basic_example_callback (i3c_callback_args_t const * const
p_args)
{
 switch (p_args->event)
```

```
    {
 case I3C_EVENT_IBI_WRITE_COMPLETE:
      {
 /* Notify the application that the IBI write is complete. */
            i3c_app_event_notify(I3C_EVENT_IBI_WRITE_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Master Common Command Codes Example

This is a basic example of sending Common Command Codes in I3C Master mode.

```
void i3c_master_ccc_example (void)
{
 static uint8_t command_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Setup the command descriptor. */
 static i3c_command_descriptor_t command_descriptor =
    {
       .command_code = I3C_CCC_DIRECT_GETSTATUS,
 /* Set a buffer for storing the data read by the command. */
       .p_buffer    = command_buffer,
 /* The length for a GETSTATUS command is 2 bytes. */
       .length       = 2,
 /* Terminate the transfer with a STOP condition. */
       .restart      = false,
 /* The GETSTATUS command is a Direct Get Command so rnw should be true. */
       .rnw          = true,
    };
 /* Send the command. */
```

```
 fsp_err_t status = R_I3C_CommandSend(&g_i3c_ctrl, &command_descriptor);
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
 /* The command_buffer will have the status info that was read from the slave device.
*/
}
void i3c_master_ccc_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* Notify the application that the command is complete. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Slave Common Command Codes Example

This is a basic example of receiving Common Command Codes in I3C Slave mode.

```
void i3c_slave_ccc_example (void)
{
 static uint8_t read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Broadcast and Direct Set commands will be read into the read_buffer the same way
that
  * a normal SDR Master Write / Slave Read transfer is read. */
 fsp_err_t status = R_I3C_Read(&g_i3c_ctrl, read_buffer, sizeof(read_buffer), false);
```

```
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
}
void i3c_slave_ccc_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* The command code and transfer size will be available in p_args.
  * If the command code is a Broadcast or Direct Set, then data will
  * be stored in the read buffer provided by i3c_api_t::read.
  * If the command code is a Direct Get, then the data will be automatically
  * sent from device SFR. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C HDR Command Example

This is a basic example of sending an HDR command

```
#define I3C_HDR_COMMAND_CODE (0x55)
#define I3C_HDR_COMMAND_LENGTH (16)
void i3c_hdr_command_example (void)
{
 static uint8_t command_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Setup the command descriptor. */
```

```
 static i3c_command_descriptor_t command_descriptor =
    {
 /* Specify the command code for the HDR transfer. */
       .command_code = I3C_HDR_COMMAND_CODE,
 /* Set a buffer for writing command data. */
       .p_buffer    = command_buffer,
 /* The length for command (HDR commands must be a multiple of 2 bytes). */
       .length       = I3C_HDR_COMMAND_LENGTH,
 /* Terminate the transfer with a HDR-EXIT. */
       .restart      = false,
 /* Specify if this is a read/write command. */
       .rnw          = false,
    };
 /* Select the HDR mode for subsequent transfers. */
 fsp_err_t status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0,
I3C_BITRATE_MODE_I3C_HDR_DDR_STDBR);
    assert(FSP_SUCCESS == status);
 /* Send the command. */
    status = R_I3C_CommandSend(&g_i3c_ctrl, &command_descriptor);
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
}
void i3c_hdr_command_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* Notify the application that the command is complete. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
```

```
      {
 break;
       }
    }
}
```

Data Structures

struct	i3c_clock_stalling_t
struct	i3c_bitrate_settings_t
struct	i3c_ibi_control_t
struct	i3c_slave_command_response_info_t
struct	i3c_instance_ctrl_t
struct	i3c_extended_cfg_t

Macros

#define	I3C_ERROR_RECOVERY_VERSION_1 Support error recovery procedure for chip version 1. More...
#define	I3C_ERROR_RECOVERY_VERSION_2 Support error recovery procedure for chip version 2.
#define	I3C_DEVICE_INDEX_EXTENDED_DEVICE
#define	I3C_EVENT_STATUS_SUCCESS The transfer was completed as expected. More...
#define	I3C_EVENT_STATUS_PARITY A parity error was detected.
#define	I3C_EVENT_STATUS_FRAME A frame error was detected.
#define	I3C_EVENT_STATUS_ADDRESS_HEADER

	An Address Header error was detected.
#define	I3C_EVENT_STATUS_NACK The transfer was NACK'd.
#define	I3C_EVENT_STATUS_OVERFLOW A Receive FIFO overflow or Transmit FIFO underflow occurred.
#define	I3C_EVENT_STATUS_ABORTED_TO_MASTER In slave mode, the write transfer was ended via the 'T' bit.
#define	I3C_EVENT_STATUS_ABORTED In master mode, the transfer was aborted.
#define	I3C_EVENT_STATUS_NOT_SUPPORTED Operation is not supported.
#define	I3C_EVENT_STATUS_IBI_NACK_DISABLED An IBI was NACK'd and the a DISEC command was sent.

Enumerations

enum	i3c_bitrate_mode_t
enum	i3c_activity_state_t
enum	i3c_data_rate_setting_t
enum	i3c_clock_data_turnaround_t

Data Structure Documentation

◆ i3c_clock_stalling_t

struct i3c_clock_stalling_t		
Clock stalling settings.		
Data Fields		
uint32_t	assigned_address_phase_enable: 1	Enable Clock Stalling during the address phase of the ENTDAA

		command.
uint32_t	transition_phase_enable: 1	Enable Clock Stalling during the transition bit in read transfers.
uint32_t	parity_phase_enable: 1	Enable Clock Stalling during the parity bit period in write transfers.
uint32_t	ack_phase_enable: 1	Enable Clock Stalling during the ACK/NACK phase.
uint16_t	clock_stalling_time	The amount of time to stall the clock in I3C source clock ticks.

◆ i3c_bitrate_settings_t

struct i3c_bitrate_settings_t		
Bitrate settings for configuring the SCL clock frequency.		
Data Fields		
uint32_t	stdbr	The standard bitrate settings.
uint32_t	extbr	The extended bitrate settings.
i3c_clock_stalling_t	clock_stalling	Clock Stalling settings (See Master Clock Stalling in the MIPI I3C Specification v1.0).

◆ i3c_ibi_control_t

struct i3c_ibi_control_t		
Settings for controlling the drivers behavior in response to IBIs.		
Data Fields		
uint32_t	hot_join_acknowledge: 1	If false, NACK all Hot Join requests.
uint32_t	notify_rejected_hot_join_requests: 1	Notify the application when an IBI Hot-Join request has been NACK'd.
uint32_t	notify_rejected_mastership_requests: 1	Notify the application when an IBI Mastership request has been NACK'd.
uint32_t	notify_rejected_interrupt_requests: 1	Notify the application when an IBI Interrupt request has been NACK'd.

◆ i3c_slave_command_response_info_t

struct i3c_slave_command_response_info_t		
Default configuration settings for the slave response to Direct Get Common Command Codes.		
Data Fields		

bool	inband_interrupt_enable	Enable IBI interrupts. Slave Event Settings (See ENEC and DISEC in the MIPI I3C Specification v1.0).
bool	mastership_request_enable	Enable Mastership requests.
bool	hotjoin_request_enable	Enable Hot-Join requests.
i3c_activity_state_t	activity_state	Starting Activity State (See ENTASn in the MIPI I3C Specification v1.0).
uint16_t	write_length	Max Write Length (See SETMWL and GETMWL in the MIPI I3C Specification v1.0).
uint16_t	read_length	Max Read Length (See SETMRL and GETMRL in the MIPI I3C Specification v1.0).
uint8_t	ibi_payload_length	Number of bytes that will be written by an IBI (See SETMRL and GETMRL in the MIPI I3C Specification v1.0).
i3c_data_rate_setting_t	write_data_rate	Max Write Data Rate. Max Data Rate Settings (See GETMXDS in the MIPI I3C Specification v1.0).
i3c_data_rate_setting_t	read_data_rate	Max Read Data Rate.
i3c_clock_data_turnaround_t	clock_data_turnaround	Max Data Speed Turnaround.
bool	read_turnaround_time_enable	Enable transmission of the of the Max Read Max Read Turnaround Time.
uint32_t	read_turnaround_time	Max Read Turnaround Time.
uint8_t	oscillator_frequency	This byte represents the Slave's internal oscillator frequency in increments of 0.5 MHz (500kHz), up to 127.5 MHz. (See GETXTIME in the MIPI I3C Specification v1.1).
uint8_t	oscillator_inaccuracy	Oscillator inaccuracy in 0.5% increments of 0% up to 25.5% (See GETXTIME in the MIPI I3C Specification v1.1).
bool	hdr_ddr_support	HDR-DDR mode is supported.
bool	hdr_tsp_support	HDR-TSP mode is supported.
bool	hdr_tsl_support	HDR-TSL mode is supported.

◆ i3c_instance_ctrl_t

struct i3c_instance_ctrl_t	
Channel control block. DO NOT INITIALIZE. Initialization occurs when i3c_api_t::open is called.	
Public Member Functions	
i3c_slave_info_t current_slave_info	BSP_ALIGN_VARIABLE (4)
	The last i3c_slave_info_t read during ENTDAA.
Data Fields	
uint32_t	open
	Indicates whether the open() API has been successfully called.
R_I3C0_Type *	p_reg
	Base register for this channel.
volatile uint32_t	internal_state
	Used to track the current state of the driver.
uint8_t	current_command_code
	The current Common Command Code that is being transferred.
uint32_t	device_index
	The device index selected using i3c_api_t::deviceSelect.
i3c_bitrate_mode_t	device_bitrate_mode
	Runtime bitrate settings to use for the next transfer.
uint32_t	next_word
	The next word that will be written to the FIFO.

uint32_t	ibi_next_word
	The next word that will be written to the IBI FIFO.
i3c_write_buffer_descriptor_t	write_buffer_descriptor
	Buffer descriptor for keeping track of a write transfer.
i3c_read_buffer_descriptor_t	read_buffer_descriptor
	Buffer descriptor for keeping track of a read transfer.
i3c_read_buffer_descriptor_t	ibi_buffer_descriptor
	Buffer descriptor for keeping track of an IBI read/write transfer.
volatile uint32_t	read_transfer_count_final
	The total number of bytes read during a read transfer.
volatile uint32_t	ibi_transfer_count_final
	The total number of bytes read during an IBI transfer.
i3c_cfg_t const *	p_cfg
	A pointer to the configuration structure provided during open.

◆ i3c_extended_cfg_t

struct i3c_extended_cfg_t		
Extended configuration for r_i3c.		
Data Fields		
i3c_bitrate_settings_t	bitrate_settings	Bitrate settings configuring the frequency and duty cycle for SCL.

i3c_ibi_control_t	ibi_control	Configure the driver's behavior in response to IBIs.
uint32_t	bus_free_detection_time	The time in I3C reference clock ticks needed in order to detect the bus free condition (See "Bus Free Condition" in the MIPI I3C Specification v1.0).
uint32_t	bus_available_detection_time	The time in I3C reference clock ticks needed in order to detect the bus available condition (See "Bus Available Condition" in the MIPI I3C Specification v1.0).
uint32_t	bus_idle_detection_time	The time in I3C reference clock ticks needed in order to detect the bus idle condition (See "Bus Idle Condition" in the MIPI I3C Specification v1.0).
bool	timeout_detection_enable	Notify the application if SCL is stuck high or low.
i3c_slave_command_response_info_t	slave_command_response_info	Initial settings for configuring the slave's responses to received commands.
IRQn_Type	resp_irq	Response Queue Full IRQ number.
IRQn_Type	rx_irq	Receive FIFO Full IRQ number.
IRQn_Type	tx_irq	Transmit FIFO Empty IRQ number.
IRQn_Type	rcv_irq	Receive Status Queue Full IRQ number.
IRQn_Type	ibi_irq	IBI IRQ number.
IRQn_Type	eei_irq	EEI IRQ number.
uint8_t	ipl	Interrupt Priority for Resp, Rx, Tx, and RCV IRQs.
uint8_t	eei_ipl	Error and Event Interrupt Priority.

Macro Definition Documentation

◆ I3C_ERROR_RECOVERY_VERSION_1

#define I3C_ERROR_RECOVERY_VERSION_1
Support error recovery procedure for chip version 1. There are two different versions of the RX MCU error recovery procedure.

◆ I3C_DEVICE_INDEX_EXTENDED_DEVICE

#define I3C_DEVICE_INDEX_EXTENDED_DEVICE
Index for selecting the device defined in the extended address table.

◆ I3C_EVENT_STATUS_SUCCESS

#define I3C_EVENT_STATUS_SUCCESS
The transfer was completed as expected. Event Status Provided by the callback.

Enumeration Type Documentation

◆ i3c_bitrate_mode_t

enum i3c_bitrate_mode_t	
Bitrate settings that can be selected at run-time using i3c_api_t::deviceSelect.	
Enumerator	
I3C_BITRATE_MODE_I2C_STDBR	Use standard period setting for subsequent I2C transfers.
I3C_BITRATE_MODE_I2C_EXTBR	Use extended period setting for subsequent I2C transfers.
I3C_BITRATE_MODE_I3C_SDR0_STDBR	Use standard period setting for subsequent I3C SDR transfers.
I3C_BITRATE_MODE_I3C_SDR1_EXTBR	Use extended period setting for subsequent I3C SDR transfers.
I3C_BITRATE_MODE_I3C_SDR2_STDBR_X2	Use standard period setting x 2 for subsequent I3C SDR transfers.
I3C_BITRATE_MODE_I3C_SDR3_EXTBR_X2	Use extended period setting x 2 for subsequent I3C SDR transfers.
I3C_BITRATE_MODE_I3C_SDR4_EXTBR_X4	Use extended period setting x 4 for subsequent I3C SDR transfers.
I3C_BITRATE_MODE_I3C_HDR_DDR_STDBR	Use standard period setting for subsequent I3C HDR-DDR transfers.

◆ i3c_activity_state_t

enum i3c_activity_state_t	
Supported activity states for ENTASn Command (See ENTASn in the MIPI I3C Specification v1.0).	
Enumerator	
I3C_ACTIVITY_STATE_ENTAS0	Activity Interval (1 microsecond).
I3C_ACTIVITY_STATE_ENTAS1	Activity Interval (100 microseconds).
I3C_ACTIVITY_STATE_ENTAS2	Activity Interval (2 milliseconds).
I3C_ACTIVITY_STATE_ENTAS3	Activity Interval (50 milliseconds).

◆ i3c_data_rate_setting_t

enum i3c_data_rate_setting_t	
Maximum Sustained Data Rate for non-CCC messages sent by Master Device to Slave Device (See GETMXDS in the MIPI I3C Specification v1.0).	
Enumerator	
I3C_DATA_RATE_SETTING_FSCL_MAX	There is no data rate limit.
I3C_DATA_RATE_SETTING_8MHZ	The max sustained data rate is 8 Mhz.
I3C_DATA_RATE_SETTING_6MHZ	The max sustained data rate is 6 Mhz.
I3C_DATA_RATE_SETTING_4MHZ	The max sustained data rate is 4 Mhz.
I3C_DATA_RATE_SETTING_2MHZ	The max sustained data rate is 2 Mhz.

◆ i3c_clock_data_turnaround_t

enum i3c_clock_data_turnaround_t	
Clock to Data Turnaround Time (See GETMXDS in the MIPI I3C Specification v1.0).	
Enumerator	
I3C_CLOCK_DATA_TURNAROUND_8NS	Clock to turnaround time is 8 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_9NS	Clock to turnaround time is 9 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_10NS	Clock to turnaround time is 10 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_11NS	Clock to turnaround time is 11 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_12NS	Clock to turnaround time is 12 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_EXTENDED	Clock to turnaround time is greater than 12 nanoseconds.

Function Documentation

◆ R_I3C_Open()

fsp_err_t R_I3C_Open (i3c_ctrl_t *const *p_api_ctrl*, i3c_cfg_t const *const *p_cfg*)

Configure an I3C instance. Implements i3c_api_t::open.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was invalid.
FSP_ERR_ALREADY_OPEN	Open has already been called for this instance.
FSP_ERR_UNSUPPORTED	A selected feature is not supported with the current configuration.

◆ R_I3C_Enable()

fsp_err_t R_I3C_Enable (i3c_ctrl_t *const *p_api_ctrl*)

Enable the I3C device.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This instance is already enabled.

◆ R_I3C_DeviceCfgSet()

fsp_err_t R_I3C_DeviceCfgSet (i3c_ctrl_t *const *p_api_ctrl*, i3c_device_cfg_t const *const *p_device_cfg*)

Set the configuration for this device.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_UNSUPPORTED	The device cannot be a secondary master if master support is disabled.

◆ R_I3C_MasterDeviceTableSet()

fsp_err_t R_I3C_MasterDeviceTableSet (i3c_ctrl_t *const *p_api_ctrl*, uint32_t *device_index*, i3c_device_table_cfg_t const *const *p_device_table_cfg*)

Configure an entry in the master device table. This function is called in master mode in order to configure the devices on the I3C bus. It may also be called in slave mode when the slave receives the DEFSVLS command.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_UNSUPPORTED	Mastership requests must be rejected is slave support is disabled.

◆ R_I3C_SlaveStatusSet()

fsp_err_t R_I3C_SlaveStatusSet (i3c_ctrl_t *const *p_api_ctrl*, i3c_device_status_t *status*)

Set the status returned to the master in response to a GETSTATUS command.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	The instance is not in slave mode.
FSP_ERR_UNSUPPORTED	Slave support is disabled.

◆ R_I3C_DeviceSelect()

fsp_err_t R_I3C_DeviceSelect (i3c_ctrl_t *const *p_api_ctrl*, uint32_t *device_index*, uint32_t *bitrate_mode*)

In master mode, select the device for the next transfer. This function is not used in slave mode.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This operation is prohibited in slave mode.
FSP_ERR_UNSUPPORTED	Master support is disabled.

◆ R_I3C_DynamicAddressAssignmentStart()

fsp_err_t R_I3C_DynamicAddressAssignmentStart (i3c_ctrl_t *const *p_api_ctrl*, i3c_address_assignment_mode_t *address_assignment_mode*, uint32_t *starting_device_index*, uint32_t *device_count*)

Start the Dynamic Address Assignment Process. Implements i3c_api_t::dynamicAddressAssignmentStart.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL or invalid.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This operation is prohibited in slave mode.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_UNSUPPORTED	Master support is disabled.

◆ R_I3C_CommandSend()

fsp_err_t R_I3C_CommandSend (i3c_ctrl_t *const *p_api_ctrl*, i3c_command_descriptor_t const *const *p_command_descriptor*)

Send a broadcast or direct command to slave devices on the bus. Implements i3c_api_t::commandSend.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This driver is not in master mode.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes. If it is a read operation, the length also be a multiple of 4 bytes.
FSP_ERR_UNSUPPORTED	Master support must be enabled to call this function. Slave support must be enabled when sending the GETACCCR command.

◆ R_I3C_Write()

fsp_err_t R_I3C_Write (i3c_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_data*, uint32_t *length*, bool *restart*)

Set the write buffer for the transfer. In master mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start. Implements i3c_api_t::write.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This driver is disabled, or an invalid bitrate mode is selected.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes.

◆ R_I3C_Read()

fsp_err_t R_I3C_Read (i3c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t *length*, bool *restart*)

Set the read buffer for the transfer. In master mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start. Implements i3c_api_t::read.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This driver is disabled, or an invalid bitrate mode is selected.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes and the length must be a multiple of 4 bytes.

◆ R_I3C_IbiWrite()

fsp_err_t R_I3C_IbiWrite (i3c_ctrl_t *const *p_api_ctrl*, i3c_ibi_type_t *ibi_type*, uint8_t const *const *p_data*, uint32_t *length*)

Initiate an IBI write operation (This function is only used in slave mode).

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This function is only called in slave mode.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes.
FSP_ERR_UNSUPPORTED	Slave supoprt is disabled.

◆ R_I3C_IbiRead()

fsp_err_t R_I3C_IbiRead (i3c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t *length*)

Set the read buffer for storing received IBI data (This function is only used in master mode).

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This function is only called in master mode.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes and the length must be a multiple of 4 bytes.
FSP_ERR_UNSUPPORTED	Master support is disabled.

◆ R_I3C_Close()

fsp_err_t R_I3C_Close (i3c_ctrl_t *const *p_api_ctrl*)

Close the I3C instance. Implements i3c_api_t::close.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.

5.2.6.4 I2C Master (r_iic_b_master)

Modules » Connectivity

Functions

fsp_err_t	R_IIC_B_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t	R_IIC_B_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t	R_IIC_B_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

fsp_err_t	R_IIC_B_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_B_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t	R_IIC_B_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_B_MASTER_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t	R_IIC_B_MASTER_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)

Detailed Description

I2C Driver for the IIC/I3C peripheral on RX MCUs. This module implements the I2C Master Interface.

Overview

The I2C master on IIC/I3C HAL module supports transactions with an I2C Slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_iic_b_master

The following build time configurations are defined in fsp_cfg/r_iic_b_master_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled	Default (BSP)	If selected code for parameter checking is

	• Disabled		included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for Connectivity > I2C Master (r_iic_b_master)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_iic_b_master).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2c_master0	Module name.
Channel	Value must be a non-negative integer	0	Specify the IIC channel.
Rate	• Standard • Fast-mode • Fast-mode plus	Standard	Select the transfer rate. If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and duty cycle are printed in a comment in the generated iic_b_master_extended_cfg_t structure.
Custom Rate (bps)	Value must be a non-negative integer	0	Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode. Standard-mode: up to 100000; Fast-mode: up to 400000; Fast-mode plus: up to 1000000

Rise Time (ns)	Value must be a non-negative integer	120	Set the rise time (tr) in nanoseconds.
Fall Time (ns)	Value must be a non-negative integer	120	Set the fall time (tf) in nanoseconds.
Duty Cycle (%)	Value must be an integer between 0 and 100	50	Set the SCL duty cycle.
Slave Address	Value must be non-negative	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here.
Timeout Mode	• Short Mode • Long Mode	Short Mode	Select the timeout mode to detect bus hang.
Timeout during SCL Low	• Enabled • Disabled	Enabled	Select if the timeout can occur when SCL is held low for a duration longer than what is set in the timeout mode.
Callback	Name must be a valid C symbol	g_iic_b_master0_callback	A user callback function must be provided. This will be called from the interrupt service routine (ISR) upon IIC transaction completion reporting the transaction status.
Interrupt Priority Level	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts.

Clock Configuration

The I3C peripheral module uses the I3CCLK or PCLKA (based on the MCU) as its clock source for the bus clock. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the clocks are configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Pin Configuration

The I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

IIC Master Rate Calculation

- The RX Configuration editor calculates the internal baud-rate setting based on the configured transfer rate.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
  - Fast-mode Plus (Fm+) : up to 1 Mbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current I3CCLK/PCLKA (based on the MCU) settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

Enabling DTC with the IIC

- DTC transfer support is configurable and is disabled from the build by default. IIC driver provides two DTC instances for transmission and reception respectively. The DTC instances can be enabled individually during configuration.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Transfer (r_dtc)

Multiple Devices on the Bus

- A single IIC instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

Restart

- IIC master can hold the the bus after an I2C transaction by issuing a repeated start condition.

Examples

Basic Example

This is a basic example of minimal use of the r_iic_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_b_master_instance_ctrl_t g_i2c_device_ctrl_1;
i2c_master_cfg_t g_i2c_device_cfg_1 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_FAST,
    .slave         = I2C_SLAVE_EEPROM,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_1,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_b_master_cfg_extend
};
void i2c_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IIC_B_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
```

```
/* Send data to I2C slave */
   g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
   err = R_IIC_B_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers*/
while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
   }
if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
   {
      __BKPT(0);
   }
/* Read data back from the I2C slave */
   g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
   timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
   err = R_IIC_B_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers*/
while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;;
   }
if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
   {
      __BKPT(0);
   }
/* Verify the read data */
if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
   {
```

```
        __BKPT(0);
    }
}
```

Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IIC driver can be used to communicate with different slave devices which are on the same channel.

Note

The callback function from the first example applies to this example as well.

```
iic_b_master_instance_ctrl_t g_i2c_device_ctrl_2;
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_b_master_cfg_extend
};
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_B_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
```

```
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_IIC_B_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	iic_b_master_clock_settings_t

struct	iic_b_master_instance_ctrl_t
struct	iic_b_master_extended_cfg_t

Enumerations

enum	iic_b_master_timeout_mode_t
enum	iic_b_master_timeout_scl_low_t

Data Structure Documentation

◆ iic_b_master_clock_settings_t

struct iic_b_master_clock_settings_t		
I2C clock settings		
Data Fields		
uint8_t	cks_value	Internal Reference Clock Select.
uint8_t	brh_value	High-level period of SCL clock.
uint8_t	brl_value	Low-level period of SCL clock.
uint8_t	sdod_value	SDA Output Delay Counter.
bool	sdodcs_value	SDA Output Delay Clock Source.

◆ iic_b_master_instance_ctrl_t

struct iic_b_master_instance_ctrl_t
I2C control structure. DO NOT INITIALIZE.

◆ iic_b_master_extended_cfg_t

struct iic_b_master_extended_cfg_t		
R_IIC_B extended configuration		
Data Fields		
iic_b_master_timeout_mode_t	timeout_mode	Timeout Detection Time Select: Long Mode = 0 and Short Mode = 1.
iic_b_master_timeout_scl_low_t	timeout_scl_low	Allows timeouts to occur when SCL is held low.
iic_b_master_clock_settings_t	clock_settings	I2C Clock settings.
uint32_t	iic_clock_freq	I2C Clock frequency in Hz.
bool	smbus_operation	SMBus operation on I2C bus.

Enumeration Type Documentation

◆ iic_b_master_timeout_mode_t

enum iic_b_master_timeout_mode_t	
I2C Timeout mode parameter definition	
Enumerator	
IIC_B_MASTER_TIMEOUT_MODE_LONG	Timeout Detection Time Select: Long Mode -> TMOS = 0.
IIC_B_MASTER_TIMEOUT_MODE_SHORT	Timeout Detection Time Select: Short Mode -> TMOS = 1.

◆ iic_b_master_timeout_scl_low_t

enum iic_b_master_timeout_scl_low_t	
Enumerator	
IIC_B_MASTER_TIMEOUT_SCL_LOW_DISABLED	Timeout detection during SCL low disabled.
IIC_B_MASTER_TIMEOUT_SCL_LOW_ENABLED	Timeout detection during SCL low enabled.

Function Documentation

◆ R_IIC_B_MASTER_Open()

fsp_err_t R_IIC_B_MASTER_Open (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg*)

Opens the I2C device.

Return values

FSP_SUCCESS	Requested clock rate was set exactly.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Set the rate to fast mode plus on a channel which does not support it. 5. Invalid IRQ number assigned

◆ R_IIC_B_MASTER_Read()

fsp_err_t R_IIC_B_MASTER_Read (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl, p_dest or bytes is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_IN_USE	Bus busy condition. Another transfer was in progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

◆ R_IIC_B_MASTER_Write()

fsp_err_t R_IIC_B_MASTER_Write (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_IN_USE	Bus busy condition. Another transfer was in progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

◆ R_IIC_B_MASTER_Abort()

fsp_err_t R_IIC_B_MASTER_Abort (i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IIC peripheral into ready state.

Return values

FSP_SUCCESS	Channel was reset successfully.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

Note

A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.

◆ R_IIC_B_MASTER_SlaveAddressSet()

fsp_err_t R_IIC_B_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

FSP_SUCCESS	Address of the slave is set correctly.
FSP_ERR_ASSERTION	Pointer to control structure is NULL.
FSP_ERR_IN_USE	Another transfer was in-progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

◆ R_IIC_B_MASTER_Close()

fsp_err_t R_IIC_B_MASTER_Close (i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. May power down IIC peripheral. This function will safely terminate any in-progress I2C transfers.

Return values

FSP_SUCCESS	Device closed without issue.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

Note

A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.

◆ R_IIC_B_MASTER_CallbackSet()

fsp_err_t R_IIC_B_MASTER_CallbackSet (i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_IIC_B_MASTER_StatusGet()

fsp_err_t R_IIC_B_MASTER_StatusGet (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status*)

Provides driver status.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.

5.2.6.5 I2C Slave (r_iic_b_slave)

Modules » Connectivity

Functions

fsp_err_t	R_IIC_B_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg)
fsp_err_t	R_IIC_B_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_IIC_B_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)
fsp_err_t	R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_B_SLAVE_CallbackSet (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, i2c_slave_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the IIC/I3C peripheral on RX MCUs. This module implements the I2C Slave Interface.

Overview

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.
- Can accept 0x00 as slave address.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.
- Additional build-time features:
  - Optional (build time) DTC support for read and write respectively.

Configuration

Build Time Configurations for r_iic_b_slave

The following build time configurations are defined in fsp_cfg/r_iic_b_slave_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.

Configurations for Connectivity > I2C Slave (r_iic_b_slave)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Slave (r_iic_b_slave).

Configuration	Options	Default	Description
Interrupt Priority Level > Transmit, Receive, and Transmit End	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI, and TEI interrupts.
Interrupt Priority Level > Error	MCU Specific Options		Select the interrupt priority level. This is set for ERI interrupt.
Name	Name must be a valid C symbol	g_i2c_slave0	Module name.
Channel	Value must be a non-negative integer	0	Specify the IIC channel.
Rate	• Standard • Fast-mode • Fast-mode plus	Standard	Select the transfer rate. If the delay for the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated delay is printed in a comment in the generated iic_b_slave_extended_cfg_t structure.
Internal Reference Clock	• I2C Clock / 1 • I2C Clock / 2 • I2C Clock / 4 • I2C Clock / 8 • I2C Clock / 16	I2C Clock / 1	Select the internal reference clock for IIC slave. The internal reference clock is used only to determine the

	• I2C Clock / 32 • I2C Clock / 64 • I2C Clock / 128		clock frequency of the noise filter samples. I2C Clock can be either IICCLK or PCLKD based on the MCU.
Digital Noise Filter Stage Select	Refer to the RX Configuration tool for available options.	3-stage filter	Select the number of digital filter stages for IIC Slave.
Slave Address	Value must be non-negative	0x08	Specify the slave address.
General Call	• Enabled • Disabled	Disabled	Allows the slave to respond to general call address: 0x00.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the slave address mode.
Clock Stretching	• Enabled • Disabled	Disabled	Configure Clock Stretching.
Callback	Name must be a valid C symbol	g_iic_b_slave0_callback	A user callback function must be provided. This will be called from the interrupt service routine (ISR) to report I2C Slave transaction events and status.

Clock Configuration

The IIC/I3C peripheral module uses the I3CCLK or PCLKA (based on the MCU) as its clock source for the bus clock. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the clocks are configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Pin Configuration

The IIC/I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel must be enabled in the properties of the selected device.
- The interrupt priority of ERI can be set higher than or equal to the interrupt priorities of RXI, TXI and TEI.
  Note
  : During master-write slave-read type of operations if the slave device requires to perform clock stretching after the last data byte is received, a higher priority ERI will ensure that the ongoing

transaction is completed (by accepting the Stop/Restart condition from the master) before the next transaction is initiated.
: To support clock stretching (Holding SCL low after the falling edge of the 9th clock cycle), 'Clock Stretching' configuration must be enabled.

Callback

- A callback function must be provided which will be invoked for the cases below:
  - An I2C Master initiates a transmission or reception: I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
  - A Transmission or reception has been completed: I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
  - An I2C Master is requesting to read or write more data: I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
  - Error conditions: I2C_SLAVE_EVENT_ABORTED
  - An I2C Master initiates a general call by passing 0x00 as slave address: I2C_SLAVE_EVENT_GENERAL_CALL
- The callback arguments will contain information about the transaction status/events, bytes transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IIC slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

IIC Slave Callback Event	IIC Slave API expected to be called
I2C_SLAVE_EVENT_ABORTED	Handle event based on application
I2C_SLAVE_EVENT_RX_COMPLETE	Handle event based on application
I2C_SLAVE_EVENT_TX_COMPLETE	Handle event based on application
I2C_SLAVE_EVENT_RX_REQUEST	R_IIC_B_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master.
I2C_SLAVE_EVENT_TX_REQUEST	R_IIC_B_SLAVE_Write API
I2C_SLAVE_EVENT_RX_MORE_REQUEST	R_IIC_B_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master.
I2C_SLAVE_EVENT_TX_MORE_REQUEST	R_IIC_B_SLAVE_Write API
I2C_SLAVE_EVENT_GENERAL_CALL	R_IIC_B_SLAVE_Read

- If parameter checking is enabled and R_IIC_B_SLAVE_Read API is not called for I2C_SLAVE_EVENT_RX_REQUEST and/or I2C_SLAVE_EVENT_RX_MORE_REQUEST, the slave will send a NACK to the master and would eventually timeout.
- R_IIC_B_SLAVE_Write API is not called for I2C_SLAVE_EVENT_TX_REQUEST and/or I2C_SLAVE_EVENT_TX_MORE_REQUEST:
  - Slave timeout is less than Master timeout: The slave will timeout and release the bus causing the master to read 0xFF for every remaining byte.
  - Slave timeout is more than Master timeout: The master will timeout first followed by the slave.

IIC Slave Rate Calculation

- The RX Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current I2C Clock (I3CCLK or PCLKA based on the MCU) settings is calculated and used.

Limitations

- When 'Clock Stretching' configuration is enabled, the receive operation will not utilize the double buffer arrangement in hardware for a continuous read. This means that the read operation would happen in single byte units such that the active master would send the next byte only when the slave has read the current byte of data.
- When DTC is enabled, the clock frequency supply for DTC must be configured greater than the "Internal Reference Clock" of the slave.

Examples

Basic Example

This is a basic example of minimal use of the R_IIC_B_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_b_master_instance_ctrl_t g_i2c_master_ctrl;
i2c_master_cfg_t g_i2c_master_cfg =
{
    .channel       = I2C_MASTER_CHANNEL_2,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_master_callback, // Callback
    .p_context    = &g_i2c_master_ctrl,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend_standard_mode
};
iic_b_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
```

```
    .addr_mode = I2C_SLAVE_ADDR_MODE_7BIT,
    .p_callback = i2c_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
void i2c_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void i2c_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IIC_B_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
 /* Write to master */
       err = R_IIC_B_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else
```

```
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Pin connections:
  * Channel 0 SDA <--> Channel 2 SDA
  * Channel 0 SCL <--> Channel 2 SCL
  */
 /* Initialize the IIC Slave module */
    err = R_IIC_B_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the IIC Master module */
    err = R_IIC_B_MASTER_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_master_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    err = R_IIC_B_MASTER_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_B_SLAVE_Read API to service the Master
Write Request.
```

```
 */
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Write API to service the Master Read
Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
```

```
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	iic_b_slave_clock_settings_t
struct	iic_b_slave_extended_cfg_t

Data Structure Documentation

◆ iic_b_slave_clock_settings_t

struct iic_b_slave_clock_settings_t		
I2C clock settings		
Data Fields		
uint8_t	cks_value	Internal Reference Clock Select.
uint8_t	brl_value	Low-level period of SCL clock.
uint8_t	digital_filter_stages	Number of digital filter stages based on brl_value.

◆ iic_b_slave_extended_cfg_t

struct iic_b_slave_extended_cfg_t		
R_IIC_SLAVE extended configuration		
Data Fields		
iic_b_slave_clock_settings_t	clock_settings	I2C Clock settings.

Function Documentation

◆ R_IIC_B_SLAVE_Open()

fsp_err_t R_IIC_B_SLAVE_Open (i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg*)

Opens the I2C slave device.

Return values

FSP_SUCCESS	I2C slave device opened successfully.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_INVALID_ARGUMENT	Error interrupt priority is lower than Transmit, Receive and Transmit End interrupt priority
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Set the rate to fast mode plus on a channel which does not support it. 5. Invalid IRQ number assigned 6. transfer instance in p_cfg is NULL when DTC support enabled

◆ R_IIC_B_SLAVE_Read()

fsp_err_t R_IIC_B_SLAVE_Read (i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

Return values

FSP_SUCCESS	Function executed without issue
FSP_ERR_ASSERTION	p_api_ctrl, bytes or p_dest is NULL.
FSP_ERR_IN_USE	Another transfer was in progress.
FSP_ERR_NOT_OPEN	Device is not open.
FSP_ERR_INVALID_SIZE	Invalid size when reading data via DTC.

◆ R_IIC_B_SLAVE_Write()

fsp_err_t R_IIC_B_SLAVE_Write (i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*)

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
FSP_ERR_IN_USE	Another transfer was in progress.
FSP_ERR_NOT_OPEN	Device is not open.
FSP_ERR_INVALID_SIZE	Invalid size when writing data via DTC.

◆ R_IIC_B_SLAVE_Close()

fsp_err_t R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t *const *p_api_ctrl*)

Closes the I2C device.

Return values

FSP_SUCCESS	Device closed successfully.
FSP_ERR_NOT_OPEN	Device not opened.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.

◆ R_IIC_B_SLAVE_CallbackSet()

fsp_err_t R_IIC_B_SLAVE_CallbackSet (i2c_slave_ctrl_t *const *p_api_ctrl*, void(*)(i2c_slave_callback_args_t *) *p_callback*, void *const *p_context*, i2c_slave_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_slave_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.6.6 I2C Master (r_sci_b_i2c)

Modules » Connectivity

Functions

fsp_err_t	R_SCI_B_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_B_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t	R_SCI_B_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
fsp_err_t	R_SCI_B_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t	R_SCI_B_I2C_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_B_I2C_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)

Detailed Description

Driver for the SCI_B peripheral on RX MCUs. This module implements the I2C Master Interface.

Overview

The Simple I2C master on SCI_B HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_sci_b_i2c

The following build time configurations are defined in fsp_cfg/r_sci_b_i2c_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for Connectivity > I2C Master (r_sci_b_i2c)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_sci_b_i2c).

Configuration	Options	Default	Description

Name	Name must be a valid C symbol	g_i2c0	Module name.
Channel	Value must be an integer between 0 and 12	0	Select the SCI channel.
Slave Address	Value must be a hex value	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the address mode.
Rate	• Standard • Fast-mode	Standard	Select the I2C data rate. If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and SDA delay are printed in a comment in the generated sci_b_i2c_extended_cfg_t structure.
Custom Rate (bps)	Value must be a non-negative integer	0	Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode. Standard-mode: up to 100000; Fast-mode: up to 400000
SDA Output Delay (nano seconds)	Must be a valid non-negative integer with maximum configurable value of 300	300	Specify the SDA output delay in nanoseconds.
Noise Filter Clock Select	• The on-chip baud rate generator source clock divided by 1 • The on-chip baud rate generator source clock divided by 2 • The on-chip baud rate	The on-chip baud rate generator source clock divided by 1	Select the on-chip baud rate generator source clock division setting for the digital noise filter

	generator source clock divided by 4 • The on-chip baud rate generator source clock divided by 8		
Clock Source	MCU Specific Options		Select the clock source for the SCI I2C module.
Bit Rate Modulation	• Enable • Disable	Enable	Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave.
Callback	Name must be a valid C symbol	sci_b_i2c_master_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Interrupt Priority Level	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI (if used), TEI interrupts.
RX Interrupt Priority Level [Only used when DTC is enabled]	MCU Specific Options		Select the interrupt priority level. This is set for RXI only when DTC is enabled.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RX74M	SCICLK
RX74N	SCICLK

The actual I2C transfer rate can be derived from either SCICLK or the peripheral clock (PCLK)[1], and will be calculated and set by the tooling depending on the selected transfer rate and the SDA delay.

If the selected clock is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Note

1. See Figure "Block Diagram of SCI" in the hardware user manual for more information.

Pin Configuration

The SCI_B I2C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- Receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

SCI_B I2C Master Rate Calculation

- The RX Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and SDA Delay.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

Enabling DTC with the SCI_B I2C

- DTC transfer support is configurable and is disabled from the build by default. SCI_B I2C driver provides two DTC instances for transmission and reception respectively.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Transfer (r_dtc)

Multiple Devices on the Bus

- A single SCI_B I2C instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

Restart

- SCI_B_I2C can hold the the bus after an I2C transaction by issuing a repeated start

condition.

Examples

Basic Example

This is a basic example of minimal use of the r_sci_b_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
    err = R_SCI_B_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_B_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
```

```
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Multiple Slave devices on the same channel (bus)

This example demonstrates how a single SCI_B I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_B_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SCI_B_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
```

```
        __BKPT(0);
    }
}
```

Data Structures

struct	sci_b_i2c_clock_settings_t
struct	sci_b_i2c_instance_ctrl_t
struct	sci_b_i2c_extended_cfg_t

Enumerations

enum	sci_b_i2c_clock_source_t

Data Structure Documentation

◆ sci_b_i2c_clock_settings_t

struct sci_b_i2c_clock_settings_t		
I2C clock settings		
Data Fields		
bool	bitrate_modulation	Bit-rate Modulation Function enable or disable.
uint8_t	brr_value	Bit rate register settings.
uint8_t	clk_divisor_value	Clock Select settings.
uint8_t	mddr_value	Modulation Duty Register settings.
uint8_t	cycles_value	SDA Delay Output Cycles Select.
uint8_t	snfr_value	Noise Filter Setting Register value.
sci_b_i2c_clock_source_t	clock_source	Clock source (PCLK or SCISPICLK)

◆ sci_b_i2c_instance_ctrl_t

struct sci_b_i2c_instance_ctrl_t
I2C control structure. DO NOT INITIALIZE.

◆ sci_b_i2c_extended_cfg_t

struct sci_b_i2c_extended_cfg_t
SCI I2C extended configuration

Data Fields		
sci_b_i2c_clock_settings_t	clock_settings	I2C Clock settings.

Enumeration Type Documentation

◆ sci_b_i2c_clock_source_t

enum sci_b_i2c_clock_source_t
SCI clock source

Function Documentation

◆ R_SCI_B_I2C_Open()

fsp_err_t R_SCI_B_I2C_Open (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg*)

Opens the I2C device.

Return values

FSP_SUCCESS	Requested clock rate was set exactly.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Clock rate requested is greater than 400KHz 5. Invalid IRQ number assigned

◆ R_SCI_B_I2C_Close()

fsp_err_t R_SCI_B_I2C_Close (i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

FSP_SUCCESS	Device closed without issue.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_Read()

fsp_err_t R_SCI_B_I2C_Read (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	The parameter p_ctrl, p_dest is NULL, bytes is 0.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_Write()

fsp_err_t R_SCI_B_I2C_Write (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_ctrl, p_src is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_Abort()

fsp_err_t R_SCI_B_I2C_Abort (i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

FSP_SUCCESS	Transaction was aborted without issue.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_SlaveAddressSet()

fsp_err_t R_SCI_B_I2C_SlaveAddressSet (i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

FSP_SUCCESS	Address of the slave is set correctly.
FSP_ERR_ASSERTION	p_ctrl or address is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.
FSP_ERR_IN_USE	An I2C Transaction is in progress.

◆ R_SCI_B_I2C_CallbackSet()

fsp_err_t R_SCI_B_I2C_CallbackSet (i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_SCI_B_I2C_StatusGet()

fsp_err_t R_SCI_B_I2C_StatusGet (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status*)

Provides driver status.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.

5.2.6.7 SPI (r_sci_b_spi)

Modules » Connectivity

Functions

fsp_err_t	R_SCI_B_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_B_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_B_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_B_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_B_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_B_SPI_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_SPI_CalculateBitrate (uint32_t bitrate, sci_b_spi_clock_source_t clock_source, sci_b_spi_div_setting_t *sclk_div)

Detailed Description

Driver for the SCI peripheral on RX MCUs. This module implements the SPI Interface.

Overview

Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Select data sampling on leading edge, data change on trailing edge
    - CPHA=1 Select data change on leading edge, data sampling on trailing edge
  - MSB/LSB first
- Configurable bit rate
- DTC Support

- DMAC Support
- Callback Events
    - Transfer Complete
    - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

Configuration

Build Time Configurations for r_sci_b_spi

The following build time configurations are defined in fsp_cfg/r_sci_b_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Enable Support for using a Transfer API	• Enabled • Disabled	Enabled	Select whether to include support for DMAC or DTC transfers in the build.
FIFO Support	• Enabled • Disabled	Disabled	Enable FIFO support for the SCI_B_SPI module.

Configurations for Connectivity > SPI (r_sci_b_spi)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_sci_b_spi).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_sci_spi0	Module name.
Channel	Value must be a non-negative integer	0	Select the SCI channel.
Operating Mode	• Master • Slave	Master	Select the SPI operating mode.
Clock Phase	• Data sampling on odd edge, data variation on even edge • Data sampling on even edge, data variation on odd edge	Data sampling on odd edge, data variation on even edge	Select the clock edge to sample data.
Clock Polarity	• Low when idle • High when idle	Low when idle	Select clock level when idle.
Mode Fault Error	• Enable • Disable	Disable	Detect master/slave mode conflicts.

Bit Order	• MSB First • LSB First	MSB First	Select the data bit order.
Clock Source	• PCLK • SCISPICLK	PCLK	Select whether the peripheral clock (PCLK) or SCISPICLK is used for generating the SCK frequency.
Callback	Name must be a valid C symbol	sci_b_spi_callback	A user callback function that is called from the sci_b_spi interrupts when a transfer is completed or an error has occurred.
Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Transmit Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Transmit End Interrupt Priority	MCU Specific Options		Select the transmit end interrupt priority.
Error Interrupt Priority	MCU Specific Options		Select the error interrupt priority.
Bitrate	Value must be an integer greater than 0	8000000	Enter the desired bitrate. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The actual bitrate is printed in a comment in the generated sci_spi_extended_cfg_t structure.
Transmit FIFO Trigger Level	Refer to the RX Configuration tool for available options.	Disabled	Set to 1 to trigger DMAC/DTC TX FIFO reload immediately when each byte is transmitted. Set to 15 to trigger the reload when the FIFO is full or after 15 bit times with no data.
Receive Sampling Timing Adjustment (Master Only)	MCU Specific Options		Select the sampling timing delay of the RXDn pin (master

	mode only). Configuring this property may be required at high bitrates for the master to sample the received data accurately. This setting has no effect in slave mode.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RX74M	PCLKA
RX74N	PCLKA

The SCI peripheral uses the SCISPICLK/SCICLK or PCLKA for communication and PCLKA for internal operations. Both can be configured via the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module uses SCIn_MOSI, SCIn_MISO, SCIn_SPCK, and SCIn_SS pins to communicate with on board devices.

Note

It is recommended to configure the pins with high drive strength (or high-speed high if available) to maintain signal integrity, particularly at high bitrates. The drive strength can be set by opening the e2 studio pin configuration editor, selecting the pin, and configuring 'Drive Capacity'.

Usage Notes

Transfer Complete Event

The transfer complete event is triggered when all of the data has been transferred. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. This may also result in SPI_EVENT_ERR_READ_OVERFLOW errors.

In order to improve performance at high bit rates, consider the following recommendations:

- Enable DTC or DMAC. Note that DMAC will provide the best performance.
- Configure the Receive Sampling Timing Adjustment in the module properties (master mode only). For more information about Receive Sampling Timing Adjustment, refer to the diagram "Reception Sampling Timing Adjustment Operation in Clock Synchronous Mode

(Master)" in the HW User's Manual for the selected MCU.
- When using DMAC, set the RX DMAC channel to a higher priority (lower channel number) than the TX channel.
- When using DMAC, consider pausing operation of higher-priority DMAC instances in the project during SPI transfers or reassigning priorities to ensure high-speed SPI channels have top priority to avoid bus contention.

Transmit From RXI Interrupt

After every byte, the SCI SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. Whenever possible, the SCI SPI module handles both interrupts in the receive buffer full interrupt. This improves performance when the DTC or DMAC is not being used.

Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

Examples

Basic Example

This is a basic example of minimal use of the SCI_B_SPI module in an application.

```
static volatile bool g_transfer_complete = false;
static void r_sci_b_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
void sci_b_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
```

```
    err = R_SCI_B_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_B_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_B_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

Data Structures

struct	sci_b_spi_div_setting_t

Data Structure Documentation

◆ sci_b_spi_div_setting_t

struct sci_b_spi_div_setting_t
Settings for adjusting the SPI CLK.

Function Documentation

◆ R_SCI_B_SPI_Open()

fsp_err_t R_SCI_B_SPI_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SCI channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

Parameters

p_api_ctrl	Pointer to the control structure.
p_cfg	Pointer to a configuration structure.

Return values

FSP_SUCCESS	Channel initialized successfully.
FSP_ERR_ASSERTION	An input parameter is invalid or NULL.
FSP_ERR_ALREADY_OPEN	The instance has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel number is invalid.
FSP_ERR_UNSUPPORTED	Fifo mode is requested but a transfer interface is not used

◆ R_SCI_B_SPI_Read()

fsp_err_t R_SCI_B_SPI_Read (spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_dest	Pointer to the destination buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Read operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Bit width is not 8 bits • Length is equal to 0 • Pointer to destination is NULL • Transfer length is larger than supported by the transfer instances.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

◆ R_SCI_B_SPI_Write()

fsp_err_t R_SCI_B_SPI_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_src	Pointer to the source buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Length is equal to 0 • Bit width is not equal to 8 bits • Transfer length is larger than supported by the transfer instances.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

◆ R_SCI_B_SPI_WriteRead()

fsp_err_t R_SCI_B_SPI_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_src	Pointer to the source buffer.
	p_dest	Pointer to the destination buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Pointer to destination is NULL • Length is equal to 0 • Bit width is not equal to 8 bits • Transfer length is larger than supported by the transfer instances.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.

FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

◆ R_SCI_B_SPI_CallbackSet()

fsp_err_t R_SCI_B_SPI_CallbackSet (spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_SCI_B_SPI_Close()

fsp_err_t R_SCI_B_SPI_Close (spi_ctrl_t *const *p_api_ctrl*)

Disable the SCI channel and set the instance as not open. Implements spi_api_t::close.

Parameters

p_api_ctrl	Pointer to an opened instance.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

◆ R_SCI_B_SPI_CalculateBitrate()

fsp_err_t R_SCI_B_SPI_CalculateBitrate (uint32_t *bitrate*, sci_b_spi_clock_source_t *clock_source*, sci_b_spi_div_setting_t * *sclk_div*)

Calculate the register settings required to achieve the desired bitrate.

Parameters

[in]	bitrate	bitrate [bps]. For example, 250,000; 500,00; 2,500,000 (max), etc.
	clock_source	clock source (PCLKA or SCISPICLK) used for bit rate calculation.
	sclk_div	Pointer to sci_b_spi_div_setting_t used to configure baudrate settings.

Return values

FSP_SUCCESS	Baud rate is set successfully.
FSP_ERR_ASSERTION	Baud rate is not achievable.
FSP_ERR_INVALID_ARGUMENT	bitrate invalid.

Note

The application must pause for 1 bit time after the BRR register is loaded before transmitting/receiving to allow time for the clock to settle.

5.2.6.8 UART (r_sci_b_uart)

Modules » Connectivity

Functions

fsp_err_t	R_SCI_B_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_B_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_SCI_B_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes)
fsp_err_t	R_SCI_B_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting)

fsp_err_t	R_SCI_B_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info)
fsp_err_t	R_SCI_B_UART_Close (uart_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort)
fsp_err_t	R_SCI_B_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, sci_b_baud_setting_t *const p_baud_setting)
fsp_err_t	R_SCI_B_UART_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_B_UART_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *remaining_bytes)
fsp_err_t	R_SCI_B_UART_ReceiveSuspend (uart_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_UART_ReceiveResume (uart_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the SCI peripheral on RX MCUs. This module implements the UART Interface.

Overview

Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- CTS/RTS hardware flow control (with an associated pin)
- RS-485 Half Duplex driver support with external RS-485 transceiver
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels

Configuration

Build Time Configurations for r_sci_b_uart

The following build time configurations are defined in fsp_cfg/r_sci_b_uart_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
FIFO Support	• Enable • Disable	Disable	Enable FIFO support for the SCI_UART module.
DTC Support	• Enable • Disable	Disable	Enable DTC support for the SCI_UART module.
Flow Control Support	• Enable • Disable	Disable	Enable RS232 and RS-485 flow control support using a user provided pin.

Configurations for Connectivity > UART (r_sci_b_uart)

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_sci_b_uart).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_uart0	Module name.
General > Channel	Value must be a non-negative integer	0	Select the SCI channel.
General > Data Bits	• 8bits • 7bits • 9bits	8bits	Select the number of bits per word.
General > Parity	• None • Odd • Even	None	Select the parity mode.
General > Stop Bits	• 1bit • 2bits	1bit	Select the number of stop bits.
Baud > Baud Rate	Value must be an integer greater than 0	115200	Enter the desired baud rate. If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated sci_b_baud_setting_t structure.
Baud > Baud Rate Modulation	• Disabled • Enabled	Disabled	Enabling baud rate modulation reduces the percent error of the

			actual baud rate with respect to the requested baud rate. It does this by modulating the number of cycles per clock, so some bits are slightly longer than others.
Baud > Max Error (%)	Must be a valid non-negative integer with a maximum configurable value of 100	5	Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings. NOTE: The baud calculation does not show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated sci_b_baud_setting_t structure.
Flow Control > CTS/RTS Selection	MCU Specific Options		Select either CTS or RTS function on the CTSn_RTSn pin of SCI channel n or select CTS function on CTSn pin and RTS function on CTSn_RTSn pin of SCI channel n (Available on selected MCUs and channels).
Flow Control > Software RTS Port	Refer to the RX Configuration tool for available options.	Disabled	Specify the flow control pin port for the MCU.
Flow Control > Software RTS Pin	Refer to the RX Configuration tool for available options.	Disabled	Specify the flow control pin for the MCU.
Extra > RS-485 > DE Pin	• Disable • Enable	Disable	Enable or disable the DE pin for use in RS-485 half-duplex mode.
Extra > RS-485 > DE Pin Polarity	• Active Low • Active High	Active High	Select the polarity of the DE pin.

Extra > RS-485 > DE Pin Assertion Time	Must be a valid integer greater than 0 and less than or equal to 31.	1	Configure the time between assertion of the DE pin and the start of a write transfer. The time is specified in multiples of the SCI base clock period.
Extra > RS-485 > DE Pin Negation Time	Must be a valid integer greater than 0 and less than or equal to 31.	1	Configure the time between the end of a write transfer and the negation of the DE pin. The time is specified in multiples of the SCI base clock period.
Extra > Clock Source	• Internal Clock • Internal Clock With Output on SCK • External Clock 8x baud rate • External Clock 16x baud rate	Internal Clock	Selection of the clock source to be used in the baud-rate clock generator. When internal clock is used the baud rate can be output on the SCK pin.
Extra > Start bit detection	• Falling Edge • Low Level	Falling Edge	Start bit detected as falling edge or low level.
Extra > Noise Filter	• Enable • Disable	Disable	Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits.
Extra > Receive FIFO Trigger Level	Refer to the RX Configuration tool for available options.	Max	Unused if the channel has no FIFO or if DTC is used for reception. Set to One to get a callback immediately when each byte is received. Set to Max to get a callback when FIFO is full or after 15 bit times with no data (fewer interrupts).
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Interrupts > Receive	MCU Specific Options		Select the receive

Interrupt Priority		interrupt priority.
Interrupts > Transmit Data Empty Interrupt Priority	MCU Specific Options	Select the transmit interrupt priority.
Interrupts > Transmit End Interrupt Priority	MCU Specific Options	Select the transmit end interrupt priority.
Interrupts > Error Interrupt Priority	MCU Specific Options	Select the error interrupt priority.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RX74M	SCICLK
RX74N	SCICLK

The clock source for the baud-rate clock generator can be selected from the internal clock, the external clock times 8 or the external clock times 16. The external clock is supplied to the SCK pin.

Pin Configuration

This module uses TXD and RXD to communicate to external devices. CTS or RTS can be controlled by the hardware. CTS or RTS or both (CTS and RTS) can be controlled by the hardware When the internal clock is the source for the baud-rate generator the SCK pin can be used to output a clock with the same frequency as the bit rate.

Usage Notes

- When configured for Hardware CTS and Software RTS the configured flow control pin will be used for RTS. The pin will be set high inside of the receive ISR while data is being read. It will be set low when all data is read.
- When configured for Hardware CTS and Hardware RTS the CSTn_RTSn pin will be used for RTS function and the CTSn pin will be used for CTS function on channel n.
- The driver will follow correct hardware flow control function when CTSn_RTSn pin is connected to CTSn pin when "Hardware CTS and Hardware RTS" flow control is selected. The data will still be transferred when CTSn_RTSn and CTSn are disconnected as the CTSn pin is internally pulled low on the hardware when CTSn pin is configured as a peripheral pin for SCI module. Do not configure CTSn pin if the hardware flow control is not desired.

Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.

DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to

R_SCI_B_UART_Read(). The logic required to combine bytes received through R_SCI_B_UART_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

Examples

SCI UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_b_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SCI_B_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
```

```
    }
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                    g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                    uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                    *p_dest             = (uint16_t) p_args->data;
                    g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
```

```
        }
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
        {
              g_transfer_complete = 1;
 break;
        }
 default:
        {
        }
    }
}
```

SCI UART Baud Set Example

```
#define SCI_B_UART_BAUDRATE_19200 (19200)
#define SCI_B_UART_BAUDRATE_ERROR_PERCENT_5 (5000)
void r_sci_b_uart_baud_example (void)
{
 sci_b_baud_setting_t baud_setting;
    uint32_t             baud_rate                = SCI_B_UART_BAUDRATE_19200;
 bool                enable_bitrate_modulation = false;
    uint32_t             error_rate_x_1000         =
SCI_B_UART_BAUDRATE_ERROR_PERCENT_5;
 fsp_err_t err = R_SCI_B_UART_BaudCalculate(baud_rate, enable_bitrate_modulation,
error_rate_x_1000, &baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	sci_b_uart_instance_ctrl_t
struct	sci_b_baud_setting_t

struct	sci_b_uart_rs485_setting_t
struct	sci_b_uart_extended_cfg_t

Enumerations

enum	sci_b_clk_src_t
enum	sci_b_uart_flow_control_t
enum	sci_b_uart_rx_fifo_trigger_t
enum	sci_b_uart_start_bit_detect_t
enum	sci_b_uart_noise_cancellation_t
enum	sci_b_uart_rs485_enable_t
enum	sci_b_uart_rs485_de_polarity_t

Data Structure Documentation

◆ sci_b_uart_instance_ctrl_t

struct sci_b_uart_instance_ctrl_t
UART instance control block.

◆ sci_b_baud_setting_t

struct sci_b_baud_setting_t
Register settings to achieve a desired baud rate and modulation duty.

◆ sci_b_uart_rs485_setting_t

struct sci_b_uart_rs485_setting_t		
Configuration settings for controlling the DE signal for RS-485.		
Data Fields		
sci_b_uart_rs485_enable_t	enable	Enable the DE signal.
sci_b_uart_rs485_de_polarity_t	polarity	DE signal polarity.
uint8_t	assertion_time: 5	Time in baseclock units after assertion of the DE signal and before the start of the write transfer.
uint8_t	negation_time: 5	Time in baseclock units after the end of a write transfer and before the DE signal is negated.

◆ sci_b_uart_extended_cfg_t

struct sci_b_uart_extended_cfg_t		
UART on SCI device Configuration		
Data Fields		
sci_b_clk_src_t	clock	The source clock for the baud-rate generator. If internal optionally output baud rate on SCK.
sci_b_uart_start_bit_detect_t	rx_edge_start	Start reception on falling edge.
sci_b_uart_noise_cancellation_t	noise_cancel	Noise cancellation setting.
sci_b_baud_setting_t *	p_baud_setting	Register settings for a desired baud rate.
sci_b_uart_rx_fifo_trigger_t	rx_fifo_trigger	Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used.
bsp_io_port_pin_t	flow_control_pin	UART Driver Enable pin.
sci_b_uart_flow_control_t	flow_control	CTS/RTS function of the SSn pin.
sci_b_uart_rs485_setting_t	rs485_setting	RS-485 settings.

Enumeration Type Documentation

◆ sci_b_clk_src_t

enum sci_b_clk_src_t	
Enumeration for SCI clock source	
Enumerator	
SCI_B_UART_CLOCK_INT	Use internal clock for baud generation.
SCI_B_UART_CLOCK_INT_WITH_BAUDRATE_OUTPUT	Use internal clock for baud generation and output on SCK.
SCI_B_UART_CLOCK_EXT8X	Use external clock 8x baud rate.
SCI_B_UART_CLOCK_EXT16X	Use external clock 16x baud rate.

◆ sci_b_uart_flow_control_t

enum sci_b_uart_flow_control_t	
UART flow control mode definition	
Enumerator	
SCI_B_UART_FLOW_CONTROL_RTS	Use CTSn_RTSn pin for RTS.
SCI_B_UART_FLOW_CONTROL_CTS	Use CTSn_RTSn pin for CTS.
SCI_B_UART_FLOW_CONTROL_HARDWARE_CTSRTS	Use CTSn pin for CTS, CTSn_RTSn pin for RTS.
SCI_B_UART_FLOW_CONTROL_CTSRTS	Use SCI pin for CTS, external pin for RTS.

◆ sci_b_uart_rx_fifo_trigger_t

enum sci_b_uart_rx_fifo_trigger_t	
Receive FIFO trigger configuration.	
Enumerator	
SCI_B_UART_RX_FIFO_TRIGGER_1	Callback after each byte is received without buffering.
SCI_B_UART_RX_FIFO_TRIGGER_2	Callback when FIFO having 2 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_3	Callback when FIFO having 3 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_4	Callback when FIFO having 4 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_5	Callback when FIFO having 5 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_6	Callback when FIFO having 6 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_7	Callback when FIFO having 7 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_8	Callback when FIFO having 8 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_9	Callback when FIFO having 9 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_10	Callback when FIFO having 10 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_11	Callback when FIFO having 11 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_12	Callback when FIFO having 12 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_13	Callback when FIFO having 13 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_14	Callback when FIFO having 14 bytes.
SCI_B_UART_RX_FIFO_TRIGGER_MAX	Callback when FIFO is full or after 15 bit times with no data (fewer interrupts)

◆ sci_b_uart_start_bit_detect_t

enum sci_b_uart_start_bit_detect_t	
Asynchronous Start Bit Edge Detection configuration.	
Enumerator	
SCI_B_UART_START_BIT_LOW_LEVEL	Detect low level on RXDn pin as start bit.
SCI_B_UART_START_BIT_FALLING_EDGE	Detect falling level on RXDn pin as start bit.

◆ sci_b_uart_noise_cancellation_t

enum sci_b_uart_noise_cancellation_t	
Noise cancellation configuration.	
Enumerator	
SCI_B_UART_NOISE_CANCELLATION_DISABLE	Disable noise cancellation.
SCI_B_UART_NOISE_CANCELLATION_ENABLE	Enable noise cancellation.

◆ sci_b_uart_rs485_enable_t

enum sci_b_uart_rs485_enable_t	
RS-485 Enable/Disable.	
Enumerator	
SCI_B_UART_RS485_DISABLE	RS-485 disabled.
SCI_B_UART_RS485_ENABLE	RS-485 enabled.

◆ sci_b_uart_rs485_de_polarity_t

enum sci_b_uart_rs485_de_polarity_t	
The polarity of the RS-485 DE signal.	
Enumerator	
SCI_B_UART_RS485_DE_POLARITY_HIGH	The DE signal is high when a write transfer is in progress.
SCI_B_UART_RS485_DE_POLARITY_LOW	The DE signal is low when a write transfer is in progress.

Function Documentation

◆ R_SCI_B_UART_Open()

fsp_err_t R_SCI_B_UART_Open (uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg*)

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function. Implements uart_api_t::open

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The requested channel does not exist on this MCU.
FSP_ERR_INVALID_ARGUMENT	Flow control is enabled but flow control pin is not defined.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

◆ R_SCI_B_UART_Read()

fsp_err_t R_SCI_B_UART_Read (uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

Return values

FSP_SUCCESS	Data reception successfully ends.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
FSP_ERR_INVALID_ARGUMENT	Destination address or data size is not valid for 9-bit mode.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_IN_USE	A previous read operation is still in progress.
FSP_ERR_UNSUPPORTED	SCI_B_UART_CFG_RX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

If 9-bit data length is specified at R_SCI_B_UART_Open call, p_dest must be aligned 16-bit boundary.

◆ R_SCI_B_UART_Write()

fsp_err_t R_SCI_B_UART_Write (uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*)

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

Return values

FSP_SUCCESS	Data transmission finished successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
FSP_ERR_INVALID_ARGUMENT	Source address or data size is not valid for 9-bit mode.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_IN_USE	A UART transmission is in progress
FSP_ERR_UNSUPPORTED	SCI_B_UART_CFG_TX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

If 9-bit data length is specified at R_SCI_B_UART_Open call, p_src must be aligned on a 16-bit boundary.

◆ R_SCI_B_UART_BaudSet()

fsp_err_t R_SCI_B_UART_BaudSet (uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting*)

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a sci_b_baud_setting_t structure. Implements uart_api_t::baudSet

Warning

This terminates any in-progress transmission.

Return values

FSP_SUCCESS	Baud rate was successfully changed.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL or the UART is not configured to use the internal clock.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_B_UART_InfoGet()

fsp_err_t R_SCI_B_UART_InfoGet (uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info*)

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

Return values

FSP_SUCCESS	Information stored in provided p_info.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_B_UART_Close()

fsp_err_t R_SCI_B_UART_Close (uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_B_UART_Abort()

fsp_err_t R_SCI_B_UART_Abort (uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort*)

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

Return values

FSP_SUCCESS	UART transaction aborted successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

◆ R_SCI_B_UART_BaudCalculate()

fsp_err_t R_SCI_B_UART_BaudCalculate (uint32_t *baudrate*, bool *bitrate_modulation*, uint32_t *baud_rate_error_x_1000*, sci_b_baud_setting_t *const *p_baud_setting*)

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

Parameters

[in]	baudrate	Baud rate [bps]. For example, 19200, 57600, 115200, etc.
[in]	bitrate_modulation	Enable bitrate modulation
[in]	baud_rate_error_x_1000	Max baud rate error. At most <baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 15000 (15%).
[out]	p_baud_setting	Baud setting information stored here if successful

Return values

FSP_SUCCESS	Baud rate is set successfully
FSP_ERR_ASSERTION	Null pointer
FSP_ERR_INVALID_ARGUMENT	Baud rate is '0', error in calculated baud rate is larger than requested max error, or requested max error in baud rate is larger than 15%.

◆ R_SCI_B_UART_CallbackSet()

fsp_err_t R_SCI_B_UART_CallbackSet (uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void *const *p_context*, uart_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements uart_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_SCI_B_UART_ReadStop()

fsp_err_t R_SCI_B_UART_ReadStop (uart_ctrl_t *const *p_api_ctrl*, uint32_t * *remaining_bytes*)

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

Return values

FSP_SUCCESS	UART transaction aborted successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

◆ R_SCI_B_UART_ReceiveSuspend()

fsp_err_t R_SCI_B_UART_ReceiveSuspend (uart_ctrl_t *const *p_api_ctrl*)

Suspend Reception

Return values

FSP_ERR_UNSUPPORTED	Functionality not supported by this driver instance

◆ R_SCI_B_UART_ReceiveResume()

fsp_err_t R_SCI_B_UART_ReceiveResume (uart_ctrl_t *const *p_api_ctrl*)

Resume Reception

Return values

FSP_ERR_UNSUPPORTED	Functionality not supported by this driver instance

5.2.6.9 SPI (r_spi_b)

Modules » Connectivity

Functions

fsp_err_t	R_SPI_B_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SPI_B_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_B_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_B_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_B_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SPI_B_CalculateBitrate (uint32_t bitrate, spi_b_clock_source_t clock_source, rspck_div_setting_t *spck_div)
fsp_err_t	R_SPI_B_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the SPI peripheral on RX MCUs. This module implements the SPI Interface.

Overview

Features

- Standard SPI Modes
  - Master or Slave Mode
  - 3-Wire (clock synchronous) or 4-Wire (SPI) Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data is sampled at an odd edge and changes at an even edge.
    - CPHA=1 Data changes at an odd edge and is sampled at an even edge.
  - MSB/LSB first
  - 8- to 32-Bit data frames
    - Hardware endian swap in 16-Bit and 32-Bit mode
  - SSL level keep (burst transfer) supported
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- DMAC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SPI shift register is copied to the data register before

previous data was read)
- TX Underrun Error (No data to load into shift register for transmitting)
- Parity Error (When parity is enabled and a parity error is detected)

Configuration

Build Time Configurations for r_spi_b

The following build time configurations are defined in fsp_cfg/r_spi_b_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Enable Support for using a transfer API	• Enabled • Disabled	Enabled	If enabled, transfer instances will be included in the build for both transmission and reception.
Enable Transmitting from RXI Interrupt	• Enabled • Disabled	Disabled	If enabled, all operations will be handled from the RX (receive) interrupt. This setting only provides a performance boost when DTC is not used. In addition, Transmit Only mode is not supported when this configuration is enabled.

Configurations for Connectivity > SPI (r_spi_b)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_spi_b).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_spi0	Module name.
Channel	Value must be a non-negative integer	0	Select the SPI channel.
Receive Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Transmit Buffer Empty Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.

Transfer Complete Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Error Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Operating Mode	• Master • Slave	Master	Select the SPI operating mode.
Clock Phase	• Data sampling on odd edge, data variation on even edge • Data sampling on even edge, data variation on odd edge	Data sampling on odd edge, data variation on even edge	Select the clock edge to sample data.
Clock Polarity	• Low when idle • High when idle	Low when idle	Select clock level when idle.
Mode Fault Error	• Enable • Disable	Disable	Detect master/slave mode conflicts.
Bit Order	• MSB First • LSB First	MSB First	Select the data bit order.
Callback	Name must be a valid C symbol	spi_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
SPI Mode	• SPI Operation • Clock Synchronous Operation	Clock Synchronous Operation	Select the clock sync mode.
Communication Mode Select	• Full Duplex • Transmit Only	Full Duplex	Select Full Duplex or Transmit Only Mode.
Slave Select Polarity	• Active Low • Active High	Active Low	Select the slave select active level.
Select SSL(Slave Select)	• SSL0 • SSL1 • SSL2 • SSL3	SSL0	Select which slave to use.
MOSI Idle State	• MOSI Idle Value Fixing Disable • MOSI Idle Value Fixing Low • MOSI Idle Value	MOSI Idle Value Fixing Disable	Select the MOSI idle level if MOSI idle is enabled.

	Fixing High		
Parity Mode	• Disabled • Odd • Even	Disabled	Select the parity mode if parity is enabled.
Byte Swapping	• Disable • Enable	Disable	Select the byte swap mode for 16/32-Bit Data Frames.
Clock Source	MCU Specific Options		Select the clock source for communication.
Bitrate	Value must be an integer greater than 0	16000000	Enter the desired bitrate, change the bitrate to a value supported by MCU. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The theoretical bitrate is printed in a comment in the generated spi_extended_cfg_t structure.
Clock Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles before each data frame.
SSL Negation Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles after each data frame.
Next Access Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles between each data frame.

Clock Configuration

The SPI peripheral uses the SCISPICLK for communication and PCLKB for internal operations. Both can be configured via the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module uses MOSI, MISO, RSPCK, and SSL pins to communicate with on board devices.

Note

At high bitrates it may be necessary to configure the pins with IOPORT_CFG_DRIVE_HIGH to maintain signal integrity.

Usage Notes

Performance

At high bitrates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in TX Underrun and RX Overflow errors.

In order to improve performance at high bitrates, it is recommended that the instance be configured to service transfers using the DTC or DMAC.

Another way to improve performance is to transfer the data in 16/32 bit wide data frames when possible. A typical use-case where this is possible is when reading/writing to a block device.

When performing CPU-driven transfers, the transfer time may not decrease as expected when increasing the SPI bitrate. To improve throughput, it is recommended to enable I-Cache and D-Cache along with "Transmit from RXI Interrupt".

Transmit From RXI Interrupt

After every data frame the SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. It is possible to configure the driver to handle transmit buffer empty interrupts in the receive buffer full ISR. This only improves performance when the DTC nor DMAC is not being used.

Note

Configuring the module to use RX DTC/DMAC instance without also providing a TX DTC/DMAC instance results in an invalid configuration when RXI transmit is enabled.
Transmit Only mode is not supported when Transmit from RXI is enabled.

Clock Auto-Stopping

In master mode, if the Receive Buffer Full Interrupts are not handled fast enough, instead of generating a RX Overflow error, the last clock cycle will be stretched until the receive buffer is read.

Parity Mode

When parity mode is configured, the LSB of each data frame is used as a parity bit. When odd parity is selected, the LSB is set such that there are an odd number of ones in the data frame. When even parity is selected, the LSB is set such that there are an even number of ones in the data frame.

Limitations

Developers should be aware of the following limitations when using the SPI:

- In master mode, the driver will only configure 4-Wire mode if the device supports SSL Level Keeping (SSLKP bit in SPCMD0) and will return FSP_ERR_UNSUPPORTED if configured for 4-Wire mode on devices without SSL Level Keeping. Without SSL Level Keeping, the SSL pin is toggled after every data frame. In most cases this is not desirable behavior so it is recommended that the SSL pin be driven in software if SSL Level Keeping is not present on the device.
- The module does not support communicating with multiple slaves using different SSL pins. In order to achieve this, the module must either be closed and re-opened to change the SSL pin or drive SSL in software. It is recommended that SSL be driven in software when controlling multiple slave devices.
- The SPI peripheral has a minimum 3 SPI CLK delay between each data frame.
- The behavior for Byte Swap operation is not guaranteed for data frames other than 8-bit, 16-bit and 32bit.
- Full-duplex transmission and reception (writeRead) may experience RX overflows at very high bitrates when using DTC or DMAC. In this case, use half-duplex (write and read) functions.
- Slave mode prohibits frame widths of 17 or 18 bits. Operating with these lengths leads to unpredictable behavior, including out-of-order data frame reception at the master.

Examples

Basic Example

This is a basic example of minimal use of the SPI in an application.

```
static volatile bool g_transfer_complete = false;
void spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start a write/read transfer */
    err = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
```

```
        ;
    }
}
static void r_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
        g_transfer_complete = true;
    }
}
```

Driving Software Slave Select Line

This is an example of communicating with multiple slave devices by asserting SSL in software.

```
void spi_software_ssl_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                 = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
```

```
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                 = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

Configuring the SPI Clock Divider Registers

This example demonstrates how to set the SPI clock divisors at runtime.

```
void spi_bitrate_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_spi_cfg.p_extend = &g_spi_extended_cfg;
 /* Configure SPI Clock divider to achieve largest bitrate less than or equal to the
desired bitrate. */
```

```
    err = R_SPI_B_CalculateBitrate(BITRATE, SPI_B_CLOCK_SOURCE_SCISPICLK,
&(g_spi_extended_cfg.spck_div));
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rspck_div_setting_t
struct	spi_b_extended_cfg_t
struct	spi_b_instance_ctrl_t

Enumerations

enum	spi_b_ssl_mode_t
enum	spi_b_communication_t
enum	spi_b_ssl_polarity_t
enum	spi_b_ssl_select_t
enum	spi_b_mosi_idle_value_fixing_t
enum	spi_b_parity_t
enum	spi_b_byte_swap_t
enum	spi_b_delay_count_t
enum	spi_b_clock_source_t

Data Structure Documentation

◆ rspck_div_setting_t

struct rspck_div_setting_t		
SPI Clock Divider settings.		
Data Fields		
uint8_t	spbr	SPBR register setting.

uint8_t	brdv: 2	BRDV setting in SPCMD0.

◆ spi_b_extended_cfg_t

struct spi_b_extended_cfg_t		
Extended SPI interface configuration		
Data Fields		
spi_b_ssl_mode_t	spi_clksyn	Select SPI or Clock Synchronous mode operation.
spi_b_communication_t	spi_comm	Select full-duplex or transmit-only communication.
spi_b_ssl_polarity_t	ssl_polarity	Select SSLn signal polarity.
spi_b_ssl_select_t	ssl_select	Select which slave to use: 0-SSL0, 1-SSL1, 2-SSL2, 3-SSL3.
spi_b_mosi_idle_value_fixing_t	mosi_idle	Select MOSI idle fixed value and selection.
spi_b_parity_t	parity	Select parity and enable/disable parity.
spi_b_byte_swap_t	byte_swap	Select byte swap mode.
spi_b_clock_source_t	clock_source	Communication clock source (TCLK).
rspck_div_setting_t	spck_div	Register values for configuring the SPI Clock Divider.
spi_b_delay_count_t	spck_delay	SPI Clock Delay Register Setting.
spi_b_delay_count_t	ssl_negation_delay	SPI Slave Select Negation Delay Register Setting.
spi_b_delay_count_t	next_access_delay	SPI Next-Access Delay Register Setting.

◆ spi_b_instance_ctrl_t

struct spi_b_instance_ctrl_t	
Channel control block. DO NOT INITIALIZE. Initialization occurs when spi_api_t::open is called.	
Data Fields	
uint32_t	open
	Indicates whether the open() API has been successfully called.
spi_cfg_t const *	p_cfg
	Pointer to instance configuration.

R_SPI_B0_Type *	p_regs
	Base register for this channel.
void const *	p_tx_data
	Buffer to transmit.
void *	p_rx_data
	Buffer to receive.
uint32_t	tx_count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
uint32_t	rx_count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
uint32_t	count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
spi_bit_width_t	bit_width
	Bits per Data frame (8-bit, 16-bit, 32-bit)

Enumeration Type Documentation

◆ spi_b_ssl_mode_t

enum spi_b_ssl_mode_t	
3-Wire or 4-Wire mode.	
Enumerator	
SPI_B_SSL_MODE_SPI	SPI operation (4-wire method)
SPI_B_SSL_MODE_CLK_SYN	Clock Synchronous operation (3-wire method)

◆ spi_b_communication_t

enum spi_b_communication_t	
Transmit Only (Half Duplex), or Full Duplex.	
Enumerator	
SPI_B_COMMUNICATION_FULL_DUPLEX	Full-Duplex synchronous serial communication.
SPI_B_COMMUNICATION_TRANSMIT_ONLY	Transit only serial communication.

◆ spi_b_ssl_polarity_t

enum spi_b_ssl_polarity_t	
Slave Select Polarity.	
Enumerator	
SPI_B_SSLP_LOW	SSLP signal polarity active low.
SPI_B_SSLP_HIGH	SSLP signal polarity active high.

◆ spi_b_ssl_select_t

enum spi_b_ssl_select_t	
The Slave Select Line	
Enumerator	
SPI_B_SSL_SELECT_SSL0	Select SSL0.
SPI_B_SSL_SELECT_SSL1	Select SSL1.
SPI_B_SSL_SELECT_SSL2	Select SSL2.
SPI_B_SSL_SELECT_SSL3	Select SSL3.

◆ spi_b_mosi_idle_value_fixing_t

enum spi_b_mosi_idle_value_fixing_t	
MOSI Idle Behavior.	
Enumerator	
SPI_B_MOSI_IDLE_VALUE_FIXING_DISABLE	MOSI output value=value set in MOIFV bit.
SPI_B_MOSI_IDLE_VALUE_FIXING_LOW	MOSIn level low during MOSI idling.
SPI_B_MOSI_IDLE_VALUE_FIXING_HIGH	MOSIn level high during MOSI idling.

◆ spi_b_parity_t

enum spi_b_parity_t	
Parity Mode	
Enumerator	
SPI_B_PARITY_MODE_DISABLE	Disable parity.
SPI_B_PARITY_MODE_ODD	Select even parity.
SPI_B_PARITY_MODE_EVEN	Select odd parity.

◆ spi_b_byte_swap_t

enum spi_b_byte_swap_t	
Byte Swapping Enable/Disable.	
Enumerator	
SPI_B_BYTE_SWAP_DISABLE	Disable Byte swapping for 16/32-Bit transfers.
SPI_B_BYTE_SWAP_ENABLE	Enable Byte swapping for 16/32-Bit transfers.

◆ spi_b_delay_count_t

enum spi_b_delay_count_t	
Delay count for SPI delay settings.	
Enumerator	
SPI_B_DELAY_COUNT_1	Set RSPCK delay count to 1 RSPCK.
SPI_B_DELAY_COUNT_2	Set RSPCK delay count to 2 RSPCK.
SPI_B_DELAY_COUNT_3	Set RSPCK delay count to 3 RSPCK.
SPI_B_DELAY_COUNT_4	Set RSPCK delay count to 4 RSPCK.
SPI_B_DELAY_COUNT_5	Set RSPCK delay count to 5 RSPCK.
SPI_B_DELAY_COUNT_6	Set RSPCK delay count to 6 RSPCK.
SPI_B_DELAY_COUNT_7	Set RSPCK delay count to 7 RSPCK.
SPI_B_DELAY_COUNT_8	Set RSPCK delay count to 8 RSPCK.

◆ spi_b_clock_source_t

enum spi_b_clock_source_t
SPI communication clock source.

Function Documentation

◆ R_SPI_B_Open()

fsp_err_t R_SPI_B_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

This functions initializes a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the peripheral registers according to the configuration.
- Initialize the control structure for use in other SPI Interface functions.

Return values

FSP_SUCCESS	Channel initialized successfully.
FSP_ERR_ALREADY_OPEN	Instance was already initialized.
FSP_ERR_ASSERTION	An invalid argument was given in the configuration structure.
FSP_ERR_UNSUPPORTED	A requested setting is not possible on this device with the current build configuration.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel number is invalid.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::open

Note

This function is reentrant.

◆ R_SPI_B_Read()

fsp_err_t R_SPI_B_Read (spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function receives data from a SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI read operation.

Return values

FSP_SUCCESS	Read operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control or destination parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_B_Write()

fsp_err_t R_SPI_B_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function transmits data to a SPI device using the TX Only Communications Operation Mode. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI write operation.

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control or source parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_B_WriteRead()

fsp_err_t R_SPI_B_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function simultaneously transmits and receive data. Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI writeRead operation.

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control, source or destination parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_B_Close()

fsp_err_t R_SPI_B_Close (spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task. Implements spi_api_t::close.

Disables SPI operations by disabling the SPI bus.

- Disables the SPI peripheral.
- Disables all the associated interrupts.
- Update control structure so it will not work with SPI Interface functions.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	A required pointer argument is NULL.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

◆ R_SPI_B_CalculateBitrate()

fsp_err_t R_SPI_B_CalculateBitrate (uint32_t *bitrate*, spi_b_clock_source_t *clock_source*, rspck_div_setting_t * *spck_div*)

Calculates the SPBR register value and the BRDV bits for a desired bitrate. If the desired bitrate is faster than the maximum bitrate, than the bitrate is set to the maximum bitrate. If the desired bitrate is slower than the minimum bitrate, than an error is returned.

Parameters

[in]	bitrate	Desired bitrate
[in]	clock_source	SPI communication clock source to be used
[out]	spck_div	Memory location to store bitrate register settings.

Return values

FSP_SUCCESS	Valid spbr and brdv values were calculated
FSP_ERR_INVALID_ARGUMENT	Bitrate is not achievable or selected clock source is not supported.

◆ R_SPI_B_CallbackSet()

fsp_err_t R_SPI_B_CallbackSet (spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.6.10 I2S (r_ssi)

Modules » Connectivity

Functions

fsp_err_t	R_SSI_Open (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
fsp_err_t	R_SSI_Stop (i2s_ctrl_t *const p_ctrl)
fsp_err_t	R_SSI_StatusGet (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)
fsp_err_t	R_SSI_Write (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)
fsp_err_t	R_SSI_Read (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)
fsp_err_t	R_SSI_WriteRead (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)
fsp_err_t	R_SSI_Mute (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
fsp_err_t	R_SSI_Close (i2s_ctrl_t *const p_ctrl)
fsp_err_t	R_SSI_CallbackSet (i2s_ctrl_t *const p_api_ctrl, void(*p_callback)(i2s_callback_args_t *), void *const p_context, i2s_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the SSIE peripheral on RX MCUs. This module implements the I2S Interface.

Overview

Features

The SSI module supports the following features:

- Transmission and reception of uncompressed audio data using the standard I2S protocol in master and slave modes
- Full-duplex I2S communication (channel 0, 1)
- Integration with the DTC transfer module
- Internal connection to GPT timer output to generate the audio clock
- Callback function notification when all data is loaded into the SSI FIFO

Configuration

Build Time Configurations for r_ssi

The following build time configurations are defined in fsp_cfg/r_ssi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC Support	• Enabled • Disabled	Enabled	If code for DTC transfer support is included in the build.

Configurations for Connectivity > I2S (r_ssi)

This module can be added to the Stacks tab via New Stack > Connectivity > I2S (r_ssi).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2s0	Module name.
Channel	Value must be a non-negative integer	0	Specify the I2S channel.
Operating Mode (Master/Slave)	• Master Mode • Slave Mode	Master Mode	Select if the MCU is I2S master or slave.
Bit Depth	• 8 Bits • 16 Bits • 18 Bits • 20 Bits • 22 Bits • 24 Bits • 32 Bits	16 Bits	Select the bit depth of one sample of audio data.

Word Length	• 8 Bits • 16 Bits • 24 Bits • 32 Bits • 48 Bits • 64 Bits • 128 Bits • 256 Bits	16 Bits	Select the word length of audio data. Must be at least as large as Data bits.
WS Continue Mode	• Enabled • Disabled	Disabled	Enable WS continue mode to output the word select (WS) pin even when transmission is idle.
Bit Clock Source(available only in Master mode)	MCU Specific Options		Select External AUDIO_CLK for external signal to AUDIO_CLK input pin or Internal AUDIO_CLK for internal connection to MCU specific GPT channel. Please refer to the hardware manual for which GPT channel is connected to the internal signal
Bit Clock Divider(available only in Master mode)	Refer to the RX Configuration tool for available options.	Audio Clock / 1	Select divider used to generate bit clock from audio clock.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from all three interrupt service routines (ISR).
Transmit Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Idle/Error Interrupt Priority	MCU Specific Options		Select the Idle/Error interrupt priority.

Clock Configuration

The SSI peripheral runs on PCLKB. The PCLKB frequency can be configured on the **Clocks** tab of the FSP Configuration editor. The SSI audio clock can optionally be supplied from an external source through the AUDIO_CLK pin in master mode.

Pin Configuration

The SSI uses the following pins:

- AUDIO_CLK (optional, master mode only): The AUDIO_CLK pin is used to supply the audio clock from an external source.
- SSIBCKn: Bit clock pin for channel n
- SSILRCKn: Channel selection pin for channel n
- SSIRXDn: Reception pin for channel n
- SSITXDn: Transmission pin for channel n

Usage Notes

SSI Frames

An SSI frame is 2 samples worth of data. The frame boundary (end of previous frame, start of next frame) is on the falling edge of the SSILRCKn signal.

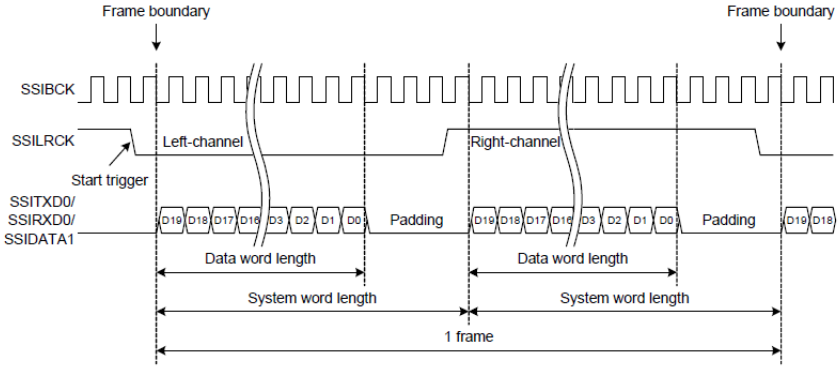


Figure 104: SSI Frame Diagram (8-bit word, 8-bit samples)

Note

If the word length is longer than the sample bit depth, padding bits (0) will be added after the sample.

Audio Data

Only uncompressed PCM data is supported.

Data arrays have the following size, alignment, and length based on the "Bit Depth" setting:

Bit Depth	Array Data Type	Required Alignment	Required Length (bytes)
8 Bits	8-bit integer	1 byte alignment	Multiple of 2
16 Bits	16-bit integer	2 byte alignment	Multiple of 4
18 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8
20 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8
22 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8

24 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8
32 Bits	32-bit integer	4 byte alignment	Multiple of 8

Note

The length of the array must be a multiple of 2 when the data type is the recommended data type. The 2 represents the frame size (left and right channel) of I2S communication. The SSIE peripheral does not support odd read/write lengths in I2S mode.

Audio Clock

The audio clock is only required for master mode.

Audio Clock Frequency

The bit clock frequency is the product of the sampling frequency and channels and bits per system word:

bit_clock (Hz) = sampling_frequency (Hz) * channels * system_word_bits

I2S data always has 2 channels.

For example, the bit clock for transmitting 2 channels of 16-bit data (using a 16-bit system word) at 44100 Hz would be:

44100 * 2 * 16 = 1,411,200 Hz

The audio clock frequency is used to generate the bit clock frequency. It must be a multiple of the bit clock frequency. Refer to the Bit Clock Divider configuration for divider options. The input audio clock frequency must be:

audio_clock (Hz) = desired_bit_clock (Hz) * bit_clock_divider

To get a bit clock of 1.4 MHz from an audio clock of 2.8 MHz, select the divider Audio Clock / 2.

Audio Clock Source

The audio clock source can come from:

- An external source input to the AUDIO_CLK pin
- An internal connection to the GPT timer output

Note

When using the internal GPT timer output, Pin Output Support must be Enabled, and GTIOCA Output Enabled must be True.
See the SSIE section in the MCU hardware manual for information about which GPT channel may be used.

Limitations

Developers should be aware of the following limitations when using the SSI:

- SSI must go idle before changing the communication mode (between read only, write only, and full duplex)

Examples

Basic Example

This is a basic example of minimal use of the SSI in an application.

```
#define SSI_EXAMPLE_SAMPLES_TO_TRANSFER (1024)
#define SSI_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t g_src[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
int16_t g_dest[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
void ssi_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t t = 0; t < SSI_EXAMPLE_SAMPLES_TO_TRANSFER / 2; t += 1)
    {
 float input = (((float) (freq * t)) * (M_TWOPI)) /
SSI_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       g_src[2 * t]    = (int16_t) ((INT16_MAX * sinf(input)));
       g_src[2 * t + 1] = (int16_t) ((INT16_MAX * sinf(input)));
    }
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Transfer data. */
    (void) R_SSI_WriteRead(&g_i2s_ctrl,
                           (uint8_t *) &g_src[0],
                           (uint8_t *) &g_dest[0],
                           SSI_EXAMPLE_SAMPLES_TO_TRANSFER * sizeof(int16_t));
}
```

Streaming Example

This is an example of using SSI to stream audio data. This application uses a double buffer to store

PCM sine wave data. It starts transmitting in the main loop, then loads the next buffer if it is ready in the callback. If the next buffer is not ready, a flag is set in the callback so the application knows to restart transmission in the main loop.

This example also checks the return code of R_SSI_Write() because R_SSI_Write() can return an error if a transmit overflow occurs before the FIFO is reloaded. If a transmit overflow occurs before the FIFO is reloaded, the SSI will be stopped in the error interrupt, and it cannot be restarted until the I2S_EVENT_IDLE callback is received.

```
#define SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ (22050)
#define SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK (1024)
#define SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t       g_stream_src[2][SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
uint32_t      g_buffer_index          = 0;
volatile bool g_send_data_in_main_loop = true;
volatile bool g_data_ready = false;
/* Example callback called when SSI is ready for more data. */
void ssi_example_callback (i2s_callback_args_t * p_args)
{
 /* Reload the FIFO if we hit the transmit watermark or restart transmission if the
SSI is idle because it was
  * stopped after a transmit FIFO overflow. */
 if ((I2S_EVENT_TX_EMPTY == p_args->event) || (I2S_EVENT_IDLE == p_args->event))
    {
 if (g_data_ready)
      {
 /* Reload FIFO and handle errors. */
            ssi_example_write();
      }
 else
      {
 /* Data was not ready yet, send it in the main loop. */
            g_send_data_in_main_loop = true;
      }
    }
}
/* Load the transmit FIFO and check for error conditions. */
void ssi_example_write (void)
```

```
{
 /* Transfer data. This call is non-blocking. */
 fsp_err_t err = R_SSI_Write(&g_i2s_ctrl,
                              (uint8_t *) &g_stream_src[g_buffer_index][0],
                              SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK * sizeof
(int16_t));
 if (FSP_SUCCESS == err)
    {
 /* Switch the buffer after data is sent. */
       g_buffer_index = !g_buffer_index;
 /* Allow loop to calculate next buffer only if transmission was successful. */
       g_data_ready = false;
    }
 else
    {
 /* Getting here most likely means a transmit overflow occurred before the FIFO could
be reloaded. The
  * application must wait until the SSI is idle, then restart transmission. In this
example, the idle
  * callback transmits data or resets the flag g_send_data_in_main_loop. */
    }
}
/* Calculate samples. This example is just a sine wave. For this type of data, it
would be better to calculate
 * one period and loop it. This example should be updated for the audio data used by
the application. */
void ssi_example_calculate_samples (uint32_t buffer_index)
{
 static uint32_t t = 0U;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t i = 0; i < SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK / 2; i += 1)
    {
```

```
 float input = (((float) (freq * t)) * M_TWOPI) /
SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       t++;
 /* Store sample twice, once for left channel and once for right channel. */
       int16_t sample = (int16_t) ((INT16_MAX * sinf(input)));
       g_stream_src[buffer_index][2 * i]    = sample;
       g_stream_src[buffer_index][2 * i + 1] = sample;
    }
 /* Data is ready to be sent in the interrupt. */
    g_data_ready = true;
}
void ssi_streaming_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Prepare data in a buffer that is not currently used for transmission. */
       ssi_example_calculate_samples(g_buffer_index);
 /* Send data in main loop the first time, and if it was not ready in the interrupt.
*/
 if (g_send_data_in_main_loop)
      {
 /* Clear flag. */
            g_send_data_in_main_loop = false;
 /* Reload FIFO and handle errors. */
            ssi_example_write();
      }
 /* If the next buffer is ready, wait for the data to be sent in the interrupt. */
 while (g_data_ready)
      {
```

```
 /* Do nothing. */
      }
    }
}
```

Data Structures

struct	ssi_instance_ctrl_t
struct	ssi_extended_cfg_t

Enumerations

enum	ssi_audio_clock_t
enum	ssi_clock_div_t

Data Structure Documentation

ssi_instance_ctrl_t

struct ssi_instance_ctrl_t
Channel instance control block. DO NOT INITIALIZE. Initialization occurs when i2s_api_t::open is called.

ssi_extended_cfg_t

struct ssi_extended_cfg_t		
SSI configuration extension. This extension is optional.		
Data Fields		
ssi_audio_clock_t	audio_clock	Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL.
ssi_clock_div_t	bit_clock_div	Select bit clock division ratio.

Enumeration Type Documentation

◆ ssi_audio_clock_t

enum ssi_audio_clock_t	
Audio clock source.	
Enumerator	
SSI_AUDIO_CLOCK_EXTERNAL	Audio clock source is the AUDIO_CLK input pin.
SSI_AUDIO_CLOCK_INTERNAL	Audio clock source is internal connection to a MCU specific GPT channel output.

◆ ssi_clock_div_t

enum ssi_clock_div_t	
Bit clock division ratio. Bit clock frequency = audio clock frequency / bit clock division ratio.	
Enumerator	
SSI_CLOCK_DIV_1	Clock divisor 1.
SSI_CLOCK_DIV_2	Clock divisor 2.
SSI_CLOCK_DIV_4	Clock divisor 4.
SSI_CLOCK_DIV_6	Clock divisor 6.
SSI_CLOCK_DIV_8	Clock divisor 8.
SSI_CLOCK_DIV_12	Clock divisor 12.
SSI_CLOCK_DIV_16	Clock divisor 16.
SSI_CLOCK_DIV_24	Clock divisor 24.
SSI_CLOCK_DIV_32	Clock divisor 32.
SSI_CLOCK_DIV_48	Clock divisor 48.
SSI_CLOCK_DIV_64	Clock divisor 64.
SSI_CLOCK_DIV_96	Clock divisor 96.
SSI_CLOCK_DIV_128	Clock divisor 128.

Function Documentation

◆ R_SSI_Open()

fsp_err_t R_SSI_Open (i2s_ctrl_t *const *p_ctrl*, i2s_cfg_t const *const *p_cfg*)

Opens the SSI. Implements i2s_api_t::open.

This function sets this clock divisor and the configurations specified in i2s_cfg_t. It also opens the timer and transfer instances if they are provided.

Return values

FSP_SUCCESS	Ready for I2S communication.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_cfg is null.
FSP_ERR_ALREADY_OPEN	The control block has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel number is not available on this MCU.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

◆ R_SSI_Stop()

fsp_err_t R_SSI_Stop (i2s_ctrl_t *const *p_ctrl*)

Stops SSI. Implements i2s_api_t::stop.

This function disables both transmission and reception, and disables any transfer instances used.

The SSI will stop on the next frame boundary. Do not restart SSI until it is idle.

Return values

FSP_SUCCESS	I2S communication stop request issued.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_SSI_StatusGet()

fsp_err_t R_SSI_StatusGet (i2s_ctrl_t *const *p_ctrl*, i2s_status_t *const *p_status*)

Gets SSI status and stores it in provided pointer p_status. Implements i2s_api_t::statusGet.

Return values

FSP_SUCCESS	Information stored successfully.
FSP_ERR_ASSERTION	The p_instance_ctrl or p_status parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_SSI_Write()

fsp_err_t R_SSI_Write (i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, uint32_t const *bytes*)

Writes data buffer to SSI. Implements i2s_api_t::write.

This function resets the transfer if the transfer interface is used, or writes the length of data that fits in the FIFO then stores the remaining write buffer in the control block to be written in the ISR.

Write() cannot be called if another write(), read() or writeRead() operation is in progress. Write can be called when the SSI is idle, or after the I2S_EVENT_TX_EMPTY event.

Return values

FSP_SUCCESS	Write initiated successfully.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_src was null, or bytes requested was 0.
FSP_ERR_IN_USE	Another transfer is in progress, data was not written.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_UNDERFLOW	A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

◆ R_SSI_Read()

fsp_err_t R_SSI_Read (i2s_ctrl_t *const *p_ctrl*, void *const *p_dest*, uint32_t const *bytes*)

Reads data into provided buffer. Implements i2s_api_t::read.

This function resets the transfer if the transfer interface is used, or reads the length of data available in the FIFO then stores the remaining read buffer in the control block to be filled in the ISR.

Read() cannot be called if another write(), read() or writeRead() operation is in progress. Read can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Return values

FSP_SUCCESS	Read initiated successfully.
FSP_ERR_IN_USE	Peripheral is in the wrong mode or not idle.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_dest was null, or bytes requested was 0.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_OVERFLOW	A receive overflow error is pending. Wait for the SSI to go idle before resuming communication.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

◆ R_SSI_WriteRead()

fsp_err_t R_SSI_WriteRead (i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, void *const *p_dest*, uint32_t const *bytes*)

Writes from source buffer and reads data into destination buffer. Implements i2s_api_t::writeRead.

This function calls R_SSI_Write and R_SSI_Read.

writeRead() cannot be called if another write(), read() or writeRead() operation is in progress. writeRead() can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Return values

FSP_SUCCESS	Write and read initiated successfully.
FSP_ERR_IN_USE	Peripheral is in the wrong mode or not idle.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_UNDERFLOW	A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication.
FSP_ERR_OVERFLOW	A receive overflow error is pending. Wait for the SSI to go idle before resuming communication.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

◆ R_SSI_Mute()

fsp_err_t R_SSI_Mute (i2s_ctrl_t *const *p_ctrl*, i2s_mute_t const *mute_enable*)

Mutes SSI on the next frame boundary. Implements i2s_api_t::mute.

Data is still written while mute is enabled, but the transmit line outputs zeros.

Return values

FSP_SUCCESS	Transmission is muted.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_SSI_Close()

fsp_err_t R_SSI_Close (i2s_ctrl_t *const *p_ctrl*)

Closes SSI. Implements i2s_api_t::close.

This function powers down the SSI and closes the lower level timer and transfer drivers if they are used.

Return values

FSP_SUCCESS	Device closed successfully.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_SSI_CallbackSet()

fsp_err_t R_SSI_CallbackSet (i2s_ctrl_t *const *p_api_ctrl*, void(*)(i2s_callback_args_t *) *p_callback*, void *const *p_context*, i2s_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2s_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.6.11 USB (r_usb_basic)

Modules » Connectivity

Functions

fsp_err_t	R_USB_Open (usb_ctrl_t *const p_ctrl, usb_cfg_t const *const p_cfg) Applies power to the USB module specified in the argument (p_ctrl). More...
fsp_err_t	R_USB_Close (usb_ctrl_t *const p_ctrl) Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module). More...

fsp_err_t	R_USB_Read (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination) Bulk/Interrupt data transfer. More...
fsp_err_t	R_USB_Write (usb_ctrl_t *const p_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination) Bulk/Interrupt data transfer. More...
fsp_err_t	R_USB_Stop (usb_ctrl_t *const p_ctrl, usb_transfer_t direction, uint8_t destination) Requests a data read/write transfer be terminated when a data read/write transfer is being performed. More...
fsp_err_t	R_USB_Suspend (usb_ctrl_t *const p_ctrl) Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure. More...
fsp_err_t	R_USB_Resume (usb_ctrl_t *const p_ctrl) Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure. More...
fsp_err_t	R_USB_VbusSet (usb_ctrl_t *const p_ctrl, uint16_t state) Specifies starting or stopping the VBUS supply. More...
fsp_err_t	R_USB_InfoGet (usb_ctrl_t *const p_ctrl, usb_info_t *p_info, uint8_t destination) Obtains completed USB-related events. More...
fsp_err_t	R_USB_PipeRead (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) Requests a data read (Bulk/Interrupt transfer) via the pipe specified in the argument. More...
fsp_err_t	R_USB_PipeWrite (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) Requests a data write (Bulk/Interrupt transfer). More...

fsp_err_t	R_USB_PipeStop (usb_ctrl_t *const p_ctrl, uint8_t pipe_number) Terminates a data read/write operation. More...
fsp_err_t	R_USB_UsedPipesGet (usb_ctrl_t *const p_ctrl, uint16_t *p_pipe, uint8_t destination) Gets the selected pipe number (number of the pipe that has completed initalization) via bit map information. More...
fsp_err_t	R_USB_PipeInfoGet (usb_ctrl_t *const p_ctrl, usb_pipe_t *p_info, uint8_t pipe_number) Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size. More...
fsp_err_t	R_USB_PullUp (usb_ctrl_t *const p_ctrl, uint8_t state) This API enables or disables pull-up of D+/D- line. More...
fsp_err_t	R_USB_EventGet (usb_ctrl_t *const p_ctrl, usb_status_t *event) Obtains completed USB related events. (OS-less Only) More...
fsp_err_t	R_USB_Callback (usb_callback_t *p_callback) Register a callback function to be called upon completion of a USB related event. (RTOS only) More...
fsp_err_t	R_USB_HostControlTransfer (usb_ctrl_t *const p_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address) Performs settings and transmission processing when transmitting a setup packet. More...
fsp_err_t	R_USB_PeriControlDataGet (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size) Receives data sent by control transfer. More...
fsp_err_t	R_USB_PeriControlDataSet (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size) Performs transfer processing for control transfer. More...

fsp_err_t	R_USB_PeriControlStatusSet (usb_ctrl_t *const p_ctrl, usb_setup_status_t status) Set the response to the setup packet. More...
fsp_err_t	R_USB_RemoteWakeup (usb_ctrl_t *const p_ctrl) Sends a remote wake-up signal to the connected Host. More...
fsp_err_t	R_USB_ModuleNumberGet (usb_ctrl_t *const p_ctrl, uint8_t *module_number) This API gets the module number. More...
fsp_err_t	R_USB_ClassTypeGet (usb_ctrl_t *const p_ctrl, usb_class_t *class_type) This API gets the class type. More...
fsp_err_t	R_USB_DeviceAddressGet (usb_ctrl_t *const p_ctrl, uint8_t *device_address) This API gets the device address. More...
fsp_err_t	R_USB_PipeNumberGet (usb_ctrl_t *const p_ctrl, uint8_t *pipe_number) This API gets the pipe number. More...
fsp_err_t	R_USB_DeviceStateGet (usb_ctrl_t *const p_ctrl, uint16_t *state) This API gets the state of the device. More...
fsp_err_t	R_USB_DataSizeGet (usb_ctrl_t *const p_ctrl, uint32_t *data_size) This API gets the read data size. More...
fsp_err_t	R_USB_SetupGet (usb_ctrl_t *const p_ctrl, usb_setup_t *setup) This API gets the setup information. More...
fsp_err_t	R_USB_OtgCallbackSet (usb_ctrl_t *const p_ctrl, usb_otg_callback_t *p_callback) Set callback function to be called when the OTG role swap was completed on Azure RTOS. More...

fsp_err_t	R_USB_OtgSRP (usb_ctrl_t *const p_ctrl) Start the SRP processing for OTG on Azure RTOS. More...
fsp_err_t	R_USB_DriverActivate (usb_ctrl_t *const p_api_ctrl) Activate USB Driver for USB Peripheral BareMetal. More...
fsp_err_t	R_USB_CallbackMemorySet (usb_ctrl_t *const p_api_ctrl, usb_callback_args_t *p_callback_memory) Set callback memory to USB Driver for USB Peripheral BareMetal. More...

Detailed Description

Driver for the USB peripheral on RX MCUs. This module implements the USB Interface.

Overview

The USB module operates in combination with the device class drivers provided by Renesas to form a complete USB stack.

Features

The USB module has the following key features:

- USB Host mode
  - Enumerates Low/Full/High-speed devices (see note below)
  - Automatic transfer error determination and retry
- USB Peripheral mode
  - Supports USB1.1/2.0/3.0 hosts
- Automatic processing of device connect/disconnect, suspend/resume, and USB bus reset
- Up to 10 pipes
  - Control transfers supported on pipe 0
  - Data transfer on pipes 1 to 9 (Bulk or Interrupt)
- Functions with or without an RTOS

Note

Supported speeds are dependent on the MCU.

Configuration

Build Time Configurations for r_usb_basic

The following build time configurations are defined in fsp_cfg/r_usb_basic_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
PLL Frequency	MCU Specific Options		Specify the PLL frequency supplied to the USB module. This setting only applies to USB1 (not USB0).
CPU Bus Access Wait Cycles	MCU Specific Options		This setting controls the delay for consecutive USB peripheral register access. Set this value to a number of CPU cycles that is equivalent to 40.8ns or more.
Battery Charging	MCU Specific Options		Specify whether or not to include battery charging functionality.
Power IC Shutdown Polarity	MCU Specific Options		Select the polarity of the Shutdown signal on the power supply IC (if provided).
Dedicated Charging Port (DCP) Mode	MCU Specific Options		When enabled, USB communication is disabled and the port is used for charging only.
Notifications for SET_INTERFACE/SET_FEATURE/CLEAR_FEATURE	• Disabled • Enabled	Enabled	When enabled, the application will receive notifications for SET_INTERFACE, SET_FEATURE and CLEAR_FEATURE messages.
Double Buffering	• Disabled • Enabled	Enabled	When enabled, the FIFOs for Pipes 1-5 are double-buffered.
Continuous Transfer Mode	MCU Specific Options		Enable or disable continuous transfer mode.
LDO Regulator	MCU Specific Options		Enable or disable LDO regulator.
Type-C Connector	• Disabled • Enabled	Disabled	Enable or disable Type-C Connector.
DMA Support	MCU Specific Options		Enable or disable DMA support for the USB module.

USB Compliance Test mode	• Enabled • Disabled	Disabled	Display the information required to take the compliance test.
USB TPL table name	Enter the TPL table name.	NULL	Enter the name of the TPL Table.

Configurations for Connectivity > USB (r_usb_basic)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_basic0	Module name.
USB Mode	• Host mode • Peri mode	Peri mode	Select the usb mode.
USB Speed	• Full Speed • Hi Speed • Low Speed	Full Speed	Select the USB speed.
USB Module Number	• USB_IP0 Port • USB_IP1 Port	USB_IP0 Port	Specify the USB module number to be used.
USB Device Class	Refer to the RX Configuration tool for available options.	Peripheral Communications Device Class	Select the USB device class.
USB Descriptor	USB Descriptor must be a valid C symbol.	g_usb_descriptor	Enter the name of the descriptor to be used. For how to create a descriptor structure, refer to the Descriptor definition chapter in the usb_basic manual. Specify NULL when using the Host class.
USB Compliance Callback	Compliance Callback must be a valid C symbol.	NULL	Set the callback for compliance tests here.
USBFS Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the main USBFS ISR.
USBFS Resume Priority	MCU Specific Options		Select the interrupt priority used by the USBFS Resume ISR.
USBFS D0FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBFS D0FIFO.
USBFS D1FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBFS D1FIFO.
USBHS Interrupt	MCU Specific Options		Select the interrupt

Priority			priority used by the main USBHS ISR.
USBHS D0FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBHS D0FIFO ISR.
USBHS D1FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBHS D1FIFO ISR.
USB RTOS Callback	Enter the address of the function.	NULL	If an FreeRTOS is used, set the callback function here.
USB Callback Context	Enter the address of the context.	NULL	Set the callback context here.

Clock Configuration

The USB module uses PLL as the clock source. The PLL frequency can be set in the **Clocks** tab of the configuration editor or by using the CGC Interface at run-time.

Pin Configuration

In peripheral mode the USB_VBUS and/or USBHS_VBUS pins are used to detect the USB connection status (connected or disconnected) and should be connected to the USB VBUS signal.

Note

USB_VBUS and USBHS_VBUS are 5V-tolerant pins.

In host mode the USBHS_VBUSEN, USBHS_OVRCURA and USBHS_OVRCURB pins should be connected to the relevant pins of an external power supply IC, if available. These pins will be used to manage the USB VBUS supply.

DMA Configuration

When using DMA with USB the following properties must be configured for each DMAC module:

Config Name	Select Name	Description
Transfer Size	2 Bytes 4 Bytes	In FS mode, select "2 Bytes" In HS mode, select "4 Bytes"
Activation source	USBFS FIFO 0 USBFS FIFO 1 USBHS FIFO 0 USBHS FIFO 1	USB FS Reception USB FS Transmission USB HS Reception USB HS Transmission

Descriptor definition

In Peripheral mode, the usb_descriptor_t structure stores descriptor information including the device and configuration descriptors. The values set in this structure are sent to the USB host as part of enumeration.

Note:

Descriptor definitions need to be defined by the user.
Template files for descriptor definitions are provided within the folder for each device class, so users can refer to these templates.
For example, in the case of PCDC, the template file is located in the following:
r_usb_pcdc/r_usb_pcdc_descriptor.c.template

```
typedef struct usb_descriptor
{
    uint8_t *p_device;    /* Pointer to device descriptor */
    uint8_t *p_config_f; /* Pointer to full-speed configuration descriptor */
    uint8_t *p_config_h; /* Pointer to high-speed configuration descriptor (HS only)
*/
    uint8_t *p_qualifier; /* Pointer to device qualifier descriptor (HS only) */
    uint8_t **pp_string; /* Pointer to string descriptor table */
    uint8_t num_string;   /* Number of strings in table */
} usb_descriptor_t;
```

Note

Even in high-speed mode the full-speed configuration must be made available:

```
/* Example USB FS descriptor struct */
usb_descriptor_t g_usb_descriptor =
{
    smp_device,
    smp_config_f,
    NULL,
    NULL,
    smp_str_table,
    3,
};
/* Example USB HS descriptor struct */
usb_descriptor_t g_usb_descriptor =
{
    smp_device,
    smp_config_f,
    smp_config_h,
```

```
    smp_qualifier,
    smp_str_table,
    3,
};
```

String Descriptor

This USB driver requires string descriptors to be registered in the string descriptor table. Use the following format to define the elements:

```
/* String descriptor 0 is reserved for language ID information */
uint8_t str_descriptor_0[]
{
    0x04,      /* Length */
    0x03,      /* Descriptor type */
    0x09, 0x04 /* Language ID */
};
uint8_t str_descriptor_manufacturer[] =
{
    0x10,      /* Length */
    0x03,      /* Descriptor type */
 'R', 0x00,
 'E', 0x00,
 'N', 0x00,
 'E', 0x00,
 'S', 0x00,
 'A', 0x00,
 'S', 0x00
};
uint8_t str_descriptor_product[] =
{
    0x12,      /* Length */
    0x03,      /* Descriptor type */
 'C', 0x00,
 'D', 0x00,
```

```
 'C', 0x00,
 '_', 0x00,
 'D', 0x00,
 'E', 0x00,
 'M', 0x00,
 'O', 0x00
};
/* String descriptor table */
uint8_t * smp_str_table[] =
{
    str_descriptor_0,            /* Index: 0 */
    str_descriptor_manufacturer, /* Index: 1 */
    str_descriptor_product,      /* Index: 2 */
};
```

Note

*Set the string index values in the device/configuration descriptors (iManufacturer, iConfiguration etc.) to the index of the desired string in the string descriptor table. For example, in the table below, the manufacturer is described in **str_descriptor_manufacturer** and the value of iManufacturer in the device descriptor is **1**.*

Other Descriptors

Refer to the Universal Serial Bus Revision 2.0 specification (http://www.usb.org/developers/docs/) for details on how to construct the device, configuration and qualifier descriptors.

Usage Notes

Program Structure

USB applications (whether using an RTOS or not) should be written as an event-handling loop. Either a callback function (RTOS only) or R_USB_EventGet should be used to provide event data to the application loop where a switch statement handles the event.

Note

1.The USB_STATUS_CONFIGURED event should be confirmed before calling R_USB_Read or R_USB_Write.
2.When attaching to USB Host, Suspend event is notified to the application program in USB peripheral mode. Notification of this event to the application program does not affect the operation.

Limitations

Developers should be aware of the following limitations when using the USB driver:

- The current USB driver does not support FreeRTOS.

- The current USB driver does not support hub.
- In USB host mode, the module does not support suspend during data transfers. Execute suspend only after confirming that all transfers are complete.
- Multiconfigurations are not supported.
- This driver does not support CPU transfers using the D0FIFO/D1FIFO register.
- Only one device-class driver may be used at a time.
- The USB Hi-Speed module only supports Hi-Speed operation.
- In USB host mode, this USB driver does not support the composite device.
- The user can not specify DMA transfer to USB IP0 and IP1 modules at the same time when using USB multi-port feature. USB multi-port function: Simultaneous operation feature of USB Host and Peripheral.

Compliance Test

Please set as follows to the following items in RX configuration (r_usb_basic) when doing the compliance test.

1.USB Compliance Test mode Set "Enabled" in this item.

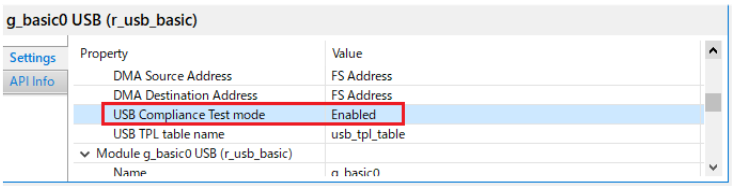


Figure 105: Compliance Test Setting

2.USB TPL table name. Set the start address of TPL(Target Peripheral List) defined in the application program.

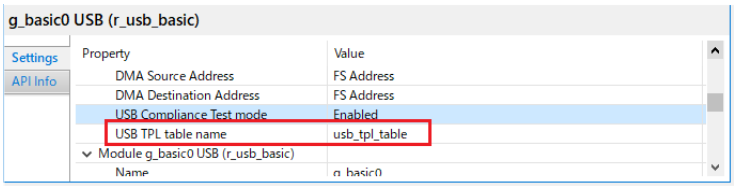


Figure 106: TPL Start Address Setting

Please refer to the following about how to define for TPL.
The following example is when two devices are set in TPL.

```
const uint16_t usb_tpl_table[] =
{
    2,                  /* Number of tpl */
    0,                  /* Reserved */
    0x0123, 0x4567,     /* Vendor ID, Product ID (1st device) */
    0x89ab, 0xcdef      /* Vendor ID, Product ID (2nd device) */
```

```
};
```

3.USB Compliance Callback Set the start address of the callback function defined in the application program.

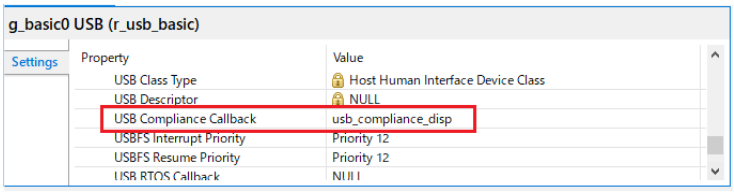


Figure 107: Compliance Callback Setting

The user needs to create this callback function by referring to the following.

```
void usb_compliance_disp (void * param)
{
    usb_compliance_t     *disp_data;
    uint8_t              print_data[32];
    disp_data = (usb_compliance_t*)param;
 switch(disp_data->status)
    {
 case USB_COMPLIANCETEST_ATTACH:            /* Device Attach Detection */
            display("\nATTACH \n");
 break;
 case USB_COMPLIANCETEST_DETACH:            /* Device Detach Detection */
            display("\nDETACH \n");
 break;
 case USB_COMPLIANCETEST_TPL:               /* TPL device connect */
            display("\nTPL PID:%04x VID:%04x \n",disp_data->pid, disp_data->vid);
 break;
 case USB_COMPLIANCETEST_NOTTPL:            /* Not TPL device connect */
            display("\nNOTPL PID:%04x VID:%04x \n",disp_data->pid, disp_data->vid);
 break;
 case USB_COMPLIANCETEST_HUB:               /* USB Hub connect */
            display("\nHub    \n");
 break;
```

```
 case USB_COMPLIANCETEST_OVRC:             /* over current */
 break;
 case USB_COMPLIANCETEST_NORES:             /* Responce Time out for Control Read
Transfer */
            display("\nNOTRESP \n");
 break;
 case USB_COMPLIANCETEST_SETUP_ERR:         /* Setup Transaction Error */
 break;
 default:
 break;
    }
} /* End of function usb_compliance_disp() */
```

Please replace the display function described in the example with the display function created by the customer.

Callback function for USB Peripheral

The user can get completed USB event using callback function in USB Peripheral for BareMetal.

- Please specify user callback function in the following item.
- Please call the *R_USB_DriverActivate* function using one of the following methods.
    - Infinite loop in user application program
    - Timer Interrupt (Add r_gpt module in user system)
- Please sepecify the area to store USB events to USB driver using *R_USB_CallbackMemorySet* function after calling *R_USB_Open* function.

```
usb_callback_args_t g_apl_usb_event_callback;
```

- Please allocate a buffer area such as a ring buffer to store USB event information in user application program.

```
#define BUFSIZE 10
usb_callback_args_t g_apl_usb_event_buffer[BUFSIZE];
```

- The USB driver calls the callback function when the USB event completes. USB event information is stored in the area indicated by the callback function argument.Copy the callback function argument to the ring buffer area or other area secured above.

```

```

```
g_apl_usb_event_buffer[g_apl_buffer_wp] = g_apl_usb_event_callback;
g_apl_buffer_wp++;
g_apl_buffer_wp %= BUFSIZE;
```

UCLK setting

Enable UCLK in "Clocks" tab on e² studio when using the following MCU.

1. RX72N

Examples

USB Basic Example

This is a basic example of minimal use of the USB in an application.

```
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 case USB_STATUS_WRITE_COMPLETE:
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_READ_COMPLETE:
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
```

```
                    g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
      }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
      }
 else
      {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
      }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
      }
    }
} /* End of function usb_main() */
```

Typedefs

typedef usb_event_info_t	usb_instance_ctrl_t

Typedef Documentation

◆ usb_instance_ctrl_t

typedef usb_event_info_t usb_instance_ctrl_t
USB private control block. DO NOT MODIFY. Initialization occurs when R_USB_Open is called.

Function Documentation

◆ R_USB_Open()

fsp_err_t R_USB_Open (usb_ctrl_t *const *p_ctrl*, usb_cfg_t const *const *p_cfg*)

Applies power to the USB module specified in the argument (p_ctrl).

Return values

FSP_SUCCESS	Success in open.
FSP_ERR_USB_BUSY	Specified USB module now in use.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Close()

fsp_err_t R_USB_Close (usb_ctrl_t *const *p_ctrl*)

Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module).

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_NOT_OPEN	USB module is not open.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Read()

fsp_err_t R_USB_Read (usb_ctrl_t *const *p_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *destination*)

Bulk/Interrupt data transfer.

Requests USB data read (bulk/interrupt transfer). The read data is stored in the area specified by argument (p_buf). After data read is completed, confirm the operation by checking the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function. The received data size is set in member (size) of the usb_ctrl_t structure. To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

Return values

FSP_SUCCESS	Successfully completed (Data read request completed).
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Data receive request already in process for USB device with same device address.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1. Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).
2. Allocate the following the storage area when using DMA transfer and specify the start address of the allocated storage area to the 2nd argument(p_buf).
(1). When using High-speed and enabling continuous transfer mode, allocate the storage area with a size that is a multiple of 2048.
(2). When using High-speed and disabling continuous transfer mode, allocate the storage area with a size that is a multiple of 512.
(3). When using Full-speed, allocate the storage area with a size that is a multiple of 64.
3. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.
(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.
(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.

◆ R_USB_Write()

fsp_err_t R_USB_Write (usb_ctrl_t *const *p_ctrl*, uint8_t const *const *p_buf*, uint32_t *size*, uint8_t *destination*)

Bulk/Interrupt data transfer.

Requests USB data write (bulk/interrupt transfer). Stores write data in area specified by argument (p_buf). Set the device class type in usb_ctrl_t structure member (type). Confirm after data write is completed by checking the return value (USB_STATUS_WRITE_COMPLETE) of the R_USB_GetEvent function. For sending a zero-length packet, please refer the following Note.

Return values

FSP_SUCCESS	Successfully completed. (Data write request completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Data write request already in process for USB device with same device address.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1. The user needs to send the zero-length packet(ZLP) since this USB driver does not send the ZLP automatically. When sending a ZLP, the user sets USB_NULL in the third argument (size) of R_USB_Write function as follow.
e.g)
R_USB_Write (&g_basic0_ctrl, &g_buf, USB_NULL);
2. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.
(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.
(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.
3. Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Stop()

fsp_err_t R_USB_Stop (usb_ctrl_t *const *p_ctrl*, usb_transfer_t *direction*, uint8_t *destination*)

Requests a data read/write transfer be terminated when a data read/write transfer is being performed.

To stop a data read, set USB_TRANSFER_READ as the argument (type); to stop a data write, specify USB_WRITE as the argument (type).

Return values

FSP_SUCCESS	Successfully completed. (stop completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_BUSY	Stop processing is called multiple times.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Suspend()

fsp_err_t R_USB_Suspend (usb_ctrl_t *const *p_ctrl*)

Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure.

After the suspend request is completed, confirm the operation with the return value (USB_STATUS_SUSPEND) of the R_USB_EventGet function.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	During a suspend request to the specified USB module, or when the USB module is already in the suspended state.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Resume()

fsp_err_t R_USB_Resume (usb_ctrl_t *const *p_ctrl*)

Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure.

After the resume request is completed, confirm the operation with the return value (USB_STATUS_RESUME) of the R_USB_EventGet function

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Resume already requested for same device address. (USB host mode only)
FSP_ERR_USB_NOT_SUSPEND	USB device is not in the SUSPEND state.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_VbusSet()

fsp_err_t R_USB_VbusSet (usb_ctrl_t *const *p_ctrl*, uint16_t *state*)

Specifies starting or stopping the VBUS supply.

Return values

FSP_SUCCESS	Successful completion. (VBUS supply start/stop completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_InfoGet()

fsp_err_t R_USB_InfoGet (usb_ctrl_t *const *p_ctrl*, usb_info_t * *p_info*, uint8_t *destination*)

Obtains completed USB-related events.

Return values

FSP_SUCCESS	Successful completion. (VBUS supply start/stop completed)
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_PipeRead()

fsp_err_t R_USB_PipeRead (usb_ctrl_t *const *p_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number*)

Requests a data read (Bulk/Interrupt transfer) via the pipe specified in the argument.

The read data is stored in the area specified in the argument (p_buf). After the data read is completed, confirm the operation with the R_USB_GetEvent function return value(USB_STATUS_READ_COMPLETE). To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1. Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).
2. Allocate the following the storage area when using DMA transfer and specify the start address of the allocated storage area to the 2nd argument(p_buf).
(1). When using High-speed and enabling continuous transfer mode, allocate the storage area with a size that is a multiple of 2048.
(2). When using High-speed and disabling continuous transfer mode, allocate the storage area with a size that is a multiple of 512.
(3). When using Full-speed, allocate the storage area with a size that is a multiple of 64.
3. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.
(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.
(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.

◆ R_USB_PipeWrite()

fsp_err_t R_USB_PipeWrite (usb_ctrl_t *const *p_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number*)

Requests a data write (Bulk/Interrupt transfer).

The write data is stored in the area specified in the argument (p_buf). After data write is completed, confirm the operation with the return value (USB_STATUS_WRITE_COMPLETE) of the EventGet function. For sending a zero-length packet, please refer the following Note.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1. The user needs to send the zero-length packet(ZLP) since this USB driver does not send the ZLP automatically. When sending a ZLP, the user sets USB_NULL in the third argument (size) of R_USB_PipeWrite function as follow.
e.g)
R_USB_PipeWrite (&g_basic0_ctrl, &g_buf, USB_NULL, pipe_number);
2. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.
(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.
(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.
3. Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PipeStop()

fsp_err_t R_USB_PipeStop (usb_ctrl_t *const *p_ctrl*, uint8_t *pipe_number*)

Terminates a data read/write operation.

Return values

FSP_SUCCESS	Successfully completed. (Stop request completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_UsedPipesGet()

fsp_err_t R_USB_UsedPipesGet (usb_ctrl_t *const *p_ctrl*, uint16_t * *p_pipe*, uint8_t *destination*)

Gets the selected pipe number (number of the pipe that has completed initalization) via bit map information.

The bit map information is stored in the area specified in argument (p_pipe). Based on the information (module member and address member) assigned to the usb_ctrl_t structure, obtains the PIPE information of that USB device.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_PipeInfoGet()

fsp_err_t R_USB_PipeInfoGet (usb_ctrl_t *const *p_ctrl*, usb_pipe_t * *p_info*, uint8_t *pipe_number*)

Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size.

The obtained pipe information is stored in the area specified in the argument (p_info).

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_PullUp()

fsp_err_t R_USB_PullUp (usb_ctrl_t *const *p_ctrl*, uint8_t *state*)

This API enables or disables pull-up of D+/D- line.

Return values

FSP_SUCCESS	Successful completion. (Pull-up enable/disable setting completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_EventGet()

fsp_err_t R_USB_EventGet (usb_ctrl_t *const *p_ctrl*, usb_status_t * *event*)

Obtains completed USB related events. (OS-less Only)

In USB host mode, the device address value of the USB device that completed an event is specified in the usb_ctrl_t structure member (address) specified by the event's argument. In USB peripheral mode, USB_NULL is specified in member (address). If this function is called in the RTOS execution environment, a failure is returned.

Return values

FSP_SUCCESS	Event Get Success.
FSP_ERR_USB_FAILED	If called in the RTOS environment, an error is returned.

Note

Do not use the same variable as the first argument of R_USB_Open for the first argument.
Do not call this API in the interrupt function.

◆ R_USB_Callback()

fsp_err_t R_USB_Callback (usb_callback_t * *p_callback*)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

This function registers a callback function to be called when a USB-related event has completed. If this function is called in the OS-less execution environment, a failure is returned.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_ASSERTION	Parameter is NULL error.

Note

Do not call this API in the interrupt function.

◆ R_USB_HostControlTransfer()

fsp_err_t R_USB_HostControlTransfer (usb_ctrl_t *const *p_ctrl*, usb_setup_t * *p_setup*, uint8_t * *p_buf*, uint8_t *device_address*)

Performs settings and transmission processing when transmitting a setup packet.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PeriControlDataGet()

fsp_err_t R_USB_PeriControlDataGet (usb_ctrl_t *const *p_ctrl*, uint8_t * *p_buf*, uint32_t *size*)

Receives data sent by control transfer.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PeriControlDataSet()

fsp_err_t R_USB_PeriControlDataSet (usb_ctrl_t *const *p_ctrl*, uint8_t * *p_buf*, uint32_t *size*)

Performs transfer processing for control transfer.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PeriControlStatusSet()

fsp_err_t R_USB_PeriControlStatusSet (usb_ctrl_t *const *p_ctrl*, usb_setup_status_t *status*)

Set the response to the setup packet.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_RemoteWakeup()

fsp_err_t R_USB_RemoteWakeup (usb_ctrl_t *const *p_ctrl*)

Sends a remote wake-up signal to the connected Host.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_NOT_SUSPEND	Device is not suspended.
FSP_ERR_USB_BUSY	The device is in resume operation.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_ModuleNumberGet()

fsp_err_t R_USB_ModuleNumberGet (usb_ctrl_t *const *p_ctrl*, uint8_t * *module_number*)

This API gets the module number.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_ClassTypeGet()

fsp_err_t R_USB_ClassTypeGet (usb_ctrl_t *const *p_ctrl*, usb_class_t * *class_type*)

This API gets the class type.

Return values

FSP_SUCCESS	Successful completion.

Note

In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.
In the FreeRTOS, please specify one of the following to the 1st argument of this API.
1. The 1st argument of the callback function specified in Conguration.
2. The start address of the area where the structure area of the 1st argument was copied.

◆ R_USB_DeviceAddressGet()

fsp_err_t R_USB_DeviceAddressGet (usb_ctrl_t *const *p_ctrl*, uint8_t * *device_address*)

This API gets the device address.

Return values

FSP_SUCCESS	Successful completion.

Note

In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.
In the FreeRTOS, please specify one of the following to the 1st argument of this API.
1. The 1st argument of the callback function specified in Conguration.
2. The start address of the area where the structure area of the 1st argument was copied.

◆ R_USB_PipeNumberGet()

fsp_err_t R_USB_PipeNumberGet (usb_ctrl_t *const *p_ctrl*, uint8_t * *pipe_number*)

This API gets the pipe number.

Return values

FSP_SUCCESS	Successful completion.

Note

In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.
In the FreeRTOS, please specify one of the following to the 1st argument of this API.
1. The 1st argument of the callback function specified in Conguration.
2. The start address of the area where the structure area of the 1st argument was copied.

◆ R_USB_DeviceStateGet()

fsp_err_t R_USB_DeviceStateGet (usb_ctrl_t *const *p_ctrl*, uint16_t * *state*)

This API gets the state of the device.

Return values

FSP_SUCCESS	Successful completion.

Note

In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.
In the FreeRTOS, please specify one of the following to the 1st argument of this API.
1. The 1st argument of the callback function specified in Conguration.
2. The start address of the area where the structure area of the 1st argument was copied.

◆ R_USB_DataSizeGet()

fsp_err_t R_USB_DataSizeGet (usb_ctrl_t *const *p_ctrl*, uint32_t * *data_size*)

This API gets the read data size.

Return values

FSP_SUCCESS	Successful completion.

Note

In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.
In the FreeRTOS, please specify one of the following to the 1st argument of this API.
1. The 1st argument of the callback function specified in Conguration.
2. The start address of the area where the structure area of the 1st argument was copied.

◆ R_USB_SetupGet()

fsp_err_t R_USB_SetupGet (usb_ctrl_t *const *p_ctrl*, usb_setup_t * *setup*)

This API gets the setup information.

Return values

FSP_SUCCESS	Successful completion.

Note

In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.
In the FreeRTOS, please specify one of the following to the 1st argument of this API.
1. The 1st argument of the callback function specified in Conguration.
2. The start address of the area where the structure area of the 1st argument was copied.

◆ R_USB_OtgCallbackSet()

fsp_err_t R_USB_OtgCallbackSet (usb_ctrl_t *const *p_ctrl*, usb_otg_callback_t * *p_callback*)

Set callback function to be called when the OTG role swap was completed on Azure RTOS.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.

◆ R_USB_OtgSRP()

fsp_err_t R_USB_OtgSRP (usb_ctrl_t *const *p_ctrl*)

Start the SRP processing for OTG on Azure RTOS.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.

Note

Do not support the VBUS Pulsing since OTG 2.0 does not support the VBUS Pulsing..

◆ R_USB_DriverActivate()

fsp_err_t R_USB_DriverActivate (usb_ctrl_t *const *p_api_ctrl*)

Activate USB Driver for USB Peripheral BareMetal.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.

Note

Call this API in the in the infinite loop of the application program or a timer interrupt.

◆ R_USB_CallbackMemorySet()

fsp_err_t R_USB_CallbackMemorySet (usb_ctrl_t *const *p_api_ctrl*, usb_callback_args_t * *p_callback_memory*)

Set callback memory to USB Driver for USB Peripheral BareMetal.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.

Note

Call this API after calling R_USB_Open function.

5.2.6.12 USB Composite (r_usb_composite)

Modules » Connectivity

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

USB composite device works as a USB Peripheral by combining two peripheral device classes and r_usb_basic module.
This USB driver supports the following composite devices:

1. PCDC + PHID
2. PCDC + PCDC
3. PHID + PHID
4. PCDC + PVND

How to Configuration

The following shows FSP configuration procedure for USB composite device.

- Select [New Stack]->[USB]->[USB Composite]

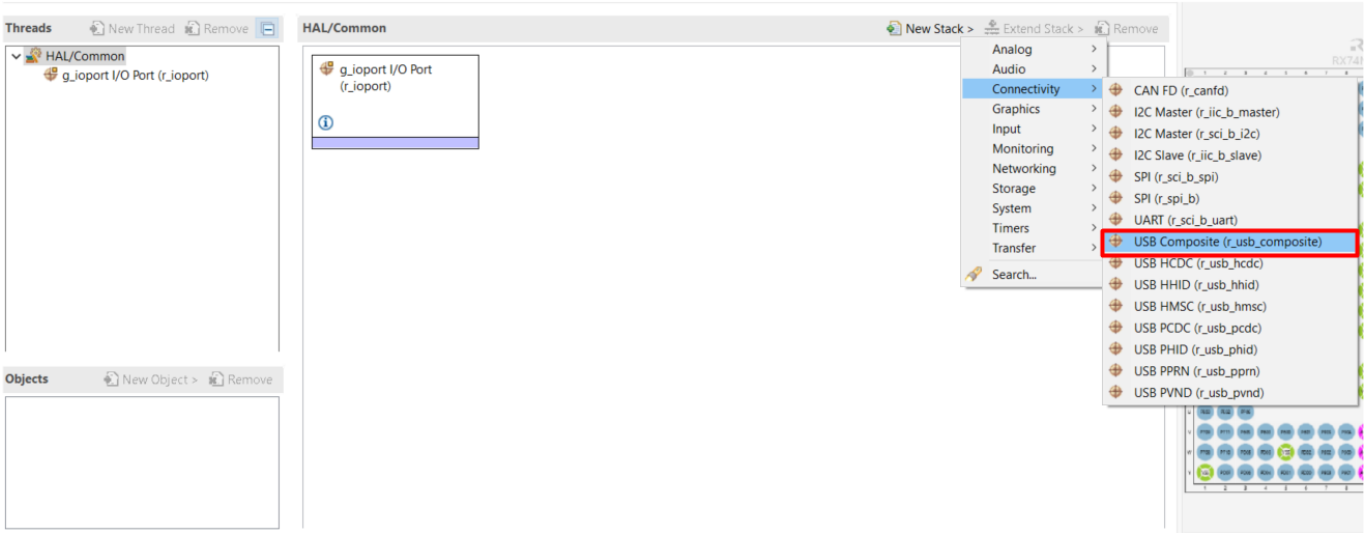


Figure 108: Select USB Composite

- The following is displayed when selecting [USB Composite].

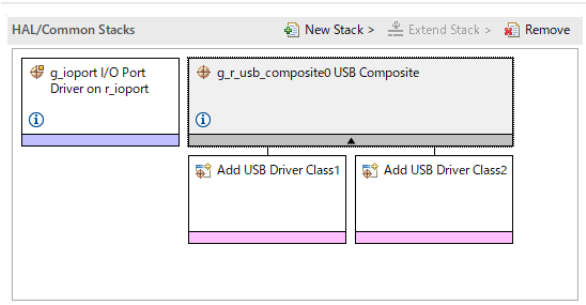


Figure 109: USB Composite Stack

- Select the supported 2 device classes as follows.

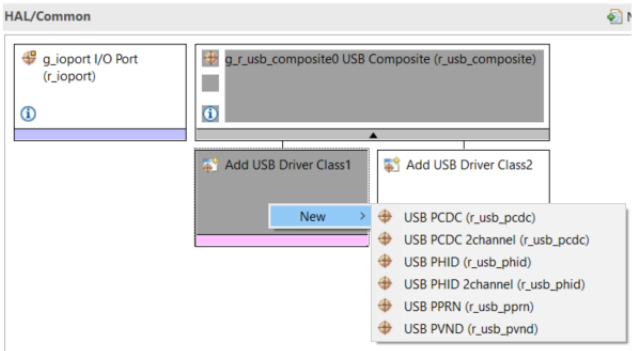


Figure 110: Select Device Classes

Note

1. *Be sure to select "USB PCDC driver on r_usb_pcdc" and "USB PCDC 2channel driver on r_usb_pcdc" when configurating for "PCDC + PCDC".*
2. *Be sure to select "USB PHID driver on r_usb_phid" and "USB PHID 2channel driver on r_usb_phid" when configurating for "PHID + PHID".*

- Select the supported 2 device classes as follows. The following is displayed when selecting 2 device classes.

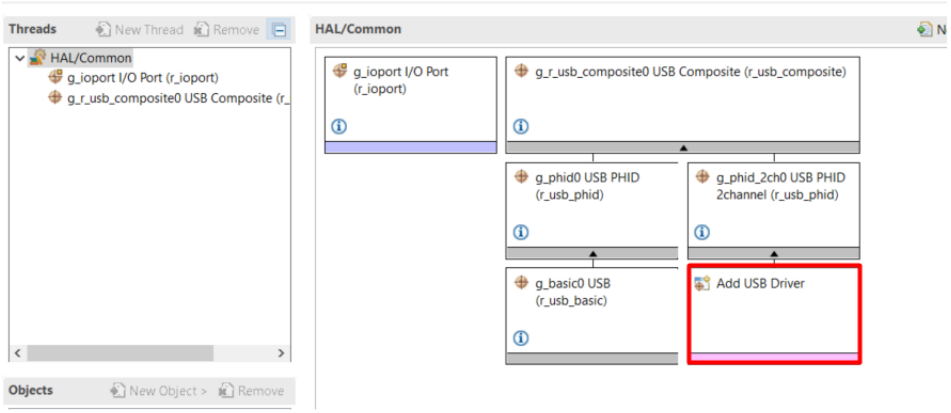


Figure 111: Delete USB Basic Instance

Note

1. *Delete the "g_basic1" instance manually since this instance is not used in composite device. (Refer to the red frame in the above figure.)*
2. *The error is output when selecting the following device classes.*
    a. *PHID + PHID*

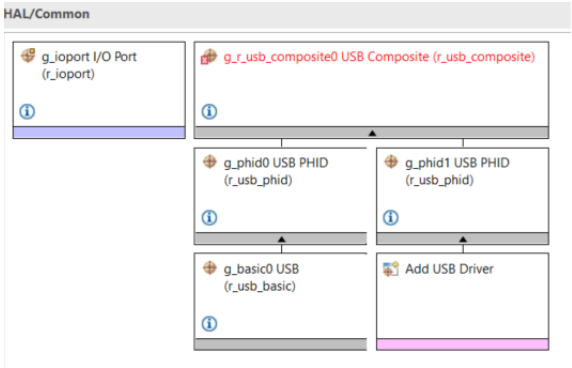


Figure 112: Device Class Selection Error

Notes

Please determine by the member "pipe" in "usb_event_info" structure when getting PCDC channel number which the write event is completed in PCDC + PCDC.
Don't refer to the member "type" in "usb_event_info" structure.

Descriptor

Templates for composite device descriptors can be found in ra/fsp/src/r_usb_composite folder. Also, please be sure to use your vendor ID.

1. r_usb_pcdc_phid_descriptor.c.template (for PCDC + PHID)
2. r_usb_pcdc_pcdc_descriptor.c.template (for PCDC + PCDC)
3. r_usb_phid_phid_descriptor.c.template (for PHID + PHID)
4. r_usb_pcdc_pvnd_descriptor.c.template (for PCDC + PVND)

Examples

USB COMPOSITE Example

- PCDC + PHID

```
void main_task (void)
{
  #if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
  #endif
```

```
   usb_event_info_t usb_event;
usb_status_t    event;
   uint8_t       * p_idle_value;
   uint8_t          sw_data;
   usb_info_t       info;
fsp_err_t       ret_code = FSP_SUCCESS;
   uint8_t          send_data[16] BSP_ALIGN_VARIABLE(4);
   uint8_t          req_comp_flag = 0;
   uint8_t          count         = 0;
   g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
   set_key_data(g_buf_phid);
/* Loop back between PC(TerminalSoft) and USB MCU */
while (1)
   {
 #if (BSP_CFG_RTOS == 2)
      USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
      usb_event = *p_mess;
event    = usb_event.event;
 #else  /* (BSP_CFG_RTOS == 2) */
R_USB_EventGet(&usb_event, &event);
 #endif /* (BSP_CFG_RTOS == 2) */
switch (event)
     {
case USB_STATUS_CONFIGURED:
     {
               g_status = NO_WRITING;
               g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
break;
          }
case USB_STATUS_WRITE_COMPLETE:
     {
if (usb_event.type == USB_CLASS_PCDC)
     {
                 g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
```

```
            }
 else if (usb_event.type == USB_CLASS_PHID)
     {
 if (DATA_WRITING == g_status)
     {
                        g_status = ZERO_WRITING;
                        g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN_PHID, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
            }
 else if (g_status == ZERO_WRITING)
     {
                g_status = NO_WRITING;
            }
            }
 break;
           }
 case USB_STATUS_READ_COMPLETE:
     {
 if (usb_event.type == USB_CLASS_PCDC)
     {
                 g_usb_on_usb.write(&g_basic0_ctrl, g_buf, usb_event.data_size,
USB_CLASS_PCDC);
 if (req_comp_flag == 1)
     {
 if (g_status == NO_WRITING)
     {
                           count++;
                           g_status = DATA_WRITING;
                           g_usb_on_usb.write(&g_basic0_ctrl, g_buf_phid,
DATA_LEN_PHID, USB_CLASS_PHID);
            }
            }
            }
 break;
```

```
            }
 case USB_STATUS_REQUEST:   /* Receive Class Request */
      {
 if (USB_PCDC_SET_LINE_CODING == (usb_event.setup.request_type & USB_BREQUEST))
      {
 R_USB_PeriControlDataGet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
              }
 else if (USB_PCDC_GET_LINE_CODING == (usb_event.setup.request_type & USB_BREQUEST))
      {
 R_USB_PeriControlDataSet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
              }
 else if (USB_SET_REPORT == (usb_event.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
              }
 else if (USB_GET_DESCRIPTOR == (usb_event.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == usb_event.setup.request_value)
      {
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                        (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
              }
 else if (USB_GET_HID_DESCRIPTOR == usb_event.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
                             send_data[i] = g_apl_configuration[84 + i];
              }
 /* Configuration Descriptor address set. */
```

```
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
             }
 else
     {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
             }
             }
 else if (USB_SET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
     {
 /* Get SetIdle value */
                  p_idle_value = (uint8_t *) &usb_event.setup.request_value;
                  g_idle       = p_idle_value[1];
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
             }
 else if (USB_GET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
             }
 else if (USB_SET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
             }
 else if (USB_GET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
             }
 else
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
```

```
USB_SETUP_STATUS_STALL);
                }
 break;
             }
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
      {
 if (USB_SET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
                    p_idle_value = (uint8_t *) &usb_event.setup.request_value;
                    g_idle = p_idle_value[1];
                }
 else if (USB_SET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
 /* None */
 /* g_protocol = event_info.setup.value; */
                }
 else
      {
                    req_comp_flag = 1;
                }
 break;
             }
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
      {
 break;
             }
 default:
      {
 break;
             }
        }
    }
} /* End of function usb_main() */
```

```
void set_key_data (uint8_t * p_buf)
{
 static uint8_t key_data;
    key_data = KBD_CODE_A;
    *(p_buf + 2) = key_data;
}
  #if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_rec_msg
 * Description : Receive a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t** mess : Message pointer
 * : usb_tm_t tm : Timeout Value
 * Return : uint16_t : USB_OK / USB_ERROR
 ******************************************************************************/
usb_er_t usb_apl_rec_msg (uint8_t id, usb_msg_t ** mess, usb_tm_t tm)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
    (void) tm;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    *mess = NULL;
    err = xQueueReceive(handle, (void *) mess, (portMAX_DELAY));
 if ((pdTRUE == err) && (NULL != (*mess)))
    {
       result = USB_APL_OK;
    }
 else
    {
```

```
        result = USB_APL_ERROR;
    }
 return result;
}
/*****************************************************************************
 * End of function usb_apl_rec_msg
 *****************************************************************************/
/*****************************************************************************
 * Function Name : usb_apl_snd_msg
 * Description : Send a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t* mess : Message pointer
 * Return : usb_er_t : USB_OK / USB_ERROR
 *****************************************************************************/
usb_er_t usb_apl_snd_msg (uint8_t id, usb_msg_t * mess)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    err = xQueueSend(handle, (const void *) &mess, (TickType_t) (0));
 if (pdTRUE == err)
    {
        result = USB_APL_OK;
    }
 else
    {
        result = USB_APL_ERROR;
    }
 return result;
```

```
}
/******************************************************************************
 * End of function usb_apl_snd_msg
 ******************************************************************************/
  #endif  /* #if (BSP_CFG_RTOS == 2) */
```

- PCDC + PVND

```
/******************************************************************************
 * Includes <System Includes> , "Project Includes"
 ******************************************************************************/
#include "r_usb_basic.h"
#include "r_usb_pcdc_api.h"
#include "r_usb_pcdc_cfg.h"
/******************************************************************************
 * Macro definitions
 ******************************************************************************/
#define DATA_LEN 2048
#define RESET_VALUE 0
#define INT_EVT_TSK_STACK_SIZE 1024  /* Stack size of Event notification thread for
vendor class's Interrupt IN/OUT test */
#define INT_EVT_TSK_PRI 2  /* Priority of Event notification thread for vendor
class's Interrupt IN/OUT test */
#define BUF_SIZE (2048)  /* Buffer size */
#define REQ_SIZE (20)  /* Request buffer size */
#define USB_VALUE_FF (0xFFU)  /* FF macro */
#define USB_APL_MXPS (64U)
#define START_PIPE (USB_PIPE1)  /* Start pipe number */
#define END_PIPE (USB_PIPE9 + 1) /* Total pipe */
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
#define SET_VENDOR_NO_DATA (USB_SET_VENDOR_NO_DATA | USB_HOST_TO_DEV | USB_VENDOR |
```

```
USB_INTERFACE)
#define SET_VENDOR (USB_SET_VENDOR | USB_HOST_TO_DEV | USB_VENDOR | USB_INTERFACE)
#define GET_VENDOR (USB_GET_VENDOR | USB_DEV_TO_HOST | USB_VENDOR | USB_INTERFACE)
#define DELAY (10U) /* Delay for print */
#define USB_STATUS_VENDOR_INTIN_TEST ((usb_status_t) 0x01)
#define USB_STATUS_VENDOR_INTOUT_TEST ((usb_status_t) 0x02)
#define USB_STATUS_COM_IN_TEST ((usb_status_t) 0x03)
#define USB_STATUS_RE_INIT_TEST ((usb_status_t) 0x04)
#define HS_MAX_PACKET_SIZE (512)
#define FS_MAX_PACKET_SIZE (64)
#define VALUE_A1H (0xA1)
#define LINE_CODING_LENGTH (0x07U)
/******************************************************************************
 * Private global variables and functions
 ******************************************************************************/
static uint8_t g_cdc_buf[BUF_SIZE] = {RESET_VALUE}; /* Data buffer for PCDC Normal
Request */
static uint8_t g_vnd_buf[BUF_SIZE] = {RESET_VALUE}; /* Data buffer for PVND Normal
Request */
static uint8_t g_request_buf[REQ_SIZE] = {RESET_VALUE}; /* Data buffer for PVND Class
Request */
static uint8_t g_bulk_in_pipe = RESET_VALUE; /* Bulk In Pipe */
static uint8_t g_bulk_out_pipe = RESET_VALUE; /* Bulk Out Pipe */
static uint8_t g_interrupt_in_pipe = RESET_VALUE; /* Interrupt In Pipe */
static uint8_t g_interrupt_out_pipe = RESET_VALUE; /* Interrupt Out Pipe */
static uint16_t g_max_packet_size = USB_APL_MXPS;
/* Variable to capture USB event. */
static volatile usb_event_info_t * p_usb_event = NULL;
static volatile bool g_err_flag = false;
static bool b_usb_attach = false;
static usb_pcdc_linecoding_t g_line_coding;
#if (BSP_CFG_RTOS == 2)
static TaskHandle_t g_app_interrupt_event_notify_tsk_hdl;
#endif  /* (BSP_CFG_RTOS == 2) */
```

```
static uint32_t g_interrupt_in_test_flag = 0;
static uint32_t g_interrupt_out_test_flag = 0;
static uint32_t g_interrupt_out_times = 0;
static uint32_t g_interrupt_com_event = 0;
static uint8_t g_interrupt_in_com_data[8] = {VALUE_A1H, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00};
static uint32_t g_test_reinit_flag = 0;
/* Interrupt Event for Application */
#if (BSP_CFG_RTOS == 2)
static usb_event_info_t g_event_info2;
static usb_event_info_t * g_p_event_info2;
#else
static usb_event_info_t g_usb_event;
static usb_status_t g_event;
static uint32_t g_event2;
#endif
extern const usb_cfg_t g_basic0_cfg;
usb_instance_ctrl_t g_basic0_ctrl;
/* Function definitions */
static fsp_err_t process_usb_events(void);
static fsp_err_t process_usb_events_for_interrupt(void);
static fsp_err_t usb_configured_event_process(void);
static fsp_err_t usb_status_request(void);
static void handle_error(fsp_err_t err);
static fsp_err_t buffer_check(uint32_t length);
void usb_composite_thread_entry(void * pvParameters);
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_app_interrupt_event_task
 * Description : Event notification thread for Interrupt IN/OUT test.
 * Argument : void * pvParameters : Ipointer to pvParameters
 * Return : none
 ******************************************************************************/
void usb_app_interrupt_event_task (void * pvParameters)
```

```
{
 while (1)
    {
 if (1 == g_interrupt_in_test_flag)
      {
            g_interrupt_in_test_flag = 0;
            g_event_info2.event = USB_STATUS_VENDOR_INTIN_TEST;
            g_p_event_info2    = &g_event_info2;
 /* Send Interrupt-IN event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
      {
                g_err_flag = true;
            }
       }
 if (1 == g_interrupt_out_test_flag)
      {
            g_interrupt_out_test_flag = 0;
            g_event_info2.event = USB_STATUS_VENDOR_INTOUT_TEST;
            g_p_event_info2    = &g_event_info2;
 /* Send Interrupt-OUT event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
      {
                g_err_flag = true;
            }
       }
 if (0 != g_interrupt_com_event)
      {
            g_interrupt_com_event = 0;
            g_event_info2.event = USB_STATUS_COM_IN_TEST;
            g_p_event_info2    = &g_event_info2;
 /* Send Interrupt-IN event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
```

```
(TickType_t) (RESET_VALUE))))
      {
                g_err_flag = true;
            }
       }
 if (0 != g_test_reinit_flag)
      {
            g_test_reinit_flag = 0;
            g_event_info2.event = USB_STATUS_RE_INIT_TEST;
            g_p_event_info2    = &g_event_info2;
 /* Send RE_INIT_TEST event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
      {
                g_err_flag = true;
            }
       }
       vTaskDelay(100);
    }
}
/*****************************************************************************
 * End of function usb_app_interrupt_event_task
 *****************************************************************************/
#endif  /* (BSP_CFG_RTOS == 2) */
/*****************************************************************************
 * Function Name : usb_composite_thread_entry
 * Description : Peripheral CDC & Vendor application main process
 * Arguments : none
 * Return value : none
 *****************************************************************************/
void usb_composite_thread_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
 fsp_err_t err = FSP_SUCCESS;
```

```
#if (BSP_CFG_RTOS == 2)
    BaseType_t err_queue = pdFALSE;
    BaseType_t err_task = pdFALSE;
#endif  /* (BSP_CFG_RTOS == 2) */
 /* Open USB instance */
 R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 if (USB_SPEED_FS == g_basic0_cfg.usb_speed)
    {
       g_max_packet_size = FS_MAX_PACKET_SIZE;
    }
 else
    {
       g_max_packet_size = HS_MAX_PACKET_SIZE;
    }
#if (BSP_CFG_RTOS == 2)
    err_task = xTaskCreate((TaskFunction_t) usb_app_interrupt_event_task,
 "INT_EVT_TSK",
                           INT_EVT_TSK_STACK_SIZE,
                           NULL,
                           INT_EVT_TSK_PRI,
                           &g_app_interrupt_event_notify_tsk_hdl);
 if (pdPASS != err_task)
    {
 return;
    }
#endif
 while (true)
    {
 /* Handle error if queue send fails*/
 if (true == g_err_flag)
      {
            handle_error(FSP_ERR_ABORTED);
       }
#if (BSP_CFG_RTOS == 2)
```

```
 /* Receive message from queue */
       err_queue = xQueueReceive(g_event_queue, &p_usb_event, (portMAX_DELAY));
 /* Handle error */
 if (pdTRUE != err_queue)
      {
            handle_error(FSP_ERR_ABORTED);
       }
 if (pdTRUE == err_queue)
      {
 if (p_usb_event == &g_event_info2)
      {
 /* process Application Interrupt events */
                err = process_usb_events_for_interrupt();
                handle_error(err);
            }
 else
      {
 /* process USB events */
                err = process_usb_events();
                handle_error(err);
            }
       }
#else  /* (BSP_CFG_RTOS == 2) */
 R_USB_EventGet(&g_usb_event, &g_event);
 if (USB_STATUS_NONE == g_event)
      {
 /* Interrupt IN Test Event for Vendor Class */
 if (1 == g_interrupt_in_test_flag)
      {
              g_interrupt_in_test_flag = 0;
                g_event2                = USB_STATUS_VENDOR_INTIN_TEST;
                err = process_usb_events_for_interrupt();
            }
 /* Interrupt OUT Test Event for Vendor Class */
```

```
else if (1 == g_interrupt_out_test_flag)
     {
             g_interrupt_out_test_flag = 0;
               g_event2 = USB_STATUS_VENDOR_INTOUT_TEST;
               err      = process_usb_events_for_interrupt();
           }
/* Interrupt IN Test Event for COM Class */
else if (0 != g_interrupt_com_event)
     {
             g_interrupt_com_event = 0;
               g_event2             = USB_STATUS_COM_IN_TEST;
               err = process_usb_events_for_interrupt();
           }
/* ReInit Test Event */
else if (0 != g_test_reinit_flag)
     {
             g_test_reinit_flag = 0;
               g_event2          = USB_STATUS_RE_INIT_TEST;
               err                = process_usb_events_for_interrupt();
           }
else
     {
               err = process_usb_events();
           }
      }
else
     {
           p_usb_event = &g_usb_event;
           err         = process_usb_events();
      }
      handle_error(err);
#endif
    }
}
```

```
/*****************************************************************************
 * End of function usb_composite_thread_entry
 *****************************************************************************/
/*****************************************************************************
 * Function Name : process_usb_events
 * Description : Function processes usb events.
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
 *****************************************************************************/
static fsp_err_t process_usb_events (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* USB event received */
#if (BSP_CFG_RTOS == 2)
 switch (p_usb_event->event)
#else  /* (BSP_CFG_RTOS == 2) */
 switch (g_event)
#endif /* (BSP_CFG_RTOS == 2) */
    {
 case USB_STATUS_CONFIGURED:    /* Configured State */
      {
 /* Process USB configured event */
            usb_configured_event_process();
 /* [PCDC] Read data from terminal software */
 R_USB_Read(&g_basic0_ctrl, g_cdc_buf, (uint32_t) g_max_packet_size, USB_CLASS_PCDC);
 /* [PVND] Read data from ra_usb_hvnd.exe */
            memset(g_vnd_buf, RESET_VALUE, BUF_SIZE);
            err = R_USB_PipeRead(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], BUF_SIZE,
g_bulk_out_pipe);
 break;
       }
 case USB_STATUS_WRITE_COMPLETE: /* Write Complete State */
      {
```

```
 if (b_usb_attach)
      {
 /* [PCDC] Read data from terminal software */
 if ((USB_CFG_PCDC_BULK_IN == p_usb_event->pipe) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
      {
                    err = R_USB_Read(&g_basic0_ctrl, g_cdc_buf, (uint32_t)
g_max_packet_size, USB_CLASS_PCDC);
              }
 /* [PVND] Read data from ra_usb_hvnd.exe */
 if ((g_bulk_in_pipe == p_usb_event->pipe) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
      {
              memset(g_vnd_buf, RESET_VALUE, BUF_SIZE);
 /* Read data back */
              err = R_USB_PipeRead(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], BUF_SIZE,
g_bulk_out_pipe);
              }
            }
 else
      {
 // Do Nothing as USB is removed and not connected yet.
            }
 break;
       }
 case USB_STATUS_READ_COMPLETE: /* Read Complete State */
      {
 if (b_usb_attach)
      {
 /* PCDC */
 if ((USB_CFG_PCDC_BULK_OUT == p_usb_event->pipe) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
      {
 /* Write back the read data from terminal software to it. */
```

```
                    err = R_USB_Write(&g_basic0_ctrl, g_cdc_buf,
p_usb_event->data_size, USB_CLASS_PCDC);
              }
 /* PVND */
 if ((g_bulk_out_pipe == p_usb_event->pipe) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
      {
 /* Data comparison read from host */
                    err = buffer_check(p_usb_event->data_size);
 if (FSP_SUCCESS == err)
      {
 /* Write data back to host */
 R_USB_PipeWrite(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], p_usb_event->data_size,
                                        g_bulk_in_pipe);
              }
 else
      {
 return FSP_ERR_USB_FAILED;
              }
              }
            }
 else
      {
 // Do Nothing as USB is removed and not connected yet.
            }
 break;
       }
 case USB_STATUS_REQUEST:       /* Receive Class Request */
      {
 /* Perform usb status request operation.*/
            err = usb_status_request();
 break;
       }
 case USB_STATUS_REQUEST_COMPLETE: /* Request Complete State */
```

```
     {
// Do Nothing.
break;
      }
case USB_STATUS_DETACH:
case USB_STATUS_SUSPEND:
     {
/* Reset the usb attached flag as indicating usb is removed.*/
           b_usb_attach = false;
           memset(g_cdc_buf, RESET_VALUE, sizeof(g_cdc_buf));
           memset(g_vnd_buf, RESET_VALUE, sizeof(g_vnd_buf));
break;
      }
case USB_STATUS_RESUME:
     {
/* set the usb attached flag*/
           b_usb_attach = true;
break;
      }
default:
     {
break;
      }
   }
return err;
}
/******************************************************************************
 * End of function process_usb_events
 ******************************************************************************/
/******************************************************************************
 * Function Name : process_usb_events_for_interrupt
 * Description : Function processes Application events (Interrupt IN/OUT Request ).
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
```

```
operation.
 ***************************************************************************/
static fsp_err_t process_usb_events_for_interrupt (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint8_t   pipe = RESET_VALUE;
 /* USB event received */
#if (BSP_CFG_RTOS == 2)
 switch (p_usb_event->event)
#else  /* (BSP_CFG_RTOS == 2) */
 switch (g_event2)
#endif /* (BSP_CFG_RTOS == 2) */
    {
 case USB_STATUS_VENDOR_INTIN_TEST:
      {
            pipe = g_interrupt_in_pipe;
            err = R_USB_PipeWrite(&g_basic0_ctrl, (uint8_t *) "interrupt-in-test",
strlen("interrupt-in-test"), pipe);
 break;
       }
 case USB_STATUS_VENDOR_INTOUT_TEST:
      {
            pipe = g_interrupt_out_pipe;
            err = R_USB_PipeRead(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], (BUF_SIZE),
pipe);
 if (FSP_SUCCESS == err)
      {
                g_interrupt_out_times++;
            }
 break;
       }
 case USB_STATUS_COM_IN_TEST:
      {
            pipe = USB_CFG_PCDC_INT_IN;
```

```
            err = R_USB_PipeWrite(&g_basic0_ctrl, g_interrupt_in_com_data,
sizeof(g_interrupt_in_com_data), pipe);
 break;
       }
 case USB_STATUS_RE_INIT_TEST:
      {
            g_usb_on_usb.close(&g_basic0_ctrl);
      g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 break;
       }
 default:
      {
 break;
       }
    }
 return err;
}
/******************************************************************************
 * End of function process_usb_events_for_interrupt
 ******************************************************************************/
/******************************************************************************
 * Function Name : usb_configured_event_process
 * Description : Function processes USB configured event (for vendor class).
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
 ******************************************************************************/
static fsp_err_t usb_configured_event_process (void)
{
 fsp_err_t err       = FSP_SUCCESS;
    uint16_t   used_pipe = RESET_VALUE;
    usb_pipe_t pipe_info = {RESET_VALUE};
    uint8_t    pipe      = RESET_VALUE;
 /* Get USB Pipe Information */
```

```
   err = R_USB_UsedPipesGet(&g_basic0_ctrl, &used_pipe, USB_CLASS_PVND);
if (FSP_SUCCESS == err)
   {
for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
     {
/* check for the used pipe */
if ((used_pipe & (START_PIPE << pipe)) != RESET_VALUE)
     {
/* Get the pipe Info */
               err = R_USB_PipeInfoGet(&g_basic0_ctrl, &pipe_info, pipe);
if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
     {
/* Out Transfer */
if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
     {
if (pipe != USB_CFG_PCDC_BULK_OUT)
     {
                         g_bulk_out_pipe = pipe;
             }
             }
else if (USB_TRANSFER_TYPE_INT == pipe_info.transfer_type)
     {
                       g_interrupt_out_pipe = pipe;
             }
else
     {
/* Do nothing */
             }
             }
else
     {
/* In Transfer */
if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
     {
```

```
 if (pipe != USB_CFG_PCDC_BULK_IN)
      {
                              g_bulk_in_pipe = pipe;
              }
              }
 else if (USB_TRANSFER_TYPE_INT == pipe_info.transfer_type)
      {
 if (pipe != USB_CFG_PCDC_INT_IN)
      {
                              g_interrupt_in_pipe = pipe;
              }
              }
 else
      {
 /* Do nothing */
              }
              }
            }
 else
      {
 /* Do nothing */
            }
       }
    }
 return err;
}
/*****************************************************************************
 * End of function usb_configured_event_process
 *****************************************************************************/
/*****************************************************************************
 * Function Name : usb_status_request
 * Description : Function processes usb status request.
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
```

```
operation.
 ***************************************************************************/
static fsp_err_t usb_status_request (void)
{
 fsp_err_t err            = FSP_SUCCESS;
    uint16_t request_length = RESET_VALUE;
 /* Check for the specific CDC class request IDs */
 if (USB_PCDC_SET_LINE_CODING == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
 /* Get the class request.*/
       err = R_USB_PeriControlDataGet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
    }
 else if (USB_PCDC_GET_LINE_CODING == (p_usb_event->setup.request_type & USB_BREQUEST
))
    {
 /* Set the class request.*/
       err = R_USB_PeriControlDataSet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
    }
 else if (USB_PCDC_SET_CONTROL_LINE_STATE == (p_usb_event->setup.request_type &
USB_BREQUEST))
    {
 /* Set the usb status as ACK response.*/
       err = R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
    }
 else if (USB_SET_VENDOR_NO_DATA == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
 /* Set ACk to host */
       err = R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
    }
 else if (USB_SET_VENDOR == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
       request_length = p_usb_event->setup.request_length;
```

```
 /* Get data length from host */
       err = R_USB_PeriControlDataGet(&g_basic0_ctrl, &g_request_buf[RESET_VALUE],
(uint32_t) request_length);
    }
 else if (USB_GET_VENDOR == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
       request_length = p_usb_event->setup.request_length;
 /* Set data length in peripheral */
       err = R_USB_PeriControlDataSet(&g_basic0_ctrl, &g_request_buf[RESET_VALUE],
(uint32_t) request_length);
    }
 else
    {
 // Do Nothing.
    }
 return err;
}
/******************************************************************************
 * End of function usb_status_request
 ******************************************************************************/
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_composite_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_event_info : Control structure for USB API.
 * usb_hdl_t handler : Task Handle
 * usb_onoff_t usbon_off_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_composite_callback (usb_event_info_t * p_event_info, usb_hdl_t handler,
usb_onoff_t on_off)
{
 FSP_PARAMETER_NOT_USED(handler);
 FSP_PARAMETER_NOT_USED(on_off);
```

```
 /* Send event received to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &p_event_info, (TickType_t)
(RESET_VALUE))))
    {
       g_err_flag = true;
    }
}
/*****************************************************************************
 * End of function usb_composite_callback
 *****************************************************************************/
#endif  /* (BSP_CFG_RTOS == 2) */
/*****************************************************************************
 * Function Name : handle_error
 * Description : Closes the USB module , Print and traps error.
 * Arguments : fsp_err_t err : error status
 * Return value : none
 *****************************************************************************/
static void handle_error (fsp_err_t err)
{
 if (FSP_SUCCESS != err)
    {
 R_USB_Close(&g_basic0_ctrl);
    }
}
/*****************************************************************************
 * End of function handle_error
 *****************************************************************************/
/*****************************************************************************
 * Function Name : buffer_check
 * Description : Check data received from vendor host tools
 * Arguments : uint32_t length : data length
 * char * err_str : error string
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
```

```
 ******************************************************************************/
static fsp_err_t buffer_check (uint32_t length)
{
 for (uint16_t cnt = RESET_VALUE; cnt < (uint16_t) length; cnt++)
    {
 if ((uint8_t) (cnt & USB_VALUE_FF) != g_vnd_buf[cnt])
      {
 return FSP_ERR_ABORTED;
       }
    }
 R_BSP_SoftwareDelay(DELAY, BSP_DELAY_UNITS_MILLISECONDS);
 return FSP_SUCCESS;
}
/******************************************************************************
 * End of function buffer_check
 ******************************************************************************/
```

5.2.6.13 USB Host Communications Device Class (r_usb_hcdc)

Modules » Connectivity

Functions

fsp_err_t	R_USB_HCDC_ControlDataRead (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t device_address) Read Control Data.(CDC Interrupt IN data) More...
fsp_err_t	R_USB_HCDC_SpecificDeviceRegister (usb_ctrl_t *const p_api_ctrl, uint16_t vendor_id, uint16_t product_id) Register the specified vendor class device in the device table. More...
fsp_err_t	R_USB_HCDC_DeviceInfoGet (usb_ctrl_t *const p_api_ctrl, usb_hcdc_device_info_t *p_info, uint8_t device_address) Get the VID, PID and subclass code of the connected device. More...

Detailed Description

This module provides a USB Host Communications Device Class (HCDC) driver. It implements the USB HCDC Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_hcdc module, when used in combination with the r_usb_basic module, operates as a USB Host Communications Device Class (HCDC) driver. The HCDC conforms to the PSTN device subclass abstract control model of the USB Communications Device Class (CDC) specification and enables communication with a CDC peripheral device.

Features

The r_usb_hcdc module has the following key features:

- Checks for connected devices
- Implementation of communication line settings
- Acquisition of the communication line state
- Data transfer to and from a CDC peripheral device

Configuration

Build Time Configurations for r_usb_hcdc

The following build time configurations are defined in fsp_cfg/r_usb_hcdc_cfg.h:

Configuration	Options	Default	Description
Target Peripheral Device Class ID	• CDC class supported device • Vendor class device	CDC class supported device	Specify the device class ID of the CDC device to be connected.
Bulk Input Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE4	Select the USB pipe to use for bulk input transfers.
Bulk Output Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE5	Select the USB pipe to use for bulk output transfers.
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8	USB PIPE6	Select the USB pipe to use for interrupts.

- USB PIPE9

Configurations for Connectivity > USB HCDC (r_usb_hcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HCDC (r_usb_hcdc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_hcdc0	Module name.

Note

Refer to the USB (r_usb_basic) module for hardware configuration options.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Communications Device Class (CDC), PSTN and ACM

This software conforms to the Abstract Control Model (ACM) subclass of the Communications Device Class specification as defined in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2. The Abstract Control Model subclass is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections) enabling use of application programs designed for older modems.

Basic Functions

The main functions of HCDC are the following:

- Verify connected devices
- Make communication line settings
- Acquire the communication line state
- Transfer data to and from the CDC peripheral device

Abstract Control Model Class Requests - Host to Device

This driver supports the following class requests:

Request	Code	Description
SendEncapsulatedCommand	0x00	Transmits an AT command as defined by the protocol used by the device (normally 0 for USB).
GetEncapsulatedResponse	0x01	Requests a response to a command transmitted by SendEncapsulatedCommand.

SetCommFeature	0x02	Enables or disables features such as device-specific 2-byte code and country setting.
GetCommFeature	0x03	Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting.
ClearCommFeature	0x04	Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting.
SetLineCoding	0x20	Makes communication line settings (communication speed, data length, parity bit, and stop bit length).
GetLineCoding	0x21	Acquires the communication line setting state.
SetControlLineState	0x22	Makes communication line control signal (RTS, DTR) settings.
SendBreak	0x23	Transmits a break signal.

Note

For more information about Abstract Control Model requests, refer to Table 11 "Requests - Abstract Control Model" in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.

The expected data format for each command is shown below followed by dependent structures.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SEND_ENCAPSULATED_COMMAND (0x00)	0x0000	0x0000	Data length	usb_hcdc_encapsulated_t
0x21	GET_ENCAPSULATED_RESPONSE (0x01)	0x0000	0x0000	Data length	usb_hcdc_encapsulated_t
0x21	SET_COMM_FEATURE (0x02)	usb_hcdc_feature_selector_t	0x0000	Data length	usb_hcdc_commfeature_t
0x21	GET_COMM_FEATURE (0x03)	usb_hcdc_feature_selector_t	0x0000	Data length	usb_hcdc_commfeature_t
0x21	CLEAR_COMM_FEATURE (0x04)	usb_hcdc_feature_selector_t	0x0000	Data length	None
0x21	SET_LINE_CODING (0x20)	0x0000	0x0000	0x0000	usb_hcdc_linecoding_t

0xA1	GET_LINE_CODING (0x21)	0x0000	0x0000	0x0007	usb_hcdc_linecoding_t
0x21	SET_CONTROL_LINE_STATE (0x22)	usb_hcdc_controllinestate_t	0x0000	0x0000	None
0x21	SEND_BREAK (0x23)	usb_hcdc_breakduration_t	0x0000	0x0000	None

ACM Notifications from Device to Host

The following class notifications are supported:

Notification	Code	Description
RESPONSE_AVAILABLE	0x01	Response to GET_ENCAPSULATED_RESPONSE
SERIAL_STATE	0x20	Notification of serial line state

The data types returned are as follows:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	RESPONSE_AVAILABLE (0x01)	0x0000	0x0000	0x0000	None
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0002	usb_hcdc_serialstate_t

Note

The host is notified with SERIAL_STATE whenever a change in the UART port state is detected.

Limitations

This driver is subject to the following limitations:

- Suspend is not supported when a data transfer is in progress. Confirm that data transfer has completed before executing suspend.
- Use of compound USB devices with CDC class support is not supported.
- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- This driver does not support Low-speed.
- This driver does not support simultaneous operation with the other device class.

Examples

USB HCDC Loopback Example

The main functions of the HCDC loopback example are as follows:

1. Virtual UART control settings are configured by transmitting the class request

SET_LINE_CODING to the CDC device.
2. Sends receive (Bulk In transfer) requests to a CDC peripheral device and receives data.
3. Loops received data back to the peripheral by means of Bulk Out transfers.

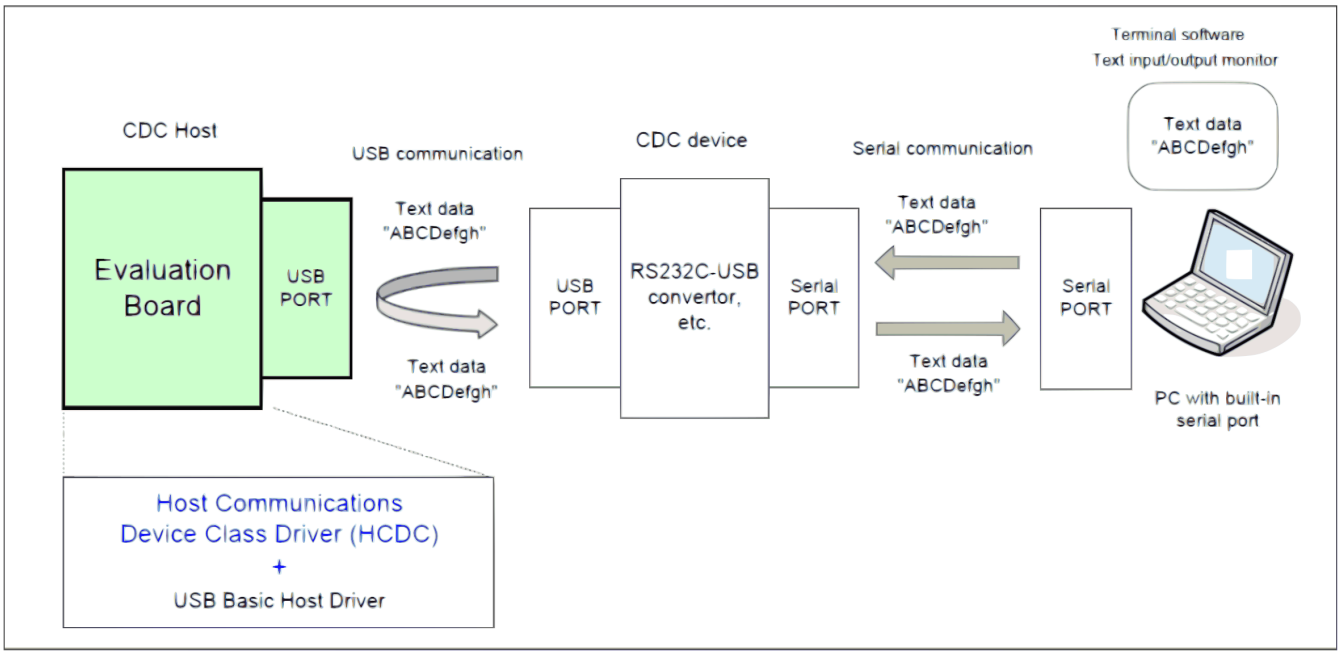


Figure 113: Data Transfer (Loopback)

The main loop performs loopback processing in which data received from a CDC peripheral device is transmitted unaltered back to the peripheral.

```
#define SET_LINE_CODING (USB_CDC_SET_LINE_CODING | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define GET_LINE_CODING (USB_CDC_GET_LINE_CODING | USB_DEV_TO_HOST | USB_CLASS |
USB_INTERFACE)
#define SET_CONTROL_LINE_STATE (USB_CDC_SET_CONTROL_LINE_STATE | USB_HOST_TO_DEV |
USB_CLASS | USB_INTERFACE)
#define COM_SPEED (9600U)
#define COM_DATA_BIT (8U)
#define COM_STOP_BIT (0)
#define COM_PARITY_BIT (0)
#define LINE_CODING_LENGTH (7)
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
```

```
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : usb_hcdc_example
 * Description : Host CDC application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_hcdc_example (void)
{
 usb_status_t    event;
    usb_event_info_t event_info;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
```

```
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 /* Handle the received event (if any) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 /* Configure virtual UART settings */
                set_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
 break;
 case USB_STATUS_READ_COMPLETE:
 if (USB_CLASS_HCDC == event_info.type)
      {
 if (event_info.data_size > 0)
      {
 /* Send the received data back to the connected peripheral */
                       g_usb_on_usb.write(&g_basic0_ctrl, g_snd_buf,
event_info.data_size, USB_DEVICE_ADDRESS_1);
                  }
 else
      {
 /* Send the data reception request when the zero-length packet is received. */
                       g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
                  }
               }
 else /* USB_HCDCC */
      {
 /* Control Class notification "SerialState" receive start */
                  g_usb_on_usb.read(&g_basic0_ctrl,
                                    (uint8_t *) &g_serial_state,
                                    USB_HCDC_SERIAL_STATE_MSG_LEN,
                                    USB_DEVICE_ADDRESS_1);
```

```
                }
 break;
 case USB_STATUS_WRITE_COMPLETE:
 /* Start receive operation */
                g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
 break;
 case USB_STATUS_REQUEST_COMPLETE:
 if (USB_CDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Set virtual RTS/DTR signal state */
                   set_control_line_state(&g_basic0_ctrl, event_info.device_address);
/* CDC Class request "SetControlLineState" */
                }
 /* Check Complete request "SetControlLineState" */
 else if (USB_CDC_SET_CONTROL_LINE_STATE == (event_info.setup.request_type &
USB_BREQUEST))
      {
 /* Read back virtual UART settings */
                   get_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
                }
 else if (USB_CDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Now that setup is complete, start loopback operation */
                   g_usb_on_usb.read(&g_basic0_ctrl, g_snd_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
                }
 else
      {
 /* Unsupported request */
                }
 break;
 default:
```

```
 /* Other event */
 break;
       }
    }
} /* End of function usb_hcdc_example() */
/******************************************************************************
 * Function Name : set_control_line_state
 * Description : Send the class request (SetControLineState) to CDC device.
 * Arguments : device_address : device address of CDC device.
 * Return value : none
 ******************************************************************************/
void set_control_line_state (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = SET_CONTROL_LINE_STATE; /*
bRequestCode:SET_CONTROL_LINE_STATE, bmRequestType */
    setup.request_value = 0x0000;                /* wValue:Zero */
    setup.request_index = 0x0000;                /* wIndex:Interface */
    setup.request_length = 0x0000;                /* wLength:Zero */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_usb_dummy,
device_address);
} /* End of function set_control_line_state() */
/******************************************************************************
 * Function Name : set_line_coding
 * Description : Send the class request (SetLineCoding) to CDC device.
 * Arguments : device_address : device address of CDC device.
 * Return value : none
 ******************************************************************************/
void set_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    g_com_parm.dwdte_rate   = (usb_hcdc_line_speed_t) COM_SPEED;
    g_com_parm.bchar_format = (usb_hcdc_stop_bit_t) COM_STOP_BIT;
    g_com_parm.bparity_type = (usb_hcdc_parity_bit_t) COM_PARITY_BIT;
```

```
    g_com_parm.bdata_bits   = (usb_hcdc_data_bit_t) COM_DATA_BIT;
    setup.request_type   = SET_LINE_CODING;    /* bRequestCode:SET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
} /* End of function set_line_coding() */
/******************************************************************************
 * Function Name : get_line_coding
 * Description : Send the class request (GetLineCoding) to CDC device.
 * Arguments : device_address : device address of CDC device.
 * Return value : none
 ******************************************************************************/
void get_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = GET_LINE_CODING;    /* bRequestCode:GET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
} /* End of function get_line_coding() */
```

Function Documentation

◆ R_USB_HCDC_ControlDataRead()

fsp_err_t R_USB_HCDC_ControlDataRead (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *device_address*)

Read Control Data.(CDC Interrupt IN data)

Return values

FSP_SUCCESS	Successfully completed (Data read request completed).
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Data receive request already in process for USB device with same device address.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1. Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_HCDC_SpecificDeviceRegister()

fsp_err_t R_USB_HCDC_SpecificDeviceRegister (usb_ctrl_t *const *p_api_ctrl*, uint16_t *vendor_id*, uint16_t *product_id*)

Register the specified vendor class device in the device table.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_HCDC_DeviceInfoGet()

fsp_err_t R_USB_HCDC_DeviceInfoGet (usb_ctrl_t *const *p_api_ctrl*, usb_hcdc_device_info_t * *p_info*, uint8_t *device_address*)

Get the VID, PID and subclass code of the connected device.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

(1). Interrupt function.
(2). Callback function (for RTOS).

5.2.6.14 USB Host Human Interface Device Class Driver (r_usb_hhid)

Modules » Connectivity

Functions

fsp_err_t	R_USB_HHID_TypeGet (usb_ctrl_t *const p_ctrl, uint8_t *p_type, uint8_t device_address) Get HID protocol.(USB Mouse/USB Keyboard/Other Type.) More...
fsp_err_t	R_USB_HHID_MaxPacketSizeGet (usb_ctrl_t *const p_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address) Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT). More...

Detailed Description

This module provides a USB Host Human Interface Device Class Driver (HHID). It implements the USB HHID Interface.

Overview

The r_usb_hhid module combines with the r_usb_basic module to provide a USB Host Human Interface Device Class (HHID) driver. The HHID driver conforms to the USB Human Interface Device

class specifications and implements communication with a HID device.

Features

The r_usb_hhid module has the following key features:

- Data communication with a connected HID device (USB mouse, keyboard etc.)
- Issuing of HID class requests to a connected HID device
- Supports Interrupt OUT transfer

Configuration

Build Time Configurations for r_usb_hhid

The following build time configurations are defined in fsp_cfg/r_usb_hhid_cfg.h:

Configuration	Options	Default	Description
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the pipe number to use for input interrupt events.
Interrupt Out Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE9	Select the pipe number to use for output interrupt events.

Configurations for Connectivity > USB HHID (r_usb_hhid)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HHID (r_usb_hhid).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_hhid0	Module name.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Note

This driver is not guaranteed to provide USB HID operation in all scenarios. The developer must verify correct operation when connected to the targeted USB peripherals.

Class Requests

The class requests supported by this driver are shown below:

Request	Code	Description
USB_GET_REPORT	0x01	Receives a report from the HID device.
USB_SET_REPORT	0x09	Sends a report to the HID device.
USB_GET_IDLE	0x02	Receives a duration (time) from the HID device.
USB_SET_IDLE	0x0A	Sends a duration (time) to the HID device.
USB_GET_PROTOCOL	0x03	Reads a protocol from the HID device.
USB_SET_PROTOCOL	0x0B	Sends a protocol to the HID device.
USB_GET_REPORT_DESCRIPTOR	0x06	Requests a report descriptor.
USB_GET_HID_DESCRIPTOR	0x06	Requests a HID descriptor.

Data Format

The boot protocol data format of data received from the keyboard or mouse through interrupt-IN transfers is shown below:

offset	Keyboard (8 Bytes)	Mouse (3 Bytes)
0 (Top Byte)	Modifier keys	b0 : Button 1 b1 : Button 2 b2 : Button 3 b3-b7 : Reserved
+1	Reserved	X displacement
+2	Keycode 1	Y displacement
+3	Keycode 2	-
+4	Keycode 3	-
+5	Keycode 4	-
+6	Keycode 5	-
+7	Keycode 6	-

Limitations

- The HID driver does not analyze the report descriptor. This driver determines the report format from the interface protocol.
- This driver does not support DMA transfers.
- This driver does not support High-speed.
- The transfer rates of Full-speed and Low-speed are the same when the max packet sizes of Full-speed and Low-speed are the same.
- This driver does not support simultaneous operation with the other device class.

Examples

USB HHID Example

The main functions of the application are as follows:

1. Performs enumeration and initialization of HID devices.
2. Transfers data to and from a connected HID device (mouse or keyboard). Data received from the device is read and discarded.
3. When an RTOS is used, the USB driver calls the callback (usb_apl_callback) in order to pass events to the main loop through a queue.

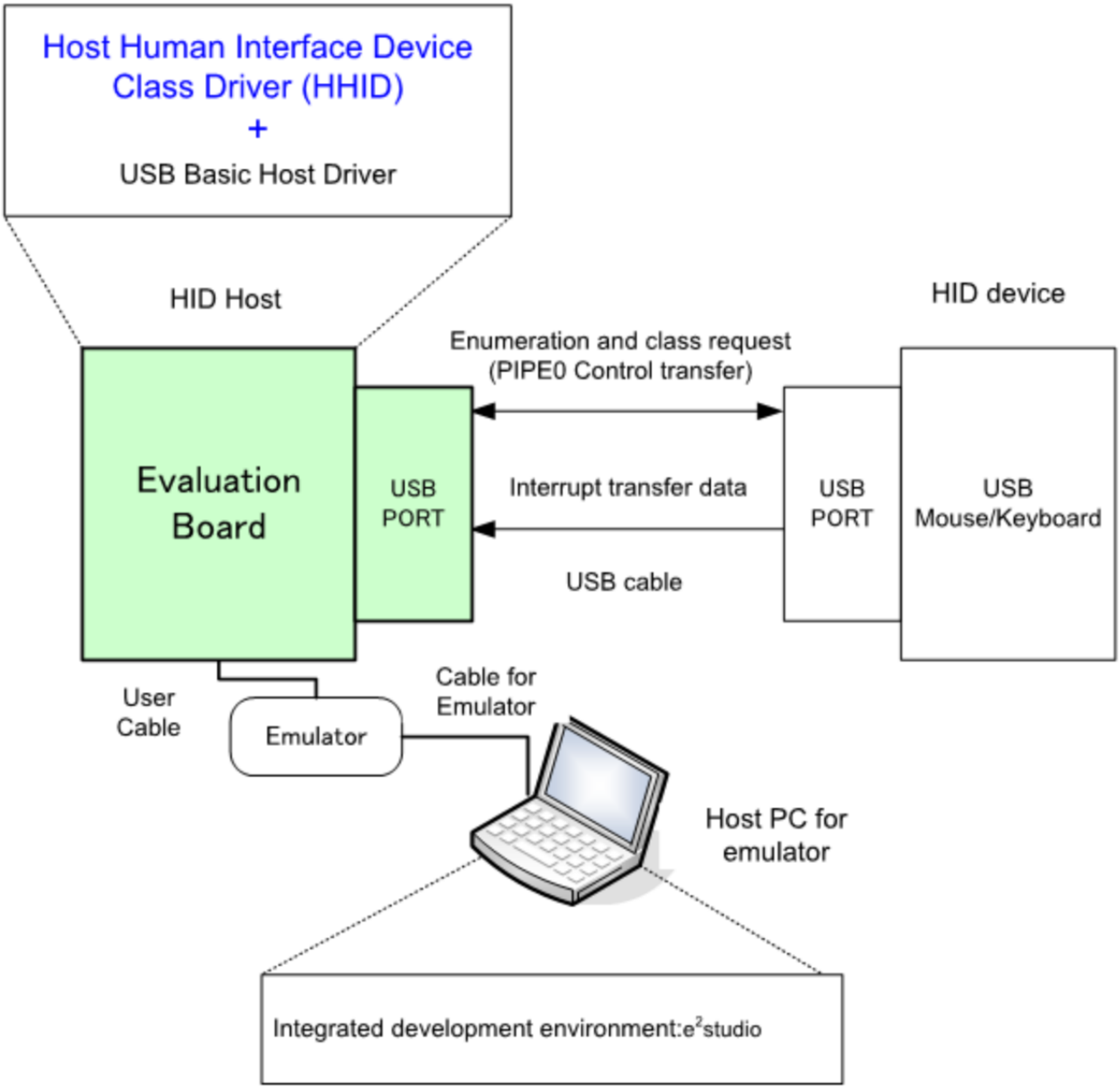


Figure 114: Example Operating Environment

Application Processing (for RTOS)

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing performed by the loop is shown below.

1. When a USB-related event has completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task (APL) is notified of the USB completion event using the real-time OS functionality.
2. In APL, information regarding the USB completion event was notified from the callback function is retrieved using the real-time OS functionality.
3. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_CONFIGURED, APL sends the class request (SET_PROTOCOL) to the HID device.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_REQUEST_COMPLETE, APL performs a data reception request to receive data transmitted from the HID device by calling the R_USB_Read function.
5. The above processing is repeated.

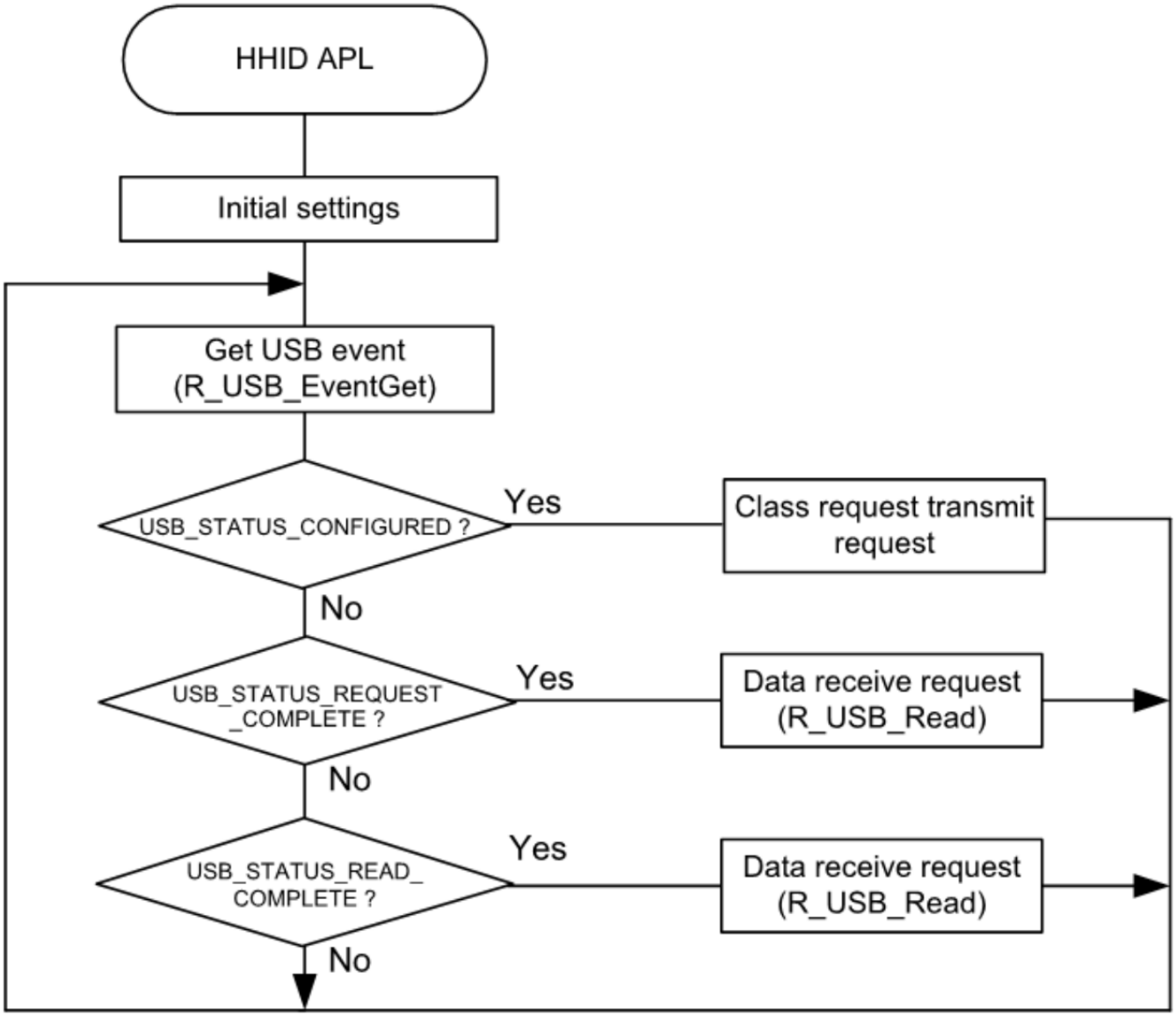


Figure 115: Main Loop (Normal mode)

Application Processing (for Non-OS)

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing of the main loop is presented below.

1. When the R_USB_GetEvent function is called after an HID device attaches to the USB host and enumeration completes, USB_STATUS_CONFIGURED is set as the return value. When the APL confirms USB_STATUS_CONFIGURED, it calls the R_USB_Write function to request transmission of data to the HID device.
2. When the R_USB_GetEvent function is called after sending of class request SET_PROTOCOL

to the HID device has completed, USB_STATUS_REQUEST_COMPLETE is set as the return value. When the APL confirms USB_STATUS_REQUEST_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.

3. When the R_USB_GetEvent function is called after reception of data from the HID device has completed, USB_STATUS_READ_COMPLETE is set as the return value. When the APL confirms USB_STATUS_READ_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
4. The processing in step 3, above, is repeated.

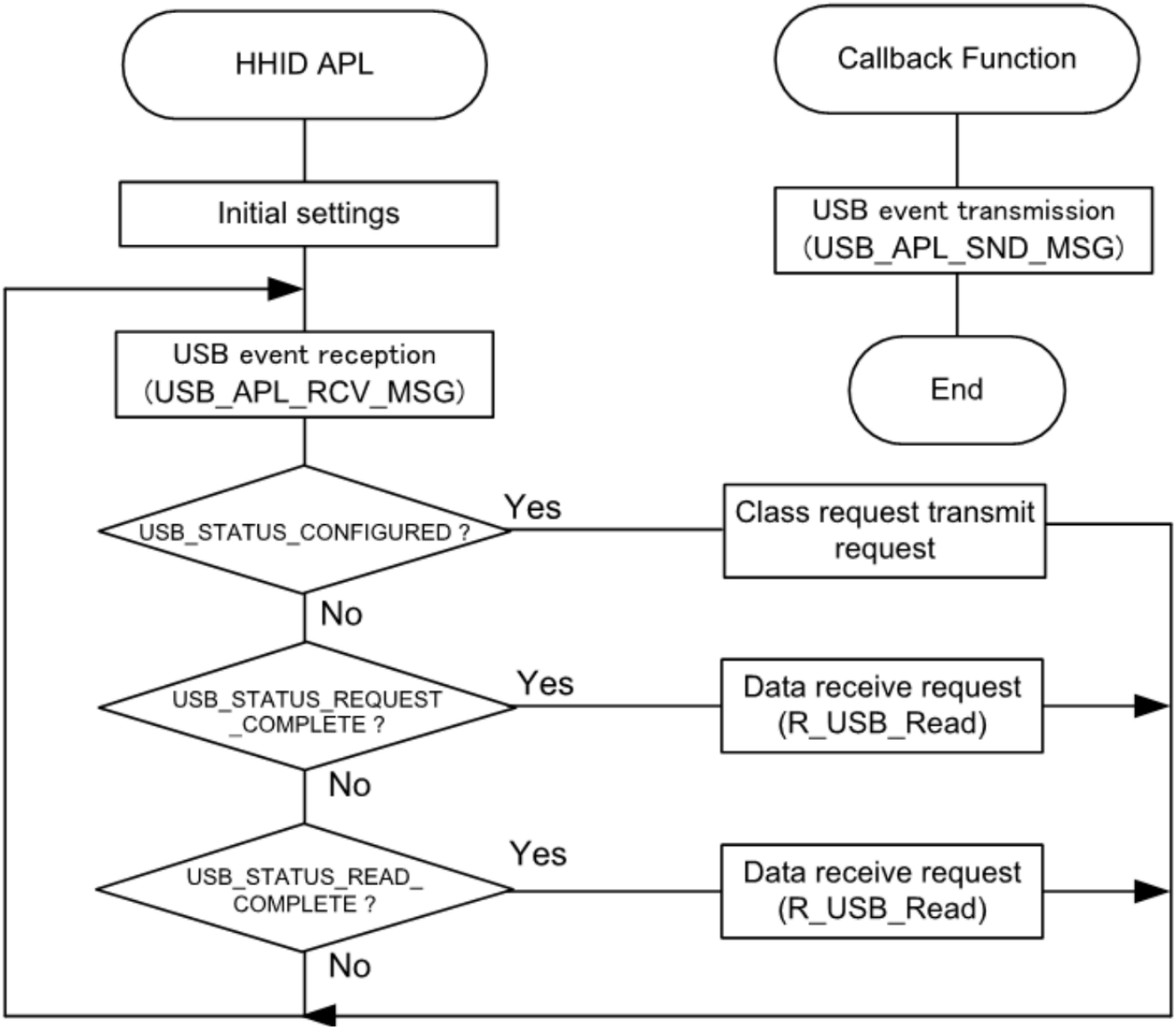


Figure 116: Main Loop (Normal mode)

```
/*****************************************************************************
```

```
 * Macro definitions
 ******************************************************************************/
#define SET_PROTOCOL (USB_HID_SET_PROTOCOL | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define BOOT_PROTOCOL (0)
#define USB_FS_DEVICE_ADDRESS_1 (1)
/******************************************************************************
 * Private global variables and functions
 ******************************************************************************/
static const usb_hhid_api_t g_hhid_on_usb =
{
    .typeGet          = R_USB_HHID_TypeGet,
    .maxPacketSizeGet = R_USB_HHID_MaxPacketSizeGet,
};
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : usb_hhid_example
 * Description : Host HID application main process
```

```
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_hhid_example (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif  /* (BSP_CFG_RTOS == 2) */
 usb_status_t    event;
    usb_event_info_t event_info;
    uint16_t         offset = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event); /* Get event code */
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                g_hhid_on_usb.typeGet(&g_basic0_ctrl, &g_hid_type,
USB_FS_DEVICE_ADDRESS_1);
                g_hhid_on_usb.maxPacketSizeGet(&g_basic0_ctrl, &g_mxps, USB_HID_IN,
USB_FS_DEVICE_ADDRESS_1);
 /* Send the HID request (SetProtocol) to HID device */
                set_protocol(&g_basic0_ctrl, BOOT_PROTOCOL, USB_FS_DEVICE_ADDRESS_1);
 break;
            }
 case USB_STATUS_READ_COMPLETE:
```

```
     {
                offset = hid_memcpy(g_store_buf, g_buf, offset, g_mxps);
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
 break;
           }
 case USB_STATUS_REQUEST_COMPLETE:
     {
 if (USB_HID_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
              }
 break;
           }
 default:
     {
 break;
           }
      }
    }
} /* End of function usb_hhid_example() */
/******************************************************************************
 * Function Name : set_protocol
 * Description : Sending SetProtocol request to HID device
 * Arguments : usb_ctrl_t *p_ctrl : Pointer to usb_instance_ctrl_t structure.
 * : uint8_t ptorocol: Protocol Type
 * : uint8_t device_address: Device address that sends this request
 * Return value : none
 ******************************************************************************/
static void set_protocol (usb_instance_ctrl_t * p_ctrl, uint8_t protocol, uint8_t
device_address)
{
    usb_setup_t setup;
```

```
    setup.request_type   =
SET_PROTOCOL;                                                          /*
bRequestCode:SET_PROTOCOL, bmRequestType */
    setup.request_value =
protocol;                                                              /* wValue:
Protocol Type */
    setup.request_index =
0x0000;                                                                /*
wIndex:Interface */
    setup.request_length =
0x0000;                                                                /* wLength:Zero
*/
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_setup_data,
device_address); /* Request Control transfer */
} /* End of function set_protocol() */
/******************************************************************************
 * Function Name : hid_memcpy
 * Description : Copy received hhid data to the application buffer
 * Arguments : uint8_t *p_dest : Pointer to application buffer
 * : uint8_t *p_src : Pointer to received buffer
 * : uint16_t offset : Application buffer offset
 * : uint16_t size : Size of receiced hhid data
 * Return value : uint16_t offset + i: Offset
 ******************************************************************************/
static uint16_t hid_memcpy (uint8_t * p_dest, uint8_t * p_src, uint16_t offset,
uint16_t size)
{
    uint16_t i;
 for (i = 0; i < size; i++)
    {
 if (BUFSIZE == (offset + i))
      {
            offset = 0;
       }
```

```
        *(p_dest + offset + i) = *(p_src + i);
    }
 return (uint16_t) (offset + i);
} /* End of function hid_memcpy() */
```

Function Documentation

◆ R_USB_HHID_TypeGet()

fsp_err_t R_USB_HHID_TypeGet (usb_ctrl_t *const *p_ctrl*, uint8_t * *p_type*, uint8_t *device_address*)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_HHID_MaxPacketSizeGet()

fsp_err_t R_USB_HHID_MaxPacketSizeGet (usb_ctrl_t *const *p_ctrl*, uint16_t * *p_size*, uint8_t *direction*, uint8_t *device_address*)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

5.2.6.15 USB Host Mass Storage Class Driver (r_usb_hmsc)

Modules » Connectivity

Detailed Description

This module provides a USB Host Mass Storage Class (HMSC) driver. It implements the USB HMSC Interface.

Overview

The r_usb_hmsc module, when used in combination with the r_usb_basic module, operates as a USB Host Mass Storage Class (HMSC) driver. It is built on the USB Mass Storage Class Bulk-Only Transport (BOT) protocol. It is possible to communicate with BOT-compatible USB storage devices by combining this module with a file system and storage device driver.

Note

This module should be used in combination with the FreeRTOS+FAT File System.

Features

The r_usb_hmsc module has the following key features:

- Checking of connected USB storage devices to determine whether or not operation is supported
- Storage command communication using the BOT protocol
- Support for SFF-8070i (ATAPI) USB mass storage subclass
- Sharing of a single pipe for IN/OUT directions or multiple devices
- Supports up to 4 connected USB storage devices

Class Requests

The class requests supported by this driver are shown below.

Request	Description
GetMaxLun	Gets the maximum number of units that are supported.
MassStorageReset	Cancels a protocol error.

Storage Commands

This driver supports the following storage commands:

- TEST_UNIT_READY
- MODE_SELECT10
- MODE_SENSE10
- PREVENT_ALLOW
- READ_FORMAT_CAPACITY
- READ10
- WRITE10

Configuration

Refer to the USB (r_usb_basic) module.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Warning

Due to the wide variety of USB mass storage device implementations, this driver is not guaranteed to work with all devices. When implementing the driver it is important to verify correct operation with the mass storage devices that the end user is expected to use.

Multi Port

This driver supports simultaneous operation with Peripheral Communication Device Class(PCDC). If the user are using MCU that supports 2 USB modules, such as RX74M, the user can run HMSC on one USB module and PCDC on the other. This driver does not support simultaneous operation using device classes other than PCDC.

For Bare Metal

1. To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.
2. In RX configurator, enter the appropriate values in the Main stack size and Heap size fields. The figure below is an example of the EK-RX74M board.

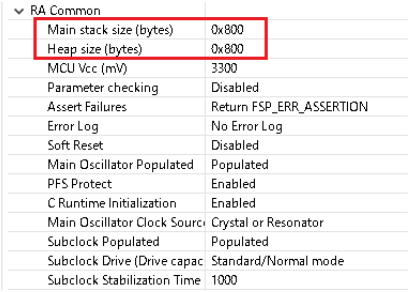


Figure 117: BSP Setting

1. In the Bare Metal version, specify "NULL" in the Callback item.

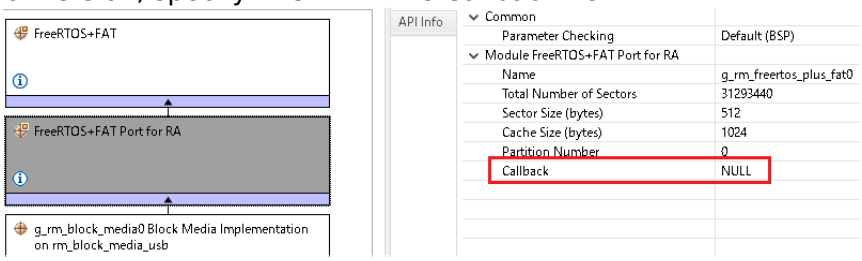


Figure 118: For Bare Metal Setting

Limitations

1. Some MSC devices may be unable to connect because they are not recognized as storage devices.
2. MSC devices that return values of 1 or higher in response to the GetMaxLun command (mass storage class command) are not supported.
3. A maximum of 4 USB storage devices can be connected.
4. Only USB storage devices with a sector size of 512 bytes can be connected.
5. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.
6. The continuous transfer mode cannot be used when using DMA.
7. This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
8. This driver does not support Low-speed.
9. The current USB driver does not support FreeRTOS.

Examples

USB HMSC Example

Example Operating Environment

The following shows an example operating environment for the HMSC.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

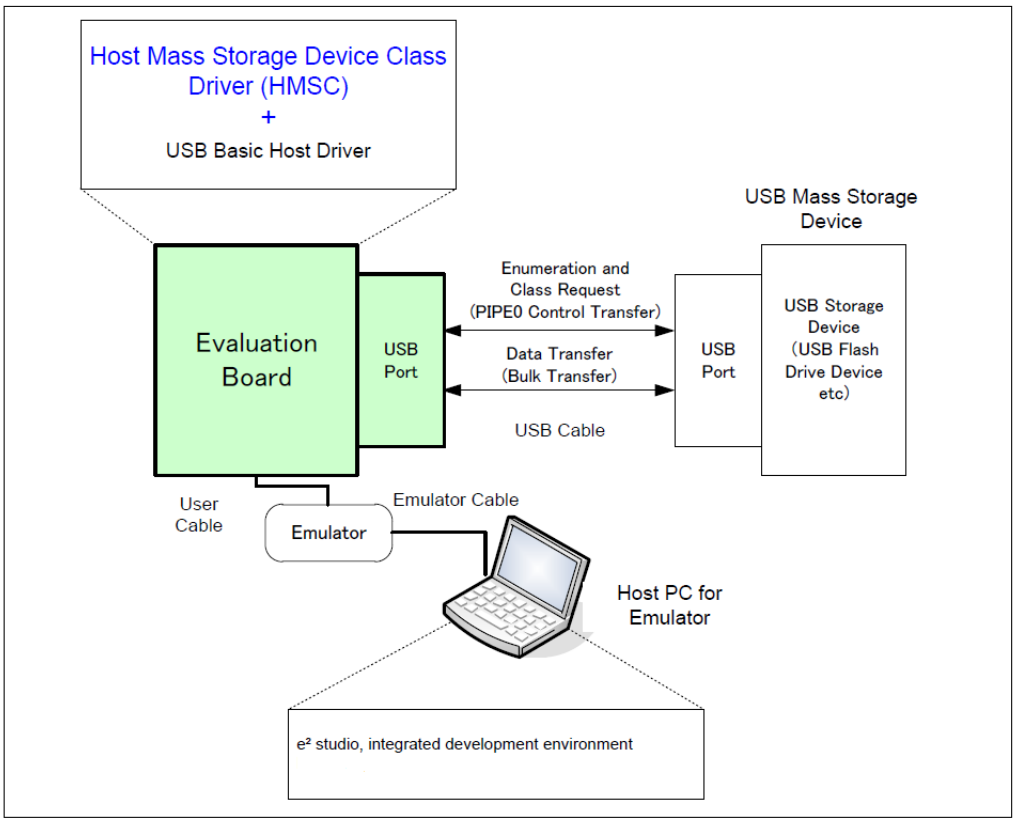


Figure 119: Example Operating Environment

Application Specifications

The main functions of the application are as follows:

1. Performs enumeration and drive recognition processing on MSC devices.
2. After the above processsing finisihes, the application writes the file to the MSC device once.
3. After writing the above file, the APL repeatedly reads the file. It continues to read the file repeatedly until the switch is pressed again.

Application Processing (for RTOS)

This application has two tasks. An overview of the processing in these two tasks is provided below.

usb_apl_task

1. After start up, MCU pin setting, USB controller initialization, and application program initialization are performed.
2. The MSC device is attached to the kit. When enumeration and drive recognition processing have completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task is notified of the USB completion event using the FreeRTOS functionality.
3. In the application task, information regarding the USB completion event about which notification was received from the callback function is retrieved using the real-time OS functionality.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_CONFIGURED then, based on the USB completion event, the MSC device is mounted and the file is written to the MSC device.
5. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_DETACH, the application initializes the variables for state management.

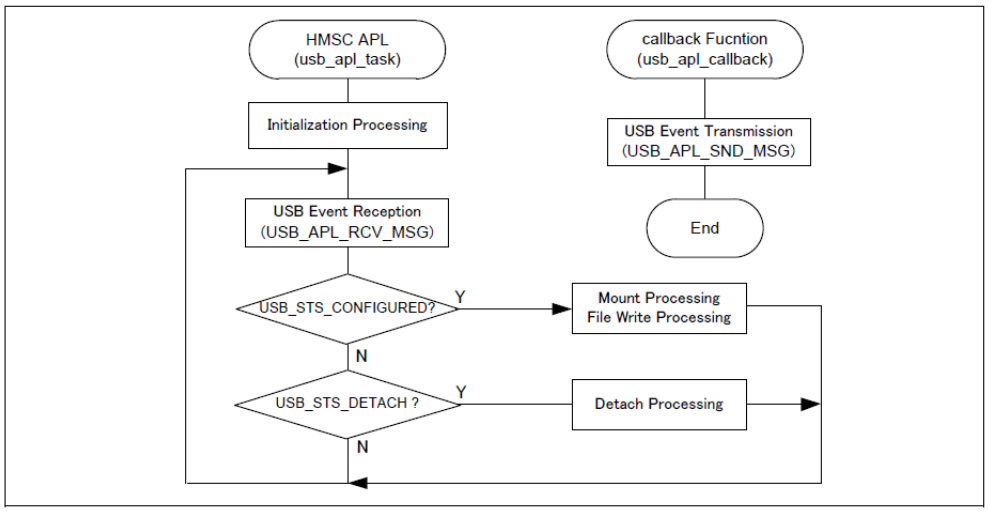


Figure 120: usb_apl_task

file_read_task

Of the application tasks usb_apl_task and file_read_task, file_read_task is processed while usb_apl_task is in the wait state. This task performs file read processing on the file that was written to the MSC device.

Example Code

```
 #if 0 // Not File system
  #define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
  #define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
  #define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
  #define RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB
typedef enum
{
    STATE_ATTACH, STATE_DATA_READY, STATE_DATA_WRITE, STATE_FILE_READ, STATE_DETACH,
STATE_ERROR,
} state_t;
extern rm_freertos_plus_fat_instance_ctrl_t g_rm_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_rm_freertos_plus_fat0_cfg;
// @@extern const rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat0_disk_cfg;
uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
static uint16_t g_state = STATE_DETACH;
void usb_hmsc_baremetal_example (void)
{
    uint16_t i;
    uint16_t k;
 fsp_err_t err;
    FF_FILE * pxSourceFile;
    FF_Disk_t disk;
 rm_freertos_plus_fat_device_t device;
 usb_status_t    event;
    usb_event_info_t event_info;
    FF_Error_t       ff_err;
 size_t          size_return;
 int             close_err;
 rm_block_media_usb_instance_ctrl_t * p_instance_ctrl;
 for (i = 0; i < RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES; i++)
    {
       g_file_data[i] = (uint8_t) i;
```

```
    }
 /* Open media driver.*/
 RM_FREERTOS_PLUS_FAT_Open(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_cfg);
/* When using USB media, enable RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB macro. */
  #ifdef RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
                p_instance_ctrl                = event_info.p_context;
                p_instance_ctrl->device_address = event_info.device_address;
 RM_FREERTOS_PLUS_FAT_MediaInit(&g_rm_freertos_plus_fat0_ctrl, &device);
 /* Initialize one disk for each partition used in the application. */
 RM_FREERTOS_PLUS_FAT_DiskInit(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_disk_cfg, &disk);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
                FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 /* Add the disk to the file system. */
                FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
                pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
 /* Write file data. */
                ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
 /* Close the file. */
                ff_fclose(pxSourceFile);
                g_state = STATE_FILE_READ;
```

```
 break;
      }
 case USB_STATUS_DETACH:
      {
                g_state = STATE_DETACH;
 RM_FREERTOS_PLUS_FAT_DiskDeinit(&g_rm_freertos_plus_fat0_ctrl, &disk);
 break;
      }
 default:
      {
 break;
      }
      }
 if (STATE_FILE_READ == g_state)
      {
            pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "r");
 for (k = 0; k < RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES; k++)
      {
                g_read_buffer[k] = (uint8_t) 0;
      }
 /* Read file data. */
            size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1,
pxSourceFile);
 /* Close the file. */
            close_err = ff_fclose(pxSourceFile);
      }
    }
  #endif
}
 #endif // Not File system
```

5.2.6.16 USB Host Vendor class (r_usb_hvnd)

Modules » Connectivity

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

USB Host Vendor class works by combining r_usb_basic module.

How to Configuration

The following shows FSP configuration procedure for USB Host Vendor class.

- Select [New Stack]->[Connectivity]->[USB Host Vendor class (r_usb_hvnd)].

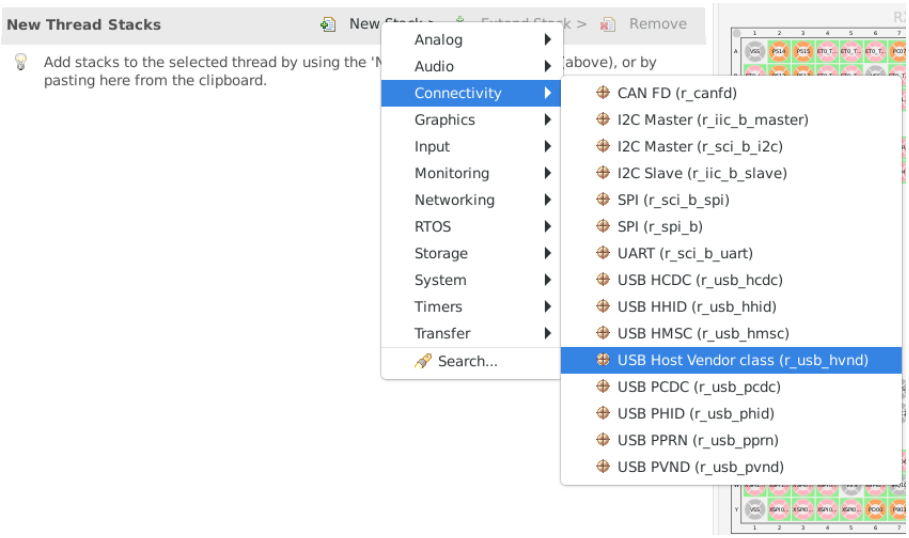


Figure 121: Select USB Host Vendor Class

- The following is displayed when selecting [USB Host Vendor class (r_usb_hvnd)]. The user does not specify USB pipe number in Vendor class.

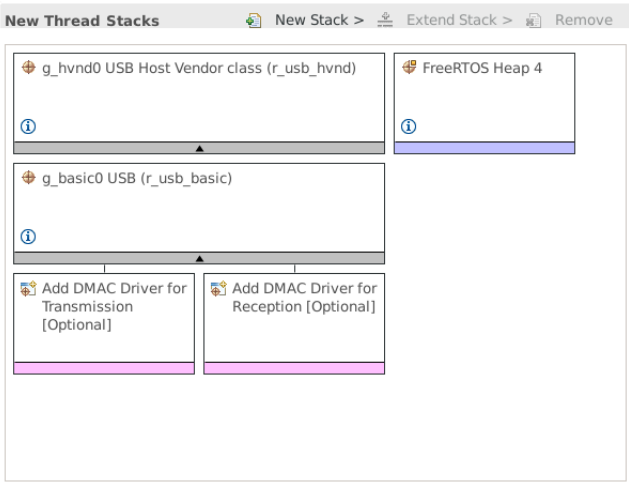


Figure 122: USB Host Vendor Class Stack

API

Use the following APIs in Host Vendor class application program.

- For Data Transfer
  Use the following APIs for data transfer for Bulk transfer or Interrupt transfer.

  1. R_USB_PipeRead()
  2. R_USB_PipeWrite()
  3. R_USB_PipeStop()

- For Control Transfer
  Use the following API for the class request processing.

  1. R_USB_HostControlTransfer()

- For USB Pipe Information
  The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class. Use the following APIs to get the allocated USB pipe information.

  1. R_USB_UsedPipesGet()
  2. R_USB_PipeInfoGet()

USB PIPE Allocation

The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class.
The USB PIPE related to the Endpoint Descriptor are allocated in order from USB PIPE1 according to the description order of the Endpoint Descriptor.

Examples

This application program processes the follwoing after the enumeration completes with USB device.

1. Getting USB Pipe Infomattion

2. Vendor Class Request Processing
3. Loopback processing of bulk transfer and interrupt transfer.

```
/*****************************************************************************
**
 * Macro definitions
 *****************************************************************************
**/
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
#define SET_VENDOR_NO_DATA (USB_SET_VENDOR_NO_DATA | USB_HOST_TO_DEV |
USB_VENDOR | USB_INTERFACE)
#define SET_VENDOR (USB_SET_VENDOR | USB_HOST_TO_DEV | USB_VENDOR |
USB_INTERFACE)
#define GET_VENDOR (USB_GET_VENDOR | USB_DEV_TO_HOST | USB_VENDOR |
USB_INTERFACE)
#if (BSP_CFG_RTOS == 2)
/*****************************************************************************
**
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 *****************************************************************************
**/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
```

```
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
**
 * Function Name : usb_pvnd_example
 * Description : Peripheral Vendor Class application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
void usb_pvnd_example (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
 usb_status_t    event;
    usb_event_info_t event_info;
    uint8_t          bulk_out_pipe = 0; /* Bulk Out Pipe       */
    uint8_t          bulk_in_pipe = 0; /* Bulk In Pipe       */
    uint8_t          int_out_pipe = 0; /* Interrupt Out Pipe   */
    uint8_t          int_in_pipe   = 0; /* Interrupt In Pipe   */
    uint16_t   buf_type = 0;
    uint8_t    pipe      = 0;
    uint8_t    is_zlp[2] = {0, 0};
    uint16_t   used_pipe = 0;
    usb_pipe_t pipe_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event);
```

```
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                buffer_init();
                is_zlp[0] = 0;
                is_zlp[1] = 0;
 /* Get USB Pipe Information */
                g_usb_on_usb.usedPipesGet(&g_basic0_ctrl, &used_pipe,
ADDRESS1);
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
      {
 if (0 != (used_pipe & (1 << pipe)))
      {
                        g_usb_on_usb.pipeInfoGet(&g_basic0_ctrl, &pipe_info,
pipe);
 if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
      {
 /* Out Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                            buf_type      = BUF_BULK;
                            bulk_out_pipe = pipe;
                        }
 else
      {
                            buf_type    = BUF_INT;
                            int_out_pipe = pipe;
                        }
                    }
 else
      {
 /* In Transfer */
```

```
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                                 buf_type    = BUF_BULK;
                                 bulk_in_pipe = pipe;
                              }
 else
      {
                                 buf_type    = BUF_INT;
                                 int_in_pipe = pipe;
                              }
                           }
                        }
                     }
                  }
 /* Send Vendor Class Request */
                  class_request_set_vendor_no_data(&g_basic0_ctrl,
event_info.device_address);
 break;
             }
 case USB_STATUS_READ_COMPLETE:
      {
 if (FSP_ERR_USB_FAILED != event_info.status)
      {
 if (bulk_in_pipe == event_info.pipe)
      {
                           buf_type = BUF_BULK;
                           pipe    = bulk_out_pipe;
                        }
 else if (int_in_pipe == event_info.pipe)
      {
                           buf_type = BUF_INT;
                           pipe    = int_out_pipe;
                        }
 else
      {
```

```
while (1)
     {
                              ;
                         }
                    }
                    buffer_check(buf_type, event_info.data_size);
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[buf_type][0], event_info.data_size, pipe);
                }
break;
           }
case USB_STATUS_WRITE_COMPLETE:
     {
if (bulk_out_pipe == event_info.pipe)
     {
                    buf_type = BUF_BULK;
if (1 == is_zlp[buf_type])
     {
                         pipe = bulk_in_pipe;
                    }
                }
else if (int_out_pipe == event_info.pipe)
     {
                    buf_type = BUF_INT;
if (1 == is_zlp[buf_type])
     {
                         pipe = int_in_pipe;
                    }
                }
else
     {
/* Nothing */
                }
if (1 == is_zlp[buf_type])
```

```
      {
                    is_zlp[buf_type] = 0;
                    buffer_clear(buf_type);
                    g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[buf_type][0],
BUF_SIZE, pipe);
                }
 else
      {
                    is_zlp[buf_type] = 1;
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl, 0, 0,
event_info.pipe); /* Send ZLP */
                }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_SET_VENDOR_NO_DATA == (event_info.setup.request_type & USB_BREQUEST
))
      {
                    class_request_set_vendor(&g_basic0_ctrl,
event_info.device_address);
                }
 else if (USB_SET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                    class_request_get_vendor(&g_basic0_ctrl,
event_info.device_address);
                }
 else if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                   buffer_init();
 /* Bulk Out Transfer */
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[BUF_BULK][0], (BUF_SIZE - USB_APL_MXPS),
                                            bulk_out_pipe);
```

```
 /* Interrupt Out Transfer */
                   g_usb_on_usb.pipeWrite(&g_basic0_ctrl, &g_buf[BUF_INT][0],
(BUF_SIZE - USB_APL_MXPS), int_out_pipe);
                }
 else
      {
 /* Unsupported request */
                }
 break;
            }
 case USB_STATUS_DETACH:
      {
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_pvnd_example() */
/*****************************************************************************
**
 * Function Name : class_request_set_vendor
 * Description : Send Vendor Class Request (SET_VENDOR) to USB device.
 * Arguments : none
 * Return value : none
 *****************************************************************************
**/
static void class_request_set_vendor (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
    uint16_t    i;
```

```
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = (uint8_t) i;
    }
    setup.request_type   = SET_VENDOR; /* bRequestCode:SET_VENDOR,
bmRequestType */
    setup.request_value = 0x0000;    /* wValue:Zero */
    setup.request_index = 0x0000;    /* wIndex:Interface */
    setup.request_length = REQ_SIZE;   /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
} /* End of function class_request_set_vendor() */
/******************************************************************************
**
 * Function Name : class_request_set_vendor_no_data
 * Description : Send Vendor Class Request (SET_VENDOR_NO_DATA) to USB
device.
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
static void class_request_set_vendor_no_data (usb_instance_ctrl_t * p_ctrl,
uint8_t device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = (uint8_t) i;
    }
    setup.request_type   = SET_VENDOR_NO_DATA; /*
bRequestCode:SET_VENDOR_NO_DATA, bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
```

```
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = 0x0000;             /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
} /* End of function class_request_set_vendor_no_data() */
/*****************************************************************************
**
 * Function Name : class_request_get_vendor
 * Description : Send Vendor Class Request (GET_VENDOR) to USB device.
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
static void class_request_get_vendor (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = 0;
    }
    setup.request_type   = GET_VENDOR; /* bRequestCode:GET_VENDOR,
bmRequestType */
    setup.request_value = 0x0000;    /* wValue:Zero */
    setup.request_index = 0x0000;    /* wIndex:Interface */
    setup.request_length = REQ_SIZE;   /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
} /* End of function class_request_get_vendor() */
/*****************************************************************************
**
```

```
 * Function Name : buffer_init
 * Description : buffer initialization
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
static void buffer_init (void)
{
    uint16_t i;
    uint16_t j;
 for (j = 0; j < 2; j++)
    {
 for (i = 0; i < BUF_SIZE; i++)
      {
            g_buf[j][i] = (uint8_t) i;
       }
    }
} /* End of function buffer_init() */
/******************************************************************************
**
 * Function Name : buffer_check
 * Description : buffer check
 * Arguments : buf_type : buffer number
 * Return value : none
 ******************************************************************************
**/
static void buffer_check (uint16_t buf_type, uint32_t size)
{
    uint16_t i;
 for (i = 0; i < (uint16_t) size; i++)
    {
 if ((uint8_t) (i & USB_VALUE_FF) != g_buf[buf_type][i])
      {
 while (1)
```

```
      {
                ;
            }
       }
    }
} /* End of function buffer_check() */
/*****************************************************************************
**
 * Function Name : buffer_clear
 * Description : buffer clear
 * Arguments : buf_type : buffer number
 * Return value : none
 ******************************************************************************
**/
static void buffer_clear (uint16_t buf_type)
{
    uint16_t i;
 for (i = 0; i < BUF_SIZE; i++)
    {
       g_buf[buf_type][i] = 0;
    }
} /* End of function buffer_clear() */
/*****************************************************************************
**
 * End of function usb_mcu_init
 ******************************************************************************
**/
```

5.2.6.17 USB PCDC (r_usb_pcdc)

Modules » Connectivity

This module provides a USB Peripheral Communications Device Class Driver (PCDC). It implements the USB PCDC Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_pcdc module combines with the r_usb_basic module to provide a USB Peripheral Communications Device Class (PCDC) driver. The PCDC driver conforms to Abstract Control Model of the USB Communications Device Class (CDC) specification and enables communication with a CDC host device.

Features

The r_usb_pcdc module has the following key features:

- Data transfer to and from a USB host
- Response to CDC class requests
- Supports CDC notifications

Configuration

Build Time Configurations for r_usb_pcdc

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

Configuration	Options	Default	Description
Bulk In Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE4	Select the USB pipe to use for bulk input transfers.
Bulk Out Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE5	Select the USB pipe to use for bulk output transfers.
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the USB pipe to use for interrupts.

Configurations for Connectivity > USB PCDC (r_usb_pcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PCDC (r_usb_pcdc).

Configuration	Options	Default	Description

Name	Name must be a valid C symbol	g_pcdc0	Module name.

Note

Refer to the USB (r_usb_basic) module for hardware configuration options.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Abstract Control Model Overview

The Abstract Control Model subclass of CDC is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems.

Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

Request	Code	Description
SetLineCoding	0x20	Sets communication line settings (bitrate, data length, parity, and stop bit length)
GetLineCoding	0x21	Acquires the communication line setting state
SetControlLineState	0x22	Set communication line control signals (RTS, DTR)

Note

For details concerning the Abstract Control Model requests, refer to Table 11 "Requests - Abstract Control Model" in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.

Data Format of Class Requests

The data format of supported class requests is described below:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING (0x20)	0x0000	0x0000	0x0007	usb_pcdc_linecoding_t
0xA1	GET_LINE_CODING (0x21)	0x0000	0x0000	0x0007	usb_pcdc_linecoding_t

0x21	SET_CONTROL_LINE_STATE (0x22)	usb_pcdc_ctrllinestate_t	0x0000	0x0000	None

Class Notifications (Peripheral to Host)

The following class notifications are supported:

Notification	Code	Description
SERIAL_STATE	0x20	Notification of serial line state

The data types returned are as follows:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0002	usb_serial_state_bitmap_t

Note

The host is notified with SERIAL_STATE whenever a change in the UART port state is detected. This driver will automatically detect overrun, parity and framing errors. A state notification is performed when a transition from normal to error state is detected.

Virtual COM-port Usage

When connected to a PC the CDC device can be used as a virtual COM port. After enumeration, the CDC class requests GetLineCoding and SetControlLineState are executed by the target, and the CDC device is registered in Windows Device Manager as a virtual COM device.

Registering the CDC device as a virtual COM-port in Windows Device Manager enables data communication with the CDC device via a terminal app such as PuTTY. When changing settings of the serial port in the terminal application, the UART setting is propagated to the firmware via the class request SetLineCoding.

Data input (or file transmission) from the terminal app window is transmitted to the board using endpoint 2 (EP2); data from the board side is transmitted to the PC using EP1.

When the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

Multi Port

This driver supports simultaneous operation with Host Mass Storage Class(HMSC). If the user are using MCU that supports 2 USB modules, such as RX74M, the user can run PCDC on one USB module and HMSC on the other. This driver does not support simultaneous operation using device classes other than HMSC.

Limitations

- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.

- This driver does not support Low-speed.
- The current USB driver does not support FreeRTOS.

Examples

USB PCDC Loopback Example

The main functions of the PCDC loopback example are as follows:

1. Receives virtual UART configuration data from the host terminal
2. Loops all other received data back to the host terminal

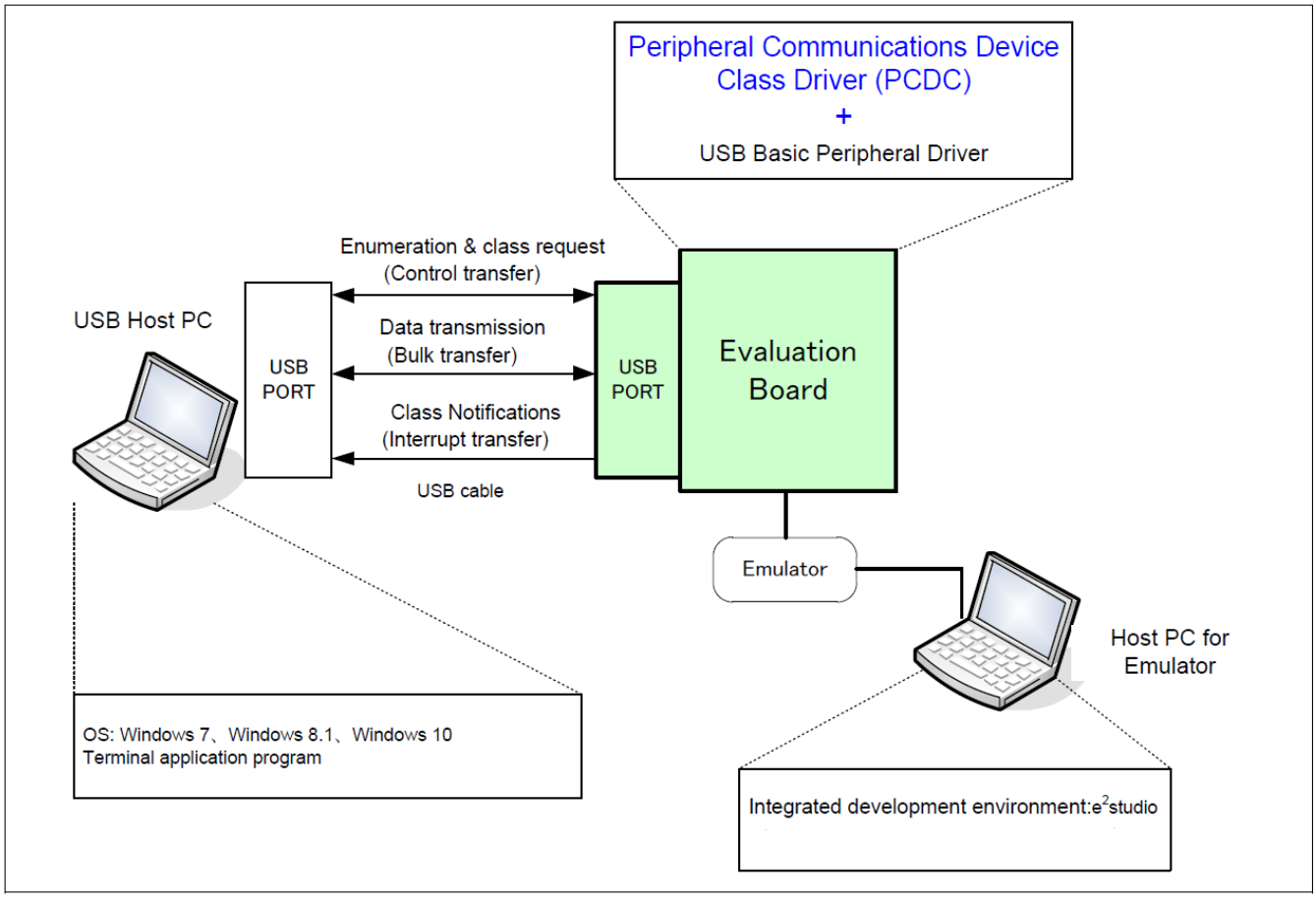


Figure 123: Example Operating Environment

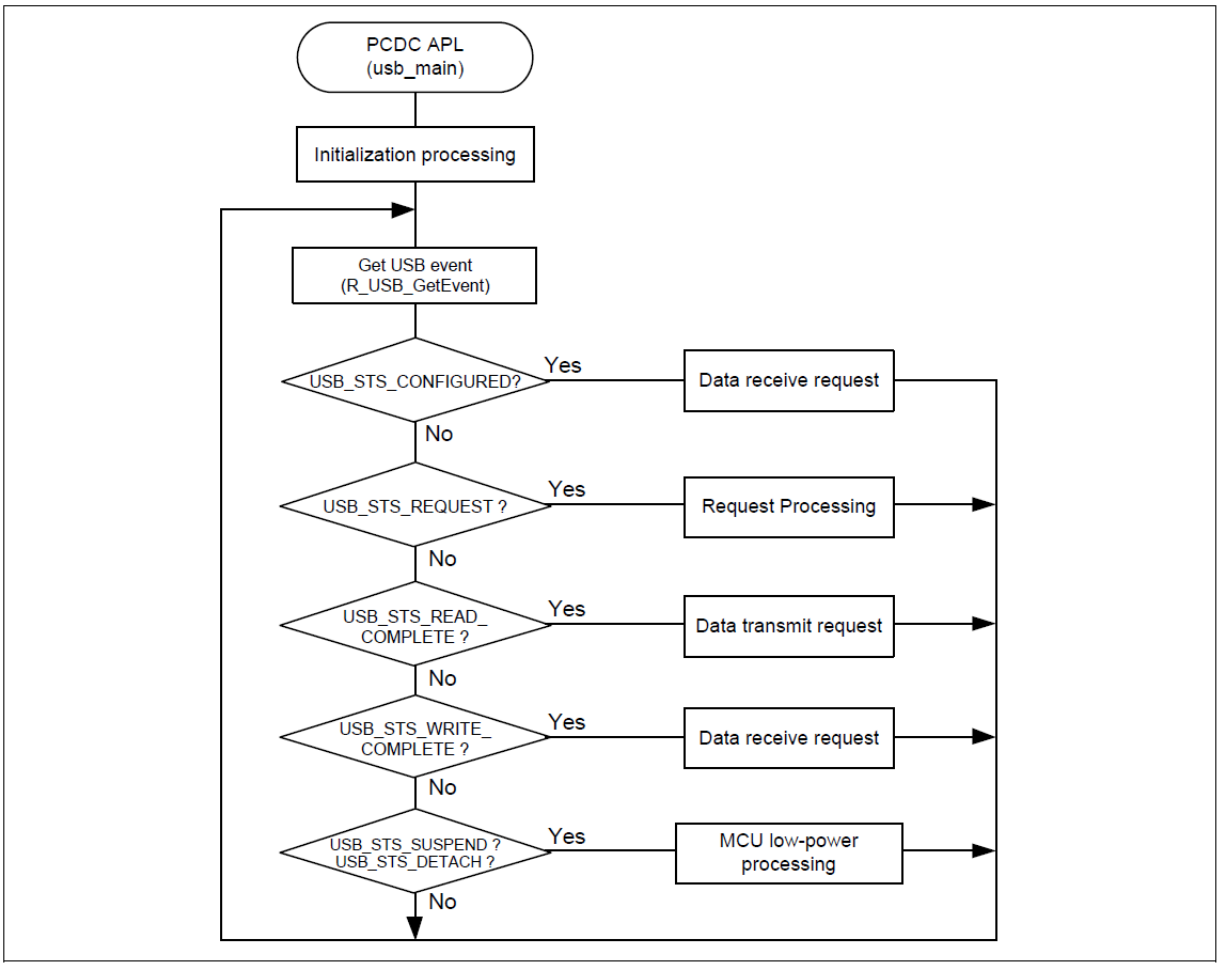


Figure 124: Main Loop processing (Echo mode)

```
#define USB_APL_YES (1U)
#define USB_APL_NO (0U)
#define APL_NUM_USB_EVENT (10U)
// #define APL_USE_BAREMETAL_CALLBACK USB_APL_NO
#define APL_USE_BAREMETAL_CALLBACK USB_APL_YES
/*****************************************************************************
 * Private global variables and functions
 *****************************************************************************/
extern const usb_cfg_t g_basic0_cfg;
static uint8_t g_buf[DATA_LEN];
static usb_pcdc_linecoding_t g_line_coding;
extern uint8_t g_apl_device[];
extern uint8_t g_apl_configuration[];
```

```
extern uint8_t g_apl_hs_configuration[];
extern uint8_t g_apl_qualifier_descriptor[];
extern uint8_t * g_apl_string_table[];
usb_instance_ctrl_t g_basic0_ctrl;
#if (BSP_CFG_RTOS == 2)
QueueHandle_t g_apl_mbx_hdl;
#endif  /* (BSP_CFG_RTOS == 2) */
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
usb_callback_args_t g_apl_usb_event;
usb_callback_args_t g_apl_usb_event_buf[APL_NUM_USB_EVENT];
uint8_t             g_apl_usb_event_wp = 0;
uint8_t             g_apl_usb_event_rp = 0;
#endif  /* (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
/******************************************************************************
 * Exported global functions (to be accessed by other files)
 ******************************************************************************/
void usb_pcdc_example(void);
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
void usb_apl_callback(usb_callback_args_t * p_event);
#endif
#if (BSP_CFG_RTOS == 2)
void usb_apl_callback(usb_event_info_t * p_api_event, usb_hdl_t cur_task, usb_onoff_t
usb_state);
#endif
/******************************************************************************
 * Renesas Peripheral Communications Devices Class Sample Code functions
 ******************************************************************************/
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
```

```
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_callback_args_t * p_event : Pointer to usb_callback_args_t
structure
 * Return value : none
 ******************************************************************************/
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
void usb_apl_callback (usb_callback_args_t * p_event)
{
    g_apl_usb_event_buf[g_apl_usb_event_wp] = *p_event;
    g_apl_usb_event_wp++;
    g_apl_usb_event_wp %= APL_NUM_USB_EVENT;
}
#endif  /* (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
/******************************************************************************
 * Function Name : usb_pcdc_example
 * Description : Peripheral CDC application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_pcdc_example (void)
{
    usb_event_info_t event_info;
```

```
 usb_status_t event = USB_STATUS_POWERED;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
    event_info.data_size = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
    g_usb_on_usb.callbackMemorySet(&g_basic0_ctrl, &g_apl_usb_event);
#endif  /* (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 #if (APL_USE_BAREMETAL_CALLBACK == USB_YES)
       g_usb_on_usb.driverActivate(&g_basic0_ctrl);
 if (g_apl_usb_event_wp != g_apl_usb_event_rp)
      {
            event_info = g_apl_usb_event_buf[g_apl_usb_event_rp];
            g_apl_usb_event_rp++;
            g_apl_usb_event_rp %= APL_NUM_USB_EVENT;
 event = event_info.event;
       }
 #else  /* (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
 #endif
#endif  /* (BSP_CFG_RTOS == 2) */
 /* Handle the received event (if any) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
```

```
 case USB_STATUS_WRITE_COMPLETE:
 /* Initialization complete; get data from host */
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_READ_COMPLETE:
 /* Loop back received data to host */
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Configure virtual UART settings */
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
         }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Send virtual UART settings back to host */
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
         }
 else
      {
 /* ACK all other status requests */
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
         }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
```

```
        }
    }
} /* End of function usb_pcdc_example() */
```

Descriptor

A template for PCDC descriptors can be found in rx/fsp/src/r_usb_pcdc/r_usb_pcdc_descriptor.c.template. Also, please be sure to use your vendor ID.

5.2.6.18 USB PHID (r_usb_phid)

Modules » Connectivity

This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_phid module combines with the r_usb_basic module to provide a USB Peripheral Human Interface Device Class (PHID) driver. The PHID driver conforms to the USB Human Interface Device class specifications and implements communication with a HID host.

Features

The r_usb_phid module has the following functions:

- Data transfer to and from a USB host
- Response to HID class requests
- Response to function references from the HID host

Note

This driver is not guaranteed to provide USB HID operation in all scenarios. The developer must verify correct operation when connected to the targeted USB hosts.

Configuration

Build Time Configurations for r_usb_phid

The following build time configurations are defined in fsp_cfg/r_usb_phid_cfg.h:

Configuration	Options	Default	Description
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the pipe number for input interrupt events.
Interrupt Out Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE7	Select the pipe number for output interrupt events.

Configurations for Connectivity > USB PHID (r_usb_phid)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PHID (r_usb_phid).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_phid0	Module name.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

Request	Code	Description
Get_Report	0x01	Receives a report from the HID host
Set_Report	0x09	Sends a report to the HID host
Get_Idle	0x02	Receives a duration (time) from the HID host
Set_Idle	0x0A	Sends a duration (time) to the HID host
Get_Protocol	0x03	Reads a protocol from the HID host
Set_Protocol	0x0B	Sends a protocol to the HID host
Get_Descriptor	0x06	Transmits a report or HID

descriptor

The data format of supported class requests is described below:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_REPORT (0x01)	ReportType & ReportID	Interface	ReportLength	Report
0x21	SET_REPORT (0x09)	ReportType & ReportID	Interface	ReportLength	Report
0xA1	GET_IDLE (0x02)	0 & ReportID	Interface	1	Idle rate
0x21	SET_IDLE (0x0A)	Duration & ReportID	Interface	0	Idle rate
0xA1	GET_PROTOCOL (0x03)	0	Interface	0	0 (Boot) or 1 (Report)
0x21	SET_PROTOCOL (0x0B)	0 (Boot) or 1 (Report)	Interface	0	Not applicable

Limitations

- This driver does not support USB Hi-speed mode.
- This driver does not support USB Low-speed mode.
- This driver does not support DMA transfers.
- This driver does not support simultaneous operation with USB Host device class.

Examples

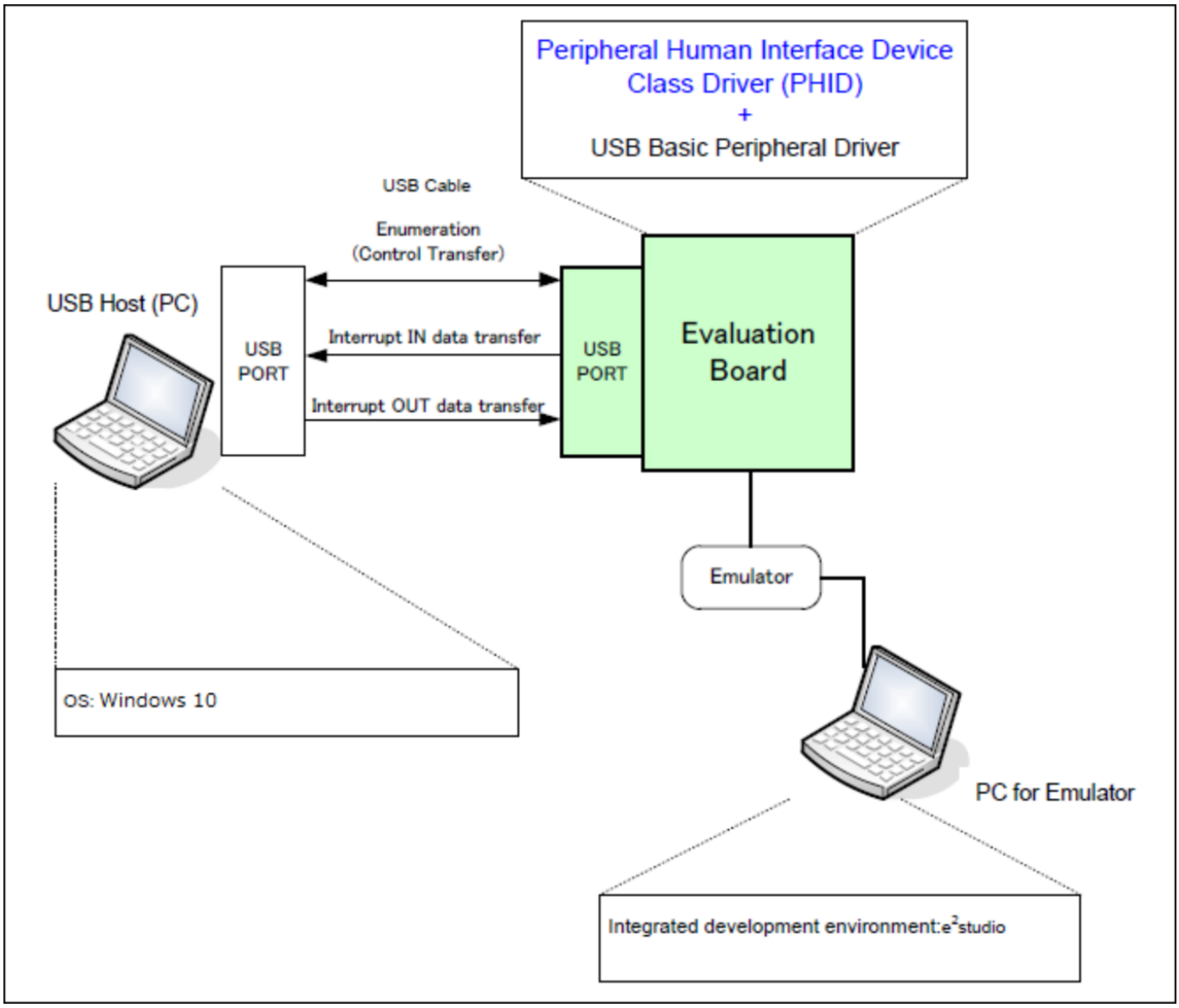


Figure 125: Example Operating Environment

USB PHID Example

This is a minimal example for implementing PHID in a non-RTOS application.

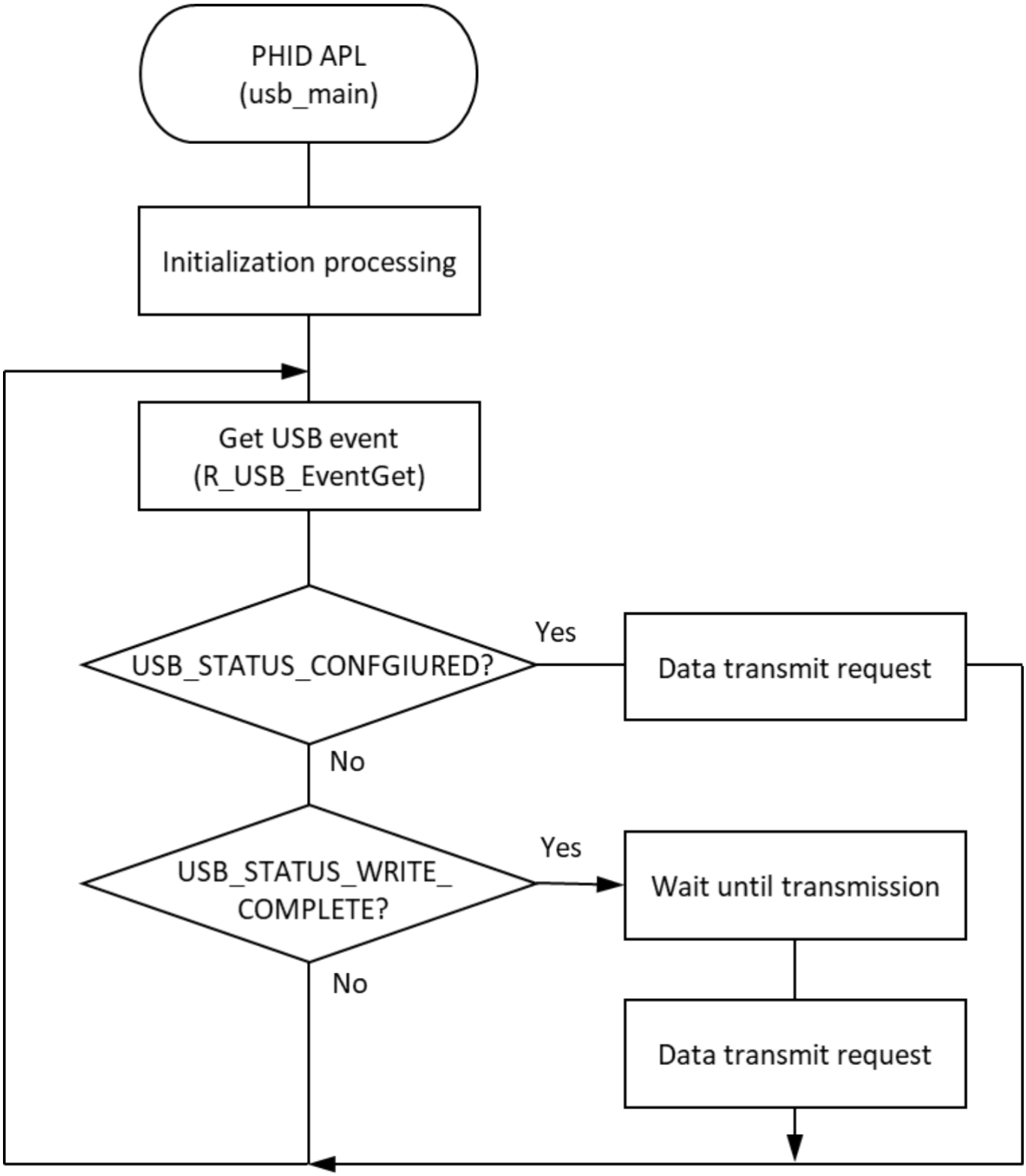


Figure 126: Main Loop processing for non-RTOS example

This is a minimal example for implementing PHID in an RTOS application.

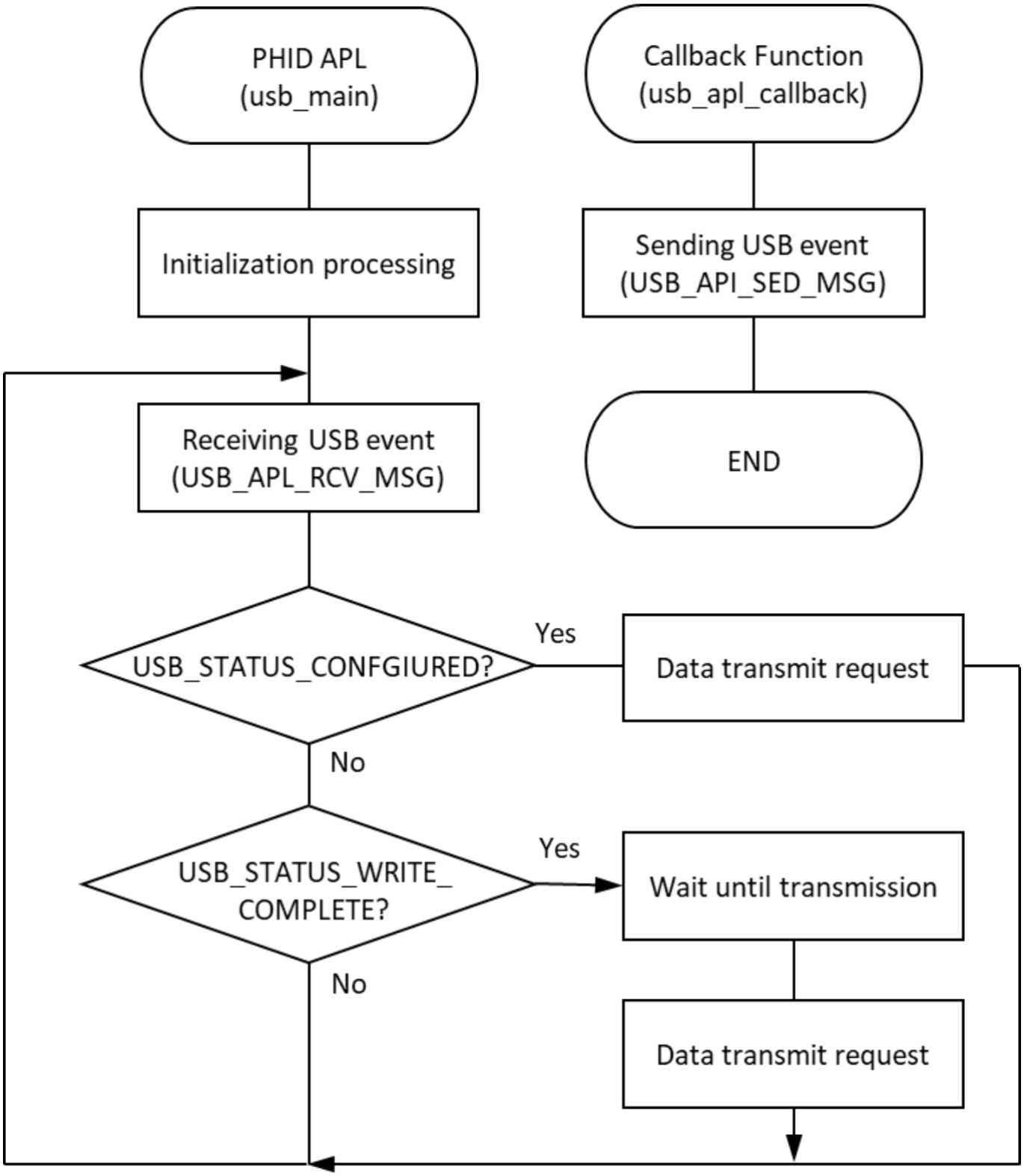


Figure 127: Main Loop processing for RTOS example

```
#define USB_RECEIVE_REPORT_DESCRIPTOR (76)
#define USB_RECEIVE_HID_DESCRIPTOR (9)
```

```
#define USB_WAIT_1000MS (1000)
#define SW_ACTIVE 0
#define SW R_PFS->PORT[0].PIN[8].PmnPFS_b.PIDR
#define SW_PDR R_PFS->PORT[0].PIN[8].PmnPFS_b.PDR
#define SW_PMR R_PFS->PORT[0].PIN[8].PmnPFS_b.PMR
static uint8_t send_data[16] BSP_ALIGN_VARIABLE(4);
static uint8_t g_buf[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* HID data */
static const uint8_t g_zero_data[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* zero data */
static uint16_t g_numlock = 0;
static uint8_t g_idle = 0;
uint8_t             g_remote_wakeup_enable = USB_OFF;
uint8_t             g_status               = NO_WRITING;
/******************************************************************************
 * Function Name : usb_cpu_getkeyno
 * Description : input key port
 * Arguments : none
 * Return value : uint16_t : key_no
 ******************************************************************************/
uint8_t usb_cpu_getkeyno (void)
{
    uint8_t key_buf = 0;
 if (SW_ACTIVE == SW)
    {
 if (sw_on_count[0] < SW_ON_THRESHOLD)
      {
            sw_on_count[0]++;
       }
    }
 else
    {
 if (sw_on_count[0] >= SW_ON_THRESHOLD)
      {
            key_buf |= SW_PUSH;
       }
```

```
        sw_on_count[0] = 0;
    }
 return key_buf;
} /* End of function usb_cpu_getkeyno() */
/******************************************************************************
 * Function Name : set_key_data
 * Description : Set key data to buffer
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void set_key_data (uint8_t * p_buf)
{
 static uint8_t key_data;
    key_data = KBD_CODE_A;
    *(p_buf + 2) = key_data;
} /* End of function set_key_data() */
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
```

```
 * Function Name : usb_phid_example
 * Description : Peripheral HID application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_phid_example (void)
{
    usb_event_info_t event_info;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
 usb_status_t event;
    uint8_t    * p_idle_value;
    uint8_t      sw_data;
    usb_info_t   info;
 fsp_err_t    ret_code = FSP_SUCCESS;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (DATA_WRITING == g_status)
```

```
      {
                    g_status = ZERO_WRITING;
                    g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
         }
 else
      {
                    g_status = DATA_WRITING;
                    usb_cpu_delay_xms(USB_WAIT_1000MS);
                    g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 break;
 case USB_STATUS_REQUEST
:                                                        /* Receive Class Request
*/
 if (USB_SET_REPORT == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_numlock, 2); /* Get the NumLock data (NumLock data is not used) */
         }
 else if (USB_GET_DESCRIPTOR == (event_info.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == event_info.setup.request_value)
      {
                         g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                         (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
         }
 else if (USB_GET_HID_DESCRIPTOR == event_info.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
```

```
                            send_data[i] = g_apl_configuration[18 + i];
        }
 /* Configuration Descriptor address set. */
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
        }
 else
     {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
        }
        }
 else if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* Get SetIdle value */
                  p_idle_value = (uint8_t *) &event_info.setup.request_value;
                  g_idle = p_idle_value[1];
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
        }
 else if (USB_GET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
        }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                  g_status = DATA_WRITING;
                  g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
        }
 else if (USB_GET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
```

```
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
 else
      {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
 break;
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
 if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                    p_idle_value = (uint8_t *) &event_info.setup.request_value;
                    g_idle = p_idle_value[1];
         }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* None */
         }
 else
      {
                    g_status = DATA_WRITING;
                    g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 break;
 case USB_STATUS_SUSPEND:
 break;
 case USB_STATUS_DETACH:
                g_remote_wakeup_enable = USB_OFF;
 break;
 default:
 break;
       }
```

```
      ret_code = g_usb_on_usb.infoGet(&g_basic0_ctrl, &info, 0);
 if (FSP_SUCCESS == ret_code)
     {
           sw_data = usb_cpu_getkeyno();
 if (USB_STATUS_SUSPEND == info.device_status)
     {
 if (0 != (sw_data & SW_PUSH))
     {
                  g_usb_on_usb.remoteWakeup(&g_basic0_ctrl);
        }
        }
      }
   }
} /* End of function usb_phid_example() */
```

Descriptors

A template for PHID descriptors can be found in ra/fsp/src/r_usb_phid. Be sure to replace the vendor ID with your own.
Keyboard templates should be referred to r_usb_phid_descriptor_keyboard.c.template.
Mouse templates should be referred to r_usb_phid_descriptor_mouse.c.template.

5.2.6.19 USB PMSC (r_usb_pmsc)

Modules » Connectivity

This module provides a USB Peripheral Mass Storage Class (PMSC) driver. It implements the USB PMSC Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_pmsc module combines with the r_usb_basic module to provide USB Peripheral It operates

as a Mass Storage class driver (hereinafter referred to as PMSC).
The USB peripheral mass storage class driver (PMSC) comprises a USB mass storage class bulk-only transport (BOT) protocol.
When combined with a USB peripheral control driver and media driver, it enables communication with a USB host as a BOT-compatible storage device.

Features

The r_usb_pmsc module has the following key features:

- Storage command control using the BOT protocol
- Supports only SFF-8070i (ATAPI) Interface Sub-Class
- Response to mass storage device class requests from a USB host

Configuration

Build Time Configurations for r_usb_pmsc

The following build time configurations are defined in fsp_cfg/r_usb_pmsc_cfg.h:

Configuration	Options	Default	Description
Bulk Input Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE4	Select the USB pipe to use for bulk input transfers.
Bulk Output Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE5	Select the USB pipe to use for bulk output transfers.
Vendor Information	Vendor Information must be 8 bytes long; pad with spaces if shorter.	Vendor	Specify the vendor information field (part of the Inquiry command response).
Product Information	Product Information must be 16 bytes long; pad with spaces if shorter.	Mass Storage	Specify the product information field (part of the Inquiry command response).
Product Revision Level	Product Revision Level must be 4 bytes long; pad with spaces if shorter.	1.00	Specify the product revision level field (part of the Inquiry command response).
Sector size	• 512 • 4096	512	Specifies the sector size.
Number of Transfer Sectors	Please enter a number between 1 and 255.	8	Specify the maximum sector size to request with one data transfer.

Configurations for Connectivity > USB PMSC (r_usb_pmsc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PMSC (r_usb_pmsc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_pmsc0	Module name.

Refer to the USB (r_usb_basic) module for hardware configuration options.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Class Requests

The class requests supported by this driver are shown below.

Request	Code	Description
Bulk-Only Mass Storage Reset	0xFF	Resets the connection interface to the mass storage device.
Get Max Logical Unit Number	0xFE	Reports the logical numbers supported by the device.

Storage Commands

This driver supports the following storage commands.

Command	Code	Description
TEST_UNIT_READY	0x00	Checks the state of the peripheral device.
REQUEST_SENSE	0x03	Gets the error information of the previous storage command execution result.
INQUIRY	0x12	Gets the parameter information of the logical unit.
READ_FORMAT_CAPACITY	0x23	Gets the formattable capacity.
READ_CAPACITY	0x25	Gets the capacity information of the logical unit.
READ10	0x28	Reads data.
WRITE10	0x1A	Writes data.

MODE_SENSE10	0x5A	Gets the parameters of the logical unit.

Note

A STALL or FAIL error is sent to the host upon receipt of any command not listed in the above table.

BOT Protocol Overview

BOT (USB MSC Bulk-Only Transport) is a transfer protocol that encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out). The ATAPI storage commands and the response status are embedded in a Command Block Wrapper (CBW) and a Command Status Wrapper (CSW). The below image shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

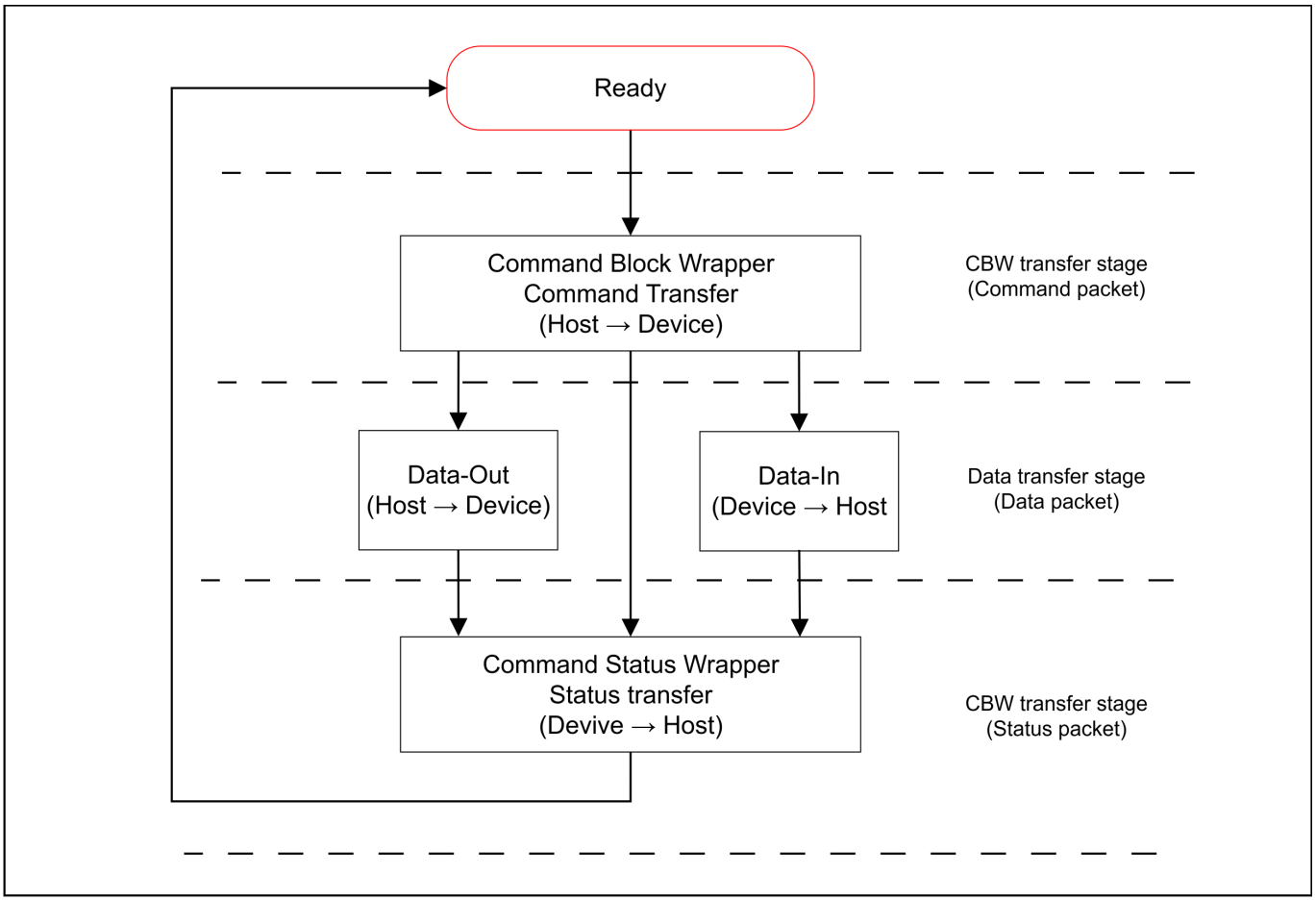


Figure 128: BOT protocol Overview

Block Media Interface

PMSC implements a block media interface to enable access to higher-level modules. If the block media interface supports multiple media, users can select any media to access.

Note

When the user develops the storage media driver, be sure to define the instance named "g_rm_block_media0".

Limitations

1. The driver always returns 0 in response to the GetMaxLun command.
2. The driver supports a sector size of 512 bytes only.
3. The only media currently supported by the block media interface is an SD card. The card must be inserted before initializing the driver.
4. When using DMA for Hi-Speed transfers continuous transfer mode must not be used in the USB Basic driver.
5. The storage area must be formatted before use.
6. When using the SD/MMC Block Media Implementation (rm_block_media_sdmmc), "Card Detection" must be set to "Not Used" in the SD/MMC Host Interface (r_sdhi) settings.
7. The driver does not support Low-speed.
8. This driver does not support simultaneous operation with USB Host device class.

Examples

USB PMSC Example

In this example, when the evaluation board is connected to the host PC it is recognized as a removable disk and reading/writing files is possible. The FAT type is either FAT12, FAT16, or FAT32 depending on the size of the media used.

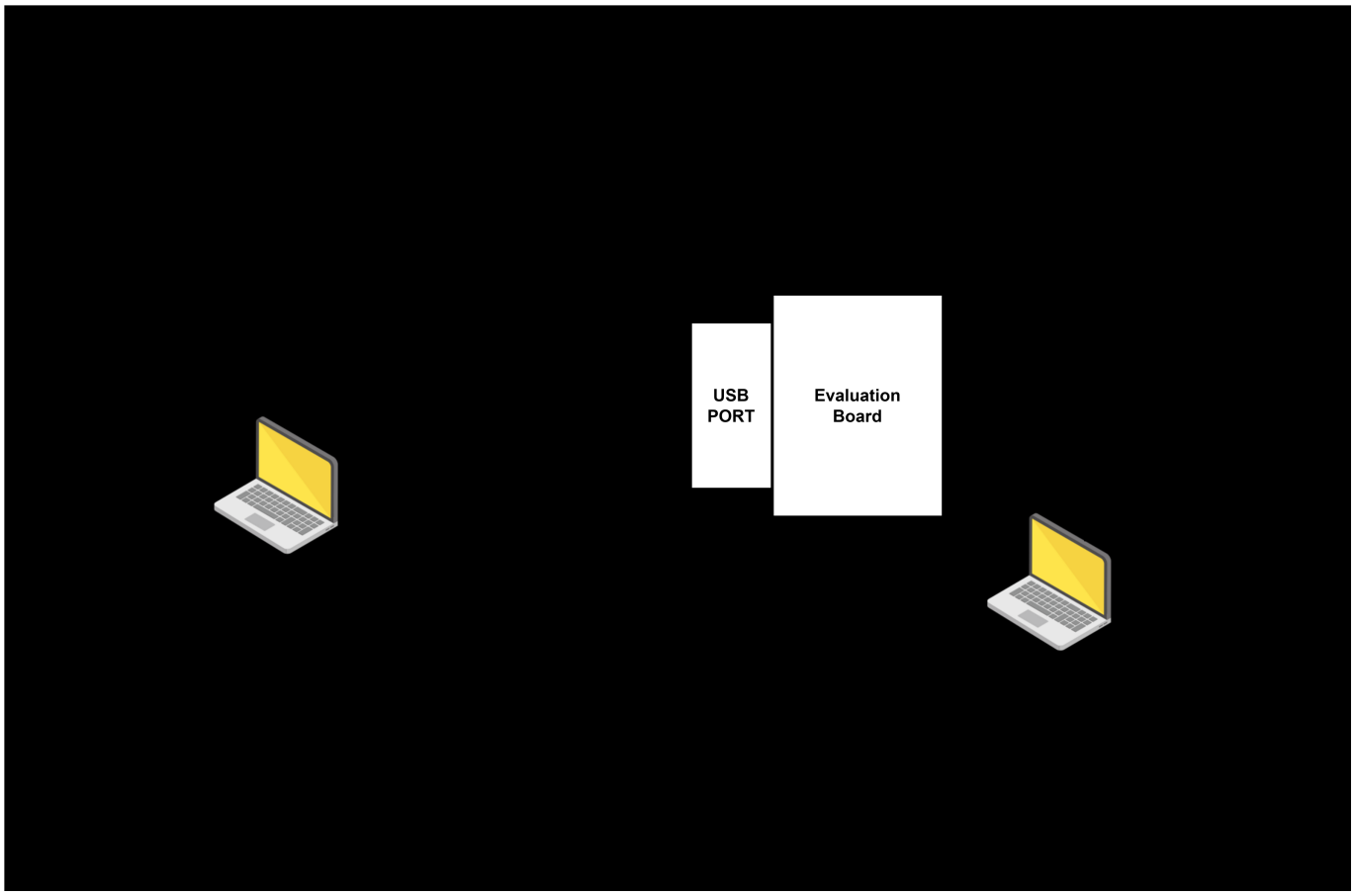


Figure 129: Example Operating Environment

```
#if (BSP_CFG_RTOS == 2)
/*******************************************************************************
```

```
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback */
#endif /* (BSP_CFG_RTOS == 2) */
void usb_pmsc_example (void)
{
    usb_event_info_t usb_event;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#else
 usb_status_t event;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&usb_event, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
```

```
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
 break;
            }
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
      {
#if USB_SUPPORT_LPW == USB_APL_ENABLE
// @@ low_power_mcu();
#endif  /* USB_SUPPORT_LPW == USB_APL_ENABLE */
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main() */
```

Descriptor

A template for PMSC descriptors can be found in rx/fsp/src/r_usb_pmsc/r_usb_pmsc_descriptor.c.template. Also, please be sure to use your vendor ID.

5.2.6.20 USB Peripheral Printer (r_usb_pprn)

Modules » Connectivity

This module is USB Peripheral Printer Device Class Driver (PPRN). It implements the USB PPRN Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_pprn module combines with the r_usb_basic module to provide a USB Peripheral Printer Device Class (PPRN) driver. The PPRN driver conforms to the USB Printer Device class specifications and implements communication with a printer host.

Features

The r_usb_pprn module has the following functions:

- Data transfer to and from a USB host
- Response to the printer class requests
- Response to function references from the printer host

Note

This driver is not guaranteed to provide USB printer operation in all scenarios. The developer must verify correct operation when connected to the targeted USB hosts.

Configuration

Build Time Configurations for r_usb_pprn

The following build time configurations are defined in fsp_cfg/r_usb_pprn_cfg.h:

Configuration	Options	Default	Description
Bulk In Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE4	Select the USB pipe to use for bulk in transfer.
Bulk Out Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE5	Select the USB pipe to use for bulk out transfer.

Configurations for Connectivity > USB PPRN (r_usb_pprn)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PPRN (r_usb_pprn).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_pprn0	Module name.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

Request	Code	Description
GET_DEVICE_ID	0x00	Get IEEE 1284 Device ID String
GET_PORT_STATUS	0x01	Get Printer's Status
SOFT_RESET	0x02	Flushes all buffers and resets the Bulk OUT and Bulk IN pipes

The data format of supported class requests is described below:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_DEVICE_ID (0x00)	Config Index	Interface Alternate Setting	Maximum Length	1284 Device ID String
0xA1	GET_PORT_STATUS (0x01)	0	Interface	1	Printer Status
0x21	SOFT_RESET (0x02)	0	Interface	0	None

Limitations

- This driver does not support USB Low-speed mode.
- This driver does not support simultaneous operation with USB Host device class.

USB PPRN Example

This is a minimal example for implementing PPRN in bare metal and FreeRTOS application.

```
/******************************************************************************
 * Macro definitions
 ******************************************************************************/
/* Define device framework. */
#define DEMO_PROTOCOL (1U)  /* 1-Uni-dir, 2-Bi-dir */
#define VALUE_5BH (0x5BU)
/******************************************************************************
 * Private global variables and functions
```

```
 ******************************************************************************/
static uint8_t g_port_status = PORT_STATUS_BENIGN;
/* Device printer device ID. */
static uint8_t g_printer_device_id[] =
{
    0x00,                               /* data length */
    VALUE_5BH,
 'M',                               /* manufacturer (case sensitive) */
 'F',
 'G',
 ':',
 'G',
 'e',
 'n',
 'e',
 'r',
 'i',
 'c',
 ';',
 'M',                               /* model (case sensitive) */
 'D',
 'L',
 ':',
 'G',
 'e',
 'n',
 'e',
 'r',
 'i',
 'c',
 '_',
 '/',
 '_',
 'T',
```

```
'e',
'x',
't',
'_',
'O',
'n',
'l',
'y',
';',
'C',                            /* PDL command set */
'M',
'D',
':',
'1',
'2',
'8',
'4',
'.',
'4',
';',
'C',                            /* class */
'L',
'S',
':',
'P',
'R',
'I',
'N',
'T',
'E',
'R',
';',
'D',                            /* description */
'E',
```

```
 'S',
 ':',
 'G',
 'e',
 'n',
 'e',
 'r',
 'i',
 'c',
 ' ',
 't',
 'e',
 'x',
 't',
 ' ',
 'o',
 'n',
 'l',
 'y',
 ' ',
 'p',
 'r',
 'i',
 'n',
 't',
 'e',
 'r',
 ';'
};
/*****************************************************************************
 * Exported global functions (to be accessed by other files)
 *****************************************************************************/
void usb_pprn_example(void);
usb_instance_ctrl_t g_basic0_ctrl;
```

```
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : usb_pprn_example
 * Description : Peripheral Printer Class(PRN) application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_pprn_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    usb_info_t       info;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
```

```
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif
 /* Handle the received event (if any) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 /* Initialization complete; get data from host */
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PPRN);
 break;
 case USB_STATUS_WRITE_COMPLETE:
#if DEMO_PROTOCOL > 1
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PPRN);
#endif
 break;
 case USB_STATUS_READ_COMPLETE:
#if DEMO_PROTOCOL > 1
 /* Loop back to Host */
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PPRN);
#else
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PPRN);
#endif
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PPRN_GET_DEVICE_ID == (event_info.setup.request_type & USB_BREQUEST))
      {
 R_USB_InfoGet(&g_basic0_ctrl, &info, event_info.device_address);
 R_USB_PeriControlDataSet(&g_basic0_ctrl, g_printer_device_id,
```

```
PRINTER_DEVICE_ID_LENGTH);
                }
 else if (USB_PRPN_GET_PORT_STATUS == (event_info.setup.request_type & USB_BREQUEST))
      {
 R_USB_InfoGet(&g_basic0_ctrl, &info, event_info.device_address);
 R_USB_PeriControlDataSet(&g_basic0_ctrl, &g_port_status, 1);
                }
 else if (USB_PPRN_SOFT_RESET == (event_info.setup.request_type & USB_BREQUEST))
      {
 R_USB_InfoGet(&g_basic0_ctrl, &info, event_info.device_address);
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
 /* [To do] Transport Abort */
                   g_usb_on_usb.stop(&g_basic0_ctrl, USB_TRANSFER_READ,
USB_CLASS_PPRN);
                }
 else
      {
                }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
       }
    }
}
```

Descriptors

A template for PPRN descriptors can be found in rx/fsp/src/r_usb_pprn folder. Be sure to replace the vendor ID with your own.

5.2.6.21 USB Peripheral Vendor class (r_usb_pvnd)

Modules » Connectivity

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

USB Peripheral Vendor class works by combining r_usb_basic module.

How to Configuration

The following shows FSP configuration procedure for USB Peripheral Vendor class.

- Select [New Stack]->[Connectivity]->[USB PVND (r_usb_pvnd)].

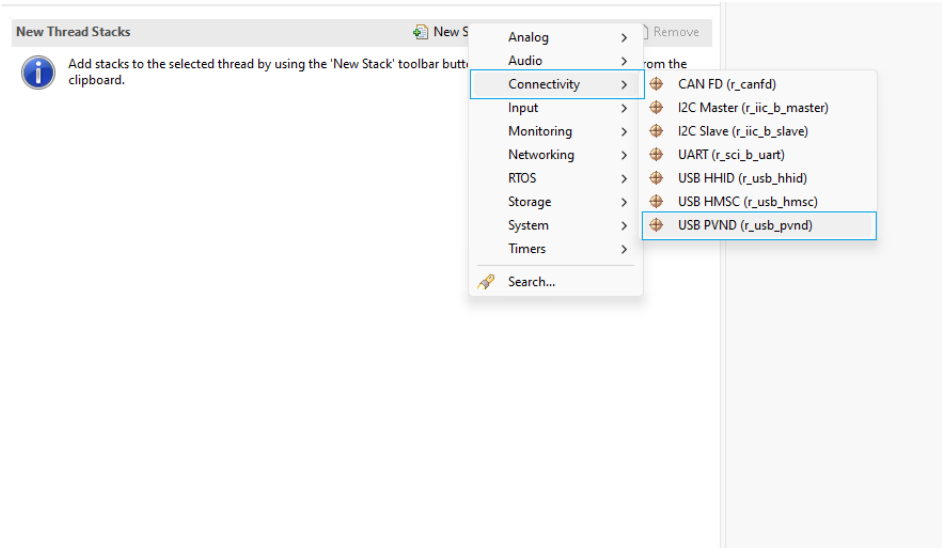


Figure 130: Select USB Peripheral Vendor Class

- The following is displayed when selecting [USB PVND (r_usb_pvnd)]. The user does not specify USB pipe number in Vendor class.

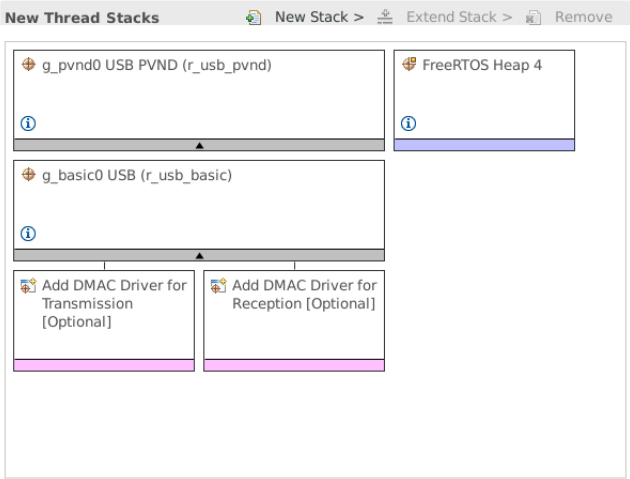


Figure 131: USB Peripheral Vendor Class Stack

API

Use the following APIs in Peripheral Vendor class application program.

- For Data Transfer
  Use the following APIs for data transfer for Bulk transfer or Interrupt transfer.

  1. R_USB_PipeRead()
  2. R_USB_PipeWrite()
  3. R_USB_PipeStop()

- For Control Transfer
  Use the following API for the class request processing.

  1. R_USB_PeriControlDataGet()
  2. R_USB_PeriControlDataSet()
  3. R_USB_PeriControlStatusSet()

- For USB Pipe Information
  The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class. Use the following APIs to get the allocated USB pipe information.

  1. R_USB_UsedPipesGet()
  2. R_USB_PipeInfoGet()

USB PIPE Allocation

The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class.
The USB PIPE related to the Endpoint Descriptor are allocated in order from USB PIPE1 according to the description order of the Endpoint Descriptor.
FSP driver pipe configuration behaviors for the PVND Class, when it is used with the Composite PCDC+PVND Class is mentioned below.

- In Composite PCDC+PVND class, PCDC class interface is placed next to the PVND class interface in descriptor file. PVND Class Pipe configurations are as follows.

- FSP driver(PVND) allocates Pipes in order from USB PIPE1 according to the order of the endpoint in descriptor(Bulk transfer).
- FSP driver(PVND) allocates Pipes in order(decremental order) from USB PIPE9 according to the order of the endpoint in descriptor(Interrupt transfer).
- PCDC and PVND must always use different USB PIPEs.

For example:

- In the Composite PCDC+PVND class, if there are 3 number of bulk transfers endpoints and one Interrupt transfer endpoint in descriptor file for PVND class and for this configuration FSP driver configures USB PIPE1, USB PIPE2 and USB PIPE3 for bulk data transfer and USB PIPE9 for Interrupt data transfer.
- Since hardware support total 5 number of Bulk transfer pipe configuration. So, in this scenario, for the PCDC class we have configurable USB PCDC Stack property to select available Pipes(USB PIPE 4 and USB PIPE 5 for Bulk and USB PIPE 6 for Interrupt transfer) from USB PCDC Stack configurator.

Descriptor

Template for Vendor class descriptor can be found in rx/fsp/src/r_usb_pvnd folder. Also, please be sure to use your vendor ID.

Examples

This application program processes the follwoing after the enumeration completes with USB device.

1. Getting USB Pipe Infomattion
2. Vendor Class Request Processing
3. Loopback processing of bulk transfer and interrupt transfer.

```
/*****************************************************************************
**
 * Macro definitions
 *****************************************************************************
**/
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
/*****************************************************************************
**
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
```

```
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************
**/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0))
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
**
 * Function Name : usb_pvnd_example
 * Description : Peripheral Vendor Class application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
void usb_pvnd_example (void)
{
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
    usb_event_info_t * p_mess;
#endif
 usb_status_t    event;
    usb_event_info_t event_info;
    uint8_t          bulk_out_pipe = 0; /* Bulk Out Pipe       */
    uint8_t          bulk_in_pipe = 0; /* Bulk In Pipe       */
    uint8_t          int_out_pipe = 0; /* Interrupt Out Pipe   */
    uint8_t          int_in_pipe   = 0; /* Interrupt In Pipe   */
    uint16_t   buf_type       = 0;
```

```
    uint8_t    pipe          = 0;
    uint8_t    is_zlp[2]      = {0, 0};
    uint32_t   request_length = 0;
    uint16_t   used_pipe      = 0;
    usb_pipe_t pipe_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    memset(&event_info, 0, sizeof(usb_event_info_t));
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                buffer_init();
                is_zlp[0] = 0;
                is_zlp[1] = 0;
                g_usb_on_usb.usedPipesGet(&g_basic0_ctrl, &used_pipe,
USB_CLASS_PVND);
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
      {
 if (0 != (used_pipe & (1 << pipe)))
      {
                        g_usb_on_usb.pipeInfoGet(&g_basic0_ctrl, &pipe_info,
pipe);
 if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
      {
 /* Out Transfer */
```

```
if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
     {
                                buf_type      = BUF_BULK;
                                bulk_out_pipe = pipe;
                            }
else
     {
                                buf_type    = BUF_INT;
                                int_out_pipe = pipe;
                            }
                        }
else
     {
/* In Transfer */
if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
     {
                                buf_type    = BUF_BULK;
                                bulk_in_pipe = pipe;
                            }
else
     {
                                buf_type    = BUF_INT;
                                int_in_pipe = pipe;
                            }
                        }
                    }
                }
break;
            }
case USB_STATUS_READ_COMPLETE:
     {
if (FSP_ERR_USB_FAILED != event_info.status)
     {
if (bulk_out_pipe == event_info.pipe)
```

```
     {
                         buf_type = BUF_BULK;
                         pipe    = bulk_in_pipe;
                    }
 else if (int_out_pipe == event_info.pipe)
     {
                         buf_type = BUF_INT;
                         pipe    = int_in_pipe;
                    }
 else
     {
 while (1)
     {
                             ;
                         }
                    }
                    buffer_check(buf_type, event_info.data_size);
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[buf_type][0], event_info.data_size, pipe);
                }
 break;
            }
 case USB_STATUS_WRITE_COMPLETE:
     {
 if (FSP_SUCCESS == event_info.status)
     {
 if (bulk_in_pipe == event_info.pipe)
     {
                         buf_type = BUF_BULK;
 if (1 == is_zlp[buf_type])
     {
                             pipe = bulk_out_pipe;
                         }
                    }
```

```
 else if (int_in_pipe == event_info.pipe)
      {
                        buf_type = BUF_INT;
 if (1 == is_zlp[buf_type])
      {
                            pipe = int_out_pipe;
                        }
                   }
 else
      {
 /* Nothing */
                   }
 if (1 == is_zlp[buf_type])
      {
                        is_zlp[buf_type] = 0;
                        buffer_clear(buf_type);
                        g_usb_on_usb.pipeRead(&g_basic0_ctrl,
&g_buf[buf_type][0], BUF_SIZE, pipe);
                   }
 else
      {
                        is_zlp[buf_type] = 1;
                        g_usb_on_usb.pipeWrite(&g_basic0_ctrl, 0, 0,
event_info.pipe); /* Send ZLP */
                   }
               }
 break;
            }
 case USB_STATUS_REQUEST:
      {
 if (USB_SET_VENDOR_NO_DATA == (event_info.setup.request_type & USB_BREQUEST
))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
```

```
USB_SETUP_STATUS_ACK);
                }
 else if (USB_SET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                   request_length = event_info.setup.request_length;
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl,
&g_request_buf[0], request_length);
                }
 else if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
&g_request_buf[0], request_length);
                }
 else
      {
 /* Nothing */
                }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[BUF_BULK][0],
BUF_SIZE, bulk_out_pipe);
                   g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[BUF_INT][0],
BUF_SIZE, int_out_pipe);
                }
 break;
            }
 case USB_STATUS_DETACH:
      {
 break;
            }
```

```
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_pvnd_example() */
/*****************************************************************************
**
 * Function Name : buffer_init
 * Description : buffer initialization
 * Arguments : none
 * Return value : none
 *****************************************************************************
**/
static void buffer_init (void)
{
    uint16_t i;
    uint16_t j;
 for (j = 0; j < 2; j++)
    {
 for (i = 0; i < BUF_SIZE; i++)
      {
            g_buf[j][i] = (uint8_t) i;
       }
    }
} /* End of function buffer_init() */
/*****************************************************************************
**
 * Function Name : buffer_check
 * Description : buffer check
 * Arguments : buf_type : buffer number
 * Return value : none
 *****************************************************************************
```

```
**/
static void buffer_check (uint16_t buf_type, uint32_t size)
{
    uint16_t i;
 for (i = 0; i < (uint16_t) size; i++)
    {
 if ((uint8_t) (i & USB_VALUE_FF) != g_buf[buf_type][i])
      {
 while (1)
      {
                ;
            }
       }
    }
} /* End of function buffer_check() */
/*****************************************************************************
**
 * Function Name : buffer_clear
 * Description : buffer clear
 * Arguments : buf_type : buffer number
 * Return value : none
 *****************************************************************************
**/
static void buffer_clear (uint16_t buf_type)
{
    uint16_t i;
 for (i = 0; i < BUF_SIZE; i++)
    {
       g_buf[buf_type][i] = 0;
    }
} /* End of function buffer_clear() */
/*****************************************************************************
**
 * End of function usb_mcu_init
```

```
 ***********************************************************************

**/
```

5.2.6.22 USB_PCDC Communication Device (rm_comms_usb_pcdc)

Modules » Connectivity

Functions

fsp_err_t	RM_COMMS_USB_PCDC_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg) Opens and configures the USB PCDC Comms module. Implements rm_comms_api_t::open. More...
fsp_err_t	RM_COMMS_USB_PCDC_Close (rm_comms_ctrl_t *const p_api_ctrl) Disables specified USB PCDC Comms module. Implements rm_comms_api_t::close. More...
fsp_err_t	RM_COMMS_USB_PCDC_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) Performs a read from the USB PCDC device. Implements rm_comms_api_t::read. More...
fsp_err_t	RM_COMMS_USB_PCDC_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes) Performs a write to the USB PCDC device. Implements rm_comms_api_t::write. More...
fsp_err_t	RM_COMMS_USB_PCDC_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params) Performs a write to, then a read from the USB device. Implements rm_comms_api_t::writeRead. More...
fsp_err_t	RM_COMMS_USB_PCDC_CallbackSet (rm_comms_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context) Updates the USB PCDC Comms callback. Implements rm_comms_api_t::callbackSet. More...

void	rm_comms_usb_pcdc_notify_application (rm_comms_usb_pcdc_instance_ctrl_t const *p_ctrl, rm_comms_event_t event)
void	rm_comms_usb_pcdc_callback_handler (usb_instance_ctrl_t *p_args, usb_hdl_t usb_handle, usb_onoff_t usb_onoff_status) Common callback function called in the USB PCDC driver callback function.

Detailed Description

Middleware to implement a generic communications interface over USB_PCDC. This module implements the Communications Middleware Interface.

Overview

The RM_COMMS_USB_PCDC module implements COMMS API for USB_PCDC interface.

Features

The implementation of the USB_PCDC communications interfacehas the following key features:

- Non-blocking API for bare metal
- Non-blocking and blocking API for RTOS
- USB connection detection mode

Configuration

Build Time Configurations for rm_comms_usb_pcdc

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Bulk In Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE4	Select the USB pipe to use for bulk input transfers.
Bulk Out Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE5	Select the USB pipe to use for bulk output transfers.
Interrupt In Pipe	• USB PIPE6	USB PIPE6	Select the USB pipe to

	• USB PIPE7 • USB PIPE8 • USB PIPE9	use for interrupts.

Configurations for Connectivity > USB PCDC Communication Device (rm_comms_usb_pcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PCDC Communication Device (rm_comms_usb_pcdc).

Configuration	Options	Default	Description
RTOS > Recursive Write Mutex	• Do Not Use • Use	Use	Lock device for writing in using RTOS.
RTOS > Recursive Read Mutex	• Do Not Use • Use	Use	Lock device for reading in using RTOS.
RTOS > Mutex Timeout	Value must be a non-negative integer	0xFFFFFFFF	Timeout for recursive mutex operation in using RTOS.
RTOS > Write Semaphore	• Do Not Use • Use	Use	Block writing in using RTOS.
RTOS > Read Semaphore	• Do Not Use • Use	Use	Block reading in using RTOS.
RTOS > Semaphore Timeout	Value must be a non-negative integer	0xFFFFFFFF	Timeout for semaphore operation in using RTOS.
Name	Name must be a valid C symbol	g_comms_usb_pcdc0	Module name.
USB Connection Detection	• Disabled • Baudrate Configuration • Control Line State	Disabled	Enable it only when application wants to process the error of USB host tool not opened.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided.

Usage Notes

USB connection detection has two modes: Baudrate Configuration mode and Control Line State mode. The former depends on line coding request, and the latter relies on control line state request. When it is enabled, rm_comms_usb_pcdc_write API will return FSP_ERR_NOT_ENABLED when USB host tool is not opened. By default it is disabled.

Limitations

- RM_COMMS_API are not reentrant in non blocking mode
- When in blocking mode, RM_COMMS_USB_PCDC_Write() and RM_COMMS_USB_PCDC_Read() cannot be called in callback.

- RM_COMMS_USB_PCDC_WriteRead API is not implemented
- USB connection detection could be platform specific when it is enabled. For Windows USB host, Baudrate configuration mode should be used.

Examples

Basic Example

This is a basic example of minimal use of USB_PCDC communications implementation in an application.

```
void rm_comms_usb_pcdc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_COMMS_USB_PCDC_Open(&g_comms_usb_pcdc_ctrl, &g_comms_usb_pcdc_cfg);
 if (FSP_SUCCESS != err)
    {
 /* Handle any errors. */
    }
 while (true)
    {
 /* Send data. */
       g_err_flag = 0;
       g_tx_flag = 0;
       err       = RM_COMMS_USB_PCDC_Write(&g_comms_usb_pcdc_ctrl, g_tx_buf,
TX_BUF_LEN);
 if (FSP_SUCCESS != err)
      {
 /* Handle any errors. */
      }
 while ((0 == g_tx_flag) && (0 == g_err_flag))
      {
 /* Wait callback */
      }
 /* Receive data. */
       g_err_flag = 0;
       g_rx_flag = 0;
       err       = RM_COMMS_USB_PCDC_Read(&g_comms_usb_pcdc_ctrl, g_rx_buf,
```

```
RX_BUF_LEN);
 if (FSP_SUCCESS != err)
      {
 /* Handle any errors.*/
      }
 while ((0 == g_rx_flag) && (0 == g_err_flag))
      {
 /* Wait callback */
      }
    }
}
static void rm_comms_usb_pcdc_callback (rm_comms_callback_args_t * p_args)
{
 if (p_args->event == RM_COMMS_EVENT_TX_OPERATION_COMPLETE)
    {
       g_tx_flag = 1;
    }
 else if (p_args->event == RM_COMMS_EVENT_RX_OPERATION_COMPLETE)
    {
       g_rx_flag = 1;
    }
 else
    {
       g_err_flag = 1;
    }
}
```

Descriptor

A template for PCDC descriptors can be found in rx/fsp/src/r_usb_pcdc/r_usb_pcdc_descriptor.c.template. Also, please be sure to use your vendor ID.

Data Structures

struct	rm_comms_usb_pcdc_instance_ctrl_t

Data Structure Documentation

◆ rm_comms_usb_pcdc_instance_ctrl_t

struct rm_comms_usb_pcdc_instance_ctrl_t	
Communications middleware control structure.	
Data Fields	
uint32_t	open
	Open flag.
rm_comms_cfg_t const *	p_cfg
	Middleware configuration.
rm_comms_usb_pcdc_extended_cfg_t const *	p_extend
	Pointer to extended configuration structure.
usb_callback_args_t *	p_usb_args
	Pointer to usb callback args.
usb_pcdc_ctrllinestate_t	ctrl_line_state
	Line State Control.
usb_pcdc_linecoding_t	line_coding
	Line Coding to store COM port settings.
void(*	p_callback)(rm_comms_callback_args_t *p_args)
	Pointer to callback that is called when a usb_status_t occurs.
void *	p_context
	Pointer to context passed into callback function.

Function Documentation

◆ RM_COMMS_USB_PCDC_Open()

fsp_err_t RM_COMMS_USB_PCDC_Open (rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg*)

Opens and configures the USB PCDC Comms module. Implements rm_comms_api_t::open.

Return values

FSP_SUCCESS	USB PCDC Comms module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_COMMS_USB_PCDC_Close()

fsp_err_t RM_COMMS_USB_PCDC_Close (rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified USB PCDC Comms module. Implements rm_comms_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_COMMS_USB_PCDC_Read()

fsp_err_t RM_COMMS_USB_PCDC_Read (rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Performs a read from the USB PCDC device. Implements rm_comms_api_t::read.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_ENABLED	USB device needs connecting from USB host to operate.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_COMMS_USB_PCDC_Write()

fsp_err_t RM_COMMS_USB_PCDC_Write (rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*)

Performs a write to the USB PCDC device. Implements rm_comms_api_t::write.

Return values

FSP_SUCCESS	Successfully writing data .
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_ENABLED	USB device needs connecting from USB host to operate.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_COMMS_USB_PCDC_WriteRead()

fsp_err_t RM_COMMS_USB_PCDC_WriteRead (rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params*)

Performs a write to, then a read from the USB device. Implements rm_comms_api_t::writeRead.

Return values

FSP_ERR_UNSUPPORTED	Not supported.

◆ RM_COMMS_USB_PCDC_CallbackSet()

fsp_err_t RM_COMMS_USB_PCDC_CallbackSet (rm_comms_ctrl_t *const *p_api_ctrl*, void(*)(rm_comms_callback_args_t *) *p_callback*, void *const *p_context*)

Updates the USB PCDC Comms callback. Implements rm_comms_api_t::callbackSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ rm_comms_usb_pcdc_notify_application()

void rm_comms_usb_pcdc_notify_application (rm_comms_usb_pcdc_instance_ctrl_t const * *p_ctrl*, rm_comms_event_t *event*)

(end addtogroup RM_COMMS_USB_PCDC)

5.2.6.23 Azure RTOS USBX Port (rm_usbx_port)

Modules » Connectivity

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

This USB driver works by combining USBX and r_usb_basic module.

How to Configuration

Using a class other than HMSC.

The following describes how to configure USBX using PCDC as an example.

- Select [New Stack]->[Connectivity]->[Azure RTOS PCDC]

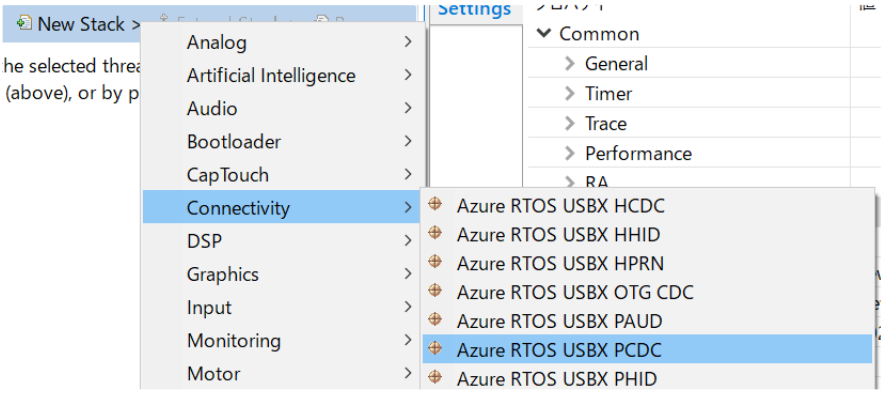


Figure 132: Select USB Device Class

- The following is displayed when selecting Azure RTOS PCDC.

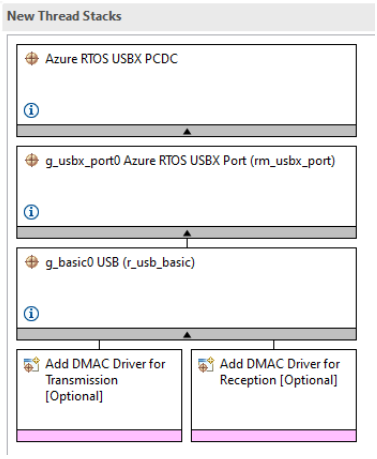


Figure 133: Select using USB Pipe

- Select the USB pipe to use.

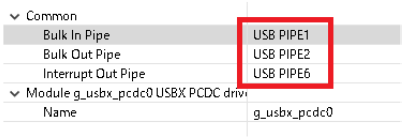


Figure 134: Select using USB Pipe

Using HMSC.

Since HMSC is used in a different way from other USBX modules, the usage is described below.

- Select [New Stack]->[Storage]->[Azure RTOS FileX on USBX]

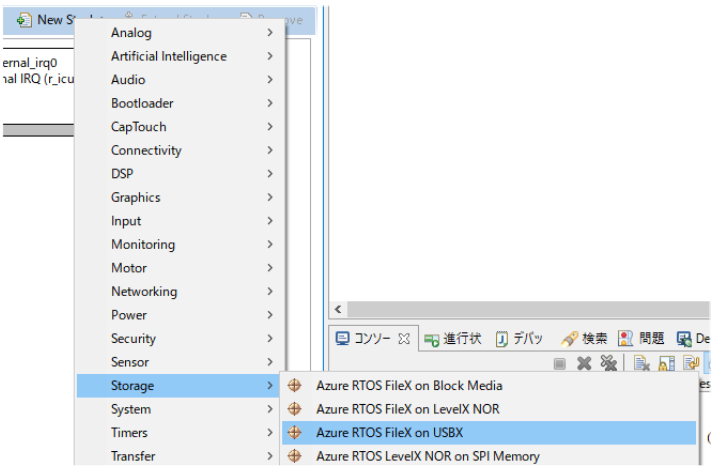


Figure 135: Select USB Device Class

- The following is displayed when selecting Filex on USBX.

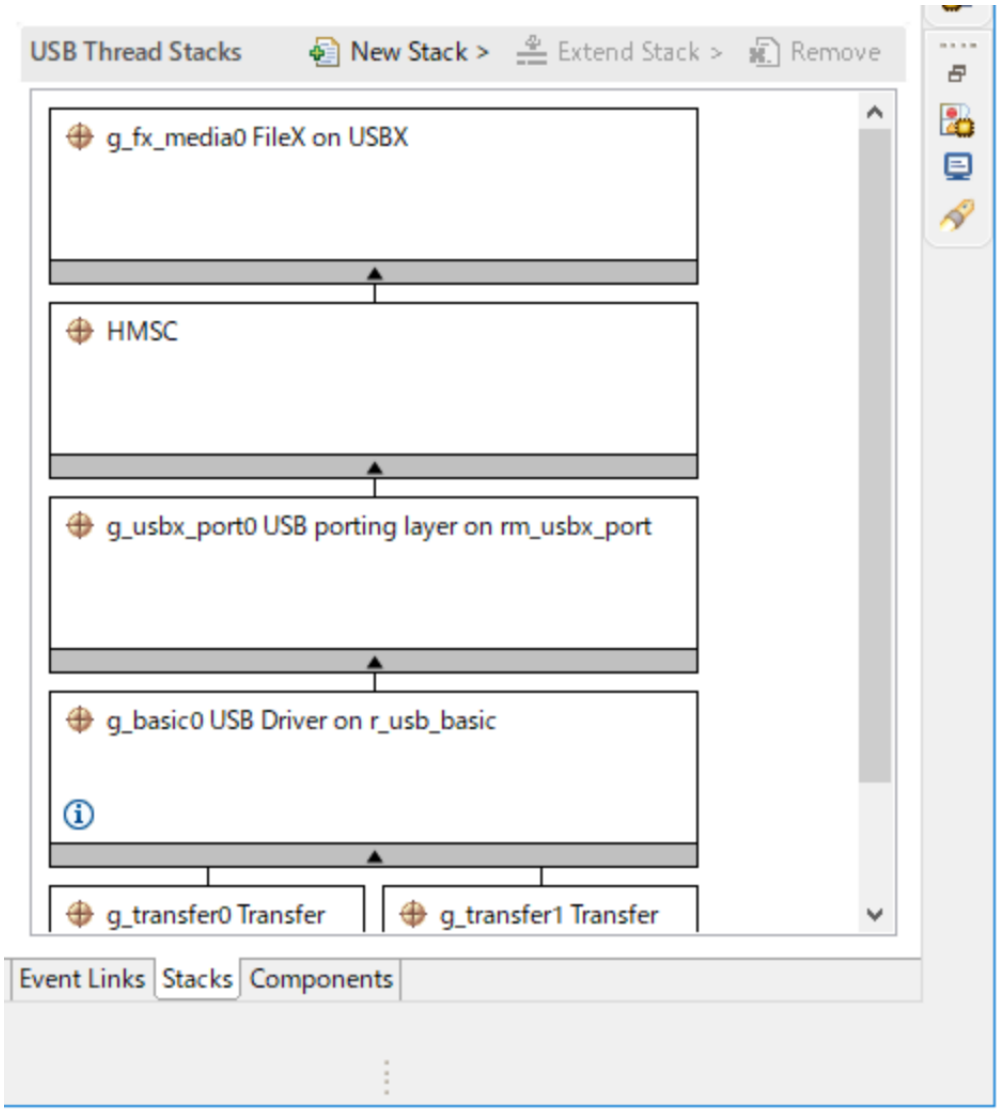


Figure 136: FileX on USBX Stack

Note

- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.
- The following are notes on using the ux_host_class_cdc_acm_read function or the ux_device_class_cdc_acm_read function
    - Please specify a multiple of MaxPacketSize to the 3rd argument(requested_length) since if the value of the 3rd argument is not multiple of MaxPacketSize, all data sent by USB Host or USB Peripheral may not be received correctly.
    - Please specify start address of the area allocated a size larger than 3rd argument(requested_length) to the 2nd argument(data_pointer or buffer).

HMSC uses FileX.

For more information on FileX, please refer to the following URL.

https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/filex/index.md

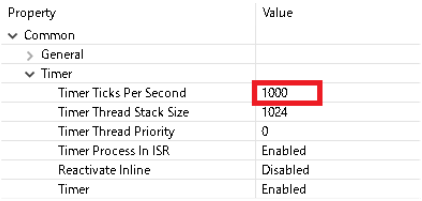
Limitations

- This USB driver does not support the multiple device class in Host mode.

- Please call the initialization function in the application program.

  Please be sure to call R_USB_Open() function after calling the following functions.

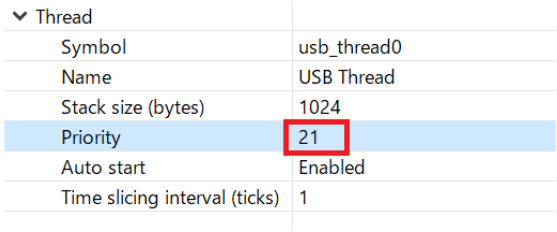
  - Peripheral
    - PCDC
      - ux_system_initialize
      - ux_device_stack_initialize
      - ux_device_stack_class_register
  - Host
    - HMSC
      - ux_system_initialize
      - ux_host_stack_initialize
      - fx_system_initialize
- Set the value for 1000 Ticks per second in the "Timer Ticks Per Second" item.



Property	Value
Common	
General	
Timer	
Timer Ticks Per Second	1000
Timer Thread Stack Size	1024
Timer Thread Priority	0
Timer Process In ISR	Enabled
Reactivate Inline	Disabled
Timer	Enabled

Figure 137: Specify value of Timer Ticks Per Second

- Set the Thread priority to a value of 21.



Thread	
Symbol	usb_thread0
Name	USB Thread
Stack size (bytes)	1024
Priority	21
Auto start	Enabled
Time slicing interval (ticks)	1

Figure 138: Specify value of Thread priority

- Call the following functions in the following order after calling the R_USB_Close() function.
  - Peripheral
    - PCDC
      - ux_device_stack_class_unregister
      - ux_device_stack_uninitialize
  - Host
    - ux_host_stack_uninitialize
- The FAT file format that HMSC supports is FAT32, FAT16 and FAT12.
- When using USBX HMSC, please be sure to check the "fx" checkbox(red frame) of the pack the user is using in the "Components" tab (green frame) as shown below.

Figure 139: Check the fx checkbox

- Please specify the correct framework size to the 2nd argument(device_framework_length_high_speed) and the 4th argument(device_framework_length_full_speed) in "ux_device_stack_initialize" function. If incorrect size is specified, USB driver does not work properly.

- ux_device_class_cdc_acm_read() API, throughput/speed may be similar in following cases.

    - Read operation in double buffer mode when compared to single buffer mode.
    - Read operation in continuous transfer mode when compared to non-continious transfer mode.

  Because of the USBX CDC device stack limits the read transfer size buffer to single packet (ie. wMaxPacketSize as per Full-speed or High-speed USB port). Please refer Azure RTOS USBX documentation for more information.

Descriptor

Templates for USBX descriptors can be found in rx/fsp/src/rm_usbx_port folder. Also, please be sure to use your vendor ID.
Change the descriptor.c.template file for each class as follows if High-speed mode is used.

- rm_usbx_pcdc_descriptor.c.template

    - Comment on lines 108 and 243.
    - Delete the "//" on lines 109 and 242.

Examples

USBX PCDC Example

PCDC loopback example is as follows.

```
#define VALUE_105 (105)

#define VALUE_2 (2)

#define VALUE_103 (103)

#define VALUE_93 (93)

/*****************************************************************************

 * Function Name : ux_cdc_device0_instance_activate
```

```
 * Description : Get instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 ******************************************************************************/
static void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_cdc = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
}
/******************************************************************************
 * End of function ux_cdc_device0_instance_activate
 ******************************************************************************/
/******************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to area store the instance pointer
 * Return value : none
 ******************************************************************************/
static void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
    g_cdc = UX_NULL;
}
/******************************************************************************
 * End of function ux_cdc_device0_instance_deactivate
 ******************************************************************************/
/******************************************************************************
 * Function Name : apl_status_change_cb
 * Description : USB callback function for USB status change
 * Arguments : ULONG status : USB status
 * Return value : UX_SUCCESS
 ******************************************************************************/
UINT apl_status_change_cb (ULONG status)
{
```

```
 switch (status)
    {
 case UX_DEVICE_ATTACHED:
            g_attach = USB_YES;
 break;
 case UX_DEVICE_REMOVED:
            g_attach = USB_NO;
 break;
 default:
 break;
    }
 return UX_SUCCESS;
}
/******************************************************************************
 * End of function apl_status_change_cb
 ******************************************************************************/
/******************************************************************************
 * Function Name : usbx_pcdc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usbx_pcdc_sample (void)
{
 fsp_err_t err;
    uint32_t ret;
    uint32_t size;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_device_stack_initialize(g_device_framework_hi_speed,
                               VALUE_103,
                               g_device_framework_full_speed,
                               VALUE_93,
                               g_string_framework,
                               VALUE_105,
```

```
                                g_language_id_framework,
                                VALUE_2,
                                apl_status_change_cb);
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_activate   =
ux_cdc_device0_instance_activate;
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_deactivate =
       ux_cdc_device0_instance_deactivate;
    ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                   _ux_device_class_cdc_acm_entry,
                                   1,
                                   0x00,
                                   (void *) &g_ux_device_class_cdc_acm0_parameter);
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
      {
 if (USB_YES == g_attach)
      {
 while (g_cdc == UX_NULL)
      {
                    ;
      }
                ret = ux_device_class_cdc_acm_read(g_cdc, g_buf, DATA_LEN,
&g_actual_length);
 if (UX_SUCCESS == ret)
      {
                    size = g_actual_length;
                    ux_device_class_cdc_acm_write(g_cdc, g_buf, size,
&g_actual_length);
      }
      }
      }
    }
```

```
}
/*****************************************************************************
 * End of function usbx_pcdc_sample
 *****************************************************************************/
```

USBX HMSC Example

HMSC storage example is as follows. See usbx_hmsc_sample for the basic operation of HMSC. Also, please refer to usbx_hmsc_sample_format for the format of the USB memory.

```
#define UX_STORAGE_BUFFER_SIZE (64 * 1024)
#define EVENT_USB_PLUG_IN (1UL << 0)
#define EVENT_USB_PLUG_OUT (1UL << 1)
#define MEMPOOL_SIZE (63488)
#define DATA_LEN (2048)
#define VALUE_32 (32)
#define VALUE_100 (100)
#define VALUE_200 (200)
#define VALUE_256 (256)
#define VALUE_1024 (1024)
#define DEVICE_INSERTION (0x01U)
#define DEVICE_REMOVAL (0x02U)
static uint16_t g_read_buf[UX_STORAGE_BUFFER_SIZE];
static uint16_t g_write_buf[UX_STORAGE_BUFFER_SIZE];
static FX_FILE g_file;
static UCHAR g_fx_media0_media_memory[UX_STORAGE_BUFFER_SIZE];
static uint8_t g_ux_pool_memory[MEMPOOL_SIZE];
static FX_MEDIA * g_p_media = UX_NULL;
TX_EVENT_FLAGS_GROUP g_usb_plug_events;
UINT usb_host_plug_event_notification(ULONG usb_event, UX_HOST_CLASS * host_class,
VOID * instance);
UINT ux_system_host_storage_fx_media_get(UX_HOST_CLASS_STORAGE       * instance,
                                         UX_HOST_CLASS_STORAGE_MEDIA **
p_storage_media,
                                         FX_MEDIA                   ** p_fx_media);
```

```
static UINT apl_change_function (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
    UINT                          status = UX_SUCCESS;
    UX_HOST_CLASS                * class;
    UX_HOST_CLASS_STORAGE        * storage;
    UX_HOST_CLASS_STORAGE_MEDIA * storage_media;
 FSP_PARAMETER_NOT_USED(host_class);
 FSP_PARAMETER_NOT_USED(instance);
 /* Check if there is a device insertion. */
 if (DEVICE_INSERTION == event)
    {
       status = ux_host_stack_class_get(_ux_system_host_class_storage_name, &class);
 if (UX_SUCCESS != status)
      {
 return status;
      }
       status = ux_host_stack_class_instance_get(class, 0, (void **) &storage);
 if (UX_SUCCESS != status)
      {
 return status;
      }
 if (UX_HOST_CLASS_INSTANCE_LIVE != storage->ux_host_class_storage_state)
      {
 return UX_ERROR;
      }
       storage_media = class->ux_host_class_media;
       g_p_media    = &storage_media->ux_host_class_storage_media;
       tx_event_flags_set(&g_usb_plug_events, EVENT_USB_PLUG_IN, TX_OR);
    }
 else if (DEVICE_REMOVAL == event) /* Check if there is a device removal. */
    {
       g_p_media = UX_NULL;
       tx_event_flags_set(&g_usb_plug_events, EVENT_USB_PLUG_OUT, TX_OR);
```

```
    }
 else
    {
 /* None */
    }
 return status;
}
/*****************************************************************************
 * Function Name : usbx_hmsc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 *****************************************************************************/
void usbx_hmsc_sample (void)
{
    ULONG      actual_length = 0;
    ULONG      actual_flags;
    UINT       tx_return;
    UINT       fx_return;
    uint16_t   data_count = 0;
    FX_MEDIA * p_media    = UX_NULL;
    CHAR       volume[VALUE_32];
    fx_system_initialize();
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(apl_change_function);
    tx_event_flags_create(&g_usb_plug_events, (CHAR *) "USB Plug Event Flags");
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
 // Wait until device inserted.
       tx_return = tx_event_flags_get(&g_usb_plug_events,
                                      EVENT_USB_PLUG_IN,
                                      TX_OR_CLEAR,
                                      &actual_flags,
```

```
                                        TX_WAIT_FOREVER);
 if (TX_SUCCESS != tx_return)
      {
            tx_thread_sleep(TX_WAIT_FOREVER);
      }
 // Get the pointer to FileX Media Control Block for a USB flash device
       p_media = g_p_media;
 // Retrieve the volume name of the opened media from the Data sector
       fx_return = fx_media_volume_get(p_media, volume, FX_DIRECTORY_SECTOR);
 if (FX_SUCCESS == fx_return)
      {
 // Set the default directory in the opened media, arbitrary name called "firstdir"
            fx_directory_default_set(p_media, "firstdir");
 // Suspend this thread for 200 time-ticks
            tx_thread_sleep(VALUE_100);
 // Try to open the file, 'counter.txt'.
            fx_return = fx_file_open(p_media, &g_file, "counter.txt",
(FX_OPEN_FOR_READ | FX_OPEN_FOR_WRITE));
 if (FX_SUCCESS != fx_return)
      {
 // The 'counter.txt' file is not found, so create a new file
                fx_return = fx_file_create(p_media, "counter.txt");
 if (FX_SUCCESS != fx_return)
      {
 break;
      }
 // Open that file
              fx_return = fx_file_open(p_media, &g_file, "counter.txt",
(FX_OPEN_FOR_READ | FX_OPEN_FOR_WRITE));
 if (FX_SUCCESS != fx_return)
      {
 break;
      }
      }
```

```
// Already open a file, then read the file in blocks
// Set a specified byte offset for reading
           fx_return = fx_file_seek(&g_file, 0);
if (FX_SUCCESS == fx_return)
     {
                fx_return = fx_file_read(&g_file, g_read_buf, DATA_LEN,
&actual_length);
if ((FX_SUCCESS == fx_return) || (FX_END_OF_FILE == fx_return))
     {
if (data_count == VALUE_1024)
     {
// empty file
                       data_count = 0;
     }
for (uint16_t data_max_count = data_count; data_count < (data_max_count +
VALUE_256); data_count++)
     {
                       g_write_buf[data_count] = data_count;
     }
// Set the specified byte offset for writing
             fx_return = fx_file_seek(&g_file, 0);
if (FX_SUCCESS == fx_return)
     {
// Write the file in blocks
                       fx_return = fx_file_write(&g_file, g_write_buf, DATA_LEN);
if (FX_SUCCESS == fx_return)
     {
     }
else
     {
             tx_thread_sleep(TX_WAIT_FOREVER);
     }
     }
     }
```

```
     }
 else
     {
             tx_thread_sleep(TX_WAIT_FOREVER);
     }
 // Close already opened file
           fx_return = fx_file_close(&g_file);
 if (FX_SUCCESS != fx_return)
     {
             tx_thread_sleep(TX_WAIT_FOREVER);
     }
           tx_thread_sleep(VALUE_200);
     }
 else
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 /* flush the media */
      fx_return = fx_media_flush(p_media);
 if (FX_SUCCESS != fx_return)
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 /* close the media */
      fx_return = fx_media_close(p_media);
 if (FX_SUCCESS != fx_return)
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 // Wait for unplugging the USB
      tx_event_flags_get(&g_usb_plug_events, EVENT_USB_PLUG_OUT, TX_OR_CLEAR,
&actual_flags, TX_WAIT_FOREVER);
    } // while(1)
}
```

```
void usbx_hmsc_sample_format (void)
{
    ULONG      actual_flags;
    UINT       tx_return;
    UINT       status = UX_SUCCESS;
    FX_MEDIA * p_media = UX_NULL;
    fx_system_initialize();
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(apl_change_function);
    tx_event_flags_create(&g_usb_plug_events, (CHAR *) "USB Plug Event Flags");
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 // Wait until device inserted.
    tx_return = tx_event_flags_get(&g_usb_plug_events, EVENT_USB_PLUG_IN,
TX_OR_CLEAR, &actual_flags, TX_WAIT_FOREVER);
 if (TX_SUCCESS != tx_return)
    {
       tx_thread_sleep(TX_WAIT_FOREVER);
    }
 // Get the pointer to FileX Media Control Block for a USB flash device
    p_media = g_p_media;
    memset(g_fx_media0_media_memory, 0x00, sizeof(g_fx_media0_media_memory));
    status = fx_media_format(p_media,                                    // Pointer to
FileX media control block.
                             p_media->fx_media_driver_entry,          // Driver entry
                             p_media->fx_media_driver_info,            // Pointer to
Block Media Driver
                             g_fx_media0_media_memory,                // Media buffer
pointer
                             p_media->fx_media_memory_size,            // Media buffer
size
 "sample",                                   // Volume Name
                             p_media->fx_media_number_of_FATs,         // Number of
FATs
                             p_media->fx_media_root_directory_entries, // Directory
```

```
Entries
                            p_media->fx_media_hidden_sectors,         // Hidden
sectors
                            (ULONG) p_media->fx_media_total_sectors, // Total sectors
                            p_media->fx_media_bytes_per_sector,       // Sector size
                            p_media->fx_media_sectors_per_cluster,    // Sectors per
cluster
                            p_media->fx_media_heads,                  // Heads (disk
media)
                            p_media->fx_media_sectors_per_track);
 if ((uint8_t) FX_SUCCESS != status)
    {
       __BKPT(0);
    }
}
```

5.2.7 DSP

Modules

DSP Modules.

5.2.8 Graphics

Modules

Detailed Description

Graphics Modules.

Modules

	Parallel Data Capture (r_pdc) Driver for the PDC peripheral on RX MCUs. This module implements the CAPTURE Interface.

	SEGGER emWin RX Serial Port (rm_emwin_rx_port) SEGGER emWin serial LCD port for RX MCUs.

5.2.8.1 Parallel Data Capture (r_pdc)

Modules » Graphics

Functions

fsp_err_t	R_PDC_Open (capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg) Powers on PDC, handles required initialization described in the hardware manual. More...
fsp_err_t	R_PDC_Close (capture_ctrl_t *const p_ctrl) Stops and closes the transfer interface, disables and powers off the PDC, clears internal driver data and disables interrupts. More...
fsp_err_t	R_PDC_CaptureStart (capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer) Starts a capture. Enables interrupts. More...
fsp_err_t	R_PDC_StatusGet (capture_ctrl_t *const p_ctrl, capture_status_t *p_status)
fsp_err_t	R_PDC_CallbackSet (capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the PDC peripheral on RX MCUs. This module implements the CAPTURE Interface.

Overview

The PDC peripheral supports interfacing with external cameras by accepting timing and data signals in order to capture incoming data. A callback is invoked every time a frame of data is accepted.

Features

- Capture incoming data into a user defined buffer
- Data bytes per pixel can be configured

- Endianess of the incoming data can be specified
- Supports configuring capture width and height
- Supports configuring vertical and horizontal sync polarity
- Horizontal and Vertical position for image/data capture can be specified
- External clock to the camera module can be adjusted
- Choice between DMAC and DTC to transfer out the captured data
- The specified user callback is invoked when a data frame is captured

Configuration

Build Time Configurations for r_pdc

The following build time configurations are defined in fsp_cfg/r_pdc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Graphics > Parallel Data Capture (r_pdc)

This module can be added to the Stacks tab via New Stack > Graphics > Parallel Data Capture (r_pdc).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_pdc0	Module name.
Input > Signal polarity > HSYNC	• High • Low	High	Specify the active polarity of the HSYNC signal.
Input > Signal polarity > VSYNC	• High • Low	High	Specify the active polarity of the VSYNC signal.
Input > Capture Specifications > Number of pixels to capture horizontally	Value must be an integer greater than 0	640	Specify the number of horizontal pixels to capture.
Input > Capture Specifications > Number of lines to capture vertically	Value must be an integer greater than 0	480	Specify the number of vertical pixels to capture.
Input > Capture Specifications > Horizontal pixel to start capture from	Value must be an integer	0	Specify the horizontal pixel to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Input > Capture	Value must be an	0	Specify the vertical line

Specifications > Line to start capture from	integer		to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Input > Bytes per pixel	Value must be an integer greater than 0	2	Specify the number of bytes per pixel of the captured image data.
Input > Clock divider	• CLK/2 • CLK/4 • CLK/6 • CLK/8 • CLK/10 • CLK/12 • CLK/14 • CLK/16	CLK/2	Specify the clock divider for clock frequency of the clock input PCKO to the PDC peripheral.
Input > Endianess	• Little • Big	Little	Specify the endianness of the captured image data.
Output > Buffer > Number of image buffers	Value must be an integer greater than 0	1	Specify the number of buffers to create.
Interrupts > Callback	Name must be a valid C symbol	g_pdc_user_callback	A user callback function must be provided. This callback is invoked for every successful frame capture and any error conditions
Interrupts > PDC Error Interrupt Priority	MCU Specific Options		Select the PDC error interrupt priority.
Interrupts > PDC Receive Data Ready Interrupt Priority	MCU Specific Options		Select the PDC receive data ready interrupt priority.

Clock Configuration

The PDC peripheral module uses the PCLKB as its clock source. The maximum clock to the camera module is PCLKB / 2.

Pin Configuration

The PCKO pin is a clock output and should be connected to the clock input of the camera. The PIXCLK pin is a clock input and should be connected to the output pixel clock of the camera. Likewise, the HSYNC and VSYNC pins must be connected to the horizontal and vertical sync signals of the camera, respectively. The PIXD0-PIXD7 pins are the 8-bit data bus input and should be connected to the relevant output pins of the camera.

Note

Camera control and serial communication pins must be configured separately and are not controlled by this

module.

Usage Notes

Interrupt Configuration

- PDC error interrupts are used by this module for reporting errors such as overrun, underrun, vertical line number setting and horizontal byte number setting errors.
- In addition to the PDC error interrupts, DMAC or DTC interrupts are also used internally to perform data transfer from this peripheral to the specified image buffer.
- Receive data ready interrupt is used as activation source for DMAC and DTC trigger.

Enabling Transfer Modules

- An option to select between DMAC or DTC is provided with DMAC as the default transfer choice.
- For further details on DMAC please refer DMAC Transfer (r_dmac)
- For further details on DTC please refer Transfer (r_dtc)

PDC setup with external camera

- Before configuring the external camera device the PDC Open API must be called in order to start clock output.
- Ensure that the memory pointed to by p_buffer is both valid and large enough to store a complete image.
- The amount of space required (in bytes) can be calculated as: size (bytes) = image width (pixels) * image height (lines) * number of bytes per pixel
- Ensure that the size above is divisible by and aligned to 32 bytes.

Examples

Basic Example

This is a basic example of minimal use of the PDC in an application. This example shows how this driver can be used for capturing data from an external I/O device such as an image sensor.

```
void g_pdc_user_callback (capture_callback_args_t * p_args)
{
 if (PDC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_capture_ready = true;
    }
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the PDC module */
```

```
    err = R_PDC_Open(&g_pdc0_ctrl, &g_pdc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the camera module at this point. This implementation is camera vendor
specific. */
    camera_module_initialization();
 /* Initialize capture ready flag to false. This gets set to true in PDC callback
upon successful frame capture. */
    g_capture_ready = false;
    err = R_PDC_CaptureStart(&g_pdc0_ctrl, g_user_buffer);
    assert(FSP_SUCCESS == err);
    uint32_t timeout_ms = PDC_DELAY_MS;
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_capture_ready) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (0U == timeout_ms)
    {
       __BKPT(0);
    }
}
static void camera_module_initialization (void)
{
 /* Camera vendor specific initialization to be done here */
}
```

Data Structures

struct	pdc_extended_cfg_t
struct	pdc_instance_ctrl_t

Enumerations

enum	pdc_event_t

enum	pdc_clock_division_t
enum	pdc_endian_t
enum	pdc_hsync_polarity_t
enum	pdc_vsync_polarity_t

Data Structure Documentation

◆ pdc_extended_cfg_t

struct pdc_extended_cfg_t		
Extended configuration structure for PDC.		
Data Fields		
pdc_clock_division_t	clock_division	Clock divider.
pdc_endian_t	endian	Endian of capture data.
pdc_hsync_polarity_t	hsync_polarity	Polarity of HSYNC input.
pdc_vsync_polarity_t	vsync_polarity	Polarity of VSYNC input.
uint8_t	pdc_ipl	PDC interrupt priority.
uint8_t	transfer_req_ipl	Transfer interrupt priority.
IRQn_Type	pdc_irq	PDC IRQ number.
IRQn_Type	transfer_req_irq	Transfer request IRQ number.
transfer_instance_t const *	p_lower_lvl_transfer	Pointer to the transfer instance the PDC should use.

◆ pdc_instance_ctrl_t

struct pdc_instance_ctrl_t
PDC instance control block. DO NOT INITIALIZE.

Enumeration Type Documentation

◆ pdc_event_t

enum pdc_event_t	
PDC events	
Enumerator	
PDC_EVENT_TRANSFER_COMPLETE	Complete frame transferred by DMAC/DTC.
PDC_EVENT_RX_DATA_READY	Receive data ready interrupt.
PDC_EVENT_FRAME_END	Frame end interrupt.
PDC_EVENT_ERR_OVERRUN	Overrun interrupt.
PDC_EVENT_ERR_UNDERRUN	Underrun interrupt.
PDC_EVENT_ERR_V_SET	Vertical line setting error interrupt.
PDC_EVENT_ERR_H_SET	Horizontal byte number setting error interrupt.

◆ pdc_clock_division_t

enum pdc_clock_division_t	
Clock divider applied to PDC clock to provide PCKO output frequency	
Enumerator	
PDC_CLOCK_DIVISION_2	CLK / 2.
PDC_CLOCK_DIVISION_4	CLK / 4.
PDC_CLOCK_DIVISION_6	CLK / 6.
PDC_CLOCK_DIVISION_8	CLK / 8.
PDC_CLOCK_DIVISION_10	CLK / 10.
PDC_CLOCK_DIVISION_12	CLK / 12.
PDC_CLOCK_DIVISION_14	CLK / 14.
PDC_CLOCK_DIVISION_16	CLK / 16.

◆ pdc_endian_t

enum pdc_endian_t	
Endian of captured data	
Enumerator	
PDC_ENDIAN_LITTLE	Data is in little endian format.
PDC_ENDIAN_BIG	Data is in big endian format.

◆ pdc_hsync_polarity_t

enum pdc_hsync_polarity_t	
Polarity of input HSYNC signal	
Enumerator	
PDC_HSYNC_POLARITY_HIGH	HSYNC signal is active high.
PDC_HSYNC_POLARITY_LOW	HSYNC signal is active low.

◆ pdc_vsync_polarity_t

enum pdc_vsync_polarity_t	
Polarity of input VSYNC signal	
Enumerator	
PDC_VSYNC_POLARITY_HIGH	VSYNC signal is active high.
PDC_VSYNC_POLARITY_LOW	VSYNC signal is active low.

Function Documentation

◆ R_PDC_Open()

fsp_err_t R_PDC_Open (capture_ctrl_t *const *p_ctrl*, capture_cfg_t const *const *p_cfg*)

Powers on PDC, handles required initialization described in the hardware manual.

Implements capture_api_t::open.

The Open function provides initial configuration for the PDC module. It powers on the module and enables the PCLKO output and the PIXCLK input. Further initialization requires the PIXCLK input to be running in order to be able to reset the PDC as part of its initialization. This clock is input from a camera module and so the reset and further initialization is performed in capture_api_t::captureStart. This function should be called once prior to calling any other PDC API functions. After the PDC is opened the Open function should not be called again without first calling the Close function.

Example:

```
/* Initialize the PDC module */

   err = R_PDC_Open(&g_pdc0_ctrl, &g_pdc0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	One or more of the following parameters is NULL 1. p_cfg is NULL 2. p_api_ctrl is NULL 3. The pointer to the transfer interface in the p_cfg parameter is NULL 4. Callback parameter is NULL. 5. Invalid IRQ number assigned
FSP_ERR_INVALID_ARGUMENT	One or more of the following parameters is incorrect 1. bytes_per_pixel is zero 2. x_capture_pixels is zero 3. y_capture_pixels is zero 4. x_capture_start_pixel + x_capture_pixels is less than 4, OR 5. x_capture_start_pixel + x_capture_pixels is greater than 4095, OR 6. y_capture_start_pixel + y_capture_pixels is greater than 4095
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_PDC_Close()

fsp_err_t R_PDC_Close (capture_ctrl_t *const *p_ctrl*)

Stops and closes the transfer interface, disables and powers off the PDC, clears internal driver data and disables interrupts.

Implements capture_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_api_ctrl is NULL
FSP_ERR_NOT_OPEN	Open has not been successfully called.

◆ R_PDC_CaptureStart()

fsp_err_t R_PDC_CaptureStart (capture_ctrl_t *const *p_ctrl*, uint8_t *const *p_buffer*)

Starts a capture. Enables interrupts.

Implements capture_api_t::captureStart.

Sets up the transfer interface to transfer data from the PDC into the specified buffer. Configures the PDC settings as previously set by the capture_api_t::open API. These settings are configured here as the PIXCLK input must be active for the PDC reset operation. When a capture is complete the callback registered during capture_api_t::open API call will be called.

Example:

```
err = R_PDC_CaptureStart(&g_pdc0_ctrl, g_user_buffer);

assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Capture start successful.
FSP_ERR_ASSERTION	One or more of the following parameters is NULL 1. p_api_ctrl is NULL 2. p_buffer is NULL
FSP_ERR_NOT_OPEN	Open has not been successfully called.
FSP_ERR_IN_USE	PDC transfer is already in progress.
FSP_ERR_TIMEOUT	Reset operation timed out.
FSP_ERR_NOT_INITIALIZED	Callback function has not been set

◆ R_PDC_StatusGet()

fsp_err_t R_PDC_StatusGet (capture_ctrl_t *const *p_ctrl*, capture_status_t * *p_status*)

Provides the pdc operating status.

Implements capture_api_t::statusGet.

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	One or more parameters is NULL.
FSP_ERR_NOT_OPEN	Open has not been successfully called.

◆ R_PDC_CallbackSet()

fsp_err_t R_PDC_CallbackSet (capture_ctrl_t *const *p_ctrl*, void(*)(capture_callback_args_t *) *p_callback*, void *const *p_context*, capture_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure.

Implements capture_api_t::callbackSet.

Return values

FSP_ERR_UNSUPPORTED	This is just a stub at present

5.2.8.2 SEGGER emWin RX Serial Port (rm_emwin_rx_port)

Modules » Graphics

Functions

fsp_err_t	RM_EMWIN_RX_PORT_PinWrite (rm_emwin_rx_port_instance_ctrl_t *const p_ctrl, bsp_io_port_pin_t const pin, bsp_io_level_t const level) Writes one GPIO pin through the configured I/O port instance. More...
fsp_err_t	RM_EMWIN_RX_PORT_Open (rm_emwin_rx_port_instance_ctrl_t *const p_ctrl, rm_emwin_rx_port_cfg_t const *const p_cfg) Opens the configured lower SPI driver. More...
fsp_err_t	RM_EMWIN_RX_PORT_Write (rm_emwin_rx_port_instance_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t const length, bool const command)

	Writes command or data bytes to the serial LCD controller through the configured lower SPI driver. More...
fsp_err_t	RM_EMWIN_RX_PORT_Close (rm_emwin_rx_port_instance_ctrl_t *const p_ctrl) Closes the configured lower SPI driver. More...
void	rm_emwin_rx_port_spi_callback (spi_callback_args_t *p_args) Records the result of a lower SPI transfer. More...
fsp_err_t	rm_emwin_rx_port_transfer_wait (rm_emwin_rx_port_instance_ctrl_t *p_ctrl, uint32_t timeout_ms) Waits for the lower SPI callback for a bounded number of milliseconds.

Detailed Description

SEGGER emWin serial LCD port for RX MCUs.

Overview

The SEGGER emWin RX Serial Port module provides the configuration and serial display support necessary to use emWin with supported LCD controllers on RX MCUs. The port connects the emWin FlexColor display driver to an FSP SPI lower driver and uses optional GPIO pins to control the display signals.

Note

The display port is managed internally by emWin through GUI_X_Config, LCD_X_Config, and LCD_X_DisplayDriver. Please consult the SEGGER emWin User Guide (UM03001) for details on how to use emWin in your project.

Features

The SEGGER emWin RX Serial Port module supports the following features:

- Runtime configuration of the GUI heap, display size, color depth, LCD controller, orientation, write cache, SPI lower driver, and optional GPIO pins
- ST7715 and ILI9341 serial LCD controller profiles
- Synchronous command and data transfers through an FSP SPI lower driver
- emWin graphics, bitmap, and widget output through the serial LCD FlexColor driver
- FreeRTOS, ThreadX, and bare-metal support

Configuration

Build Time Configurations for rm_emwin_rx_port

The following build time configurations are defined in fsp_cfg/rm_emwin_rx_port_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Graphics > SEGGER emWin RX Serial Port (rm_emwin_rx_port)

Configuration	Options	Default	Description
Memory Allocation > GUI Heap Size	Value must be a positive integer	32768	Set the heap size allocated exclusively for emWin.
Memory Allocation > Section for GUI Heap	Manual Entry	.noinit	Specify the section used for the GUI heap.
LCD Settings > Horizontal Pixels	Value must be a positive integer	480	Physical LCD horizontal pixel count.
LCD Settings > Vertical Pixels	Value must be a positive integer	272	Physical LCD vertical pixel count.
LCD Settings > Color Depth	• RGB565, 16 bpp • RGB888, 24 bpp	RGB565, 16 bpp	Serial LCD pixel format used by emWin FlexColor.
LCD Settings > Controller	• ST7715 compatible • ILI9341 compatible	ST7715 compatible	Serial LCD controller family.
LCD Settings > Orientation	• 0 degrees • 90 degrees clockwise • 180 degrees • 90 degrees counter-clockwise	0 degrees	Default display orientation.
LCD Settings > Use Write Cache	• Enabled • Disabled	Disabled	Enable the emWin FlexColor write cache for 16 bpp serial LCD output.
Pins > Data/Command Port	MCU Specific Options		Specify the port for the serial LCD D/C signal.
Pins > Data/Command Pin	Refer to the RX Configuration tool for available options.	00	Specify the pin for the serial LCD D/C signal.
Pins > Chip Select Port	MCU Specific Options		Specify the port for the serial LCD CS signal.
Pins > Chip Select Pin	Refer to the RX	00	Specify the pin for the

	Configuration tool for available options.		serial LCD CS signal.
Pins > Display Enable Port	MCU Specific Options		Specify the port for the optional serial LCD DISP signal. Select Disabled if not used.
Pins > Display Enable Pin	Refer to the RX Configuration tool for available options.	Disabled	Specify the pin for the optional serial LCD DISP signal. Select Disabled if not used.
Pins > Backlight Port	MCU Specific Options		Specify the port for the optional serial LCD backlight signal. Select Disabled if not used.
Pins > Backlight Pin	Refer to the RX Configuration tool for available options.	Disabled	Specify the pin for the optional serial LCD backlight signal. Select Disabled if not used.
Pins > Reset Port	MCU Specific Options		Specify the port for the optional serial LCD reset signal. Select Disabled if not used.
Pins > Reset Pin	Refer to the RX Configuration tool for available options.	Disabled	Specify the pin for the optional serial LCD reset signal. Select Disabled if not used.

Hardware Configuration

Add one FSP SPI lower driver instance and configure its channel, bitrate, pins, transfer support, and interrupts for the LCD panel. The RX Serial Port uses 8-bit SPI transfers.

Configure the required GPIO pins in the BSP pin configuration. Data/command and LCD chip select are normally required. Display enable, backlight, and reset can be enabled when required by the hardware. Configure each software-controlled chip-select pin as an output with an initial high level.

Display Configuration

Set the LCD controller, horizontal and vertical pixel counts, color depth, orientation, and write-cache selection in the RX Serial Port module configuration.

The horizontal and vertical pixel properties specify the physical panel dimensions before rotation. The port provides the rotated logical dimensions to emWin when a 90-degree clockwise or counter-clockwise orientation is selected.

For 16-bpp output, enable **Use Write Cache** to select the cached FlexColor controller mode. This runtime configuration is applied when GUI_Init() initializes the display.

Library Configuration

emWin is provided as a pre-compiled library. To maximize compatibility the build-time options are

configured as follows:

```
#define GUI_OS (1) // Context switch support enabled
#define GUI_MAXTASK (1) // One task supported by default
#define GUI_NUM_LAYERS (1) // Single display layer
#define GUI_SUPPORT_TOUCH (0) // Touch support excluded from this port
#define GUI_SUPPORT_MOUSE (0) // Mouse not supported by this port
#define GUI_SUPPORT_MEMDEV (1) // Memory devices available
#define GUI_WINSUPPORT (1) // Window manager available
#define GUI_SUPPORT_BIDI (1) // Bidirectional text enabled
#define GUI_DEBUG_LEVEL (2) // Parameter and consistency checks enabled (no logging)
```

All other options are left at the default settings in GUI_ConfDefaults.h.

Usage Notes

Getting Started

To get started with emWin on an RX serial LCD project the following must be performed:

1. Open the RX Configuration editor for your project
2. Add emWin to your project in the Stacks view by clicking **New Stack -> SEGGER -> SEGGER emWin**
3. Add the SEGGER emWin RX Serial Port module
4. Add and configure an FSP SPI lower driver instance
5. Configure the LCD controller, display size, color depth, orientation, write cache, and required GPIO pins
6. Ensure the lower-driver interrupts are enabled
7. Confirm the application stack and the emWin GUI heap are configured as needed
8. Click Generate Project Content to save and commit configuration changes

At this point the project is ready to build with emWin. Call GUI_Init() before using emWin drawing APIs. The port opens the configured SPI lower driver and initializes the LCD controller during display initialization. Please refer to the SEGGER emWin User Guide (UM03001) as well as demo and sample code for details on how to create a GUI application.

Serial LCD Operation

emWin calls LCD_X_Config to configure the display layer and LCD_X_DisplayDriver to initialize and control the serial LCD. The port opens the configured SPI lower driver during LCD controller initialization. The application does not need to open the lower SPI driver or call RM_EMWIN_RX_PORT_Write directly for emWin drawing. The lower driver remains responsible for the SPI clock, bitrate, pins, transfer engine, and interrupts. The port provides a bounded SPI transfer wait that reports completion, callback failure, and timeout. Call GUI_Exit() before RM_EMWIN_RX_PORT_Close(&g_rm_emwin_rx_port_ctrl) when explicit teardown is required. Closing the port releases the configured SPI lower driver.

RTOS Support

RTOS support is selected by the EMWIN_CFG_RTOS configuration macro. By default this macro follows BSP_CFG_RTOS.

- EMWIN_CFG_RTOS == 2: FreeRTOS
- EMWIN_CFG_RTOS == 1: ThreadX
- EMWIN_CFG_RTOS == 0: Bare metal

Limitations

Developers should be aware of the following limitations when using SEGGER emWin RX Serial Port:

- LCD reads are not supported
- One display layer is supported
- Because emWin is provided as a pre-compiled library, the build-time options are fixed. See the Library Configuration section for the supplied configuration

Examples

Basic Example

This is a basic example demonstrating a simple emWin application on an RX serial LCD. The screen is cleared to white, a title bar is drawn, and "Hello World!" is printed in the center.

Note

emWin manages the configured SPI lower driver internally for display output; the lower driver does not need to be opened directly by the application.

```
void emWinTask (void)
{
 /* LCD controller initialization opens the configured SPI lower driver. */
 if (0 != GUI_Init())
    {
 return;
    }
 int const x_size = LCD_GetXSize();
 int const y_size = LCD_GetYSize();
    GUI_SetBkColor(GUI_WHITE);
    GUI_Clear();
    GUI_SetColor(GUI_BLUE);
    GUI_FillRect(0, 0, x_size - 1, RM_EMWIN_RX_PORT_EXAMPLE_TITLE_HEIGHT);
    GUI_SetColor(GUI_WHITE);
    GUI_SetBkColor(GUI_BLUE);
    GUI_DispStringHCenterAt("RX Serial emWin", x_size / 2,
```

```
RM_EMWIN_RX_PORT_EXAMPLE_TITLE_TEXT_Y);
    GUI_SetBkColor(GUI_WHITE);
    GUI_SetColor(GUI_BLACK);
    GUI_DispStringHCenterAt("Hello World!", x_size / 2, y_size / 2);
 while (1)
    {
       GUI_Delay(20);
    }
}
```

Note

For further example code please consult SEGGER emWin documentation, which can be downloaded here, as well as the Quick Start Guide and example project(s) provided with your Evaluation Kit (if applicable).

Function Documentation

◆ RM_EMWIN_RX_PORT_PinWrite()

fsp_err_t RM_EMWIN_RX_PORT_PinWrite (rm_emwin_rx_port_instance_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t const *pin*, bsp_io_level_t const *level*)

Writes one GPIO pin through the configured I/O port instance.

Parameters

[in]	p_ctrl	Pointer to the control block for this instance.
[in]	pin	GPIO pin to update.
[in]	level	GPIO output level.

Return values

FSP_SUCCESS	GPIO pin write completed successfully.
FSP_ERR_ASSERTION	A required I/O port pointer is NULL.

Returns

See Common Error Codes or the lower I/O port driver for other possible return codes. This function calls

- ioport_api_t:pinWrite

◆ RM_EMWIN_RX_PORT_Open()

fsp_err_t RM_EMWIN_RX_PORT_Open (rm_emwin_rx_port_instance_ctrl_t *const *p_ctrl*, rm_emwin_rx_port_cfg_t const *const *p_cfg*)

Opens the configured lower SPI driver.

Parameters

[in]	p_ctrl	Pointer to the control block for this instance.
[in]	p_cfg	Pointer to the configuration for this instance.

Return values

FSP_SUCCESS	Lower SPI driver is open.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_ALREADY_OPEN	Lower SPI driver is already owned by another user.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- spi_api_t:open

◆ RM_EMWIN_RX_PORT_Write()

fsp_err_t RM_EMWIN_RX_PORT_Write (rm_emwin_rx_port_instance_ctrl_t *const *p_ctrl*, uint8_t const *const *p_data*, uint32_t const *length*, bool const *command*)

Writes command or data bytes to the serial LCD controller through the configured lower SPI driver.

Parameters

[in]	p_ctrl	Pointer to the control block for this instance.
[in]	p_data	Data to write.
[in]	length	Number of bytes to write.
[in]	command	True for command bytes, false for data bytes.

Return values

FSP_SUCCESS	Data write completed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_IN_USE	Another transfer is in progress.
FSP_ERR_INVALID_SIZE	Length is zero.
FSP_ERR_NOT_OPEN	The emWin RX port is not open.
FSP_ERR_TIMEOUT	The lower SPI callback was not received before the transfer timeout.
FSP_ERR_TRANSFER_ABORTED	The lower SPI callback reported an error event.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- spi_api_t:write

◆ RM_EMWIN_RX_PORT_Close()

fsp_err_t RM_EMWIN_RX_PORT_Close (rm_emwin_rx_port_instance_ctrl_t *const *p_ctrl*)

Closes the configured lower SPI driver.

Parameters

[in]	p_ctrl	Pointer to the control block for this instance.

Return values

FSP_SUCCESS	Lower SPI driver is closed.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_IN_USE	A transfer is in progress.
FSP_ERR_NOT_OPEN	The emWin RX port is not open.

Returns

See Common Error Codes or lower level drivers for other possible return codes. This function calls

- spi_api_t:close

◆ rm_emwin_rx_port_spi_callback()

void rm_emwin_rx_port_spi_callback (spi_callback_args_t * *p_args*)

Records the result of a lower SPI transfer.

Parameters

[in]	p_args	Lower SPI callback arguments.

5.2.9 Input

Modules

Detailed Description

Input Modules.

Modules

External IRQ (r_icu)	Driver for the ICU peripheral on RX MCUs. This module implements

	the External IRQ Interface.

5.2.9.1 External IRQ (r_icu)

Modules » Input

Functions

fsp_err_t	R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const p_api_ctrl, external_irq_cfg_t const *const p_cfg)
fsp_err_t	R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const p_api_ctrl)
fsp_err_t	R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const p_api_ctrl)
fsp_err_t	R_ICU_ExternalIrqCallbackSet (external_irq_ctrl_t *const p_api_ctrl, void(*p_callback)(external_irq_callback_args_t *), void *const p_context, external_irq_callback_args_t *const p_callback_memory)
fsp_err_t	R_ICU_ExternalIrqClose (external_irq_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the ICU peripheral on RX MCUs. This module implements the External IRQ Interface.

Overview

The Interrupt Controller Unit (ICU) controls various interrupt requests from the peripheral modules and the IRQ pin, and generates an interrupt request to the CPU and a transfer request to the DTC and DMAC. The R_ICU software module only implements the External IRQ Interface. The external_irq interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

Note

Multiple instances are used when more than one external interrupt is needed. Configure each instance with different channels and properties as needed for the specific interrupt.

Features

- Supports configuring interrupts for IRQ pins on the target MCUs
  - Enabling and disabling interrupt generation.
  - Configuring interrupt trigger on rising edge, falling edge, both edges, or low level signal.
  - Enabling and disabling the IRQ noise filter.
- Supports configuring a user callback function, which will be invoked by the HAL module when an external pin interrupt is generated.

Configuration

Build Time Configurations for r_icu

The following build time configurations are defined in fsp_cfg/r_icu_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Input > External IRQ (r_icu)

This module can be added to the Stacks tab via New Stack > Input > External IRQ (r_icu).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_external_irq0	Module name.
Channel	Value must be a non-negative integer	0	Specify the hardware channel.
Trigger	MCU Specific Options		Select the signal edge or state that triggers an interrupt.
Digital Filtering	MCU Specific Options		Select if the digital noise filter should be enabled.
Filter Source	MCU Specific Options		Select the clock source for digital filter (Only affected when the digital filter is enabled).
Filter Mode	MCU Specific Options		Select the digital noise filter operation mode (Only affected when the digital filter is enabled).
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	MCU Specific Options		Select the clock divider for the digital noise filter.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn

		triggers
Pin Interrupt Priority	MCU Specific Options	Select the PIN interrupt priority.

Clock Configuration

The ICU peripheral module doesn't require any specific clock settings.

Note

The digital filter uses PCLKB as the clock source for sampling the IRQ pin.

Pin Configuration

The pin for the external interrupt channel must be configured as an input with IRQ Input Enabled.

Limitation

Pin configuration does not show conflicts when same IRQ is used by multiple pins.

Usage Notes

Digital Filter

The digital filter is used to reject trigger conditions that are too short. The trigger condition must be longer than three periods of the filter clock. The filter clock frequency is determined by PCLKB and the external_irq_clock_source_div_t setting.

MIN_PULSE_WIDTH = EXTERNAL_IRQ_PCLKB_DIV / PCLKB_FREQUENCY * 3

DMAC/DTC

When using an External IRQ pin to trigger a DMAC/DTC transfer, the External IRQ pin must be opened before the transfer instance is opened.

Examples

Basic Example

This is a basic example of minimal use of the ICU in an application.

```
#define ICU_IRQN_PIN BSP_IO_PORT_09_PIN_04
#define ICU_IRQN 10
/* Called from icu_irq_isr */
void external_irq_callback (external_irq_callback_args_t * p_args)
{
    (void) p_args;
    g_external_irq_complete = 1;
}
```

```
void simple_example (void)
{
 /* Example Configuration */
 external_irq_cfg_t icu_cfg =
    {
       .channel          = ICU_IRQN,
       .trigger          = EXTERNAL_IRQ_TRIG_RISING,
       .filter_enable    = false,
       .clock_source_div = EXTERNAL_IRQ_CLOCK_SOURCE_DIV_1,
       .p_callback       = external_irq_callback,
       .p_context       = 0,
       .ipl             = 1,
       .irq             = (IRQn_Type) ICU_EVENT_FIXED_ICU_IRQ14,
    };
 /* Configure the external interrupt. */
 fsp_err_t err = R_ICU_ExternalIrqOpen(&g_icu_ctrl, &icu_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable the external interrupt. */
 /* Enable not required when used with ELC or DMAC. */
    err = R_ICU_ExternalIrqEnable(&g_icu_ctrl);
    assert(FSP_SUCCESS == err);
 while (0 == g_external_irq_complete)
    {
 /* Wait for interrupt. */
    }
}
```

Data Structures

struct	icu_extended_cfg_t
struct	icu_instance_ctrl_t

Enumerations

enum	e_external_irq_digital_filter_t
enum	e_external_irq_filter_mode_t

Data Structure Documentation

◆ icu_extended_cfg_t

struct icu_extended_cfg_t		
Extended ICU interface configuration		
Data Fields		
e_external_irq_digital_filter_t	filter_src	Select digital filter clock source when digital filter is enabled.
e_external_irq_filter_mode_t	filter_mode	Digital filter sampling mode (number of consecutive level matches).

◆ icu_instance_ctrl_t

struct icu_instance_ctrl_t	
ICU private control block. DO NOT MODIFY. Initialization occurs when R_ICU_ExternalIrqOpen is called.	
Data Fields	
uint32_t	open
	Used to determine if channel control block is in use.
IRQn_Type	irq
	ICU interrupt number.
uint8_t	channel
	Channel.
void *	p_context

Field Documentation

◆ p_context

void* icu_instance_ctrl_t::p_context
Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

Enumeration Type Documentation

◆ e_external_irq_digital_filter_t

enum e_external_irq_digital_filter_t	
External IRQ input pin digital filtering clock selection.	
Enumerator	
EXTERNAL_IRQ_DIGITAL_FILTER_PCLK_DIV	Digital filter using clock PCLK dividers.
EXTERNAL_IRQ_DIGITAL_FILTER_LOCO	Digital filter using clock LOCO.

◆ e_external_irq_filter_mode_t

enum e_external_irq_filter_mode_t	
IRQ digital noise filter operation mode.	
Enumerator	
EXTERNAL_IRQ_FILTER_MODE_3_CYCLES	Digital filter using 3 consecutive level matches.
EXTERNAL_IRQ_FILTER_MODE_4_CYCLES	Digital filter using 4 consecutive level matches.

Function Documentation

◆ R_ICU_ExternalIrqOpen()

fsp_err_t R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const *p_api_ctrl*, external_irq_cfg_t const *const *p_cfg*)

Configure an IRQ input pin for use with the external interrupt interface. Implements external_irq_api_t::open.

The Open function is responsible for preparing an external IRQ pin for operation.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	One of the following is invalid: • p_ctrl or p_cfg is NULL
FSP_ERR_ALREADY_OPEN	The channel specified has already been opened. No configurations were changed. Call the associated Close function to reconfigure the channel.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in p_cfg is not available on the device selected in r_bsp_cfg.h.
FSP_ERR_INVALID_ARGUMENT	p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function.
FSP_ERR_UNSUPPORTED	An input argument is not supported by selected mode.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

◆ R_ICU_ExternalIrqEnable()

fsp_err_t R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt for specified channel at ICU. Implements external_irq_api_t::enable.

Return values

FSP_SUCCESS	Interrupt Enabled successfully.
FSP_ERR_ASSERTION	The p_ctrl parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_IRQ_BSP_DISABLED	Requested IRQ is not defined in this system

◆ R_ICU_ExternalIrqDisable()

fsp_err_t R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const *p_api_ctrl*)

Disable external interrupt for specified channel at ICU. Implements external_irq_api_t::disable.

Return values

FSP_SUCCESS	Interrupt disabled successfully.
FSP_ERR_ASSERTION	The p_ctrl parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_IRQ_BSP_DISABLED	Requested IRQ is not defined in this system

◆ R_ICU_ExternalIrqCallbackSet()

fsp_err_t R_ICU_ExternalIrqCallbackSet (external_irq_ctrl_t *const *p_api_ctrl*, void(*)(external_irq_callback_args_t *) *p_callback*, void *const *p_context*, external_irq_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements external_irq_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_ICU_ExternalIrqClose()

fsp_err_t R_ICU_ExternalIrqClose (external_irq_ctrl_t *const *p_api_ctrl*)

Close the external interrupt channel. Implements external_irq_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The channel is not opened.

5.2.10 Monitoring

Modules

Detailed Description

Monitoring Modules.

Modules

	Clock Accuracy Circuit (r_cac) Driver for the CAC peripheral on RX MCUs. This module implements the CAC Interface.
	CRC (r_crc) Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface.
	Data Operation Circuit (r_doc) Driver for the DOC peripheral on RX MCUs. This module implements the DOC Interface.
	Independent Watchdog (r_iwdt) Driver for the IWDT peripheral on RX MCUs. This module implements the WDT Interface.
	Programmable Voltage Detection (r_pvd) Driver for the PVD and PVD peripherals on RX MCUs. This module implements the Voltage Detection Interface.
	Watchdog (r_wdt) Driver for the WDT peripheral on RX MCUs. This module implements the WDT Interface.

5.2.10.1 Clock Accuracy Circuit (r_cac)

Modules » Monitoring

Functions

fsp_err_t	R_CAC_Open (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

fsp_err_t	R_CAC_StartMeasurement (cac_ctrl_t *const p_ctrl)
fsp_err_t	R_CAC_StopMeasurement (cac_ctrl_t *const p_ctrl)
fsp_err_t	R_CAC_Read (cac_ctrl_t *const p_ctrl, uint32_t *const p_counter)
fsp_err_t	R_CAC_Close (cac_ctrl_t *const p_ctrl)
fsp_err_t	R_CAC_CallbackSet (cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void *const p_context, cac_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CAC peripheral on RX MCUs. This module implements the CAC Interface.

Overview

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of measurement clock edges that occur between two edges of the reference clock.

Features

- Supports clock frequency-measurement and monitoring based on a reference signal input
- Reference can be either an externally supplied clock source or an internal clock source
- An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow.
- A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks.
- Edge-detection options for the reference clock are configurable as rising, falling, or both.

Configuration

Build Time Configurations for r_cac

The following build time configurations are defined in fsp_cfg/r_cac_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > Clock Accuracy Circuit (r_cac)

This module can be added to the Stacks tab via New Stack > Monitoring > Clock Accuracy Circuit (r_cac).

Configuration	Options	Default	Description
General > Unit	Must be a valid CAC unit	0	Specify the hardware unit.
Name	Name must be a valid C symbol	g_cac0	Module name.
Reference clock divider	• 32 • 128 • 1024 • 8192	32	Reference clock divider.
Reference clock source	MCU Specific Options		Reference clock source.
Reference clock digital filter	• Disabled • Sampling clock =Measuring freq • Sampling clock =Measuring freq/4 • Sampling clock =Measuring freq/16	Disabled	Reference clock digital filter.
Reference clock edge detect	• Rising • Falling • Both	Rising	Reference clock edge detection.
Measurement clock divider	• 1 • 4 • 8 • 32	1	Measurement clock divider.
Measurement clock source	MCU Specific Options		Measurement clock source.
Upper Limit Threshold	Value must be a non-negative integer, between 0 to 65535	0	Top end of allowable range for measurement completion.
Lower Limit Threshold	Value must be a non-negative integer, between 0 to 65535	0	Bottom end of allowable range for measurement completion.
Frequency Error Interrupt Priority	MCU Specific Options		CAC frequency error interrupt priority.
Measurement End Interrupt Priority	MCU Specific Options		CAC measurement end interrupt priority.
Overflow Interrupt Priority	MCU Specific Options		CAC overflow interrupt priority.
Callback	Name must be a valid C symbol	NULL	Function name for callback

Clock Configuration

The CAC measurement clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO
4. MOCO
5. LOCO
6. PCLKB
7. IWDT
8. PLL1 output clock P/Q/R (divided by 16)
9. PLL2 output clock P/Q/R (divided by 16)

The CAC reference clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO
4. MOCO
5. LOCO
6. PCLKB
7. IWDT
8. External Clock Source (CACREF)
9. PLL1 output clock P/Q/R (divided by 16)
10. PLL2 output clock P/Q/R (divided by 16)

Note:

- For RX74M/RX74N device:
  - SUBCLOCK and LOCO are only used in Unit 0.
  - PLL1 output clock P/Q/R (divided by 16) are only used in Unit 1.
  - PLL2 output clock P/Q/R (divided by 16) are only used in Unit 2.

Pin Configuration

The CACREF pin can be configured to provide the reference clock for CAC measurements.

Usage Notes

Measurement Accuracy

The clock measurement result may be off by up to one pulse depending on the phase difference between the edge detection circuit, digital filter, and CACREF pin signal, if applicable.

Frequency Error Interrupt

The frequency error interrupt is only triggered at the end of a CAC measurement. This means that there will be a measurement complete interrupt in addition to the frequency error interrupt.

Examples

Basic Example

This is a basic example of minimal use of the CAC in an application.

```
volatile uint32_t g_callback_complete;
void cac_basic_example (void)
{
    g_callback_complete = 0;
 fsp_err_t err = R_CAC_Open(&g_cac_ctrl, &g_cac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    (void) R_CAC_StartMeasurement(&g_cac_ctrl);
 /* Wait for measurement to complete. */
 while (0 == g_callback_complete)
    {
    }
    uint32_t value;
 /* Read the CAC measurement. */
    (void) R_CAC_Read(&g_cac_ctrl, &value);
}
/* Called when measurement is completed. */
static void r_cac_callback (cac_callback_args_t * p_args)
{
 if (CAC_EVENT_MEASUREMENT_COMPLETE == p_args->event)
    {
       g_callback_complete = 1U;
    }
}
```

Data Structures

struct	cac_instance_ctrl_t
struct	cac_extended_cfg_t

Data Structure Documentation

◆ cac_instance_ctrl_t

struct cac_instance_ctrl_t
CAC instance control block. DO NOT INITIALIZE.

◆ cac_extended_cfg_t

struct cac_extended_cfg_t		
CAC Extended Configuration		
Data Fields		
uint32_t	unit	CAC Unit.

Function Documentation

◆ R_CAC_Open()

fsp_err_t R_CAC_Open (cac_ctrl_t *const *p_ctrl*, cac_cfg_t const *const *p_cfg*)

The Open function configures the CAC based on the provided user configuration settings.

Return values

FSP_SUCCESS	CAC is available and available for measurement(s).
FSP_ERR_ASSERTION	An argument is invalid.
FSP_ERR_ALREADY_OPEN	The CAC has already been opened.
FSP_ERR_IP_UNIT_NOT_PRESENT	Unit does not exist on this MCU.

Note

There is only a single CAC peripheral.

◆ R_CAC_StartMeasurement()

fsp_err_t R_CAC_StartMeasurement (cac_ctrl_t *const *p_ctrl*)

Start the CAC measurement process.

Return values

FSP_SUCCESS	CAC measurement started.
FSP_ERR_ASSERTION	NULL provided for p_instance_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_StopMeasurement()

fsp_err_t R_CAC_StopMeasurement (cac_ctrl_t *const *p_ctrl*)

Stop the CAC measurement process.

Return values

FSP_SUCCESS	CAC measuring has been stopped.
FSP_ERR_ASSERTION	NULL provided for p_instance_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_Read()

fsp_err_t R_CAC_Read (cac_ctrl_t *const *p_ctrl*, uint32_t *const *p_counter*)

Read and return the CAC status and counter registers.

Return values

FSP_SUCCESS	CAC read successful.
FSP_ERR_ASSERTION	An argument is NULL.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_Close()

fsp_err_t R_CAC_Close (cac_ctrl_t *const *p_ctrl*)

Release any resources that were allocated by the Open() or any subsequent CAC operations.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	NULL provided for p_instance_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_CallbackSet()

fsp_err_t R_CAC_CallbackSet (cac_ctrl_t *const *p_ctrl*, void(*)(cac_callback_args_t *) *p_callback*, void *const *p_context*, cac_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements cac_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.10.2 CRC (r_crc)

Modules » Monitoring

Functions

fsp_err_t	R_CRC_Open (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
fsp_err_t	R_CRC_Close (crc_ctrl_t *const p_ctrl)
fsp_err_t	R_CRC_Calculate (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *calculatedValue)
fsp_err_t	R_CRC_CalculatedValueGet (crc_ctrl_t *const p_ctrl, uint32_t *calculatedValue)
fsp_err_t	R_CRC_SnoopEnable (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
fsp_err_t	R_CRC_SnoopDisable (crc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface.

Overview

The CRC module provides a API to calculate 8, 16 and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using industry-standard polynomials.

Features

- CRC module supports the following 8 and 16 bit CRC polynomials which operates on 8-bit data in parallel
    - X^8+X^2+X+1 (CRC-8)
    - X^16+X^15+X^2+1 (CRC-16)
    - X^16+X^12+X^5+1 (CRC-CCITT)
- CRC module supports the following 32 bit CRC polynomials which operates on 32-bit data in parallel
    - X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X+ 1 (CRC-32)
    - X^32+ X^28+ X^27+ X^26+ X^25+ X^23+ X^22+ X^20+ X^19+ X^18+X^14+X^13+X^11+X^10+X^9+X^8+X^6+1 (CRC-32C)
- CRC module can calculate CRC with LSB first or MSB first bit order.

Configuration

Build Time Configurations for r_crc

The following build time configurations are defined in fsp_cfg/r_crc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > CRC (r_crc)

This module can be added to the Stacks tab via New Stack > Monitoring > CRC (r_crc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_crc0	Module name.
CRC Polynomial	MCU Specific Options		Select the CRC polynomial.
Bit Order	MCU Specific Options		Select the CRC bit order.
Snoop Address	MCU Specific Options		Select the SCI register address CRC snoop

Clock Configuration

There is no clock configuration for the CRC module.

Pin Configuration

This module does not use I/O pins.

Usage Notes

CRC Snoop

The CRC snoop function monitors reads from and writes to a specified I/O register address and performs CRC calculation on the data read from and written to the register address automatically. Instead of calling R_CRC_Calculate on a block of data, R_CRC_SnoopEnable is called to start monitoring reads/writes and R_CRC_CalculatedValueGet is used to obtain the current CRC.

Note

Snoop mode is available for transmit/receive operations on SCI only.

Limitations

When using CRC32 polynomial functions the CRC module produces the same results as popular online CRC32 calculators, but it is important to remember a few important points.

- Online CRC32 calculators allow the input to be any number of bytes. The FSP CRC32 API function uses 32-bit words. This means the online calculations must be 'padded' to end on a 32-bit boundary.
- Online CRC32 calculators usually invert the output prior to presenting it as a result. It is up to the application program to include this step if needed.
- The seed value of 0xFFFFFFFF needs to be used by both the online calculator and the R_CRC module API (CRC32 polynomials)
- Make sure the bit orientation of the R_CRC CRC32 is set for LSB and that you have CRC32 selected and not CRC32C.
- Some online CRC tools XOR the final result with 0xFFFFFFFF.

Examples

Basic Example

This is a basic example of minimal use of the CRC module in an application.

```
void crc_example (void)
{
    uint32_t length;
    uint32_t uint8_calculated_value;
    length = sizeof(g_data_8bit) / sizeof(g_data_8bit[0]);
 crc_input_t example_input =
    {
       .p_input_buffer = g_data_8bit,
       .num_bytes      = length,
       .crc_seed       = 0,
    };
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
```

```
 /* 8-bit CRC calculation */
 R_CRC_Calculate(&crc_ctrl, &example_input, &uint8_calculated_value);
}
```

Snoop Example

This example demonstrates CRC snoop operation.

```
void crc_snoop_example (void)
{
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* Open SCI Driver */
 /* Configure Snoop address and enable snoop mode */
 R_CRC_SnoopEnable(&crc_ctrl, 0);
 /* Perfrom SCI read/Write operation depending on the SCI snoop address configure */
 /* Read CRC value */
 R_CRC_CalculatedValueGet(&crc_ctrl, &g_crc_buff);
}
```

Data Structures

struct	crc_instance_ctrl_t

Data Structure Documentation

◆ crc_instance_ctrl_t

struct crc_instance_ctrl_t
Driver instance control structure.

Function Documentation

◆ R_CRC_Open()

fsp_err_t R_CRC_Open (crc_ctrl_t *const *p_ctrl*, crc_cfg_t const *const *p_cfg*)

Open the CRC driver module

Implements crc_api_t::open

Open the CRC driver module and initialize the driver control block according to the passed-in configuration structure.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Module already open
FSP_ERR_UNSUPPORTED	Hardware not support this feature.

◆ R_CRC_Close()

fsp_err_t R_CRC_Close (crc_ctrl_t *const *p_ctrl*)

Close the CRC module driver.

Implements crc_api_t::close

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_Calculate()

fsp_err_t R_CRC_Calculate (crc_ctrl_t *const *p_ctrl*, crc_input_t *const *p_crc_input*, uint32_t * *calculatedValue*)

Perform a CRC calculation on a block of 8-bit/32-bit (for 32-bit polynomial) data.

Implements crc_api_t::calculate

This function performs a CRC calculation on an array of 8-bit/32-bit (for 32-bit polynomial) values and returns an 8-bit/32-bit (for 32-bit polynomial) calculated value

Return values

FSP_SUCCESS	Calculation successful.
FSP_ERR_ASSERTION	Either p_ctrl, inputBuffer, or calculatedValue is NULL.
FSP_ERR_INVALID_ARGUMENT	length value is NULL, or not 4-byte aligned when 32-bit CRC polynomial function is configured.
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_CalculatedValueGet()

fsp_err_t R_CRC_CalculatedValueGet (crc_ctrl_t *const *p_ctrl*, uint32_t * *calculatedValue*)

Return the current calculated value.

Implements crc_api_t::crcResultGet

CRC calculation operates on a running value. This function returns the current calculated value.

Return values

FSP_SUCCESS	Return of calculated value successful.
FSP_ERR_ASSERTION	Either p_ctrl or calculatedValue is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_SnoopEnable()

fsp_err_t R_CRC_SnoopEnable (crc_ctrl_t *const *p_ctrl*, uint32_t *crc_seed*)

Configure the snoop channel and set the CRC seed.

Implements crc_api_t::snoopEnable

The CRC calculator can operate on reads and writes over any of the first ten SCI channels. For example, if set to channel 0, transmit, every byte written out SCI channel 0 is also sent to the CRC calculator as if the value was explicitly written directly to the CRC calculator.

Return values

FSP_SUCCESS	Snoop configured successfully.
FSP_ERR_ASSERTION	Pointer to control structure is NULL
FSP_ERR_NOT_OPEN	The driver is not opened.
FSP_ERR_UNSUPPORTED	SNOOP operation is not supported.

◆ R_CRC_SnoopDisable()

fsp_err_t R_CRC_SnoopDisable (crc_ctrl_t *const *p_ctrl*)

Disable snooping.

Implements crc_api_t::snoopDisable

Return values

FSP_SUCCESS	Snoop disabled.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.
FSP_ERR_UNSUPPORTED	SNOOP operation is not supported.

5.2.10.3 Data Operation Circuit (r_doc)

Modules » Monitoring

Functions

fsp_err_t	R_DOC_Open (doc_ctrl_t *const p_api_ctrl, doc_cfg_t const *const p_cfg)
fsp_err_t	R_DOC_Close (doc_ctrl_t *const p_api_ctrl)

fsp_err_t	R_DOC_Read (doc_ctrl_t *const p_api_ctrl, uint32_t *p_result)
fsp_err_t	R_DOC_Write (doc_ctrl_t *const p_api_ctrl, uint32_t data)
fsp_err_t	R_DOC_CallbackSet (doc_ctrl_t *const p_api_ctrl, void(*p_callback)(doc_callback_args_t *), void *const p_context, doc_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the DOC peripheral on RX MCUs. This module implements the DOC Interface.

Overview

Features

The DOC HAL module peripheral is used to compare, add or subtract 16-bit or 32-bit[1] data and can detect the following events:

- Comparison Mode
  - Data is equal to the configured reference data setting.
  - Data is not equal to the configured reference data setting.
  - Data is less than the configured reference data setting[2].
  - Data is greater than the configured reference data setting[2].
  - Data is inside of a configurable pair of reference data settings[2].
  - Data is outside of a configurable pair of reference data settings[2].
- Addition Mode - Overflow of an addition operation
- Subtraction Mode - Underflow of a subtraction operation

A user-defined callback can be created to inform the CPU when any of above events occur.

Note

1. Operating on 32-bit data is not supported on all MCUs.
2. This comparison mode is not supported on all MCUs.

Configuration

Build Time Configurations for r_doc

The following build time configurations are defined in fsp_cfg/r_doc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > Data Operation Circuit (r_doc)

This module can be added to the Stacks tab via New Stack > Monitoring > Data Operation Circuit

(r_doc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_doc0	Module name.
Event	MCU Specific Options		Select the event that will trigger the DOC interrupt.
Bit Width	MCU Specific Options		The bit width for DOC operations.
Reference/Initial Data	Value must be an integer greater than or equal to 0.	0	Enter Initial Value for Addition/Subtraction or enter reference value for comparison.
Additional Reference Data	Value must be an integer greater than or equal to 0.	0	Additional reference data used for Window Compare modes.
Callback	Name must be a valid C symbol	NULL	A user callback function must be provided. This will be called from the interrupt service routine (ISR) when the configured DOC event occurs.
DOC Interrupt Priority	MCU Specific Options		Select the DOC interrupt priority.

Clock Configuration

The DOC HAL module does not require a specific clock configuration.

Pin Configuration

The DOC HAL module does not require and specific pin configurations.

Usage Notes

DMAC/DTC Integration

DOC can be used with DMAC Transfer (r_dmac) or Transfer (r_dtc) to write to the input register without CPU intervention. DMAC is more useful for most DOC applications because it can be started directly from software. To write DOC input data with DTC/DMAC, set transfer_info_t::p_dest to R_DOC->DODIR.

Examples

Basic Example

This is a basic example of minimal use of the R_DOC in an application. This example shows how this driver can be used for continuous 16 bit addition operation while reading the result at every overflow event.

```
#define DOC_EXAMPLE_VALUE 0xF000
uint32_t g_callback_event_counter = 0;
/* This callback is called when DOC overflow event occurs. It is registered in
doc_cfg_t when R_DOC_Open is
 * called. */
void doc_callback (doc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write data to the DOC Data Input Register and read the result of addition from
status register when an
  * interrupt occurs. */
 for (int i = 0; i < 5; i++)
    {
       err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
      assert(FSP_SUCCESS == err);
 if (g_callback_event_counter >= 1)
      {
            uint32_t result;
 /* Read the result of the operation */
            err = R_DOC_Read(&g_doc_ctrl, &result);
      assert(FSP_SUCCESS == err);
       }
```

```
    }
}
```

Function Documentation

◆ R_DOC_Open()

fsp_err_t R_DOC_Open (doc_ctrl_t *const *p_api_ctrl*, doc_cfg_t const *const *p_cfg*)

Opens and configures the Data Operation Circuit (DOC) in comparison, addition or subtraction mode and sets initial data for addition or subtraction, or reference data for comparison.

Example:

```
 /* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
```

Return values

FSP_SUCCESS	DOC successfully configured.
FSP_ERR_ALREADY_OPEN	Module already open.
FSP_ERR_ASSERTION	One or more pointers point to NULL or callback is NULL or the interrupt vector is invalid.

◆ R_DOC_Close()

fsp_err_t R_DOC_Close (doc_ctrl_t *const *p_api_ctrl*)

Closes the module driver. Enables module stop mode.

Return values

FSP_SUCCESS	Module successfully closed.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	Pointer pointing to NULL.

Note

This function will disable the DOC interrupt in the NVIC.

◆ R_DOC_Read()

fsp_err_t R_DOC_Read (doc_ctrl_t *const *p_api_ctrl*, uint32_t * *p_result*)

Returns the result of addition/subtraction.

Example:

```
            uint32_t result;
/* Read the result of the operation */
            err = R_DOC_Read(&g_doc_ctrl, &result);
      assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Status successfully read.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	One or more pointers point to NULL.

◆ R_DOC_Write()

fsp_err_t R_DOC_Write (doc_ctrl_t *const *p_api_ctrl*, uint32_t *data*)

Writes to the DODIR - DOC Input Register.

Example:

```
       err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
      assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Values successfully written to the registers.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	One or more pointers point to NULL.

◆ R_DOC_CallbackSet()

fsp_err_t R_DOC_CallbackSet (doc_ctrl_t *const *p_api_ctrl*, void(*)(doc_callback_args_t *) *p_callback*, void *const *p_context*, doc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements doc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.10.4 Independent Watchdog (r_iwdt)

Modules » Monitoring

Functions

fsp_err_t	R_IWDT_Refresh (wdt_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IWDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg)
fsp_err_t	R_IWDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status)
fsp_err_t	R_IWDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status)
fsp_err_t	R_IWDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count)
fsp_err_t	R_IWDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout)
fsp_err_t	R_IWDT_CallbackSet (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the IWDT peripheral on RX MCUs. This module implements the WDT Interface.

Overview

The independent watchdog timer is used to recover from unexpected errors in an application. The timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the IWDT resets the device or generates an IRQ.

Features

The IWDT HAL module has the following key features:

- When the IWDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an IRQ
- The WDT supports following modes:[1]
  - In auto start mode, the WDT begins counting at reset.
  - In register start mode, the WDT can be started from the application[1].

Selecting a Watchdog

RX MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

	WDT	IWDT
Start Mode	The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode).	On most of MCUs, the IWDT can only be configured by hardware to start automatically[1].
Clock Source	The WDT runs off a peripheral clock.	The IWDT has its own clock source which improves safety.

Note

1. Refer to the MCU hardware user's manual or datasheet to determine if IWDT supports register start mode.

Configuration

The configuration of start mode setting is on the **BSP** tab. The default setting for the start mode is register start mode.

When using register start mode, no need to update the configuration of start mode on the BSP tab. Configure the independence watchdog timer on the Stacks tab.

Note

*When using auto start mode, configurations on the **Stacks** tab are ignored. Configure the watchdog using the OFS settings on the BSP tab. These settings include the following:*

- *Start Mode*
- *Timeout Period*
- *Dedicated Clock Frequency Divisor*
- *Window End Position*
- *Window Start Position*
- *Reset Interrupt Request Select*
- *Stop Control*

Review the OFS0 properties window to see additional details.

Build Time Configurations for r_iwdt

The following build time configurations are defined in fsp_cfg/r_iwdt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
NMI Support	MCU Specific Options		NMI is not supported.
IRQ Support	MCU Specific Options		If enabled, code for IRQ support is included in the build.

Configurations for Monitoring > Independent Watchdog (r_iwdt)

This module can be added to the Stacks tab via New Stack > Monitoring > Independent Watchdog (r_iwdt).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_wdt0	Module name.
Timeout	MCU Specific Options		Select the independent watchdog timeout in cycles.
Clock Division Ratio	MCU Specific Options		Select the independent watchdog clock divisor.
Window Start Position	MCU Specific Options		Select the allowed independent watchdog refresh start point in %.
Window End Position	MCU Specific Options		Select the allowed independent watchdog refresh end point in %.
Reset Control	MCU Specific Options		Select what happens when the independent watchdog timer expires.(NMI request output is not supported)
Stop Control	MCU Specific Options		Select the independent watchdog state in low power mode.
NMI/IRQ callback	Name must be a valid C symbol	NULL	A user callback function can be provided here. If this

		callback function is provided, it is called from the interrupt service routine (ISR) when the watchdog triggers.
IRQ Priority	MCU Specific Options	Select the IWDT interrupt priority.

Clock Configuration

The IWDT clock is based on the IWDTCLK frequency. You can set the IWDTCLK frequency divider using the **BSP** tab of the RX Configuration editor.

Pin Configuration

This module does not use I/O pins.

Usage Notes

NMI Interrupt

The independent watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

Note

NMI request output is not supported in IWDT module.
We can use IRQ instead of NMI in IWDT module.

IRQ Interrupt

The watchdog timer uses an IRQ, and the interrupt priority level must be configured before use. When the IRQ is triggered, the callback function registered during open is called.

Note

When using the IWDT in software start mode with IRQ and the timer underflows, the IWDT status must be reset by calling R_IWDT_StatusClear before restarting the timer via R_IWDT_Refresh.

Period Calculation

The IWDT operates from IWDTCLK. With a IWDTCLK of 120000 Hz, the maximum time from the last refresh to device reset or IRQ generation will be just below 35 seconds as detailed below.

IWDTCLK = 120000 Hz
Clock division ratio = IWDTCLK / 256
Timeout period = 16384 cycles
WDT clock frequency = 120000 Hz / 256 = 468.75 Hz
Cycle time = 1 / 468.75 Hz = 2.1333 ms
Timeout = 2.1333 ms x 16384 cycles = 34.95 seconds

Limitations

Developers should be aware of the following limitations when using the IWDT:

- NMI request output is not supported in IWDT module. Please use **Reset Output** or **IRQ Output** for the configuration of Reset Control. Because IWDT will stop the counter in low power mode, then your application must restart the independent watchdog by calling R_IWDT_Refresh() after the MCU wakes from low power mode.

Examples

IWDT Basic Example

This is a basic example of minimal use of the IWDT in an application.

```
void iwdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the IWDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset based on the setting. */
       (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
    }
}
```

IWDT Advanced Example

This example demonstrates using a start window and gives an example for refreshing before the counter underflows to prevent reset.

```
#define IWDT_TIMEOUT_COUNTS (2048U)
#define IWDT_MAX_COUNTER (IWDT_TIMEOUT_COUNTS - 1U)
#define IWDT_START_WINDOW_75 ((IWDT_MAX_COUNTER * 3) / 4)
void iwdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
```

```
#if defined(R_SYSTEM_RSTSR1_IWDTRF_Msk)
 /* (Optional) Check if the IWDTRF flag is set to know if the system is
  * recovering from a IWDT reset. */
 if (R_SYSTEM->RSTSR1_b.IWDTRF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
#endif
 /* Open the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize other application code. */
 /* Do not call R_IWDT_Refresh() in auto start mode unless the
  * counter is in the acceptable refresh window. */
    (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
 while (true)
    {
 /* Application work here. */
 /* (Optional) If there is a chance the application takes less time than
  * the start window, verify the IWDT counter is past the start window
  * before refreshing the IWDT. */
       uint32_t iwdt_counter = 0U;
 do
      {
 /* Read the current IWDT counter value. */
            err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
      assert(FSP_SUCCESS == err);
      } while (iwdt_counter >= IWDT_START_WINDOW_75);
 /* Refresh before the counter underflows to prevent reset or NMI. */
       (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
    }
}
```

Data Structures

struct	iwdt_instance_ctrl_t
struct	iwdt_extended_cfg_t

Data Structure Documentation

◆ iwdt_instance_ctrl_t

struct iwdt_instance_ctrl_t	
IWDT control block. DO NOT INITIALIZE. Initialization occurs when wdt_api_t::open is called.	
Data Fields	
uint32_t	wdt_open
	Indicates whether the open() API has been successfully called.
void *	p_context
	Placeholder for user data. Passed to the user callback in wdt_callback_args_t.
void(*	p_callback)(wdt_callback_args_t *p_args)
	Callback provided when a WDT NMI ISR occurs.

◆ iwdt_extended_cfg_t

struct iwdt_extended_cfg_t		
IWDT Extended Configuration		
Data Fields		
IRQn_Type	irq	IWUNI IRQ number.
uint8_t	ipl	IWUNI interrupt priority.

Function Documentation

◆ R_IWDT_Refresh()

fsp_err_t R_IWDT_Refresh (wdt_ctrl_t *const *p_api_ctrl*)

Refresh the Independent Watchdog Timer. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI/IRQ ISR to run.

Example:

```
/* Refresh before the counter underflows to prevent reset based on the setting. */

      (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
```

Return values

FSP_SUCCESS	IWDT successfully refreshed.
FSP_ERR_ASSERTION	One or more parameters are NULL pointers.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_Open()

fsp_err_t R_IWDT_Open (wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t const *const *p_cfg*)

Register the IWDT NMI callback.

Example:

```
/* Initializes the module. */

   err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
```

Return values

FSP_SUCCESS	IWDT successfully configured.
FSP_ERR_ASSERTION	Null Pointer.
FSP_ERR_NOT_ENABLED	An attempt to open the IWDT when the OFS0 register is not configured for auto-start mode.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

◆ R_IWDT_StatusClear()

fsp_err_t R_IWDT_StatusClear (wdt_ctrl_t *const *p_api_ctrl*, const wdt_status_t *status*)

Clear the IWDT status and error flags. Implements wdt_api_t::statusClear.

Return values

FSP_SUCCESS	IWDT flag(s) successfully cleared.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.
FSP_ERR_UNSUPPORTED	This function is only valid if the IWDT generates an NMI/IRQ when an error occurs.

Note

When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

◆ R_IWDT_StatusGet()

fsp_err_t R_IWDT_StatusGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_status_t *const *p_status*)

Read the IWDT status flags.

Indicates both status and error conditions.

Return values

FSP_SUCCESS	IWDT status successfully read.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.
FSP_ERR_UNSUPPORTED	This function is only valid if the IWDT generates an NMI when an error occurs.

Note

When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

◆ R_IWDT_CounterGet()

fsp_err_t R_IWDT_CounterGet (wdt_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_count*)

Read the current count value of the IWDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current IWDT counter value. */
            err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
      assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	IWDT current count successfully read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_TimeoutGet()

fsp_err_t R_IWDT_TimeoutGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_timeout_values_t *const *p_timeout*)

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

Return values

FSP_SUCCESS	IWDT timeout information retrieved successfully.
FSP_ERR_ASSERTION	One or more parameters are NULL pointers.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_CallbackSet()

fsp_err_t R_IWDT_CallbackSet (wdt_ctrl_t *const *p_ctrl*, void(*)(wdt_callback_args_t *) *p_callback*, void *const *p_context*, wdt_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements wdt_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.10.5 Programmable Voltage Detection (r_pvd)

Modules » Monitoring

Functions

fsp_err_t	R_PVD_Open (voltage_detection_ctrl_t *const p_api_ctrl, voltage_detection_cfg_t const *const p_cfg)
fsp_err_t	R_PVD_Close (voltage_detection_ctrl_t *const p_api_ctrl)
fsp_err_t	R_PVD_StatusGet (voltage_detection_ctrl_t *const p_api_ctrl, voltage_detection_status_t *p_voltage_detection_status)
fsp_err_t	R_PVD_StatusClear (voltage_detection_ctrl_t *const p_api_ctrl)
fsp_err_t	R_PVD_CallbackSet (voltage_detection_ctrl_t *const p_api_ctrl, void(*p_callback)(voltage_detection_callback_args_t *), void *const p_context, voltage_detection_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the PVD and PVD peripherals on RX MCUs. This module implements the Voltage Detection Interface.

Overview

The Programmable Voltage Detection module configures the voltage monitors to detect when a power supply pin or a voltage detector pin voltages crosses a specified threshold.

Features

The PVD HAL module supports the following functions:

- Seven run-time configurable voltage monitors (Voltage Monitor 1, Voltage Monitor 2, Voltage Monitor 4, Voltage Monitor 5)
    - Configurable voltage threshold
    - Digital filter (Available on specific MCUs)
    - Support for both interrupt or polling
        - NMI or maskable interrupt can be configured (NMI support for Voltage Monitor 1 & Voltage Monitor 2 only).
    - Voltage monitor interrupt generation condition can be configured (rising, falling, or both edge event detection).
    - Support for resetting the MCU when V_{CC} falls below/rises above configured threshold.
      Note: Voltage monitor availability is limited based on the selected MCU.

Configuration

Build Time Configurations for r_pvd

The following build time configurations are defined in fsp_cfg/r_pvd_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > Programmable Voltage Detection (r_pvd)

This module can be added to the Stacks tab via New Stack > Monitoring > Programmable Voltage Detection (r_pvd).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_pvd0	Module name.
Monitor Number	MCU Specific Options		Select the PVD monitor.
Digital Filter	MCU Specific Options		Enable the digital filter and select the digital filter clock divider.
Voltage Threshold	MCU Specific Options		Select the programmable voltage detection threshold.
Detection Response	MCU Specific Options		Select what happens when the voltage crosses the threshold voltage.
Voltage Slope	MCU Specific Options		Select interrupt generation on rising

			voltage, falling voltage or both.
Negation Delay	MCU Specific Options		Negation of the monitor signal can either be delayed from the reset event or from voltage returning to normal range.
Monitor Interrupt Callback	Name must be a valid C symbol.	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers.
PVD Monitor Interrupt Priority	MCU Specific Options		Select the PVD Monitor interrupt priority.

Clock Configuration

The LOCO clock must be enabled in order to use the digital filter.

Usage Notes

Startup Edge Detection

If V_{CC} is below the threshold prior to configuring the voltage monitor for falling edge detection, the monitor will immediately detect the a falling edge condition. If V_{CC} is above the threshold prior to configuring the monitor for rising edge detection, the monitor will not detect a rising edge condition until V_{CC} falls below the threshold and then rises above it again.

Voltage Monitor 0

The PVD HAL module only supports configuring voltage monitor 1 and voltage monitor 2. Voltage monitor 0 can be configured by setting the appropriate bits in the OFS1 register. This means that voltage monitor 0 settings cannot be changed at runtime.

Voltage monitor 0 supports the following features

- Configurable Voltage Threshold (V_{DET0})
- Reset the device when V_{CC} falls below V_{DET0}

Limitations

- The digital filter must be disabled when using voltage monitors in Software Standby or Deep Software Standby.
- Deep Software Standby mode is not possible if the voltage monitor is configured to reset the MCU.
- Reset generated by the VCC-rise detection is supported only on devices with PVD.
- Voltage Monitor 4, Voltage Monitor 5 does not support interrupt feature.
- PVD Monitor 1 and PVD Monitor 2 must have the same security attribution. Either both

monitors are used in the secure project, or both monitors are used in the non-secure project.

Examples

Basic Example

This is a basic example of minimal use of the PVD in an application.

```
void basic_example (void)
{
 fsp_err_t err = R_PVD_Open(&g_pvd_ctrl, &g_pvd_cfg);
    assert(FSP_SUCCESS == err);
 while (1)
    {
 voltage_detection_status_t status;
       err = R_PVD_StatusGet(&g_pvd_ctrl, &status);
      assert(FSP_SUCCESS == err);
 if (VOLTAGE_DETECTION_THRESHOLD_CROSSING_DETECTED == status.crossing_detected)
      {
            err = R_PVD_StatusClear(&g_pvd_ctrl);
      assert(FSP_SUCCESS == err);
 /* Do something */
       }
    }
}
```

Interrupt Example

This is a basic example of using a PVD instance that is configured to generate an interrupt.

```
void interrupt_example (void)
{
 fsp_err_t err = R_PVD_Open(&g_pvd_ctrl, &g_pvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (1)
    {
```

```
 /* Application Process */
 /* Application will be interrupted when Vcc crosses the configured threshold. */
    }
}
/* Called when Vcc crosses configured threshold. */
void pvd_callback (voltage_detection_callback_args_t * p_args)
{
 if (VOLTAGE_DETECTION_CURRENT_STATE_BELOW_THRESHOLD == p_args->current_state)
    {
 /* Do Something */
    }
}
```

Reset Example

This is a basic example of using a PVD instance that is configured to reset the MCU.

```
void reset_example (void)
{
 if (1U == R_SYSTEM->RSTSR0_b.PVD1RF)
    {
 /* The system is coming out of reset because Vcc crossed configured voltage
threshold. */
 /* Clear Voltage Monitor 1 Reset Detect Flag. */
       R_SYSTEM->RSTSR0_b.PVD1RF = 0;
    }
 fsp_err_t err = R_PVD_Open(&g_pvd_ctrl, &g_pvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (1)
    {
 /* Application Process */
 /* Application will reset when Vcc crosses the configured threshold. */
    }
}
```

Data Structures

struct	pvd_instance_ctrl_t

Data Structure Documentation

◆ pvd_instance_ctrl_t

struct pvd_instance_ctrl_t
PVD instance control structure

Function Documentation

◆ R_PVD_Open()

fsp_err_t R_PVD_Open (voltage_detection_ctrl_t *const *p_api_ctrl*, voltage_detection_cfg_t const *const *p_cfg*)

Initializes a voltage monitor and detector according to the passed-in configuration structure.

Parameters

[in]	p_api_ctrl	Pointer to the control structure for the driver instance
[in]	p_cfg	Pointer to the configuration structure for the driver instance

Note

Digital filter is not to be used with standby modes.
Startup time can take on the order of milliseconds for some configurations.

Example:

```
fsp_err_t err = R_PVD_Open(&g_pvd_ctrl, &g_pvd_cfg);
```

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	Requested configuration was invalid
FSP_ERR_INVALID_ARGUMENT	An argument is invalid.
FSP_ERR_ALREADY_OPEN	The instance was already opened
FSP_ERR_IN_USE	Another instance is already using the desired monitor
FSP_ERR_UNSUPPORTED	Digital filter was enabled on a device that does not support it

◆ R_PVD_Close()

fsp_err_t R_PVD_Close (voltage_detection_ctrl_t *const *p_api_ctrl*)

Disables the PVD peripheral. Closes the driver instance.

Parameters

[in]	p_api_ctrl	Pointer to the control block structure for the driver instance

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_NOT_OPEN	Driver is not open

◆ R_PVD_StatusGet()

fsp_err_t R_PVD_StatusGet (voltage_detection_ctrl_t *const *p_api_ctrl*, voltage_detection_status_t * *p_voltage_detection_status*)

Get the current state of the monitor (threshold crossing detected, voltage currently above or below threshold).

Parameters

[in]	p_api_ctrl	Pointer to the control structure for the driver instance
[out]	p_voltage_detection_status	Pointer to status structure

Example:

```
err = R_PVD_StatusGet(&g_pvd_ctrl, &status);
```

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_NOT_OPEN	Driver is not open
FSP_ERR_UNSUPPORTED	This monitor does not support status feature

◆ R_PVD_StatusClear()

fsp_err_t R_PVD_StatusClear (voltage_detection_ctrl_t *const *p_api_ctrl*)

Clears the latched status of the monitor.

Parameters

[in]	p_api_ctrl	Pointer to the control structure for the driver instance

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_NOT_OPEN	Driver is not open
FSP_ERR_UNSUPPORTED	This monitor does not support status feature

◆ R_PVD_CallbackSet()

fsp_err_t R_PVD_CallbackSet (voltage_detection_ctrl_t *const *p_api_ctrl*, void(*)(voltage_detection_callback_args_t *) *p_callback*, void *const *p_context*, voltage_detection_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements voltage_detection_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.10.6 Watchdog (r_wdt)

Modules » Monitoring

Functions

fsp_err_t	R_WDT_Refresh (wdt_ctrl_t *const p_ctrl)
fsp_err_t	R_WDT_Open (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

fsp_err_t	R_WDT_StatusClear (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
fsp_err_t	R_WDT_StatusGet (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
fsp_err_t	R_WDT_CounterGet (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
fsp_err_t	R_WDT_TimeoutGet (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
fsp_err_t	R_WDT_CallbackSet (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the WDT peripheral on RX MCUs. This module implements the WDT Interface.

Overview

The watchdog timer is used to recover from unexpected errors in an application. The watchdog timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the WDT resets the device or generates an IRQ.

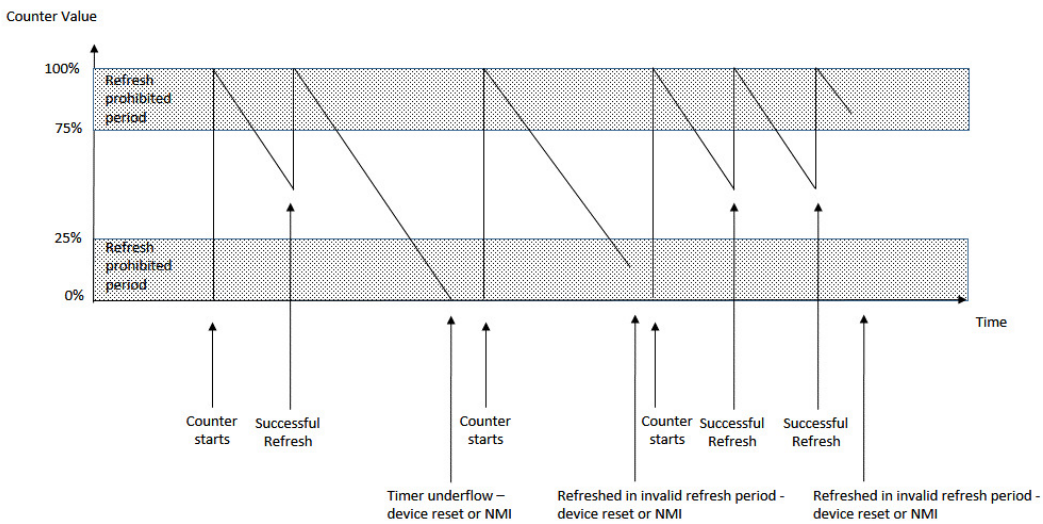


Figure 140: Watchdog Timer Operation Example

Features

The WDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an IRQ
- The WDT has two supported modes:
  - In auto start mode, the WDT begins counting at reset.
  - In register start mode, the WDT can be started from the application.

Selecting a Watchdog

RX MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

	WDT	IWDT
Start Mode	The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode).	The IWDT can only be configured by hardware to start automatically.
Clock Source	The WDT runs off a peripheral clock.	The IWDT has its own clock source which improves safety.

Configuration

The configuration of start mode setting is on the **BSP** tab. The default setting for the start mode is register start mode.

When using register start mode, no need to update the configuration of start mode on the BSP tab. Configure the watchdog timer on the Stacks tab.

Note

*When using auto start mode, configurations on the **Stacks** tab are ignored. Configure the watchdog using the OFS settings on the BSP tab.*

Build Time Configurations for r_wdt

The following build time configurations are defined in fsp_cfg/r_wdt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
NMI Support	Disabled	Disabled	NMI is not supported.

IRQ Support	• Enabled • Disabled	Disabled	If enabled, code for IRQ support is included in the build.

Configurations for Monitoring > Watchdog (r_wdt)

This module can be added to the Stacks tab via New Stack > Monitoring > Watchdog (r_wdt).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_wdt0	Module name.
Timeout	• 1,024 Cycles • 4,096 Cycles • 8,192 Cycles • 16,384 Cycles	16,384 Cycles	Select the watchdog timeout in cycles.
Clock Division Ratio	• PCLK/4 • PCLK/64 • PCLK/128 • PCLK/512 • PCLK/2048 • PCLK/8192	PCLK/8192	Select the watchdog clock divisor.
Window Start Position	• 100 (Window Position Not Specified) • 75 • 50 • 25	100 (Window Position Not Specified)	Select the allowed watchdog refresh start point in %.
Window End Position	• 0 (Window Position Not Specified) • 25 • 50 • 75	0 (Window Position Not Specified)	Select the allowed watchdog refresh end point in %.
Reset Control	• Reset Output • IRQ Output	Reset Output	Select what happens when the watchdog timer expires.(NMI request output is not supported)
Stop Control	MCU Specific Options		Select the watchdog state in low power mode.
NMI/IRQ Callback	Name must be a valid C symbol	NULL	A user callback function must be provided if the WDT is configured to generate an NMI or IRQ when the timer underflows or a refresh error occurs. If this callback function is provided, it will be

		called from the NMI/IRQ handler each time the watchdog triggers.
IRQ Priority	MCU Specific Options	Select the WDT interrupt priority.

Clock Configuration

The WDT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time. The maximum timeout period with PCLKB running at 60 MHz is approximately 2.2 seconds.

Pin Configuration

This module does not use I/O pins.

Usage Notes

NMI Interrupt

The watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

Note

NMI request output is not supported in WDT module.

IRQ Interrupt

The watchdog timer uses an IRQ, and the interrupt priority level must be configured before use. When the IRQ is triggered, the callback function registered during open is called.

Note

When using the WDT in software start mode with IRQ and the timer underflows, the WDT status must be reset by calling R_WDT_StatusClear before restarting the timer via R_WDT_Refresh.

Period Calculation

The WDT operates from PCLKB. With a PCLKB of 60 MHz, the maximum time from the last refresh to device reset or IRQ generation will be just over 2.2 seconds as detailed below.

PLCKB = 60 MHz
Clock division ratio = PCLKB / 8192
Timeout period = 16384 cycles
WDT clock frequency = 60 MHz / 8192 = 7.324 kHz
Cycle time = 1 / 7.324 kHz = 136.53 us
Timeout = 136.53 us x 16384 cycles = 2.23 seconds

Limitations

Developers should be aware of the following limitations when using the WDT:

- NMI request output is not supported in WDT module. Please use **Reset Output** or **IRQ Output** only for the configuration of Reset Control.
- If the WDT is configured to stop the counter in low power mode, then your application must

restart the watchdog by calling R_WDT_Refresh() after the MCU wakes from low power mode.

Examples

WDT Basic Example

This is a basic example of minimal use of the WDT in an application.

```
void wdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the WDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* In register start mode, start the watchdog by calling R_WDT_Refresh. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset. */
      err = R_WDT_Refresh(&g_wdt0_ctrl);
      assert(FSP_SUCCESS == err);
    }
}
```

WDT Advanced Example

This example demonstrates using a start window for refreshing before the counter underflows to prevent reset.

```
#define WDT_TIMEOUT_COUNTS (16384U)
#define WDT_MAX_COUNTER (WDT_TIMEOUT_COUNTS - 1U)
#define WDT_START_WINDOW_75 ((WDT_MAX_COUNTER * 3) / 4)
```

```
void wdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Check if the WDTRF flag is set to know if the system is
  * recovering from a WDT reset. */
 if (R_SYSTEM->RSTSR1_b.WDT0RF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
 /* Open the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize other application code. */
 /* (Register start mode) Call R_WDT_Refresh() to start the WDT in register
  * start mode. Do not call R_WDT_Refresh() in auto start mode unless the
  * counter is in the acceptable refresh window. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* (Optional) If there is a chance the application takes less time than
  * the start window, verify the WDT counter is past the start window
  * before refreshing the WDT. */
       uint32_t wdt_counter = 0U;
 do
      {
 /* Read the current WDT counter value. */
            err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
      assert(FSP_SUCCESS == err);
      } while (wdt_counter >= WDT_START_WINDOW_75);
 /* Refresh before the counter underflows to prevent reset. */
```

```
        err = R_WDT_Refresh(&g_wdt0_ctrl);
        assert(FSP_SUCCESS == err);
    }
}
```

Data Structures

struct	wdt_instance_ctrl_t
struct	wdt_extended_cfg_t

Data Structure Documentation

◆ wdt_instance_ctrl_t

struct wdt_instance_ctrl_t
WDT private control block. DO NOT MODIFY. Initialization occurs when R_WDT_Open() is called.

◆ wdt_extended_cfg_t

struct wdt_extended_cfg_t		
WDT Extended Configuration		
Data Fields		
IRQn_Type	irq	WUNI interrupt number.
uint8_t	ipl	WUNI interrupt priority.

Function Documentation

◆ R_WDT_Refresh()

fsp_err_t R_WDT_Refresh (wdt_ctrl_t *const *p_ctrl*)

Refresh the watchdog timer. Implements wdt_api_t::refresh.

In addition to refreshing the watchdog counter this function can be used to start the counter in register start mode.

Example:

```
/* Refresh before the counter underflows to prevent reset. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	WDT successfully refreshed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function only returns FSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI/IRQ ISR to run.

◆ R_WDT_Open()

fsp_err_t R_WDT_Open (wdt_ctrl_t *const *p_ctrl*, wdt_cfg_t const *const *p_cfg*)

Configure the WDT in register start mode. In auto-start_mode the NMI/IRQ callback can be registered. Implements wdt_api_t::open.

This function should only be called once as WDT configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

Return values

FSP_SUCCESS	WDT successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

Note

In auto start mode the only valid configuration option is for registering the callback for the NMI/IRQ ISR if NMI/IRQ output has been selected.

◆ R_WDT_StatusClear()

fsp_err_t R_WDT_StatusClear (wdt_ctrl_t *const *p_ctrl*, const wdt_status_t *status*)

Clear the WDT status and error flags. Implements wdt_api_t::statusClear.

Return values

FSP_SUCCESS	WDT flag(s) successfully cleared.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.
FSP_ERR_UNSUPPORTED	This function is only valid if the watchdog generates an NMI/IRQ when an error occurs.

Note

When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

◆ R_WDT_StatusGet()

fsp_err_t R_WDT_StatusGet (wdt_ctrl_t *const *p_ctrl*, wdt_status_t *const *p_status*)

Read the WDT status flags. Implements wdt_api_t::statusGet.

Indicates both status and error conditions.

Return values

FSP_SUCCESS	WDT status successfully read.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.
FSP_ERR_UNSUPPORTED	This function is only valid if the watchdog generates an NMI/IRQ when an error occurs.

Note

When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

◆ R_WDT_CounterGet()

fsp_err_t R_WDT_CounterGet (wdt_ctrl_t *const *p_ctrl*, uint32_t *const *p_count*)

Read the current count value of the WDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current WDT counter value. */
            err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
      assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	WDT current count successfully read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.

◆ R_WDT_TimeoutGet()

fsp_err_t R_WDT_TimeoutGet (wdt_ctrl_t *const *p_ctrl*, wdt_timeout_values_t *const *p_timeout*)

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

Return values

FSP_SUCCESS	WDT timeout information retrieved successfully.
FSP_ERR_ASSERTION	Null Pointer.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.

◆ R_WDT_CallbackSet()

fsp_err_t R_WDT_CallbackSet (wdt_ctrl_t *const *p_ctrl*, void(*)(wdt_callback_args_t *) *p_callback*, void *const *p_context*, wdt_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements wdt_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.11 Motor

Modules

Detailed Description

Motor Modules.

Modules

	Phase Counting Interface (r_pcif) Driver for the PCIF peripheral on RX MCUs. This module implements the PCIF Interface.
	Motor Current Controller (rm_motor_current) Calculation process for the motor control on RX MCUs. This module implements the Motor current Interface.
	ADC and PWM Modulation (rm_motor_driver) Calculation process for the motor control on RX MCUs. This module implements the Motor driver Interface.
	Motor Encoder Vector Control (rm_motor_encoder) Control a SPM motor on RX MCUs. This module implements the Motor Encoder Vector Control (rm_motor_encoder).
	Motor Angle (rm_motor_estimate) Calculation process for the motor control on RX MCUs. This module implements the Motor angle Interface.
	Motor Inertia estimate (rm_motor_inertia_estimate) Measurement and calculation process for the motor control on RX MCUs. This module implements the Motor Inertia Estimate Interface.
	Motor Position Controller (rm_motor_position) Calculation process for the motor control on RX MCUs. This module implements the Motor position Interface.

	Motor return origin (rm_motor_return_origin) Search origin position process for the motor control on RX MCUs. This module implements the Motor Return Origin Function Interface.
	Motor Angle (rm_motor_sense_encoder) Calculation process for the motor control on RX MCUs. This module implements the Motor angle Interface.
	Motor Sensorless Vector Control (rm_motor_sensorless) Usual control of a SPM motor on RX MCUs. This module implements the Motor Sensorless Vector Control (rm_motor_sensorless).
	Motor Speed Controller (rm_motor_speed) Calculation process for the motor control on RX MCUs. This module implements the Motor speed Interface.

5.2.11.1 Phase Counting Interface (r_pcif)

Modules » Motor

Functions

fsp_err_t	R_PCIF_Open (pcif_ctrl_t *const p_ctrl, pcif_cfg_t const *const p_cfg)
fsp_err_t	R_PCIF_Stop (pcif_ctrl_t *const p_ctrl, pcif_stop_mode_t stop_mode)
fsp_err_t	R_PCIF_Start (pcif_ctrl_t *const p_ctrl)
fsp_err_t	R_PCIF_SetCounter (pcif_ctrl_t *const p_ctrl, pcif_phc_counter_info_t const *const p_counter_data, bool reset_counter)
fsp_err_t	R_PCIF_StatusGet (pcif_ctrl_t *const p_ctrl, pcif_status_t *const p_status)
fsp_err_t	R_PCIF_SoftwareEventGenerate (pcif_ctrl_t *const p_ctrl)
fsp_err_t	R_PCIF_Close (pcif_ctrl_t *const p_ctrl)

Detailed Description

Driver for the PCIF peripheral on RX MCUs. This module implements the PCIF Interface.

Overview

The Phase Counting Interface (PCIF) outputs A-phase, B-phase, and Z-phase signals for encoder-style motor control systems. The PCIF has two operating modes:

- **Phase Counting Mode**: Generates ABZ output signals from position data, edge count data, and carrier period events supplied by software or ELC.
- **Incremental Mode**: Receives external ABZ encoder signals and outputs ABZ signals with a configured resolution ratio.

Features

- Generate ABZ outputs from angle or position information configured by software.
- Output ABZ pulses with reduced resolution from an external encoder signal.
- Provide position feedback information to the host controller.
- Generate ABZ pulses without external components such as an FPGA.
- Support Phase Counting mode.
    - Use internal or ELC carrier period events to update edge data.
    - Update edge data and position data at run time.
    - Notify the user callback when a carrier period occurs or an error is detected.
    - Set position data at run time.
- Support Incremental mode.
    - Configure the output resolution ratio for an external encoder signal.
    - Generate Z-phase output or clear the position counter by software trigger.
    - Use crest/trough ELC events to clear internal counters or generate Z-phase output.
- Support Z-phase generation in both modes.
- Support ABZ-phase output inversion in both modes.
- Start PCIF operation by API.
- Stop PCIF operation immediately or by phase-counting synchronized stop.
- Get current operation state and position monitor information at run time.

Configuration

Build Time Configurations for r_pcif

The following build time configurations are defined in fsp_cfg/r_pcif_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Timer Support	• Enabled • Disabled	Disabled	If select whether to use Timer module in the build.

Configurations for Motor > Phase-Counting Interface (r_pcif)

This module can be added to the Stacks tab via New Stack > Motor > Phase-Counting Interface (r_pcif).

Configuration	Options	Default	Description
General > Polarity > A-phase	• Positive • Negative	Positive	Select the polarity for A phase.
General > Polarity > B-phase	• Positive • Negative	Positive	Select the polarity for B phase.
General > Polarity > Z-phase	• Positive • Negative	Positive	Select the polarity for Z phase.
General > Name	Name must be a valid C symbol	g_pcif0	Module name.
General > Channel	Value must be an non-negative integer	0	Specify the PCIF channel.
General > Clock Source	MCU Specific Options		Select the clock source for the PCIF module.
General > Mode	• Phase Counting Mode • Incremental Mode	Phase Counting Mode	Select the clock source for the PCIF module.
General > Maximum Position	Manual Entry	0x10000	Specify The maximum value of the position counter.
Phase Counting Mode > ELC Support > Carrier Period Event Trigger Source	MCU Specific Options		Select external source that triggers the carrier period.
Phase Counting Mode > Initialize Edge Data > Start Position	Value must be a non-negative integer and less than max position	0	Set specify start counter position.
Phase Counting Mode > Initialize Edge Data > Edge Count Data	Value must be a signed 16-bit integer	0	Set the initial edge counter. This value must also be within +/- the generated carrier period count.
Phase Counting Mode > Error Detect > Error Carrier Period Detection	• Disable • Enable	Disable	Select carrier period error detection.
Phase Counting Mode > Error Detect > Error Edge Unprocessed Detection	• Disable • Enable	Disable	Select edge unprocessed error detection.
Phase Counting Mode > Error Detect > Error Edge Counting Overflow Detection	• Disable • Enable	Disable	Select edge counting overflow error detection.
Phase Counting Mode	• Disable	Disable	Select old edge error

> Error Detect > Error Old Edge Detection	• Enable		detection.
Phase Counting Mode > Carrier Period	Manual Entry	500	Specify the carrier period in microseconds selected.
Phase Counting Mode > Carrier Period Event Difference Threshold	Value must be a non-negative integer in the 16-bit range	0x100	Allowable difference range of carrier period events. This value must be less than the generated carrier period count.
Phase Counting Mode > Carrier Period Adjustment Mode	• Event Based Mode • Register Based Mode	Register Based Mode	Carrier period adjustment mode selection.
Phase Counting Mode > Z Phase Width	• Z-phase Width Disable • Z-phase Width 1 • Z-phase Width 2 • Z-phase Width 3 • Z-phase Width 4	Z-phase Width Disable	Select Z-phase width configuration.
Phase Counting Mode > Z Phase Sync	• Z-phase Sync A Phase • Z-phase Sync B Phase	Z-phase Sync A Phase	Select Z-phase synchronize configuration.
Phase Counting Mode > Edge Counter Clear	• Disable • Enable	Disable	Select automatic clear edge counter configuration.
Phase Counting Mode > Callback	Name must be a valid C symbol	NULL	A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the carrier period elapses
Incremental Mode > ELC Support > Crest Event Trigger Src	MCU Specific Options		Select external source that triggers the crest event.
Incremental Mode > ELC Support > Trough Event Trigger Src	MCU Specific Options		Select external source that triggers the trough event.
Incremental Mode >	Value must be from 1	1	Specify the downscale

Downscale Resolution Ratio	to 64		resolution ratio.
Incremental Mode > AB-Phase Input Mode	• Mode 0 (AB phase pulse) • Mode 1 (A phase pulse, B phase rotation direction) • Mode 2 (Pulse switching)	Mode 0 (AB phase pulse)	Incremental AB phase input mode selection
Incremental Mode > AB-Phase Output Mode	• Same as input • AB phase pulse	Same as input	Incremental AB phase output mode selection
Incremental Mode > Z-Phase Output Mode	• Immediate output • A-phase synchronization • B-phase synchronization • Pulse synchronization	Immediate output	Incremental Z phase output mode selection
Incremental Mode > Z-Phase Output Minimum Cycle	Value must be from 1 to 256	1	Specify the minimum output cycle.
Incremental Mode > Z Phase Generate Sources	• Counter-based • Z-Phase Input • Crest Event Trigger • Trough Event Trigger • Software Trigger		Select Z phase generate sources.
Incremental Mode > Clear Counter Sources	• Z-Phase Input • Crest Event Trigger • Trough Event Trigger • Software Trigger		Select counter clear sources.

Clock Configuration

PCIF uses CCLK as the core clock and CCLK_DIV as the output synchronization clock. CCLK can be selected from PCLKB or GPTCLK in the RX Configuration editor.

Configure the clocks so all HUM clock conditions are satisfied:

Clock	HUM range or limit
PCLKB	Maximum 75 MHz
CCLK	Maximum 200 MHz

CCLK_DIV	10 MHz to 20 MHz

The following relationships must also be satisfied:

- PCLKB >= 2 * CCLK_DIV
- PCLKB >= CCLK / 8

For example, the HUM maximum-frequency setting is PCLKB = 75 MHz, CCLK = 200 MHz, and CCLK_DIV = 20 MHz.

Pin Configuration

This module can use PCOUTAn, PCOUTBn, and PCOUTZn pins as output pins.

This module can use PCINAn, PCINBn, and PCINZn pins as input pins for encoder signals.

Usage Notes

General Operation

- Stop the PCIF before changing mode, polarity, position maximum, or other configuration registers that the HUM marks as rewrite-prohibited while SR.PCIFPS is 1.
- When using ELC events, start carrier period generation or crest/trough trigger generation only after the PCIF start flow has completed. HUM operation is not guaranteed if these events are input during start processing.
- Output inversion is configured by A/B/Z polarity settings and applies to both phase-counting and incremental modes.
- R_PCIF_StatusGet reports the PCIF operation state, rotation direction, and monitor values. In phase-counting mode, current_position_value is PHCPCMR.PHCPOSCNTM, position_write_value is PHCPCWDR.PHCPOSCNTWD, and input_counter_value is 0. In incremental mode, current_position_value is INCPCMR.INCOCNTM, input_counter_value is INCPCMR.INCICNTM, and position_write_value is 0.

Phase Counting Mode

Position And Edge Settings

- The number of edges per rotation can be set from 16 to 65,536 and must be a multiple of 4.
- The driver max_count_position field is the POSMAX register value. For an encoder with N edges per rotation, configure max_count_position = N - 1.
- The carrier period can be configured from 10 us to 3276 us. The generated carrier period count is expressed in CCLK_DIV cycles.
- The carrier period event difference threshold must be less than the carrier period count.
- The edge count written to PHCECR.PHCEDGCNT must be in the range -carrier_period_counts to +carrier_period_counts.

Carrier Period Control

There are two methods for controlling the carrier period:

- **Event-based mode**: Carrier period processing is triggered by carrier period event input from ELC.

- **Register-based mode**: Carrier period processing is generated from the configured PHCPERIODR.PHCPERIOD[15:0] value.

Set PHCPERIODR.PHCPERIOD[15:0] to the mean interval of carrier period events. If the long-term mean event interval differs from the register setting, edges may not be completely processed and may be lost during long-term operation.

Runtime Edge And Position Updates

R_PCIF_SetCounter writes the new edge count and can also write the new position value.

- When the PCIF is stopped, the new position value is reflected immediately.
- When the PCIF is running, the new position value is normally reflected at the next carrier period event.
- position_write_value from R_PCIF_StatusGet can be used to inspect the latest position write value.

Error Detection

Error detection is available only in phase-counting mode and must be enabled per error type in the configuration. If an error specified by PHCSR occurs, HUM operation is not guaranteed. Stop or reset the PCIF according to the phase-counting stop flow, clear the error state, and review the PCIF settings.

Incremental Mode

Input And Output Modes

PCIF supports the following incremental input/output combinations:

Input setting	Output setting	Input protocol mode	Output protocol mode
AB-phase pulse	Same as input	Mode 0	Mode 0
A-phase pulse, B-phase direction	Same as input	Mode 1	Mode 1
A-phase pulse, B-phase direction	AB-phase pulse	Mode 1	Mode 0
Pulse switching	Same as input	Mode 2	Mode 2
Pulse switching	AB-phase pulse	Mode 2	Mode 0

Note:

- **Mode 0** uses AB-phase pulses.
- **Mode 1** uses A-phase pulse and B-phase rotation direction.
- **Mode 2** uses pulse switching, where A-phase toggles for forward rotation and B-phase toggles for reverse rotation.

Resolution And Position Monitor

The incremental resolution setting is encoded as INCRSL = ratio - 1. A setting of 0 outputs the same

resolution as the input; a setting of 63 outputs 1/64 of the input resolution.

In incremental mode, the complete input position can be calculated from the monitor values returned by R_PCIF_StatusGet:

```
input_position = current_position_value * (resolution_ratio_setting + 1U) +
input_counter_value;
```

Z-Phase Generation And Counter Clear

The following HUM restrictions are enforced by the driver:

- Counter-based Z-phase generation and Z-phase input generation cannot be enabled at the same time.
- Clear counter by Z-phase input cannot be enabled when counter-based Z-phase generation is enabled.
- A/B synchronized Z-phase output is valid only when the output protocol is Mode 0.
- Pulse synchronized Z-phase output is valid only when the output protocol is Mode 1 or Mode 2.

z_output_min_cycles extends the Z-phase output assertion period. A configured value of 0 means 1 CCLK_DIV cycle, and 255 means 256 CCLK_DIV cycles. While the Z-phase output assertion period is being extended, other Z-phase output conditions are ignored until the extension completes.

Input Toggle Interval

The input ABZ signal must meet the HUM minimum toggle interval:

Input protocol condition	Minimum required toggle interval
Mode 0	1 CCLK cycle + 1 CCLK_DIV cycle
Mode 1	1 CCLK cycle + 1 CCLK_DIV cycle
Mode 2 with constant rotation direction	1 CCLK cycle + 1 CCLK_DIV cycle
Mode 2 when the rotation direction changes	1 CCLK cycle + 2 CCLK_DIV cycles

If the input signal does not satisfy these intervals, edges may be lost.

Limitations

- **Edge count max value setting in PHCECR**: According to the HUM, PHCECR.PHCEDGCNT[15:0] can represent values from -PHCPERIOD[15:0] to +PHCPERIOD[15:0]. The FSP API uses int16_t for edge count data, so values outside the signed 16-bit range cannot be passed through the API.

Examples

PCIF Basic Example

This is a basic example of minimal use of the PCIF in an application.

```
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the PCIF module */
    err = R_PCIF_Open(&g_pcif0_ctrl, &g_pcif0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize error flag and success flag to false.
  * This gets set to true in PCIF callback upon error occurs. */
    g_error_flag   = false;
    g_success_flag = false;
 /* Start PCIF Operation*/
    err = R_PCIF_Start(&g_pcif0_ctrl);
    assert(FSP_SUCCESS == err);
    uint32_t timeout_ms = PCIF_DELAY_MS;
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_success_flag) && (true != g_error_flag) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (0U == timeout_ms)
    {
       __BKPT(0);
    }
 /* Stop PCIF Operation*/
    err = R_PCIF_Stop(&g_pcif0_ctrl, PCIF_STOP_MODE_IMMEDIATE);
    assert(FSP_SUCCESS == err);
    err = R_PCIF_Close(&g_pcif0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

PCIF Callback

This is an example of PCIF callback.

```
/* Example callback called when pcif carrier period occurs. */
void pcif_callback (pcif_callback_args_t * p_args)
{
 if (PCIF_ERROR_NONE == p_args->error)
    {
 /* User can call R_PCIF_SetCounter to set edge data for next carrier period */
 R_PCIF_SetCounter(&g_pcif0_ctrl, &g_counter_data, true);
       g_success_flag = true;
    }
 else /* Error occur */
    {
       g_error_flag = true;
    }
}
```

PCIF Phase Counting Mode Example

This is an example of PCIF phase counting mode.

```
/* Example callback called when pcif carrier period occurs. */
void pcif_phase_counting_callback (pcif_callback_args_t * p_args)
{
 if (PCIF_ERROR_NONE == p_args->error)
    {
       g_success_flag = true;
 /* Do something with p_context if any */
    }
 else /* Error occur */
    {
       g_error_flag = true;
 /* Can check which type of error here to do something ...*/
    }
}
void pcif_phase_counting_example (void)
{
```

```
fsp_err_t err;
/* Initialize the PCIF module */
   err = R_PCIF_Open(&g_pcif0_ctrl, &g_pcif0_cfg);
   assert(FSP_SUCCESS == err);
/* Initialize error flag to false. This gets set to true in PCIF callback upon error
occurs. */
   g_error_flag   = false;
   g_success_flag = false;
/* Start PCIF Operation*/
   err = R_PCIF_Start(&g_pcif0_ctrl);
   assert(FSP_SUCCESS == err);
/* Set edge */
for (int i = 0; i < 5; i++)
   {
      err = R_PCIF_SetCounter(&g_pcif0_ctrl, &g_arr_counter_data[i], false);
     assert(FSP_SUCCESS == err);
     uint32_t timeout_ms = PCIF_DELAY_MS;
/* Since there is nothing else to do, block until Callback triggers*/
while ((true != g_success_flag) && (true != g_error_flag) && timeout_ms)
     {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;
     }
/* If timeout */
if (0U == timeout_ms)
     {
        __BKPT(0);
     }
else if (true == g_error_flag)
     {
break;
     }
else
     {
```

```
/* Do next loop*/
     g_success_flag = false;
     g_error_flag   = false;
     }
   }
/* If error occurs, PCIF will automatically stop operation */
if (true != g_error_flag)
   {
/* Stop PCIF Operation if no error occurs*/
     err = R_PCIF_Stop(&g_pcif0_ctrl, PCIF_STOP_MODE_IMMEDIATE);
     assert(FSP_SUCCESS == err);
   }
/* Deinitialize the PCIF module */
   err = R_PCIF_Close(&g_pcif0_ctrl);
   assert(FSP_SUCCESS == err);
}
```

PCIF Edge Data Update Example

This is an example of updating edge data in the PCIF main process.

```
if (PCIF_ERROR_NONE == p_args->error)
   {
/* User can call R_PCIF_SetCounter to set edge data for next carrier period */
R_PCIF_SetCounter(&g_pcif0_ctrl, &g_counter_data, true);
      g_success_flag = true;
   }
else /* Error occur */
   {
      g_error_flag = true;
   }
```

PCIF Basic Incremental Mode Example

This is an example of PCIF incremental mode.

```
void pcif_incremental_example (void)
{
 fsp_err_t err;
 /* Initialize the PCIF module */
    err = R_PCIF_Open(&g_pcif0_ctrl, &g_pcif0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start PCIF Operation*/
    err = R_PCIF_Start(&g_pcif0_ctrl);
    assert(FSP_SUCCESS == err);
 pcif_status_t pcif_status;
    err = R_PCIF_StatusGet(&g_pcif0_ctrl, &pcif_status);
    assert(FSP_SUCCESS == err);
 /* Do something with data in pcif_status.
  * In incremental mode, current_position_value is INCOCNTM and input_counter_value
is INCICNTM.
  * In phase-counting mode, current_position_value is PHCPOSCNTM and
position_write_value is PHCPCWDR. */
 /* Reset counter or Z phase generate */
    err = R_PCIF_SoftwareEventGenerate(&g_pcif0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Check if the counter is reset? or the Z phase is generated?*/
 /* Stop PCIF Operation */
 /* With incremental mode, stop mode always is PCIF_STOP_MODE_IMMEDIATE */
    err = R_PCIF_Stop(&g_pcif0_ctrl, PCIF_STOP_MODE_IMMEDIATE);
    assert(FSP_SUCCESS == err);
 /* Deinitialize the PCIF module */
    err = R_PCIF_Close(&g_pcif0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	pcif_extended_cfg_t
struct	pcif_instance_ctrl_t

Macros

#define	PCIF_INC_RESOLUTION_RATIO_SETTING_MAX
#define	PCIF_INC_Z_PHASE_GEN_COUNTER_INPUT_PROHIBITED_MASK
#define	PCIF_INC_COUNTER_CLEAR_Z_GEN_PROHIBITED_MASK

Enumerations

enum	pcif_phc_error_detection_enable_t
enum	pcif_phc_z_phase_width_t
enum	pcif_phc_z_phase_sync_t
enum	pcif_phc_carrier_period_mode_t
enum	pcif_phc_counter_clear_mode_t
enum	pcif_inc_ab_input_mode_t
enum	pcif_inc_ab_output_mode_t
enum	pcif_inc_z_output_mode_t
enum	pcif_inc_z_generate_source_t
enum	pcif_inc_position_counter_clear_t

Data Structure Documentation

◆ pcif_extended_cfg_t

struct pcif_extended_cfg_t		
Extended configuration structure for PCIF.		
Data Fields		
pcif_phc_z_phase_width_t	z_width	Set width Z phase. Setting for Phase Counting mode Setting Z phase
pcif_phc_z_phase_sync_t	z_sync	Set phase (A/B) to sync Z phase.
pcif_phc_carrier_period_mode_t	carrier_period_mode	Set carrier period mode. Carrier Period Setting
uint16_t	carrier_period_diff_threshold	Carrier period difference threshold in CCLK_DIV cycles.
uint16_t	carrier_period_counts	Carrier period in CCLK_DIV

		cycles.
pcif_phc_counter_clear_mode_t	counter_clear_mode	Set automatic clear counter mode. Edge Counter Setting
pcif_phc_counter_info_t *	p_init_counter_data	Set initial counter data.
pcif_phc_error_detection_enable_t	error_carrier_period_enable	Set carrier period error detection enable. Error detection enable setting
pcif_phc_error_detection_enable_t	error_edge_count_enable	Set edge count error detection enable.
pcif_phc_error_detection_enable_t	error_edge_count_overflow_enable	Set edge count overflow error detection enable.
pcif_phc_error_detection_enable_t	error_old_edge_count_enable	Set old edge count error detection enable.
uint8_t	resolution_ratio_setting	Output resolution: 0 = x1/1, N = x1/(N + 1), max 0x3F. Setting for Incremental mode
pcif_inc_ab_input_mode_t	ab_input_mode	Set incremental input mode. Setting A/B phase
pcif_inc_ab_output_mode_t	ab_output_mode	Set incremental output mode.
uint8_t	z_output_min_cycles	Minimum cycle Z output: 0 = 1 CCLK_DIV cycle, 0xFF = 256 cycles. Setting Z Phase
pcif_inc_z_generate_source_t	z_generate_source	Set Z phase generate source.
pcif_inc_z_output_mode_t	z_output_mode	Set incremental Z phase output mode.
pcif_inc_position_counter_clear_t	position_clear_source	Set incremental position counter clear mode.

◆ pcif_instance_ctrl_t

struct pcif_instance_ctrl_t
Control block used by driver. DO NOT INITIALIZE. Initialization occurs when pcif_api_t::open is called.

Macro Definition Documentation

◆ PCIF_INC_RESOLUTION_RATIO_SETTING_MAX

#define PCIF_INC_RESOLUTION_RATIO_SETTING_MAX
Maximum incremental mode resolution ratio setting (INCRSL[5:0]).

◆ PCIF_INC_Z_PHASE_GEN_COUNTER_INPUT_PROHIBITED_MASK

#define PCIF_INC_Z_PHASE_GEN_COUNTER_INPUT_PROHIBITED_MASK
Z phase generation sources that cannot be enabled together.

◆ PCIF_INC_COUNTER_CLEAR_Z_GEN_PROHIBITED_MASK

#define PCIF_INC_COUNTER_CLEAR_Z_GEN_PROHIBITED_MASK
Position clear source prohibited when counter-based Z phase generation is enabled.

Enumeration Type Documentation

◆ pcif_phc_error_detection_enable_t

enum pcif_phc_error_detection_enable_t	
Error detection enable setting.	
Enumerator	
PCIF_PHC_ERROR_DETECTION_DISABLE	error detection disabled
PCIF_PHC_ERROR_DETECTION_ENABLE	error detection enabled

◆ pcif_phc_z_phase_width_t

enum pcif_phc_z_phase_width_t	
Z phase width config	
Enumerator	
PCIF_PHC_Z_WIDTH_DISABLE	Z Width = 0, output Z phase disabled.
PCIF_PHC_Z_WIDTH_1	Z Width = 1.
PCIF_PHC_Z_WIDTH_2	Z Width = 2.
PCIF_PHC_Z_WIDTH_3	Z Width = 3.
PCIF_PHC_Z_WIDTH_4	Z Width = 4.

◆ pcif_phc_z_phase_sync_t

enum pcif_phc_z_phase_sync_t	
Phase Counting mode Z phase sync	
Enumerator	
PCIF_PHC_SYNC_PHASE_A	Z phase sync with A PHASE.
PCIF_PHC_SYNC_PHASE_B	Z phase sync with B PHASE.

◆ pcif_phc_carrier_period_mode_t

enum pcif_phc_carrier_period_mode_t	
Phase Counting mode carrier cycle event difference adjustment mode	
Enumerator	
PCIF_PHC_CARRIER_PERIOD_MODE_EVENT_BASED	Carrier period is generated by ELC.
PCIF_PHC_CARRIER_PERIOD_MODE_REGISTER_BASED	Carrier period is generated by internal counter.

◆ pcif_phc_counter_clear_mode_t

enum pcif_phc_counter_clear_mode_t	
Phase Counting mode automatic clear counter mode	
Enumerator	
PCIF_PHC_COUNTER_CLEAR_DISABLE	Disable automatic clear edge count register.
PCIF_PHC_COUNTER_CLEAR_ENABLE	Enable automatic clear edge count register.

◆ pcif_inc_ab_input_mode_t

enum pcif_inc_ab_input_mode_t	
Incremental mode ABZ phase input	
Enumerator	
PCIF_INC_INPUT_AB_PHASE_PULSE	Input mode 0: AB phase pulse.
PCIF_INC_INPUT_A_PHASE_PULSE	Input mode 1: A phase pulse (B phase rotation direction)
PCIF_INC_INPUT_PULSE_SWITCHING	Input mode 2: Pulse switching.

◆ pcif_inc_ab_output_mode_t

enum pcif_inc_ab_output_mode_t	
Incremental mode AB phase output	
Enumerator	
PCIF_INC_OUTPUT_SAME_AS_INPUT	Output same as input.
PCIF_INC_OUTPUT_AB_PHASE_PULSE	Output AB phase pulse.

◆ pcif_inc_z_output_mode_t

enum pcif_inc_z_output_mode_t	
Incremental mode Z phase output	
Enumerator	
PCIF_INC_Z_OUTPUT_IMMEDIATE	Immediate output.
PCIF_INC_Z_OUTPUT_A_SYNC	A-phase synchronization.
PCIF_INC_Z_OUTPUT_B_SYNC	B-phase synchronization.
PCIF_INC_Z_OUTPUT_PULSE_SYNC	Pulse synchronization.

◆ pcif_inc_z_generate_source_t

enum pcif_inc_z_generate_source_t	
Incremental mode Z phase generate source	
Enumerator	
PCIF_INC_Z_PHASE_GEN_NONE	No generation of Z phase.
PCIF_INC_Z_PHASE_GEN_COUNTER	Generate Z phase from output/internal counter.
PCIF_INC_Z_PHASE_GEN_INPUT	Generate output Z phase from external input signal.
PCIF_INC_Z_PHASE_GEN_CREST	Generate output Z phase from crest event signal.
PCIF_INC_Z_PHASE_GEN_TROUGH	Generate output Z phase from trough event signal.
PCIF_INC_Z_PHASE_GEN_SOFTWARE	Generate output Z phase from software trigger.

◆ pcif_inc_position_counter_clear_t

enum pcif_inc_position_counter_clear_t	
Incremental position counter clear enable bit	
Enumerator	
PCIF_INC_COUNTER_CLEAR_NONE	Disable position counter clear.
PCIF_INC_COUNTER_CLEAR_INPUT	Clear position counter by Z input signal.
PCIF_INC_COUNTER_CLEAR_CREST	Clear position counter by crest event signal.
PCIF_INC_COUNTER_CLEAR_TROUGH	Clear position counter by trough event signal.
PCIF_INC_COUNTER_CLEAR_SOFTWARE	Clear position counter by software trigger.

Function Documentation

◆ R_PCIF_Open()

fsp_err_t R_PCIF_Open (pcif_ctrl_t *const *p_ctrl*, pcif_cfg_t const *const *p_cfg*)

Initializes the pcif module and applies configurations. Implements pcif_api_t::open.

This function sets this clock divisor and the configurations specified in pcif_cfg_t.

Example:

```
/* Initialize the PCIF module */
   err = R_PCIF_Open(&g_pcif0_ctrl, &g_pcif0_cfg);
```

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. Pointer p_api_ctrl or p_cfg is NULL. 2. Extended parameter is NULL. 3. Timer instance in p_cfg is NULL when timer support enabled 4. Callback parameter is NULL when ELC support enabled.
FSP_ERR_INVALID_DATA	Parameter check failure due to one or more reasons below: 1. Max count position does not meet "4 * N – 1" requirement. 2. Value diff threshold is greater than or equal to value carrier period. 3. Initial position is greater than max count position. 4. Initial edge count is outside the carrier period range. 5. Incremental resolution ratio is outside the hardware bit-field range. 6. Prohibited incremental Z generation sources are enabled together. 7. Prohibited incremental position clear source is enabled.

◆ R_PCIF_Stop()

fsp_err_t R_PCIF_Stop (pcif_ctrl_t *const *p_ctrl*, pcif_stop_mode_t *stop_mode*)

Stop PCIF. Implements pcif_api_t::stop.

Example:

```
/* Stop PCIF Operation*/
   err = R_PCIF_Stop(&g_pcif0_ctrl, PCIF_STOP_MODE_IMMEDIATE);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Module successfully stops.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_PCIF_Start()

fsp_err_t R_PCIF_Start (pcif_ctrl_t *const *p_ctrl*)

Starts PCIF. Implements pcif_api_t::start.

Example:

```
/* Start PCIF Operation*/
   err = R_PCIF_Start(&g_pcif0_ctrl);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Timer successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_IN_USE	Channel is running.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_PCIF_SetCounter()

fsp_err_t R_PCIF_SetCounter (pcif_ctrl_t *const *p_ctrl*, pcif_phc_counter_info_t const *const *p_counter_data*, bool *reset_counter*)

Set counter data in phase counting mode. Implements pcif_api_t::setCounter. Only used in phase counting mode. This function returns the PCIF information.

Example:

```
/* Set edge */
for (int i = 0; i < 5; i++)
   {
      err = R_PCIF_SetCounter(&g_pcif0_ctrl, &g_arr_counter_data[i], false);
     assert(FSP_SUCCESS == err);
     uint32_t timeout_ms = PCIF_DELAY_MS;
/* Since there is nothing else to do, block until Callback triggers*/
while ((true != g_success_flag) && (true != g_error_flag) && timeout_ms)
     {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;
     }
/* If timeout */
if (0U == timeout_ms)
     {
        __BKPT(0);
     }
else if (true == g_error_flag)
     {
break;
     }
else
     {
/* Do next loop*/
     g_success_flag = false;
     g_error_flag   = false;
     }
   }
```

Return values

FSP_SUCCESS	Information successfully retrieved.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_MODE	The mode is not phase counting.
FSP_ERR_INVALID_DATA	The new position or edge count is invalid.

◆ R_PCIF_StatusGet()

fsp_err_t R_PCIF_StatusGet (pcif_ctrl_t *const *p_ctrl*, pcif_status_t *const *p_status*)

Get PCIF status. Implements pcif_api_t::statusGet.

This function returns the PCIF status.

Example:

```
pcif_status_t pcif_status;
   err = R_PCIF_StatusGet(&g_pcif0_ctrl, &pcif_status);
   assert(FSP_SUCCESS == err);
/* Do something with data in pcif_status.
 * In incremental mode, current_position_value is INCOCNTM and input_counter_value
is INCICNTM.
 * In phase-counting mode, current_position_value is PHCPOSCNTM and
position_write_value is PHCPCWDR. */
```

Return values

FSP_SUCCESS	Status successfully retrieved.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_PCIF_SoftwareEventGenerate()

fsp_err_t R_PCIF_SoftwareEventGenerate (pcif_ctrl_t *const *p_ctrl*)

Software reset PCIF. Implements pcif_api_t::softwareEventGenerate.

This function generates a software event to reset the PCIF.

Example:

```
/* Reset counter or Z phase generate */
   err = R_PCIF_SoftwareEventGenerate(&g_pcif0_ctrl);
   assert(FSP_SUCCESS == err);
/* Check if the counter is reset? or the Z phase is generated?*/
```

Return values

FSP_SUCCESS	Software event successfully generated.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_MODE	The mode is not incremental.

◆ R_PCIF_Close()

fsp_err_t R_PCIF_Close (pcif_ctrl_t *const *p_ctrl*)

Close PCIF. Implements pcif_api_t::close.

This function powers down the PCIF and closes the lower level timer drivers if they are used.

Example:

```
   err = R_PCIF_Close(&g_pcif0_ctrl);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Timer successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

5.2.11.2 Motor Current Controller (rm_motor_current)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_CURRENT_Open (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) Opens and configures the Motor Current Module. Implements motor_current_api_t::open. More...
fsp_err_t	RM_MOTOR_CURRENT_Close (motor_current_ctrl_t *const p_ctrl) Disables specified Motor Current Module. Implements motor_current_api_t::close. More...
fsp_err_t	RM_MOTOR_CURRENT_Reset (motor_current_ctrl_t *const p_ctrl) Reset variables of Motor Current Module. Implements motor_current_api_t::reset. More...
fsp_err_t	RM_MOTOR_CURRENT_Run (motor_current_ctrl_t *const p_ctrl) Run(Start) the Current Control. Implements motor_current_api_t::run. More...
fsp_err_t	RM_MOTOR_CURRENT_ParameterSet (motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input) Set (Input) Parameter Data. Implements motor_current_api_t::parameterSet. More...
fsp_err_t	RM_MOTOR_CURRENT_CurrentReferenceSet (motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference) Set Current Reference Data. Implements motor_current_api_t::currentReferenceSet. More...
fsp_err_t	RM_MOTOR_CURRENT_SpeedPhaseSet (motor_current_ctrl_t *const p_ctrl, float const speed, float const phase) Set Current Speed & rotor phase Data. Implements motor_current_api_t::speedPhaseSet. More...
fsp_err_t	RM_MOTOR_CURRENT_CurrentSet (motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage) Set d/q-axis Current & Voltage Data. Implements motor_current_api_t::currentSet. More...

fsp_err_t	RM_MOTOR_CURRENT_ParameterGet (motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output) Get Output Parameters. Implements motor_current_api_t::parameterGet. More...
fsp_err_t	RM_MOTOR_CURRENT_CurrentGet (motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq) Get d/q-axis Current. Implements motor_current_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_CURRENT_PhaseVoltageGet (motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage) Gets the set phase voltage. Implements motor_current_api_t::phaseVoltageGet. More...
fsp_err_t	RM_MOTOR_CURRENT_ParameterUpdate (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) Update the parameters of Current Control. Implements motor_current_api_t::parameterUpdate. More...
void	rm_motor_current_encoder_cyclic (motor_current_instance_t const *p_ctrl)

Detailed Description

Calculation process for the motor control on RX MCUs. This module implements the Motor current Interface.

Overview

The motor current is used to control the electric current of motor rotation in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

Features

The Motor Current Module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling Control.
- Voltage Error Compensation.

Configuration

Build Time Configurations for rm_motor_current

The following build time configurations are defined in fsp_cfg/rm_motor_current_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Current Controller (rm_motor_current)

This module can be added to the Stacks tab via New Stack > Motor > Motor Current Controller (rm_motor_current).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_current0	Module name.
General > Sensor type	• Sensorless • Encoder • Induction • Hall	Sensorless	Select sensor type
General > Shunt type	• 1 shunt • 2 shunt • 3 shunt	2 shunt	Select shunt type
General > Current control decimation	Must be a valid non-negative value.	0	Decimation of current control.
General > PWM carrier frequency (kHz)	Must be a valid value	20.0F	PWM carrier frequency.
General > Input voltage (V)	Must be a valid value	24.0F	Input voltage for limitation of current PI control.
General > Sample delay compensation	• Disable • Enable	Enable	Select enable/disable sample delay compensation.
General > Period magnification value	Must be a valid value	1.5	Period magnification value for sampling delay compensation.
General > Voltage error compensation	• Disable • Enable	Enable	Select enable/disable voltage error compensation.
General > Voltage error compensation table of voltage 1	Must be a valid value	0.672F	Voltage error compensation table of voltage.
General > Voltage error compensation table of voltage 2	Must be a valid value	0.945F	Voltage error compensation table of voltage.

General > Voltage error compensation table of voltage 3	Must be a valid value	1.054F	Voltage error compensation table of voltage.
General > Voltage error compensation table of voltage 4	Must be a valid value	1.109F	Voltage error compensation table of voltage.
General > Voltage error compensation table of voltage 5	Must be a valid value	1.192F	Voltage error compensation table of voltage.
General > Voltage error compensation table of current 1	Must be a valid value	0.013F	Voltage error compensation table of current.
General > Voltage error compensation table of current 2	Must be a valid value	0.049F	Voltage error compensation table of current.
General > Voltage error compensation table of current 3	Must be a valid value	0.080F	Voltage error compensation table of current.
General > Voltage error compensation table of current 4	Must be a valid value	0.184F	Voltage error compensation table of current.
General > Voltage error compensation table of current 5	Must be a valid value	0.751F	Voltage error compensation table of current.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt.
Design Parameter > Current PI loop omega (Hz)	Must be a valid value	300.0F	Current PI loop omega
Design Parameter > Current PI loop zeta	Must be a valid value	1.0F	Current PI loop zeta
Motor Parameter > Pole pairs	Must be a valid non-negative value.	2	Pole pairs
Motor Parameter > Resistance (ohm)	Must be a valid value	8.5F	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid value	0.0045F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid value	0.0045F	Inductance of q-axis

Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid value	0.02159F	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid value	0.0000028F	Rotor inertia

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the Period of Current Control with none-negative value.
- Set the Reference Voltage with none-negative value.

Examples

Basic Example

This is a basic example of minimal use of the Motor Current in an application.

```
void motor_current_basic_example (void)
{
 motor_current_input_current_t temp_input_current;
 motor_current_input_voltage_t temp_input_voltage;
 motor_current_get_voltage_t   temp_get_voltage;
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_CURRENT_Open(g_test_motor_current.p_ctrl,
g_test_motor_current.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Set current reference before get phase voltage */
```

```
      (void) RM_MOTOR_CURRENT_CurrentReferenceSet(g_test_motor_current.p_ctrl, 1.0F,
1.0F);
 /* Set speed and phase data before get phase voltage */
      (void) RM_MOTOR_CURRENT_SpeedPhaseSet(g_test_motor_current.p_ctrl, 104.72F,
1.0F);
      temp_input_current.iu    = 1.0F;
      temp_input_current.iv    = 1.0F;
      temp_input_current.iw    = 1.0F;
      temp_input_voltage.vdc    = 24.0F;
      temp_input_voltage.va_max = 24.0F;
 /* Set electric current and voltage before get phase voltage */
      (void) RM_MOTOR_CURRENT_CurrentSet(g_test_motor_current.p_ctrl,
temp_input_current, temp_input_voltage);
 /* Activate the process. */
      (void) RM_MOTOR_CURRENT_Run(g_test_motor_current.p_ctrl);
 /* Get d/q-axis current*/
      (void) RM_MOTOR_CURRENT_CurrentGet(g_test_motor_current.p_ctrl, &f_get_id,
&f_get_iq);
 /* Get the flag of PI control */
      (void) RM_MOTOR_CURRENT_PhaseVolageGet(g_test_motor_current.p_ctrl,
&temp_get_voltage);
 /* Get Output Parameter */
      (void) RM_MOTOR_CURRENT_ParameterGet(g_test_motor_current.p_ctrl,
&test_output);
      (void) RM_MOTOR_CURRENT_ParameterUpdate(g_test_motor_current.p_ctrl,
g_test_motor_current.p_cfg);
    }
 /* Reset the process. */
    (void) RM_MOTOR_CURRENT_Reset(g_test_motor_current.p_ctrl);
 /* Close the module. */
    (void) RM_MOTOR_CURRENT_Close(g_test_motor_current.p_ctrl);
}
```

Enumerations

enum	motor_current_shunt_type_t

Enumeration Type Documentation

◆ motor_current_shunt_type_t

enum motor_current_shunt_type_t	
Selection of shunt type	
Enumerator	
MOTOR_CURRENT_SHUNT_TYPE_1_SHUNT	Only use U phase current.
MOTOR_CURRENT_SHUNT_TYPE_2_SHUNT	Use U and W phase current.
MOTOR_CURRENT_SHUNT_TYPE_3_SHUNT	Use all phase current.

Function Documentation

◆ RM_MOTOR_CURRENT_Open()

fsp_err_t RM_MOTOR_CURRENT_Open (motor_current_ctrl_t *const *p_ctrl*, motor_current_cfg_t const *const *p_cfg*)

Opens and configures the Motor Current Module. Implements motor_current_api_t::open.

Return values

FSP_SUCCESS	Motor Current successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_CURRENT_Close()

fsp_err_t RM_MOTOR_CURRENT_Close (motor_current_ctrl_t *const *p_ctrl*)

Disables specified Motor Current Module. Implements motor_current_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_Reset()

fsp_err_t RM_MOTOR_CURRENT_Reset (motor_current_ctrl_t *const *p_ctrl*)

Reset variables of Motor Current Module. Implements motor_current_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_Run()

fsp_err_t RM_MOTOR_CURRENT_Run (motor_current_ctrl_t *const *p_ctrl*)

Run(Start) the Current Control. Implements motor_current_api_t::run.

Return values

FSP_SUCCESS	Successfully run.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_ParameterSet()

fsp_err_t RM_MOTOR_CURRENT_ParameterSet (motor_current_ctrl_t *const *p_ctrl*, motor_current_input_t const *const *p_st_input*)

Set (Input) Parameter Data. Implements motor_current_api_t::parameterSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input argument error.

◆ RM_MOTOR_CURRENT_CurrentReferenceSet()

fsp_err_t RM_MOTOR_CURRENT_CurrentReferenceSet (motor_current_ctrl_t *const *p_ctrl*, float const *id_reference*, float const *iq_reference*)

Set Current Reference Data. Implements motor_current_api_t::currentReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_SpeedPhaseSet()

fsp_err_t RM_MOTOR_CURRENT_SpeedPhaseSet (motor_current_ctrl_t *const *p_ctrl*, float const *speed*, float const *phase*)

Set Current Speed & rotor phase Data. Implements motor_current_api_t::speedPhaseSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_CurrentSet()

fsp_err_t RM_MOTOR_CURRENT_CurrentSet (motor_current_ctrl_t *const *p_ctrl*, motor_current_input_current_t const *const *p_st_current*, motor_current_input_voltage_t const *const *p_st_voltage*)

Set d/q-axis Current & Voltage Data. Implements motor_current_api_t::currentSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_ParameterGet()

fsp_err_t RM_MOTOR_CURRENT_ParameterGet (motor_current_ctrl_t *const *p_ctrl*, motor_current_output_t *const *p_st_output*)

Get Output Parameters. Implements motor_current_api_t::parameterGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_CurrentGet()

fsp_err_t RM_MOTOR_CURRENT_CurrentGet (motor_current_ctrl_t *const *p_ctrl*, float *const *p_id*, float *const *p_iq*)

Get d/q-axis Current. Implements motor_current_api_t::currentGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_PhaseVoltageGet()

fsp_err_t RM_MOTOR_CURRENT_PhaseVoltageGet (motor_current_ctrl_t *const *p_ctrl*, motor_current_get_voltage_t *const *p_voltage*)

Gets the set phase voltage. Implements motor_current_api_t::phaseVoltageGet.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_ParameterUpdate()

fsp_err_t RM_MOTOR_CURRENT_ParameterUpdate (motor_current_ctrl_t *const *p_ctrl*, motor_current_cfg_t const *const *p_cfg*)

Update the parameters of Current Control. Implements motor_current_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ rm_motor_current_encoder_cyclic()

void rm_motor_current_encoder_cyclic (motor_current_instance_t const * *p_ctrl*)

(end addtogroup MOTOR_CURRENT)

5.2.11.3 ADC and PWM Modulation (rm_motor_driver)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_DRIVER_Open (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) Opens and configures the Motor Driver module. Implements motor_driver_api_t::open. More...
fsp_err_t	RM_MOTOR_DRIVER_Close (motor_driver_ctrl_t *const p_ctrl) Disables specified Motor Driver Module. Implements motor_driver_api_t::close. More...
fsp_err_t	RM_MOTOR_DRIVER_Reset (motor_driver_ctrl_t *const p_ctrl) Reset variables of Motor Driver Module. Implements motor_driver_api_t::reset. More...
fsp_err_t	RM_MOTOR_DRIVER_PhaseVoltageSet (motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage) Set Phase Voltage Data to calculate PWM duty. Implements

	motor_driver_api_t::phaseVoltageSet. More...
fsp_err_t	RM_MOTOR_DRIVER_CurrentGet (motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get) Get calculated phase Current, Vdc & Va_max data. Implements motor_driver_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_DRIVER_FlagCurrentOffsetGet (motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset) Get the flag of finish current offset detection. Implements motor_driver_api_t::flagCurrentOffsetGet. More...
fsp_err_t	RM_MOTOR_DRIVER_CurrentOffsetRestart (motor_driver_ctrl_t *const p_ctrl) Restart the current offset detection. Implements motor_driver_api_t::currentOffsetRestart. More...
fsp_err_t	RM_MOTOR_DRIVER_ParameterUpdate (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) Update the parameters of Driver Module. Implements motor_driver_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RX MCUs. This module implements the Motor driver Interface.

Overview

The motor driver module is used to translate phase voltage to PWM duty and output PWM, and detect phase current and main line voltage. This module should be called cyclically at included A/D Conversion finish interrupt.

Features

The Motor Driver Module has below features.

- Calculate each phase(U/V/W) PWM duty according to reference and output PWM.
- Detect each phase current and main line voltage.
- Detect and correct A/D offset at phase current channel

Configuration

Build Time Configurations for rm_motor_driver

The following build time configurations are defined in fsp_cfg/rm_motor_driver_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
ADC_B Support	• Enabled • Disabled	Disabled	Select ADC_B module support.
Shared ADC support	• Enabled • Disabled	Disabled	Select Shared ADC support.
Supported Motor Number	Must be greater than 1.	1	

Configurations for Motor > ADC and PWM Modulation (rm_motor_driver)

This module can be added to the Stacks tab via New Stack > Motor > ADC and PWM Modulation (rm_motor_driver).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_driver0	Module name.
General > Shunt type	• 1 shunt • 2 shunt • 3 shunt	2 shunt	Select shunt type
General > Modulation method	• SVPWM • SPWM	SVPWM	Select PWM modulation method
General > PWM output port UP	Manual Entry	0	PWM output port UP
General > PWM output port UN	Manual Entry	0	PWM output port UN
General > PWM output port VP	Manual Entry	0	PWM output port VP
General > PWM output port VN	Manual Entry	0	PWM output port VN
General > PWM output port WP	Manual Entry	0	PWM output port WP
General > PWM output port WN	Manual Entry	0	PWM output port WN
General > PWM Timer Frequency (MHz)	Must be a valid integer.	120	GPT PWM timer frequency
General > PWM Carrier Period (Microseconds)	Must be a valid integer.	50	GPT PWM carrier period
General > Dead Time (Raw Counts)	Must be a valid integer.	240	GPT PWM dead time

General > Current Range (A)	Must be a valid value	27.5F	Current range to measure(Maximum input current)
General > Voltage Range (V)	Must be a valid value	111.0F	Voltage range to measure(Maximum input Main Line Voltage)
General > Counts for current offset measurement	Must be a valid integer.	500	How many times to measure current offset
General > A/D conversion channel for U Phase current	Value must be a supported channel number	0	Specify the A/D channel for U phase current (channel availability varies by MCU)
General > A/D conversion channel for W Phase current	Value must be a supported channel number	2	Specify the A/D channel for W phase current (channel availability varies by MCU)
General > A/D conversion channel for Main Line Voltage	Value must be a supported channel number	5	Specify the A/D channel for main line voltage (channel availability varies by MCU)
General > A/D conversion channel for V Phase current	Value must be a supported channel number	1	Specify the A/D channel for V phase current (channel availability varies by MCU)
General > A/D conversion channel for sin signal	Value must be a supported channel number	27	Specify the A/D channel for sin signal of induction sensor (channel availability varies by MCU)
General > A/D conversion channel for cos signal	Value must be a supported channel number	28	Specify the A/D channel for cos signal of induction sensor (channel availability varies by MCU)
General > Using ADC Scan Group	Value should be between 0 to 8 (valid only with ADC_B module)	0	For MCUs with ADC_B, select the scan group used.
General > A/D conversion unit for U Phase current	Must be a valid non-negative value.	0	Select the A/D conversion module for U phase current (only valid with adc module)

General > A/D conversion unit for W Phase current	Must be a valid non-negative value.	0	Select the A/D conversion module for W phase current (only valid with adc module)
General > A/D conversion unit for main line voltage	Must be a valid non-negative value.	0	Select the A/D conversion module for main line voltage (only valid with adc module)
General > A/D conversion unit for V Phase current	Must be a valid non-negative value.	0	Select the A/D conversion module for V phase current (only valid with adc module)
General > A/D conversion unit for sin signal	Must be a valid non-negative value.	0	Select the A/D conversion module for sin signal of induction sensor (only valid with adc module)
General > A/D conversion unit for cos signal	Must be a valid non-negative value.	0	Select the A/D conversion module for cos signal of induction sensor (only valid with adc module)
General > ADC interrupt module	• 1st • 2nd	1st	Select from which module ADC intterupt happens (only valid with adc module)
General > Adjustment value to current A/D	Must be a valid non-negative value.	20.0	Value to adjust 1shunt A/D double buffer
General > Minimum difference of PWM duty	Must be a valid non-negative value.	300	Minimum difference of PWM duty
General > Adjustment delay of A/D conversion	Must be a valid non-negative value.	240	Adjustment delay of A/D conversion
General > 1shunt interrupt phase	• PHASE_U • PHASE_V • PHASE_W	PHASE_U	Select the phase which occures ADC end interrupt. (Only valid with 1shunt)
General > Input Voltage (V)	Must be a valid value	24.0F	Input voltage
General > Resolution of A/D conversion	Must be a valid Resolution of ADC.	0xFFF	Resolution of A/D conversion
General > Offset of A/D conversion for current	Must be a valid non-negative value.	0x745	Offset of A/D conversion for current
General > Conversion level of A/D conversion for voltage	Must be a valid value	0.66F	Conversion level of A/D conversion for voltage
General > GTIOCA Stop	• Pin Level Low	Pin Level High	Select the behavior of

Level	• Pin Level High		the output pin when the timer is stopped.
General > GTIOCB Stop Level	• Pin Level Low • Pin Level High	Pin Level High	Select the behavior of the output pin when the timer is stopped.
Modulation > Maximum Duty	Must be a valid value	0.9375F	Maximum duty of PWM
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt.

Clock Configuration

Set used clock with included GPT timer.

Pin Configuration

Depend on included GPT Three Phase Module and ADC Module.

Usage Notes

When shared ADC instance is used, please perform below sequence.

1. Add (New) shared ADC instance.
2. Set configurations of below ADC module for your use.
3. Select "Common | Shared ADC support" to "Enabled".

Limitations

Basically no limitation exists.

Examples

Basic Example

This is a basic example of minimal use of the Motor Driver in an application.

```
void motor_driver_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_DRIVER_Open(&g_motor_driver0.p_ctrl, &g_motor_driver0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at cyclic interrupt (e.g. included GPT PWM Carrier
```

```
intterupt).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Get electric current, main line voltage and maximum voltage component */
       (void) RM_MOTOR_DRIVER_CurrentGet(&g_motor_driver0.p_ctrl, &f_get_iu,
&f_get_iw, &f_get_vdc, &f_get_va_max);
 /* Get the flag of A/D convderted current offset */
       (void) RM_MOTOR_DRIVER_FlagCurrentOffsetGet(&g_motor_driver0.p_ctrl,
&u1_get_flg_offset);
 // Perform current control process here
 /* Set phase voltage */
       (void) RM_MOTOR_DRIVER_PhaseVoltageSet(&g_motor_driver0.p_ctrl, 1.0F, 1.0F,
1.0F);
       (void) RM_MOTOR_DRIVER_ParameterUpdate(&g_motor_driver0.p_ctrl,
&g_motor_driver0.p_cfg);
    }
    (void) RM_MOTOR_DRIVER_Reset(&g_motor_driver0.p_ctrl);
 //

    (void) RM_MOTOR_DRIVER_Close(&g_motor_driver0.p_ctrl);
}
```

Data Structures

struct	motor_driver_shared_instance_ctrl_t
struct	motor_driver_extended_shared_cfg_t

Enumerations

enum	motor_driver_select_adc_instance_t
enum	motor_driver_modulation_method_t

Data Structure Documentation

◆ motor_driver_shared_instance_ctrl_t

struct motor_driver_shared_instance_ctrl_t		
For multiple motor		
Data Fields		
uint32_t	open	
uint8_t	registered_motor_count	Registered motor counts.
void *	p_context[MOTOR_DRIVER_CFG_SUPPORT_MOTOR_NUM]	

◆ motor_driver_extended_shared_cfg_t

struct motor_driver_extended_shared_cfg_t		
For multiple motor		
Data Fields		
adc_instance_t const *	p_adc_instance_first	first ADC instance
adc_instance_t const *	p_adc_instance_second	second ADC instance
motor_driver_shared_instance_ctrl_t *const	p_shared_instance_ctrl	

Enumeration Type Documentation

◆ motor_driver_select_adc_instance_t

enum motor_driver_select_adc_instance_t	
Support two ADC instance valid for adc	
Enumerator	
MOTOR_DRIVER_SELECT_ADC_INSTANCE_FIRST	Use first ADC instance.
MOTOR_DRIVER_SELECT_ADC_INSTANCE_SECOND	Use second ADC instance.

◆ motor_driver_modulation_method_t

enum motor_driver_modulation_method_t	
Enumerator	
MOTOR_DRIVER_MODULATION_METHOD_SPWM	Sinusoidal pulse-width-modulation.
MOTOR_DRIVER_MODULATION_METHOD_SVPWM	Space vector pulse-width-modulation.

Function Documentation

◆ RM_MOTOR_DRIVER_Open()

fsp_err_t RM_MOTOR_DRIVER_Open (motor_driver_ctrl_t *const *p_ctrl*, motor_driver_cfg_t const *const *p_cfg*)

Opens and configures the Motor Driver module. Implements motor_driver_api_t::open.

Return values

FSP_SUCCESS	Motor Driver successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

◆ RM_MOTOR_DRIVER_Close()

fsp_err_t RM_MOTOR_DRIVER_Close (motor_driver_ctrl_t *const *p_ctrl*)

Disables specified Motor Driver Module. Implements motor_driver_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_Reset()

fsp_err_t RM_MOTOR_DRIVER_Reset (motor_driver_ctrl_t *const *p_ctrl*)

Reset variables of Motor Driver Module. Implements motor_driver_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_PhaseVoltageSet()

fsp_err_t RM_MOTOR_DRIVER_PhaseVoltageSet (motor_driver_ctrl_t *const *p_ctrl*, float const *u_voltage*, float const *v_voltage*, float const *w_voltage*)

Set Phase Voltage Data to calculate PWM duty. Implements motor_driver_api_t::phaseVoltageSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_CurrentGet()

fsp_err_t RM_MOTOR_DRIVER_CurrentGet (motor_driver_ctrl_t *const *p_ctrl*, motor_driver_current_get_t *const *p_current_get*)

Get calculated phase Current, Vdc & Va_max data. Implements motor_driver_api_t::currentGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_DRIVER_FlagCurrentOffsetGet()

fsp_err_t RM_MOTOR_DRIVER_FlagCurrentOffsetGet (motor_driver_ctrl_t *const *p_ctrl*, uint8_t *const *p_flag_offset*)

Get the flag of finish current offset detection. Implements motor_driver_api_t::flagCurrentOffsetGet .

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_DRIVER_CurrentOffsetRestart()

fsp_err_t RM_MOTOR_DRIVER_CurrentOffsetRestart (motor_driver_ctrl_t *const *p_ctrl*)

Restart the current offset detection. Implements motor_driver_api_t::currentOffsetRestart.

Return values

FSP_SUCCESS	Successfully restarted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_ParameterUpdate()

fsp_err_t RM_MOTOR_DRIVER_ParameterUpdate (motor_driver_ctrl_t *const *p_ctrl*, motor_driver_cfg_t const *const *p_cfg*)

Update the parameters of Driver Module. Implements motor_driver_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

5.2.11.4 Motor Encoder Vector Control (rm_motor_encoder)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_ENCODER_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)
fsp_err_t	RM_MOTOR_ENCODER_Close (motor_ctrl_t *const p_ctrl) Disables specified Motor Encoder Control block. Implements motor_api_t::close. More...
fsp_err_t	RM_MOTOR_ENCODER_Run (motor_ctrl_t *const p_ctrl) Run Motor (Start motor rotation). Implements motor_api_t::run. More...

fsp_err_t	RM_MOTOR_ENCODER_Stop (motor_ctrl_t *const p_ctrl) Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More...
fsp_err_t	RM_MOTOR_ENCODER_Reset (motor_ctrl_t *const p_ctrl) Reset Motor Encoder Control block. Implements motor_api_t::reset. More...
fsp_err_t	RM_MOTOR_ENCODER_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error) Set error information. Implements motor_api_t::errorSet. More...
fsp_err_t	RM_MOTOR_ENCODER_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm) Set speed reference[rpm]. Implements motor_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_ENCODER_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position) Set position reference[degree]. Implements motor_api_t::positionSet. More...
fsp_err_t	RM_MOTOR_ENCODER_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status) Get current control status. Implements motor_api_t::statusGet. More...
fsp_err_t	RM_MOTOR_ENCODER_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad) Get current rotor angle. Implements motor_api_t::angleGet. More...
fsp_err_t	RM_MOTOR_ENCODER_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get rotational speed. Implements motor_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_ENCODER_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error) Check the occurunce of Error. Implements motor_api_t::errorCheck. More...

fsp_err_t	RM_MOTOR_ENCODER_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag) Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More...
fsp_err_t	RM_MOTOR_ENCODER_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function) Select using function. Implements motor_api_t::functionSelect. More...
fsp_err_t	RM_MOTOR_ENCODER_InertiaEstimateStart (motor_ctrl_t *const p_ctrl) Start inertia estimation function. More...
fsp_err_t	RM_MOTOR_ENCODER_InertiaEstimateStop (motor_ctrl_t *const p_ctrl) Stop(Cancel) inertia estimation function. More...
fsp_err_t	RM_MOTOR_ENCODER_ReturnOriginStart (motor_ctrl_t *const p_ctrl) Start return origin function. More...
fsp_err_t	RM_MOTOR_ENCODER_ReturnOriginStop (motor_ctrl_t *const p_ctrl) Stop(Cancel) return origin function. More...

Detailed Description

Control a SPM motor on RX MCUs. This module implements the Motor Encoder Vector Control (rm_motor_encoder).

Overview

The motor encoder vector control is used to control motor rotation in an application. This module is meant to be used with Surface Permanent Magnet (SPM) motors and allows applications to start or stop motor rotation easily.

Features

The motor encoder module has below features.

- Start/stop motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

Target Hardware

The below figure shows an example of target hardware of this Motor Encoder Module.

Block Diagram

The below figure shows block diaram of encoder vector motor control.

Modulation

- Sine wave modulation The modulation factor "m" is defined as follows.
- Space vector modulation In vector control of a permanent magnet synchronous motor, generally, the desired voltage command value of each phase is generated sinusoidally. However, if the generated value is used as-is for the modulation wave for PWM generation, voltage utilization as applied to the motor (in terms of line voltage) is limited to a maximum of 86.7 percents with respect to inverter bus voltage. As such, as shown in the following expression, the average of the maximum and minimum values is calculated for the voltage command value of each phase, and the value obtained by subtracting the average from the voltage command value of each phase is used as the modulation wave. As a result, the maximum amplitude of the modulation wave is multiplied by (square root 3)/2, while voltage utilization becomes 100 percents and line voltage is unchanged.

State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

EVENT name	Occurrence factor
STOP	by user operation
RUN	by user operation
ERROR	when the system detects an error
RESET	by user operation

Flowchart

The below figures show flowcharts of motor encoder module.

Configuration

Build Time Configurations for rm_motor_encoder

The following build time configurations are defined in fsp_cfg/rm_motor_encoder_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Encoder Vector Control (rm_motor_encoder)

This module can be added to the Stacks tab via New Stack > Motor > Motor Encoder Vector Control (rm_motor_encoder).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_encoder0	Module name.
General > Limit of over current (A)	Must be a valid value	2.0F	Limit of over current.(Detection threshold)
General > Limit of over voltage (V)	Must be a valid value	28.0F	Limit of over voltage.(Detection threshold)
General > Limit of over speed (rpm)	Must be a valid value	2100.0F	Limit of over speed.(Detection threshold)
General > Limit of low voltage (V)	Must be a valid value	18.0F	Limit of low voltage.(Detection threshold)
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt.

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

Usage Notes

Limitations

Examples

Basic Example

This is a basic example of minimal use of the motor encoder module in an application.

```
void motor_encoder_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_ENCODER_Open(g_motor_encoder0.p_ctrl, g_motor_encoder0.p_cfg);
    handle_error(err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_ENCODER_SpeedSet(g_motor_encoder0.p_ctrl,
RM_MOTOR_ENCODER_TEST_OVER_SPEED_LIMIT);
 /* Set position reference before motor run */
    (void) RM_MOTOR_ENCODER_PositionSet(g_motor_encoder0.p_ctrl, &g_posref_sample1);
 /* Start motor rotation */
    (void) RM_MOTOR_ENCODER_Run(g_motor_encoder0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_ENCODER_StatusGet(g_motor_encoder0.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_ENCODER_AngleGet(g_motor_encoder0.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_ENCODER_SpeedGet(g_motor_encoder0.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_ENCODER_ErrorCheck(g_motor_encoder0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_Stop(g_motor_encoder0.p_ctrl);
 /* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_ErrorSet(g_motor_encoder0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset Speed Control */
    (void) RM_MOTOR_ENCODER_Reset(g_motor_encoder0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_ENCODER_Close(g_motor_encoder0.p_ctrl);
}
```

Enumerations

enum	motor_encoder_ctrl_t

enum	motor_encoder_ctrl_event_t

Enumeration Type Documentation

◆ motor_encoder_ctrl_t

enum motor_encoder_ctrl_t	
Enumerator	
MOTOR_ENCODER_CTRL_STOP	Stop mode.
MOTOR_ENCODER_CTRL_RUN	Run mode.
MOTOR_ENCODER_CTRL_ERROR	Error mode.

◆ motor_encoder_ctrl_event_t

enum motor_encoder_ctrl_event_t	
Enumerator	
MOTOR_ENCODER_CTRL_EVENT_STOP	Stop event.
MOTOR_ENCODER_CTRL_EVENT_RUN	Run event.
MOTOR_ENCODER_CTRL_EVENT_ERROR	Error event.
MOTOR_ENCODER_CTRL_EVENT_RESET	Reset event.

Function Documentation

◆ RM_MOTOR_ENCODER_Open()

fsp_err_t RM_MOTOR_ENCODER_Open (motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg*)

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_ENCODER_Open(g_motor_encoder0.p_ctrl, g_motor_encoder0.p_cfg);
```

Return values

FSP_SUCCESS	MOTOR successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

Note

◆ RM_MOTOR_ENCODER_Close()

fsp_err_t RM_MOTOR_ENCODER_Close (motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Encoder Control block. Implements motor_api_t::close.

Example:

```
/* Close Speed Control */
   (void) RM_MOTOR_ENCODER_Close(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_Run()

fsp_err_t RM_MOTOR_ENCODER_Run (motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_ENCODER_Run(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_Stop()

fsp_err_t RM_MOTOR_ENCODER_Stop (motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_ENCODER_Stop(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_Reset()

fsp_err_t RM_MOTOR_ENCODER_Reset (motor_ctrl_t *const *p_ctrl*)

Reset Motor Encoder Control block. Implements motor_api_t::reset.

Example:

```
/* Reset Speed Control */
   (void) RM_MOTOR_ENCODER_Reset(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_ErrorSet()

fsp_err_t RM_MOTOR_ENCODER_ErrorSet (motor_ctrl_t *const *p_ctrl*, motor_error_t const *error*)

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_ENCODER_ErrorSet(g_motor_encoder0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_SpeedSet()

fsp_err_t RM_MOTOR_ENCODER_SpeedSet (motor_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_ENCODER_SpeedSet(g_motor_encoder0.p_ctrl,
RM_MOTOR_ENCODER_TEST_OVER_SPEED_LIMIT);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_PositionSet()

fsp_err_t RM_MOTOR_ENCODER_PositionSet (motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position*)

Set position reference[degree]. Implements motor_api_t::positionSet.

Example:

```
/* Set position reference before motor run */
   (void) RM_MOTOR_ENCODER_PositionSet(g_motor_encoder0.p_ctrl, &g_posref_sample1);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data set pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_StatusGet()

fsp_err_t RM_MOTOR_ENCODER_StatusGet (motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status*)

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_ENCODER_StatusGet(g_motor_encoder0.p_ctrl, &smpl_status);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_AngleGet()

fsp_err_t RM_MOTOR_ENCODER_AngleGet (motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad*)

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
   (void) RM_MOTOR_ENCODER_AngleGet(g_motor_encoder0.p_ctrl, &smpl_angle);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_SpeedGet()

fsp_err_t RM_MOTOR_ENCODER_SpeedGet (motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_ENCODER_SpeedGet(g_motor_encoder0.p_ctrl, &smpl_speed);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_ErrorCheck()

fsp_err_t RM_MOTOR_ENCODER_ErrorCheck (motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error*)

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_ENCODER_ErrorCheck(g_motor_encoder0.p_ctrl, &smpl_error);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_WaitStopFlagGet()

fsp_err_t RM_MOTOR_ENCODER_WaitStopFlagGet (motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag*)

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

◆ RM_MOTOR_ENCODER_FunctionSelect()

fsp_err_t RM_MOTOR_ENCODER_FunctionSelect (motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function*)

Select using function. Implements motor_api_t::functionSelect.

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Mode unmatch

Note

◆ RM_MOTOR_ENCODER_InertiaEstimateStart()

fsp_err_t RM_MOTOR_ENCODER_InertiaEstimateStart (motor_ctrl_t *const *p_ctrl*)

Start inertia estimation function.

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Mode unmatch

Note

◆ RM_MOTOR_ENCODER_InertiaEstimateStop()

fsp_err_t RM_MOTOR_ENCODER_InertiaEstimateStop (motor_ctrl_t *const *p_ctrl*)

Stop(Cancel) inertia estimation function.

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Mode unmatch

Note

◆ RM_MOTOR_ENCODER_ReturnOriginStart()

fsp_err_t RM_MOTOR_ENCODER_ReturnOriginStart (motor_ctrl_t *const *p_ctrl*)

Start return origin function.

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Mode unmatch

Note

◆ RM_MOTOR_ENCODER_ReturnOriginStop()

fsp_err_t RM_MOTOR_ENCODER_ReturnOriginStop (motor_ctrl_t *const *p_ctrl*)

Stop(Cancel) return origin function.

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Mode unmatch

Note

5.2.11.5 Motor Angle (rm_motor_estimate)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_ESTIMATE_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Opens and configures the Angle Estimation module. Implements motor_angle_api_t::open. More...
fsp_err_t	RM_MOTOR_ESTIMATE_Close (motor_angle_ctrl_t *const p_ctrl) Disables specified Angle Estimation module. Implements motor_angle_api_t::close. More...
fsp_err_t	RM_MOTOR_ESTIMATE_Reset (motor_angle_ctrl_t *const p_ctrl) Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset. More...
fsp_err_t	RM_MOTOR_ESTIMATE_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage) Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_SpeedSet (motor_angle_ctrl_t *const p_ctrl,

	float const speed_ctrl, float const damp_speed) Set Speed Information. Implements motor_angle_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi) Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err) Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq) Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate. More...
fsp_err_t	RM_MOTOR_ESTIMATE_InternalCalculate (motor_angle_ctrl_t *const p_ctrl) Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More...
fsp_err_t	RM_MOTOR_ESTIMATE_AngleAdjust (motor_angle_ctrl_t *const p_ctrl) Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More...
fsp_err_t	RM_MOTOR_ESTIMATE_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl) Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More...
fsp_err_t	RM_MOTOR_ESTIMATE_InfoGet (motor_angle_ctrl_t *const p_ctrl,

	motor_angle_encoder_info_t *const p_info) Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_CyclicProcess (motor_angle_ctrl_t *const p_ctrl) Perform induction cyclic process. Implements motor_angle_api_t::cyclicProcess. More...
fsp_err_t	RM_MOTOR_ESTIMATE_SensorDataSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data) Set sensor data. Implements motor_angle_api_t::sensorDataSet. More...

Detailed Description

Calculation process for the motor control on RX MCUs. This module implements the Motor angle Interface.

Overview

The motor angle and speed estimation module is used to calculate rotor angle and rotational speed in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application.

Features

The Motor Angle and Speed Estimation Module has below features.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

Configuration

Build Time Configurations for rm_motor_estimate

The following build time configurations are defined in fsp_cfg/rm_motor_estimate_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Angle and Speed Estimation (rm_motor_estimate)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle and Speed

Estimation (rm_motor_estimate).

Configuration	Options	Default	Description
Motor Parameter > Pole pairs	Must be a valid positive integer.	2	Pole pairs
Motor Parameter > Resistance[ohm]	Must be a valid value	8.5F	Resistance
Motor Parameter > Inductance of d-axis[H]	Must be a valid value	0.0045F	Inductance of d-axis
Motor Parameter > Inductance of q-axis[H]	Must be a valid value	0.0045F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux[Wb]	Must be a valid value	0.02159F	Permanent magnetic flux
Motor Parameter > Rotor inertia[kgm^2]	Must be a valid value	0.0000028F	Rotor inertia
Motor Parameter > Nominal current[Arms]	Must be a valid value	1.67	Nominal current
Name	Name must be a valid C symbol	g_motor_angle0	Module name.
Openloop damping	• Disable • Enable	Enable	Openloop damping functionally enable or disable
Natural frequency of BEMF observer	Must be a valid value	1000.0F	Natural frequency of BEMF observer
Damping ratio of BEMF observer	Must be a valid value	1.0F	Damping ratio of BEMF observer
Natural frequency of PLL Speed estimate loop	Must be a valid value	20.0F	Natural frequency of PLL Speed estimate loop
Damping ratio of PLL Speed estimate loop	Must be a valid value	1.0F	Damping ratio of PLL Speed estimate loop
Control period	Must be a valid value	0.00005F	Control period

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the Motor Angle and Speed Estimation:

Examples

Basic Example

This is a basic example of minimal use of the Motor Angle and Speed Estimation in an application.

```
void motor_estimate_basic_example (void)
{
 fsp_err_t             err = FSP_SUCCESS;
 motor_angle_current_t smpl_current:
 motor_angle_voltage_reference_t smpl_voltage;
 /* Initializes the module. */
    err = RM_MOTOR_ESTIMATE_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 while (true)
    {
 /* Application work here. */
 /* Set PI Control Flag before get Angle/Speed and Estimated Component */
       (void) RM_MOTOR_ESTIMATE_FlagPiCtrlSet(&g_mtr_angle0_ctrl, 1U);
       smpl_current.id = 1.0F;
       smpl_current.iq = 1.0F;
       smpl_voltage.vd = 10.0F;
       smpl_voltage.vq = 10.0F;
 /* Set Current and Speed data before get Angle/Speed and Estimated Component */
       (void) RM_MOTOR_ESTIMATE_CurrentSet(&g_mtr_angle0_ctrl, smpl_current,
smpl_voltage);
 /* Set Internal Speed Reference & damping speed data before get Angle/Speed and
Estimated Component */
       (void) RM_MOTOR_ESTIMATE_SpeedSet(&g_mtr_angle0_ctrl, 104.27F, 10.0F);
 /* Get Angle/Speed data */
       (void) RM_MOTOR_ESTIMATE_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
```

```
 /* Get Estimated Component */
       (void) RM_MOTOR_ESTIMATE_EstimatedComponentGet(&g_mtr_angle0_ctrl, &f_get_ed,
&f_get_eq);
    }
}
```

Function Documentation

◆ RM_MOTOR_ESTIMATE_Open()

fsp_err_t RM_MOTOR_ESTIMATE_Open (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Opens and configures the Angle Estimation module. Implements motor_angle_api_t::open.

Return values

FSP_SUCCESS	MTR_ANGL_EST successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_ESTIMATE_Close()

fsp_err_t RM_MOTOR_ESTIMATE_Close (motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Estimation module. Implements motor_angle_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_Reset()

fsp_err_t RM_MOTOR_ESTIMATE_Reset (motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_CurrentSet()

fsp_err_t RM_MOTOR_ESTIMATE_CurrentSet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage*)

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_SpeedSet()

fsp_err_t RM_MOTOR_ESTIMATE_SpeedSet (motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed*)

Set Speed Information. Implements motor_angle_api_t::speedSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_ESTIMATE_FlagPiCtrlSet (motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi*)

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_AngleSpeedGet()

fsp_err_t RM_MOTOR_ESTIMATE_AngleSpeedGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err*)

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_EstimatedComponentGet()

fsp_err_t RM_MOTOR_ESTIMATE_EstimatedComponentGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq*)

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

Return values

FSP_SUCCESS	Successfully data gotten.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_ParameterUpdate()

fsp_err_t RM_MOTOR_ESTIMATE_ParameterUpdate (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data is update.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_InternalCalculate()

fsp_err_t RM_MOTOR_ESTIMATE_InternalCalculate (motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_AngleAdjust()

fsp_err_t RM_MOTOR_ESTIMATE_AngleAdjust (motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_EncoderCyclic()

fsp_err_t RM_MOTOR_ESTIMATE_EncoderCyclic (motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_InfoGet()

fsp_err_t RM_MOTOR_ESTIMATE_InfoGet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info*)

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_CyclicProcess()

fsp_err_t RM_MOTOR_ESTIMATE_CyclicProcess (motor_angle_ctrl_t *const *p_ctrl*)

Perform induction cyclic process. Implements motor_angle_api_t::cyclicProcess.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

◆ RM_MOTOR_ESTIMATE_SensorDataSet()

fsp_err_t RM_MOTOR_ESTIMATE_SensorDataSet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_ad_data_t *const *p_ad_data*)

Set sensor data. Implements motor_angle_api_t::sensorDataSet.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

5.2.11.6 Motor Inertia estimate (rm_motor_inertia_estimate)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_Open (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg) Opens and configures the motor inertia estimate module. Implements motor_inertia_estimate_api_t::open. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_Close (motor_inertia_estimate_ctrl_t *const p_ctrl) Disables specified motor inertia estimate module. Implements motor_inertia_estimate_api_t::close. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_Start (motor_inertia_estimate_ctrl_t *const p_ctrl) Start inertia estimation. Implements motor_inertia_estimate_api_t::start. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_Stop (motor_inertia_estimate_ctrl_t *const p_ctrl) Stop (Cancel) inertia estimation. Implements motor_inertia_estimate_api_t::stop. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_Reset (motor_inertia_estimate_ctrl_t *const p_ctrl) Reset variables of inertia estimate module. Implements motor_inertia_estimate_api_t::reset. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_InfoGet (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_info_t *const p_info) Get information of inertia estimation. Implements motor_inertia_estimate_api_t::infoGet. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_DataSet (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_set_data_t *const p_set_data) Set necessary data to inertia estimation. Implements

	motor_inertia_estimate_api_t::dataSet. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_SpeedCyclic (motor_inertia_estimate_ctrl_t *const p_ctrl) Cyclic process of inertia estimation at speed control period. Implements motor_inertia_estimate_api_t::speedCyclic. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_CurrentCyclic (motor_inertia_estimate_ctrl_t *const p_ctrl) Cyclic process of inertia estimation at current control period (called at A/D conversion finish interrupt). Implements motor_inertia_estimate_api_t::currentCyclic. More...
fsp_err_t	RM_MOTOR_INERTIA_ESTIMATE_ParameterUpdate (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg) Update the parameters of inertia estimate. Implements motor_inertia_estimate_api_t::parameterUpdate. More...

Detailed Description

Measurement and calculation process for the motor control on RX MCUs. This module implements the Motor Inertia Estimate Interface.

Overview

The motor inertia estimation module is used to measure and calculate rotor inertia in an application. This module should be used with Renesas Motor Workbench (RMW) basically.

Features

The Motor Inertia Estimation Module has below features.

- Measurement process work automatically
- Calculate rotor inertia [kgm^2].

Configuration

Build Time Configurations for rm_motor_inertia_estimate

The following build time configurations are defined in fsp_cfg/rm_motor_inertia_estimate_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled	Default (BSP)	If selected code for parameter checking is

- Disabled included in the build.

Configurations for Motor > Motor inertia estimation (rm_motor_inertia_estimate)

This module can be added to the Stacks tab via New Stack > Motor > Motor inertia estimation (rm_motor_inertia_estimate).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_inertia_estimate0	Module name.
Moved position distance to measure (degree)	Must be set -360 to -10 or 10 to 360.	360	Moved position distance to measure inertia (degree)
Maximum speed (rpm)	Must be set over 60.	500	Maximum rotation speed (rpm)
Acceleration time	Must be a valid non-negative value.	0.3	Acceleration time
Motor inertia	Must be a valid non-negative value.	0.0000041	Motor inertia
Low threshold to judge speed	Must be set 0.1 to 0.5.	0.1	Low threshold to judge speed reached
High threshold to judge speed	Must be set 0.5 to 0.9, and greater than low threshold	0.9	High threshold to judge speed reached
Time to wait moving stability (sec)	Must be a valid non-negative value.	0.8	Time to wait moving stability
Cyclic period of current control (sec)	Must be a valid non-negative value.	0.00005	Cyclic period of current control (sec)
Cyclic period of speed control (sec)	Must be a valid non-negative value.	0.0005	Cyclic period of speed control (sec)
Motor pole pairs	Must be a valid non-negative value.	4	Motor pole pairs
Motor magnet flux (Wb)	Must be a valid non-negative value.	0.00623	Motor magnet flux (Wb)
Interval time	Must be a valid non-negative value.	400.0	Interval time. Please set same value as Position control

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

Examples

Basic Example

This is a basic example of minimal use of the Motor Inertia Estimation in an application.

```
void motor_inertia_estimate_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Open(&g_mtr_inertia_estimate0_ctrl,
&g_mtr_inertia_estimate0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start process. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Start(&g_mtr_inertia_estimate0_ctrl);
    temp_set_data.f_iq = 1.0F;
    temp_set_data.f_speed_radian_control = 1.04F;
    temp_set_data.s2_position_degree    = 180;
    temp_set_data.u1_position_state      = 0;
 /* Set data to the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_DataSet(&g_mtr_inertia_estimate0_ctrl,
&temp_set_data);
 /* Get information from the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_InfoGet(&g_mtr_inertia_estimate0_ctrl,
&temp_information);
 /* Stop process. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Stop(&g_mtr_inertia_estimate0_ctrl);
 /* Close the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Close(&g_mtr_inertia_estimate0_ctrl);
}
```

Data Structures

struct	motor_inertia_estimate_extended_cfg_t

struct	motor_inertia_estimate_instance_ctrl_t

Data Structure Documentation

◆ motor_inertia_estimate_extended_cfg_t

struct motor_inertia_estimate_extended_cfg_t		
Extended configurations for motor inertia estimate		
Data Fields		
int16_t	s2_move_degree	Moving position reference [degree].
uint16_t	u2_J_max_speed_rpm	Maximum Speed [rpm].
float	f_accel_time	Acceleration time.
float	f_rotor_inertia	Initialized rotor inertia value.
float	f_judge_low_threshold	Low threshold to judge speed.
float	f_judge_high_threshold	High threshold to judge speed.
float	f_change_mode_time	Timing value to change internal mode.
float	f_current_ctrl_period	Period of current control [sec].
float	f_speed_ctrl_period	Period of speed control [sec].
uint8_t	u1_motor_polepairs	Motor pole pairs.
float	f_motor_m	Motor magnet flux [Wb].
float	f_position_interval	Interval counts for reference position change.

◆ motor_inertia_estimate_instance_ctrl_t

struct motor_inertia_estimate_instance_ctrl_t		
Inertia estimate instance control block		
Data Fields		
uint32_t	open	Used to determine if the module is configured.
motor_inertia_estimate_start_flag_t	start_flag	start/stop flag
motor_inertia_estimate_mode_t	mode	Internal mode.
uint8_t	u1_mode_count	Use to manage internal mode.
motor_inertia_estimate_period_t	speed_period	Measure period.
motor_inertia_estimate_period_t	speed_period_buffer	Buffer of measure period to be referred by current cyclic.
uint32_t	u4_measure_count	Counter for speed control cycle.

uint32_t	u4_wait_count	Counter to wait change mode timing.
uint8_t	u1_position_move_mode	Position move mode (TRIANGLE/TRAPEZOID)
int16_t	s2_initial_position_degree	Initial position.
float	f_iq_ad	q-axis current [A]
float	f_summary_iq_ad	Summary of q-axis current.
float	f_position_mode_time	Summary of speed control period to judge the timing.
float	f_position_dt_time_sec	Differential time of move.
int16_t	s2_position_reference_degree	Position reference [degree].
float	f_estimated_value	Estimated inertia.
float	f_inertia_value1	Buffer to calculate inertia 1.
float	f_inertia_value2	Buffer to calculate inertia 2.
float	f_interval_time	Interval time about position transition.
float	f_inertia_speed_ctrl1	
float	f_inertia_speed_ctrl2	
float	f_inertia_speed_ctrl3	
float	f_inertia_speed_ctrl4	
float	f_inertia_speed_ctrl5	
float	f_inertia_speed_ctrl6	
float	f_inertia_speed_ctrl7	
float	f_inertia_speed_ctrl8	
float	f_inertia_integ_iq1	
float	f_inertia_integ_iq2	
float	f_inertia_integ_iq3	
float	f_inertia_integ_iq4	
float	f_inertia_integ_time1	
float	f_inertia_integ_time2	
float	f_inertia_integ_time3	
float	f_inertia_integ_time4	
float	f_inverse_motor_polepairs	Inverse motor pole pairs (for calculation)
motor_inertia_estimate_set_data_t	receive_data	Received data set from speed(position) and current.

motor_inertia_estimate_cfg_t const *	p_cfg	Pointer of configuration structure.

Function Documentation

◆ RM_MOTOR_INERTIA_ESTIMATE_Open()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Open (motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_cfg_t const *const *p_cfg*)

Opens and configures the motor inertia estimate module. Implements motor_inertia_estimate_api_t::open.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Open(&g_mtr_inertia_estimate0_ctrl,
&g_mtr_inertia_estimate0_cfg);
```

Return values

FSP_SUCCESS	Motor inertia estimate module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_INERTIA_ESTIMATE_Close()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Close (motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Disables specified motor inertia estimate module. Implements motor_inertia_estimate_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Close(&g_mtr_inertia_estimate0_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_INERTIA_ESTIMATE_Start()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Start (motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Start inertia estimation. Implements motor_inertia_estimate_api_t::start.

Example:

```
/* Start process. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Start(&g_mtr_inertia_estimate0_ctrl);
```

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_INERTIA_ESTIMATE_Stop()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Stop (motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Stop (Cancel) inertia estimation. Implements motor_inertia_estimate_api_t::stop.

Example:

```
/* Stop process. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Stop(&g_mtr_inertia_estimate0_ctrl);
```

Return values

FSP_SUCCESS	Successfully stopped (canceled).
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_INERTIA_ESTIMATE_Reset()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Reset (motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Reset variables of inertia estimate module. Implements motor_inertia_estimate_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_INERTIA_ESTIMATE_InfoGet()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_InfoGet (motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_info_t *const *p_info*)

Get information of inertia estimation. Implements motor_inertia_estimate_api_t::infoGet.

Example:

```
/* Get information from the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_InfoGet(&g_mtr_inertia_estimate0_ctrl,
&temp_information);
```

Return values

FSP_SUCCESS	Successfully get data.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Argument pointer is invalid.

◆ RM_MOTOR_INERTIA_ESTIMATE_DataSet()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_DataSet (motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_set_data_t *const *p_set_data*)

Set necessary data to inertia estimation. Implements motor_inertia_estimate_api_t::dataSet.

Example:

```
/* Set data to the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_DataSet(&g_mtr_inertia_estimate0_ctrl,
&temp_set_data);
```

Return values

FSP_SUCCESS	Successfully set data.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_INERTIA_ESTIMATE_SpeedCyclic()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_SpeedCyclic (motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Cyclic process of inertia estimation at speed control period. Implements motor_inertia_estimate_api_t::speedCyclic.

Return values

FSP_SUCCESS	Successfully perform the process.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_INERTIA_ESTIMATE_CurrentCyclic()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_CurrentCyclic (motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Cyclic process of inertia estimation at current control period (called at A/D conversion finish interrupt). Implements motor_inertia_estimate_api_t::currentCyclic.

Return values

FSP_SUCCESS	Successfully perform the process.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_INERTIA_ESTIMATE_ParameterUpdate()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_ParameterUpdate (motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_cfg_t const *const *p_cfg*)

Update the parameters of inertia estimate. Implements motor_inertia_estimate_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

5.2.11.7 Motor Position Controller (rm_motor_position)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_POSITION_Open (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg) Opens and configures the Motor Position Module. Implements motor_position_api_t::open. More...
fsp_err_t	RM_MOTOR_POSITION_Close (motor_position_ctrl_t *const p_ctrl) Disables specified Motor Position Module. Implements motor_position_api_t::close. More...
fsp_err_t	RM_MOTOR_POSITION_Reset (motor_position_ctrl_t *const p_ctrl) Reset the variables of Motor Position Module. Implements motor_position_api_t::reset. More...
fsp_err_t	RM_MOTOR_POSITION_PositionGet (motor_position_ctrl_t *const p_ctrl, int16_t *const p_position) Get Rotor Position Data [degree]. Implements motor_position_api_t::positionGet. More...
fsp_err_t	RM_MOTOR_POSITION_PositionSet (motor_position_ctrl_t *const p_ctrl, float const position_rad) Set Position Data from Encoder [radian]. Implements motor_position_api_t::positionSet. More...
fsp_err_t	RM_MOTOR_POSITION_PositionReferenceSet (motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg) Set Position Reference Data [degree]. Implements motor_position_api_t::positionReferenceSet. More...
fsp_err_t	RM_MOTOR_POSITION_ControlModeSet (motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode) Set Position Control Mode. Implements motor_position_api_t::controlModeSet. More...
fsp_err_t	RM_MOTOR_POSITION_PositionControl (motor_position_ctrl_t *const p_ctrl)

	Calculates internal position reference.(Main process of Position Control) Implements motor_position_api_t::positionControl. More...
fsp_err_t	RM_MOTOR_POSITION_IpdSpeedPControl (motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref) Calculates the q-axis current reference by P control. Implements motor_position_api_t::ipdSpeedPControl. More...
fsp_err_t	RM_MOTOR_POSITION_SpeedReferencePControlGet (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref) Get Speed Reference by P Control. Implements motor_position_api_t::speedReferencePControlGet. More...
fsp_err_t	RM_MOTOR_POSITION_SpeedReferenceIpdControlGet (motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref) Get Speed Reference by IPD Control. Implements motor_position_api_t::speedReferenceIpdControlGet. More...
fsp_err_t	RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref) Get Speed Reference by Feedforward. Implements motor_position_api_t::speedReferenceFeedforwardGet. More...
fsp_err_t	RM_MOTOR_POSITION_InfoGet (motor_position_ctrl_t *const p_ctrl, motor_position_info_t *const p_info) Get position information. More...
fsp_err_t	RM_MOTOR_POSITION_ParameterUpdate (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg) Update the parameters of Position Control Calculation. Implements motor_position_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RX MCUs. This module implements the Motor position Interface.

Overview

The motor position is used to control the position of motor rotor in an application. This module should be called cyclically in an application (e.g. in cyclic timer interrupt). This module calculates speed reference with inputted position reference and current rotational speed.

Features

The Motor Position Module has below features.

- Calculate speed reference.

Configuration

Build Time Configurations for rm_motor_position

The following build time configurations are defined in fsp_cfg/rm_motor_position_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Position Controller (rm_motor_position)

This module can be added to the Stacks tab via New Stack > Motor > Motor Position Controller (rm_motor_position).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_position0	Module name.
General > Position dead band	Must be a valid non-negative value.	1U	Ignored area of position control.
General > Position band limit	Must be a valid non-negative value.	3U	Ignored area of position control.
General > Speed feedforward ratio	Must be a valid value	0.8F	Speed feedforward ratio.
General > Encoder counts per one rotation	Must be a valid value	1200.0F	Encoder counts per one rotation.
General > Position omega	Must be a valid value	10.0F	Position control omega.
General > Period of speed control (sec)	Must be a valid value	0.0005F	Period of speed control.
IPD > IPD LPF	• Disable • Enable	Disable	IPD LPF process enable or disable
IPD > Position Kp ratio	Must be a valid value	0.3F	Position Kp ratio.
IPD > Position feedforward ratio	Must be a valid value	0.0F	Position feedforward ratio.

IPD > Speed K ratio	Must be a valid value	2.0F	Speed K ratio
IPD > Error Limit #1	Must be a valid value	10.0F	Error limitation #1
IPD > Error limit #2	Must be a valid value	0.2F	Error limitation #2
IPD > LPF omega	Must be a valid value	500.0F	LPF omega.
IPD > LPF zeta	Must be a valid value	1.0F	LPF zeta.
Position Profiling > Interval time	Must be a valid non-negative value.	400U	Interval time.
Position Profiling > Accel time	Must be a valid value	0.3F	Accel time.
Position Profiling > Maximum accel time (sec)	Must be a valid value	8117.96F	Maximum acceleration time (sec)
Position Profiling > Acceleration maximum speed	Must be a valid value	2000.0F	Acceleration maximum speed.
Position Profiling > Update step of timer	Must be a valid value	0.0005F	Update step of timer.
Motor Parameter > Pole pair	Value must be non-negative	7	Pole pair
Motor Parameter > Resistance (ohm)	Must be a valid value	0.453F	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid value	0.0009447F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid value	0.0009447F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid value	0.006198F	Permanent magnetic flux
Motor Parameter > Motor inertia (kgm^2)	Must be a valid value	0.00000962F	Motor inertia

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the period of speed control with non-negative value.
- Set the limit of speed change step with non-negative value.
- Set the maximum speed with non-negative value.

Examples

Basic Example

This is a basic example of using the Motor Position module in an application.

```
void motor_position_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_POSITION_Open(g_motor_position0.p_ctrl, g_motor_position0.p_cfg);
    handle_error(err);
 /* Set working mode */
    RM_MOTOR_POSITION_ModeSet(g_motor_position0.p_ctrl,
MOTOR_POSITION_CTRL_MODE_STEP);
 /* Set position reference */
 RM_MOTOR_POSITION_PositionReferenceSet(g_motor_position0.p_ctrl, 180U);
 /* Basically run this module at cyclic interrupt (e.g. AGT timer).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Perform Position Control Process */
 RM_MOTOR_POSITION_PositionControl(g_motor_position0.p_ctrl);
 /* Perform Speed IPD Control Process */
 RM_MOTOR_POSITION_IpdSpeedPControl(g_motor_position0.p_ctrl, 0.0F, 0.0F,
&temp_iq_ref);
 /* Get Position */
 RM_MOTOR_POSITION_PositionGet(g_motor_position0.p_ctrl, &temp_position);
 /* Update parameters */
 RM_MOTOR_POSITION_ParameterUpdate(g_motor_position0.p_ctrl,
&g_motor_position0.p_cfg);
    }
 /* Reset Speed Control */
```

```
RM_MOTOR_POSITION_Reset(g_motor_position0.p_ctrl);
/* Close Speed Control */
RM_MOTOR_POSITION_Close(g_motor_position0.p_ctrl);
}
```

Enumerations

enum	motor_position_ipd_lpf_t

Enumeration Type Documentation

◆ motor_position_ipd_lpf_t

enum motor_position_ipd_lpf_t

Enumerator	
MOTOR_POSITION_IPD_LPF_DISABLE	ipd control is disabled
MOTOR_POSITION_IPD_LPF_ENABLE	ipd control is enabled

Function Documentation

◆ RM_MOTOR_POSITION_Open()

fsp_err_t RM_MOTOR_POSITION_Open (motor_position_ctrl_t *const *p_ctrl*, motor_position_cfg_t const *const *p_cfg*)

Opens and configures the Motor Position Module. Implements motor_position_api_t::open.

Return values

FSP_SUCCESS	Motor Position Module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Set parameter is invalid.

◆ RM_MOTOR_POSITION_Close()

fsp_err_t RM_MOTOR_POSITION_Close (motor_position_ctrl_t *const *p_ctrl*)

Disables specified Motor Position Module. Implements motor_position_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_Reset()

fsp_err_t RM_MOTOR_POSITION_Reset (motor_position_ctrl_t *const *p_ctrl*)

Reset the variables of Motor Position Module. Implements motor_position_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_PositionGet()

fsp_err_t RM_MOTOR_POSITION_PositionGet (motor_position_ctrl_t *const *p_ctrl*, int16_t *const *p_position*)

Get Rotor Position Data [degree]. Implements motor_position_api_t::positionGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_PositionSet()

fsp_err_t RM_MOTOR_POSITION_PositionSet (motor_position_ctrl_t *const *p_ctrl*, float const *position_rad*)

Set Position Data from Encoder [radian]. Implements motor_position_api_t::positionSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_PositionReferenceSet()

fsp_err_t RM_MOTOR_POSITION_PositionReferenceSet (motor_position_ctrl_t *const *p_ctrl*, int16_t const *position_reference_deg*)

Set Position Reference Data [degree]. Implements motor_position_api_t::positionReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_ControlModeSet()

fsp_err_t RM_MOTOR_POSITION_ControlModeSet (motor_position_ctrl_t *const *p_ctrl*, motor_position_ctrl_mode_t const *mode*)

Set Position Control Mode. Implements motor_position_api_t::controlModeSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_PositionControl()

fsp_err_t RM_MOTOR_POSITION_PositionControl (motor_position_ctrl_t *const *p_ctrl*)

Calculates internal position reference.(Main process of Position Control) Implements motor_position_api_t::positionControl.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_IpdSpeedPControl()

fsp_err_t RM_MOTOR_POSITION_IpdSpeedPControl (motor_position_ctrl_t *const *p_ctrl*, float const *ref_speed_rad*, float const *speed_rad*, float *const *p_iq_ref*)

Calculates the q-axis current reference by P control. Implements motor_position_api_t::ipdSpeedPControl.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_SpeedReferencePControlGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferencePControlGet (motor_position_ctrl_t *const *p_ctrl*, float *const *p_speed_ref*)

Get Speed Reference by P Control. Implements motor_position_api_t::speedReferencePControlGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_SpeedReferenceIpdControlGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferenceIpdControlGet (motor_position_ctrl_t *const *p_ctrl*, float const *max_speed_rad*, float *const *p_speed_ref*)

Get Speed Reference by IPD Control. Implements motor_position_api_t::speedReferenceIpdControlGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet (motor_position_ctrl_t *const *p_ctrl*, float *const *p_speed_ref*)

Get Speed Reference by Feedforward. Implements motor_position_api_t::speedReferenceFeedforwardGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_InfoGet()

fsp_err_t RM_MOTOR_POSITION_InfoGet (motor_position_ctrl_t *const *p_ctrl*, motor_position_info_t *const *p_info*)

Get position information.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_ParameterUpdate()

fsp_err_t RM_MOTOR_POSITION_ParameterUpdate (motor_position_ctrl_t *const *p_ctrl*, motor_position_cfg_t const *const *p_cfg*)

Update the parameters of Position Control Calculation. Implements motor_position_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

5.2.11.8 Motor return origin (rm_motor_return_origin)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_RETURN_ORIGIN_Open (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg) Opens and configures the motor return origin module. Implements motor_return_origin_api_t::open. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_Close (motor_return_origin_ctrl_t *const p_ctrl) Disables specified motor return origin module. Implements motor_return_origin_api_t::close. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_Start (motor_return_origin_ctrl_t *const p_ctrl) Start return origin function. Implements motor_return_origin_api_t::start. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_Stop (motor_return_origin_ctrl_t *const p_ctrl) Stop (Cancel) return origin function. Implements motor_return_origin_api_t::stop. More...

fsp_err_t	RM_MOTOR_RETURN_ORIGIN_Reset (motor_return_origin_ctrl_t *const p_ctrl) Reset variables of return origin module. Implements motor_return_origin_api_t::reset. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_InfoGet (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_info_t *const p_info) Get information of return origin. Implements motor_return_origin_api_t::infoGet. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_DataSet (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_set_data_t *const p_set_data) Set necessary data to return origin function. Implements motor_return_origin_api_t::dataSet. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_SpeedCyclic (motor_return_origin_ctrl_t *const p_ctrl) Cyclic process of return origin function at speed control period. (Called at timer interrupt.) Implements motor_return_origin_api_t::speedCyclic. More...
fsp_err_t	RM_MOTOR_RETURN_ORIGIN_ParameterUpdate (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg) Update the parameters of return origin function. Implements motor_return_origin_api_t::parameterUpdate. More...

Detailed Description

Search origin position process for the motor control on RX MCUs. This module implements the Motor Return Origin Function Interface.

Overview

The motor return origin module is used to search origin position in an application. This module should be used with Renesas Motor Workbench (RMW) basically.

Features

The Motor return origin module has below features.

- Search origin position automatically.

Configuration

Build Time Configurations for rm_motor_return_origin

The following build time configurations are defined in fsp_cfg/rm_motor_return_origin_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor return origin function (rm_motor_return_origin)

This module can be added to the Stacks tab via New Stack > Motor > Motor return origin function (rm_motor_return_origin).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_return_origin0	Module name.
General > Using mode	Search by pushing	Search by pushing	Select using mode
Search speed (rpm)	Must be a valid non-negative value.	10.0	Speed to search origin position (rpm)
Acceleration of speed (rpm/sec)	Must be set 1 to 10000.	10000.0	Acceleration of rotation speed at return moving (rpm/sec)
Cyclic period of speed control (sec)	Must be a valid non-negative value.	0.0005	Cyclic period of speed control (sec)
Maximum current (A)	Must be a valid non-negative value.	1.8	Maximum current (A). Please set to match used motor.
Percentage of current to judge pushing (%)	Must be a valid non-negative value.	30.0	Perentage of current to judge pushing (%)
Pushing time (sec)	Must be a valid non-negative value.	1.0	Time to push the stopper (sec)
Degree to judge none stopper	Must be set -360 to 360.	360.0	When the motor runs over this value, it is judged impossible to search the stopper.
Degree to return	Must be a valid non-negative value.	3.0	Degree to return from the stopper (degree)
Mechanical gear ratio	Must be a valid non-negative value.	1.0	Mechanical gear ratio

Clock Configuration

This module doesn't depend on clock setting.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

Examples

Basic Example

This is a basic example of minimal use of the Motor Return origin in an application.

```
void motor_return_origin_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_RETURN_ORIGIN_Open(&g_mtr_return_origin0_ctrl,
&g_mtr_return_origin0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start process. */
    err = RM_MOTOR_RETURN_ORIGIN_Start(&g_mtr_return_origin0_ctrl);
    temp_set_data.f_iq             = 1.0F;
    temp_set_data.f_position_degree = 180.0F;
 /* Set data to the module. */
    err = RM_MOTOR_RETURN_ORIGIN_DataSet(&g_mtr_return_origin0_ctrl, &temp_set_data);
 /* Get information from the module. */
    err = RM_MOTOR_RETURN_ORIGIN_InfoGet(&g_mtr_return_origin0_ctrl, &temp_info);
 /* Stop process. */
    err = RM_MOTOR_RETURN_ORIGIN_Stop(&g_mtr_return_origin0_ctrl);
 /* Close the module. */
    err = RM_MOTOR_RETURN_ORIGIN_Close(&g_mtr_return_origin0_ctrl);
}
```

Data Structures

struct	motor_return_origin_extended_cfg_t
struct	motor_return_origin_pushing_t

struct	motor_return_origin_instance_ctrl_t

Data Structure Documentation

◆ motor_return_origin_extended_cfg_t

struct motor_return_origin_extended_cfg_t		
Extended configurations for return origin function		
Data Fields		
float	f_search_speed_rpm	Speed to search origin position [rpm].
float	f_return_accel_rpm	Acceleration speed when return [rpm/s].
float	f_speed_ctrl_period	Period of speed control [sec].
float	f_maximum_current	Maximum current [A].
float	f_current_limit_percent_push	Percentage of current at pushing.
float	f_pushing_time	Keep pushing time [sec].
float	f_over_degree	Angle to judge search impossible [degree].
float	f_return_degree	Return angle from pushing position [degree].
float	f_mechanical_gear_ratio	Mechanical gear ratio.

◆ motor_return_origin_pushing_t

struct motor_return_origin_pushing_t		
Variables for rerutn origin with pushing		
Data Fields		
uint32_t	u4_time_counter	Counter of speed cyclic (to judge the time)
float	f_sum_position	Summary of position data.
uint32_t	u4_sum_counter	Counter of summary.
float	f_move_amount	Movement amount [degree].
float	f_judge_iq	q-axis current to judge pushing
float	f_pushing_counts	Counts to measure pushing time.

◆ motor_return_origin_instance_ctrl_t

struct motor_return_origin_instance_ctrl_t
Return origin function instance control block

Data Fields		
uint32_t	open	Used to determine if the module is configured.
motor_return_origin_start_flag_t	start_flag	start/stop flag
motor_return_origin_state_t	state	State number of return origin process.
int8_t	s1_direction	Moving direction.
float	f_angle_degree_on_edge	Rotor angle on the edge [degree].
float	f_current_speed	Current speed.
float	f_origin_position_angle_degree	Searched origin position [degree].
float	f_search_speed	Speed to search origin position [rad / sampling time].
float	f_accel_speed	Speed acceleration.
float	f_position_reference_degree	Position reference [degree].
motor_return_origin_pushing_t	st_pushing	Variables for pushing.
motor_return_origin_set_data_t	receive_data	Received data from speed(position) & current.
motor_return_origin_cfg_t const *	p_cfg	Pointer of configuration structure.

Function Documentation

◆ RM_MOTOR_RETURN_ORIGIN_Open()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Open (motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_cfg_t const *const *p_cfg*)

Opens and configures the motor return origin module. Implements motor_return_origin_api_t::open .

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_RETURN_ORIGIN_Open(&g_mtr_return_origin0_ctrl,
&g_mtr_return_origin0_cfg);
```

Return values

FSP_SUCCESS	Motor return origin module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_RETURN_ORIGIN_Close()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Close (motor_return_origin_ctrl_t *const *p_ctrl*)

Disables specified motor return origin module. Implements motor_return_origin_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MOTOR_RETURN_ORIGIN_Close(&g_mtr_return_origin0_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_RETURN_ORIGIN_Start()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Start (motor_return_origin_ctrl_t *const *p_ctrl*)

Start return origin function. Implements motor_return_origin_api_t::start.

Example:

```
/* Start process. */
   err = RM_MOTOR_RETURN_ORIGIN_Start(&g_mtr_return_origin0_ctrl);
```

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_RETURN_ORIGIN_Stop()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Stop (motor_return_origin_ctrl_t *const *p_ctrl*)

Stop (Cancel) return origin function. Implements motor_return_origin_api_t::stop.

Example:

```
/* Stop process. */
   err = RM_MOTOR_RETURN_ORIGIN_Stop(&g_mtr_return_origin0_ctrl);
```

Return values

FSP_SUCCESS	Successfully stopped (canceled).
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_RETURN_ORIGIN_Reset()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Reset (motor_return_origin_ctrl_t *const *p_ctrl*)

Reset variables of return origin module. Implements motor_return_origin_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_RETURN_ORIGIN_InfoGet()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_InfoGet (motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_info_t *const *p_info*)

Get information of return origin. Implements motor_return_origin_api_t::infoGet.

Example:

```
/* Get information from the module. */

   err = RM_MOTOR_RETURN_ORIGIN_InfoGet(&g_mtr_return_origin0_ctrl, &temp_info);
```

Return values

FSP_SUCCESS	Successfully get data.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_RETURN_ORIGIN_DataSet()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_DataSet (motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_set_data_t *const *p_set_data*)

Set necessary data to return origin function. Implements motor_return_origin_api_t::dataSet.

Example:

```
/* Set data to the module. */

   err = RM_MOTOR_RETURN_ORIGIN_DataSet(&g_mtr_return_origin0_ctrl, &temp_set_data);
```

Return values

FSP_SUCCESS	Successfully set data.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_RETURN_ORIGIN_SpeedCyclic()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_SpeedCyclic (motor_return_origin_ctrl_t *const *p_ctrl*)

Cyclic process of return origin function at speed control period. (Called at timer interrupt.) Implements motor_return_origin_api_t::speedCyclic.

Return values

FSP_SUCCESS	Successfully perform the process.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_RETURN_ORIGIN_ParameterUpdate()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_ParameterUpdate (motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_cfg_t const *const *p_cfg*)

Update the parameters of return origin function. Implements motor_return_origin_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

5.2.11.9 Motor Angle (rm_motor_sense_encoder)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_SENSE_ENCODER_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Opens and configures the Angle Encoder module. Implements motor_angle_api_t::open. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_Close (motor_angle_ctrl_t *const p_ctrl) Disables specified Angle Encoder module. Implements motor_angle_api_t::close. More...

fsp_err_t	RM_MOTOR_SENSE_ENCODER_Reset (motor_angle_ctrl_t *const p_ctrl) Reset variables of Angle Encoder module. Implements motor_angle_api_t::reset. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_InternalCalculate (motor_angle_ctrl_t *const p_ctrl) Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err) Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_AngleAdjust (motor_angle_ctrl_t *const p_ctrl) Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl) Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info) Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Update the parameters of Angle&Speed calculation with an encoder. Implements motor_angle_api_t::parameterUpdate. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage) Set d/q-axis Current Data & Voltage Reference. Implements

	motor_angle_api_t::currentSet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed) Set Speed Information. Implements motor_angle_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi) Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq) Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_CyclicProcess (motor_angle_ctrl_t *const p_ctrl) Perform cyclic process. Implements motor_angle_api_t::cyclicProcess . More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_SensorDataSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data) Set sensor data. Implements motor_angle_api_t::sensorDataSet. More...

Detailed Description

Calculation process for the motor control on RX MCUs. This module implements the Motor angle Interface.

Overview

The motor angle and speed calculation with an encoder module is used to calculate rotor angle and rotational speed in an application. This module is designed to be used with the motor current module (rm_motor_current).

Features

The motor angle and speed calculation with an encoder module has the features listed below.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

Configuration

Build Time Configurations for rm_motor_sense_encoder

The following build time configurations are defined in fsp_cfg/rm_motor_sense_encoder_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Support HALL sensor	• Support • Unsupport	Support	

Configurations for Motor > Motor Angle and Speed Calculation with encoder (rm_motor_sense_encoder)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle and Speed Calculation with encoder (rm_motor_sense_encoder).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_sense_encoder0	Module name.
Hall sensor > sensor input port U	Manual Entry	BSP_IO_PORT_09_PIN_07	U-phase Hall sensor input port
Hall sensor > sensor input port V	Manual Entry	BSP_IO_PORT_09_PIN_05	V-phase Hall sensor input port
Hall sensor > sensor input port W	Manual Entry	BSP_IO_PORT_09_PIN_06	W-phase Hall sensor input port
Hall sensor > sensor pattern #1	Must be a valid integer	1	Hall sensor pattern #1
Hall sensor > sensor pattern #2	Must be a valid integer	5	Hall sensor pattern #2
Hall sensor > sensor pattern #3	Must be a valid integer	4	Hall sensor pattern #3
Hall sensor > sensor pattern #4	Must be a valid integer	6	Hall sensor pattern #4
Hall sensor > sensor pattern #5	Must be a valid integer	2	Hall sensor pattern #5
Hall sensor > sensor pattern #6	Must be a valid integer	3	Hall sensor pattern #6
Motor Parameter > Pole pairs	Must be a valid positive integer.	7	Pole pairs

Motor Parameter > Resistance (ohm)	Must be a valid non-negative value	0.453F	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid non-negative value	0.0009447F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid non-negative value	0.0009447F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid non-negative value	0.006198F	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid non-negative value	0.00000962F	Rotor inertia
Control Type	• Speed • Position	Position	Select control Type
Period of Current control (kHz)	Must be a valid non-negative value	20.0F	Period of Current control
Period of Speed control (sec)	Must be a valid non-negative value	0.0005F	Period of Speed control
PWM Carrier Frequency (kHz)	Must be a valid non-negative value	20.0F	PWM Carrier Frequency
Decimation of Interrupt	Must be a valid positive integer.	0U	Decimation of Interrupt
Counts per Rotation	Must be a valid positive integer.	1200U	Encoder Counts per One Rotation
Counts for Angle Adjust	Must be a valid positive integer.	512U	Counts for Angle Adjust (as working time)
Zero speed counts	Must be a valid positive integer.	20000000U	Threshold counts to judge zero speed
Occupancy Time	Must be a valid non-negative value	0.30F	Occupancy time of carrier interrup
Carrier Time	Must be a valid non-negative value	0.000013F	Processing time of carrier interrupt
Process Time	Must be a valid non-negative value	0.000001F	Processing time of encoder interrupt
Highspeed Change Margin (rpm)	Must be a valid positive integer.	150U	Margin of toggle speed for high speed mode
LPF parameter for Highspeed Filter	Must be a valid non-negative value	0.1F	Highspeed mode speed LPF parameter
Counts to change speed	Must be a valid positive integer.	8U	Counts for mode change of position speed calculation
Speed Detection Type	• Use capture	Use capture width	Select Speed detection

	width • Use carrier interrupt	Type

Clock Configuration

Pin Configuration

Usage Notes

Limitations

Developers should be aware of the following limitations when using the motor angle and speed calculation with an encoder: all configurations should be set as positive values.

Examples

Basic Example

This is a basic example of minimal use of the motor angle and speed calculation with an encoder in an application.

```
void motor_sense_encoder_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSE_ENCODER_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Initialize motor with the encoder */
       (void) RM_MOTOR_SENSE_ENCODER_AngleAdjust(&g_mtr_angle0_ctrl);
 /* Perform cyclic encoder process*/
       (void) RM_MOTOR_SENSE_ENCODER_CyclicProcess(&g_mtr_angle0_ctrl);
 /* Calculate information with encoder signal input */
       (void) RM_MOTOR_SENSE_ENCODER_InternalCalculate(&g_mtr_angle0_ctrl);
 /* Get angle/speed data */
       (void) RM_MOTOR_SENSE_ENCODER_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
```

```
/* Get calculated component */
      (void) RM_MOTOR_SENSE_ENCODER_InfoGet(&g_mtr_angle0_ctrl, &temp_info);
   }
/* Reset the module */
   (void) RM_MOTOR_SENSE_ENCODER_Reset(&g_mtr_angle0_ctrl);
/* Close the module */
   (void) RM_MOTOR_SENSE_ENCODER_Close(&g_mtr_angle0_ctrl);
}
```

Enumerations

enum	motor_sense_encoder_loop_t
enum	motor_sense_encoder_mode_t
enum	motor_sense_encoder_speed_detect_type_t

Enumeration Type Documentation

◆ motor_sense_encoder_loop_t

enum motor_sense_encoder_loop_t	
Enumerator	
MOTOR_SENSE_ENCODER_LOOP_SPEED	Speed control mode.
MOTOR_SENSE_ENCODER_LOOP_POSITION	Position control mode.

◆ motor_sense_encoder_mode_t

enum motor_sense_encoder_mode_t	
Enumerator	
MOTOR_SENSE_ENCODER_MODE_INIT	Initialize mode (Start status)
MOTOR_SENSE_ENCODER_MODE_BOOT	Boot mode (Angle adjustment status)
MOTOR_SENSE_ENCODER_MODE_DRIVE	Drive mode (Normal work status)

◆ motor_sense_encoder_speed_detect_type_t

enum motor_sense_encoder_speed_detect_type_t	
Enumerator	
MOTOR_SENSE_ENCODER_SPEED_DETECT_WIDTH	Use captured width.
MOTOR_SENSE_ENCODER_SPEED_DETECT_CARRIER	Use carrier interrupt.

Function Documentation

◆ RM_MOTOR_SENSE_ENCODER_Open()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Open (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Opens and configures the Angle Encoder module. Implements motor_angle_api_t::open.

Return values

FSP_SUCCESS	Angle Encoder module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_SENSE_ENCODER_Close()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Close (motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Encoder module. Implements motor_angle_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_Reset()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Reset (motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Encoder module. Implements motor_angle_api_t::reset.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_InternalCalculate()

fsp_err_t RM_MOTOR_SENSE_ENCODER_InternalCalculate (motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_AngleSpeedGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_AngleSpeedGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err*)

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_AngleAdjust()

fsp_err_t RM_MOTOR_SENSE_ENCODER_AngleAdjust (motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_EncoderCyclic()

fsp_err_t RM_MOTOR_SENSE_ENCODER_EncoderCyclic (motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_InfoGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_InfoGet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info*)

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_ParameterUpdate()

fsp_err_t RM_MOTOR_SENSE_ENCODER_ParameterUpdate (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Update the parameters of Angle&Speed calculation with an encoder. Implements motor_angle_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data is update.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_CurrentSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_CurrentSet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage*)

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software currentSet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_SpeedSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_SpeedSet (motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed*)

Set Speed Information. Implements motor_angle_api_t::speedSet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software speedSet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet (motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi*)

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software flagPiCtrlSet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq*)

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software estimatedComponentGet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_CyclicProcess()

fsp_err_t RM_MOTOR_SENSE_ENCODER_CyclicProcess (motor_angle_ctrl_t *const *p_ctrl*)

Perform cyclic process. Implements motor_angle_api_t::cyclicProcess.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_SensorDataSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_SensorDataSet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_ad_data_t *const *p_ad_data*)

Set sensor data. Implements motor_angle_api_t::sensorDataSet.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

5.2.11.10 Motor Sensorless Vector Control (rm_motor_sensorless)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_SENSORLESS_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)
fsp_err_t	RM_MOTOR_SENSORLESS_Close (motor_ctrl_t *const p_ctrl) Disables specified Motor Sensorless Control block. Implements motor_api_t::close. More...
fsp_err_t	RM_MOTOR_SENSORLESS_Run (motor_ctrl_t *const p_ctrl) Run Motor (Start motor rotation). Implements motor_api_t::run. More...
fsp_err_t	RM_MOTOR_SENSORLESS_Stop (motor_ctrl_t *const p_ctrl) Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More...
fsp_err_t	RM_MOTOR_SENSORLESS_Reset (motor_ctrl_t *const p_ctrl) Reset Motor Sensorless Control block. Implements motor_api_t::reset . More...
fsp_err_t	RM_MOTOR_SENSORLESS_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error) Set error information. Implements motor_api_t::errorSet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm) Set speed reference[rpm]. Implements motor_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status) Get current control status. Implements motor_api_t::statusGet. More...

fsp_err_t	RM_MOTOR_SENSORLESS_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad) Get current rotor angle. Implements motor_api_t::angleGet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get rotational speed. Implements motor_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error) Check the occurunce of Error. Implements motor_api_t::errorCheck. More...
fsp_err_t	RM_MOTOR_SENSORLESS_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position) Set position reference. Implements motor_api_t::positionSet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag) Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function) Select function. Implements motor_api_t::functionSelect. More...

Detailed Description

Usual control of a SPM motor on RX MCUs. This module implements the Motor Sensorless Vector Control (rm_motor_sensorless).

Overview

The motor sensorless vector control is used to control a motor rotation in an application. This module is implemented with using SPM motor. User can start/stop motor rotation simply.

Features

The Motor Sensorless Module has below features.

- Start/Stop a motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

Target Hardware

The below figure shows an example of target hardware of this Motor Sensorless Module.

Block Diagram

The below figures show block diaram of sensorless vector motor control. The 1st shows as an open-loop state, 2nd as a PI feedback loop state.

Modulation

- Sine wave modulation The modulation factor "m" is defined as follows.
- Space vector modulation In vector control of a permanent magnet synchronous motor, generally, the desired voltage command value of each phase is generated sinusoidally. However, if the generated value is used as-is for the modulation wave for PWM generation, voltage utilization as applied to the motor (in terms of line voltage) is limited to a maximum of 86.7 percents with respect to inverter bus voltage. As such, as shown in the following expression, the average of the maximum and minimum values is calculated for the voltage command value of each phase, and the value obtained by subtracting the average from the voltage command value of each phase is used as the modulation wave. As a result, the maximum amplitude of the modulation wave is multiplied by (square root 3)/2, while voltage utilization becomes 100 percents and line voltage is unchanged.

State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

EVENT name	Occurrence factor
STOP	by user operation
RUN	by user operation
ERROR	when the system detects an error
RESET	by user operation

Flowchart

The below figures show flowcharts of motor sensorless module.

Configuration

Build Time Configurations for rm_motor_sensorless

The following build time configurations are defined in fsp_cfg/rm_motor_sensorless_cfg.h:

Configuration	Options	Default	Description
Parameter checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Sensorless Vector Control (rm_motor_sensorless)

This module can be added to the Stacks tab via New Stack > Motor > Motor Sensorless Vector Control (rm_motor_sensorless).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_sensorless0	Module name.
General > Limit of over current (A)	Must be a valid value	0.42F	Limit of over current.(Detection threshold)
General > Limit of over voltage (V)	Must be a valid value	28.0F	Limit of over voltage.(Detection threshold)
General > Limit of over speed (rpm)	Must be a valid value	3000.0F	Limit of over speed.(Detection threshold)
General > Limit of low voltage (V)	Must be a valid value	14.0F	Limit of low voltage.(Detection threshold)
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt.

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

Usage Notes

Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

Examples

Basic Example

This is a basic example of minimal use of the Motor Sensorless in an application.

```
void motor_sensorless_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSORLESS_Open(g_motor_sensorless0.p_ctrl,
g_motor_sensorless0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_SENSORLESS_SpeedSet(g_motor_sensorless0.p_ctrl,
DEF_SENSORLESS_TEST_OVSPD_LIM);
 /* Start motor rotation */
    (void) RM_MOTOR_SENSORLESS_Run(g_motor_sensorless0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_SENSORLESS_StatusGet(g_motor_sensorless0.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_SENSORLESS_AngleGet(g_motor_sensorless0.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_SENSORLESS_SpeedGet(g_motor_sensorless0.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_SENSORLESS_ErrorCheck(g_motor_sensorless0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_Stop(g_motor_sensorless0.p_ctrl);
 /* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_ErrorSet(g_motor_sensorless0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset Speed Control */
    (void) RM_MOTOR_SENSORLESS_Reset(g_motor_sensorless0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_SENSORLESS_Close(g_motor_sensorless0.p_ctrl);
}
```

Function Documentation

◆ RM_MOTOR_SENSORLESS_Open()

fsp_err_t RM_MOTOR_SENSORLESS_Open (motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg*)

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_SENSORLESS_Open(g_motor_sensorless0.p_ctrl,
g_motor_sensorless0.p_cfg);
```

Return values

FSP_SUCCESS	MOTOR successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

Note

◆ RM_MOTOR_SENSORLESS_Close()

fsp_err_t RM_MOTOR_SENSORLESS_Close (motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Sensorless Control block. Implements motor_api_t::close.

Example:

```
/* Close Speed Control */
   (void) RM_MOTOR_SENSORLESS_Close(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_Run()

fsp_err_t RM_MOTOR_SENSORLESS_Run (motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_SENSORLESS_Run(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_Stop()

fsp_err_t RM_MOTOR_SENSORLESS_Stop (motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_SENSORLESS_Stop(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_Reset()

fsp_err_t RM_MOTOR_SENSORLESS_Reset (motor_ctrl_t *const *p_ctrl*)

Reset Motor Sensorless Control block. Implements motor_api_t::reset.

Example:

```
/* Reset Speed Control */
   (void) RM_MOTOR_SENSORLESS_Reset(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_ErrorSet()

fsp_err_t RM_MOTOR_SENSORLESS_ErrorSet (motor_ctrl_t *const *p_ctrl*, motor_error_t const *error*)

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_SENSORLESS_ErrorSet(g_motor_sensorless0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_SpeedSet()

fsp_err_t RM_MOTOR_SENSORLESS_SpeedSet (motor_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_SENSORLESS_SpeedSet(g_motor_sensorless0.p_ctrl,
DEF_SENSORLESS_TEST_OVSPD_LIM);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_StatusGet()

fsp_err_t RM_MOTOR_SENSORLESS_StatusGet (motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status*)

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_SENSORLESS_StatusGet(g_motor_sensorless0.p_ctrl, &smpl_status);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_AngleGet()

fsp_err_t RM_MOTOR_SENSORLESS_AngleGet (motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad*)

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */

   (void) RM_MOTOR_SENSORLESS_AngleGet(g_motor_sensorless0.p_ctrl, &smpl_angle);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_SpeedGet()

fsp_err_t RM_MOTOR_SENSORLESS_SpeedGet (motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */

   (void) RM_MOTOR_SENSORLESS_SpeedGet(g_motor_sensorless0.p_ctrl, &smpl_speed);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_ErrorCheck()

fsp_err_t RM_MOTOR_SENSORLESS_ErrorCheck (motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error*)

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */

   (void) RM_MOTOR_SENSORLESS_ErrorCheck(g_motor_sensorless0.p_ctrl, &smpl_error);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_PositionSet()

fsp_err_t RM_MOTOR_SENSORLESS_PositionSet (motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position*)

Set position reference. Implements motor_api_t::positionSet.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

◆ RM_MOTOR_SENSORLESS_WaitStopFlagGet()

fsp_err_t RM_MOTOR_SENSORLESS_WaitStopFlagGet (motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag*)

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

◆ RM_MOTOR_SENSORLESS_FunctionSelect()

fsp_err_t RM_MOTOR_SENSORLESS_FunctionSelect (motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function*)

Select function. Implements motor_api_t::functionSelect.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

5.2.11.11 Motor Speed Controller (rm_motor_speed)

Modules » Motor

Functions

fsp_err_t	RM_MOTOR_SPEED_Open (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) Opens and configures the Motor Speed Module. Implements motor_speed_api_t::open. More...
fsp_err_t	RM_MOTOR_SPEED_Close (motor_speed_ctrl_t *const p_ctrl) Disables specified Motor Speed Module. Implements motor_speed_api_t::close. More...
fsp_err_t	RM_MOTOR_SPEED_Reset (motor_speed_ctrl_t *const p_ctrl) Reset the variables of Motor Speed Module. Implements motor_speed_api_t::reset. More...
fsp_err_t	RM_MOTOR_SPEED_Run (motor_speed_ctrl_t *const p_ctrl) Run(Start) the Motor Speed Control. Implements motor_speed_api_t::run. More...
fsp_err_t	RM_MOTOR_SPEED_SpeedReferenceSet (motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm) Set Speed Reference Data. Implements motor_speed_api_t::speedReferenceSet. More...

fsp_err_t	RM_MOTOR_SPEED_PositionReferenceSet (motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data) Set Position Reference Data. Implements motor_speed_api_t::positionReferenceSet. More...
fsp_err_t	RM_MOTOR_SPEED_ParameterSet (motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input) Set Input parameters. Implements motor_speed_api_t::parameterSet. More...
fsp_err_t	RM_MOTOR_SPEED_SpeedControl (motor_speed_ctrl_t *const p_ctrl) Calculates the d/q-axis current reference.(Main process of Speed Control) Implements motor_speed_api_t::speedControl. More...
fsp_err_t	RM_MOTOR_SPEED_ParameterGet (motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output) Get Speed Control Parameters. Implements motor_speed_api_t::parameterGet. More...
fsp_err_t	RM_MOTOR_SPEED_ParameterUpdate (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) Update the parameters of Speed Control Calculation. Implements motor_speed_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RX MCUs. This module implements the Motor speed Interface.

Overview

The motor speed is used to control the speed of motor rotation in an application. This module should be called cyclically in an application (e.g. in cyclic timer interrupt). This module caliculates d/q-axis current reference with input speed reference, current rotational speed, and d/q-axis current.

Features

The motor speed module has below features.

- Calculate d/q-axis electric current reference.
- Flux weakening process at high speed rotation.
- Open loop damping control when using sensorless type.
- Low pass filter of input rotational speed.
- Speed observer function when using encoder type.

Configuration

Build Time Configurations for rm_motor_speed

The following build time configurations are defined in fsp_cfg/rm_motor_speed_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Position Support	• Enabled • Disabled	Disabled	Enable position algorithm support.

Configurations for Motor > Motor Speed Controller (rm_motor_speed)

This module can be added to the Stacks tab via New Stack > Motor > Motor Speed Controller (rm_motor_speed).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_speed0	Module name.
General > Speed control period (sec)	Must be a valid non-negative value	0.0005F	Period of speed control function.
General > Step of speed climbing (rpm)	Must be a valid non-negative value	0.5F	Step of speed change at start of open-loop.
General > Maximum rotational speed (rpm)	Must be a valid non-negative value.	2650.0F	Maximum rotational speed (Limit speed).
General > Speed LPF omega	Must be a valid non-negative value	10.0F	Design parameter for speed LPF.
General > Limit of q-axis current (A)	Must be a valid non-negative value	0.42F	Limit of q-axis current.
General > Step of speed feedback at open-loop	Must be a valid non-negative value	0.20F	Step of speed feedback at open-loop.
General > Natural frequency	Must be a valid non-negative value.	100.0F	Natural frequency for disturbance speed observer.
General > Open-loop damping	• Disable • Enable	Enable	Select enable/disable Open-loop damping control.
General > Flux weakening	• Disable • Enable	Disable	Select enable/disable flux weakening control.
General > Torque compensation for sensorless transition	• Disable • Enable	Enable	Select enable/disable torque compensation for sensorless transition.

General > Speed observer	• Disable • Enable	Enable	Select enable/disable Speed observer process.
General > Selection of speed observer	• Normal • Disturbance	Normal	Select speed observer type.
General > Control method	• PID • IPD	PID	Select the control method [PID or IPD].
General > Control type	• Sensoreless • Encoder • Induction • Hall	Sensoreless	
Open-Loop > Step of d-axis current climbing	Must be a valid non-negative value	0.3F	Step of d-axis current climbing
Open-Loop > Step of d-axis current descending	Must be a valid non-negative value	0.3F	Step of d-axis current descending
Open-Loop > Step of q-axis current descending ratio	Must be a valid non-negative value	1.0F	Step of q-axis current descending ratio
Open-Loop > Reference of d-axis current	Must be a valid non-negative value	0.3F	Reference of d-axis current
Open-Loop > Threshold of speed control descending	Must be a valid value	600.0F	When rotational speed reaches this speed, d-axis current is controlled descending.
Open-Loop > Threshold of speed control climbing	Must be a valid value	500.0F	Until rotational speed reaches this speed, d-axis current is controlled climbing.
Open-Loop > Period between open-loop to BEMF (sec)	Must be a valid non-negative value	0.025F	Margin time between open-loop control changes to BEMF PI control.
Open-Loop > Phase error(degree) to decide sensor-less switch timing	Must be a valid value	10.0F	Phase error(degree) to decide sensor-less switch timing.
Design parameter > Speed PI loop omega	Must be a valid non-negative value	5.0F	Speed PI loop omega
Design parameter > Speed PI loop zeta	Must be a valid non-negative value	1.0F	Speed PI loop zeta
Design parameter > Estimated d-axis HPF omega	Must be a valid non-negative value	2.5F	HPF cutoff frequency for ed (Hz)

Design parameter > Open-loop damping zeta	Must be a valid non-negative value	1.0F	Damping ratio of open-loop damping control
Design parameter > Cutoff frequency of phase error LPF	Must be a valid non-negative value	10.0F	Cutoff frequency of phase error LPF
Design parameter > Speed observer omega	Must be a valid non-negative value	200.0F	Speed observer loop omega
Design parameter > Speed observer zeta	Must be a valid non-negative value	1.0F	Speed observer loop zeta
Motor Parameter > Pole pairs	Must be a valid integer.	2	Pole pairs
Motor Parameter > Resistance (ohm)	Must be a valid non-negative value	8.5F	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid non-negative value	0.0045F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid non-negative value	0.0045F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid non-negative value	0.02159F	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid non-negative value	0.0000028F	Rotor inertia
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at timer interrupt.
Interrupts > Input data	Name must be a valid C symbol	NULL	Structure for Speed control Input. If you set this content, Speed Control function read these data automatically. (No need to use Set API.)
Interrupts > Output data	Name must be a valid C symbol	NULL	Structure for Speed control Output. If you set this content, Speed Control function write need data automatically. (No need to use Get API.)

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the period of speed control with none-negative value.
- Set the limit of speed change step with none-negative value.
- Set the maximum speed with none-negative value.

Examples

Basic Example

This is a basic example of minimal use of the motor speed in an application.

```
void motor_speed_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SPEED_Open(g_motor_speed0.p_ctrl, g_motor_speed0.p_cfg);
    handle_error(err);
 /* Set speed reference before get current reference */
    (void) RM_MOTOR_SPEED_SpeedReferenceSet(g_motor_speed0.p_ctrl, 104.72F);
 /* Set position reference before get current reference
  * (Basically Exclusive to SpeedReferenceSet. This is only sample,) */
    (void) RM_MOTOR_SPEED_PositionReferenceSet(g_motor_speed0.p_ctrl,
&g_posref_sample);
 /* Basically run this module at cyclic interrupt (e.g. AGT timer).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Set input parameter data before get current reference */
       (void) RM_MOTOR_SPEED_ParameterSet(g_motor_speed0.p_ctrl,
&g_test_speed_input);
 /* Activate Speed Process */
```

```
      (void) RM_MOTOR_SPEED_Run(g_motor_speed0.p_ctrl);
 /* Perform Speed Control Process */
      (void) RM_MOTOR_SPEED_SpeedControl(g_motor_speed0.p_ctrl);
 /* Get output parameters */
      (void) RM_MOTOR_SPEED_ParameterGet(g_motor_speed0.p_ctrl,
&g_test_speed_output);
 //

 /* Update parameters */
      (void) RM_MOTOR_SPEED_ParameterUpdate(g_motor_speed0.p_ctrl,
&g_motor_speed0.p_cfg);
    }
 /* Reset Speed Control */
    (void) RM_MOTOR_SPEED_Reset(g_motor_speed0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_SPEED_Close(g_motor_speed0.p_ctrl);
}
```

Enumerations

enum	motor_speed_control_type_t
enum	motor_speed_openloop_damping_t
enum	motor_speed_flux_weaken_t
enum	motor_speed_less_switch_t
enum	motor_speed_observer_switch_t
enum	motor_speed_observer_select_t
enum	motor_speed_ctrl_status_t

Enumeration Type Documentation

◆ motor_speed_control_type_t

enum motor_speed_control_type_t	
Enumerator	
MOTOR_SPEED_CONTROL_TYPE_SENSORLESS	Sensorless type.
MOTOR_SPEED_CONTROL_TYPE_ENCODER	Encoder type.
MOTOR_SPEED_CONTROL_TYPE_HALL	Hall type.
MOTOR_SPEED_CONTROL_TYPE_INDUCTION	Induction type.

◆ motor_speed_openloop_damping_t

enum motor_speed_openloop_damping_t	
Enumerator	
MOTOR_SPEED_OPENLOOP_DAMPING_DISABLE	Disable openloop damping.
MOTOR_SPEED_OPENLOOP_DAMPING_ENABLE	Enable openloop damping.

◆ motor_speed_flux_weaken_t

enum motor_speed_flux_weaken_t	
Enumerator	
MOTOR_SPEED_FLUX_WEAKEN_DISABLE	Disable flux-weakening control.
MOTOR_SPEED_FLUX_WEAKEN_ENABLE	Enable flux-weakening control.

◆ motor_speed_less_switch_t

enum motor_speed_less_switch_t	
Enumerator	
MOTOR_SPEED_LESS_SWITCH_DISABLE	Disable soft switching between open-loop mode and normal field oriented control mode.
MOTOR_SPEED_LESS_SWITCH_ENABLE	Enable soft switching between open-loop mode and normal field oriented control mode.

◆ motor_speed_observer_switch_t

enum motor_speed_observer_switch_t	
Enumerator	
MOTOR_SPEED_OBSERVER_SWITCH_DISABLE	Disable speed observer.
MOTOR_SPEED_OBSERVER_SWITCH_ENABLE	Enable speed observer.

◆ motor_speed_observer_select_t

enum motor_speed_observer_select_t	
Enumerator	
MOTOR_SPEED_OBSERVER_SELECT_NORMAL	Normal speed observer.
MOTOR_SPEED_OBSERVER_SELECT_DISTURBANCE	Disturbance speed observer.

◆ motor_speed_ctrl_status_t

enum motor_speed_ctrl_status_t	
Enumerator	
MOTOR_SPEED_CTRL_STATUS_INIT	Speed control status is INIT.
MOTOR_SPEED_CTRL_STATUS_BOOT	Speed control status is BOOT.
MOTOR_SPEED_CTRL_STATUS_RUN	Speed control status is RUN.

Function Documentation

◆ RM_MOTOR_SPEED_Open()

fsp_err_t RM_MOTOR_SPEED_Open (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_cfg_t const *const *p_cfg*)

Opens and configures the Motor Speed Module. Implements motor_speed_api_t::open.

Return values

FSP_SUCCESS	Motor Speed Module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_SPEED_Close()

fsp_err_t RM_MOTOR_SPEED_Close (motor_speed_ctrl_t *const *p_ctrl*)

Disables specified Motor Speed Module. Implements motor_speed_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_Reset()

fsp_err_t RM_MOTOR_SPEED_Reset (motor_speed_ctrl_t *const *p_ctrl*)

Reset the variables of Motor Speed Module. Implements motor_speed_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_Run()

fsp_err_t RM_MOTOR_SPEED_Run (motor_speed_ctrl_t *const *p_ctrl*)

Run(Start) the Motor Speed Control. Implements motor_speed_api_t::run.

Return values

FSP_SUCCESS	Successfully start.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_SpeedReferenceSet()

fsp_err_t RM_MOTOR_SPEED_SpeedReferenceSet (motor_speed_ctrl_t *const *p_ctrl*, float const *speed_reference_rpm*)

Set Speed Reference Data. Implements motor_speed_api_t::speedReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_PositionReferenceSet()

fsp_err_t RM_MOTOR_SPEED_PositionReferenceSet (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position_data*)

Set Position Reference Data. Implements motor_speed_api_t::positionReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input structure pointer is NULL.

◆ RM_MOTOR_SPEED_ParameterSet()

fsp_err_t RM_MOTOR_SPEED_ParameterSet (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_input_t const *const *p_st_input*)

Set Input parameters. Implements motor_speed_api_t::parameterSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_SPEED_SpeedControl()

fsp_err_t RM_MOTOR_SPEED_SpeedControl (motor_speed_ctrl_t *const *p_ctrl*)

Calculates the d/q-axis current reference.(Main process of Speed Control) Implements motor_speed_api_t::speedControl.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_ParameterGet()

fsp_err_t RM_MOTOR_SPEED_ParameterGet (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_output_t *const *p_st_output*)

Get Speed Control Parameters. Implements motor_speed_api_t::parameterGet.

Return values

FSP_SUCCESS	Successfully the flag is gotten.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_SPEED_ParameterUpdate()

fsp_err_t RM_MOTOR_SPEED_ParameterUpdate (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_cfg_t const *const *p_cfg*)

Update the parameters of Speed Control Calculation. Implements motor_speed_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

5.2.12 Networking

Modules

Detailed Description

Networking Modules.

Modules

	Ethercat (r_ethercat_phy) The Ethercat PHY module (r_ethercat_phy) provides an API for standard Ethercat PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface.
	Ethernet Generic PTP Timer (r_gptp) Driver for the Ethernet peripheral on RX MCUs. This module implements the GPTP Interface.
	Layer3 Ethernet Switch (r_layer3_switch) The LAYER3 SWITCH module (r_layer3_switch) provides an API for standard Ethernet communications applications that use the ESWM peripheral. It implements the Ethernet Switch Interface.
	Ethernet (r_rmac) Driver for the Ethernet peripheral on RX MCUs. This module

	implements the Ethernet Interface.
	Ethernet (r_rmac_phy) The RMAC PHY module (r_rmac_phy) provides an API for standard Ethernet PHY communications applications that use the RMAC peripheral. It implements the Ethernet PHY Interface.
	AWS PKCS11 PAL on LittleFS (rm_aws_pkcs11_pal_littlefs) PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS.
	EtherCAT SSC Port (rm_ethercat_ssc_port) Middleware for EtherCAT Slave Stack Code (SSC) on RX microprocessors. This module implements the EtherCAT SSC Interface.
	AWS coreHTTP This module provides the AWS coreHTTP library.
	AWS MQTT This module provides the AWS MQTT integration documentation.
	FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp) Middleware for using TCP on RX MCUs.
	gPTP Ethernet Switch Module(rm_gptp_eswm)
	gPTP Baremetal Porting Layer (rm_gptp_sys_baremetal)
	gPTP FreeRTOS Porting Layer (rm_gptp_sys_freertos)
	lwIP HTTP Server This module provides the lwIP HTTP Server integration documentation.
	lwIP Iperf This module provides the lwIP Iperf integration documentation.

	lwIP MQTT This module provides the lwIP MQTT integration documentation.
	lwIP NETBIOSNS This module implements the lwIP NETBIOSNS integration documentation.
	lwIP SMTP This module provides the lwIP SMTP integration documentation.
	lwIP SNTP This module provides the lwIP SNTP integration documentation.
	lwIP Ethernet Driver (rm_lwip_ether)
	lwIP Baremetal Porting Layer (rm_lwip_sys_baremetal)
	lwIP FreeRTOS Porting Layer (rm_lwip_sys_freertos)
	NetX Duo Ethernet Driver (rm_netxduo_ether)

5.2.12.1 Ethercat (r_ethercat_phy)

Modules » Networking

Functions

fsp_err_t	R_ETHERCAT_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) Resets Ethercat PHY device. Implements ether_phy_api_t::open. More...
fsp_err_t	R_ETHERCAT_PHY_Close (ether_phy_ctrl_t *const p_ctrl) Close Ethercat PHY device. Implements ether_phy_api_t::close. More...
fsp_err_t	R_ETHERCAT_PHY_ChipInit (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) Initialize Ethercat PHY device. Implements ether_phy_api_t::chipInit.

	More...
fsp_err_t	R_ETHERCAT_PHY_Read (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data) Read data from register of PHY-LSI . Implements ether_phy_api_t::read. More...
fsp_err_t	R_ETHERCAT_PHY_Write (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data) Write data to register of PHY-LSI . Implements ether_phy_api_t::write. More...
fsp_err_t	R_ETHERCAT_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl) Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More...
fsp_err_t	R_ETHERCAT_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause) Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More...
fsp_err_t	R_ETHERCAT_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl) Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet. More...

Detailed Description

The Ethercat PHY module (r_ethercat_phy) provides an API for standard Ethercat PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface.

Overview

The Ethercat PHY module is used to setup and manage an external Ethercat PHY device for use with the on-chip Ethercat Slave Controller (ESC) peripheral. It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

Features

The Ethercat PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Configuration

Build Time Configurations for r_ethercat_phy

The following build time configurations are defined in fsp_cfg/r_ethercat_phy_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
VSC8541 Target	• Disabled • Enabled	Disabled	Select whether to use VSC8541 Phy LSI or not.
KSZ8081MNX Target	• Disabled • Enabled	Disabled	Select whether to use KSZ8081MNX Phy LSI or not.
User Own Target	• Disabled • Enabled	Disabled	Select whether to use User own Phy LSI or not.

Configurations for Networking > Ethernet (r_ethercat_phy)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_ethercat_phy).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_ethercat_phy0	Module name.
Channel	• 0 • 1	0	Select the Ethernet controller channel number.
PHY-LSI Address	Specify a value between 0 and 31.	0	Specify the address of the PHY-LSI used.
PHY-LSI Reset Completion Timeout	Specify a value between 0x1 and 0xFFFFFFFF.	0x00020000	Specify the number of times to read the PHY-LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used.
Select MII type	MII	module.driver.ether_phy.mii_type.rmii	Specify to use MII.
PHY-LSI Type	• Kit Component	Kit Component	Select the Phy LSI

	• DEFAULT • VSC8541 • KSZ8081MNX • User own PHY		target. Selecting 'Kit Component' will automatically choose the correct option when using a Renesas development board.
Target Init Function	Name must be a valid C symbol	NULL	Set the initial function of the PHY-LSI, When using your own PHY-LSI.
Target Link Partner Ability Get Function	Name must be a valid C symbol	NULL	Set the link partner ability get function of the PHY-LSI, When using your own PHY-LSI.
MII Register Access Wait-time	Specify a value between 0x1 and 0x7FFFFFFF.	8	Specify the bit timing for MII register accesses during PHY initialization. This value should be adjusted experimentally based on the PHY-LSI used.
Flow Control	• Disable • Enable	Disable	Select whether to enable or disable flow control.

Usage Notes

Note

See the example below for details on how to initialize the Ethercat PHY module.

Accessing the MII and RMII Registers

Use the PIR register to access the MII and RMII registers in the PHY-LSI. Serial data in the MII and RMII management frame format is transmitted and received through the ET0_MDC and ET0_MDIO pins controlled by software.

MII and RMII management frame format

The below table lists the MII and RMII management frame formats.

Access type	MII and RMII management frame							
Item	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
Number of bits	32	2	2	5	5	2	16	1
Read	1...1	01	10	00001	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z

Write	1...1	01	01	00001	RRRRR	10	DDDDDDDDDDDDDDDD	Z

Note

- PRE (preamble): Send 32 consecutive 1s.
- ST (start of frame): Send 01b.
- OP (operation code): Send 10b for read or 01b for write.
- PHYAD (PHY address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the PHY-LSI address is 1, send 00001b.
- REGAD (register address): One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.
- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.
Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).
(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)
- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.
- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write
operation (Z is output). No control is required, because a bus was already released during a read operation.

Limitations

- The r_ethercat_phy module may need to be customized for PHY devices other than the ones currently supported. Use the existing code as a starting point for creating a custom implementation. Supported PHY devices are as follows.
  - VSC8541
  - KSZ8081MNX

Examples

ETHERCAT PHY Basic Example

This is a basic example of minimal use of the ETHERCAT PHY in an application.

```
void ethercat_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_ethercat_phy0_ctrl.open   = 0U;
    g_ethercat_phy0_cfg.channel = 0;
 /* Initializes the module. */
    err = R_ETHERCAT_PHY_Open(&g_ethercat_phy0_ctrl, &g_ethercat_phy0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_ETHERCAT_PHY_StartAutoNegotiate(&g_ethercat_phy0_ctrl);
```

```
    assert(FSP_SUCCESS == err);
 /* Polling until link is established. */
 while (FSP_SUCCESS != R_ETHERCAT_PHY_LinkStatusGet(&g_ethercat_phy0_ctrl))
    {
 /* Do nothing */
    }
 /* Get link partner ability from phy interface. */
    err = R_ETHERCAT_PHY_LinkPartnerAbilityGet(&g_ethercat_phy0_ctrl,
                                               &g_ethercat_phy0_line_speed_duplex,
                                               &g_ethercat_phy0_local_pause,
                                               &g_ethercat_phy0_partner_pause);
    assert(FSP_SUCCESS == err);
 /* Check current link status. */
    err = R_ETHERCAT_PHY_LinkStatusGet(&g_ethercat_phy0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	ethercat_phy_instance_ctrl_t
struct	ethercat_phy_extended_cfg_t

Enumerations

enum	ethercat_phy_interface_status_t

Data Structure Documentation

◆ ethercat_phy_instance_ctrl_t

struct ethercat_phy_instance_ctrl_t		
ETHER PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called.		
Data Fields		
uint32_t	open	Used to determine if the channel is configured.
ether_phy_cfg_t const *	p_ether_phy_cfg	Pointer to initial configurations.
uint32_t	local_advertise	Capabilities bitmap for local advertising.
ethercat_phy_interface_status_t	interface_status	Initialized status of ETHER PHY

		interface.

◆ ethercat_phy_extended_cfg_t

struct ethercat_phy_extended_cfg_t	
ETHERCAT PHY extended configuration.	
Data Fields	
void(*	p_target_init)(ethercat_phy_instance_ctrl_t *p_instance_ctrl)
	Pointer to callback that is called to initialize the target.
bool(*	p_target_link_partner_ability_get)(ethercat_phy_instance_ctrl_t *p_instance_ctrl, uint32_t line_speed_duplex)
	Pointer to callback that is called to get the link partner ability.

Enumeration Type Documentation

◆ ethercat_phy_interface_status_t

enum ethercat_phy_interface_status_t	
Initialization state for read/write	
Enumerator	
ETHERCAT_PHY_INTERFACE_STATUS_UNINITIALIZED	ETHER PHY interface is uninitialized.
ETHERCAT_PHY_INTERFACE_STATUS_INITIALIZED	ETHER PHY interface is initialized.

Function Documentation

◆ R_ETHERCAT_PHY_Open()

fsp_err_t R_ETHERCAT_PHY_Open (ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg*)

Resets Ethercat PHY device. Implements ether_phy_api_t::open.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block or configuration structure is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_INVALID_CHANNEL	Invalid channel number is given.
FSP_ERR_INVALID_POINTER	Pointer to p_cfg is NULL.

◆ R_ETHERCAT_PHY_Close()

fsp_err_t R_ETHERCAT_PHY_Close (ether_phy_ctrl_t *const *p_ctrl*)

Close Ethercat PHY device. Implements ether_phy_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_ETHERCAT_PHY_ChipInit()

fsp_err_t R_ETHERCAT_PHY_ChipInit (ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg*)

Initialize Ethercat PHY device. Implements ether_phy_api_t::chipInit.

Return values

FSP_SUCCESS	PHY device initialized successfully.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block or configuration structure is NULL.
FSP_ERR_INVALID_POINTER	Pointer to p_cfg is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_TIMEOUT	PHY-LSI Reset wait timeout.

◆ R_ETHERCAT_PHY_Read()

fsp_err_t R_ETHERCAT_PHY_Read (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *const *p_data*)

Read data from register of PHY-LSI . Implements ether_phy_api_t::read.

Return values

FSP_SUCCESS	ETHERCAT_PHY successfully read data.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block is NULL.
FSP_ERR_INVALID_POINTER	Pointer to read buffer is NULL.
FSP_ERR_INVALID_ARGUMENT	Address is not a valid size
FSP_ERR_INVALID_MODE	Command is invalid.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized

◆ R_ETHERCAT_PHY_Write()

fsp_err_t R_ETHERCAT_PHY_Write (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *data*)

Write data to register of PHY-LSI . Implements ether_phy_api_t::write.

Return values

FSP_SUCCESS	ETHERCAT_PHY successfully write data.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block is NULL.
FSP_ERR_INVALID_ARGUMENT	Address or data is not a valid size
FSP_ERR_INVALID_MODE	Command is invalid.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized

◆ R_ETHERCAT_PHY_StartAutoNegotiate()

fsp_err_t R_ETHERCAT_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

Return values

FSP_SUCCESS	ETHERCAT_PHY successfully starts auto-negotiate.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_INVALID_ARGUMENT	Register address is incorrect
FSP_ERR_INVALID_MODE	Command is invalid.
FSP_ERR_INVALID_POINTER	Pointer to read buffer is NULL.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized

◆ R_ETHERCAT_PHY_LinkPartnerAbilityGet()

fsp_err_t R_ETHERCAT_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause*)

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

Return values

FSP_SUCCESS	ETHERCAT_PHY successfully get link partner ability.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block is NULL.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY-LSI is not link up.
FSP_ERR_ETHER_PHY_NOT_READY	The auto-negotiation isn't completed
FSP_ERR_INVALID_ARGUMENT	Status register address is incorrect
FSP_ERR_INVALID_MODE	Command is invalid.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized

◆ R_ETHERCAT_PHY_LinkStatusGet()

fsp_err_t R_ETHERCAT_PHY_LinkStatusGet (ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

Return values

FSP_SUCCESS	ETHERCAT_PHY successfully get link partner ability.
FSP_ERR_ASSERTION	Pointer to ETHERCAT_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY-LSI is not link up.
FSP_ERR_INVALID_ARGUMENT	Status register address is incorrect
FSP_ERR_INVALID_MODE	Command is invalid.
FSP_ERR_INVALID_POINTER	Pointer to read buffer is NULL.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized

5.2.12.2 Ethernet Generic PTP Timer (r_gptp)

Modules » Networking

Functions

fsp_err_t	R_GPTP_Open (gptp_ctrl_t *const p_ctrl, gptp_cfg_t const *const p_cfg)
fsp_err_t	R_GPTP_Close (gptp_ctrl_t *const p_ctrl)
fsp_err_t	R_GPTP_TimerCfg (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_cfg_t const *const p_timer_cfg)
fsp_err_t	R_GPTP_Start (gptp_ctrl_t *const p_ctrl, uint8_t timer)
fsp_err_t	R_GPTP_Stop (gptp_ctrl_t *const p_ctrl, uint8_t timer)
fsp_err_t	R_GPTP_TimerValueGet (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_value_t *const p_timer_value)
fsp_err_t	R_GPTP_TimerOffsetSet (gptp_ctrl_t *const p_ctrl, uint8_t timer, int64_t offset)
fsp_err_t	R_GPTP_TimerRateSet (gptp_ctrl_t *const p_ctrl, uint8_t timer, uint32_t rate)
fsp_err_t	R_GPTP_PulseGeneratorSet (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_pulse_generator_t *p_pulse)
fsp_err_t	R_GPTP_CallbackSet (gptp_ctrl_t *const p_ctrl, void(*p_callback)(gptp_callback_args_t *), void *const p_context, gptp_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the Ethernet peripheral on RX MCUs. This module implements the GPTP Interface.

Overview

Ethernet Generic PTP timer (GPTP) enables accurate synchronization of the clock in the control system.

This module supports two GPTP timer.

Features

The GPTP module has the following features:

- Supports start and stop gPTP timer.
- Supports set timer offset.
- Supports set timer rate.
- Supports set pulse generator.

Configuration

Build Time Configurations for r_gptp

The following build time configurations are defined in fsp_cfg/r_gptp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Networking > Ethernet Generic PTP Timer (r_gptp)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet Generic PTP Timer (r_gptp).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_gptp0	Module name.
Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs

Clock Configuration

The GPTP clock is based on the ESWCLK frequency. You can set the PTPTIVCt (t = 0 or 1) register to timer increment value according to ESWCLK frequency.

Usage Notes

Examples

This is a basic example of minimal use of the GPTP in an application.

```
void gptp_basic_example (void)
{
 fsp_err_t       err       = FSP_SUCCESS;
    uint8_t          timer     = GPTP_EXAMPLE_TIMER_NUMBER;
 gptp_timer_cfg_t timer_cfg = {.clock_period = GPTP_EXAMPLE_CLOCK_PERIOD};
    int64_t          offset     = GPTP_EXAMPLE_OFFSET;
    uint32_t         rate       = GPTP_EXAMPLE_RATE;
 gptp_timer_value_t timer_value =
```

```
    {
        .time_sec_upper = 0,
        .time_sec_lower = 0,
        .time_nsec      = 0,
    };
 gptp_pulsed_output_cfg_t pulsed_output_cfg0 =
    {
        .pulse_num       = 0,
        .start_sec_upper = 0,
        .start_sec_lower = 0,
        .start_ns         = 0,
        .wide             = 0,
        .period_sec_upper = 0,
        .period_sec_lower = 0,
        .period_ns       = 0,
    };
 gptp_pulsed_output_cfg_t pulsed_output_cfg1 =
    {
        .pulse_num       = 1,
        .start_sec_upper = 0,
        .start_sec_lower = 0,
        .start_ns         = 0,
        .wide             = 0,
        .period_sec_upper = 0,
        .period_sec_lower = 0,
        .period_ns       = 0,
    };
#if (BSP_FEATURE_ESWM_GPTP_PULSE_GENERATOR_NUM == 4)
 gptp_pulsed_output_cfg_t pulsed_output_cfg2 =
    {
        .pulse_num       = 2,
        .start_sec_upper = 0,
        .start_sec_lower = 0,
        .start_ns         = 0,
```

```
        .wide              = 0,
        .period_sec_upper = 0,
        .period_sec_lower = 0,
        .period_ns        = 0,
    };
 gptp_pulsed_output_cfg_t pulsed_output_cfg3 =
    {
        .pulse_num        = 3,
        .start_sec_upper = 0,
        .start_sec_lower = 0,
        .start_ns          = 0,
        .wide              = 0,
        .period_sec_upper = 0,
        .period_sec_lower = 0,
        .period_ns        = 0,
    };
#endif
 gptp_pulse_generator_t pulse_generator =
    {
        .p_pulsed_output_cfg_list =
#if (BSP_FEATURE_ESWM_GPTP_PULSE_GENERATOR_NUM == 4)
      {&pulsed_output_cfg0, &pulsed_output_cfg1, &pulsed_output_cfg2,
&pulsed_output_cfg3},
#else
      {&pulsed_output_cfg0, &pulsed_output_cfg1},
#endif
    };
 /* Open the gptp instance with initial configuration. */
    err = R_GPTP_Open(&g_gptp0_ctrl, &g_gptp0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set configuration of gptp timer. */
    err = R_GPTP_TimerCfg(&g_gptp0_ctrl, timer, &timer_cfg);
    assert(FSP_SUCCESS == err);
```

```
/* Start gptp timer. */
   err = R_GPTP_Start(&g_gptp0_ctrl, timer);
   assert(FSP_SUCCESS == err);
/* Get value of gptp timer. */
   err = R_GPTP_TimerValueGet(&g_gptp0_ctrl, timer, &timer_value);
   assert(FSP_SUCCESS == err);
/* Set offset of gptp timer. */
   err = R_GPTP_TimerOffsetSet(&g_gptp0_ctrl, timer, offset);
   assert(FSP_SUCCESS == err);
/* Set rate of gptp timer. */
   err = R_GPTP_TimerRateSet(&g_gptp0_ctrl, timer, rate);
   assert(FSP_SUCCESS == err);
/* Set pulse generator of gptp timer. */
   err = R_GPTP_PulseGeneratorSet(&g_gptp0_ctrl, timer, &pulse_generator);
   assert(FSP_SUCCESS == err);
/* Stop gptp timer. */
   err = R_GPTP_Stop(&g_gptp0_ctrl, timer);
   assert(FSP_SUCCESS == err);
/* Close the gptp instance. */
   err = R_GPTP_Close(&g_gptp0_ctrl);
   assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	gptp_pulsed_output_cfg_t
struct	gptp_pulse_generator_t
struct	gptp_instance_ctrl_t

Data Structure Documentation

◆ gptp_pulsed_output_cfg_t

struct gptp_pulsed_output_cfg_t		
Pulsed output configuration.		
Data Fields		
uint8_t	pulse_num	Pulse generator number.

uint16_t	start_sec_upper	Pulse start second(upper 16 bit).
uint32_t	start_sec_lower	Pulse start second(lower 32 bit).
uint32_t	start_ns	Pulse start nanosecond.
uint16_t	wide	Pulse width.
uint16_t	period_sec_upper	Pulse period second(upper 16 bit).
uint32_t	period_sec_lower	Pulse period second(lower 32 bit).
uint32_t	period_ns	Pulse period nanosecond.

◆ gptp_pulse_generator_t

struct gptp_pulse_generator_t		
Pulse generator.		
Data Fields		
gptp_pulsed_output_cfg_t *	p_pulsed_output_cfg_list[BSP_FEATURE_ESWM_GPTP_PULSE_GENERATOR_NUM]	List of pointer to pulsed output configuration.

◆ gptp_instance_ctrl_t

struct gptp_instance_ctrl_t	
GPTP control block. DO NOT INITIALIZE. Initialization occurs when gptp_api_t::open is called.	
Data Fields	
uint32_t	open
	Indicates whether the open() API has been successfully called.
gptp_cfg_t const *	p_cfg
	Pointer to initial configurations.
R_GPTP_Type *	p_reg_gptp
	Pointer to GPTP timer register.
void(*	p_callback)(gptp_callback_args_t *)
	Pointer to callback that is called.

gptp_callback_args_t *	p_callback_memory
	Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory.
void const *	p_context
	Pointer to context to be passed into callback function.

Function Documentation

◆ R_GPTP_Open()

fsp_err_t R_GPTP_Open (gptp_ctrl_t *const *p_ctrl*, gptp_cfg_t const *const *p_cfg*)

Initializes the gptp module and applies configurations.

Return values

FSP_SUCCESS	Module opened successfully.
FSP_ERR_ASSERTION	Pointer to GPTP control block or configuration structure is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.

◆ R_GPTP_Close()

fsp_err_t R_GPTP_Close (gptp_ctrl_t *const *p_ctrl*)

Close the gptp module.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Pointer to GPTP control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_GPTP_TimerCfg()

fsp_err_t R_GPTP_TimerCfg (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, gptp_timer_cfg_t const *const *p_timer_cfg*)

Configures the gptp timer parameters.

Return values

FSP_SUCCESS	Command successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Invalid timer id.
FSP_ERR_INVALID_POINTER	Invalid poiter to the timer cfg.

◆ R_GPTP_Start()

fsp_err_t R_GPTP_Start (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*)

Starts gptp timer.

Return values

FSP_SUCCESS	Timer started..
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Invalid timer id.

◆ R_GPTP_Stop()

fsp_err_t R_GPTP_Stop (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*)

Stops gptp timer.

Return values

FSP_SUCCESS	Timer stopped.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Invalid timer id.

◆ R_GPTP_TimerValueGet()

fsp_err_t R_GPTP_TimerValueGet (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, gptp_timer_value_t *const *p_timer_value*)

Gets the current time value to the timer with the specified gptp timer number.

Return values

FSP_SUCCESS	Sec upper, sec lower, and nano sec srored in p_timestamp.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.

◆ R_GPTP_TimerOffsetSet()

fsp_err_t R_GPTP_TimerOffsetSet (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, int64_t *offset*)

Sets the offset correction for the specified gptp timer number.

Return values

FSP_SUCCESS	Time offset set successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.

◆ R_GPTP_TimerRateSet()

fsp_err_t R_GPTP_TimerRateSet (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, uint32_t *rate*)

Sets clock rate correction for the specified timer.

Return values

FSP_SUCCESS	Time rate set successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.

◆ R_GPTP_PulseGeneratorSet()

fsp_err_t R_GPTP_PulseGeneratorSet (gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, gptp_pulse_generator_t * *p_pulse*)

Sets pulse generator.

Return values

FSP_SUCCESS	Pulse generator set successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.

◆ R_GPTP_CallbackSet()

fsp_err_t R_GPTP_CallbackSet (gptp_ctrl_t *const *p_ctrl*, void(*)(gptp_callback_args_t *) *p_callback*, void *const *p_context*, gptp_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements gptp_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

5.2.12.3 Layer3 Ethernet Switch (r_layer3_switch)

Modules » Networking

Functions

fsp_err_t	R_LAYER3_SWITCH_Open (ether_switch_ctrl_t *const p_ctrl, ether_switch_cfg_t const *const p_cfg)
fsp_err_t	R_LAYER3_SWITCH_Close (ether_switch_ctrl_t *const p_ctrl) Disables interrupts and stop module. Implements

	ether_switch_api_t::close. More...
fsp_err_t	R_LAYER3_SWITCH_CreateDescriptorQueue (ether_switch_ctrl_t *const p_ctrl, uint32_t *const p_queue_index, const layer3_switch_descriptor_queue_cfg_t *const p_queue_cfg) Create a new descriptor queue and set it to LINKFIX table. This function must be called before calling R_LAYER3_SWITCH_SetDescriptor and R_LAYER3_SWITCH_GetDescriptor. More...
fsp_err_t	R_LAYER3_SWITCH_SetDescriptor (ether_switch_ctrl_t *const p_ctrl, uint32_t queue_index, layer3_switch_descriptor_t const *const p_descriptor) Set descriptor data to a target descriptor. More...
fsp_err_t	R_LAYER3_SWITCH_GetDescriptor (ether_switch_ctrl_t *const p_ctrl, uint32_t queue_index, layer3_switch_descriptor_t *const p_descriptor) Get descriptor data from a target descriptor. More...
fsp_err_t	R_LAYER3_SWITCH_StartDescriptorQueue (ether_switch_ctrl_t *const p_ctrl, uint32_t queue_index) Reload and enable a descriptor queue. In a TX descriptor queue, the queue start transmission. In a RX descriptor queue, the queue start reception. More...
fsp_err_t	R_LAYER3_SWITCH_CallbackSet (ether_switch_ctrl_t *const p_ctrl, void(*p_callback)(ether_switch_callback_args_t *), void *const p_context, ether_switch_callback_args_t *const p_callback_memory)
fsp_err_t	R_LAYER3_SWITCH_ConfigurePort (ether_switch_ctrl_t *const p_ctrl, uint8_t port, layer3_switch_port_cfg_t *p_port_cfg)
fsp_err_t	R_LAYER3_SWITCH_AddTableEntry (ether_switch_ctrl_t *const p_ctrl, layer3_switch_frame_filter_t const *const p_target_frame, layer3_switch_table_entry_cfg_t const *const p_entry_cfg)
fsp_err_t	R_LAYER3_SWITCH_SearchTableEntry (ether_switch_ctrl_t *const p_ctrl, layer3_switch_frame_filter_t const *const p_target_frame, layer3_switch_table_entry_cfg_t *const p_entry_cfg)
fsp_err_t	R_LAYER3_SWITCH_ConfigureTable (ether_switch_ctrl_t *const p_ctrl, layer3_switch_table_cfg_t const *const p_table_cfg)
fsp_err_t	R_LAYER3_SWITCH_GetTable (ether_switch_ctrl_t *const p_ctrl,

	layer3_switch_table_t *const p_table)
fsp_err_t	R_LAYER3_SWITCH_ConfigureTAS (ether_switch_ctrl_t *const p_ctrl, uint8_t port, layer3_switch_tas_cfg_t *p_tas_cfg)
fsp_err_t	R_LAYER3_SWITCH_EnableTAS (ether_switch_ctrl_t *const p_ctrl, uint8_t port)
fsp_err_t	R_LAYER3_SWITCH_PsfpClearErrorStatus (ether_switch_ctrl_t *const p_ctrl, layer3_switch_psfp_error_status_bitmask_t bitmasks)

Detailed Description

The LAYER3 SWITCH module (r_layer3_switch) provides an API for standard Ethernet communications applications that use the ESWM peripheral. It implements the Ethernet Switch Interface.

Overview

This module performs Ethernet frame transmission and reception using Ethernet Switch Module.

The following figure shows which IPs are used by the r_layer3_switch module.

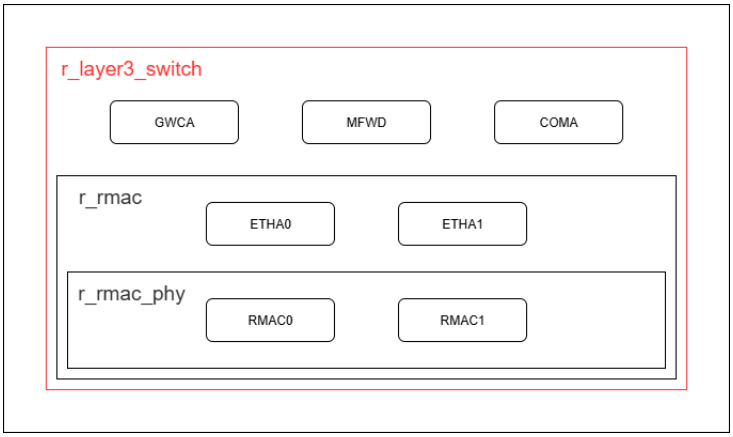


Figure 141: ESWM block diagram

Features

The layer3 switch module supports the following features:

- Transmit/receive data using a descriptor.
- Timestamp feature
- Optional callback function when transmit/receive complete or an error occurs
- Port based Forwarding [MFWD]
- Direct Forwarding [MFWD]
- L2/L3 Forwarding [MFWD/GWCA]
- L2/L3 Update [MFWD]
- Frame Replication and Elimination for Reliability (FRER) feature [MFWD]
- Per Stream Filtering and Policing (PSFP) feature [MFWD]

Configuration

Build Time Configurations for r_layer3_switch

The following build time configurations are defined in fsp_cfg/r_layer3_switch_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Available queue num	Value must be an integer between 1 and 64	4	available queue num.
gPTP enable	• Enabled • Disabled	Enabled	Select whether or not to use gPTP function.
Time Aware Shaper	• Enabled • Disabled	Disabled	Select whether or not to use the Time Aware Shaper function.

Configurations for Networking > Switch (r_layer3_switch)

This module can be added to the Stacks tab via New Stack > Networking > Switch (r_layer3_switch).

Configuration	Options	Default	Description
Interrupts > Interrupt priority	MCU Specific Options		Select the layer3_switch interrupt priority.
Interrupts > ETHA0 error interrupt priority	MCU Specific Options		Select the layer3_switch etha0 interrupt priority.
Interrupts > ETHA1 error interrupt priority	MCU Specific Options		Select the layer3_switch etha1 interrupt priority.
Ports > Port 0 > IPV queue depth > Queue 0	Value must be an integer more than 1	64	queue0 dept of port 0.
Ports > Port 0 > IPV queue depth > Queue 1	Value must be an integer more than 1	64	queue1 dept of port 0.
Ports > Port 0 > IPV queue depth > Queue 2	Value must be an integer more than 1	64	queue2 dept of port 0.
Ports > Port 0 > IPV queue depth > Queue 3	Value must be an integer more than 1	64	queue3 dept of port 0.
Ports > Port 0 > IPV	Value must be an	64	queue4 dept of port 0.

queue depth > Queue 4	integer more than 1		
Ports > Port 0 > IPV queue depth > Queue 5	Value must be an integer more than 1	64	queue5 dept of port 0.
Ports > Port 0 > IPV queue depth > Queue 6	Value must be an integer more than 1	64	queue6 dept of port 0.
Ports > Port 0 > IPV queue depth > Queue 7	Value must be an integer more than 1	64	queue7 dept of port 0.
Ports > Port 0 > Forwarding	MCU Specific Options		Destination ports for port 0 forwarding
Ports > Port 0 > IPV queues for preemptable frames	• Queue 0 • Queue 1 • Queue 2 • Queue 3 • Queue 4 • Queue 5 • Queue 6 • Queue 7		Select IPV queues that send preemptable frames from port 0.
Ports > Port 0 > Minimum fragment size of preemptable frames	• 64 bytes • 128 bytes • 192 bytes • 256 bytes	64 bytes	Select minimum fragment size of preemptable frames that send from port 0.
Ports > Port 0 > MAC address	Must be a valid MAC address	00:11:22:33:44:55	MAC address of port 0.
Ports > Port 0 > Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs for port0.
Ports > Port 0 > Context	Name must be a valid C symbol	NULL	Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t for port0.
Ports > Port 0 > GPTP timer number	• Timer 0 • Timer 1	Timer 0	GPTP timer number of port 0.
Ports > Port 1 > IPV queue depth > Queue 0	Value must be an integer more than 1	64	queue0 dept of port 1.
Ports > Port 1 > IPV queue depth > Queue 1	Value must be an integer more than 1	64	queue1 dept of port 1.
Ports > Port 1 > IPV queue depth > Queue 2	Value must be an integer more than 1	64	queue2 dept of port 1.

Ports > Port 1 > IPV queue depth > Queue 3	Value must be an integer more than 1	64	queue3 dept of port 1.
Ports > Port 1 > IPV queue depth > Queue 4	Value must be an integer more than 1	64	queue4 dept of port 1.
Ports > Port 1 > IPV queue depth > Queue 5	Value must be an integer more than 1	64	queue5 dept of port 1.
Ports > Port 1 > IPV queue depth > Queue 6	Value must be an integer more than 1	64	queue6 dept of port 1.
Ports > Port 1 > IPV queue depth > Queue 7	Value must be an integer more than 1	64	queue7 dept of port 1.
Ports > Port 1 > Forwarding	MCU Specific Options		Destination ports for port 1 forwarding
Ports > Port 1 > IPV queues for preemptable frames	• Queue 0 • Queue 1 • Queue 2 • Queue 3 • Queue 4 • Queue 5 • Queue 6 • Queue 7		Select IPV queues that send preemptable frames from port 1.
Ports > Port 1 > Minimum fragment size of preemptable frames	• 64 bytes • 128 bytes • 192 bytes • 256 bytes	64 bytes	Select minimum fragment size of preemptable frames that send from port 1.
Ports > Port 1 > MAC address	Must be a valid MAC address	00:11:22:33:44:55	MAC address of port 1.
Ports > Port 1 > Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs for port1.
Ports > Port 1 > Context	Name must be a valid C symbol	NULL	Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t for port1.
Ports > Port 1 > GPTP timer number	• Timer 0 • Timer 1	Timer 1	GPTP timer number of port 1.
Ports > CPU > Forwarding	• Port 0 • Port 1		Destination ports for CPU forwarding
CBS > Port 0 Band width (%) > Queue 0	Value must be an integer between 0 and 100	0	queue0 Band width of port 0.

CBS > Port 0 Band width (%) > Queue 1	Value must be an integer between 0 and 100	0	queue1 Band width of port 0.
CBS > Port 0 Band width (%) > Queue 2	Value must be an integer between 0 and 100	0	queue2 Band width of port 0.
CBS > Port 0 Band width (%) > Queue 3	Value must be an integer between 0 and 100	0	queue3 Band width of port 0.
CBS > Port 0 Band width (%) > Queue 4	Value must be an integer between 0 and 100	0	queue4 Band width of port 0.
CBS > Port 0 Band width (%) > Queue 5	Value must be an integer between 0 and 100	0	queue5 Band width of port 0.
CBS > Port 0 Band width (%) > Queue 6	Value must be an integer between 0 and 100	0	queue6 Band width of port 0.
CBS > Port 0 Band width (%) > Queue 7	Value must be an integer between 0 and 100	0	queue7 Band width of port 0.
CBS > Port 1 Band width (%) > Queue 0	Value must be an integer between 0 and 100	0	queue0 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 1	Value must be an integer between 0 and 100	0	queue1 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 2	Value must be an integer between 0 and 100	0	queue2 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 3	Value must be an integer between 0 and 100	0	queue3 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 4	Value must be an integer between 0 and 100	0	queue4 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 5	Value must be an integer between 0 and 100	0	queue5 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 6	Value must be an integer between 0 and 100	0	queue6 Band width of port 1.
CBS > Port 1 Band width (%) > Queue 7	Value must be an integer between 0 and 100	0	queue7 Band width of port 1.

CBS > Port 0 Maximum burst frame number > Queue 0	Value must be an integer between 0 and 255	0	queue0 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 1	Value must be an integer between 0 and 255	0	queue1 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 2	Value must be an integer between 0 and 255	0	queue2 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 3	Value must be an integer between 0 and 255	0	queue3 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 4	Value must be an integer between 0 and 255	0	queue4 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 5	Value must be an integer between 0 and 255	0	queue5 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 6	Value must be an integer between 0 and 255	0	queue6 maximum burst frame number of port 0.
CBS > Port 0 Maximum burst frame number > Queue 7	Value must be an integer between 0 and 255	0	queue7 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 0	Value must be an integer between 0 and 255	0	queue0 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 1	Value must be an integer between 0 and 255	0	queue1 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 2	Value must be an integer between 0 and 255	0	queue2 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 3	Value must be an integer between 0 and 255	0	queue3 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 4	Value must be an integer between 0 and 255	0	queue4 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 5	Value must be an integer between 0 and 255	0	queue5 maximum burst frame number of port 0.
CBS > Port 1 Maximum burst frame number > Queue 6	Value must be an integer between 0 and 255	0	queue6 maximum burst frame number of port 0.

CBS > Port 1 Maximum burst frame number > Queue 7	Value must be an integer between 0 and 255	0	queue7 maximum burst frame number of port 0.
Name	Name must be a valid C symbol	g_layer3_switch0	Module name.
Channel	0	0	Select the layer3 switch controller channel number.
Context	Name must be a valid C symbol	NULL	Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t.
Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs
Number of Layer 3 routing entries	Value must be an integer between 1 and 256	10	Maximum number of Layer 3 routing entries.

Usage Notes

Time-Sensitive Network (TSN)

Credit Based Shaper [802.1Qav]

Credit Based Shaper (CBS) is a standard defined in IEEE 802.1Qav that limits the throughput of Ethernet traffic. The figure below illustrates the operation of CBS when the bandwidth is set to 40Mbps. When the transmission throughput exceeds the bandwidth configured by CBS, frame transmission is restricted. The restricted frames are buffered in the queue and transmitted with delay.

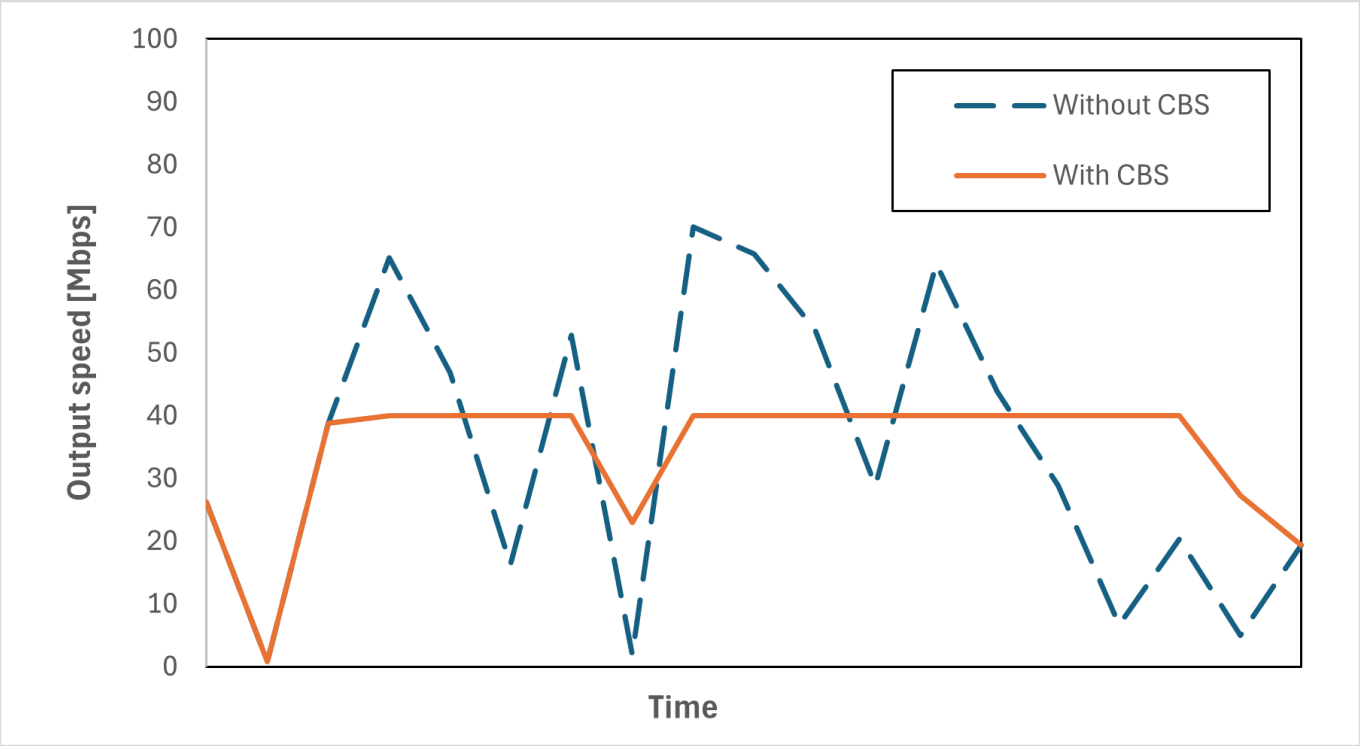


Figure 142: CBS operation image

In the r_layer3_switch module, CBS bandwidth can be configured for each Internal Priority Value (IPV) queue per transmission port. There are 8 IPV queues available for each port. On transmission ports where CBS is enabled, the throughput is limited to the configured bandwidth. CBS can be configured through stack properties or the ConfigurePort() API.

Note

The maximum number of frames that can be buffered by CBS is limited to the depth of each IPV queue. If many frames are buffered in high-priority queues, the common memory pool may be depleted, potentially limiting transmission from other queues.

Time-Aware Shaper [802.1Qbv]

Time-Aware Shaper (TAS) is a standard defined in IEEE 802.1Qbv that limits Ethernet traffic by controlling the time window when frames are transmitted. TAS controls transmission by changing gate states. Frames are transmitted while the gate is opened and when the gate is closed, frames cannot be transmitted and are delayed until the next open state. By opening each gate at different timings, it is possible to prevent frames of specific priorities from monopolizing the bandwidth. The figure below illustrates the operation of TAS. In this image, the gate cycle time is set to 10ms, Gate 0 is opened for 2.5 ms and closed for 7.5 ms, Gate 1 is closed for 1.0 + 5.0 ms and open for 4.0 ms.

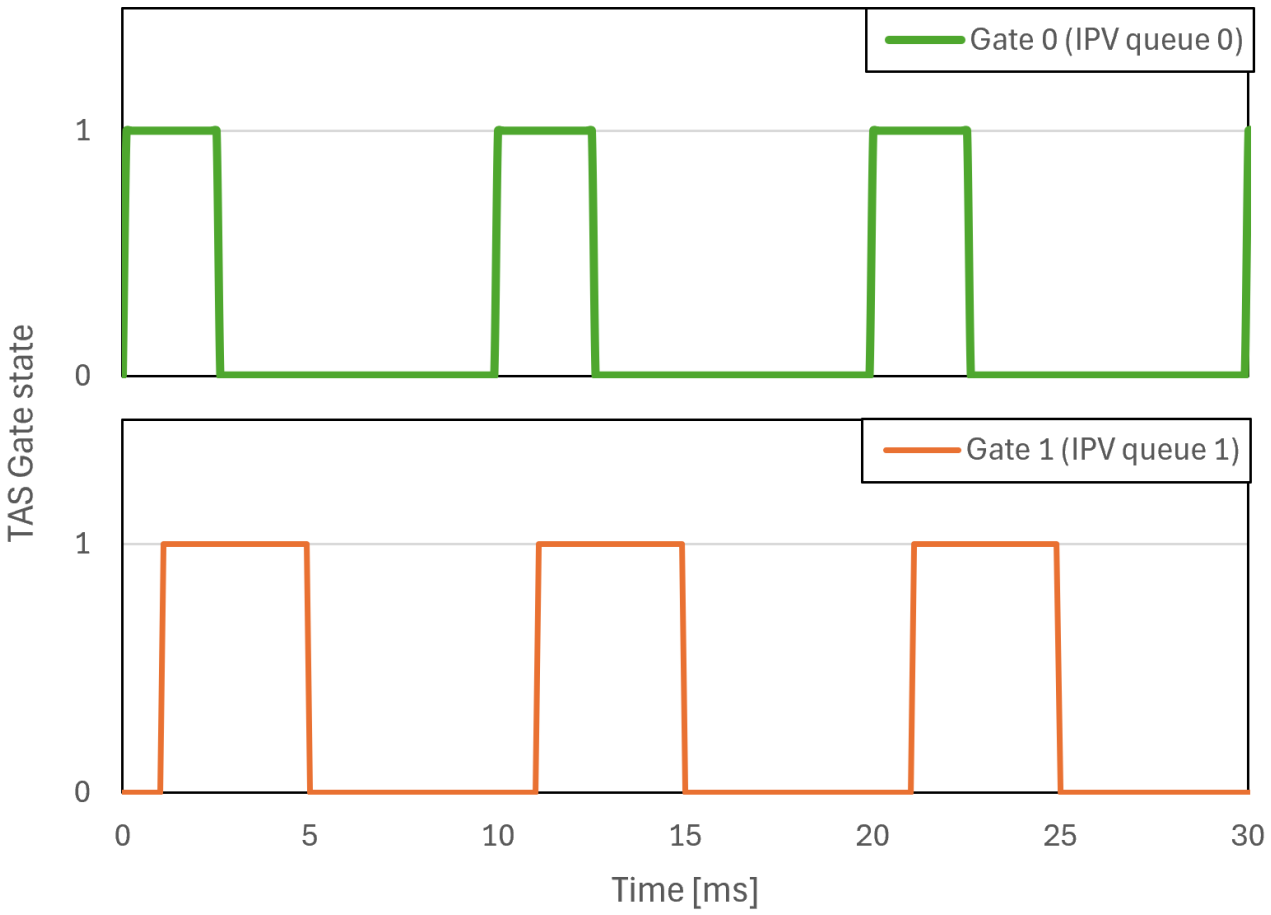


Figure 143: TAS operation image

In the r_layer3_switch module, TAS gates can be configured for each IPV queue of the transmission ports. Each gate learns entries that represent windows. An entry contains the window time and state. TAS feature is configured using the ConfigureTAS() API, and enabled using the EnableTAS() API.

Note

This TAS feature uses the gPTP timer.

Frame preemption [802.3br, 802.1Qbu]

Frame preemption is a standard defined in IEEE 802.3br and 802.1Qbu. This standard allows preempting low-priority frames to send high-priority frames.

Verification process

Frame preemption needs to verify the frame preemption capability of the link partner. The figure below shows the verification process. Verify frames have special preambles, so these frames are rejected on devices which do not support frame preemption.

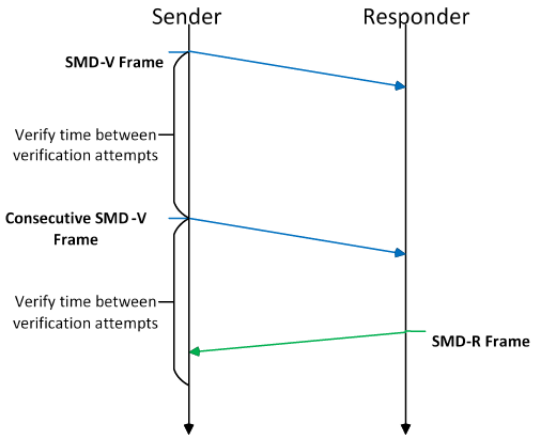


Figure 144: Frame preemption verification

In the r_layer3_switch module, the verification process is performed when the r_rmac_phy module links up and frame preemption is enabled in the configuration.

Preemption

The figure below shows the behavior of frame preemption. The low-priority frame (preemptable frame) is separated into several fragments, and they can be transferred in the intervals between high-priority frames (express frames). As a result, the negative influence on the throughput of high-priority frames is prevented.

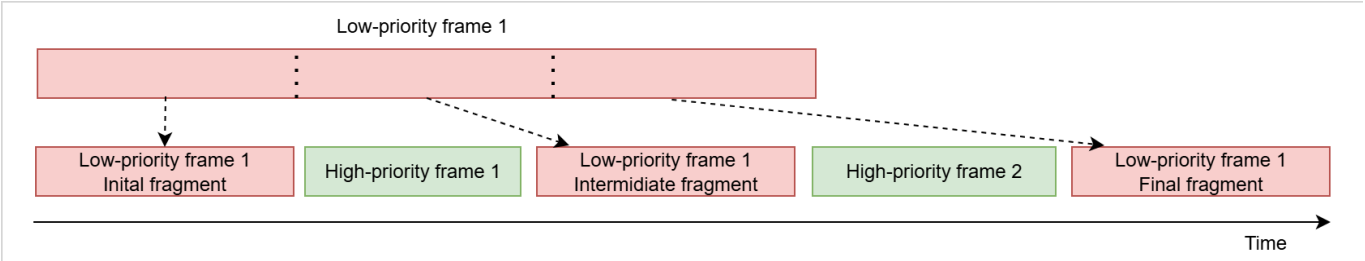


Figure 145: Frame preemption behavior

In the r_layer3_switch module, each IPV queue can be configured to send either preemptable frames or express frames. ESWM IP automatically assembles separated frames when receiving them.

Note

Sending preemptable frames is enabled when the verification process succeeds.

Per Stream Filtering and Policing (PSFP) [802.1Qci]

Per Stream Filtering and Policing (PSFP) feature is a standard defined in IEEE 802.1Qci that supports MSDU and Meter filters.

MAC Service Data Unit (MSDU) Filter

MAC Service Data Unit (MSDU)filter filters packets when Ethernet frame size (payload) is more than a

threshold.

Meter Filter

Meter filter filters packets when information rate is more than a threshold.

The filtered packets are sent to green or yellow bucket by Meter filter configuration.

Normal Mode and Throttle Mode

MSDU and Meter filters support normal mode and throttle mode.

In normal mode, the driver does not send packets filtered MSDU or Meter filter. Also, error status flag becomes 1 when the packet is more than a threshold.

In throttle mode, the driver keeps not to send all packets when error status flag is 1. User can clear error status flag using PsfpClearErrorStatus() API.

Big endian

When big endian mode is selected, the endian setting of ESWM bus is also changed to big endian. As a result, descriptor fields require endian conversion for word (16-bit) and long word (32-bit) accesses.

Most descriptor members behave correctly regardless of endianness. However, a few members require special handling before writing to the descriptor or after reading from it.

- For members of descriptor that occupy a **full 32-bit word** (such as **ptr_l**, **tss**,...), apply endian conversion to the entire value **before writing or after reading**.

```
typedef struct st_layer3_switch_basic_descriptor
{
    ...
 volatile uint32_t ptr_l;
} layer3_switch_basic_descriptor_t;
```

```
/* Endian conversion must be applied to ptr_l */
transmission_descriptor.basic.ptr_l = BYTES_SWAP_32((uintptr_t) ethernet_buffer));
```

- For descriptor containing bitfields, especially **fields larger than 1 byte** (such as **tsns**, **fesf**,...), do not convert each bitfield individually. Since multiple bitfields may share the same 32-bit word, the whole word must be converted first before accessing the bitfields.

```
struct st_reception_ethernet_descriptor
{
 volatile uint32_t tsns : 30;
```

```
 volatile uint32_t tsv : 1;
 volatile uint32_t tsd : 1;
 volatile uint32_t tss;
} reception_ethernet_descriptor;
```

```
uint32_t * p_descriptor_word = (uint32_t *)
(&descriptor.reception_ethernet_descriptor);
p_descriptor_word[index] = BYTES_SWAP_32(p_descriptor_word[index]);
```

Limitations

Developers should be aware of the following limitations when using the r_layer3_switch:

- CBS feature supports only 100 Mbps and 1 Gbps link speeds.
- PSFP feature supports only Meter filter and MSDU filter. PSFP stream gates are not supported.

Examples

Basic Example

This is a basic example of transmission use of the LAYER3 SWITCH.

```
#define LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE (8)
#define LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE (1514)
#define LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER ((uint8_t)
((LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE & \
  0xFF00) >> 8))
#define LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER ((uint8_t)
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE & \
  0x00FF)
#define LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)
#define LAYER3_SWITCH_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01
#define LAYER3_SWITCH_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79,
0x02
#define LAYER3_SWITCH_EXAMPLE_FRAME_TYPE 0x00, 0x2E
#define LAYER3_SWITCH_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
```

```
  0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
/* Swap bytes of a 32-bit value */
#if defined(__BIG)
static uint32_t layer3_switch_example_bytes_swap_32_func (uint32_t x)
{
 return ((x & 0x000000FFUL) << 24) |
         ((x & 0x0000FF00UL) << 8) |
         ((x & 0x00FF0000UL) >> 8) |
         ((x & 0xFF000000UL) >> 24);
}
 #define LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32(x)
layer3_switch_example_bytes_swap_32_func(x)
#endif
layer3_switch_descriptor_t
gp_descriptor_pool[LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE];
uint8_t gp_ether_buffer0[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer1[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer2[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer3[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer4[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer5[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer6[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer7[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t * gpp_ether_buffers[8] =
{
    &gp_ether_buffer0[0],
    &gp_ether_buffer1[0],
    &gp_ether_buffer2[0],
    &gp_ether_buffer3[0],
    &gp_ether_buffer4[0],
    &gp_ether_buffer5[0],
```

```
    &gp_ether_buffer6[0],
    &gp_ether_buffer7[0],
};
/* Transmit data buffer */
uint8_t gp_send_data[LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    LAYER3_SWITCH_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    LAYER3_SWITCH_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    LAYER3_SWITCH_EXAMPLE_FRAME_TYPE,             /* Type field */
    LAYER3_SWITCH_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void layer3_switch_transmit_example (void)
{
 fsp_err_t err         = FSP_SUCCESS;
    uint32_t queue_index = 0;
 /* Configure transmission queue. */
 layer3_switch_descriptor_queue_cfg_t transmission_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_TX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
       .p_descriptor_array    = gp_descriptor_pool,
       .p_ts_descriptor_array = NULL,
       .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
       .array_length          = LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE,
       .ports                 = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
 /* Prepare data for transmission. */
    memcpy(gpp_ether_buffers[0], gp_send_data,
LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE);
 /* Initialize transmission descriptor. */
 layer3_switch_descriptor_t transmission_descriptor = {0};
    transmission_descriptor.basic.ds_l =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER;
```

```
    transmission_descriptor.basic.ds_h =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER;
    transmission_descriptor.basic.die = 1;
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
    transmission_descriptor.basic.ptr_l =
LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32((uintptr_t) gpp_ether_buffers);
#else
    transmission_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers;
#endif
    transmission_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE;
    transmission_descriptor.info1_tx.fmt = 1;
    transmission_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize PHY-LSI chip. */
    err = R_RMAC_PHY_ChipInit(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Get the Ethernet link status read from the PHY-LSI Basic Status registers until
link-up. */
       err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    } while (FSP_SUCCESS != err);
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&transmission_queue_cfg);
    assert(FSP_SUCCESS == err);
```

```
 /* Set descriptor to queue. */
    err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Start transmission. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
 /* Get descriptor status. */
    err = R_LAYER3_SWITCH_GetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Check if transmission is completed. */
    assert(LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY == transmission_descriptor.basic.dt);
 /* Disable transmission and receive function and close the layer3 switch instance.
*/
 R_LAYER3_SWITCH_Close(&g_layer3_switch0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

This is a basic example of reception use of the LAYER3 SWITCH.

```
#define LAYER3_SWITCH_EXAMPLE_RECEPTION_QUEUE_LENGTH (4)
bool g_complete_reception_flag;
void layer3_switch_receive_example (void)
{
 fsp_err_t err         = FSP_SUCCESS;
    uint32_t queue_index = 1;
 /* Configure reception queue. */
 layer3_switch_descriptor_queue_cfg_t reception_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_RX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
```

```
        .p_descriptor_array    = gp_descriptor_pool,
        .p_ts_descriptor_array = NULL,
#if LAYER3_SWITCH_CFG_GPTP_ENABLE
        .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_ENABLE,
#else
        .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
#endif
        .array_length = LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE,
        .ports       = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
 /* Initialize reception descriptor. */
 layer3_switch_descriptor_t reception_descriptor = {0};
    reception_descriptor.basic.ds_l = LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER;
    reception_descriptor.basic.ds_h = LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER;
    reception_descriptor.basic.die = 1;
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
    reception_descriptor.basic.ptr_l =
LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32((uintptr_t) gpp_ether_buffers);
#else
    reception_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers;
#endif
    reception_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY;
    reception_descriptor.info1_tx.fmt = 1;
    reception_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Initialize PHY-LSI chip. */
    err = R_RMAC_PHY_ChipInit(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
```

```
 do
    {
 /* Get the Ethernet link status read from the PHY-LSI Basic Status registers until
link-up. */
       err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    } while (FSP_SUCCESS != err);
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&reception_queue_cfg);
    assert(FSP_SUCCESS == err);
 /* Copy descriptor to queue. */
 for (uint16_t i = 0; i < LAYER3_SWITCH_EXAMPLE_RECEPTION_QUEUE_LENGTH - 1; i++)
    {
       memset(gpp_ether_buffers[i], 0, LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE);
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
       reception_descriptor.basic.ptr_l =
LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32((uintptr_t) gpp_ether_buffers[i]);
#else
       reception_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers[i];
#endif
       err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&reception_descriptor);
      assert(FSP_SUCCESS == err);
    }
    g_complete_reception_flag = false;
 /* Start reception. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 /* Wait to complete reception. */
 while (true != g_complete_reception_flag)
    {
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
```

```
    }
 /* Disable transmission and receive function and close the layer3 switch instance.
*/
 R_LAYER3_SWITCH_Close(&g_layer3_switch0_ctrl);
    assert(FSP_SUCCESS == err);
}
void layer3_switch_example_callback (ether_switch_callback_args_t * p_args) {
 switch (p_args->event)
    {
 case ETHER_SWITCH_EVENT_RX_COMPLETE:
      {
            g_complete_reception_flag = true;
 break;
      }
 case ETHER_SWITCH_EVENT_TX_COMPLETE:
      {
 break;
      }
 case ETHER_SWITCH_EVENT_RX_QUEUE_FULL:
      {
 break;
      }
 case ETHER_SWITCH_EVENT_RX_MESSAGE_LOST:
      {
 break;
      }
 default:
      {
 break;
      }
    }
}
```

TSN Example

The below is an example of forwarding frames with Credit Based Shaper (CBS).

```
void layer3_switch_cbs_example (void)
{
 fsp_err_t                err             = FSP_SUCCESS;
    uint8_t                  source_port      = 0;
    uint8_t                  destination_port = 1;
 layer3_switch_port_cfg_t port_cfg         = {0};
 /* Set bandwidth of queue 0 to 30 Mbps, queue 1 to 20 Mbps. */
 layer3_switch_cbs_cfg_t cbs_cfg =
    {
       .band_width_list    = {30, 20, 0, 0, 0, 0, 0, 0   },
       .max_burst_num_list = {10, 10, 10, 10, 10, 10, 10, 10 },
    };
 /* Destination of forwarding target frames are 00:01:02:03:04:05 and
10:11:12:13:14:15*/
    uint8_t p_mac_address_list[2][6] =
    {
      {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, },
      {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, },
    };
 layer3_switch_table_cfg_t mac_table_cfg =
    {
       .port_cfg_list             = {{.mac_table_enable = true}, {.mac_table_enable =
true}},
       .unsecure_entry_maximum_num = 2,
    };
 layer3_switch_frame_filter_t    mac_entry_frame;
 layer3_switch_table_entry_cfg_t mac_entry_cfg =
    {
       .destination_ports               = (uint32_t) (1 << destination_port),
       .source_ports                   = (uint32_t) (1 << source_port),
       .internal_priority_update_enable = true,
       .entry_enable                   = true,
    };
```

```
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Initialize MAC table entry.*/
    mac_entry_frame.entry_type = LAYER3_SWITCH_TABLE_ENTRY_TYPE_MAC;
 /* Reset MAC table. */
    err = R_LAYER3_SWITCH_ConfigureTable(&g_layer3_switch0_ctrl, &mac_table_cfg);
    assert(FSP_SUCCESS == err);
 /* Learn destination MAC address of a forwarding frame. */
 for (uint8_t i = 0; i < 2; i++)
    {
       mac_entry_frame.p_destination_mac_address    = p_mac_address_list[i];
       mac_entry_cfg.internal_priority_update_value = i;
       err = R_LAYER3_SWITCH_AddTableEntry(&g_layer3_switch0_ctrl, &mac_entry_frame,
&mac_entry_cfg);
      assert(FSP_SUCCESS == err);
    }
 /* Configure and enable CBS. */
    port_cfg.p_cbs_cfg = &cbs_cfg;
    err                = R_LAYER3_SWITCH_ConfigurePort(&g_layer3_switch0_ctrl,
destination_port, &port_cfg);
    assert(FSP_SUCCESS == err);
 while (1)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

The below is an example of forwarding frames with Time Aware Shaper (TAS).

```
#define LAYER3_SWITCH_EXAMPLE_GPTP_TIMER (0)
#define LAYER3_SWITCH_EXAMPLE_GPTP_CLOCK_PERIOD (4)
void layer3_switch_tas_example (void)
{
```

```
 fsp_err_t err              = FSP_SUCCESS;
    uint8_t   source_port      = 0;
    uint8_t   destination_port = 1;
 layer3_switch_table_entry_t mac_table_entry;
 layer3_switch_tas_cfg_t      tas_cfg;
 layer3_switch_tas_entry_t    closed_entry_list[1] = {{.state =
LAYER3_SWITCH_TAS_GATE_STATE_CLOSE, .time = 0}};
 layer3_switch_tas_gate_cfg_t tas_gate_closed      =
    {
       .tas_entry_num    = 1, .initial_gate_state =
LAYER3_SWITCH_TAS_GATE_STATE_CLOSE,
       .p_tas_entry_list = closed_entry_list
    };
 gptp_instance_t const * p_gptp_instance =
g_layer3_switch0_extended_cfg.p_gptp_instance;
 gptp_timer_cfg_t       timer_cfg       = {.clock_period =
LAYER3_SWITCH_EXAMPLE_GPTP_CLOCK_PERIOD};
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Gate 0 and 1 are repeat Open/Close. Other gates are alwayes closed. */
    tas_cfg.gate_cfg_list[0] = g_layer3_switch_example_tas_gate_0_cfg;
    tas_cfg.gate_cfg_list[1] = g_layer3_switch_example_tas_gate_1_cfg;
 for (uint8_t i = 2; i < 8; i++)
    {
       tas_cfg.gate_cfg_list[i] = tas_gate_closed;
    }
    tas_cfg.gptp_timer_number    = LAYER3_SWITCH_EXAMPLE_GPTP_TIMER;
    tas_cfg.cycle_time_start_high =
LAYER3_SWITCH_EXAMPLE_CYCLE_START_TIME_UPPER_32BIT;
    tas_cfg.cycle_time_start_low =
LAYER3_SWITCH_EXAMPLE_CYCLE_START_TIME_UPPER_32BIT;
    tas_cfg.cycle_time            = LAYER3_SWITCH_EXAMPLE_CYCLE_TIME;
 /* Start gPTP timer before enable TAS feature. */
```

```
    p_gptp_instance->p_api->timerCfg(p_gptp_instance->p_ctrl,
LAYER3_SWITCH_EXAMPLE_GPTP_TIMER, &timer_cfg);
    p_gptp_instance->p_api->start(p_gptp_instance->p_ctrl,
LAYER3_SWITCH_EXAMPLE_GPTP_TIMER);
 /* Configure and enable TAS. */
    err = R_LAYER3_SWITCH_ConfigureTAS(&g_layer3_switch0_ctrl, destination_port,
&tas_cfg);
    assert(FSP_SUCCESS == err);
    err = R_LAYER3_SWITCH_EnableTAS(&g_layer3_switch0_ctrl, destination_port);
    assert(FSP_SUCCESS == err);
 /* Initialize MAC table entry.*/
    err = R_LAYER3_SWITCH_ConfigureTable(&g_layer3_switch0_ctrl,
&g_layer3_switch_example_mac_table_cfg);
    assert(FSP_SUCCESS == err);
 /* Learn a MAC entry. */
    mac_table_entry.target_frame.entry_type                    =
LAYER3_SWITCH_TABLE_ENTRY_TYPE_MAC;
    mac_table_entry.entry_cfg.entry_enable                    = true;
    mac_table_entry.entry_cfg.security_enable                 = true;
    mac_table_entry.entry_cfg.destination_ports                = (uint32_t) (1 <<
destination_port);
    mac_table_entry.entry_cfg.source_ports                    = (uint32_t) (1 <<
source_port);
    mac_table_entry.entry_cfg.internal_priority_update_enable = true;
 for (uint8_t i = 0; i < 2; i++)
    {
       mac_table_entry.target_frame.p_destination_mac_address   =
g_layer3_switch_example_mac_address_list[i];
       mac_table_entry.entry_cfg.internal_priority_update_value = i;
       err = R_LAYER3_SWITCH_AddTableEntry(&g_layer3_switch0_ctrl,
                                           &mac_table_entry.target_frame,
                                           &mac_table_entry.entry_cfg);
      assert(FSP_SUCCESS == err);
    }
```

```
 while (1)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

PSFP Example

MSDU Filter

The below is an example of PSFP MSDU filter.

```
#define LAYER3_SWITCH_EXAMPLE_PSFP_MSDU_FILTER_ID (0)
#define LAYER3_SWITCH_EXAMPLE_PSFP_MSDU_FILTER_FRAME_MAX_SIZE (60)
#define LAYER3_SWITCH_EXAMPLE_PSFP_MSDU_FILTER_CLEAR_BITMASK (0xFFFF)
void layer3_switch_psfp_msdu_filter_example (void)
{
 fsp_err_t err             = FSP_SUCCESS;
    uint8_t   source_port      = 0;
    uint8_t   destination_port = 1;
    uint32_t queue_index      = 0;
 layer3_switch_table_entry_t               mac_table_entry;
 layer3_switch_psfp_error_status_bitmask_t bitmasks;
 layer3_switch_psfp_msdu_filter_cfg_t      psfp_msdu_filter_cfg;
 /* Configure transmission queue. */
 layer3_switch_descriptor_queue_cfg_t transmission_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_TX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
       .p_descriptor_array    = gp_descriptor_pool,
       .p_ts_descriptor_array = NULL,
       .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
       .array_length          = LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE,
       .ports                 = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
```

```
 /* Prepare data for transmission. */
    memcpy(gpp_ether_buffers[0], gp_send_data,
LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE);
 /* Initialize transmission descriptor. */
 layer3_switch_descriptor_t transmission_descriptor = {0};
    transmission_descriptor.basic.ds_l =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER;
    transmission_descriptor.basic.ds_h =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER;
    transmission_descriptor.basic.die = 1;
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
    transmission_descriptor.basic.ptr_l =
LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32((uintptr_t) gpp_ether_buffers);
#else
    transmission_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers;
#endif
    transmission_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE;
    transmission_descriptor.info1_tx.fmt = 1;
    transmission_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Initialize MAC table entry.*/
    err = R_LAYER3_SWITCH_ConfigureTable(&g_layer3_switch0_ctrl,
&g_layer3_switch_example_mac_table_cfg);
    assert(FSP_SUCCESS == err);
 /* Learn a MAC entry. */
    mac_table_entry.target_frame.entry_type                   =
LAYER3_SWITCH_TABLE_ENTRY_TYPE_LAYER3;
    mac_table_entry.entry_cfg.entry_enable                   = true;
    mac_table_entry.entry_cfg.security_enable                = true;
    mac_table_entry.entry_cfg.destination_ports               = (uint32_t) (1 <<
destination_port);
```

```
    mac_table_entry.entry_cfg.source_ports                    = (1 << source_port);
    mac_table_entry.entry_cfg.internal_priority_update_enable = true;
 /* Configure PSFP MSDU filter. */
    psfp_msdu_filter_cfg.msdu_filter_id             =
LAYER3_SWITCH_EXAMPLE_PSFP_MSDU_FILTER_ID;
    psfp_msdu_filter_cfg.maximum_frame_size          =
LAYER3_SWITCH_EXAMPLE_PSFP_MSDU_FILTER_FRAME_MAX_SIZE;
    psfp_msdu_filter_cfg.mode                        =
LAYER3_SWITCH_MSDU_FILTER_THROTTLE_MODE;
    mac_table_entry.entry_cfg.p_psfp_msdu_filter_cfg = &psfp_msdu_filter_cfg;
 for (uint8_t i = 0; i < 2; i++)
    {
       mac_table_entry.target_frame.p_destination_mac_address   =
g_layer3_switch_example_mac_address_list[i];
       mac_table_entry.entry_cfg.internal_priority_update_value = i;
       err = R_LAYER3_SWITCH_AddTableEntry(&g_layer3_switch0_ctrl,
                                           &mac_table_entry.target_frame,
                                           &mac_table_entry.entry_cfg);
      assert(FSP_SUCCESS == err);
    }
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&transmission_queue_cfg);
    assert(FSP_SUCCESS == err);
 /* Set descriptor to queue. */
    err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Start transmission. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 /* Set bitmask for clearing error status. */
    bitmasks.msdu_filter_bitmask =
LAYER3_SWITCH_EXAMPLE_PSFP_MSDU_FILTER_CLEAR_BITMASK;
```

```
    bitmasks.meter_filter_bitmask = 0;
    err = R_LAYER3_SWITCH_PsfpClearErrorStatus(&g_layer3_switch0_ctrl, bitmasks);
    assert(FSP_SUCCESS == err);
}
```

Meter Filter

The below is an example of PSFP Meter filter.

```
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_FILTER_ID (0)
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_CBS (1000)
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_CIR (10)
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_EBS (1000)
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_EIR (10)
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_SYSTEM_CLOCK (250)
#define LAYER3_SWITCH_EXAMPLE_PSFP_INFORMATION_RATE_DIVIDE_VALUE (8)
#define LAYER3_SWITCH_EXAMPLE_PSFP_INFORMATION_RATE_1M (1000000)
#define LAYER3_SWITCH_EXAMPLE_PSFP_INFORMATION_RATE_MULTIPLY_VALUE (65536)
#define LAYER3_SWITCH_EXAMPLE_PSFP_METER_FILTER_CLEAR_BITMASK (0xFFFFFFFF)
void layer3_switch_psfp_meter_filter_example (void)
{
 fsp_err_t err             = FSP_SUCCESS;
    uint8_t   source_port      = 0;
    uint8_t   destination_port = 1;
    uint32_t queue_index      = 0;
 layer3_switch_table_entry_t               mac_table_entry;
 layer3_switch_psfp_error_status_bitmask_t bitmasks;
 layer3_switch_psfp_meter_filter_cfg_t    psfp_meter_filter_cfg;
 /* Configure transmission queue. */
 layer3_switch_descriptor_queue_cfg_t transmission_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_TX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
       .p_descriptor_array    = gp_descriptor_pool,
```

```
        .p_ts_descriptor_array = NULL,
        .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
        .array_length          = LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE,
        .ports                 = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
 /* Prepare data for transmission. */
    memcpy(gpp_ether_buffers[0], gp_send_data,
LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE);
 /* Initialize transmission descriptor. */
 layer3_switch_descriptor_t transmission_descriptor = {0};
    transmission_descriptor.basic.ds_l =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER;
    transmission_descriptor.basic.ds_h =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER;
    transmission_descriptor.basic.die = 1;
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
    transmission_descriptor.basic.ptr_l =
LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32((uintptr_t) gpp_ether_buffers);
#else
    transmission_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers;
#endif
    transmission_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE;
    transmission_descriptor.info1_tx.fmt = 1;
    transmission_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Initialize MAC table entry.*/
    err = R_LAYER3_SWITCH_ConfigureTable(&g_layer3_switch0_ctrl,
&g_layer3_switch_example_mac_table_cfg);
    assert(FSP_SUCCESS == err);
 /* Learn a MAC entry. */
    mac_table_entry.target_frame.entry_type                  =
```

```
LAYER3_SWITCH_TABLE_ENTRY_TYPE_LAYER3;
    mac_table_entry.entry_cfg.entry_enable                    = true;
    mac_table_entry.entry_cfg.security_enable                 = true;
    mac_table_entry.entry_cfg.destination_ports                = (uint32_t) (1 <<
destination_port);
    mac_table_entry.entry_cfg.source_ports                    = (1 << source_port);
    mac_table_entry.entry_cfg.internal_priority_update_enable = true;
    mac_table_entry.entry_cfg.p_psfp_msdu_filter_cfg           = NULL;
    mac_table_entry.entry_cfg.p_psfp_meter_filter_cfg          = NULL;
    mac_table_entry.entry_cfg.p_frer_entry_cfg                 = NULL;
    mac_table_entry.entry_cfg.layer3.p_update_configs          = NULL;
    mac_table_entry.entry_cfg.layer3.number_of_configs        = 0;
    mac_table_entry.entry_cfg.destination_queue_index          = 0;
 /* Configure PSFP Meter filter. */
    psfp_meter_filter_cfg.meter_filter_id =
LAYER3_SWITCH_EXAMPLE_PSFP_METER_FILTER_ID;
    psfp_meter_filter_cfg.cbs             = LAYER3_SWITCH_EXAMPLE_PSFP_METER_CBS;
    psfp_meter_filter_cfg.cir             =
layer3_switch_psfp_information_rate_example(
       LAYER3_SWITCH_EXAMPLE_PSFP_METER_CIR,
       LAYER3_SWITCH_EXAMPLE_PSFP_METER_SYSTEM_CLOCK);
    psfp_meter_filter_cfg.ebs = LAYER3_SWITCH_EXAMPLE_PSFP_METER_EBS;
    psfp_meter_filter_cfg.eir = layer3_switch_psfp_information_rate_example(
       LAYER3_SWITCH_EXAMPLE_PSFP_METER_EIR,
       LAYER3_SWITCH_EXAMPLE_PSFP_METER_SYSTEM_CLOCK);
    psfp_meter_filter_cfg.mode                =
LAYER3_SWITCH_METER_FILTER_THROTTLE_MODE;
    psfp_meter_filter_cfg.drop_red_frame      =
LAYER3_SWITCH_METER_FILTER_DROP_RED_FRAME;
    psfp_meter_filter_cfg.coupling_enable    =
LAYER3_SWITCH_METER_FILTER_COUPLING_STATUS_DISABLE;
    psfp_meter_filter_cfg.dei_handling_policy =
LAYER3_SWITCH_METER_FILTER_DEI_HANDLING_POLICY_YELLOW;
    psfp_meter_filter_cfg.pcp_handling_policy =
```

```
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_GREEN_ALL;
    mac_table_entry.entry_cfg.p_psfp_meter_filter_cfg = &psfp_meter_filter_cfg;
    mac_table_entry.target_frame.p_destination_mac_address   =
g_layer3_switch_example_mac_address_list[0];
    mac_table_entry.entry_cfg.internal_priority_update_value = 0;
    err = R_LAYER3_SWITCH_AddTableEntry(&g_layer3_switch0_ctrl,
                                        &mac_table_entry.target_frame,
                                        &mac_table_entry.entry_cfg);
    assert(FSP_SUCCESS == err);
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&transmission_queue_cfg);
    assert(FSP_SUCCESS == err);
 /* Set descriptor to queue. */
    err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Start transmission. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 /* Set bitmask for clearing error status. */
    bitmasks.msdu_filter_bitmask = 0;
    bitmasks.meter_filter_bitmask =
LAYER3_SWITCH_EXAMPLE_PSFP_METER_FILTER_CLEAR_BITMASK;
    err = R_LAYER3_SWITCH_PsfpClearErrorStatus(&g_layer3_switch0_ctrl, bitmasks);
    assert(FSP_SUCCESS == err);
}
```

Frame Preemption Example

The below is an example of Frame Preemption.

```
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_DESCRIPTOR_QUEUE_LENGTH (3)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_MAXIMUM_FRAME_SIZE (1514)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50,
```

```
0x00, 0x79, 0x01
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_DESTINATION_MAC_ADDRESS 0x44,
0x44, 0x44, 0x00, 0x00, 0x0A
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_DESTINATION_MAC_ADDRESS 0x44,
0x44, 0x44, 0x00, 0x00, 0x0B
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_FRAME_TYPE 0x05, 0xDC //
1500(=0x05DC) byte
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_FRAME_TYPE 0x00, 0x32 //
50(=0x0032) byte
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_PRIORITY (1)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_PRIORITY (0)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE (64)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE (1514)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE_UPPER \
  ((uint8_t) ((LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE & \
  0xFF00) >> 8))
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE_LOWER \
  ((uint8_t) LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE & \
  0x00FF)
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE_UPPER \
  ((uint8_t) ((LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE & \
  0xFF00) >> 8))
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE_LOWER \
  ((uint8_t) LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE & \
  0x00FF)
/* 50 byte data */
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_PAYLOAD \
  0xFF, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
  0xFF, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
  0xFF, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
  0xFF, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
  0xFF, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09
/* 1500 byte data */
#define LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_PAYLOAD \
```

```
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
```

```
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
```

```
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
```

```
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, \
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, \
```

```
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, \
  0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, \
  0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, \
  0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, \
  0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, \
  0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, \
  0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, \
  0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, \
  0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, \
  0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, \
  0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, \
  0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, \
  0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, \
  0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9,
layer3_switch_descriptor_t gp_tx_descriptor_array[LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPT
ION_DESCRIPTOR_QUEUE_LENGTH];
uint8_t gp_ether_buffer_express_frame[LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_MAXIMUM_
FRAME_SIZE];
uint8_t gp_ether_buffer_preemptable_frame[LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_MAXI
MUM_FRAME_SIZE];
/* Transmit data buffer */
uint8_t gp_high_send_data[LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_MAXIMUM_FRAME_SIZE]
=
{
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_DESTINATION_MAC_ADDRESS, /*
Destination MAC address */
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_SOURCE_MAC_ADDRESS,          /* Source MAC
address */
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_FRAME_TYPE,             /* Type field
*/
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_PAYLOAD                  /* Payload
value (46byte) */
};
uint8_t gp_low_send_data[LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_MAXIMUM_FRAME_SIZE] =
```

```
{
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_DESTINATION_MAC_ADDRESS, /*
Destination MAC address */
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_SOURCE_MAC_ADDRESS,          /* Source MAC
address */
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_FRAME_TYPE,             /* Type field
*/
    LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_PAYLOAD                  /* Payload
value (1500byte) */
};
bool g_tx_complete_flag = false;
void layer3_switch_frame_preemption_transmit_example (void)
{
 fsp_err_t err         = FSP_SUCCESS;
    uint32_t queue_index = 0;
 layer3_switch_descriptor_t transmission_descriptor = {0};
 /* Configure transmission queue. */
 layer3_switch_descriptor_queue_cfg_t transmission_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_TX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
       .p_descriptor_array    = gp_tx_descriptor_array,
       .p_ts_descriptor_array = NULL,
       .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
       .array_length          =
LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_DESCRIPTOR_QUEUE_LENGTH,
       .ports                 = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize PHY-LSI chip. */
```

```
    err = R_RMAC_PHY_ChipInit(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Get the Ethernet link status read from the PHY-LSI Basic Status registers until
link-up. */
       err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    } while (FSP_SUCCESS != err);
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&transmission_queue_cfg);
    assert(FSP_SUCCESS == err);
 /* Prepare data for preemptable frame. */
    memcpy(gp_ether_buffer_preemptable_frame,
          gp_low_send_data,
          LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE);
 /* Initialize transmission descriptor for preemptable frame. */
    transmission_descriptor.basic.ds_l   =
LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE_LOWER;
    transmission_descriptor.basic.ds_h   =
LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_SEND_FRAME_SIZE_UPPER;
    transmission_descriptor.basic.die    = 1;
    transmission_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE;
    transmission_descriptor.info1_tx.fmt = 1;
    transmission_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Sending a preemptable frame with lower IPV. */
    transmission_descriptor.info1_tx.ipv = (uint8_t)
(LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_LOW_PRIORITY & 0x07);
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
```

```
    transmission_descriptor.basic.ptr_l = LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32(
       (uintptr_t) gp_ether_buffer_preemptable_frame);
#else
    transmission_descriptor.basic.ptr_l = (uintptr_t)
gp_ether_buffer_preemptable_frame;
#endif
 /* Set descriptor to queue. */
    err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Start transmission for sending preemptable frame. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 while (!g_tx_complete_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MICROSECONDS);
    }
    g_tx_complete_flag = false;
 /* Prepare data for express frame. */
    memcpy(gp_ether_buffer_express_frame, gp_high_send_data,
          LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE);
    transmission_descriptor.basic.ds_l =
LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE_LOWER;
    transmission_descriptor.basic.ds_h =
LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_SEND_FRAME_SIZE_UPPER;
 /* Sending a preemptable frame with higher IPV. */
    transmission_descriptor.info1_tx.ipv = (uint8_t)
(LAYER3_SWITCH_EXAMPLE_FRAME_PREEMPTION_HIGH_PRIORITY & 0x07);
#if defined(__BIG)
 /* Endian conversion must be applied to ptr_l before writing. */
    transmission_descriptor.basic.ptr_l =
       LAYER3_SWITCH_EXAMPLE_BYTES_SWAP_32((uintptr_t)
gp_ether_buffer_express_frame);
#else
```

```
    transmission_descriptor.basic.ptr_l = (uintptr_t) gp_ether_buffer_express_frame;
#endif
 /* Set descriptor to queue. */
    err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Start transmission for sending express frame. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 while (!g_tx_complete_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MICROSECONDS);
    }
 /* Close the layer3 switch instance. */
 R_LAYER3_SWITCH_Close(&g_layer3_switch0_ctrl);
    assert(FSP_SUCCESS == err);
}
void layer3_switch_frame_preemption_example_callback (ether_switch_callback_args_t *
p_args) {
 switch (p_args->event)
    {
 case ETHER_SWITCH_EVENT_RX_COMPLETE:
      {
 break;
      }
 case ETHER_SWITCH_EVENT_TX_COMPLETE:
      {
            g_tx_complete_flag = true;
 break;
      }
 case ETHER_SWITCH_EVENT_RX_QUEUE_FULL:
      {
 break;
      }
```

```
 case ETHER_SWITCH_EVENT_RX_MESSAGE_LOST:
      {
 break;
      }
 default:
      {
 break;
      }
    }
}
```

Data Structures

struct	layer3_switch_basic_descriptor_t
struct	layer3_switch_descriptor_t
struct	layer3_switch_ts_reception_process_descriptor_t
struct	layer3_switch_psfp_msdu_filter_cfg_t
struct	layer3_switch_psfp_msdu_filter_info_t
struct	layer3_switch_psfp_meter_filter_cfg_t
struct	layer3_switch_psfp_meter_filter_info_t
struct	layer3_switch_psfp_error_status_bitmask_t
struct	layer3_switch_frer_cfg_t
struct	layer3_switch_frer_entry_t
struct	layer3_switch_frer_entry_cfg_t
struct	layer3_switch_frer_sequence_recovery_status_t
struct	layer3_switch_descriptor_queue_cfg_t
struct	layer3_switch_cbs_cfg_t
struct	layer3_switch_port_cfg_t
struct	layer3_switch_descriptor_queue_status_t

struct	layer3_switch_ipv6_filter_address_offset_t
struct	layer3_switch_l3_stream_filter_cfg_t
struct	layer3_switch_frame_vlan_tag_t
struct	layer3_switch_frame_filter_t
struct	layer3_switch_l3_filter_t
struct	layer3_switch_l3_update_config_t
struct	layer3_switch_table_entry_cfg_t
struct	layer3_switch_table_entry_t
struct	layer3_switch_forwarding_port_cfg_t
struct	layer3_switch_table_t
struct	layer3_switch_table_cfg_t
struct	layer3_switch_extended_cfg_t
struct	layer3_switch_instance_ctrl_t
struct	layer3_switch_tas_entry_t
struct	layer3_switch_tas_gate_cfg_t
struct	layer3_switch_tas_cfg_t

Enumerations

enum	layer3_switch_descriptor_type_t
enum	layer3_switch_port_bitmask_t
enum	layer3_switch_queue_type_t
enum	layer3_switch_descriptor_format_t
enum	layer3_switch_write_back_mode_t
enum	layer3_switch_ip_version_t
enum	layer3_switch_ip_address_direction_t
enum	layer3_switch_magic_packet_detection_t

enum	layer3_switch_vlan_mode_t
enum	layer3_switch_vlan_ingress_mode_t
enum	layer3_switch_vlan_egress_mode_t
enum	layer3_switch_ip_protocol_t
enum	layer3_switch_forwarding_r_tag_t
enum	layer3_switch_l3_filter_bitmask_t
enum	layer3_switch_l3_update_bitmask_t
enum	layer3_switch_table_entry_type_t
enum	layer3_switch_table_status_t
enum	layer3_switch_tas_gate_state_t
enum	layer3_switch_rx_timestamp_storage_t
enum	layer3_switch_ts_descriptor_queue_status_t
enum	layer3_switch_preemptable_frame_fragment_size_t
enum	layer3_switch_psfp_filter_status_t
enum	layer3_switch_msdu_filter_mode_t
enum	layer3_switch_meter_filter_mode_t
enum	layer3_switch_meter_filter_coupling_status_t
enum	layer3_switch_meter_filter_drop_red_frame_t
enum	layer3_switch_meter_filter_dei_handling_policy_t
enum	layer3_switch_meter_filter_pcp_handling_policy_t

Data Structure Documentation

◆ layer3_switch_basic_descriptor_t

struct layer3_switch_basic_descriptor_t
GWCA descriptor.
Data Fields

volatile uint8_t	ds_l	0.. 8 (8 bits), Descriptor size (low).
volatile uint8_t	ds_h: 4	9..12 (4 bits), Descriptor size (High).
volatile uint8_t	info0: 4	12..15 (4 bits), Information 0.
volatile uint8_t	err: 3	16..18 (3 bits), Error, data size error, AXI bus error.
volatile uint8_t	die: 1	19 (1 bit), Descriptor interrupt enable.
volatile uint8_t	dt: 4	20..23 (4 bits), Descriptor type.
volatile uint8_t	ptr_h	24..31 (8 bits), Pointer (High).
volatile uint32_t	ptr_l	32..63 (32 bits), Pointer (Low).

◆ layer3_switch_descriptor_t

struct layer3_switch_descriptor_t		
GWCA extended descriptor.		
Data Fields		
volatile layer3_switch_basic_descriptor_t	basic	Basic descriptor fields.
union layer3_switch_descriptor_t	__unnamed__	

◆ layer3_switch_ts_reception_process_descriptor_t

struct layer3_switch_ts_reception_process_descriptor_t
GWCA TS reception process descriptor.

◆ layer3_switch_psfp_msdu_filter_cfg_t

struct layer3_switch_psfp_msdu_filter_cfg_t		
Configuration of PSFP MSDU filter.		
Data Fields		
uint32_t	msdu_filter_id	PSFP MSDU filter ID.
uint32_t	maximum_frame_size	Maximum frame size is rejected by PSFP MSDU filter.
layer3_switch_msdu_filter_mode_t	mode	PSFP MSDU filter mode.

◆ layer3_switch_psfp_msdu_filter_info_t

struct layer3_switch_psfp_msdu_filter_info_t

Information of PSFP MSDU filter.		
Data Fields		
uint32_t	msdu_filter_hw_id	PSFP MSDU filter ID for hardware.
layer3_switch_psfp_filter_status_t	status	Status of PSFP MSDU filter.
layer3_switch_psfp_msdu_filter_cfg_t *	p_psfp_msdu_filter_cfg	Pointer to configuration of PSFP MSDU filter.

◆ layer3_switch_psfp_meter_filter_cfg_t

struct layer3_switch_psfp_meter_filter_cfg_t		
Configuration of PSFP Meter filter for user.		
Data Fields		
uint32_t	meter_filter_id	PSFP Meter filter ID.
uint32_t	cbs	Maximum number of tokens (in bytes) contained in green bucket.
uint32_t	cir	Used to increment green tokens when green bucket is not full.
uint32_t	ebs	Maximum number of tokens (in bytes) contained in yellow bucket.
uint32_t	eir	Used to increment yellow tokens when yellow bucket is not full.
layer3_switch_meter_filter_mode_t	mode	Meter filter mode.
layer3_switch_meter_filter_drop_red_frame_t	drop_red_frame	Drop or not red frame by Meter filter.
layer3_switch_meter_filter_coupling_status_t	coupling_enable	When green bucket is full, CIR tokens are 0:lost, 1:added the yellow bucket.
layer3_switch_meter_filter_dei_handling_policy_t	dei_handling_policy	DEI handling policy for determining a frame color.
layer3_switch_meter_filter_pcp_handling_policy_t	pcp_handling_policy	PCP handling policy for determining a frame color.

◆ layer3_switch_psfp_meter_filter_info_t

struct layer3_switch_psfp_meter_filter_info_t		
Information of PSFP Meter filter.		
Data Fields		

uint32_t	meter_filter_hw_id	PSFP Meter filter ID for hardware.
layer3_switch_psfp_filter_status_t	status	Status of PSFP Meter filter.
layer3_switch_psfp_meter_filter_cfg_t *	p_psfp_meter_filter_cfg	Pointer to configuration of PSFP Meter filter.

◆ layer3_switch_psfp_error_status_bitmask_t

struct layer3_switch_psfp_error_status_bitmask_t		
Bitmasks for clearing error status of throttle mode.		
Data Fields		
uint32_t	msdu_filter_bitmask	Bitmask for clearing error status bit of PSFP MSDU filter.
uint32_t	meter_filter_bitmask	Bitmask for clearing error status bit of PSFP Meter filter.

◆ layer3_switch_frer_cfg_t

struct layer3_switch_frer_cfg_t		
Configuration of FRER feature.		
Data Fields		
uint32_t	sys_clock	Timeout check time.
uint32_t	timeout_enable	Timeout check valid.
uint32_t	check_period	Microsecond prescaler used to creates an internal clock for aging at 1 MHz to derive the timeout 1 kHz clock.

◆ layer3_switch_frer_entry_t

struct layer3_switch_frer_entry_t		
FRER table entry.		
Data Fields		
uint32_t	take_no_sequence	Reject frames without R-TAG.
uint32_t	sequence_history_len	Sequence history length.
uint32_t	set_recovery_remaining_tick	Time in tick before timeout.

◆ layer3_switch_frer_entry_cfg_t

struct layer3_switch_frer_entry_cfg_t		
Configuration of FRER entry (individual and sequence recovery).		
Data Fields		
layer3_switch_frer_entry_t	individual_recovery	FRER table entry for individual

		recovery.
layer3_switch_frer_entry_t *	p_sequence_recovery	Pointer to FRER entry(sequence recovery) information referenced by FRERN in this FRER entry(individual recovery).
uint32_t	sequence_recovery_id	

◆ layer3_switch_frer_sequence_recovery_status_t

struct layer3_switch_frer_sequence_recovery_status_t		
Status of FRER sequence recovery table.		
Data Fields		
bool	learned	Flags whether the FRER entry(sequence recovery) related to this entry has been learned.
uint32_t	frer_entry_index	Real FRER table index of the FRER entry(sequence recovery) related to this entry.

◆ layer3_switch_descriptor_queue_cfg_t

struct layer3_switch_descriptor_queue_cfg_t		
Configuration of a descriptor queue.		
Data Fields		
layer3_switch_descriptor_format_t	descriptor_format	Enable or disable extended descriptors.
layer3_switch_queue_type_t	type	Reception queue or transmission queue.
layer3_switch_write_back_mode_t	write_back_mode	Configure write-back to descriptor fields.
layer3_switch_descriptor_t *	p_descriptor_array	Descriptor array that use to create queue.
layer3_switch_ts_reception_process_descriptor_t *	p_ts_descriptor_array	TS descriptor array that use to create queue.
layer3_switch_rx_timestamp_storage_t	rx_timestamp_storage	Configure RX timestamp storage.
uint32_t	array_length	Length of descriptor array. This length includes terminate descriptor at the end.
uint32_t	ports	Bitmap of ports that use this queue.

◆ layer3_switch_cbs_cfg_t

struct layer3_switch_cbs_cfg_t		
Configuration of Credit Based Shaper.		
Data Fields		
uint8_t	band_width_list[8]	CBS band width [%] of each queue.
uint8_t	max_burst_num_list[8]	Maximum burst frame number of each queue.

◆ layer3_switch_port_cfg_t

struct layer3_switch_port_cfg_t	
Configuration of each Ethernet port.	
Data Fields	
uint8_t *	p_mac_address
	Pointer to MAC address.
bool	forwarding_to_cpu_enable
	Enable or disable reception on CPU.
layer3_switch_cbs_cfg_t *	p_cbs_cfg
	Pointer to CBS configuration.
void(*	p_callback)(ether_switch_callback_args_t *p_args)
	Callback provided when an ISR occurs.
ether_switch_callback_args_t *	p_callback_memory
	Pointer to optional callback argument memory.
void *	p_context
	Pointer to context to be passed into callback function.

◆ layer3_switch_descriptor_queue_status_t

struct layer3_switch_descriptor_queue_status_t		
Status of a descriptor queue.		
Data Fields		
uint32_t	head	Index at the head of the queue. This used for GetDescriptor API.
uint32_t	tail	Index at the tail of the queue. This used for SetDescriptor API.
bool	created	This queue is already created.
layer3_switch_descriptor_queue_cfg_t const *	p_queue_cfg	Configuration of this queue.
bool	rx_available	Indicates whether the queue is available for reception.

◆ layer3_switch_ipv6_filter_address_offset_t

struct layer3_switch_ipv6_filter_address_offset_t		
IP address offset for creating IPv6 filter of Layer3 forwarding.		
Data Fields		
uint8_t	offset	Offset of IPv6 address.
layer3_switch_ip_address_direction_t	direction	Select source or destination.

◆ layer3_switch_l3_stream_filter_cfg_t

struct layer3_switch_l3_stream_filter_cfg_t		
Configuration of stream filter in Layer3 forwarding.		
Data Fields		
uint16_t	filter_field_bitmask	Bitmask of which feilds is enabled in stream filter. Use layer3_switch_l3_filter_bitmask_t.
layer3_switch_ipv6_filter_address_offset_t	ipv6_address0	Offset of IPv6 address 0.
layer3_switch_ipv6_filter_address_offset_t	ipv6_address1	Offset of IPv6 address 1.

◆ layer3_switch_frame_vlan_tag_t

struct layer3_switch_frame_vlan_tag_t		
VLAN tag structure.		
Data Fields		
uint16_t	pcp: 3	Priority Code Point (3 bit).

uint16_t	dei: 1	Drop Eligible Indicator (1 bit).
uint16_t	id: 12	VLAN ID (12 bit).

◆ layer3_switch_frame_filter_t

struct layer3_switch_frame_filter_t		
Frame filter of a MAC/VLAN/Layer3 forwarding entry. Set values for members required for filtering.		
Data Fields		
layer3_switch_table_entry_type_t	entry_type	Type of this entry. Select MAC, VLAN or Layer3.
uint8_t *	p_destination_mac_address	Destination MAC address.
uint8_t *	p_source_mac_address	Source MAC address.
layer3_switch_frame_vlan_tag_t	vlan_c_tag	VLAN C-TAG.
layer3_switch_frame_vlan_tag_t	vlan_s_tag	VLAN S-TAG.
layer3_switch_ip_version_t	ip_version	IP version. Select IPv4, IPv6, or not IP packet.
uint8_t	protocol	IP protocol.
uint8_t *	p_source_ip_address	Source IP address.
uint8_t *	p_destination_ip_address	Destination IP address.
uint16_t	layer4_source_port	TCP/UDP source port.
uint16_t	layer4_destination_port	TCP/UDP destination port.

◆ layer3_switch_l3_filter_t

struct layer3_switch_l3_filter_t		
Store filter information of Layer3 forwarding entry.		
Data Fields		
layer3_switch_frame_filter_t	frame	Target frame format that used to Layer3 forwarding.

◆ layer3_switch_l3_update_config_t

struct layer3_switch_l3_update_config_t		
Configuration of a L2/L3 update feature for output frames.		
Data Fields		
uint32_t	enable_destination_ports	Destination ports that this update config is enabled.
uint32_t	update_field_bitmask	Bit mask of which fields will be updated. Use layer3_switch_l3_update_bitmask_t.
uint8_t *	p_mac_destination_address	MAC Destination Address.

layer3_switch_frame_vlan_tag_t	vlan_c_tag	VLAN C-tag.
layer3_switch_frame_vlan_tag_t	vlan_s_tag	VLAN S-tag.
layer3_switch_forwarding_r_tag_t	r_tag_update_mode	R-TAG update setting.
bool	sequence_number_generation_enable	Flags whether sequence number generation (FRER) is enabled for this entry.

◆ layer3_switch_table_entry_cfg_t

struct layer3_switch_table_entry_cfg_t		
Table entry configuration of MAC/VLAN/Layer3 forwarding.		
Data Fields		
bool	entry_enable	Enable or disable entry. If this field is false, entry will be removed.
bool	security_enable	Entry is secure or not.
layer3_switch_psfp_msdu_filter_cfg_t *	p_psfp_msdu_filter_cfg	Pointer to configuration of PSFP MDSU filter.
layer3_switch_psfp_meter_filter_cfg_t *	p_psfp_meter_filter_cfg	Pointer to configuration of PSFP Meter filter.
layer3_switch_frer_entry_cfg_t *	p_frer_entry_cfg	Configuration of the FRER entry(individual recovery) for this L3 entry (set to NULL : FRER is not valid in this L3 entry).
uint32_t	destination_ports	Destination ports of forwarding.
uint32_t	source_ports	Source ports that enable forwarding of incoming frame.
uint32_t	destination_queue_index	Destination queue. This fields is only used when a destination port is CPU.
uint32_t	internal_priority_update_enable	Enable to update internal priority.
uint32_t	internal_priority_update_value	Internal priority when updating is enabled.
union layer3_switch_table_entry_cfg_t	__unnamed__	

◆ layer3_switch_table_entry_t

struct layer3_switch_table_entry_t
Table entry of MAC/VLAN/Layer3 forwarding.
Data Fields

layer3_switch_frame_filter_t	target_frame	Target frame of forwarding by this entry.
layer3_switch_table_entry_cfg_t	entry_cfg	Configuration of this entry.

◆ layer3_switch_forwarding_port_cfg_t

struct layer3_switch_forwarding_port_cfg_t		
Configuration of forwarding feature for each port.		
Data Fields		
bool	mac_table_enable	Enable MAC table and forwarding feature.
bool	mac_reject_unknown	Reject frame with unknown MAC address.
bool	mac_hardware_learning_enable	Enable hardware learning and migration.
bool	vlan_table_enable	Enable VLAN table and forwarding feature.
bool	vlan_reject_unknown	Reject frame with unknown VLAN ID.
layer3_switch_vlan_ingress_mode_t	vlan_ingress_mode	Select Tag-based VLAN or Port-based VLAN for incoming frame.
layer3_switch_vlan_egress_mode_t	vlan_egress_mode	Tagging/untagging mode for outgoing frame.
layer3_switch_frame_vlan_tag_t	vlan_s_tag	S-TAG of this port. When egress mode is hardware SC-TAG, add this to outgoing frame.
layer3_switch_frame_vlan_tag_t	vlan_c_tag	C-TAG of this port. When egress mode is hardware C-TAG, add this to outgoing frame.
bool	layer3_table_enable	Enable Layer3 table and forwarding feature.
bool	layer3_reject_unknown	Reject frame that not found in Layer3 table.
bool	layer3_ipv4_filter_enable	Enable IPv4 stream filter.
bool	layer3_ipv6_filter_enable	Enable IPv6 stream filter.
bool	layer3_l2_filter_enable	Enable L2 stream filter.

◆ layer3_switch_table_t

struct layer3_switch_table_t
Forwarding table containing MAC/VLAN/Layer3 forwarding entries.
Data Fields

layer3_switch_table_entry_t *	p_mac_entry_list	List of MAC entries.
uint32_t	mac_list_length	Length of the MAC entry list.
layer3_switch_table_entry_t *	p_vlan_entry_list	List of VLAN entries.
uint32_t	vlan_list_length	Length of the VLAN entry list.
layer3_switch_table_entry_t *	p_l3_entry_list	List of Layer3 entries.
uint32_t	l3_list_length	Length of the Layer3 entry list.

◆ layer3_switch_table_cfg_t

struct layer3_switch_table_cfg_t		
Configuration of the forwarding table.		
Data Fields		
layer3_switch_table_t *	p_table	Pointer to forwarding table.
layer3_switch_forwarding_port_cfg_t	port_cfg_list[BSP_FEATURE_ETHER_NUM_CHANNELS+1]	Forwarding configuration of each port.
uint32_t	unsecure_entry_maximum_num	Maximum number of unsecure entries.
bool	mac_entry_aging_enable	Enable aging feature of MAC table.
uint32_t	mac_entry_aging_time_sec	Time[s] to delete an entry by aging.
layer3_switch_vlan_mode_t	vlan_mode	VLAN mode options: NO VLAN, C-TAG, or SC-TAG.
layer3_switch_l3_stream_filter_cfg_t	l3_stream_filter_cfg	Configuration of stream filter in Layer3 forwarding.
layer3_switch_frer_cfg_t	frer_cfg	Configuration of FRER feature.

◆ layer3_switch_extended_cfg_t

struct layer3_switch_extended_cfg_t		
ESWM extension configures each Ethernet port and forwarding feature.		
Data Fields		
ether_phy_instance_t const *	p_ether_phy_instances[BSP_FEATURE_ETHER_NUM_CHANNELS]	List of pointers to ETHER_PHY instance.
gptp_instance_t const *	p_gptp_instance	Pointer to a gPTP instance.
uint32_t	fowarding_target_port_masks[BSP_FEATURE_ETHER_NUM_CHANNELS+1]	List of ports to which incoming frames are forwarded.
uint8_t *	p_mac_addresses[BSP_FEATURE_ETHER_NUM_CHANNELS]	[DEPRECATED] MAC address of each port.
uint32_t	ipv_queue_depth_list[BSP_FEAT	List of IPV queue depth for each

	URE_ETHER_NUM_CHANNELS][BSP_FEATURE_ESWM_ETHA_IPV_QUEUE_NUM]	port.
uint32_t	ipv_queue_preemptable_bitmask[BSP_FEATURE_ETHER_NUM_CHANNELS]	Bitmask of IPV queues that contain preemptable frames.
layer3_switch_preemptable_frame_fragment_size_t	frame_preemption_fragment_size[BSP_FEATURE_ETHER_NUM_CHANNELS]	Minimum fragment size of preemptable frame.
layer3_switch_l3_filter_t *	l3_filter_list	Filter list of Layer3 routing.
uint32_t	l3_filter_list_length	Length of Layer3 filter list.
layer3_switch_port_cfg_t *	p_port_cfg_list[BSP_FEATURE_ETHER_NUM_CHANNELS]	Configuration for each port.
IRQn_Type	etha_error_irq_port_0	ETHA error interrupt number for port 0.
IRQn_Type	etha_error_irq_port_1	ETHA error interrupt number for port 1.
uint8_t	etha_error_ipl_port_0	ETHA error interrupt priority for port 0.
uint8_t	etha_error_ipl_port_1	ETHA error interrupt priority for port 1.
uint8_t	gptp_timer_numbers[BSP_FEATURE_ESWM_GPTP_TIMER_NUM]	List of timer numbers for transmission/reception timestamp.

◆ layer3_switch_instance_ctrl_t

struct layer3_switch_instance_ctrl_t	
LAYER3_SWITCH control block. DO NOT INITIALIZE. Initialization occurs when ether_switch_api_t::open is called.	
Data Fields	
uint32_t	open
	Used to determine if the channel is configured.
ether_switch_cfg_t const *	p_cfg
	Pointer to initial configurations.
R_GWCA0_Type *	p_gwca_reg
	Pointer to GWCA register.

uint32_t	allocated_descriptor_queue_index
	Index of the descriptor pool.
layer3_switch_basic_descriptor_t	p_descriptor_queue_list [LAYER3_SWITCH_CFG_AVAILABLE_QUEUE_NUM]
	Descriptor queue lists used by hardware.
layer3_switch_descriptor_queue_status_t	p_queues_status [LAYER3_SWITCH_CFG_AVAILABLE_QUEUE_NUM]
	Status of each descriptor queues.
layer3_switch_port_cfg_t	p_port_cfg_list [BSP_FEATURE_ETHER_NUM_CHANNELS]
	Configuration for each port.
layer3_switch_table_status_t	table_status
	Forwarding table is initialized or not.
uint32_t	l3_entry_count
	Counts of valid LAYER3 entry.
uint8_t	l3_routing_number
	Routing number for L2/L3 update feature.
uint8_t	l3_remapping_number
	Remapping number for L2/L3 update feature.
layer3_switch_ts_descriptor_queue_status_t	ts_descriptor_queue_status_list [BSP_FEATURE_ESWM_TS_DESCRIPTOR_QUEUE_MAX_NUM]

	Status of TS reception descriptor queues.
layer3_switch_psfp_msdu_filter_info_t	psfp_msdu_filter_info_list [BSP_FEATURE_ESWM_PSFP_MSDU_FILTER_MAX_NUM]
	List of information of PSFP MSDU filter.
layer3_switch_psfp_meter_filter_info_t	psfp_meter_filter_info_list [BSP_FEATURE_ESWM_PSFP_METER_FILTER_SINGLE_BUCKET_METERS_MAX_NUM]
	List of information of PSFP Meter filter.
uint32_t	valid_frer_entry_num
	Number of valid FRER entry.
layer3_switch_frer_sequence_recovery_status_t	frer_sequence_recovery_status [BSP_FEATURE_ESWM_FRER_TABLE_SIZE]
	Status of table for each FRER sequence recovery entry.
uint32_t	used_frer_sequence_generator_num
	Number of the sequence number generator.
bool	frame_preemption_available [BSP_FEATURE_ETHER_NUM_CHANNELS]
	Status of the frame preemption verification.
void(*	p_callback)(ether_switch_callback_args_t *p_args)
	Callback provided when an ISR occurs.
ether_switch_callback_args_t *	p_callback_memory
	Pointer to optional callback argument memory.

void *	p_context
	Pointer to context to be passed into callback function.

◆ layer3_switch_tas_entry_t

struct layer3_switch_tas_entry_t		
Configuration of the gate operation.		
Data Fields		
layer3_switch_tas_gate_state_t	state	Gate state.
uint32_t	time	Time associated with the entry gate state [nsec].

◆ layer3_switch_tas_gate_cfg_t

struct layer3_switch_tas_gate_cfg_t		
Configuration of the gate.		
Data Fields		
layer3_switch_tas_gate_state_t	initial_gate_state	Initial gate state when the cycle starts.
uint8_t	tas_entry_num	Number of TAS entries included in this gate.
layer3_switch_tas_entry_t *	p_tas_entry_list	List of TAS entries included in this gate.

◆ layer3_switch_tas_cfg_t

struct layer3_switch_tas_cfg_t		
Configuration of the TAS.		
Data Fields		
uint8_t	gptp_timer_number	gPTP timer number.
uint32_t	cycle_time_start_high	Upper 32 bits of TAS cycle start time [nsec].
uint32_t	cycle_time_start_low	Lower 32 bits of TAS cycle start time [nsec].
uint32_t	cycle_time	TAS cycle time [nsec].
layer3_switch_tas_gate_cfg_t	gate_cfg_list[8]	List of TAS gate configurations.

Enumeration Type Documentation

◆ layer3_switch_descriptor_type_t

enum layer3_switch_descriptor_type_t	
GWCA descriptor type.	
Enumerator	
LAYER3_SWITCH_DESCRIPTOR_TYPE_LINKFIX	Linkfix. Control element pointing to next descriptor in chain.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_IS	Frame Empty Incremental Start.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_IC	Frame Empty Incremental Continue.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_ND	Frame Empty No Data Storage.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY	Frame Empty.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_START	Frame Empty Start.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_MID	Frame Empty Mid.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_END	Frame Empty End.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE	Frame Single.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FSTART	Frame Start.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FMID	Frame Mid.
LAYER3_SWITCH_DESCRIPTOR_TYPE_FEND	Frame End.
LAYER3_SWITCH_DESCRIPTOR_TYPE_LEMPTY	Link Empty.
LAYER3_SWITCH_DESCRIPTOR_TYPE_EEMPTY	EOS Empty.
LAYER3_SWITCH_DESCRIPTOR_TYPE_LINK	Link.
LAYER3_SWITCH_DESCRIPTOR_TYPE_EOS	End of Set.

◆ layer3_switch_port_bitmask_t

enum layer3_switch_port_bitmask_t	
Bitmasks for each port.	
Enumerator	
LAYER3_SWITCH_PORT_BITMASK_PORT0	Port 0.
LAYER3_SWITCH_PORT_BITMASK_PORT1	Port 1.
LAYER3_SWITCH_PORT_BITMASK_PORT2	Port 2.

◆ layer3_switch_queue_type_t

enum layer3_switch_queue_type_t	
Descriptor queue type.	
Enumerator	
LAYER3_SWITCH_QUEUE_TYPE_RX	Reception queue.
LAYER3_SWITCH_QUEUE_TYPE_TX	Transmission queue.

◆ layer3_switch_descriptor_format_t

enum layer3_switch_descriptor_format_t	
Descriptor format type.	
Enumerator	
LAYER3_SWITCH_DISCRIPTOR_FORMTAT_BASIC	Using basic descriptor.
LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED	Using extended descriptor with additional fields.
LAYER3_SWITCH_DISCRIPTOR_FORMTAT_TX_TIMESTAMP	Using TX timestamp descriptor.

◆ layer3_switch_write_back_mode_t

enum layer3_switch_write_back_mode_t	
Write back mode.	
Enumerator	
LAYER3_SWITCH_WRITE_BACK_MODE_FULL	All fields are updated by hardware.
LAYER3_SWITCH_WRITE_BACK_MODE_DISABLE	No fields are updated by hardware.
LAYER3_SWITCH_WRITE_BACK_MODE_KEEP_DT	Fields exclude DT are updated by hardware.

◆ layer3_switch_ip_version_t

enum layer3_switch_ip_version_t	
IP version options.	
Enumerator	
LAYER3_SWITCH_IP_VERSION_NONE	Not IP packet.
LAYER3_SWITCH_IP_VERSION_IPV4	IPv4 packet.
LAYER3_SWITCH_IP_VERSION_IPV6	IPv6 packet.

◆ layer3_switch_ip_address_direction_t

enum layer3_switch_ip_address_direction_t	
Direction of IP address.	
Enumerator	
LAYER3_SWITCH_IP_ADDRESS_DIRECTION_SOURCE	Use IP source address in stream ID.
LAYER3_SWITCH_IP_ADDRESS_DIRECTION_DESTINATION	Use IP destination address in stream ID.

◆ layer3_switch_magic_packet_detection_t

enum layer3_switch_magic_packet_detection_t	
Magic packet detection options.	
Enumerator	
LAYER3_SWITCH_MAGIC_PACKET_DETECTION_ENABLE	Enable magic packet detection.
LAYER3_SWITCH_MAGIC_PACKET_DETECTION_DISABLE	Disable magic packet detection.

◆ layer3_switch_vlan_mode_t

enum layer3_switch_vlan_mode_t	
VLAN detection mode.	
Enumerator	
LAYER3_SWITCH_VLAN_MODE_NO_VLAN	Not use VLAN feature.
LAYER3_SWITCH_VLAN_MODE_C_TAG	Detect and use VLAN C-TAG.
LAYER3_SWITCH_VLAN_MODE_SC_TAG	Detect and use VLAN SC-TAG.

◆ layer3_switch_vlan_ingress_mode_t

enum layer3_switch_vlan_ingress_mode_t	
VLAN ingress mode determines whether the switch operates with tagged VLAN or port-based VLAN.	
Enumerator	
LAYER3_SWITCH_VLAN_INGRESS_MODE_TAG_BASED	Use VLAN ID of the incoming frame.
LAYER3_SWITCH_VLAN_INGRESS_MODE_PORT_BASED	Use VLAN ID of the incoming port.

◆ layer3_switch_vlan_egress_mode_t

enum layer3_switch_vlan_egress_mode_t	
VLAN egress mode determines the VLAN tag that is added to output frames.	
Enumerator	
LAYER3_SWITCH_VLAN_EGRESS_MODE_NO_VLAN	Frame always outgoing without VLAN TAG.
LAYER3_SWITCH_VLAN_EGRESS_MODE_C_TAG	Frame outgoing with C-TAG of incoming frame.
LAYER3_SWITCH_VLAN_EGRESS_MODE_HW_C_TAG	Frame always outgoing with C-TAG of output port.
LAYER3_SWITCH_VLAN_EGRESS_MODE_SC_TAG	Frame always outgoing with SC-TAG of incoming frame.
LAYER3_SWITCH_VLAN_EGRESS_MODE_HW_SC_TAG	Frame always outgoing with SC-TAG of output port.

◆ layer3_switch_ip_protocol_t

enum layer3_switch_ip_protocol_t	
IP protocols type.	
Enumerator	
LAYER3_SWITCH_IP_PROTOCOL_TCP	IP protocol is TCP.
LAYER3_SWITCH_IP_PROTOCOL_UDP	IP protocol is UDP.

◆ layer3_switch_forwarding_r_tag_t

enum layer3_switch_forwarding_r_tag_t	
R-TAG filed update options.	
Enumerator	
LAYER3_SWITCH_FORWARDING_R_TAG_UPDATE	Incoming frame with R-TAG will be updated with new R-TAG.
LAYER3_SWITCH_FORWARDING_R_TAG_ALWAYS_ADD	ALl incoming frame will be added new R-TAG.
LAYER3_SWITCH_FORWARDING_R_TAG_ALWAYS_ELIMINATE	ALl incoming frame will be removed R-TAG.

◆ layer3_switch_l3_filter_bitmask_t

enum layer3_switch_l3_filter_bitmask_t	
Frame filters options for layer3 forwarding.	
Enumerator	
LAYER3_SWITCH_L3_FILTER_BITMASK_MAC_DESTINATION	Enable MAC destination address.
LAYER3_SWITCH_L3_FILTER_BITMASK_MAC_SOURCE	Enable MAC source address.
LAYER3_SWITCH_L3_FILTER_BITMASK_STAG_ID	Enable VLAN ID of S-TAG.
LAYER3_SWITCH_L3_FILTER_BITMASK_STAG_PCP	Enable PCP of S-TAG.
LAYER3_SWITCH_L3_FILTER_BITMASK_STAG_DEI	Enable DEI of S-TAG.
LAYER3_SWITCH_L3_FILTER_BITMASK_CTAG_ID	Enable VLAN ID of C-TAG.
LAYER3_SWITCH_L3_FILTER_BITMASK_CTAG_PCP	Enable PCP of C-TAG.
LAYER3_SWITCH_L3_FILTER_BITMASK_CTAG_DEI	Enable DEI of C-TAG.
LAYER3_SWITCH_L3_FILTER_BITMASK_IP_SOURCE_ADDRESS	Enable IP source address.
LAYER3_SWITCH_L3_FILTER_BITMASK_IP_DESTINATION_ADDRESS	Enable IP destination address.
LAYER3_SWITCH_L3_FILTER_BITMASK_PROTOCOL	Enable IP protocol.
LAYER3_SWITCH_L3_FILTER_BITMASK_IP_SOURCE_PORT	Enable layer4 source port.
LAYER3_SWITCH_L3_FILTER_BITMASK_IP_DESTINATION_PORT	Enable layer4 destination port.

◆ layer3_switch_l3_update_bitmask_t

enum layer3_switch_l3_update_bitmask_t	
Enable L2/L3 update feature.	
Enumerator	
LAYER3_SWITCH_L3_UPDATE_BITMASK_TTL	Update TTL.
LAYER3_SWITCH_L3_UPDATE_BITMASK_MAC_DESTINATION	Update MAC destination address.
LAYER3_SWITCH_L3_UPDATE_BITMASK_MAC_SOURCE	Update MAC source address.
LAYER3_SWITCH_L3_UPDATE_BITMASK_CTAG_ID	Update VLAN ID of C-TAG.
LAYER3_SWITCH_L3_UPDATE_BITMASK_CTAG_PCP	Update PCP of C-TAG.
LAYER3_SWITCH_L3_UPDATE_BITMASK_CTAG_DEI	Update DEI of C-TAG.
LAYER3_SWITCH_L3_UPDATE_BITMASK_STAG_ID	Update VLAN ID of C-TAG.
LAYER3_SWITCH_L3_UPDATE_BITMASK_STAG_PCP	Update PCP of C-TAG.
LAYER3_SWITCH_L3_UPDATE_BITMASK_STAG_DEI	Update DEI of C-TAG.

◆ layer3_switch_table_entry_type_t

enum layer3_switch_table_entry_type_t	
Table entry type for forwarding feature.	
Enumerator	
LAYER3_SWITCH_TABLE_ENTRY_TYPE_EMPTY	Entry is not initialized yet.
LAYER3_SWITCH_TABLE_ENTRY_TYPE_MAC	Entry of MAC table.
LAYER3_SWITCH_TABLE_ENTRY_TYPE_VLAN	Entry of VLAN table.
LAYER3_SWITCH_TABLE_ENTRY_TYPE_LAYER3	Entry of Layer3 table.

◆ layer3_switch_table_status_t

enum layer3_switch_table_status_t	
Table status for forwarding feature.	
Enumerator	
LAYER3_SWITCH_TABLE_STATUS_UNINITIALIZED	Forwarding table is uninitialized.
LAYER3_SWITCH_TABLE_STATUS_INITIALIZED	Forwarding table is initialized.

◆ layer3_switch_tas_gate_state_t

enum layer3_switch_tas_gate_state_t	
TAS gate state.	
Enumerator	
LAYER3_SWITCH_TAS_GATE_STATE_CLOSE	Gate is closed.
LAYER3_SWITCH_TAS_GATE_STATE_OPEN	Gate is opened.

◆ layer3_switch_rx_timestamp_storage_t

enum layer3_switch_rx_timestamp_storage_t	
Enable or disable rx timestamp storage.	
Enumerator	
LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE	Timestamp is not added in the descriptor.
LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_ENABLE	Timestamp is added in the descriptor.

◆ layer3_switch_ts_descriptor_queue_status_t

enum layer3_switch_ts_descriptor_queue_status_t	
Used or not timestamp descriptor queue.	
Enumerator	
LAYER3_SWITCH_TS_DESCRIPTOR_QUEUE_STATUS_UNUSED	TS descriptor queue is unused.
LAYER3_SWITCH_TS_DESCRIPTOR_QUEUE_STATUS_USED	TS descriptor queue is used.

◆ layer3_switch_preemptable_frame_fragment_size_t

enum layer3_switch_preemptable_frame_fragment_size_t	
Fragment size of preemptable frame.	
Enumerator	
LAYER3_SWITCH_PREEMPTABLE_FRAME_FRAGMENT_SIZE_64BYTE	Minimum fragment size is 64 byte.
LAYER3_SWITCH_PREEMPTABLE_FRAME_FRAGMENT_SIZE_128BYTE	Minimum fragment size is 64 byte.
LAYER3_SWITCH_PREEMPTABLE_FRAME_FRAGMENT_SIZE_192BYTE	Minimum fragment size is 64 byte.
LAYER3_SWITCH_PREEMPTABLE_FRAME_FRAGMENT_SIZE_256BYTE	Minimum fragment size is 64 byte.

◆ layer3_switch_psfp_filter_status_t

enum layer3_switch_psfp_filter_status_t	
Used or not PSFP MSDU filter.	
Enumerator	
LAYER3_SWITCH_PSFP_FILTER_STATUS_UNUSED	PSFP filter is unused.
LAYER3_SWITCH_PSFP_FILTER_STATUS_USED	PSFP filter is used.

◆ layer3_switch_msdu_filter_mode_t

enum layer3_switch_msdu_filter_mode_t	
PSFP MSDU filter mode.	
Enumerator	
LAYER3_SWITCH_MSDU_FILTER_NORMAL_MODE	Filtering occurs if LDESCR.TPL[GWCA] is greater than FWPMPCi.MSDUV.
LAYER3_SWITCH_MSDU_FILTER_THROTTLE_MODE	Filtering occurs if LDESCR.TPL[GWCA] is greater than FWPMPCi.MSDUV. Frames with FWEIS2.PMFS[i] set are also filtered.

◆ layer3_switch_meter_filter_mode_t

enum layer3_switch_meter_filter_mode_t	
PSFP Meter filter mode.	
Enumerator	
LAYER3_SWITCH_METER_FILTER_NORMAL_MODE	Any frame linked to Meter filter i thanks to L3 table received when not enough token is available will be red.
LAYER3_SWITCH_METER_FILTER_THROTTLE_MODE	Any frame linked to Meter filter i thanks to L3 table received when not enough token is available or when FWEIS5.PMRFSi is set will be red.

◆ layer3_switch_meter_filter_coupling_status_t

enum layer3_switch_meter_filter_coupling_status_t	
Enable or disable copling flag.	
Enumerator	
LAYER3_SWITCH_METER_FILTER_COUPLING_STATUS_DISABLE	When green bucket is full, CIR tokens are lost.
LAYER3_SWITCH_METER_FILTER_COUPLING_STATUS_ENABLE	When green bucket is full, CIR tokens are added to the yellow bucket.

◆ layer3_switch_meter_filter_drop_red_frame_t

enum layer3_switch_meter_filter_drop_red_frame_t	
Drop or not red frames.	
Enumerator	
LAYER3_SWITCH_METER_FILTER_NOT_DROP_RED_FRAME	Red frames are not dropped by Meter filter.
LAYER3_SWITCH_METER_FILTER_DROP_RED_FRAME	Red frames are dropped by Meter filter.

◆ layer3_switch_meter_filter_dei_handling_policy_t

enum layer3_switch_meter_filter_dei_handling_policy_t	
DEI handling policy for determining a frame color.	
Enumerator	
LAYER3_SWITCH_METER_FILTER_DEI_HANDLING_POLICY_IGNORE	Ignore the DEI of the VLAN TAG, and the frame color is determined only by the PCP value.
LAYER3_SWITCH_METER_FILTER_DEI_HANDLING_POLICY_YELLOW	When the DEI of the VLAN TAG is 1, treat it as a yellow frame.

◆ layer3_switch_meter_filter_pcp_handling_policy_t

enum layer3_switch_meter_filter_pcp_handling_policy_t	
PCP handling policy for determining a frame color.	
Enumerator	
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_GREEN_ALL	Treat all frames as green, regardless of the PCP.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_1	Treat a frame as yellow when the PCP is less than 1.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_2	Treat a frame as yellow when the PCP is less than 2.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_3	Treat a frame as yellow when the PCP is less than 3.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_4	Treat a frame as yellow when the PCP is less than 4.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_5	Treat a frame as yellow when the PCP is less than 5.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_6	Treat a frame as yellow when the PCP is less than 6.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_BELOW_7	Treat a frame as yellow when the PCP is less than 7.
LAYER3_SWITCH_METER_FILTER_PCP_HANDLING_POLICY_YELLOW_ALL	Treat all frames as yellow, regardless of the PCP.

Function Documentation

◆ R_LAYER3_SWITCH_Open()

fsp_err_t R_LAYER3_SWITCH_Open (ether_switch_ctrl_t *const *p_ctrl*, ether_switch_cfg_t const *const *p_cfg*)

Initializes the switch module and applies configurations. Implements ether_switch_api_t::open.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to control block, config structure or extended config structure is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	Initialization of PHY-LSI failed.
FSP_ERR_INVALID_ARGUMENT	Invalid configuration value.

◆ R_LAYER3_SWITCH_Close()

fsp_err_t R_LAYER3_SWITCH_Close (ether_switch_ctrl_t *const *p_ctrl*)

Disables interrupts and stop module. Implements ether_switch_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	Control block is not open.

◆ R_LAYER3_SWITCH_CreateDescriptorQueue()

fsp_err_t R_LAYER3_SWITCH_CreateDescriptorQueue (ether_switch_ctrl_t *const *p_ctrl*, uint32_t *const *p_queue_index*, const layer3_switch_descriptor_queue_cfg_t *const *p_queue_cfg*)

Create a new descriptor queue and set it to LINKFIX table. This function must be called before calling R_LAYER3_SWITCH_SetDescriptor and R_LAYER3_SWITCH_GetDescriptor.

Return values

FSP_SUCCESS	Descriptor created successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	Control block is not open.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_OUT_OF_MEMORY	Descriptor queue list is depleted.
FSP_ERR_OVERFLOW	TS descriptor queue is used.

◆ R_LAYER3_SWITCH_SetDescriptor()

fsp_err_t R_LAYER3_SWITCH_SetDescriptor (ether_switch_ctrl_t *const *p_ctrl*, uint32_t *queue_index*, layer3_switch_descriptor_t const *const *p_descriptor*)

Set descriptor data to a target descriptor.

Return values

FSP_SUCCESS	Descriptor set successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	Control block is not open.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_INVALID_ARGUMENT	Queue index is invalid.
FSP_ERR_NOT_INITIALIZED	This descriptor queue is not created.
FSP_ERR_IN_USE	Target descriptor is now running.
FSP_ERR_OVERFLOW	Descriptor queue is full.

◆ R_LAYER3_SWITCH_GetDescriptor()

fsp_err_t R_LAYER3_SWITCH_GetDescriptor (ether_switch_ctrl_t *const *p_ctrl*, uint32_t *queue_index*, layer3_switch_descriptor_t *const *p_descriptor*)

Get descriptor data from a target descriptor.

Return values

FSP_SUCCESS	Descriptor got successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	Control block is not open.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_INVALID_ARGUMENT	Queue index is invalid.
FSP_ERR_NOT_INITIALIZED	This descriptor queue is not created or target descriptor is not set.
FSP_ERR_IN_USE	Target descriptor is now running.

◆ R_LAYER3_SWITCH_StartDescriptorQueue()

fsp_err_t R_LAYER3_SWITCH_StartDescriptorQueue (ether_switch_ctrl_t *const *p_ctrl*, uint32_t *queue_index*)

Reload and enable a descriptor queue. In a TX descriptor queue, the queue start transmission. In a RX descriptor queue, the queue start reception.

Return values

FSP_SUCCESS	Descriptor queue started successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	Control block is not open.
FSP_ERR_INVALID_ARGUMENT	Queue index is invalid.
FSP_ERR_NOT_INITIALIZED	Target descriptor queue is not created.
FSP_ERR_IN_USE	Target descriptor queue is already running.
FSP_ERR_INVALID_DATA	Target TX queue have no data.

◆ R_LAYER3_SWITCH_CallbackSet()

fsp_err_t R_LAYER3_SWITCH_CallbackSet (ether_switch_ctrl_t *const *p_ctrl*, void(*)(ether_switch_callback_args_t *) *p_callback*, void *const *p_context*, ether_switch_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_LAYER3_SWITCH_ConfigurePort()

fsp_err_t R_LAYER3_SWITCH_ConfigurePort (ether_switch_ctrl_t *const *p_ctrl*, uint8_t *port*, layer3_switch_port_cfg_t * *p_port_cfg*)

Configure Ethernet port features, including callback function for each port.

Return values

FSP_SUCCESS	Port configured successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_INVALID_ARGUMENT	Port number is invalid.

◆ R_LAYER3_SWITCH_AddTableEntry()

fsp_err_t R_LAYER3_SWITCH_AddTableEntry (ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_frame_filter_t const *const *p_target_frame*, layer3_switch_table_entry_cfg_t const *const *p_entry_cfg*)

Add or update an entry of the forwarding table.

Return values

FSP_SUCCESS	Successfully add/updated an entry into the table.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_OVERFLOW	The number of unsecure entries exceeded the configured value.
FSP_ERR_WRITE_FAILED	A hardware error occurred while learning the entry.
FSP_ERR_INVALID_ARGUMENT	Target frame or entry type is invalid.
FSP_ERR_INVALID_MODE	VLAN feature is disabled and a VLAN entry is passed.
FSP_ERR_NOT_INITIALIZED	The forwarding table has not been initialized.
FSP_ERR_UNSUPPORTED	Function is not supported.
FSP_ERR_BUFFER_EMPTY	PSFP MSDU or Meter filter ID is invalid.

◆ R_LAYER3_SWITCH_SearchTableEntry()

fsp_err_t R_LAYER3_SWITCH_SearchTableEntry (ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_frame_filter_t const *const *p_target_frame*, layer3_switch_table_entry_cfg_t *const *p_entry_cfg*)

Search an entry from the forwarding table.

Return values

FSP_SUCCESS	The entry is found successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_NOT_FOUND	The entry is not found in the table.
FSP_ERR_INVALID_ARGUMENT	Target frame or entry type is invalid.
FSP_ERR_INVALID_MODE	VLAN feature is disabled and a VLAN entry is passed.
FSP_ERR_NOT_INITIALIZED	The forwarding table has not been initialized.
FSP_ERR_UNSUPPORTED	Function is not supported.

◆ R_LAYER3_SWITCH_ConfigureTable()

fsp_err_t R_LAYER3_SWITCH_ConfigureTable (ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_table_cfg_t const *const *p_table_cfg*)

Configure and initialize an forwarding table.

Return values

FSP_SUCCESS	Table configured successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_WRITE_FAILED	Failed to add entries.
FSP_ERR_NOT_INITIALIZED	The forwarding table has not been initialized.
FSP_ERR_OVERFLOW	The number of unsecure entries exceeded the configured value.
FSP_ERR_INVALID_ARGUMENT	Target frame or entry type is invalid.
FSP_ERR_INVALID_MODE	VLAN feature is disabled and a VLAN entry is passed.
FSP_ERR_UNSUPPORTED	Function is not supported.

◆ R_LAYER3_SWITCH_GetTable()

fsp_err_t R_LAYER3_SWITCH_GetTable (ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_table_t *const *p_table*)

Get a pointer to the forwarding table.

Return values

FSP_SUCCESS	Table got successfully.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_INVALID_MODE	VLAN feature is disabled and a VLAN entry list is passed.
FSP_ERR_NOT_INITIALIZED	The forwarding table has not been initialized.
FSP_ERR_NOT_FOUND	The entry isn't found.
FSP_ERR_INVALID_ARGUMENT	Invalid configuration value.
FSP_ERR_UNSUPPORTED	Function is not supported.

◆ R_LAYER3_SWITCH_ConfigureTAS()

fsp_err_t R_LAYER3_SWITCH_ConfigureTAS (ether_switch_ctrl_t *const *p_ctrl*, uint8_t *port*, layer3_switch_tas_cfg_t * *p_tas_cfg*)

Configure Time Aware Shaper feature.

Return values

FSP_SUCCESS	TAS configure successfully.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to a argument is NULL.
FSP_ERR_INVALID_ARGUMENT	Port number is invalid.
FSP_ERR_ASSERTION	Pointer to control block is NULL.
FSP_ERR_UNSUPPORTED	TAS feature is not enabled in the configuration.

◆ R_LAYER3_SWITCH_EnableTAS()

fsp_err_t R_LAYER3_SWITCH_EnableTAS (ether_switch_ctrl_t *const *p_ctrl*, uint8_t *port*)

Enable Time Aware Shaper feature.

Return values

FSP_SUCCESS	TAS enabled successfully.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Port number is invalid.
FSP_ERR_UNSUPPORTED	TAS feature is not enabled in the configuration.
FSP_ERR_ASSERTION	Pointer to control block is NULL.

◆ R_LAYER3_SWITCH_PsfpClearErrorStatus()

fsp_err_t R_LAYER3_SWITCH_PsfpClearErrorStatus (ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_psfp_error_status_bitmask_t *bitmasks*)

Clear error status bit of throttle mode.

Return values

FSP_SUCCESS	Status bit cleared successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	Function is not supported.

5.2.12.4 Ethernet (r_rmac)

Modules » Networking

Functions

fsp_err_t	R_RMAC_Open (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg) After RMAC, Switch and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open. More...
fsp_err_t	R_RMAC_Close (ether_ctrl_t *const p_ctrl)

	Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close. More...
fsp_err_t	R_RMAC_BufferRelease (ether_ctrl_t *const p_ctrl) Move to the next buffer in the receive buffer list. Implements ether_api_t::bufferRelease. More...
fsp_err_t	R_RMAC_RxBufferUpdate (ether_ctrl_t *const p_ctrl, void *const p_buffer) Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate. More...
fsp_err_t	R_RMAC_LinkProcess (ether_ctrl_t *const p_ctrl) The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess. More...
fsp_err_t	R_RMAC_WakeOnLANEnable (ether_ctrl_t *const p_ctrl) The setting of RMAC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable. More...
fsp_err_t	R_RMAC_Read (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes) Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read. More...
fsp_err_t	R_RMAC_Write (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length) Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write. More...
fsp_err_t	R_RMAC_TxStatusGet (ether_ctrl_t *const p_ctrl, void *const p_buffer_address)
fsp_err_t	R_RMAC_CallbackSet (ether_ctrl_t *const p_api_ctrl,

	void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory)
fsp_err_t	R_RMAC_SetWriteConfig (ether_ctrl_t *const p_ctrl, rmac_write_cfg_t *const p_write_cfg) Set configuration for tx frame. This API must call before ether_api_t::write. More...
fsp_err_t	R_RMAC_GetTxTimestamp (ether_ctrl_t *const p_ctrl, rmac_timestamp_t *const p_timestamp) Get timestamp of transmitted frame. This API must call after ether_api_t::write. More...
fsp_err_t	R_RMAC_GetRxTimestamp (ether_ctrl_t *const p_ctrl, rmac_timestamp_t *const p_timestamp) Get timestamp of received frame. This API must call before ether_api_t::read. More...

Detailed Description

Driver for the Ethernet peripheral on RX MCUs. This module implements the Ethernet Interface.

Overview

This module transmits and receives Ethernet (IEEE 802.3) frames. This module provides single MAC layer interface.

The following figure shows which IPs are used by the r_rmac module.

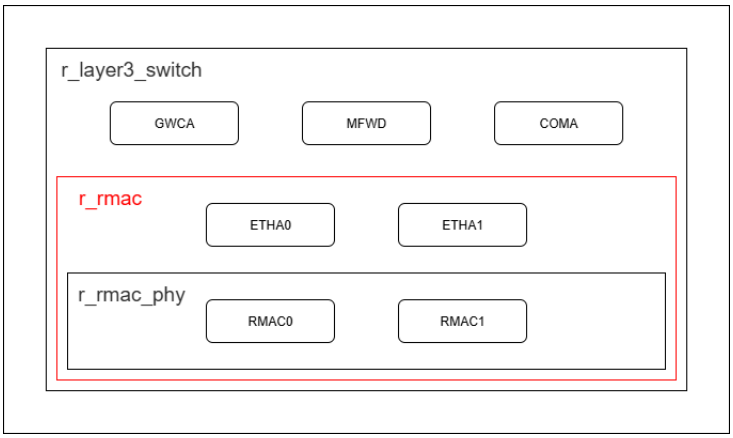


Figure 146: ESWM block diagram

Features

The RMAC module supports the following features:

- Transmit/Receive processing
- Optional zero-copy buffering
- Callback function with returned event code
- Auto negotiation support
- Flow control support
- Multicast filtering support
- Broadcast filtering support
- Promiscuous mode support
- Get the frame transmission and reception times
- Port based forwarding function for transmission/reception
- Descriptor format selection when transmitting

Configuration

Build Time Configurations for r_rmac

The following build time configurations are defined in fsp_cfg/r_rmac_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Read Buffer Size Checking	• Enabled • Disabled	Disabled	If selected code for buffer size checking is included in the build.

Configurations for Networking > Ethernet MAC (r_rmac)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet MAC (r_rmac).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_ether0	Module name.
General > Channel	Must be an integer greater than or equal to 0.	0	Write the ether channel number.
General > Zero-copy Mode	• Disable • Enable	Disable	Enable or disable zero-copy mode.
General > Flow control functionality	• Disable • Enable	Disable	Enable or disable flow control.
General > MAC address	Must be a valid MAC address	00:11:22:33:44:55	MAC address of this channel.
Filters > Multicast Mode	• Disable • Enable	Enable	Enable or disable multicast frame reception.
Filters > Promiscuous	• Disable	Disable	Enable this option to

Mode	• Enable		receive packets addressed to other NICs.
Filters > Broadcast filter	Must be a valid non-negative integer with maximum configurable value of 65535.	0	Limit of the number of broadcast frames received continuously
Buffers > Padding size	Disable	Disable	The padding size that is automatically inserted into the received packets
Buffers > Padding offset	Must be less than 64 bytes.	0	The offset into a receive buffer to insert padding bytes.
Buffers > TX queue length	Value must be an integer	3	Number of buffers per TX queue.
Buffers > RX queue length	Value must be an integer	3	Number of buffers per RX queue.
Buffers > TX queue num	Value must be an integer	2	TX queue num
Buffers > RX queue num	Value must be an integer	2	RX queue num
Buffers > Number of TX buffer	Value must be an integer	12	Number of transmit buffers
Buffers > Number of RX buffer	Value must be an integer	12	Number of receive buffers
Buffers > Buffer size	Must be at least 1514 which is the maximum Ethernet frame size.	1514	Size of Ethernet buffer
Buffers > Allocate RX buffer	• Disable • Enable	Enable	Allocates the RX buffer when generating the configuration structure
Buffers > Number of node num	Value must be an integer	24	Number of receive buffers
Interrupts > Magic packet interrupt priority	MCU Specific Options		Select the magic packet detection interrupt priority.
Interrupts > Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs
Timestamp > TS queue length	Value must be an integer	8	Timestamp queue length.
Forwarding > Transmission descriptor format	• Transmission ethernet descriptor • Transmission	Transmission direct descriptor	The transmission direct descriptor could be used, for example, for non-routable protocol

	direct descriptor		forwarding (network control) or for CPU-CPU communication and the transmission ethernet descriptor could be used for all normal ethernet transmissions.
Forwarding > Port based forwarding for reception	• Enabled • Disabled	Disabled	Port based forwarding function forwards all frames from a port to other ones when receiving.

Interrupt Configuration

The first R_RMAC_Open function call sets GWDI interrupts. The user could provide callback function which would be invoked when GWDI interrupt handler has been completed. The callback arguments will contain information about a channel number, the event code and a pointer to the user defined context.

Callback Configuration

The user could provide callback function which would be invoked when either a magic packet or a link signal change is detected. When the callback function is called, a variable in which the channel number for which the detection occurred and a constant shown in Table 2.4 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, its value should be copied into, for example, a global variable.

Clock Configuration

The ESWM peripheral uses ICLK, PCLKA, ESWCLK and ESWPHYCLK as clocks. These clocks must be set to frequencies within the ranges shown in the following table.

Interface Mode	PHY speed [Mbps]	ICLK [MHz]	PCLKA [MHz]	ESWCLK [MHz]	ESWPHYCLK [MHz]
MII/RMII/RGMII	10/100	50 to 300	up to 150	50 to 250	500 (RGMII only)
GMII/RGMII	1000	150 to 300	up to 150	150 to 250	500

Note

1. Clock restrictions are as follow: ICLK × 1.5 => ESWCLK, ESWCLK > PCLKA and ICLK > PCLKA.

Pin Configuration

To use the RMAC module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). Please perform the pin setting before calling the R_RMAC_Open function.

Usage Notes

Ethernet Frame Format

The RMAC module supports the Ethernet II/IEEE 802.3 frame format.

Frame Format for Data Transmission and Reception

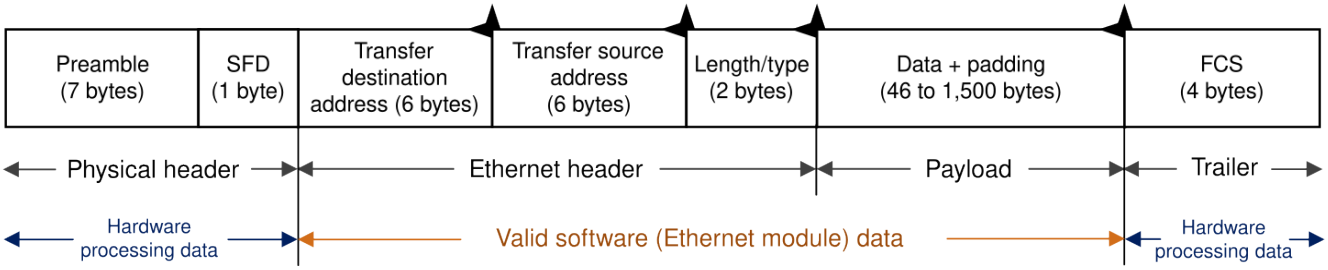


Figure 147: Frame Format Image

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match. When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

PAUSE Frame Format

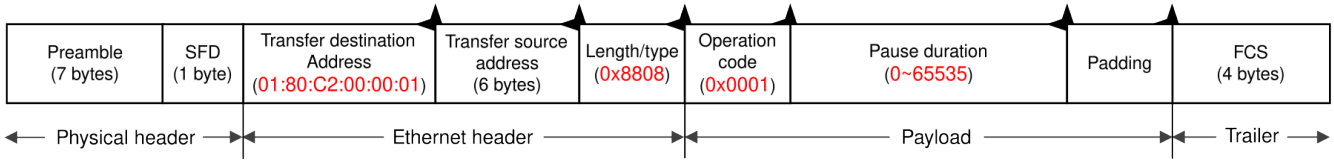


Figure 148: Pause Frame Format Image

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and the operation code as 0x0001. The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

Magic Packet Frame Format

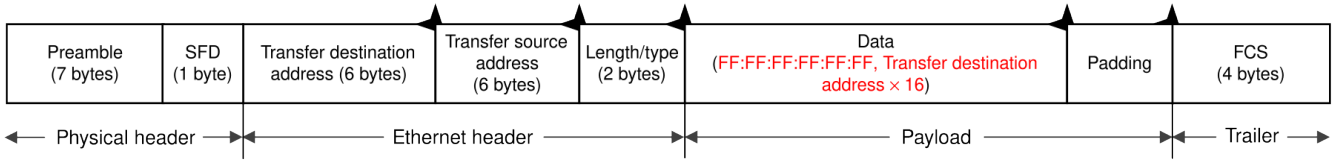


Figure 149: Magic Packet Frame Format Image

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

Examples

ETHER Basic Example

This is a basic example of minimal use of the ETHER in an application.

Note

In this example zero-copy mode is disabled and there are no restrictions on buffer alignment.

```
#define RMAC_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE (1514)
#define RMAC_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)
#define RMAC_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01
#define RMAC_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x02
#define RMAC_EXAMPLE_FRAME_TYPE 0x00, 0x2E
#define RMAC_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

/* Receive data buffer */
static uint8_t gp_read_buffer[RMAC_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE] = {0};
/* Transmit data buffer */
static uint8_t gp_send_data[RMAC_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    RMAC_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    RMAC_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    RMAC_EXAMPLE_FRAME_TYPE,             /* Type field */
    RMAC_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void rmac_basic_example (void)
{
 fsp_err_t err            = FSP_SUCCESS;
    uint32_t read_data_size = 0;
 /* Open the rmac instance with initial configuration. */
    err = R_RMAC_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
```

```
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_RMAC_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Transmission is non-blocking. */
 /* User data copy to internal buffer and is transferred by DMA in the background. */
    err = R_RMAC_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
    assert(FSP_SUCCESS == err);
 /* Received data copy to user buffer from internal buffer. */
    err = R_RMAC_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the rmac instance. */
 R_RMAC_Close(&g_ether0_ctrl);
}
```

Timestamp Example

This is a example of getting transmission and reception timestamps.

```
void rmac_timestamp_example (void)
{
 fsp_err_t err            = FSP_SUCCESS;
    uint32_t read_data_size = 0;
 rmac_timestamp_t rx_timestamp;
 rmac_timestamp_t tx_timestamp;
 rmac_write_cfg_t write_cfg = {.tx_timestamp_enable = 1};
 /* Open the rmac instance with initial configuration. */
    err = R_RMAC_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
```

```
up,
  * Initializes the module and make auto negotiation. */
       err = R_RMAC_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set write configuration. */
    err = R_RMAC_SetWriteConfig(&g_ether0_ctrl, &write_cfg);
    assert(FSP_SUCCESS == err);
 /* Transmission is non-blocking. */
 /* User data copy to internal buffer and is transferred by DMA in the background. */
    err = R_RMAC_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
    assert(FSP_SUCCESS == err);
 /* Get TX timestamp value. */
    err = R_RMAC_GetTxTimestamp(&g_ether0_ctrl, &tx_timestamp);
    assert(FSP_SUCCESS == err);
 /* Get RX timestamp value. */
    err = R_RMAC_GetRxTimestamp(&g_ether0_ctrl, &rx_timestamp);
    assert(FSP_SUCCESS == err);
 /* Received data copy to user buffer from internal buffer. */
    err = R_RMAC_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the rmac instance. */
 R_RMAC_Close(&g_ether0_ctrl);
}
```

Function Documentation

◆ R_RMAC_Open()

fsp_err_t R_RMAC_Open (ether_ctrl_t *const *p_ctrl*, ether_cfg_t const *const *p_cfg*)

After RMAC, Switch and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open.

Initializes the ether module and applies configurations. Implements ether_api_t::open.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block or configuration structure is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	Initialization of PHY-LSI failed.
FSP_ERR_INVALID_CHANNEL	Invalid channel number is given.
FSP_ERR_INVALID_POINTER	Pointer to extend config structure or MAC address is NULL.
FSP_ERR_INVALID_ARGUMENT	Interrupt is not enabled.
FSP_ERR_UNSUPPORTED	Zerocopy mode and padding feature are not supported.

◆ R_RMAC_Close()

fsp_err_t R_RMAC_Close (ether_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_RMAC_BufferRelease()

fsp_err_t R_RMAC_BufferRelease (ether_ctrl_t *const *p_ctrl*)

Move to the next buffer in the receive buffer list. Implements ether_api_t::bufferRelease.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block or internal buffers is NULL.
FSP_ERR_BUFFER_EMPTY	There is no available internal RX buffer.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_OVERFLOW	Value of the send frame size is larger than the buffer size.

◆ R_RMAC_RxBufferUpdate()

fsp_err_t R_RMAC_RxBufferUpdate (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*)

Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	A pointer argument is NULL.
FSP_ERR_BUFFER_EMPTY	There is no available internal RX buffer.
FSP_ERR_INVALID_POINTER	The pointer of buffer is NULL or not aligned on a 32-bit boundary.
FSP_ERR_INVALID_MODE	Driver is configured to non zero copy mode.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_OVERFLOW	Value of the send frame size is larger than the buffer size.

◆ R_RMAC_LinkProcess()

fsp_err_t R_RMAC_LinkProcess (ether_ctrl_t *const *p_ctrl*)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess.

Return values

FSP_SUCCESS	Link is up.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_LINK	Link is down.
FSP_ERR_ETHER_PHY_ERROR_LINK	Initialization of the PHY-LSI failed.

◆ R_RMAC_WakeOnLANEnable()

fsp_err_t R_RMAC_WakeOnLANEnable (ether_ctrl_t *const *p_ctrl*)

The setting of RMAC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_ETHER_PHY_ERROR_LINK	Initialization of PHY-LSI failed.

◆ R_RMAC_Read()

fsp_err_t R_RMAC_Read (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t *const *length_bytes*)

Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_NO_DATA	There is no data in receive buffer.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_INVALID_POINTER	Value of the pointer is NULL.
FSP_ERR_OVERFLOW	Value of the send frame size is larger than the buffer size.
FSP_ERR_BUFFER_EMPTY	There is no available internal RX buffer.
FSP_ERR_INVALID_SIZE	The buffer size is smaller than ether_cfg_t::ether_buffer_size.

◆ R_RMAC_Write()

fsp_err_t R_RMAC_Write (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t const *frame_length*)

Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL	Transmit buffer is not empty.
FSP_ERR_INVALID_POINTER	Value of the pointer is NULL.
FSP_ERR_INVALID_ARGUMENT	Value of the send frame size is out of range.
FSP_ERR_OVERFLOW	Value of the send frame size is larger than the buffer size.

◆ R_RMAC_TxStatusGet()

fsp_err_t R_RMAC_TxStatusGet (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer_address*)

Provides status of Ethernet driver in the user provided pointer. Implements ether_api_t::txStatusGet.

Return values

FSP_SUCCESS	Transmit buffer address is stored in provided p_buffer_address.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_FOUND	Transmit buffer address has been overwritten in transmit descriptor.
FSP_ERR_INVALID_POINTER	p_status is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_RMAC_CallbackSet()

fsp_err_t R_RMAC_CallbackSet (ether_ctrl_t *const *p_api_ctrl*, void(*)(ether_callback_args_t *) *p_callback*, void *const *p_context*, ether_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements ether_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_RMAC_SetWriteConfig()

fsp_err_t R_RMAC_SetWriteConfig (ether_ctrl_t *const *p_ctrl*, rmac_write_cfg_t *const *p_write_cfg*)

Set configuration for tx frame. This API must call before ether_api_t::write.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	A pointer argument is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to write configuration structure is NULL.

◆ R_RMAC_GetTxTimestamp()

fsp_err_t R_RMAC_GetTxTimestamp (ether_ctrl_t *const *p_ctrl*, rmac_timestamp_t *const *p_timestamp*)

Get timestamp of transmitted frame. This API must call after ether_api_t::write.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	A pointer argument is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to timestamp structure is NULL.

◆ R_RMAC_GetRxTimestamp()

fsp_err_t R_RMAC_GetRxTimestamp (ether_ctrl_t *const *p_ctrl*, rmac_timestamp_t *const *p_timestamp*)

Get timestamp of received frame. This API must call before ether_api_t::read.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	A pointer argument is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	Pointer to timestamp structure is NULL.

5.2.12.5 Ethernet (r_rmac_phy)

Modules » Networking

Functions

fsp_err_t	R_RMAC_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) Resets Ethernet PHY device. Implements ether_phy_api_t::open. More...
fsp_err_t	R_RMAC_PHY_Close (ether_phy_ctrl_t *const p_ctrl) Close Ethernet PHY device. Implements ether_phy_api_t::close. More...
fsp_err_t	R_RMAC_PHY_ChipInit (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) Initialize Ethernet PHY device. Implements ether_phy_api_t::chipInit. More...
fsp_err_t	R_RMAC_PHY_Read (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data) Read data from register of PHY-LSI . Implements ether_phy_api_t::read. More...
fsp_err_t	R_RMAC_PHY_Write (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)

	Write data to register of PHY-LSI . Implements ether_phy_api_t::write. More...
fsp_err_t	R_RMAC_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl) Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More...
fsp_err_t	R_RMAC_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause) Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More...
fsp_err_t	R_RMAC_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl) Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet. More...
fsp_err_t	R_RMAC_PHY_ChipSelect (ether_phy_ctrl_t *const p_ctrl, uint8_t port) Update the target PHY LSI of this driver. More...
fsp_err_t	R_RMAC_PHY_CallbackSet (ether_phy_ctrl_t *const p_ctrl, void(*p_callback)(ether_phy_callback_args_t *), void *const p_context, ether_phy_callback_args_t *const p_callback_memory)

Detailed Description

The RMAC PHY module (r_rmac_phy) provides an API for standard Ethernet PHY communications applications that use the RMAC peripheral. It implements the Ethernet PHY Interface.

Overview

The RMAC PHY module is used to setup and manage an external Ethernet PHY device. r_rmac_phy use the on-chip MAC layer peripheral(RMAC), which is part of Ethernet Switch Module(ESWM). It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

The following figure shows which IPs are used by the r_rmac_phy module.

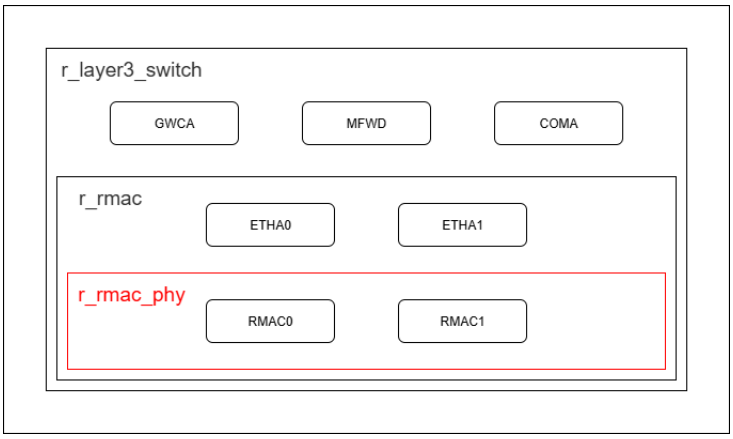


Figure 150: ESWM block diagram

Features

The RMAC PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support
- Gigabit Ethernet support

Following features are not supported:

- Extensions to management frame format defined in IEEE802.3 clause 45

Configuration

Build Time Configurations for r_rmac_phy

The following build time configurations are defined in fsp_cfg/r_rmac_phy_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
KSZ8091RNB Target	• Disabled • Enabled	Disabled	Select whether to use KSZ8091RNB PHY-LSI or not.
KSZ8041 Target	• Disabled • Enabled	Disabled	Select whether to use KSZ8041 PHY-LSI or not.
DP83620 Target	• Disabled • Enabled	Disabled	Select whether to use DP83620 PHY-LSI or not.
ICS1894 Target	• Disabled • Enabled	Disabled	Select whether to use ICS1894 PHY-LSI or not.

GPY111 Target	• Disabled • Enabled	Disabled	Select whether to use GPY111 PHY-LSI or not.
VSC8541 Target	• Disabled • Enabled	Disabled	Select whether to use VSC8541 PHY-LSI or not.
KSZ8081MNX Target	• Disabled • Enabled	Disabled	Select whether to use KSZ8081MNX PHY-LSI or not.
User Own Target	• Disabled • Enabled	Disabled	Select whether to use User own PHY-LSI or not.
Reference Clock	• Default • Enabled • Disabled	Default	Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board.
Reference Clock	• Default • Enabled • Disabled	Default	Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board.

Configurations for Networking > Ethernet (r_rmac_phy)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_rmac_phy).

Configuration	Options	Default	Description
Interrupts > Context	Name must be a valid C symbol	NULL	Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t.
Interrupts > Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs
Interrupts > ETHA0 status interrupt priority	MCU Specific Options		Select the ETHA0 status interrupt priority.
Interrupts > ETHA1 status interrupt priority	MCU Specific Options		Select the ETHA1 status interrupt priority.
Name	Name must be a valid C symbol	g_rmac_phy0	Module name.

Channel	• 0 • 1	0	Select the Ethernet controller channel number.
Default PHY-LSI port	• 0 • 1	0	Specify the default port for PHY-LSI configuration.
PHY-LSI Reset Completion Timeout	Specify a value between 0x1 and 0xFFFFFFFF.	0x00020000	Specify the number of times to read the PHY-LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used.
Select MII type	• MII • RMII • GMII • RGMII	GMII	Specify whether to use MII or RMII.
Port Custom Init Function	Name must be a valid C symbol	NULL	Set the initial function of the PHY-LSI, When using your own PHY-LSI.
Port Custom Link Partner Ability Get Function	Name must be a valid C symbol	NULL	Set the link partner ability get function of the PHY-LSI, When using your own PHY-LSI.
Flow Control	• Disable • Enable	Disable	Select whether to enable or disable flow control.
Management frame format	Clause 22 frame format	Clause 22 frame format	Select the management frame format used to access the PHY-LSI.
MDC clock rate (Hz)	Value must be a non-negative number	2500000	Specify the MDC clock frequency
MDIO hold timing adjustment	Value must be a non-negative integer less than 8	0	Specify the clock cycle for hold timing.
MDIO capture timing adjustment	Value must be a non-negative integer less than 8	0	Specify the clock cycle for capture timing.
Frame preemption	• Disable • Enable	Disable	Select whether to enable or disable frame preemption feature.
Verify frame interval	Value must be a non-negative integer less	10	Interval for sending verify frame [ms].

than 128

Usage Notes

Accessing the Registers in PHY-LSI

Use the RMAC peripheral to access the registers in the PHY-LSI. Management frame is transmitted and received through the ETn_MDC and ETn_MDIO pins controlled by hardware.

This HW supports two management frame format.

- Normal management frame format defined in Clause 22 of IEEE802.3 (Describe "MDIO" in this document)
- Extension management frame format defined in Clause 45 of IEEE802.3 (Describe "eMDIO" in this document)

Management frame format of MDIO (Clause 22)

The below table lists the management frame formats of MDIO.

Access type	MII and RMII management frame							
Item	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
Number of bits	32	2	2	5	5	2	16	1
Read	1...1	01	10	00001	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z
Write	1...1	01	01	00001	RRRRR	10	DDDDDDDDDDDDDDDD	Z

Note

- PRE (preamble): Send 32 consecutive 1s.
- ST (start of frame): Send 01b.
- OP (operation code): Send 10b for read or 01b for write.
- PHYAD (PHY address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the PHY-LSI address is 1, send 00001b.
- REGAD (register address): One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.
- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.
Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).
(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)
- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.
- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write
operation (Z is output). No control is required, because a bus was already released during a read operation.

Management frame format of eMDIO (Clause 45)

The below table lists the management frame formats of eMDIO(Clause 45).

Access type	MII and RMII management frame							
Item	PRE	ST	OP	PRTAD	DEVAD	TA	DATA	IDLE
Number of bits	32	2	2	5	5	2	16	1
Address	1...1	00	00	PPPPP	EEEEE	10	AAAAAAAAAAAAAAAA	Z
Write	1...1	00	01	PPPPP	EEEEE	10	DDDDDDDDDDDDDDDD	Z
Read	1...1	00	11	PPPPP	EEEEE	Z0	DDDDDDDDDDDDDDDD	Z
Read Increment	1...1	00	10	PPPPP	EEEEE	Z0	DDDDDDDDDDDDDDDD	Z

Note

- PRE (preamble): Send 32 consecutive 1s.
- ST (start of frame): Send 00b.
- OP (operation code): Send 00b for register address, 10b for read operation, 11b for write operation, or 10b for post-read-increment-address operation.
- PRTAD (port address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits port address. When the port address is 1, send 00001b.
- DEVAD (device address): Up to 32 MDIO Manageable Devices(MMD) are contained in a PHY-LSI. MMD are selected with these 5 bits device address.
- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.
Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).
(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)
- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.
- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write
operation (Z is output). No control is required, because a bus was already released during a read operation.

Limitations

- The r_rmac_phy module may need to be customized for PHY devices other than the ones currently supported. Use the existing code as a starting point for creating a custom implementation. Supported PHY devices are as follows.
  - KSZ8091RNB
  - KSZ8041
  - DP83620
  - ICS1894

- GPY111
- VSC8541
- KSZ8081MNX

Examples

RMAC PHY Basic Example

This is a basic example of minimal use of the RMAC PHY in an application.

```
void rmac_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_rmac_phy0_ctrl.open   = 0U;
    g_rmac_phy0_cfg.channel = 0;
 /* Initializes the module. */
    err = R_RMAC_PHY_Open(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Polling until link is established. */
 while (FSP_SUCCESS != R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl))
    {
 /* Do nothing */
    }
 /* Get link partner ability from phy interface. */
    err = R_RMAC_PHY_LinkPartnerAbilityGet(&g_rmac_phy0_ctrl,
                                           &g_rmac_phy0_line_speed_duplex,
                                           &g_rmac_phy0_local_pause,
                                           &g_rmac_phy0_partner_pause);
    assert(FSP_SUCCESS == err);
 /* Check current link status. */
    err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rmac_phy_instance_ctrl_t
struct	rmac_phy_extended_cfg_t

Enumerations

enum	rmac_phy_interface_status_t
enum	rmac_phy_frame_format_t

Data Structure Documentation

◆ rmac_phy_instance_ctrl_t

struct rmac_phy_instance_ctrl_t

RMAC PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called.

Data Fields

uint32_t	open
	Used to determine if the channel is configured.
ether_phy_cfg_t const *	p_ether_phy_cfg
	Pointer to initial configurations.
R_RMAC0_Type *	p_reg_rmac
	Pointer to RMAC peripheral registers.
uint32_t	local_advertise
	Capabilities bitmap for local advertising.
rmac_phy_interface_status_t	interface_status
	Initialized status of ETHER PHY interface.
uint8_t	phy_lsi_cfg_index

	Index of the PHY LSI that is currently the target of operation.
void(*	p_callback)(ether_phy_callback_args_t *p_args)
	Callback provided when an ISR occurs.

◆ rmac_phy_extended_cfg_t

struct rmac_phy_extended_cfg_t	
RMAC PHY extended configuration.	
Data Fields	
void(*	p_target_init)(rmac_phy_instance_ctrl_t *p_instance_ctrl)
	Pointer to callback that is called to initialize the target.
bool(*	p_target_link_partner_ability_get)(rmac_phy_instance_ctrl_t *p_instance_ctrl, uint32_t line_speed_duplex)
	Pointer to callback that is called to get the link partner ability.
rmac_phy_frame_format_t	frame_format
	Whether the management frame format is MDIO or eMDIO.
uint32_t	mdc_clock_rate
	MDC frequency division.
uint8_t	mdio_hold_time
	MDIO hold time adjustment.
uint8_t	mdio_capture_time
	MDIO capture time adjustment.
ether_phy_lsi_cfg_t const *	p_phy_lsi_cfg_list [BSP_FEATURE_ETHER_NUM_CHANNELS]

	Pointer list of PHY LSI configurations.
uint8_t	default_phy_lsi_cfg_index
	Index of the default PHY LSI condiguration.
bool	frame_preemption_enable
	Whether to enable frame preemption or not.
uint32_t	frame_preemption_verification_interval
	Interval for sending verify frame [ms].
IRQn_Type	easi_irq [BSP_FEATURE_ETHER_NUM_CHANNELS]
	EASI interrupt number.
uint32_t	easi_ipl [BSP_FEATURE_ETHER_NUM_CHANNELS]
	EASI interrupt priority.
void(*	p_callback)(ether_phy_callback_args_t *p_args)
	Callback provided when an ISR occurs.
void *	p_context
	Placeholder for user data.

Enumeration Type Documentation

◆ rmac_phy_interface_status_t

enum rmac_phy_interface_status_t	
Initialization state for read/write	
Enumerator	
RMAC_PHY_INTERFACE_STATUS_UNINITIALIZED	ETHER PHY interface is uninitialized.
RMAC_PHY_INTERFACE_STATUS_INITIALIZED	ETHER PHY interface is initialized.

◆ rmac_phy_frame_format_t

enum rmac_phy_frame_format_t	
Enumerator	
RMAC_PHY_FRAME_FORMAT_MDIO	Normal management frame format defined in clause 22.
RMAC_PHY_FRAME_FORMAT_EMDIO	Extension management frame format defined in clause 45.

Function Documentation

◆ R_RMAC_PHY_Open()

fsp_err_t R_RMAC_PHY_Open (ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg*)

Resets Ethernet PHY device. Implements ether_phy_api_t::open.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_INVALID_CHANNEL	Invalid channel number is given.
FSP_ERR_INVALID_POINTER	Pointer to p_cfg is NULL.
FSP_ERR_INVALID_ARGUMENT	Invalid configuration value.
FSP_ERR_INVALID_MODE	Function is called when not in DISABLE mode.

◆ R_RMAC_PHY_Close()

fsp_err_t R_RMAC_PHY_Close (ether_phy_ctrl_t *const *p_ctrl*)

Close Ethernet PHY device. Implements ether_phy_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_INVALID_MODE	Function is called when not in DISABLE mode.

◆ R_RMAC_PHY_ChipInit()

fsp_err_t R_RMAC_PHY_ChipInit (ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg*)

Initialize Ethernet PHY device. Implements ether_phy_api_t::chipInit.

Return values

FSP_SUCCESS	PHY device initialized successfully.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block or configuration structure is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_TIMEOUT	PHY-LSI Reset wait timeout.
FSP_ERR_INVALID_MODE	Function is called when not in OPERATION mode.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized
FSP_ERR_INVALID_ARGUMENT	Address is not a valid size

◆ R_RMAC_PHY_Read()

fsp_err_t R_RMAC_PHY_Read (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *const *p_data*)

Read data from register of PHY-LSI . Implements ether_phy_api_t::read.

Return values

FSP_SUCCESS	RMAC_PHY successfully read data.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_INVALID_POINTER	Pointer to read buffer is NULL.
FSP_ERR_INVALID_ARGUMENT	Address is not a valid size
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized

◆ R_RMAC_PHY_Write()

fsp_err_t R_RMAC_PHY_Write (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *data*)

Write data to register of PHY-LSI . Implements ether_phy_api_t::write.

Return values

FSP_SUCCESS	RMAC_PHY successfully write data.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_INVALID_ARGUMENT	Address or data is not a valid size
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized
FSP_ERR_INVALID_MODE	Function is called when not in OPERATION mode.

◆ R_RMAC_PHY_StartAutoNegotiate()

fsp_err_t R_RMAC_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

Return values

FSP_SUCCESS	RMAC_PHY successfully starts auto-negotiate.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized
FSP_ERR_INVALID_MODE	Function is called when not in OPERATION mode.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_INVALID_ARGUMENT	Address is not a valid size

◆ R_RMAC_PHY_LinkPartnerAbilityGet()

fsp_err_t R_RMAC_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause*)

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

Return values

FSP_SUCCESS	RMAC_PHY successfully get link partner ability.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY-LSI is not link up.
FSP_ERR_ETHER_PHY_NOT_READY	The auto-negotiation isn't completed
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized
FSP_ERR_INVALID_MODE	Function is called when not in OPERATION mode.
FSP_ERR_INVALID_ARGUMENT	Address is not a valid size

◆ R_RMAC_PHY_LinkStatusGet()

fsp_err_t R_RMAC_PHY_LinkStatusGet (ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

Return values

FSP_SUCCESS	RMAC_PHY successfully get link partner ability.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY-LSI is not link up.
FSP_ERR_NOT_INITIALIZED	The control block has not been initialized
FSP_ERR_INVALID_MODE	Function is called when not in OPERATION mode.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_INVALID_ARGUMENT	Address is not a valid size

◆ R_RMAC_PHY_ChipSelect()

fsp_err_t R_RMAC_PHY_ChipSelect (ether_phy_ctrl_t *const *p_ctrl*, uint8_t *port*)

Update the target PHY LSI of this driver.

Return values

FSP_SUCCESS	PHY device initialized successfully.
FSP_ERR_ASSERTION	Pointer to RMAC_PHY control block or configuration structure is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_ARGUMENT	Invalid PHY LSI is selected.

◆ R_RMAC_PHY_CallbackSet()

fsp_err_t R_RMAC_PHY_CallbackSet (ether_phy_ctrl_t *const *p_ctrl*, void(*)(ether_phy_callback_args_t *) *p_callback*, void *const *p_context*, ether_phy_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements ether_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.12.6 AWS PKCS11 PAL on LittleFS (rm_aws_pkcs11_pal_littlefs)

Modules » Networking

PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS.

Overview

Note

The PKCS#11 PAL LittleFS Interface does not provide any interfaces to the user. Consult the AWS documentation for more info: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-pkcs.html.

Configuration

There is no user configuration for this module

Usage Notes

The current implementation utilizes LittleFS on Flash (rm_littlefs_flash) for storage.

Limitations

- Credential access is not limited in any way.

5.2.12.7 EtherCAT SSC Port (rm_ethercat_ssc_port)

Modules » Networking

Functions

fsp_err_t	RM_ETHERCAT_SSC_PORT_Open (ethercat_ssc_port_ctrl_t *const p_ctrl, ethercat_ssc_port_cfg_t const *const p_cfg) EtherCAT Slave Controller is initialized with this function. This function includes PHY initialization and ESC EEPROM loading. Afterwards, EtherCAT communication begins. Also EtherCAT interrupts are permitted if the interrupts are used. In order to receive the EtherCAT, EtherCAT Sync0, EtherCAT Sync1 interrupt event, it's necessary to register a callback function. Implements ethercat_ssc_port_api_t::open. More...
fsp_err_t	RM_ETHERCAT_SSC_PORT_Close (ethercat_ssc_port_ctrl_t *const p_ctrl)

Detailed Description

Middleware for EtherCAT Slave Stack Code (SSC) on RX microprocessors. This module implements the EtherCAT SSC Interface.

Overview

This module provides the hardware port layer for EtherCAT Slave Stack Code. The control target is the EtherCAT Slave Controller (ESC). This module uses the EtherCAT slave stack code generated by the SSC tools provided by Beckhoff Automation GmbH. Users must register as a member of the EtherCAT Technology Group (ETG) to obtain SSC
tools.:
https://www.ethercat.org/en/downloads/downloads_01DCC32A10294F2EA866F7E46FB0285F.htm
After initializing this module, refer to the SSC API reference to use the EtherCAT slave stack. The hardware access layer API referenced from the stack are also implemented. Please include renesashw.h in the stack source file which has hardware access.

Features

The EtherCAT SSC port module supports the following features:

- Number of EtherCAT ports: 2
- Interrupts: EtherCAT interrupt, Sync0 interrupt, Sync1 interrupt
- Hardware access layer specified by SSC (including timer)
- Initialize ESC and PHY to enable EtherCAT communication

Configuration

Build Time Configurations for rm_ethercat_ssc_port

The following build time configurations are defined in fsp_cfg/rm_ethercat_ssc_port_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Phy Link Polarity0	• Active High • Active Low	Active Low	Set the signal porality of CAT0_LINKSTA pin.
Phy Link Polarity1	• Active High • Active Low	Active Low	Set the signal porality of CAT1_LINKSTA pin.

Configurations for Networking > EtherCAT SSC Port (rm_ethercat_ssc_port)

This module can be added to the Stacks tab via New Stack > Networking > EtherCAT SSC Port (rm_ethercat_ssc_port).

Configuration	Options	Default	Description
PHY > Reset Signal Hold Time (ms)	Specify a value between 0x1 and 0xFFFFFFFF.	1	Time to keep the PHY reset signal low.
PHY > Reset Wait Time (us)	Specify a value between 0x1 and 0xFFFFFFFF.	15000	Wait time to access the PHY register after the PHY reset signal is released.
PHY > Offset Address	Specify a value between 0x0 and 0x1F.	0	The offset address of PHY in case of using the EtherCAT.
Delay Time of TX Signal > Port 0	• 0 ns • 10 ns • 20 ns • 30 ns	0 ns	Set the amount of delay time for TX signal of the Port 0
Delay Time of TX Signal > Port 1	• 0 ns • 10 ns • 20 ns • 30 ns	0 ns	Set the amount of delay time for TX signal of the Port 1
Interrupts > EtherCAT Interrupt Priority	MCU Specific Options		Select the EtherCAT interrupt priority.
Interrupts > EtherCAT SYNC0 Interrupt Priority	MCU Specific Options		Select the EtherCAT SYNC0 interrupt priority.
Interrupts > EtherCAT SYNC1 Interrupt Priority	MCU Specific Options		Select the EtherCAT SYNC1 interrupt priority.
Interrupts > Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs
Name	Name must be a valid C symbol	g_ethercat_ssc_port0	Module name.
EEPROM Size	• Under 32Kbits • Over 32Kbits	Over 32Kbits	Specify whether to use Normal or Inverted.

Clock Configuration

The ESC uses the ESCCLK which is generated by the clock generation circuit and the ESCCLK divided by 4 as CLK100 and CLK25 of the ESC IP Core, respectively.

Set the registers in the clock generation circuit so that the frequency of the ESCCLK becomes 100 MHz. Also, set the registers that the frequency of the ETHPHYCLK signal which is used as the operating clock of a PHY-LSI becomes 25 MHz.

Pin Configuration

The RM_ETHERCAT_SSC_PORT module is used for configuring pins.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the RM_ETHERCAT_SSC_PORT:

- Port configuration is EtherCAT port only.

ETHERCAT SSC Port Basic Example

This is a basic example of minimal use of the ETHERCAT SSC Port in an application.

```
void ethercat_ssc_port_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module before using the EtherCAT Slave Stack. */
 /* The instance of EtherCAT SSC Port instance can be accessed as
gp_ether_cat_ssc_port */
    err = RM_ETHERCAT_SSC_PORT_Open(gp_ethercat_ssc_port->p_ctrl,
gp_ethercat_ssc_port->p_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Close the module after using the EtherCAT Slave Stack. */
 RM_ETHERCAT_SSC_PORT_Close(gp_ethercat_ssc_port->p_ctrl);
}
/* Example callback called when EtherCAT interrupt. */
void ethercat_ssc_port_callback (ethercat_ssc_port_callback_args_t * p_args)
{
 switch (p_args->event)
    {
```

```
 case ETHERCAT_SSC_PORT_EVENT_ESC_CAT_INTERRUPT:
      {
/* ESC CAT Interrupt application */
 break;
       }
 case ETHERCAT_SSC_PORT_EVENT_ESC_SYNC0_INTERRUPT:
      {
/* ESC SYNC0 Interrupt application */
 break;
       }
 case ETHERCAT_SSC_PORT_EVENT_ESC_SYNC1_INTERRUPT:
      {
/* ESC SYNC1 Interrupt application */
 break;
       }
 default:
      {
 /* Do Nothing! */
       }
    }
}
```

Data Structures

struct	ethercat_ssc_port_instance_ctrl_t
struct	ethercat_ssc_port_extend_cfg_t

Enumerations

enum	ethercat_ssc_port_eeprom_size_t
enum	ethercat_ssc_port_tx_signal_delay_t

Data Structure Documentation

◆ ethercat_ssc_port_instance_ctrl_t

struct ethercat_ssc_port_instance_ctrl_t
EtherCAT SSC port control block. DO NOT INITIALIZE. Initialization occurs when ethercat_ssc_port_api_t::open is called.

Data Fields		
uint32_t	open	Flag to indicate if EtherCAT SSC port instance has been initialized.
uint16_t	sys_time	System time of EtherCAT Slave Controller.
ethercat_ssc_port_cfg_t const *	p_cfg	Pointer to initial configurations.
R_ESC_Type *	p_reg_esc	Pointer to EtherCAT Slave Controller registers.

◆ ethercat_ssc_port_extend_cfg_t

struct ethercat_ssc_port_extend_cfg_t		
Extended configuration		
Data Fields		
ethercat_ssc_port_eeprom_size_t	eeprom_size	EEPROM memory size.
ethercat_ssc_port_tx_signal_delay_t	tx_signal_delay_port0	Port 0 TX signal delay time.
ethercat_ssc_port_tx_signal_delay_t	tx_signal_delay_port1	Port 1 TX signal delay time.
ether_phy_instance_t const *	p_ether_phy_instance[BSP_FEATURE_ESC_NUM_PORTS]	Pointer to ETHER_PHY instance.

Enumeration Type Documentation

◆ ethercat_ssc_port_eeprom_size_t

enum ethercat_ssc_port_eeprom_size_t	
EEPROM memory size of the EtherCAT	
Enumerator	
ETHERCAT_SSC_PORT_EEPROM_SIZE_UNDER_32KBIT	Up to 16KBit.
ETHERCAT_SSC_PORT_EEPROM_SIZE_OVER_32KBIT	32KBit to 4MBit

◆ ethercat_ssc_port_tx_signal_delay_t

enum ethercat_ssc_port_tx_signal_delay_t	
Delay time of TX signal	
Enumerator	
ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_00NS	0 ns
ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_10NS	10 ns
ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_20NS	20 ns
ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_30NS	30 ns

Function Documentation

◆ RM_ETHERCAT_SSC_PORT_Open()

fsp_err_t RM_ETHERCAT_SSC_PORT_Open (ethercat_ssc_port_ctrl_t *const *p_ctrl*, ethercat_ssc_port_cfg_t const *const *p_cfg*)

EtherCAT Slave Controller is initialized with this function. This function includes PHY initialization and ESC EEPROM loading. Afterwards, EtherCAT communication begins. Also EtherCAT interrupts are permitted if the interrupts are used. In order to receive the EtherCAT, EtherCAT Sync0, EtherCAT Sync1 interrupt event, it's necessary to register a callback function. Implements ethercat_ssc_port_api_t::open.

Return values

FSP_SUCCESS	Opened successfully
FSP_ERR_INVALID_HW_CONDITION	ESC EEPROM loading error
FSP_ERR_TIMEOUT	ESC EEPROM loading timeout error

◆ RM_ETHERCAT_SSC_PORT_Close()

fsp_err_t RM_ETHERCAT_SSC_PORT_Close (ethercat_ssc_port_ctrl_t *const *p_ctrl*)
Disables interrupts. Removes power and releases hardware lock. Implements ethercat_ssc_port_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

5.2.12.8 AWS coreHTTP

Modules » Networking

This module provides the AWS coreHTTP library.

Overview

The AWS coreHTTP library can be used to send HTTP and HTTPS requests. The documentation for the library can be found at the following link: coreHTTP.

Features

- Secure and Non-secure HTTP requests
- Mutually authenticated connections

Configuration

Memory Usage

The AWS coreHTTP stack relies on dynamic memory allocation for thread/task creation as well as other uses. It is recommended to tweak the thread stack configuration values based on usage. Notable values are:

FreeRTOS Thread

- General|Minimal Stack Size
- Memory Allocation|Total Heap Size

FreeRTOS Plus TCP

- Stack size in words (not bytes)

Usage Notes

- A transport interface is required to use the CoreHTTP library (https://www.freertos.org/network-interface.html).
- For FSP, a transport interface over MbedTLS is provided. This transport interface can connect to RX ethernet, WiFi, and cellular modules via sockets wrappers for each module.
- TLS_FreeRTOS_Connect should be used before calling any CoreHTTP APIs. See the example for more information.

Limitations

- MbedTLS must be initialized and key provisioning must be done before starting a secure connection.

Examples

HTTPS GET request

```
/* Certificate copied from https://www.amazontrust.com/repository/AmazonRootCA1.pem
*/
static const char g_server_certificate[] = "-----BEGIN CERTIFICATE-----\n" \
 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
 "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
 "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
 "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
 "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
 "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
 "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
 "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
 "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
 "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
 "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
 "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
 "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
 "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
 "rqXRfboQnoZsG4q5WTP468SQvvG5\n"                                    \
 "-----END CERTIFICATE-----";
/* Default settings to use if DHCP fails. */
```

```
const uint8_t g_default_ip_address[4] = {192, 168, 0, 100};
const uint8_t g_default_subnet_mask[4] = {255, 255, 255, 0};
const uint8_t g_default_gateway[4] = {192, 168, 0, 1};
const uint8_t g_default_dns[4] = {8, 8, 8, 8};
#if defined(ipconfigIPv4_BACKWARD_COMPATIBLE) && (ipconfigIPv4_BACKWARD_COMPATIBLE ==
0)
static NetworkInterface_t xInterfaces[1];
static NetworkEndPoint_t xEndPoints[1];
extern NetworkInterface_t * pxFillInterfaceDescriptor(BaseType_t xEMACIndex,
NetworkInterface_t * pxInterface);
#endif
void https_example_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
 /* In order to use the PKCS11 PAL, littlefs must be configured. */
 fsp_err_t fsp_err_status = RM_LITTLEFS_FLASH_Open(g_rm_littlefs0.p_ctrl,
g_rm_littlefs0.p_cfg);
    assert(FSP_SUCCESS == fsp_err_status);
 /* Reformat littlefs to ensure that data flash is in a known state. */
    assert(0 == lfs_format(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /* Mount littlefs. */
    assert(0 == lfs_mount(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /*
  * Write the keys into data flash using the PKCS11 PAL so that they can be used
during TLS setup
  * Note that in an application this will only be done when provisioning a device
with a private key.
  * Once a device has been provisioned, the keys will persist in data flash.
  */
    ProvisioningParams_t params;
    params.pucClientPrivateKey       = (uint8_t *) g_client_private_key;
    params.pucClientCertificate      = (uint8_t *) g_client_certificate;
```

```
    params.ulClientPrivateKeyLength = sizeof(g_client_private_key);
    params.ulClientCertificateLength = sizeof(g_client_certificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    uint32_t err = (uint32_t) vAlternateKeyProvisioning(&params);
    assert(0 == err);
#if defined(ipconfigIPv4_BACKWARD_COMPATIBLE) && (ipconfigIPv4_BACKWARD_COMPATIBLE ==
0)
 /* Initialize the interface descriptor. */
    pxFillInterfaceDescriptor(0, xInterfaces);
    FreeRTOS_FillEndPoint(xInterfaces,
                          xEndPoints,
                          g_default_ip_address,
                          g_default_subnet_mask,
                          g_default_gateway,
                          g_default_dns,
                          g_ether0.p_cfg->p_mac_address);
 /* Initialise the TCP/IP stack. */
    FreeRTOS_IPInit_Multi();
#else
 /* Start up the network stack. */
    FreeRTOS_IPInit(g_default_ip_address,
                    g_default_subnet_mask,
                    g_default_gateway,
                    g_default_dns,
                    g_ether0.p_cfg->p_mac_address);
#endif
#if defined(ipconfigIPv4_BACKWARD_COMPATIBLE) && (ipconfigIPv4_BACKWARD_COMPATIBLE ==
0)
 while (pdFALSE == FreeRTOS_IsEndPointUp(xEndPoints))
#else
 while (pdFALSE == FreeRTOS_IsNetworkUp())
#endif
    {
```

```
      vTaskDelay(10);
   }
   NetworkCredentials_t xSocketsConfig = {0};
   TlsTransportStatus_t xNetworkStatus = TLS_TRANSPORT_SUCCESS;
   TlsTransportParams_t transport_params;
/* Configure credentials for TLS authenticated session. */
   xSocketsConfig.pAlpnProtos = NULL;
   xSocketsConfig.disableSni = false;
   xSocketsConfig.pRootCa    = (const unsigned char *) g_server_certificate;
   xSocketsConfig.rootCaSize = sizeof(g_server_certificate);
   NetworkContext_t xNetworkContext = {0};
/* Initialize network context */
   xNetworkContext.pParams = &transport_params;
/* Attempt to create a authenticated TLS connection. */
   TLS_FreeRTOS_Connect(&xNetworkContext,
"postman-echo.com",
                        HTTPS_EXAMPLE_TLS_PORT,
                        &xSocketsConfig,
                        HTTPS_EXAMPLE_TIMEOUT,
                        HTTPS_EXAMPLE_TIMEOUT);
   assert(TLS_TRANSPORT_SUCCESS == xNetworkStatus);
   TransportInterface_t xTransportInterface;
/* Define the transport interface. */
   xTransportInterface.pNetworkContext = &xNetworkContext;
   xTransportInterface.send            = TLS_FreeRTOS_send;
   xTransportInterface.recv            = TLS_FreeRTOS_recv;
   HTTPRequestInfo_t    xRequestInfo    = {0};
   HTTPRequestHeaders_t xRequestHeaders = {0};
/* Configure a GET request. */
   xRequestInfo.pHost    = "postman-echo.com";
   xRequestInfo.hostLen   = strlen(xRequestInfo.pHost);
   xRequestInfo.pMethod   = HTTP_METHOD_GET;
   xRequestInfo.methodLen = strlen(HTTP_METHOD_GET);
   xRequestInfo.pPath    = "/get?arg1=val1&arg2=val2";
```

```
    xRequestInfo.pathLen   = strlen(xRequestInfo.pPath);
    xRequestInfo.reqFlags = HTTP_REQUEST_KEEP_ALIVE_FLAG;
 /* Set the buffer used for storing request headers. */
 static uint8_t ucUserBuffer[HTTPS_EXAMPLE_USER_BUFFER_SIZE];
    xRequestHeaders.pBuffer   = ucUserBuffer;
    xRequestHeaders.bufferLen = sizeof(ucUserBuffer);
 /* Initialize the request. */
    HTTPStatus_t xHTTPStatus = HTTPClient_InitializeRequestHeaders(&xRequestHeaders,
&xRequestInfo);
    assert(HTTPSuccess == xHTTPStatus);
 /* Reuse the user buffer for storing the response headers. */
    HTTPResponse_t xResponse = {0};
    xResponse.pBuffer   = ucUserBuffer;
    xResponse.bufferLen = sizeof(ucUserBuffer);
 /* Send the request. */
    xHTTPStatus = HTTPClient_Send(&xTransportInterface, &xRequestHeaders, (uint8_t *)
NULL, 0, &xResponse, 0);
    assert(HTTPSuccess == xHTTPStatus);
    TLS_FreeRTOS_Disconnect(&xNetworkContext);
 /* The HTTPS request has completed. The result is stored in xResponse. */
}
```

5.2.12.9 AWS MQTT

Modules » Networking

This module provides the AWS MQTT integration documentation.

Overview

The AWS MQTT library can connect to either AWS or a third party MQTT broker such as Mosquitto. The documentation for the library can be found at the following link: coreMQTT.

Features

- MQTT connections over TLS to an AWS IoT Endpoint or Mosquitto server

Configuration

Memory Usage

The AWS CoreMQTT library relies heavily on dynamic memory allocation for thread/task creation as well as other uses. Depending on the configuration it may be required to provide as much as 110k heap. To decrease this it is recommended to tweak the thread stack configuration values based on usage. Notable values are:

FreeRTOS Thread

- General|Minimal Stack Size

FreeRTOS Plus TCP

- Stack size in words (not bytes)

Usage Notes

- A transport interface is required to use the CoreMQTT library (https://www.freertos.org/network-interface.html).
- For FSP, a transport interface over MbedTLS is provided. This transport interface can connect to RX ethernet, WiFi, and cellular modules via sockets wrappers for each module.
- TLS_FreeRTOS_Connect should be used before calling any CoreMQTT APIs. See the example for more information.
- Not all MQTT servers support maintaining a TLS session after a MQTT disconnect. TLS_FreeRTOS_Disconnect and TLS_FreeRTOS_Connect should be called to establish a new TLS session before attempting to reconnect with CoreMQTT APIs if the server does not support TLS session resumption.

Limitations

- aws_clientcredential.h and aws_clientcredential_keys.h need to be added manually.
- MbedTLS must be initialized and key provisioning must be done before starting a secure connection.

Examples

Connection example using MbedTLS/PKCS11 Transport Interface

```
#define EXAMPLE_PDN_CONTEXT_NUMBER (1)

#define EXAMPLE_TIMEOUT_MS (5000)

#define EXAMPLE_DISABLE_SNI (false)

#define EXAMPLE_MQTT_HOST ("mqtt_host_server")

#define EXAMPLE_MQTT_HOST_PORT (8883)

#define EXAMPLE_MQTT_SEND_TIMEOUT (5000)

#define EXAMPLE_MQTT_RECEIVE_TIMEOUT (5000)

#define EXAMPLE_MQTT_TOPIC ("example_topic")

#define EXAMPLE_MQTT_CLIENT_IDENTIFIER ("client_id")
```

```
#define EXAMPLE_MQTT_KEEP_ALIVE_SECONDS (60)
#define EXAMPLE_MQTT_PAYLOAD ("helloworld")
struct NetworkContext
{
    TlsTransportParams_t * pParams;
};
static const char SERVER_CERTIFICATE_PEM[] =
 "-----BEGIN CERTIFICATE-----\n"
 "MIIDazCCAlOgAwIBAgIURabL79ayIywQv0y8SPnbZ1FYDRIwDQYJKoZIhvcNAQEL\n"
 "BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM\n"
 "GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAeFw0xOTA5MTEyMTIyMjZaFw0yMDA5\n"
 "MTAyMTIyMjZaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw\n"
 "HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwggEiMA0GCSqGSIb3DQEB\n"
 "AQUAA4IBDwAwggEKAoIBAQDSA3h+5sT58FHgnovnQzsVHQ0H/3TsnEKwVzyBwTQl\n"
 "s4PbG6VXCWyyJWjdJ4XMH1oU8gAlxauFbwOO98Aquei4K3Pi/ynKNBeX4VJcLyE5\n"
 "Azq7nRIIwt4+OoZ5kV7v8JIoLY5i+Ktn3zq1t0y1ZmK6Uk/rRPonb+Kx7wQPx7jq\n"
 "ZIZGda+CgF6ZedidPcABuggqD1y3U2gLiRPoBhe9nN2hG60rRp7vhbWMF0pzTDXu\n"
 "BKF7XSTbhYz3pl6NeOCLh5E3t8x908Ui5W1zDN3iOysrcwQFtCiGTvzNtxSfli1+\n"
 "PugIt9Q2vlYmuz5qI+juxHftJSXO86M5SV7exqUOXP9RAgMBAAGjUzBRMB0GA1Ud\n"
 "DgQWBBQG8VNJEJUjpTKMjmrOY3XApNp5lDAfBgNVHSMEGDAWgBQG8VNJEJUjpTKM\n"
 "jmrOY3XApNp5lDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQAt\n"
 "CabfjsYUnG8tt3/GDdhjsuG+SfeQe11S73pZi3+L616bPH5MNUv+LkgR/1AFEqt5\n"
 "WadKVTgzW5Ork1t7CfkYwrOHbyhyaaDPzERjMCfCcl8lQluBy6vE/lEb0hWq6XlO\n"
 "f6+8i+VKxWkSIXs2ZQqqYSOTTzAjHSsiiuE5WsC00ErvCvnC7uD6+3Y7W1uQRkFZ\n"
 "uSd9AN1ixPvAFi69FF/ymlJv6vII5GXOVDrIwdr50bMNuezMEx6qMNDADRH8iEaL\n"
 "JaSgfklczGiI1i7MPD4JTtsXOgKwxcBDAa0zQDVA5uBGEIOhva3m5X70N4iO7W0V\n"
 "eEhZekKeg3Fl3t/CXi8l\n"
 "-----END CERTIFICATE-----";
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
  "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
  "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
  "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
```

```
  "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
  "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
  "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
  "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
```

```
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
/* Callback to handle MQTT events, user should add their own processing */
static void prvEventCallback(MQTTContext_t * pxMQTTContext,
                             MQTTPacketInfo_t       * pxPacketInfo,
                             MQTTDeserializedInfo_t * pxDeserializedInfo);
static void prvEventCallback (MQTTContext_t * pxMQTTContext,
                              MQTTPacketInfo_t       * pxPacketInfo,
                               MQTTDeserializedInfo_t * pxDeserializedInfo)
{
 FSP_PARAMETER_NOT_USED(pxMQTTContext);
 FSP_PARAMETER_NOT_USED(pxPacketInfo);
 FSP_PARAMETER_NOT_USED(pxDeserializedInfo);
}
/* Callback to get current time */
static uint32_t prvGetTimeMs(void);
static uint32_t prvGetTimeMs (void)
{
    TickType_t xTickCount = 0;
    uint32_t   ulTimeMs   = 0UL;
 /* Get the current tick count. */
    xTickCount = xTaskGetTickCount();
 /* Convert the ticks to milliseconds. */
    ulTimeMs = (uint32_t) xTickCount * (1000U / configTICK_RATE_HZ); //
NOLINT(readability-magic-numbers)
 return ulTimeMs;
}
NetworkContext_t networkContext;
```

```
void rm_aws_transport_interface_port_basic_example (void)
{
    ProvisioningParams_t params;
    NetworkCredentials_t credentials;
    TransportInterface_t transport;
    MQTTContext_t       mqtt_context;
    TlsTransportParams_t transport_params;
 static uint8_t       buffer[1024]; // NOLINT(readability-magic-numbers)
    MQTTFixedBuffer_t    networkBuffer;
    MQTTConnectInfo_t    connectInfo;
    MQTTPublishInfo_t    lwtInfo;
 bool                session_present;
 /* Open little FS flash and format in order to store keys */
    assert(FSP_SUCCESS == RM_LITTLEFS_FLASH_Open(g_rm_littlefs0.p_ctrl,
g_rm_littlefs0.p_cfg));
    assert(0 == lfs_format(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
    assert(0 == lfs_mount(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
 /* Write the keys into a secure region in data flash. */
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientCertificateLength = sizeof(keyCLIENT_CERTIFICATE_PEM);
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.ulClientPrivateKeyLength = sizeof(keyCLIENT_PRIVATE_KEY_PEM);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
 /* Initialize network context */
    networkContext.pParams = &transport_params;
 /* Setup network credentials */
    credentials.pAlpnProtos      = NULL;
    credentials.disableSni       = EXAMPLE_DISABLE_SNI;
    credentials.pRootCa          = (const unsigned char *) SERVER_CERTIFICATE_PEM;
    credentials.rootCaSize       = sizeof(SERVER_CERTIFICATE_PEM);
```

```
    credentials.pUserName       = NULL;
    credentials.userNameSize    = 0;
    credentials.pPassword       = NULL;
    credentials.passwordSize    = 0;
    credentials.pClientCertLabel = pkcs11configLABEL_DEVICE_CERTIFICATE_FOR_TLS;
    credentials.pPrivateKeyLabel = pkcs11configLABEL_DEVICE_PRIVATE_KEY_FOR_TLS;
 /* Set transport interface */
    transport.pNetworkContext = &networkContext;
    transport.recv            = TLS_FreeRTOS_recv;
    transport.send            = TLS_FreeRTOS_send;
 /* Connect to a MQTT host */
    TLS_FreeRTOS_Connect(&networkContext,
                         EXAMPLE_MQTT_HOST,
                         EXAMPLE_MQTT_HOST_PORT,
                         &credentials,
                         EXAMPLE_MQTT_RECEIVE_TIMEOUT,
                         EXAMPLE_MQTT_SEND_TIMEOUT);
 /* Fill the values for network buffer. */
    networkBuffer.pBuffer = buffer;
    networkBuffer.size    = 1024;      // NOLINT
 /* Initialize MQTT */
    assert(MQTTSuccess == MQTT_Init(&mqtt_context, &transport, prvGetTimeMs,
prvEventCallback, &networkBuffer));
 /* Set connection info for MQTT session */
    connectInfo.cleanSession          = true;
    connectInfo.clientIdentifierLength = sizeof(EXAMPLE_MQTT_CLIENT_IDENTIFIER) - 1;
    connectInfo.pClientIdentifier      = EXAMPLE_MQTT_CLIENT_IDENTIFIER;
    connectInfo.keepAliveSeconds       = EXAMPLE_MQTT_KEEP_ALIVE_SECONDS;
    connectInfo.pUserName             = NULL;
    connectInfo.userNameLength         = 0U;
    connectInfo.pPassword             = NULL;
    connectInfo.passwordLength         = 0U;
 /* LWT Info. */
    lwtInfo.pTopicName      = EXAMPLE_MQTT_TOPIC;
```

```
    lwtInfo.topicNameLength = sizeof(EXAMPLE_MQTT_TOPIC) - 1;
    lwtInfo.pPayload       = EXAMPLE_MQTT_PAYLOAD;
    lwtInfo.payloadLength   = strlen(EXAMPLE_MQTT_PAYLOAD);
    lwtInfo.qos             = MQTTQoS0;
    lwtInfo.dup             = false;
    lwtInfo.retain          = false;
 /* Send MQTT CONNECT packet to broker. */
    assert(MQTTSuccess == MQTT_Connect(&mqtt_context, &connectInfo, &lwtInfo, 20000,
&session_present));
}
```

5.2.12.10 FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp)

Modules » Networking

Middleware for using TCP on RX MCUs.

Overview

FreeRTOS Plus TCP is a TCP stack created for use with FreeRTOS.

This module provides the NetworkInterface required to use FreeRTOS Plus TCP with the ETHER driver or the Ethernet (r_rmac) driver.

Please refer to the FreeRTOS Plus TCP documentation for further details.

Configuration

Build Time Configurations for FreeRTOS_Plus_TCP

The following build time configurations are defined in aws/FreeRTOSIPConfig.h:

Configuration	Options	Default	Description
Print debug messages	• Disable • Enable	Disable	If ipconfigHAS_DEBUG_PRINTF is set to 1 then FreeRTOS_debug_printf should be defined to the function used to print out the debugging messages.
Backward Compatible Mode	• No • Yes	Yes	Run the code in backward compatible

			mode
Enable IPV6	• Disable • Enable	Disable	Stack supports handling IPv6 packets (including handling IPv6 header, ND, RA, and so on) when enabled
Print info messages	• Disable • Enable	Disable	Set to 1 to print out non debugging messages, for example the output of the FreeRTOS_netstat() command, and ping replies. If ipconfigHAS_PRINTF is set to 1 then FreeRTOS_printf should be set to the function used to print out the messages.
Byte order of the target MCU	• pdFREERTOS_LITTLE_ENDIAN • pdFREERTOS_BIG_ENDIAN	pdFREERTOS_LITTLE_ENDIAN	Define the byte order of the target MCU
IP/TCP/UDP checksums	• Disable • Enable	Enable	If the network card/driver includes checksum offloading (IP/TCP/UDP checksums) then set ipconfigDRIVER_INCLUDED_RX_IP_CHECKSUM to 1 to prevent the software stack repeating the checksum calculations.
Receive Block Time	Value must be a non-negative integer	10000	Amount of time FreeRTOS_recv() will block for. The timeouts can be set per socket, using setsockopt().
Send Block Time	Value must be a non-negative integer	10000	Amount of time FreeRTOS_send() will block for. The timeouts can be set per socket, using setsockopt().
DNS caching	• Disable • Enable	Enable	DNS caching
DNS Request Attempts	Value must be an integer	2	When a cache is present, ipconfigDNS_REQUEST_ATTEMPTS can

			be kept low and also DNS may use small timeouts.
IP stack task priority	Only symbols, numbers and arithmetic operators are valid.	configMAX_PRIORITIES - 2	Set the priority of the task that executes the IP stack.
Stack size in words (not bytes)	Only symbols, numbers and arithmetic operators are valid.	configMINIMAL_STACK_SIZE * 5	The size, in words (not bytes), of the stack allocated to the FreeRTOS+TCP stack.
Network Events call vApplicationIPNetworkEventHook	• Disable • Enable	Enable	vApplicationIPNetworkEventHook is called when the network connects or disconnects.
Max UDP send block time	Only symbols, numbers and arithmetic operators are valid.	15000 / portTICK_PERIOD_MS	Max UDP send block time
Use DHCP	• Disable • Enable	Enable	If ipconfigUSE_DHCP is 1 then FreeRTOS+TCP will attempt to retrieve an IP address, netmask, DNS server address and gateway address from a DHCP server.
DHCP Register Hostname	• Disable • Enable	Enable	Register hostname when using DHCP
DHCP Uses Unicast	• Disable • Enable	Enable	DHCP uses unicast.
DHCP Send Discover After Auto IP	• Disable • Enable	Disable	DHCP Send Discover After Auto IP
DHCP callback function	• Disable • Enable	Disable	Provide an implementation of the DHCP callback function (xApplicationDHCPHook)
Interval between transmissions	Only symbols, numbers and arithmetic operators are valid.	120000 / portTICK_PERIOD_MS	When ipconfigUSE_DHCP is set to 1, DHCP requests will be sent out at increasing time intervals until either a reply is received from a DHCP server and accepted, or the interval between transmissions reaches ipconfigMAXIMUM_DISC

			OVER_TX_PERIOD.
ARP Cache Entries	Value must be an integer	6	The maximum number of entries that can exist in the ARP table at any one time
ARP Request Retransmissions	Value must be an integer	5	ARP requests that do not result in an ARP response will be re-transmitted a maximum of ipconfigMAX_ARP_RETRANSMISSIONS times before the ARP request is aborted.
Maximum time before ARP table entry becomes stale	Value must be an integer	150	The maximum time between an entry in the ARP table being created or refreshed and the entry being removed because it is stale
Use string for IP Address	• Disable • Enable	Enable	Take an IP in decimal dot format (for example, "192.168.0.1") as its parameter FreeRTOS_inet_addr_quick() takes an IP address as four separate numerical octets (for example, 192, 168, 0, 1) as its parameters
Total number of available network buffers	Value must be an integer	10	Define the total number of network buffer that are available to the IP stack
Set the maximum number of events	Only symbols, numbers and arithmetic operators are valid.	ipconfigNUM_NETWORK_BUFFER_DESCRIPTORS + 5	Set the maximum number of events that can be queued for processing at any one time. The event queue must be a minimum of 5 greater than the total number of network buffers
Enable FreeRTOS_sendto() without calling Bind	• Enable • Disable	Disable	Set to 1 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the IP stack automatically binding

			the socket to a port number from the range socketAUTO_PORT_ALLOCATION_START_NUMBER to 0xffff. If ipconfigALLOW_SOCKET_SEND_WITHOUT_BIND is set to 0 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the send operation being aborted.
TTL values for UDP packets	Value must be an integer	128	Define the Time To Live (TTL) values used in outgoing UDP packets
TTL values for TCP packets	Value must be an integer	128	Defines the Time To Live (TTL) values used in outgoing TCP packets
Use TCP and all its features	• Disable • Enable	Enable	Use TCP and all its features
Let TCP use windowing mechanism	• Disable • Enable	Disable	Let TCP use windowing mechanism
Maximum number of bytes the payload of a network frame can contain	Value must be an integer	1500	Maximum number of bytes the payload of a network frame can contain
Basic DNS client or resolver	• Disable • Enable	Enable	Set ipconfigUSE_DNS to 1 to include a basic DNS client/resolver. DNS is used through the FreeRTOS_gethostbyname() API function.
Reply to incoming ICMP echo (ping) requests	• Disable • Enable	Enable	If ipconfigREPLY_TO_INCOMING_PINGS is set to 1 then the IP stack will generate replies to incoming ICMP echo (ping) requests.
FreeRTOS_SendPingRequest() is available	• Disable • Enable	Disable	If ipconfigSUPPORT_OUTGOING_PINGS is set to 1 then the FreeRTOS_SendPingRequest() API function is available.
FreeRTOS_select() (and associated) API function is available	• Disable • Enable	Disable	If ipconfigSUPPORT_SELECT_FUNCTION is set to 1 then the

			FreeRTOS_select() (and associated) API function is available
Filter out non Ethernet II frames.	• Disable • Enable	Enable	If ipconfigFILTER_OUT_NON_ETHERNET_II_FRAMES is set to 1 then Ethernet frames that are not in Ethernet II format will be dropped. This option is included for potential future IP stack developments
Responsibility of the Ethernet interface to filter out packets	• Disable • Enable	Disable	If ipconfigETHERNET_DRIVER_FILTERS_FRAME_TYPES is set to 1 then it is the responsibility of the Ethernet interface to filter out packets that are of no interest.
Send RST packets, when receive unknown packets.	• Disable • Enable	Enable	TCP will not send RST packets in reply to TCP unknown or out-of-order packets
Block time to simulate MAC interrupts	Please enter a valid function name without spaces or funny characters	20 / portTICK_PERIOD_MS	The windows simulator cannot really simulate MAC interrupts, and needs to block occasionally to allow other tasks to run
Access 32-bit fields in the IP packets	Value must be an integer	2	To access 32-bit fields in the IP packets with 32-bit memory instructions, all packets will be stored 32-bit-aligned, plus 16-bits. This has to do with the contents of the IP-packets: all 32-bit fields are 32-bit-aligned, plus 16-bit
Size of the pool of TCP window descriptors	Value must be an integer	240	Define the size of the pool of TCP window descriptors
Size of Rx buffer for TCP sockets	Value must be an integer	3000	Define the size of Rx buffer for TCP sockets
Size of Tx buffer for TCP sockets	Value must be an integer	3000	Define the size of Tx buffer for TCP sockets
TCP keep-alive	• Disable • Enable	Enable	TCP keep-alive is available or not

TCP keep-alive interval	Value must be an integer	120	TCP keep-alive interval in second
The socket semaphore to unblock the MQTT task (USER_SEMAPHORE)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (WAKE_CALLBACK)	• Disable • Enable	Enable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (USE_CALLBACKS)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (TX_DRIVER)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (RX_DRIVER)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
Possible optimisation for expert users	• Disable • Enable	Disable	Possible optimisation for expert users - requires network driver support. It is is useful when there is high network traffic. If non-zero value then instead of passing received packets into the IP task one at a time the network interface can chain received packets together and pass them into the IP task in one go. If set to 0 then only one buffer will be sent at a time.

Configurations for Networking > FreeRTOS+TCP Wrapper to ethernet driver (rm_freertos_plus_tcp)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_freertos_plus_tcp0	Module name.
RX Handler Task Stack Size	Only symbols, numbers and arithmetic operators are valid.	configMINIMAL_STACK_SIZE	Stack size of RX handler task.
RX Handler Task Priority	Only symbols, numbers and arithmetic operators are valid.	configMAX_PRIORITIES - 1	Priority of RX handler task. A higher priority ensures timely processing of network

			traffic or callbacks.
Check Link Status Task Stack Size	Only symbols, numbers and arithmetic operators are valid.	configMINIMAL_STACK_SIZE	Stack size of Check link status task.
Check Link Status Task Priority	Only symbols, numbers and arithmetic operators are valid.	configMAX_PRIORITIES - 1	Priority of Check link status task. A low priority is recommended as this is an periodic task.
Link check interval	Only symbols, numbers and arithmetic operators are valid.	1000	Interval for checking link status of the ethernet instance of the network interface. Low period will consume more processing time

Usage Notes

Backward Compatible Mode

In order to use the NetworkInterface implementation provided by Renesas for RX devices:

1/ **In case of Common|Backward Compatible Mode =="NO" (FreeRTOS+TCP 4.0.0 onward)**

Note

This mode use IP_Init_Multi to initialize the TCP/IP stack.
Before calling this function, at least 1 interface and 1 end-point must have been added to the TCP/IP stack.

2/ **In case of Common|Backward Compatible Mode =="YES" (FreeRTOS+TCP 4.0.0 backward)**

Note

This mode use IP_Init to initialize stack.
Only support 1 interface.
Only one instance of the middle port layer should be generated.

- Some of the stack initialisation APIs have been deprecated as of FreeRTOS V4.0.0 onwards. Please refer to FreeRTOS+TCP Networking Tutorial: Initializing the TCP/IP Stack for the new APIs supporting IPv6, Multiple Endpoints and Multiple interfaces. To use the deprecated APIs please set it to "Yes" for the stack property "Common|Backward Compatible Mode".

Note

The MAC address passed to FreeRTOS_FillEndPoint must match the MAC address configured in the r_ether instance.
g_ether_instance must have vEtherISRCallback configured as the callback.
The xApplicationGetRandomNumber and ulApplicationGetNextSequenceNumber functions should be implemented in systems using FreeRTOS Plus TCP without Secure Sockets.
To connect to a server using an IP address the macro ipconfigINCLUDE_FULL_INET_ADDR must be set to 1.

Big endian

When big endian mode is selected, the **Common|Byte order of the target MCU** setting of FreeRTOS+TCP must be changed to pdFREERTOS_BIG_ENDIAN to ensure correct operation.

Limitations

- Zero-copy is not currently supported by the NetworkInterface.

5.2.12.11 gPTP Ethernet Switch Module(rm_gptp_eswm)

Modules » Networking

Overview

This module provides time synchronization based on the standard IEEE 802.1AS-2020.

Features

- Clock Source selection according to Best Master Clock Algorithm (BMCA)
- Time Aware Endstation
  - Clock Source (Synchronization Master)
  - Clock Sink (Synchronization Slave)

Usage Notes

Important:

When running this module with FreeRTOS

- Configure the Timer Queue Length to **at least 7** to ensure stable system operation. FreeRTOS timer operations are queued and processed asynchronously by the Timer Service task. When multiple timer commands are generated within a short period, the timer queue may become full if its length is insufficient.
- Update the Timer Queue Length setting in: Stacks -> New Thread -> Properties -> Common -> Timers -> Timer Queue Length

Limitations

The supported clock are 200/250 MHz for ESW clock and 25MHz for ETHPHY clock.

Examples

Baremetal Example

This is a example of minimal use of the gPTP Ethernet switch module in the baremetal application.

```
void rm_gptp_eswm_baremetal_example (void)
{
```

```
 fsp_err_t err;
 /* Initialize gPTP ESWM module. */
    err = rm_gptp_eswm_init(&g_gptp_eswm0_ctrl, &g_gptp_eswm0_cfg);
    assert(FSP_SUCCESS == err);
 while (1)
    {
 /* Read a gPTP packet. */
       gptp_read();
 /* Check timer expired. */
       gptp_timer_check();
 /* Send a gPTP packet. */
       gptp_send();
 /* Detect timer counts. */
 while (!(gptp_detect_count()))
      {
            ;
       }
    }
}
```

FreeRTOS Example

This is a example of minimal use of the gPTP Ethernet switch module in the freertos application.

```
void rm_gptp_eswm_freertos_example (void)
{
 fsp_err_t err;
 /* Initialize gPTP ESWM module. */
    err = rm_gptp_eswm_init(&g_gptp_eswm0_ctrl, &g_gptp_eswm0_cfg);
    assert(FSP_SUCCESS == err);
 /* Create ptp task. */
    rm_gptp_sys_freertos_create_ptp_task(g_gptp_eswm0_ctrl.p_cfg->p_gptp_sys_port_ins
tance->p_ctrl);
 /* Create ptp message generation task. */
    rm_gptp_sys_freertos_create_ptp_message_gen_task(g_gptp_eswm0_ctrl.p_cfg->p_gptp_
```

```
sys_port_instance->p_ctrl);
 /* Create ptp read task. */
    rm_gptp_sys_freertos_create_ptp_read_task(g_gptp_eswm0_ctrl.p_cfg->p_gptp_sys_por
t_instance->p_ctrl);
 while (1)
    {
       vTaskDelay(1);
    }
}
```

Configuration API

The configuration application programming interface (configuration-api) can be used to change dedicated functions of the synchronization software stack during runtime. The interface has the following functions and is located in module r_ptp_configuration_api:

Function name	Brief description
set_priority1	Set the priority 1 value for the PTP instance
set_priority2	Set the priority 2 value for the PTP instance
set_domain_number	Set the domain number for the PTP instance
set_instance_enable	Enable or disable a PTP instance
set_enable_ptp_port	Enable a port for PTP
set_delay_mechanism	Set a delay mechanism for a PTP port
set_mean_link_delay_threshold	Set the mean link delay threshold for a PTP port
set_mgt_settable_log_announce_interval	Set the log announce message interval for a PTP port
set_use_mgt_settable_log_announce_interval	Use the settable log announce message interval for a PTP port
set_announce_receipt _timeout	Set the announce receipt timeout value for a PTP port
set_initial_log_sync_interval	Set the initial log sync interval
set_mgt_settable_log_sync_interval	Set the log sync message interval for a PTP port
set_use_mgt_settale_log_sync_interval	Use the settable log sync message interval for a PTP port
set_sync_receipt _timeout	Set the sync receipt timeout value for a PTP port
set_initial_log_pdelay_req_interval	Set the initial log Pdelay request interval
set_use_mgt_settable_log_pdelay_req_interval	Use the managementt settable log pdelay

	request intervalfor a PTP port
set_mgt_settable_log_pdelay_req_interval	Set the log Pdelay request interval for a PTP port
set_allowed_lost_responses	Set the number of allowed lost Pdelay responses for a PTP port
set_initial_log_gptp_capable_message_ineterval	Set the initial log gptp capable message interval
set_use_mgt_settable_log_gptp_capable_message_interval	Use the management settable log gptp capable message interval for a PTP port
set_mgt_settable_log_gptp_capable_message_interval	Set the management settable log gptp capable message interval
set_time_value	Set the management settable log gptp capable message interval

Diagnosis API

The diagnosis application programming interface (diagnosis-api) can be used to monitor dedicated functions of the synchronization software stack. The interface has the following functions and is located at r_ptp_diagnosis_api:

Function name	Brief description
get_clock_class	Get the clock class value for a PTP instance
get_clock_accuracy	Get the clock accuracy value for a PTP instance
get_offset_scaled_log_variance	Get the offset scaled log variance value for a PTP instance
get_priority1	Get the priority 1 value for a PTP instance
get_priority2	Get the priority 2 value for a PTP instance
get_domain_number	Get the domain number value for a PTP instance
get_offset_from_master	Get the current offset from master for a PTP instance
get_cumulative_rate_ratio	Get the cumulative rate ratio value for a PTP instance
get_grandmaster_clock_class	Get the grandmaster clock class value for a PTP instance
get_grandmaster_clock_accuracy	Get the grandmaster clock accuracy value for a PTP instance
get_grandmaster_offset_scaled_log_variance	Get_the grandmaster offset scaled log variance value for a PTP instance
get_grandmaster_priority1	Get the grandmaster priority1 value for a PTP instance
get_grandmaster_priority2	Get the grandmaster priority2 value for a PTP instance
get_ptp_port_enabled	Get the PTP port enabled status for a PTP port

get_delay_measurement	Get the delay measurement method for a PTP port
get_mean_link_delay_threshold	Get the mean link delay threshold value for a PTP port
get_mean_link_delay_threshold	Get the initial log announce interval value for a PTP port
get_current_log_announce_interval	Get the current log announce interval value for a PTP port
get_use_mgt_settable_log_announce_interval	Get the management settable log announce interval use status for a PTP port
get_announce_receipt_timeout	Get the announce receipt timeout value for a PTP port
get_initial_log_sync_interval	Get the initial log sync interval value for a PTP port
get_initial_log_sync_interval	Get the current log sync interval value for a PTP port
get_use_mgt_settable_log_sync_interval	Get the management settable log sync interval use state for a PTP port
get_sync_receipt_timeout	Get the sync receipt timeout value for a PTP port
get_initial_log_pdelay_req_interval	Get the initial log Pdelay request interval value
get_current_log_pdelay_req_interval	Get the current log Pdelay request interval value
get_use_mgt_settable_log_pdelay_req_interval	Get the management settable log Pdelay request interval use status value for a PTP port
get_allowed_lost_responses	Get the value of allowed lost Pdelay responses for a PTP port
get_initial_log_gptp_capable_message_interval	Get the initial log gptp capable signaling message interval
get_current_log_gptp_capable_message_interval	Get the current gptp capable signaling message interval
get_use_mgt_settable_log_gptp_capable_messag_interval	Get the management settable log gptp capable message interval use state
get_mgt_settable_log_gptp_capble_message_interval	Get the management settable log gptp capable message interval
get_cmlds_link_port_enable	Get the cmlds link port enable status for a PTP port
get_rx_sync_count	Get the counter value for received Sync messages for a PTP port
get_rx_one_step_sync_count	Get the counter value for received one step Sync messages for a PTP port
get_rx_follow_up_count	Get the counter value for received Follow-Up messages for a PTP port

get_rx_pdelay_request_count	Get the counter value for received Pdelay request messages for a PTP port
get_rx_pdelay_resp_count	Get the counter value for received Pdelay response messages for a PTP port
get_rx_pdelay_resp_fu_count	Get the counter value for received Pdelay response follow up messages for a PTP port
get_rx_announce_count	Get the counter value for received Announce messages for a PTP port
get_rx_packet_discard_count	Get the counter value for discarded PTP packets for a PTP port
get_sync_receipt_timeout_count	Get the counter value for Sync receipt timeouts for a PTP port
get_announce_receipt_timeout_count	Get the counter value for Announce receipt timeouts for a PTP port
get_pdelay_allowed_lost_responses_exceeded_count	Get the counter value for lost responses exceeded for a PTP port
get_sync_tx_count	Get the counter value for transmitted Sync messages for a PTP port
get_tx_one_step_sync_count	Get the counter value for transmitted one step Sync messages for a PTP port
get_tx_follow_up_count	Get the counter value for transmitted Follow-Up messages for a PTP port
get_tx_pdelay_request_count	Get the counter value for transmitted Pdelay request messages for a PTP port
get_tx_Pdelay_response_count	Get the counter value for transmitted Pdelay response messages for a PTP port
get_tx_Pdelay_response_follow_up_count	Get the counter value for transmitted Pdelay response follow up messages for a PTP port
get_tx_announce_count	Get the counter value for transmitted Announce messages for a PTP port
get_cmlds_rx_pdelay_request_count	Get the counter value for received Pdelay request messages with CMLDS tag for a PTP port
get_cmlds_rx_pdelay_response_count	Get the counter value for received Pdelay response messages with CMLDS tag for a PTP port
get_cmlds_rx_pdelay_response_fu_count	Get the counter value for received Pdelay response follow up messages with CMLDS tag for a PTP port
get_cmlds_rx_ptp_packet_discard_count	Get the counter value for discarded PTP packetes with CMLDS tag for a PTP port
get_cmlds_pdelay_allowed_lost_responses_exceeded_count	Get the counter value for lost responses exceeded with CMLDS tag for a PTP port

get_cmlds_tx_pdelay_request_count	Get the counter value for transmitted Pdelay request messages with CMLDS tag for a PTP port
get_cmlds_tx_pdelay_response_count	Get the counter value for transmitted Pdelay response messages with CMLDS tag for a PTP port
get_cmlds_tx_pdelay_response_fu_count	Get the counter value for transmitted Pdelay response follow up messages with CMLDS tag for a PTP port
get_time_value	Get the current time value of the timer (PTP clock)

5.2.12.12 gPTP Baremetal Porting Layer (rm_gptp_sys_baremetal)

Modules » Networking

Overview

This module provides system timer feature for baremetal.

Features

This porting layer includes the following features.

- Timer

5.2.12.13 gPTP FreeRTOS Porting Layer (rm_gptp_sys_freertos)

Modules » Networking

Overview

This module provides a gPTP porting layer for FreeRTOS.

Features

This porting layer includes the following features.

- Support PTP task
- Support message generation task
- Support read task

5.2.12.14 lwIP HTTP Server

Modules » Networking

This module provides the lwIP HTTP Server integration documentation.

Overview

The LwIP HTTP Server library can be used to respond HTTP request. The documentation for the library can be found at the following link: lwIPServer.

Features

The LwIP HTTP Server:

- Can respond from HTTP GET and POST requests.
- Supports Server-Side Includes (SSI) and Common Gateway Interface (CGI).
- Can set up TLS connection handling (HTTPS).

Configuration

Build Time Configurations for Source

The following build time configurations are defined in lwip/lwip/apps/httpd_opts.h:

Configuration	Options	Default	Description
LwIP HTTPD CGI (old style)	• Enabled • Disabled	Disabled	Enable CGI support (old style). Use this CGI type e.g. to execute specific actions and return a page that does not depend on the CGI parameters.
LwIP HTTPD CGI SSI (new style)	• Enabled • Disabled	Disabled	Enable CGI support with SSI feature (new style). Use this CGI handler if you want CGI information passed on to SSI.
LwIP HTTPD SSI	• Enabled • Disabled	Disabled	Enable SSI support (Server-Side-Includes).
LwIP HTTPD SSI Raw	• Enabled • Disabled	Disabled	Enable to implement an SSI tag handler callback that gets a const char* to the tag (instead of an index into a pre-registered array of known tags).

LwIP HTTPD SSI By File Extension	• Overridden • Enabled • Disabled	Enabled	enable/disable or override the file SSI tag scan function. You need to provide an external function to perform the check: u8_t http_uri_is_ssi(struct fs_file *file, const char *uri)
LwIP HTTPD SSI Extension	Manual Entry	".shtml", ".shtm", ".ssi", ".xml", ".json"	List of file extensions handled as SSI files. Only used if LWIP_HTTPD_SSI_BY_FILE_EXTENSION != 0.
LwIP HTTPD Support POST	• Enabled • Disabled	Disabled	Set this to 1 to support HTTP POST.
LwIP HTTPD Maximum CGI Parameters	Must be a non zero positive interger.	16	The maximum number of parameters that the CGI handler can be sent.
LwIP HTTPD SSI Multipart	• Enabled • Disabled	Disabled	SSI handler function is called with 2 more arguments to handle string too long to be inserted at once
LwIP HTTPD Maximum Tag Name Length	Must be a non zero positive interger.	8	The maximum length of the string comprising the SSI tag name. The longer tags are ignored.
LwIP HTTPD Maximum Tag Insert Length	Must be a non zero positive interger.	192	The maximum length of string that can be returned to replace any given tag.
LwIP HTTPD POST Manual WND	• Enabled • Disabled	Disabled	Enable configurating POST window response for HTTP server
HTTPD Server Agent	Manual Entry	lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/projects/lwip)	This string is passed in the HTTP header as 'Server: HTTPD_SERVER_AGENT'.
LwIP HTTPD Dynamic Headers	• Enabled • Disabled	Disabled	Set this to 1 if you want to include code that creates HTTP headers at runtime. Else use static header created

			by the makefsdata tool.
HTTPD Debug	• Disabled • Enabled	Disabled	Configure debug for HTTPD file send timing
HTTPD Use Mem Pool	• Enabled • Disabled	Disabled	Set this to 1 to allocate http_state instead of using heap.If enabled, define MEMP_NUM_PARALLEL_HTTPD_CONNS (and MEMP_NUM_PARALLEL_HTTPD_SSI_CONNS for SSI) to set the size of the pool(s).
HTTPD Server Port	Must be an interger in range 0-65535.	80	The http server port for HTTPD to use
HTTPD Server Port HTTPS	Must be an interger in range 0-65535.	443	The https server port for HTTPD to use
HTTPD Enable HTTPS	• Enabled • Disabled	Disabled	Enable https support.
HTTPD Maximum Retries	Must be a positive integer.	4	Maximum retries before the connection is aborted/closed.
HTTPD Poll Interval	Must be a non zero positive interger.	4	How long the Server will keep the connection. The poll delay is X*500ms.
HTTPD TCP Priority	Must be an interger in range 1-127.	1	Priority for tcp pcbs created by HTTPD 1 by default range from 1-127.
LwIP HTTPD Timing	• Enabled • Disabled	Disabled	Set this to 1 to enable timing each file sent.
HTTPD Debug Timing	• Disabled • Enabled	Disabled	Set this to 1 to enable timing each file sent.
LwIP HTTPD Support Extstatus	• Enabled • Disabled	Disabled	Set this to one to show error pages when parsing request fail. Else simply close the connection
LwIP HTTPD Support V09	• Enabled • Disabled	Enabled	Set this to 0 to drop support for HTTP/0.9 clients (to save some bytes).
LwIP HTTPD Support 11 Keepalive	• Enabled • Disabled	Disabled	Set this to 1 to enable HTTP/1.1 persistent connections.
LwIP HTTPD Support	• Enabled	Enabled	Set this to 1 to support

Requestlist	• Disabled		HTTP request coming in in multiple packets/pbufs.
LwIP HTTPD Request Queuelen	Must be a non zero positive interger.	5	Number of rx pbufs to enqueue to parse an incoming request.
LwIP HTTPD Request Bufsize	Must be a non zero positive interger.	LWIP_HTTPD_MAX_REQ_LENGTH	Number of (TCP payload-) bytes (in pbufs) to enqueue to parse and incoming request.
LwIP HTTPD Maximum Request Length	Must be a non zero positive interger.	LWIP_MIN(1023, (LWIP_HTTPD_REQ_QUEUELEN * PBUF_POOL_BUFSIZE))	Defines the maximum length of a HTTP request line.
LwIP HTTPD Maximum Request URI Length	Must be a positive interger.	63	This is the size of a static buffer used when URIs end with '/'.
LwIP HTTPD POST Maximum Response URI Length	Must be a non zero positive interger.	63	Maximum length of the filename to send as response to a POST request.
LwIP HTTPD SSI Include Tag	• Enabled • Disabled	Enabled	Set this to 0 to not send the SSI tag.
LwIP HTTPD Abort On Close Memory Error	• Enabled • Disabled	Disabled	Set this to 1 to call tcp_abort when tcp_close fails with memory error.
LwIP HTTPD Kill Old On Connections Exceeded	• Enabled • Disabled	Disabled	Set this to 1 to kill the oldest connection.
LwIP HTTPD Omit Header For Extensionless URI	• Enabled • Disabled	Disabled	Set this to 1 to send URIs without extension without headers.
HTTPD Limit Sending To 2mss	• Enabled • Disabled	Enabled	By default, the httpd is limited to send 2*pcb->mss.
HTTPD Maximum Write Length	The name function must be a valid C symbol	altcp_mss	Define this to a function that returns the maximum amount of data to enqueue. The function have this signature: u16_t fn(struct altcp_pcb* pcb);.
LwIP HTTPD Custom Files	• Enabled • Disabled	Disabled	Set this to 1 and provide the functions: - int

			fs_open_custom(struct fs_file *file, const char *name) Called first for every opened file to allow opening files that are not included in fsdata(_custom).c - void fs_close_custom(struct fs_file *file) Called to free resources allocated by fs_open_custom().
LwIP HTTPD Dynamic File Read	• Enabled • Disabled	Disabled	Set this to 1 to support fs_read(). Else only one-block files are supported, and the contents must be ready after fs_open().
LwIP HTTPD File State	• Enabled • Disabled	Disabled	Set this to 1 to include an application stateargument per file that is opened.
LwIP HTTPD File Extension	• Enabled • Disabled	Disabled	Set this to 1 to add the pextension field to the fs_file structure.New code should use LWIP_HTTPD_FILE_STATE instead.
HTTPD Precalculated Checksum	• Enabled • Disabled	Disabled	Support asynchronous read operations.
LwIP HTTPD FS Async Read	• Enabled • Disabled	Disabled	Set this to 1 to add the pextension field.
HTTPD Fsdata File	Manual Entry	fsdata_sample.c	Provide source name file of FS custom data. This data file will be included by fs.c file
HTTP Is Data Volatile	• Allow Copy and More • Allow Copy • Undefined	Allow Copy	Determines if HTTP data should be copied into the TCP buffer before transmission. Allow Copy: data will be copied into memory belonging to the stack. Allow Copy and More: Notify to TCP that data will be copied into memory belonging to the stack and additional data will

			follow in subsequent TCP transmissions.
LwIP HTTPD Use Default Template	• Enabled FSdata Generator • Disabled FSdata Generator	Enabled FSdata Generator	Set this to 1 to when intent to use the default template from the generator

Usage Notes

Limitations

- In normal mode, the user must select the "HTTP Is Data Volatile" option as Allow Copy mode (TCP_WRITE_FLAG_COPY) in the FSP Configurator properties to ensure tcp_write() functions correctly.
- When using the Server-Side Includes (SSI) mode or the Common Gateway Interface (CGI) mode, make sure that the "HTTP Is Data Volatile" option is set to Allow Copy and More mode (TCP_WRITE_FLAG_COPY | TCP_WRITE_FLAG_MORE) in the FSP Configurator properties to work properly.

How to use fsdata file

There are two ways to use fsdata file as follows:

1. Use the available fsdata_sample.c file

When the stack "LwIP HTTP Server" is added, by default it includes the stack "LwIP HTTP Server FS" which is the sample file provided in the LwIP HTTP Server library repository, and the fsdata_sample.c file is generated at the src/fsdata/ path.

2. Generate new fsdata.c file using makefsdata script

In case, creating a new fsdata.c file by makefsdata script, follow these detailed steps:

Note

Reffer to this link readme.txt for an introduction by the author about the makefsdata scripts.

- ***Setup Environment***
  1. *Make sure Perl is installed on your development environment to ensure the required script runs.*
- ***Convert Server Contents to C Source Code***
  1. *Create a new folder. (the created folder can be placed outside the project workspace).*
  2. *Copy the "makefsdata" folder from the LwIP HTTP Server library repository into the created folder.*
  3. *Inside the created folder, create a new subfolder named "fs" (the name "fs" is mandatory).*
  4. *Place the files to be converted into the "fs" folder.*
  5. *Open a terminal on Linux or Git Bash/Windows Command Prompt on Windows. (if Cygwin is installed) and navigate to the created folder.*
  6. *Type the command "perl makefsdata".*
  7. *The new "fsdata.c" file will be generated in the created folder.*
- ***Configure the Generated File***
  1. *Remove the the stack "LwIP HTTP Server FS".*
  2. *Add an "include guard" to prevent multiple inclusions in the new file "fsdata.c".*
  3. *Include the headers "lwip/apps/fs.h" and "lwip/def.h" in the new file "fsdata.c".*
  4. *Copy only the new file "fsdata.c" into the project workspace.*
  5. *In the FSP Configurator properties, set the "HTTPD Fsdata File" option to the new file*

"fsdata.c".

TLS connection

The stack and heap sizes must be properly configured when using the TLS layer with MbedTLS. Depending on the specific configuration and use case, these values may need to be increased from their defaults. For more information on memory requirements, refer to the **Memory Usage** section in Mbed Crypto H/W Acceleration (rm_psa_crypto).

Note

*When using hardware acceleration for the MbedTLS implementation, verify that the target device supports all required mbedCrypto features for the TLS layer of lwIP. Refer to the **HW Overview** and **Features** sections in Mbed Crypto H/W Acceleration (rm_psa_crypto) to check hardware support. If any required feature is not supported, configure the corresponding features to use the software crypto implementation instead.*

Examples

Basic Example

This is a basic example of minimal use of the LwIP HTTP Server APIs in an application.

```
#if HTTPD_ENABLE_HTTPS
 #include "lwip/altcp_tls.h"
 #include "mbedtls/platform.h"
#endif
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#include "lwip/apps/httpd.h"
#if HTTPD_ENABLE_HTTPS
/* Example server key, needs to be updated. */
static uint8_t gs_server_key[] =
 "-----BEGIN PRIVATE KEY-----\n"
 "MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDKrJdfXTLDsPb4\n"
 "7ACEg7kfMYuISDOaIgTlZBUSfAbeozju5zZoTigOTbJG4oUgV52qkoumSQ8L6c80\n"
 "RA+0hnslzuYXo2mz3zdlLBL1PZwXOvQIJ1IdveYIiH7DIsEiQ5o1wvPz7RrKCUGY\n"
 "CfhDLFfxgZUDQ51p31V1KgU92pIgEJUfrmKyYT1HK3dQC9QP4AkmUrADmZInhzku\n"
 "NktnE9/aZAC1DqVH+bcIizT4WJDY7wY1lRAM9GghRhJOOGPDbBfCrUsTd7qOd1pT\n"
 "BPGTnGVskrmC2YsO+Xtc0Mbh+26V6rfES7QmKtlLPDnApnyEPe4EzlluMhXLKI6M\n"
 "UhGWRp8TAgMBAAECggEBAKdS2h33OaPqA9v6K2s7vKgOJV48Da4gi94+RE9gaeOA\n"
 "6c4CwR5X+CXm61ouRS4wZ5CdMAH2qx9wxool7rlhlWXfbILzfJ7tmDREsNobJTTg\n"
 "jPv2SosXbkR5O/4qQTzihvv/mZw2kjlebvLwuC/s+iX+j2GEfv8bTNhP+sfYHW+u\n"
```

```
 "CL11deCRosu5GHe5Jgs2YSGmi1sMrrzAMBIq/K6SkjzM+Ty9e6AhW0/EAkqeHZmU\n"
 "jUT7MVgfATpf1jBe5Blgu0/ZRixdb01PeOxxEmqOdjl6sKLneczNS7i1yUsAjR7J\n"
 "dlLjNvp7bz91zl08BoRBk/XdkNKkWuVA5Eo/XMotcLECgYEA64i9rymLb5EVbP56\n"
 "a9MJVnmQQ3IA3nX26Vi3ksHyzSeegazA3xQxHM2R99dNJNvXP4OkNyL1TZQVVhsb\n"
 "rHR9QFMk1WxoLmw42krba1gdQ8UMh/C21HN7Q7eyVDYoEWHOBoKWQoZI6SFgLx8W\n"
 "S4CRZObExsRxtknHr7NUuFiINrUCgYEA3EjofZrzENzzZUzzsKVm2q8R6GIhYC06\n"
 "5VFcBEsoz1NNzyf93iLDwCluvUqt8jsaQsjNDpwdeSe273f80Hp3xqe7ADg5O6rK\n"
 "S2ZMmWpcuugMI/l9KnhLmQmFz62phL4ioT5ojIUBhs8J/oXrldv1K/7ZkuOj0179\n"
 "v6vhsW2dk6cCgYABskJreIkgjJLy7gJDtIlkrgtz8OZAs+jLVXA9BliRUa+6j1v0\n"
 "P1j7NB/RASP1GAA83N3Z586MeHyO2jMg0tX0GKUzREybyPu/1rot07yiOFEWyY0H\n"
 "iq1gl3oZUSqItYOeWGJsqDruMudWb4rhkFN7vo28QL3w6iceBNSfX0Pb4QKBgEs4\n"
 "jkNN5RBuEqMV2uUIJg4o2BmdYL5B06m41Y4VyDK7xIcnpdXz0xpMS6dR2AIuUnJZ\n"
 "tUP25U2Y1SwCsg7SsxHf9shFaq3ITVnf7eo5Wr7sPDEQocAd4dq5WLspFwjgEF01\n"
 "PWg2huXszwHd4yuCN781QECbgTbNaiLIY8I9pyBjAoGBAIY8y7uN+pU2LdW7LG56\n"
 "E6wxhSl9Epz1PV31W93gQyOfY8RZSZ3upbe4uylg2ONe3m7cLB3cPg4ohSxM84RI\n"
 "60jvXHWPi+DG0Ub0Y3E1O6ZAW1UnrE3Q+J01bhNK8oy3X4sht9dg6VFZK8UFCk4V\n"
 "cPZoDs+ykQpvVD8dnII31qAw\n"
 "-----END PRIVATE KEY-----";
/* Example server certificate, needs to be updated. */
static uint8_t g_server_crt[] =
 "-----BEGIN CERTIFICATE-----\n"
 "MIIDbTCCAlWgAwIBAgIUKvNkxPfd58IpndbgRLZetCiVS84wDQYJKoZIhvcNAQEL\n"
 "BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM\n"
 "GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAgFw0yMDA5MTExNjE5NThaGA8yMjAw\n"
 "MDIxNjE2MTk1OFowRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUx\n"
 "ITAfBgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDCCASIwDQYJKoZIhvcN\n"
 "AQEBBQADggEPADCCAQoCggEBAMqsl19dMsOw9vjsAISDuR8xi4hIM5oiBOVkFRJ8\n"
 "Bt6jOO7nNmhOKA5NskbihSBXnaqSi6ZJDwvpzzRED7SGeyXO5hejabPfN2UsEvU9\n"
 "nBc69AgnUh295giIfsMiwSJDmjXC8/PtGsoJQZgJ+EMsV/GBlQNDnWnfVXUqBT3a\n"
 "kiAQlR+uYrJhPUcrd1AL1A/gCSZSsAOZkieHOS42S2cT39pkALUOpUf5twiLNPhY\n"
 "kNjvBjWVEAz0aCFGEk44Y8NsF8KtSxN3uo53WlME8ZOcZWySuYLZiw75e1zQxuH7\n"
 "bpXqt8RLtCYq2Us8OcCmfIQ97gTOWW4yFcsojoxSEZZGnxMCAwEAAaNTMFEwHQYD\n"
 "VR0OBBYEFCL0XSgJ4bT7qqLtjdNHHfDhXoV5MB8GA1UdIwQYMBaAFCL0XSgJ4bT7\n"
 "qqLtjdNHHfDhXoV5MA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEB\n"
```

```
 "AIREt6x3cwQeRHaNkBjuYbOmB1LXa1az32d4nrvHG9aVfAOlIZuI2Zz3l3JJFT1M\n"
 "7NsSyhYojOgagEFs+pfgW6o6YG62e8Stne2J1wppGNqckLTJyeYnAFzKakfc9KXU\n"
 "Cq4v+JesOwzXmz3hbZywZ5rDJJHt/BoixL6C82fHUFNgzyH4dFpGWbndH29p8wr4\n"
 "hiTR4JxH5qlogMmWqnIJ26pOu26+xfPVhLy4o84MZl0e5QqRg05LBrFd5c8/BKxq\n"
 "Cr1qcy3Gc67/skwXMO1Dm+OSmDyGgvpAC0XXX1XpY3+/Y2yIT2glx09JsnyoXfES\n"
 "pasiCAXmwq48oPhw9PKoYCQ=\n"
 "-----END CERTIFICATE-----";
#endif
/* LwIP HTTPD example */
void lwip_httpd_example (void) {
 struct netif netif;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
#if !HTTPD_ENABLE_HTTPS
 /* Initialize the HTTP server. */
    httpd_init();
#else
 /* Setup the platform; initialize the SCE. */
 mbedtls_platform_setup(NULL);
```

```
 /* Create an TLS server configuratio.n */
 struct altcp_tls_config * configuration =
altcp_tls_create_config_server_privkey_cert((const u8_t *) gs_server_key,
 sizeof(gs_server_key),

     NULL,

     0,

     (const u8_t *) g_server_crt,
 sizeof(g_server_crt));
 /* Initialize the HTTPS server. */
    httpd_inits(configuration);
#endif
 /* Program main loop. */
 while (true)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
}
```

5.2.12.15 lwIP Iperf

Modules » Networking

This module provides the lwIP Iperf integration documentation.

Overview

The lwIP Iperf is a simple performance measuring bandwidth using iPerf2. The documentation for the library can be found at the following link: lwIPerf.

Features

- Check your bandwidth using Iperf2.
- Supports TCP stream.

- Compatible with Iperf2.
- Both iPerf2 server and client modes are supported.

Configuration

Build Time Configurations for source

The following build time configurations are defined in lwip/lwip/apps/lwiperf_opts.h:

Configuration	Options	Default	Description
Idle Timeout	Value must be a positive integer value and less than or equal to 255.	10	Specify the idle timeout (in seconds) after that the test fails.
lwIPerf IP version	• IPv4 • IPv6 • IPv4+IPv6(Dual-stack)	IPv4+IPv6(Dual-stack)	Configure whether lwIPerf listens on all IP versions.
Check format of received data	• Enabled • Disabled	Disabled	Enabled to verify the format of received data.

Usage Notes

Limitations

- UDP is not supported due to the limitation of lwIP OSS.
- Iperf3 is not supported due to the limitation of lwIP OSS.

Examples

Basic Example

This is a basic example of minimal use of the lwIP Iperf APIs in an application.

Note

When using printf with e² studio, build errors may occur. Please refer to https://en-support.renesas.com/knowledgeBase/16979385 and follow Method 2, which is recommended for smooth integration and to avoid build errors.

```
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/netif.h"
#include "lwip/apps/lwiperf.h"
/* LwIP Iperf example */
static void lwiperf_report_example(void * arg,
 enum lwiperf_report_type report_type,
```

```
 const ip_addr_t       * local_addr,
                                 uint16_t                local_port,
 const ip_addr_t       * remote_addr,
                                 uint16_t                remote_port,
                                 uint32_t                bytes_transferred,
                                 uint32_t                ms_duration,
                                 uint32_t                bandwidth_kbitpsec);
/* Define a report callback function */
static void lwiperf_report_example (void * arg,
 enum lwiperf_report_type report_type,
 const ip_addr_t       * local_addr,
                                  uint16_t                local_port,
 const ip_addr_t       * remote_addr,
                                  uint16_t                remote_port,
                                  uint32_t                bytes_transferred,
                                  uint32_t                ms_duration,
                                   uint32_t                bandwidth_kbitpsec)
{
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(local_addr);
 FSP_PARAMETER_NOT_USED(local_port);
    printf(
 "IPERF report: type=%d, remote: %s:%u, total bytes: %" PRIu32 " bytes, duration in
ms: %" PRIu32 " ms, kbits/s: %" PRIu32 " kbit/s\n",
       (int) report_type,
       ipaddr_ntoa(remote_addr),
       (int) remote_port,
       bytes_transferred,
       ms_duration,
       bandwidth_kbitpsec);
}
void lwip_iperf_server_example (void) {
 struct netif netif;
    ip_addr_t ipaddr;
```

```
    ip_addr_t netmask;
    ip_addr_t gw;
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
 /* Start lwiperf server */
    lwiperf_start_tcp_server_default(lwiperf_report_example, NULL);
 /* Program main loop. */
 while (true)
    {
       sys_check_timeouts();
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
void lwip_iperf_client_example (void)
{
 struct netif netif;
    ip_addr_t   ipaddr;
    ip_addr_t   netmask;
    ip_addr_t   gw;
    ip_addr_t * server_ip_address;
 /* Initializing lwIP core. */
    lwip_init();
```

```
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
 /* Start lwiperf client */
    server_ip_address = &netif.gw;
    lwiperf_start_tcp_client_default(server_ip_address, lwiperf_report_example,
NULL);
 /* Program main loop. */
 while (true)
    {
       sys_check_timeouts();
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

5.2.12.16 lwIP MQTT

Modules » Networking

This module provides the lwIP MQTT integration documentation.

Overview

The lwIP MQTT library can connect to a third party MQTT broker such as Mosquitto. The documentation for the library can be found at the following link: lwIPMQTT.

Features

- MQTT connections to MQTT broker
- Support set up TLS connection handling with one-way server authentication or two-way server/client authentication
- The version of MQTT protocol supported in lwIP is 3.1.1. Details of the MQTT protocol can be found at: mqtt-v3.1.1

Usage Notes

TLS connection

The stack and heap sizes must be properly configured when using the TLS layer with MbedTLS. Depending on the specific configuration and use case, these values may need to be increased from their defaults. For more information on memory requirements, refer to the **Memory Usage** section in Mbed Crypto H/W Acceleration (rm_psa_crypto).

Note

*When using hardware acceleration for the MbedTLS implementation, verify that the target device supports all required mbedCrypto features for the TLS layer of lwIP. Refer to the **HW Overview** and **Features** sections in Mbed Crypto H/W Acceleration (rm_psa_crypto) to check hardware support. If any required feature is not supported, configure the corresponding features to use the software crypto implementation instead.*

Examples

Basic Example

This is a basic example of minimal use of the lwIP MQTT APIs in an application.

```
#if LWIP_ALTCP && LWIP_ALTCP_TLS
 #include "lwip/altcp_tls.h"
 #include "mbedtls/platform.h"
#endif
#include "lwip/tcp.h"
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#include "lwip/apps/mqtt.h"
#define LWIP_MQTT_EXAMPLE_PORT_NUMBER 8883
#define LWIP_MQTT_EXAMPLE_TOPIC "/lwip/example"
#define LWIP_MQTT_EXAMPLE_TOPIC_LENGTH 13
#define LWIP_MQTT_EXAMPLE_BUF_SIZE 50
#define LWIP_MQTT_EXAMPLE_KEEP_ALIVE 100
/* LwIP MQTT callback functions. */
static void lwip_mqtt_example_connection_callback(mqtt_client_t * client, void * arg,
mqtt_connection_status_t status);
```

```
static void lwip_mqtt_example_pub_sub_request_callback(void * arg, err_t result);
static void lwip_mqtt_example_incoming_publish_callback(void * arg, const char *
topic, u32_t tot_len);
static void lwip_mqtt_example_incoming_data_callback(void * arg, const u8_t * data,
u16_t len, u8_t flags);
volatile uint8_t g_lwip_mqtt_example_connected_flag;
volatile uint8_t g_lwip_mqtt_example_receive_incomming_msg_flag;
volatile uint8_t g_lwip_mqtt_example_pub_sub_flag;
static unsigned char g_mqtt_lwip_example_incoming_msg[LWIP_MQTT_EXAMPLE_BUF_SIZE];
static char g_mqtt_lwip_example_incoming_topic[LWIP_MQTT_EXAMPLE_BUF_SIZE];
/* LwIP MQTT connection information */
const struct mqtt_connect_client_info_t lwip_mqtt_example_client_info =
{
 "lwip_mqtt_example",               // Client identifier
    NULL,                              // User name
    NULL,                              // Password
    LWIP_MQTT_EXAMPLE_KEEP_ALIVE,      // Keep alive time
    NULL,                              // Will_topic
    NULL,                              // Will_msg
    0,                                 // Will_qos
    0                                  // Will_retain
#if LWIP_ALTCP && LWIP_ALTCP_TLS
    , NULL                            // TLS configuration
#endif
};
#if LWIP_ALTCP && LWIP_ALTCP_TLS
/* Example ca certificate, needs to be updated. */
static uint8_t gs_ca_crt[] =
 "-----BEGIN CERTIFICATE-----\n"                                    \
 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
 "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
 "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
 "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
```

```
 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
 "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
 "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
 "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
 "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
 "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
 "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
 "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
 "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
 "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
 "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
 "rqXRfboQnoZsG4q5WTP468SQvvG5\n"                                  \
 "-----END CERTIFICATE-----";
#endif
void lwip_mqtt_example (void) {
 struct netif    netif;
    mqtt_client_t * g_lwip_mqtt_example_client;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
    ip_addr_t server_ip_address;
 const char data[] = "lwIP MQTT Hello world.";
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable */
```

```
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
    IP4_ADDR(&server_ip_address, 192, 168, 10, 1);
 /* Initialize lwIP MQTT Client */
    g_lwip_mqtt_example_client = mqtt_client_new();
 /* Establish MQTT connection to server */
    g_lwip_mqtt_example_connected_flag = false;
#if LWIP_ALTCP && LWIP_ALTCP_TLS
 /* Create a TLS client configuration */
 struct altcp_tls_config * configuration =
       altcp_tls_create_config_client((const u8_t *) gs_ca_crt, sizeof(gs_ca_crt));
 /* Initializing TLS configuration */
    lwip_mqtt_example_client_info.tls_config = configuration;
#endif
    assert(ERR_OK ==
          mqtt_client_connect(g_lwip_mqtt_example_client, &server_ip_address,
LWIP_MQTT_EXAMPLE_PORT_NUMBER,
                              lwip_mqtt_example_connection_callback, NULL,
&lwip_mqtt_example_client_info));
 /* Waiting for connection to be established. */
 while (false == g_lwip_mqtt_example_connected_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Set callback for receiving incomming messsage from subscribed MQTT topic */
    g_lwip_mqtt_example_receive_incomming_msg_flag = false;
    mqtt_set_inpub_callback(g_lwip_mqtt_example_client,
                            lwip_mqtt_example_incoming_publish_callback,
                            lwip_mqtt_example_incoming_data_callback,
                            NULL);
 /* Subcribe to MQTT topic with Qos 0 */
    g_lwip_mqtt_example_pub_sub_flag = false;
```

```
    assert(ERR_OK ==
          mqtt_subscribe(g_lwip_mqtt_example_client, LWIP_MQTT_EXAMPLE_TOPIC, 0,
                         lwip_mqtt_example_pub_sub_request_callback, NULL));
 /* Waiting for subcribe process completed. */
 while (false == g_lwip_mqtt_example_pub_sub_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Publish message to MQTT topic with Qos 0 and retain flag disabled */
    g_lwip_mqtt_example_pub_sub_flag = false;
    assert(ERR_OK ==
          mqtt_publish(g_lwip_mqtt_example_client, LWIP_MQTT_EXAMPLE_TOPIC, data,
(u16_t) strlen(data), 0, 0,
                       lwip_mqtt_example_pub_sub_request_callback, NULL));
 /* Waiting for publish process completed. */
 while (false == g_lwip_mqtt_example_pub_sub_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Waiting for receive incomming process completed. */
 while (false == g_lwip_mqtt_example_receive_incomming_msg_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Make sure receiving correct topic and message */
    assert(0 == memcmp(g_mqtt_lwip_example_incoming_topic, LWIP_MQTT_EXAMPLE_TOPIC,
sizeof(LWIP_MQTT_EXAMPLE_TOPIC)));
    assert(0 == memcmp(g_mqtt_lwip_example_incoming_msg, data, sizeof(data)));
 /* Unsubcribe to MQTT topic */
    assert(ERR_OK ==
          mqtt_unsubscribe(g_lwip_mqtt_example_client, LWIP_MQTT_EXAMPLE_TOPIC,
```

```
                            lwip_mqtt_example_pub_sub_request_callback, NULL));
 /* Close MQTT connection and netif. */
    mqtt_disconnect(g_lwip_mqtt_example_client);
    netif_remove(&netif);
    mqtt_client_free(g_lwip_mqtt_example_client);
}
/* LwIP MQTT callback functions */
void lwip_mqtt_example_incoming_publish_callback (void * arg, const char * topic,
u32_t tot_len)
{
 FSP_PARAMETER_NOT_USED(arg);
    memcpy(g_mqtt_lwip_example_incoming_topic, topic, tot_len);
}
void lwip_mqtt_example_incoming_data_callback (void * arg, const u8_t * data, u16_t
len, u8_t flags)
{
 FSP_PARAMETER_NOT_USED(arg);
 if (flags & MQTT_DATA_FLAG_LAST)
    {
       memcpy(g_mqtt_lwip_example_incoming_msg, data, (size_t) len);
       g_lwip_mqtt_example_receive_incomming_msg_flag = true;
    }
}
void lwip_mqtt_example_connection_callback (mqtt_client_t * client, void * arg,
mqtt_connection_status_t status)
{
 FSP_PARAMETER_NOT_USED(client);
 FSP_PARAMETER_NOT_USED(arg);
 if (MQTT_CONNECT_ACCEPTED == status)
    {
       g_lwip_mqtt_example_connected_flag = true;
    }
}
void lwip_mqtt_example_pub_sub_request_callback (void * arg, err_t result)
```

```
{
 FSP_PARAMETER_NOT_USED(arg);
 if (ERR_OK == result)
    {
       g_lwip_mqtt_example_pub_sub_flag = true;
    }
}
```

5.2.12.17 lwIP NETBIOSNS

Modules » Networking

This module implements the lwIP NETBIOSNS integration documentation.

Overview

This is a simple lwIP NetBIOS nameserver. It responds to name queries from a configurable name.

Features

The lwIP NetBIOS Responder opens a UDP socket on port 137 to handle NetBIOS name resolution requests:

- Respond to NB name query type, resolving the configured hostname to IPv4 address (always enabled).
- Respond to NBSTAT name query type with both exact-name and wildcard (*) queries (configurable).

Configuration

Build Time Configurations for source

The following build time configurations are defined in lwip/lwip/apps/netbiosns_opts.h:

Configuration	Options	Default	Description
NetBIOS Name Query Response	• Enabled • Disabled	Enabled	Enabling responses to NBSTAT queries is configurable, while responses to NB name queries are always enabled.
Use static NETBIOS name	• Enabled • Disabled	Disabled	Disable this option to allow changing the NetBIOS host name at

			runtime using netbiosns_set_name().
NetBIOS Name	String length minimum 1 maximum 15, allow A-Z,0-9, -, ., must not start with period (.).	LWIPDEVICE	Specifies a string for NetBIOS name. String length 1 to 15. Allowed uppercase letters A-Z, digit0-9, hyphen -, period .. Must not start with period.

Usage Notes

Note

It is recommended to call netbiosns_stop() before reinitializing the NetBIOS Responder to avoid a potential memory leak.

Limitations

Note

1. *Name conflict resolving is not supported.*
2. *The socket does not broadcast it's own name, thus can not detect duplicate names.*
3. *The socket always resolves name queries using the IP address of the default network interface.*

Examples

Basic Example

This is a basic example of initializing NETBIOSNS responder and change hostname in run time.

```
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#include "lwip/apps/netbiosns.h"
struct netif g_netbiosns_netif;
/* LwIP netbiosns example */
void lwip_netbiosns_example (void) {
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
```

```
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&g_netbiosns_netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&g_netbiosns_netif);
    netif_set_up(&g_netbiosns_netif);
    netif_set_link_up(&g_netbiosns_netif);
#ifndef NETBIOS_LWIP_NAME
 /* Set the NetBIOS name, host name can be change again during run time using
netbiosns_set_name */
    netbiosns_set_name("LWIPDEVICE");
#endif
 /* Init netbios responder */
    netbiosns_init();
 /* Program main loop. */
 while (true)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
}
```

5.2.12.18 lwIP SMTP

Modules » Networking

This module provides the lwIP SMTP integration documentation.

Overview

The documentation for the library can be found at the following link: lwIPSMTP.

Features

- Support authentication methods and send an email via the currently selected server, username and password.
- Support set up TLS connection handling with one-way server authentication or two-way server/client authentication.
- The common ports that used for SMTP:
    - Port 25:
        - Default port for SMTP since the beginning of email.
        - Supports plain, unencrypted SMTP.
    - Port 465:
        - Used for SMTPS (SMTP Secure), where the connection is immediately encrypted using SSL/TLS upon connection.
    - Custom ports: Based on the port of SMTP server.

    Note

    SMTP port usage reference in the RFC8314 documentation.

Configuration

Build Time Configurations for source

The following build time configurations are defined in lwip/lwip/apps/smtp_opts.h:

Configuration	Options	Default	Description
Data Handler Callback on BODY	• Enabled • Disabled	Disabled	Enable this option to invoke the user-defined callback for SMTP message BODY data.
SMTP Debug	• LWIP_DBG_ON • LWIP_DBG_OFF	LWIP_DBG_OFF	Set this to 1 to debug SMTP.
Maximum SMTP Server Name Length	Value must be a positive integer value	256	Specifies the maximum length of the SMTP server name.
Maximum SMTP Username Length	Value must be a positive integer value	32	Specifies the maximum length of the SMTP username.
Maximum Password Length	Value must be a positive integer value	32	Specifies the maximum length of the SMTP password.
Copy Authentication Data	• Enabled • Disabled	Enabled	Enable to copy username and password into internal buffers instead of using pointers to user data. Disabling this if you know the authentication data will not change during the smtp session, which saves some heap space.

Check Email Data	• Enabled • Disabled	Enabled	Enable to verify SMTP message fields for validity before transmission. Disabling this to save some code space if you know for sure that all data passed to this module conforms to the requirements in the SMTP RFC. WARNING: Use this with care!
AUTH PLAIN Support	• Enabled • Disabled	Enabled	Enable to support SMTP AUTH PLAIN authentication.
AUTH LOGIN Support	• Enabled • Disabled	Enabled	Enable to support SMTP AUTH LOGIN authentication.
Memory allocation	Name must be a valid C symbol	mem_malloc	Define the allocation function used for SMTP state machine memory. By default, uses lwIP mem_malloc. Override to FreeRTOS heap for RTOS-based systems.
Memory deallocation	Name must be a valid C symbol	mem_free	Define the free function used for SMTP state machine memory. By default, uses lwIP mem_free. Override to FreeRTOS heap for RTOS-based systems.

Usage Notes

TLS connection

The stack and heap sizes must be properly configured when using the TLS layer with MbedTLS. Depending on the specific configuration and use case, these values may need to be increased from their defaults. For more information on memory requirements, refer to the **Memory Usage** section in Mbed Crypto H/W Acceleration (rm_psa_crypto).

Note

*When using hardware acceleration for the MbedTLS implementation, verify that the target device supports all required mbedCrypto features for the TLS layer of lwIP. Refer to the **HW Overview** and **Features** sections in Mbed Crypto H/W Acceleration (rm_psa_crypto) to check hardware support. If any required feature is not supported, configure the corresponding features to use the software crypto implementation instead.*

Authetication

The lwIP SMTP module supports the following authentication methods:

- **No Authentication**: Use smtp_set_auth(NULL, NULL) for servers that don't require authentication.
- **Authentication**: Use smtp_set_auth(username, password) for username/password authentication.
    - AUTH LOGIN: The username and password are sent separately. The server first asks for the username, then the password.
    - AUTH PLAIN: The username and password are sent together.
    - The client automatically selects PLAIN or LOGIN based on server capabilities.
    - Both methods use Base64 encoding (NOT encryption).

Note

Authentication credentials are sent in plain text (Base64 encoded only) unless TLS is enabled. Base64 can be easily decoded by anyone intercepting network traffic.

Always use TLS when sending credentials:

```
// INSECURE - Credentials visible on network
smtp_set_tls_config(NULL);
smtp_set_auth("user", "password");
// SECURE - Credentials encrypted
smtp_set_tls_config(tls_config); // Enable TLS encryption
smtp_set_auth("user", "password");
```

Examples

Basic Example

This is a basic example of minimal use of the lwIP SMTP APIs in an application.

Note

When using printf with e² studio, build errors may occur. Please refer to https://en-support.renesas.com/knowledgeBase/16979385 and follow Method 2, which is recommended for smooth integration and to avoid build errors.

```
#if LWIP_ALTCP && LWIP_ALTCP_TLS
 #include "lwip/altcp_tls.h"
 #include "mbedtls/platform.h"
#endif
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/netif.h"
#include "lwip/apps/smtp.h"
```

```
#define LWIP_SMTP_GW_ADDR "192.168.10.1"
#define LWIP_SMTP_PORT (1025)
#if LWIP_ALTCP && LWIP_ALTCP_TLS
/* Example ca certificate, needs to be updated. */
static uint8_t gs_ca_crt[] =
 "-----BEGIN CERTIFICATE-----\n"                                    \
 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
 "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
 "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
 "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
 "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
 "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
 "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
 "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
 "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
 "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
 "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
 "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
 "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
 "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
 "rqXRfboQnoZsG4q5WTP468SQvvG5\n"                                   \
 "-----END CERTIFICATE-----";
#endif
/* LwIP SMTP example */
/* Declare the result callback function */
static void lwip_smtp_result_example(void * arg, uint8_t smtp_result, uint16_t
srv_err, err_t err);
/* Define the result callback function */
static void lwip_smtp_result_example (void * arg, uint8_t smtp_result, uint16_t
srv_err, err_t err)
{
```

```
 FSP_PARAMETER_NOT_USED(arg);
    printf("Mail sent with results: %d, srv_err: %d, err: %d\n", smtp_result,
srv_err, err);
}
void lwip_smtp_example (void) {
 struct netif netif;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
#if LWIP_ALTCP && LWIP_ALTCP_TLS
 /* Create a TLS client configuration */
 struct altcp_tls_config * configuration =
       altcp_tls_create_config_client((const u8_t *) gs_ca_crt, sizeof(gs_ca_crt));
 /* Set TLS configuration */
    smtp_set_tls_config(configuration);
#endif
 /* Set the server address */
    smtp_set_server_addr(LWIP_SMTP_GW_ADDR);
 /* Set the server port */
    smtp_set_server_port(LWIP_SMTP_PORT);
```

```
/* Set the authentication data */
   smtp_set_auth("YourUserName", "YourPassword");
/* Send mail */
   smtp_send_mail("***@YourDomain.com",
"***@SomeOtherDomain.com",
"Subject",
"Email Body",
                  lwip_smtp_result_example,
                  NULL);
/* Program main loop. */
while (true)
   {
      sys_check_timeouts();
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
   }
}
```

5.2.12.19 lwIP SNTP

Modules » Networking

This module provides the lwIP SNTP integration documentation.

Overview

The lwIP SNTP library can be used to synchronize system time in a network by exchanging packets regularly. The documentation for the library can be found at the following link: lwIPSNTP.

Features

- This is simple "SNTP" client for the lwIP raw API. It is a minimal implementation of SNTPv4 as specified in RFC 4330.

Usage Notes

Note

MEMP_NUM_SYS_TIMEOUT at Infrastructure|Internal memory pools|Memp num sys timeout property of lwIP tcpip stack must be increased by one from the default value, as the SNTP module requires an additional timeout slot in the lwIP timer subsystem.

Examples

Basic Example

This is a basic example of minimal use of the lwIP SNTP APIs in an application.

Note

When using printf with e² studio, build errors may occur. Please refer to https://en-support.renesas.com/knowledgeBase/16979385 and follow Method 2, which is recommended for smooth integration and to avoid build errors.

```
#include <time.h>
#include "lwip/timeouts.h"
#include "lwip/ip_addr.h"
#include "lwip/init.h"
#include "lwip/apps/sntp.h"
#define LWIP_SNTP_EXAMPLE_SECONDS_PER_HOUR (60 * 60)
#define LWIP_SNTP_EXAMPLE_TIMEZONE_UTC_PLUS (7) // This may need to be updated for
your timezone
#define LWIP_SNTP_EXAMPLE_RETRY_CONNECT (5U)
#define LWIP_SNTP_EXAMPLE_SERVER_NAME "time1.google.com"
#define LWIP_SNTP_EXAMPLE_SERVER_ADDRESS "216.239.35.0"
/* TODO: Set this function name in the "SNTP Set System Time" property of lwIP SNTP
stack.
 * This function will be called to change the system time in seconds using the
timestamp received from the SNTP server.
 *
 * Example: lwip_sntp_example_set_time(sec)
 */
void lwip_sntp_example_set_time(u32_t sec);
uint8_t g_lwip_sntp_example_count;
bool g_lwip_sntp_example_flag;
/* LwIP SNTP example */
void lwip_sntp_example (void) {
 struct netif netif;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
```

```
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
 /* Set SNTP operation mode to Polling. */
    sntp_setoperatingmode(SNTP_OPMODE_POLL);
#if SNTP_SERVER_DNS && LWIP_DNS
 /* Set server as the supplied DNS. */
    sntp_setservername(0, LWIP_SNTP_EXAMPLE_SERVER_NAME);
#else
    ip_addr_t sntp_ip;
    ipaddr_aton(LWIP_SNTP_EXAMPLE_SERVER_ADDRESS, &sntp_ip);
 /* Set server as the supplied ip address. */
    sntp_setserver(0, &sntp_ip);
#endif
 /* Initialize the SNTP. */
    sntp_init();
 /* Program main loop. */
 while (true)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
 if (true == g_lwip_sntp_example_flag)
      {
```

```
            g_lwip_sntp_example_count++;
 /* Stop wait when the total count is reached. */
 if (LWIP_SNTP_EXAMPLE_RETRY_CONNECT == g_lwip_sntp_example_count)
      {
 break;
            }
       }
       g_lwip_sntp_example_flag = false;
    }
 /* Stop the current SNTP. */
 if (sntp_enabled())
    {
       sntp_stop();
    }
}
/* Change the current time responded from SNTP server example */
void lwip_sntp_example_set_time (u32_t sec)
{
 /* Set the flag to verify. */
    g_lwip_sntp_example_flag = true;
#if defined(__CCRX__) || defined(__ICCRX__)
 FSP_PARAMETER_NOT_USED(sec);
    printf("Received time value from SNTP Server\n");
#else
 /* Adjust clock to the your timezone. */
    sec += (LWIP_SNTP_EXAMPLE_TIMEZONE_UTC_PLUS *
LWIP_SNTP_EXAMPLE_SECONDS_PER_HOUR);
 /* Convert Unix time to local time structure. */
    time_t      time = (time_t) sec;
 struct tm * timeinfo;
    timeinfo = localtime(&time);
 char strftime_buf[32];             // NOLINT(readability-magic-numbers)
 /* Format time into "YYYY-MM-DD HH:MM:SS". */
    strftime(strftime_buf, sizeof(strftime_buf), "%Y-%m-%d %H:%M:%S", timeinfo);
```

```
    printf("Current UTC+%d time is: %s\n", LWIP_SNTP_EXAMPLE_TIMEZONE_UTC_PLUS,
strftime_buf);
#endif
}
```

5.2.12.20 lwIP Ethernet Driver (rm_lwip_ether)

Modules » Networking

Functions

err_t	rm_lwip_ether_init (struct netif *p_netif)
void	rm_lwip_ether_callback (ether_callback_args_t *p_args)

Detailed Description

Overview

This module provides a lwIP driver that is implemented using the Ethernet Interface.

Please refer to the lwIP documentation for further details.

Features

- Packet Types Supported
  - ARP
  - IPv4
  - IPv6
- UDP (User Datagram Protocol) including experimental UDP-lite extensions
- TCP (Transmission Control Protocol) with congestion control, RTT estimation fast recovery/fast retransmit and sending SACKs
- raw/native API for enhanced performance
- Link status callback

Configuration

Configurations for Networking > lwIP Wrapper to (rm_lwip_ether)

Configuration	Options	Default	Description
RTOS > RX thread stacksize	Must be a positive integer	1024	Stack size of RX thread.
RTOS > RX thread priority	Must be a positive integer	4	Priority of RX thread.
Name	Name must be a valid C symbol	g_lwip_ether0	Module name.

MTU	Must be an integer from 576 to 1500	1500	MTU size.
Netif flags	• UP • BROADCAST • LINK_UP • ETHARP • ETHERNET • IGMP • MLD6	module.middleware.lwip_ether.netif_flags.broadcast,module.middleware.lwip_ether.netif_flags.etharp	Flags representing the capabilities of the Ethernet device
Zero-copy mode	• Disable • Enable	Disable	Enable or disable zero-copy mode.
RX buffer pool size	Must be a positive integer	10	Buffer pool size for zero-copy RX.
Link check interval	Must be a positive integer	100	Interval for checking link status.

Usage Notes

Big endian

When big endian mode is selected, the **Networking | lwIP Compiler and Platform Abstraction** stack must be added, and its **Common|Platform abstraction|Endian Mode** setting must be changed to BIG_ENDIAN to ensure correct operation.

Limitations

Examples

Basic Example

This is a basic example of minimal use of the lwIP Ether Driver in an application.

```
#include "lwip/tcp.h"
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#define LWIP_EXAMPLE_PORT_NUMBER 9000
/* TCP callback functions. */
static err_t lwip_example_connected_callback(void * arg, struct tcp_pcb * tpcb, err_t
err);
err_t       lwip_example_sent_callback(void * arg, struct tcp_pcb * tpcb, u16_t len);
err_t       lwip_example_recv_callback(void * arg, struct tcp_pcb * tpcb, struct pbuf
* p, err_t err);
volatile uint8_t g_lwip_example_connected_flag;
```

```
volatile uint8_t g_lwip_example_sent_flag;
volatile uint8_t g_lwip_example_recv_flag;
void rm_lwip_ether_example (void) {
 struct netif    netif;
 struct tcp_pcb * pcb = NULL;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
    ip_addr_t server_ip;
 const char data[] = "lwIP Hello world.";
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
 /* Create a pcb for client. */
    pcb = tcp_new();
 /* Start connection to the server. */
    g_lwip_example_connected_flag = false;
    IP4_ADDR(&server_ip, 192, 168, 10, 1);
    tcp_connect(pcb, &server_ip, LWIP_EXAMPLE_PORT_NUMBER,
lwip_example_connected_callback);
 /* Waiting for connection to be established. */
 while (false == g_lwip_example_connected_flag)
    {
```

```
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Set callback for receiving complete. */
    g_lwip_example_recv_flag = false;
    tcp_recv(pcb, lwip_example_recv_callback);
 /* Set callback for sending complete. */
    g_lwip_example_sent_flag = false;
    tcp_sent(pcb, lwip_example_sent_callback);
 /* Send a message. */
    tcp_write(pcb, data, sizeof(data), TCP_WRITE_FLAG_COPY);
 while (false == g_lwip_example_sent_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 while (false == g_lwip_example_recv_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Close TCP client and netif. */
    tcp_close(pcb);
    netif_remove(&netif);
}
err_t lwip_example_connected_callback (void * arg, struct tcp_pcb * tpcb, err_t err)
{
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(err);
 FSP_PARAMETER_NOT_USED(tpcb);
    g_lwip_example_connected_flag = true;
 return err;
}
err_t lwip_example_sent_callback (void * arg, struct tcp_pcb * tpcb, u16_t len) {
```

```
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(tpcb);
 FSP_PARAMETER_NOT_USED(len);
    g_lwip_example_sent_flag = true;
 return ERR_OK;
}
err_t lwip_example_recv_callback (void * arg, struct tcp_pcb * tpcb, struct pbuf * p,
err_t err) {
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(tpcb);
 FSP_PARAMETER_NOT_USED(err);
    g_lwip_example_recv_flag = true;
 /* If the callback function returns ERR_OK it must have freed the pbuf, otherwise it
must not have freed it. */
 if (p != NULL)
    {
       pbuf_free(p);
    }
 return ERR_OK;
}
```

Function Documentation

◆ rm_lwip_ether_init()

err_t rm_lwip_ether_init (struct netif * *p_netif*)
Initialize ethernet hardware and lwIP network interface. This function is passed to netif_add()

◆ rm_lwip_ether_callback()

void rm_lwip_ether_callback (ether_callback_args_t * *p_args*)
Callback of Ethernet interrupt subroutine. This function is set to ethernet driver callback by configurator.

5.2.12.21 lwIP Baremetal Porting Layer (rm_lwip_sys_baremetal)

Modules » Networking

Overview

This module provides system timer feature for baremetal.

Please refer to the lwIP documentation for further details.

Features

This porting layer includes the following features.

- Timer
- Critical sections
- Compiler/Platform abstraction.

5.2.12.22 lwIP FreeRTOS Porting Layer (rm_lwip_sys_freertos)

Modules » Networking

Overview

This module provides a lwIP porting layer for FreeRTOS.

Please refer to the lwIP documentation for further details.

Features

This porting layer includes the following features.

- Compiler/Platform abstraction.

All other features of FreeRTOS porting layer are provided by the lwIP library.

5.2.12.23 NetX Duo Ethernet Driver (rm_netxduo_ether)

Modules » Networking

Overview

This module provides a NetX Duo driver that is implemented using the Ethernet Interface.

Please refer to the NetXDuo documentation for further details.

Features

- Packet Types Supported
  - ARP
  - IPv4
  - IPv6
- Link status callback
- Configurable IP MTU

Configuration

Configurations for Networking > NetX Duo Ethernet Driver (rm_netxduo_ether)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_netxduo_ether_0	Module name.
IP MTU	Value must be in the range [576, 1500] bytes.	1500	IP MTU

Usage Notes

Calculating the Packet Size for an IP instance

In order to ensure that there is enough space to store an entire Ethernet frame, the packet pool used for receiving packets must have a payload size that is 32 bytes larger than the configured ether_cfg_t::ether_buffer_size. The extra 32 bytes is needed in order to ensure that the allocated packets are properly aligned to 32 bytes.

ether_cfg_t::ether_buffer_size is calculated from the IP MTU using the following formula:

ceil((rm_netxduo_ether_cfg_t::mtu + Ethernet Header (14) + Padding Bytes (2)) / 32) * 32

Zero Copy Support

Zero copy support should only be disabled when a lower level ethernet driver doesn't support it. Disabling zero copy will negatively impact performance. When zero copy is disabled larger buffers will need to be allocated or link speed will need to be limited to counteract poor performance.

Limitations

Developers should be aware of the following limitations when using the rm_netxduo_ether with RMAC Ethernet driver:

- When using rm_netxduo_ether to establish a connection multiple times in a row, this operation is currently supported for only 1 Azure RTOS NetX Duo IP Instance.

Build Configuration

When building the NetX Duo project with CCRX compiler, the ctype.h (Character classification routines library) option must be enabled in the Library Generator settings. If this option is left

unchecked, the build process will fail.

Steps to configure:

- In e2 studio, right-click the project -> Properties
- Navigate to C/C++ Build -> Settings -> Library Generator -> Standard Library
- Check the box to enable the **ctype.h (C89/C99): Character classification routines (-head=ctype)** option
- Click **Apply and Close**

Examples

Basic Example

This is a basic example of minimal use of the NetX Duo Ether Driver in an application.

```
#define NETXDUO_EXAMPLE_IP_STACK_SIZE (2048U)
#define NETXDUO_EXAMPLE_ARP_CACHE_SIZE (2048U)
#define NETXDUO_EXAMPLE_PACKET_SIZE (1568U)
#define NETXDUO_EXAMPLE_PACKET_NUM (100U)
#define NETXDUO_EXAMPLE_PACKET_POOL_SIZE ((sizeof(NX_PACKET) +
NETXDUO_EXAMPLE_PACKET_SIZE) * \
  NETXDUO_EXAMPLE_PACKET_NUM)
#define NETXDUO_EXAMPLE_MAC_ADDRESS_MSW (0)
#define NETXDUO_EXAMPLE_MAC_ADDRESS_LSW (0)
static NX_IP g_ip;
static NX_PACKET_POOL g_packet_pool;
static uint8_t g_ip_stack_memory[NETXDUO_EXAMPLE_IP_STACK_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_packet_pool_memory[NETXDUO_EXAMPLE_PACKET_POOL_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_ip_arp_cache_memory[NETXDUO_EXAMPLE_ARP_CACHE_SIZE]
BSP_ALIGN_VARIABLE(4);
static void rm_netxduo_ether0 (NX_IP_DRIVER * driver_req_ptr)
{
 /* Pass the driver request and ethernet driver configuration to the NetX Duo Ether
Driver. */
    rm_netxduo_ether(driver_req_ptr, &g_netxduo_ether_instance);
}
void rm_netxduo_ether_example ()
```

```
{
    UINT status;
    nx_system_initialize();
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool,
 "Packet Pool",
                                   NETXDUO_EXAMPLE_PACKET_SIZE,
                                   &g_packet_pool_memory[0],
                                   NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
 /* Create an IP instance using the rm_netxduo_ether driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip,
 "IP Instance",
                          IP_ADDRESS(192, 168, 1, 2),
                          IP_ADDRESS(255, 255, 255, 0),
                          &g_packet_pool,
                          rm_netxduo_ether0,
                          &g_ip_stack_memory[0],
 sizeof(g_ip_stack_memory),
                          0);
    assert(NX_SUCCESS == status);
 /* Enable all modules that are required by the application. */
    status = nx_arp_enable(&g_ip, g_ip_arp_cache_memory, sizeof
(g_ip_arp_cache_memory));
    assert(NX_SUCCESS == status);
    status = nx_tcp_enable(&g_ip);
    assert(NX_SUCCESS == status);
    status = nx_icmp_enable(&g_ip);
    assert(NX_SUCCESS == status);
}
```

Changing MAC address Example

This is an example of changing the MAC address after an IP instance has already been created.

```
void rm_netxduo_ether_change_mac_address_example ()
{
    ULONG driver_status;
    UINT status;
 /* Disable the link. */
    status = nx_ip_driver_interface_direct_command(&g_ip, NX_LINK_DISABLE, 0,
&driver_status);
    assert(NX_SUCCESS == status);
 /* Update the MAC address. */
    status = nx_ip_interface_physical_address_set(&g_ip,
                                                  0,
                                                  NETXDUO_EXAMPLE_MAC_ADDRESS_MSW,
                                                  NETXDUO_EXAMPLE_MAC_ADDRESS_LSW,
                                                  NX_TRUE);
    assert(NX_SUCCESS == status);
 /* Re-enable the link. */
    status = nx_ip_driver_interface_direct_command(&g_ip, NX_LINK_ENABLE, 0,
&driver_status);
    assert(NX_SUCCESS == status);
}
```

5.2.13 Power

Modules

Detailed Description

Power Modules.

Modules

	Low Power Modes (r_lpm) Driver for the LPM peripheral on RX MCUs. This module implements the Low Power Modes Interface.

5.2.13.1 Low Power Modes (r_lpm)

Modules » Power

Functions

fsp_err_t	R_LPM_Open (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t	R_LPM_Close (lpm_ctrl_t *const p_api_ctrl)
fsp_err_t	R_LPM_LowPowerReconfigure (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t	R_LPM_LowPowerModeEnter (lpm_ctrl_t *const p_api_ctrl)
fsp_err_t	R_LPM_IoKeepClear (lpm_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the LPM peripheral on RX MCUs. This module implements the Low Power Modes Interface.

Overview

The low power modes driver is used to configure and place the device into the desired low power mode. Various sources can be configured to to wake from standby and deep standby mode.

Features

The LPM HAL module has the following key features:

- Supports the following low power modes:
  - Deep Software Standby mode (On supported MCUs)
  - Deep Sleep (On supported MCUs)
  - Software Standby mode
  - Sleep mode
  - Snooze mode (Not supported)
- Supports reducing power consumption when in deep software standby mode through internal power supply control and by resetting the states of I/O ports.
- Supports disabling and enabling the MCU's other hardware peripherals

Configuration

Build Time Configurations for r_lpm

The following build time configurations are defined in fsp_cfg/r_lpm_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled	Default (BSP)	If selected code for parameter checking is

	• Disabled		included in the build.
Standby Limit	• Enabled • Disabled	Disabled	If enabled, standby configuration only applies in R_LPM_LowPowerModeEnter. Otherwise, standby configuration applies to any WFI call.

Configurations for Power > Low Power Modes (r_lpm)

This module can be added to the Stacks tab via New Stack > Power > Low Power Modes (r_lpm).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_lpm0	Module name.
General > Low Power Mode	MCU Specific Options		Power mode to be entered.
General > Output port state in standby and deep standby	MCU Specific Options		Select the state of output pins during standby. Applies to address output, data output, and other bus control output pins.
General > Supply of SOSC clock to peripheral function in standby	MCU Specific Options		If enabled, SOSC clock is provided to peripheral functions in Software Standby and Snooze modes.
General > Startup speed of the HOCO in Standby and Snooze modes	MCU Specific Options		If enabled, the HOCO enters high-speed startup mode, shortening standby release time, and snooze transition time.
General > Flash mode in sleep or snooze	MCU Specific Options		Select flash mode in sleep mode or in snooze mode.
Deep Sleep and Standby Options > Snooze Options (Not available on every MCU) > Snooze Request Source	MCU Specific Options		Select the event that will enter snooze.
Deep Sleep and Standby Options > Snooze Options (Not available on every MCU) > Snooze End	MCU Specific Options		Enable wake from snooze from these sources.

Sources		
Deep Sleep and Standby Options > Snooze Options (Not available on every MCU) > DTC state in Snooze Mode	MCU Specific Options	Enable wake from snooze from this source.
Deep Sleep and Standby Options > Snooze Options (Not available on every MCU) > Snooze Cancel Source	MCU Specific Options	Select an interrupt source to cancel snooze.
Deep Sleep and Standby Options > Wake Sources	MCU Specific Options	Enable wake from deep sleep and standby from these Sources.
Deep Sleep and Standby Options > Wake Sources 2	MCU Specific Options	Enable wake from deep sleep and standby from these Sources.
Deep Sleep and Standby Options > Power-Supply Control	MCU Specific Options	Select the state of the internal power supply in standby mode.
RAM Retention Control (Not available on every MCU) > RAM retention in Standby mode	MCU Specific Options	Select the memory regions that are retained in standby mode.
RAM Retention Control (Not available on every MCU) > TCM retention in Deep Sleep and Standby modes	MCU Specific Options	Select if Tightly Coupled Memory (TCM) is retained in deep sleep and standby modes.
RAM Retention Control (Not available on every MCU) > Standby RAM retention in Standby and Deep Standby modes	MCU Specific Options	Select if Standby RAM is retained in standby and deep standby modes.
Oscillator LDO Control (Not available on every MCU) > PLL1 LDO State in standby mode	MCU Specific Options	Select the state PLL1 LDO state in standby mode.
Oscillator LDO Control (Not available on every MCU) > PLL2 LDO State in standby mode	MCU Specific Options	Select the state PLL2 LDO state in standby mode.
Oscillator LDO Control (Not available on every MCU) > HOCO LDO	MCU Specific Options	Select the state HOCO LDO state in standby mode.

State in standby mode		
Deep Standby Options (Not available on every MCU) > I/O Port Retention	MCU Specific Options	Select the state of the IO Pins after exiting deep standby mode.
Deep Standby Options (Not available on every MCU) > Power-Supply Control	MCU Specific Options	Select the state of the internal power supply in deep standby mode.
Deep Standby Options (Not available on every MCU) > Cancel Sources	MCU Specific Options	Enable wake from deep standby using these sources.
Deep Standby Options (Not available on every MCU) > Cancel Edges	MCU Specific Options	Falling edge trigger is default. Select sources to enable wake from deep standby with rising edge.

Clock Configuration

This module does not have any selectable clock sources.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Sleep Mode

At power on, by default sleep is set as the low-power mode. Sleep mode is the most convenient low-power mode available, as it does not require any special configuration (other than configuring and enabling a suitable interrupt or event to wake the MCU from sleep) to return to normal program-execution mode. The states of the SRAM, the processor registers, and the hardware peripherals are all maintained in sleep mode, and the time needed to enter and wake from sleep is minimal. Any interrupt causes the MCU device to wake from sleep mode, including the SysTick interrupt used by the RTOS scheduler.

Deep Sleep Mode

Deep sleep mode is similar to sleep mode with the exception that DTC and DMAC are stopped, access to TCM memory is not available, and only a subset of interrupts are available for waking the CPU.

Software Standby Mode

In software-standby mode, the CPU, as well as most of the on-chip peripheral functions and all of the internal oscillators, are stopped. The contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions, and I/O Ports are all retained. Software-standby mode allows significant reduction in power consumption, because most of the oscillators are stopped in this

mode. Like sleep mode, standby mode requires an interrupt or event be configured and enabled to wake up.

Snooze Mode

Snooze mode can be used with some MCU peripherals to execute basic tasks while keeping the MCU in a low-power state. Many core peripherals and all clocks can be selected to run during Snooze, allowing for more flexible low-power configuration than Software Standby mode. To enable Snooze, select "Software Standby mode with Snooze mode enabled" for the "Low Power Mode" configuration option. Snooze mode settings (including entry/exit sources) are available under "Standby Options".

Deep Software Standby Mode

The MCU always wakes from Deep Software Standby Mode by going through reset, either by the negation of the reset pin or by one of the wakeup sources configurable in the "Deep Standby Options" configuration group.

The Reset Status Registers can be used to determine if the reset occurred after coming out of deep software standby. For example, R_SYSTEM->RSTSR0_b.DPSRSTF is set to 1 after a deep software standby reset.

I/O Port Retention can be enabled to maintain I/O port configuration across a deep software standby reset. Retention can be cancelled through the R_LPM_IoKeepClear API.

Limitations

Developers should be aware of the following limitations when using the LPM:

- Reduced SRAM retention area in software standby mode is not supported.
- When using Snooze mode with SCI0 RXD as the snooze source the system clock must be HOCO and the MOCO, Main Oscillator and PLL clocks must be turned off.
- If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC module properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.
- When using the DC-DC regulator (where available), the MCU will temporarily switch to the LDO if Software Standby or Snooze is requested and back again when it is cancelled. Switching to the LDO incurs a 60 microsecond critical section wherein all interrupts AND peripherals are stopped. Switching back to DCDC from the LDO incurs an additional 22 microsecond critical section (peripherals running).
- On RX74, there are delays inserted before and after switching to a lower power mode based on BSP configuration options and current CPUCLK speed. These delays do not disable interrupts for the entire delay as required by the RX74 Hardware Manual. Interrupts should be disabled by the application when possible before sleeping to meet the requirements, or the control of interrupts and the delay should be managed by the user to ensure the requirement is fully met.

Examples

LPM Sleep Example

This is a basic example of minimal use of the LPM in an application. The LPM instance is opened and the configured low-power mode is entered.

```
void r_lpm_sleep (void)
{
 fsp_err_t err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_sleep);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
    assert(FSP_SUCCESS == err);
}
```

LPM Deep Software Standby Example

```
void r_lpm_deep_software_standby (void)
{
 fsp_err_t err;
#ifdef R_SYSTEM_RSTSR0_DPSRSTF_Msk
 /* Check the Deep Software Standby Reset Flag. If it is set, then the MCU is exiting
  * Deep Software Standby mode. */
 if (1U == R_SYSTEM->RSTSR0_b.DPSRSTF)
    {
 /* Clear the IOKEEP bit to allow I/O Port use.
  * Note that this function should be called before opening the LPM driver. */
       err = R_LPM_IoKeepClear(&g_lpm_ctrl);
      assert(FSP_SUCCESS == err);
    }
#endif
    err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Add user code here. */
 /* Reconfigure the module to set the IOKEEP bit before entering deep software
standby. */
    err = R_LPM_LowPowerReconfigure(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
    assert(FSP_SUCCESS == err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
 /* Code after R_LPM_LowPowerModeEnter when using Deep Software Standby never be
```

```
executed.
  * Deep software standby exits by resetting the MCU. */
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	lpm_instance_ctrl_t

Data Structure Documentation

◆ lpm_instance_ctrl_t

struct lpm_instance_ctrl_t
LPM private control block. DO NOT MODIFY. Initialization occurs when R_LPM_Open() is called.

Function Documentation

◆ R_LPM_Open()

fsp_err_t R_LPM_Open (lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg*)

Perform any necessary initialization

Return values

FSP_SUCCESS	LPM instance opened
FSP_ERR_ASSERTION	Null Pointer
FSP_ERR_ALREADY_OPEN	LPM instance is already open
FSP_ERR_UNSUPPORTED	This MCU does not support Deep Software Standby and Snooze Mode
FSP_ERR_INVALID_ARGUMENT	One of the following: • Invalid snooze entry source • Invalid snooze end sources
FSP_ERR_INVALID_MODE	One of the following: • Invalid low power mode • Invalid DTC option for snooze mode • Invalid deep standby end sources • Invalid deep standby end sources edges • Invalid power supply option for deep standby • Invalid IO port option for deep standby • Invalid output port state setting for standby or deep standby • Invalid sources for wake from standby mode • Invalid power supply option for standby • Invalid IO port option for standby • Invalid standby end sources • Invalid standby end sources edges

◆ R_LPM_Close()

fsp_err_t R_LPM_Close (lpm_ctrl_t *const *p_api_ctrl*)

Close the LPM Instance

Return values

FSP_SUCCESS	LPM driver closed
FSP_ERR_NOT_OPEN	LPM instance is not open
FSP_ERR_ASSERTION	Null Pointer

◆ R_LPM_LowPowerReconfigure()

fsp_err_t R_LPM_LowPowerReconfigure (lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg*)

Configure a low power mode

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

FSP_SUCCESS	Low power mode successfully applied
FSP_ERR_ASSERTION	Null Pointer
FSP_ERR_NOT_OPEN	LPM instance is not open
FSP_ERR_UNSUPPORTED	This MCU does not support Deep Software Standby
FSP_ERR_INVALID_ARGUMENT	One of the following: • Invalid snooze entry source • Invalid snooze end sources
FSP_ERR_INVALID_MODE	One of the following: • Invalid low power mode • Invalid DTC option for snooze mode • Invalid deep standby end sources • Invalid deep standby end sources edges • Invalid power supply option for deep standby • Invalid IO port option for deep standby • Invalid output port state setting for standby or deep standby • Invalid sources for wake from standby mode • Invalid power supply option for standby • Invalid IO port option for standby • Invalid standby end sources • Invalid standby end sources edges

◆ R_LPM_LowPowerModeEnter()

fsp_err_t R_LPM_LowPowerModeEnter (lpm_ctrl_t *const *p_api_ctrl*)

Enter low power mode (sleep/deep sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

FSP_SUCCESS	Successful.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	LPM instance is not open
FSP_ERR_INVALID_MODE	One of the following: • HOCO was not system clock when using snooze mode with SCI0/RXD0. • HOCO was not stable when using snooze mode with SCI0/RXD0. • MOCO was running when using snooze mode with SCI0/RXD0. • MAIN OSCILLATOR was running when using snooze mode with SCI0/RXD0. • PLL was running when using snooze mode with SCI0/RXD0. • Unable to disable ocillator stop detect when using standby or deep standby.

◆ R_LPM_IoKeepClear()

fsp_err_t R_LPM_IoKeepClear (lpm_ctrl_t *const *p_api_ctrl*)

Clear the IOKEEP bit after deep software standby

Return values

FSP_SUCCESS	DPSBYCR_b.IOKEEP bit cleared Successfully.
FSP_ERR_UNSUPPORTED	Deep standby mode not supported on this MCU.

5.2.14 RTOS

Modules

Detailed Description

RTOS Modules.

Modules

	FreeRTOS Port (rm_freertos_port) FreeRTOS port for RX MCUs.
	Azure RTOS ThreadX Port (rm_threadx_port) ThreadX port for RX MCUs.

5.2.14.1 FreeRTOS Port (rm_freertos_port)

Modules » RTOS

FreeRTOS port for RX MCUs.

Overview

Note

The FreeRTOS Port does not provide any interfaces to the user. Consult the FreeRTOS documentation at https://www.freertos.org/Documentation for further information.

Features

The RX FreeRTOS port supports the following features:

- Standard FreeRTOS configurations

Configuration

Note

The FreeRTOS Port and libraries use printf by default when logging is enabled. printf requires a heap (BSP Tab -> Properties -> RX Common -> Heap size (bytes)).

Build Time Configurations for all

The following build time configurations are defined in aws/FreeRTOSConfig.h:

Configuration	Options	Default	Description
General > Custom FreeRTOSConfig.h	Manual Entry		Add a path to your custom FreeRTOSConfig.h file. It can be used to

			override some or all of the configurations defined here, and to define additional configurations.
General > Use Preemption	• Enabled • Disabled	Enabled	Set to Enabled to use the preemptive RTOS scheduler, or Disabled to use the cooperative RTOS scheduler.
General > Use Port Optimised Task Selection	• Enabled • Disabled	Disabled	Some FreeRTOS ports have two methods of selecting the next task to execute - a generic method, and a method that is specific to that port. The Generic method: Is used when Use Port Optimized Task Selection is set to 0, or when a port specific method is not implemented. Can be used with all FreeRTOS ports. Is completely written in C, making it less efficient than a port specific method. Does not impose a limit on the maximum number of available priorities. A port specific method: Is not available for all ports. Is used when Use Port Optimized Task Selection is Enabled. Relies on one or more architecture specific assembly instructions (typically a Count Leading Zeros [CLZ] or equivalent instruction) so can only be used with the architecture for which it was specifically written. Is more efficient than the generic method.

			Typically imposes a limit of 32 on the maximum number of available priorities.
General > Use Tickless Idle	• Enabled • Disabled	Disabled	Set Use Tickless Idle to Enabled to use the low power tickless mode, or Disabled to keep the tick interrupt running at all times. Low power tickless implementations are not provided for all FreeRTOS ports.
General > Cpu Clock Hz	Manual Entry	SystemCoreClock	Enter the frequency in Hz at which the internal clock that drives the peripheral used to generate the tick interrupt will be executing - this is normally the same clock that drives the internal CPU clock. This value is required in order to correctly configure timer peripherals.
General > Tick Rate Hz	Value must be greater than 0	1000	The frequency of the RTOS tick interrupt. The tick interrupt is used to measure time. Therefore a higher tick frequency means time can be measured to a higher resolution. However, a high tick frequency also means that the RTOS kernel will use more CPU time so be less efficient. The RTOS demo applications all use a tick rate of 1000Hz. This is used to test the RTOS kernel and is higher than would normally be required. More than one task can share the same priority. The RTOS scheduler will share

			processor time between tasks of the same priority by switching between the tasks during each RTOS tick. A high tick rate frequency will therefore also have the effect of reducing the 'time slice' given to each task.
General > Max Priorities	Must be an integer and greater than 0	7	The number of priorities available to the application tasks. Any number of tasks can share the same priority. Each available priority consumes RAM within the RTOS kernel so this value should not be set any higher than actually required by your application.
General > Minimal Stack Size	Must be an integer and greater than 0	512	The size of the stack used by the idle task. Generally this should not be reduced from the value set in the FreeRTOSConfig.h file provided with the demo application for the port you are using. Like the stack size parameter to the xTaskCreate() and xTaskCreateStatic() functions, the stack size is specified in words, not bytes. If each item placed on the stack is 32-bits, then a stack size of 100 means 400 bytes (each 32-bit stack item consuming 4 bytes).
General > Max Task Name Len	Must be an integer and greater than 0	16	The maximum permissible length of the descriptive name given to a task when the task is created. The length is specified in the number of

			characters including the NULL termination byte.
General > Use 16-bit Ticks	Disabled	Disabled	Time is measured in 'ticks' - which is the number of times the tick interrupt has executed since the RTOS kernel was started. The tick count is held in a variable of type TickType_t. Defining configUSE_16_BIT_TICKS as 1 causes TickType_t to be defined (typedef'ed) as an unsigned 16bit type. Defining configUSE_16_BIT_TICKS as 0 causes TickType_t to be defined (typedef'ed) as an unsigned 32bit type. Using a 16-bit type will greatly improve performance on 8- and 16-bit architectures, but limits the maximum specifiable time period to 65535 'ticks'. Therefore, assuming a tick frequency of 250Hz, the maximum time a task can delay or block when a 16bit counter is used is 262 seconds, compared to 17179869 seconds when using a 32-bit counter.
General > Idle Should Yield	• Enabled • Disabled	Enabled	This parameter controls the behaviour of tasks at the idle priority. It only has an effect if: The preemptive scheduler is being used. The application creates tasks that run at the idle priority. If Use Time Slicing is

Enabled then tasks that share the same priority will time slice. If none of the tasks get preempted then it might be assumed that each task at a given priority will be allocated an equal amount of processing time - and if the priority is above the idle priority then this is indeed the case.
When tasks share the idle priority the behaviour can be slightly different. If Idle Should Yield is Enabled then the idle task will yield immediately if any other task at the idle priority is ready to run. This ensures the minimum amount of time is spent in the idle task when application tasks are available for scheduling. This behaviour can however have undesirable effects (depending on the needs of your application) as depicted below:

The diagram above shows the execution pattern of four tasks that are all running at the idle priority. Tasks A, B and C are application tasks. Task I is the idle task. A context switch occurs with regular period at times T0, T1, ..., T6. When the idle task yields task A starts to execute - but the idle task has already consumed some of the current time slice. This results in task I and

			task A effectively sharing the same time slice. The application tasks B and C therefore get more processing time than the application task A. This situation can be avoided by: If appropriate, using an idle hook in place of separate tasks at the idle priority. Creating all application tasks at a priority greater than the idle priority. Setting Idle Should Yield to Disabled. Setting Idle Should Yield to Disabled prevents the idle task from yielding processing time until the end of its time slice. This ensure all tasks at the idle priority are allocated an equal amount of processing time (if none of the tasks get pre-empted) - but at the cost of a greater proportion of the total processing time being allocated to the idle task.
General > Use Task Notifications	• Enabled • Disabled	Enabled	Setting Use Task Notifications to Enabled will include direct to task notification functionality and its associated API in the build. Setting Use Task Notifications to Disabled will exclude direct to task notification functionality and its associated API from the build.

			Each task consumes 8 additional bytes of RAM when direct to task notifications are included in the build.
General > Task Notification Array Entries	Must be an integer and greater than 0	1	Set the Task Notifications Array Entries to enter the size of array needed
General > Use Mutexes	• Enabled • Disabled	Disabled	Set to Enabled to include mutex functionality in the build, or Disabled to omit mutex functionality from the build. Readers should familiarise themselves with the differences between mutexes and binary semaphores in relation to the FreeRTOS functionality.
General > Use Recursive Mutexes	• Enabled • Disabled	Disabled	Set to Enabled to include recursive mutex functionality in the build, or Disabled to omit recursive mutex functionality from the build.
General > Use Counting Semaphores	• Enabled • Disabled	Enabled	Set to Enabled to include counting semaphore functionality in the build, or Disabled to omit counting semaphore functionality from the build.
General > Queue Registry Size	Value must be positive integer greater than or equal to 0	10	The queue registry has two purposes, both of which are associated with RTOS kernel aware debugging: It allows a textual name to be associated with a queue for easy queue identification within a debugging GUI. It contains the information required by

			a debugger to locate each registered queue and semaphore. The queue registry has no purpose unless you are using a RTOS kernel aware debugger. Registry Size defines the maximum number of queues and semaphores that can be registered. Only those queues and semaphores that you want to view using a RTOS kernel aware debugger need be registered. See the API reference documentation for vQueueAddToRegistry() and vQueueUnregisterQueue() for more information.
General > Use Queue Sets	• Enabled • Disabled	Enabled	Set to Enabled to include queue set functionality (the ability to block, or pend, on multiple queues and semaphores), or Disabled to omit queue set functionality.
General > Use Time Slicing	• Enabled • Disabled	Disabled	If Use Time Slicing is Enabled, FreeRTOS uses prioritised preemptive scheduling with time slicing. That means the RTOS scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on every RTOS tick interrupt. If Use Time Slicing is Disabled then the RTOS scheduler will still run the highest priority task that is in the Ready state, but will not switch between

			tasks of equal priority just because a tick interrupt has occurred.
General > Use Newlib Reentrant	• Enabled • Disabled	Disabled	If Use Newlib Reentrant is Enabled then a newlib reent structure will be allocated for each created task. Note Newlib support has been included by popular demand, but is not used by the FreeRTOS maintainers themselves. FreeRTOS is not responsible for resulting newlib operation. User must be familiar with newlib and must provide system-wide implementations of the necessary stubs. Be warned that (at the time of writing) the current newlib design implements a system-wide malloc() that must be provided with locks.
General > Enable Backward Compatibility	• Enabled • Disabled	Enabled	The FreeRTOS.h header file includes a set of #define macros that map the names of data types used in versions of FreeRTOS prior to version 8.0.0 to the names used in FreeRTOS version 8.0.0. The macros allow application code to update the version of FreeRTOS they are built against from a pre 8.0.0 version to a post 8.0.0 version without modification. Setting Enable Backward Compatibility to Disabled in FreeRTOSConfig.h excludes the macros from the build, and in so doing allowing validation that no pre version 8.0.0 names

			are being used.
General > Num Thread Local Storage Pointers	Value must be positive integer greater than or equal to 0	3	Sets the number of indexes in each task's thread local storage array.
General > Stack Depth Type	Manual Entry	uint32_t	Sets the type used to specify the stack depth in calls to xTaskCreate(), and various other places stack sizes are used (for example, when returning the stack high water mark). Older versions of FreeRTOS specified stack sizes using variables of type UBaseType_t, but that was found to be too restrictive on 8-bit microcontrollers. Stack Depth Type removes that restriction by enabling application developers to specify the type to use.
General > Message Buffer Length Type	Manual Entry	size_t	FreeRTOS Message buffers use variables of type Message Buffer Length Type to store the length of each message. If Message Buffer Length Type is not defined then it will default to size_t. If the messages stored in a message buffer will never be larger than 255 bytes then defining Message Buffer Length Type to uint8_t will save 3 bytes per message on a 32-bit microcontroller. Likewise if the messages stored in a message buffer will never be larger than 65535 bytes then defining Message Buffer Length Type to uint16_t will save 2

		bytes per message on a 32-bit microcontroller.
General > Library Max Syscall Interrupt Priority	MCU Specific Options	The highest interrupt priority that can be used by any interrupt service routine that makes calls to interrupt safe FreeRTOS API functions. DO NOT CALL INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER PRIORITY THAN THIS! Below is explanation for macros that are set based on this value from FreeRTOS website. Note in the following discussion that only API functions that end in "FromISR" can be called from within an interrupt service routine. configMAX_SYSCALL_INTERRUPT_PRIORITY sets the highest interrupt priority from which interrupt safe FreeRTOS API functions can be called. A full interrupt nesting model is achieved by setting configMAX_SYSCALL_INTERRUPT_PRIORITY above (that is, at a higher priority level) than configKERNEL_INTERRUPT_PRIORITY. This means the FreeRTOS kernel does not completely disable interrupts, even inside critical sections. Further, this is achieved without the

			disadvantages of a segmented kernel architecture. Interrupts that do not call API functions can execute at priorities above configMAX_SYSCALL_INTERRUPT_PRIORITY and therefore never be delayed by the RTOS kernel execution.
General > Assert	Manual Entry	assert(x)	The semantics of the configASSERT() macro are the same as the standard C assert() macro. An assertion is triggered if the parameter passed into configASSERT() is zero. configASSERT() is called throughout the FreeRTOS source files to check how the application is using FreeRTOS. It is highly recommended to develop FreeRTOS applications with configASSERT() defined. The example definition (shown at the top of the file and replicated below) calls vAssertCalled(), passing in the file name and line number of the triggering configASSERT() call (__FILE__ and __LINE__ are standard macros provided by most compilers). This is just for demonstration as vAssertCalled() is not a FreeRTOS function, configASSERT() can be defined to take whatever action the application writer deems appropriate.

It is normal to define configASSERT() in such a way that it will prevent the application from executing any further. This if for two reasons; stopping the application at the point of the assertion allows the cause of the assertion to be debugged, and executing past a triggered assertion will probably result in a crash anyway.

Note defining configASSERT() will increase both the application code size and execution time. When the application is stable the additional overhead can be removed by simply commenting out the configASSERT() definition in FreeRTOSConfig.h.

/* Define configASSERT() to call vAssertCalled() if the assertion fails. The assertion
has failed if the value of the parameter passed into configASSERT() equals zero. */
#define configASSERT((x)) if((x) == 0) vAssertCalled(__FILE__, __LINE__)
If running FreeRTOS under the control of a debugger, then configASSERT() can be defined to just disable interrupts and sit in a loop, as demonstrated below. That will have the effect of stopping the code on the line

			that failed the assert test - pausing the debugger will then immediately take you to the offending line so you can see why it failed. /* Define configASSERT() to disable interrupts and sit in a loop. */ #define configASSERT((x)) if((x) == 0) { taskDISABLE_INTERRUPTS(); for(;;); }
General > Include Application Defined Privileged Functions	• Enabled • Disabled	Disabled	Include Application Defined Privileged Functions is only used by FreeRTOS MPU. If Include Application Defined Privileged Functions is Enabled then the application writer must provide a header file called "application_defined_privileged_functions.h", in which functions the application writer needs to execute in privileged mode can be implemented. Note that, despite having a .h extension, the header file should contain the implementation of the C functions, not just the functions' prototypes. Functions implemented in "application_defined_privileged_functions.h" must save and restore the processor's privilege state using the prvRaisePrivilege() function and portRESET_PRIVILEGE() macro respectively. For example, if a library provided print function

			accesses RAM that is outside of the control of the application writer, and therefore cannot be allocated to a memory protected user mode task, then the print function can be encapsulated in a privileged function using the following code:
			
			void MPU_debug_printf(const char *pcMessage)
			{
			/* State the privilege level of the processor when the function was called. */
			BaseType_t xRunningPrivileged = prvRaisePrivilege();
			
			/* Call the library function, which now has access to all RAM. */
			debug_printf(pcMessage);
			
			/* Reset the processor privilege level to its original value. */
			portRESET_PRIVILEGE(xRunningPrivileged);
			}
			This technique should only be use during development, and not deployment, as it circumvents the memory protection.

General > List Member Volatile	Value must be volatile	volatile	Defining volatile can prevent loop optimizations
Hooks > Use Idle Hook	• Enabled • Disabled	Disabled	Set to Enabled if you wish to use an idle hook, or Disabled to omit an idle hook. If enabled, hook function

			vApplicationIdleHook() should be defined.
Hooks > Use Malloc Failed Hook	• Enabled • Disabled	Disabled	The kernel uses a call to pvPortMalloc() to allocate memory from the heap each time a task, queue or semaphore is created. The official FreeRTOS download includes four sample memory allocation schemes for this purpose. The schemes are implemented in the heap_1.c, heap_2.c, heap_3.c, heap_4.c and heap_5.c source files respectively. Use Malloc Failed Hook is only relevant when one of these three sample schemes is being used. The malloc() failed hook function is a hook (or callback) function that, if defined and configured, will be called if pvPortMalloc() ever returns NULL. NULL will be returned only if there is insufficient FreeRTOS heap memory remaining for the requested allocation to succeed. If Use Malloc Failed Hook is Enabled then the application must define a malloc() failed hook function. If Use Malloc Failed Hook is set to Dosab;ed then the malloc() failed hook function will not be called, even if one is defined. Malloc() failed hook functions must have the name and prototype shown below.

			void vApplicationMallocFailedHook(void);
Hooks > Use Daemon Task Startup Hook	• Enabled • Disabled	Disabled	If Use Timers and Use Daemon Task Startup Hook are both Enabled then the application must define a hook function that has the exact name and prototype as shown below. The hook function will be called exactly once when the RTOS daemon task (also known as the timer service task) executes for the first time. Any application initialisation code that needs the RTOS to be running can be placed in the hook function. void void vApplicationDaemonTaskStartupHook(void);
Hooks > Use Tick Hook	• Enabled • Disabled	Disabled	Set to Enabled if you wish to use an tick hook, or Disabled to omit an tick hook.
Hooks > Check For Stack Overflow	• Enabled • Disabled	Disabled	The stack overflow detection page describes the use of this parameter. This is not recommended for RX MCUs with hardware stack monitor support. RX MCU designs should enable the RX hardware stack monitor instead.
Stats > Use Trace Facility	• Enabled • Disabled	Disabled	Set to Enabled if you wish to include additional structure members and functions to assist with execution visualisation and tracing.
Stats > Use Stats Formatting Functions	• Enabled • Disabled	Disabled	Set Use Trace Facility and Use Stats Formatting Functions to Enabled to include the vTaskList() and vTa

			skGetRunTimeStats() functions in the build. Setting either to Disabled will omit vTaskList() and vTaskGetRunTimeStates() from the build.
Stats > Generate Run Time Stats	• Enabled • Disabled	Disabled	The Run Time Stats page describes the use of this parameter.
Memory Allocation > Clear Memory on Free	• Enabled • Disabled	Disabled	If set to 1, then blocks of memory allocated using pvPortMalloc() will be cleared when freed using vPortFree()
Memory Allocation > Support Static Allocation	• Enabled • Disabled	Enabled	If Support Static Allocation is Enabled then RTOS objects can be created using RAM provided by the application writer. If Support Static Allocation is Disabled then RTOS objects can only be created using RAM allocated from the FreeRTOS heap. If Support Static Allocation is left undefined it will default to 0. If Support Static Allocation is Enabled then the application writer must also provide two callback functions: vApplicationGetIdleTaskMemory() to provide the memory for use by the RTOS Idle task, and (if Use Timers is Enabled) vApplicationGetTimerTaskMemory() to provide memory for use by the RTOS Daemon/Timer Service task. Examples are provided below. /* Support Static

Allocation is Enabled,
so the application must
provide an
implementation of vAp
plicationGetIdleTaskMe
mory() to provide the
memory that is
used by the Idle task. */
void vApplicationGetIdl
eTaskMemory(
StaticTask_t **ppxIdleT
askTCBBuffer,
StackType_t **ppxIdleT
askStackBuffer,
uint32_t
*pulIdleTaskStackSize)
{
/* If the buffers to be
provided to the Idle
task are declared
inside this
function then they
must be declared static
- otherwise they will be
allocated on
the stack and so not
exists after this
function exits. */
static StaticTask_t
xIdleTaskTCB;
static StackType_t
uxIdleTaskStack[config
MINIMAL_STACK_SIZE];

/* Pass out a pointer to
the StaticTask_t
structure in which the
Idle task's
state will be stored. */
*ppxIdleTaskTCBBuffer
=

/* Pass out the array
that will be used as the
Idle task's stack. */
*ppxIdleTaskStackBuffe
r = uxIdleTaskStack;

/* Pass out the size of
the array pointed to by
*ppxIdleTaskStackBuffe
r.
Note that, as the array
is necessarily of type
StackType_t,

```
configMINIMAL_STACK_
SIZE is specified in
words, not bytes. */
*pulIdleTaskStackSize
= configMINIMAL_STAC
K_SIZE;
}
/*-----------------------------
-----------------------------*/

/* Support Static
Allocation and Use
Timers are both
Enabled, so the
application must
provide an
implementation of vAp
plicationGetTimerTask
Memory()
to provide the memory
that is used by the
Timer service task. */
void vApplicationGetTi
merTaskMemory(
StaticTask_t **ppxTime
rTaskTCBBuffer,<br>
StackType_t **ppxTime
rTaskStackBuffer,<br>
uint32_t
*pulTimerTaskStackSiz
e )
{
/* If the buffers to be
provided to the Timer
task are declared
inside this
function then they
must be declared static
- otherwise they will be
allocated on
the stack and so not
exists after this
function exits. */
static StaticTask_t
xTimerTaskTCB;
static StackType_t
uxTimerTaskStack[ con
figTIMER_TASK_STACK_
DEPTH ];

/* Pass out a pointer to
the StaticTask_t
structure in which the
Timer
task's state will be
```

			stored. */
			*ppxTimerTaskTCBBuffer =
			
			/* Pass out the array that will be used as the Timer task's stack. */
			*ppxTimerTaskStackBuffer = uxTimerTaskStack;
			
			/* Pass out the size of the array pointed to by *ppxTimerTaskStackBuffer.
			Note that, as the array is necessarily of type StackType_t, configTIMER_TASK_STACK_DEPTH is specified in words, not bytes. */
			*pulTimerTaskStackSize = configTIMER_TASK_STACK_DEPTH;
			}
			
			Examples of the callback functions that must be provided by the application to supply the RAM used by the Idle and Timer Service tasks if Support Static Allocation is Enabled.
			
			See the Static Vs Dynamic Memory Allocation page for more information.

Memory Allocation > Support Dynamic Allocation	• Enabled • Disabled	Enabled	If Support Dynamic Allocation is Enabled then RTOS objects can be created using RAM that is automatically allocated from the FreeRTOS heap. If Support Dynamic Allocation is set to 0 then RTOS objects can only be created using RAM provided by the application writer.

			See the Static Vs Dynamic Memory Allocation page for more information.
Memory Allocation > Total Heap Size	Value must be positive integer greater than or equal to 0	0	The total amount of RAM available in the FreeRTOS heap. This value will only be used if Support Dynamic Allocation is Enabled and the application makes use of one of the sample memory allocation schemes provided in the FreeRTOS source code download. See the memory configuration section for further details.
Memory Allocation > Application Allocated Heap	• Enabled • Disabled	Disabled	By default the FreeRTOS heap is declared by FreeRTOS and placed in memory by the linker. Setting Application Allocated Heap to Enabled allows the heap to instead be declared by the application writer, which allows the application writer to place the heap wherever they like in memory. If heap_1.c, heap_2.c or heap_4.c is used, and Application Allocated Heap is Enabled, then the application writer must provide a uint8_t array with the exact name and dimension as shown below. The array will be used as the FreeRTOS heap. How the array is placed at a specific memory location is dependent on the compiler being used - refer to your compiler's documentation.

			uint8_t ucHeap[configTOTAL_HEAP_SIZE];
Timers > Use Timers	• Enabled • Disabled	Enabled	Set to Enabled to include software timer functionality, or Disabled to omit software timer functionality. See the FreeRTOS software timers page for a full description.
Timers > Timer Task Priority	Value must be positive integer greater than or equal to 0	6	Sets the priority of the software timer service/daemon task. See the FreeRTOS software timers page for a full description.
Timers > Timer Queue Length	Value must be a non-negative integer	5	Sets the length of the software timer command queue. See the FreeRTOS software timers page for a full description.
Timers > Timer Task Stack Depth	Value must be a non-negative integer	configMINIMAL_STACK_SIZE * 6	Sets the stack depth allocated to the software timer service/daemon task. See the FreeRTOS software timers page for a full description.
Optional Functions > vTaskPrioritySet() Function	• Enabled • Disabled	Enabled	Include vTaskPrioritySet() function in build
Optional Functions > uxTaskPriorityGet() Function	• Enabled • Disabled	Enabled	Include uxTaskPriorityGet() function in build
Optional Functions > vTaskDelete() Function	• Enabled • Disabled	Enabled	Include vTaskDelete() function in build
Optional Functions > vTaskSuspend() Function	• Enabled • Disabled	Enabled	Include vTaskSuspend() function in build
Optional Functions > xResumeFromISR() Function	• Enabled • Disabled	Enabled	Include xResumeFromISR() function in build
Optional Functions > vTaskDelayUntil() Function	• Enabled • Disabled	Enabled	Include vTaskDelayUntil() function in build

Optional Functions > vTaskDelay() Function	• Enabled • Disabled	Enabled	Include vTaskDelay() function in build
Optional Functions > xTaskGetSchedulerState() Function	• Enabled • Disabled	Enabled	Include xTaskGetSchedulerState() function in build
Optional Functions > xTaskGetCurrentTaskHandle() Function	• Enabled • Disabled	Enabled	Include xTaskGetCurrentTaskHandle() function in build
Optional Functions > uxTaskGetStackHighWaterMark() Function	• Enabled • Disabled	Disabled	Include uxTaskGetStackHighWaterMark() function in build
Optional Functions > xTaskGetIdleTaskHandle() Function	• Enabled • Disabled	Disabled	Include xTaskGetIdleTaskHandle() function in build
Optional Functions > eTaskGetState() Function	• Enabled • Disabled	Disabled	Include eTaskGetState() function in build
Optional Functions > xEventGroupSetBitFromISR() Function	• Enabled • Disabled	Enabled	Include xEventGroupSetBitFromISR() function in build
Optional Functions > xTimerPendFunctionCall() Function	• Enabled • Disabled	Disabled	Include xTimerPendFunctionCall() function in build
Optional Functions > xTaskAbortDelay() Function	• Enabled • Disabled	Disabled	Include xTaskAbortDelay() function in build
Optional Functions > xTaskGetHandle() Function	• Enabled • Disabled	Disabled	Include xTaskGetHandle() function in build
Optional Functions > xTaskResumeFromISR() Function	• Enabled • Disabled	Enabled	Include xTaskResumeFromISR() function in build
Logging > Print String Function	Manual Entry	vLoggingPrint(x)	
Logging > Logging Max Message Length	Manual Entry	192	
Logging > Logging Include Time and Task Name	• Disabled • Enabled	Enabled	
Logging > Debug Logging Name	Manual Entry	Log Name	
Logging > Logging Level	• LOG_NONE • LOG_ERROR • LOG_WARN • LOG_INFO	LOG_NONE	Set the logging level

- LOG_DEBUG

Pin Configuration

This module does not use I/O pins.

Clock Configuration

The FreeRTOS port uses timer module as the system clock. Users select Timer Interface on stack configuration.

Usage Notes

Timer

Users need to match the periods of FreeRTOS and timer module.

The period of the FreeRTOS is set by the FreeRTOS component under General > Tick Rate Hz. It is calculated by (1000 / (General > Tick Rate Hz)) Milliseconds.

The period of the timer module is set by the timer component under General > Period and General > Period Unit. It is calculated by peripheral clock. For details, please refer to the usage notes of timer module.

Examples

System Timer Example

This is an example of using the cmt timer as system timer. vPortSetupTimerInterrupt function initializes timer and vPortTeardownTimerInterrupt function completes it. If users do not use Timer Interface, need to implement those functions. The actual period of the timer module is displayed in generated codes.

```
/* Vector table allocations */
#define VECTOR_NUMBER_CMT0_CMI ((IRQn_Type) 28) /* ID:28 CMT0 CMI (Compare match) */
cmt_instance_ctrl_t g_timer0_ctrl;
const timer_cfg_t g_timer0_cfg =
{
    .mode                                                = TIMER_MODE_PERIODIC,
 /* Actual period: 0.001 seconds. */ .period_counts = (uint32_t) 0x1d4c,
    .source_div    = (timer_source_div_t) 3,
    .channel       = 0,
    .p_callback    = rm_freertos_port_systick_callback,
    .p_context    = NULL,
    .p_extend      = NULL,
    .cycle_end_ipl = (10),
```

```
#if defined(VECTOR_NUMBER_CMT0_CMI)
    .cycle_end_irq                                      = VECTOR_NUMBER_CMT0_CMI,
#else
    .cycle_end_irq                                      = FSP_INVALID_VECTOR,
#endif
};
const timer_instance_t g_timer0 = {.p_ctrl = &g_timer0_ctrl, .p_cfg = &g_timer0_cfg,
.p_api = &g_timer_on_cmt};
/* Setup the system timer to generate the tick interrupts at the required frequency.
*/
void vPortSetupTimerInterrupt (void)
{
    g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
    g_timer0.p_api->callbackSet(g_timer0.p_ctrl, rm_freertos_port_systick_callback,
NULL, NULL);
    g_timer0.p_api->start(g_timer0.p_ctrl);
}
/*Teardown the system timer to terminate scheduler. */
void vPortTeardownTimerInterrupt (void)
{
    g_timer0.p_api->close(g_timer0.p_ctrl);
}
```

5.2.14.2 Azure RTOS ThreadX Port (rm_threadx_port)

Modules » RTOS

ThreadX port for RX MCUs.

Overview

Note

The ThreadX Port does not provide any interfaces to the user. Consult the ThreadX documentation at https://docs.microsoft.com/en-us/azure/rtos/threadx/ for further information.

Features

The RX ThreadX port supports the following features:

- Standard ThreadX configurations

Configuration

Build Time Configurations for ThreadX

The following build time configurations are defined in fsp_cfg/azure/tx/tx_user.h:

Configuration	Options	Default	Description
General > Custom tx_user.h	Manual Entry		Add a path to your custom tx_user.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
General > Error Checking	• Enabled • Disabled	Enabled	The ThreadX basic API error checking can be bypassed by compiling with the symbol TX_DISABLE_ERROR_CHECKING defined.
General > Max Priorities	Value must be a multiple of 32 and in range 32 to 1024 or empty	32	Define the priority levels for ThreadX. Legal values range from 32 to 1024 and MUST be evenly divisible by 32.
General > Minimum Stack	Value must be greater than 0 or empty	200	Define the minimum stack for a ThreadX thread on this processor. If the size supplied during thread creation is less than this value, the thread create call will return an error.
General > Stack Filling	• Enabled • Disabled	Enabled	Determine is stack filling is enabled. By default, ThreadX stack filling is enabled, which places an 0xEF pattern in each byte of each thread's stack. This is used by debuggers with ThreadX-awareness and by the ThreadX run-time stack

			checking feature.
General > Preemption Threshold	• Enabled • Disabled	Disabled	Determine if preemption-threshold should be disabled. By default, preemption-threshold is disabled. If the application does not use preemption-threshold, it may be disabled to reduce code size and improve performance.
General > Notify Callbacks	• Enabled • Disabled	Disabled	Determine if the notify callback option should be disabled. By default, notify callbacks are disabled. If the application does not use notify callbacks, they may be disabled to reduce code size and improve performance.
General > Inline Thread Resume Suspend	• Enabled • Disabled	Disabled	Determine if the tx_thread_resume and tx_thread_suspend services should have their internal code in-line. This results in a larger image, but improves the performance of the thread resume and suspend services.
General > Not Interruptable	• Enabled • Disabled	Disabled	Determine if the internal ThreadX code is non-interruptable. This results in smaller code size and less processing overhead, but increases the interrupt lockout time.
General > IAR Library Support	• Enabled • Disabled	Disabled	Enable IAR library support (IAR compiler only). When IAR Library Support is Enabled, enable the linker option --threaded_lib. In the IAR IDE, this can be enabled in Project > Options > General Options > Library

			Configuration > Enable thread support in library.
General > BSD Support	• Enabled • Disabled	Disabled	Defines TX_THREAD_EXTENSION_1 to bsd_err_no in order to support NXD BSD.
General > FileX Pointer	• Enabled • Disabled	Enabled	Determine if there is a FileX pointer in the thread control block. By default, the pointer is there for legacy/backwards compatibility. The pointer must also be there for applications using FileX. Disable this to save space in the thread control block.
Timer > Timer Ticks Per Second	Value must be greater than 0 or empty	100	Define the number of times the system timer runs per second. Default is 100 ticks per second, which results in a tick every 10ms.
Timer > Timer Thread Stack Size	Value must be greater than 0 or empty	1024	Define the system timer thread's default stack size and priority. These are only applicable if TX_TIMER_PROCESS_IN_ISR is disabled.
Timer > Timer Thread Priority	Value must be greater than 0 or empty	0	Define the system timer thread's default stack size and priority. These are only applicable if TX_TIMER_PROCESS_IN_ISR is disabled.
Timer > Timer Process In ISR	• Enabled • Disabled	Enabled	Determine if timer expirations (application timers, timeouts, and tx_thread_sleep calls should be processed within the a system timer thread or directly in the timer ISR. When disabled, the timer thread is used. When enabled, timer expiration processing is

			done directly from the timer ISR, thereby eliminating the timer thread control block, stack, and context switching to activate it.
Timer > Reactivate Inline	• Enabled • Disabled	Disabled	Determine if in-line timer reactivation should be used within the timer expiration processing. By default, this is disabled and a function call is used. When enabled, reactivating is performed in-line resulting in faster timer processing but slightly larger code size.
Timer > Timer	• Enabled • Disabled	Enabled	Determine if no timer processing is required. This option will help eliminate the timer processing when not needed.
Trace > Event Trace	• Enabled • Disabled	Disabled	Determine if the trace event logging code should be enabled. This causes slight increases in code size and overhead, but provides the ability to generate system trace information which is available for viewing in TraceX.
Trace > Trace Buffer Name	Name must be a valid C symbol	g_tx_trace_buffer	Name of trace buffer symbol, only used if Event Trace is enabled.
Trace > Memory section for Trace Buffer	Manual Entry	.bss	Specify the memory section where the Trace Buffer will be allocated, only used if Event Trace is enabled. To view TraceX data, export this buffer as raw binary data to a file (.trx extension recommended) and open it with Microsoft Azure RTOS TraceX.

Trace > Trace Buffer Size	Value must be greater than 0	65536	Trace buffer size in bytes, only used if Event Trace is enabled
Trace > Trace Buffer Number of Registries	Value must be greater than 0	30	Number of registries available to TraceX, only used if Event Trace is enabled
Performance > Block Pool Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers block pool performance information.
Performance > Byte Pool Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers byte pool performance information.
Performance > Event Flags Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers event flags performance information.
Performance > Mutex Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers mutex performance information.
Performance > Queue Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers queue performance information.
Performance > Semaphore Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers semaphore performance information.
Performance > Thread Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers thread performance information.
Performance > Timer Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers timer performance information.
Interrupts > Maximum Interrupt Priority Level	Value must be an integer between 1 and 31 or empty	15	The maximum value of the PSW IPL (Interrupt Priority Level) field used by ThreadX for interrupt masking. The PSW IPL field width differs between MCUs: 4-bit (0-15) on RX72N/RX26T; 5-bit

(0-31) on RX74M/RX74N. For RX74M/RX74N, Priority level 15 is the highest level in non-secure world (non-secure, highest), and levels 16-31 are available in root world or secure world only. The default value 15 allows all non-secure world interrupts to use ThreadX services on all supported MCUs.

Pin Configuration

This module does not use I/O pins.

Clock Configuration

The ThreadX port uses timer module as the system clock. Users select Timer Interface on stack configuration.

Usage Notes

Interrupt Priorities

When no threads are ready to run, the ThreadX port spins in the SWINT, which is fixed at the lowest interrupt priority (the priority level is 1, since priority level 0 disables interrupts). The MCU does not service any other interrupts of the lowest priority while no threads are ready to run.

To get around this limitation, when the ready thread pointer (_tx_thread_execute_ptr) is NULL, the RXv3 ThreadX scheduler executes the WAIT instruction in its scheduling loop if BSP_CFG_RTOS_IDLE_SLEEP is defined and set to 1. This causes the CPU to wait until an interrupt occurs. Any interrupt that is enabled and configured with a priority higher than SWINT can wake the CPU, make a thread ready, and allow the scheduler to resume normal thread execution after the interrupt service routine completes. If BSP_CFG_RTOS_IDLE_SLEEP is not enabled, the scheduler loops without entering the idle wait state. In either case, the application can create an idle thread that is always ready to run to avoid this limitation.

Warning

Do not attempt to wake a thread from an interrupt with the lowest available interrupt priority unless the application has created an idle thread.

Timer

Users need to match the periods of ThreadX and timer module. A mismatch results in incorrect system tick timing and may affect thread scheduling.

The period of the ThreadX is set by the ThreadX component under Timer > Timer Ticks Per Second.

The period of the timer module is set by the timer component under General > Period and General > Period Unit. It is calculated by peripheral clock. For details, please refer to the usage notes of timer

module.

SmartConfigurator validates that the timer period matches the ThreadX tick period at configuration time. If the periods do not match, the following error is displayed and code generation is blocked:

> Period of the timer module must be set to match the value set in Common->Timer->Timer Ticks Per Second

Note that Raw Counts cannot be used as the timer period unit when ThreadX timer tick synchronization is required. Selecting Raw Counts results in the following error:

> Raw Counts cannot be used when ThreadX timer tick synchronization is required

Stack Alignment

The thread stack size must be an integer multiple of 4 bytes. RXv3 does not use the ARM Architecture Procedure Call Standard (ARM AAPCS), so 4-byte alignment is sufficient. This differs from ARM MCUs (e.g., RA series), which require 8-byte alignment per ARM AAPCS.

Examples

System Timer Example

This is an example of using the cmt timer as system timer. _tx_timer_module_setup function initializes timer. _tx_timer_module_counter_get function gets a current counter value. If users do not use Timer Interface, need to implement those functions. The actual period of the timer module is displayed in generated codes.

```
/* Vector table allocations */
#define VECTOR_NUMBER_CMT0_CMI ((IRQn_Type) 28) /* ID:28 CMT0 CMI (Compare match) */
cmt_instance_ctrl_t g_timer0_ctrl;
const timer_cfg_t g_timer0_cfg =
{
    .mode                                          = TIMER_MODE_PERIODIC,
 /* Actual period: 0.001 seconds. */ .period_counts = (uint32_t) 0x1d4c,
    .source_div    = (timer_source_div_t) 3,
    .channel       = 0,
    .p_callback    = rm_threadx_port_systick_callback,
    .p_context    = NULL,
    .p_extend      = NULL,
    .cycle_end_ipl = (10),
```

```
#if defined(VECTOR_NUMBER_CMT0_CMI)
    .cycle_end_irq                                      = VECTOR_NUMBER_CMT0_CMI,
#else
    .cycle_end_irq                                      = FSP_INVALID_VECTOR,
#endif
};
const timer_instance_t g_timer0 = {.p_ctrl = &g_timer0_ctrl, .p_cfg = &g_timer0_cfg,
.p_api = &g_timer_on_cmt};
/* Setup the system timer to generate the tick interrupts at the required frequency.
*/
void _tx_timer_module_setup (void)
{
 R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 R_CMT_CallbackSet(&g_timer0_ctrl, rm_threadx_port_systick_callback, NULL, NULL);
 R_CMT_Start(&g_timer0_ctrl);
}
uint32_t _tx_timer_module_counter_get (void)
{
 timer_status_t status;
 R_CMT_StatusGet(&g_timer0_ctrl, &status);
 return status.counter;
}
```

5.2.15 Security

Modules

Detailed Description

Security Modules.

Modules

Secure Key Injection (r_rsip_key_injection)	Driver for the Secure Key Injection on RX MCUs.

	Renesas Secure IP (r_rsip_protected) Driver for the Renesas Secure IP on RX MPUs. This module implements the RSIP Interface.
	Azure RTOS NetX Crypto HW Acceleration (rm_netx_secure_crypto) Hardware acceleration for the Netx Crypto implementation of the Microsoft Azure RTOS NetX Crypto API.
	Mbed Crypto H/W Acceleration (rm_psa_crypto) Hardware acceleration for the mbedCrypto implementation of the Arm PSA Crypto API.

5.2.15.1 Secure Key Injection (r_rsip_key_injection)

Modules » Security

Functions

fsp_err_t	R_RSIP_AES128_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_AES192_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_AES256_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_AES128_XTS_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_AES256_XTS_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key,

	rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ChaCha20_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_chacha20_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_RSA2048_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa2048_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_RSA3072_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa3072_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_RSA4096_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa4096_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_secp256r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_secp384r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_secp521r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_secp256k1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_brainpoolP256r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const

	*const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_brainpoolP384r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_brainpoolP512r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_ECC_Ed25519_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_SHA224_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_SHA256_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_SHA384_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_SHA512_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_SHA512_224_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_SHA512_256_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const

	p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

Detailed Description

Driver for the Secure Key Injection on RX MCUs.

Overview

This module provides RSIP functions in PSA Crypto.

HW Overview

Crypto Peripheral version	Devices
RSIP-E50D	RX74M/N

Features

The RSIP Key Injection Module has two types of APIs, InitialKeyWrap and EncryptedKeyWrap. The available APIs differ depending on used RSIP. Please refer to the following for details.

Key Support

AES Key Injection

Crypto Peripheral version	128 bit	192 bit	256 bit
RSIP-E50D	Support	Support	Support

AES XTS Key Injection

Crypto Peripheral version	128 bit XTS	256 bit XTS
RSIP-E50D	Support	Support

ChaCha20 Key Injection

Crypto Peripheral version	ChaCha20
RSIP-E50D	Support

RSA Key Injection

Crypto Peripheral version	2048 bit	3072 bit	4096 bit
RSIP-E50D	Support	Support	Support

ECC Key Injection

Crypto Peripheral version	secp256r1	secp384r1	secp521r1	secp256k1	brainpool P256r1	brainpool P384r1	brainpool P512r1	Ed25519
RSIP-E50D	Support	Support	Support	Support	Support	Support	Support	Support

HMAC Key Injection

Crypto Peripheral version	SHA224	SHA256	SHA384	SHA512	SHA512/224	SHA512/256
RSIP-E50D	Support	Support	Support	Support	Support	Support

Configuration

Clock Configuration

This module does not require a specific clock configuration.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Module activation

RSIP Key Umport for PSA Crypto Module is off by default. To use it, please On the Stacks tab, add New > Security > MbedTLS or MbedTLS (Crypto Only), Add Requires RSIP Driver > New > (*1), Add Key Injection for RSA Crypto (Optional) > New > Key Injectin for RSA Crypto for PSA Crypto Module after adding MbedTLS.

Hardware Initialization

Please refer to Hardware Initialization in Mbed Crypto H/W Acceleration (rm_psa_crypto).

Enumerations

enum	rsip_key_type_t

Enumeration Type Documentation

◆ rsip_key_type_t

enum rsip_key_type_t	
Enumerator	
RSIP_KEY_TYPE_INVALID	Invalid key.
RSIP_KEY_TYPE_AES_128	AES-128.

RSIP_KEY_TYPE_AES_192	AES-192.
RSIP_KEY_TYPE_AES_256	AES-256.
RSIP_KEY_TYPE_XTS_AES_128	XTS-AES-128.
RSIP_KEY_TYPE_XTS_AES_256	XTS-AES-256.
RSIP_KEY_TYPE_CHACHA20	ChaCha20.
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC	secp256r1 public key (also known as NIST P-256, prime256v1)
RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC	secp384r1 public key (also known as NIST P-384)
RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC	secp521r1 public key (also known as NIST P-521)
RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC	secp256k1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC	brainpoolP256r1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC	brainpoolP384r1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC	brainpoolP512r1 public key
RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC	edwards25519 public key
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE	secp256r1 private key (also known as NIST P-256, prime256v1)
RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE	secp384r1 private key (also known as NIST P-384)
RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE	secp521r1 private key (also known as NIST P-521)
RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE	secp256k1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE	brainpoolP256r1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE	brainpoolP384r1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE	brainpoolP512r1 private key
RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE	edwards25519 private key

RSIP_KEY_TYPE_RSA_1024_PUBLIC	RSA-1024 public key.
RSIP_KEY_TYPE_RSA_2048_PUBLIC	RSA-2048 public key.
RSIP_KEY_TYPE_RSA_3072_PUBLIC	RSA-3072 public key.
RSIP_KEY_TYPE_RSA_4096_PUBLIC	RSA-4096 public key.
RSIP_KEY_TYPE_RSA_1024_PRIVATE	RSA-1024 private key.
RSIP_KEY_TYPE_RSA_2048_PRIVATE	RSA-2048 private key.
RSIP_KEY_TYPE_RSA_3072_PRIVATE	RSA-3072 private key.
RSIP_KEY_TYPE_RSA_4096_PRIVATE	RSA-4096 private key.
RSIP_KEY_TYPE_HMAC_SHA1	HMAC-SHA1.
RSIP_KEY_TYPE_HMAC_SHA224	HMAC-SHA224.
RSIP_KEY_TYPE_HMAC_SHA256	HMAC-SHA256.
RSIP_KEY_TYPE_HMAC_SHA384	HMAC-SHA384.
RSIP_KEY_TYPE_HMAC_SHA512	HMAC-SHA512.
RSIP_KEY_TYPE_HMAC_SHA512_224	HMAC-SHA512/224.
RSIP_KEY_TYPE_HMAC_SHA512_256	HMAC-SHA512/256.
RSIP_KEY_TYPE_HMAC_SHA3_224	HMAC-SHA3-224.
RSIP_KEY_TYPE_HMAC_SHA3_256	HMAC-SHA3-256.
RSIP_KEY_TYPE_HMAC_SHA3_384	HMAC-SHA3-384.
RSIP_KEY_TYPE_HMAC_SHA3_512	HMAC-SHA3-512.
RSIP_KEY_TYPE_KDF_HMAC_SHA256	KDF HMAC-SHA256.
RSIP_KEY_TYPE_KDF_HMAC_SHA384	KDF HMAC-SHA384.
RSIP_KEY_TYPE_KDF_HMAC_SHA512	KDF HMAC-SHA512.
RSIP_KEY_TYPE_KUK	Key Update Key (KUK)

Function Documentation

◆ R_RSIP_AES128_InitialKeyWrap()

fsp_err_t R_RSIP_AES128_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key*)

This API generates 128-bit AES key within the user routine.

Implements rsip_key_injection_api_t::AES128_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_AES192_InitialKeyWrap()

fsp_err_t R_RSIP_AES192_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key*)

This API generates 192-bit AES key within the user routine.

Implements rsip_key_injection_api_t::AES192_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	192-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_AES256_InitialKeyWrap()

fsp_err_t R_RSIP_AES256_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit AES key within the user routine.

Implements rsip_key_injection_api_t::AES256_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_AES128_XTS_InitialKeyWrap()

fsp_err_t R_RSIP_AES128_XTS_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key*)

This API generates 128-bit AES-XTS key within the user routine.

Implements rsip_key_injection_api_t::AES128_XTS_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	128-bit AES-XTS wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_AES256_XTS_InitialKeyWrap()

fsp_err_t R_RSIP_AES256_XTS_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit AES-XTS key within the user routine.

Implements rsip_key_injection_api_t::AES256_XTS_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit AES-XTS wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ChaCha20_InitialKeyWrap()

fsp_err_t R_RSIP_ChaCha20_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_chacha20_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit ChaCha20-Poly-1305 key within the user routine.

Implements rsip_key_injection_api_t::ChaCha20_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit ChaCha20 wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_RSA2048_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_RSA2048_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_rsa2048_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 2048-bit RSA private key within the user routine.

Implements rsip_key_injection_api_t::RSA2048_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	2048-bit RSA wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note
The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_RSA3072_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_RSA3072_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_rsa3072_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 3072-bit RSA private key within the user routine.

Implements rsip_key_injection_api_t::RSA3072_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	3072-bit RSA wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_RSA4096_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_RSA4096_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_rsa4096_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 4096-bit RSA private key within the user routine.

Implements rsip_key_injection_api_t::RSA4096_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	4096-bit RSA wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_secp256r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp256r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_secp256r1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_secp384r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp384r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 384-bit ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_secp384r1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	384-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_secp521r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp521r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 521-bit ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_secp521r1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	521-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_secp256k1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp256k1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_secp256k1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_brainpoolP256r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_brainpoolP256r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit brainpool ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_brainpoolP256r1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_brainpoolP384r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_brainpoolP384r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 384-bit brainpool ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_brainpoolP384r1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	384-bit ECC wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_brainpoolP512r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_brainpoolP512r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 512-bit brainpool ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_brainpoolP512r1_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	512-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_ECC_Ed25519_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_Ed25519_InitialPrivateKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit Edwards curve ECC private key within the user routine.

Implements rsip_key_injection_api_t::ECC_Ed25519_InitialPrivateKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit ECC wrapped private key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note
The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_HMAC_SHA224_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA224_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*)

This API generates 224-bit HMAC-SHA key within the user routine.

Implements rsip_key_injection_api_t::HMAC_SHA224_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	224-bit HMAC-SHA wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_HMAC_SHA256_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA256_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*)

This API generates 256-bit HMAC-SHA key within the user routine.

Implements rsip_key_injection_api_t::HMAC_SHA256_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	256-bit HMAC-SHA wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_HMAC_SHA384_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA384_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*)

This API generates 384-bit HMAC-SHA key within the user routine.

Implements rsip_key_injection_api_t::HMAC_SHA384_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	384-bit HMAC-SHA wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_HMAC_SHA512_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA512_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*)

This API generates 512-bit HMAC-SHA key within the user routine.

Implements rsip_key_injection_api_t::HMAC_SHA512_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	512-bit HMAC-SHA wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note
The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_HMAC_SHA512_224_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA512_224_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*)

This API generates 512/224-bit HMAC-SHA key within the user routine.

Implements rsip_key_injection_api_t::HMAC_SHA512_224_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	512/224-bit HMAC-SHA wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

◆ R_RSIP_HMAC_SHA512_256_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA512_256_InitialKeyWrap (rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*)

This API generates 512/256-bit HMAC-SHA key within the user routine.

Implements rsip_key_injection_api_t::HMAC_SHA512_256_InitialKeyWrap.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it is encrypted and MAC appended
[out]	p_wrapped_key	512/256-bit HMAC-SHA wrapped key

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_SCE_FAIL	MAC anomaly detection.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict.
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.
FSP_ERR_INVALID_STATE	Internal state is illegal.

FSP_ERR_UNSUPPORTED	This API is not supported on this device.

Note

The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.

5.2.15.2 Renesas Secure IP (r_rsip_protected)

Modules » Security

Functions

fsp_err_t	R_RSIP_Open (rsip_ctrl_t *const p_ctrl, rsip_cfg_t const *const p_cfg)
fsp_err_t	R_RSIP_Close (rsip_ctrl_t *const p_ctrl)
fsp_err_t	R_RSIP_RandomNumberGenerate (rsip_ctrl_t *const p_ctrl, uint8_t *const p_random)
fsp_err_t	R_RSIP_KeyGenerate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_KeyPairGenerate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_public_key, rsip_wrapped_key_t *const p_wrapped_private_key)
fsp_err_t	R_RSIP_EncryptedKeyWrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_key_update_key, void const *const p_initial_vector, void const *const p_encrypted_key, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_RFC3394_KeyWrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, rsip_wrapped_key_t const *const p_wrapped_target_key, uint8_t *const p_rfc3394_wrapped_target_key)
fsp_err_t	R_RSIP_RFC3394_KeyUnwrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, uint8_t const *const p_rfc3394_wrapped_target_key, rsip_wrapped_key_t *const p_wrapped_target_key)
fsp_err_t	R_RSIP_PublicKeyExport (rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t *const p_raw_public_key)
fsp_err_t	R_RSIP_AES_Cipher_Init (rsip_ctrl_t *const p_ctrl, rsip_aes_cipher_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_initial_vector)
fsp_err_t	R_RSIP_AES_Cipher_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint8_t *const p_output, uint32_t const length)

fsp_err_t	R_RSIP_AES_Cipher_Finish (rsip_ctrl_t *const p_ctrl)
fsp_err_t	R_RSIP_AES_AEAD_Init (rsip_ctrl_t *const p_ctrl, rsip_aes_aead_mode_t mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)
fsp_err_t	R_RSIP_AES_AEAD_LengthsSet (rsip_ctrl_t *const p_ctrl, uint32_t const total_aad_length, uint32_t const total_text_length, uint32_t const tag_length)
fsp_err_t	R_RSIP_AES_AEAD_AADUpdate (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)
fsp_err_t	R_RSIP_AES_AEAD_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t	R_RSIP_AES_AEAD_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)
fsp_err_t	R_RSIP_AES_AEAD_Verify (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)
fsp_err_t	R_RSIP_AES_MAC_Init (rsip_ctrl_t *const p_ctrl, rsip_aes_mac_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key)
fsp_err_t	R_RSIP_AES_MAC_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t	R_RSIP_AES_MAC_SignFinish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)
fsp_err_t	R_RSIP_AES_MAC_VerifyFinish (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)
fsp_err_t	R_RSIP_ChaCha20_Init (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const counter)
fsp_err_t	R_RSIP_ChaCha20_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t	R_RSIP_ChaCha20_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t	R_RSIP_ChaCha20_Poly1305_Init (rsip_ctrl_t *const p_ctrl, rsip_chacha_poly_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const

	nonce_length)
fsp_err_t	R_RSIP_ChaCha20_Poly1305_AADUpdate (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)
fsp_err_t	R_RSIP_ChaCha20_Poly1305_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t	R_RSIP_ChaCha20_Poly1305_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)
fsp_err_t	R_RSIP_ChaCha20_Poly1305_Verify (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)
fsp_err_t	R_RSIP_xDLMS_InitiateRequest_Decrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_apdu, uint8_t const *const p_nonce, uint8_t const *const p_aad, uint32_t const aad_length, rsip_wrapped_key_t *const p_dedicated_key, rsip_dlms_apdu_data_t *const p_apdu_data, rsip_dlms_ret_t *const p_dlms_ret)
fsp_err_t	R_RSIP_ECDSA_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_hash, uint8_t *const p_signature)
fsp_err_t	R_RSIP_ECDSA_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_PKI_ECDSA_CertVerify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_PureEdDSA_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t *const p_signature)
fsp_err_t	R_RSIP_PureEdDSA_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_ECDH_KeyAgree (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_wrapped_secret_t *const p_wrapped_secret)

fsp_err_t	R_RSIP_ECDH_PlainKeyAgree (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_plain_public_key, rsip_wrapped_secret_t *const p_wrapped_secret)
fsp_err_t	R_RSIP_KDF_SHA_Init (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)
fsp_err_t	R_RSIP_KDF_SHA_ECDHSecretUpdate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_t const *const p_wrapped_secret)
fsp_err_t	R_RSIP_KDF_SHA_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t	R_RSIP_KDF_SHA_Finish (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)
fsp_err_t	R_RSIP_KDF_SHA_Suspend (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_t *const p_handle)
fsp_err_t	R_RSIP_KDF_SHA_Resume (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_t const *const p_handle)
fsp_err_t	R_RSIP_KDF_HMAC_DKMKeyImport (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const key_length, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_KDF_HMAC_ECDHSecretKeyImport (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_t const *const p_wrapped_secret, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_KDF_HMAC_Init (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)
fsp_err_t	R_RSIP_KDF_HMAC_DKMUpdate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm)
fsp_err_t	R_RSIP_KDF_HMAC_ECDHSecretUpdate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_t const *const p_wrapped_secret)
fsp_err_t	R_RSIP_KDF_HMAC_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t	R_RSIP_KDF_HMAC_SignFinish (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)
fsp_err_t	R_RSIP_KDF_HMAC_Suspend (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_t *const p_handle)
fsp_err_t	R_RSIP_KDF_HMAC_Resume (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_t const *const p_handle)

fsp_err_t	R_RSIP_KDF_DKMConcatenate (rsip_wrapped_dkm_t *const p_wrapped_dkm1, rsip_wrapped_dkm_t const *const p_wrapped_dkm2, uint32_t const wrapped_dkm1_buffer_length)
fsp_err_t	R_RSIP_KDF_DerivedKeyImport (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const position, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t	R_RSIP_KDF_DerivedIVWrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, rsip_initial_vector_type_t const initial_vector_type, uint32_t const position, uint8_t const *const p_tls_sequence_num, uint8_t *const p_wrapped_initial_vector)
fsp_err_t	R_RSIP_OTF_Init (rsip_ctrl_t *const p_ctrl, rsip_otf_channel_t const channel, rsip_wrapped_key_t *const p_wrapped_key, uint8_t const *const p_seed)
fsp_err_t	R_RSIP_PKI_VerifiedCertInfoExport (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_t *const p_verified_cert_info)
fsp_err_t	R_RSIP_PKI_VerifiedCertInfoImport (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_t const *const p_verified_cert_info)
fsp_err_t	R_RSIP_PKI_CertKeyImport (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_cert, uint32_t const cert_length, uint8_t const *const p_key_param1, uint32_t const key_param1_length, uint8_t const *const p_key_param2, uint32_t const key_param2_length, rsip_hash_type_t const hash_function, rsip_wrapped_key_t *const p_wrapped_public_key)
fsp_err_t	R_RSIP_RSA_Encrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint8_t *const p_cipher)
fsp_err_t	R_RSIP_RSA_Decrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain)
fsp_err_t	R_RSIP_RSAES_PKCS1_V1_5_Encrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)
fsp_err_t	R_RSIP_RSAES_PKCS1_V1_5_Decrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)
fsp_err_t	R_RSIP_RSAES_OAEP_Encrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key,

	rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)
fsp_err_t	R_RSIP_RSAES_OAEP_Decrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)
fsp_err_t	R_RSIP_RSASSA_PKCS1_V1_5_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t *const p_signature)
fsp_err_t	R_RSIP_RSASSA_PKCS1_V1_5_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_RSASSA_PSS_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t *const p_signature)
fsp_err_t	R_RSIP_RSASSA_PSS_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_PKI_RSASSA_PSS_CertVerify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t	R_RSIP_SHA_Compute (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_digest)
fsp_err_t	R_RSIP_SHA_Init (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)

fsp_err_t	R_RSIP_SHA_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t	R_RSIP_SHA_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_digest)
fsp_err_t	R_RSIP_SHA_Suspend (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_t *const p_handle)
fsp_err_t	R_RSIP_SHA_Resume (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_t const *const p_handle)
fsp_err_t	R_RSIP_HMAC_Compute (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_mac)
fsp_err_t	R_RSIP_HMAC_Verify (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t const *const p_mac, uint32_t const mac_length)
fsp_err_t	R_RSIP_HMAC_Init (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)
fsp_err_t	R_RSIP_HMAC_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t	R_RSIP_HMAC_SignFinish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)
fsp_err_t	R_RSIP_HMAC_VerifyFinish (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)
fsp_err_t	R_RSIP_HMAC_Suspend (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_t *const p_handle)
fsp_err_t	R_RSIP_HMAC_Resume (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_t const *const p_handle)

Detailed Description

Driver for the Renesas Secure IP on RX MPUs. This module implements the RSIP Interface.

Overview

This module provides RSIP functions in protected mode.

HW Overview

Crypto Peripheral version	Devices

RSIP-E50D	RX74M/N

Features

The RSIP module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption
    - AES
      - ECB 128/192/256bit
      - CBC 128/192/256bit
      - CTR 128/192/256bit
      - GCM 128/192/256bit
      - CCM 128/192/256bit
      - XTS 128/256bit
    - ChaCha
      - ChaCha20
      - ChaCha20-Poly1305
  - Asymmetric Encryption/Decryption
    - RSA
      - RSAES-PKCS1-V1_5 2048/3072/4096bit
      - RSAES-OAEP 2048/3072/4096bit
      - RSASSA-PKCS1-V1_5 2048/3072/4096bit
      - RSASSA-PSS 2048/3072/4096bit
    - ECC
      - ECDSA
        - secp256r1/secp384r1/secp521r1
        - secp256k1
        - brainpool P256r1/P384r1/P512r1
      - EdDSA
        - Ed25519
  - Key Agreement
    - ECDH
      - secp256r1/secp384r1/secp521r1
      - brainpool P256r1/P384r1/P512r1
  - Key Derivation
    - Key Derivation Function (KDF) using SHA
      - SHA-256/384/512bit
    - Key Derivation Function (KDF) using HMAC
      - HMAC-SHA256/384/512bit
    - Key Import by derived keying material
    - IV output by derived keying material
  - Public Key Infrastructure (PKI)
    - Certificate Verification
      - RSA
        - RSASSA-PKCS1-V1_5 2048/3072/4096bit
        - RSASSA-PSS 2048/3072/4096bit
      - ECDSA
        - secp256r1/secp384r1/secp521r1
        - brainpool P256r1/P384r1/P512r1
    - Key Import from Certificate
      - RSA 2048/3072/4096bit
      - ECC
        - secp256r1/secp384r1/secp521r1
        - brainpool P256r1/P384r1/P512r1

        - Hash Functions
            - SHA-2
                - SHA-224, 256, 384, 512, 512/224, 512/256
            - SHA-3
                - SHA3-224, 256, 384, 512
        - RFC3394 Key Wrap/Unwrap
    - Message Authentication Code
        - HMAC-SHA224, 256, 384, 512, 512/224, 512/256 bit
        - HMAC-SHA3 224, 256, 384, 512 bit
        - AES-CMAC 128/192/256bit
    - Key Manage
        - Key Support
            - AES 128/192/256bit
            - RSA 2048/3072/4096bit
            - ECC
                - secp256r1/secp384r1/secp521r1
                - secp256k1
                - brainpool P256r1/P384r1/P512r1
            - HMAC-SHA224, 256, 384, 512, 512/224, 512/256 bit
            - HMAC-SHA3 224, 256, 384, 512 bit
    - Random Number Generate
    - DLMS/COSEM (Not Supported)

Supported algorithms

The following algorithms are available on each devices.

- RSIP-E50D: RX74M/N

Symmetric Key

Key type (rsip_key_type_t)	RSIP-E50D
RSIP_KEY_TYPE_AES_128	Supported
RSIP_KEY_TYPE_AES_192	Supported
RSIP_KEY_TYPE_AES_256	Supported
RSIP_KEY_TYPE_XTS_AES_128	Supported
RSIP_KEY_TYPE_XTS_AES_256	Supported
RSIP_KEY_TYPE_CHACHA20	Supported
RSIP_KEY_TYPE_HMAC_SHA224	Supported
RSIP_KEY_TYPE_HMAC_SHA256	Supported
RSIP_KEY_TYPE_HMAC_SHA384	Supported
RSIP_KEY_TYPE_HMAC_SHA512	Supported
RSIP_KEY_TYPE_HMAC_SHA512_224	Supported
RSIP_KEY_TYPE_HMAC_SHA512_256	Supported
RSIP_KEY_TYPE_HMAC_SHA3_224	Supported
RSIP_KEY_TYPE_HMAC_SHA3_256	Supported

RSIP_KEY_TYPE_HMAC_SHA3_384	Supported
RSIP_KEY_TYPE_HMAC_SHA3_512	Supported

Asymmetric Key

Public key type (rsip_key_type_t)	Private key type (rsip_key_type_t)	RSIP-E50D
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC	RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC	RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC	RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC	RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC	RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC	RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC	RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE	Supported
RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC	RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE	Supported
RSIP_KEY_TYPE_RSA_2048_PUBLIC	RSIP_KEY_TYPE_RSA_2048_PRIVATE	Supported
RSIP_KEY_TYPE_RSA_3072_PUBLIC	RSIP_KEY_TYPE_RSA_3072_PRIVATE	Supported
RSIP_KEY_TYPE_RSA_4096_PUBLIC	RSIP_KEY_TYPE_RSA_4096_PRIVATE	Supported

AES Block Cipher Mode of Operation

Cipher mode (rsip_aes_cipher_mode_t)	RSIP-E50D
RSIP_AES_CIPHER_MODE_ECB_ENC	Supported
RSIP_AES_CIPHER_MODE_ECB_DEC	Supported
RSIP_AES_CIPHER_MODE_CBC_ENC	Supported
RSIP_AES_CIPHER_MODE_CBC_DEC	Supported
RSIP_AES_CIPHER_MODE_CTR	Supported
RSIP_AES_CIPHER_MODE_XTS_ENC	Supported
RSIP_AES_CIPHER_MODE_XTS_DEC	Supported
AEAD mode (rsip_aes_aead_mode_t)	**RSIP-E50D**
RSIP_AES_AEAD_MODE_GCM_ENC	Supported

RSIP_AES_AEAD_MODE_GCM_DEC	Supported
RSIP_AES_AEAD_MODE_CCM_ENC	Supported
RSIP_AES_AEAD_MODE_CCM_DEC	Supported

MAC mode (rsip_aes_mac_mode_t)	RSIP-E50D
RSIP_AES_MAC_MODE_CMAC	Supported

ChaCha20-Poly1305 Operation

This feature has a usage constraint that requires an input of at least one byte of encrypt/decrypt text data.

ChaCha20-Poly1305 mode (rsip_chacha_poly_mode_t)	RSIP-E50D
RSIP_CHACHA20_POLY1305_MODE_ENC	Supported
RSIP_CHACHA20_POLY1305_MODE_DEC	Supported

Hash Function

Using hardware-accelerated SHA-3 in Protected Mode for use with a software PQC library is prohibited by the Protected Mode security policy due to the requirement for plaintext key usage with a software cryptographic library.

Hash Function (rsip_hash_type_t)	RSIP-E50D
RSIP_HASH_TYPE_SHA224	Supported
RSIP_HASH_TYPE_SHA256	Supported
RSIP_HASH_TYPE_SHA384	Supported
RSIP_HASH_TYPE_SHA512	Supported
RSIP_HASH_TYPE_SHA512_224	Supported
RSIP_HASH_TYPE_SHA512_256	Supported
RSIP_HASH_TYPE_SHA3_224	Supported
RSIP_HASH_TYPE_SHA3_256	Supported
RSIP_HASH_TYPE_SHA3_384	Supported
RSIP_HASH_TYPE_SHA3_512	Supported

Key Derivation Function

Key Derivation Function	RSIP-E50D
SHA-256	Supported
SHA-384	Supported
SHA-512	Supported
HMAC-SHA256	Supported

HMAC-SHA384	Supported
HMAC-SHA512	Supported

Note : KDF-SHA 512 does not support using a shared secret generated by ECDH secp521r1 key agreement as input for the message.

Configuration

Build Time Configurations for r_rsip_e50d_protected

The following build time configurations are defined in fsp_cfg/r_rsip_cfg.h:

Configuration	Options	Default	Description
Selection of algorithms to enable > AES-128	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > AES-192	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > AES-256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > AES-ECB CBC CTR	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by

			disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > XTS-AES-128	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > XTS-AES-256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > XTS-AES	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > AES-GCM	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > AES-CCM	• Enabled • Disabled	Enabled	If the application uses only some of the

			algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > AES-CMAC	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ChaCha20	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ChaCha20-Poly1305	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC SECP256R1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it

			works effectively.
Selection of algorithms to enable > ECC SECP384R1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC SECP521R1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC SECP256K1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC BRAINPOOLP256R1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC BRAINPOOLP384R1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This

			configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC BRAINPOOLP512R1	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > ECC EDWARDS25519	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > RSA-2048	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > RSA-3072	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > RSA-4096	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size

			can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA-1	Disabled	Disabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA-224	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA-256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA-384	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms	• Enabled	Enabled	If the application uses

to enable > SHA-512	• Disabled		only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA-512_224	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA-512_256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA3-224	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA3-256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the

			functions for which it works effectively.
Selection of algorithms to enable > SHA3-384	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > SHA3-512	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA1	Disabled	Disabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA224	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused

			algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA384	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA512	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA512_224	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA512_256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA3_224	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the

			function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA3_256	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA3_384	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Selection of algorithms to enable > HMAC_SHA3_512	• Enabled • Disabled	Enabled	If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively.
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Clock Configuration

This module does not require a specific clock configuration.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Getting Started: Creating a RSIP Protected Mode Project

Start by creating a new project in e² studio or RXSC. On the Stacks tab, add New > Security > RSIP Protected Mode (r_rsip_protected).

State Transition

This driver has the following states.

State Name	Details
STATE_INITIAL	Driver is not open.
STATE_MAIN	Ready to Compute.
STATE_AES	Computing AES (unauthenticated cipher)
STATE_AES_AEAD	Computing AES (AEAD).
STATE_AES_MAC	Computing AES (MAC).
STATE_CHACHA	Computing ChaCha20.
STATE_CHACHA_POLY	Computing ChaCha20-Poly1305.
STATE_SHA	Computing SHA.
STATE_HMAC	Computing HMAC.
STATE_KDF_SHA	Computing KDF SHA.
STATE_KDF_HMAC	Computing KDF HMAC.

There are two types of APIs provided by the RSIP driver for accelerating cryptographic operations: those that provide cryptographic operations in a single API and those that provide them in multiple APIs. In this document, the former is referred to as single-part operations and the latter as multi-part operations. Each corresponds to the following algorithms:

- Single-part operations: DRBG, Key management, ECC, RSA, SHA, HMAC
- Multi-part operations: AES, SHA, HMAC

Multi-part operations are APIs which split a single cryptographic operation into a sequence of separate steps (e.g. Init-Update-Finish). This enables fine control over the configuration of the cryptographic operation, and allows the message data to be processed in fragments instead of all at once.

Due to the above characteristics, it is necessary to manage the operational states in the multi-part operations.

TRNG Initialization

The seed used for CTR-DRBG in R_RSIP_RandomNumberGenerate() is generated by R_RSIP_Open(). To reseed the random number generator, you need to call R_RSIP_Open() again.

Examples

AES cipher Example

This is an example of AES-CBC encryption/decryption.

```
void r_rsip_example_aes ()
{
 fsp_err_t            err;
 static const uint8_t plain[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x00
    };
    uint8_t iv[RSIP_PRV_BYTE_SIZE_AES_BLOCK] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t plain_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, iv);
    assert(FSP_SUCCESS == err);
```

```
 /* Encrypt a plaintext */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_ENC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, plain, cipher_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_DEC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, cipher_calculated, plain_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl,
                                   cipher_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   plain_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

AES-GCM Example

This is an example of AES-GCM encryption/decryption.

```
#define AES_GCM_TEST_GCM_BLOCK_LEN (16)
```

```
#define AES_GCM_TEST_GCM_TEXT_LEN (31)
#define GCM_NONCE_LEN (12)
#define AES_GCM_TEST_GCM_AAD_LEN (8)
#define AES_GCM_TEST_GCM_TAG_LEN (8)
void r_rsip_example_gcm ()
{
 fsp_err_t            err;
 static const uint8_t plain[AES_GCM_TEST_GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[AES_GCM_TEST_GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[AES_GCM_TEST_GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                =
{0};
    uint8_t tag[AES_GCM_TEST_GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                   =
{0};
    uint8_t cipher_calculated[AES_GCM_TEST_GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) =
{0};
    uint8_t plain_calculated[AES_GCM_TEST_GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) =
{0};
    uint32_t input_length      = 0;
    uint32_t output_length    = 0;
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
```

```
rsip_wrapped_key_t wrapped_key =
   {
      .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
   };
/* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
/* Generate a aes 256-bit key */
   err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
   assert(FSP_SUCCESS == err);
/* Generate a nonce */
   err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
   assert(FSP_SUCCESS == err);
/* Encrypt a plaintext */
   err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GCM_NONCE_LEN);
   assert(FSP_SUCCESS == err);
/* Input AAD */
   err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, AES_GCM_TEST_GCM_AAD_LEN);
   assert(FSP_SUCCESS == err);
/* Input plaintext (one-shot) */
   err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                plain,
                                AES_GCM_TEST_GCM_TEXT_LEN,
                                cipher_calculated,
                                &output_length_tmp);
   output_length += output_length_tmp;
   assert(FSP_SUCCESS == err);
/* Output remaining text and generate a tag */
   err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
   assert(FSP_SUCCESS == err);
/* Decrypt a ciphertext using the same key */
   err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
```

```
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, AES_GCM_TEST_GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
    input_length = 0;
    output_length = 0;
 /* Input ciphertext (multi-shot) */
    input_length_tmp = 10;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                  &output_length_tmp);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
    input_length_tmp = AES_GCM_TEST_GCM_TEXT_LEN - input_length;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                  &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /*
  * Output remaining text and verify the tag
  * Attention: Do not use the output plaintext by R_RSIP_AES_Cipher_DecryptUpdate()
  * before R_RSIP_AES_Cipher_DecryptFinal() returns FSP_SUCCESS.
  * The integrity is checked in this function.
  */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl,
                                 plain_calculated + output_length,
                                 &output_length_tmp,
```

```
                                tag,
                                AES_GCM_TEST_GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, AES_GCM_TEST_GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

AES-GMAC Example

This is an example of AES-GMAC signature/verification.

```
#define GMAC_NONCE_LEN (12)
#define GMAC_AAD_LEN (8)
#define GMAC_TAG_LEN (16)
void r_rsip_example_gmac ()
{
 fsp_err_t err;
 static const uint8_t aad[GMAC_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GMAC_NONCE_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t tag[GMAC_TAG_LEN] BSP_ALIGN_VARIABLE(4)    = {0};
    uint8_t output_dummy_buf[1] = {0};
    uint32_t output_length       = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
```

```
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
    assert(FSP_SUCCESS == err);
 /* Initialize GMAC signature */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC generation */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, output_dummy_buf, &output_length,
tag);
    assert(FSP_SUCCESS == err);
 /*Initialize GMAC verification*/
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC verification*/
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, output_dummy_buf, &output_length, tag,
GMAC_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
```

```
    assert(FSP_SUCCESS == err);
}
```

AES-CCM Example

This is an example of AES-CCM encryption/decryption.

```
#define CCM_NONCE_LEN (12)
#define CCM_ADATA_LEN (20)
#define CCM_INPUT_DATA_LEN (24)
#define CCM_MAC_LEN (8)
void r_rsip_example_ccm ()
{
 fsp_err_t err;
 static const uint8_t nonce[CCM_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B};
 static const uint8_t adata[CCM_ADATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
       0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13
    };
 static const uint8_t plain[CCM_INPUT_DATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29,
       0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33,
       0x34, 0x35, 0x36, 0x37
    };
    uint8_t   cipher_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   plain_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   mac[CCM_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length_tmp = 0;
    uint8_t * p_output          = NULL;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
```

```
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_ENC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(cipher_calculated[output_length_tmp]);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, p_output, &output_length_tmp, mac);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_DEC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
```

```
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
 // err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
plain_calculated, &output_length_tmp);
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                 cipher_calculated,
                                 CCM_INPUT_DATA_LEN,
                                 plain_calculated,
                                 &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(plain_calculated[output_length_tmp]);
 /* Output remaining text and verify the tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, p_output, &output_length_tmp, mac,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

AES-CMAC Example

This is an example of AES-CMAC signature/verification.

```
#define CMAC_MESSAGE_LEN (10)
#define CMAC_MAC_LEN (16)
void r_rsip_example_cmac ()
{
 fsp_err_t err;
```

```
 static const uint8_t message[CMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0xf0, 0x8f, 0x89, 0x08, 0x75, 0xe1, 0x39, 0x48, 0x04, 0x89
    };
    uint8_t mac[CMAC_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* CMAC initialize for encrypt */
    err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Input message */
    err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_MAC_SignFinish(&g_rsip_ctrl, mac);
    assert(FSP_SUCCESS == err);
 /* CMAC initialize for decrypt */
    err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Input message */
    err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
```

```
 /* Verify the tag */
    err = R_RSIP_AES_MAC_VerifyFinish(&g_rsip_ctrl, mac, CMAC_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

ChaCha20 Example

This is an example of ChaCha20 encryption/decryption.

```
#define CHACHA20_BLOCK_LEN (64)
#define CHACHA20_TEXT_LEN (114)
#define CHACHA20_NONCE_LEN (12)
void r_rsip_example_chacha ()
{
 fsp_err_t            err;
 static const uint8_t plain[CHACHA20_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x4c, 0x61, 0x64, 0x69, 0x65, 0x73, 0x20, 0x61, 0x6e, 0x64, 0x20, 0x47, 0x65,
0x6e, 0x74, 0x6c,
       0x65, 0x6d, 0x65, 0x6e, 0x20, 0x6f, 0x66, 0x20, 0x74, 0x68, 0x65, 0x20, 0x63,
0x6c, 0x61, 0x73,
       0x73, 0x20, 0x6f, 0x66, 0x20, 0x27, 0x39, 0x39, 0x3a, 0x20, 0x49, 0x66, 0x20,
0x49, 0x20, 0x63,
       0x6f, 0x75, 0x6c, 0x64, 0x20, 0x6f, 0x66, 0x66, 0x65, 0x72, 0x20, 0x79, 0x6f,
0x75, 0x20, 0x6f,
       0x6e, 0x6c, 0x79, 0x20, 0x6f, 0x6e, 0x65, 0x20, 0x74, 0x69, 0x70, 0x20, 0x66,
0x6f, 0x72, 0x20,
       0x74, 0x68, 0x65, 0x20, 0x66, 0x75, 0x74, 0x75, 0x72, 0x65, 0x2c, 0x20, 0x73,
0x75, 0x6e, 0x73,
       0x63, 0x72, 0x65, 0x65, 0x6e, 0x20, 0x77, 0x6f, 0x75, 0x6c, 0x64, 0x20, 0x62,
0x65, 0x20, 0x69,
       0x74, 0x2e,
```

```
    };
 static const uint8_t nonce[CHACHA20_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,
    };
    uint32_t counter = 0;
    uint8_t cipher_calculated[CHACHA20_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[CHACHA20_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t input_length      = 0;
    uint32_t output_length    = 0;
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_CHACHA20)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_CHACHA20, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a ChaCha20 key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
/* Encrypt plaintext with one-shot update */
 /* Initialize ChaCha20 operation */
    err = R_RSIP_ChaCha20_Init(&g_rsip_ctrl, &wrapped_key, nonce, counter);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) and output encrypted text */
    err = R_RSIP_ChaCha20_Update(&g_rsip_ctrl, plain, CHACHA20_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
```

```
    output_length += output_length_tmp;
 /* Output remaining encrypted text */
    err = R_RSIP_ChaCha20_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp);
    assert(FSP_SUCCESS == err);
/* Decrypt ciphertext with multi-shot update */
 /* Initialize ChaCha20 operation */
    err = R_RSIP_ChaCha20_Init(&g_rsip_ctrl, &wrapped_key, nonce, counter);
    assert(FSP_SUCCESS == err);
    input_length = 0;
    output_length = 0;
 /* Input ciphertext (first 64 bytes) and output decrypted text */
    input_length_tmp = CHACHA20_BLOCK_LEN;
    err             = R_RSIP_ChaCha20_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                &output_length_tmp);
    assert(FSP_SUCCESS == err);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
 /* Input remaining ciphertext and output decrypted text */
    input_length_tmp = CHACHA20_TEXT_LEN - input_length;
    err             = R_RSIP_ChaCha20_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining decrypted text */
    err = R_RSIP_ChaCha20_Finish(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp);
    assert(FSP_SUCCESS == err);
```

```
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, CHACHA20_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

ChaCha20-Poly1305 Example

This is an example of ChaCha20-Poly1305 encryption/decryption.

```
#define CHACHA_POLY_BLOCK_LEN (64)
#define CHACHA_POLY_TEXT_LEN (114)
#define CHACHA_POLY_NONCE_LEN (12)
#define CHACHA_POLY_AAD_LEN (12)
#define CHACHA_POLY_TAG_LEN (16)
void r_rsip_example_chacha_poly ()
{
 fsp_err_t            err;
 static const uint8_t plain[CHACHA_POLY_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x4c, 0x61, 0x64, 0x69, 0x65, 0x73, 0x20, 0x61, 0x6e, 0x64, 0x20, 0x47, 0x65,
0x6e, 0x74, 0x6c,
       0x65, 0x6d, 0x65, 0x6e, 0x20, 0x6f, 0x66, 0x20, 0x74, 0x68, 0x65, 0x20, 0x63,
0x6c, 0x61, 0x73,
       0x73, 0x20, 0x6f, 0x66, 0x20, 0x27, 0x39, 0x39, 0x3a, 0x20, 0x49, 0x66, 0x20,
0x49, 0x20, 0x63,
       0x6f, 0x75, 0x6c, 0x64, 0x20, 0x6f, 0x66, 0x66, 0x65, 0x72, 0x20, 0x79, 0x6f,
0x75, 0x20, 0x6f,
       0x6e, 0x6c, 0x79, 0x20, 0x6f, 0x6e, 0x65, 0x20, 0x74, 0x69, 0x70, 0x20, 0x66,
0x6f, 0x72, 0x20,
       0x74, 0x68, 0x65, 0x20, 0x66, 0x75, 0x74, 0x75, 0x72, 0x65, 0x2c, 0x20, 0x73,
0x75, 0x6e, 0x73,
       0x63, 0x72, 0x65, 0x65, 0x6e, 0x20, 0x77, 0x6f, 0x75, 0x6c, 0x64, 0x20, 0x62,
0x65, 0x20, 0x69,
```

```
      0x74, 0x2e,
   };
 static const uint8_t nonce[CHACHA_POLY_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
   {
      0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,
   };
 static const uint8_t aad[CHACHA_POLY_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
   {
      0x50, 0x51, 0x52, 0x53, 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
   };
   uint8_t cipher_calculated[CHACHA_POLY_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
   uint8_t plain_calculated[CHACHA_POLY_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
   uint8_t tag[CHACHA_POLY_TAG_LEN] BSP_ALIGN_VARIABLE(4)                = {0};
   uint32_t input_length      = 0;
   uint32_t output_length    = 0;
   uint32_t input_length_tmp = 0;
   uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_CHACHA20)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
   {
      .type = RSIP_KEY_TYPE_CHACHA20,
      .p_value = &wrapped_key_value
   };
 /* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
 /* Generate a ChaCha20 key */
   err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
   assert(FSP_SUCCESS == err);
/* Encrypt plaintext with one-shot update */
 /* Initialize ChaCha20-Poly1305 operation */
```

```
    err = R_RSIP_ChaCha20_Poly1305_Init(&g_rsip_ctrl,
 RSIP_CHACHA20_POLY1305_MODE_ENC,
                                        &wrapped_key,
                                        nonce,
                                        CHACHA_POLY_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_ChaCha20_Poly1305_AADUpdate(&g_rsip_ctrl, aad, CHACHA_POLY_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) and output encrypted text */
    err = R_RSIP_ChaCha20_Poly1305_Update(&g_rsip_ctrl,
                                    plain,
                                           CHACHA_POLY_TEXT_LEN,
                                    cipher_calculated,
                                    &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining encrypted text and generate a tag */
    err = R_RSIP_ChaCha20_Poly1305_Finish(&g_rsip_ctrl, cipher_calculated +
output_length, &output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
/* Decrypt ciphertext with multi-shot update */
 /* Initialize ChaCha20 operation */
    err = R_RSIP_ChaCha20_Poly1305_Init(&g_rsip_ctrl,
 RSIP_CHACHA20_POLY1305_MODE_DEC,
                                        &wrapped_key,
                                        nonce,
                                        CHACHA_POLY_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_ChaCha20_Poly1305_AADUpdate(&g_rsip_ctrl, aad, CHACHA_POLY_AAD_LEN);
    assert(FSP_SUCCESS == err);
    input_length = 0;
    output_length = 0;
```

```
/* Input ciphertext (first 64 bytes) and output decrypted text */
   input_length_tmp = CHACHA_POLY_BLOCK_LEN;
   err              = R_RSIP_ChaCha20_Poly1305_Update(&g_rsip_ctrl,
                                                      cipher_calculated + input_length,
                                                      input_length_tmp,
                                                      plain_calculated + output_length,
                                    &output_length_tmp);
   assert(FSP_SUCCESS == err);
   input_length += input_length_tmp;
   output_length += output_length_tmp;
/* Input remaining ciphertext and output decrypted text */
   input_length_tmp = CHACHA_POLY_TEXT_LEN - input_length;
   err              = R_RSIP_ChaCha20_Poly1305_Update(&g_rsip_ctrl,
                                                      cipher_calculated + input_length,
                                                      input_length_tmp,
                                                      plain_calculated + output_length,
                                    &output_length_tmp);
   assert(FSP_SUCCESS == err);
   output_length += output_length_tmp;
/* Output remaining decrypted text and verify the tag*/
   err = R_RSIP_ChaCha20_Poly1305_Verify(&g_rsip_ctrl,
                                    plain_calculated + output_length,
                                    &output_length_tmp,
                                    tag,
                                           CHACHA_POLY_TAG_LEN);
   assert(FSP_SUCCESS == err);
/* Compare plain and plain_calculated */
   assert(0 == memcmp(plain, plain_calculated, CHACHA_POLY_TEXT_LEN));
/* Close the driver */
   err = R_RSIP_Close(&g_rsip_ctrl);
   assert(FSP_SUCCESS == err);
}
```

ECC Example

This is an example of ECDSA signature generation/verification.

```
#define SECP256R1_KEY_SIZE (32)
void r_rsip_example_ecc ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
    uint8_t hash[SECP256R1_KEY_SIZE] BSP_ALIGN_VARIABLE(4)          = {0};
    uint8_t signature[SECP256R1_KEY_SIZE * 2] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
```

```
 /* Generate the message hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
 /* Verify the signature*/
    err = R_RSIP_ECDSA_Verify(&g_rsip_ctrl, &wrapped_public_key, hash, signature);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

RSA Example

This is an example of RSA encryption/decryption (RSAES-OAEP) and RSA signature generation/verification (RSASSA-PSS).

```
#define SHA256_HASH_SIZE (32)
#define RSA_2048_KEY_SIZE (256)
void r_rsip_example_rsa ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
 static const uint8_t label[] BSP_ALIGN_VARIABLE(4) = "label";
    uint8_t cipher_calculated[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t plain_calculated_length = 0;
    uint8_t hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)       = {0};
    uint8_t signature[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_RSA_2048_PUBLIC)]
```

```
   BSP_ALIGN_VARIABLE(4);
rsip_wrapped_key_t wrapped_public_key =
   {
      .type = RSIP_KEY_TYPE_RSA_2048_PUBLIC, .p_value = &wrapped_public_key_value
   };
   uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_RSA_2048_PRIVATE)]
   BSP_ALIGN_VARIABLE(4);
rsip_wrapped_key_t wrapped_private_key =
   {
      .type = RSIP_KEY_TYPE_RSA_2048_PRIVATE, .p_value = &wrapped_private_key_value
   };
/* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
/* Generate a key pair of RSA-2048 */
   err =
R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key, &wrapped_private_key);
   assert(FSP_SUCCESS == err);
/*
 * RSAES-OAEP
 */
/* Encrypt a plaintext */
   err = R_RSIP_RSAES_OAEP_Encrypt(&g_rsip_ctrl,
                                   &wrapped_public_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
                                   label,
sizeof(label),
                                   message,
sizeof(message),
                                   cipher_calculated);
   assert(FSP_SUCCESS == err);
/* Decrypt a ciphertext using the same key */
```

```
   err = R_RSIP_RSAES_OAEP_Decrypt(&g_rsip_ctrl,
                                   &wrapped_private_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
                                   label,
sizeof(label),
                                  cipher_calculated,
                                  plain_calculated,
                                   &plain_calculated_length,
sizeof(plain_calculated));
   assert(FSP_SUCCESS == err);
/* Verify calculated plaintext */
   assert(sizeof(message) == plain_calculated_length);
   assert(0 == memcmp(message, plain_calculated, plain_calculated_length));
/*
 * RSASSA-PSS
 */
/* Generate the message hash */
   err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
   assert(FSP_SUCCESS == err);
/* Generate the signature */
   err = R_RSIP_RSASSA_PSS_Sign(&g_rsip_ctrl,
                                &wrapped_private_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
RSIP_RSA_SALT_LENGTH_AUTO,
                                hash,
                                signature);
   assert(FSP_SUCCESS == err);
/* Verify the signature*/
   err = R_RSIP_RSASSA_PSS_Verify(&g_rsip_ctrl,
                                 &wrapped_public_key,
RSIP_HASH_TYPE_SHA256,
```

```
 RSIP_MGF_TYPE_MGF1_SHA256,
 RSIP_RSA_SALT_LENGTH_AUTO,
                                  hash,
                                  signature);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

HASH Example

This is an example of calculating the SHA-256 message digest.

```
#define SHA256_HASH_SIZE (32)
void r_rsip_example_hash ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE (4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
       0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
    };
 static const uint8_t hash[] BSP_ALIGN_VARIABLE (4) =
    {
       0x63, 0x0d, 0xcd, 0x29, 0x66, 0xc4, 0x33, 0x66, 0x91, 0x12, 0x54, 0x48, 0xbb,
0xb2, 0x5b, 0x4f,
       0xf4, 0x12, 0xa4, 0x9c, 0x73, 0x2d, 0xb2, 0xc8, 0xab, 0xc1, 0xb8, 0x58, 0x1b,
0xd7, 0x10, 0xdd
    };
    uint8_t digest_single[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t digest_multi[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Initialize the driver */
```

```
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate SHA-256 digests with single-part API */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), digest_single);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_single, hash, SHA256_HASH_SIZE));
 /* (2) Calculate SHA-256 digests with multi-part API */
    err = R_RSIP_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_SHA_Update(&g_rsip_ctrl, message, sizeof(message));
    assert(FSP_SUCCESS == err);
    err = R_RSIP_SHA_Finish(&g_rsip_ctrl, digest_multi);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_multi, hash, SHA256_HASH_SIZE));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

HMAC Example

This is an example of HMAC signature/verification.

```
#define HMAC_MESSAGE_LEN (34)
#define HMAC_HASH_LEN (32)
void r_rsip_example_hmac ()
{
 fsp_err_t err;
 static const uint8_t message[HMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x53, 0x61, 0x6D, 0x70, 0x6C, 0x65, 0x20, 0x6D,
       0x65, 0x73, 0x73, 0x61, 0x67, 0x65, 0x20, 0x66,
```

```
       0x6F, 0x72, 0x20, 0x6B, 0x65, 0x79, 0x6C, 0x65,
       0x6E, 0x3C, 0x62, 0x6C, 0x6F, 0x63, 0x6B, 0x6C,
       0x65, 0x6E
    };
    uint8_t hash_1[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t hash_2[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a HMAC-SHA 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to finish */
    err = R_RSIP_HMAC_Compute(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1);
    assert(FSP_SUCCESS == err);
 /* (2) Verify HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to verify */
    err = R_RSIP_HMAC_Verify(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* (3) Calculate HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
```

```
 /* Inputs message */
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC generation */
    err = R_RSIP_HMAC_SignFinish(&g_rsip_ctrl, hash_2);
    assert(FSP_SUCCESS == err);
 /* (4) Verify HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC verification */
    err = R_RSIP_HMAC_VerifyFinish(&g_rsip_ctrl, hash_2, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare hash_1 and hash_2 */
    assert(0 == memcmp(hash_1, hash_2, HMAC_HASH_LEN));
}
```

PKI Example

This is an example that verifies the public key certificate, imports the peer's public key, and calculates the shared secret.

```
#define SHA256_HASH_SIZE (32)
#define SECP256R1_KEY_SIZE (32)
/* Simplified public key certificate structure */
typedef struct st_cert
{
    uint8_t public_key_param_qx[SECP256R1_KEY_SIZE];
    uint8_t public_key_param_qy[SECP256R1_KEY_SIZE];
```

```
} cert_t;
void r_rsip_example_pki ()
{
 fsp_err_t err;
 /* Prepare data area for generating certificate and shared secret */
    cert_t                cert;
    uint8_t               raw_public_key[SECP256R1_KEY_SIZE * 2]
BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t               hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t               signature[SECP256R1_KEY_SIZE * 2]
BSP_ALIGN_VARIABLE(4)      = {0};
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare wrapped key data area for CA (Virtually prepared for local validation) */
    uint8_t
wrapped_ca_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_ca_public_key_value
    };
    uint8_t
wrapped_ca_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_ca_private_key_value
    };
 /* Prepare wrapped key data area for peer (Virtually prepared for local validation)
*/
```

```
    uint8_t
wrapped_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_peer_public_key_value
    };
    uint8_t
wrapped_peer_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_peer_private_key_value
    };
 /* Prepare wrapped key data area for Local device */
    uint8_t
wrapped_local_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_local_public_key_value
    };
    uint8_t
wrapped_local_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_private_key =
    {
```

```
        .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_local_private_key_value
    };
 /* Prepare wrapped key data area for verified peer public key */
    uint8_t wrapped_verified_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
 RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_verified_peer_public_key =
    {
        .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_verified_peer_public_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate key pairs for CA, peer, and Local device */
 /* Generate a key pair of secp256r1 for CA (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_ca_public_key,
&wrapped_ca_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for peer (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_peer_public_key,
&wrapped_peer_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for Local device (Assume product with RSIP
driver) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_local_public_key,
&wrapped_local_private_key);
    assert(FSP_SUCCESS == err);
/* CA process :
 * In the following, generate certificate and signature of peer public key */
 /* Export raw peer public key (Qx|Qy) and store it in cert */
    err = R_RSIP_PublicKeyExport(&wrapped_peer_public_key, raw_public_key);
```

```
    assert(FSP_SUCCESS == err);
    memcpy(cert.public_key_param_qx, &raw_public_key[0], SECP256R1_KEY_SIZE);
    memcpy(cert.public_key_param_qy, &raw_public_key[SECP256R1_KEY_SIZE],
SECP256R1_KEY_SIZE);
 /* Generate the hash of cert and store it in hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, (uint8_t *) &cert,
sizeof(cert), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature of hash with CA private key and store it in signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_ca_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
/* Local device process :
 * In the following, verify the CA certificate, import peer public key, and generate
shared secret */
 /* Verify the signature of cert with CA public key */
    err = R_RSIP_PKI_ECDSA_CertVerify(&g_rsip_ctrl, &wrapped_ca_public_key, hash,
signature);
    assert(FSP_SUCCESS == err);
 /* Import verified peer public key with wrapped format */
    err = R_RSIP_PKI_CertKeyImport(&g_rsip_ctrl,
                                   (uint8_t *) &cert,
 sizeof(cert),
                                   cert.public_key_param_qx,
                                   SECP256R1_KEY_SIZE,
                                   cert.public_key_param_qy,
                                   SECP256R1_KEY_SIZE,
 RSIP_HASH_TYPE_SHA256,
                                   &wrapped_verified_peer_public_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the Local private key and the verified peer public
key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl,
                               &wrapped_local_private_key,
                               &wrapped_verified_peer_public_key,
```

```
                                &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

ECDH (with plain public key) Example

This is an example of calculating the shared secret using a plaintext key that has been received from somewhere else.

```
#define SECP256R1_KEY_SIZE (32)
#define COUNTER_BYTE_LEN (4)
#define FIXED_INFO_BYTE_LEN (16)
#define KDF_SHA256_DKM_BYTE_SIZE (RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER +
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256)
void r_rsip_example_ecdh ()
{
 fsp_err_t err;
 /* Plain public key data for ECC secp256r1 */
 static const uint8_t plain_public_key[SECP256R1_KEY_SIZE * 2] BSP_ALIGN_VARIABLE(4)
=
    {
       0x60, 0xfe, 0xd4, 0xba, 0x25, 0x5a, 0x9d, 0x31, 0xc9, 0x61, 0xeb, 0x74, 0xc6,
0x35, 0x6d, 0x68,
       0xc0, 0x49, 0xb8, 0x92, 0x3b, 0x61, 0xfa, 0x6c, 0xe6, 0x69, 0x62, 0x2e, 0x60,
0xf2, 0x9f, 0xb6,
       0x79, 0x03, 0xfe, 0x10, 0x08, 0xb8, 0xbc, 0x99, 0xa4, 0x1a, 0xe9, 0xe9, 0x56,
0x28, 0xbc, 0x64,
       0xf2, 0xf1, 0xb2, 0x0c, 0x2d, 0x7e, 0x9f, 0x51, 0x77, 0xa3, 0xc2, 0x94, 0xd4,
0x46, 0x22, 0x99,
    };
 /* Prepare data area for local key pair and shared secret */
    uint8_t
```

```
wrapped_local_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_local_public_key_value
    };
    uint8_t
wrapped_local_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_local_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm[KDF_SHA256_DKM_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm = (rsip_wrapped_dkm_t *) wrapped_dkm;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[RSIP_BYTE_SIZE_WRAPPED_IV];
 /* Input data for KDF-SHA operation */
    uint32_t             counter = 0;
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
```

```
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for Local device */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_local_public_key,
&wrapped_local_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the Local private key and the plain public key */
    err = R_RSIP_ECDH_PlainKeyAgree(&g_rsip_ctrl, &wrapped_local_private_key,
plain_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* In the following, Generate DKM (Derived Keying Material) by calculating KDF-
SHA[counter|Z|FixedInfo] as an example, where Z is shared secret obtained above */
    counter = 0x00000000;
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm);
```

```
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES from the DKM */
 /* Derive the AES-256 wrapped key from the 1st byte position of the DKM */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm, 0, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 1st byte position of the DKM */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   0,
                                   NULL,
                                   wrapped_iv);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

KDF-SHA Example

This is an example of One-Step Key Derivation (NIST SP 800-56C) using KDF-SHA.

```
#define COUNTER_BYTE_LEN (4)
#define FIXED_INFO_BYTE_LEN (16)
#define KDF_SHA256_DKM_BYTE_SIZE (RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER +
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256)
#define CONCAT_DKM_BYTE_SIZE (RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER +
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256 * 2)
#define KDF_SHA_TEST_GCM_BLOCK_LEN (16)
#define KDF_SHA_TEST_GCM_TEXT_LEN (31)
#define KDF_SHA_TEST_GCM_AAD_LEN (8)
#define KDF_SHA_TEST_GCM_TAG_LEN (16)
void r_rsip_example_kdf_sha ()
{
```

```
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_DKM_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_SHA256_DKM_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
```

```
    };
    uint8_t wrapped_iv[RSIP_BYTE_SIZE_WRAPPED_IV];
 /* Input data for KDF-SHA operation */
    uint32_t             counter = 0;
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[KDF_SHA_TEST_GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[KDF_SHA_TEST_GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    };
    uint8_t tag[KDF_SHA_TEST_GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[KDF_SHA_TEST_GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) =
{0};
    uint8_t plain_calculated[KDF_SHA_TEST_GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) =
{0};
    uint32_t output_length    = 0;
    uint32_t output_length_tmp = 0;
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
```

```
 /* Generate a key pair of secp256r1 to generate the shared secret */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* One-Step Key Derivation (NIST SP 800-56C)
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter = 0x00000000;
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM1 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
/* In the following, calculate K(2) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter++;
 /* Initialize KDF-SHA operation */
```

```
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM2 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm2);
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_DKM_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In One-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
DKM1 */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
```

```
                                    16,
                                    NULL,
                                    wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, KDF_SHA_TEST_GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext and output ciphertext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, KDF_SHA_TEST_GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, KDF_SHA_TEST_GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
```

```
                                cipher_calculated,
                                KDF_SHA_TEST_GCM_TEXT_LEN,
                                plain_calculated,
                                &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl,
                                plain_calculated + output_length,
                                &output_length_tmp,
                                tag,
                                KDF_SHA_TEST_GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, KDF_SHA_TEST_GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

KDF-HMAC Example

This is an example of Two-Step Key Derivation (NIST SP 800-56C) using KDF-HMAC.

```
#define INITIAL_VALUE_BYTE_LEN (32)
#define FIXED_INFO_BYTE_LEN (16)
#define HMAC_SHA256_KEY_LEN (32)
#define KDF_HMAC_SHA256_DKM_BYTE_SIZE (RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER +
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256)
#define CONCAT_DKM_BYTE_SIZE (RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER +
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256 * \
  2)
#define KDF_HMAC_TEST_GCM_BLOCK_LEN (16)
#define KDF_HMAC_TEST_GCM_TEXT_LEN (31)
#define KDF_HMAC_TEST_GCM_AAD_LEN (8)
```

```
#define KDF_HMAC_TEST_GCM_TAG_LEN (16)
void r_rsip_example_kdf_hmac ()
{
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped key as a salt */

uint8_t            wrapped_salt_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_salt =
    {
       .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_salt_value
    };
 /* Prepare data area for wrapped KDK (Key Derivation Key) */
```

```
uint8_t            wrapped_kdk_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_KDF_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_kdk =
    {
       .type = RSIP_KEY_TYPE_KDF_HMAC_SHA256, .p_value = &wrapped_kdk_value
    };
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_DKM_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_HMAC_SHA256_DKM_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[RSIP_BYTE_SIZE_WRAPPED_IV];
 /* Input data for KDF-HMAC operation */
 static const uint8_t initial_value[INITIAL_VALUE_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa, 0x55, 0xaa,
       0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa, 0x55, 0xaa,
    };
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
```

```
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[KDF_HMAC_TEST_GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[KDF_HMAC_TEST_GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    };
    uint8_t tag[KDF_HMAC_TEST_GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[KDF_HMAC_TEST_GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t plain_calculated[KDF_HMAC_TEST_GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) =
{0};
    uint32_t output_length    = 0;
    uint32_t output_length_tmp = 0;
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
 /* Generate a key pair of secp256r1 to generate the shared secret */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* Two-Step Key Derivation (NIST SP 800-56C) Step 1 : Randomness Extraction
```

```
 * Generate KDK (Key Derivation Key) */
/* In the following, calculate KDK = KDF-HMAC[salt, Z], where Z is shared secret */
 /* Generate a HMAC-SHA 256 key as a salt */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_salt);
    assert(FSP_SUCCESS == err);
 /* Initialize KDF-HMAC operation (Key setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_salt);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_HMAC_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped DKM */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
 /* Convert the wrapped DKM to wrapped HMAC-SHA256 key, which is called KDK (Key
Derivation Key) */
    err = R_RSIP_KDF_HMAC_DKMKeyImport(&g_rsip_ctrl, p_wrapped_dkm1,
HMAC_SHA256_KEY_LEN, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
/* Two-Step Key Derivation (NIST SP 800-56C) Step 2 : Key Expansion
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-HMAC[KDK, K(0)|FixedInfo], where K(0) = IV
*/
 /* Initialize KDF-HMAC operation (KDK setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
 /* Input K(0) = IV as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, initial_value,
INITIAL_VALUE_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input FixedInfo as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped DKM */
```

```
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
/* In the following, calculate K(2) = KDF-HMAC[KDK, K(1)|FixedInfo] */
 /* Initialize KDF-HMAC operation (KDK setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
 /* Input K(1) as a ciphertext */
    err = R_RSIP_KDF_HMAC_DKMUpdate(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
 /* Input FixedInfo as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped DKM */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm2);
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_DKM_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In Two-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
DKM1 */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
```

```
                                    16,
                                    NULL,
                                    wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, KDF_HMAC_TEST_GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext and output ciphertext */
    err =
 R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, KDF_HMAC_TEST_GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, KDF_HMAC_TEST_GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
```

```
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                 cipher_calculated,
                                 KDF_HMAC_TEST_GCM_TEXT_LEN,
                                 plain_calculated,
                                 &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl,
                                 plain_calculated + output_length,
                                 &output_length_tmp,
                                 tag,
                                 KDF_HMAC_TEST_GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, KDF_HMAC_TEST_GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rsip_sha_handle_t
struct	rsip_hmac_handle_t
struct	rsip_kdf_sha_handle_t
struct	rsip_kdf_hmac_handle_t
struct	rsip_verified_cert_info_t
struct	rsip_wrapped_secret_t
struct	rsip_instance_ctrl_t

Macros

#define	RSIP_BYTE_SIZE_PLAIN_KEY(key_type)

#define	RSIP_BYTE_SIZE_ENCRYPTED_KEY(key_type)
#define	RSIP_BYTE_SIZE_WRAPPED_KEY(key_type)

Enumerations

enum	rsip_byte_size_wrapped_key_t
enum	rsip_byte_size_encrypted_key_t
enum	rsip_byte_size_wrapped_dkm_t
enum	rsip_byte_size_wrapped_iv_t

Data Structure Documentation

◆ rsip_sha_handle_t

struct rsip_sha_handle_t
Working area for SHA functions. DO NOT MODIFY.

◆ rsip_hmac_handle_t

struct rsip_hmac_handle_t
Working area for HMAC functions. DO NOT MODIFY.

◆ rsip_kdf_sha_handle_t

struct rsip_kdf_sha_handle_t
Working area for KDF SHA functions. DO NOT MODIFY.

◆ rsip_kdf_hmac_handle_t

struct rsip_kdf_hmac_handle_t
Working area for KDF HMAC functions. DO NOT MODIFY.

◆ rsip_verified_cert_info_t

struct rsip_verified_cert_info_t
Verified certificate information

◆ rsip_wrapped_secret_t

struct rsip_wrapped_secret_t
Wrapped ECDH secret structure for ECDH and KDF APIs.

◆ rsip_instance_ctrl_t

struct rsip_instance_ctrl_t
RSIP private control block. DO NOT MODIFY. Initialization occurs when R_RSIP_Open() is called.

Macro Definition Documentation

◆ RSIP_BYTE_SIZE_PLAIN_KEY

#define RSIP_BYTE_SIZE_PLAIN_KEY (key_type)
Returns the plain key size formatted for RSIP driver of the key type.

◆ RSIP_BYTE_SIZE_ENCRYPTED_KEY

#define RSIP_BYTE_SIZE_ENCRYPTED_KEY (key_type)
Returns the encrypted key size of the key type.

◆ RSIP_BYTE_SIZE_WRAPPED_KEY

#define RSIP_BYTE_SIZE_WRAPPED_KEY (key_type)
Returns the wrapped key size of the key type.

Enumeration Type Documentation

◆ rsip_byte_size_wrapped_key_t

enum rsip_byte_size_wrapped_key_t	
Byte size of wrapped key	
Enumerator	
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_128	AES-128.
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_192	AES-192.
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256	AES-256.
RSIP_BYTE_SIZE_WRAPPED_KEY_XTS_AES_128	XTS-AES-128.
RSIP_BYTE_SIZE_WRAPPED_KEY_XTS_AES_256	XTS-AES-256.
RSIP_BYTE_SIZE_WRAPPED_KEY_CHACHA20	ChaCha20.
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R	

1_PUBLIC	secp256r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP384R1_PUBLIC	secp384r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP521R1_PUBLIC	secp521r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_PUBLIC	secp256k1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP256R1_PUBLIC	brainpoolP256r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP384R1_PUBLIC	brainpoolP384r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP512R1_PUBLIC	brainpoolP512r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_EDWARDS25519_PUBLIC	edwards25519 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_PRIVATE	secp256r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP384R1_PRIVATE	secp384r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP521R1_PRIVATE	secp521r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_PRIVATE	secp256k1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP256R1_PRIVATE	brainpoolP256r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP384R1_PRIVATE	brainpoolP384r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP512R1_PRIVATE	brainpoolP512r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_EDWARDS25519_PRIVATE	edwards25519 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_1024_PUBLIC	RSA-1024 public key.
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_2048_PUBLIC	RSA-2048 public key.
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_3072_PUBLIC	RSA-3072 public key.
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_4096_PUBLIC	RSA-4096 public key.
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_1024_PRIVATE	RSA-1024 private key.

RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_2048_PRIVATE	RSA-2048 private key.
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_3072_PRIVATE	RSA-3072 private key.
RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_4096_PRIVATE	RSA-4096 private key.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA1	HMAC-SHA1.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA224	HMAC-SHA224.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA256	HMAC-SHA256.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA384	HMAC-SHA384.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA512	HMAC-SHA512.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA512_224	HMAC-SHA512/224.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA512_256	HMAC-SHA512/256.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA3_224	HMAC-SHA3-224.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA3_256	HMAC-SHA3-256.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA3_384	HMAC-SHA3-384.
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA3_512	HMAC-SHA3-512.
RSIP_BYTE_SIZE_WRAPPED_KEY_KDF_HMAC_SHA256	KDF HMAC-SHA256.
RSIP_BYTE_SIZE_WRAPPED_KEY_KDF_HMAC_SHA384	KDF HMAC-SHA384.
RSIP_BYTE_SIZE_WRAPPED_KEY_KDF_HMAC_SHA512	KDF HMAC-SHA512.
RSIP_BYTE_SIZE_WRAPPED_KEY_KUK	Key Update Key (KUK)

◆ rsip_byte_size_encrypted_key_t

enum rsip_byte_size_encrypted_key_t	
Byte size of encrypted key	
Enumerator	
RSIP_BYTE_SIZE_ENCRYPTED_KEY_AES_128	AES-128.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_AES_192	AES-192.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_AES_256	AES-256.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_XTS_AES_128	XTS-AES-128.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_XTS_AES_256	XTS-AES-256.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_CHACHA20	ChaCha20.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256R1_PUBLIC	secp256r1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP384R1_PUBLIC	secp384r1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP521R1_PUBLIC	secp521r1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256K1_PUBLIC	secp256k1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP256R1_PUBLIC	brainpoolP256r1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP384R1_PUBLIC	brainpoolP384r1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP512R1_PUBLIC	brainpoolP512r1 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_EDWARDS25519_PUBLIC	edwards25519 public key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256R1_PRIVATE	secp256r1 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP384R1_PRIVATE	secp384r1 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP521R1_PRIVATE	secp521r1 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256K1_PRIVATE	secp256k1 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPO	brainpoolP256r1 private key

OLP256R1_PRIVATE	
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP384R1_PRIVATE	brainpoolP384r1 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP512R1_PRIVATE	brainpoolP512r1 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_EDWARDS25519_PRIVATE	edwards25519 private key
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_1024_PUBLIC	RSA-1024 public key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_2048_PUBLIC	RSA-2048 public key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_3072_PUBLIC	RSA-3072 public key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_4096_PUBLIC	RSA-4096 public key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_1024_PRIVATE	RSA-1024 private key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_2048_PRIVATE	RSA-2048 private key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_3072_PRIVATE	RSA-3072 private key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_4096_PRIVATE	RSA-4096 private key.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA1	HMAC-SHA1.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA224	HMAC-SHA224.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA256	HMAC-SHA256.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA384	HMAC-SHA384.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA512	HMAC-SHA512.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA512_224	HMAC-SHA512/224.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA512_256	HMAC-SHA512/256.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA3_224	HMAC-SHA3-224.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA3_256	HMAC-SHA3-256.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA3_	HMAC-SHA3-384.

384	
RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA3_512	HMAC-SHA3-512.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_KDF_HMAC_SHA256	KDF HMAC-SHA256.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_KDF_HMAC_SHA384	KDF HMAC-SHA384.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_KDF_HMAC_SHA512	KDF HMAC-SHA512.
RSIP_BYTE_SIZE_ENCRYPTED_KEY_KUK	Key Update Key (KUK)

◆ rsip_byte_size_wrapped_dkm_t

enum rsip_byte_size_wrapped_dkm_t	
Byte size of wrapped key	
Enumerator	
RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER	Header.
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256	A block of SHA-256.
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA384	A block of SHA-384.
RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA512	A block of SHA-512.

◆ rsip_byte_size_wrapped_iv_t

enum rsip_byte_size_wrapped_iv_t	
Byte size of wrapped IV or nonce	
Enumerator	
RSIP_BYTE_SIZE_WRAPPED_IV	Wrapped IV size for all algorithms.

Function Documentation

◆ R_RSIP_Open()

fsp_err_t R_RSIP_Open (rsip_ctrl_t *const *p_ctrl*, rsip_cfg_t const *const *p_cfg*)

Enables use of Renesas Secure IP functionality.

Implements rsip_api_t::open.

State transition

This API can only be executed in **STATE_INITIAL**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Internal key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Hardware initialization is failed.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware fault is detected.

Note

This module does not provide an optional feature to disable TRNG initialization.

◆ R_RSIP_Close()

fsp_err_t R_RSIP_Close (rsip_ctrl_t *const *p_ctrl*)

Disables use of Renesas Secure IP functionality.

Implements rsip_api_t::close.

State transition

This API can be executed in **except STATE_INITIAL**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_INITIAL
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RandomNumberGenerate()

fsp_err_t R_RSIP_RandomNumberGenerate (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_random*)

Generates a 128-bit random number.

Implements rsip_api_t::randomNumberGenerate.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KeyGenerate()

fsp_err_t R_RSIP_KeyGenerate (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t *const *p_wrapped_key*)

Generates a wrapped symmetric key from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Implements rsip_api_t::keyGenerate.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FAIL	A generated AES-XTS key is illegal (Data Key == Tweak Key).
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KeyPairGenerate()

fsp_err_t R_RSIP_KeyPairGenerate (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t *const *p_wrapped_public_key*, rsip_wrapped_key_t *const *p_wrapped_private_key*)

Generates a wrapped asymmetric key pair from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Implements rsip_api_t::keyPairGenerate.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Here, p_wrapped_public_key->type and p_wrapped_private_key->type must match. For example, if p_wrapped_public_key->type is RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, then p_wrapped_public_key->type must be RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_EncryptedKeyWrap()

fsp_err_t R_RSIP_EncryptedKeyWrap (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_key_update_key*, void const *const *p_initial_vector*, void const *const *p_encrypted_key*, rsip_wrapped_key_t *const *p_wrapped_key*)

Decrypts an encrypted user key with Key Update Key (KUK) and wrap it with the Hardware Unique Key (HUK).

Implements rsip_api_t::encryptedKeyWrap.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input KUK is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RFC3394_KeyWrap()

fsp_err_t R_RSIP_RFC3394_KeyWrap (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_kek*, rsip_wrapped_key_t const *const *p_wrapped_target_key*, uint8_t *const *p_rfc3394_wrapped_target_key*)

This function provides Key Wrap algorithm compliant with RFC3394. Using p_wrapped_kek to wrap p_wrapped_target_key, and output the result to p_rfc3394_wrapped_target_key.

Implements rsip_api_t::rfc3394_KeyWrap.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_ARGUMENT	Input key type or mode is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RFC3394_KeyUnwrap()

fsp_err_t R_RSIP_RFC3394_KeyUnwrap (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_kek*, uint8_t const *const *p_rfc3394_wrapped_target_key*, rsip_wrapped_key_t *const *p_wrapped_target_key*)

This function provides Key Unwrap algorithm compliant with RFC3394. Using p_wrapped_kek to unwrap p_rfc3394_wrapped_target_key, and output the result to p_wrapped_target_key.

Implements rsip_api_t::rfc3394_KeyUnwrap.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_ARGUMENT	Input key type or mode is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_PublicKeyExport()

fsp_err_t R_RSIP_PublicKeyExport (rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t *const *p_raw_public_key*)

Exports public key parameters from a wrapped key.

Implements rsip_api_t::publicKeyExport.

Relative position of each elements in p_raw_public_key is shown in below:

- ECC (RSIP_KEY_TYPE_ECC_*) : Qx placed first and Qy placed after that.

bit length	Qx	Qy
256	0	32
384	0	48
512	0	64
521	0	66

- RSA (RSIP_KEY_TYPE_RSA_*) : n placed first and e placed after that.

modulus	n	e
1024	0	128
2048	0	256
3072	0	384
4096	0	512

State transition

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.

◆ R_RSIP_AES_Cipher_Init()

fsp_err_t R_RSIP_AES_Cipher_Init (rsip_ctrl_t *const *p_ctrl*, rsip_aes_cipher_mode_t const *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_initial_vector*)

Starts AES cipher operation in confidentiality mode (ECB/CBC/CTR) or XTS mode.

Implements rsip_api_t::aesCipherInit.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument mode must be the following:

- AES key
  - RSIP_AES_CIPHER_MODE_ECB_ENC, RSIP_AES_CIPHER_MODE_ECB_DEC
  - RSIP_AES_CIPHER_MODE_CBC_ENC, RSIP_AES_CIPHER_MODE_CBC_DEC
  - RSIP_AES_CIPHER_MODE_CBC_ENC_WRAPPED_IV, RSIP_AES_CIPHER_MODE_CBC_DEC_WRAPPED_IV
  - RSIP_AES_CIPHER_MODE_CTR
- XTS-AES key
  - RSIP_AES_CIPHER_MODE_XTS_ENC, RSIP_AES_CIPHER_MODE_XTS_DEC

Argument p_initial_vector must be the following:

- [ECB] Not used
- [CBC] Raw initial vector
- [CTR] Raw nonce
- [XTS] Raw initial vector

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_AES
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_ARGUMENT	Input key type or mode is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FAIL	Input AES-XTS key value is illegal (Data Key == Tweak Key).
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_Cipher_Update()

fsp_err_t R_RSIP_AES_Cipher_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint8_t *const *p_output*, uint32_t const *length*)

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::aesCipherUpdate.

Conditions

Argument length must be the following:

- [ECB][CBC][CTR] 0 or a multiple of 16.
- [XTS] 0 or greater than or equal to 16.

Output length

Output length to p_output is length.

State transition

This API can only be executed in **STATE_AES**, and does not cause any state transitions.

In XTS mode, if once an integer other than 0 or a multiple of 16 is input, this API can no longer be called.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	Input length is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_Cipher_Finish()

fsp_err_t R_RSIP_AES_Cipher_Finish (rsip_ctrl_t *const *p_ctrl*)

Finishes AES operation.

Implements rsip_api_t::aesCipherFinish.

State transition

This API can only be executed in **STATE_AES**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_AEAD_Init()

fsp_err_t R_RSIP_AES_AEAD_Init (rsip_ctrl_t *const *p_ctrl*, rsip_aes_aead_mode_t *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_nonce*, uint32_t const *nonce_length*)

Starts AES AEAD function.

Implements rsip_api_t::aesAeadInit.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument mode accepts any member of enumeration rsip_aes_aead_mode_t.

Argument nonce_length must be the following:

- [GCM] Any length is accepted, but 12 bytes is generally recommended.
- [CCM] 7 to 13 bytes.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_AES_AEAD
Others	No change

The next callable API functions in STATE_AES_AEAD are as below.

- [GCM] R_RSIP_AES_AEAD_AADUpdate(), R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish() (encryption), R_RSIP_AES_AEAD_Verify() (decryption)
- [CCM] R_RSIP_AES_AEAD_LengthsSet()

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	nonce_length is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_ARGUMENT	Input key type is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_AEAD_LengthsSet()

fsp_err_t R_RSIP_AES_AEAD_LengthsSet (rsip_ctrl_t *const *p_ctrl*, uint32_t const *total_aad_length*, uint32_t const *total_text_length*, uint32_t const *tag_length*)

Sets text and tag lengths for CCM mode.

Implements rsip_api_t::aesAeadLengthsSet.

Conditions

Argument total_aad_length must be equal to the length of AAD and must be 110 or less.

Argument total_test_length must be equal to the length of the plaintext or ciphertext.

Argument tag_length must be 4, 6, 8, 10, 12, 14, or 16.

State transition

This API can only be executed in **STATE_AES_AEAD**, and does not cause any state transitions.

The next callable API functions in STATE_AES_AEAD are as below.

- [CCM] R_RSIP_AES_AEAD_AADUpdate()

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_SIZE	Input length is illegal.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

In GCM mode, this API must NOT be called. If called, FSP_ERR_INVALID_STATE will be returned. AAD length and text length are indeterminate and output tag length is fixed to 16 bytes.

◆ R_RSIP_AES_AEAD_AADUpdate()

fsp_err_t R_RSIP_AES_AEAD_AADUpdate (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_aad*, uint32_t const *aad_length*)

Inputs Additional Authentication Data (AAD).

Implements rsip_api_t::aesAeadAadUpdate.

State transition

This API can only be executed in **STATE_AES_AEAD**, and does not cause any state transitions.

- [GCM] R_RSIP_AES_AEAD_AADUpdate(), R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish() (encryption), R_RSIP_AES_AEAD_Verify() (decryption)
- [CCM] R_RSIP_AES_AEAD_AADUpdate() (AAD input is not completed), R_RSIP_AES_AEAD_Update() (AAD input is completed)

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	aad_length is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.

◆ R_RSIP_AES_AEAD_Update()

fsp_err_t R_RSIP_AES_AEAD_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint32_t const *input_length*, uint8_t *const *p_output*, uint32_t *const *p_output_length*)

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::aesAeadUpdate.

Output length

Output length to p_output (p_output_length) is calculated text that has not yet been output in multiple of 16 bytes.

State transition

This API can only be executed in **STATE_AES_AEAD**, and does not cause any state transitions.

- [GCM] R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish() (encryption), R_RSIP_AES_AEAD_Verify() (decryption)
- [CCM] R_RSIP_AES_AEAD_Update() (text input is not completed), R_RSIP_AES_AEAD_Finish() (text input is completed, encryption), R_RSIP_AES_AEAD_Verify() (text input is completed, decryption)

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_SIZE	Input length is illegal.
FSP_ERR_INVALID_STATE	Internal state is illegal.

Note

In GCM mode, if this API is skipped, GMAC will be calculated. For detailed usage, refer to example code.

◆ R_RSIP_AES_AEAD_Finish()

fsp_err_t R_RSIP_AES_AEAD_Finish (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t *const *p_tag*)

Finalizes an AES AEAD encryption.

Implements rsip_api_t::aesAeadFinish.

Output length

Output length to p_output (p_output_length) is the remaining calculated text length.

Output length to p_tag as below.

- [GCM] 16 bytes.
- [CCM] Input value as tag_length in R_RSIP_AES_AEAD_LengthsSet().

State transition

This API can only be executed in **STATE_AES_AEAD**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_AEAD_Verify()

fsp_err_t R_RSIP_AES_AEAD_Verify (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t const *const *p_tag*, uint32_t const *tag_length*)

Finalizes an AES AEAD decryption.

If there is 16-byte fractional data indicated by the total data length of the value of p_cipher that was input by R_RSIP_AES_GCM_DecryptUpdate(), this API will output the result of decrypting that fractional data to p_cipher. Here, the portion that does not reach 16 bytes will be padded with zeros.

Implements rsip_api_t::aesAeadVerify.

Conditions

Argument tag_length must be as below.

- [GCM] 1 to 16 bytes.
- [CCM] Input value as tag_length in R_RSIP_AES_AEAD_LengthsSet().

Output length

Output length to p_output (p_output_length) is the remaining calculated text length.

State transition

This API can only be executed in **STATE_AES_AEAD**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	tag_length is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	Authentication is failed.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_MAC_Init()

fsp_err_t R_RSIP_AES_MAC_Init (rsip_ctrl_t *const *p_ctrl*, rsip_aes_mac_mode_t const *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*)

Starts an AES MAC operation.

Implements rsip_api_t::aesMacInit.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument mode accepts any member of enumeration rsip_aes_aead_mode_t.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_AES_MAC
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Note

To calculate AES-GMAC, please use not R_RSIP_MAC_*() but R_RSIP_AEAD_*(). For detailed usage, refer to example code.

◆ R_RSIP_AES_MAC_Update()

fsp_err_t R_RSIP_AES_MAC_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length*)

Inputs message.

Implements rsip_api_t::aesMacUpdate.

Inside this function, the data that is input by the user is buffered until the input value of p_message exceeds 16 bytes. If the input value, p_message, is not a multiple of 16 bytes, it will be padded within the function.

State transition

This API can only be executed in **STATE_AES_MAC**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_AES_MAC_SignFinish()

fsp_err_t R_RSIP_AES_MAC_SignFinish (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_mac*)

Outputs AES MAC.

Implements rsip_api_t::aesMacSignFinish.

Output length

Output length to p_mac is 16 bytes.

State transition

This API can only be executed in **STATE_AES_MAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_AES_MAC_VerifyFinish()

fsp_err_t R_RSIP_AES_MAC_VerifyFinish (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*, uint32_t const *mac_length*)

Verifies AES MAC.

Implements rsip_api_t::aesMacVerifyFinish.

Conditions

Argument mac_length must be as below.

- [CMAC] 2 to 16 bytes.

State transition

This API can only be executed in **STATE_AES_MAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	mac_length is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error.
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	Authentication is failed.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ChaCha20_Init()

fsp_err_t R_RSIP_ChaCha20_Init (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_nonce*, uint32_t const *counter*)

Prepares a ChaCha20 function.

Implements rsip_api_t::chacha20Init.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_CHACHA
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ChaCha20_Update()

fsp_err_t R_RSIP_ChaCha20_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint32_t const *input_length*, uint8_t *const *p_output*, uint32_t *const *p_output_length*)

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::chacha20Update.

Output length

Output length to p_output (p_output_length) is calculated text that has not yet been output in multiple of 64 bytes.

State transition

This API can only be executed in **RSIP_STATE_CHACHA**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_ChaCha20_Finish()

fsp_err_t R_RSIP_ChaCha20_Finish (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*)

Finalize ChaCha20 operation.

Implements rsip_api_t::chacha20Finish.

Output length

Output length to p_output (p_output_length) is the remaining calculated text length.

State transition

This API can only be executed in **RSIP_STATE_CHACHA**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ChaCha20_Poly1305_Init()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Init (rsip_ctrl_t *const *p_ctrl*, rsip_chacha_poly_mode_t const *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_nonce*, uint32_t const *nonce_length*)

Prepares a ChaCha20-Poly1305 function.

Implements rsip_api_t::chacha20Poly1305Init.

Conditions

Key type of p_wrapped_key must be RSIP_KEY_TYPE_CHACHA20.

Argument mode accepts any member of enumeration rsip_chacha_poly_mode_t.

Argument nonce_length must be 12 bytes.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_CHACHA_POLY
Others	No change

The next callable API functions in STATE_CHACHA_POLY are as below.

- R_RSIP_ChaCha20_Poly1305_AADUpdate(), R_RSIP_ChaCha20_Poly1305_Update(), R_RSIP_ChaCha20_Poly1305_Finish() (encryption), R_RSIP_ChaCha20_Poly1305_Verify() (decryption)

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	nonce_length is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_ARGUMENT	Input mode is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ChaCha20_Poly1305_AADUpdate()

fsp_err_t R_RSIP_ChaCha20_Poly1305_AADUpdate (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_aad*, uint32_t const *aad_length*)

Inputs Additional Authentication Data (AAD).

Implements rsip_api_t::chacha20Poly1305AadUpdate.

State transition

This API can only be executed in **STATE_CHACHA_POLY**, and does not cause any state transitions.

- R_RSIP_ChaCha20_Poly1305_AADUpdate(), R_RSIP_ChaCha20_Poly1305_Update(), R_RSIP_ChaCha20_Poly1305_Finish() (encryption), R_RSIP_ChaCha20_Poly1305_Verify() (decryption)

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_ChaCha20_Poly1305_Update()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint32_t const *input_length*, uint8_t *const *p_output*, uint32_t *const *p_output_length*)

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::chacha20Poly1305Update.

Output length

Output length to p_output (p_output_length) is calculated text that has not yet been output in multiple of 64 bytes.

At least one byte of data must be input with this API before calling R_RSIP_ChaCha20_Poly1305_Finish() or R_RSIP_ChaCha20_Poly1305_Verify().

State transition

This API can only be executed in **STATE_CHACHA_POLY**, and does not cause any state transitions.

- R_RSIP_ChaCha20_Poly1305_Update(), R_RSIP_ChaCha20_Poly1305_Finish() (encryption), R_RSIP_ChaCha20_Poly1305_Verify() (decryption)

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ChaCha20_Poly1305_Finish()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Finish (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t *const *p_tag*)

Finalizes a ChaCha20-Poly1305 encryption.

Implements rsip_api_t::chacha20Poly1305Finish.

Output length

Output length to p_output (p_output_length) is the remaining calculated text length.

Output length to p_tag is 16 bytes.

State transition

This API can only be executed in **STATE_CHACHA_POLY**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
FSP_ERR_INVALID_SIZE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	Input text length is illegal. (zero byte)
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ChaCha20_Poly1305_Verify()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Verify (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t const *const *p_tag*, uint32_t const *tag_length*)

Finalizes a ChaCha20-Poly1305 decryption.

Implements rsip_api_t::chacha20Poly1305Verify.

Conditions

Argument tag_length must be 16 bytes.

Output length

Output length to p_output (p_output_length) is the remaining calculated text length.

State transition

This API can only be executed in **STATE_CHACHA_POLY**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	tag_length is illegal.
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	Authentication is failed.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_xDLMS_InitiateRequest_Decrypt()

fsp_err_t R_RSIP_xDLMS_InitiateRequest_Decrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_apdu*, uint8_t const *const *p_nonce*, uint8_t const *const *p_aad*, uint32_t const *aad_length*, rsip_wrapped_key_t *const *p_dedicated_key*, rsip_dlms_apdu_data_t *const *p_apdu_data*, rsip_dlms_ret_t *const *p_dlms_ret*)

Decrypts and Verifies ciphered InitiateRequest message, and wraps a Dedicated Key embedded in the message.

Implements rsip_api_t::xdlmsInitiateRequestDecrypt.

Conditions

Decrypted InitiateRequest message consists of the following:
Part 1 and Part 2 are obtained as p_apdu_data (rsip_dlms_apdu_data_t structure), and Dedicated Key is obtained as p_dedicated_key (wrapped key).

Decrypted InitiateRequest message		
Part 1 (The first 3 bytes of the message, consisting of APDU Choice, Key Flag, and Key Length)	Dedicated Key (AES-128 or AES-256 key)	Part 2 (Remaining message; Maximum 14 bytes)

Key type of p_wrapped_key must be one of the following and this must match Security Suite specified in SC (Security Control) byte:

- RSIP_KEY_TYPE_AES_128
- RSIP_KEY_TYPE_AES_256

The p_value of the p_dedicated_key must point to a buffer of size RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256).
The type of the p_dedicated_key will be set to either RSIP_KEY_TYPE_AES_128 or RSIP_KEY_TYPE_AES_256 as per InitiateRequest message.

If the return value is FSP_SUCCESS, also refer to p_dlms_ret to check which result was returned:

- RSIP_DLMS_RET_SUCCESS : The process completed successfully, and both the Dedicated Key and the decrypted APDU data (p_apdu_data) were output.
- RSIP_DLMS_RET_NO_DEDICATED_KEY : The process completed successfully, but the Dedicated Key was not present within the InitiateRequest message. Only the decrypted APDU data was output.

If the return value is FSP_ERR_CRYPTO_RSIP_DLMS_ERROR, also refer to p_dlms_ret to identify the more specific error.

If the return value is neither FSP_SUCCESS nor FSP_ERR_CRYPTO_RSIP_DLMS_ERROR, p_dlms_ret will be indeterminate.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination. Refer also to the

	argument p_dlms_ret.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	Authentication is failed.
FSP_ERR_CRYPTO_RSIP_DLMS_ERROR	An error occurred during DLMS processing. Refer to the argument p_dlms_ret for more details.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ECDSA_Sign()

fsp_err_t R_RSIP_ECDSA_Sign (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_hash*, uint8_t *const *p_signature*)

Generates an ECDSA signature.

Implements rsip_api_t::ecdsaSign.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length.

For secp521r1 operation, the length of p_hash must be set to 64 bytes.

For secp521r1 operation, the length of the argument p_signature must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

Data Format for secp521r1 (132 byte)			
zero padding (7 bit)	signature r (521 bit)	zero padding (7 bit)	signature s (521 bit)

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	• Input parameter is illegal. • Signature generation is failed.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ECDSA_Verify()

fsp_err_t R_RSIP_ECDSA_Verify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature*)

Verifies an ECDSA signature.

Implements rsip_api_t::ecdsaVerify.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length.

For secp521r1 operation, the length of p_hash must be set to 64 bytes.

For secp521r1 operation, the length of the argument p_signature must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

Data Format for secp521r1 (132 byte)			
zero padding (7 bit)	signature r (521 bit)	zero padding (7 bit)	signature s (521 bit)

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	• Input parameter is illegal. • Signature verification is failed.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_PKI_ECDSA_CertVerify()

fsp_err_t R_RSIP_PKI_ECDSA_CertVerify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature*)

Verifies a public key certificate with ECDSA.

Implements rsip_api_t::pkiEcdsaCertVerify.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length.

For secp521r1 operation, the length of p_hash must be set to 64 bytes.

For secp521r1 operation, the length of the argument p_signature must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

Data Format for secp521r1 (132 byte)			
zero padding (7 bit)	signature r (521 bit)	zero padding (7 bit)	signature s (521 bit)

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	• Input parameter is illegal. • Signature verification is failed.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_PureEdDSA_Sign()

fsp_err_t R_RSIP_PureEdDSA_Sign (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_message*, uint64_t const *message_length*, uint8_t *const *p_signature*)

Generates an EdDSA signature.

Implements rsip_api_t::eddsaSign.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_SIZE	Input length is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_PureEdDSA_Verify()

fsp_err_t R_RSIP_PureEdDSA_Verify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_message*, uint64_t const *message_length*, uint8_t const *const *p_signature*)

Verifies an EdDSA signature.

Implements rsip_api_t::eddsaVerify.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_INVALID_SIZE	Input length is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ECDH_KeyAgree()

fsp_err_t R_RSIP_ECDH_KeyAgree (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_wrapped_secret_t *const *p_wrapped_secret*)

Computes ECDH secret with wrapped private key and wrapped public key.

Implements rsip_api_t::ecdhKeyAgree.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_ECDH_PlainKeyAgree()

fsp_err_t R_RSIP_ECDH_PlainKeyAgree (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_plain_public_key*, rsip_wrapped_secret_t *const *p_wrapped_secret*)

Computes ECDH secret with wrapped private key and plain public key.

Implements rsip_api_t::ecdhPlainKeyAgree.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

For secp521r1 operation, the length of the argument p_plain_public_key must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

Data Format for secp521r1 (132 byte)			
zero padding (7 bit)	public_key Qx (521 bit)	zero padding (7 bit)	public_key Qy (521 bit)

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Attention

Unauthenticated public key of Diffie-Hellman key exchange generally carry the risk of man-in-the-middle (MITM) attack. Please use R_RSIP_ECDH_KeyAgree() or implement signature verification whenever possible.

◆ R_RSIP_KDF_SHA_Init()

fsp_err_t R_RSIP_KDF_SHA_Init (rsip_ctrl_t *const *p_ctrl*, rsip_hash_type_t const *hash_type*)

Prepares a SHA generation.

Implements rsip_api_t::kdfShaInit.

Conditions

Argument hash_type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_KDF_SHA
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.

◆ R_RSIP_KDF_SHA_ECDHSecretUpdate()

fsp_err_t R_RSIP_KDF_SHA_ECDHSecretUpdate (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_secret_t const *const *p_wrapped_secret*)

Inputs wrapped ECDH secret as a message. Input the messages required to perform the SHA operation using this API and R_RSIP_KDF_SHA_Update() in the desired order.

Implements rsip_api_t::kdfShaEcdhSecretUpdate.

Conditions

The argument p_wrapped_secret must be input the result of R_RSIP_ECDH_KeyAgree() or R_RSIP_ECDH_PlainKeyAgree(). This API can only be called if there is no previous message input, or if a 4-byte message has been input using R_RSIP_KDF_SHA_Update(). This API cannot be called multiple times in a process of performing SHA operation.

State transition

This API can only be executed in **STATE_KDF_SHA**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_SHA_Update()

fsp_err_t R_RSIP_KDF_SHA_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length*)

Inputs message. Input the messages required to perform the SHA operation using this API and R_RSIP_KDF_SHA_ECDHSecretUpdate() in the desired order.

Implements rsip_api_t::kdfShaUpdate.

State transition

This API can only be executed in **STATE_KDF_SHA**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_SHA_Finish()

fsp_err_t R_RSIP_KDF_SHA_Finish (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t *const *p_wrapped_dkm*)

Finalizes a SHA operation and generate DKM (Derived Keying Material).

Implements rsip_api_t::kdfShaFinish.

State transition

This API can only be executed in **STATE_KDF_SHA**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_SHA_Suspend()

fsp_err_t R_RSIP_KDF_SHA_Suspend (rsip_ctrl_t *const *p_ctrl*, rsip_kdf_sha_handle_t *const *p_handle*)

Suspends SHA operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::kdfShaSuspend.

State transition

This API can only be executed in **STATE_KDF_SHA**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_SHA_Resume()

fsp_err_t R_RSIP_KDF_SHA_Resume (rsip_ctrl_t *const *p_ctrl*, rsip_kdf_sha_handle_t const *const *p_handle*)

Resumes SHA operation suspended by R_RSIP_KDF_SHA_Suspend().

Implements rsip_api_t::kdfShaResume.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_KDF_SHA
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_KDF_HMAC_DKMKeyImport()

fsp_err_t R_RSIP_KDF_HMAC_DKMKeyImport (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*, uint32_t const *key_length*, rsip_wrapped_key_t *const *p_wrapped_key*)

Converts wrapped MAC to wrapped HMAC key for KDF.

Implements rsip_api_t::kdfHmacDkmKeyImport.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration

The argument p_wrapped_dkm must be input the result of R_RSIP_KDF_HMAC_SignFinish().

Argument kdf_data_length depends on key type.

- 32 to 64 (RSIP_KEY_TYPE_KDF_HMAC_SHA256)
- 48 to 64 (RSIP_KEY_TYPE_KDF_HMAC_SHA384)
- 64 (RSIP_KEY_TYPE_KDF_HMAC_SHA512)

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_HMAC_ECDHSecretKeyImport()

fsp_err_t R_RSIP_KDF_HMAC_ECDHSecretKeyImport (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_secret_t const *const *p_wrapped_secret*, rsip_wrapped_key_t *const *p_wrapped_key*)

Converts wrapped ECDH secret to wrapped HMAC key for KDF.

Implements rsip_api_t::kdfHmacEcdhSecretKeyImport.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_HMAC_Init()

fsp_err_t R_RSIP_KDF_HMAC_Init (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key*)

Prepares a HMAC generation.

Implements rsip_api_t::kdfHmacInit.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_KDF_HMAC
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.

◆ R_RSIP_KDF_HMAC_DKMUpdate()

fsp_err_t R_RSIP_KDF_HMAC_DKMUpdate (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*)

Inputs wrapped Derived Keying Material (DKM) as a message.

Implements rsip_api_t::kdfHmacDkmUpdate.

Conditions

The argument p_wrapped_dkm must be input the result of R_RSIP_KDF_HMAC_SignFinish().

State transition

This API can only be executed in **STATE_KDF_HMAC**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_KDF_HMAC_ECDHSecretUpdate()

fsp_err_t R_RSIP_KDF_HMAC_ECDHSecretUpdate (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_secret_t const *const *p_wrapped_secret*)

Inputs wrapped ECDH secret as a message.

Implements rsip_api_t::kdfHmacEcdhSecretUpdate.

Conditions

The argument p_wrapped_secret must be input the result of R_RSIP_ECDH_KeyAgree() or R_RSIP_ECDH_PlainKeyAgree().

State transition

This API can only be executed in **STATE_KDF_HMAC**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_HMAC_Update()

fsp_err_t R_RSIP_KDF_HMAC_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length*)

Inputs message.

Implements rsip_api_t::kdfHmacUpdate.

State transition

This API can only be executed in **STATE_KDF_HMAC**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_HMAC_SignFinish()

fsp_err_t R_RSIP_KDF_HMAC_SignFinish (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t *const *p_wrapped_dkm*)

Finalizes a HMAC generation.

Implements rsip_api_t::kdfHmacSignFinish.

State transition

This API can only be executed in **STATE_KDF_HMAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_HMAC_Suspend()

fsp_err_t R_RSIP_KDF_HMAC_Suspend (rsip_ctrl_t *const *p_ctrl*, rsip_kdf_hmac_handle_t *const *p_handle*)

Suspends HMAC operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::kdfHmacSuspend.

State transition

This API can only be executed in **STATE_KDF_HMAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_HMAC_Resume()

fsp_err_t R_RSIP_KDF_HMAC_Resume (rsip_ctrl_t *const *p_ctrl*, rsip_kdf_hmac_handle_t const *const *p_handle*)

Resumes HMAC operation suspended by R_RSIP_KDF_HMAC_Suspend().

Implements rsip_api_t::kdfHmacResume.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_KDF_HMAC
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_KDF_DKMConcatenate()

fsp_err_t R_RSIP_KDF_DKMConcatenate (rsip_wrapped_dkm_t *const *p_wrapped_dkm1*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm2*, uint32_t const *wrapped_dkm1_buffer_length*)

Concatenates two wrapped derived keying materials (DKMs).

DKM1 || DKM2 is output to p_wrapped_dkm1.

Implements rsip_api_t::kdfDkmConcatenate.

State transition

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.

◆ R_RSIP_KDF_DerivedKeyImport()

fsp_err_t R_RSIP_KDF_DerivedKeyImport (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*, uint32_t const *position*, rsip_wrapped_key_t *const *p_wrapped_key*)

Outputs a wrapped key from KDF output.

Implements rsip_api_t::kdfDerivedKeyImport.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument p_wrapped_dkm must be input the result of R_RSIP_KDF_SHA_Finish(), R_RSIP_KDF_HMAC_SignFinish() or R_RSIP_KDF_DKMConcatenate().

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_KDF_DerivedIVWrap()

fsp_err_t R_RSIP_KDF_DerivedIVWrap (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*, rsip_initial_vector_type_t const *initial_vector_type*, uint32_t const *position*, uint8_t const *const *p_tls_sequence_num*, uint8_t *const *p_wrapped_initial_vector*)

Outputs a initial vector from KDF output.

Implements rsip_api_t::kdfDerivedIvWrap

Conditions

Argument p_wrapped_dkm must be input the result of R_RSIP_KDF_SHA_Finish(), R_RSIP_KDF_HMAC_SignFinish() or R_RSIP_KDF_DKMConcatenate().

Argument initial_vector_type must be one of the following:

- RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_OTF_Init()

fsp_err_t R_RSIP_OTF_Init (rsip_ctrl_t *const *p_ctrl*, rsip_otf_channel_t const *channel*, rsip_wrapped_key_t *const *p_wrapped_key*, uint8_t const *const *p_seed*)

Initialize on-the-fly decryption on RSIP. Implements rsip_api_t::otfInit.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled or selected channel is invalid.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Attention

This function is only part of on-the-fly decryption activation process and intended to be called from a higher level FSP module. Even if a user calls this function directly, the feature will not be enabled.

◆ R_RSIP_PKI_VerifiedCertInfoExport()

fsp_err_t R_RSIP_PKI_VerifiedCertInfoExport (rsip_ctrl_t *const *p_ctrl*, rsip_verified_cert_info_t *const *p_verified_cert_info*)

Exports verified certificate information stored in this driver.

The certificate is last called function R_RSIP_PKI_ECDSA_CertVerify(), R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify(), R_RSIP_PKI_RSASSA_PSS_CertVerify(), or R_RSIP_PKI_VerifiedCertInfoImport().

Implements rsip_api_t::pkiVerifiedCertInfoExport.

State transition

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.

◆ R_RSIP_PKI_VerifiedCertInfoImport()

fsp_err_t R_RSIP_PKI_VerifiedCertInfoImport (rsip_ctrl_t *const *p_ctrl*, rsip_verified_cert_info_t const *const *p_verified_cert_info*)

Imports verified certificate information.

Implements rsip_api_t::pkiVerifiedCertInfoImport.

State transition

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.

◆ R_RSIP_PKI_CertKeyImport()

fsp_err_t R_RSIP_PKI_CertKeyImport (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_cert*, uint32_t const *cert_length*, uint8_t const *const *p_key_param1*, uint32_t const *key_param1_length*, uint8_t const *const *p_key_param2*, uint32_t const *key_param2_length*, rsip_hash_type_t const *hash_function*, rsip_wrapped_key_t *const *p_wrapped_public_key*)

Wraps the public key in the verified public key certificate.

Implements rsip_api_t::pkiCertKeyImport.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Note

- *Argument p_cert must correspond to the signature of the last called function R_RSIP_PKI_ECDSA_CertVerify(), R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify(), or R_RSIP_PKI_RSASSA_PSS_CertVerify().*
- *For arguments p_key_param1 and p_key_param2, specify an address inside the certificate. Addresses outside the certificate are invalid.*

◆ R_RSIP_RSA_Encrypt()

fsp_err_t R_RSIP_RSA_Encrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_plain*, uint8_t *const *p_cipher*)

Encrypts plaintext with raw RSA.

Implements rsip_api_t::rsaEncrypt.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Note

This API provides RSA low-level primitives (RSAEP/RSAVP1). It should be used in conjunction with any padding scheme.

◆ R_RSIP_RSA_Decrypt()

fsp_err_t R_RSIP_RSA_Decrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_cipher*, uint8_t *const *p_plain*)

Decrypts ciphertext with raw RSA.

Implements rsip_api_t::rsaDecrypt.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Note

This API provides RSA low-level primitives (RSADP/RSASP1). It should be used in conjunction with any padding scheme.

◆ R_RSIP_RSAES_PKCS1_V1_5_Encrypt()

fsp_err_t R_RSIP_RSAES_PKCS1_V1_5_Encrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_plain*, uint32_t const *plain_length*, uint8_t *const *p_cipher*)

Encrypts plaintext with RSAES-PKCS1-v1_5.

Implements rsip_api_t::rsaesPkcs1V15Encrypt.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

mLen (plain_length) and k (modulus length) must meet the following condition.

- mLen <= k - 11

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RSAES_PKCS1_V1_5_Decrypt()

fsp_err_t R_RSIP_RSAES_PKCS1_V1_5_Decrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_cipher*, uint8_t *const *p_plain*, uint32_t *const *p_plain_length*, uint32_t const *plain_buffer_length*)

Decrypts with RSAES-PKCS1-v1_5.

Implements rsip_api_t::rsaesPkcs1V15Decrypt.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

plain_buffer_length must be greater than or equal to mLen(plaintext length).

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Note

This API skips the ciphertext length checking at RFC8017 (PKCS#1 v2.2) Section 7.2.2 Step 1.

◆ R_RSIP_RSAES_OAEP_Encrypt()

fsp_err_t R_RSIP_RSAES_OAEP_Encrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint8_t const *const *p_label*, uint32_t const *label_length*, uint8_t const *const *p_plain*, uint32_t const *plain_length*, uint8_t *const *p_cipher*)

Encrypts plaintext with RSAES-OAEP.

Implements rsip_api_t::rsaesOaepEncrypt.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

mLen (plain_length), hLen (hash length of hash_function), and k (modulus length) must meet the following condition.

- mLen <= k - 2 hLen - 2

Argument hash_function and mask_generation_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RSAES_OAEP_Decrypt()

fsp_err_t R_RSIP_RSAES_OAEP_Decrypt (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const

p_wrapped_private_key, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint8_t const *const *p_label*, uint32_t const *label_length*, uint8_t const *const *p_cipher*, uint8_t *const *p_plain*, uint32_t *const *p_plain_length*, uint32_t const *plain_buffer_length*)

Decrypts ciphertext with RSAES-OAEP.

Implements rsip_api_t::rsaesOaepDecrypt.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

hLen (hash length of hash_function) and k (modulus length) must meet the following condition.

- k >= 2 hLen + 2

plain_buffer_length must be greater than or equal to mLen(plaintext length).

Argument hash_function and mask_generation_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

Note

This API skips the ciphertext length checking at RFC8017 (PKCS#1 v2.2) Section 7.1.2 Step 1.

◆ R_RSIP_RSASSA_PKCS1_V1_5_Sign()

fsp_err_t R_RSIP_RSASSA_PKCS1_V1_5_Sign (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_hash_type_t const *hash_function*, uint8_t const *const *p_hash*, uint8_t *const *p_signature*)

Signs message with RSASSA-PKCS1-v1_5.

Implements rsip_api_t::rsassaPkcs1V15Sign.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument hash_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance with hash_function.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RSASSA_PKCS1_V1_5_Verify()

fsp_err_t R_RSIP_RSASSA_PKCS1_V1_5_Verify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature*)

Verifies signature with RSASSA-PKCS1-v1_5.

Implements rsip_api_t::rsassaPkcs1V15Verify.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument hash_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance with hash_function.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RSASSA_PSS_Sign()

fsp_err_t R_RSIP_RSASSA_PSS_Sign (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint32_t const *salt_length*, uint8_t const *const *p_hash*, uint8_t *const

p_signature)

Signs message with RSASSA-PSS.

Implements rsip_api_t::rsassaPssSign.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument hash_function and mask_generation_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance with hash_function.

Salt length salt_length must be one of the following:

- Any member of enumeration rsip_rsa_salt_length_t
  - RSIP_RSA_SALT_LENGTH_AUTO
  - RSIP_RSA_SALT_LENGTH_HASH
  - RSIP_RSA_SALT_LENGTH_MAX
- Integers that satisfies the formula: sLen <= emLen - hLen - 2
  - sLen is salt_length
  - emLen is the same as modulus length
  - hLen is the hash length

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_RSASSA_PSS_Verify()

fsp_err_t R_RSIP_RSASSA_PSS_Verify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint32_t const *salt_length*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature*)

Verifies signature with RSASSA-PSS. Implements rsip_api_t::rsassaPssVerify.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument hash_function and mask_generation_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance with hash_function.

Salt length salt_length must be one of the following:

- Any member of enumeration rsip_rsa_salt_length_t
  - RSIP_RSA_SALT_LENGTH_AUTO
  - RSIP_RSA_SALT_LENGTH_HASH
  - RSIP_RSA_SALT_LENGTH_MAX
- Integers that satisfies the formula: sLen <= emLen - hLen - 2
  - sLen is salt_length
  - emLen is the same as modulus length
  - hLen is the hash length

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify()

fsp_err_t R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature*)

Verifies a certificate with RSASSA-PKCS1-v1_5.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument hash_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance with hash_function.

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	hash_function	Hash function in EMSA-PKCS1-v1_5.
[in]	p_hash	Pointer to input hash.
[in]	p_signature	Pointer to input signature.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.

FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_PKI_RSASSA_PSS_CertVerify()

fsp_err_t R_RSIP_PKI_RSASSA_PSS_CertVerify (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint32_t const *salt_length*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature*)

Verifies a certificate with RSASSA-PSS.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Argument hash_function and mask_generation_function must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

Message hash p_hash should be computed in advance with hash_function.

Salt length salt_length must be one of the following:

- Any member of enumeration rsip_rsa_salt_length_t
  - RSIP_RSA_SALT_LENGTH_AUTO
  - RSIP_RSA_SALT_LENGTH_HASH
  - RSIP_RSA_SALT_LENGTH_MAX
- Integers that satisfies the formula: sLen <= emLen - hLen - 2
  - sLen is salt_length
  - emLen is the same as modulus length
  - hLen is the hash length

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	hash_function	Hash function in EMSA-PSS-VERIFY.
[in]	mask_generation_function	Mask generation function in EMSA-PSS-VERIFY.
[in]	salt_length	Salt length. RSIP_RSA_SALT_LENGTH_AUTO, RSIP_RSA_SALT_LENGTH_HASH, RSIP_RSA_SALT_LENGTH_MA

		X, 0, or positive integers can be set, where salt_length <= emLen - hLen - 2 (emLen is the same as the key length and hLen is the hash length).
[in]	p_hash	Pointer to input hash.
[in]	p_signature	Pointer to input signature.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	Input parameter is invalid.
FSP_ERR_INVALID_SIZE	Any length is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_SHA_Compute()

fsp_err_t R_RSIP_SHA_Compute (rsip_ctrl_t *const *p_ctrl*, rsip_hash_type_t const *hash_type*, uint8_t const *const *p_message*, uint32_t const *message_length*, uint8_t *const *p_digest*)

Generates SHA message digest. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::shaCompute.

Conditions

See R_RSIP_SHA_Init().

Output length

See R_RSIP_SHA_Finish().

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.

◆ R_RSIP_SHA_Init()

fsp_err_t R_RSIP_SHA_Init (rsip_ctrl_t *const *p_ctrl*, rsip_hash_type_t const *hash_type*)

Starts SHA operation.

Implements rsip_api_t::shaInit.

Conditions

Argument hash_type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_SHA
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.

◆ R_RSIP_SHA_Update()

fsp_err_t R_RSIP_SHA_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length*)

Inputs SHA message. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::shaUpdate.

State transition

This API can only be executed in **STATE_SHA**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_SHA_Finish()

fsp_err_t R_RSIP_SHA_Finish (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_digest*)

Outputs SHA message digest.

Implements rsip_api_t::shaFinish.

Output length

Output length to p_digest depends on hash_function.

- 28 (RSIP_HASH_TYPE_SHA224, RSIP_HASH_TYPE_SHA512_224, RSIP_HASH_TYPE_SHA3_224)
- 32 (RSIP_HASH_TYPE_SHA256, RSIP_HASH_TYPE_SHA512_256, RSIP_HASH_TYPE_SHA3_256)
- 48 (RSIP_HASH_TYPE_SHA384, RSIP_HASH_TYPE_SHA3_384)
- 64 (RSIP_HASH_TYPE_SHA512, RSIP_HASH_TYPE_SHA3_512)

State transition

This API can only be executed in **STATE_SHA**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_SHA_Suspend()

fsp_err_t R_RSIP_SHA_Suspend (rsip_ctrl_t *const *p_ctrl*, rsip_sha_handle_t *const *p_handle*)

Suspends SHA operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::shaSuspend.

State transition

This API can only be executed in **STATE_SHA**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_SHA_Resume()

fsp_err_t R_RSIP_SHA_Resume (rsip_ctrl_t *const *p_ctrl*, rsip_sha_handle_t const *const *p_handle*)

Resumes SHA operation suspended by R_RSIP_SHA_Suspend().

Implements rsip_api_t::shaResume.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_SHA
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

◆ R_RSIP_HMAC_Compute()

fsp_err_t R_RSIP_HMAC_Compute (rsip_ctrl_t *const *p_ctrl*, const rsip_wrapped_key_t * *p_wrapped_key*, uint8_t const *const *p_message*, uint32_t const *message_length*, uint8_t *const *p_mac*)

Generates HMAC. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::hmacCompute.

Conditions

See R_RSIP_HMAC_Init().

Output length

See R_RSIP_HMAC_SignFinish().

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_HMAC_Verify()

fsp_err_t R_RSIP_HMAC_Verify (rsip_ctrl_t *const *p_ctrl*, const rsip_wrapped_key_t * *p_wrapped_key*, uint8_t const *const *p_message*, uint32_t const *message_length*, uint8_t const *const *p_mac*, uint32_t const *mac_length*)

Verifies HMAC. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::hmacVerify.

Conditions

See R_RSIP_HMAC_Init() and R_RSIP_HMAC_VerifyFinish().

State transition

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	mac_length is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key value is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	MAC verification is failed.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_HMAC_Init()

fsp_err_t R_RSIP_HMAC_Init (rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key*)

Starts HMAC operation.

Implements rsip_api_t::hmacInit.

Conditions

Argument rsip_wrapped_key_t::type must be supported by both the function and the device, and must also be enabled in the configuration. For more details, please refer to Supported Algorithms and Configuration.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_HMAC
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_NOT_ENABLED	Input key type is disabled in this function by configuration.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.

◆ R_RSIP_HMAC_Update()

fsp_err_t R_RSIP_HMAC_Update (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length*)

Inputs HMAC message. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::hmacUpdate.

State transition

This API can only be executed in **STATE_HMAC**, and does not cause any state transitions.

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_HMAC_SignFinish()

fsp_err_t R_RSIP_HMAC_SignFinish (rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_mac*)

Outputs HMAC.

Implements rsip_api_t::hmacSignFinish.

Output length

Output length to p_mac depends on key type of p_wrapped_key.

- 28 (RSIP_KEY_TYPE_HMAC_SHA224, RSIP_KEY_TYPE_HMAC_SHA512_224, RSIP_KEY_TYPE_HMAC_SHA3_224)
- 32 (RSIP_KEY_TYPE_HMAC_SHA256, RSIP_KEY_TYPE_HMAC_SHA512_256, RSIP_KEY_TYPE_HMAC_SHA3_256)
- 48 (RSIP_KEY_TYPE_HMAC_SHA384, RSIP_KEY_TYPE_HMAC_SHA3_384)
- 64 (RSIP_KEY_TYPE_HMAC_SHA512, RSIP_KEY_TYPE_HMAC_SHA3_512)

State transition

This API can only be executed in **STATE_HMAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_HMAC_VerifyFinish()

fsp_err_t R_RSIP_HMAC_VerifyFinish (rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*, uint32_t const *mac_length*)

Verifies HMAC.

Implements rsip_api_t::hmacVerifyFinish.

Conditions

Argument mac_length depends on key type of p_wrapped_key. Usually the longest length is recommended.

- 4 to 28 (RSIP_KEY_TYPE_HMAC_SHA224, RSIP_KEY_TYPE_HMAC_SHA512_224, RSIP_KEY_TYPE_HMAC_SHA3_224)
- 4 to 32 (RSIP_KEY_TYPE_HMAC_SHA256, RSIP_KEY_TYPE_HMAC_SHA512_256, RSIP_KEY_TYPE_HMAC_SHA3_256)
- 4 to 48 (RSIP_KEY_TYPE_HMAC_SHA384, RSIP_KEY_TYPE_HMAC_SHA3_384)
- 4 to 64 (RSIP_KEY_TYPE_HMAC_SHA512, RSIP_KEY_TYPE_HMAC_SHA3_512)

State transition

This API can only be executed in **STATE_HMAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
FSP_ERR_ASSERTION	No change
FSP_ERR_NOT_OPEN	No change
FSP_ERR_INVALID_STATE	No change
FSP_ERR_INVALID_SIZE	No change
Others	STATE_MAIN

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_INVALID_SIZE	mac_length is illegal.
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input key is illegal.
FSP_ERR_CRYPTO_RSIP_FAIL	MAC verification is failed.
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_HMAC_Suspend()

fsp_err_t R_RSIP_HMAC_Suspend (rsip_ctrl_t *const *p_ctrl*, rsip_hmac_handle_t *const *p_handle*)

Suspends HMAC operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::hmacSuspend.

State transition

This API can only be executed in **STATE_HMAC**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_MAIN
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption is detected.

◆ R_RSIP_HMAC_Resume()

fsp_err_t R_RSIP_HMAC_Resume (rsip_ctrl_t *const *p_ctrl*, rsip_hmac_handle_t const *const *p_handle*)

Resumes HMAC operation suspended by R_RSIP_HMAC_Suspend().

Implements rsip_api_t::hmacResume.

State transition

This API can only be executed in **STATE_MAIN**, and causes state transition.

Return value	Next state
FSP_SUCCESS	STATE_HMAC
Others	No change

Return values

FSP_SUCCESS	Normal termination.
FSP_ERR_ASSERTION	A required parameter is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_STATE	Internal state is illegal.

5.2.15.3 Azure RTOS NetX Crypto HW Acceleration (rm_netx_secure_crypto)

Modules » Security

Detailed Description

Hardware acceleration for the Netx Crypto implementation of the Microsoft Azure RTOS NetX Crypto API.

Overview

Please refer to the NetXDuo - NetX Crypto documentation for further details.

HW Overview

Crypto Peripheral version	Devices
RSIP-E50D	RX74M, RX74N

Features

		AES Engine	AES Engine 2	SCE5	SCE5_B	SCE7	SCE9	RSIP-E11A	RSIP-E31A	RSIP-E51A	RSIP-E50D
TRNG		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Hash	SHA224, SHA256	–	–	–	–	Yes	Yes	Yes	Yes	Yes	Yes
	SHA384, SHA512	–	–	–	–	–	–	–	Yes	Yes	Yes
	SHA512/224, SHA512/256	–	–	–	–	–	–	–	Yes	Yes	Yes
MAC	HMAC-SHA224, HMAC-SHA256	–	–	–	–	Yes	Yes	Yes	Yes	Yes	Yes
	HMAC-SHA384, HMAC-SHA512	–	–	–	–	–	–	–	Yes	Yes	Yes
	HMAC-SHA512/224	–	–	–	–	–	–	–	Yes	Yes	Yes
	HMAC-SHA512/256	–	–	–	–	–	–	–	Yes	Yes	Yes
AES	AES128-CBC, CTR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES128-GCM (1)	–	Yes	Yes	Yes	Yes	(2)	Yes	Yes	Yes	Yes
	AES128-CCM (1)	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES192-CBC, CTR	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	AES192-GCM (1)	–	–	–	–	Yes	(2)	–	–	Yes	Yes
	AES19	–	–	–	–	Yes	Yes	–	–	Yes	Yes

	2-CCM (1)										
	AES256-CBC, CTR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES256-GCM (1)	–	Yes	Yes	Yes	Yes	(2)	Yes	Yes	Yes	Yes
	AES256-CCM (1)	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ECC	secp256r1	–	–	–	–	Yes	Yes	Yes	Yes	Yes	Yes
	secp384r1	–	–	–	–	Yes	Yes	–	Yes	Yes	Yes
RSA	RSA-1024	–	–	–	–	–	Yes	–	–	Yes	Yes
	RSA-2048	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	RSA-3072 (3)	–	–	–	–	–	Yes	–	–	Yes	Yes
	RSA-4096 (3)	–	–	–	–	–	Yes	–	–	Yes	Yes

Note

1. Only block cipher unit is HW accelerated
2. Encrypt - Fully accelerated; Decrypt - Only GHASH and block cipher unit is HW accelerated
3. Signature verification only

Configuration

Build Time Configurations for rm_netx_secure_crypto_sw_port

The following build time configurations are defined in fsp_cfg/middleware/rm_netx_secure_crypto_cfg.h:

Configuration	Options	Default	Description
Standalone Usage	• Use Standalone Crypto Only • Use with TLS	Use Standalone Crypto Only	Defines NX_CRYPTO_STANDALONE_ENABLE.
Maximum RSA Modulus size (bits)	• 1024 • 2048	4096	

- 3072
- 4096

Build Time Configurations for rm_netx_secure_crypto

The following build time configurations are defined in fsp_cfg/middleware/rm_netx_secure_crypto_cfg.h:

Configuration	Options	Default	Description
Hardware Acceleration > Hash > SHA256/224	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA256_ALT.
Hardware Acceleration > Hash > SHA512/384	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA512_ALT.
Hardware Acceleration > Cipher > AES	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_AES_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECC > ECC	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_ECC_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECC > ECDSA Scratch Buffer Size (Bytes)	Value must be a non-negative integer	3016	Sets value of NX_CRYPTO_ECDSA_SCRATCH_BUFFER_SIZE
Hardware Acceleration > Public Key Cryptography (PKC) > ECC > ECDH Scratch Buffer Size (Bytes)	Value must be a non-negative integer	2464	Sets value of NX_CRYPTO_ECDH_SCRATCH_BUFFER_SIZE
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA	MCU Specific Options		Enables/Disables RSA HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 1024 (HW)	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_1024_ALT to allow HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 2048 (HW)	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_2048_ALT to allow HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 3072	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_3072_ALT to allow HW support

Verify/Encryption (HW)			
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 4096 Verify/Encryption (HW)	MCU Specific Options		Enables RSA NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_4096_ALT to allow HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA Scratch Buffer Size (Bytes)	MCU Specific Options		Sets value of NX_CRYPTO_RSA_SCRATCH_BUFFER_SIZE
Hardware Acceleration > TRNG	• Enabled • Disabled	Enabled	Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_TRNG_ALT.
Standalone Usage	• Use Standalone Crypto Only • Use with TLS	Use Standalone Crypto Only	Defines NX_CRYPTO_STANDALONE_ENABLE.

Random Number Generator Configuration

To enable hardware acceleration for the TRNG, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_TRNG_ALT must be defined in the configuration file. By default TRNG is enabled which can be disabled using the RX Configuration editor.

Once enabled 'rand' function will be mapped to HW TRNG; the 'srand' function is not supported, any calls to this function will have no effect. Functionality to re-seed the HW TRNG is not supported by the existing implementation.

If disabled, both 'rand' and 'srand' will be mapped to the C Standard Library. This would require setting up the heap as 'rand' implementation calls 'malloc'.

SHA Configuration

To enable hardware acceleration for the SHA256/224 or SHA384/512 calculation, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA256_ALT or NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA512_ALT must be defined in the configuration file. By default SHA is enabled which can be disabled using the RX Configuration editor.

AES Configuration

To enable hardware acceleration for the AES128/192/256 operation, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_AES_ALT must be defined in the configuration file. By default AES is enabled which can be disabled using the RX Configuration editor.

ECC Configuration

To enable hardware acceleration for the ECDSA and ECDH for curves SECP384R1 and SECP256R1, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_ECC_ALT must be defined in the configuration file. By default ECC operations are enabled which can be disabled using the RX Configuration editor.

RSA Configuration

To enable hardware acceleration for the RSA Encrypt/Decrypt (or Sign/Verify) operation(s), the macro(s) below must be must be defined in the configuration file:

Configuration Macro	Feature / Operation
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_1024_ALT	Signature Generation / Signature Verification (Encryption / Decryption)
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_2048_ALT	Signature Generation / Signature Verification (Encryption / Decryption)
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_3072_ALT	Signature Verification (Encryption Only)
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_4096_ALT	Signature Verification (Encryption Only)

By default RSA 2048 is enabled which can be disabled using the RX Configuration editor.

RSA software implementation is completely disabled when any of the above macros are enabled.

Usage Notes

Memory Alignment

Use 32bit aligned buffer pointers as arguments to APIs for best performance.

Hardware Initialization

_nx_crypto_initialize() must be invoked before using the NetX Crypto APIs to ensure that the SCE peripheral is initialized.

Memory Usage

Sufficient memory must be allocated to be used as 'crypto_metadata' for the chosen crypto operation(s). Refer Azure RTOS NetX Crypto API description for recommended 'crypto_metadata_size' based on selected crypto operations. Sufficient amount of memory must be allocated for the thread stack to support low level crypto operations when using this module in the standalone mode or through NetX Secure (TLS). A minimum stack of 0x1000 is required to use ECC and RSA. This is either the main stack in a bare metal application or the specific thread stack for an RTOS based application.

AES Usage

GCM mode

The first byte of the IV must indicate the length of the subsequent IV. For example if the IV is {0x00, 0x00, 0x00}, then the IV pointer passed to the _nx_crypto_method_aes_operation must store the IV as {0x03, 0x00, 0x00, 0x00}. Refer to the example code for actual usage.

CTR mode

For CTR mode the IV pointer must be as defined in Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP) under 'Figure 2. Counter Block Format'. The IV must be 8 bytes in length. The Nonce field in the reference above must be 4 bytes and should be passed to _nx_crypto_method_aes_operation through the key pointer stored after the

actual AES key. For Example, if the AES 128-bit Plain Key is {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} and the Nonce is {0x01, 0x07, 0xBD, 0xFD}, the key passed to the _nx_crypto_method_aes_operation during Encryption/Decryption should be set as {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0xBD, 0xFD}. This format would also be valid for Wrapped keys where the Nonce is appended at the end of the actual Wrapped key. Refer to the example code for actual usage.

The 'Block Counter' field in the above reference is fixed to {0x00, 0x00, 0x00, 0x01} at the beginning and increments internally after every subsequent AES block is processed. Test vectors that have the initial Block Counter not set to {0x00, 0x00, 0x00, 0x01} cannot be used in this implementation.

ECC Usage

ECC operations include ECDH and ECDSA. As a part of ECDSA operation the input message can be hashed before signing or verification, or the message digest can be provided directly. ECC Scratch buffer size can be optionally reduced as supported ECC computations are now done by the HW. This is controlled by NX_CRYPTO_ECDSA_SCRATCH_BUFFER_SIZE and NX_CRYPTO_ECDH_SCRATCH_BUFFER_SIZE macros for ECDSA and ECDH respectively.

Operation	Key Format
ECDSA Signature	Plain private key; Wrapped private key
ECDSA Signature-Verify	Uncompressed public key
Key Generation using ECDSA operation API	Wrapped private key; Uncompressed public key
ECDH private key import	Plain private key; Wrapped private key (Allows for Uncompressed and Formatted public key)
ECDH public key export	Uncompressed public key
ECDH setup	Uncompressed public key
ECDH shared secret calculate	Uncompressed public key
Key Generation using ECDH operation API	Wrapped private key; Uncompressed public key

Note:

- Uncompressed public key is of the form (0x04 || Qx || Qy). Refer Section 2.2. Subject Public Key under RFC5480
- Formatted public key is of the form (Key Info (4 bytes) || Qx || Qy || Key Info (16 bytes)). This is the key which is used internally by SCE peripheral.

RSA Usage

Wrapped Key Usage

To use the NetX Crypto stack with wrapped private keys (for signature generation/decryption), a dummy pointer (non-NULL) should be passed to the 'key' parameter during _nx_crypto_method_rsa_init API call. However, the 'key_size_in_bits' parameter should be equal to the intended RSA modulus length in bits. The actual wrapped key must be passed as the 'key' parameter to the _nx_crypto_method_rsa_operation API call with its length in bits passed through the 'key_size_in_bits' parameter.

For PKCS1V15 operation 'key' and 'key_size_in_bits' parameters of

_nx_crypto_method_pkcs1_v1_5_init are unused. These can be passed as NULL and 0 respectively. The actual wrapped key must be passed as the 'key' parameter to the _nx_crypto_method_pkcs1_v1_5_operation API call and the intended modulus length in bits must be passed through the 'key_size_in_bits' parameter.

Software Implementation

The plaintext data passed in to the RSA encryption API must be 4-byte aligned. If it is not aligned then when the resultant encrypted data is decrypted using the decryption API, the unaligned data will be endian-swapped. For instance, if {0x01, 0x02,0x03, 0x04, 0x05} is passed as input data for encryption, when decrypted, the result will be {0x01, 0x02, 0x03, 0x04, 0x00, 0x00, 0x00, 0x05}. Note that this alignment of input data need not be taken care while using Hardware accelerated RSA.

Limitations

- Only little endian mode is supported.
- RSA CRT keys are not supported.
- ECJPAKE related operations are unsupported for NIST 224, 256 and 384 bit curves when HW ECC is eanbled.

Examples

Initialization Example

This example shows how to initialize the HW crypto engine. This step must be performed before any crypto algorithm is used.

```
/* Setup the platform; initialize the SCE and the TRNG */
   err = nx_crypto_initialize();
   assert(NX_CRYPTO_SUCCESS == err);
```

Hash Example

This is an example on calculating the SHA256 hash using the NetX Crypto API.

```
extern NX_CRYPTO_METHOD crypto_method_sha256;
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
```

```
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void netx_secure_crypto_sha256_example (void)
{
 size_t   actual_hash_len = RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES;
    uint8_t actual_hash[RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES];
    uint8_t metadata[sizeof(NX_CRYPTO_SHA256)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_SHA256);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto SHA256 init */
    err = _nx_crypto_method_sha256_init(&crypto_method_sha256, NX_CRYPTO_NULL, 0,
&handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_INITIALIZE */
    err = _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_INITIALIZE,
                                             handler,
                                             &crypto_method_sha256,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             NX_CRYPTO_NULL,
                                             0,
                                             metadata,
                                             metadata_size,
                                             NX_CRYPTO_NULL,
                                             NX_CRYPTO_NULL);
```

```
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_UPDATE,
 * call this multiple times if needed to hash multiple data batches */
   err =
      _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_UPDATE,
                                         handler,
                                         &crypto_method_sha256,
                                         NX_CRYPTO_NULL,
                                         0,
                                         (uint8_t *) NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200),
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL,
                                         0,
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_CALCULATE */
   err = _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_CALCULATE,
                                            handler,
                                            &crypto_method_sha256,
                                            NX_CRYPTO_NULL,
                                            0,
                                            NX_CRYPTO_NULL,
                                            0,
                                            NX_CRYPTO_NULL,
                                            (uint8_t *) actual_hash,
                                            actual_hash_len,
                                            metadata,
                                            metadata_size,
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL);
```

```
    assert(NX_CRYPTO_SUCCESS == err);
 /* Ensure generated SHA256 hash matches the expected digest */
    err = (uint32_t) memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len);
    assert(0 == err);
}
```

AES Example

This is an example on using the NetX Crypto API to encrypt and decrypt multi-block data.

AES CBC Example

```
extern NX_CRYPTO_METHOD crypto_method_aes_cbc_256;
/* fe8901fecd3ccd2ec5fdc7c7a0b50519c245b42d611a5ef9e90268d59f3edf33 */
const uint8_t NIST_AES256_CBC_key[] =
{
    0xfe, 0x89, 0x01, 0xfe, 0xcd, 0x3c, 0xcd, 0x2e, 0xc5, 0xfd, 0xc7, 0xc7, 0xa0,
0xb5, 0x05, 0x19,
    0xc2, 0x45, 0xb4, 0x2d, 0x61, 0x1a, 0x5e, 0xf9, 0xe9, 0x02, 0x68, 0xd5, 0x9f,
0x3e, 0xdf, 0x33
};
/* 851e8764776e6796aab722dbb644ace8 */
const uint8_t NIST_AES256_CBC_iv[] =
{
    0xbd, 0x41, 0x6c, 0xb3, 0xb9, 0x89, 0x22, 0x28, 0xd8, 0xf1, 0xdf, 0x57, 0x56,
0x92, 0xe4, 0xd0
};
/* 6282b8c05c5c1530b97d4816ca434762 */
const uint8_t NIST_AES256_CBC_plaintext[] =
{
    0x8d, 0x3a, 0xa1, 0x96, 0xec, 0x3d, 0x7c, 0x9b, 0x5b, 0xb1, 0x22, 0xe7, 0xfe,
0x77, 0xfb, 0x12,
    0x95, 0xa6, 0xda, 0x75, 0xab, 0xe5, 0xd3, 0xa5, 0x10, 0x19, 0x4d, 0x3a, 0x8a,
0x41, 0x57, 0xd5,
    0xc8, 0x9d, 0x40, 0x61, 0x97, 0x16, 0x61, 0x98, 0x59, 0xda, 0x3e, 0xc9, 0xb2,
```

```
0x47, 0xce, 0xd9
};
/* 6acc04142e100a65f51b97adf5172c41 */
const uint8_t NIST_AES256_CBC_ciphertext[] =
{
    0x60, 0x8e, 0x82, 0xc7, 0xab, 0x04, 0x00, 0x7a, 0xdb, 0x22, 0xe3, 0x89, 0xa4,
0x47, 0x97, 0xfe,
    0xd7, 0xde, 0x09, 0x0c, 0x8c, 0x03, 0xca, 0x8a, 0x2c, 0x5a, 0xcd, 0x9e, 0x84,
0xdf, 0x37, 0xfb,
    0xc5, 0x8c, 0xe8, 0xed, 0xb2, 0x93, 0xe9, 0x8f, 0x02, 0xb6, 0x40, 0xd6, 0xd1,
0xd7, 0x24, 0x64
};
void netx_secure_crypto_aes256cbc_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_cbc_256, (uint8_t *)
NIST_AES256_CBC_key,
 sizeof(NIST_AES256_CBC_key) << 3U, &handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_aes_cbc_256,
                                          NULL,
```

```
                                        0,
                                        (uint8_t *) NIST_AES256_CBC_plaintext,
 sizeof(NIST_AES256_CBC_plaintext),
                                        (uint8_t *) NIST_AES256_CBC_iv,
                                        generated_ciphertext,
 sizeof(generated_ciphertext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, NIST_AES256_CBC_ciphertext,
sizeof(generated_ciphertext));
    assert(0 == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                        handler,
                                        &crypto_method_aes_cbc_256,
                                        NULL,
                                        0,
                                        (uint8_t *) NIST_AES256_CBC_ciphertext,
 sizeof(NIST_AES256_CBC_ciphertext),
                                        (uint8_t *) NIST_AES256_CBC_iv,
                                        generated_plaintext,
 sizeof(generated_plaintext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, NIST_AES256_CBC_plaintext,
sizeof(generated_ciphertext));
```

```
    assert(0 == err);
}
```

AES GCM Example

```
extern NX_CRYPTO_METHOD crypto_method_aes_128_gcm_16;
/* 83F9D97D4AB759FDDCC3EF54A0E2A8EC */
static const uint8_t key_gcm_128[] =
{
    0x83, 0xF9, 0xD9, 0x7D, 0x4A, 0xB7, 0x59, 0xFD, 0xDC, 0xC3, 0xEF, 0x54, 0xA0,
0xE2, 0xA8, 0xEC
};
/* In case of IV the IV length must be the first byte followed by the actual IV.
 * In this example the IV length is 0x01 and the actual IV is 0xCF
 */
/* 01CF */
static const uint8_t iv_gcm_128[] =
{
    0x01, 0xCF
};
/* 77E6329CF9424F71C808DF9170BFD298 */
static const uint8_t plain_gcm_128[] =
{
    0x77, 0xE6, 0x32, 0x9C, 0xF9, 0x42, 0x4F, 0x71, 0xC8, 0x08, 0xDF, 0x91, 0x70,
0xBF, 0xD2, 0x98
};
/* 6DD49EAEB4103DAC8F97E3234946DD2D */
static const uint8_t aad_gcm_128[] =
{
    0x6D, 0xD4, 0x9E, 0xAE, 0xB4, 0x10, 0x3D, 0xAC, 0x8F, 0x97, 0xE3, 0x23, 0x49,
0x46, 0xDD, 0x2D
};
/* 50DE86A7A92A8A5EA33DB5696B96CD77AA181E84BC8B4BF5A68927C409D422CB */
static const uint8_t secret_gcm_128[] =
{
```

```
 /* Ciphertext */
    0x50, 0xDE, 0x86, 0xA7, 0xA9, 0x2A, 0x8A, 0x5E, 0xA3, 0x3D, 0xB5, 0x69, 0x6B,
0x96, 0xCD, 0x77,
 /* Tag */
    0xAA, 0x18, 0x1E, 0x84, 0xBC, 0x8B, 0x4B, 0xF5, 0xA6, 0x89, 0x27, 0xC4, 0x09,
0xD4, 0x22, 0xCB
};
void netx_secure_crypto_aes128gcm_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_128_gcm_16,
                                  (uint8_t *) key_gcm_128,
 sizeof(key_gcm_128) << 3U,
           &handler,
           metadata,
           metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Setup Additional Authentication Data */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_SET_ADDITIONAL_DATA,
                                          handler,
                                          &crypto_method_aes_128_gcm_16,
                                          NULL,
                                          0,
                                          (uint8_t *) aad_gcm_128,
```

```
 sizeof(aad_gcm_128),
                                        NULL,
                                        NULL,
                                        0,
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                        handler,
                                        &crypto_method_aes_128_gcm_16,
                                        NULL,
                                        0,
                                        (uint8_t *) plain_gcm_128,
 sizeof(plain_gcm_128),
                                        (uint8_t *) iv_gcm_128,
                                        generated_ciphertext,
 sizeof(generated_ciphertext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* The 16 byte tag is appended to the generated ciphertext */
 /* Verify generated tag matches the expected tag */
    err = (uint32_t) memcmp(&generated_ciphertext[sizeof(plain_gcm_128)],
&secret_gcm_128[sizeof(plain_gcm_128)], 16U);
    assert(0 == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, secret_gcm_128, sizeof
(secret_gcm_128));
    assert(0 == err);
```

```
/* Setup Additional Authentication Data */
   err = _nx_crypto_method_aes_operation(NX_CRYPTO_SET_ADDITIONAL_DATA,
                                         handler,
                                         &crypto_method_aes_128_gcm_16,
                                         NULL,
                                         0,
                                         (uint8_t *) aad_gcm_128,
sizeof(aad_gcm_128),
                                         NULL,
                                         NULL,
                                         0,
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Decryption. */
   err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                         handler,
                                         &crypto_method_aes_128_gcm_16,
                                         NULL,
                                         0,
                                         (uint8_t *) generated_ciphertext,
sizeof(secret_gcm_128), /* ciphertext size + tag size */
                                         (uint8_t *) iv_gcm_128,
                                         generated_plaintext,
sizeof(generated_plaintext),
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Verify generated plaintext matches the input plaintext */
   err = (uint32_t) memcmp(generated_plaintext, plain_gcm_128, sizeof
```

```
(plain_gcm_128));
    assert(0 == err);
}
```

AES CTR Example

```
NX_CRYPTO_METHOD crypto_method_aes_ctr_256 =
{
    NX_CRYPTO_ENCRYPTION_AES_CTR,            /* AES crypto
algorithm                   */
    NX_CRYPTO_AES_256_KEY_LEN_IN_BITS,       /* Key size in bits
*/
    NX_CRYPTO_AES_IV_LEN_IN_BITS,            /* IV size in
bits                        */
    0,                                       /* ICV size in bits, not
used             */
    (NX_CRYPTO_AES_BLOCK_SIZE_IN_BITS >> 3), /* Block size in bytes
*/
 sizeof(NX_CRYPTO_AES),                    /* Metadata size in bytes                */
    _nx_crypto_method_aes_init,             /* AES-CBC initialization
routine         */
    _nx_crypto_method_aes_cleanup,          /* AES-CBC cleanup routine
*/
    _nx_crypto_method_aes_ctr_operation      /* AES-CBC
operation                      */
};
/*Note: For CTR, the key_ctr is the conjunction of key and nonce. */
/* D0E78C4D0B30D33F5BF4A132B2F94A4A38963511A3904B117E35A37B5AAC8A193BF0D158 */
const uint8_t key_ctr_256[] =
{
 /* AES Key */
    0xD0, 0xE7, 0x8C, 0x4D, 0x0B, 0x30, 0xD3, 0x3F, 0x5B, 0xF4, 0xA1, 0x32, 0xB2,
0xF9, 0x4A, 0x4A,
    0x38, 0x96, 0x35, 0x11, 0xA3, 0x90, 0x4B, 0x11, 0x7E, 0x35, 0xA3, 0x7B, 0x5A,
0xAC, 0x8A, 0x19,
```

```
 /* Nonce */
    0x3B, 0xF0, 0xD1, 0x58,
};
/* A1A31704C8B7E16C */
const uint8_t iv_ctr_256[] =
{
    0xA1, 0xA3, 0x17, 0x04, 0xC8, 0xB7, 0xE1, 0x6C,
};
/* 981FA33222C5451017530155A4BF7F29 */
const uint8_t plain_ctr_256[] =
{
    0x98, 0x1F, 0xA3, 0x32, 0x22, 0xC5, 0x45, 0x10, 0x17, 0x53, 0x01, 0x55, 0xA4,
0xBF, 0x7F, 0x29,
};
/* 643B91B4E541B20AAAEAB77F2D328566 */
const uint8_t secret_ctr_256[] =
{
    0x64, 0x3B, 0x91, 0xB4, 0xE5, 0x41, 0xB2, 0x0A, 0xAA, 0xEA, 0xB7, 0x7F, 0x2D,
0x32, 0x85, 0x66,
};
void netx_secure_crypto_aes256ctr_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_ctr_256,
```

```
                                    (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
             &handler,
             metadata,
             metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_aes_ctr_256,
                                    (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
                                          (uint8_t *) plain_ctr_256,
 sizeof(plain_ctr_256),
                                          (uint8_t *) iv_ctr_256,
                                          generated_ciphertext,
 sizeof(generated_ciphertext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, secret_ctr_256, sizeof
(secret_ctr_256));
    assert(0 == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                          handler,
                                          &crypto_method_aes_ctr_256,
                                    (uint8_t *) key_ctr_256,
```

```
crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
                                        (uint8_t *) secret_ctr_256,
 sizeof(secret_ctr_256),
                                        (uint8_t *) iv_ctr_256,
                                        generated_plaintext,
 sizeof(generated_plaintext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, plain_ctr_256, sizeof
(plain_ctr_256));
    assert(0 == err);
}
```

ECDSA Example

This is an example on using the NetX Crypto API to sign and verify input message data. Based on the hash algorithm selected a digest is computed of the plain input message before sign/verify.

```
extern NX_CRYPTO_METHOD crypto_method_ecdsa;
extern NX_CRYPTO_METHOD crypto_method_ec_secp256;
extern NX_CRYPTO_METHOD crypto_method_sha256; // NOLINT(readability-redundant-
declaration)
const uint8_t ECC_SECP256R1Keydata[] =
{
    0xf9, 0xa7, 0x68, 0x71, 0x24, 0x68, 0x9d, 0x32, 0x92, 0x6f, 0x1d, 0xfb, 0xbe,
0xf2, 0x61, 0x41, // NOLINT(readability-magic-numbers)
    0x07, 0x54, 0x0d, 0xb9, 0xa8, 0x8a, 0x8b, 0xc2, 0xd5, 0xe9, 0x38, 0x4b, 0xf9,
0xe5, 0x43, 0x5a // NOLINT(readability-magic-numbers)
};
const uint8_t ECC_SECP256R1PublicKeydata[] =
{
```

```
    0x04,
                /* ASN1 Constant */
    0x5b, 0xba, 0xd4, 0x2e, 0xb5, 0xc1, 0x07, 0xf2, 0x0e, 0x01, 0x95, 0x42, 0x6e,
0x90, 0xb8, 0x4e, // NOLINT(readability-magic-numbers)
    0xe9, 0x5a, 0xa1, 0xe8, 0x4c, 0x6c, 0xa5, 0x32, 0x3c, 0xf3, 0x09, 0xf5, 0xff,
0x8b, 0x3d, 0x26, // NOLINT(readability-magic-numbers)
    0xb6, 0x88, 0xc1, 0xdb, 0x02, 0xaf, 0x4d, 0xa5, 0x0e, 0x73, 0x61, 0x96, 0xb3,
0x59, 0x95, 0x6f, // NOLINT(readability-magic-numbers)
    0x5e, 0xc9, 0xa1, 0xf9, 0xb7, 0xb3, 0xb6, 0xdf, 0x54, 0x82, 0x79, 0xe3, 0xb6,
0x4e, 0xac, 0xb6 // NOLINT(readability-magic-numbers)
};
const uint8_t ECC_SECP256R1Message[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
void netx_secure_crypto_ecdsa_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_ECDSA)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_ECDSA);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
    ULONG sig_length;
    NX_CRYPTO_EXTENDED_OUTPUT extended_output;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdsa_init(&crypto_method_ecdsa, NX_CRYPTO_NULL, 0,
&handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set hash method. */
    err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_HASH_METHOD_SET,
                                            handler,
                                           &crypto_method_ecdsa,
                                            NX_CRYPTO_NULL,
                                            0,
```

```
                                          (uint8_t *) &crypto_method_sha256,
sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Set EC curve. */
   err =
      _nx_crypto_method_ecdsa_operation(NX_CRYPTO_EC_CURVE_SET,
                                        handler,
                                        &crypto_method_ecdsa,
                                        NX_CRYPTO_NULL,
                                        0,
                                        (uint8_t *) &crypto_method_ec_secp256,
sizeof(NX_CRYPTO_METHOD *),
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL,
                                        0,
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   extended_output.nx_crypto_extended_output_data          = output;
   extended_output.nx_crypto_extended_output_length_in_byte = sizeof(output);
/* Sign the hash data using ECDSA. */
   err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_SIGNATURE_GENERATE,
                                           handler,
                                          &crypto_method_ec_secp256,
                                          (uint8_t *) ECC_SECP256R1Keydata,
```

```
 sizeof(ECC_SECP256R1Keydata) << 3,
                                          (uint8_t *) ECC_SECP256R1Message,
 sizeof(ECC_SECP256R1Message),
       NX_CRYPTO_NULL,
                                          (uint8_t *) &extended_output,
 sizeof(extended_output),
       metadata,
       metadata_size,
       NX_CRYPTO_NULL,
       NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    sig_length = extended_output.nx_crypto_extended_output_actual_size;
 /* Verify the generated signature. */
    err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_SIGNATURE_VERIFY,
                                           handler,
                                          &crypto_method_ec_secp256,
                                          (uint8_t *) ECC_SECP256R1PublicKeydata,
 sizeof(ECC_SECP256R1PublicKeydata) << 3,
                                          (uint8_t *) ECC_SECP256R1Message,
 sizeof(ECC_SECP256R1Message),
       NX_CRYPTO_NULL,
       output,
       sig_length,
       metadata,
       metadata_size,
       NX_CRYPTO_NULL,
       NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
}
```

ECDH Example

This is an example on using the NetX Crypto API to generate a shared secret using ECDH. A shared secret is computed using known public key (from peer) and generated private key. Another shared secret is computed using the generated public key and known private key (imported to mimic peer). Both the shared secrets are checked to be the same.

```
extern NX_CRYPTO_METHOD crypto_method_ecdh;
/*Private key 59137e38152350b195c9718d39673d519838055ad908dd4757152fd8255c09bf */
const uint8_t ECC_SECP256R1Keydata_ecdh[] =
{
    0x59, 0x13, 0x7e, 0x38, 0x15, 0x23, 0x50, 0xb1, 0x95, 0xc9, 0x71, 0x8d, 0x39,
0x67, 0x3d, 0x51, // NOLINT(readability-magic-numbers)
    0x98, 0x38, 0x05, 0x5a, 0xd9, 0x08, 0xdd, 0x47, 0x57, 0x15, 0x2f, 0xd8, 0x25,
0x5c, 0x09, 0xbf, // NOLINT(readability-magic-numbers)
};
/*Public key 4, a8c5fdce8b62c5ada598f141adb3b26cf254c280b2857a63d2ad783a73115f6b,
806e1aafec4af80a0d786b3de45375b517a7e5b51ffb2c356537c9e6ef227d4a*/
const uint8_t ECC_SECP256R1PublicKeydata_ecdh[] =
{
    0x04,
    0xa8,0xc5, 0xfd, 0xce, 0x8b, 0x62, 0xc5, 0xad, 0xa5, 0x98, 0xf1, 0x41, 0xad,
0xb3, 0xb2, 0x6c, // NOLINT(readability-magic-numbers)
    0xf2,0x54, 0xc2, 0x80, 0xb2, 0x85, 0x7a, 0x63, 0xd2, 0xad, 0x78, 0x3a, 0x73,
0x11, 0x5f, 0x6b, // NOLINT(readability-magic-numbers)
    0x80,0x6e, 0x1a, 0xaf, 0xec, 0x4a, 0xf8, 0x0a, 0x0d, 0x78, 0x6b, 0x3d, 0xe4,
0x53, 0x75, 0xb5, // NOLINT(readability-magic-numbers)
    0x17,0xa7, 0xe5, 0xb5, 0x1f, 0xfb, 0x2c, 0x35, 0x65, 0x37, 0xc9, 0xe6, 0xef,
0x22, 0x7d, 0x4a, // NOLINT(readability-magic-numbers)
};
void netx_secure_crypto_ecdh_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_ECDH)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_ECDH);
    uint32_t err          = NX_CRYPTO_SUCCESS;
    uint8_t local_public_key[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    uint32_t local_public_key_len = 0;
    uint8_t shared_secret[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    uint32_t shared_secret_len = 0;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    NX_CRYPTO_EXTENDED_OUTPUT extended_output;
```

```
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdh_init(&crypto_method_ecdh, NX_CRYPTO_NULL, 0,
NX_CRYPTO_NULL, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_EC_CURVE_SET,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Generate local public key. This will generate a key pair.
  * The private wrapped key will be held by the ecdh context and the public key
(local_public_key)
  * will be returned for sharing with the peer.
  */
    extended_output.nx_crypto_extended_output_data          = local_public_key;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof
(local_public_key);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_SETUP,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
```

```
                                            NX_CRYPTO_NULL,
                                            0,
                                            NX_CRYPTO_NULL,
                                            0,
                                            NX_CRYPTO_NULL,
                                            (uint8_t *) &extended_output,
 sizeof(extended_output),
                                            metadata,
                                            metadata_size,
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    local_public_key_len = extended_output.nx_crypto_extended_output_actual_size;
 /* Calculate shared secret using the test (peer's) public key. */
    extended_output.nx_crypto_extended_output_data          = shared_secret;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof(shared_secret);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_CALCULATE,
                                            NX_CRYPTO_NULL,
                                            &crypto_method_ecdh,
                                            NX_CRYPTO_NULL,
                                            0,
                                            (uint8_t *)
ECC_SECP256R1PublicKeydata_ecdh,
 sizeof(ECC_SECP256R1PublicKeydata_ecdh),
                                            NX_CRYPTO_NULL,
                                            (uint8_t *) &extended_output,
 sizeof(extended_output),
                                            metadata,
                                            metadata_size,
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    shared_secret_len = extended_output.nx_crypto_extended_output_actual_size;
    err = _nx_crypto_method_ecdh_cleanup(metadata);
```

```
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify. The below operations will be carried out by the peer. */
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdh_init(&crypto_method_ecdh, NX_CRYPTO_NULL, 0,
NX_CRYPTO_NULL, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_EC_CURVE_SET,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Import the test private key. The peer could generate its own key pair,
  * in this example a test private key is used for simplicity. */
    err =
       _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_KEY_PAIR_IMPORT,
         NX_CRYPTO_NULL,
                                         &crypto_method_ecdh,
                                         (uint8_t *) ECC_SECP256R1Keydata_ecdh,
                                         (NX_CRYPTO_KEY_SIZE)
(sizeof(ECC_SECP256R1Keydata_ecdh) << 3),
                                         (uint8_t *) ECC_SECP256R1PublicKeydata_ecdh,
 sizeof(ECC_SECP256R1PublicKeydata_ecdh),
         NX_CRYPTO_NULL,
```

```
        NX_CRYPTO_NULL,
                                        0,
             metadata,
        metadata_size,
        NX_CRYPTO_NULL,
        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Calculate the shared secret using the local public key generated above and shared
with the peer. */
    extended_output.nx_crypto_extended_output_data          = output;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof(output);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_CALCULATE,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           local_public_key,
                                           local_public_key_len,
                                           NX_CRYPTO_NULL,
                                           (uint8_t *) &extended_output,
 sizeof(extended_output),
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Validate the output. Both the parties must generate the same shared secret */
    err = (extended_output.nx_crypto_extended_output_actual_size !=
shared_secret_len);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, shared_secret,
extended_output.nx_crypto_extended_output_actual_size);
    assert(NX_CRYPTO_SUCCESS == err);
}
```

RSA Example

This is an example on using the NetX Crypto API to encrypt and decrypt input message data.

```
extern NX_CRYPTO_METHOD crypto_method_rsa;
/* 00010001 */
const uint8_t public_e[] =
{
    0x00, 0x01, 0x00, 0x01,
};
/* 13FF7429F8E851F1079CCFCE3B3CD8606ABA8607AD85CBB3057501EBD58811F3C04823171F192C048E
1E883AF8CF958810151D3874AEDC8EC4F88D2065C581569F1E200852DD40B6DFD1652659085A9DD1D3B86
9EA3617D904D209DE156A60BA5929D02F16430273D10720C2F28D2B95684DCAA6B9F6A508EA2CBBC11B9F
3F30D6201EA6CFFBBF1C44255CEC58EE70DBC872442BCCF115D8F743557B5DE5F42DDDA6CEAE7977793CC
9D90ADFE65E520F5520B615CF3B8C2DC82D7AC75EDB1297CF38AB23A37EED18D4DD45D9AD051B26401BE8
6E8C8E53F9585A702D02F1B5BD65F6739DFA6BFFE560CA130B6F1D4779C556C06D9CD29FB72D8851904F9
CDEE9 */
const uint8_t private_e_2048[] =
{
    0x13, 0xFF, 0x74, 0x29, 0xF8, 0xE8, 0x51, 0xF1, 0x07, 0x9C, 0xCF, 0xCE, 0x3B,
0x3C, 0xD8, 0x60,
    0x6A, 0xBA, 0x86, 0x07, 0xAD, 0x85, 0xCB, 0xB3, 0x05, 0x75, 0x01, 0xEB, 0xD5,
0x88, 0x11, 0xF3,
    0xC0, 0x48, 0x23, 0x17, 0x1F, 0x19, 0x2C, 0x04, 0x8E, 0x1E, 0x88, 0x3A, 0xF8,
0xCF, 0x95, 0x88,
    0x10, 0x15, 0x1D, 0x38, 0x74, 0xAE, 0xDC, 0x8E, 0xC4, 0xF8, 0x8D, 0x20, 0x65,
0xC5, 0x81, 0x56,
    0x9F, 0x1E, 0x20, 0x08, 0x52, 0xDD, 0x40, 0xB6, 0xDF, 0xD1, 0x65, 0x26, 0x59,
0x08, 0x5A, 0x9D,
    0xD1, 0xD3, 0xB8, 0x69, 0xEA, 0x36, 0x17, 0xD9, 0x04, 0xD2, 0x09, 0xDE, 0x15,
0x6A, 0x60, 0xBA,
    0x59, 0x29, 0xD0, 0x2F, 0x16, 0x43, 0x02, 0x73, 0xD1, 0x07, 0x20, 0xC2, 0xF2,
0x8D, 0x2B, 0x95,
    0x68, 0x4D, 0xCA, 0xA6, 0xB9, 0xF6, 0xA5, 0x08, 0xEA, 0x2C, 0xBB, 0xC1, 0x1B,
```

```
0x9F, 0x3F, 0x30,
    0xD6, 0x20, 0x1E, 0xA6, 0xCF, 0xFB, 0xBF, 0x1C, 0x44, 0x25, 0x5C, 0xEC, 0x58,
0xEE, 0x70, 0xDB,
    0xC8, 0x72, 0x44, 0x2B, 0xCC, 0xF1, 0x15, 0xD8, 0xF7, 0x43, 0x55, 0x7B, 0x5D,
0xE5, 0xF4, 0x2D,
    0xDD, 0xA6, 0xCE, 0xAE, 0x79, 0x77, 0x79, 0x3C, 0xC9, 0xD9, 0x0A, 0xDF, 0xE6,
0x5E, 0x52, 0x0F,
    0x55, 0x20, 0xB6, 0x15, 0xCF, 0x3B, 0x8C, 0x2D, 0xC8, 0x2D, 0x7A, 0xC7, 0x5E,
0xDB, 0x12, 0x97,
    0xCF, 0x38, 0xAB, 0x23, 0xA3, 0x7E, 0xED, 0x18, 0xD4, 0xDD, 0x45, 0xD9, 0xAD,
0x05, 0x1B, 0x26,
    0x40, 0x1B, 0xE8, 0x6E, 0x8C, 0x8E, 0x53, 0xF9, 0x58, 0x5A, 0x70, 0x2D, 0x02,
0xF1, 0xB5, 0xBD,
    0x65, 0xF6, 0x73, 0x9D, 0xFA, 0x6B, 0xFF, 0xE5, 0x60, 0xCA, 0x13, 0x0B, 0x6F,
0x1D, 0x47, 0x79,
    0xC5, 0x56, 0xC0, 0x6D, 0x9C, 0xD2, 0x9F, 0xB7, 0x2D, 0x88, 0x51, 0x90, 0x4F,
0x9C, 0xDE, 0xE9,
};
/* E0F5059966A8AEC4BF7CDAC8AE2430BDF61C54D09CAB9963CBF9A52AC641E384B6431D3B6A9D181151
9A2904E1170A44446C80E7638A4AF2720A7654AB740D8A151FDD216F3D6933422FD9AC14AEDE9CCD021EA
79E46925F4B18FD1AF2C0073CFC3A69AC71A2B3673D08136CDB01C379892601C7C857D68018DAE924CB8C
D29377A14C752B92BAFF14C3A49725AE2FEFAAD4686D8A7D9F94EB11BF81E05BD5D2586526FB129E73539
F9223D496B2ACA23CCACC34D5B18533BD0F5815A76F94F4F55D965FE61599B44BD8FBAD35F42B612A4C4F
2765B2097A5C0090EA8166D9C6DA1E03B6119736B794600491C48433132D0F15D5DE3BB4270DF6BC9012B
74931 */
const uint8_t m_2048[] =
{
    0xE0, 0xF5, 0x05, 0x99, 0x66, 0xA8, 0xAE, 0xC4, 0xBF, 0x7C, 0xDA, 0xC8, 0xAE,
0x24, 0x30, 0xBD,
    0xF6, 0x1C, 0x54, 0xD0, 0x9C, 0xAB, 0x99, 0x63, 0xCB, 0xF9, 0xA5, 0x2A, 0xC6,
0x41, 0xE3, 0x84,
    0xB6, 0x43, 0x1D, 0x3B, 0x6A, 0x9D, 0x18, 0x11, 0x51, 0x9A, 0x29, 0x04, 0xE1,
0x17, 0x0A, 0x44,
    0x44, 0x6C, 0x80, 0xE7, 0x63, 0x8A, 0x4A, 0xF2, 0x72, 0x0A, 0x76, 0x54, 0xAB,
```

```
0x74, 0x0D, 0x8A,
    0x15, 0x1F, 0xDD, 0x21, 0x6F, 0x3D, 0x69, 0x33, 0x42, 0x2F, 0xD9, 0xAC, 0x14,
0xAE, 0xDE, 0x9C,
    0xCD, 0x02, 0x1E, 0xA7, 0x9E, 0x46, 0x92, 0x5F, 0x4B, 0x18, 0xFD, 0x1A, 0xF2,
0xC0, 0x07, 0x3C,
    0xFC, 0x3A, 0x69, 0xAC, 0x71, 0xA2, 0xB3, 0x67, 0x3D, 0x08, 0x13, 0x6C, 0xDB,
0x01, 0xC3, 0x79,
    0x89, 0x26, 0x01, 0xC7, 0xC8, 0x57, 0xD6, 0x80, 0x18, 0xDA, 0xE9, 0x24, 0xCB,
0x8C, 0xD2, 0x93,
    0x77, 0xA1, 0x4C, 0x75, 0x2B, 0x92, 0xBA, 0xFF, 0x14, 0xC3, 0xA4, 0x97, 0x25,
0xAE, 0x2F, 0xEF,
    0xAA, 0xD4, 0x68, 0x6D, 0x8A, 0x7D, 0x9F, 0x94, 0xEB, 0x11, 0xBF, 0x81, 0xE0,
0x5B, 0xD5, 0xD2,
    0x58, 0x65, 0x26, 0xFB, 0x12, 0x9E, 0x73, 0x53, 0x9F, 0x92, 0x23, 0xD4, 0x96,
0xB2, 0xAC, 0xA2,
    0x3C, 0xCA, 0xCC, 0x34, 0xD5, 0xB1, 0x85, 0x33, 0xBD, 0x0F, 0x58, 0x15, 0xA7,
0x6F, 0x94, 0xF4,
    0xF5, 0x5D, 0x96, 0x5F, 0xE6, 0x15, 0x99, 0xB4, 0x4B, 0xD8, 0xFB, 0xAD, 0x35,
0xF4, 0x2B, 0x61,
    0x2A, 0x4C, 0x4F, 0x27, 0x65, 0xB2, 0x09, 0x7A, 0x5C, 0x00, 0x90, 0xEA, 0x81,
0x66, 0xD9, 0xC6,
    0xDA, 0x1E, 0x03, 0xB6, 0x11, 0x97, 0x36, 0xB7, 0x94, 0x60, 0x04, 0x91, 0xC4,
0x84, 0x33, 0x13,
    0x2D, 0x0F, 0x15, 0xD5, 0xDE, 0x3B, 0xB4, 0x27, 0x0D, 0xF6, 0xBC, 0x90, 0x12,
0xB7, 0x49, 0x31,
};
/* 551C2E268F7ED44D0E8B063F5B2B510CB809F53BD54E9956971E243B2363DA123C29AB4A009EDE1FCE
C54625971A4E3490F3EA398BF7386AAC34720E43FB0C795445B520AEE4D7694EE1474F60F77E1B5F09FE2
ED004333658D212122F040322D1564512A1540400F27E18049A762A5EDC9F072CA4F49F408252D42B31BC
35523373740E90DDDA6A8CE7865EEB7C694A662C74412406AB190FE0435DA2551F0C24A48939DDA58A023
9706D40B4977473689DC36CE5A4DF4EF892816CBDE2780D9389B7384674C93B1DDAF728F292B5671679FC
7175AC0A3B2197B809E7CF410417010F3B1316D10D82466C62F3A01667B70A714E0499400E255D4C39EA7
DE55C */
const uint8_t plain_2048[] =
```

```
{
    0x55, 0x1C, 0x2E, 0x26, 0x8F, 0x7E, 0xD4, 0x4D, 0x0E, 0x8B, 0x06, 0x3F, 0x5B,
0x2B, 0x51, 0x0C,
    0xB8, 0x09, 0xF5, 0x3B, 0xD5, 0x4E, 0x99, 0x56, 0x97, 0x1E, 0x24, 0x3B, 0x23,
0x63, 0xDA, 0x12,
    0x3C, 0x29, 0xAB, 0x4A, 0x00, 0x9E, 0xDE, 0x1F, 0xCE, 0xC5, 0x46, 0x25, 0x97,
0x1A, 0x4E, 0x34,
    0x90, 0xF3, 0xEA, 0x39, 0x8B, 0xF7, 0x38, 0x6A, 0xAC, 0x34, 0x72, 0x0E, 0x43,
0xFB, 0x0C, 0x79,
    0x54, 0x45, 0xB5, 0x20, 0xAE, 0xE4, 0xD7, 0x69, 0x4E, 0xE1, 0x47, 0x4F, 0x60,
0xF7, 0x7E, 0x1B,
    0x5F, 0x09, 0xFE, 0x2E, 0xD0, 0x04, 0x33, 0x36, 0x58, 0xD2, 0x12, 0x12, 0x2F,
0x04, 0x03, 0x22,
    0xD1, 0x56, 0x45, 0x12, 0xA1, 0x54, 0x04, 0x00, 0xF2, 0x7E, 0x18, 0x04, 0x9A,
0x76, 0x2A, 0x5E,
    0xDC, 0x9F, 0x07, 0x2C, 0xA4, 0xF4, 0x9F, 0x40, 0x82, 0x52, 0xD4, 0x2B, 0x31,
0xBC, 0x35, 0x52,
    0x33, 0x73, 0x74, 0x0E, 0x90, 0xDD, 0xDA, 0x6A, 0x8C, 0xE7, 0x86, 0x5E, 0xEB,
0x7C, 0x69, 0x4A,
    0x66, 0x2C, 0x74, 0x41, 0x24, 0x06, 0xAB, 0x19, 0x0F, 0xE0, 0x43, 0x5D, 0xA2,
0x55, 0x1F, 0x0C,
    0x24, 0xA4, 0x89, 0x39, 0xDD, 0xA5, 0x8A, 0x02, 0x39, 0x70, 0x6D, 0x40, 0xB4,
0x97, 0x74, 0x73,
    0x68, 0x9D, 0xC3, 0x6C, 0xE5, 0xA4, 0xDF, 0x4E, 0xF8, 0x92, 0x81, 0x6C, 0xBD,
0xE2, 0x78, 0x0D,
    0x93, 0x89, 0xB7, 0x38, 0x46, 0x74, 0xC9, 0x3B, 0x1D, 0xDA, 0xF7, 0x28, 0xF2,
0x92, 0xB5, 0x67,
    0x16, 0x79, 0xFC, 0x71, 0x75, 0xAC, 0x0A, 0x3B, 0x21, 0x97, 0xB8, 0x09, 0xE7,
0xCF, 0x41, 0x04,
    0x17, 0x01, 0x0F, 0x3B, 0x13, 0x16, 0xD1, 0x0D, 0x82, 0x46, 0x6C, 0x62, 0xF3,
0xA0, 0x16, 0x67,
    0xB7, 0x0A, 0x71, 0x4E, 0x04, 0x99, 0x40, 0x0E, 0x25, 0x5D, 0x4C, 0x39, 0xEA,
0x7D, 0xE5, 0x5C,
};
```

```
/* 10F904E071338569EC131401A7869F42F3BCAE252B5D3C8755FD24D47997A9CD4221D992B2871E0528
3B98841FC5C379C5D0E35B3938279B344299C3CF1566E0C994D0A9013AF64174F1379A4B5E4E9DE57491F
3078F6D10011EA55535D0763E538662C9996F4FCF8B64A768685AA417ADB6978743D3D1F513CF143DD6D3
83AD6357728A88928D39E27EA4D0B2AF92FC7F63875F9D6A70FAE7993C1FF04DF9A2F99216874BC123D4B
7DA7E7E8974CFC10ACF0C7BC8747526A8D16791F969082EA9B0C36D77B67C37B325682D74178E4234D52D
5635273301A6CC35E315AE74D659B1433576DAAE6780FA39E0550D971F2CB5817CAAFC24B5220E21C8CEE
E85DD */
const uint8_t secret_2048[] =
{
    0x10, 0xF9, 0x04, 0xE0, 0x71, 0x33, 0x85, 0x69, 0xEC, 0x13, 0x14, 0x01, 0xA7,
0x86, 0x9F, 0x42,
    0xF3, 0xBC, 0xAE, 0x25, 0x2B, 0x5D, 0x3C, 0x87, 0x55, 0xFD, 0x24, 0xD4, 0x79,
0x97, 0xA9, 0xCD,
    0x42, 0x21, 0xD9, 0x92, 0xB2, 0x87, 0x1E, 0x05, 0x28, 0x3B, 0x98, 0x84, 0x1F,
0xC5, 0xC3, 0x79,
    0xC5, 0xD0, 0xE3, 0x5B, 0x39, 0x38, 0x27, 0x9B, 0x34, 0x42, 0x99, 0xC3, 0xCF,
0x15, 0x66, 0xE0,
    0xC9, 0x94, 0xD0, 0xA9, 0x01, 0x3A, 0xF6, 0x41, 0x74, 0xF1, 0x37, 0x9A, 0x4B,
0x5E, 0x4E, 0x9D,
    0xE5, 0x74, 0x91, 0xF3, 0x07, 0x8F, 0x6D, 0x10, 0x01, 0x1E, 0xA5, 0x55, 0x35,
0xD0, 0x76, 0x3E,
    0x53, 0x86, 0x62, 0xC9, 0x99, 0x6F, 0x4F, 0xCF, 0x8B, 0x64, 0xA7, 0x68, 0x68,
0x5A, 0xA4, 0x17,
    0xAD, 0xB6, 0x97, 0x87, 0x43, 0xD3, 0xD1, 0xF5, 0x13, 0xCF, 0x14, 0x3D, 0xD6,
0xD3, 0x83, 0xAD,
    0x63, 0x57, 0x72, 0x8A, 0x88, 0x92, 0x8D, 0x39, 0xE2, 0x7E, 0xA4, 0xD0, 0xB2,
0xAF, 0x92, 0xFC,
    0x7F, 0x63, 0x87, 0x5F, 0x9D, 0x6A, 0x70, 0xFA, 0xE7, 0x99, 0x3C, 0x1F, 0xF0,
0x4D, 0xF9, 0xA2,
    0xF9, 0x92, 0x16, 0x87, 0x4B, 0xC1, 0x23, 0xD4, 0xB7, 0xDA, 0x7E, 0x7E, 0x89,
0x74, 0xCF, 0xC1,
    0x0A, 0xCF, 0x0C, 0x7B, 0xC8, 0x74, 0x75, 0x26, 0xA8, 0xD1, 0x67, 0x91, 0xF9,
0x69, 0x08, 0x2E,
    0xA9, 0xB0, 0xC3, 0x6D, 0x77, 0xB6, 0x7C, 0x37, 0xB3, 0x25, 0x68, 0x2D, 0x74,
```

```
0x17, 0x8E, 0x42,
    0x34, 0xD5, 0x2D, 0x56, 0x35, 0x27, 0x33, 0x01, 0xA6, 0xCC, 0x35, 0xE3, 0x15,
0xAE, 0x74, 0xD6,
    0x59, 0xB1, 0x43, 0x35, 0x76, 0xDA, 0xAE, 0x67, 0x80, 0xFA, 0x39, 0xE0, 0x55,
0x0D, 0x97, 0x1F,
    0x2C, 0xB5, 0x81, 0x7C, 0xAA, 0xFC, 0x24, 0xB5, 0x22, 0x0E, 0x21, 0xC8, 0xCE,
0xEE, 0x85, 0xDD,
};
void netx_secure_crypto_rsa_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_RSA)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_RSA);
    uint32_t err          = NX_CRYPTO_SUCCESS;
 void   * handler       = NX_CRYPTO_NULL;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err =
       _nx_crypto_method_rsa_init(&crypto_method_rsa,
                                  (uint8_t *) m_2048,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(m_2048)),
             &handler,
             metadata,
             metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
    err = _nx_crypto_method_rsa_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_rsa,
                                          (uint8_t *) public_e,
```

```
                    RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(public_e)),
                                        (uint8_t *) plain_2048,
 sizeof(m_2048),
                                        NX_CRYPTO_NULL,
        output,
 sizeof(m_2048),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   err = (uint32_t) memcmp(output, secret_2048, sizeof(m_2048));
   assert(0 == err);
   err = _nx_crypto_method_rsa_cleanup(metadata);
   assert(NX_CRYPTO_SUCCESS == err);
 /* Decryption. */
   memset(output, 0, sizeof(output));
   err =
      _nx_crypto_method_rsa_init(&crypto_method_rsa,
                                 (uint8_t *) m_2048,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(m_2048)),
             &handler,
             metadata,
             metadata_size);
   assert(NX_CRYPTO_SUCCESS == err);
   err = _nx_crypto_method_rsa_operation(NX_CRYPTO_DECRYPT,
                                        handler,
                                        &crypto_method_rsa,
                                        (uint8_t *) private_e_2048,



                    RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(private_e_2048)),
                                        (uint8_t *) secret_2048,
```

```
 sizeof(m_2048),
                                        NX_CRYPTO_NULL,
        output,
 sizeof(m_2048),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, plain_2048, sizeof(m_2048));
    assert(0 == err);
}
```

RSA PKCS1V1.5 Example

This is an example on using the NetX Crypto API to sign and verify input message data. The plain input message is PKCS1V1.5 encoded before signature generation.

PKCS1V15 Example

HMAC SHA256 Example

This is an example on using the HMAC with SHA256 hash using the NetX Crypto API.

```
extern NX_CRYPTO_METHOD crypto_method_hmac_sha256;
/*
C4DA057B81EA740B697FFE1B6EB8591356BA6D5EA7F1B96E4F048030449ACD64E4BB271CB4DCF94937E6
*/
const uint8_t key_256[] =
{
    0xC4, 0xDA, 0x05, 0x7B, 0x81, 0xEA, 0x74, 0x0B, 0x69, 0x7F, 0xFE, 0x1B, 0x6E,
0xB8, 0x59, 0x13,
    0x56, 0xBA, 0x6D, 0x5E, 0xA7, 0xF1, 0xB9, 0x6E, 0x4F, 0x04, 0x80, 0x30, 0x44,
0x9A, 0xCD, 0x64,
    0xE4, 0xBB, 0x27, 0x1C, 0xB4, 0xDC, 0xF9, 0x49, 0x37, 0xE6,
};
/* BDACB6555D294D3AFFC245520116062D98F88D64276BDA593492AE71CFE16E46CABC287CB00DF21D96
066D5856C2224EEF609D4896302540078F3A0EE325F5337E */
```

```
const uint8_t plain_256[] =
{
    0xBD, 0xAC, 0xB6, 0x55, 0x5D, 0x29, 0x4D, 0x3A, 0xFF, 0xC2, 0x45, 0x52, 0x01,
0x16, 0x06, 0x2D,
    0x98, 0xF8, 0x8D, 0x64, 0x27, 0x6B, 0xDA, 0x59, 0x34, 0x92, 0xAE, 0x71, 0xCF,
0xE1, 0x6E, 0x46,
    0xCA, 0xBC, 0x28, 0x7C, 0xB0, 0x0D, 0xF2, 0x1D, 0x96, 0x06, 0x6D, 0x58, 0x56,
0xC2, 0x22, 0x4E,
    0xEF, 0x60, 0x9D, 0x48, 0x96, 0x30, 0x25, 0x40, 0x07, 0x8F, 0x3A, 0x0E, 0xE3,
0x25, 0xF5, 0x33,
    0x7E,
};
/* 940F986AC891C9000B72EF0CEC69AB66AF002E3A34EB8A3A5F94484E45C0396C */
const uint8_t secret_256[] =
{
    0x94, 0x0F, 0x98, 0x6A, 0xC8, 0x91, 0xC9, 0x00, 0x0B, 0x72, 0xEF, 0x0C, 0xEC,
0x69, 0xAB, 0x66,
    0xAF, 0x00, 0x2E, 0x3A, 0x34, 0xEB, 0x8A, 0x3A, 0x5F, 0x94, 0x48, 0x4E, 0x45,
0xC0, 0x39, 0x6C,
};
void netx_secure_crypto_hmac_sha256_example (void)
{
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES] = {0};
    uint8_t metadata[sizeof(NX_CRYPTO_SHA256_HMAC)] = {0};
    uint32_t metadata_size = sizeof(NX_CRYPTO_SHA256_HMAC);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 init */
    err = _nx_crypto_method_hmac_sha256_init(&crypto_method_hmac_sha256,
                                             (UCHAR *) key_256,
```

```
                RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(key_256)),
            &handler,
                                        metadata,
            metadata_size);
   assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_INITIALIZE */
   err =
      _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_INITIALIZE,
                                              handler,
                                               &crypto_method_hmac_sha256,
                                              (UCHAR *) key_256,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(key_256)),
                                              NX_CRYPTO_NULL,
                                              0,
                                              NX_CRYPTO_NULL,
                                              NX_CRYPTO_NULL,
                                              0,
                                              metadata,
                                              metadata_size,
                                              NX_CRYPTO_NULL,
                                              NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_UPDATE */
   err = _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_UPDATE,
                                              handler,
                                                 &crypto_method_hmac_sha256,
                                              NX_CRYPTO_NULL,
                                              0,
                                                (UCHAR *) plain_256,
 sizeof(plain_256),
                                              NX_CRYPTO_NULL,
```

```
                                                NX_CRYPTO_NULL,
                                                0,
                                                metadata,
                                                metadata_size,
                                                NX_CRYPTO_NULL,
                                                NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_CALCULATE */
    err = _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_CALCULATE,
                                                handler,
                                                   &crypto_method_hmac_sha256,
                                                NX_CRYPTO_NULL,
                                                0,
                                                NX_CRYPTO_NULL,
                                                0,
                                                NX_CRYPTO_NULL,
                                                  (UCHAR *) output,
 sizeof(output),
                                                metadata,
                                                metadata_size,
                                                NX_CRYPTO_NULL,
                                                NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Ensure generated HMAC-SHA256 mac matches the expected mac */
    err = (uint32_t) memcmp(output, secret_256, sizeof(secret_256));
    assert(0 == err);
}
```

5.2.15.4 Mbed Crypto H/W Acceleration (rm_psa_crypto)

Modules » Security

Functions

fsp_err_t	RM_PSA_CRYPTO_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes, uint32_t *p_num_gen_bytes)

	Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer. More...
int	mbedtls_platform_setup (mbedtls_platform_context *ctx)
void	mbedtls_platform_teardown (mbedtls_platform_context *ctx)

Detailed Description

Hardware acceleration for the mbedCrypto implementation of the Arm PSA Crypto API.

Overview

Note

The PSA Crypto module does not provide any interfaces to the user. This release uses the mbedTLS version 3.6.6 which conforms to the PSA Crypto API 1.0 specification. Consult the Arm documentation at https://armmbed.github.io/mbed-crypto/psa/#application-programming-interface for further information. *FSP 3.0 onward adopts a change by Arm where mbedCrypto has been integrated back to MbedTLS and the term mbedCrypto has been deprecated. The mbedCrypto term in FSP now refers to the crypto portion of the MbedTLS module.*

HW Overview

Crypto Peripheral version	Devices
RSIP-E50D	RX74M, RX74N

Features

		AES Engine	AES Engine 2	SCE5	SCE5_B	SCE7	SCE9	RSIP-E11A	RSIP-E31A	RSIP-E51A	RSIP-E50D
TRNG		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Hash	SHA224, SHA256	–	–	–	–	Yes	Yes	Yes	Yes	Yes	Yes
	SHA384, SHA512	–	–	–	–	–	–	–	Yes	Yes	Yes
	SHA3-(224/256/384/512)	–	–	–	–	–	–	–	–	–	Yes
MAC	HMAC-SHA224, HMAC-SHA256	–	–	–	–	Yes	Yes	Yes	Yes	Yes	Yes

	HMAC-SHA384, HMAC-SHA512	–	–	–	–	–	–	–	Yes	Yes	Yes
	AES128-CMAC	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES192-CMAC	–	–	–	–	–	–	–	–	Yes	Yes
	AES256-CMAC	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES-CMAC-PRF-128	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
AES	AES128-ECB, CBC, CTR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES128-XTS	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	AES128-GCM, GMAC, CCM	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES192-ECB, CBC, CTR	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	AES192-GCM, GMAC, CCM	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	AES256-ECB, CBC, CTR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	AES256-XTS	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	AES256-GCM,	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

	GMAC, CCM										
	Key generation - plaintext	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Key generation - wrapped	–	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ChaCha	ChaCha20	–	–	–	–	–	–	–	–	–	Yes
	ChaCha20-Poly1305	–	–	–	–	–	–	–	–	–	Yes
ECC	secp256r1	–	–	–	–	Yes	Yes	Yes	Yes	Yes	Yes
	secp384r1	–	–	–	–	Yes	Yes	–	Yes	Yes	Yes
	secp521r1	–	–	–	–	–	–	–	–	Yes	Yes
	Brainpool256r1	–	–	–	–	Yes	Yes	–	Yes	Yes	Yes
	Brainpool384r1	–	–	–	–	Yes	Yes	–	Yes	Yes	Yes
	Key generation - plaintext	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	Key generation - wrapped	–	–	–	–	–	Yes	Yes	Yes	Yes	Yes
RSA	RSA-1024	–	–	–	–	–	Yes	–	–	Yes	Yes
	RSA-2048	–	–	–	–	Yes	Yes	–	–	Yes	Yes
	RSA-3072	–	–	–	–	–	(1)	–	–	Yes	Yes

	RSA-4096	–	–	–	–	–	(1)	–	–	Yes	Yes
	Key generation - plaintext	–	–	–	–	Yes	(2)	–	–	(3)	(3)
	Key generation - wrapped	–	–	–	–	Yes	(2)	–	–	(3)	(3)
ML-KEM	Key generation - plaintext	–	–	–	–	–	–	–	–	–	(4)
	ML-KEM-512	–	–	–	–	–	–	–	–	–	(4)
	ML-KEM-768	–	–	–	–	–	–	–	–	–	(4)
	ML-KEM-1024	–	–	–	–	–	–	–	–	–	(4)

Note

1. Signature verification only
2. RSA-1024 and RSA-2048 only
3. RSA-2048, RSA-3072, RSA-4096 only
4. SHA3 only

Configuration

Build Time Configurations for mbedCrypto

The following build time configurations are defined in arm/mbedtls/config.h:

Configuration	Options	Default	Description
Hardware Acceleration > Key Format > AES	MCU Specific Options		Select AES key formats used
Hardware Acceleration > Key Format > ECC	MCU Specific Options		Select ECC key formats used
Hardware Acceleration > Key Format > RSA	MCU Specific Options		Select RSA key formats used

Hardware Acceleration > Hash > SHA256/224	MCU Specific Options	Defines MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT.
Hardware Acceleration > Hash > SHA512/384	MCU Specific Options	Defines MBEDTLS_SHA512_ALT and MBEDTLS_SHA512_PROCESS_ALT.
Hardware Acceleration > Hash > SHA3_224/256/384/512	MCU Specific Options	Defines MBEDTLS_SHA3_ALT and MBEDTLS_SHA3_PROCESS_ALT.
Hardware Acceleration > Cipher > AES	MCU Specific Options	Defines MBEDTLS_AES_ALT, MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 3072 > Key Generation	MCU Specific Options	Enables RSA 3072 Key Generation.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 3072 > Signing	MCU Specific Options	Enables RSA 3072 Key Signing.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 3072 > Verification	MCU Specific Options	Enables RSA 3072 Verify.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 4096 > Key Generation	MCU Specific Options	Enables RSA 4096 Key Generation.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 4096 > Signing	MCU Specific Options	Enables RSA 4096 Key Signing.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 4096 > Verification	MCU Specific Options	Enables RSA 4096 Verify.
Hardware Acceleration	MCU Specific Options	Defines

> Public Key Cryptography (PKC) > ECC			MBEDTLS_ECP_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECDSA	MCU Specific Options		Defines MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECDH	MCU Specific Options		Defines MBEDTLS_ECDH_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 1024	MCU Specific Options		Defines MBEDTLS_RSA_1024_ALT. RSA 1024 Key Generation, Signing and Verification are also enabled.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 2048	MCU Specific Options		Defines MBEDTLS_RSA_ALT. RSA 2048 Key Generation, Signing and Verification are also enabled.
Hardware Acceleration > TRNG	Enabled	Enabled	Defines MBEDTLS_ENTROPY_HARDWARE_ALT.
Hardware Acceleration > Crypto Engine Initialization	Enabled	Enabled	MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT
Platform > Alternate > MBEDTLS_PLATFORM_SETBUF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_SETBUF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_EXIT_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_EXIT_ALT
Platform > Alternate > MBEDTLS_PLATFORM_TIME_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_TIME_ALT
Platform > Alternate > MBEDTLS_PLATFORM_FPRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_FPRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_PRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_PRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_SNPRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_SNPRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_V	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_VSNPRINTF_ALT

SNPRINTF_ALT			
Platform > Alternate > MBEDTLS_PLATFORM_NV_SEED_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NV_SEED_ALT
Platform > Alternate > MBEDTLS_PLATFORM_MS_TIME_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_MS_TIME_ALT
Platform > Alternate > MBEDTLS_PLATFORM_ZEROIZE_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_ZEROIZE_ALT
Platform > Alternate > MBEDTLS_PLATFORM_GMTIME_R_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_GMTIME_R_ALT
Platform > MBEDTLS_HAVE_ASM	• Define • Undefine	Undefine	MBEDTLS_HAVE_ASM
Platform > MBEDTLS_NO_UDBL_DIVISION	• Define • Undefine	Undefine	MBEDTLS_NO_UDBL_DIVISION
Platform > MBEDTLS_NO_64BIT_MULTIPLICATION	• Define • Undefine	Undefine	MBEDTLS_NO_64BIT_MULTIPLICATION
Platform > MBEDTLS_HAVE_SSE2	• Define • Undefine	Undefine	MBEDTLS_HAVE_SSE2
Platform > MBEDTLS_HAVE_TIME	• Define • Undefine	Undefine	MBEDTLS_HAVE_TIME
Platform > MBEDTLS_HAVE_TIME_DATE	• Define • Undefine	Undefine	MBEDTLS_HAVE_TIME_DATE
Platform > MBEDTLS_PLATFORM_MEMORY	• Define • Undefine	Define	MBEDTLS_PLATFORM_MEMORY
Platform > MBEDTLS_PLATFORM_NO_STD_FUNCTIONS	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NO_STD_FUNCTIONS
Platform > MBEDTLS_TIMING_ALT	• Define • Undefine	Undefine	MBEDTLS_TIMING_ALT
Platform > MBEDTLS_NO_PLATFORM_ENTROPY	• Define • Undefine	Define	MBEDTLS_NO_PLATFORM_ENTROPY
Platform > MBEDTLS_ENTROPY_C	• Define • Undefine	Define	MBEDTLS_ENTROPY_C
Platform > MBEDTLS_PLATFORM_C	• Define • Undefine	Define	MBEDTLS_PLATFORM_C
Platform > MBEDTLS_PLATFORM_STD_CALLOC	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_CALLOC
Platform > MBEDTLS_PLATFORM_STD_CALLOC value	Manual Entry	calloc	MBEDTLS_PLATFORM_STD_CALLOC value

Platform > MBEDTLS_PLATFORM_STD_FREE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_FREE
Platform > MBEDTLS_PLATFORM_STD_FREE value	Manual Entry	free	MBEDTLS_PLATFORM_STD_FREE value
Platform > MBEDTLS_PLATFORM_STD_SETBUF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_SETBUF
Platform > MBEDTLS_PLATFORM_STD_SETBUF value	Manual Entry	setbuf	MBEDTLS_PLATFORM_STD_SETBUF value
Platform > MBEDTLS_PLATFORM_STD_EXIT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_EXIT
Platform > MBEDTLS_PLATFORM_STD_EXIT value	Manual Entry	exit	MBEDTLS_PLATFORM_STD_EXIT value
Platform > MBEDTLS_PLATFORM_STD_TIME	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_TIME
Platform > MBEDTLS_PLATFORM_STD_TIME value	Manual Entry	time	MBEDTLS_PLATFORM_STD_TIME value
Platform > MBEDTLS_PLATFORM_STD_FPRINTF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_FPRINTF
Platform > MBEDTLS_PLATFORM_STD_FPRINTF value	Manual Entry	fprintf	MBEDTLS_PLATFORM_STD_FPRINTF value
Platform > MBEDTLS_PLATFORM_STD_PRINTF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_PRINTF
Platform > MBEDTLS_PLATFORM_STD_PRINTF value	Manual Entry	printf	MBEDTLS_PLATFORM_STD_PRINTF value
Platform > MBEDTLS_PLATFORM_STD_SNPRINTF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_SNPRINTF
Platform > MBEDTLS_PLATFORM_STD_SNPRINTF value	Manual Entry	snprintf	MBEDTLS_PLATFORM_STD_SNPRINTF value
Platform > MBEDTLS_PLATFORM_STD_EXIT_SUCCESS	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_EXIT_SUCCESS
Platform > MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value	Manual Entry	0	MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value
Platform > MBEDTLS_PLATFORM_STD_EXIT_FA	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_EXIT_FAILURE

ILURE			
Platform > MBEDTLS_PLATFORM_STD_EXIT_FAILURE value	Manual Entry	1	MBEDTLS_PLATFORM_STD_EXIT_FAILURE value
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_READ	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_READ
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_READ value	Manual Entry	mbedtls_platform_std_nv_seed_read	MBEDTLS_PLATFORM_STD_NV_SEED_READ value
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_WRITE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_WRITE
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value	Manual Entry	mbedtls_platform_std_nv_seed_write	MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_FILE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_FILE
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_FILE value	Manual Entry		MBEDTLS_PLATFORM_STD_NV_SEED_FILE value
Platform > MBEDTLS_PLATFORM_CALLOC_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_CALLOC_MACRO
Platform > MBEDTLS_PLATFORM_CALLOC_MACRO value	Manual Entry	calloc	MBEDTLS_PLATFORM_CALLOC_MACRO value
Platform > MBEDTLS_PLATFORM_FREE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_FREE_MACRO
Platform > MBEDTLS_PLATFORM_FREE_MACRO value	Manual Entry	free	MBEDTLS_PLATFORM_FREE_MACRO value
Platform > MBEDTLS_PLATFORM_EXIT_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_EXIT_MACRO
Platform > MBEDTLS_PLATFORM_EXIT_MACRO value	Manual Entry	exit	MBEDTLS_PLATFORM_EXIT_MACRO value
Platform > MBEDTLS_PLATFORM_SETBUF_MACRO	• Define • Undefine	Define	MBEDTLS_PLATFORM_SETBUF_MACRO
Platform > MBEDTLS_PLATFORM_SETBUF_MACRO value	Manual Entry	dummy_setbuf	MBEDTLS_PLATFORM_SETBUF_MACRO value

Platform > MBEDTLS_PLATFORM_TIME_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_TIME_MACRO
Platform > MBEDTLS_PLATFORM_TIME_MACRO value	Manual Entry	time	MBEDTLS_PLATFORM_TIME_MACRO value
Platform > MBEDTLS_PLATFORM_TIME_TYPE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_TIME_TYPE_MACRO
Platform > MBEDTLS_PLATFORM_TIME_TYPE_MACRO value	Manual Entry	time_t	MBEDTLS_PLATFORM_TIME_TYPE_MACRO value
Platform > MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO
Platform > MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO value	Manual Entry	int64_t	MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO value
Platform > MBEDTLS_PRINTF_MS_TIME	• Define • Undefine	Undefine	MBEDTLS_PRINTF_MS_TIME
Platform > MBEDTLS_PLATFORM_DEV_RANDOM	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_DEV_RANDOM
Platform > MBEDTLS_PRINTF_MS_TIME value	Manual Entry	PRId64	MBEDTLS_PRINTF_MS_TIME value
Platform > MBEDTLS_PLATFORM_FPRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_FPRINTF_MACRO
Platform > MBEDTLS_PLATFORM_FPRINTF_MACRO value	Manual Entry	fprintf	MBEDTLS_PLATFORM_FPRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_PRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_PRINTF_MACRO
Platform > MBEDTLS_PLATFORM_PRINTF_MACRO value	Manual Entry	printf	MBEDTLS_PLATFORM_PRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_SNPRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_SNPRINTF_MACRO
Platform > MBEDTLS_PLATFORM_SNPRINTF_MACRO value	Manual Entry	snprintf	MBEDTLS_PLATFORM_SNPRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_VSNPRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_VSNPRINTF_MACRO

Platform > MBEDTLS_PLATFORM_VSNPRINTF_MACRO value	Manual Entry	vsnprintf	MBEDTLS_PLATFORM_VSNPRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_NV_SEED_READ_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NV_SEED_READ_MACRO
Platform > MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value	Manual Entry	mbedtls_platform_std_nv_seed_read	MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value
Platform > MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO
Platform > MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value	Manual Entry	mbedtls_platform_std_nv_seed_write	MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value
General > MBEDTLS_PSA_CRYPTO_DRIVERS	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_DRIVERS
General > MBEDTLS_DEPRECATED_WARNING	• Define • Undefine	Undefine	MBEDTLS_DEPRECATED_WARNING
General > MBEDTLS_DEPRECATED_REMOVED	• Define • Undefine	Define	MBEDTLS_DEPRECATED_REMOVED
General > MBEDTLS_CHECK_RETURN_WARNING	• Define • Undefine	Undefine	MBEDTLS_CHECK_RETURN_WARNING
General > MBEDTLS_ERROR_STRERROR_DUMMY	• Define • Undefine	Define	MBEDTLS_ERROR_STRERROR_DUMMY
General > MBEDTLS_MEMORY_DEBUG	• Define • Undefine	Undefine	MBEDTLS_MEMORY_DEBUG
General > MBEDTLS_MEMORY_BACKTRACE	• Define • Undefine	Undefine	MBEDTLS_MEMORY_BACKTRACE
General > MBEDTLS_PSA_CRYPTO_CLIENT	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_CLIENT
General > MBEDTLS_PSA_CRYPTO_SPM	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_SPM
General > MBEDTLS_PSA_KEY_STORE_DYNAMIC	• Define • Undefine	Undefine	MBEDTLS_PSA_KEY_STORE_DYNAMIC
General > MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS	• Define • Undefine	Undefine	MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS
General > MBEDTLS_SELF_TEST	• Define • Undefine	Undefine	MBEDTLS_SELF_TEST
General > MBEDTLS_T	• Define	Define	MBEDTLS_THREADING_

HREADING_ALT	• Undefine		ALT
General > MBEDTLS_THREADING_PTHREAD	• Define • Undefine	Undefine	MBEDTLS_THREADING_PTHREAD
General > MBEDTLS_USE_PSA_CRYPTO	Undefine	Undefine	MBEDTLS_USE_PSA_CRYPTO
General > MBEDTLS_VERSION_FEATURES	• Define • Undefine	Define	MBEDTLS_VERSION_FEATURES
General > MBEDTLS_ERROR_C	• Define • Undefine	Define	MBEDTLS_ERROR_C
General > MBEDTLS_MEMORY_BUFFER_ALLOC_C	• Define • Undefine	Undefine	MBEDTLS_MEMORY_BUFFER_ALLOC_C
General > MBEDTLS_PSA_CRYPTO_C	• Define • Undefine	Define	MBEDTLS_PSA_CRYPTO_C
General > MBEDTLS_PSA_CRYPTO_SE_C	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_SE_C
General > MBEDTLS_THREADING_C	• Define • Undefine	Define	MBEDTLS_THREADING_C
General > MBEDTLS_TIMING_C	• Define • Undefine	Undefine	MBEDTLS_TIMING_C
General > MBEDTLS_VERSION_C	• Define • Undefine	Define	MBEDTLS_VERSION_C
General > MBEDTLS_MEMORY_ALIGN_MULTIPLE	• Define • Undefine	Undefine	MBEDTLS_MEMORY_ALIGN_MULTIPLE
General > MBEDTLS_MEMORY_ALIGN_MULTIPLE value	Manual Entry	4	MBEDTLS_MEMORY_ALIGN_MULTIPLE value
General > MBEDTLS_CHECK_RETURN	• Define • Undefine	Define	MBEDTLS_CHECK_RETURN
General > MBEDTLS_IGNORE_RETURN	• Define • Undefine	Undefine	MBEDTLS_IGNORE_RETURN
General > MBEDTLS_PSA_CRYPTO_CONFIG	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_CONFIG
Cipher > Alternate > MBEDTLS_ARIA_ALT	• Define • Undefine	Undefine	MBEDTLS_ARIA_ALT
Cipher > Alternate > MBEDTLS_CAMELLIA_ALT	• Define • Undefine	Undefine	MBEDTLS_CAMELLIA_ALT
Cipher > Alternate > MBEDTLS_CCM_ALT	MCU Specific Options		MBEDTLS_CCM_ALT
Cipher > Alternate > MBEDTLS_CHACHA20_ALT	MCU Specific Options		MBEDTLS_CHACHA20_ALT

Cipher > Alternate > MBEDTLS_CHACHAPOLY_ALT	MCU Specific Options		MBEDTLS_CHACHAPOLY_ALT
Cipher > Alternate > MBEDTLS_CMAC_ALT	MCU Specific Options		MBEDTLS_CMAC_ALT
Cipher > Alternate > MBEDTLS_DES_ALT	• Define • Undefine	Undefine	MBEDTLS_DES_ALT
Cipher > Alternate > MBEDTLS_GCM_ALT	MCU Specific Options		MBEDTLS_GCM_ALT
Cipher > Alternate > MBEDTLS_NIST_KW_ALT	• Define • Undefine	Undefine	MBEDTLS_NIST_KW_ALT
Cipher > Alternate > MBEDTLS_DES_SETKEY_ALT	• Define • Undefine	Undefine	MBEDTLS_DES_SETKEY_ALT
Cipher > Alternate > MBEDTLS_DES_CRYPT_ECB_ALT	• Define • Undefine	Undefine	MBEDTLS_DES_CRYPT_ECB_ALT
Cipher > Alternate > MBEDTLS_DES3_CRYPT_ECB_ALT	• Define • Undefine	Undefine	MBEDTLS_DES3_CRYPT_ECB_ALT
Cipher > AES > MBEDTLS_AES_ROM_TABLES	• Define • Undefine	Undefine	MBEDTLS_AES_ROM_TABLES
Cipher > AES > MBEDTLS_AES_FEWER_TABLES	• Define • Undefine	Undefine	MBEDTLS_AES_FEWER_TABLES
Cipher > AES > MBEDTLS_AES_ONLY_128_BIT_KEY_LENGTH	• Define • Undefine	Undefine	MBEDTLS_AES_ONLY_128_BIT_KEY_LENGTH
Cipher > MBEDTLS_CAMELLIA_SMALL_MEMORY	• Define • Undefine	Undefine	MBEDTLS_CAMELLIA_SMALL_MEMORY
Cipher > MBEDTLS_CIPHER_MODE_CBC	• Define • Undefine	Define	MBEDTLS_CIPHER_MODE_CBC
Cipher > MBEDTLS_CIPHER_MODE_CFB	• Define • Undefine	Define	MBEDTLS_CIPHER_MODE_CFB
Cipher > MBEDTLS_CIPHER_MODE_CTR	• Define • Undefine	Define	MBEDTLS_CIPHER_MODE_CTR
Cipher > MBEDTLS_CIPHER_MODE_OFB	• Define • Undefine	Undefine	MBEDTLS_CIPHER_MODE_OFB
Cipher > MBEDTLS_CIPHER_MODE_XTS	• Define • Undefine	Undefine	MBEDTLS_CIPHER_MODE_XTS
Cipher > MBEDTLS_CIPHER_NULL_CIPHER	• Define • Undefine	Undefine	MBEDTLS_CIPHER_NULL_CIPHER

Cipher > MBEDTLS_CIPHER_PADDING_PKCS7	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_PKCS7
Cipher > MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS
Cipher > MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN
Cipher > MBEDTLS_CIPHER_PADDING_ZEROS	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_ZEROS
Cipher > MBEDTLS_AES_C	Define	Define	MBEDTLS_AES_C
Cipher > MBEDTLS_BLOCK_CIPHER_NO_DECRYPT	• Define • Undefine	Undefine	MBEDTLS_BLOCK_CIPHER_NO_DECRYPT
Cipher > MBEDTLS_CAMELLIA_C	• Define • Undefine	Undefine	MBEDTLS_CAMELLIA_C
Cipher > MBEDTLS_ARIA_C	• Define • Undefine	Undefine	MBEDTLS_ARIA_C
Cipher > MBEDTLS_CCM_C	• Define • Undefine	Define	MBEDTLS_CCM_C
Cipher > MBEDTLS_CHACHA20_C	• Define • Undefine	Undefine	MBEDTLS_CHACHA20_C
Cipher > MBEDTLS_CHACHAPOLY_C	• Define • Undefine	Undefine	MBEDTLS_CHACHAPOLY_C
Cipher > MBEDTLS_CIPHER_C	• Define • Undefine	Define	MBEDTLS_CIPHER_C
Cipher > MBEDTLS_DES_C	• Define • Undefine	Undefine	MBEDTLS_DES_C
Cipher > MBEDTLS_GCM_C	• Define • Undefine	Define	MBEDTLS_GCM_C
Cipher > MBEDTLS_GCM_LARGE_TABLE	• Define • Undefine	Undefine	MBEDTLS_GCM_LARGE_TABLE
Cipher > MBEDTLS_NIST_KW_C	• Define • Undefine	Undefine	MBEDTLS_NIST_KW_C
Public Key Cryptography (PKC) > DHM > Alternate > MBEDTLS_DHM_ALT	• Define • Undefine	Undefine	MBEDTLS_DHM_ALT
Public Key Cryptography (PKC) > DHM > MBEDTLS_DHM_C	• Define • Undefine	Undefine	MBEDTLS_DHM_C

Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECJPAKE_ALT	• Define • Undefine	Undefine	MBEDTLS_ECJPAKE_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECDSA_GENKEY_ALT	• Define • Undefine	Undefine	MBEDTLS_ECDSA_GENKEY_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_INTERNAL_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_INTERNAL_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_RANDOMIZE_JAC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_RANDOMIZE_JAC_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_ADD_MIXED_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_ADD_MIXED_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_DOUBLE_JAC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_DOUBLE_JAC_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_JAC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_NORMALIZE_JAC_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_RANDOMIZE_MXZ_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_RANDOMIZE_MXZ_ALT

Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_MXZ_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_NORMALIZE_MXZ_ALT
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP192R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP192R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP224R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP224R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP256R1_ENABLED	• Define • Undefine	Define	MBEDTLS_ECP_DP_SECP256R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP384R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP384R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP521R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP521R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP192K1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP192K1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP224K1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP224K1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP256K1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP256K1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP256R1_	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_BP256R1_ENABLED

ENABLED			
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP384R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_BP384R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP512R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_BP512R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_CURVE25519_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_CURVE25519_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_CURVE448_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_CURVE448_ENABLED
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_GEN_PUBLIC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECDH_GEN_PUBLIC_ALT
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_COMPUTE_SHARED_ALT	• Define • Undefine	Undefine	MBEDTLS_ECDH_COMPUTE_SHARED_ALT
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_NO_FALLBACK	• Define • Undefine	Undefine	MBEDTLS_ECP_NO_FALLBACK
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_NIST_OPTIM	• Define • Undefine	Undefine	MBEDTLS_ECP_NIST_OPTIM
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_RESTARTABLE	• Define • Undefine	Undefine	MBEDTLS_ECP_RESTARTABLE
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDSA_DETERMINISTIC	• Define • Undefine	Undefine	MBEDTLS_ECDSA_DETERMINISTIC
Public Key Cryptography (PKC) >	• Define • Undefine	Undefine	MBEDTLS_PK_PARSE_EC_COMPRESSED

ECC > MBEDTLS_PK_PARSE_EC_COMPRESSED			
Public Key Cryptography (PKC) > ECC > MBEDTLS_PK_PARSE_EC_EXTENDED	• Define • Undefine	Undefine	MBEDTLS_PK_PARSE_EC_EXTENDED
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_C	• Define • Undefine	Undefine	MBEDTLS_ECDH_C
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDSA_C	• Define • Undefine	Define	MBEDTLS_ECDSA_C
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_C	• Define • Undefine	Define	MBEDTLS_ECP_C
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECJPAKE_C	• Define • Undefine	Undefine	MBEDTLS_ECJPAKE_C
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_WINDOW_SIZE	• Define • Undefine	Undefine	MBEDTLS_ECP_WINDOW_SIZE
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_WINDOW_SIZE value	Manual Entry	6	MBEDTLS_ECP_WINDOW_SIZE value
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_FIXED_POINT_OPTIM	• Define • Undefine	Undefine	MBEDTLS_ECP_FIXED_POINT_OPTIM
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_FIXED_POINT_OPTIM value	Manual Entry	1	MBEDTLS_ECP_FIXED_POINT_OPTIM value
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED
Public Key Cryptography (PKC) > RSA > MBEDTLS_PK_RSA_ALT_SUPPORT	• Define • Undefine	Undefine	MBEDTLS_PK_RSA_ALT_SUPPORT

Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_NO_CRT	• Define • Undefine	Define	MBEDTLS_RSA_NO_CRT
Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_C	• Define • Undefine	Define	MBEDTLS_RSA_C
Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_GEN_KEY_MIN_BITS	• Define • Undefine	Undefine	MBEDTLS_RSA_GEN_KEY_MIN_BITS
Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_GEN_KEY_MIN_BITS value	Manual Entry	1024	MBEDTLS_RSA_GEN_KEY_MIN_BITS value
Public Key Cryptography (PKC) > MBEDTLS_GENPRIME	• Define • Undefine	Define	MBEDTLS_GENPRIME
Public Key Cryptography (PKC) > MBEDTLS_PKCS1_V15	• Define • Undefine	Define	MBEDTLS_PKCS1_V15
Public Key Cryptography (PKC) > MBEDTLS_PKCS1_V21	• Define • Undefine	Define	MBEDTLS_PKCS1_V21
Public Key Cryptography (PKC) > MBEDTLS_ASN1_PARSE_C	• Define • Undefine	Define	MBEDTLS_ASN1_PARSE_C
Public Key Cryptography (PKC) > MBEDTLS_ASN1_WRITE_C	• Define • Undefine	Define	MBEDTLS_ASN1_WRITE_C
Public Key Cryptography (PKC) > MBEDTLS_BASE64_C	• Define • Undefine	Define	MBEDTLS_BASE64_C
Public Key Cryptography (PKC) > MBEDTLS_BIGNUM_C	• Define • Undefine	Define	MBEDTLS_BIGNUM_C
Public Key Cryptography (PKC) > MBEDTLS_LMS_C	• Define • Undefine	Define	MBEDTLS_LMS_C
Public Key Cryptography (PKC) > MBEDTLS_LMS_PRIVATE	• Define • Undefine	Undefine	MBEDTLS_LMS_PRIVATE
Public Key	• Define	Define	MBEDTLS_OID_C

Cryptography (PKC) > MBEDTLS_OID_C	• Undefine		
Public Key Cryptography (PKC) > MBEDTLS_PEM_PARSE_C	• Define • Undefine	Define	MBEDTLS_PEM_PARSE_C
Public Key Cryptography (PKC) > MBEDTLS_PEM_WRITE_C	• Define • Undefine	Define	MBEDTLS_PEM_WRITE_C
Public Key Cryptography (PKC) > MBEDTLS_PK_C	• Define • Undefine	Define	MBEDTLS_PK_C
Public Key Cryptography (PKC) > MBEDTLS_PK_PARSE_C	• Define • Undefine	Define	MBEDTLS_PK_PARSE_C
Public Key Cryptography (PKC) > MBEDTLS_PK_WRITE_C	• Define • Undefine	Define	MBEDTLS_PK_WRITE_C
Public Key Cryptography (PKC) > MBEDTLS_PKCS5_C	• Define • Undefine	Define	MBEDTLS_PKCS5_C
Public Key Cryptography (PKC) > MBEDTLS_PKCS7_C	• Define • Undefine	Undefine	MBEDTLS_PKCS7_C
Public Key Cryptography (PKC) > MBEDTLS_PKCS12_C	• Define • Undefine	Define	MBEDTLS_PKCS12_C
Public Key Cryptography (PKC) > MBEDTLS_MPI_WINDOW_SIZE	• Define • Undefine	Undefine	MBEDTLS_MPI_WINDOW_SIZE
Public Key Cryptography (PKC) > MBEDTLS_MPI_WINDOW_SIZE value	Manual Entry	6	MBEDTLS_MPI_WINDOW_SIZE value
Public Key Cryptography (PKC) > MBEDTLS_MPI_MAX_SIZE	• Define • Undefine	Undefine	MBEDTLS_MPI_MAX_SIZE
Public Key Cryptography (PKC) > MBEDTLS_MPI_MAX_SIZE value	Manual Entry	1024	MBEDTLS_MPI_MAX_SIZE value
Hash > Alternate > MBEDTLS_MD5_ALT	• Define • Undefine	Undefine	MBEDTLS_MD5_ALT

Hash > Alternate > MBEDTLS_RIPEMD160_ALT	• Define • Undefine	Undefine	MBEDTLS_RIPEMD160_ALT
Hash > Alternate > MBEDTLS_SHA1_ALT	• Define • Undefine	Undefine	MBEDTLS_SHA1_ALT
Hash > Alternate > MBEDTLS_MD5_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_MD5_PROCESS_ALT
Hash > Alternate > MBEDTLS_RIPEMD160_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_RIPEMD160_PROCESS_ALT
Hash > Alternate > MBEDTLS_SHA1_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_SHA1_PROCESS_ALT
Hash > MBEDTLS_SHA256_SMALLER	• Define • Undefine	Undefine	MBEDTLS_SHA256_SMALLER
Hash > MBEDTLS_SHA512_SMALLER	• Define • Undefine	Undefine	MBEDTLS_SHA512_SMALLER
Hash > MBEDTLS_MD_C	• Define • Undefine	Define	MBEDTLS_MD_C
Hash > MBEDTLS_MD5_C	• Define • Undefine	Define	MBEDTLS_MD5_C
Hash > MBEDTLS_RIPEMD160_C	• Define • Undefine	Undefine	MBEDTLS_RIPEMD160_C
Hash > MBEDTLS_SHA1_C	• Define • Undefine	Define	MBEDTLS_SHA1_C
Hash > MBEDTLS_SHA3_C	• Define • Undefine	Undefine	MBEDTLS_SHA3_C
Hash > MBEDTLS_SHA224_C	• Define • Undefine	Define	MBEDTLS_SHA224_C
Hash > MBEDTLS_SHA256_C	• Define • Undefine	Define	MBEDTLS_SHA256_C
Hash > MBEDTLS_SHA384_C	• Define • Undefine	Undefine	MBEDTLS_SHA384_C
Hash > MBEDTLS_SHA512_C	• Define • Undefine	Undefine	MBEDTLS_SHA512_C
Message Authentication Code (MAC) > Alternate > MBEDTLS_POLY1305_ALT	• Define • Undefine	Undefine	MBEDTLS_POLY1305_ALT
Message Authentication Code (MAC) > MBEDTLS_CMAC_C	• Define • Undefine	Undefine	MBEDTLS_CMAC_C

Message Authentication Code (MAC) > MBEDTLS_HKDF_C	• Define • Undefine	Define	MBEDTLS_HKDF_C
Message Authentication Code (MAC) > MBEDTLS_HMAC_DRBG_C	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_C
Message Authentication Code (MAC) > MBEDTLS_POLY1305_C	• Define • Undefine	Undefine	MBEDTLS_POLY1305_C
Storage > MBEDTLS_FS_IO	• Define • Undefine	Undefine	MBEDTLS_FS_IO
Storage > MBEDTLS_PSA_CRYPTO_STORAGE_C	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_STORAGE_C
Storage > MBEDTLS_PSA_ITS_FILE_C	• Define • Undefine	Undefine	MBEDTLS_PSA_ITS_FILE_C
Storage > MBEDTLS_PSA_STATIC_KEY_SLOTS	• Define • Undefine	Undefine	MBEDTLS_PSA_STATIC_KEY_SLOTS
RNG > MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES	• Define • Undefine	Undefine	MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES
RNG > MBEDTLS_ENTROPY_FORCE_SHA256	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_FORCE_SHA256
RNG > MBEDTLS_ENTROPY_NV_SEED	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_NV_SEED
RNG > MBEDTLS_PSA_CRYPTO_EXTERNAL_RNG	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_EXTERNAL_RNG
RNG > MBEDTLS_PSA_INJECT_ENTROPY	• Define • Undefine	Undefine	MBEDTLS_PSA_INJECT_ENTROPY
RNG > MBEDTLS_CTR_DRBG_C	• Define • Undefine	Define	MBEDTLS_CTR_DRBG_C
RNG > MBEDTLS_CTR_DRBG_C_ALT	Define	Define	MBEDTLS_CTR_DRBG_C_ALT
RNG > MBEDTLS_CTR_DRBG_ENTROPY_LEN	• Define • Undefine	Undefine	RNG|MBEDTLS_CTR_DRBG_ENTROPY_LEN
RNG > MBEDTLS_CTR_DRBG_ENTROPY_LEN value	Manual Entry	48	RNG value|MBEDTLS_CTR_DRBG_ENTROPY_LEN
RNG > MBEDTLS_CTR_DRBG_RESEED_INTERVAL	• Define • Undefine	Undefine	RNG|MBEDTLS_CTR_DRBG_RESEED_INTERVAL

RNG > MBEDTLS_CTR_DRBG_RESEED_INTERVAL value	Manual Entry	10000	RNG value|MBEDTLS_CTR_DRBG_RESEED_INTERVAL
RNG > MBEDTLS_CTR_DRBG_MAX_INPUT	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_MAX_INPUT
RNG > MBEDTLS_CTR_DRBG_MAX_INPUT value	Manual Entry	256	MBEDTLS_CTR_DRBG_MAX_INPUT value
RNG > MBEDTLS_CTR_DRBG_MAX_REQUEST	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_MAX_REQUEST
RNG > MBEDTLS_CTR_DRBG_MAX_REQUEST value	Manual Entry	1024	MBEDTLS_CTR_DRBG_MAX_REQUEST value
RNG > MBEDTLS_CTR_DRBG_MAX_SEED_INPUT	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_MAX_SEED_INPUT
RNG > MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value	Manual Entry	384	MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value
RNG > MBEDTLS_CTR_DRBG_USE_128_BIT_KEY	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_USE_128_BIT_KEY
RNG > MBEDTLS_HMAC_DRBG_RESEED_INTERVAL	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_RESEED_INTERVAL
RNG > MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value	Manual Entry	10000	MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value
RNG > MBEDTLS_HMAC_DRBG_MAX_INPUT	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_MAX_INPUT
RNG > MBEDTLS_HMAC_DRBG_MAX_INPUT value	Manual Entry	256	MBEDTLS_HMAC_DRBG_MAX_INPUT value
RNG > MBEDTLS_HMAC_DRBG_MAX_REQUEST	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_MAX_REQUEST
RNG > MBEDTLS_HMAC_DRBG_MAX_REQUEST value	Manual Entry	1024	MBEDTLS_HMAC_DRBG_MAX_REQUEST value
RNG > MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT
RNG > MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value	Manual Entry	384	MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value
RNG > MBEDTLS_ENTR	• Define	Undefine	MBEDTLS_ENTROPY_M

OPY_MAX_SOURCES	• Undefine		AX_SOURCES
RNG > MBEDTLS_ENTROPY_MAX_SOURCES value	Manual Entry	20	MBEDTLS_ENTROPY_MAX_SOURCES value
RNG > MBEDTLS_ENTROPY_MAX_GATHER	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_MAX_GATHER
RNG > MBEDTLS_ENTROPY_MAX_GATHER value	Manual Entry	128	MBEDTLS_ENTROPY_MAX_GATHER value
RNG > MBEDTLS_ENTROPY_MIN_HARDWARE	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_MIN_HARDWARE
RNG > MBEDTLS_ENTROPY_MIN_HARDWARE value	Manual Entry	32	MBEDTLS_ENTROPY_MIN_HARDWARE value
Key Configuration > MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER
Key Configuration > MBEDTLS_PSA_CRYPTO_BUILTIN_KEYS	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_BUILTIN_KEYS
Key Configuration > PSA_CRYPTO_DRIVER_TFM_BUILTIN_KEY_LOADER	• Define • Undefine	Undefine	PSA_CRYPTO_DRIVER_TFM_BUILTIN_KEY_LOADER
Key Configuration > MBEDTLS_PSA_KEY_SLOT_COUNT	• Define • Undefine	Undefine	MBEDTLS_PSA_KEY_SLOT_COUNT
Key Configuration > MBEDTLS_PSA_KEY_SLOT_COUNT value	Manual Entry	32	MBEDTLS_PSA_KEY_SLOT_COUNT value
Key Configuration > MBEDTLS_PSA_STATIC_KEY_SLOT_BUFFER_SIZE value	Manual Entry	256	MBEDTLS_PSA_STATIC_KEY_SLOT_BUFFER_SIZE value
Post Quantum Cryptography (PQC) > MBEDTLS_MLKEM_C	• Define • Undefine	Undefine	MBEDTLS_MLKEM_C
Post Quantum Cryptography (PQC) > MBEDTLS_ML_DSA_C	• Define • Undefine	Undefine	MBEDTLS_ML_DSA_C
HMAC > MBEDTLS_PSA_HMAC_DRBG_MD_TYPE	• Define • Undefine	Define	MBEDTLS_PSA_HMAC_DRBG_MD_TYPE

SHA256 Configuration

To enable hardware acceleration for the SHA256/224 calculation, the macro MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT must be defined in the configuration file. By default SHA256 is enabled. SHA256 can be disabled, but SHA512 then needs to be enabled (software version) because the PSA implementation uses it for the entropy accumulator. This can be done using the FSP Configuration editor.

AES Configuration

To enable hardware acceleration for the AES128/256 operation, the macro MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT must be defined in the configuration file. By default AES is enabled. AES cannot be disabled because the PSA implementation requires it for the CTR_DRBG random number generator. This can be done using the FSP Configuration editor.

Note

Only AES XTS 128 is currently supported. RA2 devices support acceleration for ECB part alone, while other devices support full AES XTS hardware acceleration.

ECC Configuration

To enable hardware acceleration for the ECC Key Generation operation, the macro MBEDTLS_ECP_ALT must be defined in the configuration file. For ECDSA, the macros MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT must be defined. By default ECC, ECDSA and ECDHE are enabled. To disable ECC, undefine MBEDTLS_ECP_C, MBEDTLS_ECDSA_C and MBEDTLS_ECDH_C. This can be done using the FSP Configuration editor.

RSA Configuration

To enable hardware acceleration for the RSA2048 operation, the macro MBEDTLS_RSA_ALT must be defined in the configuration file. By default RSA is enabled. To disable RSA, undefine MBEDTLS_RSA_C, MBEDTLS_PK_C, MBEDTLS_PK_PARSE_C, MBEDTLS_PK_WRITE_C. This can be done using the FSP Configuration editor.

Post Quantum Cryptography (PQC) Configuration

ML-KEM Configuration

To enable software only PQC ML-KEM key generation, encapsulation, and decapsulation operations, the macro MBEDTLS_MLKEM_C must be defined in the configuration file. By default, PQC ML-KEM is disabled as the PQC stack is optional. This can be done using the FSP Configuration editor.

ML-DSA Configuration

To enable software only PQC ML-DSA key generation, signing, and verification operations, the macro MBEDTLS_ML_DSA_C must be defined in the configuration file. By default, PQC ML-DSA is disabled as the PQC stack is optional. This can be done using the FSP Configuration editor.

Wrapped Key Usage

To use the Secure Crypto Engine to generate and use wrapped keys, use PSA_KEY_TYPE_AES_WRAPPED or PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(curve) or PSA_KEY_TYPE_RSA_KEY_PAIR when setting the key type attribute. Setting the key's type attribute using this value will cause the SCE to use wrapped key mode for all operations related to that key. The user can use the export functionality to save the wrapped keys to user ROM and import it later for usage. This mode requires that Wrapped Key functionality for the algorithm is enabled in the

project configuration.

Note

On the SCE9 devices, only the RSA public key can be exported. A file system must be used to store the internally generated private key.

Persistent Key Storage

Persistent key storage can be enabled by defining MBEDTLS_FS_IO, MBEDTLS_PSA_CRYPTO_STORAGE_C, and MBEDTLS_PSA_ITS_FILE_C. The key lifetime must also be specified as PSA_KEY_LIFETIME_PERSISTENT. A lower level storage module must be added in the FSP Configuration editor and initialized in the code before generating persistent keys. Persistent storage supports the use of plaintext and vendor keys. Refer to the lower level storage module documentation for information on how it should be initialized. To generate a persistent key the key must be assigned a unique id prior to calling generate using the psa_set_key_id api.

```
if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
   {
/* Set the id to a positive integer. */
      psa_set_key_id(&attributes, (psa_key_id_t) 5);
   }
```

Platform Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT must be defined in the configuration file. This enables code that will initialize the SCE. Parameter checking (General|MBEDTLS_CHECK_PARAMS) is enabled by default. To reduce code size, disable parameter checking.

Random Number Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_ENTROPY_HARDWARE_ALT must be defined in the configuration file. This enables using the TRNG as an entropy source. None of the other cryptographic operations (even in software only mode) will work without this feature.

Usage Notes

Hardware Initialization

mbedtls_platform_setup() must be invoked before using the PSA Crypto API to ensure that the SCE peripheral is initialized.

Memory Usage

In general, depending on the mbedCrypto features being used a heap size of 0x1000 to 0x5000 bytes is required. The total allocated heap should be the **sum** of the heap requirements of the individual algorithms.

A minimum stack of 0x1000 is required where the module is used. This is either the main stack in a

bare metal application or the task stack of the task used for crypto operations. The total allocated stack should be the **sum** of the stack requirements of the individual algorithms.

Algorithm	Required Heap (bytes)	Required Stack (bytes)
SHA256/224	None	–
AES	0x200	–
Hardware ECC	0x400	–
Software ECC	0x1800	–
RSA	0x1500	–
ML-DSA	0x1800	0x1500
ML-KEM	0x1000	0x3000

Limitations

- Only little endian mode is supported.

Stdio Buffering

The MBEDTLS_PLATFORM_SETBUF_MACRO was introduced in mbedTLS 3.2.1 to prevent stdio read/write functions from buffering stream data to reduce the likelihood of key leakage by setting the buffer argument in setbuf() to NULL. FSP uses a dummy_setbuf() function in rm_psa_crypto.c to prevent build errors; since FSP uses LittleFS by default (where the usage of a buffer is mandatory) this function does not perform any action. Setting the cache size in LittleFS to the minimum supported by the Data Flash (4) can minimize but not remove the likelihood of key data leakage. The dummy function can be replaced with a user-defined function by defining a different value for MBEDTLS_PLATFORM_SETBUF_MACRO_value in the FSP configurator.

Post Quantum Cryptography (PQC)

- The PSA API Specificiation v1.3 supports ML-KEM and ML-DSA however MbedTLS currently does not provide an implementation. The implementation in the FSP release is thus likely to change and may cause backward compatibility issues when MbedTLS officially supports it.
- ASN.1 encoding not implemented. Keys are currently implemented in raw bytes.
- The underlying PQC-Lib library is a release candidate of V2.00.

ML-DSA

- The FIPS204 Standard allows the user to provide an optional context string for ML-DSA signature generation and verification. The current implementation does not allow the user to provide such a string. Internally this field is set to a value of 0 with a length of 0.
- The data format for import and export of the key pair is a byte array of the 32 byte seed value.
- ML-DSA-44, ML-DSA-65, and ML-DSA-87 supported.
- The current implementation of ML-DSA is too big for devices with less than 64KB of RAM.

ML-KEM

- ML-KEM-512, ML-KEM-768, and ML-KEM-1024 supported.
- Key derivation not supported.
- The data format for import and export of the key pair is the concatenation of the two 32

byte seed values: d || z

SCE9 Usage

The SCE9 is used in Compatibility Mode for mbedCrypto acceleration. The crypto capabilities in this mode on the SCE9 are different which results in the below usage limitations with mbedCrypto:

- The module includes **both** wrapped and plaintext keys code irrespective of whether the application requires it.
- Plaintext key generation is not supported for RSA and ECC; only wrapped keys can be generated.
- If ECDH is used, only wrapped key will be generated on SCE9 and will not return an error even if the user context is somehow set for plain key. This may be relevant only if the psa_key_agreement() function with plaintext key on SCE9 is attempted.

Note

For a detailed description of the different SCE9 operating modes, refer to Application Note R11AN0498.

Using PSA Crypto with TrustZone

Unlike FSP drivers, PSA Crypto cannot be configured as Non-secure callable in the FSP Configurator for a secure project. The reason for this is that in order to achieve the security objective of controlling access to protected keys, both the PSA Crypto code as well as the keys must be placed in the secure region. Since the PSA Crypto API requires access to the keys directly during initialization and later via a key handle, allowing non-secure code to use the API by making it Non-secure callable will require the keys to be stored in non-secure memory.

This section will provide a short explanation of how to add PSA Crypto to a secure project and have it usable by the non-secure project without exposing the keys. In this example the secure project will contain an RSA private key and the non-secure project is expected to be able to perform sign and verify operations using that key.

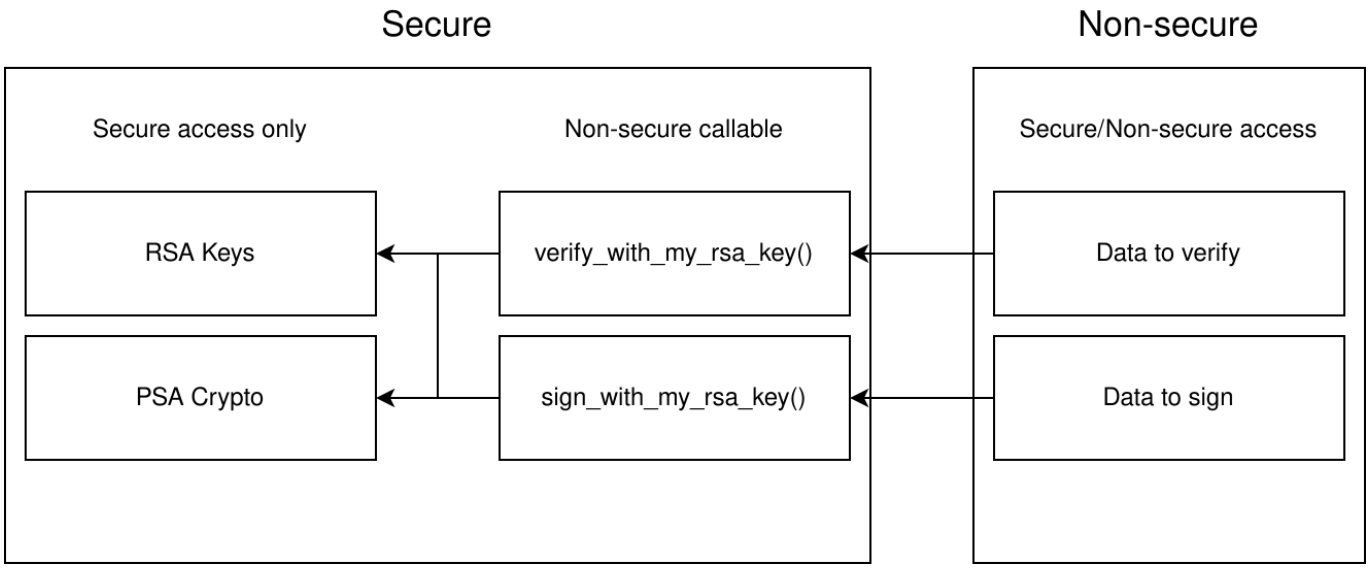


Figure 151: PSA Crypto Non-secure callable example

- Secure project
  - During secure project boot-up, psa_crypto_init() is called.

- The RSA private key is programmed into secure flash either at the factory or by calling psa_generate_key() in persistent mode. Note that the data-flash area used by the LittleFS will have to be in the secure region if the key is generated as a persistent.
- psa_import_key()/psa_open_key() are called with the resultant handle held in secure RAM.
- The Non-secure callable section contains the following **user-defined** functions
- verify_with_my_rsa_key(input_signature, input_hash, verification_result)
- The implementation of this function in secure region will call psa_verify_hash() and return the result via verification_result.
- sign_with_my_rsa_key(input_hash, output_signature)
- The implementation of this function in secure region will call psa_sign_hash() and return the signature via output_signature.

- Non-secure project
  - Calls verify_with_my_rsa_key() to verify a signature. The implementation will use the public key that is present in the secure project.
  - Calls sign_with_my_rsa_key() to generate a signature. The implementation will use the private key that is present on the secure project.

For more details on how to add user-code to the Non-secure callable region refer to the "Security Design with Arm TrustZone - IP Protection (R11AN0467EU0100)" Application Note.

Examples

Hash Example

This is an example on calculating the SHA256 hash using the PSA Crypto API.

```
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void psa_crypto_sha256_example (void)
```

```
{
    psa_algorithm_t      alg                = PSA_ALG_SHA_256;
    psa_hash_operation_t operation          = {0};
 size_t                expected_hash_len = PSA_HASH_LENGTH(alg);
    uint8_t              actual_hash[PSA_HASH_MAX_SIZE];
 size_t                actual_hash_len;
 mbedtls_platform_context ctx = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_setup(&operation, alg))
    {
 /* Hash setup failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_update(&operation, NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200)))
    {
 /* Hash calculation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_finish(&operation, &actual_hash[0], sizeof
(actual_hash), &actual_hash_len))
    {
 /* Reading calculated hash failed */
       debugger_break();
    }
 else if (0 != memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len))
    {
 /* Hash compare of calculated value with expected value failed */
```

```
        debugger_break();
    }
 else if (0 != memcmp(&expected_hash_len, &actual_hash_len, sizeof
(expected_hash_len)))
    {
 /* Hash size compare of calculated value with expected value failed */
        debugger_break();
    }
 else
    {
 /* SHA256 calculation succeeded */
        debugger_break();
    }
 /* De-initialize the platform. This is currently a placeholder function which does
not do anything. */
 mbedtls_platform_teardown(&ctx);
}
```

AES Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
static psa_status_t cipher_operation (psa_cipher_operation_t * operation,
 const uint8_t          * input,
 size_t                   input_size,
 size_t                   part_size,
                                    uint8_t                * output,
 size_t                   output_size,
 size_t                * output_len)
{
    psa_status_t status;
 size_t       bytes_to_write = 0;
 size_t       bytes_written = 0;
 size_t       len            = 0;
```

```
    *output_len = 0;
 while (bytes_written != input_size)
    {
       bytes_to_write = (input_size - bytes_written > part_size ?
                         part_size :
                         input_size - bytes_written);
       status = psa_cipher_update(operation,
                                  input + bytes_written,
                                  bytes_to_write,
                                  output + *output_len,
                                  output_size - *output_len,
                                  &len);
 if (PSA_SUCCESS != status)
      {
 return status;
      }
       bytes_written += bytes_to_write;
       *output_len   += len;
    }
    status = psa_cipher_finish(operation, output + *output_len, output_size -
*output_len, &len);
 if (PSA_SUCCESS != status)
    {
 return status;
    }
    *output_len += len;
 return status;
}
void psa_crypto_aes256cbcmultipart_example (void)
{
 enum
    {
       block_size = PSA_BLOCK_CIPHER_BLOCK_LENGTH(PSA_KEY_TYPE_AES),
       key_bits   = 256,
```

```
      input_size = 100,
      part_size = 10,
   };
mbedtls_platform_context ctx         = {0};
const psa_algorithm_t    alg         = PSA_ALG_CBC_PKCS7;
   psa_cipher_operation_t   operation_1 = PSA_CIPHER_OPERATION_INIT;
   psa_cipher_operation_t   operation_2 = PSA_CIPHER_OPERATION_INIT;
size_t iv_len = 0;
   psa_key_handle_t    key_handle       = 0;
size_t               encrypted_length = 0;
size_t               decrypted_length = 0;
   uint8_t             iv[block_size]    = {0};
   uint8_t             input[input_size] = {0};
   uint8_t             encrypted_data[input_size + block_size] = {0};
   uint8_t             decrypted_data[input_size + block_size] = {0};
   psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
   psa_key_lifetime_t   lifetime;
/* Setup the platform; initialize the SCE */
if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
   {
/* Platform initialization failed */
      debugger_break();
   }
if (PSA_SUCCESS != psa_crypto_init())
   {
/* PSA Crypto Initialization failed */
      debugger_break();
   }
/* Set key attributes */
   psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
   psa_set_key_algorithm(&attributes, alg);
/* To use wrapped keys instead of plaintext use PSA_KEY_TYPE_AES_WRAPPED. */
   psa_set_key_type(&attributes, PSA_KEY_TYPE_AES);
```

```
    psa_set_key_bits(&attributes, key_bits);
    lifetime = PSA_KEY_LIFETIME_VOLATILE;
 /* To use persistent keys:
  *......Use a lifetime value of PSA_KEY_LIFETIME_PERSISTENT
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, lifetime);
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
    {
 /* Set the id to a positive integer. */
       psa_set_key_id(&attributes, (psa_key_id_t) 5);
    }
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
```

```
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, input_size, part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
```

```
        debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
        debugger_break();
    }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Destroying the key handle failed */
        debugger_break();
    }
 else
    {
 /* All the operations succeeded */
    }
 /* Close the SCE */
 mbedtls_platform_teardown(&ctx);
}
void psa_crypto_aes128xtsmultipart_example (void)
{
 enum
    {
        block_size = PSA_BLOCK_CIPHER_BLOCK_LENGTH(PSA_KEY_TYPE_AES),
        key_bits   = 256,
        input_size = 32,
        part_size = 16,
    };
 mbedtls_platform_context ctx         = {0};
 const psa_algorithm_t    alg         = PSA_ALG_XTS;
    psa_cipher_operation_t   operation_1 = PSA_CIPHER_OPERATION_INIT;
    psa_cipher_operation_t   operation_2 = PSA_CIPHER_OPERATION_INIT;
 size_t iv_len = 0;
    psa_key_handle_t    key_handle                = 0;
```

```
 size_t               encrypted_length          = 0;
 size_t               decrypted_length          = 0;
    uint8_t             iv[block_size]             = {0};
    uint8_t             input[input_size]          = {0};
    uint8_t             encrypted_data[input_size] = {0};
    uint8_t             decrypted_data[input_size] = {0};
    psa_key_attributes_t attributes                = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_lifetime_t   lifetime;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
    psa_set_key_algorithm(&attributes, alg);
 /* To use wrapped keys instead of plaintext use PSA_KEY_TYPE_AES_WRAPPED. */
 /* When using XTS mode, since there are two keys used, the key_bits value is twice
the AES key length.
  * For AES 128 XTS mode, the key_bits value is 256. MbedTLS only supports AES 128
currently. */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_AES);
    psa_set_key_bits(&attributes, key_bits);
    lifetime = PSA_KEY_LIFETIME_VOLATILE;
 /* To use persistent keys:
  *......Use a lifetime value of PSA_KEY_LIFETIME_PERSISTENT
  * - The file system must be initialized prior to calling the generate/import key
```

```
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, lifetime);
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
    {
 /* Set the id to a positive integer. */
       psa_set_key_id(&attributes, (psa_key_id_t) 5);
    }
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, sizeof(input), part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
```

```
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
       debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
```

```
      debugger_break();
   }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
   {
 /* Destroying the key handle failed */
      debugger_break();
   }
 else
   {
 /* All the operations succeeded */
   }
 /* Close the SCE */
 mbedtls_platform_teardown(&ctx);
}
```

AES-CCM Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding using AES-CCM.

```
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
   {
 /* Random plaintext input generation failed */
      debugger_break();
   }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
   {
 /* Key generation failed */
      debugger_break();
   }
 /* AES-CCM Encryption */
 else if (PSA_SUCCESS !=
            psa_aead_encrypt(key_handle, PSA_ALG_CCM, nonce, sizeof(nonce),
additional_data, sizeof(additional_data),
                             input, sizeof(input), encrypt, sizeof(encrypt),
```

```
&output_len))
    {
 /* AES-CCM Encryption failed */
       debugger_break();
    }
 /* AES-CCM Decryption */
 else if (PSA_SUCCESS !=
             psa_aead_decrypt(key_handle, PSA_ALG_CCM, nonce, sizeof(nonce),
additional_data, sizeof(additional_data),
                              encrypt, output_len, decrypt, sizeof(decrypt),
&output_len))
    {
 /* AES-CCM Decryption failed */
       debugger_break();
    }
 else if (0U != memcmp(input, decrypt, sizeof(input)))
    {
 /* The decrypted result did not match the plaintext input */
       debugger_break();
    }
 else
    {
 /* All operations were successful */
    }
```

AES-XTS Example

This is an example on using the PSA Crypto API to generate an AES128 XTS key, encrypting and decrypting multi-block data.

```
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
```

```
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, sizeof(input), part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
```

```
else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
   {
/* Setting the IV failed */
      debugger_break();
   }
else if (PSA_SUCCESS !=
            cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
sizeof(decrypted_data), &decrypted_length))
   {
/* Decryption failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
   {
/* Terminating the decryption operation failed */
      debugger_break();
   }
else if (0 != memcmp(input, decrypted_data, sizeof(input)))
   {
/* Comparing the input data with decrypted data failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_destroy_key(key_handle))
   {
/* Destroying the key handle failed */
      debugger_break();
   }
else
   {
/* All the operations succeeded */
   }
```

CMAC Example

This is an example on using the PSA Crypto API to generate an AES256 key, followed by generation and verification of MAC for random data of known length.

```
if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
   {
/* Random number generation failure */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
   {
/* Key generation failure */
      debugger_break();
   }
/* Steps to generate the MAC */
else if (PSA_SUCCESS != psa_mac_sign_setup(&operation, key_handle, alg))
   {
/* MAC Sign setup failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_mac_update(&operation, input, input_size))
   {
/* MAC update failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_mac_sign_finish(&operation, AES_CMAC_mac, sizeof
(AES_CMAC_mac), &mac_ret))
   {
/* MAC Sign operation failed */
      debugger_break();
   }
else
   {
/* All the operations succeeded for MAC generation */
   }
/* Steps to verify the generated MAC */
```

```
 if (PSA_SUCCESS != psa_mac_verify_setup(&verify_operation, key_handle, alg))
    {
 /* MAC verification setup failure */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_mac_update(&verify_operation, input, input_size))
    {
 /* MAC update failure */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_mac_verify_finish(&verify_operation, AES_CMAC_mac,
mac_ret))
    {
 /* MAC verification failed */
       debugger_break();
    }
 else
    {
 /* All the operations succeeded for MAC verification */
    }
```

ECC Example

This is an example on using the PSA Crypto API to generate an ECC-P256R1 key, signing and verifying data after hashing it first using SHA256.

Note

Unlike RSA, ECDSA does not have any padding schemes. Thus the hash argument for the ECC sign operation MUST have a size larger than or equal to the curve size; i.e. for PSA_ECC_CURVE_SECP256R1 the payload size must be at least 256/8 bytes. nist.fips.186-4: " A hash function that provides a lower security strength than the security strength associated with the bit length of 'n' ordinarily should not be used, since this would reduce the security strength of the digital signature process to a level no greater than that provided by the hash function."

```
#define ECC_256_BIT_LENGTH 256
#define ECC_256_EXPORTED_SIZE 500
uint8_t exportedECC_SECP256R1Key[ECC_256_EXPORTED_SIZE];
size_t exportedECC_SECP256R1Keylength = 0;
void psa_ecc256R1_example (void)
```

```
{
/* This example generates an ECC-P256R1 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t                   signature_length1 = 0;
 size_t                   signature_length2 = 0;
    psa_key_attributes_t    attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t    read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 mbedtls_platform_context ctx               = {0};
    psa_key_handle_t         ecc_key_handle    = {0};
    psa_hash_operation_t hash_operation = {0};
    uint8_t             payload_hash[PSA_HASH_MAX_SIZE];
 size_t               payload_hash_len;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_ECDSA(PSA_ALG_SHA_256));
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(PSA_ECC_FAMILY_SECP_R1).*/
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ECC_KEY_PAIR(PSA_ECC_FAMILY_SECP_R1));
    psa_set_key_bits(&attributes, ECC_256_BIT_LENGTH);
```

```
 /* To use persistent keys instead of volatile:
  * - Use PSA_KEY_LIFETIME_PERSISTENT.
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate ECC P256R1 Key pair */
 if (PSA_SUCCESS != psa_generate_key(&attributes, &ecc_key_handle))
    {
       debugger_break();
    }
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(ecc_key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Calculate the hash of the message */
 if (PSA_SUCCESS != psa_hash_setup(&hash_operation, PSA_ALG_SHA_256))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_hash_update(&hash_operation, payload, sizeof(payload)))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_hash_finish(&hash_operation, &payload_hash[0], sizeof(payload_hash),
&payload_hash_len))
    {
       debugger_break();
    }
 /* Sign message using the private key
  * NOTE: The hash argument (payload_hash here) MUST have a size equal to the curve
size;
```

```
  * i.e. for SECP256R1 the payload size must be 256/8 bytes.
  * Similarly for SECP384R1 the payload size must be 384/8 bytes.
  * nist.fips.186-4: " A hash function that provides a lower security strength than
  * the security strength associated with the bit length of 'n' ordinarily should not
be used, since this
  * would reduce the security strength of the digital signature process to a level no
greater than that
  * provided by the hash function." */
 if (PSA_SUCCESS !=
       psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
                        signature_length1))
    {
       debugger_break();
    }
 /* Export the key. The exported key can then be save to flash for later usage. */
 if (PSA_SUCCESS !=
       psa_export_key(ecc_key_handle, exportedECC_SECP256R1Key, sizeof
(exportedECC_SECP256R1Key),
                      &exportedECC_SECP256R1Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(ecc_key_handle))
    {
```

```
        debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS !=
        psa_import_key(&attributes, exportedECC_SECP256R1Key,
exportedECC_SECP256R1Keylength, &ecc_key_handle))
    {
        debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
        psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
 sizeof(signature2), &signature_length2))
    {
        debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
        psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
                        signature_length2))
    {
        debugger_break();
    }
 /* Signatures cannot be compared since ECC signatures vary for the same data unless
Deterministic ECC is used which is not supported by the HW.
  * Only the verification operation can be used to validate signatures. */
}
```

RSA Example

This is an example on using the PSA Crypto API to generate an RSA2048 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
#define RSA_2048_BIT_LENGTH 2048
#define RSA_2048_EXPORTED_SIZE 1210
/* The RSA 2048 key pair export in der format is roughly as follows
 * RSA private keys:
 * RSAPrivateKey ::= SEQUENCE { ------------------------------------- 1 + 3
 * version Version, ------------------------------------- 1 + 1 + 1
 * modulus INTEGER, ---------------- n ------------------- 1 + 3 + 256 + 1
 * publicExponent INTEGER, ---------------- e ------------------- 1 + 4
 * privateExponent INTEGER, ---------------- d ------------------- 1 + 3 + 256 (276
for Wrapped)
 * prime1 INTEGER, ---------------- p ------------------- 1 + 3 + (256 / 2)
 * prime2 INTEGER, ---------------- q ------------------- 1 + 3 + (256 / 2)
 * exponent1 INTEGER, ---------------- d mod (p-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * exponent2 INTEGER, ---------------- d mod (q-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * coefficient INTEGER, ---------------- (inverse of q) mod p - 1 + 2 + (256 / 2) (4
for Wrapped)
 * otherPrimeInfos OtherPrimeInfos OPTIONAL ------------------------ 0 (not
supported)
 * }
 */
uint8_t exportedRSA2048Key[RSA_2048_EXPORTED_SIZE];
size_t exportedRSA2048Keylength = 0;
void psa_rsa2048_example (void)
{
/* This example generates an RSA2048 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
```

```
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length1 = 0;
 size_t               signature_length2 = 0;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_TYPE_RSA_KEY_PAIR_WRAPPED. */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* To use persistent keys instead of volatile:
  * - Use PSA_KEY_LIFETIME_PERSISTENT.
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate RSA 2048 Key pair */
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
       debugger_break();
    }
 /* Test the key information */
```

```
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature1,
                       signature_length1))
    {
       debugger_break();
    }
 /* Export the key */
 if (PSA_SUCCESS !=
       psa_export_key(key_handle, exportedRSA2048Key, sizeof(exportedRSA2048Key),
&exportedRSA2048Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS != psa_import_key(&attributes, exportedRSA2048Key,
```

```
exportedRSA2048Keylength, &key_handle))
    {
        debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
        psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature2,
 sizeof(signature2), &signature_length2))
    {
        debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
        psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature2,
                        signature_length2))
    {
        debugger_break();
    }
 /* Compare signatures to verify that the same signature was generated */
 if (0 != memcmp(signature2, signature1, signature_length2))
    {
        debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
}
/* Generated using "openssl genrsa -out example_private_key.pem 2048" */
const uint8_t RSAKeydata[] = "-----BEGIN RSA PRIVATE KEY-----\n"
 "MIIEowIBAAKCAQEAplaPU68h4hW/eAlH2hbQl1WjoDT1znCk0O9cnE9lAnB26IUi"
 "78wUAxuTZSlrlKez9FGGIVbUMTy7dTw0ZBanogrKCaAMaRFz5LRv4I8RQzsDxxqe"
 "nud9Y6mVRWb9hyutL/yBwDd6jzAwcviAhcatTv0E5wpEJ6ezMDyaiSCbn84hfb0T"
 "ZPXQVGhmjii3f0FfFWW9ce70qkuwdkvO6HBrOGVi2++b11As+Uh/7pxyOb7wpE7K"
```

```
 "X64nohaCtXgNnV8Hr+LOPQRHEbGTCPOfx1davKEIYFvm8hVxqv20csMD5uc8AJCd"
 "0cdom13KcOpVXCnrFPNJt0lITOcH/fBzp4xoaQIDAQABAoIBAGc56qKVWeKzempk"
 "4AlRBUwVYoEGvNDLiQz8rq12fAoCf1iXXvIP5Q90qokqJlBPrtbdTO8dsnuH5BHC"
 "NgUzJf7i0TUd9PWzVgfFjnR+dMkTM6n5NB0LLf6OfEtguc+L5GOWQXNnOpDn5/lB"
 "jIj4ng8Z6FP1RAyT/xjdU03sRYfq5OPlDZ8+ck7WwQ8VxSBKsJcZVv+K4r64Vmr1"
 "MO1zfBYuwafi4oO9Z9x89Iil9GBcYuCvhLvDoWhnTUxlOTfz+Jt+qwlmVimEz6xs"
 "6ew1Ox7e6SInly3IAXe9E84yP9MDmiGkCrVr8s8GUybZ5t/yeZ3g5lPYSQ/FiuHY"
 "kvJCH4ECgYEA2IthRDkElOQGqJ9T/LjgcIVLQs0F6i7zZWg6UKv18EmG7uHd6tsZ"
 "byFfFiBat80rq6q4AUx2OOE7+LI5MxIvgRgLk14WV8H2OKK/lfxOFb6a4feGM7bH"
 "VdERQSMNeGq0bP9IOPxU1EvFp0Vfv7cfGsQhcAPgA8up61jWREhI+fECgYEAxKVT"
 "9pKinIwgjsHgmxalaNbnRf+bcqtdS98SB2MuICg8ubcBSrjm4wtGe3oCX/PbM1GV"
 "FvEDKl4TyeWFIH4MStsyDHYxwoV2C01bBHEtHTYBnjeIShMruIomsG9GeTrNmyzg"
 "dVdSg5/lxYOxuZSXvfcbyIMsW29ddgeICu9RHfkCgYBaljQibhfUkW+Xqs9fsZdy"
 "etB5KXuH9AwuJ+P9S3KfCqM/240SaoXBT5yPjQlmSpYyQkCnim0Kbm7AIw56pujo"
 "gD6Xb4y5OZLfLnYnMF0aC5qPXRTvHU9WPxeQwDEqZrkDv+der3BtPyV4TDU55klE"
 "0TeLvJNCAzkaExfPiM2+gQKBgQC5ceBYq7hGQa+CcTvLhfO/hsrbrE2AOjLllhx+"
 "cv3QvaFm0jqZqP20J7H0R/9tVZ7mKo2a8Pa3QbkPsS92kOguv7/XGK+cbhgAWJb4"
 "/XI6FfA4sM4KbUHR6hbKVGX1dYroR831WsAp+OTK+4LjLEpdj2fYFDwEjmVpJXka"
 "Ns4coQKBgGESrtpJF7OQG4xcXwR2ZiJESPvMKVmtvxRzDrc9gmoiIIIKx1fimwCv"
 "RtOz1bQcXMRw21+ZAZpen3ahWh63KC0KxMSNBJTXdczlf2uprVkSPtmAjV4qBgwv"
 "tqjmP0lnGc9wkJsVvGmMAAfQOWgxs9h/VH/b+6biEkzaaZLOyABV"
 "-----END RSA PRIVATE KEY-----\n";
/* Expected signature below is calculated using the mbedCrypto software version */
const uint8_t RSAExpectedSignature[256] =
{
    0x25, 0xC0, 0x15, 0x64, 0xD3, 0xF7, 0xC1, 0xB7, 0xA8, 0x9B, 0x56, 0x1C, 0xB5,
0xA4, 0xA5, 0x0D, 0x61, 0x52, 0x32,
    0x0C, 0x4E, 0xE8, 0xCA, 0x4B, 0x9E, 0xA2, 0x4D, 0x35, 0x0E, 0xB1, 0xA1, 0x5B, //
NOLINT(readability-magic-numbers)
    0x5B, 0xD8, 0xC1, 0x93, 0x28, 0x60, 0x90, 0x18, 0x37, 0x88, 0x66, 0xD4, 0xED,
0x37, 0x6B, 0xEC, 0x48, 0xFF, 0x3D,
    0xFB, 0x99, 0xB7, 0xEF, 0x3C, 0x8D, 0x25, 0xE9, 0xF3, 0xCF, 0x02, 0x7C, 0x7D, //
NOLINT(readability-magic-numbers)
    0xB9, 0x4D, 0x2F, 0x2F, 0x1E, 0x30, 0x48, 0x54, 0x18, 0xE4, 0x51, 0xD5, 0xC1,
```

```
0xB7, 0x3B, 0x0D, 0xE4, 0xB4, 0x19,
    0x7B, 0x9B, 0xF4, 0x35, 0x82, 0xCC, 0x91, 0x4C, 0x10, 0xE9, 0xB6, 0xF1, 0x0A, //
NOLINT(readability-magic-numbers)
    0x23, 0xEB, 0x7D, 0x51, 0x47, 0x36, 0xFE, 0x13, 0xAF, 0x3C, 0x23, 0x9F, 0x7E,
0xFC, 0xCF, 0x7A, 0x7C, 0x2D, 0xDB,
    0xD9, 0xDA, 0xEB, 0xF7, 0xB5, 0x6B, 0x8D, 0xE0, 0x18, 0x9A, 0x5B, 0xB7, 0x0A, //
NOLINT(readability-magic-numbers)
    0xA3, 0x4E, 0xE1, 0xB7, 0xF7, 0xD1, 0x94, 0xD5, 0x7A, 0xD3, 0x27, 0xE2, 0x1F,
0x3A, 0xEB, 0xF0, 0x83, 0x10, 0x52,
    0x51, 0x5F, 0x58, 0xF8, 0x81, 0x42, 0x48, 0x83, 0x2D, 0xF0, 0xA9, 0x7D, 0x79, //
NOLINT(readability-magic-numbers)
    0x2B, 0xF1, 0x68, 0xC2, 0x22, 0xC0, 0x0C, 0x72, 0x63, 0x37, 0xBF, 0xEC, 0x72,
0x97, 0xD4, 0xA5, 0x91, 0x2E, 0x1F,
    0xA3, 0x78, 0x9A, 0xCE, 0xFE, 0x27, 0x7F, 0x2B, 0x85, 0x7D, 0x22, 0x2C, 0x0D, //
NOLINT(readability-magic-numbers)
    0x1E, 0x10, 0xB7, 0xFF, 0x9A, 0xA7, 0x99, 0xD2, 0xB9, 0x40, 0x53, 0xB3, 0xA9,
0x52, 0x5D, 0xBD, 0xC8, 0x12, 0x8D,
    0x39, 0xD7, 0x97, 0x03, 0xD2, 0x80, 0x21, 0xC3, 0xA7, 0x8B, 0xE3, 0x3D, 0xF0, //
NOLINT(readability-magic-numbers)
    0x4D, 0x4C, 0x4D, 0xC4, 0xC7, 0xE5, 0xE4, 0x35, 0x75, 0xAA, 0x45, 0x3B, 0x9C,
0x64, 0xC1, 0x94, 0x6E, 0x15, 0x0A,
    0xE8, 0x84, 0xCD, 0xFC, 0x7A, 0xBC, 0x5C, 0x8C, 0xA8, 0x95, 0x07, 0x79, 0x4E, //
NOLINT(readability-magic-numbers)
};
void psa_rsa2048_pem_format_import_example (void)
{
/* This example imports an RSA2048 keypair of PEM format, performs signing and
verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length = 0U;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
```

```
    psa_key_attributes_t read_attributes = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* Import the private key (This includes both the private and public key) */
    mbedtls_pk_context ctx_rsa;
    mbedtls_pk_init(&ctx_rsa);
 /* For keys other than ASN format, mbedtls_pk_parse_key and
mbedtls_pk_import_into_psa APIs must be called instead of psa_import_key */
 if (PSA_SUCCESS !=
       mbedtls_pk_parse_key(&ctx_rsa, RSAKeydata, sizeof(RSAKeydata), NULL, 0,
mbedtls_psa_get_random,
                            MBEDTLS_PSA_RANDOM_STATE))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != mbedtls_pk_import_into_psa(&ctx_rsa, &attributes, &key_handle))
    {
       debugger_break();
    }
    mbedtls_pk_free(&ctx_rsa);
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
```

```
    {
        debugger_break();
    }
 /* Sign message using the key */
 if (PSA_SUCCESS !=
        psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature, sizeof(signature),
                      &signature_length))
    {
        debugger_break();
    }
 if (0 != memcmp(signature, &RSAExpectedSignature, sizeof(RSAExpectedSignature)))
    {
        debugger_break();
    }
 /* Verify the signature */
 if (PSA_SUCCESS !=
        psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), RSAExpectedSignature,
 sizeof(RSAExpectedSignature)))
    {
        debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
        debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
}
```

CHACHA20 Example

This is an example on using the PSA Crypto API to import an ChaCha20 key, encrypting and

decrypting single block data.

```
/* Test vector from http://www.rfc-editor.org/rfc/rfc8439.html#section-2.8.2 */
const uint8_t chacha_key[] =
{
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, // NOLINT(readability-magic-
numbers)
    0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f, // NOLINT(readability-magic-
numbers)
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, // NOLINT(readability-magic-
numbers)
    0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f // NOLINT(readability-magic-
numbers)
};
const uint8_t chacha_iv[] =
{
    0x07, 0x00, 0x00, 0x00,                       /* 32-bit common part */
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47 /* 64-bit IV */
};
const uint8_t chacha_plaintext[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C,
0x0D, 0x0E, 0x0F, // NOLINT(readability-magic-numbers)
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C,
0x1D, 0x1E, 0x1F, // NOLINT(readability-magic-numbers)
    0x20, 0x21, 0x22,
0x23                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chacha_ciphertext[] =
{
    0x9F, 0x7A, 0xEB, 0x5E, 0x05, 0xF8, 0x46, 0xBD, 0x1D, 0xEB, 0x85, 0xF0, 0x3A,
0x8C, 0x04, 0xA1, // NOLINT(readability-magic-numbers)
    0xD1, 0xD1, 0x9A, 0x2C, 0x1D, 0x14, 0x78, 0xC9, 0xC5, 0x93, 0xCA, 0x9C, 0x49,
0x9F, 0x1D, 0xBA, // NOLINT(readability-magic-numbers)
```

```
    0x6E, 0xBF, 0xE9,
0x1B                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chacha_mac[] =
{
    0xC8, 0xB3, 0xFA, 0xDC, 0xF9, 0x1F, 0x78, 0x97, // NOLINT(readability-magic-
numbers)
    0xCD, 0xD7, 0x54, 0xE2, 0xFC, 0x0B, 0xE1, 0xE5 // NOLINT(readability-magic-
numbers)
};
typedef struct import_key_items_symmetric_s
{
 const uint8_t * key;
 size_t          key_size;
 const uint8_t * nonce_iv;          // Nomenclature 'nonce' OR 'iv' depends on the
algorithm. Eg: This is Nonce for CCM and IV for GCM.
 size_t          nonce_iv_size;
 const uint8_t * aad;
 size_t          aad_size;
 const uint8_t * mac;
 size_t          mac_size;
 const uint8_t * plaintext;
 size_t          plaintext_size;
 const uint8_t * ciphertext;
 size_t          ciphertext_size;
} import_key_items_symmetric_t;
const import_key_items_symmetric_t chacha_testdata =
{
    chacha_key,
 sizeof(chacha_key),
    chacha_iv,
 sizeof(chacha_iv),
    NULL,
```

```
    0,
    chacha_mac,
    16,
    chacha_plaintext,
 sizeof(chacha_plaintext),
    chacha_ciphertext,
 sizeof(chacha_ciphertext),
};
void psa_crypto_chacha20_example (void)
{
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = 0;
 size_t encr_len = 0;
 size_t decr_len = 0;
    uint8_t encrypt[200];             // NOLINT(readability-magic-numbers)
    uint8_t decrypt[200];             // NOLINT(readability-magic-numbers)
    import_key_items_symmetric_t * p_import_key_items = (import_key_items_symmetric_t
*) &chacha_testdata;
    psa_key_attributes_t          attributes         = PSA_KEY_ATTRIBUTES_INIT;
 const psa_algorithm_t          alg                = PSA_ALG_STREAM_CIPHER;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
```

```
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_CHACHA20);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
    psa_set_key_bits(&attributes, 256); // NOLINT(readability-magic-numbers)
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(PSA_KEY_LIFETIME_VOLATILE))
    {
       psa_set_key_id(&attributes, (psa_key_id_t) 2);
    }
 if (PSA_SUCCESS != psa_import_key(&attributes, p_import_key_items->key,
p_import_key_items->key_size, &key_handle))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_cipher_encrypt(key_handle, alg, p_import_key_items->plaintext,
p_import_key_items->plaintext_size, encrypt,
 sizeof(encrypt), &encr_len))
    {
 /* Encryption failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_cipher_decrypt(key_handle, alg, encrypt, encr_len, decrypt,
sizeof(decrypt), &decr_len))
    {
 /* Decryption failed */
       debugger_break();
    }
 if (p_import_key_items->plaintext_size == decr_len)
    {
       debugger_break();
    }
 if (0 != memcmp(p_import_key_items->plaintext, decrypt,
p_import_key_items->plaintext_size))
    {
```

```
        debugger_break();
    }
 /* Destroy the key */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Destroying the key handle failed */
        debugger_break();
    }
}
```

CHACHA20-POLY1305 Example

This is an example on using the PSA Crypto API to import an ChaCha20 key, encrypting/decrypting single block data and generates the MAC (authentication tag)

```
const uint8_t chachapoly_key[] =
{
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, // NOLINT(readability-magic-
numbers)
    0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f, // NOLINT(readability-magic-
numbers)
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, // NOLINT(readability-magic-
numbers)
    0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f // NOLINT(readability-magic-
numbers)
};
const uint8_t chachapoly_iv[] =
{
    0x07, 0x00, 0x00, 0x00,                        /* 32-bit common part */
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47 /* 64-bit IV */
};
const uint8_t chachapoly_plaintext[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C,
0x0D, 0x0E, 0x0F, // NOLINT(readability-magic-numbers)
```

```
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C,
0x1D, 0x1E, 0x1F, // NOLINT(readability-magic-numbers)
    0x20, 0x21, 0x22,
0x23                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chachapoly_ciphertext[] =
{
    0x9F, 0x7A, 0xEB, 0x5E, 0x05, 0xF8, 0x46, 0xBD, 0x1D, 0xEB, 0x85, 0xF0, 0x3A,
0x8C, 0x04, 0xA1, // NOLINT(readability-magic-numbers)
    0xD1, 0xD1, 0x9A, 0x2C, 0x1D, 0x14, 0x78, 0xC9, 0xC5, 0x93, 0xCA, 0x9C, 0x49,
0x9F, 0x1D, 0xBA, // NOLINT(readability-magic-numbers)
    0x6E, 0xBF, 0xE9,
0x1B                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chachapoly_mac[] =
{
    0xC8, 0xB3, 0xFA, 0xDC, 0xF9, 0x1F, 0x78, 0x97, // NOLINT(readability-magic-
numbers)
    0xCD, 0xD7, 0x54, 0xE2, 0xFC, 0x0B, 0xE1, 0xE5 // NOLINT(readability-magic-
numbers)
};
// typedef struct import_key_items_symmetric_s
// {
// const uint8_t * key;
// size_t key_size;
// const uint8_t * nonce_iv; // Nomenclature 'nonce' OR 'iv' depends on the
algorithm. Eg: This is Nonce for CCM and IV for GCM.
// size_t nonce_iv_size;
// const uint8_t * aad;
// size_t aad_size;
// const uint8_t * mac;
// size_t mac_size;
```

```
// const uint8_t * plaintext;
// size_t plaintext_size;
// const uint8_t * ciphertext;
// size_t ciphertext_size;
// } import_key_items_symmetric_t;
const import_key_items_symmetric_t chachapoly_testdata =
{
    chachapoly_key,
 sizeof(chachapoly_key),
    chachapoly_iv,
 sizeof(chachapoly_iv),
    NULL,
    0,
    chachapoly_mac,
    16,
    chachapoly_plaintext,
 sizeof(chachapoly_plaintext),
    chachapoly_ciphertext,
 sizeof(chachapoly_ciphertext),
};
void psa_crypto_chachapoly_example (void)
{
 enum
    {
       block_size = 16,
       part_size = block_size,
    };
 const psa_algorithm_t    alg       = PSA_ALG_CHACHA20_POLY1305;
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = 0;
 size_t encr_len = 0;
 size_t decr_len = 0;
    uint8_t encrypt[200];             // NOLINT(readability-magic-numbers)
    uint8_t decrypt[200];             // NOLINT(readability-magic-numbers)
```

```
    import_key_items_symmetric_t * p_import_key_items = (import_key_items_symmetric_t
*) &chachapoly_testdata;
    psa_key_attributes_t          attributes         = PSA_KEY_ATTRIBUTES_INIT;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_CHACHA20);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
    psa_set_key_bits(&attributes, 256); // NOLINT(readability-magic-numbers)
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(PSA_KEY_LIFETIME_VOLATILE))
    {
       psa_set_key_id(&attributes, (psa_key_id_t) 2);
    }
 if (PSA_SUCCESS != psa_import_key(&attributes, p_import_key_items->key,
p_import_key_items->key_size, &key_handle))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_aead_encrypt(key_handle, alg, p_import_key_items->nonce_iv,
p_import_key_items->nonce_iv_size, NULL, 0,
                         p_import_key_items->plaintext,
```

```
p_import_key_items->plaintext_size, encrypt, sizeof(encrypt),
                        &encr_len))
    {
       debugger_break();
    }
 if (p_import_key_items->plaintext_size + p_import_key_items->mac_size != encr_len)
    {
       debugger_break();
    }
 if (0 != memcmp(p_import_key_items->ciphertext, encrypt,
p_import_key_items->plaintext_size))
    {
       debugger_break();
    }
 if (0 !=
       memcmp(p_import_key_items->mac, &encrypt[p_import_key_items->plaintext_size],
p_import_key_items->mac_size))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_aead_decrypt(key_handle, alg, p_import_key_items->nonce_iv,
p_import_key_items->nonce_iv_size, NULL, 0,
                        encrypt, encr_len, decrypt, sizeof(decrypt), &decr_len))
    {
       debugger_break();
    }
 if (p_import_key_items->plaintext_size != decr_len)
    {
       debugger_break();
    }
 if (0 != memcmp(p_import_key_items->plaintext, decrypt,
p_import_key_items->plaintext_size))
    {
```

```
        debugger_break();
    }
 /* Destroy the key */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
        debugger_break();
    }
}
```

ML-KEM Example

This is an example on using the PSA Crypto API to generate an MLKEM-512 key, encapsulating and decapsulating ciphertext.

```
void psa_mlkem512_keygen_example (void)
{
 mbedtls_platform_context ctx                  = {0};
 size_t               bits                     = PSA_KEY_BITS_ML_KEM_512;
    psa_key_handle_t    key_handle               = 0;
    psa_key_id_t         output_key_handle       = 0;
    psa_key_id_t         decap_output_key_handle = 0;
    psa_key_attributes_t attributes               = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t shared_secret_attributes = PSA_KEY_ATTRIBUTES_INIT;
    uint8_t
ciphertext[PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512)] __ALIGNED(4);
 size_t               ciphertext_len =
PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512);
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
        debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
```

```
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE);
    psa_set_key_algorithm(&attributes, PSA_ALG_ML_KEM);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_KEM_KEY_PAIR);
    psa_set_key_bits(&attributes, bits);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Key generation failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&shared_secret_attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE);
    psa_set_key_algorithm(&shared_secret_attributes, PSA_ALG_ML_KEM);
    psa_set_key_type(&shared_secret_attributes, PSA_KEY_TYPE_RAW_DATA);
    psa_set_key_bits(&shared_secret_attributes,
PSA_BYTES_TO_BITS(PSA_ML_KEM_SHARED_SECRET_SIZE));
    psa_set_key_lifetime(&shared_secret_attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS !=
       psa_encapsulate(key_handle, PSA_ALG_ML_KEM, &shared_secret_attributes,
&output_key_handle, &ciphertext[0],
                        ciphertext_len, &ciphertext_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_decapsulate(key_handle, PSA_ALG_ML_KEM, &ciphertext[0], ciphertext_len,
&shared_secret_attributes,
```

```
                        &decap_output_key_handle))
    {
 /* Key decapsulation failed */
       debugger_break();
    }
}
/* Two 32-byte seed values, d and z, concatenated */
uint8_t mlkem_import_keypair[PSA_KEY_EXPORT_ML_KEM_KEY_PAIR_MAX_SIZE] __ALIGNED(4) =
{
    0x1E, 0xB4, 0x40, 0x0A, 0x01, 0x62, 0x9D, 0x51, 0x79, 0x74, 0xE2, 0xCD, 0x85,
0xB9, 0xDE, 0xF5, // NOLINT(readability-magic-numbers)
    0x90, 0x82, 0xDE, 0x50, 0x8E, 0x6F, 0x9C, 0x2B, 0x0E, 0x34, 0x1E, 0x12, 0x96,
0x59, 0x55, 0xCA, // NOLINT(readability-magic-numbers)
    0x1A, 0x39, 0x41, 0x11, 0x16, 0x38, 0x03, 0xFE, 0x2E, 0x85, 0x19, 0xC3, 0x35,
0xA6, 0x86, 0x75, // NOLINT(readability-magic-numbers)
    0x56, 0x33, 0x8E, 0xAD, 0xAF, 0xA2, 0x2B, 0x5F, 0xC5, 0x57, 0x43, 0x05, 0x60,
0xCC, 0xD6, 0x93 // NOLINT(readability-magic-numbers)
};
void psa_mlkem512_keypair_import_export_example (void)
{
 mbedtls_platform_context ctx                  = {0};
 size_t               bits                     = PSA_KEY_BITS_ML_KEM_512;
    psa_key_handle_t    key_handle               = 0;
    psa_key_id_t         output_key_handle       = 0;
    psa_key_id_t         decap_output_key_handle = 0;
    psa_key_attributes_t attributes               = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t shared_secret_attributes = PSA_KEY_ATTRIBUTES_INIT;
    uint8_t
ciphertext[PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512)] __ALIGNED(4);
 size_t               ciphertext_len =
PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512);
    uint8_t
export_keypair[PSA_KEY_GEN_ML_KEM_KEY_PAIR_MAX_SIZE(PSA_KEY_BITS_ML_KEM_512)];
 size_t               export_keypair_len = 0;
```

```
/* Setup the platform; initialize the SCE */
if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
   {
/* Platform initialization failed */
      debugger_break();
   }
if (PSA_SUCCESS != psa_crypto_init())
   {
/* PSA Crypto Initialization failed */
      debugger_break();
   }
/* Set key attributes */
   psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE | PSA_KEY_USAGE_EXPORT);
   psa_set_key_algorithm(&attributes, PSA_ALG_ML_KEM);
   psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_KEM_KEY_PAIR);
   psa_set_key_bits(&attributes, bits);
   psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
/* Import key pair */
if (PSA_SUCCESS !=
      psa_import_key(&attributes, &mlkem_import_keypair[0],
PSA_KEY_GEN_ML_KEM_KEY_PAIR_MAX_SIZE(bits), &key_handle))
   {
/* Key generation failed */
      debugger_break();
   }
/* Set key attributes */
   psa_set_key_usage_flags(&shared_secret_attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE);
   psa_set_key_algorithm(&shared_secret_attributes, PSA_ALG_ML_KEM);
   psa_set_key_type(&shared_secret_attributes, PSA_KEY_TYPE_RAW_DATA);
   psa_set_key_bits(&shared_secret_attributes,
PSA_BYTES_TO_BITS(PSA_ML_KEM_SHARED_SECRET_SIZE));
   psa_set_key_lifetime(&shared_secret_attributes, PSA_KEY_LIFETIME_VOLATILE);
```

```
 if (PSA_SUCCESS !=
       psa_encapsulate(key_handle, PSA_ALG_ML_KEM, &shared_secret_attributes,
&output_key_handle, &ciphertext[0],
                        ciphertext_len, &ciphertext_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_decapsulate(key_handle, PSA_ALG_ML_KEM, &ciphertext[0], ciphertext_len,
&shared_secret_attributes,
                        &decap_output_key_handle))
    {
 /* Key decapsulation failed */
       debugger_break();
    }
 /* Export the key pair */
 if (PSA_SUCCESS !=
       psa_export_key(key_handle, &export_keypair[0],
PSA_KEY_GEN_ML_KEM_KEY_PAIR_MAX_SIZE(bits), &export_keypair_len))
    {
 /* Key pair export failed */
       debugger_break();
    }
 /* Make sure the exported key pair matches the imported buffer */
 if (0 != memcmp(mlkem_import_keypair, export_keypair, export_keypair_len))
    {
 /* Exported key does not match imported key */
       debugger_break();
    }
}
```

ML-DSA Example

These are examples on using the PSA Crypto API to generate an MLDSA-44 key, sign and verify a

message.

Note

There are two ways to use this API. The first uses PSA_ALG_HASH_ML_DSA(hash_alg) which will require the message hash (calculated with the specified hash algorithm) as an input to sign and verify hash functions. See psa_mldsa44_hash_example(). Alternatively, you may use PSA_ALG_HASH_ML_DSA(hash_alg) with sign and verify message functions. In this case, PSA Crypto API will prehash the message for you using the specified hashing algorithm. The second way to use the API is with PSA_ALG_ML_DSA. This will require the use of sign and verify message functions. But this will perform the SHAKE hash calculation using an internal hash implementation as mbedTLS does not currently support SHAKE. Refer to psa_mldsa_message_example().

```
#define MLDSA_MESSAGE_LEN 512U
static uint8_t mldsa_test_message[MLDSA_MESSAGE_LEN] __ALIGNED(4) =
{
    0xF0, 0xA4, 0xA1, 0x8C, 0xCF, 0x98, 0x2E, 0x89, 0x1F, 0x3E, 0x91, 0x73, 0x27,
0x90, 0x1F, 0x73, // NOLINT(readability-magic-numbers)
    0x83, 0x90, 0xE1, 0x7A, 0xB9, 0x5F, 0xF1, 0xFB, 0x7C, 0x76, 0x83, 0x50, 0x38,
0x88, 0x9F, 0x73, // NOLINT(readability-magic-numbers)
    0xAF, 0x0D, 0x24, 0x49, 0x76, 0xDB, 0x62, 0x21, 0x94, 0xFA, 0xDA, 0x41, 0x14,
0x9E, 0x53, 0x4A, // NOLINT(readability-magic-numbers)
    0xCD, 0x96, 0xB9, 0x23, 0x38, 0x52, 0x98, 0x14, 0xEE, 0x18, 0x68, 0x32, 0xB9,
0xC3, 0x83, 0xF9, // NOLINT(readability-magic-numbers)
    0x80, 0xA5, 0x11, 0x7B, 0xDB, 0xED, 0x09, 0x3D, 0xB3, 0x1A, 0xCD, 0x52, 0x73,
0xDF, 0x13, 0xE5, // NOLINT(readability-magic-numbers)
    0xA8, 0x3D, 0x14, 0xBB, 0x82, 0x1F, 0x18, 0xC1, 0x88, 0xFF, 0x36, 0x80, 0x19,
0x16, 0x3C, 0x61, // NOLINT(readability-magic-numbers)
    0x4C, 0x4D, 0x20, 0x27, 0x83, 0xFC, 0x88, 0xE2, 0x52, 0x3C, 0x42, 0x51, 0x14,
0x21, 0x14, 0x25, // NOLINT(readability-magic-numbers)
    0x98, 0x81, 0x48, 0xF5, 0x25, 0xD0, 0x47, 0xA0, 0xCD, 0x95, 0x77, 0x63, 0xCF,
0x5B, 0x79, 0xF6, // NOLINT(readability-magic-numbers)
    0x6B, 0xE1, 0x73, 0x87, 0x04, 0xAC, 0x66, 0xF1, 0x33, 0x39, 0x8B, 0xC9, 0xC0,
0xE5, 0x57, 0x00, // NOLINT(readability-magic-numbers)
    0x63, 0x06, 0xDB, 0x92, 0x0C, 0xA8, 0x43, 0xC0, 0xD5, 0x24, 0xD4, 0xEA, 0xED,
0xBC, 0x1B, 0xDE, // NOLINT(readability-magic-numbers)
    0x05, 0x03, 0x37, 0x6B, 0x22, 0x77, 0x2B, 0x74, 0x14, 0x55, 0x9D, 0x68, 0x90,
0x1C, 0xC5, 0xA1, // NOLINT(readability-magic-numbers)
    0x80, 0x65, 0x20, 0xEB, 0x39, 0x7F, 0xFA, 0xA2, 0x4D, 0x96, 0xFF, 0x2F, 0x20,
0x7B, 0x47, 0xD1, // NOLINT(readability-magic-numbers)
```

```
    0xA2, 0x1C, 0xE6, 0x86, 0x87, 0xD7, 0x65, 0x83, 0xF2, 0x15, 0x11, 0x07, 0x4B,
0x90, 0x56, 0xD4, // NOLINT(readability-magic-numbers)
    0x54, 0xF5, 0x65, 0x35, 0x8D, 0xBE, 0xFA, 0xEC, 0x8A, 0xBE, 0x43, 0x88, 0xFB,
0x1D, 0x8A, 0x25, // NOLINT(readability-magic-numbers)
    0x5D, 0xE8, 0xF9, 0xE7, 0xF1, 0x1A, 0x96, 0xE6, 0x9A, 0xAD, 0xA9, 0x34, 0x6C,
0x84, 0x61, 0x90, // NOLINT(readability-magic-numbers)
    0x1E, 0xFA, 0x4D, 0xAA, 0xA1, 0xA4, 0xBA, 0x9B, 0x04, 0xD8, 0xA1, 0x1E, 0xBD,
0x00, 0x31, 0x9B, // NOLINT(readability-magic-numbers)
    0x4A, 0x7D, 0x8B, 0xAC, 0x21, 0x74, 0xCF, 0xA2, 0xA0, 0x06, 0xD9, 0x48, 0x59,
0xFE, 0xC5, 0x2B, // NOLINT(readability-magic-numbers)
    0xDC, 0x01, 0x88, 0x2C, 0xB9, 0xD4, 0xA3, 0xE8, 0x95, 0x1A, 0x5C, 0xC1, 0xBD,
0x36, 0xEC, 0x74, // NOLINT(readability-magic-numbers)
    0xF6, 0x03, 0x33, 0x31, 0xB0, 0x3E, 0x92, 0xA7, 0x27, 0x07, 0x7F, 0x44, 0x8D,
0x4F, 0xB5, 0x60, // NOLINT(readability-magic-numbers)
    0x9A, 0x5E, 0x55, 0xAA, 0x75, 0x28, 0x9F, 0x7D, 0xAF, 0x3B, 0xEA, 0xB3, 0x1C,
0xF7, 0xAA, 0x1F, // NOLINT(readability-magic-numbers)
    0x4B, 0x66, 0x98, 0x0F, 0x5F, 0x5F, 0x8A, 0x31, 0xAC, 0xFE, 0x55, 0x58, 0x52,
0x52, 0x24, 0x7F, // NOLINT(readability-magic-numbers)
    0xCD, 0x78, 0xB9, 0x67, 0x56, 0x38, 0xDB, 0x8E, 0x12, 0x92, 0x91, 0x3A, 0x6F,
0x3E, 0xA5, 0x7C, // NOLINT(readability-magic-numbers)
    0x60, 0xD1, 0x97, 0xB1, 0xCF, 0x83, 0xD8, 0x53, 0xC6, 0xAB, 0xE3, 0x50, 0xB7,
0xB1, 0x10, 0x40, // NOLINT(readability-magic-numbers)
    0x76, 0x5A, 0xF1, 0xC1, 0x74, 0x0A, 0xF1, 0xE3, 0x18, 0x68, 0x92, 0x37, 0xB3,
0x50, 0xDD, 0x8F, // NOLINT(readability-magic-numbers)
    0x4F, 0x0C, 0x0A, 0x63, 0x93, 0x9A, 0xDA, 0xBF, 0x39, 0x2D, 0x71, 0x47, 0x1B,
0xF3, 0xF7, 0xA5, // NOLINT(readability-magic-numbers)
    0xCA, 0x44, 0x8A, 0x4D, 0xCD, 0xF4, 0x89, 0x6A, 0x34, 0x2B, 0x4B, 0xA9, 0x42,
0x12, 0x81, 0x37, // NOLINT(readability-magic-numbers)
    0xF3, 0xFF, 0x44, 0xAE, 0x57, 0xDB, 0x85, 0x00, 0xA7, 0xBF, 0xB6, 0xFD, 0x74,
0x25, 0x0E, 0x63, // NOLINT(readability-magic-numbers)
    0x12, 0xD5, 0x4A, 0x35, 0x88, 0xD0, 0x8E, 0x93, 0xBA, 0x62, 0x3E, 0x18, 0x9B,
0x81, 0x7E, 0xEF, // NOLINT(readability-magic-numbers)
    0x2B, 0xE6, 0x6D, 0x2A, 0x57, 0x09, 0x7B, 0x14, 0x7A, 0x13, 0x1D, 0x5C, 0x88,
0x43, 0xCC, 0x73, // NOLINT(readability-magic-numbers)
```

```
    0x43, 0xE6, 0x2B, 0xC0, 0x02, 0xAD, 0x1E, 0x60, 0x29, 0x9B, 0xB7, 0xD2, 0xF0,
0xFF, 0x14, 0xC5, // NOLINT(readability-magic-numbers)
    0x1B, 0x3E, 0x41, 0x41, 0x98, 0x52, 0x4A, 0x3D, 0x96, 0x4C, 0xE2, 0xA6, 0x31,
0xB7, 0x2F, 0x71, // NOLINT(readability-magic-numbers)
    0x34, 0x2E, 0x17, 0xAF, 0x47, 0x99, 0x2E, 0x6C, 0x27, 0x06, 0x7C, 0xA1, 0x47,
0x11, 0x61, 0x5C // NOLINT(readability-magic-numbers)
};
void psa_mldsa44_hash_example (void)
{
 mbedtls_platform_context ctx       = {0};
    psa_algorithm_t          hash_alg   = PSA_ALG_SHA3_256;
    psa_algorithm_t          alg       = PSA_ALG_HASH_ML_DSA(hash_alg);
    psa_key_handle_t         key_handle = 0;
    psa_key_attributes_t    attributes = PSA_KEY_ATTRIBUTES_INIT;
 size_t               sign_len;
    uint8_t             sign[PSA_ML_DSA_SIGNATURE_SIZE(PSA_KEY_BITS_ML_DSA_44)]
__ALIGNED(4);
    psa_hash_operation_t hash_operation = {0};
    uint8_t             payload_hash[PSA_HASH_MAX_SIZE] __ALIGNED(4);
 size_t               payload_hash_len;
 unsigned char       payload[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
```

```
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_KEY_PAIR);
    psa_set_key_bits(&attributes, PSA_KEY_BITS_ML_DSA_44);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Key generation failed */
       debugger_break();
    }
 /* Calculate the hash of the message */
 if (PSA_SUCCESS != psa_hash_setup(&hash_operation, hash_alg))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_hash_update(&hash_operation, payload, sizeof(payload)))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_hash_finish(&hash_operation, &payload_hash[0], sizeof(payload_hash),
&payload_hash_len))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, alg, payload_hash, payload_hash_len, sign,
sizeof(sign), &sign_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, alg, payload_hash, payload_hash_len, sign,
```

```
sign_len))
    {
 /* Key decapsulation failed */
       debugger_break();
    }
}
void psa_mldsa44_message_example (void)
{
 mbedtls_platform_context ctx       = {0};
    psa_algorithm_t          alg       = PSA_ALG_ML_DSA;
    psa_key_handle_t         key_handle = 0;
    psa_key_attributes_t    attributes = PSA_KEY_ATTRIBUTES_INIT;
 size_t sign_len;
    uint8_t sign[PSA_ML_DSA_SIGNATURE_SIZE(PSA_KEY_BITS_ML_DSA_44)] __ALIGNED(4);
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes,
                            PSA_KEY_USAGE_SIGN_MESSAGE | PSA_KEY_USAGE_VERIFY_MESSAGE
| PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_KEY_PAIR);
    psa_set_key_bits(&attributes, PSA_KEY_BITS_ML_DSA_44);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
```

```
    {
 /* Key generation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_sign_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sizeof(sign),
                        &sign_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_verify_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sign_len))
    {
 /* Key decapsulation failed */
       debugger_break();
    }
}
/* The 32-byte seed value */
uint8_t mldsa_import_keypair[PSA_KEY_EXPORT_ML_DSA_KEY_PAIR_MAX_SIZE] __ALIGNED(4) =
{
    0x4B, 0xE7, 0xA0, 0x1A, 0x99, 0xA5, 0xE5, 0xBC, 0xFE, 0x3C, 0x06, 0x78, 0x5D,
0x8E, 0x4E, 0xC6, // NOLINT(readability-magic-numbers)
    0x64, 0x08, 0x22, 0x27, 0xD8, 0x67, 0x04, 0xE9, 0xE4, 0x48, 0x62, 0x62, 0x3A,
0x05, 0xC8, 0xB3, // NOLINT(readability-magic-numbers)
};
void psa_mldsa44_keypair_import_export_example (void)
{
    psa_key_id_t         key_handle = 0;
    psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
    uint8_t             sign[PSA_ML_DSA_SIGNATURE_SIZE(PSA_KEY_BITS_ML_DSA_44)]
__ALIGNED(4);
```

```
 size_t               sign_len;
    uint8_t
public_key[PSA_KEY_GEN_ML_DSA_PUB_KEY_SIZE(PSA_KEY_BITS_ML_DSA_44)] __ALIGNED(4);
 size_t               public_key_len = 0;
    psa_algorithm_t      alg            = PSA_ALG_HASH_ML_DSA(PSA_ALG_SHA3_512);
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_MESSAGE |
PSA_KEY_USAGE_VERIFY_MESSAGE);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_KEY_PAIR);
    psa_set_key_bits(&attributes, PSA_KEY_BITS_ML_DSA_44);
 /* Import key pair */
 if (PSA_SUCCESS !=
       psa_import_key(&attributes, &mldsa_import_keypair[0],
                      PSA_KEY_GEN_ML_DSA_KEY_PAIR_MAX_SIZE(PSA_KEY_BITS_ML_DSA_44),
&key_handle))
    {
 /* Key import failed */
       debugger_break();
    }
 /* Sign the message */
 if (PSA_SUCCESS !=
       psa_sign_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sizeof(sign),
                         &sign_len))
    {
 /* Sign message failed */
       debugger_break();
    }
 /* Verify the signature */
 if (PSA_SUCCESS !=
       psa_verify_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sign_len))
```

```
    {
 /* Verify message failed */
       debugger_break();
    }
 /* Export public key */
 if (PSA_SUCCESS !=
       psa_export_public_key(key_handle, &public_key[0],
PSA_KEY_GEN_ML_DSA_PUB_KEY_SIZE(PSA_KEY_BITS_ML_DSA_44),
                              &public_key_len))
    {
 /* Public key export failed */
       debugger_break();
    }
 /* Destroy key pair */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Key destroy failed */
       debugger_break();
    }
 /* Import public key */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_PUBLIC_KEY);
 if (PSA_SUCCESS != psa_import_key(&attributes, &public_key[0], public_key_len,
&key_handle))
    {
 /* Key import failed */
       debugger_break();
    }
 /* Verify signature */
 if (PSA_SUCCESS !=
       psa_verify_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sign_len))
    {
 /* Verify message failed */
       debugger_break();
```

```
    }
}
```

Migrating to MbedTLS 3.6.0

MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS macro is disabled by default to handle shared memory in a secure manner. Note that this increases heap memory usage and code size. The macro can be enabled if all buffers passed to any PSA function reside in memory that is accessible only to the PSA function during its execution or if it possible otherwise for the user to conclude that the user buffers passed into PSA are secure from modification. Refer to https://github.com/Mbed-TLS/mbedtls-docs/blob/main/security-advisories/mbedtls-security-advisory-2024-03.md for more details.

psa_import_key() now only accepts RSA keys in the PSA standard formats. Applications that need other formats like: PKCS#8, SubjectPublicKey, PEM can call mbedtls_pk_parse_{public,}key() followed by mbedtls_pk_import_into_psa().

```
/* Generated using "openssl genrsa -out example_private_key.pem 2048" */
const uint8_t RSAKeydata[] = "-----BEGIN RSA PRIVATE KEY-----\n"
 "MIIEowIBAAKCAQEAplaPU68h4hW/eAlH2hbQl1WjoDT1znCk0O9cnE9lAnB26IUi"
 "78wUAxuTZSlrlKez9FGGIVbUMTy7dTw0ZBanogrKCaAMaRFz5LRv4I8RQzsDxxqe"
 "nud9Y6mVRWb9hyutL/yBwDd6jzAwcviAhcatTv0E5wpEJ6ezMDyaiSCbn84hfb0T"
 "ZPXQVGhmjii3f0FfFWW9ce70qkuwdkvO6HBrOGVi2++b11As+Uh/7pxyOb7wpE7K"
 "X64nohaCtXgNnV8Hr+LOPQRHEbGTCPOfx1davKEIYFvm8hVxqv20csMD5uc8AJCd"
 "0cdom13KcOpVXCnrFPNJt0lITOcH/fBzp4xoaQIDAQABAoIBAGc56qKVWeKzempk"
 "4AlRBUwVYoEGvNDLiQz8rq12fAoCf1iXXvIP5Q90qokqJlBPrtbdTO8dsnuH5BHC"
 "NgUzJf7i0TUd9PWzVgfFjnR+dMkTM6n5NB0LLf6OfEtguc+L5GOWQXNnOpDn5/lB"
 "jIj4ng8Z6FP1RAyT/xjdU03sRYfq5OPlDZ8+ck7WwQ8VxSBKsJcZVv+K4r64Vmr1"
 "MO1zfBYuwafi4oO9Z9x89Iil9GBcYuCvhLvDoWhnTUxlOTfz+Jt+qwlmVimEz6xs"
 "6ew1Ox7e6SInly3IAXe9E84yP9MDmiGkCrVr8s8GUybZ5t/yeZ3g5lPYSQ/FiuHY"
 "kvJCH4ECgYEA2IthRDkElOQGqJ9T/LjgcIVLQs0F6i7zZWg6UKv18EmG7uHd6tsZ"
 "byFfFiBat80rq6q4AUx2OOE7+LI5MxIvgRgLk14WV8H2OKK/lfxOFb6a4feGM7bH"
 "VdERQSMNeGq0bP9IOPxU1EvFp0Vfv7cfGsQhcAPgA8up61jWREhI+fECgYEAxKVT"
 "9pKinIwgjsHgmxalaNbnRf+bcqtdS98SB2MuICg8ubcBSrjm4wtGe3oCX/PbM1GV"
 "FvEDKl4TyeWFIH4MStsyDHYxwoV2C01bBHEtHTYBnjeIShMruIomsG9GeTrNmyzg"
 "dVdSg5/lxYOxuZSXvfcbyIMsW29ddgeICu9RHfkCgYBaljQibhfUkW+Xqs9fsZdy"
 "etB5KXuH9AwuJ+P9S3KfCqM/240SaoXBT5yPjQlmSpYyQkCnim0Kbm7AIw56pujo"
 "gD6Xb4y5OZLfLnYnMF0aC5qPXRTvHU9WPxeQwDEqZrkDv+der3BtPyV4TDU55klE"
 "0TeLvJNCAzkaExfPiM2+gQKBgQC5ceBYq7hGQa+CcTvLhfO/hsrbrE2AOjLllhx+"
```

```
 "cv3QvaFm0jqZqP20J7H0R/9tVZ7mKo2a8Pa3QbkPsS92kOguv7/XGK+cbhgAWJb4"
 "/XI6FfA4sM4KbUHR6hbKVGX1dYroR831WsAp+OTK+4LjLEpdj2fYFDwEjmVpJXka"
 "Ns4coQKBgGESrtpJF7OQG4xcXwR2ZiJESPvMKVmtvxRzDrc9gmoiIIIKx1fimwCv"
 "RtOz1bQcXMRw21+ZAZpen3ahWh63KC0KxMSNBJTXdczlf2uprVkSPtmAjV4qBgwv"
 "tqjmP0lnGc9wkJsVvGmMAAfQOWgxs9h/VH/b+6biEkzaaZLOyABV"
 "-----END RSA PRIVATE KEY-----\n";
/* Expected signature below is calculated using the mbedCrypto software version */
const uint8_t RSAExpectedSignature[256] =
{
    0x25, 0xC0, 0x15, 0x64, 0xD3, 0xF7, 0xC1, 0xB7, 0xA8, 0x9B, 0x56, 0x1C, 0xB5,
0xA4, 0xA5, 0x0D, 0x61, 0x52, 0x32,
    0x0C, 0x4E, 0xE8, 0xCA, 0x4B, 0x9E, 0xA2, 0x4D, 0x35, 0x0E, 0xB1, 0xA1, 0x5B, //
NOLINT(readability-magic-numbers)
    0x5B, 0xD8, 0xC1, 0x93, 0x28, 0x60, 0x90, 0x18, 0x37, 0x88, 0x66, 0xD4, 0xED,
0x37, 0x6B, 0xEC, 0x48, 0xFF, 0x3D,
    0xFB, 0x99, 0xB7, 0xEF, 0x3C, 0x8D, 0x25, 0xE9, 0xF3, 0xCF, 0x02, 0x7C, 0x7D, //
NOLINT(readability-magic-numbers)
    0xB9, 0x4D, 0x2F, 0x2F, 0x1E, 0x30, 0x48, 0x54, 0x18, 0xE4, 0x51, 0xD5, 0xC1,
0xB7, 0x3B, 0x0D, 0xE4, 0xB4, 0x19,
    0x7B, 0x9B, 0xF4, 0x35, 0x82, 0xCC, 0x91, 0x4C, 0x10, 0xE9, 0xB6, 0xF1, 0x0A, //
NOLINT(readability-magic-numbers)
    0x23, 0xEB, 0x7D, 0x51, 0x47, 0x36, 0xFE, 0x13, 0xAF, 0x3C, 0x23, 0x9F, 0x7E,
0xFC, 0xCF, 0x7A, 0x7C, 0x2D, 0xDB,
    0xD9, 0xDA, 0xEB, 0xF7, 0xB5, 0x6B, 0x8D, 0xE0, 0x18, 0x9A, 0x5B, 0xB7, 0x0A, //
NOLINT(readability-magic-numbers)
    0xA3, 0x4E, 0xE1, 0xB7, 0xF7, 0xD1, 0x94, 0xD5, 0x7A, 0xD3, 0x27, 0xE2, 0x1F,
0x3A, 0xEB, 0xF0, 0x83, 0x10, 0x52,
    0x51, 0x5F, 0x58, 0xF8, 0x81, 0x42, 0x48, 0x83, 0x2D, 0xF0, 0xA9, 0x7D, 0x79, //
NOLINT(readability-magic-numbers)
    0x2B, 0xF1, 0x68, 0xC2, 0x22, 0xC0, 0x0C, 0x72, 0x63, 0x37, 0xBF, 0xEC, 0x72,
0x97, 0xD4, 0xA5, 0x91, 0x2E, 0x1F,
    0xA3, 0x78, 0x9A, 0xCE, 0xFE, 0x27, 0x7F, 0x2B, 0x85, 0x7D, 0x22, 0x2C, 0x0D, //
NOLINT(readability-magic-numbers)
    0x1E, 0x10, 0xB7, 0xFF, 0x9A, 0xA7, 0x99, 0xD2, 0xB9, 0x40, 0x53, 0xB3, 0xA9,
```

```
0x52, 0x5D, 0xBD, 0xC8, 0x12, 0x8D,
    0x39, 0xD7, 0x97, 0x03, 0xD2, 0x80, 0x21, 0xC3, 0xA7, 0x8B, 0xE3, 0x3D, 0xF0, //
NOLINT(readability-magic-numbers)
    0x4D, 0x4C, 0x4D, 0xC4, 0xC7, 0xE5, 0xE4, 0x35, 0x75, 0xAA, 0x45, 0x3B, 0x9C,
0x64, 0xC1, 0x94, 0x6E, 0x15, 0x0A,
    0xE8, 0x84, 0xCD, 0xFC, 0x7A, 0xBC, 0x5C, 0x8C, 0xA8, 0x95, 0x07, 0x79, 0x4E, //
NOLINT(readability-magic-numbers)
};
void psa_rsa2048_pem_format_import_example (void)
{
/* This example imports an RSA2048 keypair of PEM format, performs signing and
verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length = 0U;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* Import the private key (This includes both the private and public key) */
```

```
    mbedtls_pk_context ctx_rsa;
    mbedtls_pk_init(&ctx_rsa);
 /* For keys other than ASN format, mbedtls_pk_parse_key and
mbedtls_pk_import_into_psa APIs must be called instead of psa_import_key */
 if (PSA_SUCCESS !=
       mbedtls_pk_parse_key(&ctx_rsa, RSAKeydata, sizeof(RSAKeydata), NULL, 0,
mbedtls_psa_get_random,
                            MBEDTLS_PSA_RANDOM_STATE))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != mbedtls_pk_import_into_psa(&ctx_rsa, &attributes, &key_handle))
    {
       debugger_break();
    }
    mbedtls_pk_free(&ctx_rsa);
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Sign message using the key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature, sizeof(signature),
                      &signature_length))
    {
       debugger_break();
    }
 if (0 != memcmp(signature, &RSAExpectedSignature, sizeof(RSAExpectedSignature)))
    {
       debugger_break();
    }
 /* Verify the signature */
```

```
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), RSAExpectedSignature,
 sizeof(RSAExpectedSignature)))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
}
```

Function Documentation

◆ RM_PSA_CRYPTO_TRNG_Read()

fsp_err_t RM_PSA_CRYPTO_TRNG_Read (uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes*, uint32_t * *p_num_gen_bytes*)

Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer.

Return values

FSP_SUCCESS	Random number generation successful
FSP_ERR_ASSERTION	NULL input parameter(s).
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- s_generate_16byte_random_data

◆ mbedtls_platform_setup()

int mbedtls_platform_setup (mbedtls_platform_context * *ctx*)

This function initializes the SCE and the TRNG. It **must** be invoked before the crypto library can be used. This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined.

Example:

```
mbedtls_platform_context ctx = {0};

/* Setup the platform; initialize the SCE and the TRNG */

if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
```

Return values

0	Initialization was successful.
MBEDTLS_ERR_PLATFORM_HW_ACCEL_FAILED	SCE Initialization error.

◆ mbedtls_platform_teardown()

void mbedtls_platform_teardown (mbedtls_platform_context * *ctx*)

This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined. It is intended to de-initialize any items that were initialized in the mbedtls_platform_setup() function, but currently is only a placeholder function.

Example:

```
/* De-initialize the platform. This is currently a placeholder function which does
not do anything. */

mbedtls_platform_teardown(&ctx);
```

Return values

N/A	

5.2.16 Sensor

Modules

Sensor Modules.

5.2.17 Storage

Modules

Detailed Description

Storage Modules.

Modules

	MRAM (r_mram) Driver for the MRAM memory on RX MCUs. This module implements the Flash Interface.
	SD/MMC (r_sdhi) Driver for the SD/MMC Host Interface (SDHI) peripheral on RX MCUs. This module implements the SD/MMC Interface.
	XSPI (r_xspi_controller) Driver for the XSPI_CONTROLLER peripheral on RX MCUs. This module implements the SPI Flash Interface.
	XSPI (r_xspi_target) Driver for the XSPI_TARGET peripheral on RX MCUs. This module implements the SPI Flash Interface.
	Block Media RAM (rm_block_media_ram) Middleware that implements a block media interface on RAM. This module implements the Block Media Interface.
	Block Media SD/MMC (rm_block_media_sdmmc) Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface.
	Block Media SPI Flash (rm_block_media_spi) Middleware to implement the block media interface on SPI flash memory. This module implements the Block Media Interface.
	Block Media USB (rm_block_media_usb)

	Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface.
	Block Media Custom Implementation (rm_block_media_user) Middleware that implements a block media interface on the media of your choice. This module implements the Block Media Interface.
	FreeRTOS+FAT Port for RX (rm_freertos_plus_fat) Middleware for the FAT File System control on RX MCUs.
	LittleFS on Flash (rm_littlefs_flash) Middleware for the LittleFS File System control on RX MCUs.
	LittleFS on SPI Flash (rm_littlefs_spi_flash) Middleware for the LittleFS File System control on RX MCUs using external SPI Flash.

5.2.17.1 MRAM (r_mram)

Modules » Storage

Functions

fsp_err_t	R_MRAM_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t	R_MRAM_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t mram_address, uint32_t const num_bytes) PLACE_IN_RAM_SECTION
fsp_err_t	R_MRAM_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t	R_MRAM_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result)
fsp_err_t	R_MRAM_Close (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_MRAM_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status)
fsp_err_t	R_MRAM_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t

	const start_addr, uint32_t const end_addr)
fsp_err_t	R_MRAM_AccessWindowClear (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_MRAM_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode)
fsp_err_t	R_MRAM_Reset (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_MRAM_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_MRAM_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
fsp_err_t	R_MRAM_CallbackSet (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory)
fsp_err_t	R_MRAM_BankSwap (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_MRAM_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info)
fsp_err_t	R_MRAM_AntiRollbackCounterIncrement (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter)
fsp_err_t	R_MRAM_AntiRollbackCounterRefresh (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter)
fsp_err_t	R_MRAM_AntiRollbackCounterRead (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter, uint32_t *const p_count)
fsp_err_t	R_MRAM_UserLockableAreaWrite (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t mram_address, uint32_t const num_bytes)

Detailed Description

Driver for the MRAM memory on RX MCUs. This module implements the Flash Interface.

Overview

The MRAM HAL module APIs allow an application to write and erase MRAM areas that reside within the MCU. The amount of MRAM memory available varies across MCU parts.

Features

The R_MRAM module has the following key features:

- Blocking writing and erasing of MRAM.

- Anti-rollback counter support (select MCUs).

Configuration

Build Time Configurations for r_mram

The following build time configurations are defined in fsp_cfg/r_mram_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > MRAM (r_mram)

This module can be added to the Stacks tab via New Stack > Storage > MRAM (r_mram).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_mram0	Module name.

Clock Configuration

MRAM uses MRAM bus clock for Code MRAM and MRAM clock for Extra MRAM.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Code MRAM Precautions

Code MRAM cannot be accessed while writing or erasing code MRAM. In order to support modifying code MRAM all supporting code must reside in RAM. Code MRAM operation will not return before the operation has completed. By default, the vector table resides in the code MRAM. If an interrupt occurs during the code MRAM operation, then code MRAM will be accessed to fetch the interrupt's starting address and an error will occur. The simplest work-around is to disable interrupts during code MRAM operations. Another option is to copy the vector table to RAM, update the INTB (Interrupt Table Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must ensure that if in a multi-threaded environment, threads running from code MRAM cannot become active while a code MRAM operation is in progress.

Erase condsiderations

The erase API was designed with flash blocks in mind. The Erase API provides a mechanism for quickly overwriting blocks of data at a single time. The block size is determined by the MRAM Unit of Programming size.a

Anti Rollback Counter

Anti-rollback counters are available on selected MCUs and can be used to prevent or limit firmware

downgrade. The counters can only be incremented and never decremented or reset, even with an external programming tool.

Note

The OEM bootloader anti-rollback counter is protected by the security engine and cannot be incremented past the version number specified in the code certificate. This counter works with the First Stage Bootloader (FSBL) set to enabled. For more information, read the 'Secure Boot' section in the 'Security Features' chapter of the user manual of the target MCU.

Anti-Rollback Counter Configuration Setting

When anti-rollback counters are used for one or more non-secure applications, the Anti-Rollback Counter Configuration Setting for ARC_NSEC (ARCCS) must be configured as either a single line (1xn bits) or multi-line counter (mxn bits). Single line is suitable when there is only one non-secure application using the anti-rollback counter. Multi-line is suitable when there are multiple non-secure applications. Before the counter is configured, it can't be used.

This configuration setting is permanent and must be performed with an external programming tool such as Renesas Flash Programmer (RFP). It cannot be configured in e² studio.

The secure application and OEM bootloader anti-rollback counters do not require any initial configuration.

Viewing MRAM contents in e² studio

By default, the contents of code MRAM are cached by e^2 studio. This means that during a debug session, modifications to these memory regions will not be observed by e^2 studio. When debugging applications using e^2 studio, disable the "Allow caching of flash contents" option in the debug configuration in order to view modified MRAM contents (Debug Configuration > Debugger > Debug Tool Settings > Allow caching of flash contents).

Examples

MRAM Basic Example

This is a basic example of writing to MRAM.

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
void r_mram_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the MRAM instance. */
 fsp_err_t err = R_MRAM_Open(&g_mram_ctrl, &g_mram_cfg);
```

```
    assert(FSP_SUCCESS == err);
 /* Disable interrupts to prevent vector table access while MRAM is programming. */
    __CLRPSW_I();
 /* Write 32 bytes to an area of MRAM. */
    err = R_MRAM_Write(&g_mram_ctrl, (uint32_t) g_src, MRAM_WRITE_ADDRESS,
MRAM_EXAMPLE_WRITE_SIZE);
 /* Enable interrupts after MRAM operations are complete. */
    __SETPSW_I();
    assert(FSP_SUCCESS == err);
    assert(0 == memcmp(g_src, (uint8_t *) MRAM_WRITE_ADDRESS,
MRAM_EXAMPLE_WRITE_SIZE));
 /* Disable interrupts to prevent vector table access while MRAM is programming. */
    __CLRPSW_I();
 /* Write 32 bytes to an area of DataMRAM. */
    err = R_MRAM_Write(&g_mram_ctrl, (uint32_t) g_src, DATA_MRAM_WRITE_ADDRESS,
DATA_MRAM_EXAMPLE_WRITE_SIZE);
 /* Enable interrupts after MRAM operations are complete. */
    __SETPSW_I();
    assert(FSP_SUCCESS == err);
    assert(0 == memcmp(g_src, (uint8_t *) DATA_MRAM_WRITE_ADDRESS,
DATA_MRAM_EXAMPLE_WRITE_SIZE));
}
```

Anti-Rollback Counter Basic Example

This example demonstrates basic usage of the anti-rollback counter functions.

```
void r_mram_anti_rollback_counter_example (void)
{
 /* Open the MRAM instance. */
 fsp_err_t err = R_MRAM_Open(&g_mram_ctrl, &g_mram_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
    uint32_t current_count;
 /* Read the current counter value for the secure application */
```

```
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_SEC,
&current_count);
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for the OEM bootloader */
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_OEMBL,
&current_count);
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for the non-secure application if counter is
configured as 1xn bits */
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_0,
&current_count); // NSEC App
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for 4 non-secure applications if counter is
configured as mxn bits */
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_0,
&current_count); // NSEC App 0
    err |= R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_1,
&current_count); // NSEC App 1
    err |= R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_2,
&current_count); // NSEC App 2
    err |= R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_3,
&current_count); // NSEC App 3
    assert(FSP_SUCCESS == err);
 /* Increment the counter value for the secure application. Caution: The increment
cannot be reversed. */
    err = R_MRAM_AntiRollbackCounterIncrement(&g_mram_ctrl, FLASH_ARC_SEC);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	mram_instance_ctrl_t

Data Structure Documentation

◆ mram_instance_ctrl_t

struct mram_instance_ctrl_t

MRAM instance control block. DO NOT INITIALIZE.

Function Documentation

◆ R_MRAM_Open()

fsp_err_t R_MRAM_Open (flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg*)

Initializes the MRAM peripheral. Implements flash_api_t::open.

The Open function initializes the MRAM.

Example:

```
/* Open the MRAM instance. */
fsp_err_t err = R_MRAM_Open(&g_mram_ctrl, &g_mram_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ALREADY_OPEN	The MRAM control block is already open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
FSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the MRAM interrupts are not supported.

◆ R_MRAM_Write()

fsp_err_t R_MRAM_Write (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *mram_address*, uint32_t const *num_bytes*)

Writes to the specified MRAM memory area. Implements flash_api_t::write.

Example:

```
/* Disable interrupts to prevent vector table access while MRAM is programming. */
   __CLRPSW_I();
/* Write 32 bytes to an area of MRAM. */
   err = R_MRAM_Write(&g_mram_ctrl, (uint32_t) g_src, MRAM_WRITE_ADDRESS,
MRAM_EXAMPLE_WRITE_SIZE);
/* Enable interrupts after MRAM operations are complete. */
   __SETPSW_I();
```

Return values

FSP_SUCCESS	Operation successful.
FSP_ERR_IN_USE	The MRAM peripheral is busy with a prior on-going transaction.
FSP_ERR_INVALID_SIZE	Number of bytes provided was invalid
FSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
FSP_ERR_NOT_OPEN	The MRAM API is not Open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.

◆ R_MRAM_Erase()

fsp_err_t R_MRAM_Erase (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks*)

Erases the specified MRAM blocks. Implements flash_api_t::erase by the block_erase_address.

Example:

```
/* Disable interrupts to prevent vector table access while MRAM is programming. */
   __CLRPSW_I();
/* Erase 10 programmable units of MRAM */
fsp_err_t err = R_MRAM_Erase(&g_mram_ctrl, MRAM_ERASE_ADDRESS, 10);
/* Enable interrupts after MRAM operations are complete. */
   __SETPSW_I();
```

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
FSP_ERR_INVALID_ADDRESS	Invalid address specified.
FSP_ERR_IN_USE	Other MRAM operation in progress, or API not initialized
FSP_ERR_NOT_OPEN	The MRAM API is not Open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl

◆ R_MRAM_BlankCheck()

fsp_err_t R_MRAM_BlankCheck (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result*)

Stub function Implements flash_api_t::blankCheck.

Return values

FSP_ERR_UNSUPPORTED	Blank Check is not supported on this MCU.

◆ R_MRAM_Close()

fsp_err_t R_MRAM_Close (flash_ctrl_t *const *p_api_ctrl*)

Releases any resources that were allocated by the Open() or any subsequent MRAM operations. Implements flash_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.

◆ R_MRAM_StatusGet()

fsp_err_t R_MRAM_StatusGet (flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status*)

Query the MRAM peripheral for its status. Implements flash_api_t::statusGet.

Return values

FSP_SUCCESS	MRAM peripheral is ready to use.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_NOT_OPEN	The MRAM API is not Open.

◆ R_MRAM_AccessWindowSet()

fsp_err_t R_MRAM_AccessWindowSet (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr*)

Stub function Implements flash_api_t::accessWindowSet.

Return values

FSP_ERR_UNSUPPORTED	Access window is not supported on this MCU.

◆ R_MRAM_AccessWindowClear()

fsp_err_t R_MRAM_AccessWindowClear (flash_ctrl_t *const *p_api_ctrl*)

Stub function Implements flash_api_t::accessWindowClear.

Return values

FSP_ERR_UNSUPPORTED	Access window is not supported on this MCU.

◆ R_MRAM_IdCodeSet()

fsp_err_t R_MRAM_IdCodeSet (flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode*)

Stub function Implements flash_api_t::idCodeSet.

Return values

FSP_ERR_UNSUPPORTED	ID Code is not supported on this MCU.

◆ R_MRAM_Reset()

fsp_err_t R_MRAM_Reset (flash_ctrl_t *const *p_api_ctrl*)

Reset the MRAM peripheral. Implements flash_api_t::reset.

No attempt is made to check if the MRAM is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

Return values

FSP_ERR_UNSUPPORTED	Reset is not supported on this MCU.

◆ R_MRAM_UpdateFlashClockFreq()

fsp_err_t R_MRAM_UpdateFlashClockFreq (flash_ctrl_t *const *p_api_ctrl*)

Indicate to the MRAM module that the system clock has changed. Implements flash_api_t::updateFlashClockFreq.

This should be the case if the application has changed the system clock.

Return values

FSP_ERR_UNSUPPORTED	UpdateFlashClockFreq is not supported on this MCU.

◆ R_MRAM_StartUpAreaSelect()

fsp_err_t R_MRAM_StartUpAreaSelect (flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary*)

Selects which block, Default (Block 0) or Alternate (Block 1), is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Implements flash_api_t::startupAreaSelect.

Return values

FSP_ERR_UNSUPPORTED	StartUpAreaSelect is not supported on this MCU.

◆ R_MRAM_CallbackSet()

fsp_err_t R_MRAM_CallbackSet (flash_ctrl_t *const *p_api_ctrl*, void(*)(flash_callback_args_t *) *p_callback*, void *const *p_context*, flash_callback_args_t *const *p_callback_memory*)

Stub function Implements flash_api_t::callbackSet.

Return values

FSP_ERR_UNSUPPORTED	Interrupt driven operation is not supported on this MCU.

◆ R_MRAM_BankSwap()

fsp_err_t R_MRAM_BankSwap (flash_ctrl_t *const *p_api_ctrl*)

Stub function flash_api_t::bankSwap.

Return values

FSP_ERR_UNSUPPORTED	Bank Swap is not supported on this MCU.

◆ R_MRAM_InfoGet()

fsp_err_t R_MRAM_InfoGet (flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info*)

Returns the information about the MRAM regions. Implements flash_api_t::infoGet.

Return values

FSP_SUCCESS	Successful retrieved the request information.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.

◆ R_MRAM_AntiRollbackCounterIncrement()

fsp_err_t R_MRAM_AntiRollbackCounterIncrement (flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter*)

Increments the selected anti-rollback counter.

Note

The counter is read internally before increment.

Implements flash_api_t::antiRollbackCounterIncrement

Return values

FSP_SUCCESS	Counter incremented successfully
FSP_ERR_ASSERTION	NULL provided for p_api_ctrl
FSP_ERR_NOT_OPEN	The MRAM module is not open.
FSP_ERR_IN_USE	The MRAM peripheral is busy with a prior on-going transaction.
FSP_ERR_NOT_ENABLED	The specified counter has not been configured (configuration is only required for ARC_NSEC)
FSP_ERR_OVERFLOW	The counter cannot be incremented because it is already at its max value
FSP_ERR_TIMEOUT	Timed out waiting for the MACI to become ready.
FSP_ERR_CMD_LOCKED	MRAM Application Command Interface is in locked state
FSP_ERR_HUK_ZEROIZATION	HUK ZEROIZATION command is executing
FSP_ERR_UNSUPPORTED	Selected anti-rollback counter is not supported on this MCU

◆ R_MRAM_AntiRollbackCounterRefresh()

fsp_err_t R_MRAM_AntiRollbackCounterRefresh (flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter*)

Stub function Implements flash_api_t::antiRollbackCounterRefresh

Return values

FSP_ERR_UNSUPPORTED	Anti-Rollback counter refresh is not supported on this MCU

◆ R_MRAM_AntiRollbackCounterRead()

fsp_err_t R_MRAM_AntiRollbackCounterRead (flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter*, uint32_t *const *p_count*)

Reads the selected anti-rollback counter and returns the number of counter bits set.

Implements flash_api_t::antiRollbackCounterRead

Return values

FSP_SUCCESS	Counter read successfully into p_count
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_ASSERTION	NULL provided for p_api_ctrl or p_count
FSP_ERR_NOT_ENABLED	The specified counter has not been configured (configuration is only required for ARC_NSEC)
FSP_ERR_TIMEOUT	Timed out waiting for the MACI to become ready.
FSP_ERR_CMD_LOCKED	MRAM Application Command Interface is in locked state
FSP_ERR_IN_USE	The MRAM peripheral is busy with a prior on-going transaction.
FSP_ERR_UNSUPPORTED	The Anti Rollback Counter requested is unsupported on this MCU.

◆ R_MRAM_UserLockableAreaWrite()

fsp_err_t R_MRAM_UserLockableAreaWrite (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *mram_address*, uint32_t const *num_bytes*)

Stub function Implements flash_api_t::userLockableAreaWrite

Return values

FSP_ERR_UNSUPPORTED	User Lockable Area is not supported on this MCU

5.2.17.2 SD/MMC (r_sdhi)

Modules » Storage

Functions

fsp_err_t	R_SDHI_Open (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_cfg_t const *const p_cfg)
fsp_err_t	R_SDHI_MediaInit (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_device_t *const p_device)
fsp_err_t	R_SDHI_Read (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_Write (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_ReadIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
fsp_err_t	R_SDHI_WriteIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
fsp_err_t	R_SDHI_ReadIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t	R_SDHI_WriteIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t	R_SDHI_IoIntEnable (sdmmc_ctrl_t *const p_api_ctrl, bool enable)

fsp_err_t	R_SDHI_StatusGet (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_status_t *const p_status)
fsp_err_t	R_SDHI_Erase (sdmmc_ctrl_t *const p_api_ctrl, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_CallbackSet (sdmmc_ctrl_t *const p_api_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void *const p_context, sdmmc_callback_args_t *const p_callback_memory)
fsp_err_t	R_SDHI_Close (sdmmc_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the SD/MMC Host Interface (SDHI) peripheral on RX MCUs. This module implements the SD/MMC Interface.

Overview

Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity) and eMMC (embedded Multi Media Card)
    - Supports reading, writing and erasing SD memory devices
    - Supports 1, 4 or 8-bit data bus (8-bit bus is supported for eMMC only)
    - Supports detection of device write protection (SD cards only)
    - Supports high speed mode
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device
- Supports hardware acceleration using DMAC or DTC
- Supports callback notification when an operation completes or an error occurs

Configuration

Build Time Configurations for r_sdhi

The following build time configurations are defined in fsp_cfg/r_sdhi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Unaligned Access Support	• Disabled • Enabled (DMAC Support) • Enabled (DTC Support)	Disabled	If enabled, code for supporting buffers that are not aligned on a 4-byte boundary is included in the build. Only disable this if all buffers passed to the driver are 4-byte

			aligned.
SD Support	• Disabled • Enabled	Enabled	If selected code for SD card support is included in the build.
eMMC Support	• Disabled • Enabled	Disabled	If selected code for eMMC device support is included in the build.

Configurations for Storage > SD/MMC (r_sdhi)

This module can be added to the Stacks tab via New Stack > Storage > SD/MMC (r_sdhi).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_sdmmc0	Module name.
Channel	Value must be a non-negative integer	0	Select the channel.
Bus Width	MCU Specific Options		Select the bus width.
Card Detection	• Not Used • CD Pin	CD Pin	Select the card detection method.
Write Protection	• Not Used • WP Pin	WP Pin	Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Access Interrupt Priority	MCU Specific Options		Select the access interrupt priority.
Card Interrupt Priority	MCU Specific Options		Select the card interrupt priority.
SD Buffer Access Interrupt Priority	MCU Specific Options		Select the SD buffer access interrupt priority.

Interrupt Configurations:

The following interrupts are required to use the r_sdhi module:

Using SD/MMC with DTC:

- Access Interrupt
- DTC Interrupt

Using SD/MMC with DMAC:

- Access Interrupt
- DMAC Interrupt (in DMAC instance)

The Card interrupt is optional and only available on MCU packages that have the SDnCD pin (n = channel number).

Clock Configuration

The SDMMC MCU peripheral (SDHI) uses the PCLKB for its clock source. The SDMMC driver selects the optimal built-in divider based on the PCLKB frequency and the maximum clock rate allowed by the device obtained at media initialization.

Pin Configuration

The SDMMC driver supports the following pins (n = channel number):

- SDnCLK
- SDnCMD
- SDnDAT0
- SDnDAT1
- SDnDAT2
- SDnDAT3
- SDnDAT4 (not available on all MCUs)
- SDnDAT5 (not available on all MCUs)
- SDnDAT6 (not available on all MCUs)
- SDnDAT7 (not available on all MCUs)
- SDnCD (not available on all MCUs)
- SDnWP

The drive capacity for each pin should be set to "Medium" or "High" for most hardware designs. This can be configured in the **Pins** tab of the RX Configuration editor by selecting the pin under Pin Selection -> Ports.

Usage Notes

Card Detection

When Card Detection is configured to "CD Pin" in the RX Configuration editor, interrupt flags are cleared and card detection is enabled during R_SDHI_Open().

R_SDHI_StatusGet() can be called to retrieve the current status of the card (including whether a card is present). If the Card Interrupt Priority is enabled, a callback is called when a card is inserted or removed.

If a card is removed and reinserted, R_SDHI_MediaInit() must be called before reading from the card or writing to the card.

Note

R_SDHI_StatusGet() should be used to initially determine the card state after opening the interface.

DMA Request Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the DMA Request interrupt. This blocks all other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

Timing Notes for R_SDHI_MediaInit

The R_SDHI_MediaInit() API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1 bit and a bus speed of 400 kHz or less.

Limitations

Developers should be aware of the following limitations when using the SDHI:

Blocking Calls

The following functions block execution until the response is received for at least one command:

- R_SDHI_MediaInit
- R_SDHI_Erase

Once the function returns the status of the operation can be determined via R_SDHI_StatusGet or through receipt of a callback.

Note

Due to the variability in clocking configurations it is recommended to determine blocking delays experimentally on the target system.

Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the read(), write(), readIoExt(), and writeIoExt() APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdhi driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes an extra CPU interrupt is required for each block transferred and a software copy is used to move data to the destination buffer.

Note Due to the hardware behavior in ICU module, the data transfers by using DTC have to be 4-byte aligned address and 4-byte multiple size.

Examples

Basic Example

This is a basic example of minimal use of the r_sdhi in an application.

```
uint8_t g_dest[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);

uint8_t g_src[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);

uint32_t g_transfer_complete = 0;

void r_sdhi_basic_example (void)
```

```
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < SDHI_MAX_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the SDHI driver. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
    assert(FSP_SUCCESS == err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
    err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
    err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
}
/* The callback is called when a transfer completes. */
void r_sdhi_example_callback (sdmmc_callback_args_t * p_args)
{
```

```
 if (SDMMC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = 1;
    }
}
```

Card Detection Example

This is an example of using SDHI when the card may not be plugged in. The card detection interrupt must be enabled to use this example.

```
bool g_card_inserted = false;
void r_sdhi_card_detect_example (void)
{
 /* Open the SDHI driver. This enables the card detection interrupt. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Check if card is inserted. */
 sdmmc_status_t status;
    err = R_SDHI_StatusGet(&g_sdmmc0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 if (!status.card_inserted)
    {
 while (!g_card_inserted)
      {
 /* Wait for a card insertion interrupt. */
      }
    }
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card after card insertion is detected. */
```

```
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
}
/* The callback is called when a card detection event occurs if the card detection
interrupt is enabled. */
void r_sdhi_card_detect_example_callback (sdmmc_callback_args_t * p_args)
{
 if (SDMMC_EVENT_CARD_INSERTED == p_args->event)
    {
       g_card_inserted = true;
    }
 if (SDMMC_EVENT_CARD_REMOVED == p_args->event)
    {
       g_card_inserted = false;
    }
}
```

Function Documentation

◆ R_SDHI_Open()

fsp_err_t R_SDHI_Open (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_cfg_t const *const *p_cfg*)

Opens the driver. Resets SDHI, and enables card detection interrupts if card detection is enabled. R_SDHI_MediaInit must be called after this function before any other functions can be used.

Implements sdmmc_api_t::open().

Example:

```
/* Open the SDHI driver. */

fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
```

Return values

FSP_SUCCESS	Module is now open.
FSP_ERR_ASSERTION	Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices.
FSP_ERR_ALREADY_OPEN	Driver has already been opened with this instance of the control structure.
FSP_ERR_IRQ_BSP_DISABLED	Access interrupt is not enabled.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this MCU.

◆ R_SDHI_MediaInit()

fsp_err_t R_SDHI_MediaInit (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_device_t *const *p_device*)

Initializes the SDHI hardware and completes identification and configuration for the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements sdmmc_api_t::mediaInit().

Example:

```
 /* A device shall be ready to accept the first command within 1ms from detecting VDD min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version 6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
```

Return values

FSP_SUCCESS	Module is now ready for read/write access.
FSP_ERR_ASSERTION	Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_INIT_FAILED	Device was not identified as an SD card or eMMC device.
FSP_ERR_RESPONSE	Device responded with an error.
FSP_ERR_TIMEOUT	Device did not respond.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low (device is busy) or another operation is ongoing.

◆ R_SDHI_Read()

fsp_err_t R_SDHI_Read (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements sdmmc_api_t::read().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Example:

```
err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
```

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.

◆ R_SDHI_Write()

fsp_err_t R_SDHI_Write (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements sdmmc_api_t::write().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written and the device is no longer holding DAT0 low to indicate it is busy.

Example:

```
err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
```

Return values

FSP_SUCCESS	Card write finished successfully.
FSP_ERR_ASSERTION	Handle or Source address is NULL.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.
FSP_ERR_CARD_WRITE_PROTECTED	SD card is Write Protected.
FSP_ERR_WRITE_FAILED	Write operation failed.

◆ R_SDHI_ReadIo()

fsp_err_t R_SDHI_ReadIo (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*)

The Read function reads a one byte register from an SDIO card. Implements sdmmc_api_t::readIo().

Note

This function is not supported.

Return values

FSP_ERR_UNSUPPORTED	SDIO is not supported.

◆ R_SDHI_WriteIo()

fsp_err_t R_SDHI_WriteIo (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*, sdmmc_io_write_mode_t const *read_after_write*)

Writes a one byte register to an SDIO card. Implements sdmmc_api_t::writeIo().

Note

This function is not supported.

Return values

FSP_ERR_UNSUPPORTED	SDIO is not supported.

◆ R_SDHI_ReadIoExt()

fsp_err_t R_SDHI_ReadIoExt (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *function*, uint32_t const *address*, uint32_t *const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode*)

Reads data from an SDIO card function. Implements sdmmc_api_t::readIoExt().

Note

This function is not supported.

Return values

FSP_ERR_UNSUPPORTED	SDIO is not supported.

◆ R_SDHI_WriteIoExt()

fsp_err_t R_SDHI_WriteIoExt (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *function*, uint32_t const *address*, uint32_t const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode*)

Writes data to an SDIO card function. Implements sdmmc_api_t::writeIoExt().

Note

This function is not supported.

Return values

FSP_ERR_UNSUPPORTED	SDIO is not supported.

◆ R_SDHI_IoIntEnable()

fsp_err_t R_SDHI_IoIntEnable (sdmmc_ctrl_t *const *p_api_ctrl*, bool *enable*)

Enables or disables the SDIO Interrupt. Implements sdmmc_api_t::ioIntEnable().

Note

This function is not supported.

Return values

FSP_ERR_UNSUPPORTED	SDIO is not supported.

◆ R_SDHI_StatusGet()

fsp_err_t R_SDHI_StatusGet (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_status_t *const *p_status*)

Provides driver status. Implements sdmmc_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Driver has not been initialized.

◆ R_SDHI_Erase()

fsp_err_t R_SDHI_Erase (sdmmc_ctrl_t *const *p_api_ctrl*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Erases sectors of an SD card or eMMC device. Implements sdmmc_api_t::erase().

This function blocks until the erase command is sent. Poll the status to determine when erase is complete.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	A required pointer is NULL or an argument is invalid.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_CARD_WRITE_PROTECTED	SD card is Write Protected.
FSP_ERR_RESPONSE	Device responded with an error.
FSP_ERR_TIMEOUT	Device did not respond.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low (device is busy) or another operation is ongoing.

◆ R_SDHI_CallbackSet()

fsp_err_t R_SDHI_CallbackSet (sdmmc_ctrl_t *const *p_api_ctrl*, void(*)(sdmmc_callback_args_t *) *p_callback*, void *const *p_context*, sdmmc_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements sdmmc_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_SDHI_Close()

fsp_err_t R_SDHI_Close (sdmmc_ctrl_t *const *p_api_ctrl*)
Closes an open SD/MMC device. Implements sdmmc_api_t::close().

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver has not been initialized.

5.2.17.3 XSPI (r_xspi_controller)

Modules » Storage

Functions

fsp_err_t	R_XSPI_CONTROLLER_Open (spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg)
fsp_err_t	R_XSPI_CONTROLLER_Close (spi_flash_ctrl_t *const p_ctrl)
fsp_err_t	R_XSPI_CONTROLLER_DirectWrite (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)
fsp_err_t	R_XSPI_CONTROLLER_DirectRead (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_XSPI_CONTROLLER_DirectTransfer (spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)
fsp_err_t	R_XSPI_CONTROLLER_SpiProtocolSet (spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol)
fsp_err_t	R_XSPI_CONTROLLER_XipEnter (spi_flash_ctrl_t *const p_ctrl)
fsp_err_t	R_XSPI_CONTROLLER_XipExit (spi_flash_ctrl_t *const p_ctrl)
fsp_err_t	R_XSPI_CONTROLLER_Write (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)
fsp_err_t	R_XSPI_CONTROLLER_Erase (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)

fsp_err_t	R_XSPI_CONTROLLER_StatusGet (spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status)
fsp_err_t	R_XSPI_CONTROLLER_BankSet (spi_flash_ctrl_t *const _ctrl, uint32_t bank)
fsp_err_t	R_XSPI_CONTROLLER_AutoCalibrate (spi_flash_ctrl_t *const p_ctrl)
fsp_err_t	R_XSPI_CONTROLLER_DOTF_Configure (spi_flash_ctrl_t *const p_ctrl, xspi_controller_dotf_cfg_t *const p_dotf_cfg)
fsp_err_t	R_XSPI_CONTROLLER_RowLoad (spi_flash_ctrl_t *const p_ctrl, uint32_t row_index)
fsp_err_t	R_XSPI_CONTROLLER_RowStore (spi_flash_ctrl_t *const p_ctrl, uint32_t row_index)

Detailed Description

Driver for the XSPI_CONTROLLER peripheral on RX MCUs. This module implements the SPI Flash Interface.

Overview

The XSPI_CONTROLLER peripheral supports xSPI compatible external memory devices, and it interfaces with these devices to perform data I/O Operations. The XSPI_CONTROLLER peripheral does not support addressable devices, so all connected memory devices must be connected to an individual chip-select pin. Please note that this document will reference the xSPI protocol to which XSPI is a subset. The XSPI_CONTROLLER peripheral is compatible with a variety of xSPI protocol operating modes.

Features

The XSPI_CONTROLLER driver has the following key features to support the **xSPI** device:

- Perform data I/O Operation
- Direct memory-mapped access to the xSPI device memory up to 256 MB.
- Can be configured with xSPI devices on either of the 2 channels
- Programming the xSPI device using combination write (up to 64 bytes)
- Erasing the xSPI device
- Sending device specific commands and reading back responses of up to 8 bytes
- 1- and 2-byte addressing intended for row-column memories
- 3 byte addressing
- 4 byte addressing
- Auto-calibration
- Decryption-on-the-fly
- Support for standard SPI frames, xSPI Protocol 1.0, xSPI Protocol 2.0, and xSPI Protocol 2.0 Extended frames.

Additional build-time features:

- Optional (build-time) DMAC support for data transmission when used with XSPI_CONTROLLER.
- Optional (build-time) XiP support for entering/exiting XiP mode of the target device.
- Optional (build-time) Data-strobe (DS) auto-calibration support for target devices using the DS signal.
- Optional (build-time) Decryption on the fly (DOTF)
- Optional (build-time) Support for memory which use row-column addressing.
- Optional (build-time) BSP initialization of SiP Flash.

Note

For XSPI_CONTROLLER, use of DMAC for data transmission is strongly recommended. Without the use of DMAC, due to the high-speed hardware design of the XSPI peripheral, data transmission can be sensitive to timing variance, which could cause software-based memory-mapped operations to fail unexpectedly.

Configuration

XSPI_CONTROLLER Flash:

Build Time Configurations for r_xspi_controller

The following build time configurations are defined in fsp_cfg/r_xspi_controller_cfg.h:

Configuration	Options	Default	Description
Memory-mapping Support > Prefetch Function	• Enable • Disable	Enable	Enable prefetch function on memory-mapped reads.
Memory-mapping Support > Combination Function	Refer to the RX Configuration tool for available options.	64 Bytes	Enable combination function on memory-mapped writes.
Memory-mapping Support > XiP Support	• Enable • Disable	Disable	Enable the use of XiP enter and exit codes.
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DMAC Support	• Enable • Disable	Disable	Enable DMAC support for the XSPI module.
Autocalibration Support	• Enable • Disable	Disable	Enable DS autocalibration for dual-data-rate modes.
DOTF Support	• Enable (Protected Mode) • Enable (Compatibility Mode) • Disable	Disable	Enable DOTF support for the XSPI module.
Row Addressing Support	• Enable • Disable	Disable	Enable support for xSPI devices which utilize row-column addressing

			of the memory array.

Configurations for Storage > XSPI (r_xspi_controller)

This module can be added to the Stacks tab via New Stack > Storage > XSPI (r_xspi_controller).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_xspi0	Module name.
General > Unit	MCU Specific Options		Specify the XSPI peripheral to use.
General > Chip Select	• CS0 • CS1	CS0	Specify the XSPI chip select line to use.
General > Memory Type	• XSPI with Flash • XSPI with RAM	XSPI with Flash	Specify the XSPI memory type to use.
General > Write Status Bit	• b0 • b1 • b2 • b3 • b4 • b5 • b6 • b7	b0	Which bit contains the write in progress status returned from the Write Status Command.
General > Write Enable Bit	• b0 • b1 • b2 • b3 • b4 • b5 • b6 • b7	b1	Which bit contains the write enable status returned from the Write Enable Command.
General > DS Auto-calibration Pattern Address	Must be a valid pointer address	0	Address to the auto-calibration pattern in the target flash memory's address space.
Command Sets > Custom Table > Table Pointer	Must be a vaild C symbol		Specify the custom command set table (xspi_controller_command_set_t[]) to use. If provided, all other command set settings are ignored.
Command Sets > Custom Table > Table Length	Must be an integer greater-than or equal to 0.	0	Length of the custom command set table.
Command Sets >	Must be a valid C		Specify the primary

Custom Table > Erase Command Table	symbol name.		erase command table (spi_flash_erase_command_t[]). Required for some stacks which can accept r_spi_flash_api. This table should contain the erase sizes used by the application during normal operation.
Command Sets > Custom Table > Erase Command Table Length	Must be an integer greater-than or equal to 0.	0	Length of the erase command table.
Command Sets > Erase Sizes > Sector Erase	Erase size must be an integer greater-than or equal-to zero and less than 4,294,967,296 (0xFFFF_FFFF).	4096	Size of memory region erased by Sector Erase.
Command Sets > Erase Sizes > Block Erase	Erase size must be an integer greater-than or equal-to zero and less than 4,294,967,296 (0xFFFF_FFFF).	262144	Size of memory region erased by Block Erase.
Command Sets > Initial Mode > Read > Command Code	Command codes must be positive and fit in the selected command byte length.	0x13	Read command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Read > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Read commands. Set to 0 to disable dummy cycles.
Command Sets > Initial Mode > Program > Command Code	Command codes must be positive and fit in the selected command byte length.	0x12	Program command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Program > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Program commands. Set to 0 to disable dummy cycles.
Command Sets > Initial Mode > Row Load > Command Code	Command codes must be positive and fit in the selected command byte length.	0x0	Row Load command code. Set to 0x00 to disable.

Command Sets > Initial Mode > Row Load > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Row Load commands. Set to 0 to disable dummy cycles.
Command Sets > Initial Mode > Row Store > Command Code	Command codes must be positive and fit in the selected command byte length.	0x0	Row Store command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Row Store > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Row Store commands. Set to 0 to disable dummy cycles.
Command Sets > Initial Mode > Write Enable > Command Code	Command codes must be positive and fit in the selected command byte length.	0x06	Write Enable command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Status Read > Command Code	Command codes must be positive and fit in the selected command byte length.	0x05	Status Read command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Status Read > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Status Read commands. Set to 0 to disable dummy cycles.
Command Sets > Initial Mode > Sector Erase > Command Code	Command codes must be positive and fit in the selected command byte length.	0x21	Sector Erase command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Block Erase > Command Code	Command codes must be positive and fit in the selected command byte length.	0xDC	Block Erase command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Chip Erase > Command Code	Command codes must be positive and fit in the selected command byte length.	0x60	Chip Erase command code. Set to 0x00 to disable.
Command Sets > Initial Mode > Protocol Mode	• SPI (1S-1S-1S) • DSPI (1S-2S-2S) • DSPI (2S-2S-2S) • QSPI (1S-4S-4S) • QSPI (4S-4S-4S) • QSPI (4S-4D-4D)	SPI (1S-1S-1S)	Signal protocol mode to use for this command set.

	• Dual data rate OPI (8D-8D-8D)		
Command Sets > Initial Mode > Frame Format	• Standard • xSPI Profile 1.0 • xSPI Profile 2.0 • xSPI Profile 2.0 Extended	Standard	Frame format to use with this command set.
Command Sets > Initial Mode > Address Mode	• Normal Address • Array Address	Normal Address	Address mode to use for external device. Note that not supported by flash devices.
Command Sets > Initial Mode > Latency Mode	• Fixed • Variable (DQS)	Fixed	Latency mode to use with this command set.
Command Sets > Initial Mode > Address Length	• 1 byte • 2 bytes • 3 bytes • 4 bytes	3 bytes	Number of bytes used to address data in a memory page or row.
Command Sets > Initial Mode > Address MSB Mask	Address mask must be between 0x00 and 0xFF.	0xF0	Mask of bits to zero for the most-significant byte of the address stage during memory-mapped operations.
Command Sets > Initial Mode > Command Code Length	• 1 byte • 2 bytes	1 byte	Number of bytes used for command codes.
Command Sets > Initial Mode > Status Register	• No address • 1 byte • 2 bytes • 3 bytes • 4 bytes	No address	Number of bytes used for addressing the status register.
Command Sets > Initial Mode > Status Register Address	Status register address must be between 0x00 and 0xFFFF_FFFF, inclusive.	0x0	Address of the status register that reports Write-in-Progress and Write-Enable flags.
Command Sets > High-speed Mode > Read > Command Code	Command codes must be positive and fit in the selected command byte length.	0xEEEE	Read command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Read > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	20	Dummy cycles to use between the address and data phase for Read commands. Set to 0 to disable dummy cycles.
Command Sets > High-speed Mode > Program > Command Code	Command codes must be positive and fit in the selected command	0x1212	Program command code. Set to 0x00 to disable.

	byte length.		
Command Sets > High-speed Mode > Program > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Program commands. Set to 0 to disable dummy cycles.
Command Sets > High-speed Mode > Row Load > Command Code	Command codes must be positive and fit in the selected command byte length.	0x0	Row Load command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Row Load > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Row Load commands. Set to 0 to disable dummy cycles.
Command Sets > High-speed Mode > Row Store > Command Code	Command codes must be positive and fit in the selected command byte length.	0x0	Row Store command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Row Store > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	Dummy cycles to use between the address and data phase for Row Store commands. Set to 0 to disable dummy cycles.
Command Sets > High-speed Mode > Write Enable > Command Code	Command codes must be positive and fit in the selected command byte length.	0x0606	Write Enable command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Status Read > Command Code	Command codes must be positive and fit in the selected command byte length.	0x0505	Status Read command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Status Read > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	3	Dummy cycles to use between the address and data phase for Status Read commands. Set to 0 to disable dummy cycles.
Command Sets > High-speed Mode > Sector Erase > Command Code	Command codes must be positive and fit in the selected command byte length.	0x2121	Sector Erase command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Block Erase > Command Code	Command codes must be positive and fit in the selected command byte length.	0xDCDC	Block Erase command code. Set to 0x00 to disable.

Command Sets > High-speed Mode > Chip Erase > Command Code	Command codes must be positive and fit in the selected command byte length.	0x6060	Chip Erase command code. Set to 0x00 to disable.
Command Sets > High-speed Mode > Protocol Mode	• SPI (1S-1S-1S) • DSPI (1S-2S-2S) • DSPI (2S-2S-2S) • QSPI (1S-4S-4S) • QSPI (4S-4S-4S) • QSPI (4S-4D-4D) • Dual data rate OPI (8D-8D-8D)	Dual data rate OPI (8D-8D-8D)	Signal protocol mode to use for this command set.
Command Sets > High-speed Mode > Frame Format	• Standard • xSPI Profile 1.0 • xSPI Profile 2.0 • xSPI Profile 2.0 Extended	xSPI Profile 1.0	Frame format to use with this command set.
Command Sets > High-speed Mode > Address Mode	• Normal Address • Array Address	Normal Address	Address mode to use for external device. Note that not supported by flash devices.
Command Sets > High-speed Mode > Latency Mode	• Fixed • Variable (DQS)	Fixed	Latency mode to use with this command set.
Command Sets > High-speed Mode > Address Length	• 1 byte • 2 bytes • 3 bytes • 4 bytes	4 bytes	Number of bytes used to address data in a memory page or row.
Command Sets > High-speed Mode > Address MSB Mask	Address mask must be between 0x00 and 0xFF.	0xF0	Mask of bits to zero for the most-significant byte of the address stage during memory-mapped operations.
Command Sets > High-speed Mode > Command Code Length	• 1 byte • 2 bytes	2 bytes	Number of bytes used for command codes.
Command Sets > High-speed Mode > Status Register	• No address • 1 byte • 2 bytes • 3 bytes • 4 bytes	4 bytes	Number of bytes used for addressing the status register.
Command Sets > High-speed Mode > Status Register Address	Status register address must be between 0x00 and 0xFFFF_FFFF, inclusive.	0x00	Address of the status register that reports Write-in-Progress and Write-Enable flags.
Command Sets > Write	• Enabled	Enabled	As a mask signal for

Data Mask	• Disabled		each byte in the write data. The write data mask function is only used in 8D-8D-8D protocol mode.
Timing Settings > Chip Select > Command Interval	Refer to the RX Configuration tool for available options.	2 cycles	Minimum clock cycles between two consecutive xSPI frames (CS idle time).
Timing Settings > Chip Select > Assertion Extension	• No extension • 1 cycle extension	No extension	Signal pull-up timing (CS asserting extension) setting in OCTACLK units.
Timing Settings > Chip Select > Negation Extension	• No extension • 1 cycle extension	module.driver.xspi_controller.pull_down_timing.clocks_5	Signal pull-down timing (CS negating extension) setting in OCTACLK units.
Timing Settings > SDR > Drive Timing	• Drive 1/2 cycle before CK • Drive at CK	Drive 1/2 cycle before CK	Data signal timing relative to the rising-edge of the CK signal.
Timing Settings > SDR > CK Sampling Edge	• Falling edge • Rising edge	Falling edge	Data signal timing relative to the rising-edge of the CK signal.
Timing Settings > SDR > Sampling Delay	• None • 1 cycle • 2 cycles • 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles	None	Number of cycles to delay before sampling the data signal.
Timing Settings > DDR > Sampling Window Extension	• None • 1 cycle • 2 cycles • 3 cycles • 4 cycles • 5 cycles • 6 cycles • 7 cycles	None	Number of cycles to delay before sampling the data signal.
Timing Settings > DS Latching Delay	Refer to the RX Configuration tool for available options.	Auto-calibrate	Number of delay cells for the data strobe signal. This value must be set to zero if auto-calibration is used.
Timing Settings > Output Assert Delay	• Disabled • Enabled	Disabled	Assertion timing of the output buffer control signal. Setting this bit to 1 delays the start of output from the IO pins

			and the DS pin by one cycle.
XiP Mode > XiP Enter Code	Must be an integer between 0 and 255, inclusive.	0	XiP enter code.
XiP Mode > XiP Exit Code	Must be an integer between 0 and 255, inclusive.	0	XiP exit code.
DOTF > Name	Must be a valid C symbol name.	g_xspi_dotf	DOTF configuration name.
DOTF > AES Key	Must be a valid C symbol name.	g_xspi_dotf_key	Name of the encryption key variable.
DOTF > AES IV	Must be a valid C symbol name.	g_xspi_dotf_iv	Name of the encryption IV variable.
DOTF > AES Key Length	MCU Specific Options		AES key length.
DOTF > Key Format	MCU Specific Options		Key format.
DOTF > Decryption Start Address	Must be an integer between 0x20000000 and 0x3FFFF000	0x20000000	XSPI decryption start address
DOTF > Decryption End Address	Must be an integer between 0x20000FFF and 0x3FFFFFFF	0x20001FFF	XSPI decryption end address
Row Addressing > Row Address Length	• No address • 1 byte • 2 bytes • 3 bytes • 4 bytes	No address	Number of bytes used for addressing the row index.

Note

The user is expected to modify the command definitions based on the xSPI chip and SPI communication mode. The default mode is SPI mode and default erase commands are set for DOPI mode based on Macronix MX25LW51245G.

Clock Configuration

The signals to the xSPI target device are derived from OCTACLK. OCTACLK must be set to twice the desired xSPI_CLK signal frequency. This allows data to be latched on either clock edge for Dual-Data-Rate modes (4S-4D-4D or 8D-8D-8D).

Please refer to the Hardware User Manual for the clocks which control the XSPI_CONTROLLER registers. These along with OCTACLK can be set on the **Clocks** tab of the RA Configuration editor.

Pin Configuration

The following pins are available to connect to an external XSPI device:

- XSPIn_CK: clock output (XSPICLK / 2)

- XSPIn_DS: data strobe signal
- XSPIn_CS0: device 0 select
- XSPIn_CS1: device 1 select
- XSPIn_IO0: Data 0 I/O
- XSPIn_IO1: Data 1 I/O
- XSPIn_IO2: Data 2 I/O
- XSPIn_IO3: Data 3 I/O
- XSPIn_IO4: Data 4 I/O
- XSPIn_IO5: Data 5 I/O
- XSPIn_IO6: Data 6 I/O
- XSPIn_IO7: Data 7 I/O

Note

Data pins can be configured with IOPORT_CFG_DRIVE_6 or IOPORT_CFG_DRIVE_8 according to the Electrical Characteristics chapter.
Chip Select pins should be configured with at least IOPORT_CFG_DRIVE_4.

Usage Notes

Usage Notes for xSPI support

After R_XSPI_CONTROLLER_Open() completes successfully, the xSPI device contents are mapped to the appropriate external memory regions for each XSPI_CONTROLLER unit. For example, on the RX74M, R_XSPI_CONTROLLER0 maps the CS0 device to address 0x30000000 and the CS1 device to address 0x20000000. After opening, the addresses can be accessed like on-chip flash.

Bank CS0 and CS1 support up to 256 MB of address space each depending on the platform.

Auto-calibration

If support is enabled, auto-calibration procedures are triggered automatically when the 'Data latching delay' field in the configurator properties is set to 0. The user application is responsible for setting the appropriate preamble pattern before calling R_XSPI_CONTROLLER_Open() when using a data strobe (DS) mode or changing the SPI protocol to a DS mode using the R_XSPI_CONTROLLER_SpiProtocolSet() API. The appropriate preamble pattern can be written to the desired address using the R_XSPI_CONTROLLER_Write() API while in simple SPI mode (recommended), or a non-DS mode. Ensure that the same address is passed through the configurator. If the xSPI device is already in a DS mode, the preamble pattern must be programmed using the debugger before calling R_XSPI_CONTROLLER_Open().

The preamble pattern is expected to be { 0x00, 0x00, 0xFF, 0xFF, 0x00, 0xFF, 0x00, 0x08, 0xF7, 0x00, 0x00, 0xFF, 0xF7, 0x08, 0xF7, 0x00 }.

For high frequency operation (e.g. 266 MHz), it is recommended to enable the Output Assertion Delay to improve signal stability.

Chip Select Latencies

Chip select latencies can be set through the configurator. The default settings support SPI at minimum latency. In case the driver is opened in SPI mode and will be switched to DOPI mode later using R_XSPI_CONTROLLER_SpiProtocolSet(), please select latencies required for DOPI before calling R_XSPI_CONTROLLER_Open().

Usage Notes for OctaRAM support

Auto-calibration

If support is enabled, auto-calibration procedures are triggered automatically when the 'Data latching delay' field in the configurator properties is set to 0, the same as with OctaFlash.

Unlike OctaFlash, the user does **not** need to write a preamble pattern to the device before calling R_XSPI_CONTROLLER_Open() or R_XSPI_CONTROLLER_SpiProtocolSet(). For OctaRAM devices, calibration instead uses the target device's built-in Preamble Bit Data Pattern READ command (command code F0h, address 00h), which the OctaRAM drives unconditionally regardless of calibration or timing state. This avoids the chicken-and-egg problem of needing correctly calibrated timing to write the pattern in the first place.

Make sure to set up the option of OctaRAM via R_XSPI_CONTROLLER_DirectTransfer() after calling R_XSPI_CONTROLLER_Open() with 'Data latching delay' field in the configurator properties is other than 0.

Auto-calibration may also be triggered manually at any time by calling R_XSPI_CONTROLLER_AutoCalibrate() once the device is open.

Note

> *This behavior is specific to OctaRAM devices which support the Preamble Bit Data Pattern READ command (e.g. S66/S67WVO32M8DALL/BLL, see datasheet section 5.3). Confirm the connected OctaRAM part supports this command before relying on auto-calibration.*

XiP Support

XSPI_CONTROLLER supports eXecute in Place (XiP) modes of operation. This can be used for read-only memory-mapped accesses to reduce overall read latency by skipping the command sequence in the xSPI transaction. Separate XiP enter and exit codes may be specified for either attached target device. Upon calling R_XSPI_CONTROLLER_XipEnter(), the associated memory region for the target device is switched to read-only mode and the enter code sent to the device. Calling R_XSPI_CONTROLLER_XipExit() will transmit the exit code and transition the memory region back to read-write access.

Only one flash device should be used after entering XiP mode. Once entered, XiP codes will be transmitted to all attached devices.

xSPI Commands

Command sets and erase commands may be specified individually for each supported protocol mode. By default, the configurator only supports an alternative command set for "High-Speed Mode" (DOPI 8D-8D-8D) mode. The command sets cannot be changed during run-time, but the appropriate command set will be selected when changing protocol modes. If a command set is not found, an FSP_ERR_INVALID_MODE error code will be returned.

If custom DOPI erase commands are not specified, ensure the erase commands are the appropriate 2-byte DOPI commands. The lower byte will be used for 1-byte command protocols.

If a custom command set table is provided, the primary erase command table must also be provided for backwards compatibility with some modules. The primary table contains the erase sizes being used during normal application runtime.

DOTF Support

Decryption-On-The-Fly is configurable for XSPI with OctaFlash and is disabled from the build by

default. When DOTF support is enabled, each XSPI unit uses the matching DOTF unit:

- **DOTF0** is used with **XSPI0**.
- **DOTF1** is used with **XSPI1**.

DOTF decrypts data during memory-mapped reads from the configured XSPI address range. The encrypted data must be generated for the same physical address range that will be used by the application because the DOTF counter is address dependent.

- **DOTF0** operates within the address range 0x20000000 to 0x3FFFFFFF.
- **DOTF1** operates within the address range 0x40000000 to 0x5FFFFFFF.

Using the DOTF feature requires first creating the encrypted blob on the PC and then configuring the DOTF module with the same parameters so the programmed XSPI contents can be decrypted at run time. Use the Security Key Management Tool (https://www.renesas.com/us/en/software-tool/security-key-management-tool) to create the encrypted blob.

Example: To encrypt a 4096 byte area in an input srec file from 0x20000000 to 0x20000FFF using a 128-bit AES encryption key "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF" and IV "00000000000000000000000000000000", use SKMT with the following arguments:

```
skmt.exe /encdotf /keytype "AES-128" /enckey "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
/nonce "00000000000000000000000000000000" /startaddr "20000000" /endaddr "20000FFF"
/prg "input.srec" /incplain /output "encrypted_output.srec"
```

The values for key, iv and decryption area start and end addresses that were used to create the blob using SKMT must be set in the DOTF configuration in FSP. Make sure that the Key and IV passed into DOTF configuration are 4 byte aligned. This can be done using a compiler alignment attribute as shown below:

```
uint8_t aes_key[] BSP_ALIGN_VARIABLE(4) =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f
};
uint8_t aes_iv[] BSP_ALIGN_VARIABLE(4) =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f
};
```

See the DOTF example for how to attach a DOTF configuration to an XSPI instance before calling R_XSPI_CONTROLLER_Open() and how to reconfigure DOTF with R_XSPI_CONTROLLER_DOTF_Configure() before memory-mapped reads.

Supported Key Formats

- When the Compatibility Mode stack is added for the crypto engine, the user can provide either a wrapped or plaintext AES key to the DOTF module.
- When the Protected Mode stack is used, only wrapped keys are supported.

When configuring the DOTF options, make sure that the key format is specified.

Note

For a detailed description of the Crypto Engine operating modes, refer to Application Note R11AN0498.

SiP Flash

SiP or System-in-Package devices, can contain an internally bonded XSPI flash memory in the package. The BSP supports startup initialization of SiP flash memories using the R_OSPI_B driver. To enable startup initialization, please select an appropriate stack under Storage > SiP Flash and enable the following properties in the BSP section of the configurator: OSPI > SiP Flash Support and RA Common > Early BSP Initialization. Both of these BSP properties must be enabled for successful SiP flash startup and initialization of other memory regions (e.g. RAM) which load data from the SiP flash.

Limitations

Developers should be aware of the following limitations when using the XSPI_CONTROLLER driver:

XSPI_CONTROLLER

- The R_XSPI_CONTROLLER driver expects memory devices to have uniform sector sizes. If a design requires using a memory with non-uniform sectors, the end-user application must appropriately manage sectors.
- Prefetch and combination support for memory-mapped access is applied globally to all slave devices per XSPI_CONTROLLER unit.
- Combination writes are limited to a maximum of 64 bytes. The user should verify the write has completed before attempting to write more bytes.
- Writing to the memory-mapped regions with the CPU is restricted to 64-bit accesses with 8-byte destination alignments. This restriction is not applicable to other bus masters (e.g. DTC or DMAC).
- Take care to restrict concurrent accesses of the XSPI memory area. Collisions on the bus can occur if other bus masters attempt to write to the XSPI memory area while another master is reading the XSPI memory area.
- When using 8D-8D-8D mode, care should be taken to access on even-aligned addresses. Problems may occur if odd address alignment is used. This restriction applies to all bus masters using XSPI_CONTROLLER.
- DOTF: AES-192 is not supported when the Protected Mode stack is used.
- When using the IAR compiler, XSPI data must be const qualified to be downloaded by the debugger.
- Memories which use row-column addressing must be managed by the end-user application to ensure the appropriate memory row is loaded and stored.
- When using SiP Flash with encrypted data, the encryption key and IV must be const qualified if BSP startup initialization is used. The SiP flash is initialized before the C-runtime

has been initialized and before any RAM variables have been loaded from their respective non-volatile memories.
- When using SiP Flash, secure partitions must be 32K aligned or programming and access may fail.

Examples

XSPI with OctaFlash:

The following definitions are used across various example snippets below.

```
#define XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_SPI_MODE (0x06U)
#define XSPI_CONTROLLER_COMMAND_WRITE_DISABLE_SPI_MODE (0x04U)
#define XSPI_CONTROLLER_COMMAND_READ_STATUS_SPI_MODE (0x05U)
#define XSPI_CONTROLLER_COMMAND_READ_REGISTER_SPI_MODE (0x65U)
#define XSPI_CONTROLLER_COMMAND_READ_QSPI_MODE (0x0BU)
#define XSPI_CONTROLLER_COMMAND_PROGRAM_QSPI_MODE (0x12U)
#define XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_QSPI_MODE (0x06U)
#define XSPI_CONTROLLER_COMMAND_STATUS_QSPI_MODE (0x05U)
#define XSPI_CONTROLLER_COMMAND_BLOCK_ERASE_DOPI_MODE (0x21DEU)
#define XSPI_CONTROLLER_COMMAND_SECTOR_ERASE_DOPI_MODE (0xDC23U)
#define XSPI_CONTROLLER_COMMAND_CHIP_ERASE_DOPI_MODE (0x609FU)
#define XSPI_CONTROLLER_COMMAND_READ_DOPI_MODE (0xEE11U)
#define XSPI_CONTROLLER_COMMAND_PROGRAM_DOPI_MODE (0x12EDU)
#define XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_DOPI_MODE (0x06F9U)
#define XSPI_CONTROLLER_COMMAND_STATUS_DOPI_MODE (0x05FAU)
#define XSPI_CONTROLLER_WEN_BIT_MASK (0x02U)
#define XSPI_CONTROLLER_WIP_BIT_MASK (0x01U)
#define XSPI_CONTROLLER_BLOCK_ERASE_SIZE (4096U)
#define XSPI_CONTROLLER_SECTOR_ERASE_SIZE (262144U)
#define XSPI_CONTROLLER_EXAMPLE_DESTINATION_ADDRESS ((void *) 0x20000000U) /* Device
on CS1 */
```

```
static void xspi_controller_change_mode (spi_flash_protocol_t new_protocol)
{
 /* Step 1. Identify current device configuration and reconfigure as needed... */
 /* Step 2. Update clocks... */
 /* Step 3. Transition driver to the new protocol. */
```

```
 fsp_err_t err = R_XSPI_CONTROLLER_SpiProtocolSet(&g_xspi0_controller_ctrl,
new_protocol);
    assert(FSP_SUCCESS == err);
 /* Step 4. Verify mode change... */
}
```

Basic Example

This is a basic example of minimal use of the XSPI in an application with OctaFlash.

```
#define XSPI_CONTROLLER_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[XSPI_CONTROLLER_EXAMPLE_DATA_LENGTH];
/* Place data in the .xspi0_cs0 section to flash it during programming. */
const uint8_t g_src[XSPI_CONTROLLER_EXAMPLE_DATA_LENGTH]
BSP_PLACE_IN_SECTION(".xspi0_cs0") =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .xspi0_cs0_code section to flash it during programming. */
void r_xspi_controller_example_function(void) BSP_PLACE_IN_SECTION(".xspi0_cs0_code")
__attribute__((noinline));
void r_xspi_controller_example_function (void)
{
 /* Add code here. */
}
void r_xspi_controller_basic_example (void)
{
 /* Open the XSPI instance. */
 fsp_err_t err = R_XSPI_CONTROLLER_Open(&g_xspi0_controller_ctrl,
&g_xspi0_controller_cfg);
    assert(FSP_SUCCESS == err);
 /* (Optional) Change SPI to DOPI mode */
    xspi_controller_change_mode(SPI_FLASH_PROTOCOL_8D_8D_8D);
 /* After R_XSPI_CONTROLLER_Open() and any required device specific initialization,
data can be read directly from the XSPI flash. */
    memcpy(&g_dest[0], &g_src[0], XSPI_CONTROLLER_EXAMPLE_DATA_LENGTH);
 /* After R_XSPI_CONTROLLER_Open() and any required device specific initialization,
```

```
functions in the XSPI flash can be called. */
    r_xspi_controller_example_function();
}
```

Reading Status Register Example (R_XSPI_CONTROLLER_DirectTransfer)

This is an example of using R_XSPI_CONTROLLER_DirectWrite followed by R_XSPI_CONTROLLER_DirectRead to send the read status register command and read back the status register from the device.

```
#define XSPI_CONTROLLER_COMMAND_READ_STATUS_REGISTER (0x05U)
void r_xspi_controller_direct_example (void)
{
 spi_flash_direct_transfer_t xspi_controller_test_direct_transfer =
    {
       .command       = XSPI_CONTROLLER_COMMAND_READ_STATUS_SPI_MODE,
       .address       = 0U,
       .data_u.data    = 0U,
       .command_length = 1U,
       .address_length = 0U,
       .data_length    = 0U,
       .dummy_cycles   = 0U
    };
 /* Open the XSPI instance. */
 fsp_err_t err = R_XSPI_CONTROLLER_Open(&g_xspi0_controller_ctrl,
&g_xspi0_controller_cfg);
    assert(FSP_SUCCESS == err);
 /* Write Enable */
    err = R_XSPI_CONTROLLER_DirectTransfer(&g_xspi0_controller_ctrl,
                                           &xspi_controller_test_direct_transfer,
                                           SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Read Status Register */
    xspi_controller_test_direct_transfer.command    =
XSPI_CONTROLLER_COMMAND_READ_STATUS_SPI_MODE;
    xspi_controller_test_direct_transfer.data_length = 1U;
```

```
    err = R_XSPI_CONTROLLER_DirectTransfer(&g_xspi0_controller_ctrl,
                                           &xspi_controller_test_direct_transfer,
                                           SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 /* Check if Write Enable is set */
 if (XSPI_CONTROLLER_WEN_BIT_MASK !=
(xspi_controller_test_direct_transfer.data_u.data & XSPI_CONTROLLER_WEN_BIT_MASK))
    {
       __BKPT(0);
    }
}
```

Auto-calibration Example (R_XSPI_CONTROLLER_DirectTransfer, R_XSPI_CONTROLLER_Write, R_XSPI_CONTROLLER_SpiProtocolSet)

This is an example of using R_XSPI_CONTROLLER_SpiProtocolSet to change the operating mode from SPI to DOPI and allow the driver to initiate auto-calibration.

```
#define XSPI_CONTROLLER_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES (16U)
#define XSPI_CONTROLLER_EXAMPLE_PREAMBLE_ADDRESS (0x90000000U) /* Device connected to
CS1 */
const uint8_t g_preamble_bytes[XSPI_CONTROLLER_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES] =
{
    0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x08, 0x00, 0x00, 0xF7, 0xFF, 0x00, 0x08,
0xF7, 0x00, 0xF7
};
static void xspi_controller_wait_until_wip (void)
{
 fsp_err_t          err = FSP_SUCCESS;
 spi_flash_status_t status;
    status.write_in_progress = true;
    uint32_t timeout = UINT32_MAX;
 while ((status.write_in_progress) && (--timeout))
    {
       err = R_XSPI_CONTROLLER_StatusGet(&g_xspi0_controller_ctrl, &status);
      assert(FSP_SUCCESS == err);
```

```
    }
 if (0 == timeout)
    {
      assert(FSP_SUCCESS == err);
    }
}
void r_xspi_controller_auto_calibrate_example (void)
{
 /* Open the XSPI instance. */
 /* Set data_latch_delay_clocks to 0x0 to enable auto-calibration */
 fsp_err_t err = R_XSPI_CONTROLLER_Open(&g_xspi0_controller_ctrl,
&g_xspi0_controller_cfg);
    assert(FSP_SUCCESS == err);
    uint8_t * preamble_pattern_addr = (uint8_t *)
XSPI_CONTROLLER_EXAMPLE_PREAMBLE_ADDRESS;
    err = R_XSPI_CONTROLLER_Write(&g_xspi0_controller_ctrl,
                                  g_preamble_bytes,
                                  preamble_pattern_addr,
                                  XSPI_CONTROLLER_EXAMPLE_PREAMBLE_ADDRESS);
    assert(FSP_SUCCESS == err);
 /* Wait until write has been completed */
    xspi_controller_wait_until_wip();
 /* Change from SPI to DOPI mode */
    xspi_controller_change_mode(SPI_FLASH_PROTOCOL_8D_8D_8D);
}
```

Octaclk Update Example (R_XSPI_CONTROLLER_SpiProtocolSet)

This is an example of using R_BSP_OctaclkUpdate to change the Octal-SPI clock frequency during run time. The OCTACLK frequency must be updated before calling the R_XSPI_CONTROLLER_SpiProtocolSet with appropriate clock source and divider settings required to be set for the new SPI protocol mode. Ensure that the clock source selected is started.

```
/*
 * Changes the OCTACLK divider; assumes PLL2 is being used as the source for OCTACLK.
 * @param divider New clock divider to switch to.
```

```
 */
static void xspi_controller_change_omclk (bsp_clocks_octaclk_div_t divider)
{
 /* Ensure clock source (PLL2 in this example) is running before changing the OCTACLK
frequency */
    bsp_octaclk_settings_t octaclk_settings;
    octaclk_settings.source_clock = BSP_CLOCKS_CLOCK_PLL2;
    octaclk_settings.divider      = divider;
    R_BSP_OctaclkUpdate(&octaclk_settings);
}
```

Using a Custom xSPI Command Set Example

This is an example of using custom command sets for 8D-8D-8D and 4S-4S-4S protocol modes.

```
/* Custom two-byte erase commands. */
spi_flash_erase_command_t g_2B_erase_commands[] =
{
    {.command = XSPI_CONTROLLER_COMMAND_BLOCK_ERASE_DOPI_MODE, .size =
XSPI_CONTROLLER_BLOCK_ERASE_SIZE },
    {.command = XSPI_CONTROLLER_COMMAND_SECTOR_ERASE_DOPI_MODE, .size =
XSPI_CONTROLLER_SECTOR_ERASE_SIZE},
    {.command = XSPI_CONTROLLER_COMMAND_CHIP_ERASE_DOPI_MODE, .size =
SPI_FLASH_ERASE_SIZE_CHIP_ERASE },
};
xspi_controller_table_t g_2B_erase_table =
{
    .p_table = g_2B_erase_commands,
    .length = (uint8_t) (sizeof(g_2B_erase_commands) / sizeof
(g_2B_erase_commands[0])),
};
/* Custom command sets. */
xspi_controller_command_set_t g_command_sets[] =
{
 /* 8D-8D-8D example with inverted lower command byte. */
```

```
    {
        .protocol             = SPI_FLASH_PROTOCOL_8D_8D_8D,
        .command_bytes       = XSPI_CONTROLLER_COMMAND_BYTES_2,
        .read_command         = XSPI_CONTROLLER_COMMAND_READ_DOPI_MODE,
        .program_command      = XSPI_CONTROLLER_COMMAND_PROGRAM_DOPI_MODE,
        .write_enable_command = XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_DOPI_MODE,
        .status_command       = XSPI_CONTROLLER_COMMAND_STATUS_DOPI_MODE,
        .read_dummy_cycles    = 20U,
        .program_dummy_cycles = 0U,
        .status_dummy_cycles = 3U,
        .p_erase_commands    = &g_2B_erase_table
    },
 /* 4S-4S-4S example with different .read_command and dummy cycles. */
    {
        .protocol             = SPI_FLASH_PROTOCOL_4S_4S_4S,
        .command_bytes       = XSPI_CONTROLLER_COMMAND_BYTES_1,
        .read_command         = XSPI_CONTROLLER_COMMAND_READ_QSPI_MODE,
        .program_command      = XSPI_CONTROLLER_COMMAND_PROGRAM_QSPI_MODE,
        .write_enable_command = XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_QSPI_MODE,
        .status_command       = XSPI_CONTROLLER_COMMAND_STATUS_QSPI_MODE,
        .read_dummy_cycles    = 10U,
        .program_dummy_cycles = 0U,
        .status_dummy_cycles = 1U,
        .p_erase_commands    = NULL, // Use the common erase definitions.
    },
};
void r_xspi_controller_command_table_example (void)
{
 /* Open the XSPI_CONTROLLER instance.
  * Specify `g_command_sets` using the configurator. */
 fsp_err_t err = R_XSPI_CONTROLLER_Open(&g_xspi0_controller_ctrl,
&g_xspi0_controller_cfg);
    assert(FSP_SUCCESS == err);
 /* Change SPI to DOPI mode */
```

```
    xspi_controller_change_mode(SPI_FLASH_PROTOCOL_8D_8D_8D);
 /* After R_XSPI_CONTROLLER_Open() and any required device specific initialization,
data can be read directly from the XSPI flash. */
    memcpy(&g_dest[0], &g_src[0], XSPI_CONTROLLER_EXAMPLE_DATA_LENGTH);
 /* After R_XSPI_CONTROLLER_Open() and any required device specific initialization,
functions in the XSPI flash can be called. */
    r_xspi_controller_example_function();
 /* Change SPI to QSPI mode */
    xspi_controller_change_mode(SPI_FLASH_PROTOCOL_4S_4S_4S);
 /* Read data from an external device, this time in QSPI mode. */
    memcpy(&g_dest[0], &g_src[0], XSPI_CONTROLLER_EXAMPLE_DATA_LENGTH);
 /* Run the example function again in QSPI mode. */
    r_xspi_controller_example_function();
}
```

DOTF Memory-Mapped Read Example (R_XSPI_CONTROLLER_Open, R_XSPI_CONTROLLER_DOTF_Configure)

This example shows how to configure DOTF for memory-mapped reads. The encrypted image must be generated with the same start address, end address, key, IV, and key format shown in the DOTF configuration.

The encrypted data is represented by a normal C object placed in an XSPI section. Use the linked address of this object as the SKMT /encdotf start address, and replace the example initializer with the encrypted output generated for that same address range.

```
/* Define the DOTF data length and the section used to place the encrypted data in
the XSPI memory region. */
#define XSPI_CONTROLLER_DOTF_EXAMPLE_DATA_LENGTH (4096U)
#define XSPI_CONTROLLER_DOTF_EXAMPLE_DATA_SECTION BSP_ALIGN_VARIABLE(4096)
BSP_PLACE_IN_SECTION(".xspi0_cs0")
/* Place the encrypted data in the XSPI memory region. Generate this data with SKMT
/encdotf using the linked address
 * of g_xspi_controller_dotf_example_data as the start address. */
const uint8_t
g_xspi_controller_dotf_example_data[XSPI_CONTROLLER_DOTF_EXAMPLE_DATA_LENGTH]
    XSPI_CONTROLLER_DOTF_EXAMPLE_DATA_SECTION =
{
```

```
    0U
};
```

The DOTF configuration points to the placed data object for the conversion range. The key, IV, key type, and key format must match the SKMT command used to generate the encrypted data. Protected Mode requires a wrapped key.

```
/* Key and IV must match the values used by SKMT /encdotf and must be 4-byte aligned.
*/
static uint8_t g_xspi_controller_dotf_example_key[16] BSP_ALIGN_VARIABLE(4) =
{
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
};
static uint8_t g_xspi_controller_dotf_example_iv[16] BSP_ALIGN_VARIABLE(4) =
{
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00
};
xspi_controller_dotf_cfg_t g_xspi_controller_dotf_example_cfg =
{
    .key_type    = XSPI_CONTROLLER_DOTF_AES_KEY_TYPE_128,
 /* Protected Mode requires XSPI_CONTROLLER_DOTF_KEY_FORMAT_WRAPPED and a wrapped
key. */
    .format       = XSPI_CONTROLLER_DOTF_KEY_FORMAT_PLAINTEXT,
    .p_start_addr = (uint32_t *) &g_xspi_controller_dotf_example_data[0],
    .p_end_addr   = (uint32_t *)
&g_xspi_controller_dotf_example_data[XSPI_CONTROLLER_DOTF_EXAMPLE_DATA_LENGTH - 1U],
    .p_key       = (uint32_t *) g_xspi_controller_dotf_example_key,
    .p_iv         = (uint32_t *) g_xspi_controller_dotf_example_iv,
};
```

The application can attach the DOTF configuration to a local copy of the generated XSPI configuration before opening the driver. If the DOTF range or key is changed after opening, call R_XSPI_CONTROLLER_DOTF_Configure() before reading through the memory-mapped address.

```
void r_xspi_controller_dotf_example (void)
{
#if XSPI_CONTROLLER_CFG_DOTF_SUPPORT_ENABLE
 /* Use a local copy of the generated configuration so the DOTF configuration can be
attached before open. */
    memcpy(&g_xspi_controller_dotf_example_xspi_cfg,
          &g_xspi0_controller_cfg,
 sizeof(g_xspi_controller_dotf_example_xspi_cfg));
    memcpy(&g_xspi_controller_dotf_example_extend_cfg,
          g_xspi0_controller_cfg.p_extend,
 sizeof(g_xspi_controller_dotf_example_extend_cfg));
    g_xspi_controller_dotf_example_extend_cfg.p_dotf_cfg =
&g_xspi_controller_dotf_example_cfg;
    g_xspi_controller_dotf_example_xspi_cfg.p_extend    =
&g_xspi_controller_dotf_example_extend_cfg;
 /* Open the XSPI instance. DOTF is configured by R_XSPI_CONTROLLER_Open() when
p_dotf_cfg is not NULL. */
 fsp_err_t err = R_XSPI_CONTROLLER_Open(&g_xspi0_controller_ctrl,
&g_xspi_controller_dotf_example_xspi_cfg);
    assert(FSP_SUCCESS == err);
 /* If the DOTF range or key must be changed after open, configure DOTF before memory-
mapped reads. */
    err = R_XSPI_CONTROLLER_DOTF_Configure(&g_xspi0_controller_ctrl,
&g_xspi_controller_dotf_example_cfg);
    assert(FSP_SUCCESS == err);
 /* DOTF decrypts only memory-mapped reads from the configured range. */
    memcpy(g_xspi_controller_dotf_example_dest,
          &g_xspi_controller_dotf_example_data[0],
          XSPI_CONTROLLER_DOTF_EXAMPLE_DATA_LENGTH);
#else
    __BKPT(0);
#endif
```

```
}
```

Change Protocol Mode Example (R_XSPI_CONTROLLER_DirectTransfer, R_XSPI_CONTROLLER_SpiProtocolSet)

This is an example of using R_XSPI_CONTROLLER_DirectTransfer to change the attached flash device to a new protocol mode during run time.

```
/*
 * This example file is only intended for use with Macronix MX25LW51245GX flash
devices and is non-exhaustive.
 * Other flash devices will require modification of command codes, addresses, and/or
command sequences.
 */
#include "r_xspi_controller.h"
/*********************************************************************************
**********************************
 * Macro definitions
 *********************************************************************************
**********************************/
/* Command code for enabling write operations. */
#define XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_SPI_MODE (0x06U)
/* Command code to query the status register. */
#define XSPI_CONTROLLER_COMMAND_READ_STATUS_SPI_MODE (0x05U)
/* Command code to read a configuration register. */
#define XSPI_CONTROLLER_COMMAND_READ_REGISTER_SPI_MODE (0x65U)
/* Command code to write to a configuration register. */
#define XSPI_CONTROLLER_COMMAND_WRITE_REGISTER_SPI_MODE (0x72U)
/* Location of the write-enable bit in the status register. */
#define XSPI_CONTROLLER_WEN_BIT_MASK (0x02U)
/* Register setting for the Macronix MX25LW51245GX to enable 8D-8D-8D mode. */
#define XSPI_CONTROLLER_DOPI_REGISTER_SETTING (0x02U)
/* Register setting for the Macronix MX25LW51245GX to support faster clock rates (up
to 166 MHz DOPI). */
#define XSPI_CONTROLLER_FAST_CLOCK_REGISTER_SETTING (0x80U)
/* Address of the dummy cycle register for the Macronix MX25LW51245GX. */
```

```
#define XSPI_CONTROLLER_DC_ADDRESS (0x0300U)
/* Address of the mode register for the Macronix MX25LW51245GX. */
#define XSPI_CONTROLLER_MODE_ADDRESS (0x0000U)
/**********************************************************************************
**********************************
 * Function prototypes
 **********************************************************************************
**********************************/
void r_xspi_controller_example_spi_to_dopi(xspi_controller_instance_ctrl_t * const
p_ctrl);
static void xspi_controller_write_enable_and_verify(xspi_controller_instance_ctrl_t *
const p_ctrl, bool is_8d);
static void xspi_controller_change_omclk(bsp_clocks_octaclk_div_t divider);
/*
 * Enables write operations on the target flash and verifies the change by reading
the status register.
 * @param [in] p_ctrl Pointer to the XSPI_CONTROLLER control struct.
 * @param is_8d Indicates the target is currently configured for 8D-8D-8D protocol
mode.
 */
static void xspi_controller_write_enable_and_verify (xspi_controller_instance_ctrl_t
* const p_ctrl, bool is_8d)
{
 spi_flash_direct_transfer_t xspi_controller_test_direct_transfer =
    {
       .command       = XSPI_CONTROLLER_COMMAND_WRITE_ENABLE_SPI_MODE,
       .address       = 0U,
       .data_u.data    = 0U,
       .command_length = 1U,
       .address_length = 0U,
       .data_length    = 0U,
       .dummy_cycles   = 0U
    };
 /* Write Enable */
```

```
 fsp_err_t err = R_XSPI_CONTROLLER_DirectTransfer(p_ctrl,
                                                  &xspi_controller_test_direct_transfer,
                                                  SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
    xspi_controller_test_direct_transfer.command    =
XSPI_CONTROLLER_COMMAND_READ_STATUS_SPI_MODE;
    xspi_controller_test_direct_transfer.data_length = 1U;
 /* Adjust the dummy cycles if currently operating in 8D-8D-8D mode. */
 if (is_8d)
    {
       xspi_controller_test_direct_transfer.dummy_cycles = 5U;
    }
 /* Read Status Register */
    err = R_XSPI_CONTROLLER_DirectTransfer(p_ctrl,
                                           &xspi_controller_test_direct_transfer,
                                           SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 if (XSPI_CONTROLLER_WEN_BIT_MASK !=
(xspi_controller_test_direct_transfer.data_u.data & XSPI_CONTROLLER_WEN_BIT_MASK))
    {
 /* Verification of the write enable status failed. */
       __BKPT(0);
    }
}
/*
 * Changes the OCTACLK divider; assumes PLL2 is being used as the source for OCTACLK.
 * @param divider New clock divider to switch to.
 */
static void xspi_controller_change_omclk (bsp_clocks_octaclk_div_t divider)
{
 /* Ensure clock source (PLL2 in this example) is running before changing the OCTACLK
frequency */
    bsp_octaclk_settings_t octaclk_settings;
    octaclk_settings.source_clock = BSP_CLOCKS_CLOCK_PLL2;
```

```
    octaclk_settings.divider      = divider;
    R_BSP_OctaclkUpdate(&octaclk_settings);
}
/*
 * Example function to change from 1S-1S-1S (SPI) to 8D-8D-8D (DOPI) protocol mode.
 *
 * This assumes S28HS256 is in default factory configuration:
 * - 3-byte address lengths
 * - Memory read latency of 8/20 (1S/8S) up to 166 MHz DOPI
 * - Register read latency of 0/3 up to 25 MHz DPOI
 *
 * @param [in] p_ctrl Pointer to the XSPI_CONTROLLER control struct.
 *
 * The OCTACLK source must be within range to support this example as written. Clocks
are intended to be configured
 * as follows to obtain a PLL2P frequency of 240 MHz and OCTACLK starting frequency
of 60 MHz.
 * HOCO (48 MHz) -> PLL2 -> /3 -> x60.0 {960 MHz} -> PLL2P /4 {240 MHz} -> OCTACLK ->
/4 -> {60 MHz}.
 */
void r_xspi_controller_example_spi_to_dopi (xspi_controller_instance_ctrl_t * const
p_ctrl)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Step 1: Enable write access to the registers. */
    xspi_controller_write_enable_and_verify(p_ctrl, false);
 /* Step 2: Adjust the dummy cycles for a faster clock later.
  * Changes the dummy cycles for register commands to 1 for SPI and 5 for OSPI.
  */
 spi_flash_direct_transfer_t xfer_adjust_cfr3 =
    {
       .command       = XSPI_CONTROLLER_COMMAND_WRITE_REGISTER_SPI_MODE,
       .address       = XSPI_CONTROLLER_DC_ADDRESS,
       .data_u.data    = XSPI_CONTROLLER_FAST_CLOCK_REGISTER_SETTING,
```

```
        .command_length = 1U,
        .address_length = 3U,
        .data_length    = 1U,
        .dummy_cycles   = 0U
    };
    err = R_XSPI_CONTROLLER_DirectTransfer(p_ctrl, &xfer_adjust_cfr3,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Step 3: Change the target flash's configuration to use 8D-8D-8D mode. */
 spi_flash_direct_transfer_t xfer_change_mode =
    {
        .command       = XSPI_CONTROLLER_COMMAND_WRITE_REGISTER_SPI_MODE,
        .address       = XSPI_CONTROLLER_MODE_ADDRESS,
        .data_u.data    = XSPI_CONTROLLER_DOPI_REGISTER_SETTING,
        .command_length = 1U,
        .address_length = 3U,
        .data_length    = 1U,
        .dummy_cycles   = 0U
    };
    err = R_XSPI_CONTROLLER_DirectTransfer(p_ctrl, &xfer_change_mode,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Step 4: Speed up the bus by lowering the input divider.
  * If using the suggested clock settings this will increase the bus speed from 60
MHz to 120 MHz. */
    xspi_controller_change_omclk(BSP_CLOCKS_OCTACLK_DIV_2);
 /* Step 5: Transition the OSPI peripheral to DOPI mode. This will initiate auto
calibration as MDTR is 0.
  * This must happen after the device has been re-configured to the new mode.
Otherwise, communications will be
  * lost as both MCU and flash device will be in different modes. */
    err = R_XSPI_CONTROLLER_SpiProtocolSet(p_ctrl, SPI_FLASH_PROTOCOL_DOPI);
    assert(FSP_SUCCESS == err);
 /* Step 6: Verify that the chip is in requested DOPI mode. */
```

```
spi_flash_direct_transfer_t xfer_read_register =
   {
      .command       = XSPI_CONTROLLER_COMMAND_READ_REGISTER_SPI_MODE,
      .address       = XSPI_CONTROLLER_MODE_ADDRESS,
      .data_u.data    = 0,
      .command_length = 1U,
      .address_length = 4U,
      .data_length    = 1U,
      .dummy_cycles   = 5U,
   };
   err = R_XSPI_CONTROLLER_DirectTransfer(p_ctrl, &xfer_read_register,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
   assert(FSP_SUCCESS == err);
 if (XSPI_CONTROLLER_DOPI_REGISTER_SETTING != xfer_read_register.data_u.data)
   {
      __BKPT(0);
   }
}
```

Data Structures

struct	xspi_controller_table_t
struct	xspi_controller_timing_setting_t
struct	xspi_controller_command_set_t
struct	xspi_controller_dotf_cfg_t
struct	xspi_controller_extended_cfg_t
struct	xspi_controller_instance_ctrl_t

Macros

#define	XSPI_CONTROLLER_MAX_WRITE_ENABLE_POLLING_LOOPS

Enumerations

enum	xspi_controller_device_number_t
enum	xspi_controller_device_type_t

enum	xspi_controller_command_bytes_t
enum	xspi_controller_command_interval_clocks_t
enum	xspi_controller_command_cs_pullup_clocks_t
enum	xspi_controller_command_cs_pulldown_clocks_t
enum	xspi_controller_ds_timing_delay_t
enum	xspi_controller_sdr_drive_timing_t
enum	xspi_controller_ck_edge_t
enum	xspi_controller_sdr_sampling_delay_t
enum	xspi_controller_ddr_sampling_extension_t
enum	xspi_controller_output_assert_delay_t
enum	xspi_controller_frame_format_t
enum	xspi_controller_address_mode_t
enum	xspi_controller_latency_mode_t
enum	xspi_controller_prefetch_function_t
enum	xspi_controller_write_data_mask_t
enum	xspi_controller_combination_function_t
enum	xspi_controller_dotf_aes_key_length_words_t
enum	xspi_controller_dotf_aes_key_type_t
enum	xspi_controller_dotf_key_format_t

Data Structure Documentation

◆ xspi_controller_table_t

struct xspi_controller_table_t		
Simple array length table structure.		
Data Fields		
void *	p_table	Pointer to the table array.
uint8_t	length	Number of entries in the table.

◆ xspi_controller_timing_setting_t

struct xspi_controller_timing_setting_t		
Fixed timing configuration for bus signals.		
Data Fields		
xspi_controller_command_interval_clocks_t	command_to_command_interval	Interval between 2 consecutive commands.
xspi_controller_command_cs_pullup_clocks_t	cs_pullup_lag	Duration to de-assert CS line after the last command.
xspi_controller_command_cs_pulldown_clocks_t	cs_pulldown_lead	Duration to assert CS line before the first command.
xspi_controller_sdr_drive_timing_t	sdr_drive_timing	Data signal timing relative to the rising-edge of the CK signal.
xspi_controller_ck_edge_t	sdr_sampling_edge	Selects the clock edge to sample the data signal.
xspi_controller_sdr_sampling_delay_t	sdr_sampling_delay	Number of cycles to delay before sampling the data signal.
xspi_controller_ddr_sampling_extension_t	ddr_sampling_extension	Number of cycles to extending the data sampling window in DDR mode.
xspi_controller_output_assert_delay_t	output_assert_delay	Delays the assertion timing of IO[7:0] and DS signals by one cycle.

◆ xspi_controller_command_set_t

struct xspi_controller_command_set_t		
Command set used for a protocol mode.		
Data Fields		
spi_flash_protocol_t	protocol	Protocol mode associated with this command set.
xspi_controller_frame_format_t	frame_format	Frame format to use for this command set.
xspi_controller_latency_mode_t	latency_mode	Configurable or variable latency, only valid for XSPI_CONTROLLER_FRAME_FORMAT_XSPI_PROFILE_2 and XSPI_CONTROLLER_FRAME_FORMAT_XSPI_PROFILE_2_EXTENDED.
xspi_controller_address_mode_t	address_mode	Address mode to use for external device.
xspi_controller_command_bytes_t	command_bytes	Number of command bytes for each command code.

spi_flash_address_bytes_t	address_bytes	Number of bytes used during the address phase.
xspi_controller_write_data_mask_t	write_data_mask	Mask signal for each byte in the write data.
uint16_t	read_command	Read command.
uint16_t	program_command	Memory program/write command.
uint16_t	write_enable_command	Command to enable write or erase, set to 0x00 to ignore.
uint16_t	status_command	Command to read the write status, set to 0x00 to ignore.
uint16_t	row_load_command	Load a page into the device's internal buffer, set to 0x00 to ignore.
uint16_t	row_store_command	Stores the device's internal buffer to the memory page, set to 0x00 to ignore.
uint8_t	read_dummy_cycles	Dummy cycles to be inserted for read commands.
uint8_t	program_dummy_cycles	Dummy cycles to be inserted for page program commands.
uint8_t	status_dummy_cycles	Dummy cycles to be inserted for status read commands.
uint8_t	row_load_dummy_cycles	Dummy cycles to be inserted for the page load command.
uint8_t	row_store_dummy_cycles	Dummy cycles to be inserted for the page store command.
uint8_t	address_msb_mask	Mask of bits to zero when using memory-mapped operations; only applies to the most-significant byte.
bool	status_needs_address	Indicates that reading the status register requires an address stage.
uint32_t	status_address	Address to use for reading the status register with "busy" and "write-enable" flags.
spi_flash_address_bytes_t	status_address_bytes	Number of bytes used for status register addressing.
xspi_controller_table_t const *	p_erase_commands	List of all erase commands and associated sizes.

◆ xspi_controller_dotf_cfg_t

struct xspi_controller_dotf_cfg_t
This structure is used to hold all the DOTF related configuration.

◆ xspi_controller_extended_cfg_t

struct xspi_controller_extended_cfg_t		
XSPI_CONTROLLER Extended configuration.		
Data Fields		
uint8_t	xspi_controller_unit	The XSPI_CONTROLLER unit corresponding to the selected channel.
xspi_controller_device_number_t	channel	Device number to be used for memory device.
xspi_controller_device_type_t	memory_type	Type of memory device.
xspi_controller_timing_setting_t const *	p_timing_settings	Fixed protocol timing settings.
xspi_controller_table_t const *	p_xspi_command_set	Additional protocol command sets; if additional protocol commands set are not used set this to NULL.
xspi_controller_ds_timing_delay_t	data_latch_delay_clocks	Delay after assertion of the DS signal where data should be latched.
uint8_t *	p_autocalibration_preamble_pattern_addr	OctaFlash memory address holding the preamble pattern.
transfer_instance_t const *	p_lower_lvl_transfer	DMA Transfer instance used for data transmission.

◆ xspi_controller_instance_ctrl_t

struct xspi_controller_instance_ctrl_t		
Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called		
Data Fields		
spi_flash_cfg_t const *	p_cfg	Pointer to initial configuration.
uint32_t	open	Whether or not driver is open.
spi_flash_protocol_t	spi_protocol	Current XSPI protocol selected.
xspi_controller_device_number_t	channel	Device number to be used for memory device.
uint8_t	xspi_controller_unit	XSPI_CONTROLLER instance number.
xspi_controller_command_set_t const *	p_cmd_set	Command set for the active protocol mode.

R_XSPI0_Type *	p_reg	Address for the XSPI peripheral associated with this channel.

Macro Definition Documentation

◆ XSPI_CONTROLLER_MAX_WRITE_ENABLE_POLLING_LOOPS

#define XSPI_CONTROLLER_MAX_WRITE_ENABLE_POLLING_LOOPS
Maximum number of status polling checks after enabling memory writes.

Enumeration Type Documentation

◆ xspi_controller_device_number_t

enum xspi_controller_device_number_t	
XSPI Flash chip select	
Enumerator	
XSPI_CONTROLLER_DEVICE_NUMBER_0	Device connected to Chip-Select 0.
XSPI_CONTROLLER_DEVICE_NUMBER_1	Device connected to Chip-Select 1.

◆ xspi_controller_device_type_t

enum xspi_controller_device_type_t	
Enumerator	
XSPI_CONTROLLER_DEVICE_TYPE_FLASH	Device Memory type OctaFlash.
XSPI_CONTROLLER_DEVICE_TYPE_RAM	Device Memory type OctaRAM.

◆ xspi_controller_command_bytes_t

enum xspi_controller_command_bytes_t	
XSPI flash number of command code bytes.	
Enumerator	
XSPI_CONTROLLER_COMMAND_BYTES_1	Command codes are 1 byte long.
XSPI_CONTROLLER_COMMAND_BYTES_2	Command codes are 2 bytes long.

◆ xspi_controller_command_interval_clocks_t

enum xspi_controller_command_interval_clocks_t	
XSPI frame to frame interval	
Enumerator	
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_1	1 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_2	2 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_3	3 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_4	4 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_5	5 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_6	6 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_7	7 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_8	8 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_9	9 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_10	10 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_11	11 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_12	12 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_13	13 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_14	14 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_15	15 interval clocks
XSPI_CONTROLLER_COMMAND_INTERVAL_CLOCKS_16	16 interval clocks

◆ xspi_controller_command_cs_pullup_clocks_t

enum xspi_controller_command_cs_pullup_clocks_t	
XSPI chip select de-assertion duration	
Enumerator	
XSPI_CONTROLLER_COMMAND_CS_PULLUP_CLOCKS_NO_EXTENSION	CS asserting No extension.
XSPI_CONTROLLER_COMMAND_CS_PULLUP_CLOCKS_1	CS asserting Extend 1 cycle.

◆ xspi_controller_command_cs_pulldown_clocks_t

enum xspi_controller_command_cs_pulldown_clocks_t	
XSPI chip select assertion duration	
Enumerator	
XSPI_CONTROLLER_COMMAND_CS_PULLDOWN_CLOCKS_NO_EXTENSION	CS negating No extension.
XSPI_CONTROLLER_COMMAND_CS_PULLDOWN_CLOCKS_1	CS negating Extend 1 cycle.

◆ xspi_controller_ds_timing_delay_t

enum xspi_controller_ds_timing_delay_t	
XSPI data strobe delay.	
Enumerator	
XSPI_CONTROLLER_DS_TIMING_DELAY_NONE	Sample without delay.
XSPI_CONTROLLER_DS_TIMING_DELAY_1	Delay sampling by 1 clock cell.
XSPI_CONTROLLER_DS_TIMING_DELAY_2	Delay sampling by 2 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_3	Delay sampling by 3 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_4	Delay sampling by 4 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_5	Delay sampling by 5 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_6	Delay sampling by 6 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_7	Delay sampling by 7 clock cells.

XSPI_CONTROLLER_DS_TIMING_DELAY_8	Delay sampling by 8 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_9	Delay sampling by 9 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_10	Delay sampling by 10 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_11	Delay sampling by 11 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_12	Delay sampling by 12 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_13	Delay sampling by 13 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_14	Delay sampling by 14 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_15	Delay sampling by 15 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_16	Delay sampling by 16 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_17	Delay sampling by 17 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_18	Delay sampling by 18 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_19	Delay sampling by 19 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_20	Delay sampling by 20 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_21	Delay sampling by 21 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_22	Delay sampling by 22 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_23	Delay sampling by 23 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_24	Delay sampling by 24 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_25	Delay sampling by 25 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_26	Delay sampling by 26 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_27	Delay sampling by 27 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_28	Delay sampling by 28 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_29	Delay sampling by 29 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_30	Delay sampling by 30 clock cells.
XSPI_CONTROLLER_DS_TIMING_DELAY_31	Delay sampling by 31 clock cells.

◆ xspi_controller_sdr_drive_timing_t

enum xspi_controller_sdr_drive_timing_t	
XSPI SDR signal drive timing.	
Enumerator	
XSPI_CONTROLLER_SDR_DRIVE_TIMING_BEFORE_CK	SDR is asserted 1/2 cycle before the rising-edge of CK.
XSPI_CONTROLLER_SDR_DRIVE_TIMING_AT_CK	SDR is asserted at the rising-edge of CK.

◆ xspi_controller_ck_edge_t

enum xspi_controller_ck_edge_t	
Clock edge useed to sample data in SDR mode.	
Enumerator	
XSPI_CONTROLLER_CK_EDGE_FALLING	Falling-edge of CK signal.
XSPI_CONTROLLER_CK_EDGE_RISING	Rising-edge of CK signal.

◆ xspi_controller_sdr_sampling_delay_t

enum xspi_controller_sdr_sampling_delay_t	
SDR sampling window delay.	
Enumerator	
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_NONE	No sampling delay.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_1	Delay sampling by 1 cycle.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_2	Delay sampling by 2 cycles.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_3	Delay sampling by 3 cycles.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_4	Delay sampling by 4 cycles.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_5	Delay sampling by 5 cycles.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_6	Delay sampling by 6 cycles.
XSPI_CONTROLLER_SDR_SAMPLING_DELAY_7	Delay sampling by 7 cycles.

◆ xspi_controller_ddr_sampling_extension_t

enum xspi_controller_ddr_sampling_extension_t	
DDR sampling window extension.	
Enumerator	
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_NONE	No sampling extension.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_1	Sampling extended by 1 cycle.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_2	Sampling extended by 2 cycles.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_3	Sampling extended by 3 cycles.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_4	Sampling extended by 4 cycles.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_5	Sampling extended by 5 cycles.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_6	Sampling extended by 6 cycles.
XSPI_CONTROLLER_DDR_SAMPLING_EXTENSION_7	Sampling extended by 7 cycles.

◆ xspi_controller_output_assert_delay_t

enum xspi_controller_output_assert_delay_t	
Output enable delay settings.	
Enumerator	
XSPI_CONTROLLER_ASSERTION_DELAY_DISABLED	Assertion delay disable.
XSPI_CONTROLLER_ASSERTION_DELAY_ENABLED	Assertion delay enable.

◆ xspi_controller_frame_format_t

enum xspi_controller_frame_format_t	
Format of data frames used for communicating with the target device.	
Enumerator	
XSPI_CONTROLLER_FRAME_FORMAT_STANDARD	Standard frame with command, address, and data phases.
XSPI_CONTROLLER_FRAME_FORMAT_XSPI_PROFILE_1	JEDEC XSPI 8D-8D-8D Profile 1.0 frame.
XSPI_CONTROLLER_FRAME_FORMAT_XSPI_PROFILE_2	JEDEC XSPI 8D-8D-8D Profile 2.0 frame.
XSPI_CONTROLLER_FRAME_FORMAT_XSPI_PROFILE_2_EXTENDED	JEDEC XSPI 8D-8D-8D Profile 2.0 extended 6-byte command-address frame, used with HyperRAM.

◆ xspi_controller_address_mode_t

enum xspi_controller_address_mode_t	
Mode address setting.	
Enumerator	
XSPI_CONTROLLER_ADDRESS_MODE_NORMAL	Address is fixed to the number of address bytes for the command.
XSPI_CONTROLLER_ADDRESS_MODE_ARRAY	The flash target signifies additional address bytes by asserting the DQS line during the address phase.

◆ xspi_controller_latency_mode_t

enum xspi_controller_latency_mode_t	
Variable or fixed latency selection for flash devices which can notify the host of requiring additional time.	
Enumerator	
XSPI_CONTROLLER_LATENCY_MODE_FIXED	Latency is fixed to the number of dummy cycles for the command.
XSPI_CONTROLLER_LATENCY_MODE_VARIABLE	The flash target signifies additional latency (2x dummy cycles) by asserting the DQS line during the address phase.

◆ xspi_controller_prefetch_function_t

enum xspi_controller_prefetch_function_t	
Prefetch function settings	
Enumerator	
XSPI_CONTROLLER_PREFETCH_FUNCTION_DISABLE	Prefetch function disable.
XSPI_CONTROLLER_PREFETCH_FUNCTION_ENABLE	Prefetch function enable.

◆ xspi_controller_write_data_mask_t

enum xspi_controller_write_data_mask_t	
Write data mask settings	
Enumerator	
XSPI_CONTROLLER_WRITE_DATA_MASK_DISABLE	Write data mask disable.
XSPI_CONTROLLER_WRITE_DATA_MASK_ENABLE	Write data mask enable.

◆ xspi_controller_combination_function_t

enum xspi_controller_combination_function_t	
Combination function settings	
Enumerator	
XSPI_CONTROLLER_COMBINATION_FUNCTION_DISABLE	Combination function disable.
XSPI_CONTROLLER_COMBINATION_FUNCTION_4BYTE	Combine up to 4 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_8BYTE	Combine up to 8 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_12BYTE	Combine up to 12 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_16BYTE	Combine up to 16 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_20BYTE	Combine up to 20 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_24BYTE	Combine up to 24 bytes.

XSPI_CONTROLLER_COMBINATION_FUNCTION_28BYTE	Combine up to 28 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_32BYTE	Combine up to 32 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_36BYTE	Combine up to 36 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_40BYTE	Combine up to 40 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_44BYTE	Combine up to 44 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_48BYTE	Combine up to 48 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_52BYTE	Combine up to 52 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_56BYTE	Combine up to 56 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_60BYTE	Combine up to 60 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_64BYTE	Combine up to 64 bytes.
XSPI_CONTROLLER_COMBINATION_FUNCTION_2BYTE	Combine up to 2 bytes.

◆ xspi_controller_dotf_aes_key_length_words_t

enum xspi_controller_dotf_aes_key_length_words_t
XSPI DOTF AES Key Lengths.

◆ xspi_controller_dotf_aes_key_type_t

enum xspi_controller_dotf_aes_key_type_t
PSPI DOTF AES Type.

◆ xspi_controller_dotf_key_format_t

enum xspi_controller_dotf_key_format_t
XSPI DOTF key Type.

Function Documentation

◆ R_XSPI_CONTROLLER_Open()

fsp_err_t R_XSPI_CONTROLLER_Open (spi_flash_ctrl_t *const *p_ctrl*, spi_flash_cfg_t const *const *p_cfg*)

Open the xSPI device. After the driver is open, the xSPI device can be accessed like internal flash memory.

Implements spi_flash_api_t::open.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Driver has already been opened with the same p_ctrl.
FSP_ERR_DEVICE_BUSY	The device is busy.
FSP_ERR_INVALID_MODE	Invalid configuration for p_cmd_set.
FSP_ERR_CALIBRATE_FAILED	Failed to perform auto-calibrate.

◆ R_XSPI_CONTROLLER_Close()

fsp_err_t R_XSPI_CONTROLLER_Close (spi_flash_ctrl_t * *p_ctrl*)

Close the XSPI driver module.

Implements spi_flash_api_t::close.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_instance_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_XSPI_CONTROLLER_DirectWrite()

fsp_err_t R_XSPI_CONTROLLER_DirectWrite (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write*)

Writes raw data directly to the OctaFlash. API not supported. Use R_XSPI_CONTROLLER_DirectTransfer

Implements spi_flash_api_t::directWrite.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI.

◆ R_XSPI_CONTROLLER_DirectRead()

fsp_err_t R_XSPI_CONTROLLER_DirectRead (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Reads raw data directly from the OctaFlash. API not supported. Use R_XSPI_CONTROLLER_DirectTransfer.

Implements spi_flash_api_t::directRead.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI.

◆ R_XSPI_CONTROLLER_DirectTransfer()

fsp_err_t R_XSPI_CONTROLLER_DirectTransfer (spi_flash_ctrl_t * *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction*)

Read/Write raw data directly with the OctaFlash.

Implements spi_flash_api_t::directTransfer.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_XSPI_CONTROLLER_SpiProtocolSet()

fsp_err_t R_XSPI_CONTROLLER_SpiProtocolSet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol*)

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

Return values

FSP_SUCCESS	SPI protocol updated on MPU peripheral.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_DEVICE_BUSY	The device is busy.
FSP_ERR_INVALID_MODE	Invalid configuration for p_cmd_set.
FSP_ERR_CALIBRATE_FAILED	Failed to perform auto-calibrate.

◆ R_XSPI_CONTROLLER_XipEnter()

fsp_err_t R_XSPI_CONTROLLER_XipEnter (spi_flash_ctrl_t * *p_ctrl*)

Enters XIP (execute in place) mode.

Implements spi_flash_api_t::xipEnter.

Return values

FSP_SUCCESS	XiP mode was entered successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	XiP support is not enabled.

◆ R_XSPI_CONTROLLER_XipExit()

fsp_err_t R_XSPI_CONTROLLER_XipExit (spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

Return values

FSP_SUCCESS	XiP mode was entered successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	XiP support is not enabled.

◆ R_XSPI_CONTROLLER_Write()

fsp_err_t R_XSPI_CONTROLLER_Write (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count*)

Program a page of data to the flash.

Implements spi_flash_api_t::write.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_NOT_ENABLED	Write enable failed.
FSP_ERR_INVALID_SIZE	Insufficient space remaining in page or write length is not a multiple of CPU access size when not using the DMAC.
FSP_ERR_DEVICE_BUSY	Another Write/Erase transaction is in progress.
FSP_ERR_INVALID_ADDRESS	Destination or source is not aligned to CPU access alignment when not using the DMAC.

◆ R_XSPI_CONTROLLER_Erase()

fsp_err_t R_XSPI_CONTROLLER_Erase (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count*)

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

Return values

FSP_SUCCESS	The command to erase the flash was executed successfully.
FSP_ERR_ASSERTION	p_instance_ctrl or p_device_address is NULL, byte_count doesn't match an erase size defined in spi_flash_cfg_t, or byte_count is set to 0.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_NOT_ENABLED	Write enable failed.
FSP_ERR_DEVICE_BUSY	The device is busy.

◆ R_XSPI_CONTROLLER_StatusGet()

fsp_err_t R_XSPI_CONTROLLER_StatusGet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status*)

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

Return values

FSP_SUCCESS	The write status is in p_status.
FSP_ERR_ASSERTION	p_instance_ctrl or p_status is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_XSPI_CONTROLLER_BankSet()

fsp_err_t R_XSPI_CONTROLLER_BankSet (spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank*)

Selects the bank to access. Use XSPI_CONTROLLER_bank_select_t as the bank value.

Implements spi_flash_api_t::bankSet.

Return values

FSP_ERR_UNSUPPORTED	This function is unsupported.

◆ R_XSPI_CONTROLLER_AutoCalibrate()

fsp_err_t R_XSPI_CONTROLLER_AutoCalibrate (spi_flash_ctrl_t *const *p_ctrl*)

AutoCalibrate the XSPI_CONTROLLER DS signal.

Implements spi_flash_api_t::autoCalibrate.

Return values

FSP_SUCCESS	Autocalibration completed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	Autocalibration support is not enabled.
FSP_ERR_DEVICE_BUSY	The device is busy.
FSP_ERR_CALIBRATE_FAILED	Failed to perform auto-calibrate.

◆ R_XSPI_CONTROLLER_DOTF_Configure()

fsp_err_t R_XSPI_CONTROLLER_DOTF_Configure (spi_flash_ctrl_t *const *p_ctrl*, xspi_controller_dotf_cfg_t *const *p_dotf_cfg*)

Configure DOTF

Parameters

[in]	p_ctrl	Pointer to XSPI specific control structure
[in]	p_dotf_cfg	Pointer to DOTF configuration structure

Return values

FSP_SUCCESS	DOTF enabled successfully
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	DOTF support is not enabled

◆ R_XSPI_CONTROLLER_RowLoad()

fsp_err_t R_XSPI_CONTROLLER_RowLoad (spi_flash_ctrl_t *const *p_ctrl*, uint32_t *row_index*)

Loads a row into the page buffer on devices which support row addressing.

Parameters

[in]	p_ctrl	Pointer to a driver handle.
	row_index	Index of the row in the XSPI device.

Return values

FSP_SUCCESS	Page transaction succeeded.
FSP_ERR_INVALID_ARGUMENT	Invalid argument.
FSP_ERR_NOT_OPEN	Driver has not been opened.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_UNSUPPORTED	Row addressing support is not enabled

◆ R_XSPI_CONTROLLER_RowStore()

fsp_err_t R_XSPI_CONTROLLER_RowStore (spi_flash_ctrl_t *const *p_ctrl*, uint32_t *row_index*)

Stores (programs) the page buffer to a row on devices which support row addressing.

Parameters

[in]	p_ctrl	Pointer to a driver handle.
	row_index	Index of the row in the XSPI device.

Return values

FSP_SUCCESS	Page transaction succeeded.
FSP_ERR_INVALID_ARGUMENT	Invalid argument.
FSP_ERR_NOT_OPEN	Driver has not been opened.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_UNSUPPORTED	Row addressing support is not enabled

5.2.17.4 XSPI (r_xspi_target)

Modules » Storage

Functions

fsp_err_t	R_XSPI_TARGET_Open (spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg)
fsp_err_t	R_XSPI_TARGET_Close (spi_flash_ctrl_t *const p_ctrl)
fsp_err_t	R_XSPI_TARGET_DirectWrite (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)
fsp_err_t	R_XSPI_TARGET_DirectRead (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_XSPI_TARGET_DirectTransfer (spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)
fsp_err_t	R_XSPI_TARGET_SpiProtocolSet (spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol)
fsp_err_t	R_XSPI_TARGET_XipEnter (spi_flash_ctrl_t *const p_ctrl)

fsp_err_t	R_XSPI_TARGET_XipExit (spi_flash_ctrl_t *const p_ctrl)
fsp_err_t	R_XSPI_TARGET_Write (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)
fsp_err_t	R_XSPI_TARGET_Erase (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)
fsp_err_t	R_XSPI_TARGET_StatusGet (spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status)
fsp_err_t	R_XSPI_TARGET_BankSet (spi_flash_ctrl_t *const _ctrl, uint32_t bank)
fsp_err_t	R_XSPI_TARGET_AutoCalibrate (spi_flash_ctrl_t *const p_ctrl)

Detailed Description

Driver for the XSPI_TARGET peripheral on RX MCUs. This module implements the SPI Flash Interface.

Overview

The XSPI_TARGET peripheral supports xSPI access SRAM, and it interfaces with these devices to perform data I/O Operations.

Please note that this document will reference the xSPI protocol to which XSPI is a subset. The XSPI_TARGET peripheral is compatible with a variety of xSPI protocol operating modes.

Features

The XSPI_TARGET driver has the following key features to support the **xSPI** device:

- Perform data I/O Operation
- Can be configured with xSPI devices on either of the 2 channels
- Support for standard SPI frames, xSPI Protocol 1.0, xSPI Protocol 2.0, and xSPI Protocol 2.0 Extended frames.

Configuration

XSPI_TARGET:

Build Time Configurations for r_xspi_target

The following build time configurations are defined in fsp_cfg/r_xspi_target_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > XSPI Target (r_xspi_target)

This module can be added to the Stacks tab via New Stack > Storage > XSPI Target (r_xspi_target).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_xspi_target0	Module name.
General > Unit	MCU Specific Options		Specify the hardware unit.
Command Sets > Read > Command Code > Register	Command codes must be positive and fit in the selected command byte length.	0x13	Read register command code.
Command Sets > Read > Command Code > Memory	Command codes must be positive and fit in the selected command byte length.	0x13	Read memory command code.
Command Sets > Read > Dummy Cycles	15 cycles	15 cycles	The number of latency cycles to insert during read transactions in memory mapped mode. Set Read Dummy Cycles to 0Fh (15 cycles).
Command Sets > Write > Command Code > Register	Command codes must be positive and fit in the selected command byte length.	0x12	Write register command code.
Command Sets > Write > Command Code > Memory	Command codes must be positive and fit in the selected command byte length.	0x12	Write memory command code.
Command Sets > Write > Dummy Cycles	Dummy cycle counts must be an integer between 0 and 31.	0	The number of latency cycles to insert during write transactions in memory mapped mode.
Command Sets > Protocol Mode	• 1S-1S-1S • 4S-4D-4D • 8D-8D-8D	1S-1S-1S	Signal protocol mode to use for this command set.
Command Sets > Frame Format	• Normal format • 8D-8D-8D profile 1.0 format • 8D-8D-8D profile 2.0 command modifier format • 8D-8D-8D profile 2.0	Normal format	Frame format to use with this command set.

	extended command modifier format		
Command Sets > Address Length	• 2 bytes (64-Kbyte address space) • 3 bytes (16-Mbyte address space) • 4 bytes (4-Gbyte address space)	3 bytes (16-Mbyte address space)	Number of bytes used to address data in the address field in memory mapped mode.
Size Setting > Maximum write frame size	• 8 • 16 • 32 • 64 • 128 • 256 • 512	64	Maximum data size for the write frame
Size Setting > Maximum read frame size	• 8 • 16 • 32 • 64 • 128 • 256 • 512	64	Maximum data size for the read frame
Size Setting > Memory size	• 4 Kbytes • 8 Kbytes • 16 Kbytes • 32 Kbytes • 64 Kbytes • 128 Kbytes • 256 Kbytes • 512 Kbytes • 1 Mbytes	4 Kbytes	Setting address space of the XSPI target module
Timing Setting > Output Assert Delay	• Disabled • Enabled	Disabled	Assertion timing of the output buffer control signal. Setting this bit to 1 delays the start of output from the IO pins and the DS pin by one cycle
Write Data Mask	• Disabled • Enabled	Disabled	Disable/Enable Write Data Mask

Note

The user is expected to modify the command definitions based on the xSPI chip and SPI communication mode.

Clock Configuration

The signals to the xSPI target device are derived from OCTACLK. OCTACLK must be set to twice the

desired xSPI_CLK signal frequency. This allows data to be latched on either clock edge for Dual-Data-Rate modes (4S-4D-4D or 8D-8D-8D).

Please refer to the Hardware User Manual for the clocks which control the XSPI_TARGET registers. These along with OCTACLK can be set on the **Clocks** tab of the RA Configuration editor.

Pin Configuration

The following pins are available to connect to an external XSPI device:

- XSPIn_CK: clock output (XSPICLK / 2)
- XSPIn_DS: data strobe signal
- XSPIn_CS0: device 0 select
- XSPIn_IO0: Data 0 I/O
- XSPIn_IO1: Data 1 I/O
- XSPIn_IO2: Data 2 I/O
- XSPIn_IO3: Data 3 I/O
- XSPIn_IO4: Data 4 I/O
- XSPIn_IO5: Data 5 I/O
- XSPIn_IO6: Data 6 I/O
- XSPIn_IO7: Data 7 I/O

Note

Data pins can be configured with IOPORT_CFG_DRIVE_6 or IOPORT_CFG_DRIVE_8 according to the Electrical Characteristics chapter.
Chip Select pins should be configured with at least IOPORT_CFG_DRIVE_4.

Usage Notes

Usage Notes for xSPI support

XSPI Controller module will access SRAM via the XSPI Target module. So the protocol-specific configuration of the XSPI Target module must be mapped to XSPI Controller module.

Protocol mode supported:

- 1S-1S-1S
- 4S-4D-4D
- 8D-8D-8D

XSPI Target module can config can configure itself, or it can be configured by the XSPI Controller module.

Access mode:

- Register Access Mode:
  - XSPI Controller module can read register of XSPI Target module
  - XSPI Controller module can write register of XSPI Target module
  - XSPI Controller module can change IO protocol mode of XSPI Target module
  - XSPI Controller module can reset XSPI Target module
- Memory Access Mode:
  - XSPI Controller module can read memory of XSPI Target module
  - XSPI Controller module can write memory of XSPI Target module

xSPI Commands

The command of XSPI Target has 4 types:

- Read memory command
- Read register command
- Write memory command
- Write register command

The command of XSPI Target module can be modified. If a command set is not found, an FSP_ERR_INVALID_MODE error code will be returned.

Examples

XSPI Target Basic Example

This is a basic example of minimal use of the XSPI Target in an application.

```
void r_xspi_target_basic_example (void)
{
 /* Open the XSPI instance. */
 fsp_err_t err = R_XSPI_TARGET_Open(&g_xspi_target_ctrl, &g_xspi_target_cfg);
    assert(FSP_SUCCESS == err);
 /* (Optional) Change SPI to DOPI mode */
    err = R_XSPI_TARGET_SpiProtocolSet(&g_xspi_target_ctrl,
SPI_FLASH_PROTOCOL_8D_8D_8D);
    assert(FSP_SUCCESS == err);
    err = R_XSPI_TARGET_Close(&g_xspi_target_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	xspi_target_timing_setting_t
struct	xspi_target_command_set_t
struct	xspi_target_instance_ctrl_t
struct	xspi_target_extended_cfg_t

Enumerations

enum	xspi_target_frame_format_t
enum	xspi_target_maximum_frame_size_t

enum	xspi_target_memory_size_t
enum	xspi_target_output_assert_delay_t
enum	xspi_target_write_data_mask_enable_t

Data Structure Documentation

◆ xspi_target_timing_setting_t

struct xspi_target_timing_setting_t		
Fixed timing configuration for bus signals.		
Data Fields		
xspi_target_output_assert_delay_t	output_assert_delay	Delays the assertion timing of IO[7:0] and DS signals by one cycle.

◆ xspi_target_command_set_t

struct xspi_target_command_set_t		
Command set used for a protocol mode.		
Data Fields		
xspi_target_frame_format_t	frame_format	Frame format to use for this command set.
uint8_t	read_register_command	Read register command.
uint8_t	read_memory_command	Read memory command.
uint8_t	write_register_command	Write register command.
uint8_t	write_memory_command	Write memory command.
uint8_t	read_dummy_cycles	Dummy cycles to be inserted for read commands.
uint8_t	program_dummy_cycles	Dummy cycles to be inserted for page program commands.

◆ xspi_target_instance_ctrl_t

struct xspi_target_instance_ctrl_t		
Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called		
Data Fields		
spi_flash_cfg_t const *	p_cfg	Pointer to initial configuration.
uint32_t	open	Whether or not driver is open.
spi_flash_protocol_t	spi_protocol	Current XSPI protocol selected.
R_XSPI_TARGET0_Type *	p_reg	Address for the XSPI peripheral

		associated with this channel.

◆ xspi_target_extended_cfg_t

struct xspi_target_extended_cfg_t		
XSPI_TARGET Extended configuration.		
Data Fields		
uint8_t	unit	XSPI_TARGET instance number.
xspi_target_timing_setting_t const *	p_timing_settings	Fixed protocol timing settings.
xspi_target_maximum_frame_size_t	max_read_frame_size	
xspi_target_maximum_frame_size_t	max_write_frame_size	
xspi_target_memory_size_t	memory_size	Determine address space of XSPI Target module.
xspi_target_write_data_mask_enable_t	write_data_mask	Mask signal for each byte in the write data;.
xspi_target_command_set_t const *	p_cmd_set	Command set for the active protocol mode.

Enumeration Type Documentation

◆ xspi_target_frame_format_t

enum xspi_target_frame_format_t	
Format of data frames used for communicating with the target device.	
Enumerator	
XSPI_TARGET_FRAME_FORMAT_STANDARD	Standard frame with command, address, and data phases.
XSPI_TARGET_FRAME_FORMAT_XSPI_PROFILE_1	JEDEC XSPI 8D-8D-8D Profile 1.0 frame.
XSPI_TARGET_FRAME_FORMAT_XSPI_PROFILE_2	JEDEC XSPI 8D-8D-8D Profile 2.0 frame.
XSPI_TARGET_FRAME_FORMAT_XSPI_PROFILE_2_EXTENDED	JEDEC XSPI 8D-8D-8D Profile 2.0 extended 6-byte command-address frame, used with HyperRAM.

◆ xspi_target_maximum_frame_size_t

enum xspi_target_maximum_frame_size_t	
Enumerator	
XSPI_TARGET_MAX_FRAME_SIZE_8	Maximum supported frame size is 8 bytes.
XSPI_TARGET_MAX_FRAME_SIZE_16	Maximum supported frame size is 16 bytes.
XSPI_TARGET_MAX_FRAME_SIZE_32	Maximum supported frame size is 32 bytes.
XSPI_TARGET_MAX_FRAME_SIZE_64	Maximum supported frame size is 64 bytes.
XSPI_TARGET_MAX_FRAME_SIZE_128	Maximum supported frame size is 128 bytes.
XSPI_TARGET_MAX_FRAME_SIZE_256	Maximum supported frame size is 256 bytes.
XSPI_TARGET_MAX_FRAME_SIZE_512	Maximum supported frame size is 512 bytes.

◆ xspi_target_memory_size_t

enum xspi_target_memory_size_t	
Memory size of XSPI Target module	
Enumerator	
XSPI_TARGET_MEMORY_SIZE_4_KBYTES	4 KB memory size (address range 0x0000-0x0FFF).
XSPI_TARGET_MEMORY_SIZE_8_KBYTES	8 KB memory size (address range 0x0000-0x1FFF).
XSPI_TARGET_MEMORY_SIZE_16_KBYTES	16 KB memory size (address range 0x0000-0x3FFF).
XSPI_TARGET_MEMORY_SIZE_32_KBYTES	32 KB memory size (address range 0x0000-0x7FFF).
XSPI_TARGET_MEMORY_SIZE_64_KBYTES	64 KB memory size (address range 0x0000-0xFFFF).
XSPI_TARGET_MEMORY_SIZE_128_KBYTES	128 KB memory size (address range 0x0000-0x1FFFF).
XSPI_TARGET_MEMORY_SIZE_256_KBYTES	256 KB memory size (address range 0x0000-0x3FFFF).
XSPI_TARGET_MEMORY_SIZE_512_KBYTES	512 KB memory size (address range 0x0000-0x7FFFF).
XSPI_TARGET_MEMORY_SIZE_1_MBYTE	1 MB memory size (address range 0x0000-0xFFFFF).

◆ xspi_target_output_assert_delay_t

enum xspi_target_output_assert_delay_t	
Output enable delay settings.	
Enumerator	
XSPI_TARGET_ASSERTION_DELAY_DISABLED	Assertion delay disable.
XSPI_TARGET_ASSERTION_DELAY_ENABLED	Assertion delay enable.

◆ xspi_target_write_data_mask_enable_t

enum xspi_target_write_data_mask_enable_t	
Enumerator	
XSPI_TARGET_WRITE_DATA_MASK_DISABLE	Disable write data mask.
XSPI_TARGET_WRITE_DATA_MASK_ENABLE	Enable write data mask.

Function Documentation

◆ R_XSPI_TARGET_Open()

fsp_err_t R_XSPI_TARGET_Open (spi_flash_ctrl_t *const *p_ctrl*, spi_flash_cfg_t const *const *p_cfg*)

Open the xSPI device. After the driver is open, the xSPI device can be accessed like internal flash memory.

Implements spi_flash_api_t::open.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Driver has already been opened with the same p_ctrl.
FSP_ERR_INVALID_MODE	Invalid configuration for p_cmd_set.

◆ R_XSPI_TARGET_Close()

fsp_err_t R_XSPI_TARGET_Close (spi_flash_ctrl_t * *p_ctrl*)

Close the XSPI driver module.

Implements spi_flash_api_t::close.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_instance_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_XSPI_TARGET_DirectWrite()

fsp_err_t R_XSPI_TARGET_DirectWrite (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write*)

Writes raw data directly to the OctaFlash. API not supported. Use R_XSPI_TARGET_DirectTransfer

Implements spi_flash_api_t::directWrite.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_DirectRead()

fsp_err_t R_XSPI_TARGET_DirectRead (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Reads raw data directly from the OctaFlash. API not supported. Use R_XSPI_TARGET_DirectTransfer.

Implements spi_flash_api_t::directRead.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_DirectTransfer()

fsp_err_t R_XSPI_TARGET_DirectTransfer (spi_flash_ctrl_t * *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction*)

Read/Write a transfer to XSPI Target.

Implements spi_flash_api_t::directTransfer.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_SpiProtocolSet()

fsp_err_t R_XSPI_TARGET_SpiProtocolSet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol*)

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

Return values

FSP_SUCCESS	SPI protocol updated on MPU peripheral.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_XSPI_TARGET_XipEnter()

fsp_err_t R_XSPI_TARGET_XipEnter (spi_flash_ctrl_t * *p_ctrl*)

Enters XIP (execute in place) mode.

Implements spi_flash_api_t::xipEnter.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_XipExit()

fsp_err_t R_XSPI_TARGET_XipExit (spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_Write()

fsp_err_t R_XSPI_TARGET_Write (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count*)

Program a page of data to the flash.

Implements spi_flash_api_t::write.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_Erase()

fsp_err_t R_XSPI_TARGET_Erase (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count*)

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_StatusGet()

fsp_err_t R_XSPI_TARGET_StatusGet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status*)

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

Return values

FSP_ERR_UNSUPPORTED	API not supported by XSPI Target module.

◆ R_XSPI_TARGET_BankSet()

fsp_err_t R_XSPI_TARGET_BankSet (spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank*)
Selects the bank to access. Use XSPI_TARGET_bank_select_t as the bank value. Implements spi_flash_api_t::bankSet. **Return values** FSP_ERR_UNSUPPORTED — This function is unsupported.

◆ R_XSPI_TARGET_AutoCalibrate()

fsp_err_t R_XSPI_TARGET_AutoCalibrate (spi_flash_ctrl_t *const *p_ctrl*)
AutoCalibrate the XSPI_TARGET DS signal. Implements spi_flash_api_t::autoCalibrate. **Return values** FSP_ERR_UNSUPPORTED — This function is unsupported.

5.2.17.5 Block Media RAM (rm_block_media_ram)

Modules » Storage

Functions

fsp_err_t	RM_BLOCK_MEDIA_RAM_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_RAM_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_RAM_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_RAM_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_RAM_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_RAM_StatusGet (rm_block_media_ctrl_t *const

	p_api_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_RAM_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_RAM_Close (rm_block_media_ctrl_t *const p_ctrl)

Detailed Description

Middleware that implements a block media interface on RAM. This module implements the Block Media Interface.

Overview

Features

- This module is for using RAM as a media area.
- Supports using any read/write section of the address space as the media area.

Note

When using an external RAM as the media area (like SDRAM or OctaRAM), it must be initialized before calling RM_BLOCK_MEDIA_RAM_Open().
When using PMSC driver in combination with this module, the API of this module is called from the PMSC driver.

Configuration

Build Time Configurations for rm_block_media_ram

The following build time configurations are defined in fsp_cfg/rm_block_media_ram_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
RAM Media Size	Please enter the RAM media size.	65536	Enter a RAM media size of 20K bytes or more.
Memory Storage Section	Manual Entry	.bss	This is the section of memory that will be used for the backing storage. Must be a valid Linker section. ('.bss', '.sdram', '.OSPI_DEVICE_0_RAM', etc). For AC6 preface any section with '.bss' e.g. '.bss.sdram'
Format during mediaInit	• Enabled • Disabled	Enabled	If disabled RAM must be formatted with filex_media_format

			before calling filex_media_open.

Configurations for Storage > Block Media RAM Implementation (rm_block_media_ram)

This module can be added to the Stacks tab via New Stack > Storage > Block Media RAM Implementation (rm_block_media_ram).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_block_media0	Module name.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed.

Note

Specify RAM media size of 20 Kbytes or more. This module cannot be used with an MCU that cannot allocate at least 20 Kbytes of RAM as media area.
When using AC6 it is recommended to start section names with ".bss" (ex. ".bss.sdram") to prevent unnecessarily large binaries. This tells AC6 to create ZI sections rather than initialized sections.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

This is a basic example of minimal use of the RAM block media implementation in an application.

```
#define RM_BLOCK_MEDIA_RAM_BLOCK_SIZE (512U)
uint8_t g_dest[RM_BLOCK_MEDIA_RAM_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_RAM_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
rm_block_media_status_t status;
void rm_block_media_ram_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_RAM_BLOCK_SIZE; i++)
```

```
    {
        g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_RAM driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_RAM_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* When enabled, format RAM as FAT file system. */
    err = RM_BLOCK_MEDIA_RAM_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Get the RM_BLOCK_MEDIA_RAM driver status. */
    err = RM_BLOCK_MEDIA_RAM_StatusGet(&g_rm_block_media0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 if (true == status.initialized)
    {
 /* Write a block of data to block 3 of the RAM memory. */
       err = RM_BLOCK_MEDIA_RAM_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
      assert(FSP_SUCCESS == err);
 /* Read a block of data from block 3 of the RAM memory. */
       err = RM_BLOCK_MEDIA_RAM_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
      assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Close the RM_BLOCK_MEDIA_RAM driver. */
 RM_BLOCK_MEDIA_RAM_Close(&g_rm_block_media0_ctrl);
    }
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_RAM_Open()

fsp_err_t RM_BLOCK_MEDIA_RAM_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Opens the module.

Implements rm_block_media_api_t::open().

Return values

FSP_SUCCESS	Module is available and is now open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Module has already been opened with this instance of the control structure.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_RAM_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_RAM_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the RAM media area.

Implements rm_block_media_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_RAM_Read()

fsp_err_t RM_BLOCK_MEDIA_RAM_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Reads data from RAM media.

Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_RAM_Write()

fsp_err_t RM_BLOCK_MEDIA_RAM_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Writes data to RAM media.

Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

Return values

FSP_SUCCESS	Write finished successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_RAM_Erase()

fsp_err_t RM_BLOCK_MEDIA_RAM_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Erases sectors of RAM media.

Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_RAM_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_RAM_StatusGet (rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status.

Implements rm_block_media_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_RAM_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_RAM_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information.

Implements rm_block_media_api_t::infoGet().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_RAM_Close()

fsp_err_t RM_BLOCK_MEDIA_RAM_Close (rm_block_media_ctrl_t *const *p_ctrl*)

Closes the module.

Implements rm_block_media_api_t::close().

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

5.2.17.6 Block Media SD/MMC (rm_block_media_sdmmc)

Modules » Storage

Functions

fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Write (rm_block_media_ctrl_t *const

	p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_CallbackSet (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Close (rm_block_media_ctrl_t *const p_ctrl)

Detailed Description

Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface.

Overview

Features

The SD/MMC implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from an SD card
- Callback called when card insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

Configuration

Build Time Configurations for rm_block_media_sdmmc

The following build time configurations are defined in driver/rm_block_media_sdmmc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > Block Media SD/MMC (rm_block_media_sdmmc)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SD/MMC (rm_block_media_sdmmc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_block_media0	Module name.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

This is a basic example of minimal use of the SD/MMC block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE (512)
uint8_t g_dest[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void rm_block_media_sdmmc_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SDMMC driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SDMMC_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
```

```
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = RM_BLOCK_MEDIA_SDMMC_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_SDMMC_Open()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Opens the module.

Implements rm_block_media_api_t::open().

Return values

FSP_SUCCESS	Module is available and is now open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Module has already been opened with this instance of the control structure.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::open

◆ RM_BLOCK_MEDIA_SDMMC_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements rm_block_media_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::mediaInit

◆ RM_BLOCK_MEDIA_SDMMC_Read()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements rm_block_media_api_t::read().

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::read

◆ RM_BLOCK_MEDIA_SDMMC_Write()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements rm_block_media_api_t::write().

Return values

FSP_SUCCESS	Write finished successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::write

◆ RM_BLOCK_MEDIA_SDMMC_Erase()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Erases sectors of an SD card or eMMC device. Implements rm_block_media_api_t::erase().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::erase
- sdmmc_api_t::statusGet

◆ RM_BLOCK_MEDIA_SDMMC_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_CallbackSet (rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements rm_block_media_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ RM_BLOCK_MEDIA_SDMMC_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_StatusGet (rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status. Implements rm_block_media_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_SDMMC_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information. Implements rm_block_media_api_t::infoGet().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

◆ RM_BLOCK_MEDIA_SDMMC_Close()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Close (rm_block_media_ctrl_t *const *p_ctrl*)
Closes an open SD/MMC device. Implements rm_block_media_api_t::close(). **Return values** FSP_SUCCESS — Successful close. FSP_ERR_ASSERTION — An input parameter is invalid. FSP_ERR_NOT_OPEN — Module is not open.

5.2.17.7 Block Media SPI Flash (rm_block_media_spi)

Modules » Storage

Functions

fsp_err_t	RM_BLOCK_MEDIA_SPI_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_SPI_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_SPI_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_block, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SPI_StatusGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const start_block, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SPI_CallbackSet (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Close (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const start_block, uint32_t const num_blocks)

Detailed Description

Middleware to implement the block media interface on SPI flash memory. This module implements the Block Media Interface.

Overview

Features

The SPI implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from SPI flash memory
- Provides media information such as sector size and total number of sectors.

Note

By default, Block Media SPI Read, Write, and Erase are blocking operations. Non-blocking operation may be achieved by yielding control within the optional callback function.

Configuration

Build Time Configurations for rm_block_media_spi

The following build time configurations are defined in driver/rm_block_media_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected, code for parameter checking is included in the build

Build Time Configurations for rm_block_media_spi

The following build time configurations are defined in driver/rm_block_media_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected, code for parameter checking is included in the build

Configurations for Storage > Block Media SPI Flash (rm_block_media_spi)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SPI Flash (rm_block_media_spi).

Configuration	Options	Default	Description
Module Instance Name	Name must be a valid C symbol	g_rm_block_media0	Module name
Block size (bytes)	Manual Entry	4096	Specify the size of a block in bytes.
Block count	Minimum block count is	8192	Number of blocks

	1, maximum is defined by hardware and software design.		available for use by this driver instance.
Base Address	Manual Entry	0	Base address offset (bytes) for instance memory region.
Callback Function	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback is provided, it will be called after the completion of Read, Write, and Erase operations, or anytime these functions are waiting on hardware.
Write frame size (bytes)	Value must be a non-negative integer	64	Number of bytes passed to the lower level SPI flash driver per write call. The default value writes the full page size per call. When the lower level driver limits single writes below the page size (for example the XSPI_CONTROLLER Combination Function), set this to a frame size that divides the page size evenly.

Configurations for Storage > Block Media SPI Flash (rm_block_media_spi)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SPI Flash (rm_block_media_spi).

Configuration	Options	Default	Description
Module Instance Name	Name must be a valid C symbol	g_rm_block_media0	Module name
Block size (bytes)	Manual Entry	4096	Specify the size of a block in bytes.
Block count	Minimum block count is 1, maximum is defined by hardware and software design.	8192	Number of blocks available for use by this driver instance.
Base Address	Manual Entry	0	Base address offset (bytes) for instance memory region.
Callback Function	Name must be a valid	NULL	A user callback

	C symbol	function can be provided. If this callback is provided, it will be called after the completion of Read, Write, and Erase operations, or anytime these functions are waiting on hardware.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Limitations

Developers should be aware of the following limitations when using RM_BLOCK_MEDIA_SPI:

- Getting and setting Block Protection or Advanced Sector Protection modes is not supported.
- Addressing QSPI memory address ranges greater than 64 MB (one bank) is not supported.
- write_frame_size_bytes in rm_block_media_spi_extended_cfg_t specifies the number of bytes passed to the lower-level SPI flash driver in each write call. When using XSPI_CONTROLLER, set write_frame_size_bytes to the Combination Function frame size. The value must be greater than zero, must not exceed page_size_bytes, and must divide page_size_bytes evenly; otherwise, RM_BLOCK_MEDIA_SPI_Open() returns FSP_ERR_INVALID_SIZE. DMAC supports frame sizes of 2, 4, 8, 16, 32, and 64 bytes; without DMAC, only frame sizes of 8, 16, 32, and 64 bytes are supported.

Examples

Basic Example

This is a basic example of minimal use of the SPI block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SPI_BLOCK_SIZE (256U)
uint8_t g_dest[RM_BLOCK_MEDIA_SPI_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SPI_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_spi_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SPI_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
```

```
 /* Open the RM_BLOCK_MEDIA_SPI driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to block 3 of the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from block 3 of the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

Non-Blocking Example

This is a basic example of using the optional SPI callback to impliment non-blocking operation.

```
#define RM_BLOCK_MEDIA_EXAMPLE_DEVICE_BLOCK_COUNT 0x1000
void rm_block_media_spi_non_blocking_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SPI_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SPI driver. This enables the card detection interrupt. */
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the Block Media SPI driver. */
    err = RM_BLOCK_MEDIA_SPI_MediaInit(&g_rm_block_media0_ctrl);
```

```
    assert(FSP_SUCCESS == err);
 /* Erase a large quantity of data from SPI Flash Memory */
    err = RM_BLOCK_MEDIA_SPI_Erase(&g_rm_block_media0_ctrl, 0,
RM_BLOCK_MEDIA_EXAMPLE_DEVICE_BLOCK_COUNT);
    assert(FSP_SUCCESS == err);
}
/* The optional callback is invoked for Read, Write, and Erase operations, whenever
the operation completes or has
 * been blocked by the lower level SPI driver busy indication.
 */
void rm_block_media_spi_example_callback (rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE == p_args->event)
    {
 /* Process operation complete. */
    }
 else if (RM_BLOCK_MEDIA_EVENT_POLL_STATUS == p_args->event)
    {
 rm_block_media_status_t status;
 rm_block_media_ctrl_t * p_ctrl = (rm_block_media_ctrl_t *) p_args->p_context;
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_StatusGet(p_ctrl, &status);
      assert(FSP_SUCCESS == err);
 if (true == status.busy)
      {
 /* Run waiting tasks */
            vTaskSuspend(xTaskGetCurrentTaskHandle());
      }
    }
 else
    {
       assert(RM_BLOCK_MEDIA_EVENT_ERROR == p_args->event);
 /* Process Read, Write, or Erase error. */
    }
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_SPI_Open()

fsp_err_t RM_BLOCK_MEDIA_SPI_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Parameter checking and Acquires mutex, then handles driver initialization at the HAL SPI layer and marking the open flag in control block.

Implements rm_block_media_api_t::open.

Return values

FSP_SUCCESS	Block media for SPI framework is successfully opened.
FSP_ERR_ASSERTION	One of the input parameters or their data references may be null.
FSP_ERR_ALREADY_OPEN	The channel specified has already been opened. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::open

◆ RM_BLOCK_MEDIA_SPI_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SPI_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information.

Implements rm_block_media_api_t::infoGet.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

◆ RM_BLOCK_MEDIA_SPI_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SPI_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the Block Media SPI Flash device.

Implements rm_block_media_api_t::mediaInit.

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_SPI_Read()

fsp_err_t RM_BLOCK_MEDIA_SPI_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_block*, uint32_t const *num_blocks*)

Reads a number of blocks from spi flash memory. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::read.

Return values

FSP_SUCCESS	SPI data read successfully
FSP_ERR_ASSERTION	p_ctrl or p_dest is NULL, or num_blocks is zero
FSP_ERR_NOT_OPEN	Block Media SPI module is not yet open
FSP_ERR_INVALID_ADDRESS	Invalid address range for read operation
FSP_ERR_NOT_INITIALIZED	Block Media SPI module is not yet initialized

◆ RM_BLOCK_MEDIA_SPI_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SPI_StatusGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status.

Implements rm_block_media_api_t::statusGet.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::statusGet

◆ RM_BLOCK_MEDIA_SPI_Write()

fsp_err_t RM_BLOCK_MEDIA_SPI_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_block*, uint32_t const *num_blocks*)

Writes provided data to a number of blocks of spi flash memory. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::write.

Return values

FSP_SUCCESS	Flash write finished successfully.
FSP_ERR_ASSERTION	p_ctrl or p_src is NULL. Or num_blocks is zero.
FSP_ERR_NOT_OPEN	Block media SPI Framework module is not yet initialized.
FSP_ERR_INVALID_ADDRESS	Invalid address range
FSP_ERR_NOT_INITIALIZED	Block Media SPI module is not yet initialized

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::write

◆ RM_BLOCK_MEDIA_SPI_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_SPI_CallbackSet (rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. API not supported.

Implements rm_block_media_api_t::callbackSet.

Return values

FSP_ERR_UNSUPPORTED	API not supported by RM_BLOCK_MEDIA_SPI.

◆ RM_BLOCK_MEDIA_SPI_Close()

fsp_err_t RM_BLOCK_MEDIA_SPI_Close (rm_block_media_ctrl_t *const *p_ctrl*)

Closes the Block Media SPI device. Implements rm_block_media_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	One of the following parameters may be null: p_ctrl.
FSP_ERR_NOT_OPEN	Block media SPI Framework module is not yet initialized.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::close

◆ RM_BLOCK_MEDIA_SPI_Erase()

fsp_err_t RM_BLOCK_MEDIA_SPI_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *start_block*, uint32_t const *num_blocks*)

This function erases blocks of the SPI device. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::erase.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_INVALID_ADDRESS	Invalid address range

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- spi_flash_api_t::erase
- spi_flash_api_t::statusGet

5.2.17.8 Block Media USB (rm_block_media_usb)

Modules » Storage

Functions

fsp_err_t	RM_BLOCK_MEDIA_USB_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_USB_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_USB_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_USB_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_USB_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_USB_CallbackSet (rm_block_media_ctrl_t *const

	p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t	RM_BLOCK_MEDIA_USB_StatusGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_USB_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_USB_Close (rm_block_media_ctrl_t *const p_ctrl)

Detailed Description

Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface.

Overview

Features

The USB implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from a USB mass storage device
- Callback called when device insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

Configuration

Build Time Configurations for rm_block_media_usb

The following build time configurations are defined in driver/rm_block_media_usb_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > Block Media USB (rm_block_media_usb)

This module can be added to the Stacks tab via New Stack > Storage > Block Media USB (rm_block_media_usb).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_block_media0	Module name.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this

			callback function is provided, it will be called when a device is attached or removed.
Pointer to user context	Name must be a valid C symbol	NULL	A user context can be provided. If this context is provided, it will be passed to callback function when a device is attached or removed.

Note

RM_BLOCK_MEDIA_USB_MediaInit function must be called after receiving the insert event notification.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

This is a basic example of minimal use of the USB mass storage block media implementation in an application.

```
#define RM_BLOCK_MEDIA_USB_BLOCK_SIZE (512)
volatile bool g_usb_inserted = false;
uint8_t       g_dest[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t       g_src[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_usb_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_USB_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_USB driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_USB_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 while (!g_usb_inserted)
    {
 /* Wait for a card insertion interrupt. */
    }
 /* Initialize the mass storage device. This should not be done until the device is
plugged in and initialized. */
    err = RM_BLOCK_MEDIA_USB_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

Device Insertion

This is an example of using a callback to determine when a mass storage device is plugged in and enumerated.

```
/* The callback is called when a media insertion event occurs. */
void rm_block_media_usb_media_insertion_example_callback
(rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED == p_args->event)
    {
       g_usb_inserted = true;
    }
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED == p_args->event)
    {
       g_usb_inserted = false;
    }
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_USB_Open()

fsp_err_t RM_BLOCK_MEDIA_USB_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Opens the module.

Implements rm_block_media_api_t::open().

Return values

FSP_SUCCESS	Module is available and is now open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Module has already been opened with this instance of the control structure.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_USB_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the USB device.

Implements rm_block_media_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_USB_Read()

fsp_err_t RM_BLOCK_MEDIA_USB_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Reads data from an USB device. Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_USB_FAILED	The message could not received completed successfully.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_Write()

fsp_err_t RM_BLOCK_MEDIA_USB_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Writes data to an USB device. Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

Return values

FSP_SUCCESS	Write finished successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_USB_FAILED	The message could not received completed successfully.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_Erase()

fsp_err_t RM_BLOCK_MEDIA_USB_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Erases sectors of an USB device. Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_USB_FAILED	The message could not received completed successfully.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_USB_CallbackSet (rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements rm_block_media_api_t::callbackSet.

Note

This function is currently unsupported for Block Media over USB.

Return values

FSP_SUCCESS	Successful CallbackSet.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_USB_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_USB_StatusGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status. Implements rm_block_media_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_USB_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information. Implements rm_block_media_api_t::infoGet().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

◆ RM_BLOCK_MEDIA_USB_Close()

fsp_err_t RM_BLOCK_MEDIA_USB_Close (rm_block_media_ctrl_t *const *p_ctrl*)

Closes an open USB device. Implements rm_block_media_api_t::close().

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

5.2.17.9 Block Media Custom Implementation (rm_block_media_user)

Modules » Storage

Middleware that implements a block media interface on the media of your choice. This module implements the Block Media Interface.

Overview

Features

This module is for using Block media with user-selected media.

Configuration

Block Media User has no output config settings.
The user is required to create the config settings etc. in the application.
The figure below is an example of the config definition when the user media in USB PMSC is RAM.

```
const rm_block_media_api_t g_rm_block_media_on_user_media =
{
    .open      = RM_BLOCK_MEDIA_RAM_Open,
    .mediaInit = RM_BLOCK_MEDIA_RAM_MediaInit,
    .read      = RM_BLOCK_MEDIA_RAM_Read,
    .write    = RM_BLOCK_MEDIA_RAM_Write,
    .erase    = RM_BLOCK_MEDIA_RAM_Erase,
    .infoGet   = RM_BLOCK_MEDIA_RAM_InfoGet,
    .statusGet = RM_BLOCK_MEDIA_RAM_StatusGet,
    .close    = RM_BLOCK_MEDIA_RAM_Close,
};
extern void r_usb_pmsc_block_media_event_callback(rm_block_media_callback_args_t *
p_args);
const rm_block_media_cfg_t g_rm_block_media0_cfg =
{.p_extend = NULL, .p_callback = r_usb_pmsc_block_media_event_callback, .p_context =
NULL, };
rm_block_media_instance_t g_rm_block_media0 =
{.p_api = &g_rm_block_media_on_user_media, .p_ctrl = NULL, .p_cfg =
&g_rm_block_media0_cfg, };
```

Note

If you use block_media_user, you need to create a function that matches the media you are using.
In the above example, this is the function with RM_BLOCK_MEDIA_.
Register the created function in rm_block_media_api_t.
The registered rm_block_media_api_t is registered in p_api, which is a member of rm_block_media_instance_t.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

Please refer to USB PMSC (r_usb_pmsc) for the PMSC application given as an example.

5.2.17.10 FreeRTOS+FAT Port for RX (rm_freertos_plus_fat)

Modules » Storage

Functions

fsp_err_t	RM_FREERTOS_PLUS_FAT_Open (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)
fsp_err_t	RM_FREERTOS_PLUS_FAT_MediaInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)
fsp_err_t	RM_FREERTOS_PLUS_FAT_DiskInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)
fsp_err_t	RM_FREERTOS_PLUS_FAT_DiskDeinit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)
fsp_err_t	RM_FREERTOS_PLUS_FAT_InfoGet (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)
fsp_err_t	RM_FREERTOS_PLUS_FAT_Close (rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Detailed Description

Middleware for the FAT File System control on RX MCUs.

Overview

This module provides the hardware port layer for FreeRTOS+FAT file system. After initializing this module, refer to the FreeRTOS+FAT API reference to use the file system: https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_FAT/index.html

Features

The FreeRTOS+FAT port module supports the following features:

- Callbacks for insertion and removal for removable devices.
- Helper function to initialize FF_Disk_t
- Blocking read and write port functions that use FreeRTOS task notification to pend if FreeRTOS is used
- FreeRTOS is optional

Configuration

Build Time Configurations for rm_freertos_plus_fat

The following build time configurations are defined in fsp_cfg/middleware/rm_freertos_plus_fat_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > FreeRTOS+FAT Port for RX (rm_freertos_plus_fat)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_freertos_plus_fat0	Module name.
Total Number of Sectors	Must be a non-negative integer	31293440	Enter the total number of sectors on the device. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit().
Sector Size (bytes)	Must be a power of 2 multiple of 512	512	Select the sector size. Must match the underlying media sector size and at least 512. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after

			calling RM_FREERTOS_PLUS_FAT_MediaInit().
Cache Size (bytes)	Must be a power of 2 multiple of 512	1024	Select the cache size. Must be a multiple of the sector size and at least 2 times the sector size.
Partition Number	Must be a non-negative integer	0	Select the partition number for this disk.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed.

Usage Notes

Pending during Read/Write

If the underlying driver supports non-blocking operations, the FreeRTOS+FAT port pends the active FreeRTOS task during read and write operations so other tasks can run in the background.

If FreeRTOS is not used, the FreeRTOS+FAT port spins in a while loop waiting for read and write operations to complete.

FreeRTOS+FAT without FreeRTOS

To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.

Also, update the Malloc function to malloc and the Free function to free in the Common configurations.

Examples

Basic Example

This is a basic example of FreeRTOS+FAT in an application.

```
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
extern rm_freertos_plus_fat_instance_ctrl_t g_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_freertos_plus_fat0_cfg;
extern rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat_disk_cfg;
```

```
extern uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_freertos_plus_fat_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
```

```
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
}
```

Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
void rm_freertos_plus_fat_format_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
```

```
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Try to mount the disk. If the disk is not formatted, mount will fail. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 if (FF_isERR((uint32_t) ff_err))
    {
 /* The disk is likely not formatted. Partition and format the disk, then mount
again. */
       FF_PartitionParameters_t partition_params;
       partition_params.ulSectorCount   =
g_rm_freertos_plus_fat_disk_cfg.device.sector_count;
       partition_params.ulHiddenSectors = 1;
       partition_params.ulInterSpace    = 0;
       memset(partition_params.xSizes, 0, sizeof(partition_params.xSizes));
       partition_params.xSizes[RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER] =
            (BaseType_t) partition_params.ulSectorCount - 1;
       partition_params.xPrimaryCount = 1;
       partition_params.eSizeType    = eSizeIsSectors;
       ff_err = FF_Partition(&disk, &partition_params);
      handle_ff_error(ff_err);
       ff_err = FF_Format(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER,
pdFALSE, pdFALSE);
      handle_ff_error(ff_err);
       ff_err = FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
      handle_ff_error(ff_err);
    }
}
```

Media Insertion Example

This shows how to use the callback to wait for media insertion.

```
#if 2 == BSP_CFG_RTOS
```

```
static EventGroupHandle_t xUSBEventGroupHandle = NULL;
#else
volatile uint32_t g_rm_freertos_plus_fat_insertion_events = 0;
volatile uint32_t g_rm_freertos_plus_fat_removal_events = 0;
#endif
/* Callback called by media driver when a removable device is inserted or removed. */
void rm_freertos_plus_fat_test_callback (rm_freertos_plus_fat_callback_args_t *
p_args)
{
#if 2 == BSP_CFG_RTOS
 /* Post an event if FreeRTOS is available. */
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xEventGroupSetBitsFromISR(xUSBEventGroupHandle, p_args->event,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
#else
 /* If FreeRTOS is not used, set a global flag. */
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED)
    {
       g_rm_freertos_plus_fat_insertion_events++;
    }
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED)
    {
       g_rm_freertos_plus_fat_removal_events++;
    }
#endif
}
void rm_freertos_plus_fat_media_insertion_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
    TEST_ASSERT_NOT_EQUAL(NULL, xUSBEventGroupHandle);
#endif
```

```
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for media insertion. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
                                                       pdTRUE,
                                                       pdFALSE,
                                                       portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
          (RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
 /* Wait for media insertion. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
}
```

Media Insertion Example for USB

This shows how to use the callback to read and write to USB media.

```
void rm_freertos_plus_fat_usb_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
#endif
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for the USB media to be attached. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
                                                       pdTRUE,
                                                       pdFALSE,
                                                       portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
          (RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
 /* Wait for the USB media to be attached. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
```

```
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
}
```

Data Structures

struct	rm_freertos_plus_fat_instance_ctrl_t

Data Structure Documentation

◆ rm_freertos_plus_fat_instance_ctrl_t

struct rm_freertos_plus_fat_instance_ctrl_t

FreeRTOS plus FAT private control block. DO NOT MODIFY. Initialization occurs when RM_FREERTOS_PLUS_FAT_Open is called.

Function Documentation

◆ RM_FREERTOS_PLUS_FAT_Open()

fsp_err_t RM_FREERTOS_PLUS_FAT_Open (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_cfg_t const *const *p_cfg*)

Initializes lower layer media device.

Implements rm_freertos_plus_fat_api_t::open().

Return values

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_OUT_OF_MEMORY	Not enough memory to create semaphore.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::open

◆ RM_FREERTOS_PLUS_FAT_MediaInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_MediaInit (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_device_t *const *p_device*)

Initializes the media device. This function blocks until all identification and configuration commands are complete.

Implements rm_freertos_plus_fat_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::mediaInit
- rm_block_media_api_t::infoGet

◆ RM_FREERTOS_PLUS_FAT_DiskInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskInit (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_disk_cfg_t const *const *p_disk_cfg*, FF_Disk_t *const *p_disk*)

Initializes a FreeRTOS+FAT disk structure. This function calls FF_CreateIOManger.

Implements rm_freertos_plus_fat_api_t::diskInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module has not been initialized.
FSP_ERR_INTERNAL	Call to FF_CreateIOManger failed.

◆ RM_FREERTOS_PLUS_FAT_DiskDeinit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskDeinit (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk*)

Deinitializes a FreeRTOS+FAT disk structure. This function calls FF_DeleteIOManger.

Implements rm_freertos_plus_fat_api_t::diskDeinit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module has not been initialized.

◆ RM_FREERTOS_PLUS_FAT_InfoGet()

fsp_err_t RM_FREERTOS_PLUS_FAT_InfoGet (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk*, rm_freertos_plus_fat_info_t *const *p_info*)

Get partition information. This function can only be called after rm_freertos_plus_fat_api_t::diskInit() .

Implements rm_freertos_plus_fat_api_t::infoGet().

Return values

FSP_SUCCESS	Information stored in p_info.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.
FSP_ERR_NOT_FOUND	The value of p_iomanager is NULL.

◆ RM_FREERTOS_PLUS_FAT_Close()

fsp_err_t RM_FREERTOS_PLUS_FAT_Close (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*)

Closes media device.

Implements rm_freertos_plus_fat_api_t::close().

Return values

FSP_SUCCESS	Media device closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::close

5.2.17.11 LittleFS on Flash (rm_littlefs_flash)

Modules » Storage

Functions

fsp_err_t	RM_LITTLEFS_FLASH_Open (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)
fsp_err_t	RM_LITTLEFS_FLASH_Close (rm_littlefs_ctrl_t *const p_ctrl)

Detailed Description

Middleware for the LittleFS File System control on RX MCUs.

Overview

This module provides the hardware port layer for the LittleFS file system. After initializing this module, refer to the LittleFS documentation to use the file system: https://github.com/ARMmbed/littlefs

Configuration

Build Time Configurations for rm_littlefs_flash

The following build time configurations are defined in fsp_cfg/rm_littlefs_flash_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > LittleFS on Flash (rm_littlefs_flash)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_littlefs0	Module name.
Read Size	Must be a non-negative integer	1	Minimum size of a block read. All read operations will be a multiple of this value.
Program Size	Must be a non-negative integer	4	Minimum size of a block program. All program operations will be a multiple of this value.
Block Size (bytes)	Must be a multiple of 64	128	Size of an erasable block. This does not impact RAM consumption and may be larger than the physical erase size. However, non-inlined files take up at minimum one block. Must be a multiple of the read and program sizes.
Block Count	Manual Entry	(BSP_DATA_FLASH_SIZE_BYTES/128)	Number of erasable blocks on the device.
Block Cycles	Must be an integer	1024	Number of erase cycles before LittleFS evicts metadata logs and moves the metadata to another block. Suggested values are in the range 100-1000, with large values having better performance at the cost of less consistent wear distribution. Set to -1 to disable block-level wear-leveling.
Cache Size	Must be a non-negative integer	64	Size of block caches. Each cache buffers a portion of a block in RAM. The LittleFS needs a read cache, a

			program cache, and one additional cache per file. Larger caches can improve performance by storing more data and reducing the number of disk accesses. Must be a multiple of the read and program sizes, and a factor of the block size.
Lookahead Size	Must be a non-negative multiple of 8	16	Size of the lookahead buffer in bytes. A larger lookahead buffer increases the number of blocks found during an allocation pass. The lookahead buffer is stored as a compact bitmap, so each byte of RAM can track 8 blocks. Must be a multiple of 8.

Common LittleFS Configuration

Build Time Configurations for LittleFS

The following build time configurations are defined in arm/littlefs/lfs_util.h:

Configuration	Options	Default	Description
Custom lfs_util.h	Manual Entry		Add a path to your custom lfs_util.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
Thread Safe	• Enabled • Disabled	Disabled	Enables thread safety in LittleFS.
Read Only	• Enabled • Disabled	Disabled	Enables Read Only mode in LittleFS.
Use Malloc	• Enabled • Disabled	Enabled	Configures the use of malloc by LittleFS.
Use Assert	• Enabled • Disabled	Enabled	Configures the use of assert by LittleFS.
Debug Messages	• Enabled • Disabled	Disabled	Configures debug messages.

Warning Messages	• Enabled • Disabled	Disabled	Configures warning messages.
Error Messages	• Enabled • Disabled	Disabled	Configures error messages.
Trace Messages	• Enabled • Disabled	Disabled	Configures trace messages.
Intrinsics	• Enabled • Disabled	Enabled	Configures intrinsic functions such as __builtin_clz.
Instance Name for STDIO wrapper	Name must be a valid C symbol	g_rm_littlefs0	The rm_littlefs instance name to use with the STDIO wrapper.

Usage Notes

Blocking Read/Write/Erase

The LittleFS port blocks on Read/Write/Erase calls until the operation has completed.

Memory Constraints

The block size defined in the LittleFS configuration must be a multiple of the data flash erase size of the MCU. It must be greater than 104 bytes which is the minimum block size of a LittleFS block. For information about data flash erase sizes refer to the "Specifications of the code flash memory and data flash memory" table of the "Flash Memory" chapter's "Overview" section.

CCRX low-level I/O

Some low-level I/O are not supported in CCRX; therefore, add user implementation codes to write(), abort(), and exit(). Default dummy definition of these functions have been defined in bsp_common.c

Examples

Basic Example

This is a basic example of LittleFS on Flash in an application.

```
extern const rm_littlefs_cfg_t g_rm_littlefs_flash0_cfg;
#ifdef LFS_NO_MALLOC
static uint8_t g_file_buffer[LFS_CACHE_SIZE];
static struct lfs_file_config g_file_cfg =
{
    .buffer = g_file_buffer
};
#endif
void rm_littlefs_example (void)
```

```
{
    uint8_t    buffer[30];
    lfs_file_t file;
 /* Open LittleFS Flash port.*/
 fsp_err_t err = RM_LITTLEFS_FLASH_Open(&g_rm_littlefs_flash0_ctrl,
&g_rm_littlefs_flash0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the filesystem. */
 int lfs_err = lfs_format(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Mount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Create a breakfast directory. */
    lfs_err = lfs_mkdir(&g_rm_littlefs_flash0_lfs, "breakfast");
    handle_lfs_error(lfs_err);
 /* Create a file toast in the breakfast directory. */
 const char * path = "breakfast/toast";
#ifdef LFS_NO_MALLOC
 /*************************************************************************************
************************************
  * By default LittleFS uses malloc to allocate buffers. This can be disabled in the
FSP Configuration editor.
  * Buffers will be generated from the configuration for the read, program and
lookahead buffers.
  * When opening a file a unique buffer must be passed in for use as a file buffer.
  * The buffer size must be equal to the cache size.
  *************************************************************************************
************************************/
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs,
                               &file,
                               path,
                               LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND,
```

```
                                &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_WRONLY |
LFS_O_CREAT | LFS_O_APPEND);
    handle_lfs_error(lfs_err);
#endif
 const char * contents = "butter";
    lfs_size_t   len      = strlen(contents);
 /* Apply butter to toast 10 times. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_write(&g_rm_littlefs_flash0_lfs, &file, contents, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
 /* Unmount the filesystem. */
    lfs_err = lfs_unmount(&g_rm_littlefs_flash0_lfs);
    handle_lfs_error(lfs_err);
 /* Remount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Open breakfast/toast. */
#ifdef LFS_NO_MALLOC
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY,
&g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY);
```

```
    handle_lfs_error(lfs_err);
#endif
    handle_lfs_error(lfs_err);
 /* Verify the toast is buttered the correct amount. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_read(&g_rm_littlefs_flash0_lfs, &file, buffer, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
       assert(0 == memcmp(buffer, contents, len));
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
}
```

Function Documentation

◆ RM_LITTLEFS_FLASH_Open()

fsp_err_t RM_LITTLEFS_FLASH_Open (rm_littlefs_ctrl_t *const *p_ctrl*, rm_littlefs_cfg_t const *const *p_cfg*)

Opens the driver and initializes lower layer driver.

Implements rm_littlefs_api_t::open().

Return values

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_INVALID_SIZE	The provided block size is invalid.
FSP_ERR_INVALID_ARGUMENT	Flash BGO mode must be disabled.
FSP_ERR_INTERNAL	Failed to create the semaphore.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- flash_api_t::open

◆ RM_LITTLEFS_FLASH_Close()

fsp_err_t RM_LITTLEFS_FLASH_Close (rm_littlefs_ctrl_t *const *p_ctrl*)

Closes the lower level driver.

Implements rm_littlefs_api_t::close().

Return values

FSP_SUCCESS	Media device closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- flash_api_t::close

5.2.17.12 LittleFS on SPI Flash (rm_littlefs_spi_flash)

Modules » Storage

Functions

fsp_err_t	RM_LITTLEFS_SPI_FLASH_Open (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)
fsp_err_t	RM_LITTLEFS_SPI_FLASH_Close (rm_littlefs_ctrl_t *const p_ctrl)

Detailed Description

Middleware for the LittleFS File System control on RX MCUs using external SPI Flash.

Overview

This module provides the hardware port layer for the LittleFS file system. After initializing this module, refer to the LittleFS documentation to use the file system:
https://github.com/ARMmbed/littlefs

Configuration

Build Time Configurations for rm_littlefs_spi_flash

The following build time configurations are defined in fsp_cfg/rm_littlefs_spi_flash_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > LittleFS on SPI Flash (rm_littlefs_spi_flash)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_littlefs0	Module name.
Delay Callback	Name must be a valid C symbol	NULL	Optional delay callback for custom delay while waiting for the flash erase/write operation to complete.
Read Size	Must be a non-negative integer	1	Minimum size of a block read. All read operations will be a multiple of this value.
Program Size	Must be a non-negative integer	4	Minimum size of a block program. All program operations will be a multiple of this value.
Block Size (bytes)	Must be a multiple of 64	4096	Size of an erasable block. This does not impact RAM

			consumption and may be larger than the physical erase size. However, non-inlined files take up at minimum one block. Must be a multiple of the read and program sizes.
Block Cycles	Must be an integer	1024	Number of erase cycles before LittleFS evicts metadata logs and moves the metadata to another block. Suggested values are in the range 100-1000, with large values having better performance at the cost of less consistent wear distribution. Set to -1 to disable block-level wear-leveling.
Cache Size	Must be a non-negative integer	64	Size of block caches. Each cache buffers a portion of a block in RAM. The LittleFS needs a read cache, a program cache, and one additional cache per file. Larger caches can improve performance by storing more data and reducing the number of disk accesses. Must be a multiple of the read and program sizes, and a factor of the block size.
Lookahead Size	Must be a non-negative multiple of 8	16	Size of the lookahead buffer in bytes. A larger lookahead buffer increases the number of blocks found during an allocation pass. The lookahead buffer is stored as a compact bitmap, so each byte of RAM can track 8 blocks. Must be a multiple of 8.

Memory Start Address Offset (bytes)	Offset should be greater than or equal to zero	0	Enter the starting offset to use in the SPI memory. The starting address for LittleFS memory will be the SPI memory base address plus this offset.
Memory Size (bytes)	Size should be greater than zero	33554432	Enter the size that the LittleFS Memory should be. This can be smaller than the SPI memory size in order to use a subset of SPI memory.
Poll Status Count	Poll Status Count should be greater than or equal to zero	0xFFFFFFFF	Number of times to poll for operation complete status for blocking memory operations.

Common LittleFS Configuration

Build Time Configurations for LittleFS

The following build time configurations are defined in arm/littlefs/lfs_util.h:

Configuration	Options	Default	Description
Custom lfs_util.h	Manual Entry		Add a path to your custom lfs_util.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
Thread Safe	• Enabled • Disabled	Disabled	Enables thread safety in LittleFS.
Read Only	• Enabled • Disabled	Disabled	Enables Read Only mode in LittleFS.
Use Malloc	• Enabled • Disabled	Enabled	Configures the use of malloc by LittleFS.
Use Assert	• Enabled • Disabled	Enabled	Configures the use of assert by LittleFS.
Debug Messages	• Enabled • Disabled	Disabled	Configures debug messages.
Warning Messages	• Enabled • Disabled	Disabled	Configures warning messages.
Error Messages	• Enabled • Disabled	Disabled	Configures error messages.
Trace Messages	• Enabled	Disabled	Configures trace

	• Disabled		messages.
Intrinsics	• Enabled • Disabled	Enabled	Configures intrinsic functions such as __builtin_clz.
Instance Name for STDIO wrapper	Name must be a valid C symbol	g_rm_littlefs0	The rm_littlefs instance name to use with the STDIO wrapper.

Usage Notes

Blocking Read/Write/Erase

The LittleFS port blocks on Read/Write/Erase calls until the operation has completed.

Memory Constraints

The block size defined in the LittleFS configuration must be a multiple of the sector erase size of the SPI Flash. It must be greater than 104 bytes which is the minimum block size of a LittleFS block.

CCRX low-level I/O

Some low-level I/O are not supported in CCRX; therefore, add user implementation codes to write(), abort(), and exit(). Default dummy definition of these functions have been defined in bsp_common.c

Examples

Basic Example

This is a basic example of LittleFS on Flash in an application.

```
extern const rm_littlefs_cfg_t g_rm_littlefs_spi_flash0_cfg;
#ifdef LFS_NO_MALLOC
static uint8_t g_file_buffer[LFS_CACHE_SIZE];
static struct lfs_file_config g_file_cfg =
{
    .buffer = g_file_buffer
};
#endif
void rm_littlefs_example (void)
{
    uint8_t    buffer[30];
    lfs_file_t file;
 /* Open LittleFS Flash port.*/
 fsp_err_t err = RM_LITTLEFS_SPI_FLASH_Open(&g_rm_littlefs_spi_flash0_ctrl,
```

```
&g_rm_littlefs_spi_flash0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the filesystem. */
 int lfs_err = lfs_format(&g_rm_littlefs_spi_flash0_lfs,
&g_rm_littlefs_spi_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Mount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_spi_flash0_lfs,
&g_rm_littlefs_spi_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Create a breakfast directory. */
    lfs_err = lfs_mkdir(&g_rm_littlefs_spi_flash0_lfs, "breakfast");
    handle_lfs_error(lfs_err);
 /* Create a file toast in the breakfast directory. */
 const char * path = "breakfast/toast";
#ifdef LFS_NO_MALLOC
 /*********************************************************************************
***********************************
  * By default LittleFS uses malloc to allocate buffers. This can be disabled in the
RX Configuration editor.
  * Buffers will be generated from the configuration for the read, program and
lookahead buffers.
  * When opening a file a unique buffer must be passed in for use as a file buffer.
  * The buffer size must be equal to the cache size.
  *********************************************************************************
***********************************/
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_spi_flash0_lfs,
                               &file,
                               path,
                               LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND,
                               &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
```

```
    lfs_err = lfs_file_open(&g_rm_littlefs_spi_flash0_lfs, &file, path, LFS_O_WRONLY
| LFS_O_CREAT | LFS_O_APPEND);
    handle_lfs_error(lfs_err);
#endif
 const char * contents = "butter";
    lfs_size_t   len      = strlen(contents);
 /* Apply butter to toast 10 times. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_write(&g_rm_littlefs_spi_flash0_lfs, &file, contents, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_spi_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
 /* Unmount the filesystem. */
    lfs_err = lfs_unmount(&g_rm_littlefs_spi_flash0_lfs);
    handle_lfs_error(lfs_err);
 /* Remount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_spi_flash0_lfs,
&g_rm_littlefs_spi_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Open breakfast/toast. */
#ifdef LFS_NO_MALLOC
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_spi_flash0_lfs, &file, path,
LFS_O_RDONLY, &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_spi_flash0_lfs, &file, path,
LFS_O_RDONLY);
    handle_lfs_error(lfs_err);
```

```
#endif
   handle_lfs_error(lfs_err);
 /* Verify the toast is buttered the correct amount. */
 for (uint32_t i = 0; i < 10; i++)
   {
      lfs_err = lfs_file_read(&g_rm_littlefs_spi_flash0_lfs, &file, buffer, len);
 if (lfs_err < 0)
     {
     handle_lfs_error(lfs_err);
      }
      assert(0 == memcmp(buffer, contents, len));
   }
 /* Close the file. */
   lfs_err = lfs_file_close(&g_rm_littlefs_spi_flash0_lfs, &file);
   handle_lfs_error(lfs_err);
}
```

Function Documentation

◆ RM_LITTLEFS_SPI_FLASH_Open()

fsp_err_t RM_LITTLEFS_SPI_FLASH_Open (rm_littlefs_ctrl_t *const *p_ctrl*, rm_littlefs_cfg_t const *const *p_cfg*)

Opens the driver and initializes lower layer driver.

Implements rm_littlefs_api_t::open().

Return values

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_INVALID_SIZE	The provided block size is invalid.
FSP_ERR_INVALID_ARGUMENT	Flash BGO mode must be disabled.
FSP_ERR_INTERNAL	Failed to create the semaphore.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- spi_flash_api_t::open

◆ RM_LITTLEFS_SPI_FLASH_Close()

fsp_err_t RM_LITTLEFS_SPI_FLASH_Close (rm_littlefs_ctrl_t *const *p_ctrl*)

Closes the lower level driver.

Implements rm_littlefs_api_t::close().

Return values

FSP_SUCCESS	Media device closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- spi_flash_api_t::close

5.2.18 System

Modules

Detailed Description

System Modules.

Modules

	Clock Generation Circuit (r_cgc) Driver for the CGC peripheral on RX MCUs. This module implements the CGC Interface.
	Event Link Controller (r_elc) Driver for the ELC peripheral on RX MCUs. This module implements the ELC Interface.
	I/O Ports (r_ioport) Driver for the I/O Ports peripheral on RX MCUs. This module implements the I/O Port Interface.

5.2.18.1 Clock Generation Circuit (r_cgc)

Modules » System

Functions

fsp_err_t	R_CGC_Open (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)
fsp_err_t	R_CGC_ClocksCfg (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)
fsp_err_t	R_CGC_ClockStart (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)
fsp_err_t	R_CGC_ClockStop (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t	R_CGC_ClockCheck (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t	R_CGC_SystemClockSet (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)
fsp_err_t	R_CGC_SystemClockGet (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)
fsp_err_t	R_CGC_OscStopDetectEnable (cgc_ctrl_t *const p_ctrl)
fsp_err_t	R_CGC_OscStopDetectDisable (cgc_ctrl_t *const p_ctrl)
fsp_err_t	R_CGC_OscStopStatusClear (cgc_ctrl_t *const p_ctrl)
fsp_err_t	R_CGC_CallbackSet (cgc_ctrl_t *const p_api_ctrl, void(*p_callback)(cgc_callback_args_t *), void *const p_context, cgc_callback_args_t *const p_callback_memory)
fsp_err_t	R_CGC_Close (cgc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the CGC peripheral on RX MCUs. This module implements the CGC Interface.

Note

This module is not required for the initial clock configuration. Initial clock settings are configurable on the **Clocks** tab of the RX Configuration editor. The initial clock settings are applied by the BSP during the startup process before main.

Overview

Features

The CGC module supports runtime modifications of clock settings. Key features include the following:

- Supports changing the system clock source to any of the following options (provided they are supported on the MCU):
  - High-speed on-chip oscillator (HOCO)
  - Middle-speed on-chip oscillator (MOCO)
  - Low-speed on-chip oscillator (LOCO)
  - Main oscillator (external resonator or external clock input frequency)
  - Sub-clock oscillator (external resonator)
  - PLL/PLL2 (not available on all MCUs)
- When the system core clock frequency changes, the following things are updated:
  - The CMSIS standard global variable SystemCoreClock is updated to reflect the new clock frequency.
  - Wait states for ROM and RAM are adjusted to the minimum supported value for the new clock frequency.
  - The operating power control mode is updated to the minimum supported value for the new clock settings.
- Supports starting or stopping any of the system clock sources
- Supports changing dividers for the internal clocks
- Supports the oscillation stop detection feature

Internal Clocks

The RX microcontrollers have up to seven internal clocks. Not all internal clocks exist on all MCUs. Each clock domain has its own divider that can be updated in R_CGC_SystemClockSet(). The dividers are subject to constraints described in the footnote of the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual.

The internal clocks include:

- System clock (ICLK): core clock used for CPU, flash, internal SRAM, DTC, and DMAC
- PCLKA/PCLKB/PCLKC/PCLKD: Peripheral clocks, refer to the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual to see which peripherals are controlled by which clocks.
- FCLK: Clock source for reading data flash and for programming/erasure of both code and data flash.
- BCLK: External bus clock

Configuration

Note

*The initial clock settings are configurable on the **Clocks** tab of the RX Configuration editor.*
There is a configuration to enable the HOCO on reset in the OFS1 settings on the BSP tab.
The following clock related settings are configurable in the RX Common section on the BSP tab:

- *Main Oscillator Wait Time*
- *Main Oscillator Clock Source (external oscillator or crystal/resonator)*
- *Subclock Populated*
- *Subclock Drive*
- *Subclock Stabilization Time (ms)*

The default stabilization times are determined based on development boards provided by Renesas, but are generally valid for most designs. Depending on the target board hardware configuration and requirements these values may need to be adjusted for reliability or startup speed.

Build Time Configurations for r_cgc

The following build time configurations are defined in fsp_cfg/r_cgc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for System > Clock Generation Circuit (r_cgc)

This module can be added to the Stacks tab via New Stack > System > Clock Generation Circuit (r_cgc).

Configuration	Options	Default	Description
Sub Clock Oscillation Stop Detection Settings > Enable sub clock oscillator stop detection via the SOSC_STOP interrupt	• Enabled • Disabled	Disabled	Oscillation Stop Detection for Sub Clock enabled.
Sub Clock Oscillation Stop Detection Settings > Sub Clock Oscillation Stop Detection Time	Manual Entry	0	These bits specify the oscillation stop detection time. It is detected that oscillation has stopped when oscillation has been stopped for (A-2) to (A+1) clock cycles, where A refers to the time specified by these bits. Oscillation stop detection time = Low-speed on-chip oscillator clock (LOCO) cycle x ((value of OSDCCMP) + 1).
Sub Clock Oscillation Stop Detection Settings > Sub Clock Oscillation Stop Detection Interrupt Priority	MCU Specific Options		[Optional] Select the interrupt priority for the Sub Clock Oscillation Stop Detection interrupt.
Main Oscillation Stop Detection Settings > Enable main clock oscillator stop detection via the MOSC_STOP interrupt	• Enabled • Disabled	Disabled	Oscillation Stop Detection for Main Clock enabled.
Main Oscillation Stop Detection Settings > Main Oscillation Stop Detection Time	Manual Entry	0	These bits specify the oscillation stop detection time. It is detected that oscillation has stopped

			when oscillation has been stopped for (A - 2) to (A + 1) clock cycles, where A refers to the time specified by these bits. Oscillation stop detection time = High speed on-chip oscillator clock (HOCO) cycle x ((value of OSDCCMP) + 1).
Main Oscillation Stop Detection Settings > Main Oscillation Stop Detection Interrupt Priority	MCU Specific Options		[Optional] Select the interrupt priority for the Main Oscillation Stop Detection interrupt.
Oscillator Stop Detection Settings > Clock Switch Enable for Oscillation Stop Detected of SDADCCLK	• Enabled • Disabled	Disabled	The OSTDCSE bit specifies the source clock of the 24-bit Sigma-delta A/D Converter Clock switched to HOCO when oscillation stop detected.
Name	Name must be a valid C symbol	g_cgc0	Module name.
Enable main oscillator stop detection via the NMI interrupt	• Enabled • Disabled	Enabled	Oscillation Stop Detection for System Clock enabled.
Oscillation Stop Detection Callback	Name must be a valid C symbol	NULL	A user callback function must be provided if oscillation stop detection for System Clock/Main Oscillation/Sub Clock Oscillation is used. If this callback function is provided, it is called from the interrupt service routine (ISR) if the main oscillator stops.

Clock Configuration

This module is used to configure the system clocks. There are no module specific clock configurations required to use it.

Pin Configuration

The CGC module controls the output of the CLOCKOUT signal.

If an external oscillator is used the XTAL and EXTAL pins must be configured accordingly. When running from an on chip oscillator there is no requirement for the main clock external oscillator. In this case, the XTAL and EXTAL pins can be set to a different function in the RX Configuration editor.

The functionality of the subclock external oscillator pins XCIN and XCOUT is fixed.

Usage Notes

Oscillation Stop Detection

CGC driver supports following oscillation stop detection function.

NMI Interrupt

Note:

- For RX74M/RX74N device: NMI is not supported in the CGC module

Main Oscillator Stop Detection Interrupt

A main clock oscillation stop detection interrupt (MOSTD_STOP) is generated when the Oscillation Stop Detection for Main Clock is enabled. The main oscillation stop detection interrupt is a maskable interrupt. When the MOSTD_STOP is triggered, the callback function registered during R_CGC_Open() is called.

Subclk Oscillator Stop Detection Interrupt

Note:

- For RX74M/RX74N device: Subclk Oscillation Stop Detection Interrupt is not supported in the CGC module

Starting or Stopping the Subclock

If the Subclock Populated property is set to Populated on the BSP configuration tab, then the subclock is started in the BSP startup routine. Otherwise, it is stopped in the BSP startup routine. Starting and stopping the subclock at runtime is not recommended since the stabilization requirements typically negate the negligible power savings.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take up to several seconds to stabilize. RX startup code does not wait for subclock stabilization unless the subclock is the main clock source. In this case the default wait time is 1000ms (1 second). When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation. Because there is no hardware stabilization status bit for the subclock R_CGC_ClockCheck cannot be used to optimize this wait.

Changing the subclock state during R_CGC_ClocksCfg() is not supported.

Low Power Operation

If "Use Low Voltage Mode" is enabled in the BSP MCU specific properties (not available on all MCUs), the MCU is always in low voltage mode and no other power modes are considered. The following conditions must be met for the MCU to run in low voltage mode:

- Requires HOCO to be running, so HOCO cannot be stopped in low voltage mode
- Requires PLL to be stopped, so PLL APIs are not available in low voltage mode
- Requires ICLK <= 4 MHz
- If oscillation stop detection is used, dividers of 1 or 2 cannot be used for any clock

If "Use Low Voltage Mode" is not enabled, the MCU applies the lowest power mode by searching through the following list in order and applying the first power mode that is supported under the current conditions:

- Subosc-speed mode (lowest power)
    - Requires system clock to be LOCO or subclock
    - Requires MOCO, HOCO, main oscillator, and PLL (if present) to be stopped
    - Requires ICLK and FCLK dividers to be 1
- Low-speed mode
    - Requires PLL to be stopped
    - Requires ICLK <= 1 MHz
    - If oscillation stop detection is used, dividers of 1, 2, 4, or 8 cannot be used for any clock
- Middle-speed mode (not supported on all MCUs)
    - Requires ICLK <= 8 MHz
- High-speed mode
    - Default mode if no other operating mode is supported

Refer to the section "Function for Lower Operating Power Consumption" in the "Low Power Modes" chapter of the hardware manual for MCU specific information about operating power control modes.

Note

The DCDC regulator (if present) is only available in Middle- and High-speed modes. The BSP will automatically switch between DCDC and LDO when switching between compatible and incompatible modes if the DCDC regulator is in use. Switching to the LDO incurs a 60 microsecond critical section wherein all interrupts AND peripherals are stopped. Switching back to DCDC from the LDO incurs an additional 22 microsecond critical section (peripherals running).

When low voltage mode is not used, the following functions adjust the operating power control mode to ensure it remains within the hardware specification and to ensure the MCU is running at the optimal operating power control mode:

- R_CGC_ClockStart()
- R_CGC_ClockStop()
- R_CGC_SystemClockSet()
- R_CGC_OscStopDetectEnable()
- R_CGC_OscStopDetectDisable()

Note

FSP APIs, including these APIs, are not thread safe. These APIs and any other user code that modifies the operating power control mode must not be allowed to interrupt each other. Proper care must be taken during application design if these APIs are used in threads or interrupts to ensure this constraint is met.

No action is required by the user of these APIs. This section is provided for informational purposes only.

Examples

Basic Example

This is a basic example of minimal use of the CGC in an application.

```
void cgc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Change the system clock to LOCO for power saving. */
 /* Start the LOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);
    assert(FSP_SUCCESS == err);
 /* Wait for the LOCO stabilization wait time.
  *
  * NOTE: The MOCO, LOCO and subclock do not have stabilization status bits, so any
stabilization time must be
  * performed via a software wait when starting these oscillators. For all other
oscillators, R_CGC_ClockCheck can
  * be used to verify stabilization status.
  */
 R_BSP_SoftwareDelay(BSP_FEATURE_CGC_LOCO_STABILIZATION_MAX_US,
BSP_DELAY_UNITS_MICROSECONDS);
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
#if !BSP_FEATURE_CGC_REGISTER_SET_B
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
```

```
        .sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
        .sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
        .sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_1,
        .sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_1,
        .sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_1,
        .sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_1,
#else
        .moco_divider = CGC_SYS_CLOCK_DIV_2,
        .hoco_divider = CGC_SYS_CLOCK_DIV_1,
        .mosc_divider = CGC_SYS_CLOCK_DIV_4,
#endif
    };
 /* Switch the system clock to LOCO. */
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
    assert(FSP_SUCCESS == err);
}
```

Configuring Multiple Clocks

This example demonstrates switching to a new source clock and stopping the previous source clock in a single function call using R_CGC_ClocksCfg().

```
void cgc_clocks_cfg_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Change the system clock to PLL running from the main oscillator. */
 /* Assuming the system clock is MOCO, switch to HOCO. */
 cgc_clocks_cfg_t clocks_cfg;
```

```
#if !BSP_FEATURE_CGC_REGISTER_SET_B
    clocks_cfg.system_clock         = CGC_CLOCK_PLL;
    clocks_cfg.pll_state            = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC; // unused
    clocks_cfg.pll_cfg.multiplier   = CGC_PLL_MUL_10_0;   // unused
    clocks_cfg.pll_cfg.divider      = CGC_PLL_DIV_2;      // unused
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4;
#else
    clocks_cfg.system_clock             = CGC_CLOCK_HOCO;
    clocks_cfg.divider_cfg.moco_divider = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.hoco_divider = CGC_SYS_CLOCK_DIV_2;
    clocks_cfg.divider_cfg.mosc_divider = CGC_SYS_CLOCK_DIV_1;
#endif
    clocks_cfg.mainosc_state = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.hoco_state    = CGC_CLOCK_CHANGE_START;
    clocks_cfg.moco_state    = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.loco_state    = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* Assuming the system clock is HOCO, switch to PLL running from main oscillator and
stop MOCO. */
    clocks_cfg.system_clock         = CGC_CLOCK_PLL;
    clocks_cfg.pll_state            = CGC_CLOCK_CHANGE_START;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC;
    clocks_cfg.pll_cfg.multiplier   = (cgc_pll_mul_t) BSP_CFG_PLL_MUL;
    clocks_cfg.pll_cfg.divider      = (cgc_pll_div_t) BSP_CFG_PLL_DIV;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1;
```

```
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state = CGC_CLOCK_CHANGE_START;
    clocks_cfg.hoco_state    = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.moco_state    = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.loco_state    = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
#endif
}
```

Oscillation Stop Detection

This example demonstrates registering a callback for oscillation stop detection of the main oscillator.

```
/* Example callback called when oscillation stop is detected. */
void oscillation_stop_callback (cgc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If the MCU was running on the main oscillator, the MCU is now running
on MOCO. Switch clocks if
  * desired. This example shows switching to HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for HOCO to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t dividers =
```

```
    {
#if !BSP_FEATURE_CGC_REGISTER_SET_B
        .sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4,
        .sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4,
        .sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
        .sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4,
        .sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4,
        .sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4,
        .sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4,
#else
        .moco_divider = CGC_SYS_CLOCK_DIV_2,
        .hoco_divider = CGC_SYS_CLOCK_DIV_1,
        .mosc_divider = CGC_SYS_CLOCK_DIV_4,
#endif
    };
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_HOCO, &dividers);
    assert(FSP_SUCCESS == err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* (Optional) If the MCU was running on the PLL, the PLL is now in free-running
mode. Switch clocks if
  * desired. This example shows switching to the PLL running on HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for HOCO to stabilize. */
        err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_pll_cfg_t pll_cfg =
    {
        .source_clock = CGC_CLOCK_HOCO,
        .multiplier   = (cgc_pll_mul_t) BSP_CFG_PLL_MUL,
        .divider      = (cgc_pll_div_t) BSP_CFG_PLL_DIV,
    };
```

```
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_cfg);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for PLL to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_PLL);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t pll_dividers =
    {
       .sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4,
    };
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_dividers);
    assert(FSP_SUCCESS == err);
#endif
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
}
void cgc_osc_stop_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable oscillation stop detection. The main oscillator must be running at this
```

```
point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
    __WAIT();
 /* (Optional) Reenable oscillation stop detection after waking from low power mode.
*/
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	cgc_instance_ctrl_t

Data Structure Documentation

◆ cgc_instance_ctrl_t

struct cgc_instance_ctrl_t

CGC private control block. DO NOT MODIFY. Initialization occurs when R_CGC_Open() is called.

Data Fields

void *	p_context

Field Documentation

◆ p_context

void* cgc_instance_ctrl_t::p_context

Placeholder for user data. Passed to the user callback in cgc_callback_args_t.

Function Documentation

◆ R_CGC_Open()

fsp_err_t R_CGC_Open (cgc_ctrl_t *const *p_ctrl*, cgc_cfg_t const *const *p_cfg*)

Initialize the CGC API. Implements cgc_api_t::open.

Example:

```
/* Initializes the CGC module. */
   err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
```

Return values

FSP_SUCCESS	CGC successfully initialized.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_CGC_ClocksCfg()

fsp_err_t R_CGC_ClocksCfg (cgc_ctrl_t *const *p_ctrl*, cgc_clocks_cfg_t const *const *p_clock_cfg*)

Reconfigures all main system clocks. This API can be used for any of the following purposes:

- start or stop clocks
- change the system clock source
- configure the PLL/PLL2 multiplication and division ratios when starting the PLL
- configure the division ratios when starting the HOCO/MOCO/MOSC
- change the system dividers

If the requested system clock source has a stabilization flag, this function blocks waiting for the stabilization flag of the requested system clock source to be set. If the requested system clock source was just started and it has no stabilization flag, this function blocks for the stabilization time required by the requested system clock source according to the Electrical Characteristics section of the hardware manual. If the requested system clock source has no stabilization flag and it is already running, it is assumed to be stable and this function will not block. If the requested system clock is the subclock, the subclock must be stable prior to calling this function.

The internal dividers (cgc_clocks_cfg_t::divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes of table "Specifications of the clock generation circuit for the internal clocks" in the CGC section of the relevant hardware manual.
- RA2A1: see footnotes of table "Clock generation circuit specifications for the internal clocks" in the CGC section of the relevant hardware manual.

Do not attempt to stop the requested clock source or the source of a PLL if the PLL will be running after this operation completes.

Implements cgc_api_t::clocksCfg.

Example:

```
/* Assuming the system clock is MOCO, switch to HOCO. */
cgc_clocks_cfg_t clocks_cfg;
#if !BSP_FEATURE_CGC_REGISTER_SET_B
    clocks_cfg.system_clock         = CGC_CLOCK_PLL;
    clocks_cfg.pll_state            = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC; // unused
    clocks_cfg.pll_cfg.multiplier   = CGC_PLL_MUL_10_0;   // unused
    clocks_cfg.pll_cfg.divider      = CGC_PLL_DIV_2;      // unused
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4;
```

```
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4;
#else
    clocks_cfg.system_clock             = CGC_CLOCK_HOCO;
    clocks_cfg.divider_cfg.moco_divider = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.hoco_divider = CGC_SYS_CLOCK_DIV_2;
    clocks_cfg.divider_cfg.mosc_divider = CGC_SYS_CLOCK_DIV_1;
#endif
    clocks_cfg.mainosc_state = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.hoco_state    = CGC_CLOCK_CHANGE_START;
    clocks_cfg.moco_state    = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.loco_state    = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Clock configuration applied successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_IN_USE	Attempt to stop the current system clock or the PLL source clock.
FSP_ERR_CLOCK_ACTIVE	PLL configuration cannot be changed while PLL is running.
FSP_ERR_OSC_STOP_DET_ENABLED	PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz.
FSP_ERR_NOT_STABILIZED	PLL clock source is not stable.
FSP_ERR_PLL_SRC_INACTIVE	PLL clock source is not running.
FSP_ERR_INVALID_STATE	The subclock must be running before activating HOCO with FLL.

◆ R_CGC_ClockStart()

fsp_err_t R_CGC_ClockStart (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_pll_cfg_t const *const *p_pll_cfg*)

Start the specified clock if it is not currently active. The PLL configuration cannot be changed while the PLL is running. Implements cgc_api_t::clockStart.

The PLL source clock must be operating and stable prior to starting the PLL.

Example:

```
/* Start the LOCO. */

   err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);

   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Clock initialized successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_STABILIZED	The clock source is not stabilized after being turned off or PLL clock source is not stable.
FSP_ERR_PLL_SRC_INACTIVE	PLL clock source is not running.
FSP_ERR_CLOCK_ACTIVE	PLL configuration cannot be changed while PLL is running.
FSP_ERR_OSC_STOP_DET_ENABLED	PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz.
FSP_ERR_INVALID_STATE	The subclock must be running before activating HOCO with FLL.

◆ R_CGC_ClockStop()

fsp_err_t R_CGC_ClockStop (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*)

Stop the specified clock if it is active. Implements cgc_api_t::clockStop.

Do not attempt to stop the current system clock source. Do not attempt to stop the source clock of a PLL if the PLL is running.

Return values

FSP_SUCCESS	Clock stopped successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_IN_USE	Attempt to stop the current system clock or the PLL source clock.
FSP_ERR_OSC_STOP_DET_ENABLED	Attempt to stop MOCO when Oscillation stop is enabled.
FSP_ERR_NOT_STABILIZED	Clock not stabilized after starting.

◆ R_CGC_ClockCheck()

fsp_err_t R_CGC_ClockCheck (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*)

Check the specified clock for stability. Implements cgc_api_t::clockCheck.

Return values

FSP_SUCCESS	Clock is running and stable.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_STABILIZED	Clock not stabilized.
FSP_ERR_CLOCK_INACTIVE	Clock not turned on.

◆ R_CGC_SystemClockSet()

fsp_err_t R_CGC_SystemClockSet (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_divider_cfg_t const *const *p_divider_cfg*)

Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD, BCLK, and FCLK. Implements cgc_api_t::systemClockSet.

The requested clock source must be running and stable prior to calling this function. The internal dividers are subject to constraints described in the hardware manual table "Specifications of the Clock Generation Circuit for the internal clocks".

The internal dividers (p_divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes "Specifications of the clock generation circuit for the internal clocks" in the CGC section of the relevant hardware manual.
- RA2A1: see footnotes "Clock generation circuit specifications for the internal clocks" in the CGC section of the relevant hardware manual.

This function also updates the RAM and ROM wait states, the operating power control mode, and the SystemCoreClock CMSIS global variable.

Example:

```
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
#if !BSP_FEATURE_CGC_REGISTER_SET_B
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_u.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_u.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
       .sckdivcr_u.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
       .sckdivcr_u.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_u.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_u.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_u.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_1,
#else
```

```
        .moco_divider = CGC_SYS_CLOCK_DIV_2,
        .hoco_divider = CGC_SYS_CLOCK_DIV_1,
        .mosc_divider = CGC_SYS_CLOCK_DIV_4,
#endif
    };
 /* Switch the system clock to LOCO. */
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CLOCK_INACTIVE	The specified clock source is inactive.
FSP_ERR_NOT_STABILIZED	The clock source has not stabilized

◆ R_CGC_SystemClockGet()

fsp_err_t R_CGC_SystemClockGet (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *const *p_clock_source*, cgc_divider_cfg_t *const *p_divider_cfg*)

Return the current system clock source and configuration. Implements cgc_api_t::systemClockGet.

Return values

FSP_SUCCESS	Parameters returned successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.

◆ R_CGC_OscStopDetectEnable()

fsp_err_t R_CGC_OscStopDetectEnable (cgc_ctrl_t *const *p_ctrl*)

Enable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectEnable.

The MCU will automatically switch the system clock to MOCO when a stop is detected if Main Clock is the system clock. If the system clock is the PLL, then the clock source will not be changed and the PLL free running frequency will be the system clock frequency.

Example:

```
/* Enable oscillation stop detection. The main oscillator must be running at this point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_LOW_VOLTAGE_MODE	Settings not allowed in low voltage mode.
FSP_ERR_UNSUPPORTED	Function not supported.

◆ R_CGC_OscStopDetectDisable()

fsp_err_t R_CGC_OscStopDetectDisable (cgc_ctrl_t *const *p_ctrl*)

Disable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectDisable.

Example:

```
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
    __WAIT();
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function.
FSP_ERR_UNSUPPORTED	Function not supported.

◆ R_CGC_OscStopStatusClear()

fsp_err_t R_CGC_OscStopStatusClear (cgc_ctrl_t *const *p_ctrl*)

Clear the Oscillation Stop Detection Status register. This register is not cleared automatically if the stopped clock is restarted. Implements cgc_api_t::oscStopStatusClear.

After clearing the status, oscillation stop detection is no longer enabled.

This register cannot be cleared while the main oscillator is the system clock or the PLL source clock.

Example:

```
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CLOCK_INACTIVE	Main oscillator must be running to clear the oscillation stop detection flag.
FSP_ERR_OSC_STOP_CLOCK_ACTIVE	The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit.
FSP_ERR_INVALID_HW_CONDITION	Oscillation stop status was not cleared. Check preconditions and try again.
FSP_ERR_UNSUPPORTED	Function not supported.

◆ R_CGC_CallbackSet()

fsp_err_t R_CGC_CallbackSet (cgc_ctrl_t *const *p_api_ctrl*, void(*)(cgc_callback_args_t *) *p_callback*, void *const *p_context*, cgc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements cgc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	Function not supported.

◆ R_CGC_Close()

fsp_err_t R_CGC_Close (cgc_ctrl_t *const *p_ctrl*)

Closes the CGC module. Implements cgc_api_t::close.

Return values

FSP_SUCCESS	The module is successfully closed.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.

5.2.18.2 Event Link Controller (r_elc)

Modules » System

Functions

fsp_err_t	R_ELC_Open (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)
fsp_err_t	R_ELC_Close (elc_ctrl_t *const p_ctrl)
fsp_err_t	R_ELC_SoftwareEventGenerate (elc_ctrl_t *const p_ctrl, elc_software_event_t event_number)
fsp_err_t	R_ELC_LinkSet (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)
fsp_err_t	R_ELC_LinkBreak (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

fsp_err_t	R_ELC_Enable (elc_ctrl_t *const p_ctrl)
fsp_err_t	R_ELC_Disable (elc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the ELC peripheral on RX MCUs. This module implements the ELC Interface.

Overview

The event link controller (ELC) uses the event requests generated by various peripheral modules as source signals to connect (link) them to different modules, allowing direct cooperation between the modules without central processing unit (CPU) intervention. The conceptual diagram below illustrates a potential setup where a pin interrupt triggers a timer which later triggers an ADC conversion and CTSU scan, while at the same time a serial communication interrupt automatically starts a data transfer. These tasks would be automatically handled without the need for polling or interrupt management.

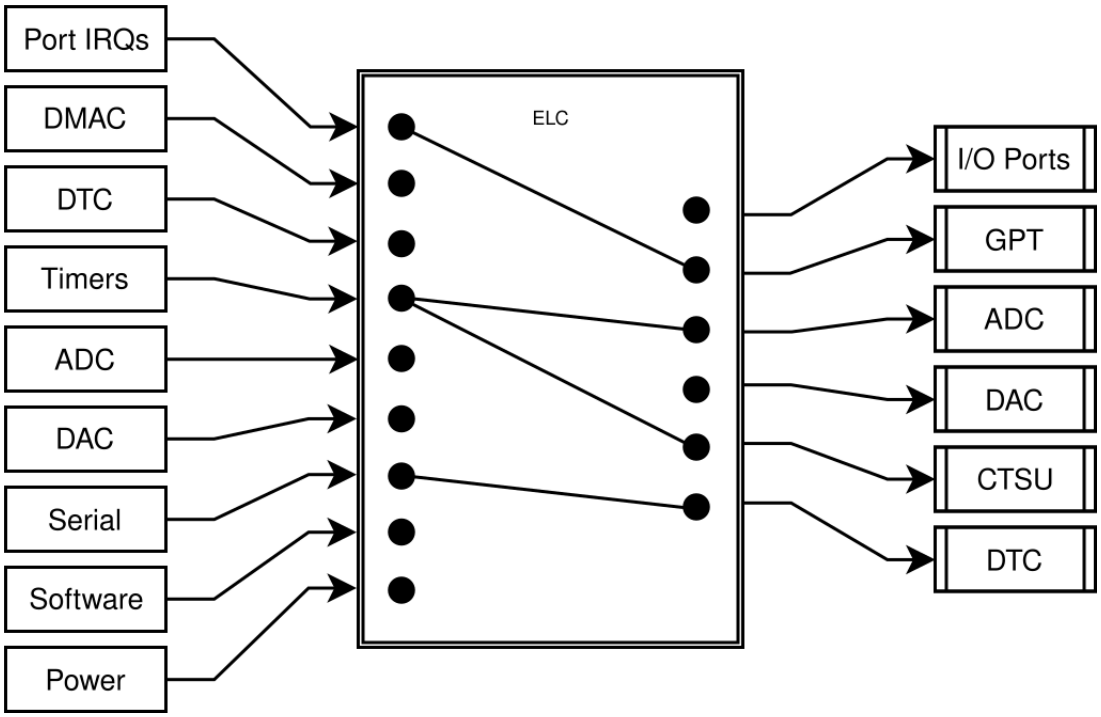


Figure 152: Event Link Controller Conceptual Diagram

In essence, the ELC is an array of multiplexers to route a wide variety of interrupt signals to a subset of peripheral functions. Events are linked by setting the multiplexer for the desired function to the desired signal (through R_ELC_LinkSet). The diagram below illustrates one peripheral output of the ELC. In this example, a conversion start is triggered for ADC0 Group A when the GPT0 counter overflows:

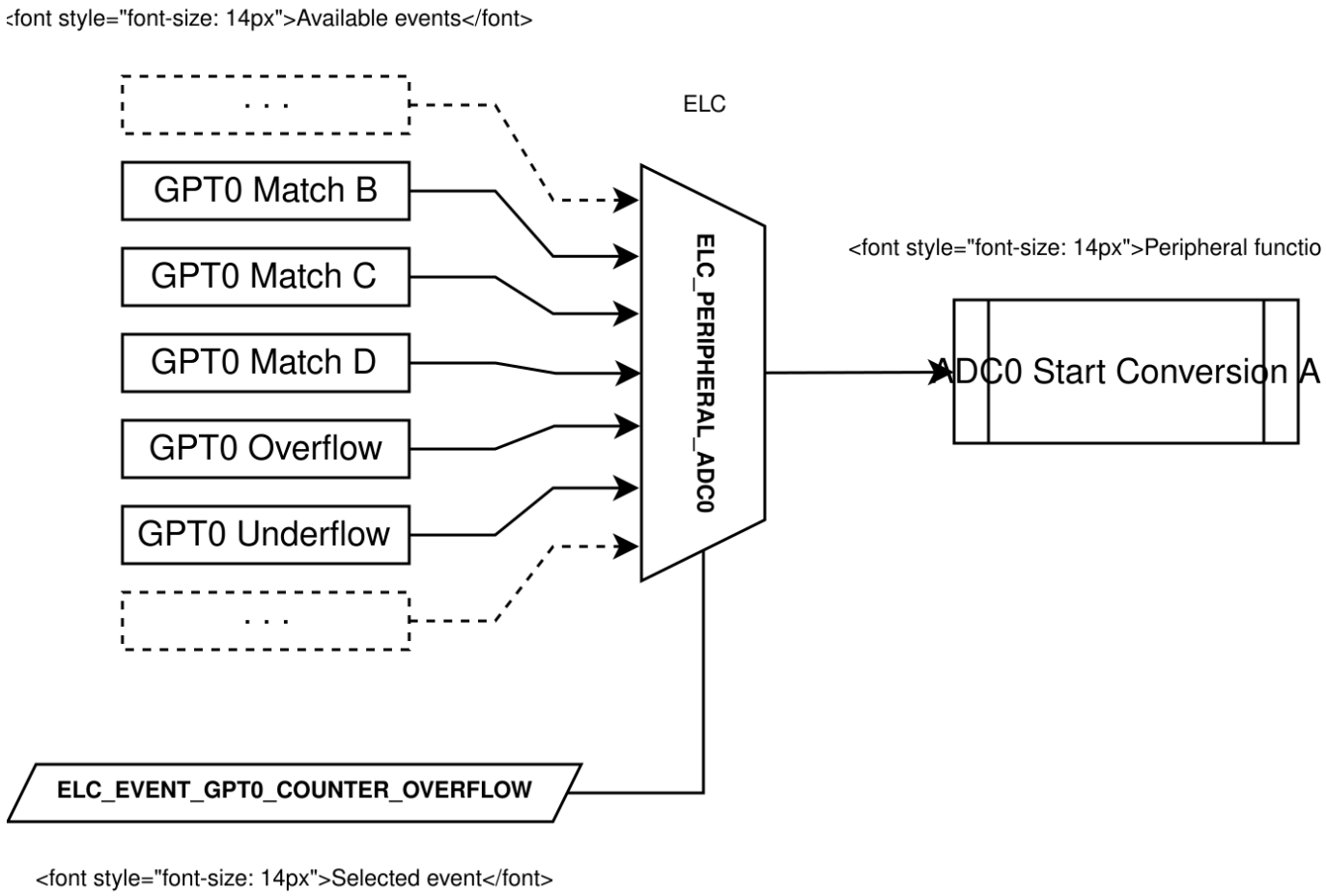


Figure 153: ELC Example

Features

The ELC HAL module can perform the following functions:

- Initialize the ELC to a pre-defined set of links
- Create an event link between two blocks
- Break an event link between two blocks
- Generate one of two software events that interrupt the CPU
- Globally enable or disable event links

A variety of functions can be activated via events, including:

- General-purpose timer (GPT) control
- ADC and DAC conversion start
- Synchronized I/O port output (ports 1-4 only)
- Capacitive touch unit (CTSU) measurement activation

Note

The available sources and peripherals may differ between devices. A full list of selectable peripherals and events is available in the User's Manual for your device.
Some peripherals have specific settings related to ELC event generation and/or reception. Details on how to enable event functionality for each peripheral are located in the usage notes for the related module(s) as well as in the User's Manual for your device.

Configuration

Note

Event links will be automatically generated based on the selections made in module properties. To view the currently linked events check the Event Links tab in the RX Configuration editor.
Calling R_ELC_Open followed by R_ELC_Enable will automatically link all events shown in the Event Links tab.

To manually link an event to a peripheral at runtime perform the following steps:

1. Configure the operation of the destination peripheral (including any configuration necessary to receive events)
2. Use R_ELC_LinkSet to set the desired event link to the peripheral
3. Use R_ELC_Enable to enable transmission of event signals
4. Configure the signaling module to output the desired event (typically an interrupt)

To disable the event, either use R_ELC_LinkBreak to clear the link for a specific event or R_ELC_Disable to globally disable event linking.

Note

The ELC module needs no pin, clocking or interrupt configuration; it is merely a mechanism to connect signals between peripherals. However, when linking I/O Ports via the ELC the relevant I/O pins need to be configured as inputs or outputs.

Build Time Configurations for r_elc

The following build time configurations are defined in fsp_cfg/r_elc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for System > Event Link Controller (r_elc)

This module can be added to the Stacks tab via New Stack > System > Event Link Controller (r_elc).

Configuration	Options	Default	Description
General > Event Link Option Settings > CMT > CMT1	MCU Specific Options		Event Link Option Setting Register CMT
General > Event Link Option Settings > CMTW > CMTW0	MCU Specific Options		Event Link Option Setting Register CMTW
General > Event Link Option Settings > TMR > TMR0	MCU Specific Options		Event Link Option Setting Register TMR
General > Event Link Option Settings > TMR > TMR1	MCU Specific Options		Event Link Option Setting Register TMR
General > Event Link	MCU Specific Options		Event Link Option

Option Settings > TMR > TMR2			Setting Register TMR
General > Event Link Option Settings > TMR > TMR3	MCU Specific Options		Event Link Option Setting Register TMR
General > Name	ELC instance name must be g_elc to match elc_cfg_t data structure created in elc_data.c	g_elc	Module name. Fixed to g_elc.
Interrupts > ELC1 Interrupt Priority	MCU Specific Options		Select the ELC interrupt 1 priority.
Interrupts > ELC2 Interrupt Priority	MCU Specific Options		Select the ELC interrupt 2 priority.
Interrupts > ELC3 Interrupt Priority	MCU Specific Options		Select the ELC interrupt 3 priority.
Interrupts > ELC4 Interrupt Priority	MCU Specific Options		Select the ELC interrupt 4 priority.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).

Usage Notes

Limitations

Developers should be aware of the following limitations when using the ELC:

- To link events it is necessary for the ELC and the related modules to be enabled. The ELC cannot operate if the related modules are in the module stop state or the MCU is in a low power consumption mode for which the module is stopped.
- If two modules are linked across clock domains there may be a 1 to 2 cycle delay between event signaling and reception. The delay timing is based on the frequency of the slowest clock.
- Software events (ELS bits = 50h to 53h) and the Data Operation Condition Met event from the DOC module (ELS bit = 313h) are the only events that can be assigned to ELSR88, ELSR89, ELSR90, and ELSR91 (ELC interrupts).
- The interrupt callback function can only be set through the Open API. If you want to use a callback function, register it in the ELC Properties section on the Stack tab of the FSP Configurator.

Examples

Basic Example

Below is a basic example of minimal use of event linking in an application.

```
/* This struct is automatically generated based on the events configured by
peripherals in the RX Configuration editor. */
static const elc_cfg_t g_elc_cfg =
{
    .link[ELC_PERIPHERAL_GPT_A]    = ELC_EVENT_IOPORT_EVENT_C,
    .link[ELC_EVENT_IOPORT_EVENT_B] = ELC_EVENT_GPT4_COUNTER_OVERFLOW
};
void elc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT4_COUNTER_OVERFLOW);
    assert(FSP_SUCCESS == err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Software-Generated Events

This example demonstrates how to use a software-generated event to signal a peripheral. This can be useful when the desired event source is not supported by the ELC hardware.

```
/* Interrupt handler for peripheral event not supported by the ELC */
void peripheral_isr (void)
{
 fsp_err_t err;
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
```

```
    assert(FSP_SUCCESS == err);
}
void elc_software_event (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Link ADC0 conversion start to software event 0. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application code here. */
    }
}
```

Data Structures

struct	elc_callback_args_t
struct	elc_cfg_extend_t
struct	elc_instance_ctrl_t

Enumerations

enum	elc_operation_option_t

Data Structure Documentation

◆ elc_callback_args_t

struct elc_callback_args_t		
Event link callback arguments definitions		
Data Fields		
elc_interrupt_t	event	ELC interrupt event.
void *	p_context	Placeholder for user data.

◆ elc_cfg_extend_t

struct elc_cfg_extend_t	
Extended configuration for ELC.	
Data Fields	
elc_operation_option_t	cmt1_md
	CMT1 Operation Select (placed at ELOPA[3:2])
elc_operation_option_t	cmtw0_md
	CMTW0 Operation Select (placed at ELOPB[1:0])
elc_operation_option_t	tmr0_md
	TMR0 Operation Select (placed at ELOPC[1:0])
elc_operation_option_t	tmr1_md
	TMR1 Operation Select (placed at ELOPC[3:2])
elc_operation_option_t	tmr2_md
	TMR2 Operation Select (placed at ELOPD[1:0])
elc_operation_option_t	tmr3_md
	TMR3 Operation Select (placed at ELOPD[3:2])
uint32_t	ipl [ELC_INTERRUPT_NUM]
	ELC interrupt priority.
IRQn_Type	irq [ELC_INTERRUPT_NUM]
	ELC interrupt IRQ number.
void(*	p_callback)(elc_callback_args_t *p_args)

	Pointer to callback function.
void *	p_context
	User defined context passed into callback function.

◆ elc_instance_ctrl_t

struct elc_instance_ctrl_t
ELC private control block. DO NOT MODIFY. Initialization occurs when R_ELC_Open() is called.

Enumeration Type Documentation

◆ elc_operation_option_t

enum elc_operation_option_t	
Event link option operation.	
Enumerator	
ELC_OPERATION_SELECT_COUNT_START	Counting is started.
ELC_OPERATION_SELECT_COUNT_RESTART	Counting is restarted.
ELC_OPERATION_SELECT_COUNTER	Event counter.
ELC_EVENT_OPERATION_SELECT_DISABLED	Event output is disabled.

Function Documentation

◆ R_ELC_Open()

fsp_err_t R_ELC_Open (elc_ctrl_t *const *p_ctrl*, elc_cfg_t const *const *p_cfg*)

Initialize all the links in the Event Link Controller. Implements elc_api_t::open

The configuration structure passed in to this function includes links for every event source included in the ELC and sets them all at once. To set or clear an individual link use R_ELC_LinkSet and R_ELC_LinkBreak respectively.

Example:

```
/* Initializes the software and sets the links defined in the control structure. */

   err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful
FSP_ERR_ASSERTION	p_ctrl or p_cfg was NULL
FSP_ERR_ALREADY_OPEN	The module is currently open

◆ R_ELC_Close()

fsp_err_t R_ELC_Close (elc_ctrl_t *const *p_ctrl*)

Globally disable ELC linking. Implements elc_api_t::close

Return values

FSP_SUCCESS	The ELC was successfully disabled
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_SoftwareEventGenerate()

fsp_err_t R_ELC_SoftwareEventGenerate (elc_ctrl_t *const *p_ctrl*, elc_software_event_t *event_number*)

Generate a software event in the Event Link Controller. Implements elc_api_t::softwareEventGenerate

Example:

```
/* Generate an event signal through software to the linked peripheral. */
   err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Initialization was successful
FSP_ERR_ASSERTION	Invalid event number or p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_LinkSet()

fsp_err_t R_ELC_LinkSet (elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral*, elc_event_t *signal*)

Create a single event link. Implements elc_api_t::linkSet

Example:

```
/* Create or modify a link between a peripheral function and an event source. */
   err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT4_COUNTER_OVERFLOW);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Initialization was successful
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_LinkBreak()

fsp_err_t R_ELC_LinkBreak (elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral*)

Break an event link. Implements elc_api_t::linkBreak

Return values

FSP_SUCCESS	Event link broken
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_Enable()

fsp_err_t R_ELC_Enable (elc_ctrl_t *const *p_ctrl*)

Enable the operation of the Event Link Controller. Implements elc_api_t::enable

Return values

FSP_SUCCESS	ELC enabled.
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_Disable()

fsp_err_t R_ELC_Disable (elc_ctrl_t *const *p_ctrl*)

Disable the operation of the Event Link Controller. Implements elc_api_t::disable

Return values

FSP_SUCCESS	ELC disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

5.2.18.3 I/O Ports (r_ioport)

Modules » System

Functions

fsp_err_t R_IOPORT_Open (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

fsp_err_t	R_IOPORT_Close (ioport_ctrl_t *const p_ctrl)
fsp_err_t	R_IOPORT_PinsCfg (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t	R_IOPORT_PinCfg (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)
fsp_err_t	R_IOPORT_PinEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)
fsp_err_t	R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)
fsp_err_t	R_IOPORT_PinRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)
fsp_err_t	R_IOPORT_PinWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)
fsp_err_t	R_IOPORT_PortDirectionSet (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)
fsp_err_t	R_IOPORT_PortEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *event_data)
fsp_err_t	R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
fsp_err_t	R_IOPORT_PortRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)
fsp_err_t	R_IOPORT_PortWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Detailed Description

Driver for the I/O Ports peripheral on RX MCUs. This module implements the I/O Port Interface.

Overview

The I/O port pins operate as general I/O port pins, I/O pins for peripheral modules, interrupt input pins, analog I/O, port group function for the ELC, or bus control pins.

Features

The IOPORT HAL module can configure the following pin settings:

- Pin direction
- Default output state
- Pull-up
- NMOS
- Drive strength
- Event edge trigger (falling, rising or both)
- Whether the pin is to be used as an IRQ pin
- Whether the pin is to be used as an analog pin
- Peripheral connection

The module also provides the following functionality:

- Reads/writes GPIO pins/ports
- Sets event output data
- Reads event input data

Configuration

The I/O PORT HAL module must be configured by the user for the desired operation. The operating state of an I/O pin can be set via the FSP Configuraton tool. When the project is built a pin configuration file is created. The BSP will automatically configure the MCU IO ports accordingly at startup using the same API functions mentioned in this document.

Build Time Configurations for r_ioport

The following build time configurations are defined in fsp_cfg/r_ioport_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for System > I/O Port (r_ioport)

This module can be added to the Stacks tab via New Stack > System > I/O Port (r_ioport).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_ioport	Module name.
1st Port ELC Trigger Source	MCU Specific Options		ELC source that will trigger the 1st port
2nd Port ELC Trigger Source	MCU Specific Options		ELC source that will trigger the 2nd port
3rd Port ELC Trigger Source	MCU Specific Options		ELC source that will trigger the 3rd port
4th Port ELC Trigger Source	MCU Specific Options		ELC source that will trigger the 4th port
Pin Configuration Name	Name must be a valid	g_bsp_pin_cfg	Name for pin

C symbol	configuration structure

Clock Configuration

The I/O PORT HAL module does not require a specific clock configuration.

Pin Configuration

The IOPORT module is used for configuring pins.

Usage Notes

Port Group Function for ELC

Depending on pin configuration, the IOPORT module can perform automatic reads and writes on 4 ports on receipt of an ELC event.

ELC triggers	RX74M, RX74N
1st Port	Port B
2nd Port	Port C
3rd Port	Port D
4th Port	Port E

When an event is received by a port, the state of the input pins on the port is saved in a hardware register. Simultaneously, the state of output pins on the port is set or cleared based on settings configured by the user. The functions R_IOPORT_PinEventInputRead and R_IOPORT_PortEventInputRead allow reading the last event input state of a pin or port, and event-triggered pin output can be configured through R_IOPORT_PinEventOutputWrite and R_IOPORT_PortEventOutputWrite.

In addition, each pin on these ports can be configured to trigger an ELC event on rising, falling or both edges. This event can be used to activate other modules when the pin changes state.

Note

The number of ELC-aware ports vary across MCUs. Refer to the Hardware User's Manual for your device for more details.

Examples

Basic Example

This is a basic example of minimal use of the IOPORT in an application.

```
void basic_example (void)
{
 bsp_io_level_t readLevel;
 fsp_err_t      err;
 /* Initialize the IOPORT module and configure the pins
```

```
  * Note: The default pin configuration name in the FSP Configuration tool is
g_bsp_pin_cfg */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Call R_IOPORT_PinsCfg if the configuration was not part of initial configurations
made in open */
    err = R_IOPORT_PinsCfg(&g_ioport_ctrl, &g_runtime_pin_cfg);
    assert(FSP_SUCCESS == err);
 /* Set Pin 07 of Port 08 to High */
    err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_08_PIN_07, BSP_IO_LEVEL_HIGH
);
    assert(FSP_SUCCESS == err);
 /* Read Pin 07 of Port 08*/
    err = R_IOPORT_PinRead(&g_ioport_ctrl, BSP_IO_PORT_08_PIN_07, &readLevel);
    assert(FSP_SUCCESS == err);
}
```

Blinky Example

This example uses IOPORT to configure and toggle a pin to blink an LED.

```
void blinky_example (void)
{
 fsp_err_t err;
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure Pin as output
  * Call the R_IOPORT_PinCfg if the configuration was not part of initial
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_08_PIN_07,
IOPORT_CFG_PORT_DIRECTION_OUTPUT);
    assert(FSP_SUCCESS == err);
```

```
 bsp_io_level_t level = BSP_IO_LEVEL_LOW;
 while (1)
    {
 /* Determine the next state of the LEDs */
 if (BSP_IO_LEVEL_LOW == level)
      {
            level = BSP_IO_LEVEL_HIGH;
       }
 else
      {
            level = BSP_IO_LEVEL_LOW;
       }
 /* Update LED on EK-RX74M */
       err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_08_PIN_07, level);
      assert(FSP_SUCCESS == err);
 /* Delay */
 R_BSP_SoftwareDelay(100, BSP_DELAY_UNITS_MILLISECONDS); // NOLINT
    }
}
```

ELC Example

This is an example of using IOPORT with ELC events. The ELC event system allows the captured data to be stored when it occurs and then read back at a later time.

```
static elc_instance_ctrl_t g_elc_ctrl;
static elc_cfg_t g_elc_cfg;
void ioport_elc_example (void)
{
 bsp_io_level_t eventValue;
 fsp_err_t      err;
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, (elc_peripheral_t) ELC_PERIPHERAL_IOPORT_C,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    assert(FSP_SUCCESS == err);
 /* Call the R_IOPORT_PinCfg if the configuration was not part of initial
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_12_PIN_00,
IOPORT_CFG_PORT_DIRECTION_INPUT);
    assert(FSP_SUCCESS == err);
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 /* Read Pin Event Input. The data(BSP_IO_LEVEL_HIGH/ BSP_IO_LEVEL_LOW) from
BSP_IO_PORT_12_PIN_00 is read into the
  * EIDR bit */
    err = R_IOPORT_PinEventInputRead(&g_ioport_ctrl, BSP_IO_PORT_12_PIN_00,
&eventValue);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	ioport_instance_ctrl_t

Enumerations

enum	ioport_port_pin_t
enum	ioport_peripheral_t
enum	ioport_cfg_options_t

Data Structure Documentation

◆ ioport_instance_ctrl_t

struct ioport_instance_ctrl_t
IOPORT private control block. DO NOT MODIFY. Initialization occurs when R_IOPORT_Open() is called.

Enumeration Type Documentation

◆ ioport_port_pin_t

enum ioport_port_pin_t	
Superset list of all possible IO port pins.	
Enumerator	
IOPORT_PORT_00_PIN_00	IO port 0 pin 0.
IOPORT_PORT_00_PIN_01	IO port 0 pin 1.
IOPORT_PORT_00_PIN_02	IO port 0 pin 2.
IOPORT_PORT_00_PIN_03	IO port 0 pin 3.
IOPORT_PORT_00_PIN_04	IO port 0 pin 4.
IOPORT_PORT_00_PIN_05	IO port 0 pin 5.
IOPORT_PORT_00_PIN_06	IO port 0 pin 6.
IOPORT_PORT_00_PIN_07	IO port 0 pin 7.
IOPORT_PORT_00_PIN_08	IO port 0 pin 8.
IOPORT_PORT_00_PIN_09	IO port 0 pin 9.
IOPORT_PORT_00_PIN_10	IO port 0 pin 10.
IOPORT_PORT_00_PIN_11	IO port 0 pin 11.
IOPORT_PORT_00_PIN_12	IO port 0 pin 12.
IOPORT_PORT_00_PIN_13	IO port 0 pin 13.
IOPORT_PORT_00_PIN_14	IO port 0 pin 14.
IOPORT_PORT_00_PIN_15	IO port 0 pin 15.
IOPORT_PORT_01_PIN_00	IO port 1 pin 0.

IOPORT_PORT_01_PIN_01	IO port 1 pin 1.
IOPORT_PORT_01_PIN_02	IO port 1 pin 2.
IOPORT_PORT_01_PIN_03	IO port 1 pin 3.
IOPORT_PORT_01_PIN_04	IO port 1 pin 4.
IOPORT_PORT_01_PIN_05	IO port 1 pin 5.
IOPORT_PORT_01_PIN_06	IO port 1 pin 6.
IOPORT_PORT_01_PIN_07	IO port 1 pin 7.
IOPORT_PORT_01_PIN_08	IO port 1 pin 8.
IOPORT_PORT_01_PIN_09	IO port 1 pin 9.
IOPORT_PORT_01_PIN_10	IO port 1 pin 10.
IOPORT_PORT_01_PIN_11	IO port 1 pin 11.
IOPORT_PORT_01_PIN_12	IO port 1 pin 12.
IOPORT_PORT_01_PIN_13	IO port 1 pin 13.
IOPORT_PORT_01_PIN_14	IO port 1 pin 14.
IOPORT_PORT_01_PIN_15	IO port 1 pin 15.
IOPORT_PORT_02_PIN_00	IO port 2 pin 0.
IOPORT_PORT_02_PIN_01	IO port 2 pin 1.
IOPORT_PORT_02_PIN_02	IO port 2 pin 2.
IOPORT_PORT_02_PIN_03	IO port 2 pin 3.
IOPORT_PORT_02_PIN_04	IO port 2 pin 4.
IOPORT_PORT_02_PIN_05	IO port 2 pin 5.
IOPORT_PORT_02_PIN_06	IO port 2 pin 6.
IOPORT_PORT_02_PIN_07	IO port 2 pin 7.
IOPORT_PORT_02_PIN_08	IO port 2 pin 8.

IOPORT_PORT_02_PIN_09	IO port 2 pin 9.
IOPORT_PORT_02_PIN_10	IO port 2 pin 10.
IOPORT_PORT_02_PIN_11	IO port 2 pin 11.
IOPORT_PORT_02_PIN_12	IO port 2 pin 12.
IOPORT_PORT_02_PIN_13	IO port 2 pin 13.
IOPORT_PORT_02_PIN_14	IO port 2 pin 14.
IOPORT_PORT_02_PIN_15	IO port 2 pin 15.
IOPORT_PORT_03_PIN_00	IO port 3 pin 0.
IOPORT_PORT_03_PIN_01	IO port 3 pin 1.
IOPORT_PORT_03_PIN_02	IO port 3 pin 2.
IOPORT_PORT_03_PIN_03	IO port 3 pin 3.
IOPORT_PORT_03_PIN_04	IO port 3 pin 4.
IOPORT_PORT_03_PIN_05	IO port 3 pin 5.
IOPORT_PORT_03_PIN_06	IO port 3 pin 6.
IOPORT_PORT_03_PIN_07	IO port 3 pin 7.
IOPORT_PORT_03_PIN_08	IO port 3 pin 8.
IOPORT_PORT_03_PIN_09	IO port 3 pin 9.
IOPORT_PORT_03_PIN_10	IO port 3 pin 10.
IOPORT_PORT_03_PIN_11	IO port 3 pin 11.
IOPORT_PORT_03_PIN_12	IO port 3 pin 12.
IOPORT_PORT_03_PIN_13	IO port 3 pin 13.
IOPORT_PORT_03_PIN_14	IO port 3 pin 14.
IOPORT_PORT_03_PIN_15	IO port 3 pin 15.
IOPORT_PORT_04_PIN_00	IO port 4 pin 0.

IOPORT_PORT_04_PIN_01	IO port 4 pin 1.
IOPORT_PORT_04_PIN_02	IO port 4 pin 2.
IOPORT_PORT_04_PIN_03	IO port 4 pin 3.
IOPORT_PORT_04_PIN_04	IO port 4 pin 4.
IOPORT_PORT_04_PIN_05	IO port 4 pin 5.
IOPORT_PORT_04_PIN_06	IO port 4 pin 6.
IOPORT_PORT_04_PIN_07	IO port 4 pin 7.
IOPORT_PORT_04_PIN_08	IO port 4 pin 8.
IOPORT_PORT_04_PIN_09	IO port 4 pin 9.
IOPORT_PORT_04_PIN_10	IO port 4 pin 10.
IOPORT_PORT_04_PIN_11	IO port 4 pin 11.
IOPORT_PORT_04_PIN_12	IO port 4 pin 12.
IOPORT_PORT_04_PIN_13	IO port 4 pin 13.
IOPORT_PORT_04_PIN_14	IO port 4 pin 14.
IOPORT_PORT_04_PIN_15	IO port 4 pin 15.
IOPORT_PORT_05_PIN_00	IO port 5 pin 0.
IOPORT_PORT_05_PIN_01	IO port 5 pin 1.
IOPORT_PORT_05_PIN_02	IO port 5 pin 2.
IOPORT_PORT_05_PIN_03	IO port 5 pin 3.
IOPORT_PORT_05_PIN_04	IO port 5 pin 4.
IOPORT_PORT_05_PIN_05	IO port 5 pin 5.
IOPORT_PORT_05_PIN_06	IO port 5 pin 6.
IOPORT_PORT_05_PIN_07	IO port 5 pin 7.
IOPORT_PORT_05_PIN_08	IO port 5 pin 8.

IOPORT_PORT_05_PIN_09	IO port 5 pin 9.
IOPORT_PORT_05_PIN_10	IO port 5 pin 10.
IOPORT_PORT_05_PIN_11	IO port 5 pin 11.
IOPORT_PORT_05_PIN_12	IO port 5 pin 12.
IOPORT_PORT_05_PIN_13	IO port 5 pin 13.
IOPORT_PORT_05_PIN_14	IO port 5 pin 14.
IOPORT_PORT_05_PIN_15	IO port 5 pin 15.
IOPORT_PORT_06_PIN_00	IO port 6 pin 0.
IOPORT_PORT_06_PIN_01	IO port 6 pin 1.
IOPORT_PORT_06_PIN_02	IO port 6 pin 2.
IOPORT_PORT_06_PIN_03	IO port 6 pin 3.
IOPORT_PORT_06_PIN_04	IO port 6 pin 4.
IOPORT_PORT_06_PIN_05	IO port 6 pin 5.
IOPORT_PORT_06_PIN_06	IO port 6 pin 6.
IOPORT_PORT_06_PIN_07	IO port 6 pin 7.
IOPORT_PORT_06_PIN_08	IO port 6 pin 8.
IOPORT_PORT_06_PIN_09	IO port 6 pin 9.
IOPORT_PORT_06_PIN_10	IO port 6 pin 10.
IOPORT_PORT_06_PIN_11	IO port 6 pin 11.
IOPORT_PORT_06_PIN_12	IO port 6 pin 12.
IOPORT_PORT_06_PIN_13	IO port 6 pin 13.
IOPORT_PORT_06_PIN_14	IO port 6 pin 14.
IOPORT_PORT_06_PIN_15	IO port 6 pin 15.
IOPORT_PORT_07_PIN_00	IO port 7 pin 0.

IOPORT_PORT_07_PIN_01	IO port 7 pin 1.
IOPORT_PORT_07_PIN_02	IO port 7 pin 2.
IOPORT_PORT_07_PIN_03	IO port 7 pin 3.
IOPORT_PORT_07_PIN_04	IO port 7 pin 4.
IOPORT_PORT_07_PIN_05	IO port 7 pin 5.
IOPORT_PORT_07_PIN_06	IO port 7 pin 6.
IOPORT_PORT_07_PIN_07	IO port 7 pin 7.
IOPORT_PORT_07_PIN_08	IO port 7 pin 8.
IOPORT_PORT_07_PIN_09	IO port 7 pin 9.
IOPORT_PORT_07_PIN_10	IO port 7 pin 10.
IOPORT_PORT_07_PIN_11	IO port 7 pin 11.
IOPORT_PORT_07_PIN_12	IO port 7 pin 12.
IOPORT_PORT_07_PIN_13	IO port 7 pin 13.
IOPORT_PORT_07_PIN_14	IO port 7 pin 14.
IOPORT_PORT_07_PIN_15	IO port 7 pin 15.
IOPORT_PORT_08_PIN_00	IO port 8 pin 0.
IOPORT_PORT_08_PIN_01	IO port 8 pin 1.
IOPORT_PORT_08_PIN_02	IO port 8 pin 2.
IOPORT_PORT_08_PIN_03	IO port 8 pin 3.
IOPORT_PORT_08_PIN_04	IO port 8 pin 4.
IOPORT_PORT_08_PIN_05	IO port 8 pin 5.
IOPORT_PORT_08_PIN_06	IO port 8 pin 6.
IOPORT_PORT_08_PIN_07	IO port 8 pin 7.
IOPORT_PORT_08_PIN_08	IO port 8 pin 8.

IOPORT_PORT_08_PIN_09	IO port 8 pin 9.
IOPORT_PORT_08_PIN_10	IO port 8 pin 10.
IOPORT_PORT_08_PIN_11	IO port 8 pin 11.
IOPORT_PORT_08_PIN_12	IO port 8 pin 12.
IOPORT_PORT_08_PIN_13	IO port 8 pin 13.
IOPORT_PORT_08_PIN_14	IO port 8 pin 14.
IOPORT_PORT_08_PIN_15	IO port 8 pin 15.
IOPORT_PORT_09_PIN_00	IO port 9 pin 0.
IOPORT_PORT_09_PIN_01	IO port 9 pin 1.
IOPORT_PORT_09_PIN_02	IO port 9 pin 2.
IOPORT_PORT_09_PIN_03	IO port 9 pin 3.
IOPORT_PORT_09_PIN_04	IO port 9 pin 4.
IOPORT_PORT_09_PIN_05	IO port 9 pin 5.
IOPORT_PORT_09_PIN_06	IO port 9 pin 6.
IOPORT_PORT_09_PIN_07	IO port 9 pin 7.
IOPORT_PORT_09_PIN_08	IO port 9 pin 8.
IOPORT_PORT_09_PIN_09	IO port 9 pin 9.
IOPORT_PORT_09_PIN_10	IO port 9 pin 10.
IOPORT_PORT_09_PIN_11	IO port 9 pin 11.
IOPORT_PORT_09_PIN_12	IO port 9 pin 12.
IOPORT_PORT_09_PIN_13	IO port 9 pin 13.
IOPORT_PORT_09_PIN_14	IO port 9 pin 14.
IOPORT_PORT_09_PIN_15	IO port 9 pin 15.
IOPORT_PORT_10_PIN_00	IO port 10 pin 0.

IOPORT_PORT_10_PIN_01	IO port 10 pin 1.
IOPORT_PORT_10_PIN_02	IO port 10 pin 2.
IOPORT_PORT_10_PIN_03	IO port 10 pin 3.
IOPORT_PORT_10_PIN_04	IO port 10 pin 4.
IOPORT_PORT_10_PIN_05	IO port 10 pin 5.
IOPORT_PORT_10_PIN_06	IO port 10 pin 6.
IOPORT_PORT_10_PIN_07	IO port 10 pin 7.
IOPORT_PORT_10_PIN_08	IO port 10 pin 8.
IOPORT_PORT_10_PIN_09	IO port 10 pin 9.
IOPORT_PORT_10_PIN_10	IO port 10 pin 10.
IOPORT_PORT_10_PIN_11	IO port 10 pin 11.
IOPORT_PORT_10_PIN_12	IO port 10 pin 12.
IOPORT_PORT_10_PIN_13	IO port 10 pin 13.
IOPORT_PORT_10_PIN_14	IO port 10 pin 14.
IOPORT_PORT_10_PIN_15	IO port 10 pin 15.
IOPORT_PORT_11_PIN_00	IO port 11 pin 0.
IOPORT_PORT_11_PIN_01	IO port 11 pin 1.
IOPORT_PORT_11_PIN_02	IO port 11 pin 2.
IOPORT_PORT_11_PIN_03	IO port 11 pin 3.
IOPORT_PORT_11_PIN_04	IO port 11 pin 4.
IOPORT_PORT_11_PIN_05	IO port 11 pin 5.
IOPORT_PORT_11_PIN_06	IO port 11 pin 6.
IOPORT_PORT_11_PIN_07	IO port 11 pin 7.
IOPORT_PORT_11_PIN_08	IO port 11 pin 8.

IOPORT_PORT_11_PIN_09	IO port 11 pin 9.
IOPORT_PORT_11_PIN_10	IO port 11 pin 10.
IOPORT_PORT_11_PIN_11	IO port 11 pin 11.
IOPORT_PORT_11_PIN_12	IO port 11 pin 12.
IOPORT_PORT_11_PIN_13	IO port 11 pin 13.
IOPORT_PORT_11_PIN_14	IO port 11 pin 14.
IOPORT_PORT_11_PIN_15	IO port 11 pin 15.
IOPORT_PORT_12_PIN_00	IO port 12 pin 0.
IOPORT_PORT_12_PIN_01	IO port 12 pin 1.
IOPORT_PORT_12_PIN_02	IO port 12 pin 2.
IOPORT_PORT_12_PIN_03	IO port 12 pin 3.
IOPORT_PORT_12_PIN_04	IO port 12 pin 4.
IOPORT_PORT_12_PIN_05	IO port 12 pin 5.
IOPORT_PORT_12_PIN_06	IO port 12 pin 6.
IOPORT_PORT_12_PIN_07	IO port 12 pin 7.
IOPORT_PORT_12_PIN_08	IO port 12 pin 8.
IOPORT_PORT_12_PIN_09	IO port 12 pin 9.
IOPORT_PORT_12_PIN_10	IO port 12 pin 10.
IOPORT_PORT_12_PIN_11	IO port 12 pin 11.
IOPORT_PORT_12_PIN_12	IO port 12 pin 12.
IOPORT_PORT_12_PIN_13	IO port 12 pin 13.
IOPORT_PORT_12_PIN_14	IO port 12 pin 14.
IOPORT_PORT_12_PIN_15	IO port 12 pin 15.
IOPORT_PORT_13_PIN_00	IO port 13 pin 0.

IOPORT_PORT_13_PIN_01	IO port 13 pin 1.
IOPORT_PORT_13_PIN_02	IO port 13 pin 2.
IOPORT_PORT_13_PIN_03	IO port 13 pin 3.
IOPORT_PORT_13_PIN_04	IO port 13 pin 4.
IOPORT_PORT_13_PIN_05	IO port 13 pin 5.
IOPORT_PORT_13_PIN_06	IO port 13 pin 6.
IOPORT_PORT_13_PIN_07	IO port 13 pin 7.
IOPORT_PORT_13_PIN_08	IO port 13 pin 8.
IOPORT_PORT_13_PIN_09	IO port 13 pin 9.
IOPORT_PORT_13_PIN_10	IO port 13 pin 10.
IOPORT_PORT_13_PIN_11	IO port 13 pin 11.
IOPORT_PORT_13_PIN_12	IO port 13 pin 12.
IOPORT_PORT_13_PIN_13	IO port 13 pin 13.
IOPORT_PORT_13_PIN_14	IO port 13 pin 14.
IOPORT_PORT_13_PIN_15	IO port 13 pin 15.
IOPORT_PORT_14_PIN_00	IO port 14 pin 0.
IOPORT_PORT_14_PIN_01	IO port 14 pin 1.
IOPORT_PORT_14_PIN_02	IO port 14 pin 2.
IOPORT_PORT_14_PIN_03	IO port 14 pin 3.
IOPORT_PORT_14_PIN_04	IO port 14 pin 4.
IOPORT_PORT_14_PIN_05	IO port 14 pin 5.
IOPORT_PORT_14_PIN_06	IO port 14 pin 6.
IOPORT_PORT_14_PIN_07	IO port 14 pin 7.
IOPORT_PORT_14_PIN_08	IO port 14 pin 8.

IOPORT_PORT_14_PIN_09	IO port 14 pin 9.
IOPORT_PORT_14_PIN_10	IO port 14 pin 10.
IOPORT_PORT_14_PIN_11	IO port 14 pin 11.
IOPORT_PORT_14_PIN_12	IO port 14 pin 12.
IOPORT_PORT_14_PIN_13	IO port 14 pin 13.
IOPORT_PORT_14_PIN_14	IO port 14 pin 14.
IOPORT_PORT_14_PIN_15	IO port 14 pin 15.
IOPORT_PORT_15_PIN_00	IO port 15 pin 0.
IOPORT_PORT_15_PIN_01	IO port 15 pin 1.
IOPORT_PORT_15_PIN_02	IO port 15 pin 2.
IOPORT_PORT_15_PIN_03	IO port 15 pin 3.
IOPORT_PORT_15_PIN_04	IO port 15 pin 4.
IOPORT_PORT_15_PIN_05	IO port 15 pin 5.
IOPORT_PORT_15_PIN_06	IO port 15 pin 6.
IOPORT_PORT_15_PIN_07	IO port 15 pin 7.
IOPORT_PORT_15_PIN_08	IO port 15 pin 8.
IOPORT_PORT_15_PIN_09	IO port 15 pin 9.
IOPORT_PORT_15_PIN_10	IO port 15 pin 10.
IOPORT_PORT_15_PIN_11	IO port 15 pin 11.
IOPORT_PORT_15_PIN_12	IO port 15 pin 12.
IOPORT_PORT_15_PIN_13	IO port 15 pin 13.
IOPORT_PORT_15_PIN_14	IO port 15 pin 14.
IOPORT_PORT_15_PIN_15	IO port 15 pin 15.
IOPORT_PORT_16_PIN_00	IO port 16 pin 0.

IOPORT_PORT_16_PIN_01	IO port 16 pin 1.
IOPORT_PORT_16_PIN_02	IO port 16 pin 2.
IOPORT_PORT_16_PIN_03	IO port 16 pin 3.
IOPORT_PORT_16_PIN_04	IO port 16 pin 4.
IOPORT_PORT_16_PIN_05	IO port 16 pin 5.
IOPORT_PORT_16_PIN_06	IO port 16 pin 6.
IOPORT_PORT_16_PIN_07	IO port 16 pin 7.
IOPORT_PORT_16_PIN_08	IO port 16 pin 8.
IOPORT_PORT_16_PIN_09	IO port 16 pin 9.
IOPORT_PORT_16_PIN_10	IO port 16 pin 10.
IOPORT_PORT_16_PIN_11	IO port 16 pin 11.
IOPORT_PORT_16_PIN_12	IO port 16 pin 12.
IOPORT_PORT_16_PIN_13	IO port 16 pin 13.
IOPORT_PORT_16_PIN_14	IO port 16 pin 14.
IOPORT_PORT_16_PIN_15	IO port 16 pin 15.

◆ ioport_peripheral_t

enum ioport_peripheral_t	
Superset of all peripheral functions.	
Enumerator	
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin
IOPORT_PERIPHERAL_AGT	Pin will function as an AGT peripheral pin
IOPORT_PERIPHERAL_AGTW	Pin will function as an AGT peripheral pin
IOPORT_PERIPHERAL_AGT1	Pin will function as an AGT peripheral pin
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8	Pin will function as an SCI peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9	Pin will function as an SCI peripheral pin
IOPORT_PERIPHERAL_SPI	Pin will function as a SPI peripheral pin
IOPORT_PERIPHERAL_IIC	Pin will function as a IIC peripheral pin
IOPORT_PERIPHERAL_KEY	Pin will function as a KEY peripheral pin
IOPORT_PERIPHERAL_CLKOUT_COMP_RTC	Pin will function as a clock/comparator/RTC peripheral pin
IOPORT_PERIPHERAL_CAC_AD	Pin will function as a CAC/ADC peripheral pin
IOPORT_PERIPHERAL_BUS	Pin will function as a BUS peripheral pin
IOPORT_PERIPHERAL_CTSU	Pin will function as a CTSU peripheral pin
IOPORT_PERIPHERAL_ACMPHS	Pin will function as a CMPHS peripheral pin
IOPORT_PERIPHERAL_LCDC	Pin will function as a segment LCD peripheral pin
IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8	Pin will function as an SCI peripheral DEn pin
IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9	Pin will function as an SCI DEn peripheral pin

IOPORT_PERIPHERAL_DALI	Pin will function as a DALI peripheral pin
IOPORT_PERIPHERAL_CEU	Pin will function as a CEU peripheral pin
IOPORT_PERIPHERAL_CAN	Pin will function as a CAN peripheral pin
IOPORT_PERIPHERAL_QSPI	Pin will function as a QSPI peripheral pin
IOPORT_PERIPHERAL_SSI	Pin will function as an SSI peripheral pin
IOPORT_PERIPHERAL_USB_FS	Pin will function as a USB full speed peripheral pin
IOPORT_PERIPHERAL_USB_HS	Pin will function as a USB high speed peripheral pin
IOPORT_PERIPHERAL_GPT2	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_SDHI_MMC	Pin will function as an SD/MMC peripheral pin
IOPORT_PERIPHERAL_GPT3	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_ETHER_MII	Pin will function as an Ethernet MMI peripheral pin
IOPORT_PERIPHERAL_GPT4	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT5	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet RMMI peripheral pin
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC peripheral pin
IOPORT_PERIPHERAL_LCD_GRAPHICS	Pin will function as a graphics LCD peripheral pin
IOPORT_PERIPHERAL_CAC	Pin will function as a CAC peripheral pin
IOPORT_PERIPHERAL_TRACE	Pin will function as a debug trace peripheral pin
IOPORT_PERIPHERAL_OSPI	Pin will function as a OSPI peripheral pin
IOPORT_PERIPHERAL_CEC	Pin will function as a CEC peripheral pin
IOPORT_PERIPHERAL_PGAOUT0	Pin will function as a PGAOUT peripheral pin
IOPORT_PERIPHERAL_PGAOUT1	Pin will function as a PGAOUT peripheral pin

IOPORT_PERIPHERAL_ULPT	Pin will function as a ULPT peripheral pin
IOPORT_PERIPHERAL_MIPI	Pin will function as a MIPI DSI peripheral pin
IOPORT_PERIPHERAL_UARTA	Pin will function as an UARTA peripheral pin
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin
IOPORT_PERIPHERAL_RTCIC	Pin will function as a RTC(RTCIC) pin
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT2	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT3	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_TMR	Pin will function as a TMR peripheral pin
IOPORT_PERIPHERAL_RTC	Pin will function as a RTC(RTCOUT) peripheral pin
IOPORT_PERIPHERAL_CAC	Pin will function as a CAC peripheral pin
IOPORT_PERIPHERAL_POEG	Pin will function as a POEG peripheral pin
IOPORT_PERIPHERAL_ADC	Pin will function as a ADC (Digital input) peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8_10_12_0A	Pin will function as an Even RSCI peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8_10_12_0B	Pin will function as an Even RSCI(CTS,DE) peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8_10_12_0C	Pin will function as an Even RSCI HBS and others peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0D	Pin will function as an Odd RSCI peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0E	Pin will function as an Odd RSCI(CTS) peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0F	Pin will function as an Odd RSCI HBS and others peripheral pin

IOPORT_PERIPHERAL_SPI	Pin will function as a RSPI peripheral pin
IOPORT_PERIPHERAL_I3C	Pin will function as a RI3C peripheral pin
IOPORT_PERIPHERAL_CANFD	Pin will function as a CANFD peripheral pin
IOPORT_PERIPHERAL_USB0	Pin will function as a USB full speed peripheral pin
IOPORT_PERIPHERAL_USBHS	Pin will function as a USB high speed peripheral pin
IOPORT_PERIPHERAL_DSMIF	Pin will function as a DSMIF peripheral pin
IOPORT_PERIPHERAL_PDMIF	Pin will function as a PDMIF peripheral pin
IOPORT_PERIPHERAL_SSIE	Pin will function as an SSIE peripheral pin
IOPORT_PERIPHERAL_PCIF	Pin will function as an PCIF peripheral pin
IOPORT_PERIPHERAL_SDHI	Pin will function as an SDHI peripheral pin
IOPORT_PERIPHERAL_XSPI0_1A	Pin will function as a XSPI first peripheral pin
IOPORT_PERIPHERAL_XSPI0_1B	Pin will function as a XSPI first peripheral pin
IOPORT_PERIPHERAL_XSPI1_1A	Pin will function as a XSPI second peripheral pin
IOPORT_PERIPHERAL_XSPI1_1B	Pin will function as a XSPI second peripheral pin
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC peripheral pin
IOPORT_PERIPHERAL_CMTW	Pin will function as a CMTW peripheral pin
IOPORT_PERIPHERAL_CLKOUT	Pin will function as a clock peripheral pin
IOPORT_PERIPHERAL_ETHER_MII_GMII	Pin will function as an Ethernet GMII/MII peripheral pin
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet RMMI peripheral pin
IOPORT_PERIPHERAL_ETHER_RGMII	Pin will function as an Ethernet RGMMI peripheral pin
IOPORT_PERIPHERAL_ETHER_PTP	Pin will function as an Ethernet PTP peripheral pin

IOPORT_PERIPHERAL_ESC0	Pin will function as an Ethernet EtherCAT first peripheral pin
IOPORT_PERIPHERAL_ESC1	Pin will function as an Ethernet EtherCAT second peripheral pin
IOPORT_PERIPHERAL_BUS0	Pin will function as an EXBUS/SDRAM first peripheral pin
IOPORT_PERIPHERAL_BUS1	Pin will function as an EXBUS/SDRAM second peripheral pin
IOPORT_PERIPHERAL_PIO	Pin will function as an 32-bit board function peripheral pin
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin
IOPORT_PERIPHERAL_RTCIC	Pin will function as a RTC(RTCIC) pin
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT2	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT3	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_TMR	Pin will function as a TMR peripheral pin
IOPORT_PERIPHERAL_RTC	Pin will function as a RTC(RTCOUT) peripheral pin
IOPORT_PERIPHERAL_CAC	Pin will function as a CAC peripheral pin
IOPORT_PERIPHERAL_POEG	Pin will function as a POEG peripheral pin
IOPORT_PERIPHERAL_ADC	Pin will function as a ADC (Digital input) peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8_10_12_0A	Pin will function as an Even RSCI peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8_10_12_0B	Pin will function as an Even RSCI(CTS,DE) peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8_10_12_0C	Pin will function as an Even RSCI HBS and others peripheral pin

IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0D	Pin will function as an Odd RSCI peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0E	Pin will function as an Odd RSCI(CTS) peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0F	Pin will function as an Odd RSCI HBS and others peripheral pin
IOPORT_PERIPHERAL_SPI	Pin will function as a RSPI peripheral pin
IOPORT_PERIPHERAL_I3C	Pin will function as a RI3C peripheral pin
IOPORT_PERIPHERAL_CANFD	Pin will function as a CANFD peripheral pin
IOPORT_PERIPHERAL_USB0	Pin will function as a USB full speed peripheral pin
IOPORT_PERIPHERAL_USBHS	Pin will function as a USB high speed peripheral pin
IOPORT_PERIPHERAL_PDMIF	Pin will function as a PDMIF peripheral pin
IOPORT_PERIPHERAL_SSIE	Pin will function as an SSIE peripheral pin
IOPORT_PERIPHERAL_PCIF	Pin will function as an PCIF peripheral pin
IOPORT_PERIPHERAL_SDHI	Pin will function as an SDHI peripheral pin
IOPORT_PERIPHERAL_XSPI0_1A	Pin will function as a XSPI first peripheral pin
IOPORT_PERIPHERAL_XSPI0_1B	Pin will function as a XSPI first peripheral pin
IOPORT_PERIPHERAL_XSPI1_1A	Pin will function as a XSPI second peripheral pin
IOPORT_PERIPHERAL_XSPI1_1B	Pin will function as a XSPI second peripheral pin
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC peripheral pin
IOPORT_PERIPHERAL_CMTW	Pin will function as a CMTW peripheral pin
IOPORT_PERIPHERAL_CLKOUT	Pin will function as a clock peripheral pin
IOPORT_PERIPHERAL_ETHER_MII_GMII	Pin will function as an Ethernet GMII/MII peripheral pin
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet RMMI peripheral pin

IOPORT_PERIPHERAL_ETHER_RGMII	Pin will function as an Ethernet RGMMI peripheral pin
IOPORT_PERIPHERAL_ETHER_PTP	Pin will function as an Ethernet PTP peripheral pin
IOPORT_PERIPHERAL_BUS0	Pin will function as an EXBUS/SDRAM first peripheral pin
IOPORT_PERIPHERAL_BUS1	Pin will function as an EXBUS/SDRAM second peripheral pin
IOPORT_PERIPHERAL_PIO	Pin will function as an 32-bit board function peripheral pin

◆ ioport_cfg_options_t

enum ioport_cfg_options_t	
Options to configure pin functions	
Enumerator	
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_PIM_TTL	Enables the pin's input mode.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.
IOPORT_CFG_PMOS_ENABLE	Enables the pin's PMOS open-drain output.
IOPORT_CFG_DRIVE_MID	Sets pin drive output to medium.
IOPORT_CFG_DRIVE_HS_HIGH	Sets pin drive output to high along with supporting high speed.
IOPORT_CFG_DRIVE_MID_IIC	Sets pin to drive output needed for IIC on a 20mA port.
IOPORT_CFG_DRIVE_HIGH	Sets pin drive output to high.

IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.
IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.
IOPORT_CFG_DRIVE_1	Sets pin drive output to DRV1.
IOPORT_CFG_DRIVE_2	Sets pin drive output to DRV2.
IOPORT_CFG_DRIVE_3	Sets pin drive output to DRV3.
IOPORT_CFG_DRIVE_4	Sets pin drive output to DRV4.
IOPORT_CFG_DRIVE_5	Sets pin drive output to DRV5.
IOPORT_CFG_DRIVE_6	Sets pin drive output to DRV6.
IOPORT_CFG_DRIVE_7	Sets pin drive output to DRV7.
IOPORT_CFG_DRIVE_8	Sets pin drive output to DRV8.
IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.

IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.
IOPORT_CFG_DRIVE_1	Sets pin drive output to DRV1.
IOPORT_CFG_DRIVE_2	Sets pin drive output to DRV2.
IOPORT_CFG_DRIVE_3	Sets pin drive output to DRV3.
IOPORT_CFG_DRIVE_4	Sets pin drive output to DRV4.
IOPORT_CFG_DRIVE_5	Sets pin drive output to DRV5.
IOPORT_CFG_DRIVE_6	Sets pin drive output to DRV6.
IOPORT_CFG_DRIVE_7	Sets pin drive output to DRV7.
IOPORT_CFG_DRIVE_8	Sets pin drive output to DRV8.
IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.
IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.

Function Documentation

◆ R_IOPORT_Open()

fsp_err_t R_IOPORT_Open (ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg*)

Initializes internal driver data, then calls pin configuration function to configure pins.

Return values

FSP_SUCCESS	Pin configuration data written to PFS register(s)
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_IOPORT_Close()

fsp_err_t R_IOPORT_Close (ioport_ctrl_t *const *p_ctrl*)

Resets IOPORT registers. Implements ioport_api_t::close

Return values

FSP_SUCCESS	The IOPORT was successfully uninitialized
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_IOPORT_PinsCfg()

fsp_err_t R_IOPORT_PinsCfg (ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg*)

Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements ioport_api_t::pinsCfg.

This function initializes the supplied list of PmnPFS registers with the supplied values. This data can be generated by the Pins tab of the RX Configuration editor or manually by the developer. Different pin configurations can be loaded for different situations such as low power modes and testing.

Return values

FSP_SUCCESS	Pin configuration data written to PFS register(s)
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

◆ R_IOPORT_PinCfg()

fsp_err_t R_IOPORT_PinCfg (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, uint32_t *cfg*)

Configures the settings of a pin. Implements ioport_api_t::pinCfg.

Return values

FSP_SUCCESS	Pin configured
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different pins. This function will change the configuration of the pin with the new configuration. For example it is not possible with this function to change the drive strength of a pin while leaving all the other pin settings unchanged. To achieve this the original settings with the required change will need to be written using this function.

◆ R_IOPORT_PinEventInputRead()

fsp_err_t R_IOPORT_PinEventInputRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_event*)

Reads the value of the event input data of a specific pin. Implements ioport_api_t::pinEventInputRead.

The pin event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Return values

FSP_SUCCESS	Pin read
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_INVALID_ARGUMENT	Port is not valid ELC PORT.
FSP_ERR_UNSUPPORTED	Function not supported.

Note

This function is re-entrant.

◆ R_IOPORT_PinEventOutputWrite()

fsp_err_t R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *pin_value*)

This function writes the event output data value to a pin. Implements ioport_api_t::pinEventOutputWrite.

Using the event system enables a pin state to be stored by this function in advance of being output on the pin. The output to the pin will occur when the ELC event occurs.

Return values

FSP_SUCCESS	Pin event data written
FSP_ERR_INVALID_ARGUMENT	Port or Pin or value not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_UNSUPPORTED	Function not supported

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PinRead()

fsp_err_t R_IOPORT_PinRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_value*)

Reads the level on a pin. Implements ioport_api_t::pinRead.

Return values

FSP_SUCCESS	Pin read
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened

Note

This function is re-entrant for different pins.

◆ R_IOPORT_PinWrite()

fsp_err_t R_IOPORT_PinWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *level*)

Sets a pin's output either high or low. Implements ioport_api_t::pinWrite.

Return values

FSP_SUCCESS	Pin written to
FSP_ERR_INVALID_ARGUMENT	The pin and/or level not valid
FSP_ERR_NOT_OPEN	The module has not been open
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different pins. This function makes use of the PCNTR3 register to atomically modify the level on the specified pin on a port.

◆ R_IOPORT_PortDirectionSet()

fsp_err_t R_IOPORT_PortDirectionSet (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *direction_values*, ioport_size_t *mask*)

Sets the direction of individual pins on a port. Implements ioport_api_t::portDirectionSet().

Multiple pins on a port can be set to inputs or outputs at once. Each bit in the mask parameter corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. If a bit is set to 1 then the corresponding pin will be changed to an input or an output as specified by the direction values. If a mask bit is set to 0 then the direction of the pin will not be changed.

Return values

FSP_SUCCESS	Port direction updated
FSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortEventInputRead()

fsp_err_t R_IOPORT_PortEventInputRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_event_data*)

Reads the value of the event input data. Implements ioport_api_t::portEventInputRead().

The event input data for the port will be read. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

The port event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Return values

FSP_SUCCESS	Port read
FSP_ERR_INVALID_ARGUMENT	Port not a valid ELC port
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_UNSUPPORTED	Function not supported.

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortEventOutputWrite()

fsp_err_t R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *event_data*, ioport_size_t *mask_value*)

This function writes the set and reset event output data for a port. Implements ioport_api_t::portEventOutputWrite.

Using the event system enables a port state to be stored by this function in advance of being output on the port. The output to the port will occur when the ELC event occurs.

The input value will be written to the specified port when an ELC event configured for that port occurs. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Return values

FSP_SUCCESS	Port event data written
FSP_ERR_INVALID_ARGUMENT	Port or Mask not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_UNSUPPORTED	Function not supported

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortRead()

fsp_err_t R_IOPORT_PortRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_port_value*)

Reads the value on an IO port. Implements ioport_api_t::portRead.

The specified port will be read, and the levels for all the pins will be returned. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

Return values

FSP_SUCCESS	Port read
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortWrite()

fsp_err_t R_IOPORT_PortWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *value*, ioport_size_t *mask*)

Writes to multiple pins on a port. Implements ioport_api_t::portWrite.

The input value will be written to the specified port. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Only the bits with the corresponding bit in the mask value set will be updated. For example, value = 0xFFFF, mask = 0x0003 results in only bits 0 and 1 being updated.

Return values

FSP_SUCCESS	Port written to
FSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports. This function makes use of the PCNTR3 register to atomically modify the levels on the specified pins on a port.

5.2.19 Timers

Modules

Detailed Description

Timers Modules.

Modules

	Compare Match Timer (r_cmt) Driver for the CMT peripherals on RX MCU. This module implements the Timer Interface.
	Compare Match Timer W (r_cmtw) Driver for the CMTW peripherals on RX MCUs. This module implements the Timer Interface.
	Timer, General PWM (r_gpt)

	Driver for the GPT32 and GPT16 peripherals on RX MCUs. This module implements the Timer Interface.
	Three-Phase PWM (r_gpt_three_phase) Driver for 3-phase motor control using the GPT peripheral on RX MCUs. This module implements the Three-Phase Interface.
	Port Output Enable for GPT (r_poeg) Driver for the POEG peripheral on RX MCUs. This module implements the POEG Interface.
	Realtime Clock (r_rtc) Driver for the RTC peripheral on RX MCUs. This module implements the RTC Interface.
	Timer, 8-Bit Timer (r_tmr) Driver for the TMR peripheral on RX MCUs. This module implements the Timer Interface.

5.2.19.1 Compare Match Timer (r_cmt)

Modules » Timers

Functions

fsp_err_t	R_CMT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t	R_CMT_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMT_Start (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMT_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMT_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMT_Disable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_CMT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const

	duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_CMT_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel)
fsp_err_t	R_CMT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t	R_CMT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t	R_CMT_CounterSet (timer_ctrl_t *const p_ctrl, uint32_t counter)
fsp_err_t	R_CMT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t	R_CMT_Close (timer_ctrl_t *const p_ctrl)

Detailed Description

Driver for the CMT peripherals on RX MCU. This module implements the Timer Interface.

Overview

RX has two on-chip CMT units (unit 0 and unit 1), each consisting of a two-channel 16-bit timer.

The CMT has a 16-bit counter, and can generate interrupts at set intervals.

Features

The CMT module has the following features:

- Supports periodic mode, one-shot mode.
- Supports count source of PCLK or ELC events.
- Configurable period (counts per timer cycle).
- Supports runtime reconfiguration of period.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, counting direction.
- APIs are provided to get the current timer status and counter value.
- Supports linking to the specified module, CMT1 count start, event count, or count restart by ELC events.

Selecting a Timer

When selecting CMT, consider these factors:

	GPT	CMTW	CMT	TMR
Timer Resolution	All MCUs have at least one 32-bit	The CMTW timer have at least one	The CMT timers are 16-bit timers.	The TMR timers are 8-bit timer or

	GPT timer.	32-bit CMTW timer.		16-bit timer.
Clock Source	The GPT runs off PCLKA with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The CMTW runs off PCLKB.	The CMT runs off PCLKB with a configurable divider from 8 to 512. It can also be configured to count ELC events.	The TMR runs off PCLKB.
Low Power Modes	The GPT can operate in sleep mode.	The CMTW can operate in sleep mode.	The CMT can operate in in sleep mode.	The TMR can operate in sleep mode.

Configuration

Build Time Configurations for r_cmt

The following build time configurations are defined in fsp_cfg/r_cmt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Timer, Compare Match Timer (r_cmt)

This module can be added to the Stacks tab via New Stack > Timers > Timer, Compare Match Timer (r_cmt).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_timer0	Module name.
General > Channel	Value must be a non-negative integer	0	Specify the hardware channel.
General > Mode	• Periodic • One-Shot	Periodic	Mode selection. Periodic: Generates periodic interrupts or square waves. One-shot: Generate a single interrupt or a pulse wave. Note: One-shot mode is implemented in software. ISRs must be enabled for one-shot even if callback is unused.
General > Period	Value must be non-	0x10000	Specify the timer

	negative		period based on the selected unit.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Raw Counts	Unit of the period specified above
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses
Interrupts > Compare Match Interrupt Priority	MCU Specific Options		Select the compare match interrupt priority.

Clock Configuration

The CMT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time.

Usage Notes

Period

When period unit is selected as "Raw Counts", the output waveform period is calculated based on default clock frequency PCLKB/8

Maximum Period

The RX Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time and units.

When the selected period unit is "Raw counts", the maximum period setting is 0x10000 on a 16-bit timer. This will configure the timer with the maximum period and a count clock divisor of 512.

Updating Period

The period is updated after the next counter overflow after calling R_CMT_PeriodSet(). To force them to update before the next counter overflow, call R_CMT_Reset() while the counter is running.

Note

When manually changing the timer period counts the maximum value for a 16-bit CMT is 0x10000. This number overflows the 16-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.

One-Shot Mode

After a compare match interrupt is generated, the counter then reset to 0000h and stop.

Periodic Mode:

Frequency dividing clock is selected from four frequency dividing clocks obtained by dividing the peripheral module clock (PCLKB), the CMCNT counter starts counting up using the selected clock. When the value in the counter and the value in the register match, a compare match interrupt (CMIn) is generated. The CMCNT counter then starts counting up again from 0000h.

Event Counting

Event counting can be done by selecting counting sources from CMCR, ELC events.

Note

In event counting mode, the application must call R_CMT_Start() to enable event counting. The counter will not change after calling R_CMT_Start() until an event occurs.

Controlling CMT with ELC Events

The cmt timer can be configured to start, count, restart when an ELC event occurs.

Note

Event signal input and output are available for CMT1 only (channel 2, channel 3).

Examples

CMT Basic Example

This is a basic example of minimal use of the cmt in an application.

```
void cmt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMT_Start(&g_timer0_ctrl);
}
void cmt_reset_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMT_Start(&g_timer0_ctrl);
 /* Reset the counter. */
    (void) R_CMT_Reset(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_CMT_StatusGet(&g_timer0_ctrl, &status);
}
```

CMT Callback example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

CMT Free Running Counter Example

To use the CMT as a free running counter, select periodic mode and set the the Period to 0xFFFF for a 16-bit timer.

```
void cmt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_CMT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_CMT_StatusGet(&g_timer0_ctrl, &status);
}
```

CMT Period Update Example

This an example of updating the period.

```
#define CMT_EXAMPLE_MSEC_PER_SEC (1000)
#define CMT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void cmt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKB frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_CMT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
```

```
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclk_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclk_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclk_freq_hz * CMT_EXAMPLE_DESIRED_PERIOD_MSEC) /
CMT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_CMT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	cmt_instance_ctrl_t

Data Structure Documentation

◆ cmt_instance_ctrl_t

struct cmt_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

Function Documentation

◆ R_CMT_Open()

fsp_err_t R_CMT_Open (timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg*)

Initializes the timer module and applies configurations. Implements timer_api_t::open.

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL or the source divider is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IRQ_BSP_DISABLED	timer_cfg_t::mode is TIMER_MODE_ONE_SHOT or timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.

◆ R_CMT_Stop()

fsp_err_t R_CMT_Stop (timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Return values

FSP_SUCCESS	Timer successfully stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMT_Start()

fsp_err_t R_CMT_Start (timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Return values

FSP_SUCCESS	Timer successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMT_Reset()

fsp_err_t R_CMT_Reset (timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMT_Enable()

fsp_err_t R_CMT_Enable (timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, event count, or restart the counter. Implements timer_api_t::enable.

Return values

FSP_ERR_UNSUPPORTED	API not supported by CMT.

◆ R_CMT_Disable()

fsp_err_t R_CMT_Disable (timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, event count or restart the counter. Implements timer_api_t::disable.

Note

The timer could be running after R_CMT_Disable(). To ensure it is stopped, call R_CMT_Stop().

Return values

FSP_ERR_UNSUPPORTED	API not supported by CMT.

◆ R_CMT_PeriodSet()

fsp_err_t R_CMT_PeriodSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts*)

Updates period. The new period is updated immediately and the counter is reset to 0. Period unit is raw. Other units will be converted to raw before setting.

Implements timer_api_t::periodSet.

Return values

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

◆ R_CMT_DutyCycleSet()

fsp_err_t R_CMT_DutyCycleSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin*)

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Return values

FSP_ERR_UNSUPPORTED	API not supported by CMT.

◆ R_CMT_CompareMatchSet()

fsp_err_t R_CMT_CompareMatchSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel*)

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

Return values

FSP_ERR_UNSUPPORTED	CMT compare match is not supported.

◆ R_CMT_InfoGet()

fsp_err_t R_CMT_InfoGet (timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info*)

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Return values

FSP_SUCCESS	Period, count direction, frequency, and ELC event written to caller's structure successfully.
FSP_ERR_ASSERTION	p_ctrl or p_info was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMT_StatusGet()

fsp_err_t R_CMT_StatusGet (timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status*)

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Return values

FSP_SUCCESS	Current timer state and counter value set successfully.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMT_CounterSet()

fsp_err_t R_CMT_CounterSet (timer_ctrl_t *const *p_ctrl*, uint32_t *counter*)

Set counter value.

Note

Do not call this API while the counter is counting.

Return values

FSP_SUCCESS	Counter value updated.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMT_CallbackSet()

fsp_err_t R_CMT_CallbackSet (timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_CMT_Close()

fsp_err_t R_CMT_Close (timer_ctrl_t *const *p_ctrl*)

Stops counter, clears internal driver data. Implements timer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

5.2.19.2 Compare Match Timer W (r_cmtw)

Modules » Timers

Functions

fsp_err_t	R_CMTW_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t	R_CMTW_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMTW_Start (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMTW_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMTW_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMTW_Disable (timer_ctrl_t *const p_ctrl)

fsp_err_t	R_CMTW_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_CMTW_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_CMTW_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t	R_CMTW_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t	R_CMTW_OutputEnable (timer_ctrl_t *const p_ctrl, cmtw_io_pin_t pin)
fsp_err_t	R_CMTW_OutputDisable (timer_ctrl_t *const p_ctrl, cmtw_io_pin_t pin)
fsp_err_t	R_CMTW_CallbackSet (timer_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t	R_CMTW_Close (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_CMTW_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel)

Detailed Description

Driver for the CMTW peripherals on RX MCUs. This module implements the Timer Interface.

Overview

The Compare Match Timer W (CMTW) module can be used to count events, measure external input signals, generate a periodic interrupt, and output a periodic to a TOC pin. This module supports two units (unit 0 and unit 1) with one channel of 32-bit.

Features

The CMTW module has the following features:

- Supports periodic mode, one-shot mode.
- Supports count source of PCLK
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Supports runtime reconfiguration of period.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, counting direction.
- APIs are provided to get the current timer status and counter value.
- Supports linking to the specified module, CMTW0 count start, event count, or count restart

by ELC events.

Selecting a Timer

RX MCUs have two timer peripherals: the General PWM Timer (GPT), the Compare Match Timer (CMT) and the Compare Match Timer W (CMTW). When selecting between them, consider these factors:

	GPT	CMT	CMTW	TMR
Low Power Modes	The GPT can operate in sleep mode.	The CMT can operate in sleep mode.	The CMTW can operate in sleep mode.	The TMR can operate in sleep mode.
Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 4 GPT channels.	All MCUs have 4 CMT channels.	All MCUs have 2 CMTW channels.	The number of TMR channels is device specific. All currently supported MCUs have at least 4 TMR channels.
Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The CMT timers are 16-bit timers.	The CMTWs timers are 32-bit timers.	The TMR timers are 8-bit timers or 16-bit timers.
Clock Source	The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The CMT runs off PCLKB.	The CMTW runs off PCLKB.	The TMR runs off PCLKB.

Configuration

Build Time Configurations for r_cmtw

The following build time configurations are defined in fsp_cfg/r_cmtw_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Timer, Compare Match Timer W (r_cmtw)

This module can be added to the Stacks tab via New Stack > Timers > Timer, Compare Match Timer W (r_cmtw).

Configuration	Options	Default	Description
General > Name	Name must be a valid	g_timer0	Module name.

	C symbol		
General > Channel	Value must be a non-negative integer	0	Specify the hardware channel.
General > Mode	• Periodic • One-Shot	Periodic	Mode selection. Note: One-shot mode is implemented in software. ISR's must be enabled for one shot even if callback is unused.
General > Period	Value must be non-negative	0x10000	Specify the timer period based on the selected unit.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Raw Counts	Unit of the period specified above
General > Counter Size	• 32-bit • 16-bit	32-bit	Size of the timer counter
General > Counter Clear Source	• Compare Match • Disabled • Input Capture0 • Input Capture1 • Output Compare0 • Output Compare1	Compare Match	Select the Counter Clear Source.
Output > TOC0 Output	• Disabled • Start Level Low and Toggle Output • Start Level High and Toggle Output	Disabled	Configure TOC0 output.
Output > TOC1 Output	• Disabled • Start Level Low and Toggle Output • Start Level High and Toggle Output	Disabled	Configure TOC1 output.
Output > TOC0 Output Enable	• Output Disable • Output Enable	Output Disable	Configure TOC0 output enable.
Output > TOC1 Output Enable	• Output Disable • Output Enable	Output Disable	Configure TOC1 output enable.

Input > TIC0 Capture Trigger	• Rising Edge • Falling Edge • Both Edge	Rising Edge	Select the trigger edge.
Input > TIC1 Capture Trigger	• Rising Edge • Falling Edge • Both Edge	Rising Edge	Select the trigger edge.
Input > TIC0 Input Enable	• Input Disable • Input Enable	Input Disable	Configure TIC0 output enable.
Input > TIC1 Input Enable	• Input Disable • Input Enable	Input Disable	Configure TIC1 output enable.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses.
Interrupts > Compare Match Interrupt Priority	MCU Specific Options		Select the compare match interrupt priority.
Interrupts > Input Capture 0 Interrupt Priority	MCU Specific Options		Select the input capture 0 interrupt priority.
Interrupts > Input Capture 1 Interrupt Priority	MCU Specific Options		Select the input capture 1 interrupt priority.
Interrupts > Output Compare 0 Interrupt Priority	MCU Specific Options		Select the output compare 0 interrupt priority.
Interrupts > Output Compare 1 Interrupt Priority	MCU Specific Options		Select the output compare 1 interrupt priority.

Clock Configuration

The CMTW clock is used to the PCLKB frequency. You can set the PCLKB frequency using the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module can use CMWCOR, CMWICR0, CMWICR1, CMWOCR0, and CMWOCR1 event as counter clear source.

This module can use TIC0, TIC1, TIC2, and TIC3 as input pins to measure input signals.

This module can use TOC0, TOC1, TOC2, and TOC3 as output pins for periodic signals.

Usage Notes

Maximum Period

The RX Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units and clock speed.

When the selected period unit is "Raw counts", the maximum period setting is 0x2000000 for 16-bit or 0x20000000000 for 32-bit. This will configure the timer with the maximum period and a count clock divisor of 512.

Note

When manually changing the timer period counts the maximum value for a 16-bit CMTW is 0x10000, and 32-bit is 0x100000000. This number overflows the 16-bit or 32-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.

Updating Period

The period is updated after the next counter overflow after calling R_CMTW_PeriodSet(). To force them to update before the next counter overflow, call R_CMTW_Reset() while the counter is running.

One-Shot Mode

The CMTW timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Transfer (r_dtc).

One-Shot Mode Output

Examples of one-shot signals that can be generated by this module are shown below:

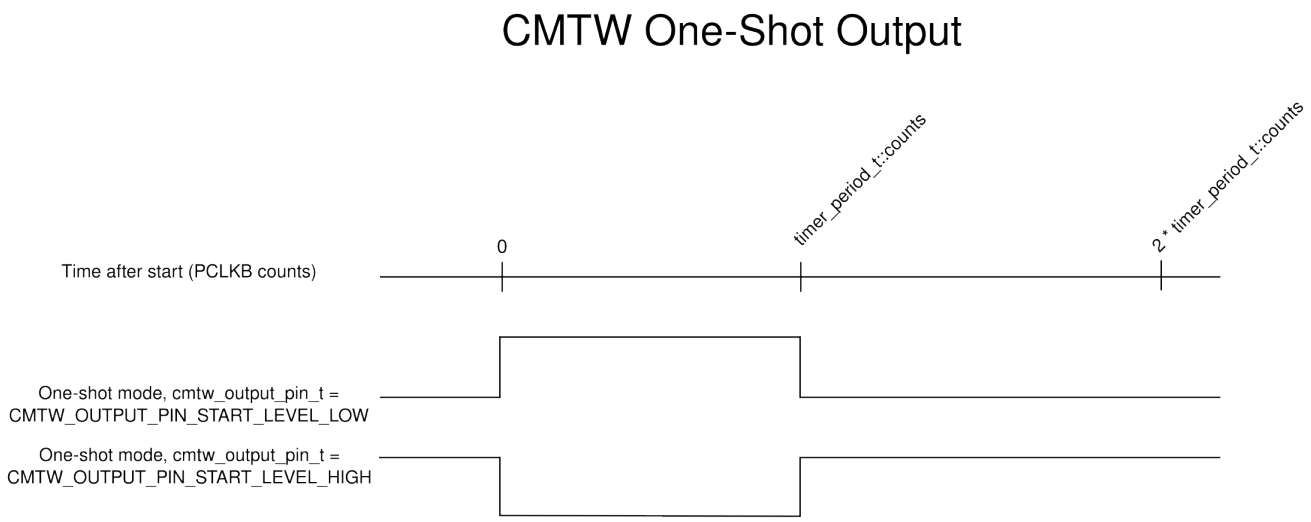


Figure 154: CMTW One-Shot Output

Periodic Mode

Frequency dividing clock is selected from four frequency dividing clocks obtained by dividing the

peripheral module clock (PCLKB), the CMWCNT counter starts counting up using the selected clock. When clearing of the counter is selected by counter clear register. The CMWCNT counter becomes 0000 0000h and continues counting. Otherwise, an overflow is generated when FFFF FFFFh changes to 0000 0000h during 32-bit count operation and 0000 FFFFh changes to 0000 0000h during 16-bit operation, and the CMWCNT counter continues counting.

Periodic Mode Output:

The TOC pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square (from rising edge to rising edge) matches the period of the CMTW timer.

Examples of periodic signals that can be generated by this module are shown below:

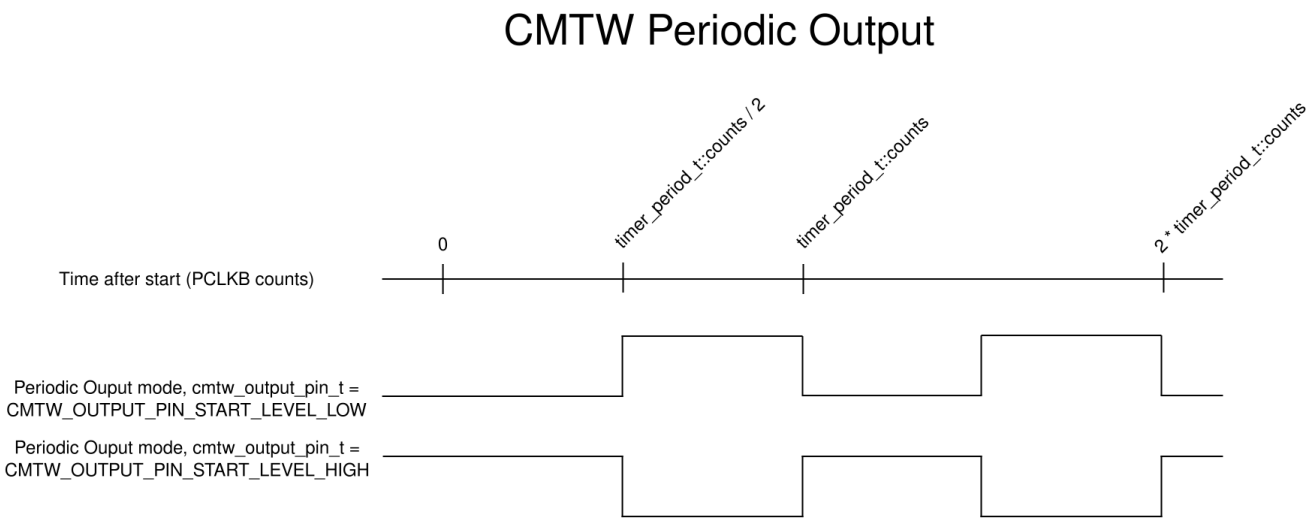


Figure 155: CMTW Periodic Output

Event Counting

Event counting can be done by selecting counting sources from CMWCR, ELC events.

Note

In event counting mode, the application must call R_CMTW_Start() to enable event counting. The counter will not change after calling R_CMTW_Start() until an event occurs.

Controlling CMTW with ELC Events (optional)

The CMTW timer can be configured to start, count, restart when an ELC event occurs.

Note

Event signal input and output are available for CMTW0 only (channel 0).

Enabling External Sources for Input Capture

R_CMTW_Enable() must be called when external sources are used for the input capture.

Examples

CMTW Basic Example

This is a basic example of minimal use of the CMTW in an application.

```
void cmtw_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMTW_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMTW_Start(&g_timer0_ctrl);
}
void cmtw_reset_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMTW_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMTW_Start(&g_timer0_ctrl);
 /* Reset the counter. */
    (void) R_CMTW_Reset(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_CMTW_StatusGet(&g_timer0_ctrl, &status);
}
```

CMTW Callback example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
```

```
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

CMTW Free Running Counter Example

To use the CMTW as a free running counter, select periodic mode and set the the Period to 0xFFFF for a 16-bit timer or 0xFFFFFFFF for a 32-bit timer.

```
void cmtw_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMTW_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMTW_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_CMTW_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_CMTW_StatusGet(&g_timer0_ctrl, &status);
}
```

CMTW Input Capture Example

This is an example of using the CMTW to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
```

```
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if ((TIMER_EVENT_CAPTURE_A == p_args->event) || (TIMER_EVENT_CAPTURE_B ==
p_args->event))
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_CMTW_InfoGet(&g_timer0_ctrl, &info);
       uint64_t period = info.period_counts;
 /* The maximum period is one more than the maximum 32-bits number, but will be
reflected as 0 in
  * timer_info_t::period_counts. */
 if (0U == period)
      {
            period = UINT32_MAX + 1U;
      }
       g_captured_time    = (period * g_capture_overflows) + p_args->capture;
       g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
       g_capture_overflows++;
    }
}
void cmtw_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMTW_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
```

```
    (void) R_CMTW_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_CMTW_Disable(&g_timer0_ctrl);
}
```

CMTW Period Update Example

This an example of updating the period.

```
#define CMTW_EXAMPLE_MSEC_PER_SEC (1000)
#define CMTW_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void cmtw_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_CMTW_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_CMTW_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKB frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_CMTW_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclk_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
```

```
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclk_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclk_freq_hz * CMTW_EXAMPLE_DESIRED_PERIOD_MSEC) /
CMTW_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_CMTW_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	cmtw_instance_ctrl_t
struct	cmtw_extended_cfg_t

Enumerations

enum	cmtw_source_edge_t
enum	cmtw_output_pin_t
enum	cmtw_clear_source_t
enum	cmtw_io_pin_t
enum	cmtw_output_control_t
enum	cmtw_input_control_t

Data Structure Documentation

◆ cmtw_instance_ctrl_t

struct cmtw_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

◆ cmtw_extended_cfg_t

struct cmtw_extended_cfg_t		
Optional CMTW extension data structure.		
Data Fields		
cmtw_output_pin_t	toc0	Configure TOC0 pin.

cmtw_output_pin_t	toc1	Configure TOC1 pin.
cmtw_output_control_t	toc0_control	Output Compare 0 enable setting.
cmtw_output_control_t	toc1_control	Output Compare 1 enable setting.
cmtw_source_edge_t	capture_ic0_source	Event sources that trigger capture of TIC0.
cmtw_source_edge_t	capture_ic1_source	Event sources that trigger capture of TIC1.
cmtw_input_control_t	ic0_control	Input Capture 0 enable setting.
cmtw_input_control_t	ic1_control	Input Capture 1 enable setting.
cmtw_clear_source_t	clear_source	Counter clear source.
timer_variant_t	counter_size	Timer counter size.
uint8_t	capture_ic0_ipl	Input Capture 0 interrupt priority.
uint8_t	capture_ic1_ipl	Input Capture 1 interrupt priority.
IRQn_Type	capture_ic0_irq	Input Capture 0 interrupt.
IRQn_Type	capture_ic1_irq	Input Capture 1 interrupt.
uint8_t	compare_oc0_ipl	Output Compare 0 interrupt priority.
uint8_t	compare_oc1_ipl	Output Compare 1 interrupt priority.
IRQn_Type	compare_oc0_irq	Output Compare 0 interrupt.
IRQn_Type	compare_oc1_irq	Output Compare 1 interrupt.

Enumeration Type Documentation

◆ cmtw_source_edge_t

enum cmtw_source_edge_t	
Trigger edge for Input capture function.	
Enumerator	
CMTW_SOURCE_EDGE_RISING	Input capture trigger on rising edge.
CMTW_SOURCE_EDGE_FALLING	Input capture trigger on falling edge.
CMTW_SOURCE_EDGE_BOTH	Input capture trigger on both edge.

◆ cmtw_output_pin_t

enum cmtw_output_pin_t	
Level of CMTW pin	
Enumerator	
CMTW_OUTPUT_PIN_RETAIN	Retain output value.
CMTW_OUTPUT_PIN_START_LEVEL_LOW	Pin level low.
CMTW_OUTPUT_PIN_START_LEVEL_HIGH	Pin level high.

◆ cmtw_clear_source_t

enum cmtw_clear_source_t	
Clear source of CMWCNT	
Enumerator	
CMTW_CLEAR_SOURCE_COMPARE_MATCH_CMWCOR	Counter is cleared by CMWCOR compare match.
CMTW_CLEAR_SOURCE_DISABLED	Counter is not cleared.
CMTW_CLEAR_SOURCE_INPUT_CAPTURE_CMWICR0	Counter is cleared by CMWICR0 input capture.
CMTW_CLEAR_SOURCE_INPUT_CAPTURE_CMWICR1	Counter is cleared by CMWICR1 input capture.
CMTW_CLEAR_SOURCE_COMPARE_MATCH_CMWOCR0	Counter is cleared by CMWOCR0 output compare.
CMTW_CLEAR_SOURCE_COMPARE_MATCH_CMWOCR1	Counter is cleared by CMWOCR1 output compare.

◆ cmtw_io_pin_t

enum cmtw_io_pin_t	
Output pins, used to select	
Enumerator	
CMTW_IO_PIN_TOC0	TOC0.
CMTW_IO_PIN_TOC1	TOC1.
CMTW_IO_PIN_TOC0_AND_TOC1	TOC0 and TOC1.

◆ cmtw_output_control_t

enum cmtw_output_control_t	
Enumerator	
CMTW_OUTPUT_CONTROL_DISABLED	OutputCompare disabled.
CMTW_OUTPUT_CONTROL_ENABLED	OutputCompare enabled.

◆ cmtw_input_control_t

enum cmtw_input_control_t	
Enumerator	
CMTW_INPUT_CONTROL_DISABLED	InputCapture disabled.
CMTW_INPUT_CONTROL_ENABLED	InputCapture enabled.

Function Documentation

◆ R_CMTW_Open()

fsp_err_t R_CMTW_Open (timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg*)

Initializes the CMTW module instance. Implements timer_api_t::open.

The CMTW hardware does not support one-shot functionality natively. The one-shot feature is therefore implemented in the CMTW HAL layer. For a timer configured as a one-shot timer, the timer is stopped upon the first timer expiration.

The CMTW implementation of the general timer can accept an optional cmtw_extended_cfg_t extension parameter.

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL or the period is not in the valid range of 1 to 0xFFFF in 16bit counter or 1 to 0xFFFFFFFF in 32bit counter.
FSP_ERR_ALREADY_OPEN	R_CMTW_Open has already been called for this p_ctrl.
FSP_ERR_IRQ_BSP_DISABLED	A required interrupt has not been enabled in the vector table.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel number is not available on CMTW.

◆ R_CMTW_Stop()

fsp_err_t R_CMTW_Stop (timer_ctrl_t *const *p_ctrl*)

Stops the timer. Implements timer_api_t::stop.

Return values

FSP_SUCCESS	Timer stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_Start()

fsp_err_t R_CMTW_Start (timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Return values

FSP_SUCCESS	Timer started.
FSP_ERR_ASSERTION	p_ctrl is null.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_Reset()

fsp_err_t R_CMTW_Reset (timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

Return values

FSP_SUCCESS	Counter reset.
FSP_ERR_ASSERTION	p_ctrl is NULL
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_Enable()

fsp_err_t R_CMTW_Enable (timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Return values

FSP_SUCCESS	External events successfully enabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMTW_Disable()

fsp_err_t R_CMTW_Disable (timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Return values

FSP_SUCCESS	External events successfully disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMTW_PeriodSet()

fsp_err_t R_CMTW_PeriodSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts*)

Updates period. The new period is updated immediately. Implements timer_api_t::periodSet.

Warning

Stop the timer before calling this function if one-shot output is used.

Return values

FSP_SUCCESS	Period value updated.
FSP_ERR_ASSERTION	A required pointer was NULL, or the period was not in the valid range of 1 to 0xFFFF in 16bit counter or 1 to 0xFFFFFFFF in 32bit counter.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_DutyCycleSet()

fsp_err_t R_CMTW_DutyCycleSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin*)

Updates duty cycle. If the timer is counting, the new duty cycle is reflected after the next counter underflow. Implements timer_api_t::dutyCycleSet.

Return values

FSP_ERR_UNSUPPORTED	CMTW duty cycle set is not supported.

◆ R_CMTW_InfoGet()

fsp_err_t R_CMTW_InfoGet (timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info*)

Gets timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Return values

FSP_SUCCESS	Period, count direction, and frequency stored in p_info.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_StatusGet()

fsp_err_t R_CMTW_StatusGet (timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status*)

Retrieves the current state and counter value stores them in p_status. Implements timer_api_t::statusGet.

Return values

FSP_SUCCESS	Current status and counter value provided in p_status.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_OutputEnable()

fsp_err_t R_CMTW_OutputEnable (timer_ctrl_t *const *p_ctrl*, cmtw_io_pin_t *pin*)

Enable output for TOC0 and/or TOC1.

Return values

FSP_SUCCESS	Output is enabled.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMTW_OutputDisable()

fsp_err_t R_CMTW_OutputDisable (timer_ctrl_t *const *p_ctrl*, cmtw_io_pin_t *pin*)

Disable output for TOC0 and/or TOC1.

Return values

FSP_SUCCESS	Output is disabled.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_CMTW_CallbackSet()

fsp_err_t R_CMTW_CallbackSet (timer_ctrl_t *const *p_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_CMTW_Close()

fsp_err_t R_CMTW_Close (timer_ctrl_t *const *p_ctrl*)

Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements timer_api_t::close.

Return values

FSP_SUCCESS	Timer closed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_CMTW_CompareMatchSet()

fsp_err_t R_CMTW_CompareMatchSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel*)

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

Return values

FSP_ERR_UNSUPPORTED	CMTW compare match is not supported.

5.2.19.3 Timer, General PWM (r_gpt)

Modules » Timers

Functions

fsp_err_t	R_GPT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t	R_GPT_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Start (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Disable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_GPT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_GPT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t	R_GPT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t	R_GPT_CounterSet (timer_ctrl_t *const p_ctrl, uint32_t counter)
fsp_err_t	R_GPT_OutputEnable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin)
fsp_err_t	R_GPT_OutputDisable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin)

fsp_err_t	R_GPT_AdcTriggerSet (timer_ctrl_t *const p_ctrl, gpt_adc_compare_match_t which_compare_match, uint32_t compare_match_value)
fsp_err_t	R_GPT_PwmOutputDelaySet (timer_ctrl_t *const p_ctrl, gpt_pwm_output_delay_edge_t edge, gpt_pwm_output_delay_setting_t delay_setting, uint32_t const pin)
fsp_err_t	R_GPT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t	R_GPT_Close (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_PwmOutputDelayInitialize (void)
fsp_err_t	R_GPT_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel)

Detailed Description

Driver for the GPT32 and GPT16 peripherals on RX MCUs. This module implements the Timer Interface.

Overview

The GPT module can be used to count events, measure external input signals, generate a periodic interrupt, or output a periodic or PWM signal to a GTIOC pin.

This module supports the GPT peripherals GPT32EH, GPT32E, GPT32, and GPT16. GPT16 is a 16-bit timer. The other peripherals (GPT32EH, GPT32E, and GPT32) are 32-bit timers. The 32-bit timers are all treated the same in this module from the API perspective.

Features

The GPT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Supports count source of PCLK, GTETRG pins, GTIOC pins, or ELC events.
- Supports debounce filter on GTIOC pins.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports runtime reconfiguration of compare match value.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Supports start, stop, clear, count up, count down, and capture by external sources from GTETRG pins, GTIOC pins, or ELC events.

- Supports symmetric and asymmetric PWM waveform generation.
- Supports One shot synchronous pulse waveform generation.
- Supports automatic addition of dead time.
- Supports generating ELC events to start an ADC scan at a compare match value (see ref ELC) and updating the compare match value.
- Supports linking with a POEG channel to automatically disable GPT output when an error condition is detected.
- Supports setting the counter value while the timer is stopped.
- Supports enabling and disabling output pins.
- Supports skipping up to seven overflow/underflow (crest/trough) interrupts at a time
- Supports generating custom PWM waveforms by configuring the pin's output level at each compare match and cycle end.

Selecting a Timer

RX MCUs have some timer peripherals: the General PWM Timer (GPT), the Compare Match Timer (CMT), the Compare Match Timer W (CMTW) and the 8-Bit Timer (TMR). When selecting between them, consider these factors:

	GPT	CMT	CMTW	TMR
Low Power Modes	The GPT can operate in sleep mode.	The CMT can operate in sleep mode.	The CMTW can operate in sleep mode.	The TMR can operate in sleep mode.
Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 4 GPT channels.	All MCUs have 4 CMT channels.	All MCUs have 2 CMTW channels.	The number of TMR channels is device specific. All currently supported MCUs have at least 4 TMR channels.
Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The CMT timers are 16-bit timers.	The CMTWs timers are 32-bit timers.	The TMR timers are 8-bit timers or 16-bit timers.
Clock Source	The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The CMT runs off PCLKB.	The CMTW runs off PCLKB.	The TMR runs off PCLKB.

Configuration

Build Time Configurations for r_gpt

The following build time configurations are defined in fsp_cfg/r_gpt_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Pin Output Support	• Disabled • Enabled • Enabled with Extra Features	Disabled	Enables or disables support for outputting PWM waveforms on GTIOCx pins. The "Enabled with Extra Features" option enables support for Triangle wave modes and also enables the features located in the "Extra Features" section of each module instance.
Write Protect Enable	• Enabled • Disabled	Disabled	If selected write protection is applied to all GPT channels.

Configurations for Timers > Timer, General PWM (r_gpt)

This module can be added to the Stacks tab via New Stack > Timers > Timer, General PWM (r_gpt).

Configuration	Options	Default	Description
General > Compare Match > Compare Match A > Status	• Enabled • Disabled	Disabled	
General > Compare Match > Compare Match A > Compare match value	Value must be a non-negative integer less than or equal to 0x40000000000	0	Specify the compare match A value in units that selected in Period Unit section.
General > Compare Match > Compare Match B > Status	• Enabled • Disabled	Disabled	
General > Compare Match > Compare Match B > Compare match value	Value must be a non-negative integer less than or equal to 0x40000000000	0	Specify the compare match B value in units that selected in Period Unit section.
General > Name	Name must be a valid C symbol	g_timer0	Module name.
General > Channel	Enter the supported Channel number	0	Specify the hardware channel.
General > Mode	• Periodic • One-Shot • One-Shot Pulse • Saw-wave PWM • Triangle-wave PWM	Periodic	Mode selection. Periodic: Generates periodic interrupts or square waves. One-shot: Generate a single interrupt or a

	(symmetric, Mode 1) • Triangle-wave PWM (asymmetric, Mode 2) • Triangle-wave PWM (asymmetric, Mode 3)		pulse wave. Note: One-shot mode is implemented in software. ISRs must be enabled for one-shot even if callback is unused. One-Shot Pulse: Counter performs saw-wave operation with fixed buffer operation. Saw-wave PWM: Generates basic saw-wave PWM waveforms. Triangle-wave PWM (symmetric, Mode 1): Generates symmetric PWM waveforms with duty cycle determined by compare match set with 32-bit transfer during a crest event and updated at the next trough with single or double buffer operation. Triangle-wave PWM (asymmetric, Mode 2): Generates asymmetric PWM waveforms with duty cycle determined by compare match set with 32-bit transfer during a crest/trough event and updated at the next trough/crest. Triangle-wave PWM (asymmetric, Mode 3): Generates PWM waveforms with duty cycle determined by compare match set with 64-bit transfer during a crest interrupt and updated at the next trough with fixed buffer operation.
General > Period	Value must be a non-negative integer less than or equal to 0x40000000000	0x10000	Specify the timer period in units selected below. Set the period to 0x100000000 (32-bit) or 0x10000 (16-bit) raw counts for a free running timer or

			an input capture configuration. The period can be set up to 0x40000000000 (32-bit) or 0x4000000 (16-bit) which will use a divider of 1024 with the maximum period. If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the generated timer_cfg_t structure.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Raw Counts	Unit of the period specified above
Output > Custom Waveform > GTIOA > Initial Output Level	• Pin Level Low • Pin Level High	Pin Level Low	Set the initial output level of GTIOCxA.
Output > Custom Waveform > GTIOA > Cycle End Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxA at cycle end.
Output > Custom Waveform > GTIOA > Compare Match Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxA at compare match.
Output > Custom Waveform > GTIOA > Retain Output Level at Count Stop	• Disabled • Enabled	Disabled	Retain the current GTIOxA output level when counting is stopped.
Output > Custom Waveform > GTIOB > Initial Output Level	• Pin Level Low • Pin Level High	Pin Level Low	Set the initial output level of GTIOCxB.
Output > Custom Waveform > GTIOB > Cycle End Output Level	• Pin Level Retain • Pin Level Low • Pin Level High	Pin Level Retain	Set the output level of GTIOCxB at cycle end.

	• Pin Level Toggle		
Output > Custom Waveform > GTIOB > Compare Match Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxB at compare match.
Output > Custom Waveform > GTIOB > Retain Output Level at Count Stop	• Disabled • Enabled	Disabled	Retain the current GTIOxB output level when counting is stopped.
Output > Custom Waveform > Custom Waveform Enable	• Disabled • Enabled	Disabled	Enable custom waveform configuration.
Output > Duty Cycle Percent (only applicable in PWM mode)	Value must be between 0 and 100	50	Specify the timer duty cycle percent. Only used in PWM mode.
Output > GTIOCA Output Enabled	• True • False	False	Enable the output of GTIOCA on a pin.
Output > GTIOCA Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
Output > GTIOCB Output Enabled	• True • False	False	Enable the output of GTIOCB on a pin.
Output > GTIOCB Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
Input > Count Up Source	MCU Specific Options		Select external source that will increment the counter. If any count up source is selected, the timer will count the external sources only. It will not count PCLKD cycles.
Input > Count Down Source	MCU Specific Options		Select external source that will decrement the counter. If any count down source is selected, the timer will count the external sources only. It will not count PCLKD cycles.
Input > Start Source	MCU Specific Options		Select external source that will start the timer. For pulse width

			measurement, set the Start Source and the Clear Source to the trigger edge (the edge to start the measurement), and set the Stop Source and Capture Source (either A or B) to the opposite edge (the edge to stop the measurement). For pulse period measurement, set the Start Source, the Clear Source, and the Capture Source (either A or B) to the trigger edge (the edge to start the measurement).
Input > Stop Source	MCU Specific Options		Select external source that will stop the timer.
Input > Clear Source	MCU Specific Options		Select external source that will clear the timer.
Input > Capture A Source	MCU Specific Options		Select external source that will trigger a capture A event.
Input > Capture B Source	MCU Specific Options		Select external source that will trigger a capture B event.
Input > Noise Filter A Sampling Clock Select	• No Filter • Filter PCLKD / 1 • Filter PCLKD / 4 • Filter PCLKD / 16 • Filter PCLKD / 64	No Filter	Select the input filter for GTIOCA.
Input > Noise Filter B Sampling Clock Select	• No Filter • Filter PCLKD / 1 • Filter PCLKD / 4 • Filter PCLKD / 16 • Filter PCLKD / 64	No Filter	Select the input filter for GTIOCB.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be specified here. If this callback function is provided, it will be called from the

			interrupt service routine (ISR) each time the timer period elapses
Interrupts > Overflow/Crest Interrupt Priority	MCU Specific Options		Select the overflow interrupt priority. This is the crest interrupt for triangle-wave PWM.
Interrupts > Capture/Compare match A Interrupt Priority	MCU Specific Options		Select the interrupt priority for Capture/Compare match A.
Interrupts > Capture/Compare match B Interrupt Priority	MCU Specific Options		Select the interrupt priority for Capture/Compare match B.
Interrupts > Underflow/Trough Interrupt Priority	MCU Specific Options		Select the interrupt priority for the trough interrupt (triangle-wave PWM only).
Extra Features > Output Disable > POEG Link	• POEG Channel 0 • POEG Channel 1 • POEG Channel 2 • POEG Channel 3	POEG Channel 0	Select which POEG to link this GPT channel to.
Extra Features > Output Disable > Output Disable POEG Trigger	• Dead Time Error • GTIOCA and GTIOCB High Level • GTIOCA and GTIOCB Low Level		Select which errors send an output disable trigger to POEG. Dead time error is only available GPT channels that have GTINTAD.GRPDTE.
Extra Features > Output Disable > GTIOCA Disable Setting	• Disable Prohibited • Set Hi Z • Level Low • Level High	Disable Prohibited	Select the disable setting for GTIOCA.
Extra Features > Output Disable > GTIOCB Disable Setting	• Disable Prohibited • Set Hi Z • Level Low • Level High	Disable Prohibited	Select the disable setting for GTIOCB.
Extra Features > ADC Trigger > Start Event Trigger (Channels with	• Trigger Event A/D Converter Start Request A		Select which A/D converter start request interrupts to generate

GTINTAD only)	During Up Counting • Trigger Event A/D Converter Start Request A During Down Counting • Trigger Event A/D Converter Start Request B During Up Counting • Trigger Event A/D Converter Start Request B During Down Counting		and at which point in the cycle to generate them.
Extra Features > ADC Trigger > ADC A Compare Match (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the compare match value that generates a GPTn AD TRIG A event.
Extra Features > ADC Trigger > ADC B Compare Match (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the compare match value that generates a GPTn AD TRIG B event.
Extra Features > Dead Time (Value range varies with Channel) > Dead Time Count Up (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during up counting. This value also applies during down counting for channels that do not have GTDVD. The dead time count up value can be set up to 0xffffffff (32-bit) or 0xffff (16-bit).
Extra Features > Dead Time (Value range varies with Channel) > Dead Time Count Down (Raw Counts) (Channels with GTDVD only)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during down counting. The dead time count down value can be set up to 0xffffffff (32-bit) or 0xffff (16-bit).
Extra Features > Interrupt Skipping (Channels with GTITC only) > Interrupt to Count	• None • Overflow and Underflow (sawtooth) • Crest (triangle) • Trough (triangle)	None	Select the count source for interrupt skipping. The interrupt skip counter increments after each source event. All crest/overflow and

			trough/underflow interrupts are skipped when the interrupt skip counter is non-zero.
Extra Features > Interrupt Skipping (Channels with GTITC only) > Interrupt Skip Count	• 0 • 1 • 2 • 3 • 4 • 5 • 6 • 7	0	Select the number of interrupts to skip.
Extra Features > Interrupt Skipping (Channels with GTITC only) > Skip ADC Events	• None • ADC A Compare Match • ADC B Compare Match • ADC A and B Compare Match	module.driver.timer.interrupt_skip.adc.none	Select ADC events to suppress when the interrupt skip count is not zero.
Extra Features > Extra Features	• Enabled • Disabled	Disabled	Select whether to enable extra features on this channel.

Clock Configuration

The GPT clock is based on the PCLKD frequency. You can set the PCLKD frequency using the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, GTETRGD, GTIOCA and GTIOCB pins as count sources.

This module can use GTIOCA and GTIOCB pins as output pins for periodic or PWM signals.

This module can use GTIOCA and GTIOCB as input pins to measure input signals.

Usage Notes

Maximum Period for GPT

The RX Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units and clock speed.

When the selected period unit is "Raw counts", the maximum period setting is 0x40000000000. This will configure the timer with the maximum period and a count clock divisor of 1024.

Note

When manually changing the timer period counts the maximum value for GPT is 0x100000000. This number overflows the 32-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.

Updating Period and Duty Cycle

The period and duty cycle are updated after the next counter overflow after calling R_GPT_PeriodSet() or R_GPT_DutyCycleSet(). To force them to update before the next counter overflow, call R_GPT_Reset() while the counter is running.

Note

When manually changing the timer period counts the maximum value for a 32-bit GPT is 0x100000000. This number overflows the 32-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.
When changing the period and duty cycle at the same time when Parameter Checking is enabled, call R_GPT_PeriodSet first followed by R_GPT_DutyCycleSet to ensure input validation is performed correctly.

One-Shot Mode

The GPT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the ref DTC.

One-Shot Mode Output

The output waveform in one-shot mode is one PCLKD cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

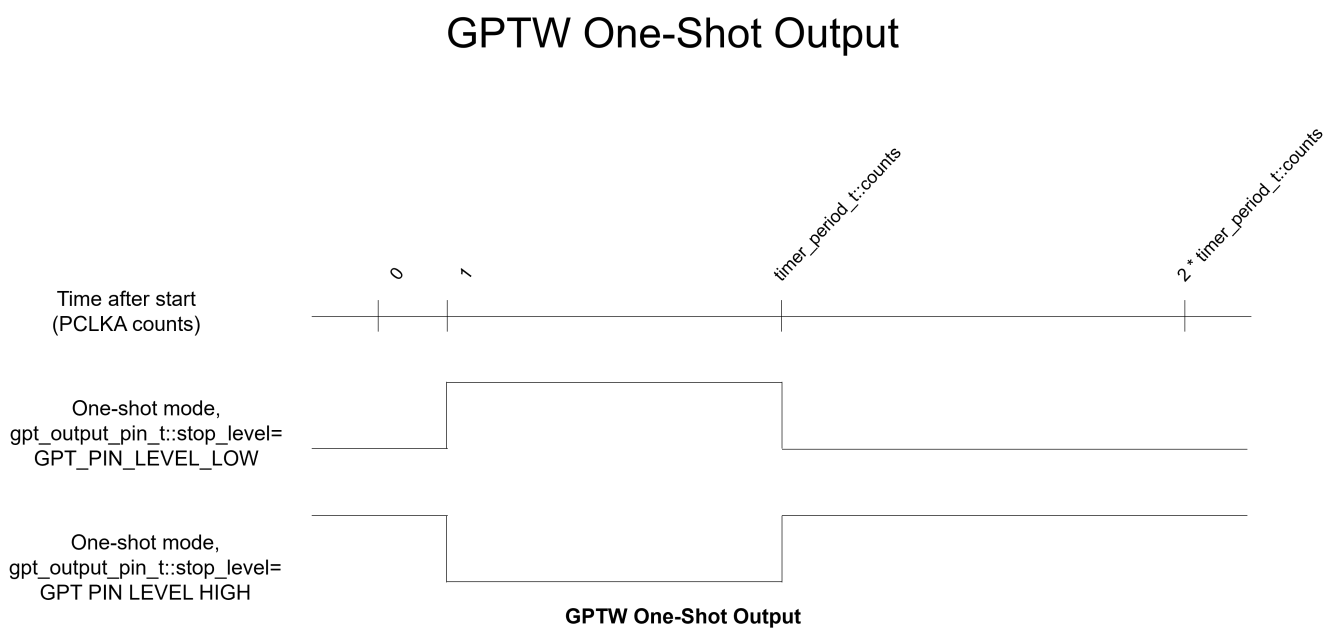


Figure 156: GPT One-Shot Output

One-Shot Pulse Mode

The one-shot pulse mode is an asymmetric PWM mode that provides more control over the rising and falling edges of the output. The user provides a period and initial output level and controls the signal by specifying compare match values for the leading and trailing edges each period.

Note

Despite its name, the One-Shot Pulse Mode operates continuously and does not stop after the first period.

One-Shot Pulse Mode Output

Examples of PWM signals that can be generated by this module are shown below. The leading and trailing edge match values can be modified using R_GPT_DutyCycleSet() in the overflow interrupt.

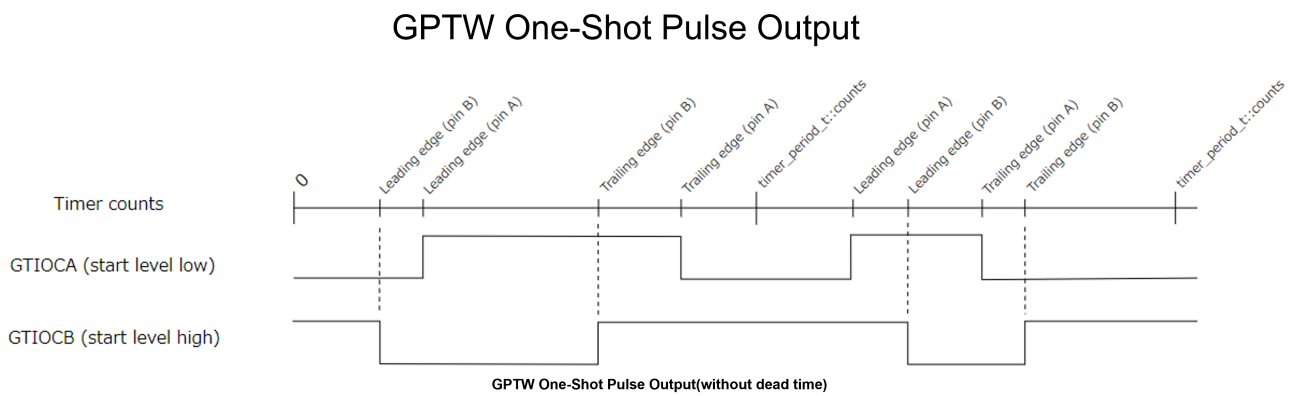


Figure 157: GPT One-Shot Pulse Output

If dead time is enabled only match values for GTIOCnA need to be set; the match values for GTIOCnB will be automatically configured in hardware.

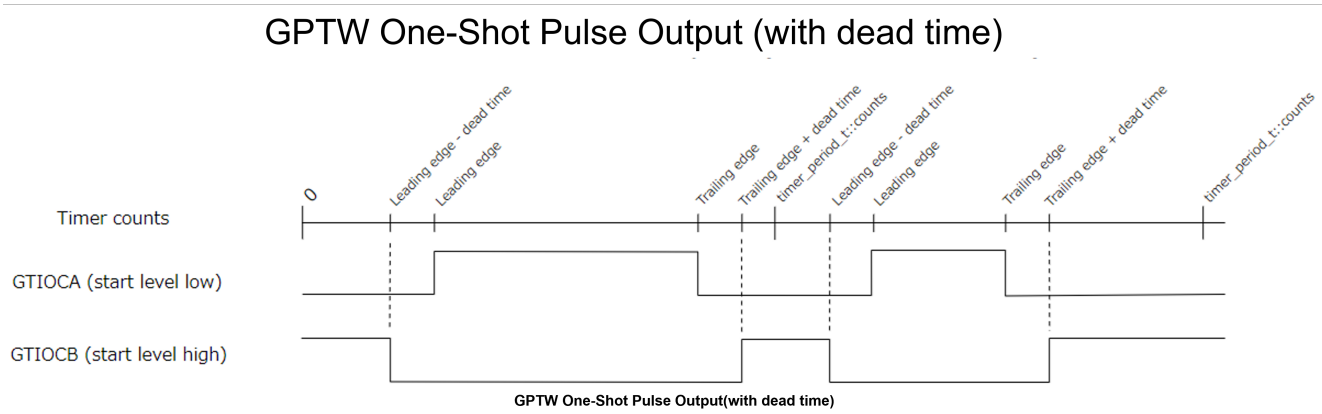


Figure 158: GPT One-Shot Pulse deadtime Output

Periodic Output

The GTIOC pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square wave (from rising edge to rising edge) matches the period of the GPT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

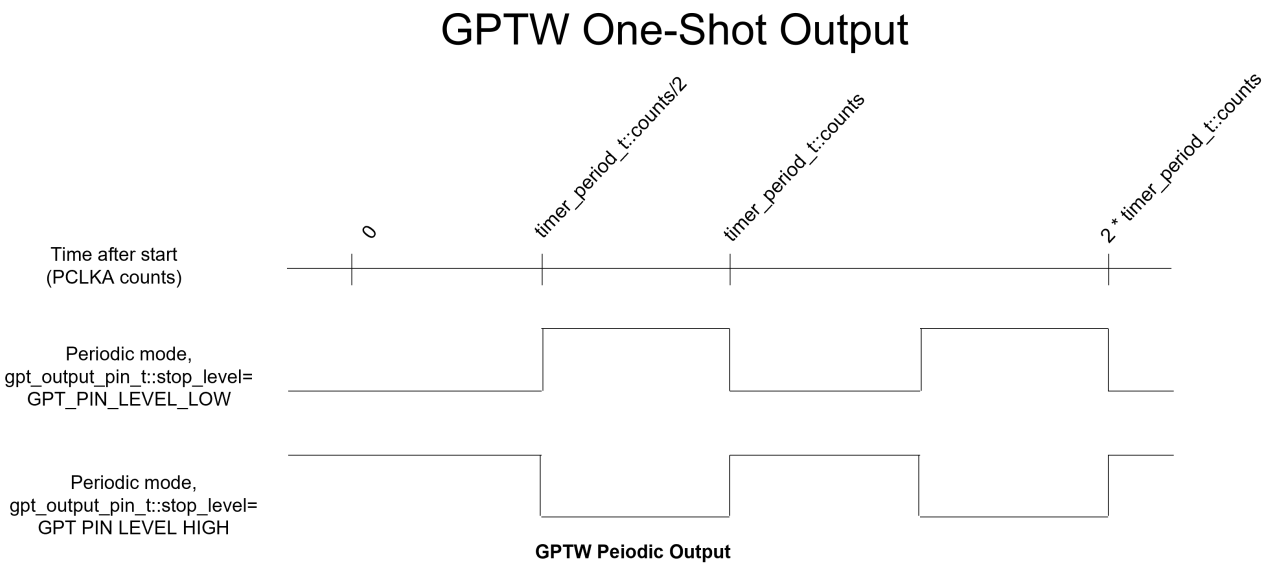


Figure 159: GPT Periodic Output

PWM Output

The PWM output signal is high at the beginning of the cycle and low at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

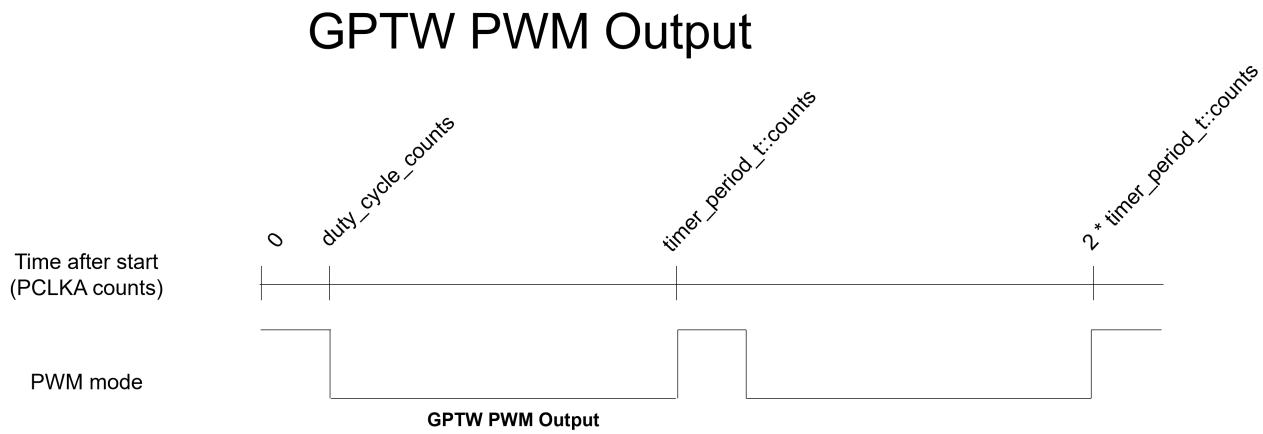


Figure 160: GPT PWM Output

Triangle-Wave PWM Output

Examples of PWM signals that can be generated by this module are shown below. The duty_cycle_counts can be modified using R_GPT_DutyCycleSet() in the crest interrupt and updated at the following trough for symmetric PWM or modified in both the crest/trough interrupts and updated at the following trough/crest for asymmetric PWM.

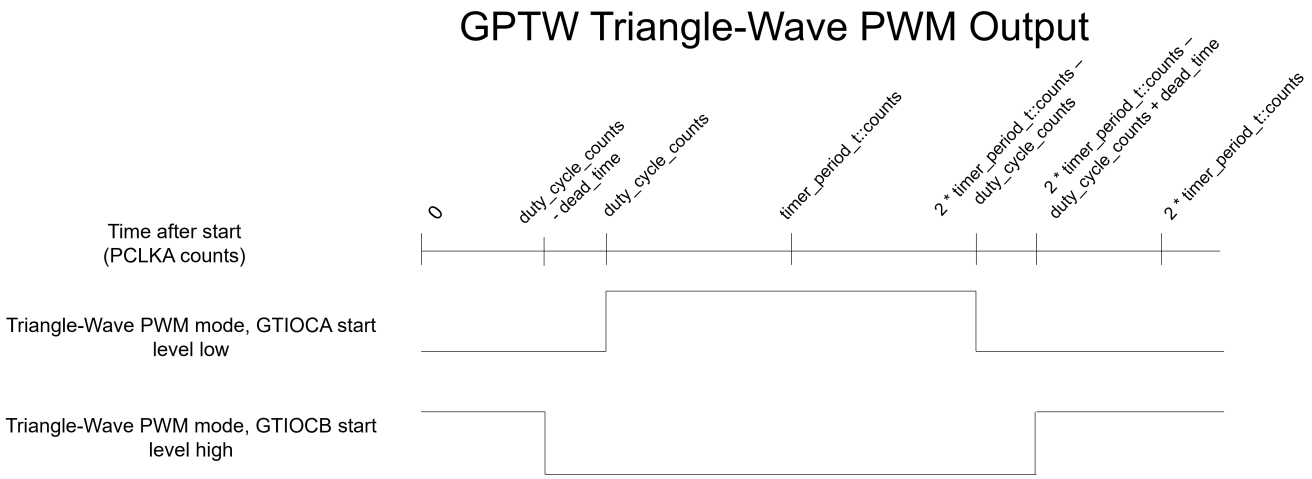


Figure 161: GPT Triangle-Wave PWM Output

PWM Output Delay Circuit

On select MCUs, an additional PWM output delay circuit can be configured in order to fine tune the rising and falling edge delays in increments of 1/32 times the period of the GPT core clock. The PWM output delay function must be configured prior to initializing the GPT channels using R_GPT_PwmOutputDelayInitialize.

Note

1. In Saw-wave PWM mode, the output delay setting cannot be changed while the capture compare setting (GTCCRn) is greater than or equal the period setting (GTPR) - 2.
2. In Triangle PWM modes, the output delay setting cannot be changed while the counter is counting down, and the capture compare setting (GTCCRn) is less than or equal to 2.
3. When the PWM Output Delay Circuit is enabled, the PWM signal is delayed by 3 GPT core clock cycles.
4. When the GPTCLK is used as the GPT core clock, the following delay is required between writes to the rising or falling edge output delay setting for a given pin: Write_Interval[ns] = Period_of_PCLKD [ns] × 6 + Period_of_GPTCLK [ns] × 4).

Event Counting

Event counting can be done by selecting up or down counting sources from GTETRG pins, ELC events, or GTIOC pins. In event counting mode, the GPT counter is not affected by PCLKD.

Note

In event counting mode, the application must call R_GPT_Start() to enable event counting. The counter will not change after calling R_GPT_Start() until an event occurs.

Pulse Measurement

If the capture edge occurs before the start edge in pulse measurement, the first capture is invalid (0).

Controlling GPT with GTETRG Edges

The GPT timer can be configured to stop, start, clear, count up, or count down when a GTETRG rising or falling edge occurs.

Note

The GTETRG pins are shared by all GPT channels.
GTETRG pins require POEG to be on (example code for this is provided in GPT Free Running Counter Example).
If input filtering is required on the GTETRG pins, that must also be handled outside this module.

Controlling GPT with ELC Events

The GPT timer can be configured to stop, start, clear, count up, or count down when an ELC event occurs.

Note

The configurable ELC GPT sources are shared by all GPT channels.
The event links for the ELC must be configured outside this module.

Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals.

Enabling External Sources for Start, Stop, Clear, or Capture

R_GPT_Enable() must be called when external sources are used for start, stop, clear, or capture.

Interrupt Skipping

When an interrupt skipping source is selected a hardware counter will increment each time the selected event occurs. Each interrupt past the first (up to the specified skip count) will be suppressed. If ADC events are selected for skipping they will also be suppressed except during the timer period leading to the selected interrupt skipping event (see below diagram).

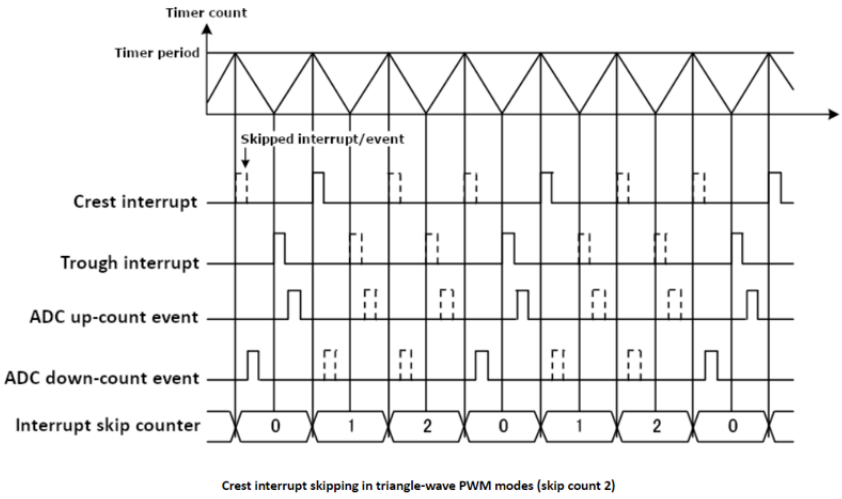


Figure 162: Crest interrupt skipping in triangle-wave PWM modes (skip count 2)

Complementary Output

By using the Custom Waveform option the output pins can be made to output complementary waveforms. To ensure these waveforms stay in sync, the duty cycle for both pins can be set simultaneously by calling R_GPT_DutyCycleSet() once with a pin parameter of GPT_IO_PIN_GTIOCA_AND_GTIOCB.

Note

The pin level for 0% and 100% duty cycle is determined by the Cycle End Output Level in normal PWM mode and the Initial Output Level in triangle PWM modes. 100% duty will output the configured level and 0% will output the opposite. Do not use Pin Level Toggle or Pin Level Retain for the Cycle End Output Level if normal PWM waveforms are desired.

Examples

GPT Basic Example

This is a basic example of minimal use of the GPT in an application.

```
void gpt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
}
```

GPT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

GPT Free Running Counter Example

To use the GPT as a free running counter, select periodic mode and set the the Period to 0xFFFFFFFF.

```
void gpt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If event count mode is used to count edges on a GTETRG pin, POEG must
be started to use GTETRG.
  * Reference Note 1 of Table "GPT functions" in the relevant hardware manual. */
 R_BSP_MODULE_START(FSP_IP_POEG, 0U);
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_GPT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
}
```

GPT Input Capture Example

This is an example of using the GPT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if ((TIMER_EVENT_CAPTURE_A == p_args->event) || (TIMER_EVENT_CAPTURE_B ==
p_args->event))
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
```

```
      (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
      uint64_t period = info.period_counts;
 /* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
  * timer_info_t::period_counts. */
 if (0U == period)
     {
           period = UINT32_MAX + 1U;
     }
      g_captured_time    = (period * g_capture_overflows) + p_args->capture;
      g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
      g_capture_overflows++;
    }
}
void gpt_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_GPT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_GPT_Disable(&g_timer0_ctrl);
}
```

GPT Period Update Example

This an example of updating the period.

```
#define GPT_EXAMPLE_MSEC_PER_SEC (1000)
#define GPT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void gpt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
  * - Use the R_GPT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
```

```
    assert(FSP_SUCCESS == err);
}
```

GPT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define GPT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void gpt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
  * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    assert(FSP_SUCCESS == err);
```

```
}
```

GPT A/D Converter Start Request Example

This is an example of using the GPT to start the ADC at a configurable A/D converter compare match value.

GPT One-Shot Pulse Mode Example

This example demonstrates the configuration and use of one-shot pulse mode with GPT timer.

```
/* Example callback called when timer overflows. */
void gpt_overflow_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
       g_timer_cycle_end_counter++;
 /* Use R_GPT_DutyCycleSet() API to set new values for each cycle.
  * - Use GPT_IO_PIN_ONE_SHOT_LEADING_EDGE to set the leading edge transition match
value (GTCCRC or GTCCRE register).
  * - Use GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE to set the trailing edge transition match
value (GTCCRD or GTCCRF register).
  */
    }
}
/* GTIOCA (configured for active-high) first and second pulse will be:
 * - 1/2 period, offset by 3/8 period
 * - 1/4 period, offset by 1/4 period */
#define GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_A ((g_timer0_cfg.period_counts * 3) / 8)
#define GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_A ((g_timer0_cfg.period_counts * 7) / 8)
#define GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_A ((g_timer0_cfg.period_counts * 1) / 4)
#define GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_A ((g_timer0_cfg.period_counts * 2) / 4)
/* GTIOCB (configured for active-low) first and second pulse will be:
 * - 1/2 period, offset by 1/4 period
 * - 1/4 period, offset by 1/2 period */
#define GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_B ((g_timer0_cfg.period_counts * 1) / 4)
#define GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_B ((g_timer0_cfg.period_counts * 3) / 4)
```

```
#define GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_B ((g_timer0_cfg.period_counts * 2) / 4)
#define GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_B ((g_timer0_cfg.period_counts * 3) / 4)
/* Configure and output a one-shot pulse that resembles what is shown in the example
above. */
void gpt_one_shot_pulse_mode_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    assert(FSP_SUCCESS == err);
 /* Set first edge transition count for GTIOCA and GTIOCB.
  *
  * Notes:
  * - Two temporary registers (A and B) are used for the second edge.
  * - Temporary register A and Temporary register B may only be loaded by shifting
from GTCCRD and GTCCRF, which is why the GPT_BUFFER_FORCE_PUSH is necessary
  * - If GPT_BUFFER_FORCE_PUSH is not used, timing for the first pulse will be
undefined.
  * - For more information, see "Example setting for saw-wave one-shot pulse mode" in
the UM for your MCU. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
 /* Set third edge transition count for GTIOCA and GTIOCB
  * Also force-push to shift all four counts into active buffer locations. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE);
    assert(FSP_SUCCESS == err);
```

```
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE | GPT_BUFFER_FORCE_PUSH);
    assert(FSP_SUCCESS == err);
 /* Set third leading edge transition count for GTIOCA and GTIOCB.
  * Note: It is not necessary to set third and fourth transition edges if it is not
required by the application. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
 /* Set fourth edge transition count for GTIOCA and GTIOCB
  * Do not force-push, shifting third and fourth edges would overwrite active buffer
locations (first and second edge counts). */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE);
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
 R_GPT_Start(&g_timer0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Wait for the one-shot pulse output to complete then stop GPT
  *
  * Notes:
  * - FSP driver does not currently support automatically stopping one-shot pulse. If
GPT is not stopped the last waveform timing will be repeated indefinitely.
```

```
  * - Some (more recent) MCUs added a new register (GTPC) that could be used to
automatically stop GPT after a set number of periods. Software driver support for
this is scheduled to be added in the future.
  * - The last transition edge timings will continue to be used indefinitely until
the user application stops GPT operation. */
 while (g_timer_cycle_end_counter < 2)
    {
 /* Wait for one-shot pulse output to complete (two pulses) */
    }
 /* (Optional) Stop the timer. */
    err = R_GPT_Stop(&g_timer0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

GPT Compare Match Set Example

This example demonstrates the configuration and use of compare match with GPT timer.

```
/* Example callback called when compare match occurs. */
void gpt_compare_match_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_COMPARE_A == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
#define GPT_COMPARE_MATCH_EXAMPLE_VALUE (0x2000U)
void gpt_compare_match_set_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the compare match value (GPT_COMPARE_MATCH_EXAMPLE_VALUE). This value must be
```

```
less than or equal to period value. */
    err = R_GPT_CompareMatchSet(&g_timer0_ctrl, GPT_COMPARE_MATCH_EXAMPLE_VALUE,
TIMER_COMPARE_MATCH_A);
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_GPT_Stop(&g_timer0_ctrl);
}
```

Data Structures

struct	gpt_output_pin_t
struct	gpt_gtior_setting_t
struct	gpt_instance_ctrl_t
struct	gpt_extended_pwm_cfg_t
struct	gpt_extended_cfg_t

Enumerations

enum	gpt_io_pin_t
enum	gpt_buffer_force_push
enum	gpt_pin_level_t
enum	gpt_source_t
enum	gpt_capture_filter_t
enum	gpt_adc_trigger_t
enum	gpt_poeg_link_t
enum	gpt_output_disable_t
enum	gpt_gtioc_disable_t
enum	gpt_adc_compare_match_t
enum	gpt_interrupt_skip_source_t

enum	gpt_interrupt_skip_count_t
enum	gpt_interrupt_skip_adc_t
enum	gpt_pwm_output_delay_setting_t
enum	gpt_pwm_output_delay_edge_t

Data Structure Documentation

◆ gpt_output_pin_t

struct gpt_output_pin_t		
Configurations for output pins.		
Data Fields		
bool	output_enabled	Set to true to enable output, false to disable output.
gpt_pin_level_t	stop_level	Select a stop level from gpt_pin_level_t.

◆ gpt_gtior_setting_t

struct gpt_gtior_setting_t
Custom GTIOR settings used for configuring GTIOCxA and GTIOCxB pins.

◆ gpt_instance_ctrl_t

struct gpt_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

◆ gpt_extended_pwm_cfg_t

struct gpt_extended_pwm_cfg_t		
GPT extension for advanced PWM features.		
Data Fields		
uint8_t	trough_ipl	Trough interrupt priority.
IRQn_Type	trough_irq	Trough interrupt.
gpt_poeg_link_t	poeg_link	Select which POEG channel controls output disable for this GPT channel.
gpt_output_disable_t	output_disable	Select which trigger sources request output disable from POEG.
gpt_adc_trigger_t	adc_trigger	Select trigger sources to start

		A/D conversion.
uint32_t	dead_time_count_up	Set a dead time value for counting up.
uint32_t	dead_time_count_down	Set a dead time value for counting down (available on GPT32E and GPT32EH only)
uint32_t	adc_a_compare_match	Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_A.
uint32_t	adc_b_compare_match	Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_B.
gpt_interrupt_skip_source_t	interrupt_skip_source	Interrupt source to count for interrupt skipping.
gpt_interrupt_skip_count_t	interrupt_skip_count	Number of interrupts to skip between events.
gpt_interrupt_skip_adc_t	interrupt_skip_adc	ADC events to skip when interrupt skipping is enabled.
gpt_gtioc_disable_t	gtioca_disable_setting	Select how to configure GTIOCA when output is disabled.
gpt_gtioc_disable_t	gtiocb_disable_setting	Select how to configure GTIOCB when output is disabled.

◆ gpt_extended_cfg_t

struct gpt_extended_cfg_t		
GPT extension configures the output pins for GPT.		
Data Fields		
gpt_output_pin_t	gtioca	Configuration for GPT I/O pin A.
gpt_output_pin_t	gtiocb	Configuration for GPT I/O pin B.
gpt_source_t	start_source	Event sources that trigger the timer to start.
gpt_source_t	stop_source	Event sources that trigger the timer to stop.
gpt_source_t	clear_source	Event sources that trigger the timer to clear.
gpt_source_t	capture_a_source	Event sources that trigger capture of GTIOCA.
gpt_source_t	capture_b_source	Event sources that trigger capture of GTIOCB.

gpt_source_t	count_up_source	Event sources that trigger a single up count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK.
gpt_source_t	count_down_source	Event sources that trigger a single down count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK.
gpt_capture_filter_t	capture_filter_gtioca	
gpt_capture_filter_t	capture_filter_gtiocb	
uint8_t	capture_a_ipl	Capture A interrupt priority.
uint8_t	capture_b_ipl	Capture B interrupt priority.
IRQn_Type	capture_a_irq	Capture A interrupt.
IRQn_Type	capture_b_irq	Capture B interrupt.
uint32_t	compare_match_value[2]	Storing compare match value for channels.
uint8_t	compare_match_status	Storing the compare match register status.
gpt_extended_pwm_cfg_t const *	p_pwm_cfg	Advanced PWM features, optional.
gpt_gtior_setting_t	gtior_setting	Custom GTIOR settings used for configuring GTIOCxA and GTIOCxB pins.

Enumeration Type Documentation

◆ gpt_io_pin_t

enum gpt_io_pin_t	
Input/Output pins, used to select which duty cycle to update in R_GPT_DutyCycleSet().	
Enumerator	
GPT_IO_PIN_GTIOCA	GTIOCA.
GPT_IO_PIN_GTIOCB	GTIOCB.
GPT_IO_PIN_GTIOCA_AND_GTIOCB	GTIOCA and GTIOCB.
GPT_IO_PIN_TROUGH	Used in R_GPT_DutyCycleSet when Triangle-wave PWM Mode 3 is selected.
GPT_IO_PIN_CREST	Used in R_GPT_DutyCycleSet when Triangle-wave PWM Mode 3 is selected.
GPT_IO_PIN_ONE_SHOT_LEADING_EDGE	Used in R_GPT_DutyCycleSet to set GTCCRC and GTCCRE registers when One-Shot Pulse mode is selected.
GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE	Used in R_GPT_DutyCycleSet to set GTCCRD and GTCCRF registers when One-Shot Pulse mode is selected.

◆ gpt_buffer_force_push

enum gpt_buffer_force_push	
Forced buffer push operation used in One-Sot Pulse mode with R_GPT_DutyCycleSet().	
Enumerator	
GPT_BUFFER_FORCE_PUSH	Used in R_GPT_DutyCycleSet to force push the data from GTCCRn registers to temporary buffer A or B when One-Shot Pulse mode is selected.

◆ gpt_pin_level_t

enum gpt_pin_level_t	
Level of GPT pin	
Enumerator	
GPT_PIN_LEVEL_LOW	Pin level low.
GPT_PIN_LEVEL_HIGH	Pin level high.

◆ gpt_source_t

enum gpt_source_t	
Sources can be used to start the timer, stop the timer, count up, or count down. These enumerations represent a bitmask. Multiple sources can be ORed together.	
Enumerator	
GPT_SOURCE_NONE	No active event sources.
GPT_SOURCE_GTETRGA_RISING	Action performed on GTETRGA rising edge.
GPT_SOURCE_GTETRGA_FALLING	Action performed on GTETRGA falling edge.
GPT_SOURCE_GTETRGB_RISING	Action performed on GTETRGB rising edge.
GPT_SOURCE_GTETRGB_FALLING	Action performed on GTETRGB falling edge.
GPT_SOURCE_GTETRGC_RISING	Action performed on GTETRGC rising edge.
GPT_SOURCE_GTETRGC_FALLING	Action performed on GTETRGC falling edge.
GPT_SOURCE_GTETRGD_RISING	Action performed on GTETRGB rising edge.
GPT_SOURCE_GTETRGD_FALLING	Action performed on GTETRGB falling edge.
GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input rises while GTIOCB is low.
GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input rises while GTIOCB is high.
GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input falls while GTIOCB is low.
GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input falls while GTIOCB is high.

GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input rises while GTIOCA is low.
GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input rises while GTIOCA is high.
GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input falls while GTIOCA is low.
GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input falls while GTIOCA is high.
GPT_SOURCE_GPT_A	Action performed on ELC GPTA event.
GPT_SOURCE_GPT_B	Action performed on ELC GPTB event.
GPT_SOURCE_GPT_C	Action performed on ELC GPTC event.
GPT_SOURCE_GPT_D	Action performed on ELC GPTD event.
GPT_SOURCE_GPT_E	Action performed on ELC GPTE event.
GPT_SOURCE_GPT_F	Action performed on ELC GPTF event.
GPT_SOURCE_GPT_G	Action performed on ELC GPTG event.
GPT_SOURCE_GPT_H	Action performed on ELC GPTH event.

◆ gpt_capture_filter_t

enum gpt_capture_filter_t	
Input capture signal noise filter (debounce) setting. Only available for input signals GTIOCxA and GTIOCxB. The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal. See "Noise Filter Function" in the hardware manual, GPT section.	
Enumerator	
GPT_CAPTURE_FILTER_NONE	None - no filtering.
GPT_CAPTURE_FILTER_PCLKD_DIV_1	PCLK/1 - fast sampling.
GPT_CAPTURE_FILTER_PCLKD_DIV_4	PCLK/4.
GPT_CAPTURE_FILTER_PCLKD_DIV_16	PCLK/16.
GPT_CAPTURE_FILTER_PCLKD_DIV_64	PCLK/64 - slow sampling.

◆ gpt_adc_trigger_t

enum gpt_adc_trigger_t	
Trigger options to start A/D conversion.	
Enumerator	
GPT_ADC_TRIGGER_NONE	None - no output disable request.
GPT_ADC_TRIGGER_UP_COUNT_START_ADC_A	Request A/D conversion from ADC unit 0 at up counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match.
GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_A	Request A/D conversion from ADC unit 0 at down counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match.
GPT_ADC_TRIGGER_UP_COUNT_START_ADC_B	Request A/D conversion from ADC unit 1 at up counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match.
GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_B	Request A/D conversion from ADC unit 1 at down counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match.

◆ gpt_poeg_link_t

enum gpt_poeg_link_t	
POEG channel to link to this channel.	
Enumerator	
GPT_POEG_LINK_POEG0	Link this GPT channel to POEG channel 0 (GTETRGA)
GPT_POEG_LINK_POEG1	Link this GPT channel to POEG channel 1 (GTETRGB)
GPT_POEG_LINK_POEG2	Link this GPT channel to POEG channel 2 (GTETRGC)
GPT_POEG_LINK_POEG3	Link this GPT channel to POEG channel 3 (GTETRGD)

◆ gpt_output_disable_t

enum gpt_output_disable_t	
Select trigger to send output disable request to POEG.	
Enumerator	
GPT_OUTPUT_DISABLE_NONE	None - no output disable request.
GPT_OUTPUT_DISABLE_DEAD_TIME_ERROR	Request output disable if a dead time error occurs.
GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_HIGH	Request output disable if GTIOCA and GTIOCB are high at the same time.
GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_LOW	Request output disable if GTIOCA and GTIOCB are low at the same time.

◆ gpt_gtioc_disable_t

enum gpt_gtioc_disable_t	
Disable level options for GTIOC pins.	
Enumerator	
GPT_GTIOC_DISABLE_PROHIBITED	Do not allow output disable.
GPT_GTIOC_DISABLE_SET_HI_Z	Set GTIOC to high impedance when output is disabled.
GPT_GTIOC_DISABLE_LEVEL_LOW	Set GTIOC level low when output is disabled.
GPT_GTIOC_DISABLE_LEVEL_HIGH	Set GTIOC level high when output is disabled.

◆ gpt_adc_compare_match_t

enum gpt_adc_compare_match_t	
Trigger options to start A/D conversion.	
Enumerator	
GPT_ADC_COMPARE_MATCH_ADC_A	Set A/D conversion start request value for GPT A/D converter start request A.
GPT_ADC_COMPARE_MATCH_ADC_B	Set A/D conversion start request value for GPT A/D converter start request B.

◆ gpt_interrupt_skip_source_t

enum gpt_interrupt_skip_source_t	
Interrupt skipping modes	
Enumerator	
GPT_INTERRUPT_SKIP_SOURCE_NONE	Do not skip interrupts.
GPT_INTERRUPT_SKIP_SOURCE_OVERFLOW_UNDERFLOW	Count and skip overflow and underflow interrupts.
GPT_INTERRUPT_SKIP_SOURCE_CREST	Count crest interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the trough interrupt fires before the crest interrupt.
GPT_INTERRUPT_SKIP_SOURCE_TROUGH	Count trough interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the crest interrupt fires before the trough interrupt.

◆ gpt_interrupt_skip_count_t

enum gpt_interrupt_skip_count_t	
Number of interrupts to skip between events	
Enumerator	
GPT_INTERRUPT_SKIP_COUNT_0	Do not skip interrupts.
GPT_INTERRUPT_SKIP_COUNT_1	Skip one interrupt.
GPT_INTERRUPT_SKIP_COUNT_2	Skip two interrupts.
GPT_INTERRUPT_SKIP_COUNT_3	Skip three interrupts.
GPT_INTERRUPT_SKIP_COUNT_4	Skip four interrupts.
GPT_INTERRUPT_SKIP_COUNT_5	Skip five interrupts.
GPT_INTERRUPT_SKIP_COUNT_6	Skip six interrupts.
GPT_INTERRUPT_SKIP_COUNT_7	Skip seven interrupts.

◆ gpt_interrupt_skip_adc_t

enum gpt_interrupt_skip_adc_t	
ADC events to skip during interrupt skipping	
Enumerator	
GPT_INTERRUPT_SKIP_ADC_NONE	Do not skip ADC events.
GPT_INTERRUPT_SKIP_ADC_A	Skip ADC A events.
GPT_INTERRUPT_SKIP_ADC_B	Skip ADC B events.
GPT_INTERRUPT_SKIP_ADC_A_AND_B	Skip ADC A and B events.

◆ gpt_pwm_output_delay_setting_t

enum gpt_pwm_output_delay_setting_t	
Delay setting for the PWM Delay Generation Circuit (PDG).	
Enumerator	
GPT_PWM_OUTPUT_DELAY_SETTING_0_32	Delay is not applied.
GPT_PWM_OUTPUT_DELAY_SETTING_1_32	Delay of 1 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_2_32	Delay of 2 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_3_32	Delay of 3 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_4_32	Delay of 4 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_5_32	Delay of 5 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_6_32	Delay of 6 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_7_32	Delay of 7 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_8_32	Delay of 8 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_9_32	Delay of 9 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_10_32	Delay of 10 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_11_32	Delay of 11 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_12_32	Delay of 12 / 32 GTCLK period applied.

GPT_PWM_OUTPUT_DELAY_SETTING_13_32	Delay of 13 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_14_32	Delay of 14 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_15_32	Delay of 15 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_16_32	Delay of 16 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_17_32	Delay of 17 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_18_32	Delay of 18 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_19_32	Delay of 19 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_20_32	Delay of 20 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_21_32	Delay of 21 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_22_32	Delay of 22 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_23_32	Delay of 23 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_24_32	Delay of 24 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_25_32	Delay of 25 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_26_32	Delay of 26 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_27_32	Delay of 27 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_28_32	Delay of 28 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_29_32	Delay of 29 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_30_32	Delay of 30 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_31_32	Delay of 31 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_BYPASS	Bypass the PWM Output Delay Circuit.

◆ gpt_pwm_output_delay_edge_t

enum gpt_pwm_output_delay_edge_t	
Select which PWM Output Delay setting to apply.	
Enumerator	
GPT_PWM_OUTPUT_DELAY_EDGE_RISING	Configure the PWM Output Delay setting for rising edge.
GPT_PWM_OUTPUT_DELAY_EDGE_FALLING	Configure the PWM Output Delay setting for falling edge.

Function Documentation

◆ R_GPT_Open()

fsp_err_t R_GPT_Open (timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg*)

Initializes the timer module and applies configurations. Implements timer_api_t::open.

GPT hardware does not support one-shot functionality natively. When using one-shot mode, the timer will be stopped in an ISR after the requested period has elapsed.

The GPT implementation of the general timer can accept a gpt_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL or the source divider is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IRQ_BSP_DISABLED	timer_cfg_t::mode is TIMER_MODE_ONE_SHOT or timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback.
FSP_ERR_INVALID_MODE	Triangle wave PWM is only supported if GPT_CFG_OUTPUT_SUPPORT_ENABLE is 2. Selected channel does not support external count sources. External and event count sources not are available in this mode.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.

◆ R_GPT_Stop()

fsp_err_t R_GPT_Stop (timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_GPT_Stop(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	Timer successfully stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Start()

fsp_err_t R_GPT_Start (timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_Start(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	Timer successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Reset()

fsp_err_t R_GPT_Reset (timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

Note

This function also updates to the new period if no counter overflow has occurred since the last call to R_GPT_PeriodSet().

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Enable()

fsp_err_t R_GPT_Enable (timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_GPT_Enable(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	External events successfully enabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Disable()

fsp_err_t R_GPT_Disable (timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Note

The timer could be running after R_GPT_Disable(). To ensure it is stopped, call R_GPT_Stop().

Example:

```
/* (Optional) Disable captures. */
   (void) R_GPT_Disable(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	External events successfully disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_PeriodSet()

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

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

Warning

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

Example:

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

```
 * This example uses the 3rd option (R_FSP_SystemClockHzGet).
 */
   uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
/* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
 * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
   uint32_t period_counts =
      (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
/* Set the calculated period. */
   err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
   assert(FSP_SUCCESS == err);
```

Return values

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

◆ R_GPT_DutyCycleSet()

fsp_err_t R_GPT_DutyCycleSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin*)

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Duty cycle is updated in the buffer register. The updated duty cycle is reflected after the next cycle end (counter overflow).

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
 * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
```

```
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    assert(FSP_SUCCESS == err);
```

Parameters

[in]	p_ctrl	Pointer to instance control block.
[in]	duty_cycle_counts	Duty cycle to set in counts.
[in]	pin	Use gpt_io_pin_t to select GPT_IO_PIN_GTIOCA or GPT_IO_PIN_GTIOCB

Return values

FSP_SUCCESS	Duty cycle updated successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL or the pin is not one of gpt_io_pin_t
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_ARGUMENT	Duty cycle is larger than period.
FSP_ERR_INVALID_MODE	GPT_IO_PIN_TROUGH, and GPT_IO_PIN_CREST settings are invalid in the this mode.
FSP_ERR_UNSUPPORTED	GPT_CFG_OUTPUT_SUPPORT_ENABLE is 0.

◆ R_GPT_InfoGet()

fsp_err_t R_GPT_InfoGet (timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info*)

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
      uint64_t period = info.period_counts;
/* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
 * timer_info_t::period_counts. */
if (0U == period)
     {
           period = UINT32_MAX + 1U;
     }
```

Return values

FSP_SUCCESS	Period, count direction, frequency, and ELC event written to caller's structure successfully.
FSP_ERR_ASSERTION	p_ctrl or p_info was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_StatusGet()

fsp_err_t R_GPT_StatusGet (timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status*)

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
```

Return values

FSP_SUCCESS	Current timer state and counter value set successfully.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_CounterSet()

fsp_err_t R_GPT_CounterSet (timer_ctrl_t *const *p_ctrl*, uint32_t *counter*)

Set counter value.

Note

Do not call this API while the counter is counting. The counter value can only be updated while the counter is stopped.

Return values

FSP_SUCCESS	Counter value updated.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_IN_USE	The timer is running. Stop the timer before calling this function.

◆ R_GPT_OutputEnable()

fsp_err_t R_GPT_OutputEnable (timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin*)

Enable output for GTIOCA and/or GTIOCB.

Return values

FSP_SUCCESS	Output is enabled.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_OutputDisable()

fsp_err_t R_GPT_OutputDisable (timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin*)

Disable output for GTIOCA and/or GTIOCB.

Return values

FSP_SUCCESS	Output is disabled.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_AdcTriggerSet()

fsp_err_t R_GPT_AdcTriggerSet (timer_ctrl_t *const *p_ctrl*, gpt_adc_compare_match_t *which_compare_match*, uint32_t *compare_match_value*)

Set A/D converter start request compare match value.

Return values

FSP_SUCCESS	Counter value updated.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_PwmOutputDelaySet()

fsp_err_t R_GPT_PwmOutputDelaySet (timer_ctrl_t *const *p_ctrl*, gpt_pwm_output_delay_edge_t *edge*, gpt_pwm_output_delay_setting_t *delay_setting*, uint32_t const *pin*)

Set the Output Delay setting for the PWM output pin.

Return values

FSP_SUCCESS	The output delay was set.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_CHANNEL	The channel does not support this feature.
FSP_ERR_NOT_INITIALIZED	The PWM Output Delay Circuit has not been initialized.
FSP_ERR_INVALID_STATE	The PWM Output Delay setting cannot be updated in the current state.
FSP_ERR_UNSUPPORTED	This feature is not supported on this MCU.

◆ R_GPT_CallbackSet()

fsp_err_t R_GPT_CallbackSet (timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_GPT_Close()

fsp_err_t R_GPT_Close (timer_ctrl_t *const *p_ctrl*)

Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_PwmOutputDelayInitialize()

fsp_err_t R_GPT_PwmOutputDelayInitialize (void)

Initialize the PWM Delay Generation Circuit (PDG). This function must be called before calling R_GPT_PwmOutputDelaySet.

Note

This function will delay for 20 microseconds.

Return values

FSP_SUCCESS	Initialization sequence completed successfully.
FSP_ERR_INVALID_STATE	The source clock frequnecy is out of the required range for the PDG.
FSP_ERR_UNSUPPORTED	This feature is not supported.

◆ R_GPT_CompareMatchSet()

fsp_err_t R_GPT_CompareMatchSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel*)

Set value for compare match feature. Implements timer_api_t::compareMatchSet.

Note

This API should be used when timer is stop counting. And shall not be used along with PWM operation.

Example:

```
 /* Set the compare match value (GPT_COMPARE_MATCH_EXAMPLE_VALUE). This value must be
less than or equal to period value. */
    err = R_GPT_CompareMatchSet(&g_timer0_ctrl, GPT_COMPARE_MATCH_EXAMPLE_VALUE,
TIMER_COMPARE_MATCH_A);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Set the compare match value successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_NOT_ENABLED	Requested compare channel is disabled.

5.2.19.4 Three-Phase PWM (r_gpt_three_phase)

Modules » Timers

Functions

fsp_err_t	R_GPT_THREE_PHASE_Open (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)
fsp_err_t	R_GPT_THREE_PHASE_Stop (three_phase_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_THREE_PHASE_Start (three_phase_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_THREE_PHASE_Reset (three_phase_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_THREE_PHASE_DutyCycleSet (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)
fsp_err_t	R_GPT_THREE_PHASE_CallbackSet (three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)

fsp_err_t	R_GPT_THREE_PHASE_Close (three_phase_ctrl_t *const p_ctrl)

Detailed Description

Driver for 3-phase motor control using the GPT peripheral on RX MCUs. This module implements the Three-Phase Interface.

Overview

The General PWM Timer (GPT) Three-Phase driver provides basic functionality for synchronously starting and stopping three PWM channels for use in 3-phase motor control applications. A function is additionally provided to allow setting duty cycle values for all three channels, optionally with double-buffering.

Features

The GPT Three-Phase driver provides the following functions:

- Synchronize configuration of three GPT channels
- Synchronously start, stop and reset all three GPT channels
- Set duty cycle on all three channels with one function

Configuration

Build Time Configurations for r_gpt_three_phase

The following build time configurations are defined in fsp_cfg/r_gpt_three_phase_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Three-Phase PWM (r_gpt_three_phase)

This module can be added to the Stacks tab via New Stack > Timers > Three-Phase PWM (r_gpt_three_phase).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_three_phase0	Module name.
General > Mode	• Triangle-Wave Symmetric PWM • Triangle-Wave Asymmetric PWM	Triangle-Wave Symmetric PWM	Mode selection. Triangle-Wave Symmetric PWM: Generates symmetric PWM waveforms with duty cycle determined

	• Triangle-Wave Asymmetric PWM (Mode 3)		by compare match set during a crest interrupt and updated at the next trough. Triangle-Wave Asymmetric PWM: Generates asymmetric PWM waveforms with duty cycle determined by compare match set during a crest/trough interrupt and updated at the next trough/crest.
General > Period	Value must be a non-negative integer less than or equal to 0x40000000000	15	Specify the timer period in units selected below. Setting the period to 0x100000000 raw counts results in the maximum period. Set the period to 0x100000000 raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000, which will use a divider of 1024 with the maximum period. If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the generated timer_cfg_t structures for each timer.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Kilohertz	Unit of the period specified above
General > GPT U-	Value must be an	0	Specify the GPT

Channel	integer greater than or equal to 0		channel for U signal output.
General > GPT V-Channel	Value must be an integer greater than or equal to 0	1	Specify the GPT channel for V signal output.
General > GPT W-Channel	Value must be an integer greater than or equal to 0	2	Specify the GPT channel for W signal output.
General > Callback Channel	• U-Channel • V-Channel • W-Channel	U-Channel	Specify the GPT channel to set a callback for when using R_GPT_THREE_PHASE_CallbackSet.
General > Buffer Mode	• Single Buffer • Double Buffer	Single Buffer	When Double Buffer is selected the 'duty_buffer' array in three_phase_duty_cycle_t is used as a buffer for the 'duty' array. This allows setting the duty cycle for the next two crest/trough events in asymmetric mode with only one call to R_GPT_THREE_PHASE_DutyCycleSet.
General > GTIOCA Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
General > GTIOCB Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
Extra Features > Dead Time > Dead Time Count Up (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during up counting. This value also applies during down counting for the GPT32 and GPT16 variants.
Extra Features > Dead Time > Dead Time Count Down (Raw Counts) (GPTE/GPTEH only)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during down counting. This value only applies to the GPT32E and GPT32EH variants.

Clock Configuration

Please refer to the Timer, General PWM (r_gpt) section for more information.

Pin Configuration

Please refer to the Timer, General PWM (r_gpt) section for more information.

Usage Notes

Warning

Be sure the GTIOCA/B stop level and dead time values are set appropriately for your application before attempting to drive a motor. Failure to do so may result in damage to the motor drive circuitry and/or the motor itself if the timer is stopped by software.

Initial Setup

The following should be configured once the GPT Three-Phase module has been added to a project:

1. Set "Pin Output Support" in one of the GPT submodules to "Enabled with Extra Features"
2. Configure common settings in the GPT Three-Phase module properties
3. Set the crest and trough interrupt priority and callback function in **one** of the three GPT submodules (if desired)
4. Set the "Extra Features -> Output Disable" settings in each GPT submodule as needed for your application

Note

Because all three modules are operated synchronously with the same period interrupts only need to be enabled in one of the three GPT modules.

Buffer Modes

There are two buffering modes available for duty cycle values - single- and double-buffered. In single buffer mode only the values specified in the duty array element of three_phase_duty_cycle_t are used by R_GPT_THREE_PHASE_DutyCycleSet. At the next trough or crest event the output duty cycle will be internally updated to the set values.

In double buffer mode the duty_buffer array values are used as buffer values for the duty elements. Once passed to R_GPT_THREE_PHASE_DutyCycleSet, the next trough or crest event will update the output duty cycle to the values specified in duty as before. However, at the following crest or trough event the output duty cycle will be updated to the values in duty_buffer. This allows the duty cycle for both sides of an asymmetric PWM waveform to be set at only one trough or crest event per period instead of at every event.

Examples

GPT Three-Phase Basic Example

This is a basic example of minimal use of the GPT Three-Phase module in an application. The duty cycle is updated at every timer trough with the previously loaded buffer value, then the duty cycle buffer is reloaded in the trough interrupt callback.

```
void gpt_callback (timer_callback_args_t * p_args)
{
 fsp_err_t                err;
```

```
 three_phase_duty_cycle_t duty_cycle;
 if (TIMER_EVENT_TROUGH == p_args->event)
    {
 /* Update duty cycle values (example) */
       duty_cycle.duty[THREE_PHASE_CHANNEL_U] =
get_duty_counts(THREE_PHASE_CHANNEL_U);
       duty_cycle.duty[THREE_PHASE_CHANNEL_V] =
get_duty_counts(THREE_PHASE_CHANNEL_V);
       duty_cycle.duty[THREE_PHASE_CHANNEL_W] =
get_duty_counts(THREE_PHASE_CHANNEL_W);
 /* Update duty cycle values */
       err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Handle crest event. */
    }
}
void gpt_three_phase_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
}
```

Data Structures

struct	gpt_three_phase_instance_ctrl_t

Data Structure Documentation

◆ gpt_three_phase_instance_ctrl_t

struct gpt_three_phase_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when three_phase_api_t::open is called.

Function Documentation

◆ R_GPT_THREE_PHASE_Open()

fsp_err_t R_GPT_THREE_PHASE_Open (three_phase_ctrl_t *const *p_ctrl*, three_phase_cfg_t const *const *p_cfg*)

Initializes the 3-phase timer module (and associated timers) and applies configurations. Implements three_phase_api_t::open.

Example:

```
/* Initializes the module. */

   err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL.
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_GPT_THREE_PHASE_Stop()

fsp_err_t R_GPT_THREE_PHASE_Stop (three_phase_ctrl_t *const *p_ctrl*)

Stops all timers synchronously. Implements three_phase_api_t::stop.

Return values

FSP_SUCCESS	Timers successfully stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_THREE_PHASE_Start()

fsp_err_t R_GPT_THREE_PHASE_Start (three_phase_ctrl_t *const *p_ctrl*)

Starts all timers synchronously. Implements three_phase_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
```

Return values

FSP_SUCCESS	Timers successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_THREE_PHASE_Reset()

fsp_err_t R_GPT_THREE_PHASE_Reset (three_phase_ctrl_t *const *p_ctrl*)

Resets the counter values to 0. Implements three_phase_api_t::reset.

Return values

FSP_SUCCESS	Counters were reset successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_THREE_PHASE_DutyCycleSet()

fsp_err_t R_GPT_THREE_PHASE_DutyCycleSet (three_phase_ctrl_t *const *p_ctrl*, three_phase_duty_cycle_t *const *p_duty_cycle*)

Sets duty cycle for all three timers. Implements three_phase_api_t::dutyCycleSet.

In symmetric PWM mode duty cycle values are reflected after the next trough. In asymmetric PWM mode values are reflected at the next trough OR crest, whichever comes first.

When double-buffering is enabled the values in three_phase_duty_cycle_t::duty_buffer are set to the double-buffer registers. When values are reflected the first time the single buffer values (three_phase_duty_cycle_t::duty) are used. On the second reflection the duty_buffer values are used. In asymmetric PWM mode this enables both count-up and count-down PWM values to be set at trough (or crest) exclusively.

Note

It is recommended to call this function in a high-priority callback to ensure that it is not interrupted and that no GPT events occur during setting that would result in a duty cycle buffer load operation.

Example:

```
/* Update duty cycle values */
      err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
      assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Duty cycle updated successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_ARGUMENT	One or more duty cycle count values was outside the range 0..(period - 1).

◆ R_GPT_THREE_PHASE_CallbackSet()

fsp_err_t R_GPT_THREE_PHASE_CallbackSet (three_phase_ctrl_t *const *p_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback for the GPT U-channel with the option to provide memory for the callback argument structure. Implements three_phase_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_GPT_THREE_PHASE_Close()

fsp_err_t R_GPT_THREE_PHASE_Close (three_phase_ctrl_t *const *p_ctrl*)
Stops counters, disables output pins, and clears internal driver data. Implements three_phase_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

5.2.19.5 Port Output Enable for GPT (r_poeg)

Modules » Timers

Functions

fsp_err_t	R_POEG_Open (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)
fsp_err_t	R_POEG_StatusGet (poeg_ctrl_t *const p_ctrl, poeg_status_t *const p_status)
fsp_err_t	R_POEG_CallbackSet (poeg_ctrl_t *const p_ctrl, void(*p_callback)(poeg_callback_args_t *), void *const p_context, poeg_callback_args_t *const p_callback_memory)
fsp_err_t	R_POEG_OutputDisable (poeg_ctrl_t *const p_ctrl)
fsp_err_t	R_POEG_Reset (poeg_ctrl_t *const p_ctrl)
fsp_err_t	R_POEG_Close (poeg_ctrl_t *const p_ctrl)

Detailed Description

Driver for the POEG peripheral on RX MCUs. This module implements the POEG Interface.

Overview

The POEG module can be used to configure events to disable GPT GTIOC output pins.

Features

The POEG module has the following features:

- Supports disabling GPT output pins based on GTETRG input pin level.
- Supports disabling GPT output pins when GTIOC pins are the same level (configurable in the Timer, General PWM (r_gpt) driver).
- Supports disabling GPT output pins when main oscillator stop is detected.
- Supports disabling GPT output pins by software API.
- Supports notifying the application when GPT output pins are disabled by POEG.
- Supports resetting POEG status.

Configuration

Build Time Configurations for r_poeg

The following build time configurations are defined in fsp_cfg/r_poeg_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Port Output Enable for GPT (r_poeg)

This module can be added to the Stacks tab via New Stack > Timers > Port Output Enable for GPT (r_poeg).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_poeg0	Module name.
General > Channel	Must be a valid POEG channel	0	Specify the hardware channel.
General > Trigger	MCU Specific Options		Select the trigger sources that will enable POEG. Software disable is always supported. This configuration can only be set once after reset. It cannot be modified after the initial setting.
Input > GTETRG Polarity	• Active High • Active Low	Active High	Select the polarity of the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger.
Input > GTETRG Noise Filter	• Disabled • PCLKA/1 • PCLKA/8 • PCLKA/32 • PCLKA/128	Disabled	Configure the noise filter for the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger.

Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) when GPT output pins are disabled by POEG.
Interrupts > Interrupt Priority	MCU Specific Options		Select the POEG interrupt priority.

Clock Configuration

The POEG clock is based on the PCLKA frequency.

Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, or GTETRGD as an input signal to disable GPT output pins.

Usage Notes

POEG GTETRG Pin and Channel

The POEG channel number corresponds to the GTETRG input pin that can be used with the channel. GTETRGA must be used with POEG channel 0, GTETRGB must be used with POEG channel 1, etc.

Limitations

The user should be aware of the following limitations when using POEG:

- The POEG trigger source can only be set once per channel. Modifying the POEG trigger source after it is set is not allowed by the hardware.
- The POEG cannot be disabled using this API. The interrupt is disabled in R_POEG_Close(), but the POEG will still disable the GPT output pins if a trigger is detected even if the module is closed.

Examples

POEG Basic Example

This is a basic example of minimal use of the POEG in an application.

```
void poeg_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the POEG. */
    err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

POEG Callback Example

This is an example of a using the POEG callback to restore GPT output operation.

```
/* Example callback called when POEG disables GPT output pins. */
void poeg_callback (poeg_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 /* (Optional) Determine the cause of the POEG event. */
 poeg_status_t status;
    (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
 /* Correct the cause of the POEG event before resetting POEG. */
 /* Reset the POEG before exiting the callback. */
    (void) R_POEG_Reset(&g_poeg0_ctrl);
 /* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
  * again. */
 do
    {
      (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
    } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
 /* Alternatively, if the POEG cannot be reset, disable the POEG interrupt to prevent
it from firing continuously.
  * Update the 0 in the macro below to match the POEG channel number. */
 R_BSP_IrqDisable(VECTOR_NUMBER_POEG0_EVENT);
}
```

Data Structures

struct	poeg_instance_ctrl_t

Data Structure Documentation

◆ poeg_instance_ctrl_t

struct poeg_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when poeg_api_t::open is called.

Function Documentation

◆ R_POEG_Open()

fsp_err_t R_POEG_Open (poeg_ctrl_t *const *p_ctrl*, poeg_cfg_t const *const *p_cfg*)

Initializes the POEG module and applies configurations. Implements poeg_api_t::open.

Note

The poeg_cfg_t::trigger setting can only be configured once after reset. Reopening with a different trigger configuration is not possible.

Example:

```
/* Initializes the POEG. */

   err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL or the source divider is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IRQ_BSP_DISABLED	poeg_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.

◆ R_POEG_StatusGet()

fsp_err_t R_POEG_StatusGet (poeg_ctrl_t *const *p_ctrl*, poeg_status_t *const *p_status*)

Get current POEG status and store it in provided pointer p_status. Implements poeg_api_t::statusGet.

Example:

```
/* (Optional) Determine the cause of the POEG event. */
poeg_status_t status;
   (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
```

Return values

FSP_SUCCESS	Current POEG state stored successfully.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_POEG_CallbackSet()

fsp_err_t R_POEG_CallbackSet (poeg_ctrl_t *const *p_ctrl*, void(*)(poeg_callback_args_t *) *p_callback*, void *const *p_context*, poeg_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements poeg_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_POEG_OutputDisable()

fsp_err_t R_POEG_OutputDisable (poeg_ctrl_t *const *p_ctrl*)

Disables GPT output pins. Implements poeg_api_t::outputDisable.

Return values

FSP_SUCCESS	GPT output pins successfully disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_POEG_Reset()

fsp_err_t R_POEG_Reset (poeg_ctrl_t *const *p_ctrl*)

Resets status flags. Implements poeg_api_t::reset.

Note

Status flags are only reset if the original POEG trigger is resolved. Check the status using R_POEG_StatusGet after calling this function to verify the status is cleared.

Example:

```
/* Correct the cause of the POEG event before resetting POEG. */
/* Reset the POEG before exiting the callback. */
   (void) R_POEG_Reset(&g_poeg0_ctrl);
/* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
 * again. */
do
   {
     (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
   } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
```

Return values

FSP_SUCCESS	Function attempted to clear status flags.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_POEG_Close()

fsp_err_t R_POEG_Close (poeg_ctrl_t *const *p_ctrl*)

Disables POEG interrupt. Implements poeg_api_t::close.

Note

This function does not disable the POEG.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

5.2.19.6 Realtime Clock (r_rtc)

Modules » Timers

Functions

fsp_err_t	R_RTC_Open (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
fsp_err_t	R_RTC_Close (rtc_ctrl_t *const p_ctrl)
fsp_err_t	R_RTC_ClockSourceSet (rtc_ctrl_t *const p_ctrl)
fsp_err_t	R_RTC_CalendarTimeSet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t	R_RTC_CalendarTimeGet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t	R_RTC_CalendarAlarmSet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t	R_RTC_CalendarAlarmGet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t	R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)
fsp_err_t	R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)
fsp_err_t	R_RTC_InfoGet (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)
fsp_err_t	R_RTC_CallbackSet (rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory)
fsp_err_t	R_RTC_TimeCaptureSet (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)
fsp_err_t	R_RTC_TimeCaptureGet (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)

Detailed Description

Driver for the RTC peripheral on RX MCUs. This module implements the RTC Interface.

Overview

The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A

callback can be used to respond to the alarm and periodic interrupt.

Features

- RTC time and date get and set.
- RTC time and date alarm get and set.
- RTC alarm and periodic event notification.
- RTC time capture.

The RTC HAL module supports three different interrupt types:

- An alarm interrupt generated on a match of any combination of year, month, day, day of the week, hour, minute or second
- A periodic interrupt generated every 2, 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, or 1/256 second(s)
- A carry interrupt is used internally when reading time from the RTC calendar to get accurate time readings.
  Note
  See section "Reading 64-Hz Counter and Time" of the relevant hardware manual for more details.
  A user-defined callback function can be registered (in the rtc_api_t::open API call) and will be called from the interrupt service routine (ISR) for alarm and periodic interrupt.When called, it is passed a pointer to a structure (rtc_callback_args_t) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

Date and Time validation

"Parameter Checking" needs to be enabled if date and time validation is required for calendarTimeSet and calendarAlarmSet APIs. If "Parameter Checking" is enabled, the 'day of the week' field is automatically calculated and updated by the driver for the provided date. When using the calendarAlarmSet API, only the fields which have their corresponding match flag set are written to the registers. Other register fields are reset to default value.

Sub-Clock error adjustment (Time Error Adjustment Function)

The time error adjustment function is used to correct errors, running fast or slow, in the time caused by variation in the precision of oscillation by the sub-clock oscillator. Because 32,768 cycles of the sub-clock oscillator constitute 1 second of operation when the sub-clock oscillator is selected, the clock runs fast if the sub-clock frequency is high and slow if the sub-clock frequency is low. The time error adjustment functions include:

- Automatic adjustment
- Adjustment by software

The error adjustment is reset every time RTC is reconfigured or time is set.

Note

RTC driver configurations do not do error adjustment internally while initializing the driver. Application must make calls to the error adjustment api's for desired adjustment. See section "Time Error Adjustment Function" of the relevant hardware manual for more details on this feature

Configuration

Build Time Configurations for r_rtc

The following build time configurations are defined in fsp_cfg/r_rtc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Set Source Clock in Open	• Enabled • Disabled	Enabled	If enabled RTC source clock is initialized in R_RTC_Open. If disabled, R_RTC_ClockSourceSet must be called to set the RTC source clock. Disable if user wants to control the setting of RTC source clock after warm start.

Configurations for Timers > Realtime Clock (r_rtc)

This module can be added to the Stacks tab via New Stack > Timers > Realtime Clock (r_rtc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rtc0	Module name.
Clock Source	MCU Specific Options		Select the RTC clock source.
Frequency Comparison Value (LOCO)	Value must be a positive integer between 7 and 511	255	Frequency comparison value when using LOCO
Automatic Adjustment Mode	• Enabled • Disabled	Enabled	Enable/ Disable the Error Adjustment mode
Automatic Adjustment Period	• 10 Seconds • 1 Minute • NONE	10 Seconds	Select the Error Adjustment Period for Automatic Adjustment
Adjustment Type (Plus-Minus)	• NONE • Addition • Subtraction	NONE	Select the Error Adjustment type
Error Adjustment Value	Manual Entry	0	Specify the Adjustment Value (the number of sub-clock cycles) from the prescaler
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the

		interrupt service routine (ISR).
Alarm Interrupt Priority	MCU Specific Options	Select the alarm interrupt priority.
Period Interrupt Priority	MCU Specific Options	Select the period interrupt priority.
Carry Interrupt Priority	MCU Specific Options	Select the carry interrupt priority.

Note

See Frequency Register (RFRH/RFRL) of the relevant hardware manual for more details

Interrupt Configuration

To activate interrupts for the RTC module, the desired interrupts must be enabled, the underlying implementation will be expected to handle any interrupts it can support and notify higher layers via callback.

Clock Configuration

The RTC HAL module can use the following clock sources:

- LOCO (Low Speed On-Chip Oscillator) with less accuracy
- Sub-clock oscillator with increased accuracy

Users have to select the right source for their application. LOCO is the default during configuration when it is available.

Pin Configuration

This module does not use I/O pins.

Usage Notes

System Initialization

- RTC driver does not start the sub-clock. The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RX startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running RTC off the subclock, the application must ensure the subclock is stable before starting operation.

- Carry interrupt priority must be set to avoid incorrect time returned from calendarTimeGet API during roll-over.
- R_RTC_CalendarTimeSet must be called successfully at least once to start the RTC count operation.
- In case of warm start or a hard reset, user might not want to reinitialize the clock source after reset.In this case, disable the Set Source Clock in Open property for RTC and call R_RTC_ClockSourceSet() only when RTC clock source needs to be set. Application should check for the reset type and should accordingly call the R_RTC_ClockSourceSet().

Limitations

Developers should be aware of the following limitations when using the RTC:

- R_RTC operates in 24-hour mode.
- Binary-count mode is not supported.
- The R_RTC_CalendarTimeGet() cannot be used from an interrupt that has higher priority than the carry interrupt.Also, it must not be called with interrupts disabled globally, as this API internally uses carry interrupt for its processing. API may return incorrect time if this is done.
- Time capture input pins should be configured prior to opening RTC.
- When multiple events are detected, the capture time for the first event is retained. Time capture value must be got and reset status bit for the next capture when having event input.

Time Capture

The RTC is capable of storing the month, day, hour, minute and second when detecting an edge of a signal on a time capture event input pin. On RX boards up to three capture channels can be configured.

- Use R_RTC_TimeCaptureSet to configure the detection source and noise filter for each capture channel.
- Use R_RTC_TimeCaptureGet to check the latest capture. If no capture has been triggered the function will return FSP_ERR_INVALID_STATE.

Examples

RTC Basic Example

This is a basic example of minimal use of the RTC in an application.

```
/* rtc_time_t is an alias for the C Standard time.h struct 'tm' */
rtc_time_t set_time =
{
    .tm_sec = 10,
    .tm_min = 11,
    .tm_hour = 12,
    .tm_mday = 6,
    .tm_wday = 3,
    .tm_mon = 11,
    .tm_year = YEARS_SINCE_1900,
};
rtc_time_t get_time;
void rtc_example (void)
{
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module */
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 /* Set the calendar time */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Get the calendar time */
 R_RTC_CalendarTimeGet(&g_rtc0_ctrl, &get_time);
}
```

RTC Clock Source Set Example

This is an example of how to handle call to set the RTC clock.

```
void rtc_clock_source_set_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* This condition can differ based on use case. */
 if (R_SYSTEM->RSTSR0 == 1)
    {
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
    }
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
```

```
 R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

RTC Periodic interrupt example

This is an example of periodic interrupt in RTC.

```
void rtc_periodic_irq_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
 R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

RTC Alarm interrupt example

This is an example of alarm interrupt in RTC.

```
void rtc_alarm_irq_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /*Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 R_RTC_CalendarAlarmSet(&g_rtc0_ctrl, &set_time1);
 /* Wait for the Alarm interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

RTC Error Adjustment example

This is an example of modifying error adjustment in RTC.

```
void rtc_erroradj_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /*Open the RTC module*/
 R_RTC_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 /* Modify Error Adjustment after RTC is running */
    err = R_RTC_ErrorAdjustmentSet(&g_rtc0_ctrl, &err_cfg2);
    assert(FSP_SUCCESS == err);
```

```
}
```

RTC Time Capture example

This is an example of calling Time Capture API on RTC.

```
void rtc_time_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure a GPIO pin to trigger the Time Capture pin */
 R_BSP_PinCfg(RTC_TIME_CAPTURE_TRIGGER_CHANNEL_0, (uint32_t) BSP_IO_DIRECTION_OUTPUT
);
 /* Preset trigger pin */
 R_BSP_PinWrite(RTC_TIME_CAPTURE_TRIGGER_CHANNEL_0, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set time capture configuration for the provided channel when RTC is running */
    err = R_RTC_TimeCaptureSet(&g_rtc0_ctrl, &g_rtc_time_capture);
    assert(FSP_SUCCESS == err);
 /* Update trigger pin */
 R_BSP_PinWrite(RTC_TIME_CAPTURE_TRIGGER_CHANNEL_0, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Get the time capture for the provided channel */
    err = R_RTC_TimeCaptureGet(&g_rtc0_ctrl, &g_rtc_time_capture);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rtc_extended_cfg_t
struct	rtc_instance_ctrl_t

Data Structure Documentation

◆ rtc_extended_cfg_t

struct rtc_extended_cfg_t		
RTC extend configuration		
Data Fields		
uint8_t	alarm1_ipl	Alarm 1 interrupt priority.
IRQn_Type	alarm1_irq	Alarm 1 interrupt vector.

◆ rtc_instance_ctrl_t

struct rtc_instance_ctrl_t	
Channel control block. DO NOT INITIALIZE. Initialization occurs when rtc_api_t::open is called	
Data Fields	
uint32_t	open
	Whether or not driver is open.
const rtc_cfg_t *	p_cfg
	Pointer to initial configurations.
volatile bool	carry_isr_triggered
	Was the carry isr triggered.

Function Documentation

◆ R_RTC_Open()

fsp_err_t R_RTC_Open (rtc_ctrl_t *const *p_ctrl*, rtc_cfg_t const *const *p_cfg*)

Opens and configures the RTC driver module. Implements rtc_api_t::open. Configuration includes clock source, and interrupt callback function.

Example:

```
/* Open the RTC module */

   err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	Invalid p_ctrl or p_cfg pointer.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.

◆ R_RTC_Close()

fsp_err_t R_RTC_Close (rtc_ctrl_t *const *p_ctrl*)

Close the RTC driver. Implements rtc_api_t::close

Return values

FSP_SUCCESS	De-Initialization was successful and RTC driver closed.
FSP_ERR_ASSERTION	Invalid p_ctrl.
FSP_ERR_NOT_OPEN	Driver not open already for close.

◆ R_RTC_ClockSourceSet()

fsp_err_t R_RTC_ClockSourceSet (rtc_ctrl_t *const *p_ctrl*)

Sets the RTC clock source. Implements rtc_api_t::clockSourceSet.

Example:

```
/* This condition can differ based on use case. */
if (R_SYSTEM->RSTSR0 == 1)
   {
/* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
R_RTC_ClockSourceSet(&g_rtc0_ctrl);
   }
```

Return values

FSP_SUCCESS	Clock source set was successful.
FSP_ERR_ASSERTION	Invalid p_ctrl or p_cfg pointer.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_ARGUMENT	Invalid clock source.

◆ R_RTC_CalendarTimeSet()

fsp_err_t R_RTC_CalendarTimeSet (rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time*)

Set the calendar time.

Implements rtc_api_t::calendarTimeSet.

Note

This API starts the RTC by setting RCR2.START bit to 1. It must be called at least once to start the RTC count operation.

Return values

FSP_SUCCESS	Calendar time set operation was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.

◆ R_RTC_CalendarTimeGet()

fsp_err_t R_RTC_CalendarTimeGet (rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time*)

Get the calendar time.

Warning

Do not call this function from a critical section or from an interrupt with higher priority than the carry interrupt, or the time returned may be inaccurate.

Implements rtc_api_t::calendarTimeGet

Return values

FSP_SUCCESS	Calendar time get operation was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid

◆ R_RTC_CalendarAlarmSet()

fsp_err_t R_RTC_CalendarAlarmSet (rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm*)

Set the calendar alarm time.

Implements rtc_api_t::calendarAlarmSet.

Precondition

The calendar counter must be running before the alarm can be set.

Return values

FSP_SUCCESS	Calendar alarm time set operation was successful.
FSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid

◆ R_RTC_CalendarAlarmGet()

fsp_err_t R_RTC_CalendarAlarmGet (rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm*)

Get the calendar alarm time.

Implements rtc_api_t::calendarAlarmGet

Return values

FSP_SUCCESS	Calendar alarm time get operation was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_PeriodicIrqRateSet()

fsp_err_t R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const *p_ctrl*, rtc_periodic_irq_select_t const *rate*)

Set the periodic interrupt rate and enable periodic interrupt.

Implements rtc_api_t::periodicIrqRateSet

Note

To start the RTC R_RTC_CalendarTimeSet must be called at least once.

Example:

```
/* Set the periodic interrupt rate to 1 second */
R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
```

Return values

FSP_SUCCESS	The periodic interrupt rate was successfully set.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid

◆ R_RTC_ErrorAdjustmentSet()

fsp_err_t R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const *p_ctrl*, rtc_error_adjustment_cfg_t const *const *err_adj_cfg*)

This function sets time error adjustment

Implements rtc_api_t::errorAdjustmentSet

Return values

FSP_SUCCESS	Time error adjustment successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open for operation.
FSP_ERR_UNSUPPORTED	The clock source is not sub-clock.
FSP_ERR_INVALID_ARGUMENT	Invalid error adjustment value.

◆ R_RTC_InfoGet()

fsp_err_t R_RTC_InfoGet (rtc_ctrl_t *const *p_ctrl*, rtc_info_t *const *p_rtc_info*)

Set RTC clock source and running status information ad store it in provided pointer p_rtc_info

Implements rtc_api_t::infoGet

Return values

FSP_SUCCESS	Get information Successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_CallbackSet()

fsp_err_t R_RTC_CallbackSet (rtc_ctrl_t *const *p_ctrl*, void(*)(rtc_callback_args_t *) *p_callback*, void *const *p_context*, rtc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements rtc_api_t::callbackSet

Return values

FSP_SUCCESS	Baud rate was successfully changed.
FSP_ERR_ASSERTION	Pointer to RTC control block is NULL or the RTC is not configured to use the internal clock.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_RTC_TimeCaptureSet()

fsp_err_t R_RTC_TimeCaptureSet (rtc_ctrl_t *const *p_ctrl*, rtc_time_capture_t *const *p_time_capture*)

Set time capture configuration for the provided channel.

Implements rtc_api_t::timeCaptureSet

Note

Updating capture settings requires significant software delay. Timing considerations should be carefully considered when calling this function.

Return values

FSP_SUCCESS	Setting for Time capture was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_INVALID_CHANNEL	Invalid input channel set.
FSP_ERR_UNSUPPORTED	Hardware not support this feature.
FSP_ERR_INVALID_ARGUMENT	Invalid input noise filter mode.

◆ R_RTC_TimeCaptureGet()

fsp_err_t R_RTC_TimeCaptureGet (rtc_ctrl_t *const *p_ctrl*, rtc_time_capture_t *const *p_time_capture*)

Get time capture value of the provided channel.

Implements rtc_api_t::timeCaptureGet

Return values

FSP_SUCCESS	Get time capture successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_INVALID_CHANNEL	Invalid input channel get.
FSP_ERR_INVALID_STATE	Invalid operation state.
FSP_ERR_UNSUPPORTED	Hardware not support this feature.
FSP_ERR_INVALID_ARGUMENT	Invalid input noise filter mode.

5.2.19.7 Timer, 8-Bit Timer (r_tmr)

Modules » Timers

Functions

fsp_err_t	R_TMR_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t	R_TMR_Start (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_TMR_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_TMR_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_TMR_Disable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_TMR_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_TMR_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_TMR_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_TMR_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const

	compare_match_value, timer_compare_match_t const match_channel)
fsp_err_t	R_TMR_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t	R_TMR_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t	R_TMR_AdcTriggerSet (timer_ctrl_t *const p_ctrl, tmr_adc_trigger_t adc_request_enable)
fsp_err_t	R_TMR_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t	R_TMR_Close (timer_ctrl_t *const p_ctrl)

Detailed Description

Driver for the TMR peripheral on RX MCUs. This module implements the Timer Interface.

Overview

Features

The TMR module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Configurable clock source, including PCLKB.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Supports linking to the specified module, TMR count start, event count, or count restart by ELC events.

Selecting a Timer

RX MCUs have four timer peripherals: the General PWM Timer (GPT), the Compare Match Timer (CMT), the Compare Match Timer W (CMTW) and the 8-bit Timer (TMR). When selecting between them, consider these factors:

	GPT	CMT	CMTW	TMR
Low Power Modes	The GPT can operate in sleep mode.	The CMT can operate in sleep mode.	The CMTW can operate in sleep mode.	The TMR can operate in sleep mode.

Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 4 GPT channels.	The number of CMT channels is device specific. All MCUs have 4 CMT channels.	The number of CMTW channels is device specific. All MCUs have 2 CMTW channels.	All MCUs have 4 TMR channels.
Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The CMT timers are 16-bit timers.	The CMTW timer have at least one 32-bit CMTW timer	The TMR timers are 8-bit timer or 16-bit timer.
Clock Source	The GPT runs off PCLKA with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The CMT runs off PCLKB.	The CMTW runs off PCLKB.	The TMR runs off PCLKB.

Configuration

Build Time Configurations for r_tmr

The following build time configurations are defined in fsp_cfg/r_tmr_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Pin Output Support	• Disabled • Enabled	Disabled	If selected code for outputting a waveform to a pin is included in the build.

Configurations for Timers > Timer, 8-bit timer (r_tmr)

This module can be added to the Stacks tab via New Stack > Timers > Timer, 8-bit timer (r_tmr).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_timer0	Module name.
General > Counter Size	MCU Specific Options		Counter size register bit width (8-bit or 16-bit))
General > Channel	Channel number must be a non-negative integer	0	Physical hardware channel.

General > Mode	• Periodic • One-Shot • PWM	Periodic	Mode selection. Note: One-shot mode is implemented in software. ISR's must be enabled for one shot even if callback is unused.
General > Period	Value must be non-negative	0x100	Specify the timer period based on the selected unit.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Raw Counts	Unit of the period specified above
Output > Duty Cycle Percent (only applicable in PWM mode)	Value must be between 0 and 100	50	Specify the timer duty cycle percent. Only used in PWM mode.
Extra Features > ADC Trigger	MCU Specific Options		Select the ADC trigger enable.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses.
Interrupts > Overflow Interrupt Priority	MCU Specific Options		Select the interrupt priority for overflow.
Interrupts > Compare Match Interrupt Priority	MCU Specific Options		Select the interrupt priority for compare match.

Clock Configuration

The TMR clock is based on the PCLKB frequency. You can set the clock frequency using the **Clocks** tab of the RX Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module can use the TMOn pin as output pins for periodic, one-shot, or PWM signals.

Timer Period

The RX Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units, and clock speed.

When the selected unit is "Raw counts", the maximum allowed period setting varies depending on the selected clock source:

Clock source	8-bit Timer Maximum period (counts)	16-bit Timer Maximum period (counts)
PCLKB	0xFF	0xFFFF

Usage Notes

One-Shot Mode

The TMR timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Transfer (r_dtc).

One-Shot Mode Output

The output waveform in one-shot mode is one TMR clock cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

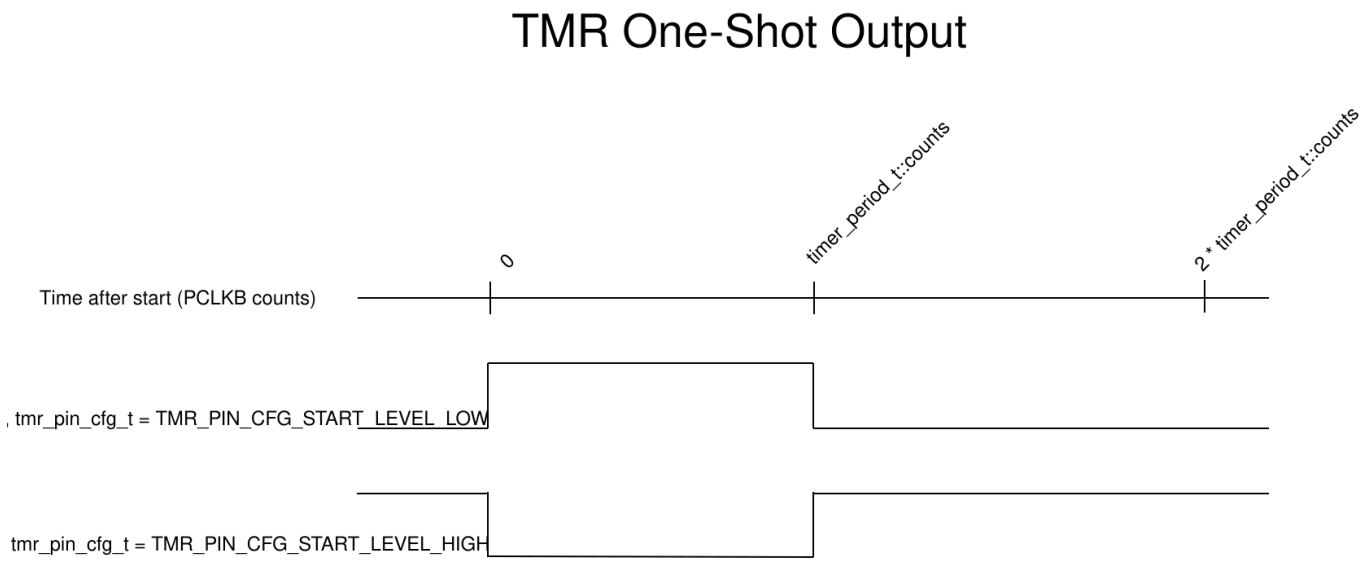


Figure 163: TMR One-Shot Output

Periodic Output

The TMOn pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square (from rising edge to rising edge) matches the period of the TMR timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

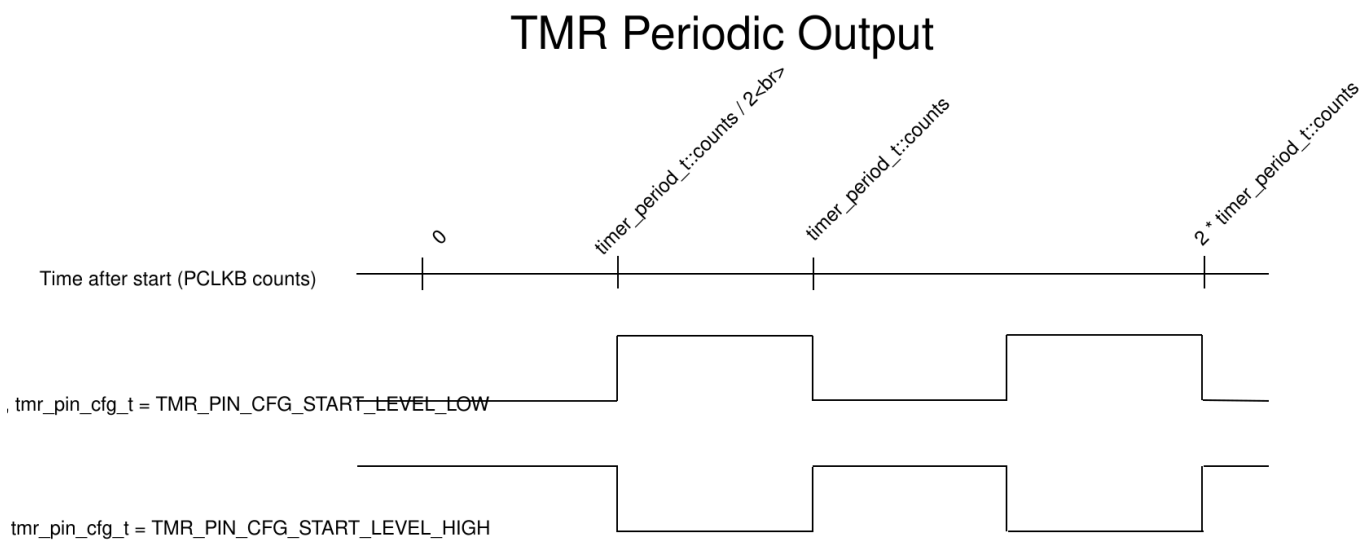


Figure 164: TMR Periodic Output

PWM Output

This module does not support in three-phase PWM output. The PWM output signal is low at the beginning of the cycle and high at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

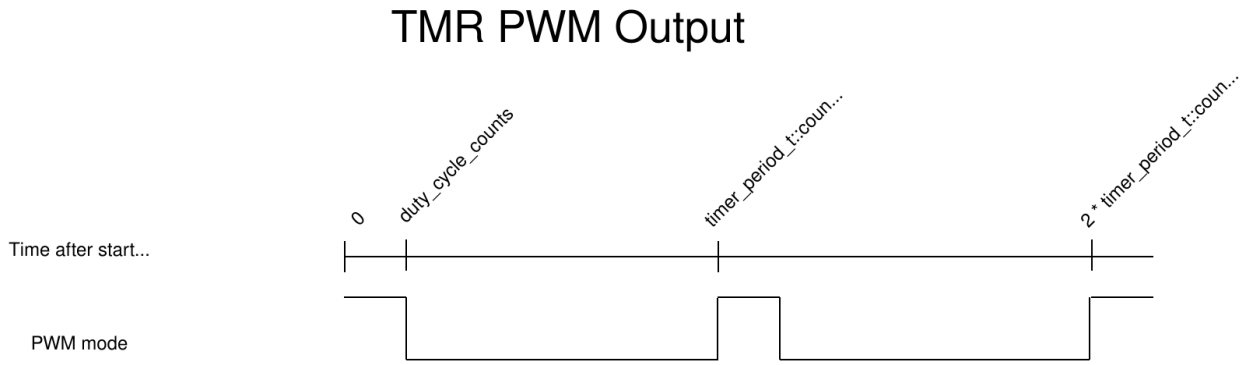


Figure 165: TMR PWM Output

Triggering ADC with TMR

The TMR timer can trigger the start request in response to compare match A This feature not support on MCU RX74M.

Triggering ELC Events with TMR

The TMR timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

Event Counting

Event counting can be done by selecting counting sources from TCCR, ELC events.

Note

In event counting mode, the application must call R_TMR_Start() to enable event counting. The counter will not change after calling R_TMR_Start() until an event occurs.

Controlling TMR with ELC Events

The TMR timer can be configured to start, count, and restart when an ELC event occurs.

Note

Event signal input and output are available for TMR channels supported by ELC on the target MCU.

Limitations

Developers should be aware of the following limitations when using the TMR:

- TMR operates with 2 units. Each unit has 2 channels with a maximum counter size of 16-bit per unit.
- When a channel is configured as a 16-bit timer, the remaining channel in the same unit cannot be used.
- When a channel is configured as an 8-bit timer, the remaining channel in the same unit can only be used as an 8-bit timer.

Examples

TMR Basic Example

This is a basic example of minimal use of the TMR in an application.

```
void tmr_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TMR_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TMR_Start(&g_timer0_ctrl);
}
```

TMR Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
```

```
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

TMR Free Running Counter Example

To use the TMR as a free running counter, select periodic mode and set the the Period to 0xFF for a 8-bit timer or 0xFFFF for a 16-bit timer.

```
void tmr_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TMR_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TMR_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_TMR_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TMR_StatusGet(&g_timer0_ctrl, &status);
}
```

TMR Period Update Example

This is an example of updating the period.

```
#define TMR_EXAMPLE_MSEC_PER_SEC (1000)
#define TMR_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
```

```
void tmr_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TMR_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TMR_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
  * - If LOCO or subclock is chosen in tmr_extended_cfg_t::clock_source
  * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
  * - If PCLKB is chosen in tmr_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
  * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_TMR_InfoGet function (it accounts for the clock source and divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * TMR_EXAMPLE_DESIRED_PERIOD_MSEC) /
TMR_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
```

```
    err = R_TMR_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

TMR Duty Cycle Update Example

This is an example of updating the duty cycle.

```
#define TMR_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define TMR_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void tmr_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TMR_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TMR_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_TMR_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. */
    uint32_t duty_cycle_counts = (current_period_counts *
TMR_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                 TMR_EXAMPLE_MAX_PERCENT;
 /* Set the calculated duty cycle. */
    err = R_TMR_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, 0);
    assert(FSP_SUCCESS == err);
}
```

TMR Cascaded Timers Example

This is an example of using underflow from an even TMR channel as the count source for the next

channel (in this case, TMR0 and TMR1).

```
/* This example shows how use cascaded timers. The count source for TMR channel 1 is
set to TMR0 underflow. */
void tmr_cascaded_timers_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers in any order. */
    err = R_TMR_Open(&g_timer_channel0_ctrl, &g_timer_channel0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_TMR_Open(&g_timer_channel1_ctrl, &g_timer_channel1_cfg);
    assert(FSP_SUCCESS == err);
 /* Start TMR channel 1 first. */
    (void) R_TMR_Start(&g_timer_channel1_ctrl);
    (void) R_TMR_Start(&g_timer_channel0_ctrl);
 /* (Optional) Stop TMR channel 0 first. */
    (void) R_TMR_Stop(&g_timer_channel0_ctrl);
    (void) R_TMR_Stop(&g_timer_channel1_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TMR_StatusGet(&g_timer_channel1_ctrl, &status);
}
```

Data Structures

struct	tmr_instance_ctrl_t
struct	tmr_extended_cfg_t

Enumerations

enum	tmr_adc_trigger_t
enum	tmr_counter_size_t

Data Structure Documentation

◆ tmr_instance_ctrl_t

struct tmr_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

Data Fields	
uint32_t	open
	Whether or not channel is open.
const timer_cfg_t *	p_cfg
	Pointer to initial configurations.
R_TMR_Type *	p_reg
	Base register for this channel.
uint32_t	period
	Current timer period (counts)
uint8_t	index
	register array index for this channel.
uint8_t	control_index
	This field indicates the index of the odd-numbered channel that will be used as the lower byte in 16-bit mode.
void(*	p_callback)(timer_callback_args_t *)
	Pointer to callback that is called when a timer_event_t occurs.
timer_callback_args_t *	p_callback_memory
	Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory.
void *	p_context
	Pointer to context to be passed into callback function.

◆ tmr_extended_cfg_t

struct tmr_extended_cfg_t		
Optional TMR extension data structure.		
Data Fields		
tmr_counter_size_t	counter_size	Selected timer 16-bit or 8-bit.
tmr_adc_trigger_t	adc_request_enable	A/D Converter Start Request.
uint8_t	output_compare_ipl	Output Compare match interrupt priority.
IRQn_Type	output_compare_irq	Output Compare match interrupt.

Enumeration Type Documentation

◆ tmr_adc_trigger_t

enum tmr_adc_trigger_t	
Trigger options to start A/D conversion.	
Enumerator	
TMR_ADC_TRIGGER_DISABLE	A/D conversion start request in response to compare match A is disabled.
TMR_ADC_TRIGGER_ENABLE	A/D conversion start request in response to compare match A is enable.

◆ tmr_counter_size_t

enum tmr_counter_size_t	
Trigger options to start A/D conversion.	
Enumerator	
TMR_COUNTER_SIZE_8_BIT	Counter timer 8-bit selected.
TMR_COUNTER_SIZE_16_BIT	Counter timer 16-bit selected.

Function Documentation

◆ R_TMR_Open()

fsp_err_t R_TMR_Open (timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg*)

Initializes the timer module and applies configurations. Implements timer_api_t::open.

TMR hardware does not support one-shot functionality natively. When using one-shot mode, the timer will be stopped in an ISR after the requested period has elapsed.

The TMR implementation of the general timer can accept a tmr_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_TMR_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL or the source divider is invalid or channel does not support a 16bit counter size or the period was not in the valid range of 1 to 0xFF in 8bit counter or 1 to 0xFFFF in 16bit counter or an input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IRQ_BSP_DISABLED	timer_cfg_t::mode is TIMER_MODE_ONE_SHOT or timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback. Or a required interrupt has not been enabled in the vector table.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.

◆ R_TMR_Start()

fsp_err_t R_TMR_Start (timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Return values

FSP_SUCCESS	Timer successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_TMR_Stop()

fsp_err_t R_TMR_Stop (timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Return values

FSP_SUCCESS	Timer successfully stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_TMR_Enable()

fsp_err_t R_TMR_Enable (timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, event count, or restart the counter. Implements timer_api_t::enable.

Return values

FSP_ERR_UNSUPPORTED	API not supported by TMR.

◆ R_TMR_Disable()

fsp_err_t R_TMR_Disable (timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, event count or restart the counter. Implements timer_api_t::disable.

Note

The timer could be running after R_TMR_Disable(). To ensure it is stopped, call R_TMR_Stop().

Return values

FSP_ERR_UNSUPPORTED	API not supported by TMR.

◆ R_TMR_Reset()

fsp_err_t R_TMR_Reset (timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

Return values

FSP_SUCCESS	Counter reset.
FSP_ERR_ASSERTION	p_ctrl is NULL
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_TMR_PeriodSet()

fsp_err_t R_TMR_PeriodSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts*)

Updates period. The new period is updated immediately. Implements timer_api_t::periodSet.

Warning

Stop the timer before calling this function if one-shot output is used.

Return values

FSP_SUCCESS	Period value updated.
FSP_ERR_ASSERTION	A required pointer was NULL, or the period was not in the valid range of 1 to 0xFF in 8bit counter or 1 to 0xFFFF in 16bit counter.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_TMR_DutyCycleSet()

fsp_err_t R_TMR_DutyCycleSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin*)

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_TMR_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. */
   uint32_t duty_cycle_counts = (current_period_counts *
TMR_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                TMR_EXAMPLE_MAX_PERCENT;
/* Set the calculated duty cycle. */
   err = R_TMR_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, 0);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Duty cycle updated.
FSP_ERR_ASSERTION	A required pointer was NULL, or the pin was invalid.
FSP_ERR_INVALID_ARGUMENT	Duty cycle was not in the valid range of 0 to period (counts) - 1
FSP_ERR_NOT_OPEN	The instance control structure is not opened.
FSP_ERR_UNSUPPORTED	TMR_CFG_OUTPUT_SUPPORT_ENABLE is 0.

◆ R_TMR_CompareMatchSet()

fsp_err_t R_TMR_CompareMatchSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel*)

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

Return values

FSP_ERR_UNSUPPORTED	TMR compare match is not supported.

◆ R_TMR_InfoGet()

fsp_err_t R_TMR_InfoGet (timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info*)

Gets timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Return values

FSP_SUCCESS	Period, count direction, and frequency stored in p_info.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_TMR_StatusGet()

fsp_err_t R_TMR_StatusGet (timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status*)

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Return values

FSP_SUCCESS	Current timer state and counter value set successfully.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_TMR_AdcTriggerSet()

fsp_err_t R_TMR_AdcTriggerSet (timer_ctrl_t *const *p_ctrl*, tmr_adc_trigger_t *adc_request_enable*)

Set A/D converter start request compare match value.

Return values

FSP_SUCCESS	Counter value updated.
FSP_ERR_ASSERTION	p_ctrl was NULL or channel unsupport Adc trigger.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device 16-bit timer.
FSP_ERR_UNSUPPORTED	The feature not available on this device.

◆ R_TMR_CallbackSet()

fsp_err_t R_TMR_CallbackSet (timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_TMR_Close()

fsp_err_t R_TMR_Close (timer_ctrl_t *const *p_ctrl*)

Stops counter, clears internal driver data. Implements timer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

5.2.20 Transfer

Modules

Detailed Description

Transfer Modules.

Modules

	DMAC Transfer (r_dmac) Driver for the DMAC peripheral on RX MCUs. This module implements the Transfer Interface.
	Transfer (r_dtc) Driver for the DTC peripheral on RX MCUs. This module implements the Transfer Interface.
	EXDMAC Transfer (r_exdmac) Driver for the EXDMAC peripheral on RX MCUs. This module implements the Transfer Interface.

5.2.20.1 DMAC Transfer (r_dmac)

Modules » Transfer

Functions

fsp_err_t	R_DMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_DMAC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_DMAC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_DMAC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_DMAC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)

fsp_err_t	R_DMAC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info)
fsp_err_t	R_DMAC_Reload (transfer_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers)
fsp_err_t	R_DMAC_Close (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_CallbackSet (transfer_ctrl_t *const p_api_ctrl, void(*p_callback)(dmac_callback_args_t *), void *const p_context, dmac_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the DMAC peripheral on RX MCUs. This module implements the Transfer Interface.

Overview

The Direct Memory Access Controller (DMAC) transfers data from one memory location to another without using the CPU.

Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
  - Repeat-Block transfer (Not available on all MCUs)
- Address increment, decrement, fixed, or offset modes
- Triggered by interrupt source
  - Refer to the Interrupt Vector Table in Interrupt Controller Unit chapter of the hardware manual for interrupt vector numbers of interrupt sources.
- Supports 1, 2, and 4 byte data units
- Configurable interrupt select options
  - Clear interrupt status at the beginning of each transfer
  - Issue an interrupt at the end of each transfer

Configuration

Build Time Configurations for r_dmac

The following build time configurations are defined in fsp_cfg/r_dmac_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Transfer > Transfer (r_dmac)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_dmac).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_transfer0	Module name.
Channel	Value must be a non-negative integer	0	Specify the hardware channel.
Mode	MCU Specific Options		Select the transfer mode. Normal: One transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks.
Transfer Size	• 1 Byte • 2 Bytes • 4 Bytes	2 Bytes	Select the transfer size.
Destination Address Mode	• Fixed • Offset addition • Incremented • Decremented	Fixed	Select the address mode for the destination.
Source Address Mode	• Fixed • Offset addition • Incremented • Decremented	Fixed	Select the address mode for the source.
Repeat Area (Unused in Normal Mode)	• Destination • Source	Source	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Number of Transfers	Value must be a non-	1	Specify the number of

	negative integer		transfers for repeat and normal mode or block size for repeat-block transfer mode.
Number of Blocks (Valid only in Repeat,Block or Repeat-Block Mode)	Value must be a non-negative integer	0	Specify the number of blocks to transfer in Repeat,Block or Repeat-Block mode.
Activation Source	MCU Specific Options		Select the DMAC transfer start event.
Callback	Name must be a valid C symbol	NULL	A user callback that is called at the end of the transfer.
Transfer End Interrupt Priority	MCU Specific Options		Select the transfer end interrupt priority.
Interrupt Frequency	• Interrupt after all transfers have completed • Interrupt after each block, or repeat size is transferred	Interrupt after all transfers have completed	Select to have interrupt after each transfer or after last transfer.
Interrupt Select	• Clear interrupt status flag at the beginning of each transfer • Issues an interrupt to the CPU at the end of each transfer	Clear interrupt status flag at the beginning of each transfer	Select to clear the interrupt status flag at the beginning of each transfer or issues an interrupt at the end of each transfer.
Offset value (Valid only when address mode is \'Offset\')	Value must be a 24 bit signed integer.	1	Offset value is added to the address after each transfer.
Source Buffer Size	Value must be a non-negative integer with a maximum configurable value of 65535.	1	Specify the size of whole source buffer (valid only for Repeat-Block transfer mode with source address update mode other than offset addition).

Clock Configuration

The DMAC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RX Configuration editor prior to a build, or by using the CGC module at run-time.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Source and Destination Configuration

R_DMAC_Reset() API function should be called to set the source and destination before starting transfer operation.

Transfer Modes

The DMAC Module supports four modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time the configured interrupt source is received by the DMAC channel. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, incremented or decremented, or add an offset to the next data unit after each transfer. A 16-bit counter decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,1024]. When the transfer counter reaches 0, the counter is reset to its configured value, the repeat area (source or destination address) resets to its starting address and the block count remaining will decrement by 1. When the block count reaches 0, transfers will no longer be triggered by the interrupt source and the CPU may be interrupted to signal that all transfers have finished.
- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.
- **Repeat-Block Mode** - In repeat-block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can be configured to a 16 bit number. If the destination address mode is offset mode, maximum configurable number of blocks is 0xFFFF for block size(length) of one with data transfer size as byte,0x7FFF for block size of one with data transfer size as half word and 0x3FFF for block size of one with data size as word. After each block transfer the source address and the destination address will be incremented or decremented to the next block address. In case of offset address mode for source address, the source address size is the total size of source buffer after which the source area is rolled over, block size can be smaller than the source buffer size.For source address mode as offset mode, the maximum configurable source buffer size is 0xFFFF for transfer data size of a byte,0x7FFF for transfer data size of half word and 0x3FFF for transfer data size of word. Repeat-block mode can be used to implement single ring buffer to multiple ring buffer transfer type design.

Selecting the DTC or DMAC or EXDMAC

The Transfer API is implemented by DTC, DMAC and EXDMAC so that applications can switch between the DTC, DMAC and the EXDMAC. When selecting between them, consider these factors:

	DTC	DMAC	EXDMAC
Repeat Mode	• Repeats forever • Max repeat size is 256 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes

Block Mode	• Max block size is 256 x 4 bytes	• Max block size is 1024 x 4 bytes	• Max block size is 1024 x 4 bytes
Channels	• One instance per interrupt	• MCU specific (8 channels or less)	• MCU specific (2 channels)
Chained Transfers	• Supported	• Not Supported	• Not Supported
Software Trigger	• Must use the ICU Software Interrupt	• Has support for software trigger by selecting the software trigger activation source • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT	• Has support for software trigger by selecting the software trigger activation source • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT
Offset Address Mode	• Not supported	• Supported	• Supported
Transfer Space	• 0000 0000h to FFFF FFFFh excluding reserved areas	• 0000 0000h to FFFF FFFFh excluding reserved areas	Exclusively for external bus transfer • SRAM0/SRAM1 (0200 0000h to 0217 FFFFh) • CS0–CS7 / SDRAM (7000 0000h to 7FFF FFFFh) • xSPI0/xSPI1 (2000 0000h to 5FFF FFFFh)

Interrupts

The DTC, DMAC and EXDMAC interrupts behave differently. The DTC uses the activating peripheral's IRQ as the interrupt source whereas each DMAC and EXDMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer	Interrupt after last transfer

Block Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block	Interrupt after last block

Repeat-block Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	N/A	N/A	N/A
TRANSFER_IRQ_END	N/A	Interrupt after last block	Interrupt after last block

Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC and EXDMAC store all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.

Offset Address Mode

When the source or destination mode is configured to offset mode, a configurable offset is added to the source or destination pointer after each transfer. The offset is a signed 24 bit number.

Examples

Basic Example

This is a basic example of minimal use of the DMAC in an application. In this case, one or more interrupt sources have been routed to the DMAC for handling so it only needs to be enabled to start accepting transfers.

```
void dmac_minimal_example (void)
{
```

```
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the DMAC so that it responds to transfer requests. */
    err = R_DMAC_Enable(&g_transfer_ctrl);
    assert(FSP_SUCCESS == err);
}
```

CRC32 Example

In this example the DMAC is used to feed the CRC peripheral to perform a CRC32 operation.

```
volatile bool g_transfer_complete = false;
void dmac_callback (dmac_callback_args_t * cb_data)
{
 FSP_PARAMETER_NOT_USED(cb_data);
    g_transfer_complete = true;
}
void dmac_crc_example (void)
{
    uint8_t p_src[TRANSFER_LENGTH];
 /* Initialize p_src to [ABC..OP] */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       p_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Set transfer source address to p_src */
    g_transfer_cfg.p_info->p_src = (void *) p_src;
 /* Set transfer destination address to the CRC data input register */
    g_transfer_cfg.p_info->p_dest = (void *) &R_CRC->CRCDIR;
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
/* Enable DMAC transfers. */
   (void) R_DMAC_Enable(&g_transfer_ctrl);
/* Open the CRC module. */
   err = R_CRC_Open(&g_crc_ctrl, &g_crc_cfg);
   assert(FSP_SUCCESS == err);
/* Clear the transfer complete flag. */
   g_transfer_complete = false;
/* Trigger the transfer using software. */
   err = R_DMAC_SoftwareStart(&g_transfer_ctrl, TRANSFER_START_MODE_SINGLE);
   assert(FSP_SUCCESS == err);
while (!g_transfer_complete)
   {
/* Wait for transfer complete interrupt */
   }
/* Get CRC result and perform final XOR. */
   uint32_t crc32;
   (void) R_CRC_CalculatedValueGet(&g_crc_ctrl, &crc32);
   crc32 ^= CRC32_FINAL_XOR_VALUE;
/* Verify that the CRC32 is calculated correctly. */
/* CRC32("ABCD...NOP") = 0xE0E8FF4D. */
const uint32_t expected_crc32 = 0xE0E8FF4D;
if (expected_crc32 != crc32)
   {
/* Handle any CRC errors. This function should be defined by the user. */
      handle_crc_error();
   }
}
```

Data Structures

struct	dmac_instance_ctrl_t
struct	dmac_extended_cfg_t

Macros

#define	DMAC_MAX_NORMAL_TRANSFER_LENGTH

#define	DMAC_MAX_REPEAT_TRANSFER_LENGTH
#define	DMAC_MAX_BLOCK_TRANSFER_LENGTH
#define	DMAC_MAX_REPEAT_COUNT
#define	DMAC_MAX_BLOCK_COUNT

Data Structure Documentation

◆ dmac_instance_ctrl_t

struct dmac_instance_ctrl_t
Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

◆ dmac_extended_cfg_t

struct dmac_extended_cfg_t	
DMAC transfer configuration extension. This extension is required.	
Data Fields	
uint8_t	channel
	Channel number, does not apply to all HAL drivers.
IRQn_Type	irq
	DMAC interrupt number.
uint8_t	ipl
	DMAC interrupt priority.
int32_t	offset
	Offset value used with transfer_addr_mode_t::TRANSFER_ADDR_MODE_OFFSET.
uint16_t	src_buffer_size
elc_event_t	activation_source

IRQn_Type	irq_activation_source
void(*	p_callback)(dmac_callback_args_t *cb_data)
void *	p_context

Field Documentation

◆ src_buffer_size

uint16_t dmac_extended_cfg_t::src_buffer_size

Source ring buffer size for TRANSFER_MODE_REPEAT_BLOCK.

◆ activation_source

elc_event_t dmac_extended_cfg_t::activation_source

Select which event will trigger the transfer.

Note

Select ELC_EVENT_NONE for software activation in order to use softwareStart and softwareStart to trigger transfers.

◆ irq_activation_source

IRQn_Type dmac_extended_cfg_t::irq_activation_source

Select which interrupt source will trigger the transfer.

◆ p_callback

void(* dmac_extended_cfg_t::p_callback) (dmac_callback_args_t *cb_data)

Callback for transfer end interrupt.

◆ p_context

void* dmac_extended_cfg_t::p_context

Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

Macro Definition Documentation

◆ DMAC_MAX_NORMAL_TRANSFER_LENGTH

#define DMAC_MAX_NORMAL_TRANSFER_LENGTH

Max configurable number of transfers in TRANSFER_MODE_NORMAL.

◆ DMAC_MAX_REPEAT_TRANSFER_LENGTH

#define DMAC_MAX_REPEAT_TRANSFER_LENGTH
Max number of transfers per repeat for TRANSFER_MODE_REPEAT.

◆ DMAC_MAX_BLOCK_TRANSFER_LENGTH

#define DMAC_MAX_BLOCK_TRANSFER_LENGTH
Max number of transfers per block in TRANSFER_MODE_BLOCK

◆ DMAC_MAX_REPEAT_COUNT

#define DMAC_MAX_REPEAT_COUNT
Max configurable number of repeats to transfer in TRANSFER_MODE_REPEAT

◆ DMAC_MAX_BLOCK_COUNT

#define DMAC_MAX_BLOCK_COUNT
Max configurable number of blocks to transfer in TRANSFER_MODE_BLOCK

Function Documentation

◆ R_DMAC_Open()

fsp_err_t R_DMAC_Open (transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg*)

Configure a DMAC channel.

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The configured channel is invalid.
FSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.

◆ R_DMAC_Reconfigure()

fsp_err_t R_DMAC_Reconfigure (transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info*)

Reconfigure the transfer with new transfer info.

Return values

FSP_SUCCESS	Transfer is configured and will start when trigger occurs.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_ENABLED	DMAC is not enabled. The current configuration must not be valid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_Reset()

fsp_err_t R_DMAC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers.

Return values

FSP_SUCCESS	Transfer reset successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_ENABLED	DMAC is not enabled. The current configuration must not be valid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_SoftwareStart()

fsp_err_t R_DMAC_SoftwareStart (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

If the mode is TRANSFER_START_MODE_SINGLE initiate a single transfer with software. If the mode is TRANSFER_START_MODE_REPEAT continue triggering transfers until all of the transfers are completed.

Return values

FSP_SUCCESS	Transfer started written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
FSP_ERR_UNSUPPORTED	Handle was not configured for software activation.

◆ R_DMAC_SoftwareStop()

fsp_err_t R_DMAC_SoftwareStop (transfer_ctrl_t *const *p_api_ctrl*)

Stop software transfers if they were started with TRANSFER_START_MODE_REPEAT.

Return values

FSP_SUCCESS	Transfer stopped written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_Enable()

fsp_err_t R_DMAC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers for the configured activation source.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_Disable()

fsp_err_t R_DMAC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfers so that they are no longer triggered by the activation source.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_InfoGet()

fsp_err_t R_DMAC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info*)

Set driver specific information in provided pointer.

Return values

FSP_SUCCESS	Information has been written to p_info.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_DMAC_Reload()

fsp_err_t R_DMAC_Reload (transfer_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *num_transfers*)

To update next transfer information without interruption during transfer.

Return values

FSP_ERR_UNSUPPORTED	This feature is not supported.

◆ R_DMAC_Close()

fsp_err_t R_DMAC_Close (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer and clean up internal data. Implements transfer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_CallbackSet()

fsp_err_t R_DMAC_CallbackSet (transfer_ctrl_t *const *p_api_ctrl*, void(*)(dmac_callback_args_t *) *p_callback*, void *const *p_context*, dmac_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.20.2 Transfer (r_dtc)

Modules » Transfer

Functions

fsp_err_t	R_DTC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_DTC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_DTC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_DTC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_DTC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)

fsp_err_t	R_DTC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_properties)
fsp_err_t	R_DTC_Reload (transfer_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers)
fsp_err_t	R_DTC_CallbackSet (transfer_ctrl_t *const p_api_ctrl, void(*p_callback)(transfer_callback_args_t *), void *const p_context, transfer_callback_args_t *const p_callback_memory)
fsp_err_t	R_DTC_Close (transfer_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the DTC peripheral on RX MCUs. This module implements the Transfer Interface.

Overview

The Data Transfer Controller (DTC) transfers data from one memory location to another without using the CPU.

The DTC uses a RAM based vector table. Each entry in the vector table corresponds to an entry in the ISR vector table. When the DTC is triggered by an interrupt, it reads the DTC vector table, fetches the transfer information, and then executes the transfer. After the transfer is executed, the DTC writes the updated transfer info back to the location pointed to by the DTC vector table.

Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
- Chain transfers
- Address increment, decrement or fixed modes
- Can be triggered by any event that has reserved a slot in the interrupt vector table.
  - Some exceptions apply, see the Interrupt Vector Table in the Vector Table section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

Configuration

Build Time Configurations for r_dtc

The following build time configurations are defined in fsp_cfg/r_dtc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Linker section to keep DTC vector table	Manual Entry	.fsp_dtc_vector_table	Section to place the DTC vector table.

Configurations for Transfer > Transfer (r_dtc)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_dtc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_transfer0	Module name.
Mode	• Normal • Repeat • Block	Normal	Select the transfer mode. Normal: One transfer per activation, transfer ends after Number of Transfers. Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer repeats until stopped. Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks.
Transfer Size	• 1 Byte • 2 Bytes • 4 Bytes	2 Bytes	Select the transfer size.
Destination Address Mode	• Fixed • Incremented • Decremented	Fixed	Select the address mode for the destination.
Source Address Mode	• Fixed • Incremented • Decremented	Fixed	Select the address mode for the source.
Repeat Area (Unused in Normal Mode)	• Destination • Source	Source	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.

Interrupt Frequency	• After all transfers have completed • After each transfer	After all transfers have completed	Select to have interrupt after each transfer or after last transfer.
Number of Transfers	Value must be a non-negative integer	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	Must be a valid non-negative integer with a maximum configurable value of 65536. Applicable only in Block Mode.	0	Specify the number of blocks to transfer in Block mode.
Number of Transfer Descriptors	Value must be a non-negative integer	1	Specify the number of transfer descriptors. Users have to initialize descriptors if its value is greater than 1.
Activation Source	MCU Specific Options		Select the DTC transfer start event.

Clock Configuration

The DTC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RX Configuration editor prior to a build or by using the CGC module at runtime.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Source and Destination Configuration

R_DTC_Reset() API function should be called to set the Source and Destination address before starting the transfer operation.

Transfer Modes

The DTC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time an interrupt is received by the DTC. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, incremented or decremented to the next data unit after each transfer. A 16-bit counter (length) decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,256]. When the transfer counter reaches 0, the counter is reset to its configured value and the repeat area (source or destination address) resets to its starting address and transfers will still be triggered by the interrupt.
- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be

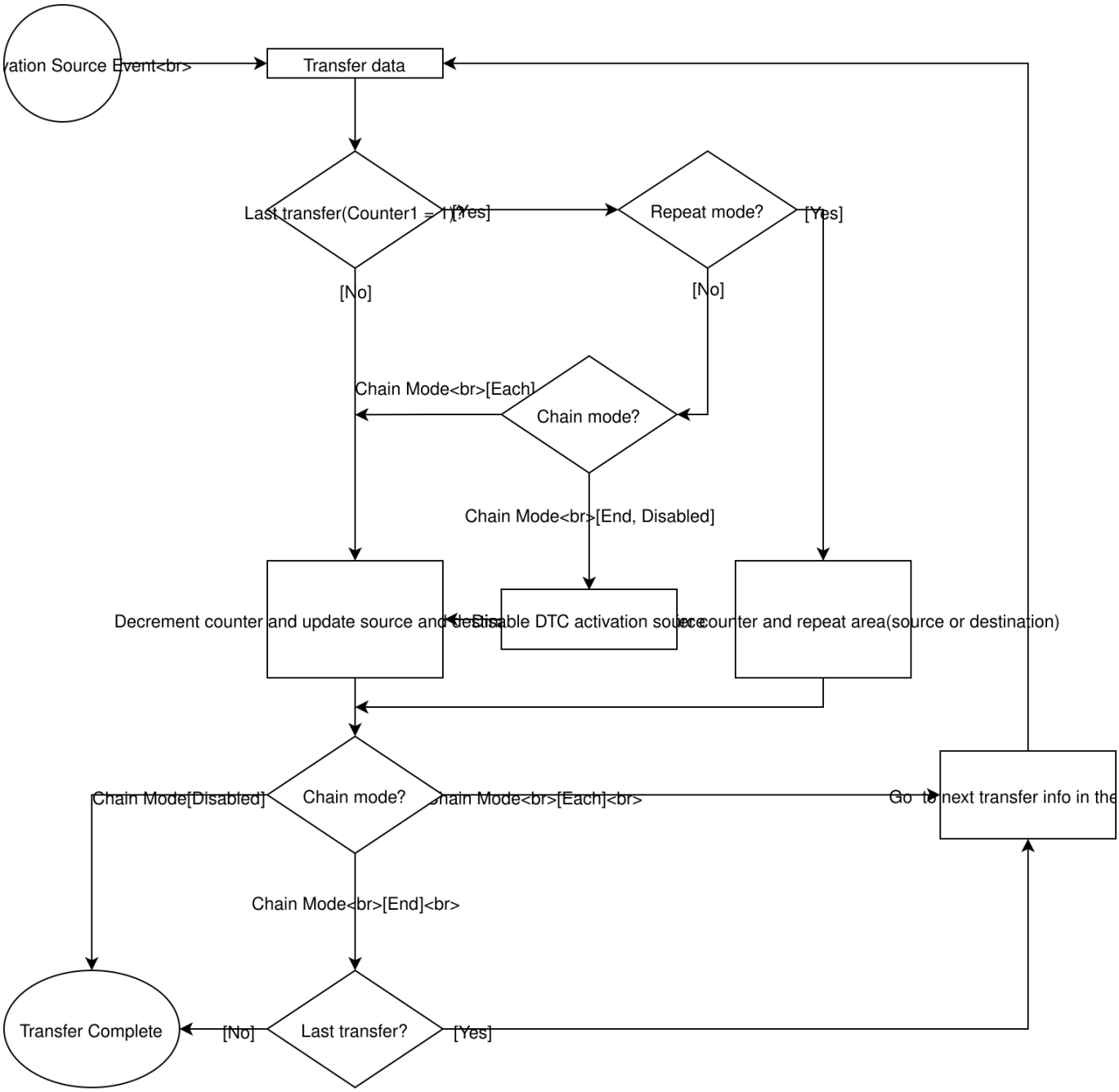
set to an integer in the range [1,256]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

Note

The source and destination address of the transfer must be aligned to the configured data unit.

Chaining Transfers

Multiple transfers can be configured for the same interrupt source by specifying an array of transfer_info_t structs instead of just passing a pointer to one. In this configuration, every transfer_info_t struct must be configured for a chain mode except for the last one. There are two types of chain modes; CHAIN_MODE_EACH and CHAIN_MODE_END. If a transfer is configured in CHAIN_MODE_EACH then it triggers the next transfer in the chain after it completes each transfer. If a transfer is configured in CHAIN_MODE_END then it triggers the next transfer in the chain after it completes its last transfer.



nter refers to transfer_info_t::length in normal and repeat mode and transfer_info_t::num_blocks in block mode.

Figure 166: DTC Transfer Flowchart

Selecting the DTC or DMAC or EXDMAC

The Transfer API is implemented by DTC, DMAC and EXDMAC so that applications can switch between the DTC, DMAC and the EXDMAC. When selecting between them, consider these factors:

	DTC	DMAC	EXDMAC
Repeat Mode	• Repeats forever • Max repeat size is 256 x 4 bytes	• Configurable number of repeats	• Configurable number of repeats

		• Max repeat size is 1024 x 4 bytes	• Max repeat size is 1024 x 4 bytes
Block Mode	• Max block size is 256 x 4 bytes	• Max block size is 1024 x 4 bytes	• Max block size is 1024 x 4 bytes
Channels	• One instance per interrupt	• MCU specific (8 channels or less)	• MCU specific (2 channels)
Chained Transfers	• Supported	• Not Supported	• Not Supported
Software Trigger	• Must use the ICU Software Interrupt	• Has support for software trigger by selecting the software trigger activation source • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT	• Has support for software trigger by selecting the software trigger activation source • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT
Offset Address Mode	• Not supported	• Supported	• Supported
Transfer Space	• 0000 0000h to FFFF FFFFh excluding reserved areas	• 0000 0000h to FFFF FFFFh excluding reserved areas	Exclusively for external bus transfer • SRAM0/SRAM1 (0200 0000h to 0217 FFFFh) • CS0–CS7 / SDRAM (7000 0000h to 7FFF FFFFh) • xSPI0/xSPI1 (2000 0000h to 5FFF FFFFh)

Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC and EXDMAC store all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.
- The DTC interrupts the CPU using the activation source's IRQ. Each DMAC and EXDMAC channel has its own IRQ.
- The necessary alignment of the transfer_info_t structs depends on the underlying MCU. The DTC_TRANSFER_INFO_ALIGNMENT macro is supplied to align the structs as necessary.

Interrupts

The DTC, DMAC and EXDMAC interrupts behave differently. The DTC uses the activating peripheral's IRQ as the interrupt source whereas each DMAC and EXDMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer	Interrupt after last transfer

Block Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block	Interrupt after last block

Repeat-block Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	N/A	N/A	N/A
TRANSFER_IRQ_END	N/A	Interrupt after last block	Interrupt after last block

Note

DTC_VECTOR_TABLE_SIZE = (ICU_IRQ_SOURCES x 4) Bytes

Peripheral Interrupts and DTC

When an interrupt is configured to trigger DTC transfers, the peripheral ISR will trigger on the following conditions:

- Each transfer completed (transfer_info_t::irq = TRANSFER_IRQ_EACH)
- Last transfer completed (transfer_info_t::irq = TRANSFER_IRQ_END)

For example, if SCI1_RXI is configured to trigger DTC transfers and a SCI1_RXI event occurs, the interrupt will not fire until the DTC transfer is completed. If the DTC transfer_info_t::irq is configured to only interrupt on the last transfer, then no RXI interrupts will occur until the last transfer is completed.

Note

1. The DTC activation source must be enabled in the ICU in order to trigger DTC transfers (Modules that are designed to integrate the R_DTC module will automatically handle this).
2. The DTC prioritizes activation sources by granting the smaller interrupt vector numbers higher priority. The priority of interrupts to the CPU is determined by the ICU priority.

Low Power Modes

DTCST must be set to 0 before transitioning to any of the following:

- Module-stop state
- Software Standby mode
- Deep Software Standby mode

Note

1. R_LPM Module stops the DTC before entering deep software standby mode and software standby mode transition.
2. For more information see "Module-Stop Function" in the DTC section of the relevant hardware manual.

Limitations

Developers should be aware of the following limitations when using the DTC:

- If the DTC is configured to service many different activation sources, the system could run into performance issues due to memory contention. To address this issue, it is recommended that the DTC vector table and transfer information be moved to their own dedicated memory area. This allows memory accesses from different BUS Masters to occur in parallel.

The following features are NOT supported in RX DTC:

- Sequence Transfer mode
- Address Displacement feature
- Write-back disable feature

Examples

Basic Example

This is a basic example of minimal use of the DTC in an application.

```
void dtc_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Enable the DTC to handle incoming transfer requests. */
    err = R_DTC_Enable(&g_transfer_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	dtc_extended_cfg_t
struct	dtc_instance_ctrl_t

Macros

#define	DTC_MAX_NORMAL_TRANSFER_LENGTH
#define	DTC_MAX_REPEAT_TRANSFER_LENGTH
#define	DTC_MAX_BLOCK_TRANSFER_LENGTH
#define	DTC_MAX_BLOCK_COUNT
#define	DTC_TRANSFER_INFO_ALIGNMENT

Data Structure Documentation

◆ dtc_extended_cfg_t

struct dtc_extended_cfg_t		
DTC transfer configuration extension. This extension is required.		
Data Fields		
IRQn_Type	activation_source	Select which IRQ will trigger the transfer.

◆ dtc_instance_ctrl_t

struct dtc_instance_ctrl_t
Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

Macro Definition Documentation

◆ DTC_MAX_NORMAL_TRANSFER_LENGTH

#define DTC_MAX_NORMAL_TRANSFER_LENGTH
Max configurable number of transfers in NORMAL MODE

◆ DTC_MAX_REPEAT_TRANSFER_LENGTH

#define DTC_MAX_REPEAT_TRANSFER_LENGTH
Max number of transfers per repeat for REPEAT MODE

◆ DTC_MAX_BLOCK_TRANSFER_LENGTH

#define DTC_MAX_BLOCK_TRANSFER_LENGTH
Max number of transfers per block in BLOCK MODE

◆ DTC_MAX_BLOCK_COUNT

#define DTC_MAX_BLOCK_COUNT
Max configurable number of blocks to transfer in BLOCK MODE

◆ DTC_TRANSFER_INFO_ALIGNMENT

#define DTC_TRANSFER_INFO_ALIGNMENT
Alignment required for transfer_info_t structures.

Function Documentation

◆ R_DTC_Open()

fsp_err_t R_DTC_Open (transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg*)

Configure the vector table if it hasn't been configured, enable the Module and copy the pointer to the transfer info into the DTC vector table. Implements transfer_api_t::open.

Example:

```
/* Open the transfer instance with initial configuration. */

fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
```

Return values

FSP_SUCCESS	Successful open. Transfer transfer info pointer copied to DTC Vector table. Module started. DTC vector table configured.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_UNSUPPORTED	Address Mode Offset is selected.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.
FSP_ERR_IN_USE	The index for this IRQ in the DTC vector table is already configured.
FSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.

◆ R_DTC_Reconfigure()

fsp_err_t R_DTC_Reconfigure (transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info*)

Copy pointer to transfer info into the DTC vector table and enable transfer in ICU. Implements transfer_api_t::reconfigure.

Return values

FSP_SUCCESS	Transfer is configured and will start when trigger occurs.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_NOT_ENABLED	Transfer source address is NULL or is not aligned correctly. Transfer destination address is NULL or is not aligned correctly.

Note

p_info must persist until all transfers are completed.

◆ R_DTC_Reset()

fsp_err_t R_DTC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers. Implements transfer_api_t::reset.

Return values

FSP_SUCCESS	Transfer reset successfully (transfers are enabled).
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_NOT_ENABLED	Transfer source address is NULL or is not aligned correctly. Transfer destination address is NULL or is not aligned correctly.

◆ R_DTC_SoftwareStart()

fsp_err_t R_DTC_SoftwareStart (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

Placeholder for unsupported softwareStart function. Implements transfer_api_t::softwareStart.

Return values

FSP_ERR_UNSUPPORTED	DTC software start is not supported.

◆ R_DTC_SoftwareStop()

fsp_err_t R_DTC_SoftwareStop (transfer_ctrl_t *const *p_api_ctrl*)

Placeholder for unsupported softwareStop function. Implements transfer_api_t::softwareStop.

Return values

FSP_ERR_UNSUPPORTED	DTC software stop is not supported.

◆ R_DTC_Enable()

fsp_err_t R_DTC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers on this activation source. Implements transfer_api_t::enable.

Example:

```
/* Enable the DTC to handle incoming transfer requests. */
   err = R_DTC_Enable(&g_transfer_ctrl);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Transfers will be triggered by the activation source
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_UNSUPPORTED	Address Mode Offset is selected.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

◆ R_DTC_Disable()

fsp_err_t R_DTC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer on this activation source. Implements transfer_api_t::disable.

Return values

FSP_SUCCESS	Transfers will not occur on activation events.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_DTC_InfoGet()

fsp_err_t R_DTC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_properties*)

Provides information about this transfer. Implements transfer_api_t::infoGet.

Return values

FSP_SUCCESS	p_info updated with current instance information.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_DTC_Reload()

fsp_err_t R_DTC_Reload (transfer_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *num_transfers*)

To update next transfer information without interruption during transfer.

Return values

FSP_ERR_UNSUPPORTED	This feature is not supported.

◆ R_DTC_CallbackSet()

fsp_err_t R_DTC_CallbackSet (transfer_ctrl_t *const *p_api_ctrl*, void(*)(transfer_callback_args_t *) *p_callback*, void *const *p_context*, transfer_callback_args_t *const *p_callback_memory*)

Placeholder for unsupported callbackset function. Implements transfer_api_t::callbackSet.

Return values

FSP_ERR_UNSUPPORTED	DTC does not support direct callbacks.

◆ R_DTC_Close()

fsp_err_t R_DTC_Close (transfer_ctrl_t *const *p_api_ctrl*)
Disables DTC activation in the ICU, then clears transfer data from the DTC vector table. Implements transfer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

5.2.20.3 EXDMAC Transfer (r_exdmac)

Modules » Transfer

Functions

fsp_err_t	R_EXDMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_EXDMAC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_EXDMAC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_EXDMAC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_EXDMAC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_EXDMAC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_EXDMAC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_EXDMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info)
fsp_err_t	R_EXDMAC_Reload (transfer_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers)
fsp_err_t	R_EXDMAC_Close (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_EXDMAC_CallbackSet (transfer_ctrl_t *const p_api_ctrl,

	void(*p_callback)(exdmac_callback_args_t *), void *const p_context, exdmac_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the EXDMAC peripheral on RX MCUs. This module implements the Transfer Interface.

Overview

The Direct Memory Access Controller (EXDMAC) designed exclusively for external bus. It can transfers data from one memory location to another without using the CPU.

Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
  - Repeat-Block transfer (Not available on all MCUs)
- Address increment, decrement, fixed, or offset modes
- Triggered by interrupt source of software configurable interrupts and external pins interrupts
  - Refer to the Interrupt Controller Unit Chapter of the hardware manual for the interrupt source numbers and the interrupt vector table for external pin interrupts.
- Supports 1, 2, and 4 byte data units

Configuration

Build Time Configurations for r_exdmac

The following build time configurations are defined in fsp_cfg/r_exdmac_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Transfer > Transfer (r_exdmac)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_exdmac).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_transfer0	Module name.
Channel	Value must be a non-negative integer	0	Specify the hardware channel.
Mode	MCU Specific Options		Select the transfer mode. Normal: One

			transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks; Repeat-Block: Number of Blocks per activation, Repeat Area address reset after each Block transfer, transfer ends after Number of Blocks.
Transfer Size	• 1 Byte • 2 Bytes • 4 Bytes	2 Bytes	Select the transfer size.
Destination Address Mode	• Fixed • Offset addition • Incremented • Decremented	Fixed	Select the address mode for the destination.
Source Address Mode	• Fixed • Offset addition • Incremented • Decremented	Fixed	Select the address mode for the source.
Repeat Area (Unused in Normal Mode)	• Destination • Source	Source	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Number of Transfers	Value must be a non-negative integer	1	Specify the number of transfers for repeat and normal mode or block size for repeat-block transfer mode.
Number of Blocks (Valid only in Repeat,Block or Repeat-Block Mode)	Value must be a non-negative integer	0	Specify the number of blocks to transfer in Repeat,Block or Repeat-Block mode.
Activation Source	MCU Specific Options		Select the EXDMAC

			transfer start event.
Callback	Name must be a valid C symbol	NULL	A user callback that is called at the end of the transfer.
Transfer End Interrupt Priority	MCU Specific Options		Select the transfer end interrupt priority.
Interrupt Frequency	• Interrupt after all transfers have completed • Interrupt after each block, or repeat size is transferred	Interrupt after all transfers have completed	Select to have interrupt after each transfer or after last transfer.
Offset value (Valid only when address mode is \'Offset\')	Value must be a 24 bit signed integer.	1	Offset value is added to the address after each transfer.
Source Buffer Size	Value must be a non-negative integer with a maximum configurable value of 65535.	1	Specify the size of whole source buffer (valid only for Repeat-Block transfer mode with source address update mode other than offset addition).

Clock Configuration

The EXDMAC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RX Configuration editor prior to a build, or by using the CGC module at run-time.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Source and Destination Configuration

R_EXDMAC_Reset() API function should be called to set the source and destination before starting transfer operation.

Transfer Modes

The EXDMAC Module supports four modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time the configured interrupt source is received by the EXDMAC channel. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment, decrement, or add an offset to the next data unit after each transfer. A 16-bit counter decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is

limited to an integer in the range[1,1024]. When the transfer counter reaches 0, the counter is reset to its configured value, the repeat area (source or destination address) resets to its starting address and the block count remaining will decrement by 1. When the block count reaches 0, transfers will no longer be triggered by the interrupt source and the CPU may be interrupted to signal that all transfers have finished.

- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.
- **Repeat-Block Mode** - In repeat-block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can be configured to a 16 bit number. If the destination address mode is offset mode, maximum configurable number of blocks is 0xFFFF for block size(length) of one with data transfer size as byte,0x7FFF for block size of one with data transfer size as half word and 0x3FFF for block size of one with data size as word. After each block transfer the source address and the destination address will be incremented or decremented to the next block address. In case of offset address mode for source address, the source address size is the total size of source buffer after which the source area is rolled over, block size can be smaller than the source buffer size.For source address mode as offset mode, the maximum configurable source buffer size is 0xFFFF for transfer data size of a byte,0x7FFF for transfer data size of half word and 0x3FFF for transfer data size of word. Repeat-block mode can be used to implement single ring buffer to multiple ring buffer transfer type design.

Selecting the DTC or DMAC or EXDMAC

The Transfer API is implemented by DTC, DMAC and EXDMAC so that applications can switch between the DTC, DMAC and the EXDMAC. When selecting between them, consider these factors:

	DTC	DMAC	EXDMAC
Repeat Mode	• Repeats forever • Max repeat size is 256 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes
Block Mode	• Max block size is 256 x 4 bytes	• Max block size is 1024 x 4 bytes	• Max block size is 1024 x 4 bytes
Channels	• One instance per interrupt	• MCU specific (8 channels or less)	• MCU specific (2 channels)
Chained Transfers	• Supported	• Not Supported	• Not Supported
Software Trigger	• Must use the ICU Software Interrupt	• Has support for software trigger by selecting the software trigger activation source • Supports TRANSFER_START_MODE_SINGLE	• Has support for software trigger by selecting the software trigger activation source • Supports TRANSFER_START_MODE_SINGLE

		and TRANSFER_START_MODE_REPEAT	and TRANSFER_START_MODE_REPEAT
Offset Address Mode	• Not supported	• Supported	• Supported
Transfer Space	• 0000 0000h to FFFF FFFFh excluding reserved areas	• 0000 0000h to FFFF FFFFh excluding reserved areas	Exclusively for external bus transfer • SRAM0/SRAM1 (0200 0000h to 0217 FFFFh) • CS0–CS7 / SDRAM (7000 0000h to 7FFF FFFFh) • xSPI0/xSPI1 (2000 0000h to 5FFF FFFFh)

Interrupts

The DTC, DMAC and EXDMAC interrupts behave differently. The DTC uses the activating peripheral's IRQ as the interrupt source whereas each DMAC and EXDMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer	Interrupt after last transfer

Block Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block	Interrupt after last block

Repeat-block Mode

	DTC	DMAC	EXDMAC
TRANSFER_IRQ_EACH	N/A	N/A	N/A
TRANSFER_IRQ_END	N/A	Interrupt after last block	Interrupt after last block

Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC and EXDMAC store all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.

Offset Address Mode

When the source or destination mode is configured to offset mode, a configurable offset is added to the source or destination pointer after each transfer. The offset is a signed 24 bit number.

Examples

Basic Example

This is a basic example of minimal use of the EXDMAC in an application. In this case, software configurable interrupts or external pins interrupt have been routed to the EXDMAC for handling so it only needs to be enabled to start accepting transfers.

Since EXDMAC is an exclusive peripheral used for external bus transfers, when transferring from or to an external address, the external memory must be initialized properly before starting the transfer.

```
void exdmac_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_EXDMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the EXDMAC so that it responds to transfer requests. */
    err = R_EXDMAC_Enable(&g_transfer_ctrl);
    assert(FSP_SUCCESS == err);
}
```

SDRAM Transfer Example

In this example the EXDMAC is used to copy data between two regions in external SDRAM. Both the source and destination addresses are located in the SDRAM transfer space, and SDRAM must be initialized by the user before starting the transfer.

```
volatile bool g_transfer_complete = false;
void exdmac_callback (exdmac_callback_args_t * cb_data)
{
 FSP_PARAMETER_NOT_USED(cb_data);
    g_transfer_complete = true;
}
void exdmac_sdram_example (void)
{
 /* Allocate source and destination addresses in SDRAM manually.
  * These addresses must be aligned to the transfer size (4 bytes).
  */
#define EXDMAC_SDRAM_SRC_ADDR ((uint8_t *) 0x70000000)
#define EXDMAC_SDRAM_DEST_ADDR ((uint8_t *) 0x70001000)
    uint8_t * p_src = EXDMAC_SDRAM_SRC_ADDR;
    uint8_t * p_dest = EXDMAC_SDRAM_DEST_ADDR;
 /* Initialize source buffer in SDRAM with a known pattern [A..Z]. */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       p_src[i] = (uint8_t) ('A' + (i % 26));
       p_dest[i] = 0x00;
    }
 /* Set EXDMAC source and destination addresses. */
    g_transfer_cfg.p_info->p_src = (void *) p_src;
    g_transfer_cfg.p_info->p_dest = (void *) p_dest;
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_EXDMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable EXDMAC transfers. */
    g_transfer_complete = false;
    err                 = R_EXDMAC_Enable(&g_transfer_ctrl);
```

```
    assert(FSP_SUCCESS == err);
/* Trigger the transfer using software. */
    err = R_EXDMAC_SoftwareStart(&g_transfer_ctrl, TRANSFER_START_MODE_SINGLE);
    assert(FSP_SUCCESS == err);
while (!g_transfer_complete)
    {
/* Wait for transfer complete interrupt */
    }
/* Verify that the data was copied correctly in SDRAM. */
for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
if (p_src[i] != p_dest[i])
      {
/* Handle any transfer errors. This function should be defined by the user. */
            __BKPT(0);
      }
    }
}
```

Data Structures

struct	exdmac_instance_ctrl_t
struct	exdmac_extended_cfg_t

Macros

#define	EXDMAC_MAX_NORMAL_TRANSFER_LENGTH
#define	EXDMAC_MAX_REPEAT_TRANSFER_LENGTH
#define	EXDMAC_MAX_BLOCK_TRANSFER_LENGTH
#define	EXDMAC_MAX_REPEAT_COUNT
#define	EXDMAC_MAX_BLOCK_COUNT

Data Structure Documentation

◆ exdmac_instance_ctrl_t

struct exdmac_instance_ctrl_t

Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

◆ exdmac_extended_cfg_t

struct exdmac_extended_cfg_t

EXDMAC transfer configuration extension. This extension is required.

Data Fields

uint8_t	channel
	Channel number, does not apply to all HAL drivers.
IRQn_Type	irq
	EXDMAC interrupt number.
uint8_t	ipl
	EXDMAC interrupt priority.
int32_t	offset
	Offset value used with transfer_addr_mode_t::TRANSFER_ADDR_MODE_OFFSET.
uint16_t	src_buffer_size
IRQn_Type	irq_activation_source
void(*	p_callback)(exdmac_callback_args_t *cb_data)
void *	p_context

Field Documentation

◆ src_buffer_size

uint16_t exdmac_extended_cfg_t::src_buffer_size

Source ring buffer size for TRANSFER_MODE_REPEAT_BLOCK.

◆ irq_activation_source

IRQn_Type exdmac_extended_cfg_t::irq_activation_source

Select which interrupt source will trigger the transfer.

◆ p_callback

void(* exdmac_extended_cfg_t::p_callback) (exdmac_callback_args_t *cb_data)

Callback for transfer end interrupt.

◆ p_context

void* exdmac_extended_cfg_t::p_context

Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

Macro Definition Documentation

◆ EXDMAC_MAX_NORMAL_TRANSFER_LENGTH

#define EXDMAC_MAX_NORMAL_TRANSFER_LENGTH

Max configurable number of transfers in TRANSFER_MODE_NORMAL.

◆ EXDMAC_MAX_REPEAT_TRANSFER_LENGTH

#define EXDMAC_MAX_REPEAT_TRANSFER_LENGTH

Max number of transfers per repeat for TRANSFER_MODE_REPEAT.

◆ EXDMAC_MAX_BLOCK_TRANSFER_LENGTH

#define EXDMAC_MAX_BLOCK_TRANSFER_LENGTH

Max number of transfers per block in TRANSFER_MODE_BLOCK

◆ EXDMAC_MAX_REPEAT_COUNT

#define EXDMAC_MAX_REPEAT_COUNT

Max configurable number of repeats to transfer in TRANSFER_MODE_REPEAT

◆ EXDMAC_MAX_BLOCK_COUNT

#define EXDMAC_MAX_BLOCK_COUNT
Max configurable number of blocks to transfer in TRANSFER_MODE_BLOCK

Function Documentation

◆ R_EXDMAC_Open()

fsp_err_t R_EXDMAC_Open (transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg*)
Configure an EXDMAC channel.

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The configured channel is invalid.
FSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.

◆ R_EXDMAC_Reconfigure()

fsp_err_t R_EXDMAC_Reconfigure (transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info*)
Reconfigure the transfer with new transfer info.

Return values

FSP_SUCCESS	Transfer is configured and will start when trigger occurs.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_ENABLED	EXDMAC is not enabled. The current configuration must not be valid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.

◆ R_EXDMAC_Reset()

fsp_err_t R_EXDMAC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers.

Return values

FSP_SUCCESS	Transfer reset successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_ENABLED	EXDMAC is not enabled. The current configuration must not be valid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.

◆ R_EXDMAC_SoftwareStart()

fsp_err_t R_EXDMAC_SoftwareStart (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

If the mode is TRANSFER_START_MODE_SINGLE initiate a single transfer with software. If the mode is TRANSFER_START_MODE_REPEAT continue triggering transfers until all of the transfers are completed.

Return values

FSP_SUCCESS	Transfer started written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.
FSP_ERR_UNSUPPORTED	Handle was not configured for software activation.

◆ R_EXDMAC_SoftwareStop()

fsp_err_t R_EXDMAC_SoftwareStop (transfer_ctrl_t *const *p_api_ctrl*)

Stop software transfers if they were started with TRANSFER_START_MODE_REPEAT.

Return values

FSP_SUCCESS	Transfer stopped written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.

◆ R_EXDMAC_Enable()

fsp_err_t R_EXDMAC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers for the configured activation source.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.

◆ R_EXDMAC_Disable()

fsp_err_t R_EXDMAC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfers so that they are no longer triggered by the activation source.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.

◆ R_EXDMAC_InfoGet()

fsp_err_t R_EXDMAC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info*)

Set driver specific information in provided pointer.

Return values

FSP_SUCCESS	Information has been written to p_info.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_EXDMAC_Reload()

fsp_err_t R_EXDMAC_Reload (transfer_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *num_transfers*)

To update next transfer information without interruption during transfer.

Return values

FSP_ERR_UNSUPPORTED	This feature is not supported.

◆ R_EXDMAC_Close()

fsp_err_t R_EXDMAC_Close (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer and clean up internal data. Implements transfer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_EXDMAC_Open to initialize the control block.

◆ R_EXDMAC_CallbackSet()

fsp_err_t R_EXDMAC_CallbackSet (transfer_ctrl_t *const *p_api_ctrl*, void(*)(exdmac_callback_args_t *) *p_callback*, void *const *p_context*, exdmac_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

5.2.21 ProtectZone

Modules

ProtectZone Modules.

5.3 Interfaces

Detailed Description

The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer.

Modules

	Analog Analog Interfaces.
	AI AI Interfaces.
	Audio Audio Interfaces.

	CapTouch CapTouch Interfaces.
	Connectivity Connectivity Interfaces.
	DSP DSP Interfaces.
	Graphics Graphics Interfaces.
	Input Input Interfaces.
	Monitoring Monitoring Interfaces.
	Motor Motor Interfaces.
	Networking Networking Interfaces.
	Power Power Interfaces.
	Security Security Interfaces.
	Sensor Sensor Interfaces.
	Storage

	Storage Interfaces.
	System System Interfaces.
	Timers Timers Interfaces.
	Transfer Transfer Interfaces.

5.3.1 Analog
Interfaces

Detailed Description

Analog Interfaces.

Modules

	ADC Interface Interface for A/D Converters.
	DAC Interface Interface for D/A converters.

5.3.1.1 ADC Interface
Interfaces » Analog

Detailed Description

Interface for A/D Converters.

Summary

The ADC interface provides standard ADC functionality including one-shot mode (single scan), continuous scan and group scan. It also allows configuration of hardware and software triggers for starting scans. After each conversion an interrupt can be triggered, and if a callback function is provided, the call back is invoked with the appropriate event information.

Data Structures

struct	adc_status_t
struct	adc_callback_args_t
struct	adc_info_t
struct	adc_cfg_t
struct	adc_api_t
struct	adc_instance_t

Typedefs

typedef void	adc_ctrl_t

Enumerations

enum	adc_mode_t
enum	adc_resolution_t
enum	adc_alignment_t
enum	adc_trigger_t
enum	adc_event_t
enum	adc_channel_t
enum	adc_group_id_t
enum	adc_group_mask_t
enum	adc_state_t

Data Structure Documentation

◆ adc_status_t

struct adc_status_t
ADC status.
Data Fields

adc_state_t	state	Current state.

◆ adc_callback_args_t

struct adc_callback_args_t		
ADC callback arguments definitions		
Data Fields		
uint16_t	unit	ADC device in use.
adc_event_t	event	ADC callback event.
void *	p_context	Placeholder for user data.
adc_channel_t	channel	Channel of conversion result.
uint64_t	channel_mask	Channel mask for conversion result. Only valid for r_adc_b and r_sdadc_b.
adc_group_mask_t	group_mask	Group Mask.

◆ adc_info_t

struct adc_info_t		
ADC Information Structure for Transfer Interface		
Data Fields		
__I void *	p_address	The address to start reading the data from.
uint32_t	length	The total number of transfers to read.
transfer_size_t	transfer_size	The size of each transfer.
elc_peripheral_t	elc_peripheral	Name of the peripheral in the ELC list.
elc_event_t	elc_event	Name of the ELC event for the peripheral.
uint32_t	calibration_data	Temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage.
uint16_t	room_calibration_data	Room temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage.
uint16_t	low_calibration_data	Low temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage.
int16_t	slope_microvolts	Temperature sensor slope in

		microvolts/degrees C.

◆ adc_cfg_t

struct adc_cfg_t	
ADC general configuration	
Data Fields	
uint16_t	unit
	ADC unit to be used.
adc_mode_t	mode
	ADC operation mode.
adc_resolution_t	resolution
	ADC resolution.
adc_alignment_t	alignment
	Specify left or right alignment; ignored if addition used.
adc_trigger_t	trigger
	Default and Group A trigger source.
IRQn_Type	scan_end_irq
	Scan end IRQ number.
IRQn_Type	scan_end_b_irq
	Scan end group B IRQ number.
IRQn_Type	scan_end_c_irq
	Scan end group C IRQ number.

uint8_t	scan_end_ipl
	Scan end interrupt priority.
uint8_t	scan_end_b_ipl
	Scan end group B interrupt priority.
uint8_t	scan_end_c_ipl
	Scan end group C interrupt priority.
void(*	p_callback)(adc_callback_args_t *p_args)
	Callback function; set to NULL for none.
void *	p_context
	Placeholder for user data. Passed to the user callback in adc_callback_args_t.
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ adc_api_t

struct adc_api_t	
ADC functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t(*	scanCfg)(adc_ctrl_t *const p_ctrl, void const *const p_extend)
fsp_err_t(*	scanStart)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	scanGroupStart)(adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)

fsp_err_t(*	scanStop)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	scanStatusGet)(adc_ctrl_t *const p_ctrl, adc_status_t *p_status)
fsp_err_t(*	read)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)
fsp_err_t(*	read32)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)
fsp_err_t(*	calibrate)(adc_ctrl_t *const p_ctrl, void const *p_extend)
fsp_err_t(*	offsetSet)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)
fsp_err_t(*	callbackSet)(adc_ctrl_t *const p_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)

Field Documentation

◆ open

fsp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.

Precondition

Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_cfg	Pointer to configuration structure

◆ scanCfg

fsp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, void const *const p_extend)

Configure the scan including the channels, groups, and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_extend	See implementation for details

◆ scanStart

fsp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)

Start the scan (in case of a software trigger), or enable the hardware trigger.

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ scanGroupStart

fsp_err_t(* adc_api_t::scanGroupStart) (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)

Start the scan group (in case of a software trigger), or enable the hardware trigger.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	group_mask	Mask of groups to start

◆ scanStop

fsp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ scanStatusGet

fsp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl, adc_status_t *p_status)

Check scan status.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_status	Pointer to store current status in

◆ read

fsp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)

Read ADC conversion result.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	p_data	Pointer to variable to load value into.

◆ read32

fsp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)

Read ADC conversion result into a 32-bit word.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	p_data	Pointer to variable to load value into.

◆ calibrate

fsp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void const *p_extend)

Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_extend	Pointer to implementation specific arguments

◆ offsetSet

fsp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	offset	See implementation for details.

◆ callbackSet

fsp_err_t(* adc_api_t::callbackSet) (adc_ctrl_t *const p_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the ADC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)

Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ infoGet

fsp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)

Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_adc_info	Pointer to ADC information structure

◆ adc_instance_t

struct adc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
adc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
adc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
void const *	p_channel_cfg	Pointer to the channel configuration structure for this instance.
adc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ adc_ctrl_t

typedef void adc_ctrl_t
ADC control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ adc_mode_t

enum adc_mode_t	
ADC operation mode definitions	
Enumerator	
ADC_MODE_SINGLE_SCAN	Single scan - one or more channels.
ADC_MODE_GROUP_SCAN	Two trigger sources to trigger scan for two groups which contain one or more channels.
ADC_MODE_CONTINUOUS_SCAN	Continuous scan - one or more channels.

◆ adc_resolution_t

enum adc_resolution_t	
ADC data resolution definitions	
Enumerator	
ADC_RESOLUTION_12_BIT	12 bit resolution
ADC_RESOLUTION_10_BIT	10 bit resolution
ADC_RESOLUTION_8_BIT	8 bit resolution
ADC_RESOLUTION_14_BIT	14 bit resolution
ADC_RESOLUTION_16_BIT	16 bit resolution
ADC_RESOLUTION_24_BIT	24 bit resolution

◆ adc_alignment_t

enum adc_alignment_t	
ADC data alignment definitions	
Enumerator	
ADC_ALIGNMENT_RIGHT	Data alignment right.
ADC_ALIGNMENT_LEFT	Data alignment left.

◆ adc_trigger_t

enum adc_trigger_t	
ADC trigger mode definitions	
Enumerator	
ADC_TRIGGER_SOFTWARE	Software trigger; not for group modes.
ADC_TRIGGER_SYNC_ELC	Synchronous trigger via ELC.
ADC_TRIGGER_ASYNC_EXTERNAL	External asynchronous trigger; not for group modes.

◆ adc_event_t

enum adc_event_t	
ADC callback event definitions	
Enumerator	
ADC_EVENT_SCAN_COMPLETE	Normal/Group A scan complete.
ADC_EVENT_SCAN_COMPLETE_GROUP_B	Group B scan complete.
ADC_EVENT_SCAN_COMPLETE_GROUP_C	Group C scan complete.
ADC_EVENT_CALIBRATION_COMPLETE	Calibration complete.
ADC_EVENT_CONVERSION_COMPLETE	Conversion complete.
ADC_EVENT_CALIBRATION_REQUEST	Calibration requested.
ADC_EVENT_CONVERSION_ERROR	Scan error.
ADC_EVENT_OVERFLOW	Overflow occurred.
ADC_EVENT_LIMIT_CLIP	Limiter clipping occurred.
ADC_EVENT_FIFO_READ_REQUEST	FIFO read requested.
ADC_EVENT_FIFO_OVERFLOW	FIFO overflow occurred.
ADC_EVENT_WINDOW_COMPARE_A	Window A comparison condition met.
ADC_EVENT_WINDOW_COMPARE_B	Window B comparison condition met.
ADC_EVENT_ZERO_CROSS_DETECTION	Zero-cross detection interrupt.
ADC_EVENT_CAPTURE_A	Capture A conversion complete.
ADC_EVENT_CAPTURE_B	Capture B conversion complete.

◆ adc_channel_t

enum adc_channel_t	
ADC channels	
Enumerator	
ADC_CHANNEL_0	ADC channel 0.

ADC_CHANNEL_1	ADC channel 1.
ADC_CHANNEL_2	ADC channel 2.
ADC_CHANNEL_3	ADC channel 3.
ADC_CHANNEL_4	ADC channel 4.
ADC_CHANNEL_5	ADC channel 5.
ADC_CHANNEL_6	ADC channel 6.
ADC_CHANNEL_7	ADC channel 7.
ADC_CHANNEL_8	ADC channel 8.
ADC_CHANNEL_9	ADC channel 9.
ADC_CHANNEL_10	ADC channel 10.
ADC_CHANNEL_11	ADC channel 11.
ADC_CHANNEL_12	ADC channel 12.
ADC_CHANNEL_13	ADC channel 13.
ADC_CHANNEL_14	ADC channel 14.
ADC_CHANNEL_15	ADC channel 15.
ADC_CHANNEL_16	ADC channel 16.
ADC_CHANNEL_17	ADC channel 17.
ADC_CHANNEL_18	ADC channel 18.
ADC_CHANNEL_19	ADC channel 19.
ADC_CHANNEL_20	ADC channel 20.
ADC_CHANNEL_21	ADC channel 21.
ADC_CHANNEL_22	ADC channel 22.
ADC_CHANNEL_23	ADC channel 23.
ADC_CHANNEL_24	ADC channel 24.

ADC_CHANNEL_25	ADC channel 25.
ADC_CHANNEL_26	ADC channel 26.
ADC_CHANNEL_27	ADC channel 27.
ADC_CHANNEL_28	ADC channel 28.
ADC_CHANNEL_DUPLEX_A	Data duplexing register A.
ADC_CHANNEL_DUPLEX_B	Data duplexing register B.
ADC_CHANNEL_DUPLEX	Data duplexing register.
ADC_CHANNEL_TEMPERATURE	Temperature sensor output.
ADC_CHANNEL_VOLT	Internal reference voltage.
ADC_CHANNEL_0	ADC channel 0.
ADC_CHANNEL_1	ADC channel 1.
ADC_CHANNEL_2	ADC channel 2.
ADC_CHANNEL_3	ADC channel 3.
ADC_CHANNEL_4	ADC channel 4.
ADC_CHANNEL_5	ADC channel 5.
ADC_CHANNEL_6	ADC channel 6.
ADC_CHANNEL_7	ADC channel 7.
ADC_CHANNEL_8	ADC channel 8.
ADC_CHANNEL_9	ADC channel 9.
ADC_CHANNEL_10	ADC channel 10.
ADC_CHANNEL_11	ADC channel 11.
ADC_CHANNEL_12	ADC channel 12.
ADC_CHANNEL_13	ADC channel 13.
ADC_CHANNEL_14	ADC channel 14.

ADC_CHANNEL_15	ADC channel 15.
ADC_CHANNEL_16	ADC channel 16.
ADC_CHANNEL_17	ADC channel 17.
ADC_CHANNEL_18	ADC channel 18.
ADC_CHANNEL_19	ADC channel 19.
ADC_CHANNEL_20	ADC channel 20.
ADC_CHANNEL_21	ADC channel 21.
ADC_CHANNEL_22	ADC channel 22.
ADC_CHANNEL_23	ADC channel 23.
ADC_CHANNEL_24	ADC channel 24.
ADC_CHANNEL_25	ADC channel 25.
ADC_CHANNEL_26	ADC channel 26.
ADC_CHANNEL_27	ADC channel 27.
ADC_CHANNEL_28	ADC channel 28.
ADC_CHANNEL_29	ADC channel 29.
ADC_CHANNEL_30	ADC channel 30.
ADC_CHANNEL_31	ADC channel 31.
ADC_CHANNEL_32	ADC channel 32.
ADC_CHANNEL_33	ADC channel 33.
ADC_CHANNEL_34	ADC channel 34.
ADC_CHANNEL_35	ADC channel 35.
ADC_CHANNEL_SELF_DIAGNOSIS_ADC0	Self-Diagnosis channel for ADC Unit 0.
ADC_CHANNEL_SELF_DIAGNOSIS_ADC1	Self-Diagnosis channel for ADC Unit 1.
ADC_CHANNEL_SELF_DIAGNOSIS_ADC2	Self-Diagnosis channel for ADC Unit 1.

ADC_CHANNEL_TEMPERATURE	Temperature sensor output.
ADC_CHANNEL_VOLT	Internal reference voltage.
ADC_CHANNEL_VBATT	VBATT 1/3 voltage monitor.
ADC_CHANNEL_DA0	D/A converter channel 0.
ADC_CHANNEL_DA1	D/A converter channel 1.
ADC_CHANNEL_SELF_DIAGNOSIS_SH0	Self-Diagnosis Sample-and-hold Circuit Unit 0.
ADC_CHANNEL_SELF_DIAGNOSIS_SH1	Self-Diagnosis Sample-and-hold Circuit Unit 1.
ADC_CHANNEL_SELF_DIAGNOSIS_SH2	Self-Diagnosis Sample-and-hold Circuit Unit 2.
ADC_CHANNEL_FIRST_INTERNAL	Highest physical channel.
ADC_CHANNEL_LAST_INTERNAL	First internal channel.
ADC_CHANNEL_CAPTURE_A_MASK	Last internal channel. Capture A channel mask
ADC_CHANNEL_CAPTURE_B_MASK	Capture B channel mask.
ADC_CHANNEL_OVERCURRENT_MASK	Overcurrent channel mask.
ADC_CHANNEL_OVERCURRENT0_CAPTURE_MASK	Overcurrent0 capture channel mask.
ADC_CHANNEL_OVERCURRENT1_CAPTURE_MASK	Overcurrent1 capture channel mask.
ADC_CHANNEL_OVERCURRENT2_CAPTURE_MASK	Overcurrent2 capture channel mask.
ADC_CHANNEL_OVERCURRENT_SUM_CAPTURE_MASK	Overcurrent sum capture channel mask.
ADC_CHANNEL_0_CAPTURE_A	Channel 0 - Capture A channel.
ADC_CHANNEL_0_CAPTURE_B	Channel 0 - Capture B channel.
ADC_CHANNEL_0_OVERCURRENT	Channel 0 - Overcurrent channel.
ADC_CHANNEL_0_OVERCURRENT0_CAPTURE	Channel 0 - Overcurrent0 capture channel.
ADC_CHANNEL_0_OVERCURRENT1_CAPTURE	Channel 0 - Overcurrent1 capture channel.
ADC_CHANNEL_0_OVERCURRENT2_CAPTURE	Channel 0 - Overcurrent2 capture channel.

ADC_CHANNEL_0_OVERCURRENT_SUM_CAPTURE	Channel 0 - Overcurrent sum capture channel.
ADC_CHANNEL_1_CAPTURE_A	Channel 1 - Capture A channel.
ADC_CHANNEL_1_CAPTURE_B	Channel 1 - Capture B channel.
ADC_CHANNEL_1_OVERCURRENT	Channel 1 - Overcurrent channel.
ADC_CHANNEL_1_OVERCURRENT0_CAPTURE	Channel 1 - Overcurrent0 capture channel.
ADC_CHANNEL_1_OVERCURRENT1_CAPTURE	Channel 1 - Overcurrent1 capture channel.
ADC_CHANNEL_1_OVERCURRENT2_CAPTURE	Channel 1 - Overcurrent2 capture channel.
ADC_CHANNEL_1_OVERCURRENT_SUM_CAPTURE	Channel 1 - Overcurrent sum capture channel.
ADC_CHANNEL_2_CAPTURE_A	Channel 2 - Capture A channel.
ADC_CHANNEL_2_CAPTURE_B	Channel 2 - Capture B channel.
ADC_CHANNEL_2_OVERCURRENT	Channel 2 - Overcurrent channel.
ADC_CHANNEL_2_OVERCURRENT0_CAPTURE	Channel 2 - Overcurrent0 capture channel.
ADC_CHANNEL_2_OVERCURRENT1_CAPTURE	Channel 2 - Overcurrent1 capture channel.
ADC_CHANNEL_2_OVERCURRENT2_CAPTURE	Channel 2 - Overcurrent2 capture channel.
ADC_CHANNEL_2_OVERCURRENT_SUM_CAPTURE	Channel 2 - Overcurrent sum capture channel.
ADC_CHANNEL_0	ADC channel 0.
ADC_CHANNEL_1	ADC channel 1.
ADC_CHANNEL_2	ADC channel 2.
ADC_CHANNEL_3	ADC channel 3.
ADC_CHANNEL_4	ADC channel 4.
ADC_CHANNEL_5	ADC channel 5.
ADC_CHANNEL_6	ADC channel 6.
ADC_CHANNEL_7	ADC channel 7.
ADC_CHANNEL_8	ADC channel 8.

ADC_CHANNEL_9	ADC channel 9.
ADC_CHANNEL_10	ADC channel 10.
ADC_CHANNEL_11	ADC channel 11.
ADC_CHANNEL_12	ADC channel 12.
ADC_CHANNEL_13	ADC channel 13.
ADC_CHANNEL_14	ADC channel 14.
ADC_CHANNEL_15	ADC channel 15.
ADC_CHANNEL_16	ADC channel 16.
ADC_CHANNEL_17	ADC channel 17.
ADC_CHANNEL_18	ADC channel 18.
ADC_CHANNEL_19	ADC channel 19.
ADC_CHANNEL_20	ADC channel 20.
ADC_CHANNEL_21	ADC channel 21.
ADC_CHANNEL_22	ADC channel 22.
ADC_CHANNEL_23	ADC channel 23.
ADC_CHANNEL_24	ADC channel 24.
ADC_CHANNEL_25	ADC channel 25.
ADC_CHANNEL_26	ADC channel 26.
ADC_CHANNEL_27	ADC channel 27.
ADC_CHANNEL_28	ADC channel 28.
ADC_CHANNEL_29	ADC channel 29.
ADC_CHANNEL_30	ADC channel 30.
ADC_CHANNEL_31	ADC channel 31.
ADC_CHANNEL_32	ADC channel 32.

ADC_CHANNEL_33	ADC channel 33.
ADC_CHANNEL_34	ADC channel 34.
ADC_CHANNEL_35	ADC channel 35.
ADC_CHANNEL_SELF_DIAGNOSIS_ADC0	Self-Diagnosis channel for ADC Unit 0.
ADC_CHANNEL_SELF_DIAGNOSIS_ADC1	Self-Diagnosis channel for ADC Unit 1.
ADC_CHANNEL_SELF_DIAGNOSIS_ADC2	Self-Diagnosis channel for ADC Unit 1.
ADC_CHANNEL_TEMPERATURE	Temperature sensor output.
ADC_CHANNEL_VOLT	Internal reference voltage.
ADC_CHANNEL_VBATT	VBATT 1/3 voltage monitor.
ADC_CHANNEL_DA0	D/A converter channel 0.
ADC_CHANNEL_DA1	D/A converter channel 1.
ADC_CHANNEL_SELF_DIAGNOSIS_SH0	Self-Diagnosis Sample-and-hold Circuit Unit 0.
ADC_CHANNEL_SELF_DIAGNOSIS_SH1	Self-Diagnosis Sample-and-hold Circuit Unit 1.
ADC_CHANNEL_SELF_DIAGNOSIS_SH2	Self-Diagnosis Sample-and-hold Circuit Unit 2.
ADC_CHANNEL_FIRST_INTERNAL	Highest physical channel.
ADC_CHANNEL_LAST_INTERNAL	First internal channel.
ADC_CHANNEL_CAPTURE_A_MASK	Last internal channel. Capture A channel mask
ADC_CHANNEL_CAPTURE_B_MASK	Capture B channel mask.
ADC_CHANNEL_OVERCURRENT_MASK	Overcurrent channel mask.
ADC_CHANNEL_OVERCURRENT0_CAPTURE_MASK	Overcurrent0 capture channel mask.
ADC_CHANNEL_OVERCURRENT1_CAPTURE_MASK	Overcurrent1 capture channel mask.
ADC_CHANNEL_OVERCURRENT2_CAPTURE_MASK	Overcurrent2 capture channel mask.
ADC_CHANNEL_OVERCURRENT_SUM_CAPTURE_MASK	Overcurrent sum capture channel mask.

ADC_CHANNEL_0_CAPTURE_A	Channel 0 - Capture A channel.
ADC_CHANNEL_0_CAPTURE_B	Channel 0 - Capture B channel.
ADC_CHANNEL_0_OVERCURRENT	Channel 0 - Overcurrent channel.
ADC_CHANNEL_0_OVERCURRENT0_CAPTURE	Channel 0 - Overcurrent0 capture channel.
ADC_CHANNEL_0_OVERCURRENT1_CAPTURE	Channel 0 - Overcurrent1 capture channel.
ADC_CHANNEL_0_OVERCURRENT2_CAPTURE	Channel 0 - Overcurrent2 capture channel.
ADC_CHANNEL_0_OVERCURRENT_SUM_CAPTURE	Channel 0 - Overcurrent sum capture channel.
ADC_CHANNEL_1_CAPTURE_A	Channel 1 - Capture A channel.
ADC_CHANNEL_1_CAPTURE_B	Channel 1 - Capture B channel.
ADC_CHANNEL_1_OVERCURRENT	Channel 1 - Overcurrent channel.
ADC_CHANNEL_1_OVERCURRENT0_CAPTURE	Channel 1 - Overcurrent0 capture channel.
ADC_CHANNEL_1_OVERCURRENT1_CAPTURE	Channel 1 - Overcurrent1 capture channel.
ADC_CHANNEL_1_OVERCURRENT2_CAPTURE	Channel 1 - Overcurrent2 capture channel.
ADC_CHANNEL_1_OVERCURRENT_SUM_CAPTURE	Channel 1 - Overcurrent sum capture channel.
ADC_CHANNEL_2_CAPTURE_A	Channel 2 - Capture A channel.
ADC_CHANNEL_2_CAPTURE_B	Channel 2 - Capture B channel.
ADC_CHANNEL_2_OVERCURRENT	Channel 2 - Overcurrent channel.
ADC_CHANNEL_2_OVERCURRENT0_CAPTURE	Channel 2 - Overcurrent0 capture channel.
ADC_CHANNEL_2_OVERCURRENT1_CAPTURE	Channel 2 - Overcurrent1 capture channel.
ADC_CHANNEL_2_OVERCURRENT2_CAPTURE	Channel 2 - Overcurrent2 capture channel.
ADC_CHANNEL_2_OVERCURRENT_SUM_CAPTURE	Channel 2 - Overcurrent sum capture channel.

◆ adc_group_id_t

enum adc_group_id_t	
Enumerator	
ADC_GROUP_ID_0	Group ID 0.
ADC_GROUP_ID_1	Group ID 1.
ADC_GROUP_ID_2	Group ID 2.
ADC_GROUP_ID_3	Group ID 3.
ADC_GROUP_ID_4	Group ID 4.
ADC_GROUP_ID_5	Group ID 5.
ADC_GROUP_ID_6	Group ID 6.
ADC_GROUP_ID_7	Group ID 7.
ADC_GROUP_ID_8	Group ID 8.
ADC_GROUP_ID_0	Group ID 0.
ADC_GROUP_ID_1	Group ID 1.
ADC_GROUP_ID_2	Group ID 2.
ADC_GROUP_ID_3	Group ID 3.
ADC_GROUP_ID_4	Group ID 4.
ADC_GROUP_ID_5	Group ID 5.
ADC_GROUP_ID_6	Group ID 6.
ADC_GROUP_ID_7	Group ID 7.
ADC_GROUP_ID_8	Group ID 8.
ADC_GROUP_ID_9	Group ID 9.
ADC_GROUP_ID_10	Group ID 10.
ADC_GROUP_ID_11	Group ID 11.
ADC_GROUP_ID_0	

	Group ID 0.
ADC_GROUP_ID_1	Group ID 1.
ADC_GROUP_ID_2	Group ID 2.
ADC_GROUP_ID_3	Group ID 3.
ADC_GROUP_ID_4	Group ID 4.
ADC_GROUP_ID_5	Group ID 5.
ADC_GROUP_ID_6	Group ID 6.
ADC_GROUP_ID_7	Group ID 7.
ADC_GROUP_ID_8	Group ID 8.
ADC_GROUP_ID_9	Group ID 9.
ADC_GROUP_ID_10	Group ID 10.
ADC_GROUP_ID_11	Group ID 11.

◆ adc_group_mask_t

enum adc_group_mask_t	
Enumerator	
ADC_GROUP_MASK_NONE	Group Mask Unknown or None.
ADC_GROUP_MASK_0	Group Mask 0.
ADC_GROUP_MASK_1	Group Mask 1.
ADC_GROUP_MASK_2	Group Mask 2.
ADC_GROUP_MASK_3	Group Mask 3.
ADC_GROUP_MASK_4	Group Mask 4.
ADC_GROUP_MASK_5	Group Mask 5.
ADC_GROUP_MASK_6	Group Mask 6.
ADC_GROUP_MASK_7	Group Mask 7.
ADC_GROUP_MASK_8	Group Mask 8.
ADC_GROUP_MASK_ALL	All Groups.
ADC_GROUP_MASK_NONE	Group Mask Unknown or None.
ADC_GROUP_MASK_0	Group Mask 0.
ADC_GROUP_MASK_1	Group Mask 1.
ADC_GROUP_MASK_2	Group Mask 2.
ADC_GROUP_MASK_3	Group Mask 3.
ADC_GROUP_MASK_4	Group Mask 4.
ADC_GROUP_MASK_5	Group Mask 5.
ADC_GROUP_MASK_6	Group Mask 6.
ADC_GROUP_MASK_7	Group Mask 7.
ADC_GROUP_MASK_8	Group Mask 8.
ADC_GROUP_MASK_9	

	Group Mask 9.
ADC_GROUP_MASK_10	Group Mask 10.
ADC_GROUP_MASK_11	Group Mask 11.
ADC_GROUP_MASK_ALL	All Groups.
ADC_GROUP_MASK_NONE	Group Mask Unknown or None.
ADC_GROUP_MASK_0	Group Mask 0.
ADC_GROUP_MASK_1	Group Mask 1.
ADC_GROUP_MASK_2	Group Mask 2.
ADC_GROUP_MASK_3	Group Mask 3.
ADC_GROUP_MASK_4	Group Mask 4.
ADC_GROUP_MASK_5	Group Mask 5.
ADC_GROUP_MASK_6	Group Mask 6.
ADC_GROUP_MASK_7	Group Mask 7.
ADC_GROUP_MASK_8	Group Mask 8.
ADC_GROUP_MASK_9	Group Mask 9.
ADC_GROUP_MASK_10	Group Mask 10.
ADC_GROUP_MASK_11	Group Mask 11.
ADC_GROUP_MASK_ALL	All Groups.

◆ adc_state_t

enum adc_state_t	
ADC states.	
Enumerator	
ADC_STATE_IDLE	ADC is idle.
ADC_STATE_SCAN_IN_PROGRESS	ADC scan in progress.
ADC_STATE_CALIBRATION_IN_PROGRESS	ADC calibration in progress - Not used by all ADC instances.

5.3.1.2 DAC Interface

Interfaces » Analog

Detailed Description

Interface for D/A converters.

Summary

The DAC interface provides standard Digital/Analog Converter functionality. A DAC application writes digital sample data to the device and generates analog output on the DAC output pin.

Data Structures

struct	dac_info_t
struct	dac_cfg_t
struct	dac_api_t
struct	dac_instance_t

Typedefs

typedef void	dac_ctrl_t

Enumerations

enum	dac_data_format_t

Data Structure Documentation

◆ dac_info_t

struct dac_info_t		
DAC information structure to store various information for a DAC		
Data Fields		
uint8_t	bit_width	Resolution of the DAC.

◆ dac_cfg_t

struct dac_cfg_t		
DAC Open API configuration parameter		
Data Fields		
uint8_t	channel	ID associated with this DAC channel.
bool	ad_da_synchronized	AD/DA synchronization.
void const *	p_extend	

◆ dac_api_t

struct dac_api_t	
DAC driver structure. General DAC functions implemented at the HAL layer follow this API.	
Data Fields	
fsp_err_t(*	open)(dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)
fsp_err_t(*	close)(dac_ctrl_t *const p_ctrl)
fsp_err_t(*	write)(dac_ctrl_t *const p_ctrl, uint16_t value)
fsp_err_t(*	start)(dac_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(dac_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* dac_api_t::open) (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

fsp_err_t(* dac_api_t::close) (dac_ctrl_t *const p_ctrl)

Close the D/A Converter.

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ write

fsp_err_t(* dac_api_t::write) (dac_ctrl_t *const p_ctrl, uint16_t value)

Write sample value to the D/A Converter.

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.
[in]	value	Sample value to be written to the D/A Converter.

◆ start

fsp_err_t(* dac_api_t::start) (dac_ctrl_t *const p_ctrl)

Start the D/A Converter if it has not been started yet.

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ stop

fsp_err_t(* dac_api_t::stop) (dac_ctrl_t *const p_ctrl)

Stop the D/A Converter if the converter is running.

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ dac_instance_t

struct dac_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
dac_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
dac_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
dac_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ dac_ctrl_t

typedef void dac_ctrl_t
DAC control block. Allocate an instance specific control block to pass into the DAC API calls.

Enumeration Type Documentation

◆ dac_data_format_t

enum dac_data_format_t	
DAC Open API data format settings.	
Enumerator	
DAC_DATA_FORMAT_FLUSH_RIGHT	LSB of data is flush to the right leaving the top 4 bits unused.
DAC_DATA_FORMAT_FLUSH_LEFT	MSB of data is flush to the left leaving the bottom 4 bits unused.

5.3.2 AI
Interfaces

AI Interfaces.

5.3.3 Audio
Interfaces

Detailed Description

Audio Interfaces.

Modules

	PDM Interface Interface for PDM audio communication.

5.3.3.1 PDM Interface
Interfaces » Audio

Detailed Description

Interface for PDM audio communication.

Summary

The pulse-density modulation(PDM) interface provides APIs and definitions for PDM audio communication.

Data Structures

struct	pdm_callback_args_t
struct	pdm_sound_detection_setting_t
struct	pdm_status_t
struct	pdm_cfg_t
struct	pdm_api_t
struct	pdm_instance_t

Typedefs

typedef void	pdm_ctrl_t

Enumerations

enum	pdm_pcm_width_t
enum	pdm_input_data_edge_t
enum	pdm_event_t
enum	pdm_error_t
enum	pdm_state_t

Data Structure Documentation

◆ pdm_callback_args_t

struct pdm_callback_args_t		
Callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data. Set in pdm_api_t::open function in pdm_cfg_t.
pdm_event_t	event	The event can be used to identify what caused the callback (overflow or error).
pdm_error_t	error	The kind of error.

◆ pdm_sound_detection_setting_t

struct pdm_sound_detection_setting_t		
Sound detection window setting		
Data Fields		
uint32_t	sound_detection_lower_limit	Sound detection lower limit (20-bit signed fixed point data)
uint32_t	sound_detection_upper_limit	Sound detection upper limit (20-bit signed fixed point data)

◆ pdm_status_t

struct pdm_status_t		
PDM status.		
Data Fields		
pdm_state_t	state	Current PDM data state.
bool	sound_detection_enabled	Current PDM sound detection status.
pdm_error_t	error	Bitfield of current PDM errors.

◆ pdm_cfg_t

struct pdm_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint32_t	unit
pdm_pcm_width_t	pcm_width
	Audio PCM data width.
pdm_input_data_edge_t	pcm_edge
	Input data select.
transfer_instance_t const *	p_transfer_rx
void(*	p_callback)(pdm_callback_args_t *p_args)
void *	p_context

void const *	p_extend
	Extension parameter for hardware specific settings.
uint8_t	dat_ipl
	Data reception interrupt priority.
uint8_t	sdet_ipl
	Sound detection interrupt priority.
uint8_t	err_ipl
	Idle/Error interrupt priority.
IRQn_Type	dat_irq
	IRQ number of data reception interrupt.
IRQn_Type	sdet_irq
	IRQ number of sound detection interrupt.
IRQn_Type	err_irq
	IRQ number of error detection interrupt.

Field Documentation

◆ unit

uint32_t pdm_cfg_t::unit

Select a unit/channel corresponding to the unit/channel number of the hardware.

◆ p_transfer_rx

transfer_instance_t const* pdm_cfg_t::p_transfer_rx

To use DMA for receiving link a Transfer instance here. Set to NULL if unused.

◆ p_callback

void(* pdm_cfg_t::p_callback) (pdm_callback_args_t *p_args)

Callback provided when an PDM ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void* pdm_cfg_t::p_context

Placeholder for user data. Passed to the user callback in pdm_callback_args_t.

◆ pdm_api_t

struct pdm_api_t

PDM functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(pdm_ctrl_t *const p_ctrl, pdm_cfg_t const *const p_cfg)
fsp_err_t(*	start)(pdm_ctrl_t *const p_ctrl, void *const p_buffer, size_t const buffer_size, uint32_t const number_of_data_to_callback)
fsp_err_t(*	stop)(pdm_ctrl_t *const p_ctrl)
fsp_err_t(*	soundDetectionEnable)(pdm_ctrl_t *const p_ctrl, pdm_sound_detection_setting_t pdm_sound_detection_setting)
fsp_err_t(*	soundDetectionDisable)(pdm_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(pdm_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)
fsp_err_t(*	statusGet)(pdm_ctrl_t *const p_ctrl, pdm_status_t *const p_status)
fsp_err_t(*	close)(pdm_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(pdm_ctrl_t *const p_ctrl, void(*p_callback)(pdm_callback_args_t *), void *const p_context,

	pdm_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* pdm_api_t::open) (pdm_ctrl_t *const p_ctrl, pdm_cfg_t const *const p_cfg)

Initial configuration.

Precondition

Peripheral clocks and any required output pins should be configured prior to calling this function.

Note

To reconfigure after calling this function, call pdm_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ start

fsp_err_t(* pdm_api_t::start) (pdm_ctrl_t *const p_ctrl, void *const p_buffer, size_t const buffer_size, uint32_t const number_of_data_to_callback)

Start communication.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.
[in]	p_buffer	Pointer to the reception buffer
[in]	buffer_size	Size of the buffer
[in]	number_of_data_to_callback	Number of the data to call the callback function from dat_isr

◆ stop

fsp_err_t(* pdm_api_t::stop) (pdm_ctrl_t *const p_ctrl)

Stop communication. Communication is stopped when callback is called with PDM_EVENT_IDLE.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.

◆ soundDetectionEnable

fsp_err_t(* pdm_api_t::soundDetectionEnable) (pdm_ctrl_t *const p_ctrl, pdm_sound_detection_setting_t pdm_sound_detection_setting)

Enable sound detection with the specified settings.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.
[in]	pdm_sound_detection_setting	Settings to configure sound detection.

◆ soundDetectionDisable

fsp_err_t(* pdm_api_t::soundDetectionDisable) (pdm_ctrl_t *const p_ctrl)

Disable sound detection.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.

◆ read

fsp_err_t(* pdm_api_t::read) (pdm_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)

Read remaining PDM data. This function can be called during PDM_STATE_STOPPED.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes.

◆ statusGet

fsp_err_t(* pdm_api_t::statusGet) (pdm_ctrl_t *const p_ctrl, pdm_status_t *const p_status)

Get current status and store it in provided pointer p_status.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.
[out]	p_status	Current status of the driver.

◆ close

fsp_err_t(* pdm_api_t::close) (pdm_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in pdm_api_t::open call for this instance.

◆ callbackSet

fsp_err_t(* pdm_api_t::callbackSet) (pdm_ctrl_t *const p_ctrl, void(*p_callback)(pdm_callback_args_t *), void *const p_context, pdm_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the PDM control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_callback_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ pdm_instance_t

struct pdm_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
pdm_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
pdm_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
pdm_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ pdm_ctrl_t

typedef void pdm_ctrl_t

PDM control block. Allocate an instance specific control block to pass into the PDM API calls.

Enumeration Type Documentation

◆ pdm_pcm_width_t

enum pdm_pcm_width_t	
Audio PCM width	
Enumerator	
PDM_PCM_WIDTH_20_BITS_0_18	Using 20-bit PCM Clips [18:0] bit of the buffer and 1-bit of sign.
PDM_PCM_WIDTH_20_BITS_1_18	Using 20-bit PCM Clips [18:1] bit of the buffer and 2-bits of sign.
PDM_PCM_WIDTH_20_BITS_2_18	Using 20-bit PCM Clips [18:2] bit of the buffer and 3-bits of sign.
PDM_PCM_WIDTH_20_BITS_3_18	Using 20-bit PCM Clips [18:3] bit of the buffer and 4-bits of sign.
PDM_PCM_WIDTH_16_BITS_4_18	Using 16-bit PCM Clips [18:4] bit of the buffer and 1-bit of sign.
PDM_PCM_WIDTH_16_BITS_3_17	Using 16-bit PCM Clips [17:3] bit of the buffer and 1-bit of sign.
PDM_PCM_WIDTH_16_BITS_2_16	Using 16-bit PCM Clips [16:2] bit of the buffer and 1-bit of sign.
PDM_PCM_WIDTH_16_BITS_1_15	Using 16-bit PCM Clips [15:1] bit of the buffer and 1-bit of sign.
PDM_PCM_WIDTH_16_BITS_0_14	Using 16-bit PCM Clips [14:0] bit of the buffer and 1-bit of sign.

◆ pdm_input_data_edge_t

enum pdm_input_data_edge_t	
Input data select.	
Enumerator	
PDM_INPUT_DATA_EDGE_RISE	Rise-edge data of channel n.
PDM_INPUT_DATA_EDGE_FALL	Fall-edge data of channel n-1 (In case of n = 0, fall-edge data of channel 2 is selected.)

◆ pdm_event_t

enum pdm_event_t	
Events that can trigger a callback function	
Enumerator	
PDM_EVENT_IDLE	IDLE.
PDM_EVENT_DATA	Data reception.
PDM_EVENT_SOUND_DETECTION	Sound detection.
PDM_EVENT_ERROR	Error.

◆ pdm_error_t

enum pdm_error_t	
Error information included in a callback function	
Enumerator	
PDM_ERROR_NONE	No error.
PDM_ERROR_SHORT_CIRCUIT	Short circuit detection.
PDM_ERROR_OVERVOLTAGE_LOWER	Overvoltage lower limit exceeded detection.
PDM_ERROR_OVERVOLTAGE_UPPER	Overvoltage upper limit exceeded detection.
PDM_ERROR_BUFFER_OVERWRITE	Buffer overwriting detection.

◆ pdm_state_t

enum pdm_state_t	
Possible status values returned by pdm_api_t::statusGet.	
Enumerator	
PDM_STATE_IN_USE	PDM is actively reading data.
PDM_STATE_STOPPED	PDM is stopped and not actively reading data.

5.3.4 CapTouch
Interfaces

CapTouch Interfaces.

5.3.5 Connectivity
Interfaces

Detailed Description

Connectivity Interfaces.

Modules

	CAN Interface Interface for CAN peripheral.
	I2C Master Interface Interface for I2C master communication.
	I2C Slave Interface Interface for I2C slave communication.
	I2S Interface Interface for I2S audio communication.
	I3C Interface Interface for I3C.
	PCIF Interface Interface for PCIF functions.
	SPI Interface Interface for SPI communications.

	UART Interface Interface for UART communications.
	USB Interface Interface for USB functions.
	USB HCDC Interface Interface for USB HCDC functions.
	USB HHID Interface Interface for USB HHID functions.
	USB HMSC Interface Interface for USB HMSC functions.
	USB PCDC Interface Interface for USB PCDC functions.
	USB PHID Interface Interface for USB PHID functions.
	USB PMSC Interface Interface for USB PMSC functions.
	USB PPRN Interface Interface for USB PPRN functions.
	Communications Middleware Interface Interface for Communications Middleware functions.

5.3.5.1 CAN Interface

Interfaces » Connectivity

Detailed Description

Interface for CAN peripheral.

Summary

The CAN interface provides common APIs for CAN HAL drivers. CAN interface supports following features.

- Full-duplex CAN communication
- Generic CAN parameter setting
- Interrupt driven transmit/receive processing
- Callback function support with returning event code
- Hardware resource locking during a transaction

Data Structures

struct	can_info_t
struct	can_bit_timing_cfg_t
struct	can_frame_t
struct	can_callback_args_t
struct	can_cfg_t
struct	can_api_t
struct	can_instance_t

Typedefs

typedef void	can_ctrl_t

Enumerations

enum	can_event_t
enum	can_operation_mode_t
enum	can_test_mode_t
enum	can_id_mode_t
enum	can_frame_type_t

Data Structure Documentation

◆ can_info_t

struct can_info_t		
CAN status info		
Data Fields		
uint32_t	status	Useful information from the CAN status register.
uint32_t	rx_mb_status	RX Message Buffer New Data flags.
uint32_t	rx_fifo_status	RX FIFO Empty flags.
uint8_t	error_count_transmit	Transmit error count.
uint8_t	error_count_receive	Receive error count.
uint32_t	error_code	Error code, cleared after reading.

◆ can_bit_timing_cfg_t

struct can_bit_timing_cfg_t		
CAN bit rate configuration.		
Data Fields		
uint32_t	baud_rate_prescaler	Baud rate prescaler. Valid values: 1 - 1024.
uint32_t	time_segment_1	Time segment 1 control.
uint32_t	time_segment_2	Time segment 2 control.
uint32_t	synchronization_jump_width	Synchronization jump width.

◆ can_frame_t

struct can_frame_t		
CAN data Frame		
Data Fields		
uint32_t	id	CAN ID.
can_id_mode_t	id_mode	Standard or Extended ID (IDE).
can_frame_type_t	type	Frame type (RTR).
uint8_t	data_length_code	CAN Data Length Code (DLC).
uint32_t	options	Implementation-specific options.
uint8_t	data[CAN_DATA_BUFFER_LENGTH]	CAN data.

◆ can_callback_args_t

struct can_callback_args_t		
CAN callback parameter definition		
Data Fields		
uint32_t	channel	Device channel number.
can_event_t	event	Event code.
uint32_t	error	Error code.
union can_callback_args_t	mb_u	
void *	p_context	Context provided to user during callback.
can_frame_t	frame	Received frame data.

◆ can_cfg_t

struct can_cfg_t	
CAN Configuration	
Data Fields	
uint32_t	channel
	CAN channel.
can_bit_timing_cfg_t *	p_bit_timing
	CAN bit timing.
void(*	p_callback)(can_callback_args_t *p_args)
	Pointer to callback function.
void *	p_context
	User defined callback context.
void const *	p_extend
	CAN hardware dependent configuration.
uint8_t	ipl
	Error/Transmit/Receive interrupt priority.

IRQn_Type	error_irq
	Error IRQ number.
IRQn_Type	rx_irq
	Receive IRQ number.
IRQn_Type	tx_irq
	Transmit IRQ number.

◆ can_api_t

struct can_api_t	
Shared Interface definition for CAN	
Data Fields	
fsp_err_t(*	open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
fsp_err_t(*	write)(can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)
fsp_err_t(*	read)(can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)
fsp_err_t(*	close)(can_ctrl_t *const p_ctrl)
fsp_err_t(*	modeTransition)(can_ctrl_t *const p_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)
fsp_err_t(*	infoGet)(can_ctrl_t *const p_ctrl, can_info_t *const p_info)
fsp_err_t(*	callbackSet)(can_ctrl_t *const p_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)

Open function for CAN device

Parameters

[in,out]	p_ctrl	Pointer to the CAN control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to CAN configuration structure. All elements of this structure must be set by user.

◆ write

fsp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)

Write function for CAN device

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	buffer	Buffer number (mailbox or message buffer) to write to.
[in]	p_frame	Pointer for frame of CAN ID, DLC, data and frame type to write.

◆ read

fsp_err_t(* can_api_t::read) (can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)

Read function for CAN device

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	buffer	Message buffer (number) to read from.
[in]	p_frame	Pointer to store the CAN ID, DLC, data and frame type.

◆ close

fsp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)

Close function for CAN device

Parameters

[in]	p_ctrl	Pointer to the CAN control block.

◆ modeTransition

fsp_err_t(* can_api_t::modeTransition) (can_ctrl_t *const p_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)

Mode Transition function for CAN device

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	operation_mode	Destination CAN operation state.
[in]	test_mode	Destination CAN test state.

◆ infoGet

fsp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const p_info)

Get CAN channel info.

Parameters

[in]	p_ctrl	Handle for channel (pointer to channel control block)
[out]	p_info	Memory address to return channel specific data to.

◆ callbackSet

fsp_err_t(* can_api_t::callbackSet) (can_ctrl_t *const p_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in can_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ can_instance_t

struct can_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
can_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
can_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
can_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ can_ctrl_t

typedef void can_ctrl_t

CAN control block. Allocate an instance specific control block to pass into the CAN API calls.

Enumeration Type Documentation

◆ can_event_t

enum can_event_t	
CAN event codes	
Enumerator	
CAN_EVENT_ERR_WARNING	Error Warning event.
CAN_EVENT_ERR_PASSIVE	Error Passive event.
CAN_EVENT_ERR_BUS_OFF	Bus Off event.
CAN_EVENT_BUS_RECOVERY	Bus Off Recovery event.
CAN_EVENT_MAILBOX_MESSAGE_LOST	Mailbox has been overrun.
CAN_EVENT_ERR_BUS_LOCK	Bus lock detected (32 consecutive dominant bits).
CAN_EVENT_ERR_CHANNEL	Channel error has occurred.
CAN_EVENT_TX_ABORTED	Transmit abort event.
CAN_EVENT_RX_COMPLETE	Receive complete event.
CAN_EVENT_TX_COMPLETE	Transmit complete event.
CAN_EVENT_ERR_GLOBAL	Global error has occurred.
CAN_EVENT_TX_FIFO_EMPTY	Transmit FIFO is empty.
CAN_EVENT_FIFO_MESSAGE_LOST	Receive FIFO overrun.

◆ can_operation_mode_t

enum can_operation_mode_t	
CAN Operation modes	
Enumerator	
CAN_OPERATION_MODE_NORMAL	CAN Normal Operation Mode.
CAN_OPERATION_MODE_RESET	CAN Reset Operation Mode.
CAN_OPERATION_MODE_HALT	CAN Halt Operation Mode.
CAN_OPERATION_MODE_SLEEP	CAN Sleep Operation Mode.

◆ can_test_mode_t

enum can_test_mode_t	
CAN Test modes	
Enumerator	
CAN_TEST_MODE_DISABLED	CAN Test Mode Disabled.
CAN_TEST_MODE_LISTEN	CAN Test Listen Mode.
CAN_TEST_MODE_LOOPBACK_EXTERNAL	CAN Test External Loopback Mode.
CAN_TEST_MODE_LOOPBACK_INTERNAL	CAN Test Internal Loopback Mode.
CAN_TEST_MODE_INTERNAL_BUS	CANFD Internal CAN Bus Communication Test Mode.

◆ can_id_mode_t

enum can_id_mode_t	
CAN ID modes	
Enumerator	
CAN_ID_MODE_STANDARD	Standard IDs of 11 bits used.
CAN_ID_MODE_EXTENDED	Extended IDs of 29 bits used.

◆ can_frame_type_t

enum can_frame_type_t	
CAN frame types	
Enumerator	
CAN_FRAME_TYPE_DATA	Data frame.
CAN_FRAME_TYPE_REMOTE	Remote frame.

5.3.5.2 I2C Master Interface

Interfaces » Connectivity

Detailed Description

Interface for I2C master communication.

Summary

The I2C master interface provides a common API for I2C HAL drivers. The I2C master interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which can return an event code

Data Structures

struct	i2c_master_callback_args_t
struct	i2c_master_status_t
struct	i2c_master_cfg_t
struct	i2c_master_api_t
struct	i2c_master_instance_t

Typedefs

typedef void	i2c_master_ctrl_t

Enumerations

enum	i2c_master_rate_t

enum	i2c_master_addr_mode_t
enum	i2c_master_event_t

Data Structure Documentation

◆ i2c_master_callback_args_t

struct i2c_master_callback_args_t		
I2C callback parameter definition		
Data Fields		
void *	p_context	Pointer to user-provided context.
i2c_master_event_t	event	Event code.

◆ i2c_master_status_t

struct i2c_master_status_t		
I2C status indicators		
Data Fields		
bool	open	True if driver is open.

◆ i2c_master_cfg_t

struct i2c_master_cfg_t	
I2C configuration block	
Data Fields	
uint8_t	channel
	Identifier recognizable by implementation. More...
i2c_master_rate_t	rate
	Device's maximum clock rate from enum i2c_rate_t.
uint32_t	slave
	The address of the slave device.
i2c_master_addr_mode_t	addr_mode
	Indicates how slave fields should be interpreted.

uint8_t	ipl
	Interrupt priority level. Same for RXI, TXI, TEI and ERI.
IRQn_Type	rxi_irq
	Receive IRQ number.
IRQn_Type	txi_irq
	Transmit IRQ number.
IRQn_Type	tei_irq
	Transmit end IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
transfer_instance_t const *	p_transfer_tx
	Transfer instance for I2C transmit. Set to NULL if unused. More...
transfer_instance_t const *	p_transfer_rx
	Transfer instance for I2C receive. Set to NULL if unused.
void(*	p_callback)(i2c_master_callback_args_t *p_args)
	Pointer to callback function. More...
void *	p_context
	Pointer to the user-provided context.

void const *	p_extend
	Any configuration data needed by the hardware. More...

Field Documentation

◆ channel

uint8_t i2c_master_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

◆ p_transfer_tx

transfer_instance_t const* i2c_master_cfg_t::p_transfer_tx

Transfer instance for I2C transmit. Set to NULL if unused.

Transfer API support

◆ p_callback

void(* i2c_master_cfg_t::p_callback) (i2c_master_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

◆ p_extend

void const* i2c_master_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

◆ i2c_master_api_t

struct i2c_master_api_t

Interface definition for I2C access as master

Data Fields

fsp_err_t(*	open)(i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t(*	read)(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t(*	write)(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

fsp_err_t(*	abort)(i2c_master_ctrl_t *const p_ctrl)
fsp_err_t(*	slaveAddressSet)(i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t(*	callbackSet)(i2c_master_ctrl_t *const p_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t(*	statusGet)(i2c_master_ctrl_t *const p_ctrl, i2c_master_status_t *p_status)
fsp_err_t(*	close)(i2c_master_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* i2c_master_api_t::open) (i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

Opens the I2C Master driver and initializes the hardware.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

fsp_err_t(* i2c_master_api_t::read) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

Performs a read operation on an I2C Master device.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_master_api_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.
[in]	restart	Specify if the restart condition should be issued after reading.

◆ write

fsp_err_t(* i2c_master_api_t::write) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

Performs a write operation on an I2C Master device.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_master_api_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.
[in]	restart	Specify if the restart condition should be issued after writing.

◆ abort

fsp_err_t(* i2c_master_api_t::abort) (i2c_master_ctrl_t *const p_ctrl)

Performs a reset of the peripheral.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_master_api_t::open call.

◆ slaveAddressSet

fsp_err_t(* i2c_master_api_t::slaveAddressSet) (i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)

Sets address of the slave device without reconfiguring the bus.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_master_api_t::open call.
[in]	slave_address	Address of the slave device.
[in]	address_mode	Addressing mode.

◆ callbackSet

fsp_err_t(* i2c_master_api_t::callbackSet) (i2c_master_ctrl_t *const p_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the IIC Master control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ statusGet

fsp_err_t(* i2c_master_api_t::statusGet) (i2c_master_ctrl_t *const p_ctrl, i2c_master_status_t *p_status)

Gets the status of the configured I2C device.

Parameters

[in]	p_ctrl	Pointer to the IIC Master control block.
[out]	p_status	Pointer to store current status.

◆ close

fsp_err_t(* i2c_master_api_t::close) (i2c_master_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Master device.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_master_api_t::open call.

◆ i2c_master_instance_t

struct i2c_master_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
i2c_master_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2c_master_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2c_master_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ i2c_master_ctrl_t

typedef void i2c_master_ctrl_t
I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

Enumeration Type Documentation

◆ i2c_master_rate_t

enum i2c_master_rate_t	
Communication speed options	
Enumerator	
I2C_MASTER_RATE_STANDARD	100 kHz
I2C_MASTER_RATE_FAST	400 kHz
I2C_MASTER_RATE_FASTPLUS	1 MHz
I2C_MASTER_RATE_HIGHSPEED	3.4 MHz

◆ i2c_master_addr_mode_t

enum i2c_master_addr_mode_t	
Addressing mode options	
Enumerator	
I2C_MASTER_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_MASTER_ADDR_MODE_10BIT	Use 10-bit addressing mode.

◆ i2c_master_event_t

enum i2c_master_event_t	
Callback events	
Enumerator	
I2C_MASTER_EVENT_ABORTED	A transfer was aborted.
I2C_MASTER_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_MASTER_EVENT_TX_COMPLETE	A transmit operation was completed successfully.
I2C_MASTER_EVENT_START	I2C sent a start condition.
I2C_MASTER_EVENT_BYTE_ACK	I2C finished sending/receiving 1 data byte.

5.3.5.3 I2C Slave Interface

Interfaces » Connectivity

Detailed Description

Interface for I2C slave communication.

Summary

The I2C slave interface provides a common API for I2C HAL drivers. The I2C slave interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which returns a event codes

Data Structures

struct	i2c_slave_callback_args_t
struct	i2c_slave_cfg_t
struct	i2c_slave_api_t
struct	i2c_slave_instance_t

Typedefs

typedef void	i2c_slave_ctrl_t

Enumerations

enum	i2c_slave_rate_t
enum	i2c_slave_addr_mode_t
enum	i2c_slave_event_t

Data Structure Documentation

◆ i2c_slave_callback_args_t

struct i2c_slave_callback_args_t		
I2C callback parameter definition		
Data Fields		
void *	p_context	Pointer to user-provided context.
uint32_t	bytes	Number of received/transmitted bytes in buffer.

i2c_slave_event_t	event	Event code.

◆ i2c_slave_cfg_t

struct i2c_slave_cfg_t	
I2C configuration block	
Data Fields	
uint8_t	channel
	Identifier recognizable by implementation. More...
i2c_slave_rate_t	rate
	Device's maximum clock rate from enum i2c_rate_t.
uint16_t	slave
	The address of the slave device.
i2c_slave_addr_mode_t	addr_mode
	Indicates how slave fields should be interpreted.
bool	general_call_enable
	Allow a General call from master.
IRQn_Type	rxi_irq
	Receive IRQ number.
IRQn_Type	txi_irq
	Transmit IRQ number.
IRQn_Type	tei_irq
	Transmit end IRQ number.

IRQn_Type	eri_irq
	Error IRQ number.
uint8_t	ipl
	Interrupt priority level for receive, transmit, and transmit end interrupts.
uint8_t	eri_ipl
	Interrupt priority level for error interrupt.
bool	clock_stretching_enable
	Low Hold SCL during reception for the period between the 9th and the 1st clock cycle.
transfer_instance_t const *	p_transfer_tx
	Transfer instance for I2C transmit. Set to NULL if unused. More...
transfer_instance_t const *	p_transfer_rx
	Transfer instance for I2C receive. Set to NULL if unused.
void(*	p_callback)(i2c_slave_callback_args_t *p_args)
	Pointer to callback function. More...
void *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Any configuration data needed by the hardware. More...

Field Documentation

◆ channel

uint8_t i2c_slave_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

◆ p_transfer_tx

transfer_instance_t const* i2c_slave_cfg_t::p_transfer_tx

Transfer instance for I2C transmit. Set to NULL if unused.

Transfer API support

◆ p_callback

void(* i2c_slave_cfg_t::p_callback) (i2c_slave_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

◆ p_extend

void const* i2c_slave_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

◆ i2c_slave_api_t

struct i2c_slave_api_t

Interface definition for I2C access as slave

Data Fields

fsp_err_t(*	open)(i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)
fsp_err_t(*	read)(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t(*	write)(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)
fsp_err_t(*	callbackSet)(i2c_slave_ctrl_t *const p_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context,

	i2c_slave_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(i2c_slave_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* i2c_slave_api_t::open) (i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)

Opens the I2C Slave driver and initializes the hardware.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

fsp_err_t(* i2c_slave_api_t::read) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Performs a read operation on an I2C Slave device.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.

◆ write

fsp_err_t(* i2c_slave_api_t::write) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Performs a write operation on an I2C Slave device.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.

◆ callbackSet

fsp_err_t(* i2c_slave_api_t::callbackSet) (i2c_slave_ctrl_t *const p_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, i2c_slave_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the IIC Slave control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* i2c_slave_api_t::close) (i2c_slave_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Slave device.

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.

◆ i2c_slave_instance_t

struct i2c_slave_instance_t

This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
i2c_slave_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2c_slave_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2c_slave_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ i2c_slave_ctrl_t

typedef void i2c_slave_ctrl_t
I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

Enumeration Type Documentation

◆ i2c_slave_rate_t

enum i2c_slave_rate_t	
Communication speed options	
Enumerator	
I2C_SLAVE_RATE_STANDARD	100 kHz
I2C_SLAVE_RATE_FAST	400 kHz
I2C_SLAVE_RATE_FASTPLUS	1 MHz
I2C_SLAVE_RATE_HIGHSPEED	3.4 MHz

◆ i2c_slave_addr_mode_t

enum i2c_slave_addr_mode_t	
Addressing mode options	
Enumerator	
I2C_SLAVE_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_SLAVE_ADDR_MODE_10BIT	Use 10-bit addressing mode.

◆ i2c_slave_event_t

enum i2c_slave_event_t	
Callback events	
Enumerator	
I2C_SLAVE_EVENT_ABORTED	A transfer was aborted.
I2C_SLAVE_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_SLAVE_EVENT_TX_COMPLETE	A transmit operation was completed successfully.
I2C_SLAVE_EVENT_RX_REQUEST	A read operation expected from slave. Detected a write from master.
I2C_SLAVE_EVENT_TX_REQUEST	A write operation expected from slave. Detected a read from master.
I2C_SLAVE_EVENT_RX_MORE_REQUEST	A read operation expected from slave. Master sends out more data than configured to be read in slave.
I2C_SLAVE_EVENT_TX_MORE_REQUEST	A write operation expected from slave. Master requests more data than configured to be written by slave.
I2C_SLAVE_EVENT_GENERAL_CALL	General Call address received from Master. Detected a write from master.

5.3.5.4 I2S Interface

Interfaces » Connectivity

Detailed Description

Interface for I2S audio communication.

Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

Data Structures

struct	i2s_callback_args_t
struct	i2s_status_t
struct	i2s_cfg_t
struct	i2s_api_t
struct	i2s_instance_t

Typedefs

typedef void	i2s_ctrl_t

Enumerations

enum	i2s_pcm_width_t
enum	i2s_word_length_t
enum	i2s_event_t
enum	i2s_mode_t
enum	i2s_mute_t
enum	i2s_ws_continue_t
enum	i2s_state_t

Data Structure Documentation

◆ i2s_callback_args_t

struct i2s_callback_args_t		
Callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data. Set in i2s_api_t::open function in i2s_cfg_t.
i2s_event_t	event	The event can be used to identify what caused the callback (overflow or error).

◆ i2s_status_t

struct i2s_status_t
I2S status.

Data Fields		
i2s_state_t	state	Current I2S state.

◆ i2s_cfg_t

struct i2s_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint32_t	channel
i2s_pcm_width_t	pcm_width
	Audio PCM data width.
i2s_word_length_t	word_length
	Audio word length, bits must be >= i2s_cfg_t::pcm_width bits.
i2s_ws_continue_t	ws_continue
	Whether to continue WS transmission during idle state.
i2s_mode_t	operating_mode
	Master or slave mode.
transfer_instance_t const *	p_transfer_tx
transfer_instance_t const *	p_transfer_rx
void(*	p_callback)(i2s_callback_args_t *p_args)
void *	p_context
void const *	p_extend
	Extension parameter for hardware specific settings.

uint8_t	rxi_ipl
	Receive interrupt priority.
uint8_t	txi_ipl
	Transmit interrupt priority.
uint8_t	idle_err_ipl
	Idle/Error interrupt priority.
IRQn_Type	txi_irq
	Transmit IRQ number.
IRQn_Type	rxi_irq
	Receive IRQ number.
IRQn_Type	int_irq
	Idle/Error IRQ number.

Field Documentation

◆ channel

uint32_t i2s_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ p_transfer_tx

transfer_instance_t const* i2s_cfg_t::p_transfer_tx

To use DMA for transmitting link a Transfer instance here. Set to NULL if unused.

◆ p_transfer_rx

transfer_instance_t const* i2s_cfg_t::p_transfer_rx

To use DMA for receiving link a Transfer instance here. Set to NULL if unused.

◆ p_callback

void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)

Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void* i2s_cfg_t::p_context

Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

◆ i2s_api_t

struct i2s_api_t

I2S functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
fsp_err_t(*	stop)(i2s_ctrl_t *const p_ctrl)
fsp_err_t(*	mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
fsp_err_t(*	write)(i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)
fsp_err_t(*	read)(i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)
fsp_err_t(*	writeRead)(i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)
fsp_err_t(*	statusGet)(i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)
fsp_err_t(*	close)(i2s_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(i2s_ctrl_t *const p_ctrl, void(*p_callback)(i2s_callback_args_t *), void *const p_context,

	i2s_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)

Initial configuration.

Precondition

Peripheral clocks and any required output pins should be configured prior to calling this function.

Note

To reconfigure after calling this function, call i2s_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ stop

fsp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl)

Stop communication. Communication is stopped when callback is called with I2S_EVENT_IDLE.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ mute

fsp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

Enable or disable mute.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	mute_enable	Whether to enable or disable mute.

◆ write

fsp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)

Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8.

◆ read

fsp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes.

◆ writeRead

fsp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffers. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8, and receive will stop at the multiple of 8 below requested bytes.

◆ statusGet

fsp_err_t(* i2s_api_t::statusGet) (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)

Get current status and store it in provided pointer p_status.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[out]	p_status	Current status of the driver.

◆ close

fsp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ callbackSet

fsp_err_t(* i2s_api_t::callbackSet) (i2s_ctrl_t *const p_ctrl, void(*p_callback)(i2s_callback_args_t *), void *const p_context, i2s_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the I2S control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ i2s_instance_t

struct i2s_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
i2s_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2s_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2s_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ i2s_ctrl_t

typedef void i2s_ctrl_t

I2S control block. Allocate an instance specific control block to pass into the I2S API calls.

Enumeration Type Documentation

◆ i2s_pcm_width_t

enum i2s_pcm_width_t	
Audio PCM width	
Enumerator	
I2S_PCM_WIDTH_8_BITS	Using 8-bit PCM.
I2S_PCM_WIDTH_16_BITS	Using 16-bit PCM.
I2S_PCM_WIDTH_18_BITS	Using 18-bit PCM.
I2S_PCM_WIDTH_20_BITS	Using 20-bit PCM.
I2S_PCM_WIDTH_22_BITS	Using 22-bit PCM.
I2S_PCM_WIDTH_24_BITS	Using 24-bit PCM.
I2S_PCM_WIDTH_32_BITS	Using 32-bit PCM.

◆ i2s_word_length_t

enum i2s_word_length_t	
Audio system word length.	
Enumerator	
I2S_WORD_LENGTH_8_BITS	Using 8-bit system word length.
I2S_WORD_LENGTH_16_BITS	Using 16-bit system word length.
I2S_WORD_LENGTH_24_BITS	Using 24-bit system word length.
I2S_WORD_LENGTH_32_BITS	Using 32-bit system word length.
I2S_WORD_LENGTH_48_BITS	Using 48-bit system word length.
I2S_WORD_LENGTH_64_BITS	Using 64-bit system word length.
I2S_WORD_LENGTH_128_BITS	Using 128-bit system word length.
I2S_WORD_LENGTH_256_BITS	Using 256-bit system word length.

◆ i2s_event_t

enum i2s_event_t	
Events that can trigger a callback function	
Enumerator	
I2S_EVENT_IDLE	Communication is idle.
I2S_EVENT_TX_EMPTY	Transmit buffer is below FIFO trigger level.
I2S_EVENT_RX_FULL	Receive buffer is above FIFO trigger level.

◆ i2s_mode_t

enum i2s_mode_t	
I2S communication mode	
Enumerator	
I2S_MODE_SLAVE	Slave mode.
I2S_MODE_MASTER	Master mode.

◆ i2s_mute_t

enum i2s_mute_t	
Mute audio samples.	
Enumerator	
I2S_MUTE_OFF	Disable mute.
I2S_MUTE_ON	Enable mute.

◆ i2s_ws_continue_t

enum i2s_ws_continue_t	
Whether to continue WS (word select line) transmission during idle state.	
Enumerator	
I2S_WS_CONTINUE_ON	Enable WS continue mode.
I2S_WS_CONTINUE_OFF	Disable WS continue mode.

◆ i2s_state_t

enum i2s_state_t	
Possible status values returned by i2s_api_t::statusGet.	
Enumerator	
I2S_STATE_IN_USE	I2S is in use.
I2S_STATE_STOPPED	I2S is stopped.

5.3.5.5 I3C Interface

Interfaces » Connectivity

Detailed Description

Interface for I3C.

Summary

The I3C interface provides APIs and definitions for I3C communication.

Data Structures

struct	i3c_device_status_t
struct	i3c_slave_info_t
struct	i3c_device_table_cfg_t
struct	i3c_device_cfg_t
struct	i3c_command_descriptor_t
struct	i3c_callback_args_t
struct	i3c_cfg_t
struct	i3c_api_t
struct	i3c_instance_t

Typedefs

typedef void	i3c_ctrl_t

Enumerations

enum	i3c_common_command_code_t
enum	i3c_event_t
enum	i3c_device_type_t
enum	i3c_device_protocol_t
enum	i3c_address_assignment_mode_t
enum	i3c_ibi_type_t

Data Structure Documentation

◆ i3c_device_status_t

struct i3c_device_status_t		
The current status of the slave device (See GETSTATUS in the MIPI I3C Specification v1.0).		
Data Fields		
uint8_t	pending_interrupt	Contains the interrupt number of any pending interrupt, or 0 if no interrupts are pending.
uint8_t	vendor_status	Reserved for vendor-specific meaning.

◆ i3c_slave_info_t

struct i3c_slave_info_t		
Device characteristics that define the I3C capabilities of a slave.		
Data Fields		
uint8_t	pid[6]	Provisional ID.
union i3c_slave_info_t	bcr_u	
uint8_t	dcr	Device Characteristics Register.
union i3c_slave_info_t	bcr_u	

◆ i3c_device_table_cfg_t

struct i3c_device_table_cfg_t		
Structure for configuring an entry in the device table when the driver is in master mode (See i3c_api_t::masterDeviceTableSet).		
Data Fields		
uint8_t	static_address	I3C Static address / I2C address for this device.

uint8_t	dynamic_address	Dynamic address for the device. This address will be assigned during Dynamic Address Assignment.
i3c_device_protocol_t	device_protocol	The protocol used to communicate with this device (I3C / I2C Legacy).
bool	ibi_accept	Accept or reject IBI requests from this device.
bool	master_request_accept	Accept mastership requests from this device.
bool	ibi_payload	IBI requests from this device have a data payload. Note: When the device is configured using ENTDAA, the ibi_payload will automatically be updated based on the value of BCR.

◆ i3c_device_cfg_t

struct i3c_device_cfg_t		
Structure for configuring a slave address when the driver is in slave mode (See i3c_api_t::deviceCfgSet).		
Data Fields		
uint8_t	static_address	I3C Static address / I2C address for this device.
uint8_t	dynamic_address	Dynamic address for this device. Note that the dynamic address will automatically be updated when ENTDAA is completed.
i3c_slave_info_t	slave_info	PID, BCR, and DCR registers for the device (Slave mode only).

◆ i3c_command_descriptor_t

struct i3c_command_descriptor_t		
Descriptor for completing CCC/HDR transfers.		
Data Fields		
uint8_t	command_code	Common Command Code / HDR Command Code for the transfer.
uint8_t *	p_buffer	Buffer for reading or writing data.

uint32_t	length	Length of the data portion of the command.
bool	restart	If true, issue a repeated-start after the transfer is completed.
bool	rnw	Set to true if the command type is Direct Get.

◆ i3c_callback_args_t

struct i3c_callback_args_t		
Arguments that are passed to the user callback when an event occurs.		
Data Fields		
i3c_event_t	event	The type of event that has occurred.
uint32_t	event_status	Status flags associated with the event.
uint32_t	transfer_size	Number of bytes transferred.
i3c_slave_info_t const *	p_slave_info	A pointer to the Characteristics Registers read during ENTDAA.
uint8_t	dynamic_address	The dynamic address that was assigned to the slave during ENTDAA.
i3c_ibi_type_t	ibi_type	The type of IBI that has been received.
uint8_t	ibi_address	The address of the device that sent the IBI.
uint8_t	command_code	The command code of the received command.
void *	p_context	User defined context.

◆ i3c_cfg_t

struct i3c_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint32_t	channel
i3c_device_type_t	device_type
transfer_instance_t const *	p_transfer_tx
	Transfer instance for I3C transmit. Set to NULL if unused. More...

transfer_instance_t const *	p_transfer_rx
	Transfer instance for I3C receive. Set to NULL if unused.
void(*	p_callback)(i3c_callback_args_t const *const p_args)
void *	p_context
	Pointer to the user-provided context.
void const *	p_extend

Field Documentation

◆ channel

uint32_t i3c_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ device_type

i3c_device_type_t i3c_cfg_t::device_type

The type of device.

◆ p_transfer_tx

transfer_instance_t const* i3c_cfg_t::p_transfer_tx

Transfer instance for I3C transmit. Set to NULL if unused.

Transfer API support.

◆ p_callback

void(* i3c_cfg_t::p_callback) (i3c_callback_args_t const *const p_args)

Pointer to the user callback.

◆ p_extend

void const* i3c_cfg_t::p_extend

Pointer to extended configuration.

◆ i3c_api_t

struct i3c_api_t

I3C functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(i3c_ctrl_t *const p_ctrl, i3c_cfg_t const *const p_cfg)
fsp_err_t(*	enable)(i3c_ctrl_t *const p_ctrl)
fsp_err_t(*	deviceCfgSet)(i3c_ctrl_t *const p_ctrl, i3c_device_cfg_t const *const p_device_cfg)
fsp_err_t(*	masterDeviceTableSet)(i3c_ctrl_t *const p_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)
fsp_err_t(*	deviceSelect)(i3c_ctrl_t *const p_ctrl, uint32_t device_index, uint32_t bitrate_mode)
fsp_err_t(*	dynamicAddressAssignmentStart)(i3c_ctrl_t *const p_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)
fsp_err_t(*	slaveStatusSet)(i3c_ctrl_t *const p_ctrl, i3c_device_status_t device_status)
fsp_err_t(*	commandSend)(i3c_ctrl_t *const p_ctrl, i3c_command_descriptor_t const *const p_command_descriptor)
fsp_err_t(*	write)(i3c_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)
fsp_err_t(*	read)(i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length, bool restart)
fsp_err_t(*	ibiWrite)(i3c_ctrl_t *const p_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)
fsp_err_t(*	ibiRead)(i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length)

fsp_err_t(*	close)(i3c_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* i3c_api_t::open) (i3c_ctrl_t *const p_ctrl, i3c_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ enable

fsp_err_t(* i3c_api_t::enable) (i3c_ctrl_t *const p_ctrl)

Enable the I3C device.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.

◆ deviceCfgSet

fsp_err_t(* i3c_api_t::deviceCfgSet) (i3c_ctrl_t *const p_ctrl, i3c_device_cfg_t const *const p_device_cfg)

Set the configuration of this device.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_device_cfg	Pointer to device configuration.

◆ masterDeviceTableSet

fsp_err_t(* i3c_api_t::masterDeviceTableSet) (i3c_ctrl_t *const p_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)

Set the configuration for the device at the given index in the device table. The configuration will be used by transfers when it is selected by deviceSelect.

Note: This function is not used in slave mode.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	device_index	Index into the device table.
[in]	p_device_table_cfg	Pointer to the table settings for the entry in the master device table.

◆ deviceSelect

fsp_err_t(* i3c_api_t::deviceSelect) (i3c_ctrl_t *const p_ctrl, uint32_t device_index, uint32_t bitrate_mode)

In master mode, select the device for the next transfer.

Note: This function is not used in slave mode.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	device_index	Index into the device table.
[in]	bitrate_setting	The bitrate settings for the selected device.

◆ dynamicAddressAssignmentStart

fsp_err_t(* i3c_api_t::dynamicAddressAssignmentStart) (i3c_ctrl_t *const p_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)

Start Dynamic Address Assignment by sending either the ENTDAA or SETDASA command See i3c_address_assignment_mode_t for more information.

Note: This function is not used in slave mode.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	address_assignment_mode	The command to use for Dynamic Address Assignment.
[in]	starting_device_index	The device index that will be used to assign the first device during Dynamic Address Assignment.
[in]	device_count	The number of devices to assign (Only used with I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA).

◆ slaveStatusSet

fsp_err_t(* i3c_api_t::slaveStatusSet) (i3c_ctrl_t *const p_ctrl, i3c_device_status_t device_status)

Set the status returned to the master in response to a GETSTATUS command.

Note: This function is not used in master mode.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	device_status	New status settings for responding to the GETSTATUS command code.

◆ commandSend

fsp_err_t(* i3c_api_t::commandSend) (i3c_ctrl_t *const p_ctrl, i3c_command_descriptor_t const *const p_command_descriptor)

Send a read/write/broadcast CCC or HDR command.

Note: This function is not used in slave mode.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_command_descriptor	A descriptor for executing the command.

◆ write

fsp_err_t(* i3c_api_t::write) (i3c_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)

In master mode: Start a write transfer. When the transfer is completed send a stop condition or a repeated-start. In slave mode: Set the write buffer and configure the number of bytes that will be transferred before the the transfer is ended by the slave via the 'T' bit or by the master issuing a stop condition.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to write.
[in]	length	Number of bytes to transfer.
[in]	restart	If true, issue a repeated-start after the transfer is completed (Master only).

◆ read

fsp_err_t(* i3c_api_t::read) (i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length, bool restart)

In master mode: Start a read transfer. When the transfer is completed, send a stop condition or a repeated-start. In slave mode: Set the read buffer for storing data read during the transfer. When the buffer is full, the application will receive a callback requesting a new read buffer. If no buffer is provided by the application, the driver will discard any remaining bytes read during the transfer.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to store the bytes read during the transfer.
[in]	length	Number of bytes to transfer.
[in]	restart	If true, issue a repeated-start after the transfer is completed (Master only).

◆ ibiWrite

fsp_err_t(* i3c_api_t::ibiWrite) (i3c_ctrl_t *const p_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)

Initiate an IBI write operation.

Note: This function is not used in master mode.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	ibi_type	The type of In-Band Interrupt.
[in]	p_data	Pointer to a buffer to start the bytes read during the transfer.
[in]	length	Number of bytes to transfer.

◆ ibiRead

fsp_err_t(* i3c_api_t::ibiRead) (i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length)

Set the read buffer for storing received IBI data (This function is not used in slave mode).

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to store the bytes read during the transfer.
[in]	length	Number of bytes to transfer.

◆ close

fsp_err_t(* i3c_api_t::close) (i3c_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.

◆ i3c_instance_t

struct i3c_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
i3c_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i3c_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i3c_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ i3c_ctrl_t

typedef void i3c_ctrl_t
I3C control block. Allocate an instance specific control block to pass into the I3C API calls.

Enumeration Type Documentation

◆ i3c_common_command_code_t

enum i3c_common_command_code_t	
Common Command Codes defined by MIPI I3C Specification v1.1.	
Enumerator	
I3C_CCC_BROADCAST_ENEC	Enable Slave initiated events.
I3C_CCC_BROADCAST_DISEC	Disable Slave initiated events.
I3C_CCC_BROADCAST_ENTAS0	Enter Activity State 0.
I3C_CCC_BROADCAST_ENTAS1	Enter Activity State 1.
I3C_CCC_BROADCAST_ENTAS2	Enter Activity State 2.
I3C_CCC_BROADCAST_ENTAS3	Enter Activity State 3.
I3C_CCC_BROADCAST_RSTDAA	Reset Dynamic Address Assignment.
I3C_CCC_BROADCAST_ENTDAA	Enter Dynamic Address Assignment.
I3C_CCC_BROADCAST_DEFSVLS	Define List of Slaves.
I3C_CCC_BROADCAST_SETMWL	Set Max Write Length.
I3C_CCC_BROADCAST_SETMRL	Set Max Read Length.
I3C_CCC_BROADCAST_ENTTM	Enter Test Mode.
I3C_CCC_BROADCAST_SETBUSCON	Set BUS Context.
I3C_CCC_BROADCAST_ENDXFER	Data Transfer Ending Procedure Control.
I3C_CCC_BROADCAST_ENTHDR0	Enter HDR Mode 0.
I3C_CCC_BROADCAST_ENTHDR1	Enter HDR Mode 1.
I3C_CCC_BROADCAST_ENTHDR2	Enter HDR Mode 2.
I3C_CCC_BROADCAST_ENTHDR3	Enter HDR Mode 3.
I3C_CCC_BROADCAST_ENTHDR4	Enter HDR Mode 4 (Reserved for future definition).
I3C_CCC_BROADCAST_ENTHDR5	Enter HDR Mode 5 (Reserved for future definition).

I3C_CCC_BROADCAST_ENTHDR6	Enter HDR Mode 6 (Reserved for future definition).
I3C_CCC_BROADCAST_ENTHDR7	Enter HDR Mode 7 (Reserved for future definition).
I3C_CCC_BROADCAST_SETXTIME	Set Exchange Timing Info.
I3C_CCC_BROADCAST_SETAASA	Set All Addresses to Static Address.
I3C_CCC_BROADCAST_RSTACT	Slave Reset Action.
I3C_CCC_BROADCAST_DEFGRPA	Define List of Group Address.
I3C_CCC_BROADCAST_RSTGRPA	Reset Group Address.
I3C_CCC_BROADCAST_MLANE	Multi-Lane Data Transfer Control.
I3C_CCC_DIRECT_ENEC	Enable Slave initiated events.
I3C_CCC_DIRECT_DISEC	Disable Slave initiated events.
I3C_CCC_DIRECT_ENTAS0	Enter Activity State 0.
I3C_CCC_DIRECT_ENTAS1	Enter Activity State 1.
I3C_CCC_DIRECT_ENTAS2	Enter Activity State 2.
I3C_CCC_DIRECT_ENTAS3	Enter Activity State 3.
I3C_CCC_DIRECT_RSTDAA	Reset Dynamic Address Assignment (DEPRECATED v1.0).
I3C_CCC_DIRECT_SETDASA	Set Dynamic Address from Static Address.
I3C_CCC_DIRECT_SETNEWDA	Set New Dynamic Address.
I3C_CCC_DIRECT_SETMWL	Set Max Write Length.
I3C_CCC_DIRECT_SETMRL	Set Max Read Length.
I3C_CCC_DIRECT_GETMWL	Get Max Write Length.
I3C_CCC_DIRECT_GETMRL	Get Max Read Length.
I3C_CCC_DIRECT_GETPID	Get Provisional ID.
I3C_CCC_DIRECT_GETBCR	Get Bus Characteristic Register.

I3C_CCC_DIRECT_GETDCR	Get Device Characteristic Register.
I3C_CCC_DIRECT_GETSTATUS	Get Device Status.
I3C_CCC_DIRECT_GETACCMST	Get Accept Mastership.
I3C_CCC_DIRECT_ENDXFER	Data Transfer Ending Procedure Control.
I3C_CCC_DIRECT_SETBRGTGT	Set Bridge Targets.
I3C_CCC_DIRECT_GETMXDS	Get Max Data Speed.
I3C_CCC_DIRECT_GETHDRCAP	Get HDR Capability.
I3C_CCC_DIRECT_SETROUTE	Set Route.
I3C_CCC_DIRECT_D2DXFER	Device to Device(s) Tunneling Control.
I3C_CCC_DIRECT_SETXTIME	Set Exchange Timing Information.
I3C_CCC_DIRECT_GETXTIME	Get Exchange Timing Information.
I3C_CCC_DIRECT_RSTACT	Reset Slave Action.
I3C_CCC_DIRECT_SETGRPA	Set Group Address.
I3C_CCC_DIRECT_RSTGRPA	Reset Group Address.
I3C_CCC_DIRECT_MLANE	Multi-Lane Data Transfer Control.

◆ i3c_event_t

enum i3c_event_t	
I3C Events that result in a callback.	
Enumerator	
I3C_EVENT_ENTDAA_ADDRESS_PHASE	Events that only occur in Master mode. A Slave device has finished writing its PID, BCR, and DCR. This information is provided in i3c_callback_args_t::p_slave_info.
I3C_EVENT_IBI_READ_COMPLETE	An IBI has successfully been read.
I3C_EVENT_IBI_READ_BUFFER_FULL	There is no more space in the IBI read buffer. The application may provide another buffer by calling i3c_api_t::ibiRead.
I3C_EVENT_READ_BUFFER_FULL	Events that only occur in Slave mode. There is no more space in the read buffer. The application may provide another buffer by calling i3c_api_t::read.
I3C_EVENT_IBI_WRITE_COMPLETE	A IBI was written successfully.
I3C_EVENT_HDR_EXIT_PATTERN_DETECTED	The HDR exit pattern was detected on the bus.
I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE	Dynamic Address Assignment has completed. Events that are common to Master and Slave mode.
I3C_EVENT_COMMAND_COMPLETE	A command was completed.
I3C_EVENT_WRITE_COMPLETE	A write transfer has completed.
I3C_EVENT_READ_COMPLETE	A read transfer has completed.
I3C_EVENT_TIMEOUT_DETECTED	SCL is stuck at the logic high or logic low level during a transfer.
I3C_EVENT_INTERNAL_ERROR	An internal error occurred.
I3C_EVENT_SDA_WRITE_COMPLETE	An SDA (Short Data Argument) write transfer has completed.

◆ i3c_device_type_t

enum i3c_device_type_t	
The type of device.	
Enumerator	
I3C_DEVICE_TYPE_MAIN_MASTER	The main master starts in master mode and is responsible for configuring the bus.
I3C_DEVICE_TYPE_SLAVE	A slave device listens to the bus for relevant I3C Commands (CCCs) sent by the current master, and responds accordingly. Slave devices may also initiate In-band interrupts and Hot-Join requests.

◆ i3c_device_protocol_t

enum i3c_device_protocol_t	
Identifies the protocol for transferring data with the device on the bus.	
Enumerator	
I3C_DEVICE_PROTOCOL_I2C	Transfers will use legacy I2C protocol with open-drain output at a reduced baudrate.
I3C_DEVICE_PROTOCOL_I3C	Transfers will use I3C SDR mode.

◆ i3c_address_assignment_mode_t

enum i3c_address_assignment_mode_t	
Address Assignment Mode.	
Enumerator	
I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA	Send the ENTDAA command to enter Dynamic Address Assignment mode and assign dynamic addresses in order, starting with the starting device index. The procedure is completed after the specified number of devices have been configured. The callback will be called after the PID, DCR, and BCR registers have been read for each device.

◆ i3c_ibi_type_t

enum i3c_ibi_type_t	
The type of In-Band Interrupt.	
Enumerator	
I3C_IBI_TYPE_INTERRUPT	Application specific In-Band Interrupt for notifying the master when an event occurs.
I3C_IBI_TYPE_HOT_JOIN	Request the master to perform the Dynamic Address Assignment process.
I3C_IBI_TYPE_MASTERSHIP_REQUEST	Request the master to give up control of the bus.

5.3.5.6 PCIF Interface

Interfaces » Connectivity

Detailed Description

Interface for PCIF functions.

Summary

Phase Counting Interface(PCIF) interface provides APIs and definitions for motor communication.

Data Structures

struct	pcif_callback_args_t
struct	pcif_phc_counter_info_t
struct	pcif_status_t
struct	pcif_cfg_t
struct	pcif_api_t
struct	pcif_instance_t

Typedefs

typedef void	pcif_ctrl_t

Enumerations

enum	pcif_error_t
enum	pcif_state_t
enum	pcif_mode_t
enum	pcif_clock_source_t
enum	pcif_clock_div_t
enum	pcif_phase_polarity_t
enum	pcif_stop_mode_t
enum	pcif_rotation_direction_t

Data Structure Documentation

◆ pcif_callback_args_t

struct pcif_callback_args_t		
Callback parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in pcif_api_t::open function in pcif_cfg_t.
pcif_error_t	error	

◆ pcif_phc_counter_info_t

struct pcif_phc_counter_info_t		
Phase Counting counter data configuration structure		
Data Fields		
uint16_t	new_position	New position of counter to be set.
int16_t	edge_count	Number of edge count.

◆ pcif_status_t

struct pcif_status_t		
Current PCIF status.		
Data Fields		
pcif_state_t	state	Current PCIF state (stopped or running)

pcif_rotation_direction_t	rotation_direction	Rotation direction (clockwise or counter-clockwise)
uint16_t	current_position_value	Current output counter value (PHCPOSCNTM or INCOCNTM)
uint16_t	position_write_value	Latest PHCPCWDR write value (0 in incremental mode)
uint8_t	input_counter_value	Incremental input counter value (INCICNTM, 0 in phase-counting mode)

◆ pcif_cfg_t

struct pcif_cfg_t	
User configuration structure, used in open function	
Data Fields	
pcif_mode_t	mode
	Select enumerated value from pcif_mode_t.
uint8_t	channel
pcif_clock_source_t	clock_src
	Source clock selection.
pcif_clock_div_t	clock_div
	Source clock divider.
pcif_phase_polarity_t	a_phase_polarity
	A phase polarity output. More...
pcif_phase_polarity_t	b_phase_polarity
	B phase polarity output.
pcif_phase_polarity_t	z_phase_polarity
	Z phase polarity output.

uint16_t	max_count_position
	Maximum position value.
timer_instance_t const *	p_lower_lvl_timer
	Pointer to the timer instance structs.
void(*	p_callback)(pcif_callback_args_t *p_args)
void *	p_context
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ channel

uint8_t pcif_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ a_phase_polarity

pcif_phase_polarity_t pcif_cfg_t::a_phase_polarity

A phase polarity output.

Common setting

◆ p_callback

void(* pcif_cfg_t::p_callback) (pcif_callback_args_t *p_args)

Callback provided when an PCIF carrier period occurs. Set to NULL if not use.

◆ p_context

void* pcif_cfg_t::p_context

Placeholder for user data. Passed to the user callback in pcif_callback_args_t.

◆ pcif_api_t

struct pcif_api_t

PCIF_API functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(pcif_ctrl_t *const p_ctrl, pcif_cfg_t const *const p_cfg)
fsp_err_t(*	start)(pcif_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(pcif_ctrl_t *const p_ctrl, pcif_stop_mode_t stop_mode)
fsp_err_t(*	setCounter)(pcif_ctrl_t *const p_ctrl, pcif_phc_counter_info_t const *const p_counter_info, bool pos_reset)
fsp_err_t(*	statusGet)(pcif_ctrl_t *const p_ctrl, pcif_status_t *const p_status)
fsp_err_t(*	softwareEventGenerate)(pcif_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(pcif_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* pcif_api_t::open) (pcif_ctrl_t *const p_ctrl, pcif_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_PCIF_Open()

Precondition

Peripheral clocks and any required output pins should be configured prior to calling this function.

Note

To reconfigure after calling this function, call pcif_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control structure. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to pin configuration structure. All elements of this structure must be set by user.

◆ start

fsp_err_t(* pcif_api_t::start) (pcif_ctrl_t *const p_ctrl)

Start the output waveform.

Implemented as

- R_PCIF_Start()

Parameters

[in]	p_ctrl	Control block set in pcif_api_t::open call for this module.

◆ stop

fsp_err_t(* pcif_api_t::stop) (pcif_ctrl_t *const p_ctrl, pcif_stop_mode_t stop_mode)

Stop the output waveform.

Implemented as

- R_PCIF_Stop()

Parameters

[in]	p_ctrl	Control block set in pcif_api_t::open call for this module.
[in]	stop_mode	Stop mode

◆ setCounter

fsp_err_t(* pcif_api_t::setCounter) (pcif_ctrl_t *const p_ctrl, pcif_phc_counter_info_t const *const p_counter_info, bool pos_reset)

Set the counter and the initial value.

Implemented as

- R_PCIF_SetCounter()

Note

This function is only available in phase counting mode.

Parameters

[in]	p_ctrl	Control block set in pcif_api_t::open call for this module.
[in]	p_counter_info	Pointer to counter info structure. All elements of this structure must be set by user.
[in]	pos_reset	Reset counter position.

◆ statusGet

fsp_err_t(* pcif_api_t::statusGet) (pcif_ctrl_t *const p_ctrl, pcif_status_t *const p_status)

Stores current status in p_status.

Implemented as

- R_PCIF_StatusGet()

Parameters

[in]	p_ctrl	Control block set in pcif_api_t::open call for this module.
[out]	p_status	Current status for this module.

◆ softwareEventGenerate

fsp_err_t(* pcif_api_t::softwareEventGenerate) (pcif_ctrl_t *const p_ctrl)

Generates a software event to trigger the PCIF.

Implemented as

- R_PCIF_SoftwareEventGenerate()

Note

This function is only available in incremental mode.
This function is used to generate a Z phase signal.

Parameters

[in]	p_ctrl	Control block set in pcif_api_t::open call for this module.

◆ close

fsp_err_t(* pcif_api_t::close) (pcif_ctrl_t *const p_ctrl)

Closes the driver and allows reconfiguration. May reduce power consumption.

Implemented as

- R_PCIF_Close()

Parameters

[in]	p_ctrl	Control block set in pcif_api_t::open call for this module.

◆ pcif_instance_t

struct pcif_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
pcif_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
pcif_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
pcif_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ pcif_ctrl_t

typedef void pcif_ctrl_t
PCIF_API control block. Allocate an instance specific control block to pass into the PCIF_API API calls.

Enumeration Type Documentation

◆ pcif_error_t

enum pcif_error_t	
PCIF Error Event	
Enumerator	
PCIF_ERROR_NONE	No error.
PCIF_ERROR_CARRIER_PERIOD	Carrier Period Event error.
PCIF_ERROR_INTERNAL_EDGE_COUNTING	Internal Edge Count error.
PCIF_ERROR_EDGE_COUNTING_OVERFLOW	Internal Edge Count Overflow error.
PCIF_ERROR_OLD_EDGE_COUNTING	Old Edge Count Usage error.

◆ pcif_state_t

enum pcif_state_t	
PCIF Status	
Enumerator	
PCIF_STATE_STOPPED	PCIF is idle.
PCIF_STATE_RUNNING	PCIF is busy.

◆ pcif_mode_t

enum pcif_mode_t	
PCIF Mode	
Enumerator	
PCIF_MODE_PHASE_COUNTING	Phase counting mode.
PCIF_MODE_INCREMENTAL	Incremental mode.

◆ pcif_clock_source_t

enum pcif_clock_source_t	
Selection of source clock	
Enumerator	
PCIF_CLOCK_SOURCE_ASYN_CLK	Select asynchronous clock source (GPTCLK)
PCIF_CLOCK_SOURCE_SYN_CLK	Select synchronous clock source (PCLK)

◆ pcif_clock_div_t

enum pcif_clock_div_t	
Division clock ratio for source clock	
Enumerator	
PCIF_CLOCK_DIV_1	Clock source divided by 1.
PCIF_CLOCK_DIV_2	Clock source divided by 2 with duty ratio 1:1.
PCIF_CLOCK_DIV_3	Clock source divided by 3 with duty ratio 2:1.
PCIF_CLOCK_DIV_4	Clock source divided by 4 with duty ratio 2:2.
PCIF_CLOCK_DIV_5	Clock source divided by 5 with duty ratio 3:2.
PCIF_CLOCK_DIV_6	Clock source divided by 6 with duty ratio 3:3.
PCIF_CLOCK_DIV_7	Clock source divided by 7 with duty ratio 4:3.
PCIF_CLOCK_DIV_8	Clock source divided by 8 with duty ratio 4:4.
PCIF_CLOCK_DIV_9	Clock source divided by 9 with duty ratio 5:4.
PCIF_CLOCK_DIV_10	Clock source divided by 10 with duty ratio 5:5.
PCIF_CLOCK_DIV_11	Clock source divided by 11 with duty ratio 6:5.
PCIF_CLOCK_DIV_12	Clock source divided by 12 with duty ratio 6:6.
PCIF_CLOCK_DIV_13	Clock source divided by 13 with duty ratio 7:6.
PCIF_CLOCK_DIV_14	Clock source divided by 14 with duty ratio 7:7.
PCIF_CLOCK_DIV_15	Clock source divided by 15 with duty ratio 8:7.
PCIF_CLOCK_DIV_16	Clock source divided by 16 with duty ratio 8:8.
PCIF_CLOCK_DIV_17	Clock source divided by 17 with duty ratio 9:8.
PCIF_CLOCK_DIV_18	Clock source divided by 18 with duty ratio 9:9.
PCIF_CLOCK_DIV_19	Clock source divided by 19 with duty ratio 10:9.
PCIF_CLOCK_DIV_20	Clock source divided by 20 with duty ratio 10:10.

PCIF_CLOCK_DIV_21	Clock source divided by 21 with duty ratio 11:10.
PCIF_CLOCK_DIV_22	Clock source divided by 22 with duty ratio 11:11.
PCIF_CLOCK_DIV_23	Clock source divided by 23 with duty ratio 12:11.
PCIF_CLOCK_DIV_24	Clock source divided by 24 with duty ratio 12:12.
PCIF_CLOCK_DIV_25	Clock source divided by 25 with duty ratio 13:12.
PCIF_CLOCK_DIV_26	Clock source divided by 26 with duty ratio 13:13.
PCIF_CLOCK_DIV_27	Clock source divided by 27 with duty ratio 14:13.
PCIF_CLOCK_DIV_28	Clock source divided by 28 with duty ratio 14:14.
PCIF_CLOCK_DIV_29	Clock source divided by 29 with duty ratio 15:14.
PCIF_CLOCK_DIV_30	Clock source divided by 30 with duty ratio 15:15.
PCIF_CLOCK_DIV_31	Clock source divided by 31 with duty ratio 16:15.
PCIF_CLOCK_DIV_32	Clock source divided by 32 with duty ratio 16:16.

◆ pcif_phase_polarity_t

enum pcif_phase_polarity_t	
Output phase polarity selection	
Enumerator	
PCIF_PHASE_POSITIVE	Positive phase.
PCIF_PHASE_NEGATIVE	Negative phase.

◆ pcif_stop_mode_t

enum pcif_stop_mode_t	
Stop mode	
Enumerator	
PCIF_STOP_MODE_IMMEDIATE	PCIF stops immediately.
PCIF_STOP_MODE_SYNC	PCIF stops after generating the ABZ phase and synchronizing with the carrier cycle.

◆ pcif_rotation_direction_t

enum pcif_rotation_direction_t	
Rotation direction	
Enumerator	
PCIF_ROTATION_DIRECTION_CW	Clockwise rotation.
PCIF_ROTATION_DIRECTION_CCW	Counter-clockwise rotation.

5.3.5.7 SPI Interface

Interfaces » Connectivity

Detailed Description

Interface for SPI communications.

Summary

Provides a common interface for communication using the SPI Protocol.

Data Structures

struct	spi_callback_args_t
struct	spi_write_read_guard_args_t
struct	spi_cfg_t
struct	spi_api_t

struct	spi_instance_t

Typedefs

typedef void	spi_ctrl_t

Enumerations

enum	spi_bit_width_t
enum	spi_mode_t
enum	spi_clk_phase_t
enum	spi_clk_polarity_t
enum	spi_mode_fault_t
enum	spi_bit_order_t
enum	spi_event_t

Data Structure Documentation

◆ spi_callback_args_t

struct spi_callback_args_t		
Common callback parameter definition		
Data Fields		
uint32_t	channel	Device channel number.
spi_event_t	event	Event code.
void *	p_context	Context provided to user during callback.

◆ spi_write_read_guard_args_t

struct spi_write_read_guard_args_t
Non-secure arguments for write-read guard function

◆ spi_cfg_t

struct spi_cfg_t	
SPI interface configuration	
Data Fields	
uint8_t	channel
	Channel number to be used.

IRQn_Type	rxi_irq
	Receive Buffer Full IRQ number.
IRQn_Type	txi_irq
	Transmit Buffer Empty IRQ number.
IRQn_Type	tei_irq
	Transfer Complete IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
uint8_t	rxi_ipl
	Receive Interrupt priority.
uint8_t	txi_ipl
	Transmit Interrupt priority.
uint8_t	tei_ipl
	Transfer Complete Interrupt priority.
uint8_t	eri_ipl
	Error Interrupt priority.
spi_mode_t	operating_mode
	Select master or slave operating mode.

spi_clk_phase_t	clk_phase
	Data sampling on odd or even clock edge.
spi_clk_polarity_t	clk_polarity
	Clock level when idle.
spi_mode_fault_t	mode_fault
	Mode fault error (master/slave conflict) flag.
spi_bit_order_t	bit_order
	Select to transmit MSB/LSB first.
transfer_instance_t const *	p_transfer_tx
	To use SPI DTC/DMAC write transfer, link a transfer instance here. Set to NULL if unused.
transfer_instance_t const *	p_transfer_rx
	To use SPI DTC/DMAC read transfer, link a transfer instance here. Set to NULL if unused.
void(*	p_callback)(spi_callback_args_t *p_args)
	Pointer to user callback function.
void *	p_context
	User defined context passed to callback function.
void const *	p_extend
	Extended SPI hardware dependent configuration.

◆ spi_api_t

struct spi_api_t

Shared Interface definition for SPI

Data Fields

fsp_err_t(*	open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t(*	read)(spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t(*	write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t(*	writeRead)(spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t(*	callbackSet)(spi_ctrl_t *const p_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(spi_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Initialize a channel for SPI communication mode.

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to SPI configuration structure.

◆ read

fsp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Receive data from a SPI device.

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ write

fsp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)

Transmit data to a SPI device.

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ writeRead

fsp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Simultaneously transmit data to a SPI device while receiving data from a SPI device (full duplex).

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ callbackSet

fsp_err_t(* spi_api_t::callbackSet) (spi_ctrl_t *const p_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the SPI control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)

Remove power to the SPI channel designated by the handle and disable the associated interrupts.

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.

◆ spi_instance_t

struct spi_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
spi_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
spi_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
spi_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ spi_ctrl_t

typedef void spi_ctrl_t
SPI control block. Allocate an instance specific control block to pass into the SPI API calls.

Enumeration Type Documentation

◆ spi_bit_width_t

enum spi_bit_width_t	
Data bit width	
Enumerator	
SPI_BIT_WIDTH_4_BITS	Data bit width is 4 bits.
SPI_BIT_WIDTH_5_BITS	Data bit width is 5 bits.
SPI_BIT_WIDTH_6_BITS	Data bit width is 6 bits.
SPI_BIT_WIDTH_7_BITS	Data bit width is 7 bits.
SPI_BIT_WIDTH_8_BITS	Data bit width is 8 bits.
SPI_BIT_WIDTH_9_BITS	Data bit width is 9 bits.
SPI_BIT_WIDTH_10_BITS	Data bit width is 10 bits.
SPI_BIT_WIDTH_11_BITS	Data bit width is 11 bits.
SPI_BIT_WIDTH_12_BITS	Data bit width is 12 bits.
SPI_BIT_WIDTH_13_BITS	Data bit width is 13 bits.
SPI_BIT_WIDTH_14_BITS	Data bit width is 14 bits.
SPI_BIT_WIDTH_15_BITS	Data bit width is 15 bits.
SPI_BIT_WIDTH_16_BITS	Data bit width is 16 bits.
SPI_BIT_WIDTH_17_BITS	Data bit width is 17 bits.
SPI_BIT_WIDTH_18_BITS	Data bit width is 18 bits.
SPI_BIT_WIDTH_19_BITS	Data bit width is 19 bits.
SPI_BIT_WIDTH_20_BITS	Data bit width is 20 bits.

SPI_BIT_WIDTH_21_BITS	Data bit width is 21 bits.
SPI_BIT_WIDTH_22_BITS	Data bit width is 22 bits.
SPI_BIT_WIDTH_23_BITS	Data bit width is 23 bits.
SPI_BIT_WIDTH_24_BITS	Data bit width is 24 bits.
SPI_BIT_WIDTH_25_BITS	Data bit width is 25 bits.
SPI_BIT_WIDTH_26_BITS	Data bit width is 26 bits.
SPI_BIT_WIDTH_27_BITS	Data bit width is 27 bits.
SPI_BIT_WIDTH_28_BITS	Data bit width is 28 bits.
SPI_BIT_WIDTH_29_BITS	Data bit width is 29 bits.
SPI_BIT_WIDTH_30_BITS	Data bit width is 30 bits.
SPI_BIT_WIDTH_31_BITS	Data bit width is 31 bits.
SPI_BIT_WIDTH_32_BITS	Data bit width is 32 bits.

◆ spi_mode_t

enum spi_mode_t	
Master or slave operating mode	
Enumerator	
SPI_MODE_MASTER	Channel operates as SPI master.
SPI_MODE_SLAVE	Channel operates as SPI slave.

◆ spi_clk_phase_t

enum spi_clk_phase_t	
Clock phase	
Enumerator	
SPI_CLK_PHASE_EDGE_ODD	0: Data sampling on odd edge, data variation on even edge
SPI_CLK_PHASE_EDGE_EVEN	1: Data variation on odd edge, data sampling on even edge

◆ spi_clk_polarity_t

enum spi_clk_polarity_t	
Clock polarity	
Enumerator	
SPI_CLK_POLARITY_LOW	0: Clock polarity is low when idle
SPI_CLK_POLARITY_HIGH	1: Clock polarity is high when idle

◆ spi_mode_fault_t

enum spi_mode_fault_t	
Mode fault error flag. This error occurs when the device is setup as a master, but the SSLA line does not seem to be controlled by the master. This usually happens when the connecting device is also acting as master. A similar situation can also happen when configured as a slave.	
Enumerator	
SPI_MODE_FAULT_ERROR_ENABLE	Mode fault error flag on.
SPI_MODE_FAULT_ERROR_DISABLE	Mode fault error flag off.

◆ spi_bit_order_t

enum spi_bit_order_t	
Bit order	
Enumerator	
SPI_BIT_ORDER_MSB_FIRST	Send MSB first in transmission.
SPI_BIT_ORDER_LSB_FIRST	Send LSB first in transmission.

◆ spi_event_t

enum spi_event_t	
SPI events	
Enumerator	
SPI_EVENT_TRANSFER_COMPLETE	The data transfer was completed.
SPI_EVENT_TRANSFER_ABORTED	The data transfer was aborted.
SPI_EVENT_ERR_MODE_FAULT	Mode fault error.
SPI_EVENT_ERR_READ_OVERFLOW	Read overflow error.
SPI_EVENT_ERR_PARITY	Parity error.
SPI_EVENT_ERR_OVERRUN	Overrun error.
SPI_EVENT_ERR_FRAMING	Framing error.
SPI_EVENT_ERR_MODE_UNDERRUN	Underrun error.

5.3.5.8 UART Interface

Interfaces » Connectivity

Detailed Description

Interface for UART communications.

Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change
- Hardware resource locking during a transaction
- CTS/RTS hardware flow control support (with an associated IOPORT pin)

Data Structures

struct	uart_info_t
struct	uart_callback_args_t
struct	uart_cfg_t
struct	uart_api_t
struct	uart_instance_t

Typedefs

typedef void	uart_ctrl_t

Enumerations

enum	uart_event_t
enum	uart_data_bits_t
enum	uart_parity_t
enum	uart_stop_bits_t
enum	uart_dir_t

Data Structure Documentation

◆ uart_info_t

struct uart_info_t		
UART driver specific information		
Data Fields		
uint32_t	write_bytes_max	Maximum bytes that can be written at this time. Only applies if uart_cfg_t::p_transfer_tx is not NULL.

uint32_t	read_bytes_max	Maximum bytes that are available to read at one time. Only applies if uart_cfg_t::p_transfer_rx is not NULL.

◆ uart_callback_args_t

struct uart_callback_args_t		
UART Callback parameter definition		
Data Fields		
uint32_t	channel	Device channel number.
uart_event_t	event	Event code.
uint32_t	data	Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused.
void *	p_context	Context provided to user during callback.

◆ uart_cfg_t

struct uart_cfg_t	
UART Configuration	
Data Fields	
uint8_t	channel
	Select a channel corresponding to the channel number of the hardware.
uart_data_bits_t	data_bits
	Data bit length (8 or 7 or 9)
uart_parity_t	parity
	Parity type (none or odd or even)
uart_stop_bits_t	stop_bits
	Stop bit length (1 or 2)

uint8_t	rxi_ipl
	Receive interrupt priority.
IRQn_Type	rxi_irq
	Receive interrupt IRQ number.
uint8_t	txi_ipl
	Transmit interrupt priority.
IRQn_Type	txi_irq
	Transmit interrupt IRQ number.
uint8_t	tei_ipl
	Transmit end interrupt priority.
IRQn_Type	tei_irq
	Transmit end interrupt IRQ number.
uint8_t	eri_ipl
	Error interrupt priority.
IRQn_Type	eri_irq
	Error interrupt IRQ number.
transfer_instance_t const *	p_transfer_rx
transfer_instance_t const *	p_transfer_tx

void(*	p_callback)(uart_callback_args_t *p_args)
	Pointer to callback function.
void *	p_context
	User defined context passed into callback function.
void const *	p_extend
	UART hardware dependent configuration.

Field Documentation

◆ p_transfer_rx

transfer_instance_t const* uart_cfg_t::p_transfer_rx

Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

◆ p_transfer_tx

transfer_instance_t const* uart_cfg_t::p_transfer_tx

Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

◆ uart_api_t

struct uart_api_t

Shared Interface definition for UART

Data Fields

fsp_err_t(*	open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
fsp_err_t(*	read)(uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t(*	write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)
fsp_err_t(*	baudSet)(uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)

fsp_err_t(*	infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
fsp_err_t(*	communicationAbort)(uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)
fsp_err_t(*	callbackSet)(uart_ctrl_t *const p_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(uart_ctrl_t *const p_ctrl)
fsp_err_t(*	readStop)(uart_ctrl_t *const p_ctrl, uint32_t *remaining_bytes)
fsp_err_t(*	receiveSuspend)(uart_ctrl_t *const p_ctrl)
fsp_err_t(*	receiveResume)(uart_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)

Open UART device.

Parameters

[in,out]	p_ctrl	Pointer to the UART control block. Must be declared by user. Value set here.
[in]	uart_cfg_t	Pointer to UART configuration structure. All elements of this structure must be set by user.

◆ read

fsp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read from UART device. The read buffer is used until the read is complete. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR. The maximum transfer size is reported by infoGet().

Parameters

[in]	p_ctrl	Pointer to the UART control block for the channel.
[in]	p_dest	Destination address to read data from.
[in]	bytes	Read data length.

◆ write

fsp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE. The maximum transfer size is reported by infoGet().

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_src	Source address to write data to.
[in]	bytes	Write data length.

◆ baudSet

fsp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)

Change baud rate.

Warning

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_baudrate_info	Pointer to module specific information for configuring baud rate.

◆ infoGet

fsp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

Get the driver specific information.

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[out]	p_info	Pointer to UART information structure.

◆ communicationAbort

fsp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

Abort ongoing transfer.

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	communication_to_abort	Type of abort request.

◆ callbackSet

fsp_err_t(* uart_api_t::callbackSet) (uart_ctrl_t *const p_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)

Close UART device.

Parameters

[in]	p_ctrl	Pointer to the UART control block.

◆ readStop

fsp_err_t(* uart_api_t::readStop) (uart_ctrl_t *const p_ctrl, uint32_t *remaining_bytes)

Stop ongoing read and return the number of bytes remaining in the read.

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in,out]	remaining_bytes	Pointer to location to store remaining bytes for read.

◆ receiveSuspend

fsp_err_t(* uart_api_t::receiveSuspend) (uart_ctrl_t *const p_ctrl)

Suspend RX operations for UART device.

Parameters

[in]	p_ctrl	Pointer to the UART control block.

◆ receiveResume

fsp_err_t(* uart_api_t::receiveResume) (uart_ctrl_t *const p_ctrl)

Resume RX operations for UART device.

Parameters

[in]	p_ctrl	Pointer to the UART control block.

◆ uart_instance_t

struct uart_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
uart_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
uart_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
uart_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ uart_ctrl_t

typedef void uart_ctrl_t
UART control block. Allocate an instance specific control block to pass into the UART API calls.

Enumeration Type Documentation

◆ uart_event_t

enum uart_event_t	
UART Event codes	
Enumerator	
UART_EVENT_RX_COMPLETE	Receive complete event.
UART_EVENT_TX_COMPLETE	Transmit complete event.
UART_EVENT_RX_CHAR	Character received.
UART_EVENT_ERR_PARITY	Parity error event.
UART_EVENT_ERR_FRAMING	Mode fault error event.
UART_EVENT_ERR_OVERFLOW	FIFO Overflow error event.
UART_EVENT_BREAK_DETECT	Break detect error event.
UART_EVENT_TX_DATA_EMPTY	Last byte is transmitting, ready for more data.

◆ uart_data_bits_t

enum uart_data_bits_t	
UART Data bit length definition	
Enumerator	
UART_DATA_BITS_9	Data bits 9-bit.
UART_DATA_BITS_8	Data bits 8-bit.
UART_DATA_BITS_7	Data bits 7-bit.

◆ uart_parity_t

enum uart_parity_t	
UART Parity definition	
Enumerator	
UART_PARITY_OFF	No parity.
UART_PARITY_ZERO	Zero parity.
UART_PARITY_EVEN	Even parity.
UART_PARITY_ODD	Odd parity.

◆ uart_stop_bits_t

enum uart_stop_bits_t	
UART Stop bits definition	
Enumerator	
UART_STOP_BITS_1	Stop bit 1-bit.
UART_STOP_BITS_2	Stop bits 2-bit.

◆ uart_dir_t

enum uart_dir_t	
UART transaction definition	
Enumerator	
UART_DIR_RX_TX	Both RX and TX.
UART_DIR_RX	Only RX.
UART_DIR_TX	Only TX.

5.3.5.9 USB Interface

Interfaces » Connectivity

Detailed Description

Interface for USB functions.

Summary

The USB interface provides USB functionality.

Data Structures

struct	usb_cfg_t
struct	usb_api_t
struct	usb_instance_t

Macros

#define	USB_BREQUEST b15-8
#define	USB_GET_STATUS USB Standard request Get Status.
#define	USB_CLEAR_FEATURE USB Standard request Clear Feature.
#define	USB_REQRESERVED USB Standard request Reqreserved.
#define	USB_SET_FEATURE USB Standard request Set Feature.
#define	USB_REQRESERVED1 USB Standard request Reqreserved1.
#define	USB_SET_ADDRESS USB Standard request Set Address.
#define	USB_GET_DESCRIPTOR

	USB Standard request Get Descriptor.
#define	USB_SET_DESCRIPTOR USB Standard request Set Descriptor.
#define	USB_GET_CONFIGURATION USB Standard request Get Configuration.
#define	USB_SET_CONFIGURATION USB Standard request Set Configuration.
#define	USB_GET_INTERFACE USB Standard request Get Interface.
#define	USB_SET_INTERFACE USB Standard request Set Interface.
#define	USB_SYNCH_FRAME USB Standard request Synch Frame.
#define	USB_HOST_TO_DEV From host to device.
#define	USB_DEV_TO_HOST From device to host.
#define	USB_STANDARD Standard Request.
#define	USB_CLASS Class Request.
#define	USB_VENDOR Vendor Request.

#define	USB_DEVICE Device.
#define	USB_INTERFACE Interface.
#define	USB_ENDPOINT End Point.
#define	USB_OTHER Other.
#define	USB_NULL NULL pointer.
#define	USB_IP0 USB0 module.
#define	USB_IP1 USB1 module.
#define	USB_PIPE0 Pipe Number0.
#define	USB_PIPE1 Pipe Number1.
#define	USB_PIPE2 Pipe Number2.
#define	USB_PIPE3 Pipe Number3.

#define USB_PIPE4

Pipe Number4.

#define USB_PIPE5

Pipe Number5.

#define USB_PIPE6

Pipe Number6.

#define USB_PIPE7

Pipe Number7.

#define USB_PIPE8

Pipe Number8.

#define USB_PIPE9

Pipe Number9.

#define USB_EP0

End Point Number0.

#define USB_EP1

End Point Number1.

#define USB_EP2

End Point Number2.

#define USB_EP3

End Point Number3.

#define USB_EP4

End Point Number4.

#define USB_EP5

	End Point Number5.
#define	USB_EP6 End Point Number6.
#define	USB_EP7 End Point Number7.
#define	USB_EP8 End Point Number8.
#define	USB_EP9 End Point Number9.
#define	USB_EP10 End Point Number10.
#define	USB_EP11 End Point Number11.
#define	USB_EP12 End Point Number12.
#define	USB_EP13 End Point Number13.
#define	USB_EP14 End Point Number14.
#define	USB_EP15 End Point Number15.
#define	USB_EP_DIR b7: Endpoint Direction

#define	USB_EP_DIR_IN b7: Endpoint Direction In
#define	USB_EP_DIR_OUT b7: Endpoint Direction Out
#define	USB_DT_DEVICE Device Descriptor.
#define	USB_DT_CONFIGURATION Configuration Descriptor.
#define	USB_DT_STRING String Descriptor.
#define	USB_DT_INTERFACE Interface Descriptor.
#define	USB_DT_ENDPOINT Endpoint Descriptor.
#define	USB_DT_DEVICE_QUALIFIER Device Qualifier Descriptor.
#define	USB_DT_OTHER_SPEED_CONF Other Speed Configuration Descriptor.
#define	USB_DT_INTERFACE_POWER Interface Power Descriptor.
#define	USB_DT_OTGDESCRIPTOR OTG Descriptor.

#define	USB_DT_HUBDESCRIPTOR HUB descriptor.
#define	USB_IFCLS_NOT Un corresponding Class.
#define	USB_IFCLS_AUD Audio Class.
#define	USB_IFCLS_CDC CDC Class.
#define	USB_IFCLS_CDCC CDC-Control Class.
#define	USB_IFCLS_HID HID Class.
#define	USB_IFCLS_PHY Physical Class.
#define	USB_IFCLS_IMG Image Class.
#define	USB_IFCLS_PRN Printer Class.
#define	USB_IFCLS_MAS Mass Storage Class.
#define	USB_IFCLS_HUB HUB Class.
#define	USB_IFCLS_CDCD

	CDC-Data Class.
#define	USB_IFCLS_CHIP Chip/Smart Card Class.
#define	USB_IFCLS_CNT Content-Security Class.
#define	USB_IFCLS_VID Video Class.
#define	USB_IFCLS_DIAG Diagnostic Device.
#define	USB_IFCLS_WIRE Wireless Controller.
#define	USB_IFCLS_APL Application-Specific.
#define	USB_IFCLS_VEN Vendor-Specific Class.
#define	USB_EP_IN In Endpoint.
#define	USB_EP_OUT Out Endpoint.
#define	USB_EP_ISO Isochronous Transfer.
#define	USB_EP_BULK Bulk Transfer.

#define	USB_EP_INT Interrupt Transfer.
#define	USB_CF_RESERVED Reserved(set to 1)
#define	USB_CF_SELFP Self Powered.
#define	USB_CF_BUSP Bus Powered.
#define	USB_CF_RWUPON Remote Wake up ON.
#define	USB_CF_RWUPOFF Remote Wake up OFF.
#define	USB_DD_BLENGTH Device Descriptor Length.
#define	USB_CD_BLENGTH Configuration Descriptor Length.
#define	USB_ID_BLENGTH Interface Descriptor Length.
#define	USB_ED_BLENGTH Endpoint Descriptor Length.

Typedefs

typedef void	usb_ctrl_t

Enumerations

enum	usb_speed_t
enum	usb_setup_status_t
enum	usb_status_t
enum	usb_class_t
enum	usb_bcport_t
enum	usb_onoff_t
enum	usb_transfer_t
enum	usb_transfer_type_t
enum	usb_mode_t
enum	usb_compliancetest_status_t

Data Structure Documentation

◆ usb_cfg_t

struct usb_cfg_t		
USB configuration.		
Data Fields		
usb_mode_t	usb_mode	USB_MODE_HOST/USB_MODE_PERI.
usb_speed_t	usb_speed	USB speed (USB_HS/USB_FS/USB_LS)
uint8_t	module_number	USB module number (USB_IP0/USB_IP1)
usb_class_t	type	USB device class etc.
usb_descriptor_t *	p_usb_reg	Pointer to the usb_decriptor_t structure area.
usb_compliance_cb_t *	usb_complience_cb	
IRQn_Type	irq	USBI dedicated interrupt number storage variable.
IRQn_Type	irq_r	USBR dedicated interrupt number storage variable.
IRQn_Type	irq_d0	FS D0FIFO dedicated interrupt number storage variable.
IRQn_Type	irq_d1	FS D1FIFO dedicated interrupt

		number storage variable.
IRQn_Type	hsirq	USBIR dedicated interrupt number storage variable.
IRQn_Type	hsirq_d0	HS D0FIFO dedicated interrupt number storage variable.
IRQn_Type	hsirq_d1	HS D1FIFO dedicated interrupt number storage variable.
uint8_t	ipl	Variable to store the interrupt priority of USBI.
uint8_t	ipl_r	Variable to store the interrupt priority of USBR.
uint8_t	ipl_d0	Variable to store the interrupt priority of FS D0FIFO.
uint8_t	ipl_d1	Variable to store the interrupt priority of FS D1FIFO.
uint8_t	hsipl	Variable to store the interrupt priority of USBIR.
uint8_t	hsipl_d0	Variable to store the interrupt priority of HS D0FIFO.
uint8_t	hsipl_d1	Variable to store the interrupt priority of HS D1FIFO.
usb_callback_t *	p_usb_apl_callback	Application Callback.
void *	p_context	Other Context.
const transfer_instance_t *	p_transfer_tx	Send context.
const transfer_instance_t *	p_transfer_rx	Receive context.
void const *	p_extend	Pointer to extended configuration by instance of interface.

◆ usb_api_t

struct usb_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(usb_ctrl_t *const p_ctrl, usb_cfg_t const *const p_cfg)
fsp_err_t(*	close)(usb_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)

fsp_err_t(*	write)(usb_ctrl_t *const p_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)
fsp_err_t(*	stop)(usb_ctrl_t *const p_ctrl, usb_transfer_t direction, uint8_t destination)
fsp_err_t(*	suspend)(usb_ctrl_t *const p_ctrl)
fsp_err_t(*	resume)(usb_ctrl_t *const p_ctrl)
fsp_err_t(*	vbusSet)(usb_ctrl_t *const p_ctrl, uint16_t state)
fsp_err_t(*	infoGet)(usb_ctrl_t *const p_ctrl, usb_info_t *p_info, uint8_t destination)
fsp_err_t(*	pipeRead)(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)
fsp_err_t(*	pipeWrite)(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)
fsp_err_t(*	pipeStop)(usb_ctrl_t *const p_ctrl, uint8_t pipe_number)
fsp_err_t(*	usedPipesGet)(usb_ctrl_t *const p_ctrl, uint16_t *p_pipe, uint8_t destination)
fsp_err_t(*	pipeInfoGet)(usb_ctrl_t *const p_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)
fsp_err_t(*	eventGet)(usb_ctrl_t *const p_ctrl, usb_status_t *event)
fsp_err_t(*	callback)(usb_callback_t *p_callback)
fsp_err_t(*	pullUp)(usb_ctrl_t *const p_ctrl, uint8_t state)
fsp_err_t(*	hostControlTransfer)(usb_ctrl_t *const p_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)

fsp_err_t(*	periControlDataGet)(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size)
fsp_err_t(*	periControlDataSet)(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size)
fsp_err_t(*	periControlStatusSet)(usb_ctrl_t *const p_ctrl, usb_setup_status_t status)
fsp_err_t(*	remoteWakeup)(usb_ctrl_t *const p_ctrl)
fsp_err_t(*	driverActivate)(usb_ctrl_t *const p_api_ctrl)
fsp_err_t(*	callbackMemorySet)(usb_ctrl_t *const p_api_ctrl, usb_callback_args_t *p_callback_memory)
fsp_err_t(*	moduleNumberGet)(usb_ctrl_t *const p_ctrl, uint8_t *module_number)
fsp_err_t(*	classTypeGet)(usb_ctrl_t *const p_ctrl, usb_class_t *class_type)
fsp_err_t(*	deviceAddressGet)(usb_ctrl_t *const p_ctrl, uint8_t *device_address)
fsp_err_t(*	pipeNumberGet)(usb_ctrl_t *const p_ctrl, uint8_t *pipe_number)
fsp_err_t(*	deviceStateGet)(usb_ctrl_t *const p_ctrl, uint16_t *state)
fsp_err_t(*	dataSizeGet)(usb_ctrl_t *const p_ctrl, uint32_t *data_size)
fsp_err_t(*	setupGet)(usb_ctrl_t *const p_ctrl, usb_setup_t *setup)
fsp_err_t(*	otgCallbackSet)(usb_ctrl_t *const p_ctrl, usb_otg_callback_t *p_callback)
fsp_err_t(*	otgSRP)(usb_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* usb_api_t::open) (usb_ctrl_t *const p_ctrl, usb_cfg_t const *const p_cfg)

Start the USB module

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* usb_api_t::close) (usb_ctrl_t *const p_ctrl)

Stop the USB module

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* usb_api_t::read) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)

Request USB data read

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	destination	In Host mode, it represents the device address, and in Peripheral mode, it represents the device class.

◆ write

fsp_err_t(* usb_api_t::write) (usb_ctrl_t *const p_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)

Request USB data write

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores write data.
[in]	size	Read request size.
[in]	destination	In Host mode, it represents the device address, and in Peripheral mode, it represents the device class.

◆ stop

fsp_err_t(* usb_api_t::stop) (usb_ctrl_t *const p_ctrl, usb_transfer_t direction, uint8_t destination)

Stop USB data read/write processing

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	direction	Receive (USB_TRANSFER_READ) or send (USB_TRANSFER_WRITE).
[in]	destination	In Host mode, it represents the device address, and in Peripheral mode, it represents the device class.

◆ suspend

fsp_err_t(* usb_api_t::suspend) (usb_ctrl_t *const p_ctrl)

Request suspend

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ resume

fsp_err_t(* usb_api_t::resume) (usb_ctrl_t *const p_ctrl)

Request resume

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ vbusSet

fsp_err_t(* usb_api_t::vbusSet) (usb_ctrl_t *const p_ctrl, uint16_t state)

Sets VBUS supply start/stop.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	state	VBUS supply start/stop specification

◆ infoGet

fsp_err_t(* usb_api_t::infoGet) (usb_ctrl_t *const p_ctrl, usb_info_t *p_info, uint8_t destination)

Get information on USB device.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_info	Pointer to usb_info_t structure area.
[in]	destination	Device address for Host.

◆ pipeRead

fsp_err_t(* usb_api_t::pipeRead) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data read from specified pipe

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	pipe_number	Pipe Number.

◆ pipeWrite

fsp_err_t(* usb_api_t::pipeWrite) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data write to specified pipe

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores write data.
[in]	size	Read request size.
[in]	pipe_number	Pipe Number.

◆ pipeStop

fsp_err_t(* usb_api_t::pipeStop) (usb_ctrl_t *const p_ctrl, uint8_t pipe_number)

Stop USB data read/write processing to specified pipe

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pipe_number	Pipe Number.

◆ usedPipesGet

fsp_err_t(* usb_api_t::usedPipesGet) (usb_ctrl_t *const p_ctrl, uint16_t *p_pipe, uint8_t destination)

Get pipe number

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_pipe	Pointer to area that stores the selected pipe number (bit map information).
[in]	destination	Device address for Host.

◆ pipeInfoGet

fsp_err_t(* usb_api_t::pipeInfoGet) (usb_ctrl_t *const p_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)

Get pipe information

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_info	Pointer to usb_pipe_t structure area.
[in]	pipe_number	Pipe Number.

◆ eventGet

fsp_err_t(* usb_api_t::eventGet) (usb_ctrl_t *const p_ctrl, usb_status_t *event)

Return USB-related completed events (OS less only)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	event	Pointer to event.

◆ callback

fsp_err_t(* usb_api_t::callback) (usb_callback_t *p_callback)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

Parameters

[in]	p_callback	Pointer to Callback function.

◆ pullUp

fsp_err_t(* usb_api_t::pullUp) (usb_ctrl_t *const p_ctrl, uint8_t state)

Pull-up enable/disable setting of D+/D- line.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	state	Pull-up enable/disable setting.

◆ hostControlTransfer

fsp_err_t(* usb_api_t::hostControlTransfer) (usb_ctrl_t *const p_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)

Performs settings and transmission processing when transmitting a setup packet.

Parameters

[in]	p_ctrl	USB control structure.
[in]	p_setup	Setup packet information.
[in]	p_buf	Transfer area information.
[in]	device_address	Device address information.

◆ periControlDataGet

fsp_err_t(* usb_api_t::periControlDataGet) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size)

Receives data sent by control transfer.

Parameters

[in]	p_ctrl	USB control structure.
[in]	p_buf	Data reception area information.
[in]	size	Data reception size information.

◆ periControlDataSet

fsp_err_t(* usb_api_t::periControlDataSet) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size)

Performs transfer processing for control transfer.

Parameters

[in]	p_ctrl	USB control structure.
[in]	p_buf	Area information for data transfer.
[in]	size	Transfer size information.

◆ periControlStatusSet

fsp_err_t(* usb_api_t::periControlStatusSet) (usb_ctrl_t *const p_ctrl, usb_setup_status_t status)

Set the response to the setup packet.

Parameters

[in]	p_ctrl	USB control structure.
[in]	status	USB port startup information.

◆ remoteWakeup

fsp_err_t(* usb_api_t::remoteWakeup) (usb_ctrl_t *const p_ctrl)

Sends a remote wake-up signal to the connected Host.

Parameters

[in]	p_ctrl	USB control structure.

◆ driverActivate

fsp_err_t(* usb_api_t::driverActivate) (usb_ctrl_t *const p_api_ctrl)

Activate USB Driver

Parameters

[in]	p_api_ctrl	USB control structure.

◆ callbackMemorySet

fsp_err_t(* usb_api_t::callbackMemorySet) (usb_ctrl_t *const p_api_ctrl, usb_callback_args_t *p_callback_memory)

Set callback memory to USB driver.

Parameters

[in]	p_api_ctrl	USB control structure.
[in]	p_callback_memory	Pointer to store USB event information.

◆ moduleNumberGet

fsp_err_t(* usb_api_t::moduleNumberGet) (usb_ctrl_t *const p_ctrl, uint8_t *module_number)

This API gets the module number.

Parameters

[in]	p_ctrl	USB control structure.
[out]	module_number	Module number to get.

◆ classTypeGet

fsp_err_t(* usb_api_t::classTypeGet) (usb_ctrl_t *const p_ctrl, usb_class_t *class_type)

This API gets the module number.

Parameters

[in]	p_ctrl	USB control structure.
[out]	class_type	Class type to get.

◆ deviceAddressGet

fsp_err_t(* usb_api_t::deviceAddressGet) (usb_ctrl_t *const p_ctrl, uint8_t *device_address)

This API gets the device address.

Parameters

[in]	p_ctrl	USB control structure.
[out]	device_address	Device address to get.

◆ pipeNumberGet

fsp_err_t(* usb_api_t::pipeNumberGet) (usb_ctrl_t *const p_ctrl, uint8_t *pipe_number)

This API gets the pipe number.

Parameters

[in]	p_ctrl	USB control structure.
[out]	pipe_number	Pipe number to get.

◆ deviceStateGet

fsp_err_t(* usb_api_t::deviceStateGet) (usb_ctrl_t *const p_ctrl, uint16_t *state)

This API gets the state of the device.

Parameters

[in]	p_ctrl	USB control structure.
[out]	state	Device state to get.

◆ dataSizeGet

fsp_err_t(* usb_api_t::dataSizeGet) (usb_ctrl_t *const p_ctrl, uint32_t *data_size)

This API gets the data size.

Parameters

[in]	p_ctrl	USB control structure.
[out]	data_size	Data size to get.

◆ setupGet

fsp_err_t(* usb_api_t::setupGet) (usb_ctrl_t *const p_ctrl, usb_setup_t *setup)

This API gets the setup type.

Parameters

[in]	p_ctrl	USB control structure.
[out]	setup	Setup type to get.

◆ otgCallbackSet

fsp_err_t(* usb_api_t::otgCallbackSet) (usb_ctrl_t *const p_ctrl, usb_otg_callback_t *p_callback)

This API sets the callback function for OTG.

Parameters

[in]	p_ctrl	USB control structure.
[in]	p_callback	Pointer to the callback function for OTG.

◆ otgSRP

fsp_err_t(* usb_api_t::otgSRP) (usb_ctrl_t *const p_ctrl)
This API starts SRP processing for OTG.

Parameters

[in]	p_ctrl	USB control structure.

◆ usb_instance_t

struct usb_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
usb_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
usb_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
usb_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ usb_ctrl_t

typedef void usb_ctrl_t
USB control block. Allocate an instance specific control block to pass into the USB API calls.

Enumeration Type Documentation

◆ usb_speed_t

enum usb_speed_t	
USB speed type	
Enumerator	
USB_SPEED_LS	Low speed operation.
USB_SPEED_FS	Full speed operation.
USB_SPEED_HS	Hi speed operation.

◆ usb_setup_status_t

enum usb_setup_status_t	
USB request result	
Enumerator	
USB_SETUP_STATUS_ACK	ACK response.
USB_SETUP_STATUS_STALL	STALL response.

◆ usb_status_t

enum usb_status_t	
USB driver status	
Enumerator	
USB_STATUS_POWERED	Powered State.
USB_STATUS_DEFAULT	Default State.
USB_STATUS_ADDRESS	Address State.
USB_STATUS_CONFIGURED	Configured State.
USB_STATUS_SUSPEND	Suspend State.
USB_STATUS_RESUME	Resume State.
USB_STATUS_DETACH	Detach State.
USB_STATUS_REQUEST	Request State.
USB_STATUS_REQUEST_COMPLETE	Request Complete State.
USB_STATUS_READ_COMPLETE	Read Complete State.
USB_STATUS_WRITE_COMPLETE	Write Complete State.
USB_STATUS_BC	battery Charge State
USB_STATUS_OVERCURRENT	Over Current state.
USB_STATUS_NOT_SUPPORT	Device Not Support.
USB_STATUS_NONE	None Status.
USB_STATUS_MSC_CMD_COMPLETE	MSC_CMD Complete.

◆ usb_class_t

enum usb_class_t	
USB class type	
Enumerator	
USB_CLASS_PCDC	PCDC Class.

USB_CLASS_PCDCC	PCDCC Class.
USB_CLASS_PCDC2	PCDC2 Class.
USB_CLASS_PCDCC2	PCDCC2 Class.
USB_CLASS_PHID	PHID Class.
USB_CLASS_PHID2	PHID2 Class.
USB_CLASS_PAUD	PAUD Class.
USB_CLASS_PPRN	PPRN Class.
USB_CLASS_DFU	DFU Class.
USB_CLASS_PVND	PVND Class.
USB_CLASS_HCDC	HCDC Class.
USB_CLASS_HCDCC	HCDCC Class.
USB_CLASS_HHID	HHID Class.
USB_CLASS_HVND	HVND Class.
USB_CLASS_HMSC	HMSC Class.
USB_CLASS_PMSC	PMSC Class.
USB_CLASS_HPRN	HPRN Class.
USB_CLASS_HUVC	HUVC Class.
USB_CLASS_HAUD	HAUD Class.
USB_CLASS_REQUEST	USB Class Request.
USB_CLASS_HUB	HUB Class.
USB_CLASS_END	USB Class End Code.

◆ usb_bcport_t

enum usb_bcport_t	
USB battery charging type	
Enumerator	
USB_BCPORT_SDP	SDP port settings.
USB_BCPORT_CDP	CDP port settings.
USB_BCPORT_DCP	DCP port settings.

◆ usb_onoff_t

enum usb_onoff_t	
USB status	
Enumerator	
USB_OFF	USB Off State.
USB_ON	USB On State.

◆ usb_transfer_t

enum usb_transfer_t	
USB read/write type	
Enumerator	
USB_TRANSFER_READ	Data Receive communication.
USB_TRANSFER_WRITE	Data transmission communication.

◆ usb_transfer_type_t

enum usb_transfer_type_t	
USB transfer type	
Enumerator	
USB_TRANSFER_TYPE_BULK	Bulk communication.
USB_TRANSFER_TYPE_INT	Interrupt communication.
USB_TRANSFER_TYPE_ISO	Isochronous communication.

◆ usb_mode_t

enum usb_mode_t	
Enumerator	
USB_MODE_HOST	Host mode.
USB_MODE_PERI	Peripheral mode.

◆ usb_compliancetest_status_t

enum usb_compliancetest_status_t	
Enumerator	
USB_COMPLIANCETEST_ATTACH	Device Attach Detection.
USB_COMPLIANCETEST_DETACH	Device Detach Detection.
USB_COMPLIANCETEST_TPL	TPL device connect.
USB_COMPLIANCETEST_NOTTPL	Not TPL device connect.
USB_COMPLIANCETEST_HUB	USB Hub connect.
USB_COMPLIANCETEST_OVRC	Over current.
USB_COMPLIANCETEST_NORES	Response Time out for Control Read Transfer.
USB_COMPLIANCETEST_SETUP_ERR	Setup Transaction Error.

5.3.5.10 USB HCDC Interface

Interfaces » Connectivity

Detailed Description

Interface for USB HCDC functions.

Summary

The USB HCDC interface provides USB HCDC functionality.

Data Structures

struct	usb_hcdc_encapsulated_t
struct	usb_hcdc_abstractstate_t
struct	usb_hcdc_countrysetting_t
union	usb_hcdc_commfeature_t
struct	usb_hcdc_linecoding_t
struct	usb_hcdc_controllinestate_t
struct	usb_hcdc_serialstate_t
struct	usb_hcdc_breakduration_t
struct	usb_hcdc_device_info_t
struct	usb_hcdc_api_t
struct	usb_hcdc_instance_t

Enumerations

enum	usb_hcdc_data_bit_t
enum	usb_hcdc_stop_bit_t
enum	usb_hcdc_parity_bit_t
enum	usb_hcdc_line_speed_t
enum	usb_hcdc_feature_selector_t

Data Structure Documentation

◆ usb_hcdc_encapsulated_t

struct usb_hcdc_encapsulated_t		
Encapsulated data		
Data Fields		
uint8_t *	p_data	Protocol dependent data.
uint16_t	wlength	Data length in bytes.

◆ usb_hcdc_abstractstate_t

struct usb_hcdc_abstractstate_t		
Abstract Control Model (ACM) settings bitmap		
Data Fields		
uint16_t	bis: 1	Idle enable.
uint16_t	bdms: 1	Data multiplexing enable.
uint16_t	rsv: 14	Reserved.

◆ usb_hcdc_countrysetting_t

struct usb_hcdc_countrysetting_t		
Country code data		
Data Fields		
uint16_t	country_code	Country code.

◆ usb_hcdc_commfeature_t

union usb_hcdc_commfeature_t		
Feature setting data		
Data Fields		
usb_hcdc_abstractstate_t	abstract_state	ACM settings bitmap.
usb_hcdc_countrysetting_t	country_setting	Country code.

◆ usb_hcdc_linecoding_t

struct usb_hcdc_linecoding_t		
Virtual UART configuration (line coding)		
Data Fields		
usb_hcdc_line_speed_t	dwdte_rate	Data terminal rate in bits per second.
usb_hcdc_stop_bit_t	bchar_format	Stop bits.

usb_hcdc_parity_bit_t	bparity_type	Parity.
usb_hcdc_data_bit_t	bdata_bits	Data bits.
uint8_t	rsv	Reserved.

◆ usb_hcdc_controllinestate_t

struct usb_hcdc_controllinestate_t		
Virtual UART control signal bitmap		
Data Fields		
uint16_t	bdtr: 1	DTR.
uint16_t	brts: 1	RTS.
uint16_t	rsv: 14	Reserved.

◆ usb_hcdc_serialstate_t

struct usb_hcdc_serialstate_t		
Virtual UART state bitmap		
Data Fields		
uint16_t	brx_carrier: 1	DCD signal.
uint16_t	btx_carrier: 1	DSR signal.
uint16_t	bbreak: 1	Break detection status.
uint16_t	bring_signal: 1	Ring signal.
uint16_t	bframing: 1	Framing error.
uint16_t	bparity: 1	Parity error.
uint16_t	bover_run: 1	Over Run error.
uint16_t	rsv: 9	Reserved.

◆ usb_hcdc_breakduration_t

struct usb_hcdc_breakduration_t		
Break duration data		
Data Fields		
uint16_t	wtime_ms	Duration of Break.

◆ usb_hcdc_device_info_t

struct usb_hcdc_device_info_t		
Break duration data		
Data Fields		
uint16_t	vendor_id	Vendor ID.
uint16_t	product_id	Product ID.

uint8_t	subclass	Subclass code.

◆ usb_hcdc_api_t

struct usb_hcdc_api_t

USB HCDC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	controlDataRead)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t device_address)
fsp_err_t(*	deviceRegister)(usb_ctrl_t *const p_api_ctrl, uint16_t vendor_id, uint16_t product_id)
fsp_err_t(*	infoGet)(usb_ctrl_t *const p_api_ctrl, usb_hcdc_device_info_t *p_info, uint8_t device_address)

Field Documentation

◆ controlDataRead

fsp_err_t(* usb_hcdc_api_t::controlDataRead) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t device_address)

Read Control Data (CDC Interrupt IN data)

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	device_address	Device address.

◆ deviceRegister

fsp_err_t(* usb_hcdc_api_t::deviceRegister) (usb_ctrl_t *const p_api_ctrl, uint16_t vendor_id, uint16_t product_id)

Register the specified vendor class device in the device table.

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	vendor_id	Vendor ID.
[in]	product_id	Product ID.

◆ infoGet

fsp_err_t(* usb_hcdc_api_t::infoGet) (usb_ctrl_t *const p_api_ctrl, usb_hcdc_device_info_t *p_info, uint8_t device_address)

Get connected device information.

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_info	Pointer to store CDC device information.
[in]	device_address	Device address.

◆ usb_hcdc_instance_t

struct usb_hcdc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
usb_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
usb_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
usb_hcdc_api_t const *	p_api	Pointer to the API structure for this instance.

Enumeration Type Documentation

◆ usb_hcdc_data_bit_t

enum usb_hcdc_data_bit_t

Virtual UART data length

Enumerator	
USB_HCDC_DATA_BIT_7	7 bits
USB_HCDC_DATA_BIT_8	8 bits

◆ usb_hcdc_stop_bit_t

enum usb_hcdc_stop_bit_t	
Virtual UART stop bit length	
Enumerator	
USB_HCDC_STOP_BIT_1	1 bit
USB_HCDC_STOP_BIT_15	1.5 bits
USB_HCDC_STOP_BIT_2	2 bits

◆ usb_hcdc_parity_bit_t

enum usb_hcdc_parity_bit_t	
Virtual UART parity bit setting	
Enumerator	
USB_HCDC_PARITY_BIT_NONE	No parity bit.
USB_HCDC_PARITY_BIT_ODD	Odd parity.
USB_HCDC_PARITY_BIT_EVEN	Even parity.

◆ usb_hcdc_line_speed_t

enum usb_hcdc_line_speed_t
Virtual UART bitrate

◆ usb_hcdc_feature_selector_t

enum usb_hcdc_feature_selector_t
Feature Selector

5.3.5.11 USB HHID Interface

Interfaces » Connectivity

Detailed Description

Interface for USB HHID functions.

Summary

The USB HHID interface provides USB HHID functionality.

Data Structures

struct	usb_hhid_api_t
struct	usb_hhid_instance_t

Macros

#define	USB_HID_OTHER Other.
#define	USB_HID_KEYBOARD Keyboard.
#define	USB_HID_MOUSE Mouse.
#define	USB_HID_IN In Transfer.
#define	USB_HID_OUT Out Transfer.

Data Structure Documentation

◆ usb_hhid_api_t

struct usb_hhid_api_t	
USB HHID functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	typeGet)(usb_ctrl_t *const p_ctrl, uint8_t *p_type, uint8_t device_address)
fsp_err_t(*	maxPacketSizeGet)(usb_ctrl_t *const p_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Field Documentation

◆ typeGet

fsp_err_t(* usb_hhid_api_t::typeGet) (usb_ctrl_t *const p_ctrl, uint8_t *p_type, uint8_t device_address)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_type	Pointer to store HID protocol value.
[in]	device_address	Device Address.

◆ maxPacketSizeGet

fsp_err_t(* usb_hhid_api_t::maxPacketSizeGet) (usb_ctrl_t *const p_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_size	Pointer to the area to store the max package size.
[in]	direction	Transfer direction.
[in]	device_address	Device Address.

◆ usb_hhid_instance_t

struct usb_hhid_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

usb_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
usb_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
usb_hhid_api_t const *	p_api	Pointer to the API structure for this instance.

5.3.5.12 USB HMSC Interface

Interfaces » Connectivity

Detailed Description

Interface for USB HMSC functions.

Summary

The USB HMSC interface provides USB HMSC functionality.

Data Structures

struct	usb_hmsc_api_t

Enumerations

enum	usb_atapi_t
enum	usb_csw_result_t

Data Structure Documentation

◆ usb_hmsc_api_t

struct usb_hmsc_api_t	
USB HMSC functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	storageCommand)(usb_ctrl_t *const p_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)
fsp_err_t(*	driveNumberGet)(usb_ctrl_t *const p_ctrl, uint8_t *p_drive, uint8_t destination)
fsp_err_t(*	storageReadSector)(uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)
fsp_err_t(*	storageWriteSector)(uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)
fsp_err_t(*	semaphoreGet)(void)
fsp_err_t(*	semaphoreRelease)(void)

Field Documentation

◆ storageCommand

fsp_err_t(* usb_hmsc_api_t::storageCommand) (usb_ctrl_t *const p_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)

Processing for MassStorage(ATAPI) command.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	*buf	Pointer to the buffer area to store the transfer data.
[in]	command	ATAPI command.
[in]	destination	Represents a device address.

◆ driveNumberGet

fsp_err_t(* usb_hmsc_api_t::driveNumberGet) (usb_ctrl_t *const p_ctrl, uint8_t *p_drive, uint8_t destination)

Get number of Storage drive.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_drive	Store address for Drive No.
[in]	destination	Represents a device address.

◆ storageReadSector

fsp_err_t(* usb_hmsc_api_t::storageReadSector) (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)

Read sector information.

Parameters

[in]	drive_number	Drive number.
[out]	*buff	Pointer to the buffer area to store the transfer data.
[in]	sector_number	The sector number to start with.
[in]	sector_count	Transmit with the sector size of the number of times.

◆ storageWriteSector

fsp_err_t(* usb_hmsc_api_t::storageWriteSector) (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)

Write sector information.

Parameters

[in]	drive_number	Drive number.
[in]	*buff	Pointer to the buffer area to store the transfer data.
[in]	sector_number	The sector number to start with.
[in]	sector_count	Transmit with the sector size of the number of times.

◆ semaphoreGet

fsp_err_t(* usb_hmsc_api_t::semaphoreGet) (void)

Get Semaphore.

◆ semaphoreRelease

fsp_err_t(* usb_hmsc_api_t::semaphoreRelease) (void)

Release Semaphore.

Enumeration Type Documentation

◆ usb_atapi_t

enum usb_atapi_t	
ATAPI commands	
Enumerator	
USB_ATAPI_TEST_UNIT_READY	Test Unit Ready.
USB_ATAPI_REQUEST_SENSE	Request Sense.
USB_ATAPI_FORMAT_UNIT	Format Unit.
USB_ATAPI_INQUIRY	Inquiry.
USB_ATAPI_MODE_SELECT6	Mode Select6.
USB_ATAPI_MODE_SENSE6	Mode Sense6.
USB_ATAPI_START_STOP_UNIT	Start Stop Unit.
USB_ATAPI_PREVENT_ALLOW	Prevent Allow.
USB_ATAPI_READ_FORMAT_CAPACITY	Read Format Capacity.
USB_ATAPI_READ_CAPACITY	Read Capacity.
USB_ATAPI_READ10	Read10.
USB_ATAPI_WRITE10	Write10.
USB_ATAPI_SEEK	Seek.
USB_ATAPI_WRITE_AND_VERIFY	Write and Verify.
USB_ATAPI_VERIFY10	Verify10.
USB_ATAPI_MODE_SELECT10	Mode Select10.
USB_ATAPI_MODE_SENSE10	Mode Sense10.

◆ usb_csw_result_t

enum usb_csw_result_t	
Command Status Wrapper (CSW)	
Enumerator	
USB_CSW_RESULT_SUCCESS	CSW was successful.
USB_CSW_RESULT_FAIL	CSW failed.
USB_CSW_RESULT_PHASE	CSW has phase error.

5.3.5.13 USB PCDC Interface

Interfaces » Connectivity

Detailed Description

Interface for USB PCDC functions.

Summary

The USB PCDC interface provides USB PCDC functionality.

Data Structures

struct	usb_serial_state_bitmap_t
union	usb_sci_serialstate_t
struct	usb_pcdc_linecoding_t
struct	usb_pcdc_ctrllinestate_t

Macros

#define	USB_PCDC_SET_LINE_CODING Set Line Coding.
#define	USB_PCDC_GET_LINE_CODING Get Line Coding.

#define	USB_PCDC_SET_CONTROL_LINE_STATE Control Line State.
#define	USB_PCDC_SERIAL_STATE Serial State Code.
#define	USB_PCDC_SETUP_TBL_BSIZE Setup packet table size (uint16_t * 5)

Data Structure Documentation

◆ usb_serial_state_bitmap_t

struct usb_serial_state_bitmap_t		
Virtual UART signal state		
Data Fields		
uint16_t	b_rx_carrier: 1	DCD signal.
uint16_t	b_tx_carrier: 1	DSR signal.
uint16_t	b_break: 1	Break signal.
uint16_t	b_ring_signal: 1	Ring signal.
uint16_t	b_framing: 1	Framing error.
uint16_t	b_parity: 1	Parity error.
uint16_t	b_over_run: 1	Overrun error.
uint16_t	rsv: 9	Reserved.

◆ usb_sci_serialstate_t

union usb_sci_serialstate_t		
Class Notification Serial State		
Data Fields		
uint32_t	word	Word Access.
usb_serial_state_bitmap_t	bit	Bit Access.

◆ usb_pcdc_linecoding_t

struct usb_pcdc_linecoding_t		
Virtual UART communication settings		
Data Fields		
uint32_t	dw_dte_rate	Bitrate.

uint8_t	b_char_format	Stop bits.
uint8_t	b_parity_type	Parity.
uint8_t	b_data_bits	Data bits.
uint8_t	rsv	Reserved.

◆ **usb_pcdc_ctrllinestate_t**

struct usb_pcdc_ctrllinestate_t		
Virtual UART control line state		
Data Fields		
uint16_t	bdtr: 1	DTR.
uint16_t	brts: 1	RTS.
uint16_t	rsv: 14	Reserved.

5.3.5.14 USB PHID Interface

Interfaces » Connectivity

Detailed Description

Interface for USB PHID functions.

Summary

The USB interface provides USB functionality.

5.3.5.15 USB PMSC Interface

Interfaces » Connectivity

Detailed Description

Interface for USB PMSC functions.

Summary

The USB PMSC interface provides USB PMSC functionality.

Macros

```
#define USB_MASS_STORAGE_RESET
```

	Mass storage reset request code.
#define	USB_GET_MAX_LUN Get max logical unit number request code.

5.3.5.16 USB PPRN Interface

Interfaces » Connectivity

Detailed Description

Interface for USB PPRN functions.

Summary

The USB PPRN interface provides USB PPRN functionality.

Macros

#define	USB_PPRN_GET_DEVICE_ID Get Device ID.
#define	USB_PRPN_GET_PORT_STATUS Get Port Status.
#define	USB_PPRN_SOFT_RESET Soft Reset.
#define	USB_PPRN_PORT_STATUS_PAPER_EMPTY 1: Paper Empty, 0: Paper Not Empty
#define	USB_PPRN_PORT_STATUS_SELECT 1: Selected, 0: Not Selected
#define	USB_PPRN_PORT_STATUS_NOT_ERROR 1: No Error, 0; Error

5.3.5.17 Communications Middleware Interface

Interfaces » Connectivity

Detailed Description

Interface for Communications Middleware functions.

Summary

The Communications interface provides multiple communications functionality.

Data Structures

struct	rm_comms_write_read_params_t
struct	rm_comms_callback_args_t
struct	rm_comms_cfg_t
struct	rm_comms_api_t
struct	rm_comms_instance_t

Typedefs

typedef void	rm_comms_ctrl_t

Enumerations

enum	rm_comms_event_t

Data Structure Documentation

◆ **rm_comms_write_read_params_t**

struct rm_comms_write_read_params_t
Struct to pack params for writeRead

◆ **rm_comms_callback_args_t**

struct rm_comms_callback_args_t
Communications middleware callback parameter definition

◆ **rm_comms_cfg_t**

struct rm_comms_cfg_t

Communications middleware configuration block	
Data Fields	
uint32_t	semaphore_timeout
	Timeout for read/write.
void const *	p_extend
	Pointer to extended configuration by instance of interface.
void const *	p_lower_level_cfg
	Pointer to lower level driver configuration structure.
void *	p_context
	Pointer to the user-provided context.
void(*	p_callback)(rm_comms_callback_args_t *p_args)
	Pointer to callback function, mostly used if using non-blocking functionality.

◆ rm_comms_api_t

struct rm_comms_api_t	
COMM APIs	
Data Fields	
fsp_err_t(*	open)(rm_comms_ctrl_t *const p_ctrl, rm_comms_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rm_comms_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t(*	write)(rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

fsp_err_t(*	writeRead)(rm_comms_ctrl_t *const p_ctrl, rm_comms_write_read_params_t write_read_params)
fsp_err_t(*	callbackSet)(rm_comms_ctrl_t *const p_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)

Field Documentation

◆ open

fsp_err_t(* rm_comms_api_t::open) (rm_comms_ctrl_t *const p_ctrl, rm_comms_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* rm_comms_api_t::close) (rm_comms_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* rm_comms_api_t::read) (rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read data.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.

◆ write

fsp_err_t(* rm_comms_api_t::write) (rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Write data.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.

◆ writeRead

fsp_err_t(* rm_comms_api_t::writeRead) (rm_comms_ctrl_t *const p_ctrl, rm_comms_write_read_params_t write_read_params)

Write bytes over comms followed by a read, will have a struct for params.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	write_read_params	Parameters structure.

◆ callbackSet

fsp_err_t(* rm_comms_api_t::callbackSet) (rm_comms_ctrl_t *const p_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)

Specify callback function and optional context pointer.

Parameters

[in]	p_ctrl	Pointer to the control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function

◆ rm_comms_instance_t

struct rm_comms_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Typedef Documentation

◆ rm_comms_ctrl_t

typedef void rm_comms_ctrl_t
Communications control block. Allocate an instance specific control block to pass into the Communications API calls.

Enumeration Type Documentation

◆ rm_comms_event_t

enum rm_comms_event_t
Event in the callback function

5.3.6 DSP

Interfaces

DSP Interfaces.

5.3.7 Graphics

Interfaces

Detailed Description

Graphics Interfaces.

Modules

	CAPTURE Interface Interface for CAPTURE functions.

5.3.7.1 CAPTURE Interface

Interfaces » Graphics

Detailed Description

Interface for CAPTURE functions.

Summary

The CAPTURE interface provides the functionality for capturing an image from an image sensor/camera. When a capture is complete a capture complete interrupt is triggered.

Data Structures

struct	capture_status_t
struct	capture_callback_args_t
struct	capture_cfg_t
struct	capture_api_t
struct	capture_instance_t

Typedefs

typedef uint32_t	capture_event_t
typedef void	capture_ctrl_t

Enumerations

enum	capture_state_t

Data Structure Documentation

◆ capture_status_t

struct capture_status_t		
CAPTURE status		
Data Fields		
capture_state_t	state	Current capture state.
uint32_t *	p_buffer	Pointer to active buffer.
uint32_t	data_size	Size of data written to provided buffer.

◆ capture_callback_args_t

struct capture_callback_args_t		
CAPTURE callback function parameter data		
Data Fields		
capture_event_t	event	Event causing the callback - See instance header for details.

uint32_t	event_status	Capture status data corresponding to given event type - See instance header for details.
uint32_t	interrupt_status	Module interrupt status data corresponding to given event type - See instance header for details.
uint8_t *	p_buffer	Pointer to buffer that contains the most recently captured data.
void *	p_context	Placeholder for user data. Set in capture_api_t::open function in capture_cfg_t.

◆ capture_cfg_t

struct capture_cfg_t	
CAPTURE configuration parameters.	
Data Fields	
uint16_t	x_capture_start_pixel
	Horizontal position to start capture.
uint16_t	x_capture_pixels
	Number of horizontal pixels to capture.
uint16_t	y_capture_start_pixel
	Vertical position to start capture.
uint16_t	y_capture_pixels
	Number of vertical lines/pixels to capture.
uint8_t	bytes_per_pixel
	Number of bytes per pixel.
void(*	p_callback)(capture_callback_args_t *p_args)

	Callback provided when a CAPTURE transfer ISR occurs.
void *	p_context
	User defined context passed to callback function.
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ capture_api_t

struct capture_api_t	
CAPTURE functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg)
fsp_err_t(*	close)(capture_ctrl_t *const p_ctrl)
fsp_err_t(*	captureStart)(capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer)
fsp_err_t(*	callbackSet)(capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory)
fsp_err_t(*	statusGet)(capture_ctrl_t *const p_ctrl, capture_status_t *p_status)

Field Documentation

◆ open

fsp_err_t(* capture_api_t::open) (capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg)

Initial configuration.

Note

To reconfigure after calling this function, call capture_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* capture_api_t::close) (capture_ctrl_t *const p_ctrl)

Closes the driver and allows reconfiguration. May reduce power consumption.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ captureStart

fsp_err_t(* capture_api_t::captureStart) (capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer)

Start a capture.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buffer	New pointer to store captured image data.

◆ callbackSet

fsp_err_t(* capture_api_t::callbackSet) (capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the CAPTURE control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ statusGet

fsp_err_t(* capture_api_t::statusGet) (capture_ctrl_t *const p_ctrl, capture_status_t *p_status)

Check scan status.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_status	Pointer to store current status in

◆ capture_instance_t

struct capture_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
capture_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
capture_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
capture_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ capture_event_t

typedef uint32_t capture_event_t
CAPTURE callback event ID - see implementation for details

◆ capture_ctrl_t

typedef void capture_ctrl_t
CAPTURE control block. Allocate an instance specific control block to pass into the CAPTURE API calls.

Enumeration Type Documentation

◆ capture_state_t

enum capture_state_t	
CAPTURE states.	
Enumerator	
CAPTURE_STATE_IDLE	CAPTURE is idle.
CAPTURE_STATE_IN_PROGRESS	CAPTURE capture in progress.
CAPTURE_STATE_BUSY	CAPTURE reset in progress.

5.3.8 Input

Interfaces

Detailed Description

Input Interfaces.

Modules

External IRQ Interface	Interface for detecting external interrupts.

5.3.8.1 External IRQ Interface

Interfaces » Input

Detailed Description

Interface for detecting external interrupts.

Summary

The External IRQ Interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

Data Structures

struct	external_irq_callback_args_t
struct	external_irq_cfg_t
struct	external_irq_api_t
struct	external_irq_instance_t

Typedefs

typedef void	external_irq_ctrl_t

Enumerations

enum	external_irq_trigger_t
enum	external_irq_clock_source_div_t

Data Structure Documentation

◆ **external_irq_callback_args_t**

struct external_irq_callback_args_t		
Callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data. Set in external_irq_api_t::open function in external_irq_cfg_t.
uint32_t	channel	The physical hardware channel that caused the interrupt.

◆ **external_irq_cfg_t**

struct external_irq_cfg_t
User configuration structure, used in open function

Data Fields

uint8_t	channel
	Hardware channel used.
uint8_t	ipl
	Interrupt priority.
IRQn_Type	irq
	Interrupt number assigned to this instance.
external_irq_trigger_t	trigger
	Trigger setting.
external_irq_clock_source_div_t	clock_source_div
	Digital filter clock divisor setting.
bool	filter_enable
	Digital filter enable/disable selection.
void(*	p_callback)(external_irq_callback_args_t *p_args)
void *	p_context
void const *	p_extend
	External IRQ hardware dependent configuration.

Field Documentation

◆ p_callback

void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)

Callback provided external input trigger occurs.

◆ p_context

void* external_irq_cfg_t::p_context

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

◆ external_irq_api_t

struct external_irq_api_t

External interrupt driver structure. External interrupt functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
fsp_err_t(*	enable)(external_irq_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(external_irq_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(external_irq_ctrl_t *const p_ctrl, void(*p_callback)(external_irq_callback_args_t *), void *const p_context, external_irq_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(external_irq_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ enable

fsp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)

Enable callback when an external trigger condition occurs.

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ disable

fsp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)

Disable callback when external trigger condition occurs.

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ callbackSet

fsp_err_t(* external_irq_api_t::callbackSet) (external_irq_ctrl_t *const p_ctrl, void(*p_callback)(external_irq_callback_args_t *), void *const p_context, external_irq_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the External IRQ control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ external_irq_instance_t

struct external_irq_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
external_irq_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
external_irq_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
external_irq_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ external_irq_ctrl_t

typedef void external_irq_ctrl_t
External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls.

Enumeration Type Documentation

◆ external_irq_trigger_t

enum external_irq_trigger_t	
Condition that will trigger an interrupt when detected.	
Enumerator	
EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.
EXTERNAL_IRQ_TRIG_LEVEL_HIGH	High level trigger.
EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.
EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.

◆ external_irq_clock_source_div_t

enum external_irq_clock_source_div_t	
External IRQ input pin digital filtering sample clock divisor settings. The digital filter rejects trigger conditions that are shorter than 3 periods of the filter clock.	
Enumerator	
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_1	Filter using clock source divided by 1.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_8	Filter using clock source divided by 8.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_32	Filter using clock source divided by 32.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_64	Filter using clock source divided by 64.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_1	Filter using clock source divided by 1.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_8	Filter using clock source divided by 8.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_32	Filter using clock source divided by 32.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_64	Filter using clock source divided by 64.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_128	Filter using clock source divided by 128.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_4	Filter using clock source divided by 4.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_16	Filter using clock source divided by 16.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_1	Filter using clock source divided by 1.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_8	Filter using clock source divided by 8.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_32	Filter using clock source divided by 32.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_64	Filter using clock source divided by 64.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_128	Filter using clock source divided by 128.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_4	Filter using clock source divided by 4.
EXTERNAL_IRQ_CLOCK_SOURCE_DIV_16	Filter using clock source divided by 16.

5.3.9 Monitoring

Interfaces

Detailed Description

Monitoring Interfaces.

Modules

	CAC Interface Interface for clock frequency accuracy measurements.
	CRC Interface Interface for cyclic redundancy checking.
	DOC Interface Interface for the Data Operation Circuit.
	Voltage Detection Interface Interface for Voltage Detection.
	WDT Interface Interface for watch dog timer functions.

5.3.9.1 CAC Interface

Interfaces » Monitoring

Detailed Description

Interface for clock frequency accuracy measurements.

Summary

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of pulses of the clock to be measured.

Data Structures

struct	cac_ref_clock_config_t
struct	cac_meas_clock_config_t
struct	cac_callback_args_t
struct	cac_cfg_t
struct	cac_api_t
struct	cac_instance_t

Typedefs

typedef void	cac_ctrl_t

Enumerations

enum	cac_event_t
enum	cac_clock_type_t
enum	cac_clock_source_t
enum	cac_ref_divider_t
enum	cac_ref_digfilter_t
enum	cac_ref_edge_t
enum	cac_meas_divider_t

Data Structure Documentation

◆ cac_ref_clock_config_t

struct cac_ref_clock_config_t		
Structure defining the settings that apply to reference clock configuration.		
Data Fields		
cac_ref_divider_t	divider	Divider specification for the Reference clock.
cac_clock_source_t	clock	Clock source for the Reference clock.
cac_ref_digfilter_t	digfilter	Digital filter selection for the CACREF ext clock.
cac_ref_edge_t	edge	Edge detection for the Reference clock.

◆ cac_meas_clock_config_t

struct cac_meas_clock_config_t		
Structure defining the settings that apply to measurement clock configuration.		
Data Fields		
cac_meas_divider_t	divider	Divider specification for the Measurement clock.
cac_clock_source_t	clock	Clock source for the Measurement clock.

◆ cac_callback_args_t

struct cac_callback_args_t		
Callback function parameter data		
Data Fields		
cac_event_t	event	The event can be used to identify what caused the callback.
void *	p_context	Value provided in configuration structure.

◆ cac_cfg_t

struct cac_cfg_t	
CAC Configuration	
Data Fields	
cac_ref_clock_config_t	cac_ref_clock
	Reference clock specific settings.
cac_meas_clock_config_t	cac_meas_clock
	Measurement clock specific settings.
uint16_t	cac_upper_limit
	The upper limit counter threshold.
uint16_t	cac_lower_limit
	The lower limit counter threshold.

IRQn_Type	mendi_irq
	Measurement End IRQ number.
IRQn_Type	ovfi_irq
	Measurement Overflow IRQ number.
IRQn_Type	ferri_irq
	Frequency Error IRQ number.
uint8_t	mendi_ipl
	Measurement end interrupt priority.
uint8_t	ovfi_ipl
	Overflow interrupt priority.
uint8_t	ferri_ipl
	Frequency error interrupt priority.
void(*	p_callback)(cac_callback_args_t *p_args)
	Callback provided when a CAC interrupt ISR occurs.
void *	p_context
	Passed to user callback in cac_callback_args_t.
void const *	p_extend
	CAC hardware dependent configuration */.

◆ cac_api_t

struct cac_api_t

CAC functions implemented at the HAL layer API

Data Fields

fsp_err_t(*	open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
fsp_err_t(*	startMeasurement)(cac_ctrl_t *const p_ctrl)
fsp_err_t(*	stopMeasurement)(cac_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(cac_ctrl_t *const p_ctrl, uint32_t *const p_counter)
fsp_err_t(*	callbackSet)(cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void *const p_context, cac_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(cac_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

Open function for CAC device.

Parameters

[out]	p_ctrl	Pointer to CAC device control. Must be declared by user.
[in]	cac_cfg_t	Pointer to CAC configuration structure.

◆ startMeasurement

fsp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)

Begin a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ stopMeasurement

fsp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)

End a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ read

fsp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint32_t *const p_counter)

Read function for CAC peripheral.

Parameters

[in]	p_ctrl	Control for the CAC device context.
[in]	p_counter	Pointer to variable in which to store the current CACNTBR register contents.

◆ callbackSet

fsp_err_t(* cac_api_t::callbackSet) (cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void *const p_context, cac_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in cac_api_t::open call
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)

Close function for CAC device.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ cac_instance_t

struct cac_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
cac_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
cac_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
cac_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ cac_ctrl_t

typedef void cac_ctrl_t
CAC control block. Allocate an instance specific control block to pass into the CAC API calls.

Enumeration Type Documentation

◆ cac_event_t

enum cac_event_t	
Event types returned by the ISR callback when used in CAC interrupt mode	
Enumerator	
CAC_EVENT_FREQUENCY_ERROR	Frequency error.
CAC_EVENT_MEASUREMENT_COMPLETE	Measurement complete.
CAC_EVENT_COUNTER_OVERFLOW	Counter overflow.

◆ cac_clock_type_t

enum cac_clock_type_t	
Enumeration of the two possible clocks.	
Enumerator	
CAC_CLOCK_MEASURED	Measurement clock.
CAC_CLOCK_REFERENCE	Reference clock.

◆ cac_clock_source_t

enum cac_clock_source_t	
Enumeration of the possible clock sources for both the reference and measurement clocks.	
Enumerator	
CAC_CLOCK_SOURCE_MAIN_OSC	Main clock oscillator.
CAC_CLOCK_SOURCE_SUBCLOCK	Sub-clock.
CAC_CLOCK_SOURCE_HOCO	HOCO (High speed on chip oscillator)
CAC_CLOCK_SOURCE_MOCO	MOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_LOCO	LOCO (Low speed on chip oscillator)
CAC_CLOCK_SOURCE_PCLKB	PCLKB (Peripheral Clock B)
CAC_CLOCK_SOURCE_IWDT	IWDT-dedicated on-chip oscillator.
CAC_CLOCK_SOURCE_EXTERNAL	Externally supplied measurement clock on CACREF pin.
CAC_CLOCK_SOURCE_MAIN_OSC	Main clock oscillator.
CAC_CLOCK_SOURCE_SUBCLOCK	Sub-clock.
CAC_CLOCK_SOURCE_PLL1P	PLL1 output clock P (divided by 16)
CAC_CLOCK_SOURCE_PLL2P	PLL2 output clock P (divided by 16)
CAC_CLOCK_SOURCE_HOCO	HOCO (High speed on chip oscillator)
CAC_CLOCK_SOURCE_MOCO	MOCO (Middle speed on chip oscillator)

CAC_CLOCK_SOURCE_LOCO	LOCO (Low speed on chip oscillator)
CAC_CLOCK_SOURCE_PLL1Q	PLL1 output clock Q (divided by 16)
CAC_CLOCK_SOURCE_PLL2Q	PLL2 output clock Q (divided by 16)
CAC_CLOCK_SOURCE_PCLKB	PCLKB (Peripheral Clock B)
CAC_CLOCK_SOURCE_PLL1R	PLL1 output clock R (divided by 16)
CAC_CLOCK_SOURCE_PLL2R	PLL2 output clock R (divided by 16)
CAC_CLOCK_SOURCE_EXTERNAL	Externally supplied measurement clock on CACREF pin.
CAC_CLOCK_SOURCE_MAIN_OSC	Main clock oscillator.
CAC_CLOCK_SOURCE_SUBCLOCK	Sub-clock.
CAC_CLOCK_SOURCE_PLL1P	PLL1 output clock P (divided by 16)
CAC_CLOCK_SOURCE_PLL2P	PLL2 output clock P (divided by 16)
CAC_CLOCK_SOURCE_HOCO	HOCO (High speed on chip oscillator)
CAC_CLOCK_SOURCE_MOCO	MOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_LOCO	LOCO (Low speed on chip oscillator)
CAC_CLOCK_SOURCE_PLL1Q	PLL1 output clock Q (divided by 16)
CAC_CLOCK_SOURCE_PLL2Q	PLL2 output clock Q (divided by 16)
CAC_CLOCK_SOURCE_PCLKB	PCLKB (Peripheral Clock B)
CAC_CLOCK_SOURCE_PLL1R	PLL1 output clock R (divided by 16)
CAC_CLOCK_SOURCE_PLL2R	PLL2 output clock R (divided by 16)
CAC_CLOCK_SOURCE_EXTERNAL	Externally supplied measurement clock on CACREF pin.

◆ cac_ref_divider_t

enum cac_ref_divider_t	
Enumeration of available dividers for the reference clock.	
Enumerator	
CAC_REF_DIV_32	Reference clock divided by 32.
CAC_REF_DIV_128	Reference clock divided by 128.
CAC_REF_DIV_1024	Reference clock divided by 1024.
CAC_REF_DIV_8192	Reference clock divided by 8192.

◆ cac_ref_digfilter_t

enum cac_ref_digfilter_t	
Enumeration of available digital filter settings for an external reference clock.	
Enumerator	
CAC_REF_DIGITAL_FILTER_OFF	No digital filter on the CACREF pin for reference clock.
CAC_REF_DIGITAL_FILTER_1	Sampling clock for digital filter = measuring frequency.
CAC_REF_DIGITAL_FILTER_4	Sampling clock for digital filter = measuring frequency/4.
CAC_REF_DIGITAL_FILTER_16	Sampling clock for digital filter = measuring frequency/16.

◆ cac_ref_edge_t

enum cac_ref_edge_t	
Enumeration of available edge detect settings for the reference clock.	
Enumerator	
CAC_REF_EDGE_RISE	Rising edge detect for the Reference clock.
CAC_REF_EDGE_FALL	Falling edge detect for the Reference clock.
CAC_REF_EDGE_BOTH	Both Rising and Falling edges detect for the Reference clock.

◆ cac_meas_divider_t

enum cac_meas_divider_t	
Enumeration of available dividers for the measurement clock	
Enumerator	
CAC_MEAS_DIV_1	Measurement clock divided by 1.
CAC_MEAS_DIV_4	Measurement clock divided by 4.
CAC_MEAS_DIV_8	Measurement clock divided by 8.
CAC_MEAS_DIV_32	Measurement clock divided by 32.

5.3.9.2 CRC Interface

Interfaces » Monitoring

Detailed Description

Interface for cyclic redundancy checking.

Summary

The CRC (Cyclic Redundancy Check) calculator generates CRC codes using five different polynomials including 8 bit, 16 bit, and 32 bit variations. Calculation can be performed by sending data to the block using the CPU or by snooping on read or write activity on one of SCI channels.

Data Structures

struct	crc_input_t
struct	crc_cfg_t
struct	crc_api_t
struct	crc_instance_t

Typedefs

typedef void	crc_ctrl_t

Enumerations

enum	crc_polynomial_t
enum	crc_bit_order_t
enum	crc_snoop_direction_t

Data Structure Documentation

◆ crc_input_t

struct crc_input_t		
Structure for CRC inputs		
Data Fields		
uint32_t	num_bytes	Length of input buffer. It must be 4-byte aligned when a 32-bit CRC polynomial function is used.
uint32_t	crc_seed	CRC seed value.
const void *	p_input_buffer	Pointer to input buffer.

◆ crc_cfg_t

struct crc_cfg_t		
User configuration structure, used in open function		
Data Fields		
uint8_t	channel	Channel number.
crc_polynomial_t	polynomial	CRC Generating Polynomial Switching (GPS)
crc_bit_order_t	bit_order	CRC Calculation Switching (LMS)
uint32_t	snoop_address	Register Snoop Address (CRCSA)

void const *	p_extend	CRC Hardware Dependent Configuration.

◆ crc_api_t

struct crc_api_t

CRC driver structure. General CRC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(crc_ctrl_t *const p_ctrl)
fsp_err_t(*	crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
fsp_err_t(*	snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
fsp_err_t(*	snoopDisable)(crc_ctrl_t *const p_ctrl)
fsp_err_t(*	calculate)(crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)

Field Documentation

◆ open

fsp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)

Open the CRC driver module.

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	p_cfg	Pointer to a configuration structure.

◆ close

fsp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)

Close the CRC module driver

Parameters

[in]	p_ctrl	Pointer to CRC device handle

Return values

FSP_SUCCESS	Configuration was successful.

◆ crcResultGet

fsp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)

Return the current calculated value.

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[out]	crc_result	The calculated value from the last CRC calculation.

◆ snoopEnable

fsp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

Configure and Enable snooping.

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	crc_seed	CRC seed.

◆ snoopDisable

fsp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)

Disable snooping.

Parameters

[in]	p_ctrl	Pointer to CRC device handle.

◆ calculate

fsp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)

Perform a CRC calculation on a block of data.

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	p_crc_input	A pointer to structure for CRC inputs
[out]	crc_result	The calculated value of the CRC calculation.

◆ crc_instance_t

struct crc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
crc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
crc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
crc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ crc_ctrl_t

typedef void crc_ctrl_t

CRC control block. Allocate an instance specific control block to pass into the CRC API calls.

Enumeration Type Documentation

◆ crc_polynomial_t

enum crc_polynomial_t	
CRC Generating Polynomial Switching (GPS).	
Enumerator	
CRC_POLYNOMIAL_CRC_8	8-bit CRC-8 (X^8 + X^2 + X + 1)
CRC_POLYNOMIAL_CRC_16	16-bit CRC-16 (X^16 + X^15 + X^2 + 1)
CRC_POLYNOMIAL_CRC_CCITT	16-bit CRC-CCITT (X^16 + X^12 + X^5 + 1)
CRC_POLYNOMIAL_CRC_32	32-bit CRC-32 (X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1)
CRC_POLYNOMIAL_CRC_32C	32-bit CRC-32C (X^32 + X^28 + X^27 + X^26 + X^25 + X^23 + X^22 + X^20 + X^19 + X^18 + X^14 + X^13 + X^11 + X^10 + X^9 + X^8 + X^6 + 1)

◆ crc_bit_order_t

enum crc_bit_order_t	
CRC Calculation Switching (LMS)	
Enumerator	
CRC_BIT_ORDER_LMS_LSB	Generates CRC for LSB first communication.
CRC_BIT_ORDER_LMS_MSB	Generates CRC for MSB first communication.

◆ crc_snoop_direction_t

enum crc_snoop_direction_t	
Snoop-On-Write/Read Switch (CRCSWR)	
Enumerator	
CRC_SNOOP_DIRECTION_RECEIVE	Snoop-on-read.
CRC_SNOOP_DIRECTION_TRANSMIT	Snoop-on-write.

5.3.9.3 DOC Interface

Interfaces » Monitoring

Detailed Description

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

Data Structures

struct	doc_callback_args_t
struct	doc_cfg_t
struct	doc_api_t
struct	doc_instance_t

Typedefs

typedef void	doc_ctrl_t

Enumerations

enum	doc_event_t
enum	doc_bit_width_t

Data Structure Documentation

◆ doc_callback_args_t

struct doc_callback_args_t		
Callback function parameter data.		
Data Fields		
void *	p_context	Set in doc_api_t::open function in doc_cfg_t. Placeholder for user data.

◆ doc_cfg_t

struct doc_cfg_t

User configuration structure, used in the open function.

Data Fields

doc_event_t	event
	Select enumerated value from doc_event_t.
doc_bit_width_t	bit_width
	The bit width of operations.
uint32_t	doc_data
uint32_t	doc_data_extra
uint8_t	ipl
	DOC interrupt priority.
IRQn_Type	irq
	Interrupt number assigned to this instance.
void(*	p_callback)(doc_callback_args_t *p_args)
void *	p_context

Field Documentation

◆ doc_data

uint32_t doc_cfg_t::doc_data

Initial/Reference data for addition, subtraction, and comparison operations.

- In Addition and Subtraction mode, this value sets the initial value of the operations.
- In Comparison match, mismatch, lower, and upper modes, this value is compared with data that is written.
- In Comparison inside window and outside window modes, this value is used as the lower bound for comparisons.

◆ doc_data_extra

uint32_t doc_cfg_t::doc_data_extra

Additional reference data for use in Window Comparison operations.

- In Comparison inside window and outside window modes, this value is used as the upper bound for comparisons.

◆ p_callback

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)

Callback provided when a DOC ISR occurs.

◆ p_context

void* doc_cfg_t::p_context

Placeholder for user data. Passed to the user callback in doc_callback_args_t.

◆ doc_api_t

struct doc_api_t

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(doc_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(doc_ctrl_t *const p_ctrl, uint32_t *p_result)
fsp_err_t(*	write)(doc_ctrl_t *const p_ctrl, uint32_t data)
fsp_err_t(*	callbackSet)(doc_ctrl_t *const p_ctrl, void(*p_callback)(doc_callback_args_t *), void *const p_context, doc_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

fsp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)

Allow the driver to be reconfigured. Will reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.

◆ read

fsp_err_t(* doc_api_t::read) (doc_ctrl_t *const p_ctrl, uint32_t *p_result)

Gets the result of addition/subtraction operations and stores it in the provided pointer p_result.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	p_result	The result of the DOC operation.

◆ write

fsp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, uint32_t data)

Write to the DODIR register.

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	data	data to be written to DOC DODIR register.

◆ callbackSet

fsp_err_t(* doc_api_t::callbackSet) (doc_ctrl_t *const p_ctrl, void(*p_callback)(doc_callback_args_t *), void *const p_context, doc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the DOC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ doc_instance_t

struct doc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
doc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
doc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
doc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ doc_ctrl_t

typedef void doc_ctrl_t
DOC control block. Allocate an instance specific control block to pass into the DOC API calls.

Enumeration Type Documentation

◆ doc_event_t

enum doc_event_t	
Event that can trigger a callback function.	
Enumerator	
DOC_EVENT_COMPARISON_MISMATCH	The data is not equal to the reference data setting.
DOC_EVENT_ADDITION	Addition resulted in a value greater than the max for the configured bit width.
DOC_EVENT_SUBTRACTION	Subtraction resulted in a value less than 0.
DOC_EVENT_COMPARISON_MATCH	The data is equal to the reference data setting.
DOC_EVENT_COMPARISON_LOWER	The data is less than the reference data setting.
DOC_EVENT_COMPARISON_UPPER	The data is greater than the reference data setting.
DOC_EVENT_COMPARISON_INSIDE_WINDOW	The data is between the two reference data settings.
DOC_EVENT_COMPARISON_OUTSIDE_WINDOW	The data is outside the two reference data setttings.

◆ doc_bit_width_t

enum doc_bit_width_t	
The bit width used during operations.	
Enumerator	
DOC_BIT_WIDTH_16	Operations are 16-bit.
DOC_BIT_WIDTH_32	Operations are 32-bit.

5.3.9.4 Voltage Detection Interface

Interfaces » Monitoring

Detailed Description

Interface for Voltage Detection.

Summary

The VOLTAGE DETECTION driver provides functions for configuring the VOLTAGE DETECTION voltage monitors and detectors.

Data Structures

struct	voltage_detection_status_t
struct	voltage_detection_callback_args_t
struct	voltage_detection_cfg_t
struct	voltage_detection_api_t
struct	voltage_detection_instance_t

Typedefs

typedef void	voltage_detection_ctrl_t

Enumerations

enum	voltage_detection_threshold_t
enum	voltage_detection_response_t
enum	voltage_detection_voltage_slope_t
enum	voltage_detection_sample_clock_t
enum	voltage_detection_negation_delay_t
enum	voltage_detection_threshold_crossing_t
enum	voltage_detection_current_state_t

Data Structure Documentation

◆ voltage_detection_status_t

struct voltage_detection_status_t		
Current state of a voltage monitor.		
Data Fields		
voltage_detection_threshold_crossing_t	crossing_detected	Threshold crossing detection (latched)
voltage_detection_current_state_t	current_state	Instantaneous status of monitored voltage (above or below threshold)

◆ voltage_detection_callback_args_t

struct voltage_detection_callback_args_t		
VOLTAGE DETECTION callback parameter definition		
Data Fields		
uint32_t	monitor_number	Monitor number.
voltage_detection_current_state_t	current_state	Current state of the voltage monitor.
void *	p_context	Placeholder for user data.

◆ voltage_detection_cfg_t

struct voltage_detection_cfg_t	
VOLTAGE DETECTION configuration structure	
Data Fields	
uint32_t	monitor_number
voltage_detection_threshold_t	voltage_threshold
voltage_detection_response_t	detection_response
voltage_detection_voltage_slope_t	voltage_slope
voltage_detection_negation_delay_t	negation_delay
voltage_detection_sample_clock_t	sample_clock_divisor

IRQn_Type	irq
uint8_t	monitor_ipl
void(*	p_callback)(voltage_detection_callback_args_t *p_args)
void *	p_context
void const *	p_extend

Field Documentation

◆ monitor_number

uint32_t voltage_detection_cfg_t::monitor_number

Monitor number, 1, 2, ...

◆ voltage_threshold

voltage_detection_threshold_t voltage_detection_cfg_t::voltage_threshold

Threshold for out of range voltage detection

◆ detection_response

voltage_detection_response_t voltage_detection_cfg_t::detection_response

Response on detecting a threshold crossing

◆ voltage_slope

voltage_detection_voltage_slope_t voltage_detection_cfg_t::voltage_slope

Direction of voltage crossing that will trigger a detection (Rising Edge, Falling Edge, Both).

◆ negation_delay

voltage_detection_negation_delay_t voltage_detection_cfg_t::negation_delay

Negation of VOLTAGE DETECTION signal follows reset or voltage in range

◆ sample_clock_divisor

voltage_detection_sample_clock_t voltage_detection_cfg_t::sample_clock_divisor

Sample clock divider, use VOLTAGE DETECTION_SAMPLE_CLOCK_DISABLED to disable digital filtering

◆ irq

IRQn_Type voltage_detection_cfg_t::irq

Interrupt number.

◆ monitor_ipl

uint8_t voltage_detection_cfg_t::monitor_ipl

Interrupt priority level.

◆ p_callback

void(* voltage_detection_cfg_t::p_callback) (voltage_detection_callback_args_t *p_args)

User function to be called from interrupt

◆ p_context

void* voltage_detection_cfg_t::p_context

Placeholder for user data. Passed to the user callback in

◆ p_extend

void const* voltage_detection_cfg_t::p_extend

Extension parameter for hardware specific settings

◆ voltage_detection_api_t

struct voltage_detection_api_t

VOLTAGE_DETECTION driver API structure. VOLTAGE_DETECTION driver functions implemented at the HAL layer will adhere to this API.

Data Fields

fsp_err_t(*	open)(voltage_detection_ctrl_t *const p_ctrl, voltage_detection_cfg_t const *const p_cfg)
fsp_err_t(*	statusGet)(voltage_detection_ctrl_t *const p_ctrl, voltage_detection_status_t *p_voltage_detection_status)
fsp_err_t(*	statusClear)(voltage_detection_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(voltage_detection_ctrl_t *const p_ctrl, void(*p_callback)(voltage_detection_callback_args_t *), void *const p_context, voltage_detection_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(voltage_detection_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* voltage_detection_api_t::open) (voltage_detection_ctrl_t *const p_ctrl, voltage_detection_cfg_t const *const p_cfg)

Initializes a programmable voltage detection driver according to the passed-in configuration structure.

Parameters

[in]	p_ctrl	Pointer to control structure for the driver instance
[in]	p_cfg	Pointer to the configuration structure for the driver instance

◆ statusGet

fsp_err_t(* voltage_detection_api_t::statusGet) (voltage_detection_ctrl_t *const p_ctrl, voltage_detection_status_t *p_voltage_detection_status)

Get the current state of the monitor, (threshold crossing detected, voltage currently above or below threshold). Must be used if the peripheral was initialized with voltage_detection_response_t set to VOLTAGE_DETECTION_RESPONSE_NONE.

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance
[in,out]	p_voltage_detection_status	Pointer to a voltage_detection_status_t structure

◆ statusClear

fsp_err_t(* voltage_detection_api_t::statusClear) (voltage_detection_ctrl_t *const p_ctrl)

Clears the latched status of the monitor. Must be used if the peripheral was initialized with voltage_detection_response_t set to VOLTAGE_DETECTION_RESPONSE_NONE.

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ callbackSet

fsp_err_t(* voltage_detection_api_t::callbackSet) (voltage_detection_ctrl_t *const p_ctrl, void(*p_callback)(voltage_detection_callback_args_t *), void *const p_context, voltage_detection_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the VOLTAGE_DETECTION control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* voltage_detection_api_t::close) (voltage_detection_ctrl_t *const p_ctrl)

Disables the VOLTAGE_DETECTION peripheral. Closes the driver instance.

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ voltage_detection_instance_t

struct voltage_detection_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

voltage_detection_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
voltage_detection_cfg_t const *	p_cfg	Pointer to the configuration structure for this interface instance.
voltage_detection_api_t const *	p_api	Pointer to the API structure for this interface instance.

Typedef Documentation

◆ voltage_detection_ctrl_t

typedef void voltage_detection_ctrl_t
VOLTAGE DETECTION control block. Allocate an instance specific control block to pass into the VOLTAGE_DETECTION API calls.

Enumeration Type Documentation

◆ voltage_detection_threshold_t

enum voltage_detection_threshold_t	
Register definitions, common services, and error codes. Voltage detection level The thresholds supported by each MCU are in the MCU User's Manual as well as in the r_voltage_detection module description on the stack tab of the RA project.	
Enumerator	
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_4_29V	4.29V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_4_16V	4.16V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_4_14V	4.14V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_4_03V	4.03V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_4_02V	4.02V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_3_86V	3.86V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_3_84V	3.84V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_3_10V	3.10V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_3_00V	3.00V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_90V	2.90V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_79V	2.79V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_68V	2.68V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_58V	2.58V

VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_48V	2.48V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_20V	2.20V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_1_96V	1.96V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_1_86V	1.86V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_1_75V	1.75V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_1_65V	1.65V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_99V	2.99V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_92V	2.92V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_1_LEVEL_2_85V	2.85V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_4_31V	4.31V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_4_29V	4.29V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_4_17V	4.17V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_4_14V	4.14V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_4_03V	4.03V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_4_02V	4.02V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_3_84V	3.84V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_2_99V	2.99V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_2_92V	2.92V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_2_LEVEL_2_85V	2.85V
VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_3_1V	3.1V
VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_2_9V	2.9V

VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_2_8V	2.8V
VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_2_7V	2.7V
VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_2_6V	2.6V
VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_2_4V	2.4V
VOLTAGE_DETECTION_THRESHOLD_EXPVDVBAT_LEVEL_2_2V	2.2V
VOLTAGE_DETECTION_THRESHOLD_PVDVRTC_LEVEL_2_8V	2.8V
VOLTAGE_DETECTION_THRESHOLD_PVDVRTC_LEVEL_2_6V	2.6V
VOLTAGE_DETECTION_THRESHOLD_PVDVRTC_LEVEL_2_4V	2.4V
VOLTAGE_DETECTION_THRESHOLD_PVDVRTC_LEVEL_2_2V	2.2V
VOLTAGE_DETECTION_THRESHOLD_NOT_AVAILABLE	Not Used.
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_86V	3.86V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_14V	3.14V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_10V	3.10V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_08V	3.08V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_85V	2.85V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_83V	2.83V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_80V	2.80V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_62V	2.62V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_33V	2.33V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_90V	1.90V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_86V	1.86V

VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_74V	1.74V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_71V	1.71V
VOLTAGE_DETECTION_THRESHOLD_NOT_AVAILABLE	Not Used.
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_86V	3.86V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_14V	3.14V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_10V	3.10V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_3_08V	3.08V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_85V	2.85V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_83V	2.83V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_80V	2.80V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_62V	2.62V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_2_33V	2.33V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_90V	1.90V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_86V	1.86V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_74V	1.74V
VOLTAGE_DETECTION_THRESHOLD_MONITOR_LEVEL_1_71V	1.71V
VOLTAGE_DETECTION_THRESHOLD_NOT_AVAILABLE	Not Used.

◆ voltage_detection_response_t

enum voltage_detection_response_t	
Response types for handling threshold crossing event.	
Enumerator	
VOLTAGE_DETECTION_RESPONSE_NMI	Non-maskable interrupt.
VOLTAGE_DETECTION_RESPONSE_INTERRUPT	Maskable interrupt.
VOLTAGE_DETECTION_RESPONSE_RESET	Reset on VCC-fall.
VOLTAGE_DETECTION_RESPONSE_RESET_ON_RISING	Reset on VCC-rise.
VOLTAGE_DETECTION_RESPONSE_NONE	No response, status must be requested via statusGet function.

◆ voltage_detection_voltage_slope_t

enum voltage_detection_voltage_slope_t	
The direction from which VCC must cross the threshold to trigger a detection (rising, falling, or both).	
Enumerator	
VOLTAGE_DETECTION_VOLTAGE_SLOPE_RISING	When VCC >= Vdet2 (rise) is detected.
VOLTAGE_DETECTION_VOLTAGE_SLOPE_FALLING	When VCC < Vdet2 (drop) is detected.
VOLTAGE_DETECTION_VOLTAGE_SLOPE_BOTH	When drop and rise are detected.

◆ voltage_detection_sample_clock_t

enum voltage_detection_sample_clock_t	
Sample clock divider, use VOLTAGE_DETECTION_SAMPLE_CLOCK_DISABLED to disable digital filtering	
Enumerator	
VOLTAGE_DETECTION_SAMPLE_CLOCK_LOCO_DIV_2	Digital filter sample clock is LOCO divided by 2.
VOLTAGE_DETECTION_SAMPLE_CLOCK_LOCO_DIV_4	Digital filter sample clock is LOCO divided by 4.
VOLTAGE_DETECTION_SAMPLE_CLOCK_LOCO_DIV_8	Digital filter sample clock is LOCO divided by 8.
VOLTAGE_DETECTION_SAMPLE_CLOCK_LOCO_DIV_16	Digital filter sample clock is LOCO divided by 16.
VOLTAGE_DETECTION_SAMPLE_CLOCK_DISABLED	Digital filter is disabled.

◆ voltage_detection_negation_delay_t

enum voltage_detection_negation_delay_t	
Negation delay of VOLTAGE DETECTION reset signal follows reset or voltage in range	
Enumerator	
VOLTAGE_DETECTION_NEGATION_DELAY_FROM_VOLTAGE	Negation follows a stabilization time (tVDn) after VCC > Vdet1 is detected. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is VOLTAGE_DETECTION_NEGATION_DELAY_FROM_VOLTAGE
VOLTAGE_DETECTION_NEGATION_DELAY_FROM_RESET	Negation follows a stabilization time (tVDn) after assertion of the VDn reset. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is VOLTAGE_DETECTION_NEGATION_DELAY_FROM_VOLTAGE

◆ voltage_detection_threshold_crossing_t

enum voltage_detection_threshold_crossing_t	
Threshold crossing detection (latched)	
Enumerator	
VOLTAGE_DETECTION_THRESHOLD_CROSSING_NOT_DETECTED	Threshold crossing has not been detected.
VOLTAGE_DETECTION_THRESHOLD_CROSSING_DETECTED	Threshold crossing has been detected.

◆ voltage_detection_current_state_t

enum voltage_detection_current_state_t	
Instantaneous status of VCC (above or below threshold)	
Enumerator	
VOLTAGE_DETECTION_CURRENT_STATE_BELOW_THRESHOLD	VCC < threshold.
VOLTAGE_DETECTION_CURRENT_STATE_ABOVE_THRESHOLD	VCC >= threshold or monitor is disabled.

5.3.9.5 WDT Interface

Interfaces » Monitoring

Detailed Description

Interface for watch dog timer functions.

Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

Data Structures

struct	wdt_callback_args_t
struct	wdt_timeout_values_t
struct	wdt_cfg_t

struct	wdt_api_t
struct	wdt_instance_t

Typedefs

typedef void	wdt_ctrl_t

Enumerations

enum	wdt_timeout_t
enum	wdt_clock_division_t
enum	wdt_window_start_t
enum	wdt_window_end_t
enum	wdt_reset_control_t
enum	wdt_stop_control_t
enum	wdt_status_t

Data Structure Documentation

◆ wdt_callback_args_t

struct wdt_callback_args_t		
Callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t.

◆ wdt_timeout_values_t

struct wdt_timeout_values_t		
WDT timeout data. Used to return frequency of WDT clock and timeout period		
Data Fields		
uint32_t	clock_frequency_hz	Frequency of watchdog clock after divider.
uint32_t	timeout_clocks	Timeout period in units of watchdog clock ticks.

◆ wdt_cfg_t

struct wdt_cfg_t

WDT configuration parameters.

Data Fields

wdt_timeout_t	timeout
	Timeout period.
wdt_clock_division_t	clock_division
	Clock divider.
wdt_window_start_t	window_start
	Refresh permitted window start position.
wdt_window_end_t	window_end
	Refresh permitted window end position.
wdt_reset_control_t	reset_control
	Select NMI/IRQ or reset generated on underflow.
wdt_stop_control_t	stop_control
	Select whether counter operates in sleep mode.
void(*	p_callback)(wdt_callback_args_t *p_args)
	Callback provided when a WDT ISR occurs.
void *	p_context
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_context

void* wdt_cfg_t::p_context

Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

◆ wdt_api_t

struct wdt_api_t

WDT functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
fsp_err_t(*	refresh)(wdt_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
fsp_err_t(*	statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
fsp_err_t(*	counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
fsp_err_t(*	timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
fsp_err_t(*	callbackSet)(wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

Initialize the WDT in register start mode. In auto-start mode (Supported devices only) with NMI output it registers the NMI callback.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ refresh

fsp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)

Refresh the watchdog timer.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ statusGet

fsp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)

Read the status of the WDT.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_status	Pointer to variable to return status information through.

◆ statusClear

fsp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)

Clear the status flags of the WDT.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	status	Status condition(s) to clear.

◆ counterGet

fsp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)

Read the current WDT counter value.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_count	Pointer to variable to return current WDT counter value.

◆ timeoutGet

fsp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)

Read the watchdog timeout values.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_timeout	Pointer to structure to return timeout values.

◆ callbackSet

fsp_err_t(* wdt_api_t::callbackSet) (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the WDT control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_callback_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ wdt_instance_t

struct wdt_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
wdt_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
wdt_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
wdt_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ wdt_ctrl_t

typedef void wdt_ctrl_t
WDT control block. Allocate an instance specific control block to pass into the WDT API calls.

Enumeration Type Documentation

◆ wdt_timeout_t

enum wdt_timeout_t	
WDT time-out periods.	
Enumerator	
WDT_TIMEOUT_128	128 clock cycles
WDT_TIMEOUT_512	512 clock cycles
WDT_TIMEOUT_1024	1024 clock cycles
WDT_TIMEOUT_2048	2048 clock cycles
WDT_TIMEOUT_4096	4096 clock cycles
WDT_TIMEOUT_8192	8192 clock cycles
WDT_TIMEOUT_16384	16384 clock cycles

◆ wdt_clock_division_t

enum wdt_clock_division_t	
WDT clock division ratio.	
Enumerator	
WDT_CLOCK_DIVISION_1	CLK/1.
WDT_CLOCK_DIVISION_4	CLK/4.
WDT_CLOCK_DIVISION_16	CLK/16.
WDT_CLOCK_DIVISION_32	CLK/32.
WDT_CLOCK_DIVISION_64	CLK/64.
WDT_CLOCK_DIVISION_128	CLK/128.
WDT_CLOCK_DIVISION_256	CLK/256.
WDT_CLOCK_DIVISION_512	CLK/512.
WDT_CLOCK_DIVISION_2048	CLK/2048.
WDT_CLOCK_DIVISION_8192	CLK/8192.

◆ wdt_window_start_t

enum wdt_window_start_t	
WDT refresh permitted period window start position.	
Enumerator	
WDT_WINDOW_START_25	Start position = 25%.
WDT_WINDOW_START_50	Start position = 50%.
WDT_WINDOW_START_75	Start position = 75%.
WDT_WINDOW_START_100	Start position = 100%.

◆ wdt_window_end_t

enum wdt_window_end_t	
WDT refresh permitted period window end position.	
Enumerator	
WDT_WINDOW_END_75	End position = 75%.
WDT_WINDOW_END_50	End position = 50%.
WDT_WINDOW_END_25	End position = 25%.
WDT_WINDOW_END_0	End position = 0%.

◆ wdt_reset_control_t

enum wdt_reset_control_t	
WDT Counter underflow and refresh error control.	
Enumerator	
WDT_RESET_CONTROL_NMI	NMI/IRQ request when counter underflows.
WDT_RESET_CONTROL_RESET	Reset request when counter underflows.

◆ wdt_stop_control_t

enum wdt_stop_control_t	
WDT Counter operation in sleep mode.	
Enumerator	
WDT_STOP_CONTROL_DISABLE	Count will not stop when device enters sleep mode.
WDT_STOP_CONTROL_ENABLE	Count will automatically stop when device enters sleep mode.

◆ wdt_status_t

enum wdt_status_t	
WDT status	
Enumerator	
WDT_STATUS_NO_ERROR	No status flags set.
WDT_STATUS_UNDERFLOW	Underflow flag set.
WDT_STATUS_REFRESH_ERROR	Refresh error flag set. Refresh outside of permitted window.
WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR	Underflow and refresh error flags set.
WDT_STATUS_OVERFLOW	Overflow flag set.

5.3.10 Motor

Interfaces

Detailed Description

Motor Interfaces.

Modules

	Motor angle Interface Interface for motor angle and speed calculation functions.
	Motor Interface Interface for Motor functions.
	Motor current Interface Interface for motor current functions.
	Motor driver Interface Interface for motor driver functions.

	Motor Inertia Estimate Interface Interface for Motor inertia estimate functions.
	Motor position Interface Interface for motor position functions.
	Motor Return Origin Function Interface Interface for Motor return origin functions.
	Motor speed Interface Interface for motor speed functions.

5.3.10.1 Motor angle Interface

Interfaces » Motor

Detailed Description

Interface for motor angle and speed calculation functions.

Summary

The Motor angle interface calculates the rotor angle and rotational speed from other data.

Data Structures

struct	motor_angle_cfg_t
struct	motor_angle_current_t
struct	motor_angle_voltage_reference_t
struct	motor_angle_ad_data_t
struct	motor_angle_encoder_info_t
struct	motor_angle_api_t
struct	motor_angle_instance_t

Typedefs

typedef void	motor_angle_ctrl_t

Enumerations

enum	motor_sense_encoder_angle_adjust_t
enum	motor_angle_open_loop_t
enum	motor_angle_error_t

Data Structure Documentation

◆ motor_angle_cfg_t

struct motor_angle_cfg_t
Configuration parameters.

◆ motor_angle_current_t

struct motor_angle_current_t		
Interface structure		
Data Fields		
float	id	d-axis current
float	iq	q-axis current

◆ motor_angle_voltage_reference_t

struct motor_angle_voltage_reference_t		
Motor angle voltage reference		
Data Fields		
float	vd	d-axis voltage reference
float	vq	q-axis voltage reference

◆ motor_angle_ad_data_t

struct motor_angle_ad_data_t		
A/D conversion data		
Data Fields		
float	sin_ad_data	sin A/D data of induction sensor
float	cos_ad_data	cos A/D data of induction sensor

◆ motor_angle_encoder_info_t

struct motor_angle_encoder_info_t

Motor angle encoder adjustment info		
Data Fields		
motor_sense_encoder_angle_adjust_t	e_adjust_status	Encoder Adjustment Status.
uint8_t	u1_adjust_count_full	Adjustment count became full.
uint8_t	u1_adjust_mode	Adjustment mode (only valid with ENCODER)
motor_angle_open_loop_t	e_open_loop_status	Openloop status.
float	f_openloop_speed	Openloop speed.
float	f_openloop_id_ref	Openloop d-axis current.

◆ motor_angle_api_t

struct motor_angle_api_t	
Functions implemented as application interface will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	currentSet)(motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)
fsp_err_t(*	speedSet)(motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)
fsp_err_t(*	flagPiCtrlSet)(motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)
fsp_err_t(*	internalCalculate)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	angleSpeedGet)(motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)

fsp_err_t(*	angleAdjust)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	encoderCyclic)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	cyclicProcess)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	sensorDataSet)(motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data)
fsp_err_t(*	estimatedComponentGet)(motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)
fsp_err_t(*	infoGet)(motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)
fsp_err_t(*	parameterUpdate)(motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_angle_api_t::open) (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)

Initialize the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_angle_api_t::close) (motor_angle_ctrl_t *const p_ctrl)

Close (Finish) the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_angle_api_t::reset) (motor_angle_ctrl_t *const p_ctrl)

Reset the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ currentSet

fsp_err_t(* motor_angle_api_t::currentSet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)

Set (Input) Current & Voltage Reference data into the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_current	Pointer to current structure
[in]	p_st_voltage	Pointer to voltage Reference structure

◆ speedSet

fsp_err_t(* motor_angle_api_t::speedSet) (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)

Set (Input) Speed Information into the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_ctrl	Control reference of rotational speed [rad/s]
[in]	damp_speed	Damping rotational speed [rad/s]

◆ flagPiCtrlSet

fsp_err_t(* motor_angle_api_t::flagPiCtrlSet) (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)

Set the flag of PI Control runs.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	flag_pi	The flag of PI control runs

◆ internalCalculate

fsp_err_t(* motor_angle_api_t::internalCalculate) (motor_angle_ctrl_t *const p_ctrl)

Calculate internal parameters of encoder process.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ angleSpeedGet

fsp_err_t(* motor_angle_api_t::angleSpeedGet) (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)

Get rotor angle and rotational speed from the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_angl	Memory address to get rotor angle data
[out]	p_speed	Memory address to get rotational speed data
[out]	p_phase_err	Memory address to get phase(angle) error data

◆ angleAdjust

fsp_err_t(* motor_angle_api_t::angleAdjust) (motor_angle_ctrl_t *const p_ctrl)

Angle Adjustment Process.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ encoderCyclic

fsp_err_t(* motor_angle_api_t::encoderCyclic) (motor_angle_ctrl_t *const p_ctrl)

DEPRECATED Encoder Cyclic Process.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ cyclicProcess

fsp_err_t(* motor_angle_api_t::cyclicProcess) (motor_angle_ctrl_t *const p_ctrl)

Cyclic Process. please

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ sensorDataSet

fsp_err_t(* motor_angle_api_t::sensorDataSet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data)

Set sensor A/D data into the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_ad_data	Pointer to A/D conversion data

◆ estimatedComponentGet

fsp_err_t(* motor_angle_api_t::estimatedComponentGet) (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)

Get estimated d/q-axis component from the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_ed	Memory address to get estimated d-axis component
[out]	p_eq	Memory address to get estimated q-axis component

◆ infoGet

fsp_err_t(* motor_angle_api_t::infoGet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)

Get Encoder Calculate Information.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Memory address to get angle internal information

◆ parameterUpdate

fsp_err_t(* motor_angle_api_t::parameterUpdate) (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *p_cfg)

Update Parameters for the calculation in the Motor_Angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_angle_instance_t

struct motor_angle_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_angle_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_angle_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_angle_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_angle_ctrl_t

typedef void motor_angle_ctrl_t

Motor Angle Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_sense_encoder_angle_adjust_t

enum motor_sense_encoder_angle_adjust_t	
Enumerator	
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_90_DEGREE	Roter Angle Adjustment to pull in 90degree.
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_0_DEGREE	Roter Angle Adjustment to pull in 0degree.
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_FINISH	Roter Angle Adjustment Finish.
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_OPEN LOOP	Roter Angle Adjustment Finish.

◆ motor_angle_open_loop_t

enum motor_angle_open_loop_t	
Enumerator	
MOTOR_ANGLE_OPEN_LOOP_INACTIVE	Openloop inactive.
MOTOR_ANGLE_OPEN_LOOP_ACTIVE	Openloop active.

◆ motor_angle_error_t

enum motor_angle_error_t	
Flag for induction correction error status	
Enumerator	
MOTOR_ANGLE_ERROR_NONE	No error happen.
MOTOR_ANGLE_ERROR_INDUCTION	Error happens in induction sensor correction process.

5.3.10.2 Motor Interface

Interfaces » Motor

Detailed Description

Interface for Motor functions.

Summary

The Motor interface provides Motor functionality.

Data Structures

struct	motor_callback_args_t
struct	motor_cfg_t
struct	motor_api_t
struct	motor_instance_t

Typedefs

typedef void	motor_ctrl_t

Enumerations

enum	motor_error_t
enum	motor_callback_event_t
enum	motor_wait_stop_flag_t
enum	motor_function_select_t

Data Structure Documentation

◆ motor_callback_args_t

struct motor_callback_args_t		
callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data.
motor_callback_event_t	event	

◆ motor_cfg_t

struct motor_cfg_t	
Configuration parameters.	
Data Fields	
motor_speed_instance_t const *	p_motor_speed_instance
	Speed Instance.

motor_current_instance_t const *	p_motor_current_instance
	Current Instance.
void(*	p_callback)(motor_callback_args_t *p_args)
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_callback

void(* motor_cfg_t::p_callback) (motor_callback_args_t *p_args)

Placeholder for user data. Passed to the user callback in motor_callback_args_t.

◆ motor_api_t

struct motor_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	errorSet)(motor_ctrl_t *const p_ctrl, motor_error_t const error)
fsp_err_t(*	speedSet)(motor_ctrl_t *const p_ctrl, float const speed_rpm)

fsp_err_t(*	positionSet)(motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)
fsp_err_t(*	statusGet)(motor_ctrl_t *const p_ctrl, uint8_t *const p_status)
fsp_err_t(*	angleGet)(motor_ctrl_t *const p_ctrl, float *const p_angle_rad)
fsp_err_t(*	speedGet)(motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)
fsp_err_t(*	waitStopFlagGet)(motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)
fsp_err_t(*	errorCheck)(motor_ctrl_t *const p_ctrl, uint16_t *const p_error)
fsp_err_t(*	functionSelect)(motor_ctrl_t *const p_ctrl, motor_function_select_t const function)

Field Documentation

◆ open

fsp_err_t(* motor_api_t::open) (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_api_t::close) (motor_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_api_t::run) (motor_ctrl_t *const p_ctrl)

Run the motor. (Start the motor rotation.)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ stop

fsp_err_t(* motor_api_t::stop) (motor_ctrl_t *const p_ctrl)

Stop the motor. (Stop the motor rotation.)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_api_t::reset) (motor_ctrl_t *const p_ctrl)

Reset the motor control. (Recover from the error status.)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ errorSet

fsp_err_t(* motor_api_t::errorSet) (motor_ctrl_t *const p_ctrl, motor_error_t const error)

Set Error Information.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	error	Happened error code

◆ speedSet

fsp_err_t(* motor_api_t::speedSet) (motor_ctrl_t *const p_ctrl, float const speed_rpm)

Set rotation speed.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_rpm	Required rotation speed [rpm]

◆ positionSet

fsp_err_t(* motor_api_t::positionSet) (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)

Set reference position.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_position	Pointer to set required data

◆ statusGet

fsp_err_t(* motor_api_t::statusGet) (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)

Get the motor control status.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_status	Pointer to get the motor control status

◆ angleGet

fsp_err_t(* motor_api_t::angleGet) (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)

Get the rotor angle.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_angle_rad	Pointer to get the rotor angle [rad]

◆ speedGet

fsp_err_t(* motor_api_t::speedGet) (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)

Get the rotation speed.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_rpm	Pointer to get the rotation speed [rpm]

◆ waitStopFlagGet

fsp_err_t(* motor_api_t::waitStopFlagGet) (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)

Get wait stop flag.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_flag	Pointer to wait stop flag

◆ errorCheck

fsp_err_t(* motor_api_t::errorCheck) (motor_ctrl_t *const p_ctrl, uint16_t *const p_error)

Check the error occurrence

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_error	Pointer to get occurred error

◆ functionSelect

fsp_err_t(* motor_api_t::functionSelect) (motor_ctrl_t *const p_ctrl, motor_function_select_t const function)

FunctionSelect.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	function	Selected function

◆ motor_instance_t

struct motor_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_ctrl_t

typedef void motor_ctrl_t
Motor Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_error_t

enum motor_error_t
Error information

◆ motor_callback_event_t

enum motor_callback_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_CALLBACK_EVENT_SPEED_FORWARD	Event forward speed control.
MOTOR_CALLBACK_EVENT_SPEED_BACKWARD	Event backward speed control.
MOTOR_CALLBACK_EVENT_CURRENT_FORWARD	Event forward current control.
MOTOR_CALLBACK_EVENT_CURRENT_BACKWARD	Event backward current control.
MOTOR_CALLBACK_EVENT_ADC_FORWARD	Event before motor 120 driver process.
MOTOR_CALLBACK_EVENT_ADC_BACKWARD	Event after motor 120 driver process.
MOTOR_CALLBACK_EVENT_CYCLE_FORWARD	Before cyclic process of speed control.
MOTOR_CALLBACK_EVENT_CYCLE_BACKWARD	After cyclic process of speed control.

◆ motor_wait_stop_flag_t

enum motor_wait_stop_flag_t	
Flag for waiting for motor stop	
Enumerator	
MOTOR_WAIT_STOP_FLAG_CLEAR	Wait stop flag clear.
MOTOR_WAIT_STOP_FLAG_SET	Wait stop flag set.

◆ motor_function_select_t

enum motor_function_select_t	
Function select	
Enumerator	
MOTOR_FUNCTION_SELECT_NONE	No function selected.
MOTOR_FUNCTION_SELECT_INERTIA_ESTIMATE	Inertia estimation.
MOTOR_FUNCTION_SELECT_RETURN_ORIGIN	Return origin position.

5.3.10.3 Motor current Interface

Interfaces » Motor

Detailed Description

Interface for motor current functions.

Summary

The Motor current interface for getting the PWM modulation duty from electric current and speed

Data Structures

struct	motor_current_output_t
struct	motor_current_input_current_t
struct	motor_current_input_voltage_t

struct	motor_current_get_voltage_t
struct	motor_current_cfg_t
struct	motor_current_api_t
struct	motor_current_instance_t

Typedefs

typedef void	motor_current_ctrl_t

Enumerations

enum	motor_current_event_t

Data Structure Documentation

◆ motor_current_output_t

struct motor_current_output_t		
Structure of interface to speed control Output parameters		
Data Fields		
float	f_id	D-axis current [A].
float	f_iq	Q-axis current [A].
float	f_vamax	
float	f_speed_rad	Speed value [rad/s].
float	f_speed_rpm	Speed value [rpm].
float	f_rotor_angle	Motor rotor angle [rad].
float	f_position_rad	Motor rotor position [rad].
float	f_ed	Estimated d-axis component[V] of flux due to the permanent magnet.
float	f_eq	Estimated q-axis component[V] of flux due to the permanent magnet.
float	f_phase_err_rad	Phase error [rad].
uint8_t	u1_flag_get_iref	Flag to set d/q-axis current reference.
uint8_t	u1_adjust_status	Angle adjustment satatus.
uint8_t	u1_adjust_count_full	Angle adjustment count full.
uint8_t	u1_adjust_mode	Angle adjustment mode (only valid with ENCODER)

uint8_t	u1_openloop_status	Openloop status.
float	f_openloop_speed	Openloop speed.
float	f_openloop_id_ref	Openloop d-axis current.

◆ motor_current_input_current_t

struct motor_current_input_current_t		
Three-phase input current		
Data Fields		
float	iu	U phase current[A].
float	iv	V phase current[A].
float	iw	W phase current[A].

◆ motor_current_input_voltage_t

struct motor_current_input_voltage_t		
Input voltage		
Data Fields		
float	vdc	Main line voltage[V].
float	va_max	Maximum magnitude of voltage vector[V].

◆ motor_current_get_voltage_t

struct motor_current_get_voltage_t		
Struct to get motor current		
Data Fields		
float	u_voltage	U phase voltage[V].
float	v_voltage	V phase voltage[V].
float	w_voltage	W phase voltage[V].
float	vd_reference	d-axis voltage reference
float	vq_reference	q-axis voltage reference

◆ motor_current_cfg_t

struct motor_current_cfg_t
Configuration parameters.

◆ motor_current_api_t

struct motor_current_api_t
Functions implemented at the Motor Current Module will follow these APIs.

Data Fields

fsp_err_t(*	open)(motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_current_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_current_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_current_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterSet)(motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input)
fsp_err_t(*	currentReferenceSet)(motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference)
fsp_err_t(*	speedPhaseSet)(motor_current_ctrl_t *const p_ctrl, float const speed_rad, float const phase_rad)
fsp_err_t(*	currentSet)(motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage)
fsp_err_t(*	parameterGet)(motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output)
fsp_err_t(*	currentGet)(motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq)
fsp_err_t(*	phaseVoltageGet)(motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage)
fsp_err_t(*	parameterUpdate)(motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_current_api_t::open) (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Initialize the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_current_api_t::close) (motor_current_ctrl_t *const p_ctrl)

Close (Finish) the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_current_api_t::reset) (motor_current_ctrl_t *const p_ctrl)

Reset variables for the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_current_api_t::run) (motor_current_ctrl_t *const p_ctrl)

Activate the motor current control.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterSet

fsp_err_t(* motor_current_api_t::parameterSet) (motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input)

Set (Input) parameters into the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_input	Pointer to input data structure(speed control output data)

◆ currentReferenceSet

fsp_err_t(* motor_current_api_t::currentReferenceSet) (motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference)

Set (Input) Current reference into the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	id_reference	D-axis current reference [A]
[in]	iq_reference	Q-axis current reference [A]

◆ speedPhaseSet

fsp_err_t(* motor_current_api_t::speedPhaseSet) (motor_current_ctrl_t *const p_ctrl, float const speed_rad, float const phase_rad)

Set (Input) Speed & Phase data into the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_rad	Rotational speed [rad/s]
[in]	phase_rad	Rotor phase [rad]

◆ currentSet

fsp_err_t(* motor_current_api_t::currentSet) (motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage)

Set (Input) Current data into the motor current module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_current	Pointer to input current structure
[in]	p_st_voltage	Pointer to input voltage structure

◆ parameterGet

fsp_err_t(* motor_current_api_t::parameterGet) (motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output)

Get (output) parameters from the motor current module

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_st_output	Pointer to output data structure(speed control input data)

◆ currentGet

fsp_err_t(* motor_current_api_t::currentGet) (motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq)

Get d/q-axis current

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_id	Pointer to get d-axis current [A]
[out]	p_iq	Pointer to get q-axis current [A]

◆ phaseVoltageGet

fsp_err_t(* motor_current_api_t::phaseVoltageGet) (motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage)

Get phase output voltage

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_voltage	Pointer to get voltages

◆ parameterUpdate

fsp_err_t(* motor_current_api_t::parameterUpdate) (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Update parameters for the calculation in the motor current control.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_current_instance_t

struct motor_current_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_current_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_current_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_current_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_current_ctrl_t

typedef void motor_current_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_current_event_t

enum motor_current_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_CURRENT_EVENT_FORWARD	Event forward current control.
MOTOR_CURRENT_EVENT_DATA_SET	Event set speed control output data.
MOTOR_CURRENT_EVENT_BACKWARD	Event backward current control.

5.3.10.4 Motor driver Interface

Interfaces » Motor

Detailed Description

Interface for motor driver functions.

Summary

The Motor driver interface for setting the PWM modulation duty

Data Structures

struct	motor_driver_callback_args_t
struct	motor_driver_current_get_t
struct	motor_driver_cfg_t
struct	motor_driver_api_t
struct	motor_driver_instance_t

Typedefs

typedef void	motor_driver_ctrl_t

Enumerations

enum	motor_driver_event_t
enum	motor_driver_shunt_type_t

Data Structure Documentation

◆ motor_driver_callback_args_t

struct motor_driver_callback_args_t		
Callback function parameter data		
Data Fields		
motor_driver_event_t	event	Event trigger.
void *	p_context	Placeholder for user data.

◆ motor_driver_current_get_t

struct motor_driver_current_get_t		
Current Data Get Structure		
Data Fields		
float	iu	U phase current [A].
float	iv	V phase current [A].
float	iw	W phase current [A].
float	vdc	Main Line Voltage [V].
float	va_max	maximum magnitude of voltage vector
float	sin_ad	Induction sensor sin signal.
float	cos_ad	Induction sensor cos signal.

◆ motor_driver_cfg_t

struct motor_driver_cfg_t	
Configuration parameters.	
Data Fields	
adc_channel_t	iu_ad_ch
	A/D Channel for U Phase Current.
adc_channel_t	iv_ad_ch
	A/D Channel for V Phase Current.
adc_channel_t	iw_ad_ch
	A/D Channel for W Phase Current.

adc_channel_t	vdc_ad_ch
	A/D Channel for Main Line Voltage.
adc_channel_t	sin_ad_ch
	A/D Channel for induction sensor sin signal.
adc_channel_t	cos_ad_ch
	A/D Channel for induction sensor cos signal.
motor_driver_shunt_type_t	shunt
	Selection of shunt type.
void *	p_context
	Placeholder for user data.

◆ motor_driver_api_t

struct motor_driver_api_t	
Functions implemented at the HAL layer will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	phaseVoltageSet)(motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)
fsp_err_t(*	currentGet)(motor_driver_ctrl_t *const p_ctrl,

	motor_driver_current_get_t *const p_current_get)
fsp_err_t(*	flagCurrentOffsetGet)(motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset)
fsp_err_t(*	currentOffsetRestart)(motor_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterUpdate)(motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_driver_api_t::open) (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)

Initialize the Motor Driver Module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_driver_api_t::close) (motor_driver_ctrl_t *const p_ctrl)

Close the Motor Driver Module

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_driver_api_t::reset) (motor_driver_ctrl_t *const p_ctrl)

Reset variables of the Motor Driver Module

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ phaseVoltageSet

fsp_err_t(* motor_driver_api_t::phaseVoltageSet) (motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)

Set (Input) Phase Voltage data into the Motor Driver Module

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	u_voltage	U phase voltage [V]
[in]	v_voltage	V phase voltage [V]
[in]	w_voltage	W phase voltage [V]

◆ currentGet

fsp_err_t(* motor_driver_api_t::currentGet) (motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get)

Get Phase current, Vdc and Va_max data from the Motor Driver Module

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_current_get	Pointer to get data structure.

◆ flagCurrentOffsetGet

fsp_err_t(* motor_driver_api_t::flagCurrentOffsetGet) (motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset)

Get the flag of finish current offset detection from the Motor Driver Module

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_flag_offset	Flag of finish current offset detection

◆ currentOffsetRestart

fsp_err_t(* motor_driver_api_t::currentOffsetRestart) (motor_driver_ctrl_t *const p_ctrl)

Restart current offset detection

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterUpdate

fsp_err_t(* motor_driver_api_t::parameterUpdate) (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)

Update Configuration Parameters for the calculation in the Motor Driver Module

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_driver_instance_t

struct motor_driver_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_driver_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_driver_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_driver_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_driver_ctrl_t

typedef void motor_driver_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_driver_event_t

enum motor_driver_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_DRIVER_EVENT_FORWARD	Event before Motor Driver Process (before Current Control timing)
MOTOR_DRIVER_EVENT_CURRENT	Event Current Control timing.
MOTOR_DRIVER_EVENT_BACKWARD	Event after Motor Driver Process (after PWM duty setting)

◆ motor_driver_shunt_type_t

enum motor_driver_shunt_type_t	
Selection of shunt type	
Enumerator	
MOTOR_DRIVER_SHUNT_TYPE_1_SHUNT	Only use U phase current.
MOTOR_DRIVER_SHUNT_TYPE_2_SHUNT	Use U and W phase current.
MOTOR_DRIVER_SHUNT_TYPE_3_SHUNT	Use all phase current.

5.3.10.5 Motor Inertia Estimate Interface

Interfaces » Motor

Detailed Description

Interface for Motor inertia estimate functions.

Summary

The Motor interface provides Motor inertia estimate functionality.

Data Structures

struct	motor_inertia_estimate_info_t
struct	motor_inertia_estimate_cfg_t

struct	motor_inertia_estimate_api_t
struct	motor_inertia_estimate_instance_t

Typedefs

typedef void	motor_inertia_estimate_ctrl_t

Data Structure Documentation

◆ motor_inertia_estimate_info_t

struct motor_inertia_estimate_info_t		
Interface data structure		
Data Fields		
int16_t	s2_position_reference_degree	Position reference [degree].
motor_inertia_estimate_mode_t	mode	Internal mode of inertia estimation.
float	f_estimated_inertia	Estimated inertia data.

◆ motor_inertia_estimate_cfg_t

struct motor_inertia_estimate_cfg_t		
Configuration parameters.		
Data Fields		
void *	p_context	
void const *	p_extend	Placeholder for user extension.

◆ motor_inertia_estimate_api_t

struct motor_inertia_estimate_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_inertia_estimate_ctrl_t *const p_ctrl)
fsp_err_t(*	start)(motor_inertia_estimate_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(motor_inertia_estimate_ctrl_t *const p_ctrl)

fsp_err_t(*	reset)(motor_inertia_estimate_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_info_t *const p_info)
fsp_err_t(*	dataSet)(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_set_data_t *const p_set_data)
fsp_err_t(*	speedCyclic)(motor_inertia_estimate_ctrl_t *const p_ctrl)
fsp_err_t(*	currentCyclic)(motor_inertia_estimate_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterUpdate)(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_inertia_estimate_api_t::open) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_inertia_estimate_api_t::close) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ start

fsp_err_t(* motor_inertia_estimate_api_t::start) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Start the function.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ stop

fsp_err_t(* motor_inertia_estimate_api_t::stop) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Stop(same as cancel) the function.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_inertia_estimate_api_t::reset) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Reset the function. (recover from error state)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ infoGet

fsp_err_t(* motor_inertia_estimate_api_t::infoGet) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_info_t *const p_info)

Get information from the function (to set speed & position control)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Pointer to information

◆ dataSet

fsp_err_t(* motor_inertia_estimate_api_t::dataSet) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_set_data_t *const p_set_data)

Set the data to the function (from speed, position and current control)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_set_data	Pointer to set the data

◆ speedCyclic

fsp_err_t(* motor_inertia_estimate_api_t::speedCyclic) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Speed cyclic process of the function

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ currentCyclic

fsp_err_t(* motor_inertia_estimate_api_t::currentCyclic) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Current cyclic process of the function

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterUpdate

fsp_err_t(* motor_inertia_estimate_api_t::parameterUpdate) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *p_cfg)

Update parameters for the function.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_inertia_estimate_instance_t

struct motor_inertia_estimate_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

motor_inertia_estimate_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_inertia_estimate_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_inertia_estimate_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_inertia_estimate_ctrl_t

typedef void motor_inertia_estimate_ctrl_t
Motor inertia estimate block. Allocate an instance specific control block to pass into the API calls.

5.3.10.6 Motor position Interface

Interfaces » Motor

Detailed Description

Interface for motor position functions.

Summary

The Motor position interface for getting the speed references from Encoder Sensor

Data Structures

struct	motor_position_info_t
struct	motor_position_cfg_t
struct	motor_position_api_t
struct	motor_position_instance_t

Typedefs

typedef void	motor_position_ctrl_t

Enumerations

enum	motor_position_ctrl_mode_t

Data Structure Documentation

◆ motor_position_info_t

struct motor_position_info_t		
Position information		
Data Fields		
uint8_t	u1_state_position_profile	Position control profile state.
int16_t	s2_position_degree	Position data [degree].

◆ motor_position_cfg_t

struct motor_position_cfg_t		
Configuration parameters.		
Data Fields		
void *	p_context	Placeholder for user data.
void const *	p_extend	

◆ motor_position_api_t

struct motor_position_api_t	
Functions implemented at the HAL layer will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_position_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_position_ctrl_t *const p_ctrl)
fsp_err_t(*	positionGet)(motor_position_ctrl_t *const p_ctrl, int16_t *const p_position)
fsp_err_t(*	positionSet)(motor_position_ctrl_t *const p_ctrl, float const position_rad)
fsp_err_t(*	positionReferenceSet)(motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg)
fsp_err_t(*	controlModeSet)(motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode)

fsp_err_t(*	positionControl)(motor_position_ctrl_t *const p_ctrl)
fsp_err_t(*	ipdSpeedPControl)(motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref)
fsp_err_t(*	speedReferencePControlGet)(motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)
fsp_err_t(*	speedReferenceIpdControlGet)(motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref)
fsp_err_t(*	speedReferenceFeedforwardGet)(motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)
fsp_err_t(*	infoGet)(motor_position_ctrl_t *const p_ctrl, motor_position_info_t *const p_info)
fsp_err_t(*	parameterUpdate)(motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_position_api_t::open) (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Initialize the Motor Position Module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_position_api_t::close) (motor_position_ctrl_t *const p_ctrl)

Close (Finish) the Motor Position Module.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_position_api_t::reset) (motor_position_ctrl_t *const p_ctrl)

Reset(Stop) the Motor Position Module.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ positionGet

fsp_err_t(* motor_position_api_t::positionGet) (motor_position_ctrl_t *const p_ctrl, int16_t *const p_position)

Get Position data.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_position	Pointer to get position data

◆ positionSet

fsp_err_t(* motor_position_api_t::positionSet) (motor_position_ctrl_t *const p_ctrl, float const position_rad)

Set Position data from Encoder.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	position_rad	Position data [radian]

◆ positionReferenceSet

fsp_err_t(* motor_position_api_t::positionReferenceSet) (motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg)

Set (Input) Position reference into the Motor Position Module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	position_reference_deg	Position reference [degree]

◆ controlModeSet

fsp_err_t(* motor_position_api_t::controlModeSet) (motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode)

Set (Input) Position Control Mode.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	mode	Position Control Mode

◆ positionControl

fsp_err_t(* motor_position_api_t::positionControl) (motor_position_ctrl_t *const p_ctrl)

Calculate internal position reference

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ ipdSpeedPControl

fsp_err_t(* motor_position_api_t::ipdSpeedPControl) (motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref)

Calculate iq reference

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	ref_speed_rad	Speed Reference [rad/sec]
[in]	speed_rad	Current Speed [rad/sec]
[out]	p_iq_ref	Pointer to get iq reference

◆ speedReferencePControlGet

fsp_err_t(* motor_position_api_t::speedReferencePControlGet) (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)

Get Speed Reference by P Control

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_ref	Pointer to get speed reference

◆ speedReferenceIpdControlGet

fsp_err_t(* motor_position_api_t::speedReferenceIpdControlGet) (motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref)

Get Speed Reference by IPD Control

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_ref	Pointer to get speed reference

◆ speedReferenceFeedforwardGet

fsp_err_t(* motor_position_api_t::speedReferenceFeedforwardGet) (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)

Get Speed Reference by Speed Feedforward

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_ref	Pointer to get speed reference

◆ infoGet

fsp_err_t(* motor_position_api_t::infoGet) (motor_position_ctrl_t *const p_ctrl, motor_position_info_t *const p_info)

Get Position information.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Pointer to get information

◆ parameterUpdate

fsp_err_t(* motor_position_api_t::parameterUpdate) (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Update Parameters for the calculation in the Motor Position Module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_position_instance_t

struct motor_position_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
motor_position_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_position_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_position_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_position_ctrl_t

typedef void motor_position_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_position_ctrl_mode_t

enum motor_position_ctrl_mode_t
Position Control Mode

5.3.10.7 Motor Return Origin Function Interface

Interfaces » Motor

Detailed Description

Interface for Motor return origin functions.

Summary

The Motor interface provides Motor return origin functionality.

Data Structures

struct	motor_return_origin_info_t
struct	motor_return_origin_cfg_t

struct	motor_return_origin_api_t
struct	motor_return_origin_instance_t

Typedefs

typedef void	motor_return_origin_ctrl_t

Enumerations

enum	motor_return_origin_mode_t

Data Structure Documentation

◆ motor_return_origin_info_t

struct motor_return_origin_info_t		
Interface data structure		
Data Fields		
float	f_position_reference_degree	Position reference [degree].
motor_return_origin_state_t	state	
float	f_result_angle	Result angle position.

◆ motor_return_origin_cfg_t

struct motor_return_origin_cfg_t		
Configuration parameters.		
Data Fields		
motor_return_origin_mode_t	mode	
void *	p_context	
void const *	p_extend	Placeholder for user extension.

◆ motor_return_origin_api_t

struct motor_return_origin_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_return_origin_ctrl_t *const p_ctrl)
fsp_err_t(*	start)(motor_return_origin_ctrl_t *const p_ctrl)

fsp_err_t(*	stop)(motor_return_origin_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_return_origin_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_info_t *const p_info)
fsp_err_t(*	dataSet)(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_set_data_t *const p_set_data)
fsp_err_t(*	speedCyclic)(motor_return_origin_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterUpdate)(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_return_origin_api_t::open) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_return_origin_api_t::close) (motor_return_origin_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ start

fsp_err_t(* motor_return_origin_api_t::start) (motor_return_origin_ctrl_t *const p_ctrl)

Start the function.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ stop

fsp_err_t(* motor_return_origin_api_t::stop) (motor_return_origin_ctrl_t *const p_ctrl)

Stop the function. (Cancel the function works.)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_return_origin_api_t::reset) (motor_return_origin_ctrl_t *const p_ctrl)

Reset the function. (Initialize the function.)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ infoGet

fsp_err_t(* motor_return_origin_api_t::infoGet) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_info_t *const p_info)

Get the function information.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Pointer to info

◆ dataSet

fsp_err_t(* motor_return_origin_api_t::dataSet) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_set_data_t *const p_set_data)

Set the data to the function

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_set_data	Pointer to set the data

◆ speedCyclic

fsp_err_t(* motor_return_origin_api_t::speedCyclic) (motor_return_origin_ctrl_t *const p_ctrl)

Speed cyclic process of the function

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterUpdate

fsp_err_t(* motor_return_origin_api_t::parameterUpdate) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *p_cfg)

Update parameters for the function.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_return_origin_instance_t

struct motor_return_origin_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_return_origin_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_return_origin_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_return_origin_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_return_origin_ctrl_t

typedef void motor_return_origin_ctrl_t
Motor return origin function block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_return_origin_mode_t

enum motor_return_origin_mode_t	
Selection type of return origin function	
Enumerator	
MOTOR_RETURN_ORIGIN_MODE_PUSH	Return origin position with pushing.
MOTOR_RETURN_ORIGIN_MODE_SENSOR	Return origin position with origin sensor.
MOTOR_RETURN_ORIGIN_MODE_2_SENSOR	Return origin position with 2 sensors.
MOTOR_RETURN_ORIGIN_MODE_3_SENSOR	Return origin position with 3 sensors.

5.3.10.8 Motor speed Interface

Interfaces » Motor

Detailed Description

Interface for motor speed functions.

Summary

The Motor speed interface for getting the current references from electric current and rotational speed

Data Structures

struct	motor_speed_callback_args_t
struct	motor_speed_position_data_t
struct	motor_speed_cfg_t
struct	motor_speed_api_t

struct	motor_speed_instance_t

Typedefs

typedef void	motor_speed_ctrl_t

Enumerations

enum	motor_speed_event_t
enum	motor_speed_loop_mode_t
enum	motor_speed_step_t

Data Structure Documentation

◆ motor_speed_callback_args_t

struct motor_speed_callback_args_t		
Callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data.
motor_speed_event_t	event	

◆ motor_speed_position_data_t

struct motor_speed_position_data_t		
Motor speed and position structure		
Data Fields		
motor_speed_step_t	e_step_mode	Select step mode.
motor_speed_loop_mode_t	e_loop_mode	Select control mode.
int16_t	position_reference_degree	Position reference [degree].

◆ motor_speed_cfg_t

struct motor_speed_cfg_t	
Configuration parameters.	
Data Fields	
motor_speed_input_t *	st_input
	Input data structure for automatic set.
motor_speed_output_t *	st_output
	Output data structure for automatic receive.

motor_position_instance_t const *	p_position_instance
	Position module instance.
void *	p_context
	Placeholder for user data.

◆ motor_speed_api_t

struct motor_speed_api_t	
Functions implemented at the HAL layer will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	speedReferenceSet)(motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm)
fsp_err_t(*	positionReferenceSet)(motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data)
fsp_err_t(*	parameterSet)(motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input)
fsp_err_t(*	speedControl)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterGet)(motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output)

fsp_err_t(*	parameterUpdate)(motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_speed_api_t::open) (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Initialize the motor speed module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_speed_api_t::close) (motor_speed_ctrl_t *const p_ctrl)

Close (Finish) the motor speed module.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_speed_api_t::reset) (motor_speed_ctrl_t *const p_ctrl)

Reset(Stop) the motor speed module.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_speed_api_t::run) (motor_speed_ctrl_t *const p_ctrl)

Activate the motor speed control.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ speedReferenceSet

fsp_err_t(* motor_speed_api_t::speedReferenceSet) (motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm)

Set (Input) speed reference into the motor speed module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_reference_rpm	Speed reference [rpm]

◆ positionReferenceSet

fsp_err_t(* motor_speed_api_t::positionReferenceSet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data)

Set (Input) position reference and control mode

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_position_data	Pointer to structure position data

◆ parameterSet

fsp_err_t(* motor_speed_api_t::parameterSet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input)

Set (Input) speed parameters into the motor speed module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_input	Pointer to structure to input parameters.

◆ speedControl

fsp_err_t(* motor_speed_api_t::speedControl) (motor_speed_ctrl_t *const p_ctrl)

Calculate current reference

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterGet

fsp_err_t(* motor_speed_api_t::parameterGet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output)

Get speed control output parameters

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_st_output	Pointer to get speed control parameters

◆ parameterUpdate

fsp_err_t(* motor_speed_api_t::parameterUpdate) (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Update Parameters for the calculation in the motor speed module.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_speed_instance_t

struct motor_speed_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_speed_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_speed_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_speed_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_speed_ctrl_t

typedef void motor_speed_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ motor_speed_event_t

enum motor_speed_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_SPEED_EVENT_FORWARD	Event forward speed control.
MOTOR_SPEED_EVENT_BACKWARD	Event backward speed control.
MOTOR_SPEED_EVENT_ENCODER_CYCLIC	Event encoder cyclic.
MOTOR_SPEED_EVENT_ENCODER_ADJUST	Event encoder adjust.

◆ motor_speed_loop_mode_t

enum motor_speed_loop_mode_t	
Enumerator	
MOTOR_SPEED_LOOP_MODE_SPEED	Speed control mode.
MOTOR_SPEED_LOOP_MODE_POSITION	Position control mode.

◆ motor_speed_step_t

enum motor_speed_step_t	
Enumerator	
MOTOR_SPEED_STEP_DISABLE	Position control works without step mode.
MOTOR_SPEED_STEP_ENABLE	Position control works with step mode.

5.3.11 Networking

Interfaces

Detailed Description

Networking Interfaces.

Modules

	Ethernet Interface Interface for Ethernet functions.
	Ethernet PHY Interface Interface for Ethernet PHY functions.
	Ethernet Switch Interface Interface for Ethernet Switch functions.
	GPTP Interface Interface for gPTP timing.
	EtherCAT SSC Interface Interface for EtherCAT SSC functions.

5.3.11.1 Ethernet Interface

Interfaces » Networking

Detailed Description

Interface for Ethernet functions.

Summary

The Ethernet interface provides Ethernet functionality. The Ethernet interface supports the following features:

- Transmit/receive processing (Blocking and Non-Blocking)
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support

Data Structures

struct	ether_nic_info_t
struct	ether_callback_args_t

struct	ether_cfg_t
struct	ether_api_t
struct	ether_instance_t

Typedefs

typedef void	ether_ctrl_t

Enumerations

enum	ether_wake_on_lan_t
enum	ether_flow_control_t
enum	ether_multicast_t
enum	ether_promiscuous_t
enum	ether_zerocopy_t
enum	ether_event_t

Data Structure Documentation

◆ ether_nic_info_t

struct ether_nic_info_t		
Network interface device information		
Data Fields		
uint8_t *	p_mac_address	MAC address.

◆ ether_callback_args_t

struct ether_callback_args_t		
Callback Function Parameter Data		
Data Fields		
uint32_t	channel	Device channel number.
ether_event_t	event	Event code.
uint32_t	status_ecsr	ETHERC status register for interrupt handler.
uint32_t	status_eesr	ETHERC/EDMAC status register for interrupt handler.
ether_nic_info_t *	p_nic_info	Pointer of NIC info.
void *	p_context	Placeholder for user data. Set in

		ether_api_t::open function in ether_cfg_t.

◆ ether_cfg_t

struct ether_cfg_t	
Configuration parameters.	
Data Fields	
uint8_t	channel
	Channel.
ether_zerocopy_t	zerocopy
	Zero copy enable or disable in Read/Write function.
ether_multicast_t	multicast
	Multicast enable or disable.
ether_promiscuous_t	promiscuous
	Promiscuous mode enable or disable.
ether_flow_control_t	flow_control
	Flow control functionally enable or disable.
ether_padding_t	padding
	Padding length inserted into the received Ethernet frame.
uint32_t	padding_offset
	Offset of the padding inserted into the received Ethernet frame.
uint32_t	broadcast_filter
	Limit of the number of broadcast frames received continuously.

uint8_t *	p_mac_address
	Pointer of MAC address.
uint8_t	num_tx_descriptors
	Number of transmission descriptor.
uint8_t	num_rx_descriptors
	Number of receive descriptor.
uint8_t **	pp_ether_buffers
	Transmit and receive buffer.
uint32_t	ether_buffer_size
	Size of transmit and receive buffer.
IRQn_Type	irq
	Interrupt number.
uint32_t	interrupt_priority
	Interrupt priority.
void(*	p_callback)(ether_callback_args_t *p_args)
	Callback provided when an ISR occurs.
ether_phy_instance_t const *	p_ether_phy_instance
	Pointer to ETHER_PHY instance.

void *	p_context
	Placeholder for user data. More...
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_context

void* ether_cfg_t::p_context

Placeholder for user data.

Placeholder for user data. Passed to the user callback in ether_callback_args_t.

◆ ether_api_t

struct ether_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ether_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes)
fsp_err_t(*	bufferRelease)(ether_ctrl_t *const p_ctrl)
fsp_err_t(*	rxBufferUpdate)(ether_ctrl_t *const p_ctrl, void *const p_buffer)
fsp_err_t(*	write)(ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length)
fsp_err_t(*	linkProcess)(ether_ctrl_t *const p_ctrl)

fsp_err_t(*	wakeOnLANEnable)(ether_ctrl_t *const p_ctrl)
fsp_err_t(*	txStatusGet)(ether_ctrl_t *const p_ctrl, void *const p_buffer_address)
fsp_err_t(*	callbackSet)(ether_ctrl_t *const p_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* ether_api_t::open) (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* ether_api_t::close) (ether_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* ether_api_t::read) (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes)

Read packet if data is available.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to where to store read data.
[in,out]	length_bytes	Number of bytes in buffer

◆ bufferRelease

fsp_err_t(* ether_api_t::bufferRelease) (ether_ctrl_t *const p_ctrl)

Release rx buffer from buffer pool process in zero-copy read operation.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ rxBufferUpdate

fsp_err_t(* ether_api_t::rxBufferUpdate) (ether_ctrl_t *const p_ctrl, void *const p_buffer)

Update the buffer pointer in the current receive descriptor.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buffer	New address to write into the rx buffer descriptor.

◆ write

fsp_err_t(* ether_api_t::write) (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length)

Write packet.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to data to write.
[in]	frame_length	Send ethernet frame size (without 4 bytes of CRC data size).

◆ linkProcess

fsp_err_t(* ether_api_t::linkProcess) (ether_ctrl_t *const p_ctrl)

Process link.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ wakeOnLANEnable

fsp_err_t(* ether_api_t::wakeOnLANEnable) (ether_ctrl_t *const p_ctrl)

Enable magic packet detection.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ txStatusGet

fsp_err_t(* ether_api_t::txStatusGet) (ether_ctrl_t *const p_ctrl, void *const p_buffer_address)

Get the address of the most recently sent buffer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_buffer_address	Pointer to the address of the most recently sent buffer.

◆ callbackSet

fsp_err_t(* ether_api_t::callbackSet) (ether_ctrl_t *const p_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the ETHER control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ ether_instance_t

struct ether_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ether_ctrl_t *	p_ctrl	Pointer to the control structure

		for this instance.
ether_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ether_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ether_ctrl_t

typedef void ether_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ ether_wake_on_lan_t

enum ether_wake_on_lan_t	
Wake on LAN	
Enumerator	
ETHER_WAKE_ON_LAN_DISABLE	Disable Wake on LAN.
ETHER_WAKE_ON_LAN_ENABLE	Enable Wake on LAN.

◆ ether_flow_control_t

enum ether_flow_control_t	
Flow control functionality	
Enumerator	
ETHER_FLOW_CONTROL_DISABLE	Disable flow control functionality.
ETHER_FLOW_CONTROL_ENABLE	Enable flow control functionality with pause frames.

◆ ether_multicast_t

enum ether_multicast_t	
Multicast Filter	
Enumerator	
ETHER_MULTICAST_DISABLE	Disable reception of multicast frames.
ETHER_MULTICAST_ENABLE	Enable reception of multicast frames.

◆ ether_promiscuous_t

enum ether_promiscuous_t	
Promiscuous Mode	
Enumerator	
ETHER_PROMISCUOUS_DISABLE	Only receive packets with current MAC address, multicast, and broadcast.
ETHER_PROMISCUOUS_ENABLE	Receive all packets.

◆ ether_zerocopy_t

enum ether_zerocopy_t	
Zero copy	
Enumerator	
ETHER_ZEROCOPY_DISABLE	Disable zero copy in Read/Write function.
ETHER_ZEROCOPY_ENABLE	Enable zero copy in Read/Write function.

◆ ether_event_t

enum ether_event_t	
Event code of callback function	
Enumerator	
ETHER_EVENT_WAKEON_LAN	Magic packet detection event.
ETHER_EVENT_LINK_ON	Link up detection event.
ETHER_EVENT_LINK_OFF	Link down detection event.
ETHER_EVENT_RX_COMPLETE	Receive complete event.
ETHER_EVENT_RX_MESSAGE_LOST	Receive FIFO overflow or Receive descriptor is full.
ETHER_EVENT_TX_COMPLETE	Transmit complete event.
ETHER_EVENT_TX_BUFFER_EMPTY	Transmit descriptor or FIFO is empty.
ETHER_EVENT_TX_ABORTED	Transmit abort event.
ETHER_EVENT_ERR_GLOBAL	Global error has occurred.
ETHER_EVENT_GET_NIC_INFO	Get NIC Info.

5.3.11.2 Ethernet PHY Interface

Interfaces » Networking

Detailed Description

Interface for Ethernet PHY functions.

Summary

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the Ethernet peripheral.

The Ethernet PHY interface supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Data Structures

struct	ether_phy_callback_args_t
struct	ether_phy_cfg_t
struct	ether_phy_lsi_cfg_t
struct	ether_phy_api_t
struct	ether_phy_instance_t

Typedefs

typedef void	ether_phy_ctrl_t

Enumerations

enum	ether_phy_lsi_type_t
enum	ether_phy_lsi_clk_delay_t
enum	ether_phy_flow_control_t
enum	ether_phy_link_speed_t
enum	ether_phy_mii_type_t
enum	ether_phy_event_t

Data Structure Documentation

◆ ether_phy_callback_args_t

struct ether_phy_callback_args_t		
Callback function parameter data		
Data Fields		
uint32_t	channel	Device channel number.
uint32_t	port	Device port number.
ether_phy_event_t	event	Event code.
void *	p_context	Placeholder for user data. Set in ether_api_t::open function in ether_cfg_t.

◆ ether_phy_cfg_t

struct ether_phy_cfg_t
Configuration parameters.

Data Fields		
uint8_t	channel	Channel.
uint8_t	phy_lsi_address	DEPRECATED Address of PHY-LSI.
uint32_t	phy_reset_wait_time	Wait time for PHY-LSI reboot.
int32_t	mii_bit_access_wait_time	Wait time for MII/RMII access.
ether_phy_lsi_type_t	phy_lsi_type	DEPRECATED Phy LSI type.
ether_phy_flow_control_t	flow_control	Flow control functionally enable or disable.
ether_phy_mii_type_t	mii_type	Interface type: MII, RMII, GMII, or RGMII.
void *	p_context	Placeholder for user data. Passed to the user callback in ether_phy_callback_args_t.
void const *	p_extend	Placeholder for user extension.

◆ ether_phy_lsi_cfg_t

struct ether_phy_lsi_cfg_t		
PHY LSI configuration.		
Data Fields		
uint8_t	address	Address of PHY-LSI.
ether_phy_lsi_type_t	type	Phy LSI type.
ether_phy_lsi_clk_delay_t	tx_clk_delay	Transmit Timing Skew (RGMII)
ether_phy_lsi_clk_delay_t	rx_clk_delay	Receive Timing Skew (RGMII)

◆ ether_phy_api_t

struct ether_phy_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ether_phy_ctrl_t *const p_ctrl)
fsp_err_t(*	chipInit)(ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)
fsp_err_t(*	read)(ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t

	*const p_data)
fsp_err_t(*	write)(ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)
fsp_err_t(*	startAutoNegotiate)(ether_phy_ctrl_t *const p_ctrl)
fsp_err_t(*	linkPartnerAbilityGet)(ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)
fsp_err_t(*	linkStatusGet)(ether_phy_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* ether_phy_api_t::open) (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* ether_phy_api_t::close) (ether_phy_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ chipInit

fsp_err_t(* ether_phy_api_t::chipInit) (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)

Initialize PHY-LSI.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ read

fsp_err_t(* ether_phy_api_t::read) (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data)

Read register value of PHY-LSI.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	reg_addr	Register address.
[out]	p_data	Pointer to the location to store read data.

◆ write

fsp_err_t(* ether_phy_api_t::write) (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)

Write data to register of PHY-LSI.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	reg_addr	Register address.
[in]	data	Data written to register.

◆ startAutoNegotiate

fsp_err_t(* ether_phy_api_t::startAutoNegotiate) (ether_phy_ctrl_t *const p_ctrl)

Start auto negotiation.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ linkPartnerAbilityGet

fsp_err_t(* ether_phy_api_t::linkPartnerAbilityGet) (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)

Get the partner ability.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_line_speed_duplex	Pointer to the location of both the line speed and the duplex.
[out]	p_local_pause	Pointer to the location to store the local pause bits.
[out]	p_partner_pause	Pointer to the location to store the partner pause bits.

◆ linkStatusGet

fsp_err_t(* ether_phy_api_t::linkStatusGet) (ether_phy_ctrl_t *const p_ctrl)

Get Link status from PHY-LSI interface.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ ether_phy_instance_t

struct ether_phy_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ether_phy_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ether_phy_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ether_phy_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ether_phy_ctrl_t

typedef void ether_phy_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ ether_phy_lsi_type_t

enum ether_phy_lsi_type_t	
Phy LSI	
Enumerator	
ETHER_PHY_LSI_TYPE_DEFAULT	Select default configuration. This type does not change Phy LSI default setting by strapping option.
ETHER_PHY_LSI_TYPE_KSZ8091RNB	Select configuration for KSZ8091RNB.
ETHER_PHY_LSI_TYPE_KSZ8041	Select configuration for KSZ8041.
ETHER_PHY_LSI_TYPE_DP83620	Select configuration for DP83620.
ETHER_PHY_LSI_TYPE_ICS1894	Select configuration for ICS1894.
ETHER_PHY_LSI_TYPE_GPY111	Select configuration for GPY111.
ETHER_PHY_LSI_TYPE_VSC8541	Select configuration for VSC8541.
ETHER_PHY_LSI_TYPE_KSZ8081MNX	Select configuration for KSZ8081MNX.
ETHER_PHY_LSI_TYPE_CUSTOM	Select configuration for User custom.

◆ ether_phy_lsi_clk_delay_t

enum ether_phy_lsi_clk_delay_t	
Transmit /Receive Timing Skew (RGMII)	
Enumerator	
ETHER_PHY_LSI_CLK_DELAY_0_0_NS	0.0 ns
ETHER_PHY_LSI_CLK_DELAY_0_2_NS	0.2 ns
ETHER_PHY_LSI_CLK_DELAY_0_5_NS	0.5 ns
ETHER_PHY_LSI_CLK_DELAY_0_8_NS	0.8 ns
ETHER_PHY_LSI_CLK_DELAY_1_0_NS	1.0 ns
ETHER_PHY_LSI_CLK_DELAY_1_1_NS	1.1 ns
ETHER_PHY_LSI_CLK_DELAY_1_5_NS	1.5 ns
ETHER_PHY_LSI_CLK_DELAY_1_7_NS	1.7 ns
ETHER_PHY_LSI_CLK_DELAY_2_0_NS	2.0 ns
ETHER_PHY_LSI_CLK_DELAY_2_3_NS	2.3 ns
ETHER_PHY_LSI_CLK_DELAY_2_5_NS	2.5 ns
ETHER_PHY_LSI_CLK_DELAY_2_6_NS	2.6 ns
ETHER_PHY_LSI_CLK_DELAY_3_0_NS	3.0 ns
ETHER_PHY_LSI_CLK_DELAY_3_4_NS	3.4 ns
ETHER_PHY_LSI_CLK_DELAY_3_5_NS	3.5 ns

◆ ether_phy_flow_control_t

enum ether_phy_flow_control_t	
Flow control functionality	
Enumerator	
ETHER_PHY_FLOW_CONTROL_DISABLE	Disable flow control functionality.
ETHER_PHY_FLOW_CONTROL_ENABLE	Enable flow control functionality with pause frames.

◆ ether_phy_link_speed_t

enum ether_phy_link_speed_t	
Link speed	
Enumerator	
ETHER_PHY_LINK_SPEED_NO_LINK	Link is not established.
ETHER_PHY_LINK_SPEED_10H	Link status is 10Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_10F	Link status is 10Mbit/s and full duplex.
ETHER_PHY_LINK_SPEED_100H	Link status is 100Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_100F	Link status is 100Mbit/s and full duplex.
ETHER_PHY_LINK_SPEED_1000H	Link status is 1000Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_1000F	Link status is 1000Mbit/s and full duplex.

◆ ether_phy_mii_type_t

enum ether_phy_mii_type_t	
Media-independent interface	
Enumerator	
ETHER_PHY_MII_TYPE_MII	MII.
ETHER_PHY_MII_TYPE_RMII	RMII.
ETHER_PHY_MII_TYPE_GMII	GMII.
ETHER_PHY_MII_TYPE_RGMII	RGMII.

◆ ether_phy_event_t

enum ether_phy_event_t	
Event code of callback function	
Enumerator	
ETHER_PHY_EVENT_PREEMPTION_LINK_VERIFY_FAIL	Preemption link verification failed.
ETHER_PHY_EVENT_PREEMPTION_LINK_VERIFY_SUCCESS	Preemption link verification succeeded.
ETHER_PHY_EVENT_PREEMPTION_LINK_VERIFY_RECEIVE	Preemption link verification frame received.

5.3.11.3 Ethernet Switch Interface

Interfaces » Networking

Detailed Description

Interface for Ethernet Switch functions.

Summary

The Ether Switch module provides an API for ethernet switch peripheral. And the general ethernet switch peripheral have forwarding functionality.

Data Structures

struct	ether_switch_callback_args_t

struct	ether_switch_cfg_t
struct	ether_switch_api_t
struct	ether_switch_instance_t

Typedefs

typedef void	ether_switch_ctrl_t

Enumerations

enum	ether_switch_event_t

Data Structure Documentation

◆ ether_switch_callback_args_t

struct ether_switch_callback_args_t		
Callback function parameter data		
Data Fields		
uint32_t	channel	Device channel number.
uint32_t	ports	Bitmap of ports on which the interrupt occurred.
uint32_t	queue_index	Queue index where a interrupt occurs.
ether_switch_event_t	event	The event can be used to identify what caused the callback.
void *	p_context	Placeholder for user data. Set in ether_switch_api_t::open function in ether_switch_cfg_t.

◆ ether_switch_cfg_t

struct ether_switch_cfg_t	
Configuration parameters.	
Data Fields	
uint8_t	channel
	Channel.
IRQn_Type	irq
	MCU interrupt number.

uint8_t	ipl
	MCU interrupt priority.
void(*	p_callback)(ether_switch_callback_args_t *p_args)
	Callback provided when an ISR occurs.
void *	p_context
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_context

void* ether_switch_cfg_t::p_context

Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t.

◆ ether_switch_api_t

struct ether_switch_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ether_switch_ctrl_t *const p_ctrl, ether_switch_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ether_switch_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* ether_switch_api_t::open) (ether_switch_ctrl_t *const p_ctrl, ether_switch_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* ether_switch_api_t::close) (ether_switch_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ ether_switch_instance_t

struct ether_switch_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
ether_switch_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ether_switch_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ether_switch_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ether_switch_ctrl_t

typedef void ether_switch_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ ether_switch_event_t

enum ether_switch_event_t	
Events that can trigger a callback function	
Enumerator	
ETHER_SWITCH_EVENT_RX_COMPLETE	A descriptor complete to receive a flame.
ETHER_SWITCH_EVENT_TX_COMPLETE	A descriptor complete to transmit a flame.
ETHER_SWITCH_EVENT_RX_QUEUE_FULL	A RX descriptor queue is full.
ETHER_SWITCH_EVENT_RX_MESSAGE_LOST	Receive a frame when a RX descriptor queue is full.
ETHER_SWITCH_EVENT_TAS_ERROR	TAS gate error.

5.3.11.4 GPTP Interface

Interfaces » Networking

Detailed Description

Interface for gPTP timing.

Summary

The gPTP API provides a generic interface for gPTP timer operation.

Data Structures

struct	gptp_timer_value_t
struct	gptp_timer_cfg_t
struct	gptp_callback_args_t
struct	gptp_cfg_t
struct	gptp_api_t
struct	gptp_instance_t

Typedefs

typedef void	gptp_ctrl_t

Data Structure Documentation

◆ gptp_timer_value_t

struct gptp_timer_value_t		
Timer value.		
Data Fields		
uint16_t	time_sec_upper	Second(Upper 16 bit).
uint32_t	time_sec_lower	Second(Lower 32 bit).
uint32_t	time_nsec	Nanosecond.

◆ gptp_timer_cfg_t

struct gptp_timer_cfg_t		
Configuration of gPTP timer.		
Data Fields		
uint8_t	clock_period	Timer increment value.

◆ gptp_callback_args_t

struct gptp_callback_args_t		
GPTP callback arguments definitions.		
Data Fields		
void *	p_context	Placeholder for user data. Set in gptp_api_t::open function in gptp_cfg_t.

◆ gptp_cfg_t

struct gptp_cfg_t	
Configuration parameters.	
Data Fields	
void(*	p_callback)(gptp_callback_args_t *p_args)
	Pointer to callback function.
void *	p_context
	Placeholder for user data. Passed to the user callback in gptp_callback_args_t.
void const *	p_extend

	Placeholder for user extension.

◆ gptp_api_t

struct gptp_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(gptp_ctrl_t *const p_ctrl, gptp_cfg_t const *const p_cfg)
fsp_err_t(*	close)(gptp_ctrl_t *const p_ctrl)
fsp_err_t(*	timerCfg)(gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_cfg_t const *const p_timer_cfg)
fsp_err_t(*	start)(gptp_ctrl_t *const p_ctrl, uint8_t timer)
fsp_err_t(*	stop)(gptp_ctrl_t *const p_ctrl, uint8_t timer)
fsp_err_t(*	timerValueGet)(gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_value_t *const p_timer_value)
fsp_err_t(*	timerOffsetSet)(gptp_ctrl_t *const p_ctrl, uint8_t timer, int64_t offset)
fsp_err_t(*	timerRateSet)(gptp_ctrl_t *const p_ctrl, uint8_t timer, uint32_t rate)
fsp_err_t(*	callbackSet)(gptp_ctrl_t *const p_ctrl, void(*p_callback)(gptp_callback_args_t *), void *const p_context, gptp_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* gptp_api_t::open) (gptp_ctrl_t *const p_ctrl, gptp_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* gptp_api_t::close) (gptp_ctrl_t *const p_ctrl)

Close driver.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ timerCfg

fsp_err_t(* gptp_api_t::timerCfg) (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_cfg_t const *const p_timer_cfg)

Configure gptp timer parameters.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	timer	Timer index.
[in]	p_timer_cfg	Configuration of the timer.

◆ start

fsp_err_t(* gptp_api_t::start) (gptp_ctrl_t *const p_ctrl, uint8_t timer)

Start gptp timer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	timer	Timer index.

◆ stop

fsp_err_t(* gptp_api_t::stop) (gptp_ctrl_t *const p_ctrl, uint8_t timer)

Stop gptp timer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	timer	Timer index.

◆ timerValueGet

fsp_err_t(* gptp_api_t::timerValueGet) (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_value_t *const p_timer_value)

Get the current time value to gptp timer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	timer	Timer index.
[out]	p_timer_value	Pointer to timer value structure.

◆ timerOffsetSet

fsp_err_t(* gptp_api_t::timerOffsetSet) (gptp_ctrl_t *const p_ctrl, uint8_t timer, int64_t offset)

Set time offset correction to gptp timer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	timer	Timer index.
[in]	offset	Time offset value.

◆ timerRateSet

fsp_err_t(* gptp_api_t::timerRateSet) (gptp_ctrl_t *const p_ctrl, uint8_t timer, uint32_t rate)

Set clock rate correction to gptp timer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	timer	Timer index.
[in]	rate	Clock rate value.

◆ callbackSet

fsp_err_t(* gptp_api_t::callbackSet) (gptp_ctrl_t *const p_ctrl, void(*p_callback)(gptp_callback_args_t *), void *const p_context, gptp_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_callback	Callback function.
[in]	p_context	Pointer to send to callback function.
[in]	p_callback_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ gptp_instance_t

struct gptp_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
gptp_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
gptp_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
gptp_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ gptp_ctrl_t

typedef void gptp_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

5.3.11.5 EtherCAT SSC Interface

Interfaces » Networking

Detailed Description

Interface for EtherCAT SSC functions.

Summary

The EtherCAT SSC port module (rm_ethercat_ssc_port) provides an API for EtherCAT Slave Stack Code that use the EtherCAT Slave Controler peripheral.

Data Structures

struct	ethercat_ssc_port_callback_args_t
struct	ethercat_ssc_port_cfg_t
struct	ethercat_ssc_port_api_t
struct	ethercat_ssc_port_instance_t

Typedefs

typedef void	ethercat_ssc_port_ctrl_t

Enumerations

enum	ethercat_ssc_port_event_t

Data Structure Documentation

◆ ethercat_ssc_port_callback_args_t

struct ethercat_ssc_port_callback_args_t		
Callback function parameter data		
Data Fields		
ethercat_ssc_port_event_t	event	Event code.
void *	p_context	Placeholder for user data. Set in ethercat_ssc_port_api_t::open function in ethercat_ssc_port_cfg_t.

◆ ethercat_ssc_port_cfg_t

struct ethercat_ssc_port_cfg_t	
Configuration parameters.	
Data Fields	
uint32_t	reset_hold_time
	PHY Reset signal hold time (ms)

uint32_t	reset_wait_time
	Wait time after PHY reset relase (us)
uint32_t	address_offset
	Offset of the PHY address.
IRQn_Type	common_irq
	EtherCAT IRQ interrupt number.
uint8_t	common_ipl
	EtherCAT interrupt priority.
IRQn_Type	sync0_irq
	EtherCAT Sync0 IRQ interrupt number.
uint8_t	sync0_ipl
	EtherCAT Sync0 interrupt priority.
IRQn_Type	sync1_irq
	EtherCAT Sync1 IRQ interrupt number.
uint8_t	sync1_ipl
	EtherCAT Sync1 interrupt priority.
void(*	p_callback)(ethercat_ssc_port_callback_args_t *p_args)
	Callback provided when an ISR occurs.

timer_instance_t const *	p_timer_instance
	Pointer to Timer instance.
void *	p_context
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_context

void* ethercat_ssc_port_cfg_t::p_context

Placeholder for user data. Passed to the user callback in ethercat_ssc_port_callback_args_t.

◆ ethercat_ssc_port_api_t

struct ethercat_ssc_port_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ethercat_ssc_port_ctrl_t *const p_api_ctrl, ethercat_ssc_port_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ethercat_ssc_port_ctrl_t *const p_api_ctrl)

Field Documentation

◆ open

fsp_err_t(* ethercat_ssc_port_api_t::open) (ethercat_ssc_port_ctrl_t *const p_api_ctrl, ethercat_ssc_port_cfg_t const *const p_cfg)

Open driver.

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to EtherCAT SSC port configuration structure.

◆ close

fsp_err_t(* ethercat_ssc_port_api_t::close) (ethercat_ssc_port_ctrl_t *const p_api_ctrl)

Close driver.

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ ethercat_ssc_port_instance_t

struct ethercat_ssc_port_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
ethercat_ssc_port_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ethercat_ssc_port_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ethercat_ssc_port_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ethercat_ssc_port_ctrl_t

typedef void ethercat_ssc_port_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ ethercat_ssc_port_event_t

enum ethercat_ssc_port_event_t	
Event code of callback function	
Enumerator	
ETHERCAT_SSC_PORT_EVENT_ESC_CAT_INTERRUPT	EtherCAT Interrupt event.
ETHERCAT_SSC_PORT_EVENT_ESC_SYNC0_INTERRUPT	EtherCAT Sync0 Interrupt event.
ETHERCAT_SSC_PORT_EVENT_ESC_SYNC1_INTERRUPT	EtherCAT Sync1 Interrupt event.

5.3.12 Power

Interfaces

Detailed Description

Power Interfaces.

Modules

	Low Power Modes Interface Interface for accessing low power modes.

5.3.12.1 Low Power Modes Interface

Interfaces » Power

Detailed Description

Interface for accessing low power modes.

Summary

This section defines the API for the LPM (Low Power Mode) Driver. The LPM Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPM driver supports configuration of MCU low power modes using the LPM hardware peripheral. The LPM driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCUs.

Data Structures

struct	lpm_ldo_standby_cfg_t
struct	lpm_cfg_t
struct	lpm_api_t
struct	lpm_instance_t

Typedefs

typedef void	lpm_ctrl_t

Enumerations

enum	lpm_mode_t
enum	lpm_snooze_request_t
enum	lpm_snooze_end_t
enum	lpm_snooze_cancel_t
enum	lpm_snooze_dtc_t
enum	lpm_standby_wake_source_t
enum	lpm_io_port_t
enum	lpm_power_supply_t
enum	lpm_deep_standby_cancel_edge_t
enum	lpm_deep_standby_cancel_source_t
enum	lpm_output_port_enable_t
enum	lpm_ldo_standby_operation_t
enum	lpm_flash_mode_select_t
enum	lpm_hoco_startup_speed_t
enum	lpm_standby_sosc_t

Data Structure Documentation

◆ lpm_ldo_standby_cfg_t

struct lpm_ldo_standby_cfg_t		
Configure LDO operation in standby mode.		
Data Fields		
lpm_ldo_standby_operation_t	pll1_ldo	Configure the state of PLL1 LDO in standby mode.
lpm_ldo_standby_operation_t	pll2_ldo	Configure the state of PLL2 LDO in standby mode.
lpm_ldo_standby_operation_t	hoco_ldo	Configure the state of HOCO LDO in standby mode.

◆ lpm_cfg_t

struct lpm_cfg_t		
User configuration structure, used in open function		
Data Fields		
lpm_mode_t	low_power_mode	Low Power Mode
lpm_standby_wake_source_bits_t	standby_wake_sources	Bitwise list of sources to wake from deep sleep and standby mode
lpm_snooze_dtc_t	dtc_state_in_snooze	State of DTC in snooze mode, enabled or disabled
void const *	p_extend	Placeholder for extension.

◆ lpm_api_t

struct lpm_api_t	
LPM driver structure. General LPM functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t(*	close)(lpm_ctrl_t *const p_ctrl)
fsp_err_t(*	lowPowerReconfigure)(lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t(*	lowPowerModeEnter)(lpm_ctrl_t *const p_ctrl)
fsp_err_t(*	ioKeepClear)(lpm_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* lpm_api_t::open) (lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg)

Initialization function

◆ close

fsp_err_t(* lpm_api_t::close) (lpm_ctrl_t *const p_ctrl)

Initialization function

◆ lowPowerReconfigure

fsp_err_t(* lpm_api_t::lowPowerReconfigure) (lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg)

Configure a low power mode.

Parameters

[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ lowPowerModeEnter

fsp_err_t(* lpm_api_t::lowPowerModeEnter) (lpm_ctrl_t *const p_ctrl)

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.

◆ ioKeepClear

fsp_err_t(* lpm_api_t::ioKeepClear) (lpm_ctrl_t *const p_ctrl)

Clear the IOKEEP bit after deep software standby.

◆ lpm_instance_t

struct lpm_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
lpm_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
lpm_cfg_t const *const	p_cfg	Pointer to the configuration structure for this instance.
lpm_api_t const *const	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ lpm_ctrl_t

typedef void lpm_ctrl_t

LPM control block. Allocate an instance specific control block to pass into the LPM API calls.

Enumeration Type Documentation

◆ lpm_mode_t

enum lpm_mode_t	
Low power modes	
Enumerator	
LPM_MODE_SLEEP	Sleep mode.
LPM_MODE_DEEP_SLEEP	Deep Sleep mode.
LPM_MODE_STANDBY	Software Standby mode.
LPM_MODE_STANDBY_SNOOZE	Software Standby mode with Snooze mode enabled.
LPM_MODE_DEEP	Deep Software Standby mode.

◆ lpm_snooze_request_t

enum lpm_snooze_request_t	
Snooze request sources	
Enumerator	
LPM_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPM_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.

LPM_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPM_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPM_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.
LPM_SNOOZE_REQUEST_RTC_ALARM1	Enable RTC alarm 1 snooze request.
LPM_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPM_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPM_SNOOZE_REQUEST_AGTW1_UNDERFLOW	Enable AGTW1 underflow snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGTW1_COMPARE_A	Enable AGTW1 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPM_SNOOZE_REQUEST_AGTW1_COMPARE_B	Enable AGTW1 compare match B snooze request.
LPM_SNOOZE_REQUEST_AGT3_UNDERFLOW	Enable AGT3 underflow snooze request.
LPM_SNOOZE_REQUEST_AGT3_COMPARE_A	Enable AGT3 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGT3_COMPARE_B	Enable AGT3 compare match B snooze request.

◆ lpm_snooze_end_t

enum lpm_snooze_end_t	
Snooze end control	
Enumerator	
LPM_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPM_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPM_SNOOZE_END_AGTW1_UNDERFLOW	AGTW1 underflow.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPM_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPM_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPM_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.
LPM_SNOOZE_END_SCI0_ADDRESS_MISMATCH	SCI0 address mismatch.
LPM_SNOOZE_END_AGT3_UNDERFLOW	AGT3 underflow.

◆ lpm_snooze_cancel_t

enum lpm_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPM_SNOOZE_CANCEL_SOURCE_NONE	No snooze cancel source.
LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match.
LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch.
LPM_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion.
LPM_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt.

◆ lpm_snooze_dtc_t

enum lpm_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPM_SNOOZE_DTC_DISABLE	Disable DTC operation.
LPM_SNOOZE_DTC_ENABLE	Enable DTC operation.

◆ lpm_standby_wake_source_t

enum lpm_standby_wake_source_t	
Wake from deep sleep or standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPM_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPM_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPM_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPM_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPM_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPM_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPM_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPM_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPM_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPM_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPM_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPM_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPM_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPM_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPM_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.

LPM_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPM_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPM_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPM_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPM_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPM_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPM_STANDBY_WAKE_SOURCE_VRTC	LVDVRTC interrupt.
LPM_STANDBY_WAKE_SOURCE_EXLVD	LVDEXLVD interrupt.
LPM_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_RTCALM1	RTC Alarm interrupt 1.
LPM_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPM_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPM_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.
LPM_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPM_STANDBY_WAKE_SOURCE_AGTW0UD	AGTW0 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGTW1UD	AGTW1 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGTW1CA	AGTW1 Compare Match A interrupt.
LPM_STANDBY_WAKE_SOURCE_AGTW1CB	AGTW1 Compare Match B interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 Compare Match A interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 Compare Match B interrupt.
LPM_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT0UD	AGT0 Underflow interrupt.

LPM_STANDBY_WAKE_SOURCE_AGT3UD	AGT3 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1UD_S	AGT1 Underflow interrupt for specific board.
LPM_STANDBY_WAKE_SOURCE_AGT3CA	AGT3 Compare Match A interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT2UD	AGT2 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT3CB	AGT3 Compare Match B interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT3UD_S	AGT3 Underflow interrupt for specific board.
LPM_STANDBY_WAKE_SOURCE_COMPHS0	Comparator-HS0 Interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT4UD	AGT4 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT5UD	AGT5 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT6UD	AGT6 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT7UD	AGT7 Underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_SOSTD	SOSTD interrupt.
LPM_STANDBY_WAKE_SOURCE_ULP0U	ULPT0 Underflow Interrupt.
LPM_STANDBY_WAKE_SOURCE_ULP0A	ULPT0 Compare Match A Interrupt.
LPM_STANDBY_WAKE_SOURCE_ULP0B	ULPT0 Compare Match B Interrupt.
LPM_STANDBY_WAKE_SOURCE_I3C0	I3C0 address match interrupt.
LPM_STANDBY_WAKE_SOURCE_ULP1U	ULPT1 Underflow Interrupt.
LPM_STANDBY_WAKE_SOURCE_ULP1A	ULPT1 Compare Match A Interrupt.
LPM_STANDBY_WAKE_SOURCE_ULP1B	ULPT1 Compare Match B Interrupt.

◆ lpm_io_port_t

enum lpm_io_port_t	
I/O port state after Deep Software Standby mode	
Enumerator	
LPM_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state
LPM_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

◆ lpm_power_supply_t

enum lpm_power_supply_t	
Power supply control	
Enumerator	
LPM_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE1	Power to the standby RAM, Low-speed on-chip oscillator, Programmable Voltage Detection Unit 0, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE2	Power to standby RAM, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode. Power to the Programmable Voltage Detection Unit 0 is supplied in deep software standby mode.

LPM_POWER_SUPPLY_DEEP_STANDBY_MODE3	Power to standby RAM, Programmable Voltage Detection Unit 0, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode.
LPM_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled
LPM_POWER_SUPPLY_STANDBY_MODE1	Power to standby ESWM, EtherCAT slave controller ESC, PDC, CANFDG, CANFD, SDHIA is supplied in deep software standby mode.
LPM_POWER_SUPPLY_STANDBY_MODE2	Power to standby ESWM, EtherCAT slave controller ESC, PDC, CANFDG, CANFD, SDHIA is disabled in software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE1	Power to the standby RAM, Low-speed on-chip oscillator, Programmable Voltage Detection Unit 0, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE2	Power to standby RAM, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode. Power to the Programmable Voltage Detection Unit 0 is supplied in deep software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE3	Power to standby RAM, Programmable Voltage Detection Unit 0, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode.
LPM_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode

LPM_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled
LPM_POWER_SUPPLY_STANDBY_MODE1	Power to standby ESWM, EtherCAT slave controller ESC, PDC, CANFDG, CANFD, SDHIA is supplied in deep software standby mode.
LPM_POWER_SUPPLY_STANDBY_MODE2	Power to standby ESWM, EtherCAT slave controller ESC, PDC, CANFDG, CANFD, SDHIA is disabled in software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE1	Power to the standby RAM, Low-speed on-chip oscillator, Programmable Voltage Detection Unit 0, and USBFS/HS resume detecting unit is supplied in deep software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE2	Power to standby RAM, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode. Power to the Programmable Voltage Detection Unit 0 is supplied in deep software standby mode.
LPM_POWER_SUPPLY_DEEP_STANDBY_MODE3	Power to standby RAM, Programmable Voltage Detection Unit 0, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode.

◆ lpm_deep_standby_cancel_edge_t

enum lpm_deep_standby_cancel_edge_t	
Deep Standby Interrupt Edge	
Enumerator	
LPM_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ14-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ14-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ16_RISING	IRQ16-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ16_FALLING	IRQ16-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ17_RISING	IRQ17-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ17_FALLING	IRQ17-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ18_RISING	IRQ18-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ18_FALLING	IRQ18-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ19_RISING	IRQ19-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ19_FALLING	IRQ19-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ20_RISING	IRQ20-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ20_FALLING	IRQ20-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ21_RISING	IRQ21-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ21_FALLING	IRQ21-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ22_RISING	IRQ22-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ22_FALLING	IRQ22-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ23_RISING	IRQ23-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ23_FALLING	IRQ23-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ24_RISING	IRQ24-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ24_FALLING	IRQ24-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ25_RISING	IRQ25-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ25_FALLING	IRQ25-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ26_RISING	IRQ26-DS Pin Rising Edge.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ26_FALLING	IRQ26-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ27_RISING	IRQ27-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ27_FALLING	IRQ27-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ28_RISING	IRQ28-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ28_FALLING	IRQ28-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ29_RISING	IRQ29-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ29_FALLING	IRQ29-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ30_RISING	IRQ30-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ30_FALLING	IRQ30-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ31_RISING	IRQ31-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ31_FALLING	IRQ31-DS Pin Falling Edge.

◆ lpm_deep_standby_cancel_source_t

enum lpm_deep_standby_cancel_source_t	
Deep Standby cancel sources	
Enumerator	
LPM_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2.
LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt.
LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI.
LPM_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume.
LPM_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume.
LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow.
LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT3	AGT3 Underflow.
LPM_DEEP_STANDBY_CANCEL_SOURCE_ULPT0	ULPT0 Overflow.
LPM_DEEP_STANDBY_CANCEL_SOURCE_ULPT1	ULPT1 Overflow.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IWDT	IWDT Underflow.
LPM_DEEP_STANDBY_CANCEL_SOURCE_VBATT	VBATT Tamper Detection.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ16	IRQ16.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ17	IRQ17.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ18	IRQ18.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ19	IRQ19.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ20	IRQ20.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ21	IRQ21.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ22	IRQ22.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ23	IRQ23.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ24	IRQ24.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ25	IRQ25.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ26	IRQ26.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ27	IRQ27.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ28	IRQ28.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ29	IRQ29.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ30	IRQ30.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ31	IRQ31.

◆ lpm_output_port_enable_t

enum lpm_output_port_enable_t	
Output port enable	
Enumerator	
LPM_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPM_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

◆ lpm_ldo_standby_operation_t

enum lpm_ldo_standby_operation_t	
Configure the behavior of an oscillator's LDO in standby mode.	
Enumerator	
LPM_LDO_STANDBY_OPERATION_DISABLED	The LDO is disabled in standby mode.
LPM_LDO_STANDBY_OPERATION_RETAINED	The LDO state is retained during standby mode.

◆ lpm_flash_mode_select_t

enum lpm_flash_mode_select_t	
Flash operating mode select.	
Enumerator	
LPM_FLASH_MODE_SELECT_ACTIVE	Flash active.
LPM_FLASH_MODE_SELECT_STOP	Flash stop.

◆ lpm_hoco_startup_speed_t

enum lpm_hoco_startup_speed_t	
Starting the high-speed on-chip oscillator at the times of release from SSTBY mode and of transitions to SNOOZE mode.	
Enumerator	
LPM_HOCO_STARTUP_SPEED_NORMAL_SPEED	Starting of the high-speed on-chip oscillator is at normal speed.
LPM_HOCO_STARTUP_SPEED_HIGH_SPEED	Starting of the high-speed on-chip oscillator is at high speed.

◆ lpm_standby_sosc_t

enum lpm_standby_sosc_t	
SOSC setting in SSTBY mode or in SNOOZE mode.	
Enumerator	
LPM_STANDBY_SOSC_ENABLE	Enables supply of SOSC clock to peripheral functions.
LPM_STANDBY_SOSC_DISABLE	Stops supply SOSC clock to peripheral functions other than the Realtime clock.

5.3.13 Security

Interfaces

Detailed Description

Security Interfaces.

Modules

	RSIP key injection Interface Interface for key injection by Renesas Secure IP (RSIP) functions.
	RSIP Interface Interface for Renesas Secure IP (RSIP) functions.

5.3.13.1 RSIP key injection Interface

Interfaces » Security

Detailed Description

Interface for key injection by Renesas Secure IP (RSIP) functions.

Summary

The RSIP key injection interface provides RSIP functionality.

Data Structures

struct	rsip_aes_wrapped_key_t
struct	rsip_chacha20_wrapped_key_t
struct	rsip_rsa2048_private_wrapped_key_t
struct	rsip_rsa3072_private_wrapped_key_t
struct	rsip_rsa4096_private_wrapped_key_t
struct	rsip_ecc_private_wrapped_key_t
struct	rsip_hmac_sha_wrapped_key_t
struct	rsip_key_injection_api_t

Enumerations

enum	rsip_key_injection_type_t

Data Structure Documentation

◆ rsip_aes_wrapped_key_t

struct rsip_aes_wrapped_key_t		
AES128/256 wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_AES256_XTS_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_chacha20_wrapped_key_t

struct rsip_chacha20_wrapped_key_t		
CHACHA20 wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_CHACHA20_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_rsa2048_private_wrapped_key_t

struct rsip_rsa2048_private_wrapped_key_t		
RSA 2048bit private wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_RSA2048_PRIVATE_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_rsa3072_private_wrapped_key_t

struct rsip_rsa3072_private_wrapped_key_t		
RSA 3072bit private wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_RSA3072_PRIVATE_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_rsa4096_private_wrapped_key_t

struct rsip_rsa4096_private_wrapped_key_t		
RSA 4096bit private wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_RSA4096_PRIVATE_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_ecc_private_wrapped_key_t

struct rsip_ecc_private_wrapped_key_t		
ECC 256/384/512/521 private wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_ECCP521_PRIVATE_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_hmac_sha_wrapped_key_t

struct rsip_hmac_sha_wrapped_key_t		
HMAC 256/342/512 wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
uint32_t	value[R_RSIP_HMAC_SHA512_KEY_INDEX_WORD_SIZE]	Wrapped key value.

◆ rsip_key_injection_api_t

struct rsip_key_injection_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	AES128_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	AES192_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	AES256_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	AES128_XTS_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	AES256_XTS_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ChaCha20_InitialKeyWrap)(rsip_key_injection_type_t const

	key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_chacha20_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	RSA2048_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa2048_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	RSA3072_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa3072_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	RSA4096_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa4096_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_secp256r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_secp384r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_secp521r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_secp256k1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

fsp_err_t(*	ECC_brainpoolP256r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_brainpoolP384r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_brainpoolP512r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	ECC_Ed25519_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	HMAC_SHA224_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	HMAC_SHA256_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	HMAC_SHA384_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	HMAC_SHA512_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const

	p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	HMAC_SHA512_224_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	HMAC_SHA512_256_InitialKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

Field Documentation

◆ AES128_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES128_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 128-bit AES wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	128-bit AES wrapped key

◆ AES192_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES192_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 192-bit AES wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	192-bit AES wrapped key

◆ AES256_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES256_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit AES wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit AES wrapped key

◆ AES128_XTS_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES128_XTS_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 128-bit AES-XTS wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	128-bit AES-XTS wrapped key

◆ AES256_XTS_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES256_XTS_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit AES-XTS wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit AES-XTS wrapped key

◆ ChaCha20_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ChaCha20_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_chacha20_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit ChaCha20 wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit ChaCha20 wrapped key

◆ RSA2048_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::RSA2048_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa2048_private_wrapped_key_t *const p_wrapped_key)

This API outputs 2048-bit RSA wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	2048-bit RSA wrapped private key

◆ RSA3072_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::RSA3072_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa3072_private_wrapped_key_t *const p_wrapped_key)

This API outputs 3072-bit RSA wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	3072-bit RSA wrapped private key

◆ RSA4096_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::RSA4096_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa4096_private_wrapped_key_t *const p_wrapped_key)

This API outputs 4096-bit RSA wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	4096-bit RSA wrapped private key

◆ ECC_secp256r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp256r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit ECC wrapped private key

◆ ECC_secp384r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp384r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 384-bit ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	384-bit ECC wrapped private key

◆ ECC_secp521r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp521r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 521-bit ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	521-bit ECC wrapped private key

◆ ECC_secp256k1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp256k1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit ECC wrapped private key

◆ ECC_brainpoolP256r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_brainpoolP256r1_InitialPrivateKeyWrap)(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit brainpool ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit ECC wrapped private key

◆ ECC_brainpoolP384r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_brainpoolP384r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 384-bit brainpool ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	384-bit ECC wrapped private key

◆ ECC_brainpoolP512r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_brainpoolP512r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 512-bit brainpool ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	512-bit ECC wrapped private key

◆ ECC_Ed25519_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_Ed25519_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit Edwards curve ECC wrapped private key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit ECC wrapped private key

◆ HMAC_SHA224_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA224_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 224-bit HMAC-SHA wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	224-bit HMAC-SHA wrapped key

◆ HMAC_SHA256_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA256_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit HMAC-SHA wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	256-bit HMAC-SHA wrapped key

◆ HMAC_SHA384_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA384_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 384-bit HMAC-SHA wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	384-bit HMAC-SHA wrapped key

◆ HMAC_SHA512_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA512_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 512-bit HMAC-SHA wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	512-bit HMAC-SHA wrapped key

◆ HMAC_SHA512_224_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA512_224_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 512/224-bit HMAC-SHA wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	512/224-bit HMAC-SHA wrapped key

◆ HMAC_SHA512_256_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA512_256_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 512/256-bit HMAC-SHA wrapped key.

Parameters

[in]	key_injection_type	Selection key injection type when generating wrapped key
[in]	p_wrapped_user_factory_programming_key	Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified.
[in]	p_initial_vector	Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified.
[in]	p_user_key	User key. If key injection type is not plain, it must be encrypted and have MAC appended.
[out]	p_wrapped_key	512/256-bit HMAC-SHA wrapped key

Enumeration Type Documentation

◆ rsip_key_injection_type_t

enum rsip_key_injection_type_t

Key injection type.

Enumerator	
RSIP_KEY_INJECTION_TYPE_ENCRYPTED	Input encrypted user key.
RSIP_KEY_INJECTION_TYPE_PLAIN	Input plain user key.

5.3.13.2 RSIP Interface

Interfaces » Security

Detailed Description

Interface for Renesas Secure IP (RSIP) functions.

Summary

The RSIP interface provides RSIP functionality.

Data Structures

struct	rsip_dlms_apdu_data_t
struct	rsip_wrapped_key_t
struct	rsip_wrapped_dkm_t
struct	rsip_cfg_t
struct	rsip_api_t
struct	rsip_instance_t

Typedefs

typedef void	rsip_sha_handle_abstractor_t
typedef void	rsip_hmac_handle_abstractor_t
typedef void	rsip_kdf_sha_handle_abstractor_t
typedef void	rsip_kdf_hmac_handle_abstractor_t
typedef void	rsip_wrapped_secret_abstractor_t
typedef void	rsip_verified_cert_info_abstractor_t
typedef void	rsip_ctrl_t

Enumerations

enum	rsip_key_type_t
enum	rsip_aes_cipher_mode_t
enum	rsip_aes_aead_mode_t

enum	rsip_chacha_poly_mode_t
enum	rsip_aes_mac_mode_t
enum	rsip_hash_type_t
enum	rsip_mgf_type_t
enum	rsip_rsa_salt_length_t
enum	rsip_initial_vector_type_t
enum	rsip_otf_channel_t
enum	rsip_dlms_ret_t

Data Structure Documentation

◆ rsip_dlms_apdu_data_t

struct rsip_dlms_apdu_data_t		
APDU data structure for DLMS/COSEM API.		
Data Fields		
uint8_t	part_1[3]	Buffer to store the first 3 bytes of the decrypted ADPU (APDU Choice, Key Flag, and Key Length)
uint8_t	part_2[14]	Buffer to store the decrypted ADPU, excluding part_1 and the dedicated key.

◆ rsip_wrapped_key_t

struct rsip_wrapped_key_t		
Wrapped key structure for all supported algorithms.		
Data Fields		
rsip_key_type_t	type	Key type.
void *	p_value	Key value.

◆ rsip_wrapped_dkm_t

struct rsip_wrapped_dkm_t
Wrapped Derived Keying Material (DKM) structure for KDF APIs.

◆ rsip_cfg_t

struct rsip_cfg_t		
User configuration structure, used in open function		
Data Fields		
void const *	p_extend	Hardware-dependent configuration.

◆ rsip_api_t

struct rsip_api_t	
RSIP driver structure. General RSIP functions implemented at the HAL layer follow this API.	
Data Fields	
fsp_err_t(*	open)(rsip_ctrl_t *const p_ctrl, rsip_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rsip_ctrl_t *const p_ctrl)
fsp_err_t(*	randomNumberGenerate)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_random)
fsp_err_t(*	keyGenerate)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	keyPairGenerate)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_public_key, rsip_wrapped_key_t *const p_wrapped_private_key)
fsp_err_t(*	encryptedKeyWrap)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_key_update_key, void const *const p_initial_vector, void const *const p_encrypted_key, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	rfc3394_KeyWrap)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, rsip_wrapped_key_t const *const p_wrapped_target_key, uint8_t *const p_rfc3394_wrapped_target_key)
fsp_err_t(*	rfc3394_KeyUnwrap)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, uint8_t const *const p_rfc3394_wrapped_target_key, rsip_wrapped_key_t *const p_wrapped_target_key)
fsp_err_t(*	publicKeyExport)(rsip_wrapped_key_t const *const

	p_wrapped_public_key, uint8_t *const p_raw_public_key)
fsp_err_t(*	aesCipherInit)(rsip_ctrl_t *const p_ctrl, rsip_aes_cipher_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_initial_vector)
fsp_err_t(*	aesCipherUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint8_t *const p_output, uint32_t const length)
fsp_err_t(*	aesCipherFinish)(rsip_ctrl_t *const p_ctrl)
fsp_err_t(*	aesAeadInit)(rsip_ctrl_t *const p_ctrl, rsip_aes_aead_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)
fsp_err_t(*	aesAeadLengthsSet)(rsip_ctrl_t *const p_ctrl, uint32_t const total_aad_length, uint32_t const total_text_length, uint32_t const tag_length)
fsp_err_t(*	aesAeadAadUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)
fsp_err_t(*	aesAeadUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t(*	aesAeadFinish)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)
fsp_err_t(*	aesAeadVerify)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)
fsp_err_t(*	aesMacInit)(rsip_ctrl_t *const p_ctrl, rsip_aes_mac_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key)
fsp_err_t(*	aesMacUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t(*	aesMacSignFinish)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)

fsp_err_t(*	aesMacVerifyFinish)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)
fsp_err_t(*	chacha20Init)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const counter)
fsp_err_t(*	chacha20Update)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t(*	chacha20Finish)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t(*	chacha20Poly1305Init)(rsip_ctrl_t *const p_ctrl, rsip_chacha_poly_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)
fsp_err_t(*	chacha20Poly1305AadUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)
fsp_err_t(*	chacha20Poly1305Update)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)
fsp_err_t(*	chacha20Poly1305Finish)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)
fsp_err_t(*	chacha20Poly1305Verify)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)
fsp_err_t(*	ecdsaSign)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_hash, uint8_t *const p_signature)
fsp_err_t(*	ecdsaVerify)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)

fsp_err_t(*	eddsaSign)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t *const p_signature)
fsp_err_t(*	eddsaVerify)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t const *const p_signature)
fsp_err_t(*	ecdhKeyAgree)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret)
fsp_err_t(*	ecdhPlainKeyAgree)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_plain_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret)
fsp_err_t(*	rsaEncrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint8_t *const p_cipher)
fsp_err_t(*	rsaDecrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain)
fsp_err_t(*	rsaesPkcs1V15Encrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)
fsp_err_t(*	rsaesPkcs1V15Decrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)
fsp_err_t(*	rsaesOaepEncrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)

fsp_err_t(*	rsaesOaepDecrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)
fsp_err_t(*	rsassaPkcs1V15Sign)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t *const p_signature)
fsp_err_t(*	rsassaPkcs1V15Verify)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t(*	rsassaPssSign)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t *const p_signature)
fsp_err_t(*	rsassaPssVerify)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t(*	shaCompute)(rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_digest)
fsp_err_t(*	shaInit)(rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)
fsp_err_t(*	shaUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t(*	shaFinish)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_digest)
fsp_err_t(*	shaSuspend)(rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t *const p_handle)

fsp_err_t(*	shaResume)(rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t const *const p_handle)
fsp_err_t(*	hmacCompute)(rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_mac)
fsp_err_t(*	hmacVerify)(rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t const *const p_mac, uint32_t const mac_length)
fsp_err_t(*	hmacInit)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)
fsp_err_t(*	hmacUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t(*	hmacSignFinish)(rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)
fsp_err_t(*	hmacVerifyFinish)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)
fsp_err_t(*	hmacSuspend)(rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t *const p_handle)
fsp_err_t(*	hmacResume)(rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t const *const p_handle)
fsp_err_t(*	pkiEcdsaCertVerify)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)
fsp_err_t(*	pkiVerifiedCertInfoExport)(rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t *const p_verified_cert_info)
fsp_err_t(*	pkiVerifiedCertInfoImport)(rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t const *const p_verified_cert_info)

fsp_err_t(*	pkiCertKeyImport)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_cert, uint32_t const cert_length, uint8_t const *const p_key_param1, uint32_t const key_param1_length, uint8_t const *const p_key_param2, uint32_t const key_param2_length, rsip_hash_type_t const hash_function, rsip_wrapped_key_t *const p_wrapped_public_key)
fsp_err_t(*	kdfShaInit)(rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)
fsp_err_t(*	kdfShaEcdhSecretUpdate)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret)
fsp_err_t(*	kdfShaUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t(*	kdfShaFinish)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)
fsp_err_t(*	kdfShaSuspend)(rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t *const p_handle)
fsp_err_t(*	kdfShaResume)(rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t const *const p_handle)
fsp_err_t(*	kdfHmacDkmKeyImport)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const key_length, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	kdfHmacEcdhSecretKeyImport)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	kdfHmacInit)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)
fsp_err_t(*	kdfHmacDkmUpdate)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm)

fsp_err_t(*	kdfHmacEcdhSecretUpdate)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret)
fsp_err_t(*	kdfHmacUpdate)(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)
fsp_err_t(*	kdfHmacSignFinish)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)
fsp_err_t(*	kdfHmacSuspend)(rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t *const p_handle)
fsp_err_t(*	kdfHmacResume)(rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t const *const p_handle)
fsp_err_t(*	kdfDkmConcatenate)(rsip_wrapped_dkm_t *const p_wrapped_dkm1, rsip_wrapped_dkm_t const *const p_wrapped_dkm2, uint32_t const wrapped_dkm1_buffer_length)
fsp_err_t(*	kdfDerivedKeyImport)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const position, rsip_wrapped_key_t *const p_wrapped_key)
fsp_err_t(*	kdfDerivedIvWrap)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, rsip_initial_vector_type_t const initial_vector_type, uint32_t const position, uint8_t const *const p_tls_sequence_num, uint8_t *const p_wrapped_initial_vector)
fsp_err_t(*	otfInit)(rsip_ctrl_t *const p_ctrl, rsip_otf_channel_t const channel, rsip_wrapped_key_t *const p_wrapped_key, uint8_t const *const p_seed)
fsp_err_t(*	xdlmsInitiateRequestDecrypt)(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_apdu, uint8_t const *const p_nonce, uint8_t const *const p_aad, uint32_t const aad_length, rsip_wrapped_key_t *const p_dedicated_key, rsip_dlms_apdu_data_t *const p_apdu_data, rsip_dlms_ret_t *const p_dlms_ret)

Field Documentation

◆ open

fsp_err_t(* rsip_api_t::open) (rsip_ctrl_t *const p_ctrl, rsip_cfg_t const *const p_cfg)

Enables use of Renesas Secure IP functionality.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* rsip_api_t::close) (rsip_ctrl_t *const p_ctrl)

Disables use of Renesas Secure IP functionality.

Parameters

[in,out]	p_ctrl	Pointer to control block.

◆ randomNumberGenerate

fsp_err_t(* rsip_api_t::randomNumberGenerate) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_random)

Generates a 128-bit random number.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_random	128bit random numbers.

◆ keyGenerate

fsp_err_t(* rsip_api_t::keyGenerate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_key)

Generate a wrapped symmetric key from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in,out]	p_wrapped_key	Pointer to destination of wrapped key. The length depends on key type. Refer "Key Size Table".

◆ keyPairGenerate

fsp_err_t(* rsip_api_t::keyPairGenerate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_public_key, rsip_wrapped_key_t *const p_wrapped_private_key)

Generate a wrapped asymmetric key pair from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in,out]	p_wrapped_public_key	Key index for Public Key. The length depends on the key type. Refer "Key Size Table".
[in,out]	p_wrapped_private_key	Key index for Private Key. The length depends on the key type. Refer "Key Size Table".

◆ encryptedKeyWrap

fsp_err_t(* rsip_api_t::encryptedKeyWrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_key_update_key, void const *const p_initial_vector, void const *const p_encrypted_key, rsip_wrapped_key_t *const p_wrapped_key)

Decrypt the encrypted user key with Key Update Key (KUK) and wrap it with the Hardware Unique Key (HUK).

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_key_update_key	Pointer to Key Update Key.
[in]	p_initial_vector	Initialization vector when generating encrypted_key. The length is 16 bytes.
[in]	p_encrypted_key	Encrypted user key. The length depends on the key type. Refer "Key Size Table".
[in,out]	p_wrapped_key	Pointer to destination of wrapped key. The length depends on key type. Refer "Key Size Table".

◆ rfc3394_KeyWrap

fsp_err_t(* rsip_api_t::rfc3394_KeyWrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, rsip_wrapped_key_t const *const p_wrapped_target_key, uint8_t *const p_rfc3394_wrapped_target_key)

This function provides Key Wrap algorithm compliant with RFC3394. Using p_wrapped_kek to wrap p_wrapped_target_key, and output the result to p_rfc3394_wrapped_target_key.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_kek	Pointer to wrapped key-encryption-key used to RFC3394-wrap the target key.
[in]	p_wrapped_target_key	Pointer to wrapped target key to be RFC3394-wrapped.
[out]	p_rfc3394_wrapped_target_key	Pointer to destination of RFC3394-wrapped target key.

◆ rfc3394_KeyUnwrap

fsp_err_t(* rsip_api_t::rfc3394_KeyUnwrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, uint8_t const *const p_rfc3394_wrapped_target_key, rsip_wrapped_key_t *const p_wrapped_target_key)

This function provides Key Unwrap algorithm compliant with RFC3394. Using p_wrapped_kek to unwrap p_rfc3394_wrapped_target_key, and output the result to p_wrapped_target_key.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_kek	Pointer to wrapped key-encryption-key used to RFC3394-unwrap the target key.
[in]	p_rfc3394_wrapped_target_key	Pointer to AES-wrapped target key to be RFC3394-unwrapped.
[in,out]	p_wrapped_target_key	Pointer to destination of RFC3394-unwrapped target key.

◆ publicKeyExport

fsp_err_t(* rsip_api_t::publicKeyExport) (rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t *const p_raw_public_key)

Exports public key parameters from a wrapped key.

Parameters

[in]	p_wrapped_public_key	Key index for Public Key. The length depends on the key type. Refer "Key Size Table".
[out]	p_raw_public_key	Pointer to destination of raw public key. The length depends on the key length.

◆ aesCipherInit

fsp_err_t(* rsip_api_t::aesCipherInit) (rsip_ctrl_t *const p_ctrl, rsip_aes_cipher_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_initial_vector)

Set parameters of AES cipher.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	mode	Block cipher modes of operation for AES.
[in]	p_wrapped_key	Pointer to wrapped key of AES or XTS-AES key.
[in]	p_initial_vector	Pointer to initialization vector (IV) or nonce. The length is 16 bytes. • [ECB] Not required • [CBC][XTS] IV • [CTR] Nonce

◆ aesCipherUpdate

fsp_err_t(* rsip_api_t::aesCipherUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint8_t *const p_output, uint32_t const length)

Encrypt plaintext.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_input	Pointer to input text. The length is given as the argument.
[out]	p_output	Pointer to destination of output text. The length is given as the argument.
[in]	length	Byte length of input and output. • [ECB][CBC][CTR] Must be 0 or a multiple of 16. • [XTS] Must be 0 or greater than or equal to 16. After an integer not divisible by 16 is input, update can no longer be executed.

◆ aesCipherFinish

fsp_err_t(* rsip_api_t::aesCipherFinish) (rsip_ctrl_t *const p_ctrl)

Finalize AES operation.

Parameters

[in,out]	p_ctrl	Pointer to control block.

◆ aesAeadInit

fsp_err_t(* rsip_api_t::aesAeadInit) (rsip_ctrl_t *const p_ctrl, rsip_aes_aead_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)

Prepares an AES-AEAD function.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	mode	AEAD mode of operation.
[in]	p_wrapped_key	Pointer to wrapped key of AES key.
[in]	p_nonce	Pointer to nonce. The length is nonce_length.
[in]	nonce_length	Byte length of nonce. Input 1 or more.

◆ aesAeadLengthsSet

fsp_err_t(* rsip_api_t::aesAeadLengthsSet) (rsip_ctrl_t *const p_ctrl, uint32_t const total_aad_length, uint32_t const total_text_length, uint32_t const tag_length)

Set text and tag lengths for specific mode.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	total_aad_length	Total AAD length.
[in]	total_text_length	Total input and output text length.
[in]	tag_length	Input or output tag length.

◆ aesAeadAadUpdate

fsp_err_t(* rsip_api_t::aesAeadAadUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)

Inputs additional authentication data.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_aad	Additional authentication data. The length depends on aad_length.
[in]	aad_length	Byte length of additional authentication data (0 or more bytes). After starting input of plaintext, this value must always be 0.

◆ aesAeadUpdate

fsp_err_t(* rsip_api_t::aesAeadUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)

Inputs test and executes encryption and decryption.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_input	Pointer to input text. The length is input_length.
[in]	input_length	Byte length of input text (0 or more bytes).
[out]	p_output	Pointer to destination of output text. The length is p_output_length.
[out]	p_output_length	Pointer to destination of output text length.

◆ aesAeadFinish

fsp_err_t(* rsip_api_t::aesAeadFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)

Finalizes an AES-GCM encryption.

If there is 16-byte fractional data indicated by the total data length of the value of p_plain that was input by R_RSIP_AES_GCM_EncryptUpdate(), this API will output the result of encrypting that fractional data to p_cipher. Here, the portion that does not reach 16 bytes will be padded with zeros.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_output	Pointer to destination of output text. The fractional block is output.
[out]	p_output_length	Pointer to destination of output text length.
[out]	p_tag	Pointer to destination of tag for authentication. GCM : The length is 16 bytes. *If a different tag length is required, truncate the 16-byte tag to the required tag length (NIST SP800-38D 7.1). CCM : The length is the value set by the API R_RSIP_AES_AEAD_LengthsSet().

◆ aesAeadVerify

fsp_err_t(* rsip_api_t::aesAeadVerify) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)

Finalizes an AES-GCM decryption.

If there is 16-byte fractional data indicated by the total data length of the value of p_cipher that was input by R_RSIP_AES_GCM_DecryptUpdate(), this API will output the result of decrypting that fractional data to p_cipher. Here, the portion that does not reach 16 bytes will be padded with zeros.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_output	Pointer to destination of decrypted data.
[out]	p_output_length	Pointer to destination of decrypted data length.
[in]	p_tag	Pointer to destination of tag for authentication. The length depends on tag_length.
[in]	tag_length	Byte length of tag. Must be 1 to 16.

◆ aesMacInit

fsp_err_t(* rsip_api_t::aesMacInit) (rsip_ctrl_t *const p_ctrl, rsip_aes_mac_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key)

Prepares an AES-MAC generation and verification.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	mode	MAC mode of operation
[in]	p_wrapped_key	Pointer to wrapped key of AES key.

◆ aesMacUpdate

fsp_err_t(* rsip_api_t::aesMacUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Input message. Inside this function, the data that is input by the user is buffered until the input value of p_message exceeds 16 bytes. If the input value, p_message, is not a multiple of 16 bytes, it will be padded within the function.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).

◆ aesMacSignFinish

fsp_err_t(* rsip_api_t::aesMacSignFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)

Finalizes an AES-CMAC generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_mac	Pointer to destination of MAC. The length is 16 bytes.

◆ aesMacVerifyFinish

fsp_err_t(* rsip_api_t::aesMacVerifyFinish) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)

Finalizes an AES-CMAC verification.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_mac	Pointer to MAC. The length depends on mac_length.
[in]	mac_length	Byte length of MAC. Must be 2 to 16.

◆ chacha20Init

fsp_err_t(* rsip_api_t::chacha20Init) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const counter)

Prepares a ChaCha20 function.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_key	Pointer to wrapped key of ChaCha20 key.
[in]	p_nonce	Pointer to nonce. The length is 12 bytes.
[in]	counter	32-bit initial counter. This is usually zero.

◆ chacha20Update

fsp_err_t(* rsip_api_t::chacha20Update) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)

Encrypts plaintext or decrypts ciphertext.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_input	Pointer to input text. The length is input_length.
[in]	input_length	Byte length of input text (0 or more bytes).
[out]	p_output	Pointer to destination of output text. The length is p_output_length.
[out]	p_output_length	Pointer to destination of output text length.

◆ chacha20Finish

fsp_err_t(* rsip_api_t::chacha20Finish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length)

Finalize ChaCha20 operation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_output	Pointer to destination of output text. The fractional block is output.
[out]	p_output_length	Pointer to destination of output text length.

◆ chacha20Poly1305Init

fsp_err_t(* rsip_api_t::chacha20Poly1305Init) (rsip_ctrl_t *const p_ctrl, rsip_chacha_poly_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)

Prepares a ChaCha20-Poly1305 function.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	mode	AEAD mode of operation.
[in]	p_wrapped_key	Pointer to wrapped key of AES key.
[in]	p_nonce	Pointer to nonce. The length is nonce_length.
[in]	nonce_length	Byte length of nonce. Input 1 or more.

◆ chacha20Poly1305AadUpdate

fsp_err_t(* rsip_api_t::chacha20Poly1305AadUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)

Inputs Additional Authentication Data (AAD).

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_aad	Additional authentication data. The length depends on aad_length.
[in]	aad_length	Byte length of additional authentication data (0 or more bytes). After starting input of plaintext, this value must always be 0.

◆ chacha20Poly1305Update

fsp_err_t(* rsip_api_t::chacha20Poly1305Update) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)

Encrypts plaintext or decrypts ciphertext.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_input	Pointer to input text. The length is input_length.
[in]	input_length	Byte length of input text (0 or more bytes).
[out]	p_output	Pointer to destination of output text. The length is p_output_length.
[out]	p_output_length	Pointer to destination of output text length.

◆ chacha20Poly1305Finish

fsp_err_t(* rsip_api_t::chacha20Poly1305Finish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)

Finalizes a ChaCha20-Poly1305 encryption.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_output	Pointer to destination of output text. The fractional block is output.
[out]	p_output_length	Pointer to destination of output text length.
[out]	p_tag	Pointer to destination of tag for authentication. The length is 16 bytes.

◆ chacha20Poly1305Verify

fsp_err_t(* rsip_api_t::chacha20Poly1305Verify) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)

Finalizes a ChaCha20-Poly1305 decryption.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_output	Pointer to destination of decrypted data.
[out]	p_output_length	Pointer to destination of decrypted data length.
[in]	p_tag	Pointer to destination of tag for authentication. The length depends on tag_length.
[in]	tag_length	Byte length of tag. Must be 16.

◆ ecdsaSign

fsp_err_t(* rsip_api_t::ecdsaSign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_hash, uint8_t *const p_signature)

Signs a hashed message. The message hash should be generated in advance.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of ECC private key.
[in]	p_hash	Pointer to hash value. The length is as same as the key length.
[out]	p_signature	Pointer to destination of signature (r, s). The length is twice as long as the key length.

◆ ecdsaVerify

fsp_err_t(* rsip_api_t::ecdsaVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies a hashed message. The message hash should be generated in advance.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of ECC public key.
[in]	p_hash	Pointer to hash value. The length is as same as the key length.
[in]	p_signature	Pointer to signature (r, s). The length is twice as long as the key length.

◆ eddsaSign

fsp_err_t(* rsip_api_t::eddsaSign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t *const p_signature)

Sign the message using EdDSA.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of ECC private key.
[in]	p_wrapped_public_key	Pointer to wrapped key of ECC public key.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message.
[out]	p_signature	Pointer to destination of signature (r, s). The length is twice as long as the key length.

◆ eddsaVerify

fsp_err_t(* rsip_api_t::eddsaVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t const *const p_signature)

Verifies the message using EdDSA.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of ECC public key.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message.
[in]	p_signature	Pointer to signature (r, s). The length is twice as long as the key length.

◆ ecdhKeyAgree

fsp_err_t(* rsip_api_t::ecdhKeyAgree) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret)

Computes ECDH secret with wrapped private key and wrapped public key.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Wrapped secp256r1 private key.
[in]	p_wrapped_public_key	Wrapped secp256r1 public key.
[out]	p_wrapped_secret	Pointer to destination of wrapped secret

◆ ecdhPlainKeyAgree

fsp_err_t(* rsip_api_t::ecdhPlainKeyAgree) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_plain_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret)

Computes ECDH secret with wrapped private key and plain public key.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Wrapped secp256r1 private key.
[in]	p_plain_public_key	Plain secp256r1 public key.
[out]	p_wrapped_secret	Pointer to destination of wrapped secret.

◆ rsaEncrypt

fsp_err_t(* rsip_api_t::rsaEncrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint8_t *const p_cipher)

Encrypts plaintext with raw RSA.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	p_plain	Pointer to plaintext. The length is as same as the key length.
[out]	p_cipher	Pointer to destination of ciphertext. The length is as same as the key length.

◆ rsaDecrypt

fsp_err_t(* rsip_api_t::rsaDecrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain)

Decrypts ciphertext with raw RSA.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of RSA private key.
[in]	p_cipher	Pointer to ciphertext. The length is as same as the key length.
[out]	p_plain	Pointer to destination of plaintext. The length is as same as the key length.

◆ rsaesPkcs1V15Encrypt

fsp_err_t(* rsip_api_t::rsaesPkcs1V15Encrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)

Encrypts plaintext with RSAES-PKCS1-v1_5.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	p_plain	Pointer to plaintext.
[in]	plain_length	Length of plaintext.
[out]	p_cipher	Pointer to destination of ciphertext. The length is as same as the key length.

◆ rsaesPkcs1V15Decrypt

fsp_err_t(* rsip_api_t::rsaesPkcs1V15Decrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)

Decrypts with RSAES-PKCS1-v1_5.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of RSA private key.
[in]	p_cipher	Pointer to ciphertext. The length is as same as the key length.
[out]	p_plain	Pointer to destination of plaintext.
[out]	p_plain_length	Pointer to destination of actual plaintext length.
[in]	plain_buffer_length	Length of plaintext destination. It must be equal to or greater than *p_plain_length.

◆ rsaesOaepEncrypt

fsp_err_t(* rsip_api_t::rsaesOaepEncrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)

Encrypts plaintext with RSAES-OAEP.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	hash_function	Hash function for label.
[in]	mask_generation_function	Mask generation function in EME-OAEP encoding.
[in]	p_label	Pointer to label. If label_length != 0, p_label must not be NULL.
[in]	label_length	Length of label. Please set 0 or more.
[in]	p_plain	Pointer to plaintext.
[in]	plain_length	Length of plaintext.
[out]	p_cipher	Pointer to destination of ciphertext. The length is as same as the key length.

◆ rsaesOaepDecrypt

fsp_err_t(* rsip_api_t::rsaesOaepDecrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)

Decrypts ciphertext with RSAES-OAEP.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of RSA private key.
[in]	hash_function	Hash function for label.
[in]	mask_generation_function	Mask generation function in EME-OAEP encoding.
[in]	p_label	Pointer to label. If label_length != 0, p_label must not be NULL.
[in]	label_length	Length of label. Please set 0 or more.
[in]	p_cipher	Pointer to ciphertext. The length is as same as the key length.
[out]	p_plain	Pointer to destination of plaintext.
[out]	p_plain_length	Pointer to destination of actual plaintext length.
[in]	plain_buffer_length	Length of plaintext destination. It must be equal to or greater than *p_plain_length.

◆ rsassaPkcs1V15Sign

fsp_err_t(* rsip_api_t::rsassaPkcs1V15Sign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t *const p_signature)

Signs message with RSASSA-PKCS1-v1_5.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of RSA private key.
[in]	hash_function	Hash function in EMSA-PKCS1-v1_5.
[in]	p_hash	Pointer to input hash.
[out]	p_signature	Pointer to destination of signature.

◆ rsassaPkcs1V15Verify

fsp_err_t(* rsip_api_t::rsassaPkcs1V15Verify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies signature with RSASSA-PKCS1-v1_5.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	hash_function	Hash function in EMSA-PKCS1-v1_5.
[in]	p_hash	Pointer to input hash.
[in]	p_signature	Pointer to input signature.

◆ rsassaPssSign

fsp_err_t(* rsip_api_t::rsassaPssSign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t *const p_signature)

Signs message with RSASSA-PSS.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_private_key	Pointer to wrapped key of RSA private key.
[in]	hash_function	Hash function in EMSA-PSS-ENCODE.
[in]	mask_generation_function	Mask generation function in EMSA-PSS-ENCODE.
[in]	salt_length	Salt length.
[in]	p_hash	Pointer to input hash.
[out]	p_signature	Pointer to destination of signature.

◆ rsassaPssVerify

fsp_err_t(* rsip_api_t::rsassaPssVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies signature with RSASSA-PSS.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of RSA public key.
[in]	hash_function	Hash function in EMSA-PSS-VERIFY.
[in]	mask_generation_function	Mask generation function in EMSA-PSS-VERIFY.
[in]	salt_length	Salt length.
[in]	p_hash	Pointer to input hash.
[in]	p_signature	Pointer to input signature.

◆ shaCompute

fsp_err_t(* rsip_api_t::shaCompute) (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_digest)

Generates SHA message digest.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	hash_type	Generating hash type.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).
[out]	p_digest	Pointer to destination of message digest. The length depends on hash type.

◆ shaInit

fsp_err_t(* rsip_api_t::shaInit) (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)

Prepares a SHA generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	hash_type	Generating hash type.

◆ shaUpdate

fsp_err_t(* rsip_api_t::shaUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).

◆ shaFinish

fsp_err_t(* rsip_api_t::shaFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_digest)

Finalizes a SHA generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_digest	Pointer to destination of message digest. The length depends on hash type.

◆ shaSuspend

fsp_err_t(* rsip_api_t::shaSuspend) (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t *const p_handle)

Suspend SHA generation. This API allows you to suspend processing, for example, if you are in the middle of computing digest value for successive chunks of the message and need to perform another process.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_handle	Pointer to destination of SHA control block.

◆ shaResume

fsp_err_t(* rsip_api_t::shaResume) (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t const *const p_handle)

Resume SHA generation. This API allows you to resume a process that has been suspended by R_RSIP_SHA_Suspend() API.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_handle	Pointer to SHA control block.

◆ hmacCompute

fsp_err_t(* rsip_api_t::hmacCompute) (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_mac)

Generates HMAC.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_key	Pointer to wrapped key of HMAC key.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).
[out]	p_mac	Pointer to destination of message digest. The length depends on MAC type.

◆ hmacVerify

fsp_err_t(* rsip_api_t::hmacVerify) (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t const *const p_mac, uint32_t const mac_length)

Verifies HMAC.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_key	Pointer to wrapped key of HMAC key.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).
[in]	p_mac	Pointer to MAC. The length depends on mac_length.
[in]	mac_length	Byte length of MAC.

◆ hmacInit

fsp_err_t(* rsip_api_t::hmacInit) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)

Prepares a HMAC generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_key	Pointer to wrapped key of HMAC key.

◆ hmacUpdate

fsp_err_t(* rsip_api_t::hmacUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).

◆ hmacSignFinish

fsp_err_t(* rsip_api_t::hmacSignFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)

Finalizes a HMAC generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_mac	Pointer to destination of message digest. The length depends on MAC type.

◆ hmacVerifyFinish

fsp_err_t(* rsip_api_t::hmacVerifyFinish) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)

Finalizes a HMAC verification.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_mac	Pointer to MAC. The length depends on mac_length.
[in]	mac_length	Byte length of MAC.

◆ hmacSuspend

fsp_err_t(* rsip_api_t::hmacSuspend) (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t *const p_handle)

Suspend HMAC generation. This API allows you to suspend processing, for example, if you are in the middle of computing HMAC for successive chunks of the message and need to perform another process.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_handle	Pointer to destination of HMAC control block.

◆ hmacResume

fsp_err_t(* rsip_api_t::hmacResume) (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t const *const p_handle)

Resume HMAC generation. This API allows you to resume a process that has been suspended by R_RSIP_HMAC_Suspend() API.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_handle	Pointer to HMAC control block.

◆ pkiEcdsaCertVerify

fsp_err_t(* rsip_api_t::pkiEcdsaCertVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies a public key certificate with ECDSA.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_public_key	Pointer to wrapped key of ECC public key.
[in]	p_hash	Pointer to hash value. The length is as same as the key length.
[in]	p_signature	Pointer to signature (r, s). The length is twice as long as the key length.

◆ pkiVerifiedCertInfoExport

fsp_err_t(* rsip_api_t::pkiVerifiedCertInfoExport) (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t *const p_verified_cert_info)

Exports verified certificate information stored in this driver.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_verified_cert_info	Pointer to certificate to be signed

◆ pkiVerifiedCertInfoImport

fsp_err_t(* rsip_api_t::pkiVerifiedCertInfoImport) (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t const *const p_verified_cert_info)

Imports verified certificate information.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_verified_cert_info	Pointer to certificate to be signed

◆ pkiCertKeyImport

fsp_err_t(* rsip_api_t::pkiCertKeyImport) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_cert, uint32_t const cert_length, uint8_t const *const p_key_param1, uint32_t const key_param1_length, uint8_t const *const p_key_param2, uint32_t const key_param2_length, rsip_hash_type_t const hash_function, rsip_wrapped_key_t *const p_wrapped_public_key)

Wraps the public key in the verified public key certificate.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_cert	Pointer to certificate.
[in]	cert_length	Certificate length.
[in]	p_key_param1	Pointer to start address of the public key parameter in certificate. • [ECC] Qx
[in]	key_param1_length	Length of key_param1 stored in the certificate.
[in]	p_key_param2	Pointer to start address of the public key parameter in certificate. • [ECC] Qy
[in]	key_param2_length	Length of key_param2 stored in the certificate.
[in]	hash_function	The hash function used when verifying certificate signature.
[in,out]	p_wrapped_public_key	Pointer to wrapped key of public key in the certificate.

◆ kdfShaInit

fsp_err_t(* rsip_api_t::kdfShaInit) (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)

Prepares a SHA generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	hash_type	Generating hash type.

◆ kdfShaEcdhSecretUpdate

fsp_err_t(* rsip_api_t::kdfShaEcdhSecretUpdate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret)

Inputs wrapped ECDH secret as a message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_secret	Pointer to wrapped secret.

◆ kdfShaUpdate

fsp_err_t(* rsip_api_t::kdfShaUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).

◆ kdfShaFinish

fsp_err_t(* rsip_api_t::kdfShaFinish) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)

Finalizes a SHA operation and generate DKM (Derived Keying Material).

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_wrapped_dkm	Pointer to destination of wrapped DKM.

◆ kdfShaSuspend

fsp_err_t(* rsip_api_t::kdfShaSuspend) (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t *const p_handle)

Suspend SHA generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_handle	Pointer to destination of KDF SHA control block.

◆ kdfShaResume

fsp_err_t(* rsip_api_t::kdfShaResume) (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t const *const p_handle)

Resume SHA generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_handle	Pointer to KDF SHA control block.

◆ kdfHmacDkmKeyImport

fsp_err_t(* rsip_api_t::kdfHmacDkmKeyImport) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const key_length, rsip_wrapped_key_t *const p_wrapped_key)

Converts wrapped Derived Keying Material (DKM) to HMAC key for KDF.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_dkm	Pointer to wrapped DKM.
[in]	key_length	Length of HMAC key to be extracted from DKM.
[in,out]	p_wrapped_key	Pointer to destination wrapped HMAC key for KDF.

◆ kdfHmacEcdhSecretKeyImport

fsp_err_t(* rsip_api_t::kdfHmacEcdhSecretKeyImport) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret, rsip_wrapped_key_t *const p_wrapped_key)

Converts wrapped ECDH secret to wrapped HMAC key for KDF.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_secret	Pointer to wrapped secret.
[in,out]	p_wrapped_key	Pointer to destination wrapped HMAC key for KDF.

◆ kdfHmacInit

fsp_err_t(* rsip_api_t::kdfHmacInit) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)

Prepares a HMAC generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_key	Pointer to wrapped key of HMAC key.

◆ kdfHmacDkmUpdate

fsp_err_t(* rsip_api_t::kdfHmacDkmUpdate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm)

Inputs wrapped Derived Keying Material (DKM) as a message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_dkm	Pointer to wrapped DKM.

◆ kdfHmacEcdhSecretUpdate

fsp_err_t(* rsip_api_t::kdfHmacEcdhSecretUpdate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret)

Inputs wrapped ECDH secret as a message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_secret	Pointer to wrapped secret.

◆ kdfHmacUpdate

fsp_err_t(* rsip_api_t::kdfHmacUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_message	Pointer to message. The length is message_length.
[in]	message_length	Byte length of message (0 or more bytes).

◆ kdfHmacSignFinish

fsp_err_t(* rsip_api_t::kdfHmacSignFinish) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)

Finalizes a HMAC generation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_wrapped_dkm	Pointer to destination of wrapped Derived Keying Material (DKM).

◆ kdfHmacSuspend

fsp_err_t(* rsip_api_t::kdfHmacSuspend) (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t *const p_handle)

Suspends HMAC operation.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[out]	p_handle	Pointer to destination of KDF HMAC control block.

◆ kdfHmacResume

fsp_err_t(* rsip_api_t::kdfHmacResume) (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t const *const p_handle)

Resumes HMAC operation suspended by R_RSIP_KDF_HMAC_Suspend().

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_handle	Pointer to KDF HMAC control block.

◆ kdfDkmConcatenate

fsp_err_t(* rsip_api_t::kdfDkmConcatenate) (rsip_wrapped_dkm_t *const p_wrapped_dkm1, rsip_wrapped_dkm_t const *const p_wrapped_dkm2, uint32_t const wrapped_dkm1_buffer_length)

Concatenates two wrapped derived keying materials (DKMs).

Parameters

[in,out]	p_wrapped_dkm1	Pointer to first DKM (DKM1).
[in]	p_wrapped_dkm2	Pointer to second DKM (DKM2).
[in]	wrapped_dkm1_buffer_length	Length of wrapped_dkm1 buffer. It must be equal to or greater than DKM1 || DKM2.

◆ kdfDerivedKeyImport

fsp_err_t(* rsip_api_t::kdfDerivedKeyImport) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const position, rsip_wrapped_key_t *const p_wrapped_key)

Outputs a wrapped key from Derived Keying Material (DKM).

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_dkm	Pointer to wrapped DKM.
[in]	position	Start position of output key value in the DKM.
[in,out]	p_wrapped_key	Pointer to destination of wrapped key.

◆ kdfDerivedIvWrap

fsp_err_t(* rsip_api_t::kdfDerivedIvWrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, rsip_initial_vector_type_t const initial_vector_type, uint32_t const position, uint8_t const *const p_tls_sequence_num, uint8_t *const p_wrapped_initial_vector)

Outputs a initial vector from Derived Keying Material (DKM).

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_dkm	Pointer to wrapped DKM.
[in]	initial_vector_type	Initial vector type.
[in]	position	Start position of output data value in the DKM.
[in]	p_tls_sequence_num	TLS sequence number. This argument is valid only for TLS-related data type.
[out]	p_wrapped_initial_vector	Pointer to destination of wrapped initial vector.

◆ otfInit

fsp_err_t(* rsip_api_t::otfInit) (rsip_ctrl_t *const p_ctrl, rsip_otf_channel_t const channel, rsip_wrapped_key_t *const p_wrapped_key, uint8_t const *const p_seed)

Initialize on-the-fly decryption on RSIP.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	channel	Channel number.
[in]	p_wrapped_key	Pointer to wrapped AES key.
[in]	p_seed	Pointer to seed.

◆ xdlmsInitiateRequestDecrypt

fsp_err_t(* rsip_api_t::xdlmsInitiateRequestDecrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_apdu, uint8_t const *const p_nonce, uint8_t const *const p_aad, uint32_t const aad_length, rsip_wrapped_key_t *const p_dedicated_key, rsip_dlms_apdu_data_t *const p_apdu_data, rsip_dlms_ret_t *const p_dlms_ret)

Decrypts and Verifies ciphered InitiateRequest message, and wraps Dedicated Key (AES-128 or AES-256) embedded in the message.

Parameters

[in,out]	p_ctrl	Pointer to control block.
[in]	p_wrapped_key	Pointer to wrapped key of ciphered InitiateRequest APDU decryption key.
[in]	p_apdu	Pointer to ciphered xDLMS InitiateRequest APDU (TAG || LEN || SH || C || T).
[in]	p_nonce	Pointer to nonce (Sys-T || IC). The length must be 12 bytes.
[in]	p_aad	Pointer to additional authentication data (SC || AK). The length is aad_length.
[in]	aad_length	Byte length of additional authentication data.
[in]	p_dedicated_key	Pointer to Dedicated Key wrapped with HUK.
[in]	p_apdu_data	Pointer to decrypted xDLMS InitiateRequest APDU structure.
[out]	p_dlms_ret	DLMS/COSEM specific return value.

◆ rsip_instance_t

struct rsip_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rsip_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rsip_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rsip_api_t const *	p_api	Pointer to the API structure for

		this instance.

Typedef Documentation

◆ rsip_sha_handle_abstractor_t

typedef void rsip_sha_handle_abstractor_t
Allocate an instance specific handle to pass into the API calls.

◆ rsip_hmac_handle_abstractor_t

typedef void rsip_hmac_handle_abstractor_t
Allocate an instance specific handle to pass into the API calls.

◆ rsip_kdf_sha_handle_abstractor_t

typedef void rsip_kdf_sha_handle_abstractor_t
Allocate an instance specific handle to pass into the API calls.

◆ rsip_kdf_hmac_handle_abstractor_t

typedef void rsip_kdf_hmac_handle_abstractor_t
Allocate an instance specific handle to pass into the API calls.

◆ rsip_wrapped_secret_abstractor_t

typedef void rsip_wrapped_secret_abstractor_t
Allocate an instance specific handle to pass into the API calls.

◆ rsip_verified_cert_info_abstractor_t

typedef void rsip_verified_cert_info_abstractor_t
Allocate an instance specific handle to pass into the API calls.

◆ rsip_ctrl_t

typedef void rsip_ctrl_t
RSIP Control block. Allocate an instance specific control block to pass into the API calls.

Enumeration Type Documentation

◆ rsip_key_type_t

enum rsip_key_type_t	
Key types	
Enumerator	
RSIP_KEY_TYPE_INVALID	Invalid key.
RSIP_KEY_TYPE_AES_128	AES-128.
RSIP_KEY_TYPE_AES_192	AES-192.
RSIP_KEY_TYPE_AES_256	AES-256.
RSIP_KEY_TYPE_XTS_AES_128	XTS-AES-128.
RSIP_KEY_TYPE_XTS_AES_256	XTS-AES-256.
RSIP_KEY_TYPE_CHACHA20	ChaCha20.
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC	secp256r1 public key (also known as NIST P-256, prime256v1)
RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC	secp384r1 public key (also known as NIST P-384)
RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC	secp521r1 public key (also known as NIST P-521)
RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC	secp256k1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC	brainpoolP256r1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC	brainpoolP384r1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC	brainpoolP512r1 public key
RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC	edwards25519 public key
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE	secp256r1 private key (also known as NIST P-256, prime256v1)
RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE	secp384r1 private key (also known as NIST P-384)
RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE	secp521r1 private key (also known as NIST

	P-521)
RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE	secp256k1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE	brainpoolP256r1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE	brainpoolP384r1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE	brainpoolP512r1 private key
RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE	edwards25519 private key
RSIP_KEY_TYPE_RSA_1024_PUBLIC	RSA-1024 public key.
RSIP_KEY_TYPE_RSA_2048_PUBLIC	RSA-2048 public key.
RSIP_KEY_TYPE_RSA_3072_PUBLIC	RSA-3072 public key.
RSIP_KEY_TYPE_RSA_4096_PUBLIC	RSA-4096 public key.
RSIP_KEY_TYPE_RSA_1024_PRIVATE	RSA-1024 private key.
RSIP_KEY_TYPE_RSA_2048_PRIVATE	RSA-2048 private key.
RSIP_KEY_TYPE_RSA_3072_PRIVATE	RSA-3072 private key.
RSIP_KEY_TYPE_RSA_4096_PRIVATE	RSA-4096 private key.
RSIP_KEY_TYPE_HMAC_SHA1	HMAC-SHA1.
RSIP_KEY_TYPE_HMAC_SHA224	HMAC-SHA224.
RSIP_KEY_TYPE_HMAC_SHA256	HMAC-SHA256.
RSIP_KEY_TYPE_HMAC_SHA384	HMAC-SHA384.
RSIP_KEY_TYPE_HMAC_SHA512	HMAC-SHA512.
RSIP_KEY_TYPE_HMAC_SHA512_224	HMAC-SHA512/224.
RSIP_KEY_TYPE_HMAC_SHA512_256	HMAC-SHA512/256.
RSIP_KEY_TYPE_HMAC_SHA3_224	HMAC-SHA3-224.
RSIP_KEY_TYPE_HMAC_SHA3_256	HMAC-SHA3-256.
RSIP_KEY_TYPE_HMAC_SHA3_384	HMAC-SHA3-384.

RSIP_KEY_TYPE_HMAC_SHA3_512	HMAC-SHA3-512.
RSIP_KEY_TYPE_KDF_HMAC_SHA256	KDF HMAC-SHA256.
RSIP_KEY_TYPE_KDF_HMAC_SHA384	KDF HMAC-SHA384.
RSIP_KEY_TYPE_KDF_HMAC_SHA512	KDF HMAC-SHA512.
RSIP_KEY_TYPE_KUK	Key Update Key (KUK)

◆ rsip_aes_cipher_mode_t

enum rsip_aes_cipher_mode_t	
Block cipher modes of operation for AES	
Enumerator	
RSIP_AES_CIPHER_MODE_ECB_ENC	Electronic Codebook (ECB) mode encryption.
RSIP_AES_CIPHER_MODE_ECB_DEC	Electronic Codebook (ECB) mode decryption.
RSIP_AES_CIPHER_MODE_CBC_ENC	Cipher Block Chaining (CBC) mode encryption.
RSIP_AES_CIPHER_MODE_CBC_DEC	Cipher Block Chaining (CBC) mode decryption.
RSIP_AES_CIPHER_MODE_CTR	Counter (CTR) mode encryption or decryption.
RSIP_AES_CIPHER_MODE_XTS_ENC	XEX-based tweaked-codebook mode with ciphertext stealing (XTS) encryption.
RSIP_AES_CIPHER_MODE_XTS_DEC	XEX-based tweaked-codebook mode with ciphertext stealing (XTS) decryption.
RSIP_AES_CIPHER_MODE_CBC_ENC_WRAPPED_IV	Cipher Block Chaining (CBC) mode encryption with wrapped IV.
RSIP_AES_CIPHER_MODE_CBC_DEC_WRAPPED_IV	Cipher Block Chaining (CBC) mode decryption with wrapped IV.

◆ rsip_aes_aead_mode_t

enum rsip_aes_aead_mode_t	
AEAD modes of operation for AES	
Enumerator	
RSIP_AES_AEAD_MODE_GCM_ENC	Galois/Counter Mode (GCM) encryption.
RSIP_AES_AEAD_MODE_GCM_DEC	Galois/Counter Mode (GCM) decryption.
RSIP_AES_AEAD_MODE_CCM_ENC	Counter with CBC-MAC (CCM) encryption.
RSIP_AES_AEAD_MODE_CCM_DEC	Counter with CBC-MAC (CCM) decryption.
RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV	Galois/Counter Mode (GCM) encryption with wrapped IV.
RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV	Galois/Counter Mode (GCM) decryption with wrapped IV.

◆ rsip_chacha_poly_mode_t

enum rsip_chacha_poly_mode_t	
ChaCha20-Poly1305 mode	
Enumerator	
RSIP_CHACHA20_POLY1305_MODE_ENC	ChaCha20-Poly1305 encryption.
RSIP_CHACHA20_POLY1305_MODE_DEC	ChaCha20-Poly1305 decryption.

◆ rsip_aes_mac_mode_t

enum rsip_aes_mac_mode_t	
MAC modes of operation for AES	
Enumerator	
RSIP_AES_MAC_MODE_CMAC	Cipher-based MAC (CMAC)

◆ rsip_hash_type_t

enum rsip_hash_type_t	
Hash type	
Enumerator	
RSIP_HASH_TYPE_SHA1	SHA-1.
RSIP_HASH_TYPE_SHA224	SHA-224.
RSIP_HASH_TYPE_SHA256	SHA-256.
RSIP_HASH_TYPE_SHA384	SHA-384.
RSIP_HASH_TYPE_SHA512	SHA-512.
RSIP_HASH_TYPE_SHA512_224	SHA-512/224.
RSIP_HASH_TYPE_SHA512_256	SHA-512/256.
RSIP_HASH_TYPE_SHA3_224	SHA3-224.
RSIP_HASH_TYPE_SHA3_256	SHA3-256.
RSIP_HASH_TYPE_SHA3_384	SHA3-384.
RSIP_HASH_TYPE_SHA3_512	SHA3-512.

◆ rsip_mgf_type_t

enum rsip_mgf_type_t	
MGF type	
Enumerator	
RSIP_MGF_TYPE_MGF1_SHA1	MGF1 with SHA-1.
RSIP_MGF_TYPE_MGF1_SHA224	MGF1 with SHA-224.
RSIP_MGF_TYPE_MGF1_SHA256	MGF1 with SHA-256.
RSIP_MGF_TYPE_MGF1_SHA384	MGF1 with SHA-384.
RSIP_MGF_TYPE_MGF1_SHA512	MGF1 with SHA-512.
RSIP_MGF_TYPE_MGF1_SHA512_224	MGF1 with SHA-512/224.
RSIP_MGF_TYPE_MGF1_SHA512_256	MGF1 with SHA-512/256.

◆ rsip_rsa_salt_length_t

enum rsip_rsa_salt_length_t	
RSA salt length	
Enumerator	
RSIP_RSA_SALT_LENGTH_AUTO	When signing, the salt length is set to RSIP_RSA_SALT_LENGTH_MAX or RSIP_RSA_SALT_LENGTH_HASH, whichever is shorter. When verifying, the salt length is detected automatically.
RSIP_RSA_SALT_LENGTH_HASH	The salt length is set to the hash length.
RSIP_RSA_SALT_LENGTH_MAX	The salt length is set to emLen - hLen - 2, where emLen is the same as the key length and hLen is the hash length.

◆ rsip_initial_vector_type_t

enum rsip_initial_vector_type_t	
IV data types	
Enumerator	
RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE	Wrapped AES initial vector (unprocessed)
RSIP_INITIAL_VECTOR_TYPE_AES_TLS12_AEAD	Wrapped AES nonce (TLS1.2 AES AEAD)
RSIP_INITIAL_VECTOR_TYPE_AES_TLS13_AEAD	Wrapped AES nonce (TLS1.3 AES AEAD)

◆ rsip_otf_channel_t

enum rsip_otf_channel_t	
Channel number of on-the-fly decryption/encryption module.	
Enumerator	
RSIP_OTF_CHANNEL_0	Channel 0.
RSIP_OTF_CHANNEL_1	Channel 1.
RSIP_OTF_CHANNEL_2	Channel 2.

◆ rsip_dlms_ret_t

enum rsip_dlms_ret_t	
Return values for DLMS/COSEM.	
Enumerator	
RSIP_DLMS_RET_SUCCESS	Normal termination.
RSIP_DLMS_RET_INVALID_OPERATIONAL_MODE	xDLMS Operational Mode is invalid (not Authenticated encryption mode).
RSIP_DLMS_RET_INVALID_SECURITY_SUITE	Security Suite id is unexpected value.
RSIP_DLMS_RET_INVALID_APDU_CHOICE	APDU CHOICE is unexpected value.
RSIP_DLMS_RET_INVALID_KEY_LENGTH	Key Length is unexpected value.
RSIP_DLMS_RET_INVALID_SIZE_AAD	Input AAD length is invalid.
RSIP_DLMS_RET_INVALID_SIZE_LEN	LEN field within the APDU is invalid.
RSIP_DLMS_RET_SECURITY_SUITE_MISMATCH	Key type is mismatch with Security Suite id.
RSIP_DLMS_RET_NO_DEDICATED_KEY	Dedicated key not present.

5.3.14 Sensor
Interfaces

Sensor Interfaces.

5.3.15 Storage
Interfaces

Detailed Description

Storage Interfaces.

Modules

	Flash Interface Interface for the Flash Memory.
	SD/MMC Interface Interface for accessing SD, eMMC, and SDIO devices.
	SPI Flash Interface Interface for accessing external SPI flash devices.
	Block Media Interface Interface for block media memory access.
	FreeRTOS+FAT Port Interface Interface for FreeRTOS+FAT port.
	LittleFS Interface Interface for LittleFS access.

5.3.15.1 Flash Interface

Interfaces » Storage

Detailed Description

Interface for the Flash Memory.

Summary

The Flash interface provides the ability to read, write, erase, and blank check the code flash and data flash regions.

Data Structures

struct	flash_block_info_t
struct	flash_regions_t
struct	flash_info_t

struct	flash_callback_args_t
struct	flash_cfg_t
struct	flash_api_t
struct	flash_instance_t

Typedefs

typedef void	flash_ctrl_t

Enumerations

enum	flash_result_t
enum	flash_startup_area_swap_t
enum	flash_event_t
enum	flash_id_code_mode_t
enum	flash_status_t
enum	flash_arc_t

Data Structure Documentation

◆ flash_block_info_t

struct flash_block_info_t		
Flash block details stored in factory flash.		
Data Fields		
uint32_t	block_section_st_addr	Starting address for this block section (blocks of this size)
uint32_t	block_section_end_addr	Ending address for this block section (blocks of this size)
uint32_t	block_size	Flash erase block size.
uint32_t	block_size_write	Flash write block size.

◆ flash_regions_t

struct flash_regions_t		
Flash block details		
Data Fields		
uint32_t	num_regions	Length of block info array.

flash_block_info_t const *	p_block_array	Block info array base address.

◆ flash_info_t

struct flash_info_t		
Information about the flash blocks		
Data Fields		
flash_regions_t	code_flash	Information about the code flash regions.
flash_regions_t	data_flash	Information about the code flash regions.

◆ flash_callback_args_t

struct flash_callback_args_t		
Callback function parameter data		
Data Fields		
flash_event_t	event	Event can be used to identify what caused the callback (flash ready or error).
void *	p_context	Placeholder for user data. Set in flash_api_t::open function in::flash_cfg_t.

◆ flash_cfg_t

struct flash_cfg_t	
FLASH Configuration	
Data Fields	
bool	data_flash_bgo
	True if BGO (Background Operation) is enabled for Data Flash.
void(*	p_callback)(flash_callback_args_t *p_args)
	Callback provided when a Flash interrupt ISR occurs.
void const *	p_extend
	FLASH hardware dependent configuration.
void *	p_context

	Placeholder for user data. Passed to user callback in flash_callback_args_t.
uint8_t	ipl
	Flash ready interrupt priority.
IRQn_Type	irq
	Flash ready interrupt number.
uint8_t	err_ipl
	Flash error interrupt priority (unused in r_flash_lp)
IRQn_Type	err_irq
	Flash error interrupt number (unused in r_flash_lp)

◆ flash_api_t

struct flash_api_t	
Shared Interface definition for FLASH	
Data Fields	
fsp_err_t(*	open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t(*	write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
fsp_err_t(*	erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t(*	blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
fsp_err_t(*	infoGet)(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

fsp_err_t(*	close)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)
fsp_err_t(*	accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)
fsp_err_t(*	accessWindowClear)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	idCodeSet)(flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)
fsp_err_t(*	reset)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	startupAreaSelect)(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
fsp_err_t(*	bankSwap)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(flash_ctrl_t *const p_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory)
fsp_err_t(*	antiRollbackCounterIncrement)(flash_ctrl_t *const p_ctrl, flash_arc_t counter)
fsp_err_t(*	antiRollbackCounterRefresh)(flash_ctrl_t *const p_ctrl, flash_arc_t counter)
fsp_err_t(*	antiRollbackCounterRead)(flash_ctrl_t *const p_ctrl, flash_arc_t counter, uint32_t *const p_count)
fsp_err_t(*	userLockableAreaWrite)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Field Documentation

◆ open

fsp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)

Open FLASH device.

Parameters

[out]	p_ctrl	Pointer to FLASH device control. Must be declared by user. Value set here.
[in]	flash_cfg_t	Pointer to FLASH configuration structure. All elements of this structure must be set by the user.

◆ write

fsp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Write FLASH device.

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	src_address	Address of the buffer containing the data to write to Flash.
[in]	flash_address	Code Flash or Data Flash address to write. The address must be on a programming line boundary.
[in]	num_bytes	The number of bytes to write. This number must be a multiple of the programming size. For Code Flash this is FLASH_MIN_PGM_SIZE_CF. For Data Flash this is FLASH_MIN_PGM_SIZE_DF.

Warning

Specifying a number that is not a multiple of the programming size will result in SF_FLASH_ERR_BYTES being returned and no data written.

◆ erase

fsp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

Erase FLASH device.

Parameters

[in]	p_ctrl	Control for the FLASH device.
[in]	address	The block containing this address is the first block erased.
[in]	num_blocks	Specifies the number of blocks to be erased, the starting block determined by the block_erase_address.

◆ blankCheck

fsp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)

Blank check FLASH device.

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	address	The starting address of the Flash area to blank check.
[in]	num_bytes	Specifies the number of bytes that need to be checked. See the specific handler for details.
[out]	p_blank_check_result	Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode. In this case the blank check operation completes here and the result is returned. In Data Flash BGO mode the blank check operation is only started here and the result obtained later when the supplied callback routine is called. In this case FLASH_RESULT_BGO_ACTIVE will be returned in p_blank_check_result.

◆ infoGet

fsp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

Close FLASH device.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_info	Pointer to FLASH info structure.

◆ close

fsp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)

Close FLASH device.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ statusGet

fsp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)

Get Status for FLASH device.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_ctrl	Pointer to the current flash status.

◆ accessWindowSet

fsp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)

Set Access Window for FLASH device.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window. This address will not be within the access window.

◆ accessWindowClear

fsp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)

Clear any existing Code Flash access window for FLASH device.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window.

◆ idCodeSet

fsp_err_t(* flash_api_t::idCodeSet) (flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)

Set ID Code for FLASH device. Setting the ID code can restrict access to the device. The ID code will be required to connect to the device. Bits 126 and 127 are set based on the mode.

For example, uint8_t id_bytes[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00}; with mode FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT will result in an ID code of 00112233445566778899aabbccddeec0

With mode FLASH_ID_CODE_MODE_LOCKED, it will result in an ID code of 00112233445566778899aabbccddee80

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	p_id_bytes	Ponter to the ID Code to be written.
[in]	mode	Mode used for checking the ID code.

◆ reset

fsp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)

Reset function for FLASH device.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ updateFlashClockFreq

fsp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)

Update Flash clock frequency (FCLK) and recalculate timeout values

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ startupAreaSelect

fsp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)

Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	swap_type	FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG.
[in]	is_temporary	True or false. See table below.

swap_type	is_temporary	Operation
FLASH_STARTUP_AREA_BLOCK0	false	On next reset Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK1	true	Startup area is immediately, but temporarily switched to Block 1.
FLASH_STARTUP_AREA_BTFLG	true	Startup area is immediately, but temporarily switched to the Block determined by the Configuration BTFLG.

◆ bankSwap

fsp_err_t(* flash_api_t::bankSwap) (flash_ctrl_t *const p_ctrl)

Swap the bank used as the startup area. On Flash HP, need to change into dual bank mode to use this feature.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ callbackSet

fsp_err_t(* flash_api_t::callbackSet) (flash_ctrl_t *const p_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ antiRollbackCounterIncrement

fsp_err_t(* flash_api_t::antiRollbackCounterIncrement) (flash_ctrl_t *const p_ctrl, flash_arc_t counter)

Increment the selected anti-rollback counter.

Warning

Once incremented, the counter can never be decremented or reset, even with a debug probe.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	counter	The anti-rollback counter to increment

◆ antiRollbackCounterRefresh

fsp_err_t(* flash_api_t::antiRollbackCounterRefresh) (flash_ctrl_t *const p_ctrl, flash_arc_t counter)

Refresh the selected anti-rollback counter flash area.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	counter	The anti-rollback counter to refresh

◆ antiRollbackCounterRead

fsp_err_t(* flash_api_t::antiRollbackCounterRead) (flash_ctrl_t *const p_ctrl, flash_arc_t counter, uint32_t *const p_count)

Read current count value of the selected anti-rollback counter.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	counter	The anti-rollback counter to read
[out]	p_count	The current count value

◆ userLockableAreaWrite

fsp_err_t(* flash_api_t::userLockableAreaWrite) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Write to the user lockable area for the flash device

Warning

Data will not be written if any segment of the specified area is locked.

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	src_address	Address of the buffer containing the data to write to flash.
[in]	flash_address	User lockable area flash address to write. The address must be on a programming line boundary.
[in]	num_bytes	The number of bytes to write. This number must be a multiple of the programming size.

◆ flash_instance_t

struct flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
flash_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
flash_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
flash_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ flash_ctrl_t

typedef void flash_ctrl_t

Flash control block. Allocate an instance specific control block to pass into the flash API calls.

Enumeration Type Documentation

◆ flash_result_t

enum flash_result_t	
Result type for certain operations	
Enumerator	
FLASH_RESULT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_NOT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_BGO_ACTIVE	Flash is configured for BGO mode. Result is returned in callback.

◆ flash_startup_area_swap_t

enum flash_startup_area_swap_t	
Parameter for specifying the startup area swap being requested by startupAreaSelect()	
Enumerator	
FLASH_STARTUP_AREA_BTFLG	Startup area will be set based on the value of the BTFLG.
FLASH_STARTUP_AREA_BLOCK0	Startup area will be set to Block 0.
FLASH_STARTUP_AREA_BLOCK1	Startup area will be set to Block 1.

◆ flash_event_t

enum flash_event_t	
Event types returned by the ISR callback when used in Data Flash BGO mode	
Enumerator	
FLASH_EVENT_ERASE_COMPLETE	Erase operation successfully completed.
FLASH_EVENT_WRITE_COMPLETE	Write operation successfully completed.
FLASH_EVENT_BLANK	Blank check operation successfully completed. Specified area is blank.
FLASH_EVENT_NOT_BLANK	Blank check operation successfully completed. Specified area is NOT blank.
FLASH_EVENT_ERR_DF_ACCESS	Data Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CF_ACCESS	Code Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CMD_LOCKED	Operation failed, FCU is in Locked state (often result of an illegal command)
FLASH_EVENT_ERR_FAILURE	Erase or Program Operation failed.
FLASH_EVENT_ERR_ONE_BIT	A 1-bit error has been corrected when reading the flash memory area by the sequencer.

◆ flash_id_code_mode_t

enum flash_id_code_mode_t	
ID Code Modes for writing to ID code registers	
Enumerator	
FLASH_ID_CODE_MODE_UNLOCKED	ID code is ignored.
FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT	ID code is checked. All erase is available.
FLASH_ID_CODE_MODE_LOCKED	ID code is checked.

◆ flash_status_t

enum flash_status_t	
Flash status	
Enumerator	
FLASH_STATUS_IDLE	The flash is idle.
FLASH_STATUS_BUSY	The flash is currently processing a command.

◆ flash_arc_t

enum flash_arc_t	
Anti-rollback counter selection.	
Enumerator	
FLASH_ARC_SEC	Anti-Rollback Counter for the secure application.
FLASH_ARC_OEMBL	Anti-Rollback Counter for the OEM bootloader.
FLASH_ARC_NSEC_0	Anti-Rollback Counter for non-secure application 0 (use when a single non-secure counter is available or when multiple non-secure counters are available)
FLASH_ARC_NSEC_1	Anti-Rollback Counter for non-secure application 1 (use when multiple non-secure counters are available)
FLASH_ARC_NSEC_2	Anti-Rollback Counter for non-secure application 2 (use when multiple non-secure counters are available)
FLASH_ARC_NSEC_3	Anti-Rollback Counter for non-secure application 3 (use when multiple non-secure counters are available)

5.3.15.2 SD/MMC Interface

Interfaces » Storage

Detailed Description

Interface for accessing SD, eMMC, and SDIO devices.

Summary

The r_sdhi interface provides standard SD and eMMC media functionality. This interface also supports SDIO.

Data Structures

struct	sdmmc_response_t.status_b
struct	sdmmc_status_t
struct	sdmmc_device_t
struct	sdmmc_callback_args_t
struct	sdmmc_write_io_args_t
struct	sdmmc_read_io_ext_args_t
struct	sdmmc_write_io_ext_args_t
struct	sdmmc_cfg_t
struct	sdmmc_api_t
struct	sdmmc_instance_t

Typedefs

typedef void	sdmmc_ctrl_t

Enumerations

enum	sdmmc_card_type_t
enum	sdmmc_bus_width_t
enum	sdmmc_io_transfer_mode_t
enum	sdmmc_io_address_mode_t
enum	sdmmc_io_write_mode_t
enum	sdmmc_event_t
enum	sdmmc_card_detect_t
enum	sdmmc_write_protect_t

enum	sdmmc_r1_state_t

Data Structure Documentation

◆ sdmmc_response_t.status_b

struct sdmmc_response_t.status_b
SDIO Card Status Register.

◆ sdmmc_status_t

struct sdmmc_status_t		
Current status.		
Data Fields		
bool	initialized	False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected), true otherwise. If ready is false, call sdmmc_api_t::mediaInit to reinitialize it
bool	transfer_in_progress	true = Card is busy
bool	card_inserted	Card detect status, true if card detect is not used.

◆ sdmmc_device_t

struct sdmmc_device_t		
Information obtained from the media device.		
Data Fields		
sdmmc_card_type_t	card_type	SD, eMMC, or SDIO.
bool	write_protected	true = Card is write protected
uint32_t	clock_rate	Current clock rate.
uint32_t	sector_count	Sector count.
uint32_t	sector_size_bytes	Sector size.
uint32_t	erase_sector_count	Minimum erasable unit (in 512 byte sectors)

◆ sdmmc_callback_args_t

struct sdmmc_callback_args_t
Callback function parameter data

Data Fields		
sdmmc_event_t	event	The event can be used to identify what caused the callback.
sdmmc_response_t	response	Response from card, only valid if SDMMC_EVENT_RESPONSE is set in event.
void *	p_context	Placeholder for user data.

◆ sdmmc_write_io_args_t

struct sdmmc_write_io_args_t
Non-secure arguments for writeIo guard function

◆ sdmmc_read_io_ext_args_t

struct sdmmc_read_io_ext_args_t
Non-secure arguments for readIoExt guard function

◆ sdmmc_write_io_ext_args_t

struct sdmmc_write_io_ext_args_t
Non-secure arguments for writeIoExt guard function

◆ sdmmc_cfg_t

struct sdmmc_cfg_t	
SD/MMC Configuration	
Data Fields	
uint8_t	channel
	Channel of SD/MMC host interface.
sdmmc_bus_width_t	bus_width
	Device bus width is 1, 4 or 8 bits wide.
transfer_instance_t const *	p_lower_lvl_transfer
	Transfer instance used to move data with DMA or DTC.
void(*	p_callback)(sdmmc_callback_args_t *p_args)

	Pointer to callback function.
void *	p_context
	User defined context passed into callback function.
void const *	p_extend
	SD/MMC hardware dependent configuration.
uint32_t	block_size
sdmmc_card_detect_t	card_detect
sdmmc_write_protect_t	write_protect
IRQn_Type	access_irq
	Access IRQ number.
IRQn_Type	sdio_irq
	SDIO IRQ number.
IRQn_Type	card_irq
	Card IRQ number.
IRQn_Type	dma_req_irq
	DMA request IRQ number.
uint8_t	access_ipl
	Access interrupt priority.

uint8_t	sdio_ipl
	SDIO interrupt priority.
uint8_t	card_ipl
	Card interrupt priority.
uint8_t	dma_req_ipl
	DMA request interrupt priority.

Field Documentation

◆ block_size

uint32_t sdmmc_cfg_t::block_size

Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

◆ card_detect

sdmmc_card_detect_t sdmmc_cfg_t::card_detect

Whether or not card detection is used.

◆ write_protect

sdmmc_write_protect_t sdmmc_cfg_t::write_protect

Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.

◆ sdmmc_api_t

struct sdmmc_api_t

SD/MMC functions implemented at the HAL layer API.

Data Fields

fsp_err_t(*	open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)
fsp_err_t(*	read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

fsp_err_t(*	write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t(*	readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
fsp_err_t(*	writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
fsp_err_t(*	readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t(*	writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t(*	ioIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)
fsp_err_t(*	statusGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)
fsp_err_t(*	erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t(*	callbackSet)(sdmmc_ctrl_t *const p_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void *const p_context, sdmmc_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(sdmmc_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)

Open the SD/MMC driver.

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[in]	p_cfg	Pointer to SD/MMC instance configuration structure.

◆ mediaInit

fsp_err_t(* sdmmc_api_t::mediaInit) (sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)

Initializes an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[out]	p_device	Pointer to store device information.

◆ read

fsp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	start_sector	First sector address to read.
[in]	sector_count	Number of sectors to read. All sectors must be in the range of sdmmc_device_t::sector_count.

◆ write

fsp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)

Write data to SD/MMC channel. This API is not supported for SDIO devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	start_sector	First sector address to write to.
[in]	sector_count	Number of sectors to write. All sectors must be in the range of sdmmc_device_t::sector_count.

◆ readIo

fsp_err_t(* sdmmc_api_t::readIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_data	Pointer to location to store data byte.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.

◆ writeIo

fsp_err_t(* sdmmc_api_t::writeIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in,out]	p_data	Pointer to data byte to write. Read data is also provided here if read_after_write is true.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	read_after_write	Whether or not to read back the same register after writing

◆ readIoExt

fsp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to read, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ writeIoExt

fsp_err_t(* sdmmc_api_t::writeIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	function_number	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to write, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ ioIntEnable

fsp_err_t(* sdmmc_api_t::ioIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	enable	Interrupt enable = true, interrupt disable = false.

◆ statusGet

fsp_err_t(* sdmmc_api_t::statusGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)

Get SD/MMC device status.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_status	Pointer to current driver status.

◆ erase

fsp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	start_sector	First sector to erase. Must be a multiple of sdmmc_device_t::erase_sector_count.
[in]	sector_count	Number of sectors to erase. Must be a multiple of sdmmc_device_t::erase_sector_count. All sectors must be in the range of sdmmc_device_t::sector_count.

◆ callbackSet

fsp_err_t(* sdmmc_api_t::callbackSet) (sdmmc_ctrl_t *const p_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void *const p_context, sdmmc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in sdmmc_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)

Close open SD/MMC device.

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.

◆ sdmmc_instance_t

struct sdmmc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
sdmmc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
sdmmc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
sdmmc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ sdmmc_ctrl_t

typedef void sdmmc_ctrl_t
SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls.

Enumeration Type Documentation

◆ sdmmc_card_type_t

enum sdmmc_card_type_t	
SD/MMC media uses SD protocol or MMC protocol.	
Enumerator	
SDMMC_CARD_TYPE_MMC	The media is an eMMC device.
SDMMC_CARD_TYPE_SD	The media is an SD card.
SDMMC_CARD_TYPE_SDIO	The media is an SDIO card.

◆ sdmmc_bus_width_t

enum sdmmc_bus_width_t	
SD/MMC data bus is 1, 4 or 8 bits wide.	
Enumerator	
SDMMC_BUS_WIDTH_1_BIT	Data bus is 1 bit wide.
SDMMC_BUS_WIDTH_4_BITS	Data bus is 4 bits wide.
SDMMC_BUS_WIDTH_8_BITS	Data bus is 8 bits wide.

◆ sdmmc_io_transfer_mode_t

enum sdmmc_io_transfer_mode_t	
SDIO transfer mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_MODE_TRANSFER_BYTE	SDIO byte transfer mode.
SDMMC_IO_MODE_TRANSFER_BLOCK	SDIO block transfer mode.

◆ sdmmc_io_address_mode_t

enum sdmmc_io_address_mode_t	
SDIO address mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_ADDRESS_MODE_FIXED	Write all data to the same address.
SDMMC_IO_ADDRESS_MODE_INCREMENT	Increment destination address after each write.

◆ sdmmc_io_write_mode_t

enum sdmmc_io_write_mode_t	
Controls the RAW (read after write) flag of CMD52. Used to read back the status after writing a control register.	
Enumerator	
SDMMC_IO_WRITE_MODE_NO_READ	Write only (do not read back)
SDMMC_IO_WRITE_READ_AFTER_WRITE	Read back the register after write.

◆ sdmmc_event_t

enum sdmmc_event_t	
Events that can trigger a callback function	
Enumerator	
SDMMC_EVENT_CARD_REMOVED	Card removed event.
SDMMC_EVENT_CARD_INSERTED	Card inserted event.
SDMMC_EVENT_RESPONSE	Response event.
SDMMC_EVENT_SDIO	IO event.
SDMMC_EVENT_TRANSFER_COMPLETE	Read or write complete.
SDMMC_EVENT_TRANSFER_ERROR	Read or write failed.
SDMMC_EVENT_ERASE_COMPLETE	Erase completed.
SDMMC_EVENT_ERASE_BUSY	Erase timeout, poll sdmmc_api_t::statusGet.

◆ sdmmc_card_detect_t

enum sdmmc_card_detect_t	
Card detection configuration options.	
Enumerator	
SDMMC_CARD_DETECT_NONE	Card detection unused.
SDMMC_CARD_DETECT_CD	Card detection using the CD pin.

◆ sdmmc_write_protect_t

enum sdmmc_write_protect_t	
Write protection configuration options.	
Enumerator	
SDMMC_WRITE_PROTECT_NONE	Write protection unused.
SDMMC_WRITE_PROTECT_WP	Write protection using WP pin.

◆ sdmmc_r1_state_t

enum sdmmc_r1_state_t	
Card state when receiving the prior command.	
Enumerator	
SDMMC_R1_STATE_IDLE	Idle State.
SDMMC_R1_STATE_READY	Ready State.
SDMMC_R1_STATE_IDENT	Identification State.
SDMMC_R1_STATE_STBY	Stand-by State.
SDMMC_R1_STATE_TRAN	Transfer State.
SDMMC_R1_STATE_DATA	Sending-data State.
SDMMC_R1_STATE_RCV	Receive-data State.
SDMMC_R1_STATE_PRG	Programming State.
SDMMC_R1_STATE_DIS	Disconnect State (between programming and

	stand-by)
SDMMC_R1_STATE_IO	This is an I/O card and memory states do not apply.
SDMMC_R1_STATE_IDLE	Idle State.
SDMMC_R1_STATE_READY	Ready State.
SDMMC_R1_STATE_IDENT	Identification State.
SDMMC_R1_STATE_STBY	Stand-by State.
SDMMC_R1_STATE_TRAN	Transfer State.
SDMMC_R1_STATE_DATA	Sending-data State.
SDMMC_R1_STATE_RCV	Receive-data State.
SDMMC_R1_STATE_PRG	Programming State.
SDMMC_R1_STATE_DIS	Disconnect State (between programming and stand-by)
SDMMC_R1_STATE_IO	This is an I/O card and memory states do not apply.

5.3.15.3 SPI Flash Interface

Interfaces » Storage

Detailed Description

Interface for accessing external SPI flash devices.

Summary

The SPI flash API provides an interface that configures, writes, and erases sectors in SPI flash devices.

Data Structures

struct	spi_flash_erase_command_t
struct	spi_flash_direct_transfer_t

struct	spi_flash_cfg_t
struct	spi_flash_status_t
struct	spi_flash_api_t
struct	spi_flash_instance_t

Typedefs

typedef void	spi_flash_ctrl_t

Enumerations

enum	spi_flash_read_mode_t
enum	spi_flash_protocol_t
enum	spi_flash_address_bytes_t
enum	spi_flash_data_lines_t
enum	spi_flash_dummy_clocks_t
enum	spi_flash_direct_transfer_dir_t

Data Structure Documentation

◆ spi_flash_erase_command_t

struct spi_flash_erase_command_t		
Structure to define an erase command and associated erase size.		
Data Fields		
uint16_t	command	Erase command.
uint32_t	size	Size of erase for associated command, set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE for chip erase.

◆ spi_flash_direct_transfer_t

struct spi_flash_direct_transfer_t		
Structure to define a direct transfer.		
Data Fields		
union spi_flash_direct_transfer_t	data_u	
uint32_t	address	Starting address.
uint16_t	command	Transfer command.

uint8_t	dummy_cycles	Number of dummy cycles.
uint8_t	command_length	Command length.
uint8_t	address_length	Address length.
uint8_t	data_length	Data length.

◆ spi_flash_cfg_t

struct spi_flash_cfg_t		
User configuration structure used by the open function		
Data Fields		
spi_flash_protocol_t	spi_protocol	Initial SPI protocol. SPI protocol can be changed in spi_flash_api_t::spiProtocolSet.
spi_flash_read_mode_t	read_mode	Read mode.
spi_flash_address_bytes_t	address_bytes	Number of bytes used to represent the address.
spi_flash_dummy_clocks_t	dummy_clocks	Number of dummy clocks to use for fast read operations.
spi_flash_data_lines_t	page_program_address_lines	Number of lines used to send address for page program command. This should either be 1 or match the number of lines used in the selected read mode.
uint8_t	write_status_bit	Which bit determines write status.
uint8_t	write_enable_bit	Which bit determines write status.
uint32_t	page_size_bytes	Page size in bytes (maximum number of bytes for page program). Used to specify single continuous write size (bytes) in case of OSPI RAM.
uint8_t	page_program_command	Page program command.
uint8_t	write_enable_command	Command to enable write or erase, typically 0x06.
uint8_t	status_command	Command to read the write status.
uint8_t	read_command	Read command - OSPI SPI mode only.
uint8_t	xip_enter_command	Command to enter XIP mode.
uint8_t	xip_exit_command	Command to exit XIP mode.

uint8_t	erase_command_list_length	Length of erase command list.
spi_flash_erase_command_t const *	p_erase_command_list	List of all erase commands and associated sizes.
void const *	p_extend	Pointer to implementation specific extended configurations.

◆ spi_flash_status_t

struct spi_flash_status_t		
Status.		
Data Fields		
bool	write_in_progress	Whether or not a write is in progress. This is determined by reading the spi_flash_cfg_t::write_status_bit from the spi_flash_cfg_t::status_command.

◆ spi_flash_api_t

struct spi_flash_api_t	
SPI flash implementations follow this API.	
Data Fields	
fsp_err_t(*	open)(spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg)
fsp_err_t(*	directWrite)(spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)
fsp_err_t(*	directRead)(spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t(*	directTransfer)(spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)
fsp_err_t(*	spiProtocolSet)(spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol)
fsp_err_t(*	write)(spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)

fsp_err_t(*	erase)(spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)
fsp_err_t(*	statusGet)(spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status)
fsp_err_t(*	xipEnter)(spi_flash_ctrl_t *const p_ctrl)
fsp_err_t(*	xipExit)(spi_flash_ctrl_t *const p_ctrl)
fsp_err_t(*	bankSet)(spi_flash_ctrl_t *const p_ctrl, uint32_t bank)
fsp_err_t(*	autoCalibrate)(spi_flash_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(spi_flash_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* spi_flash_api_t::open) (spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg)

Open the SPI flash driver module.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_cfg	Pointer to a configuration structure

◆ directWrite

fsp_err_t(* spi_flash_api_t::directWrite) (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)

Write raw data to the SPI flash.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_src	Pointer to raw data to write, must include any required command/address
[in]	bytes	Number of bytes to write
[in]	read_after_write	If true, the slave select remains asserted and the peripheral does not return to direct communications mode. If false, the slave select is deasserted and memory mapped access is possible after this function returns if the device is not busy.

◆ directRead

fsp_err_t(* spi_flash_api_t::directRead) (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read raw data from the SPI flash. Must follow a call to spi_flash_api_t::directWrite.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[out]	p_dest	Pointer to read raw data into
[in]	bytes	Number of bytes to read

◆ directTransfer

fsp_err_t(* spi_flash_api_t::directTransfer) (spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)

Direct Read/Write raw data to the SPI flash.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_data	Pointer to command, address and data values and lengths
[in]	direction	Direct Read/Write

◆ spiProtocolSet

fsp_err_t(* spi_flash_api_t::spiProtocolSet) (spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol)

Change the SPI protocol in the driver. The application must change the SPI protocol on the device.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	spi_protocol	Desired SPI protocol

◆ write

fsp_err_t(* spi_flash_api_t::write) (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)

Program a page of data to the flash.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_src	The memory address of the data to write to the flash device
[in]	p_dest	The location in the flash device address space to write the data to
[in]	byte_count	The number of bytes to write

◆ erase

fsp_err_t(* spi_flash_api_t::erase) (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)

Erase a certain number of bytes of the flash.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_device_address	The location in the flash device address space to start the erase from
[in]	byte_count	The number of bytes to erase. Set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE to erase entire chip.

◆ statusGet

fsp_err_t(* spi_flash_api_t::statusGet) (spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status)

Get the write or erase status of the flash.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[out]	p_status	Current status of the SPI flash device stored here.

◆ xipEnter

fsp_err_t(* spi_flash_api_t::xipEnter) (spi_flash_ctrl_t *const p_ctrl)

Enter XIP mode.

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ xipExit

fsp_err_t(* spi_flash_api_t::xipExit) (spi_flash_ctrl_t *const p_ctrl)

Exit XIP mode.

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ bankSet

fsp_err_t(* spi_flash_api_t::bankSet) (spi_flash_ctrl_t *const p_ctrl, uint32_t bank)

Select the bank to access. See implementation for details.

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	bank	The bank number

◆ autoCalibrate

fsp_err_t(* spi_flash_api_t::autoCalibrate) (spi_flash_ctrl_t *const p_ctrl)

AutoCalibrate the SPI flash driver module. Expected to be used when auto-calibrating OSPI RAM device.

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ close

fsp_err_t(* spi_flash_api_t::close) (spi_flash_ctrl_t *const p_ctrl)

Close the SPI flash driver module.

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ spi_flash_instance_t

struct spi_flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
spi_flash_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
spi_flash_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
spi_flash_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ spi_flash_ctrl_t

typedef void spi_flash_ctrl_t
SPI flash control block. Allocate an instance specific control block to pass into the SPI flash API calls.

Enumeration Type Documentation

◆ spi_flash_read_mode_t

enum spi_flash_read_mode_t	
Read mode.	
Enumerator	
SPI_FLASH_READ_MODE_STANDARD	Standard Read Mode (no dummy cycles)
SPI_FLASH_READ_MODE_FAST_READ	Fast Read Mode (dummy cycles between address and data)
SPI_FLASH_READ_MODE_FAST_READ_DUAL_OUTPUT	Fast Read Dual Output Mode (data on 2 lines)
SPI_FLASH_READ_MODE_FAST_READ_DUAL_IO	Fast Read Dual I/O Mode (address and data on 2 lines)
SPI_FLASH_READ_MODE_FAST_READ_QUAD_OUTPUT	Fast Read Quad Output Mode (data on 4 lines)
SPI_FLASH_READ_MODE_FAST_READ_QUAD_IO	Fast Read Quad I/O Mode (address and data on 4 lines)

◆ spi_flash_protocol_t

enum spi_flash_protocol_t	
SPI protocol.	
Enumerator	
SPI_FLASH_PROTOCOL_EXTENDED_SPI	Extended SPI mode (commands on 1 line)
SPI_FLASH_PROTOCOL_QPI	QPI mode (commands on 4 lines). Note that the application must ensure the device is in QPI mode.
SPI_FLASH_PROTOCOL_SOPI	SOPI mode (command and data on 8 lines). Note that the application must ensure the device is in SOPI mode.
SPI_FLASH_PROTOCOL_DOPI	DOPI mode (command and data on 8 lines, dual data rate). Note that the application must ensure the device is in DOPI mode.
SPI_FLASH_PROTOCOL_1S_1S_1S	1S-1S-1S protocol mode
SPI_FLASH_PROTOCOL_4S_4D_4D	4S-4D-4D protocol mode
SPI_FLASH_PROTOCOL_8D_8D_8D	8D-8D-8D protocol mode
SPI_FLASH_PROTOCOL_1S_2S_2S	1S-2S-2S protocol mode
SPI_FLASH_PROTOCOL_2S_2S_2S	2S-2S-2S protocol mode
SPI_FLASH_PROTOCOL_1S_4S_4S	1S-4S-4S protocol mode
SPI_FLASH_PROTOCOL_4S_4S_4S	4S-4S-4S protocol mode

◆ spi_flash_address_bytes_t

enum spi_flash_address_bytes_t	
Number of bytes in the address.	
Enumerator	
SPI_FLASH_ADDRESS_BYTES_1	1-byte address phase
SPI_FLASH_ADDRESS_BYTES_2	2-byte address phase
SPI_FLASH_ADDRESS_BYTES_3	3-byte address phase
SPI_FLASH_ADDRESS_BYTES_4	4-byte address phase
SPI_FLASH_ADDRESS_BYTES_4_4BYTE_READ_CODE	4 address bytes using standard 4-byte command set.

◆ spi_flash_data_lines_t

enum spi_flash_data_lines_t	
Number of data lines used.	
Enumerator	
SPI_FLASH_DATA_LINES_1	1 data line
SPI_FLASH_DATA_LINES_2	2 data lines
SPI_FLASH_DATA_LINES_4	4 data lines

◆ spi_flash_dummy_clocks_t

enum spi_flash_dummy_clocks_t	
Number of dummy cycles for fast read operations.	
Enumerator	
SPI_FLASH_DUMMY_CLOCKS_0	0 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_1	1 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_2	2 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_3	3 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_4	4 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_5	5 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_6	6 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_7	7 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_8	8 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_9	9 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_10	10 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_11	11 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_12	12 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_13	13 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_14	14 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_15	15 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_16	16 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_17	17 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_18	18 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_19	19 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_20	20 dummy clocks

SPI_FLASH_DUMMY_CLOCKS_21	21 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_22	22 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_23	23 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_24	24 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_25	25 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_26	26 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_27	27 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_28	28 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_29	29 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_30	30 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_31	31 dummy clocks

◆ spi_flash_direct_transfer_dir_t

enum spi_flash_direct_transfer_dir_t
Direct Read and Write direction

5.3.15.4 Block Media Interface

Interfaces » Storage

Detailed Description

Interface for block media memory access.

Summary

The block media interface supports reading, writing, and erasing media devices. All functions are non-blocking if possible. The callback is used to determine when an operation completes.

Data Structures

struct	rm_block_media_info_t

struct	rm_block_media_callback_args_t
struct	rm_block_media_cfg_t
struct	rm_block_media_status_t
struct	rm_block_media_api_t
struct	rm_block_media_instance_t

Typedefs

typedef void	rm_block_media_ctrl_t

Enumerations

enum	rm_block_media_event_t

Data Structure Documentation

◆ rm_block_media_info_t

struct rm_block_media_info_t		
Block media device information supported by the instance		
Data Fields		
uint32_t	sector_size_bytes	Sector size in bytes.
uint32_t	num_sectors	Total number of sectors.
bool	reentrant	True if connected block media driver is reentrant.
bool	write_protected	True if connected block media device is write protected.

◆ rm_block_media_callback_args_t

struct rm_block_media_callback_args_t		
Callback function parameter data		
Data Fields		
rm_block_media_event_t	event	The event can be used to identify what caused the callback.
void *	p_context	Placeholder for user data.

◆ rm_block_media_cfg_t

struct rm_block_media_cfg_t
User configuration structure, used in open function
Data Fields

void(*	p_callback)(rm_block_media_callback_args_t *p_args)
	Pointer to callback function.
void *	p_context
	User defined context passed into callback function.
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ rm_block_media_status_t

struct rm_block_media_status_t		
Current status		
Data Fields		
bool	initialized	False if rm_block_media_api_t::mediaInit has not been called since media was inserted, true otherwise.
bool	busy	True if media is busy with a previous write/erase operation.
bool	media_inserted	Media insertion status, true if media is not removable.

◆ rm_block_media_api_t

struct rm_block_media_api_t	
Block media interface API.	
Data Fields	
fsp_err_t(*	open)(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)

fsp_err_t(*	write)(rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t(*	erase)(rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t(*	callbackSet)(rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t(*	statusGet)(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t(*	infoGet)(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t(*	close)(rm_block_media_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_block_media_api_t::open) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)

Initialize block media device. rm_block_media_api_t::mediaInit must be called to complete the initialization procedure.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ mediaInit

fsp_err_t(* rm_block_media_api_t::mediaInit) (rm_block_media_ctrl_t *const p_ctrl)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.

◆ read

fsp_err_t(* rm_block_media_api_t::read) (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)

Reads blocks of data from the specified memory device address to the location specified by the caller.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[out]	p_dest_address	Destination to read the data into.
[in]	block_address	Block address to read the data from.
[in]	num_blocks	Number of blocks of data to read.

◆ write

fsp_err_t(* rm_block_media_api_t::write) (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)

Writes blocks of data to the specified device memory address.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[in]	p_src_address	Address to read the data to be written.
[in]	block_address	Block address to write the data to.
[in]	num_blocks	Number of blocks of data to write.

◆ erase

fsp_err_t(* rm_block_media_api_t::erase) (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)

Erases blocks of data from the memory device.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[in]	block_address	Block address to start the erase process at.
[in]	num_blocks	Number of blocks of data to erase.

◆ callbackSet

fsp_err_t(* rm_block_media_api_t::callbackSet) (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ statusGet

fsp_err_t(* rm_block_media_api_t::statusGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)

Get status of connected device.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[out]	p_status	Pointer to store current status.

◆ infoGet

fsp_err_t(* rm_block_media_api_t::infoGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)

Returns information about the block media device.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[out]	p_info	Pointer to information structure. All elements of this structure will be set by the function.

◆ close

fsp_err_t(* rm_block_media_api_t::close) (rm_block_media_ctrl_t *const p_ctrl)

Closes the module.

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.

◆ rm_block_media_instance_t

struct rm_block_media_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rm_block_media_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_block_media_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rm_block_media_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rm_block_media_ctrl_t

typedef void rm_block_media_ctrl_t

Block media API control block. Allocate an instance specific control block to pass into the block media API calls.

Enumeration Type Documentation

◆ rm_block_media_event_t

enum rm_block_media_event_t	
Events that can trigger a callback function	
Enumerator	
RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED	Media removed event.
RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED	Media inserted event.
RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE	Read, write, or erase completed.
RM_BLOCK_MEDIA_EVENT_ERROR	Error on media operation.
RM_BLOCK_MEDIA_EVENT_POLL_STATUS	Poll rm_block_media_api_t::statusGet for write/erase completion.
RM_BLOCK_MEDIA_EVENT_MEDIA_SUSPEND	Media suspended event.
RM_BLOCK_MEDIA_EVENT_MEDIA_RESUME	Media resumed event.
RM_BLOCK_MEDIA_EVENT_WAIT	Indication to user that they should wait for an interrupt on a pending operation.
RM_BLOCK_MEDIA_EVENT_WAIT_END	Indication to user that interrupt has been received and waiting can end.

5.3.15.5 FreeRTOS+FAT Port Interface

Interfaces » Storage

Detailed Description

Interface for FreeRTOS+FAT port.

Summary

The FreeRTOS+FAT port provides notifications for insertion and removal of removable media and provides initialization functions required by FreeRTOS+FAT.

Data Structures

struct	rm_freertos_plus_fat_callback_args_t

struct	rm_freertos_plus_fat_device_t
struct	rm_freertos_plus_fat_info_t
struct	rm_freertos_plus_fat_cfg_t
struct	rm_freertos_plus_fat_disk_cfg_t
struct	rm_freertos_plus_fat_api_t
struct	rm_freertos_plus_fat_instance_t

Typedefs

typedef void	rm_freertos_plus_fat_ctrl_t

Enumerations

enum	rm_freertos_plus_fat_event_t
enum	rm_freertos_plus_fat_type_t

Data Structure Documentation

◆ rm_freertos_plus_fat_callback_args_t

struct rm_freertos_plus_fat_callback_args_t		
Callback function parameter data		
Data Fields		
rm_freertos_plus_fat_event_t	event	The event can be used to identify what caused the callback.
void *	p_context	Placeholder for user data.

◆ rm_freertos_plus_fat_device_t

struct rm_freertos_plus_fat_device_t		
Information obtained from the media device.		
Data Fields		
uint32_t	sector_count	Sector count.
uint32_t	sector_size_bytes	Sector size in bytes.

◆ rm_freertos_plus_fat_info_t

struct rm_freertos_plus_fat_info_t
Basic information about a FAT volume.

Data Fields		
uint32_t	sector_size	Sector size.
uint32_t	sectors_per_cluster	Sectors per cluster.
uint32_t	total_sectors	Total number of sectors.
uint32_t	free_sectors	Current free sectors.
uint8_t *	p_volume_label	Volume label.
rm_freertos_plus_fat_type_t	type	FAT type for this disk.
uint8_t	partition_number	Partition number for this disk.

◆ rm_freertos_plus_fat_cfg_t

struct rm_freertos_plus_fat_cfg_t

FreeRTOS+FAT configuration structure.

Data Fields

void(*	p_callback)(rm_freertos_plus_fat_callback_args_t *p_args)
	Pointer to callback function.
void *	p_context
	User defined context passed into callback function.
fsp_err_t(*	p_busy_callback)(void *p_busy_context)
void *	p_busy_context
	User defined context passed into busy callback function.
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ p_busy_callback

fsp_err_t(* rm_freertos_plus_fat_cfg_t::p_busy_callback) (void *p_busy_context)
The busy callback function is called when the device is busy with a write or erase operation. It can be used to pend the current thread while waiting for the operation to complete and to manage timeouts if required. Set to NULL if pending the thread is not required and the thread will block waiting for the device busy status to clear. If this function returns an error (anything except FSP_SUCCESS), the upper layer function returns FF_ERR_IOMAN_DRIVER_FATAL_ERROR.

◆ rm_freertos_plus_fat_disk_cfg_t

struct rm_freertos_plus_fat_disk_cfg_t		
Initializing a FAT disk.		
Data Fields		
rm_freertos_plus_fat_device_t	device	Device sector data.
uint32_t	cache_size_bytes	Chache memory size.
uint8_t *	p_cache	Pointer to cache memory.
uint8_t	partition_number	Partition number for this disk.

◆ rm_freertos_plus_fat_api_t

struct rm_freertos_plus_fat_api_t	
FreeRTOS+FAT functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)
fsp_err_t(*	diskInit)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)
fsp_err_t(*	diskDeinit)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)
fsp_err_t(*	infoGet)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)
fsp_err_t(*	close)(rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_freertos_plus_fat_api_t::open) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)

Open media device.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ mediaInit

fsp_err_t(* rm_freertos_plus_fat_api_t::mediaInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_device	Pointer to store device information.

◆ diskInit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)

Initializes a FreeRTOS+FAT FF_Disk_t structure.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_disk_cfg	Pointer to disk configurations
[out]	p_disk	Pointer to store FreeRTOS+FAT disk structure.

◆ diskDeinit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskDeinit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)

Deinitializes a FreeRTOS+FAT FF_Disk_t structure.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_disk_cfg	Pointer to disk configurations
[out]	p_disk	Pointer to store FreeRTOS+FAT disk structure.

◆ infoGet

fsp_err_t(* rm_freertos_plus_fat_api_t::infoGet) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)

Returns information about the media device.

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Pointer to information structure. All elements of this structure will be set by the function.

◆ close

fsp_err_t(* rm_freertos_plus_fat_api_t::close) (rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Close media device.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ rm_freertos_plus_fat_instance_t

struct rm_freertos_plus_fat_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rm_freertos_plus_fat_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_freertos_plus_fat_cfg_t const *const	p_cfg	Pointer to the configuration structure for this instance.

rm_freertos_plus_fat_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rm_freertos_plus_fat_ctrl_t

typedef void rm_freertos_plus_fat_ctrl_t
FreeRTOS+FAT control structure

Enumeration Type Documentation

◆ rm_freertos_plus_fat_event_t

enum rm_freertos_plus_fat_event_t	
Events that can trigger a callback function	
Enumerator	
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED	Media removed event.
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED	Media inserted event.
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_SUSPEND	Media suspended event.
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_RESUME	Media resumed event.

◆ rm_freertos_plus_fat_type_t

enum rm_freertos_plus_fat_type_t	
FAT filesystem types.	
Enumerator	
RM_FREERTOS_PLUS_FAT_TYPE_FAT32	FAT32 disk.
RM_FREERTOS_PLUS_FAT_TYPE_FAT16	FAT16 disk.
RM_FREERTOS_PLUS_FAT_TYPE_FAT12	FAT12 disk.

5.3.15.6 LittleFS Interface

Interfaces » Storage

Detailed Description

Interface for LittleFS access.

Summary

The LittleFS Port configures a fail-safe filesystem designed for microcontrollers on top of a lower level storage device.

Data Structures

struct	rm_littlefs_cfg_t
struct	rm_littlefs_api_t
struct	rm_littlefs_instance_t

Typedefs

typedef void	rm_littlefs_ctrl_t

Data Structure Documentation

◆ rm_littlefs_cfg_t

struct rm_littlefs_cfg_t		
User configuration structure, used in open function		
Data Fields		
struct lfs_config const *	p_lfs_cfg	Pointer LittleFS configuration structure.
void const *	p_extend	Pointer to hardware dependent configuration.

◆ rm_littlefs_api_t

struct rm_littlefs_api_t	
LittleFS Port interface API.	
Data Fields	
fsp_err_t(*	open)(rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rm_littlefs_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_littlefs_api_t::open) (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)

Initialize The lower level storage device.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

fsp_err_t(* rm_littlefs_api_t::close) (rm_littlefs_ctrl_t *const p_ctrl)

Closes the module and lower level storage device.

Parameters

[in]	p_ctrl	Control block set in rm_littlefs_api_t::open call.

◆ rm_littlefs_instance_t

struct rm_littlefs_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rm_littlefs_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_littlefs_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rm_littlefs_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rm_littlefs_ctrl_t

typedef void rm_littlefs_ctrl_t

LittleFS Port API control block. Allocate an instance specific control block to pass into the LittleFS Port API calls.

5.3.16 System

Interfaces

Detailed Description

System Interfaces.

Modules

	CGC Interface Interface for clock generation.
	ELC Interface Interface for the Event Link Controller.
	I/O Port Interface Interface for accessing I/O ports and configuring I/O functionality.

5.3.16.1 CGC Interface

Interfaces » System

Detailed Description

Interface for clock generation.

Summary

The CGC interface provides the ability to configure and use all of the CGC module's capabilities. Among the capabilities is the selection of several clock sources to use as the system clock source. Additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

Data Structures

struct	cgc_callback_args_t

struct	cgc_pll_cfg_t
struct	cgc_divider_cfg_t
struct	cgc_cfg_t
struct	cgc_clocks_cfg_t
struct	cgc_api_t
struct	cgc_instance_t

Typedefs

typedef void	cgc_ctrl_t

Enumerations

enum	cgc_event_t
enum	cgc_clock_t
enum	cgc_pll_div_t
enum	cgc_pll_out_div_t
enum	cgc_sys_clock_div_t
enum	cgc_pin_output_control_t
enum	cgc_usb_clock_div_t
enum	cgc_clock_change_t

Data Structure Documentation

◆ cgc_callback_args_t

struct cgc_callback_args_t		
Callback function parameter data		
Data Fields		
cgc_event_t	event	The event can be used to identify what caused the callback.
void *	p_context	Placeholder for user data.

◆ cgc_pll_cfg_t

struct cgc_pll_cfg_t		
Clock configuration structure - Used as an input parameter to the cgc_api_t::clockStart function for the PLL clock.		
Data Fields		
cgc_clock_t	source_clock	PLL source clock (main oscillator or HOCO)
cgc_pll_div_t	divider	PLL divider.
cgc_pll_mul_t	multiplier	PLL multiplier.
cgc_pll_out_div_t	out_div_p	PLL divisor for output clock P.
cgc_pll_out_div_t	out_div_q	PLL divisor for output clock Q.
cgc_pll_out_div_t	out_div_r	PLL divisor for output clock R.

◆ cgc_divider_cfg_t

struct cgc_divider_cfg_t
Clock configuration structure - Used as an input parameter to the cgc_api_t::systemClockSet and cgc_api_t::systemClockGet functions.

◆ cgc_cfg_t

struct cgc_cfg_t	
Configuration options.	
Data Fields	
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ cgc_clocks_cfg_t

struct cgc_clocks_cfg_t		
Clock configuration		
Data Fields		
cgc_clock_t	system_clock	System clock source enumeration.
cgc_pll_cfg_t	pll_cfg	PLL configuration structure.
cgc_pll_cfg_t	pll2_cfg	PLL2 configuration structure.
cgc_divider_cfg_t	divider_cfg	Clock dividers structure.
cgc_clock_change_t	loco_state	State of LOCO.
cgc_clock_change_t	moco_state	State of MOCO.

cgc_clock_change_t	hoco_state	State of HOCO.
cgc_clock_change_t	mainosc_state	State of Main oscillator.
cgc_clock_change_t	pll_state	State of PLL.
cgc_clock_change_t	pll2_state	State of PLL2.
cgc_clock_change_t	subosc_state	State of Sub oscillator.

◆ cgc_api_t

struct cgc_api_t	
CGC functions implemented at the HAL layer follow this API.	
Data Fields	
fsp_err_t(*	open)(cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)
fsp_err_t(*	clocksCfg)(cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)
fsp_err_t(*	clockStart)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)
fsp_err_t(*	clockStop)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t(*	clockCheck)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t(*	systemClockSet)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)
fsp_err_t(*	systemClockGet)(cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)
fsp_err_t(*	oscStopDetectEnable)(cgc_ctrl_t *const p_ctrl)
fsp_err_t(*	oscStopDetectDisable)(cgc_ctrl_t *const p_ctrl)
fsp_err_t(*	oscStopStatusClear)(cgc_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(cgc_ctrl_t *const p_ctrl, void(*p_callback)(cgc_callback_args_t *), void *const p_context, cgc_callback_args_t *const p_callback_memory)

fsp_err_t(*	close)(cgc_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* cgc_api_t::open) (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)

Initial configuration

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ clocksCfg

fsp_err_t(* cgc_api_t::clocksCfg) (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)

Configure all system clocks.

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_clock_cfg	Pointer to desired configuration of system clocks

◆ clockStart

fsp_err_t(* cgc_api_t::clockStart) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)

Start a clock.

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Clock source to start
[in]	p_pll_cfg	Pointer to PLL configuration, can be NULL if clock_source is not CGC_CLOCK_PLL or CGC_CLOCK_PLL2

◆ clockStop

fsp_err_t(* cgc_api_t::clockStop) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Stop a clock.

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	The clock source to stop

◆ clockCheck

fsp_err_t(* cgc_api_t::clockCheck) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Check the stability of the selected clock.

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Which clock source to check for stability

◆ systemClockSet

fsp_err_t(* cgc_api_t::systemClockSet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)

Set the system clock.

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Clock source to set as system clock
[in]	p_divider_cfg	Pointer to the clock divider configuration

◆ systemClockGet

fsp_err_t(* cgc_api_t::systemClockGet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)

Get the system clock information.

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_clock_source	Returns the current system clock
[out]	p_divider_cfg	Returns the current system clock dividers

◆ oscStopDetectEnable

fsp_err_t(* cgc_api_t::oscStopDetectEnable) (cgc_ctrl_t *const p_ctrl)

Enable and optionally register a callback for Main Oscillator stop detection.

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ oscStopDetectDisable

fsp_err_t(* cgc_api_t::oscStopDetectDisable) (cgc_ctrl_t *const p_ctrl)

Disable Main Oscillator stop detection.

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ oscStopStatusClear

fsp_err_t(* cgc_api_t::oscStopStatusClear) (cgc_ctrl_t *const p_ctrl)

Clear the oscillator stop detection flag.

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ callbackSet

fsp_err_t(* cgc_api_t::callbackSet) (cgc_ctrl_t *const p_ctrl, void(*p_callback)(cgc_callback_args_t *), void *const p_context, cgc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the CGC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* cgc_api_t::close) (cgc_ctrl_t *const p_ctrl)

Close the CGC driver.

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ cgc_instance_t

struct cgc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
cgc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
cgc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
cgc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ cgc_ctrl_t

typedef void cgc_ctrl_t
CGC control block. Allocate an instance specific control block to pass into the CGC API calls.

Enumeration Type Documentation

◆ cgc_event_t

enum cgc_event_t	
Events that can trigger a callback function	
Enumerator	
CGC_EVENT_OSC_STOP_DETECT_NMI	Main oscillator stop detection has caused the NMI event.
CGC_EVENT_OSC_STOP_DETECT_MAIN_OSC	Main oscillator stop detection has caused the interrupt event.
CGC_EVENT_OSC_STOP_DETECT_SUBCLOCK	Subclock oscillator stop detection has caused the interrupt event.

◆ cgc_clock_t

enum cgc_clock_t	
System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider	
Enumerator	
CGC_CLOCK_HOCO	The high speed on chip oscillator.
CGC_CLOCK_MOCO	The middle speed on chip oscillator.
CGC_CLOCK_LOCO	The low speed on chip oscillator.
CGC_CLOCK_MAIN_OSC	The main oscillator.
CGC_CLOCK_SUBCLOCK	The subclock oscillator.
CGC_CLOCK_PLL	The PLL oscillator.
CGC_CLOCK_PLL2	The PLL2 oscillator.

◆ cgc_pll_div_t

enum cgc_pll_div_t	
PLL divider values	
Enumerator	
CGC_PLL_DIV_1	PLL divider of 1.
CGC_PLL_DIV_2	PLL divider of 2.
CGC_PLL_DIV_3	PLL divider of 3.
CGC_PLL_DIV_4	PLL divider of 4.
CGC_PLL_DIV_6	PLL divider of 6.

◆ cgc_pll_out_div_t

enum cgc_pll_out_div_t	
PLL clock output divisor.	
Enumerator	
CGC_PLL_OUT_DIV_2	PLL output clock divided by 2.
CGC_PLL_OUT_DIV_3	PLL output clock divided by 3.
CGC_PLL_OUT_DIV_4	PLL output clock divided by 4.
CGC_PLL_OUT_DIV_5	PLL output clock divided by 5.
CGC_PLL_OUT_DIV_6	PLL output clock divided by 6.
CGC_PLL_OUT_DIV_8	PLL output clock divided by 8.
CGC_PLL_OUT_DIV_9	PLL output clock divided by 9.
CGC_PLL_OUT_DIV_1_5	PLL output clock divided by 1.5.
CGC_PLL_OUT_DIV_16	PLL output clock divided by 16.

◆ cgc_sys_clock_div_t

enum cgc_sys_clock_div_t	
System clock divider values - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D.	
Enumerator	
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.
CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.
CGC_SYS_CLOCK_DIV_3	System clock divided by 3 (BCLK only)
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.
CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.
CGC_SYS_CLOCK_DIV_3	System clock divided by 3.
CGC_SYS_CLOCK_DIV_6	System clock divided by 6.
CGC_SYS_CLOCK_DIV_12	System clock divided by 12.
CGC_SYS_CLOCK_DIV_24	System clock divided by 24.
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.

CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.
CGC_SYS_CLOCK_DIV_3	System clock divided by 3.
CGC_SYS_CLOCK_DIV_6	System clock divided by 6.
CGC_SYS_CLOCK_DIV_12	System clock divided by 12.
CGC_SYS_CLOCK_DIV_24	System clock divided by 24.

◆ cgc_pin_output_control_t

enum cgc_pin_output_control_t	
Pin output control enable/disable (SDCLK, BCLK).	
Enumerator	
CGC_PIN_OUTPUT_CONTROL_ENABLE	Enable pin output.
CGC_PIN_OUTPUT_CONTROL_DISABLE	Disable pin output.

◆ cgc_usb_clock_div_t

enum cgc_usb_clock_div_t	
USB clock divider values	
Enumerator	
CGC_USB_CLOCK_DIV_2	Divide USB source clock by 2.
CGC_USB_CLOCK_DIV_3	Divide USB source clock by 3.
CGC_USB_CLOCK_DIV_4	Divide USB source clock by 4.
CGC_USB_CLOCK_DIV_5	Divide USB source clock by 5.

◆ cgc_clock_change_t

enum cgc_clock_change_t	
Clock options	
Enumerator	
CGC_CLOCK_CHANGE_START	Start the clock.
CGC_CLOCK_CHANGE_STOP	Stop the clock.
CGC_CLOCK_CHANGE_NONE	No change to the clock.

5.3.16.2 ELC Interface

Interfaces » System

Detailed Description

Interface for the Event Link Controller.

Data Structures

struct	elc_cfg_t
struct	elc_api_t
struct	elc_instance_t

Typedefs

typedef void	elc_ctrl_t

Enumerations

enum	elc_peripheral_t
enum	elc_software_event_t

Data Structure Documentation

◆ elc_cfg_t

struct elc_cfg_t
Main configuration structure for the Event Link Controller
Data Fields

elc_event_t const	link[ELC_PERIPHERAL_NUM]	Event link register settings.
void const *	p_extend	Extension parameter for hardware specific settings.

◆ elc_api_t

struct elc_api_t

ELC driver structure. General ELC functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(*	open)(elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(elc_ctrl_t *const p_ctrl)
fsp_err_t(*	softwareEventGenerate)(elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)
fsp_err_t(*	linkSet)(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)
fsp_err_t(*	linkBreak)(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)
fsp_err_t(*	enable)(elc_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(elc_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* elc_api_t::open) (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)

Initialize all links in the Event Link Controller.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* elc_api_t::close) (elc_ctrl_t *const p_ctrl)

Disable all links in the Event Link Controller and close the API.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ softwareEventGenerate

fsp_err_t(* elc_api_t::softwareEventGenerate) (elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)

Generate a software event in the Event Link Controller.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	eventNum	Software event number to be generated.

◆ linkSet

fsp_err_t(* elc_api_t::linkSet) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)

Create a single event link.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	peripheral	The peripheral block that will receive the event signal.
[in]	signal	The event signal.

◆ linkBreak

fsp_err_t(* elc_api_t::linkBreak) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

Break an event link.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	peripheral	The peripheral that should no longer be linked.

◆ enable

fsp_err_t(* elc_api_t::enable) (elc_ctrl_t *const p_ctrl)

Enable the operation of the Event Link Controller.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ disable

fsp_err_t(* elc_api_t::disable) (elc_ctrl_t *const p_ctrl)

Disable the operation of the Event Link Controller.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ elc_instance_t

struct elc_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
elc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
elc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
elc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ elc_ctrl_t

typedef void elc_ctrl_t
ELC control block. Allocate an instance specific control block to pass into the ELC API calls.

Enumeration Type Documentation

◆ elc_peripheral_t

enum elc_peripheral_t
Possible peripherals to be linked to event signals (not all available on all MCUs)

◆ elc_software_event_t

enum elc_software_event_t	
Software event number	
Enumerator	
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_2	Software event 2.
ELC_SOFTWARE_EVENT_3	Software event 3.
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.
ELC_SOFTWARE_EVENT_2	Software event 2.
ELC_SOFTWARE_EVENT_3	Software event 3.

5.3.16.3 I/O Port Interface

Interfaces » System

Detailed Description

Interface for accessing I/O ports and configuring I/O functionality.

Summary

The IOPort shared interface provides the ability to access the IOPorts of a device at both bit and port level. Port and pin direction can be changed.

Data Structures

struct	ioport_pin_cfg_t
struct	ioport_cfg_t

struct	ioport_api_t
struct	ioport_instance_t

Typedefs

typedef uint16_t	ioport_size_t IO port size. More...
typedef void	ioport_ctrl_t

Data Structure Documentation

◆ ioport_pin_cfg_t

struct ioport_pin_cfg_t		
Pin identifier and pin configuration value		
Data Fields		
uint32_t	pin_cfg	Pin configuration - Use ioport_cfg_options_t parameters to configure.
bsp_io_port_pin_t	pin	Pin identifier.

◆ ioport_cfg_t

struct ioport_cfg_t		
Multiple pin configuration data for loading into registers by R_IOPORT_Open()		
Data Fields		
uint16_t	number_of_pins	Number of pins for which there is configuration data.
ioport_pin_cfg_t const *	p_pin_cfg_data	Pin configuration data.
const void *	p_extend	Pointer to hardware extend configuration.

◆ ioport_api_t

struct ioport_api_t	
IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t(*	close)(ioport_ctrl_t *const p_ctrl)

fsp_err_t(*	pinsCfg)(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t(*	pinCfg)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)
fsp_err_t(*	pinEventInputRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)
fsp_err_t(*	pinEventOutputWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)
fsp_err_t(*	pinRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)
fsp_err_t(*	pinWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)
fsp_err_t(*	portDirectionSet)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)
fsp_err_t(*	portEventInputRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)
fsp_err_t(*	portEventOutputWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
fsp_err_t(*	portRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)
fsp_err_t(*	portWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Field Documentation

◆ open

fsp_err_t(* ioport_api_t::open) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use ioport_api_t::pinsCfg for runtime reconfiguration of multiple pins.

Parameters

[in]	p_ctrl	Pointer to control structure. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to pin configuration data array.

◆ close

fsp_err_t(* ioport_api_t::close) (ioport_ctrl_t *const p_ctrl)

Close the API.

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ pinsCfg

fsp_err_t(* ioport_api_t::pinsCfg) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Configure multiple pins.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration data array.

◆ pinCfg

fsp_err_t(* ioport_api_t::pinCfg) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)

Configure settings for an individual pin.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pin	Pin to be read.
[in]	cfg	Configuration options for the pin.

◆ pinEventInputRead

fsp_err_t(* ioport_api_t::pinEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)

Read the event input data of the specified pin and return the level.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pin	Pin to be read.
[in]	p_pin_event	Pointer to return the event data.

◆ pinEventOutputWrite

fsp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)

Write pin event data.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pin	Pin event data is to be written to.
[in]	pin_value	Level to be written to pin output event.

◆ pinRead

fsp_err_t(* ioport_api_t::pinRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)

Read level of a pin.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pin	Pin to be read.
[in]	p_pin_value	Pointer to return the pin level.

◆ pinWrite

fsp_err_t(* ioport_api_t::pinWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)

Write specified level to a pin.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pin	Pin to be written to.
[in]	level	State to be written to the pin.

◆ portDirectionSet

fsp_err_t(* ioport_api_t::portDirectionSet) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)

Set the direction of one or more pins on a port.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	port	Port being configured.
[in]	direction_values	Value controlling direction of pins on port.
[in]	mask	Mask controlling which pins on the port are to be configured.

◆ portEventInputRead

fsp_err_t(* ioport_api_t::portEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)

Read captured event data for a port.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	port	Port to be read.
[in]	p_event_data	Pointer to return the event data.

◆ portEventOutputWrite

fsp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)

Write event output data for a port.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	port	Port event data will be written to.
[in]	event_data	Data to be written as event data to specified port.
[in]	mask_value	Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port.

◆ portRead

fsp_err_t(* ioport_api_t::portRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)

Read states of pins on the specified port.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	port	Port to be read.
[in]	p_port_value	Pointer to return the port value.

◆ portWrite

fsp_err_t(* ioport_api_t::portWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Write to multiple pins on a port.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	port	Port to be written to.
[in]	value	Value to be written to the port.
[in]	mask	Mask controlling which pins on the port are written to.

◆ ioport_instance_t

struct ioport_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
ioport_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ioport_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ioport_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ioport_size_t

typedef uint16_t ioport_size_t
IO port size. IO port type used with ports

◆ ioport_ctrl_t

typedef void ioport_ctrl_t
IOPORT control block. Allocate an instance specific control block to pass into the IOPORT API calls.

5.3.17 Timers

Interfaces

Detailed Description

Timers Interfaces.

Modules

	POEG Interface Interface for the Port Output Enable for GPT.
	RTC Interface Interface for accessing the Realtime Clock.

	Three-Phase Interface Interface for three-phase timer functions.
	Timer Interface Interface for timer functions.

5.3.17.1 POEG Interface

Interfaces » Timers

Detailed Description

Interface for the Port Output Enable for GPT.

Defines the API and data structures for the Port Output Enable for GPT (POEG) interface.

Summary

The POEG disables GPT output pins based on configurable events.

Data Structures

struct	poeg_status_t
struct	poeg_callback_args_t
struct	poeg_cfg_t
struct	poeg_api_t
struct	poeg_instance_t

Typedefs

typedef void	poeg_ctrl_t

Enumerations

enum	poeg_state_t
enum	poeg_trigger_t
enum	poeg_gtetrg_polarity_t

enum	poeg_gtetrg_noise_filter_t

Data Structure Documentation

◆ poeg_status_t

struct poeg_status_t		
POEG status		
Data Fields		
poeg_state_t	state	Current state of POEG.

◆ poeg_callback_args_t

struct poeg_callback_args_t		
Callback function parameter data.		
Data Fields		
void *	p_context	Placeholder for user data, set in poeg_cfg_t.

◆ poeg_cfg_t

struct poeg_cfg_t	
User configuration structure, used in the open function.	
Data Fields	
poeg_trigger_t	trigger
	Select one or more triggers for the POEG.
poeg_gtetrg_polarity_t	polarity
	Select the polarity for the GTETRG pin.
poeg_gtetrg_noise_filter_t	noise_filter
	Configure the GTETRG noise filter.
void(*	p_callback)(poeg_callback_args_t *p_args)
void *	p_context
uint32_t	unit

	POEG unit to be used.
uint32_t	channel
	Channel 0 corresponds to GTETRGA, 1 to GTETRGB, etc.
IRQn_Type	irq
	Interrupt number assigned to this instance.
uint8_t	ipl
	POEG interrupt priority.
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ p_callback

void(* poeg_cfg_t::p_callback) (poeg_callback_args_t *p_args)

Callback called when a POEG interrupt occurs.

◆ p_context

void* poeg_cfg_t::p_context

Placeholder for user data. Passed to the user callback in poeg_callback_args_t.

◆ poeg_api_t

struct poeg_api_t

Port Output Enable for GPT (POEG) API structure. POEG functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)
fsp_err_t(*	statusGet)(poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status)

fsp_err_t(*	callbackSet)(poeg_ctrl_t *const p_ctrl, void(*p_callback)(poeg_callback_args_t *), void *const p_context, poeg_callback_args_t *const p_callback_memory)
fsp_err_t(*	outputDisable)(poeg_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(poeg_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(poeg_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* poeg_api_t::open) (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ statusGet

fsp_err_t(* poeg_api_t::statusGet) (poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status)

Gets the current driver state.

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.
[out]	p_status	Provides the current state of the POEG.

◆ callbackSet

fsp_err_t(* poeg_api_t::callbackSet) (poeg_ctrl_t *const p_ctrl, void(*p_callback)(poeg_callback_args_t *), void *const p_context, poeg_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call for this timer.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ outputDisable

fsp_err_t(* poeg_api_t::outputDisable) (poeg_ctrl_t *const p_ctrl)

Disables GPT output pins by software request.

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.

◆ reset

fsp_err_t(* poeg_api_t::reset) (poeg_ctrl_t *const p_ctrl)

Attempts to clear status flags to reenable GPT output pins. Confirm all status flags are cleared after calling this function by calling poeg_api_t::statusGet().

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.

◆ close

fsp_err_t(* poeg_api_t::close) (poeg_ctrl_t *const p_ctrl)

Disables POEG interrupt.

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.

◆ poeg_instance_t

struct poeg_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
poeg_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
poeg_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
poeg_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ poeg_ctrl_t

typedef void poeg_ctrl_t
POEG control block. Allocate an instance specific control block to pass into the POEG API calls.

Enumeration Type Documentation

◆ poeg_state_t

enum poeg_state_t	
POEG states.	
Enumerator	
POEG_STATE_NO_DISABLE_REQUEST	GPT output is not disabled by POEG.
POEG_STATE_PIN_DISABLE_REQUEST	GPT output disabled due to GTETRG pin level.
POEG_STATE_GPT_OR_COMPARATOR_DISABLE_REQUEST	GPT output disabled due to high speed analog comparator or GPT.
POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST	GPT output disabled due to main oscillator stop.
POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST	GPT output disabled due to poeg_api_t::outputDisable()
POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE	GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input.
POEG_STATE_NO_DISABLE_REQUEST	GPT output is not disabled by POEG.
POEG_STATE_PIN_DISABLE_REQUEST	GPT output disabled due to GTETRG pin level.
POEG_STATE_GPT_OR_COMPARATOR_DISABLE_REQUEST	GPT output disabled due to high speed analog comparator or GPT.
POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST	GPT output disabled due to main oscillator stop.
POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST	GPT output disabled due to poeg_api_t::outputDisable()
POEG_STATE_DSMIF0_DISABLE_REQUEST	GPT output disabled due to DSMIF0.
POEG_STATE_DSMIF1_DISABLE_REQUEST	GPT output disabled due to DSMIF1.
POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE	GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input.
POEG_STATE_NO_DISABLE_REQUEST	GPT output is not disabled by POEG.
POEG_STATE_PIN_DISABLE_REQUEST	GPT output disabled due to GTETRG pin level.
POEG_STATE_GPT_OR_COMPARATOR_DISABLE_R	GPT output disabled due to high speed analog

EQUEST	comparator or GPT.
POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST	GPT output disabled due to main oscillator stop.
POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST	GPT output disabled due to poeg_api_t::outputDisable()
POEG_STATE_DSMIF0_DISABLE_REQUEST	GPT output disabled due to DSMIF0.
POEG_STATE_DSMIF1_DISABLE_REQUEST	GPT output disabled due to DSMIF1.
POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE	GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input.

◆ poeg_trigger_t

enum poeg_trigger_t	
Triggers that will disable GPT output pins.	
Enumerator	
POEG_TRIGGER_SOFTWARE	Software disable is always supported with POEG. Select this option if no other triggers are used.
POEG_TRIGGER_PIN	Disable GPT output based on GTETRG input level.
POEG_TRIGGER_GPT_OUTPUT_LEVEL	Disable GPT output based on GPT output pin levels.
POEG_TRIGGER_OSCILLATION_STOP	Disable GPT output based on main oscillator stop.
POEG_TRIGGER_ACMPHS0	Disable GPT output based on ACMPHS0 comparator result.
POEG_TRIGGER_ACMPHS1	Disable GPT output based on ACMPHS1 comparator result.
POEG_TRIGGER_ACMPHS2	Disable GPT output based on ACMPHS2 comparator result.
POEG_TRIGGER_ACMPHS3	Disable GPT output based on ACMPHS3 comparator result.
POEG_TRIGGER_ACMPHS4	Disable GPT output based on ACMPHS4

	comparator result.
POEG_TRIGGER_ACMPHS5	Disable GPT output based on ACMPHS5 comparator result.
POEG_TRIGGER_SOFTWARE	Software disable is always supported with POEG. Select this option if no other triggers are used.
POEG_TRIGGER_PIN	Disable GPT output based on GTETRG input level.
POEG_TRIGGER_GPT_OUTPUT_LEVEL	Disable GPT output based on GPT output pin levels.
POEG_TRIGGER_OSCILLATION_STOP	Disable GPT output based on main oscillator stop.
POEG_TRIGGER_DSMIF0	Disable GPT output based on DSMIF0 overcurrent detection window notification.
POEG_TRIGGER_DSMIF1	Disable GPT output based on DSMIF1 overcurrent detection window notification.
POEG_TRIGGER_SOFTWARE	Software disable is always supported with POEG. Select this option if no other triggers are used.
POEG_TRIGGER_PIN	Disable GPT output based on GTETRG input level.
POEG_TRIGGER_GPT_OUTPUT_LEVEL	Disable GPT output based on GPT output pin levels.
POEG_TRIGGER_OSCILLATION_STOP	Disable GPT output based on main oscillator stop.
POEG_TRIGGER_DSMIF0	Disable GPT output based on DSMIF0 overcurrent detection window notification.
POEG_TRIGGER_DSMIF1	Disable GPT output based on DSMIF1 overcurrent detection window notification.

◆ poeg_gtetrg_polarity_t

enum poeg_gtetrg_polarity_t	
GTETRG polarity.	
Enumerator	
POEG_GTETRG_POLARITY_ACTIVE_HIGH	Disable GPT output based when GTETRG input level is high.
POEG_GTETRG_POLARITY_ACTIVE_LOW	Disable GPT output based when GTETRG input level is low.

◆ poeg_gtetrg_noise_filter_t

enum poeg_gtetrg_noise_filter_t	
GTETRG noise filter. For the input signal to pass through the noise filter, the active level set in poeg_gtetrg_polarity_t must be read 3 consecutive times at the sampling clock selected.	
Enumerator	
POEG_GTETRG_NOISE_FILTER_DISABLED	No noise filter applied to GTETRG input.
POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_1	Apply noise filter with sample clock equal to Clock source/1.
POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_8	Apply noise filter with sample clock equal to Clock source/8.
POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_32	Apply noise filter with sample clock equal to Clock source/32.
POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_128	Apply noise filter with sample clock equal to Clock source/128.

5.3.17.2 RTC Interface

Interfaces » Timers

Detailed Description

Interface for accessing the Realtime Clock.

Summary

The RTC Interface is for configuring Real Time Clock (RTC) functionality including alarm, periodic notification and error adjustment.

Data Structures

struct	rtc_callback_args_t
struct	rtc_error_adjustment_cfg_t
struct	rtc_alarm_time_t
struct	rtc_time_capture_t
struct	rtc_info_t
struct	rtc_cfg_t
struct	rtc_api_t
struct	rtc_instance_t

Typedefs

typedef struct tm	rtc_time_t
typedef void	rtc_ctrl_t

Enumerations

enum	rtc_event_t
enum	rtc_alarm_channel_t
enum	rtc_clock_source_t
enum	rtc_status_t
enum	rtc_error_adjustment_t
enum	rtc_error_adjustment_mode_t
enum	rtc_error_adjustment_period_t
enum	rtc_periodic_irq_select_t
enum	rtc_time_capture_source_t
enum	rtc_time_capture_mode_t
enum	rtc_time_capture_noise_filter_t

Data Structure Documentation

◆ rtc_callback_args_t

struct rtc_callback_args_t		
Callback function parameter data		
Data Fields		
rtc_event_t	event	The event can be used to identify what caused the callback (compare match or error).
void *	p_context	Placeholder for user data.

◆ rtc_error_adjustment_cfg_t

struct rtc_error_adjustment_cfg_t		
Time error adjustment value configuration		
Data Fields		
rtc_error_adjustment_mode_t	adjustment_mode	Automatic Adjustment Enable/Disable.
rtc_error_adjustment_period_t	adjustment_period	Error Adjustment period.
rtc_error_adjustment_t	adjustment_type	Time error adjustment setting.
uint32_t	adjustment_value	Value of the prescaler for error adjustment.

◆ rtc_alarm_time_t

struct rtc_alarm_time_t		
Alarm time setting structure		
Data Fields		
rtc_time_t	time	Time structure.
bool	sec_match	Enable the alarm based on a match of the seconds field.
bool	min_match	Enable the alarm based on a match of the minutes field.
bool	hour_match	Enable the alarm based on a match of the hours field.
bool	mday_match	Enable the alarm based on a match of the days field.
bool	mon_match	Enable the alarm based on a match of the months field.
bool	year_match	Enable the alarm based on a

		match of the years field.
bool	dayofweek_match	Enable the alarm based on a match of the dayofweek field.
bool	sunday_match	Enable the alarm on Sunday.
bool	monday_match	Enable the alarm on Monday.
bool	tuesday_match	Enable the alarm on Tuesday.
bool	wednesday_match	Enable the alarm on Wednesday.
bool	thursday_match	Enable the alarm on Thursday.
bool	friday_match	Enable the alarm on Friday.
bool	saturday_match	Enable the alarm on Saturday.
rtc_alarm_channel_t	channel	Select alarm 0 or alarm 1.

◆ rtc_time_capture_t

struct rtc_time_capture_t		
Time capture configuration structure		
Data Fields		
rtc_time_t	time	Time structure.
uint8_t	channel	Capture channel.
rtc_time_capture_source_t	source	Trigger source.
rtc_time_capture_noise_filter_t	noise_filter	Noise filter.
rtc_time_capture_mode_t	mode	Capture mode.

◆ rtc_info_t

struct rtc_info_t		
RTC Information Structure for information returned by infoGet()		
Data Fields		
rtc_clock_source_t	clock_source	Clock source for the RTC block.
rtc_status_t	status	RTC run status.

◆ rtc_cfg_t

struct rtc_cfg_t	
User configuration structure, used in open function	
Data Fields	
rtc_clock_source_t	clock_source
	Clock source for the RTC block.

uint32_t	freq_compare_value
	The frequency comparison value.
rtc_error_adjustment_cfg_t const *const	p_err_cfg
	Pointer to Error Adjustment configuration.
uint8_t	alarm_ipl
	Alarm interrupt priority.
IRQn_Type	alarm_irq
	Alarm interrupt vector.
uint8_t	periodic_ipl
	Periodic interrupt priority.
IRQn_Type	periodic_irq
	Periodic interrupt vector.
uint8_t	carry_ipl
	Carry interrupt priority.
IRQn_Type	carry_irq
	Carry interrupt vector.
void(*	p_callback)(rtc_callback_args_t *p_args)
	Called from the ISR.

void *	p_context
	User defined context passed into callback function.
void const *	p_extend
	RTC hardware dependant configuration.

◆ rtc_api_t

struct rtc_api_t	
RTC driver structure. General RTC functions implemented at the HAL layer follow this API.	
Data Fields	
fsp_err_t(*	open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rtc_ctrl_t *const p_ctrl)
fsp_err_t(*	clockSourceSet)(rtc_ctrl_t *const p_ctrl)
fsp_err_t(*	calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t(*	calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t(*	calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t(*	calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t(*	periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)
fsp_err_t(*	errorAdjustmentSet)(rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)
fsp_err_t(*	callbackSet)(rtc_ctrl_t *const p_ctrl,

	void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory)
fsp_err_t(*	infoGet)(rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)
fsp_err_t(*	timeCaptureSet)(rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)
fsp_err_t(*	timeCaptureGet)(rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)

Field Documentation

◆ open

fsp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)

Open the RTC driver.

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_cfg	Pointer to the configuration structure

◆ close

fsp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)

Close the RTC driver.

Parameters

[in]	p_ctrl	Pointer to RTC device handle.

◆ clockSourceSet

fsp_err_t(* rtc_api_t::clockSourceSet) (rtc_ctrl_t *const p_ctrl)

Sets the RTC clock source.

Parameters

[in]	p_ctrl	Pointer to RTC device handle

◆ calendarTimeSet

fsp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Set the calendar time and start the calendar counter

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_time	Pointer to a time structure that contains the time to set

◆ calendarTimeGet

fsp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Get the calendar time.

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_time	Pointer to a time structure that contains the time to get

◆ calendarAlarmSet

fsp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Set the calendar alarm time and enable the alarm interrupt.

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_alarm	Pointer to an alarm structure that contains the alarm time to set

◆ calendarAlarmGet

fsp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Get the calendar alarm time.

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_alarm	Pointer to an alarm structure to fill up with the alarm time

◆ periodicIrqRateSet

fsp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)

Set the periodic irq rate

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	rate	Rate of periodic interrupts

◆ errorAdjustmentSet

fsp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)

Set time error adjustment.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	err_adj_cfg	Pointer to the Error Adjustment Config

◆ callbackSet

fsp_err_t(* rtc_api_t::callbackSet) (rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Pointer to the RTC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated

◆ infoGet

fsp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)

Return the currently configure clock source for the RTC

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_rtc_info	Pointer to RTC information structure

◆ timeCaptureSet

fsp_err_t(* rtc_api_t::timeCaptureSet) (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)

Config Time capture

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_time_capture	Pointer to a time capture structure that contains the configuration

◆ timeCaptureGet

fsp_err_t(* rtc_api_t::timeCaptureGet) (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)

Get the capture time and clear bit status.

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_time_capture	Pointer to a time capture structure to fill up with the time capture

◆ rtc_instance_t

struct rtc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rtc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rtc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.

rtc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rtc_time_t

typedef struct tm rtc_time_t
Date and time structure defined in C standard library <time.h>

◆ rtc_ctrl_t

typedef void rtc_ctrl_t
RTC control block. Allocate an instance specific control block to pass into the RTC API calls.

Enumeration Type Documentation

◆ rtc_event_t

enum rtc_event_t	
Events that can trigger a callback function	
Enumerator	
RTC_EVENT_ALARM_IRQ	Real Time Clock ALARM 0 IRQ.
RTC_EVENT_ALARM1_IRQ	Real Time Clock ALARM 1 IRQ.
RTC_EVENT_PERIODIC_IRQ	Real Time Clock PERIODIC IRQ.

◆ rtc_alarm_channel_t

enum rtc_alarm_channel_t
RTC alarm channel

◆ rtc_clock_source_t

enum rtc_clock_source_t	
Clock source for the RTC block	
Enumerator	
RTC_CLOCK_SOURCE_SUBCLK	Sub-clock oscillator.
RTC_CLOCK_SOURCE_LOCO	Low power On Chip Oscillator.
RTC_CLOCK_SOURCE_MAINCLK	Main clock oscillator.

◆ rtc_status_t

enum rtc_status_t	
RTC run state	
Enumerator	
RTC_STATUS_STOPPED	RTC counter is stopped.
RTC_STATUS_RUNNING	RTC counter is running.

◆ rtc_error_adjustment_t

enum rtc_error_adjustment_t	
Time error adjustment settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_NONE	Adjustment is not performed.
RTC_ERROR_ADJUSTMENT_ADD_PRESCALER	Adjustment is performed by the addition to the prescaler.
RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER	Adjustment is performed by the subtraction from the prescaler.

◆ rtc_error_adjustment_mode_t

enum rtc_error_adjustment_mode_t	
Time error adjustment mode settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_MODE_MANUAL	Adjustment mode is set to manual.
RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC	Adjustment mode is set to automatic.

◆ rtc_error_adjustment_period_t

enum rtc_error_adjustment_period_t	
Time error adjustment period settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE	Adjustment period is set to every one minute.
RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND	Adjustment period is set to every ten second.
RTC_ERROR_ADJUSTMENT_PERIOD_NONE	Adjustment period not supported in manual mode.
RTC_ERROR_ADJUSTMENT_PERIOD_20_SECOND	Adjustment period is set to every twenty seconds.

◆ rtc_periodic_irq_select_t

enum rtc_periodic_irq_select_t	
Periodic Interrupt select	
Enumerator	
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND	A periodic irq is generated every 1/256 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND	A periodic irq is generated every 1/128 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND	A periodic irq is generated every 1/64 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND	A periodic irq is generated every 1/32 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND	A periodic irq is generated every 1/16 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND	A periodic irq is generated every 1/8 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND	A periodic irq is generated every 1/4 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND	A periodic irq is generated every 1/2 second.
RTC_PERIODIC_IRQ_SELECT_1_SECOND	A periodic irq is generated every 1 second.
RTC_PERIODIC_IRQ_SELECT_2_SECOND	A periodic irq is generated every 2 seconds.
RTC_PERIODIC_IRQ_SELECT_1_MINUTE	A periodic irq is generated every 1 minute.
RTC_PERIODIC_IRQ_SELECT_1_HOUR	A periodic irq is generated every 1 hour.
RTC_PERIODIC_IRQ_SELECT_1_DAY	A periodic irq is generated every 1 day.
RTC_PERIODIC_IRQ_SELECT_1_MONTH	A periodic irq is generated every 1 month.

◆ rtc_time_capture_source_t

enum rtc_time_capture_source_t	
Time capture trigger source	
Enumerator	
RTC_TIME_CAPTURE_SOURCE_DISABLED	Disable trigger.
RTC_TIME_CAPTURE_SOURCE_PIN_RISING	Rising edge pin trigger.
RTC_TIME_CAPTURE_SOURCE_PIN_FALLING	Falling edge pin trigger.
RTC_TIME_CAPTURE_SOURCE_PIN_BOTH	Both edges pin trigger.
RTC_TIME_CAPTURE_SOURCE_SOFTWARE	Software trigger.
RTC_TIME_CAPTURE_SOURCE_ELC_EVENT	ELC event trigger.

◆ rtc_time_capture_mode_t

enum rtc_time_capture_mode_t	
Time capture trigger mode	
Enumerator	
RTC_TIME_CAPTURE_MODE_CONTINUOUS	Continuous capturing to all capturing channels.
RTC_TIME_CAPTURE_MODE_ONE_SHOT	Single capture to a particular channel.

◆ rtc_time_capture_noise_filter_t

enum rtc_time_capture_noise_filter_t	
Time capture noise filter control	
Enumerator	
RTC_TIME_CAPTURE_NOISE_FILTER_OFF	Turn noise filter off.
RTC_TIME_CAPTURE_NOISE_FILTER_ON	Turn noise filter on (count source)
RTC_TIME_CAPTURE_NOISE_FILTER_ON_DIVIDER_32	Turn noise filter on (count source by divided by 32)
RTC_TIME_CAPTURE_NOISE_FILTER_ON_DIVIDER_4096	Turn noise filter on (count source by divided by 4096)
RTC_TIME_CAPTURE_NOISE_FILTER_ON_DIVIDER_8192	Turn noise filter on (count source by divided by 8192)

5.3.17.3 Three-Phase Interface

Interfaces » Timers

Detailed Description

Interface for three-phase timer functions.

Summary

The Three-Phase interface provides functionality for synchronous start/stop/reset control of three timer channels for use in 3-phase motor control applications.

Data Structures

struct	three_phase_duty_cycle_t
struct	three_phase_cfg_t
struct	three_phase_api_t
struct	three_phase_instance_t

Typedefs

typedef void	three_phase_ctrl_t

Enumerations

enum	three_phase_channel_t
enum	three_phase_buffer_mode_t

Data Structure Documentation

◆ three_phase_duty_cycle_t

struct three_phase_duty_cycle_t		
Struct for passing duty cycle values to three_phase_api_t::dutyCycleSet		
Data Fields		
uint32_t	duty[3]	Duty cycle. Note: When the GPT instances are configured in TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3, this value sets the duty cycle count that is transferred to GTCCRA/B at the trough.
uint32_t	duty_buffer[3]	Double-buffer for duty cycle values. Note: When the GPT instances are configured in TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3, this value sets the duty cycle count that is transferred to GTCCRA/B at the crest.

◆ three_phase_cfg_t

struct three_phase_cfg_t		
User configuration structure, used in open function		
Data Fields		
three_phase_buffer_mode_t	buffer_mode	Single or double-buffer mode.
timer_instance_t const *	p_timer_instance[3]	Pointer to the timer instance structs.
three_phase_channel_t	callback_ch	Channel to enable callback when using three_phase_api_t::callbackSet.
uint32_t	channel_mask	Bitmask of timer channels used by this module.
void *	p_context	Placeholder for user data. Passed to the user callback in timer_callback_args_t.
void const *	p_extend	Extension parameter for hardware specific settings.

◆ three_phase_api_t

struct three_phase_api_t

Three-Phase API structure.

Data Fields

fsp_err_t(*	open)(three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)
fsp_err_t(*	start)(three_phase_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(three_phase_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(three_phase_ctrl_t *const p_ctrl)
fsp_err_t(*	dutyCycleSet)(three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)
fsp_err_t(*	callbackSet)(three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(three_phase_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* three_phase_api_t::open) (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ start

fsp_err_t(* three_phase_api_t::start) (three_phase_ctrl_t *const p_ctrl)

Start all three timers synchronously.

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ stop

fsp_err_t(* three_phase_api_t::stop) (three_phase_ctrl_t *const p_ctrl)

Stop all three timers synchronously.

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ reset

fsp_err_t(* three_phase_api_t::reset) (three_phase_ctrl_t *const p_ctrl)

Reset all three timers synchronously.

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ dutyCycleSet

fsp_err_t(* three_phase_api_t::dutyCycleSet) (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)

Sets the duty cycle match values. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.
[in]	p_duty_cycle	Duty cycle values for all three timer channels.

◆ callbackSet

fsp_err_t(* three_phase_api_t::callbackSet) (three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call.
[in]	p_callback	Callback function to register with GPT U-channel
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* three_phase_api_t::close) (three_phase_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ three_phase_instance_t

struct three_phase_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
three_phase_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
three_phase_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
three_phase_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ three_phase_ctrl_t

typedef void three_phase_ctrl_t
Three-Phase control block. Allocate an instance specific control block to pass into the timer API calls.

Enumeration Type Documentation

◆ three_phase_channel_t

enum three_phase_channel_t	
Timer channel indices	
Enumerator	
THREE_PHASE_CHANNEL_U	U-channel index.
THREE_PHASE_CHANNEL_V	V-channel index.
THREE_PHASE_CHANNEL_W	W-channel index.

◆ three_phase_buffer_mode_t

enum three_phase_buffer_mode_t	
Buffering mode	
Enumerator	
THREE_PHASE_BUFFER_MODE_SINGLE	Single-buffer mode.
THREE_PHASE_BUFFER_MODE_DOUBLE	Double-buffer mode.

5.3.17.4 Timer Interface

Interfaces » Timers

Detailed Description

Interface for timer functions.

Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, PWM output, and free-running timer mode. After each timer cycle (overflow or underflow), an

interrupt can be triggered.

If an instance supports output compare mode, it is provided in the extension configuration timer_on_<instance>_cfg_t defined in r_<instance>.h.

Data Structures

struct	timer_callback_args_t
struct	timer_info_t
struct	timer_status_t
struct	timer_cfg_t
struct	timer_api_t
struct	timer_instance_t

Typedefs

typedef void	timer_ctrl_t

Enumerations

enum	timer_event_t
enum	timer_variant_t
enum	timer_compare_match_t
enum	timer_state_t
enum	timer_mode_t
enum	timer_direction_t
enum	timer_source_div_t

Data Structure Documentation

◆ timer_callback_args_t

struct timer_callback_args_t		
Callback function parameter data		
Data Fields		
void *	p_context	Placeholder for user data. Set in timer_api_t::open function in timer_cfg_t.

timer_event_t	event	The event can be used to identify what caused the callback.
uint32_t	capture	Most recent capture, only valid if event is TIMER_EVENT_CAPTURE_A or TIMER_EVENT_CAPTURE_B.

◆ timer_info_t

struct timer_info_t		
Timer information structure to store various information for a timer resource		
Data Fields		
timer_direction_t	count_direction	Clock counting direction of the timer.
uint32_t	clock_frequency	Clock frequency of the timer counter.
uint32_t	period_counts	Period in raw timer counts. *Note* *For triangle wave PWM modes, the full period is double this value.*

◆ timer_status_t

struct timer_status_t		
Current timer status.		
Data Fields		
uint32_t	counter	Current counter value.
timer_state_t	state	Current timer state (running or stopped)

◆ timer_cfg_t

struct timer_cfg_t	
User configuration structure, used in open function	
Data Fields	
timer_mode_t	mode
	Select enumerated value from timer_mode_t.
uint32_t	period_counts
	Period in raw timer counts.

timer_source_div_t	source_div
	Source clock divider.
uint32_t	duty_cycle_counts
	Duty cycle in counts.
uint8_t	channel
uint8_t	cycle_end_ipl
	Cycle end interrupt priority.
IRQn_Type	cycle_end_irq
	Cycle end interrupt.
void(*	p_callback)(timer_callback_args_t *p_args)
void *	p_context
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ channel

uint8_t timer_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ p_callback

void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)

Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void* timer_cfg_t::p_context

Placeholder for user data. Passed to the user callback in timer_callback_args_t.

◆ timer_api_t

struct timer_api_t

Timer API structure. General timer functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(*	open)(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t(*	start)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	enable)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	periodSet)(timer_ctrl_t *const p_ctrl, uint32_t const period)
fsp_err_t(*	dutyCycleSet)(timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t(*	compareMatchSet)(timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel)
fsp_err_t(*	infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t(*	statusGet)(timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t(*	callbackSet)(timer_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)

fsp_err_t(*	close)(timer_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ start

fsp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)

Start the counter.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ stop

fsp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)

Stop the counter.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ reset

fsp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)

Reset the counter to the initial value.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ enable

fsp_err_t(* timer_api_t::enable) (timer_ctrl_t *const p_ctrl)

Enables input capture.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ disable

fsp_err_t(* timer_api_t::disable) (timer_ctrl_t *const p_ctrl)

Disables input capture.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ periodSet

fsp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, uint32_t const period)

Set the time until the timer expires. See implementation for details of period update timing.

Note

Timer expiration may or may not generate a CPU interrupt based on how the timer is configured in timer_api_t::open.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	period	Time until timer should expire.

◆ dutyCycleSet

fsp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)

Sets the number of counts for the pin level to be high. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	duty_cycle_counts	Time until duty cycle should expire.
[in]	pin	Which output pin to update. See implementation for details.

◆ compareMatchSet

fsp_err_t(* timer_api_t::compareMatchSet) (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel)

Set a compare match value in raw counts.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	compare_match_value	Timer value to trigger a compare match event.
[in]	match_channel	Which channel to update.

◆ infoGet

fsp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

Stores timer information in p_info.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_info	Collection of information for this timer.

◆ statusGet

fsp_err_t(* timer_api_t::statusGet) (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)

Get the current counter value and timer state and store it in p_status.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_status	Current status of this timer.

◆ callbackSet

fsp_err_t(* timer_api_t::callbackSet) (timer_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ timer_instance_t

struct timer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
timer_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
timer_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
timer_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ timer_ctrl_t

typedef void timer_ctrl_t
Timer control block. Allocate an instance specific control block to pass into the timer API calls.

Enumeration Type Documentation

◆ timer_event_t

enum timer_event_t	
Events that can trigger a callback function	
Enumerator	
TIMER_EVENT_CYCLE_END	Requested timer delay has expired or timer has wrapped around.
TIMER_EVENT_CREST	Timer crest event (counter is at a maximum, triangle-wave PWM only)
TIMER_EVENT_CAPTURE_A	A capture has occurred on signal A.
TIMER_EVENT_CAPTURE_B	A capture has occurred on signal B.
TIMER_EVENT_TROUGH	Timer trough event (counter is 0, triangle-wave PWM only.
TIMER_EVENT_COMPARE_A	A compare has occurred on signal A.
TIMER_EVENT_COMPARE_B	A compare has occurred on signal B.
TIMER_EVENT_COMPARE_C	A compare has occurred on signal C.
TIMER_EVENT_COMPARE_D	A compare has occurred on signal D.
TIMER_EVENT_COMPARE_E	A compare has occurred on signal E.
TIMER_EVENT_COMPARE_F	A compare has occurred on signal F.
TIMER_EVENT_DEAD_TIME	Dead time event.

◆ timer_variant_t

enum timer_variant_t	
Timer variant types.	
Enumerator	
TIMER_VARIANT_32_BIT	32-bit timer
TIMER_VARIANT_16_BIT	16-bit timer

◆ timer_compare_match_t

enum timer_compare_match_t	
Options for storing compare match value	
Enumerator	
TIMER_COMPARE_MATCH_A	Compare match A value.
TIMER_COMPARE_MATCH_B	Compare match B value.
TIMER_COMPARE_MATCH_C	Compare match C value.
TIMER_COMPARE_MATCH_D	Compare match D value.
TIMER_COMPARE_MATCH_E	Compare match E value.
TIMER_COMPARE_MATCH_F	Compare match F value.
TIMER_COMPARE_MATCH_G	Compare match G value.
TIMER_COMPARE_MATCH_H	Compare match H value.

◆ timer_state_t

enum timer_state_t	
Possible status values returned by timer_api_t::statusGet.	
Enumerator	
TIMER_STATE_STOPPED	Timer is stopped.
TIMER_STATE_COUNTING	Timer is running.
TIMER_STATE_UNKNOWN	Timer state could not be defined.

◆ timer_mode_t

enum timer_mode_t	
Timer operational modes	
Enumerator	
TIMER_MODE_PERIODIC	Timer restarts after period elapses.
TIMER_MODE_ONE_SHOT	Timer stops after period elapses.
TIMER_MODE_PWM	Timer generates saw-wave PWM output.
TIMER_MODE_ONE_SHOT_PULSE	Saw-wave one-shot pulse mode (fixed buffer operation).
TIMER_MODE_TRIANGLE_WAVE_SYMMETRIC_PWM	Timer generates symmetric triangle-wave PWM output.
TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM	Timer generates asymmetric triangle-wave PWM output.
TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3	Timer generates Asymmetric Triangle-wave PWM output. In PWM mode 3, the duty cycle does not need to be updated at each tough/crest interrupt. Instead, the trough and crest duty cycle values can be set once and only need to be updated when the application needs to change the duty cycle.

◆ timer_direction_t

enum timer_direction_t	
Direction of timer count	
Enumerator	
TIMER_DIRECTION_DOWN	Timer count goes up.
TIMER_DIRECTION_UP	Timer count goes down.

◆ timer_source_div_t

enum timer_source_div_t	
Clock source divisors	
Enumerator	
TIMER_SOURCE_DIV_1	Timer clock source divided by 1.
TIMER_SOURCE_DIV_2	Timer clock source divided by 2.
TIMER_SOURCE_DIV_4	Timer clock source divided by 4.
TIMER_SOURCE_DIV_8	Timer clock source divided by 8.
TIMER_SOURCE_DIV_16	Timer clock source divided by 16.
TIMER_SOURCE_DIV_32	Timer clock source divided by 32.
TIMER_SOURCE_DIV_64	Timer clock source divided by 64.
TIMER_SOURCE_DIV_128	Timer clock source divided by 128.
TIMER_SOURCE_DIV_256	Timer clock source divided by 256.
TIMER_SOURCE_DIV_512	Timer clock source divided by 512.
TIMER_SOURCE_DIV_1024	Timer clock source divided by 1024.
TIMER_SOURCE_DIV_8192	Timer clock source divided by 8192.

5.3.18 Transfer

Interfaces

Detailed Description

Transfer Interfaces.

Modules

Transfer Interface	Interface for data transfer functions.

5.3.18.1 Transfer Interface

Interfaces » Transfer

Detailed Description

Interface for data transfer functions.

Summary

The transfer interface supports background data transfer (no CPU intervention).

Data Structures

struct	transfer_callback_args_t
struct	transfer_properties_t
struct	transfer_info_t
struct	transfer_cfg_t
struct	transfer_api_t
struct	transfer_instance_t

Typedefs

typedef void	transfer_ctrl_t

Enumerations

enum	transfer_mode_t
enum	transfer_size_t
enum	transfer_addr_mode_t
enum	transfer_repeat_area_t
enum	transfer_chain_mode_t
enum	transfer_irq_t
enum	transfer_start_mode_t

Data Structure Documentation

◆ transfer_callback_args_t

struct transfer_callback_args_t		
Callback function parameter data.		
Data Fields		
void *	p_context	Placeholder for user data. Set in transfer_api_t::open function in transfer_cfg_t.

◆ transfer_properties_t

struct transfer_properties_t		
Driver specific information.		
Data Fields		
uint32_t	block_count_max	Maximum number of blocks.
uint32_t	block_count_remaining	Number of blocks remaining.
uint32_t	transfer_length_max	Maximum number of transfers.
uint32_t	transfer_length_remaining	Number of transfers remaining.

◆ transfer_info_t

struct transfer_info_t		
This structure specifies the properties of the transfer. **Warning** When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: p_src, p_dest, num_blocks, and length. When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique transfer_info_t. When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed. ***Note*** *When using DTC, consider placing instances of this structure in a protected section of memory.*		
Data Fields		
union transfer_info_t	transfer_settings_u	
void const *volatile	p_src	Source pointer.
void *volatile	p_dest	Destination pointer.
volatile uint16_t	num_blocks	Number of blocks to transfer when using TRANSFER_MODE_BLOCK (DTC, DMAC and EXDMAC) or TRANSFER_MODE_REPEAT (DMAC and EXDMAC only) or TRANSFER_MODE_REPEAT_BLOCK (DMAC and EXDMAC only),

		unused in other modes. Number of blocks to transfer when using TRANSFER_MODE_BLOCK (DTC, DMAC or EXDMAC) or TRANSFER_MODE_REPEAT (DMAC and EXDMAC only) or TRANSFER_MODE_REPEAT_BLOCK (DMAC and EXDMAC only), unused in other modes.
volatile uint16_t	length	Length of each transfer. Range limited for TRANSFER_MODE_BLOCK, TRANSFER_MODE_REPEAT, and TRANSFER_MODE_REPEAT_BLOCK see HAL driver for details.
union transfer_info_t	transfer_settings_u	

◆ transfer_cfg_t

struct transfer_cfg_t		
Driver configuration set in transfer_api_t::open. All elements except p_extend are required and must be initialized.		
Data Fields		
transfer_info_t *	p_info	Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.
void const *	p_extend	Extension parameter for hardware specific settings.

◆ transfer_api_t

struct transfer_api_t	
Transfer functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t(*	reconfigure)(transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)
fsp_err_t(*	reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)

fsp_err_t(*	enable)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	softwareStart)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)
fsp_err_t(*	softwareStop)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)
fsp_err_t(*	close)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	reload)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers)
fsp_err_t(*	callbackSet)(transfer_ctrl_t *const p_ctrl, void(*p_callback)(transfer_callback_args_t *), void *const p_context, transfer_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)

Initial configuration.

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ reconfigure

fsp_err_t(* transfer_api_t::reconfigure) (transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)

Reconfigure the transfer. Enable the transfer if p_info is valid.

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_info	Pointer to a new transfer info structure.

◆ reset

fsp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)

Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_src	Pointer to source. Set to NULL if source pointer should not change.
[in]	p_dest	Pointer to destination. Set to NULL if destination pointer should not change.
[in]	num_transfers	Transfer length in normal mode or number of blocks in block mode. In DMAC and EXDMAC only, resets number of repeats (initially stored in transfer_info_t::num_blocks) in repeat mode. Not used in repeat mode for DTC.

◆ enable

fsp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)

Enable transfer. Transfers occur after the activation source event (or when transfer_api_t::softwareStart is called if no peripheral event is chosen as activation source).

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ disable

fsp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)

Disable transfer. Transfers do not occur after the activation source event (or when transfer_api_t::softwareStart is called if no peripheral event is chosen as the DMAC or EXDMAC activation source).

Note

If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ softwareStart

fsp_err_t(* transfer_api_t::softwareStart) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)

Start transfer in software.

Warning

Only works if no peripheral event is chosen as the DMAC or EXDMAC activation source.

Note

Not supported for DTC.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	mode	Select mode from transfer_start_mode_t.

◆ softwareStop

fsp_err_t(* transfer_api_t::softwareStop) (transfer_ctrl_t *const p_ctrl)

Stop transfer in software. The transfer will stop after completion of the current transfer.

Note

Not supported for DTC.
Only applies for transfers started with TRANSFER_START_MODE_REPEAT.

Warning

Only works if no peripheral event is chosen as the DMAC or EXDMAC activation source.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ infoGet

fsp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)

Provides information about this transfer.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[out]	p_properties	Driver specific information.

◆ close

fsp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)

Releases hardware lock. This allows a transfer to be reconfigured using transfer_api_t::open.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ reload

fsp_err_t(* transfer_api_t::reload) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers)

To update next transfer information without interruption during transfer. Allow further transfer continuation.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_src	Pointer to source. Set to NULL if source pointer should not change.
[in]	p_dest	Pointer to destination. Set to NULL if destination pointer should not change.
[in]	num_transfers	Transfer length in normal mode or block mode.

◆ callbackSet

fsp_err_t(* transfer_api_t::callbackSet) (transfer_ctrl_t *const p_ctrl, void(*p_callback)(transfer_callback_args_t *), void *const p_context, transfer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_callback_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ transfer_instance_t

struct transfer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
transfer_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
transfer_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
transfer_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ transfer_ctrl_t

typedef void transfer_ctrl_t
Transfer control block. Allocate an instance specific control block to pass into the transfer API calls.

Enumeration Type Documentation

◆ transfer_mode_t

enum transfer_mode_t	
Transfer mode describes what will happen when a transfer request occurs.	
Enumerator	
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC or EXDMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are

	updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT_BLOCK	In addition to block mode features, repeat-block mode supports a ring buffer of blocks and offsets within a block (to split blocks into arrays of their first data, second data, etc.)
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC or EXDMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT_BLOCK	In addition to block mode features, repeat-block mode supports a ring buffer of blocks and offsets within a block (to split blocks into arrays of their first data, second data, etc.)

◆ transfer_size_t

enum transfer_size_t	
Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t	
Enumerator	
TRANSFER_SIZE_1_BYTE	Each transfer transfers a 8-bit value.
TRANSFER_SIZE_2_BYTE	Each transfer transfers a 16-bit value.
TRANSFER_SIZE_4_BYTE	Each transfer transfers a 32-bit value.
TRANSFER_SIZE_8_BYTE	Each transfer transfers a 64-bit value.
TRANSFER_SIZE_1_BYTE	Each transfer transfers a 8-bit value.
TRANSFER_SIZE_2_BYTE	Each transfer transfers a 16-bit value.
TRANSFER_SIZE_4_BYTE	Each transfer transfers a 32-bit value.
TRANSFER_SIZE_8_BYTE	Each transfer transfers a 64-bit value.

◆ transfer_addr_mode_t

enum transfer_addr_mode_t	
Address mode specifies whether to modify (increment or decrement) pointer after each transfer.	
Enumerator	
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Offset is added to the address pointer after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Offset is added to the address pointer after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.

◆ transfer_repeat_area_t

enum transfer_repeat_area_t	
Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT_BLOCK, the selected pointer returns to its original value after each transfer.	
Enumerator	
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.

◆ transfer_chain_mode_t

enum transfer_chain_mode_t	
Chain transfer mode options. ***Note*** *Only applies for DTC.*	
Enumerator	
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.

◆ transfer_irq_t

enum transfer_irq_t	
Interrupt options.	
Enumerator	
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. *Note* *Not available in all HAL drivers. See HAL driver for details.*
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. *Note* *Not available in all HAL drivers. See HAL driver for details.*

◆ transfer_start_mode_t

enum transfer_start_mode_t	
Select whether to start single or repeated transfer with software start.	
Enumerator	
TRANSFER_START_MODE_SINGLE	Software start triggers single transfer.
TRANSFER_START_MODE_REPEAT	Software start transfer continues until transfer is complete.

Chapter 6 Copyright

© Renesas Electronics Corporation. All rights reserved.

Renesas FSP
Copyright © (2026) Renesas Electronics Corporation. All Rights Reserved.

Software Manual

Publication Date: v1.0.0 Sept.29.2026

Renesas FSP v1.0.0

Software Manual

Renesas Electronics Corporation