RL78 Family

Reality AI Control Modules Software Integration System

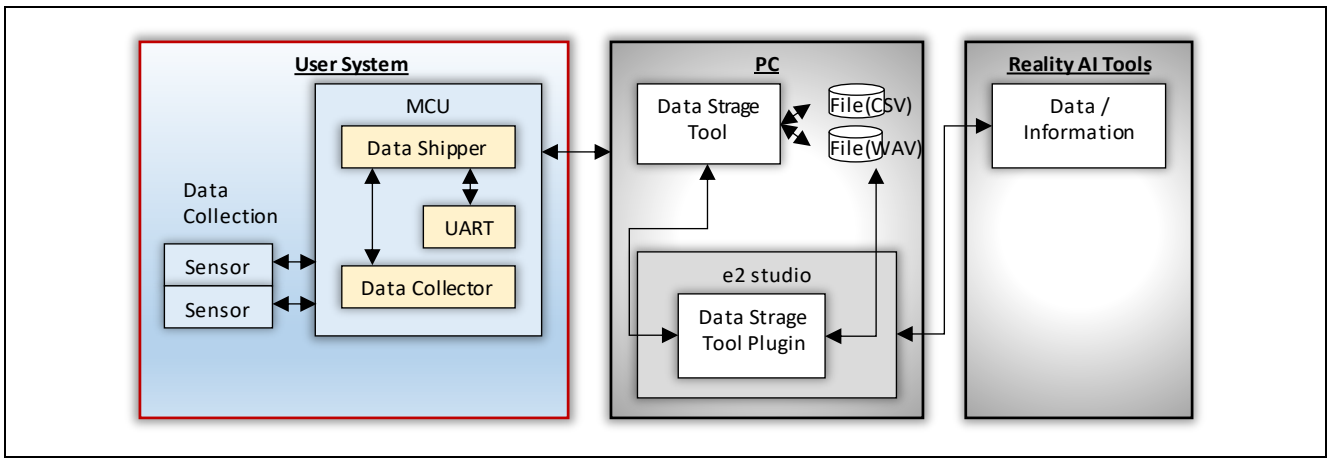
Introduction

This application note explains Data Shipper and Data Collector control modules for Renesas Reality AI, and general UART communication module using Software Integration System (SIS).

These control modules can obtain the sensing data using A/D converter etc. and transmit the data to PC.

Hereinafter, the modules described in this application note is abbreviated as follows,

- The Data Shipper control module for Renesas Reality AI: Data Shipper SIS module
- The Data Collector control module for Renesas Reality AI: Data Collector SIS module
- The general UART Communication module: COMMS UART SIS module



Target Device

- **RL78 Family MCUs:**
  MCUs supported the following modules:
  - Timer Array Unit (TAU) Interval Timer Module (Code Generator Module)
  - Data Transfer Controller Module (Code Generator Module)
  - UART Communication Module (Code Generator Module)

- **Operation confirmed MCU:**
  - RL78/G23

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compiler

- Renesas Electronics C/C++ Compiler Package for RL78 Family

Reference Documents

- Smart Configurator User’s Manual: RL78 API Reference (R20UT4852)
- RL78/G23 User’s Manual: The latest version can be downloaded from the Renesas Electronics website.
- RL78/G14 User’s Manual: The latest version can be downloaded from the Renesas Electronics website.
- Technical Update/Technical News
  The latest information can be downloaded from the Renesas Electronics website.
- RL78 Family Compiler CC-RL User's Manual (R20UT3123)
  The latest versions can be downloaded from the Renesas Electronics website.

Contents

1. Overview of Data Shipper and Data Collector Control Modules for Renesas Reality AI 5
1.1 Data Shipper Module 6
1.1.1 Overview 6
1.1.2 Features 6
1.1.3 Outline of Data Shipper SIS Module 6
1.2 Data Collector Module 6
1.2.1 Overview 6
1.2.2 Features 7
1.2.3 Outline of Data Collector SIS Module 8
1.3 UART Communication Module 8
1.3.1 Overview 8
1.3.2 Features 8
1.3.3 Outline of COMMS UART SIS Module 9
1.4 Operating Test Environment 9
1.5 Notes/Restrictions 9

2. API Information 10
2.1 Hardware Requirements 10
2.2 Software Requirements 10
2.3 Supported Toolchains 10
2.4 Usage of Interrupt Vector 10
2.5 Header Files 10
2.6 Integer Types 11
2.7 Configuration Overview 11
2.7.1 Data Shipper SIS Module Configuration (rm_rai_data_shipper_rl_config.h) 11
2.7.2 Data Collector SIS Module Configuration (rm_rai_data_collector_rl_config.h) 11
2.7.3 COMMS UART SIS Module Configuration (r_comms_uart_rl_config.h) 13
2.8 Code Size 15
2.9 Parameters 16
2.9.1 Configuration Structure and Control Structure of Data Shipper SIS Module 16
2.9.2 Configuration Structure and Control Structure of Data Collector SIS Module 17
2.9.3 Configuration Structure and Control Structure of COMMS UART SIS Module 18
2.10 Return Values 19
2.11 Adding the SIS Module to Your Project 20

3. Data Shipper API Functions 21
3.1 RM_RAI_DATA_SHIPPER_Open() 21
3.2 RM_RAI_DATA_SHIPPER_Close() 22
3.3 RM_RAI_DATA_SHIPPER_Read() 23
3.4 RM_RAI_DATA_SHIPPER_Write() 24

3.5 rai_data_shipper_write_callback() 25

4. Data Collector API Functions 26
4.1 RM_RAI_DATA_COLLECTOR_Open() 26
4.2 RM_RAI_DATA_COLLECTOR_Close() 27
4.3 RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister() 28
4.4 RM_RAI_DATA_COLLECTOR_BufferReset() 29
4.5 RM_RAI_DATA_COLLECTOR_BufferRelease() 30
4.6 RM_RAI_DATA_COLLECTOR_ChannelBufferGet() 31
4.7 RM_RAI_DATA_COLLECTOR_ChannelWrite() 32
4.8 RM_RAI_DATA_COLLECTOR_SnapshotStart() 33
4.9 RM_RAI_DATA_COLLECTOR_SnapshotStop() 34

5. UART Communication API Functions 35
5.1 RM_COMMS_UART_Open() 35
5.2 RM_COMMS_UART_Close() 36
5.3 RM_COMMS_UART_CallbackSet() 37
5.4 RM_COMMS_UART_Read() 38
5.5 RM_COMMS_UART_Write() 39
5.6 RM_COMMS_UART_WriteRead() 40
5.7 rm_comms_uart_callback 41

1. Overview of Data Shipper and Data Collector Control Modules for Renesas Reality AI

The Data Shipper and Data Collector control modules described in this application note are included in a hardware abstraction layer for Renesas Reality AI.

The software architecture of Renesas Reality AI hardware abstraction layer is shown below “Figure 1-1 User System Software Architecture”.

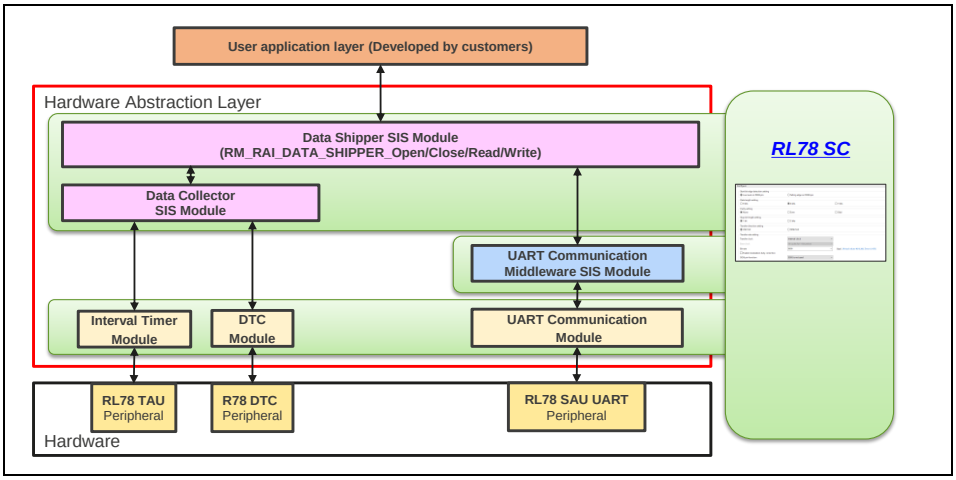


Figure 1-1 User System Software Architecture for Renesas Reality AI

The hardware abstraction layer has three layers, “Data Shipper APIs and Data Collector APIs”, “UART Communication Middleware” and SIS modules/Code Generator modules for RL78 peripherals.

The Data Shipper APIs and Data Collector APIs are provided as “Data Shipper SIS module”, “Data Collector SIS module” and the UART Communication SIS module is provided as “COMMS UART SIS module”.

The “Data Shipper SIS module” provides a method to send sensing data obtained by the “Data Collector SIS module” over UART using “COMMS UART SIS module”.

Table 1-1 shows the peripheral modules used.

Table 1-1 Peripheral Modules used

Peripheral Modules	Reference Application Notes
Timer Array Unit (TAU) Interval Timer Module (Code Generator Module)	Smart Configurator User’s Manual: RL78 API Reference
Data Transfer Controller (Code Generator Module)	
UART Communication Module (Code Generator Module)	

1.1 Data Shipper Module

1.1.1 Overview

Data Shipper usage is relatively straightforward with all communications being fully asynchronous. It utilizes the Communications Module Interface. The data being transported may be any combination of the following:

- Sensor data
- System events/errors
- Debug data and diagnostic information, including Reality AI (RAI) runtime output

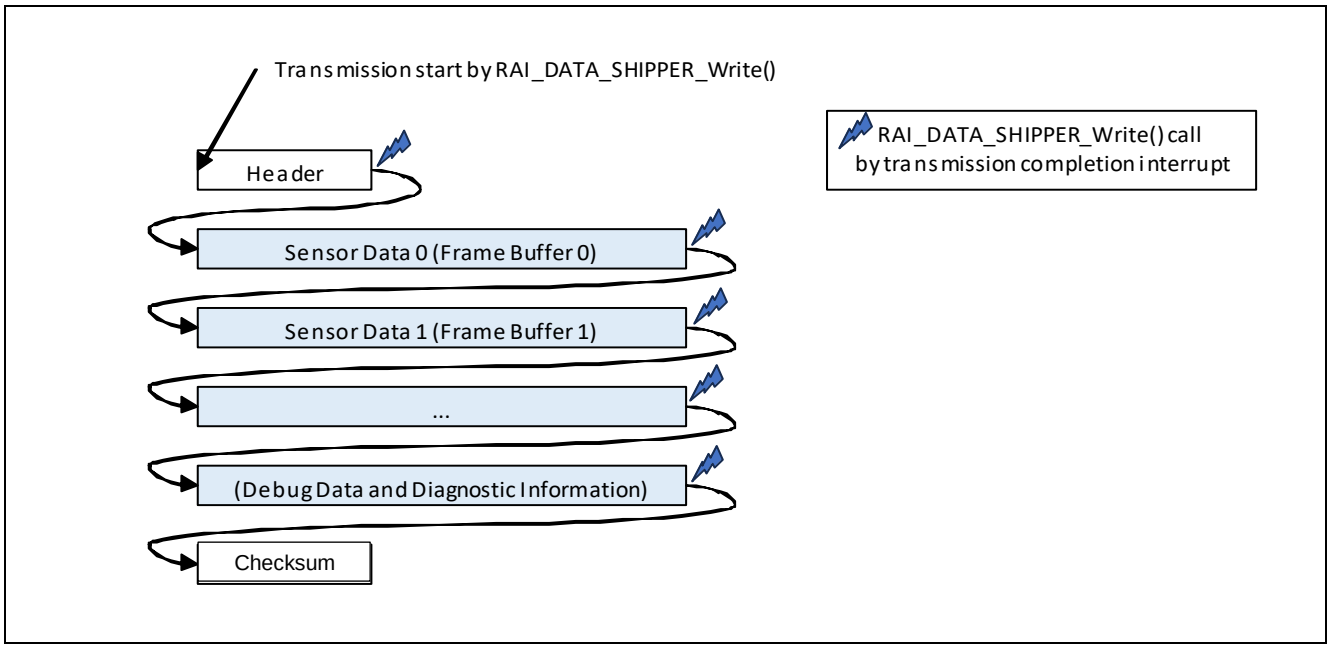


Figure 1-2 Data Transmission Using RM_RAI_DATA_SHIPPER_Write() Function

1.1.2 Features

The Data Shipper supports following interfaces:

- Multiple Data Collector instances
- Various interfaces, e.g. UART Communication Device (rm_comms_uart)

1.1.3 Outline of Data Shipper SIS Module

Table 1-2 lists the Data Shipper SIS module API functions.

Table 1-2 Data Shipper SIS Module API Functions

Function	Description
RM_RAI_DATA_SHIPPER_Open()	Opens and configures the Data Shipper module.
RM_RAI_DATA_SHIPPER_Close()	Closes the Data Shipper module instance.
RM_RAI_DATA_SHIPPER_Read()	Reads data.
RM_RAI_DATA_SHIPPER_Write()	Sends the data over UART bus.

1.2 Data Collector Module

1.2.1 Overview

Data Collector is to abstract the collection of data from sensors so that samples are collected and accumulated into fixed length frames before being made available to upper modules/application. Support of "snapshot" mode and "data feed" mode are required to accommodate for background and cooperative data collection. Each sensor will be captured into a separate frame buffer. Frame buffers are allocated by users, and they shall have the same amount of data samples (int32_t, float, uint8_t etc.). User application has to

make sure that each frame buffer will be filled up at the same rate. PING-PONG buffer is required for seamless operation.

When all frame buffers are filled up, they will be provided to the upper modules/application via data ready callback. When they are consumed, upper module has to release them. Ideally buffers will be released before the other set of buffers are filled up. However, it is possible that frame buffers will overrun due to the fact that upper module may take longer time to process the data in some cases. If it happens, application will be notified via the error callback. No intervention is required from user side in this case. Buffer overrun will go away when frame buffers are released. However, if there is a buffer-out-of-sync error, users need to find out whether all sensors work at the same pace.

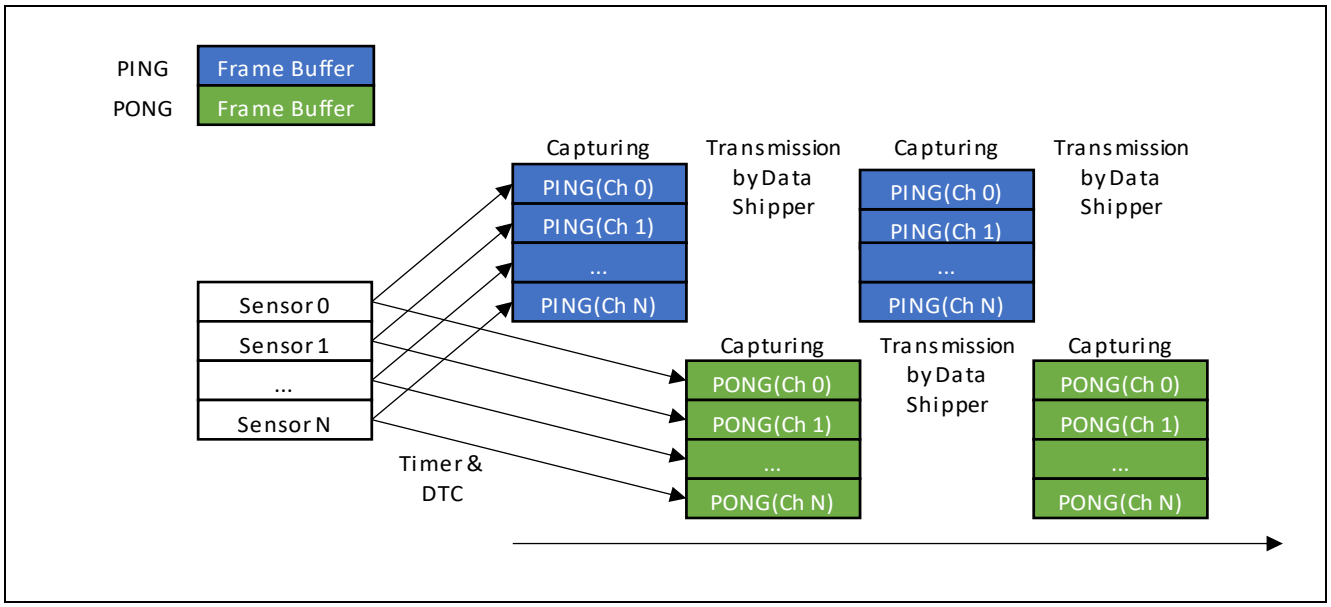


Figure 1-3 Frame Buffers and Data Capturing Using Snapshot Mode

1.2.2 Features

- Snapshot mode and data feed mode are supported
- Maximumly 8 sensors are supported for each mode
- Mix mode is supported (Snapshot mode and data feed mode work simultaneously)

(1) Snapshot Mode

Snapshot mode will periodically pull data from the user-specified places and save to designated frame buffers in the background. Renesas MCUs are well equipped to support this mode. DTC with its chain mode enables data collection from various, potentially non-linear and different-sized sources, while the Timer Array Unit (TAU) provides periodic interrupt that can be used as the activation source of DTC. When sensor source addresses are registered, application needs to start the timer so that DTC will start data collection.

(2) Data Feed Mode

Data feed mode will require data producer to push data directly to the designated frame buffer whenever data is ready. Data can be pushed synchronously or asynchronously. Synchronous mode is for the use case that the data producer has a short amount of data to be copied to the frame buffer. Asynchronous mode is for dmac/dtc transfer. Application is responsible to initialize dmac/dtc transfer descriptors.

Note when work in mixed mode, users must take care to make sure snapshot mode channels and data mode channels work at the same pace.

1.2.3 Outline of Data Collector SIS Module

Table 1-3 lists the Data Collector SIS module API functions.

Table 1-3 Data Collector SIS Module API Functions

Function	Description
RM_RAI_DATA_COLLECTOR_Open()	Opens and configures the Data Collector module.
RM_RAI_DATA_COLLECTOR_Close()	Closes Data Collector module instance.
RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister()	Configures transfer source address for snapshot mode channel.
RM_RAI_DATA_COLLECTOR_BufferReset()	Resets to discard accumulated data and start with PING buffer.
RM_RAI_DATA_COLLECTOR_BufferRelease()	Releases frame buffer.
RM_RAI_DATA_COLLECTOR_ChannelBufferGet()	Gets channel destination buffer address for asynchronous data transfer.
RM_RAI_DATA_COLLECTOR_ChannelWrite()	Synchronous data transfer using CPU copy.
RM_RAI_DATA_COLLECTOR_SnapshotStart()	Starts snapshot mode channels.
RM_RAI_DATA_COLLECTOR_SnapshotStop()	Stops snapshot mode channels.

1.3 UART Communication Module

1.3.1 Overview

The UART Communication module implements COMMS API for UART interface.

User has to associate the bus with the device as shown in the following diagram.

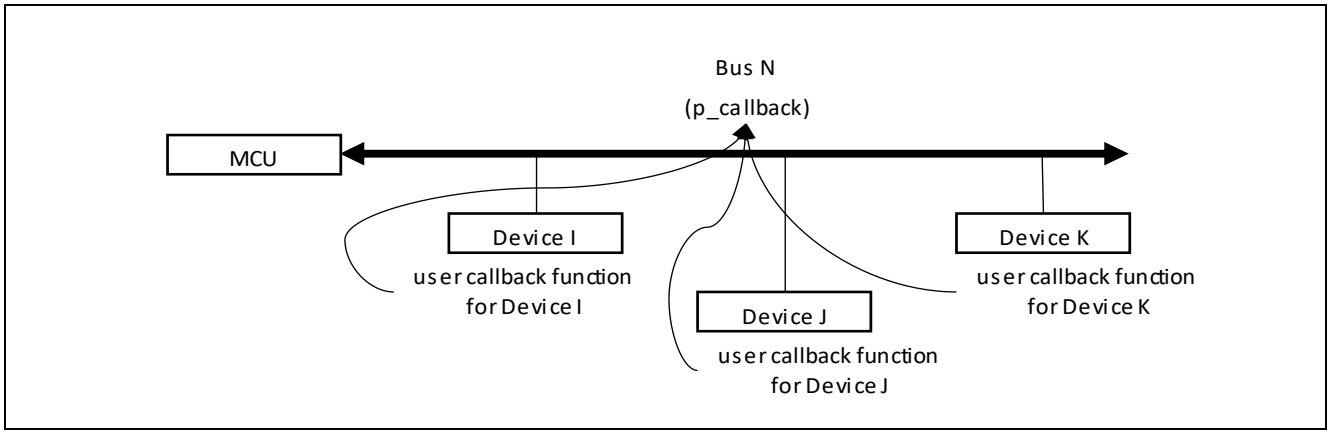


Figure 1-4 Diagram of Bus and Devices of COMMS Communication Module

1.3.2 Features

The implementation of the UART communications interface has the following key features:

- Non-blocking API for bare metal

1.3.3 Outline of COMMS UART SIS Module

Table 1-4 lists the COMMS UART SIS module API functions.

Table 1-4 COMMS UART SIS Module API Functions

Function	Description
RM_COMMS_UART_Open()	Opens and configures the Communications module.
RM_COMMS_UART_Close()	Disables the specified Communications module.
RM_COMMS_UART_CallbackSet()	Updates the user callback.
RM_COMMS_UART_Read()	Performs a read from the UART device.
RM_COMMS_UART_Write()	Performs a write to the UART device.
RM_COMMS_UART_WriteRead()	Performs a write to, then a read from the UART device.

1.4 Operating Test Environment

This section describes for detailed the operating test environments of these SIS modules.

Table 1-5 Operation Test Environment

Item	Contents
Integrated Development Environment	Renesas Electronics e2 studio 2023-07
C Compiler	Renesas Electronics C/C++ compiler for RL78 family V1.12.00 Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99 -far_rom (configurable)
Endian Order	Little-endian
Module Version	Board Support Package Module (r_bsp) Ver.1.60 Interval Timer Module (Code Generator Module) Ver.1.4.0 Data Transfer Controller Module (Code Generator Module) Ver.1.3.1 UART Communication Module (Code Generator Module) Ver.1.5.0
Board Used	RL78/G23-128p Fast Prototyping Board (RTK7RLG230CSN000BJ) RL78/G14 Fast Prototyping Board (RTK5RLG140C00000BJ)
UART Bit Rate	2Mbps using USB/UART adaptor

1.5 Notes/Restrictions

- Data Collector Module
  Maximum numbers of Collector channels: 16 fixed.
  Data type “Double”: Not supported.

- UART Communication Module
  Serial Interface UARTAx channels are not supported.

- Warning when Configuring Compiler “-far_rom” Option Setting
  The following warning will occur.
  Set the lower 16 bits of the address to DTCSAR. Therefore, there is no problem.

```
rm_rai_data_collector_driver_rl.c(253):W0520767:Conversion from pointer to smaller integer
```

2. API Information

2.1 Hardware Requirements

The MCU used must support the following functions.

- Timer Array Unit (TAU) Interval Timer for Data Collector SIS module
- Data Transfer Controller (DTC) for Data Collector SIS module
- Serial Array Unit (SAI) for COMMS UART SIS module

2.2 Software Requirements

The SIS modules are dependent upon the following packages:

- Board Support Package Module (r_bsp) Ver.1.60 or higher
- TAU Interval Timer Module (Code Generator Module) Ver.1.4.0 or higher
- Data Transfer Controller Module (Code Generator Module) Ver.1.3.1 or higher
- UART Communication Module (Code Generator Module) Ver.1.5.0 or higher

2.3 Supported Toolchains

The SIS modules are tested and work with the following toolchain:

- Renesas RL78 Toolchain v.1.12.00 or higher

2.4 Usage of Interrupt Vector

The SIS modules do not use interrupts. However, the following modules to be used use interrupts. Refer to each application note for detail information.

- TAU Interval Timer Module
- Data Transfer Controller Module
- UART Communication Module

2.5 Header Files

All API calls and their supporting interface definitions are located as follows.

- Data Shipper SIS module
  rm_rai_data_shipper_if.h
  rm_rai_data_shipper_api.h
  rm_rai_data_shipper_r_crc_api_complement.h
  rm_rai_data_shipper.h
- Data Collector SIS module
  rm_rai_data_collector_if.h
  rm_rai_data_collector_api.h
  rm_rai_data_collector_r_timer_api_complement.h
  rm_rai_data_collector_r_transfer_api_complement.h
  rm_rai_data_collector.h
- COMMS UART SIS Module
  rm_comms_usrt_if.h
  rm_comms_uart_api.h
  rm_comms_uart_r_uart_api_complement.h
  rm_comms_uart.h

2.6 Integer Types

The projects for these SIS modules use ANSI C99. These types are defined in stdint.h.

2.7 Configuration Overview

The configuration options in these SIS modules are specified in
rm_rai_data_shipper_rl_config.h and rm_rai_data_shipper_instance_rl.c for Data Shipper SIS module,
rm_rai_data_collector_rl_config.h and rm_rai_data_collector_instance_rl.c for Data Collector SIS module,
rm_comms_uart_rl_config.h and rm_comms_uart_instance_rl.c. for COMMS UART SIS module.

It is also necessary to set the following modules to be used.

- TAU Interval Timer Module
- Data Transfer Controller Module
- UART Communication Module

For other SIS modules, refer to each application note for detail information.

2.7.1 Data Shipper SIS Module Configuration (rm_rai_data_shipper_rl_config.h)

The following explains the option names and setting values of this SIS module. The configuration settings shown in following table are set on Smart Configurator.

Configuration options	Description (Smart Configurator display)
RM_RAI_DATA_SHIPPER_CFG_PARAM_CHECKING_ENABLE	Specify whether to include code for API parameter checking. Selection: BSP_CFG_PARAM_CHECKING_ENABLE Enabled (1) Disabled (0) Default: BSP_CFG_PARAM_CHECKING_ENABLE
RM_RAI_DATA_SHIPPER_CFG_FRAME_RATE_DIVIDER	Specify the number of consecutive skips after sending data. Fill the value. (Need user to input.) Default: 0 (No Skip)
RM_RAI_DATA_SHIPPER_CFG_CALLBACK	Specify user callback function name. Fill the function name. (Need user to input.) Default: rai_data_shipper0_callback
RM_RAI_DATA_SHIPPER_CFG_DEVICE_COMMS_INSTANCE	Specify using UART communication device number. Selection: UART Communication Device(x) (x: 0 – 4) Default: UART Communication Device0 (g_comms_uart_device0)
RM_RAI_DATA_SHIPPER_CFG_CRC_ENABLE	Specify whether to execute the Checksum calculation. Selection: Enabled (1) Disabled (0) Default: 1

2.7.2 Data Collector SIS Module Configuration (rm_rai_data_collector_rl_config.h)

The following explains the option names and setting values of this SIS module. The configuration settings shown in following table are set on Smart Configurator.

Configuration options	Description (Smart Configurator display)
RM_RAI_DATA_COLLECTOR_CFG_PARAM _CHECKING_ENABLE	Specify whether to include code for API parameter checking. Selection: BSP_CFG_PARAM_CHECKING_ENABLE Enabled (1) Disabled (0) Default: BSP_CFG_PARAM_CHECKING_ENABLE
RM_RAI_DATA_COLLECTOR_CFG _MAX_CHANNELS	Specify maximum numbers of Collector channels. Selection: 16 (fixed) Refer to “1.5 Notes/Restrictions”. Default: 16
RM_RAI_DATA_COLLECTOR_CFG_DC0_ID	Specify any value. Fill the value (0 – 255). (Need user to input.) Default: 0
RM_RAI_DATA_COLLECTOR_CFG_DC0 _FRAME_BUF_LEN	Specify the frame buffer length. Selection: 1 - Default: 50
RM_RAI_DATA_COLLECTOR_CFG_DC0 _DATA_READY_CALLBACK	Specify user callback function name. Fill the function name. (Need user to input.) Default: rai_data_collector0_callback
RM_RAI_DATA_COLLECTOR_CFG_DC0 _ERROR_CALLBACK	Specify user callback function name. Fill the function name. (Need user to input.) Default: rai_data_collector0_error_callback
RM_RAI_DATA_COLLECTOR_CFG_DC0 _SNAPSHOT_CHANNELn_NAME (“n”: 0 - 7)	Specify channel name for Snapshot mode of Device0. Fill the function name. (Need user to input.) Default: NULL
RM_RAI_DATA_COLLECTOR_CFG_DC0 _SNAPSHOT_CHANNELn_DATA_TYPE (“n”: 0 - 7) (Note 1)	Specify data type of channel for Snapshot mode of Device0. Selection: Refer to “2.9.2(3)Data Type e_rai_data_collector_data_type”. Default: 1 (Signed 8-bit)
RM_RAI_DATA_COLLECTOR_CFG_DC0 _SNAPSHOT_CHANNELS	Specify number of Snapshot mode channels of Device0. Selection: 0 - 8 Default: 1
RM_RAI_DATA_COLLECTOR_CFG_DC0 _SNAPSHOT_COUNT	Specify DTC transfer count for Snapshot mode of Device0. Selection: 1 - 255 Default: 1
RM_RAI_DATA_COLLECTOR_CFG_DC0 _DATA_FEED_CHANNELn_NAME (“n”: 0 - 7)	Specify channel name for Data Feed mode of Device0. Fill the function name. (Need user to input.) Default: NULL
RM_RAI_DATA_COLLECTOR_CFG_DC0 _DATA_FEED_CHANNELn_DATA_TYPE (“n”: 0 - 7)	Specify data type of channel for Data Feed mode of Device0. Selection: Refer to “2.9.2(3)Data Type e_rai_data_collector_data_type”. Default: 1 (Signed 8-bit)
RM_RAI_DATA_COLLECTOR_CFG_DC0 _DATA_FEED_CHANNELS	Specify number of Data Feed mode channels of Device0. Selection: 0 - 8 Default: 1
RM_RAI_DATA_COLLECTOR_CFG_DC0 _TIMER_DRIVER_TYPE	Specify driver type of Device0. Selection: 16-bits counter (1) 32-bits counter (2) Not selected (0) Default: Not selected
RM_RAI_DATA_COLLECTOR_CFG_DC0 _TIMER_COMPONENT	Specify timer component name of Device0. Fill the timer component name. (Need user to input.) Default: Config_TAU0_1
RM_RAI_DATA_COLLECTOR_CFG_DC0 _DTC_COMPONENT	Specify DTC component name of Device0. Fill the DTC component name. (Need user to input.) Default: Config_DTC

RM_RAI_DATA_COLLECTOR_CFG_DC0 _DTC_CONTROL_DATA_START_NUMBER	Specify the start number for DTC control data of Device0. Fill the start number. (Need user to input.) Default: 0

Note 1: Data type “Double” is not supported. Refer to “1.5 Notes/Restrictions”.

2.7.3 COMMS UART SIS Module Configuration (r_comms_uart_rl_config.h)

The following explains the option names and setting values of this SIS module. The configuration settings shown in following table are set on Smart Configurator.

Configuration	Description (Smart Configurator display)
RM_COMMS_UART_CFG_PARAM_CHECKING_ENABLE	Specify whether to include code for API parameter checking. Selection: BSP_CFG_PARAM_CHECKING_ENABLE Enabled (1) Disabled (0) Default: BSP_CFG_PARAM_CHECKING_ENABLE
COMMS_UART_CFG_BUS_NUM_MAX (Note 1)	Set the numbers (max.) of UART shared buses. Selection: Unused (0), 1 Default: 1
COMMS_UART_CFG_DEVICE_NUM_MAX (Note 1)	Set the numbers (max.) of UART devices. Selection: Unused (0), 1 Default: 1
COMMS_UART_CFG_RTOS_TX_BLOCKING_SUPPORT_ENABLE (Note 2)	Specify TX blocking operation of RTOS project. Selection: Enabled (1) – Not supported Disabled (0) Default: Enabled (1)
COMMS_UART_CFG_RTOS_TX_BUS_LOCK_SUPPORT_ENABLE (Note 2)	Specify TX bus locked operation of RTOS project. Selection: Enabled (1) – Not supported Disabled (0) Default: Enabled (1)
COMMS_UART_CFG_RTOS_RX_BLOCKING_SUPPORT_ENABLE (Note 2)	Specify RX blocking operation of RTOS project. Selection: Enabled (1) – Not supported Disabled (0) Default: Enabled (1)
COMMS_UART_CFG_RTOS_RX_BUS_LOCK_SUPPORT_ENABLE (Note 2)	Specify RX bus locked operation of RTOS project. Selection: Enabled (1) – Not supported Disabled (0) Default: Enabled (1)
COMMS_UART_CFG_BUS0_SAU_UART_COMPONENT (Note 6)	Specify the component name of the specified UART bus. Fill “Config_UART(x)”. (“x” = 0-) Default: Config_UART0
COMMS_UART_CFG_BUS0_SAU_UART_CH (Note 6)	Specify the channel number of the specified UART bus. Fill the channel number. (Need user to input) Default: 0

COMMS_UART_CFG_BUS0_TIMEOUT (Note 4)	Specify the timeout for bus lock of RTOS project. Fill the value. (Need user to input) Default: 0xFFFFFFFF
COMMS_UART_CFG_DEVICE0_BUS_CH	Specify the channel number of the specified UART device. Selection: g_comms_uart_bus(x)_extended_cfg (Note 1) Default: g_comms_uart_bus0_extended_cfg
COMMS_UART_CFG_DEVICE0_CALLBACK_ENABLE (Note 5)	Specify the callback operation of the specified UART device. Selection: 1: Enabled 0: Disabled Default: 1: Enabled
COMMS_UART_CFG_DEVICE0_CALLBACK	Specify the Callback function of the specified UART device. Fill the callback function name. (Need user to input) Default: rm_comms_uart_user_callback
COMMS_UART_CFG_DEVICE0_BLOCKING_TIMEOUT (Note 4)	Specify the blocking timeout of RTOS project. Fill the value. (Need user to input) Default: 0xFFFFFFFF

Note 1: Only “1” supported.
Note 2: Not supported. Therefore, set to “Disabled (0)”.
Note 3: Setting of None-parity, 8-bit, 1 stop-bit has been confirmed.
Note 4: These options are disabled because the following options are not supported.
COMMS_UART_CFG_RTOS_TX_BLOCKING_SUPPORT_ENABLE
COMMS_UART_CFG_RTOS_TX_BUS_LOCK_SUPPORT_ENABLE
COMMS_UART_CFG_RTOS_RX_BLOCKING_SUPPORT_ENABLE
COMMS_UART_CFG_RTOS_RX_BUS_LOCK_SUPPORT_ENABLE
Note 5: This should be set to “1”.
Note 6: Serial Interface UARTAx channels are not supported. Refer to “1.5 Notes/Restrictions”.

2.8 Code Size

Typical code sizes associated with this SIS module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in “2.7 Configuration Overview”. The table lists reference values when the C compiler’s compile options are set to their default values, as described in “2.3 Supported Toolchains”.
The compiler option default values.

- optimization level: -Odefault “-g -o ${workspace_loc:/${ProjName}/${ConfigName}}”
- data endianness: little-endian

The code size varies depending on the C compiler version and compile options.
The values in the table below are confirmed under the following conditions.

- Module Version:
  TAU Interval Timer Module (Code Generator Module) Ver.1.4.0
  Data Transfer Controller Module (Code Generator Module) Ver.1.3.1
  UART Communication Module (Code Generator Module) Ver.1.5.0
- Compiler Version:
  Renesas Electronics C/C++ Compiler Package for RL78 Family V1.12.00
  (The option of “-lang = c99” is added to the default settings of the integrated development environment.)
  -far_rom (configurable)
- Configuration Options: Default settings

Table 2-1 ROM/RAM Size

OS supporting	MCU	SIS Module	Category	Size [bytes]		Condition
				without -far_rom option	with -far_rom option	
Non	RL78/G23 RL78/G14 (S3 core)	Data Shipper	ROM	1,060	1,655	
			RAM	136	140	
		Data Collector	ROM	2,373	3,814	Note 1
			RAM	206	218	
		COMMS UART	ROM	559	839	
			RAM	18	28	

Note 1: Condition is as follows.
RM_COMMS_UART_CFG_PARAM_CHECKING_ENABLE= 1 (Enable)
RM_RAI_DATA_COLLECTOR_CFG_PARAM_CHECKING_ENABLE = 1 (Enable)
RM_RAI_DATA_SHIPPER_CFG_PARAM_CHECKING_ENABLE = 1 (Enable)
COMMS_UART_CFG_BUS_NUM_MAX = 1
COMMS_UART_CFG_DEVICE_NUM_MAX = 1
COMMS_UART_CFG_RTOS_TX_BLOCKING_SUPPORT_ENABLE = 0
COMMS_UART_CFG_RTOS_RX_BLOCKING_SUPPORT_ENABLE = 0
RM_RAI_DATA_COLLECTOR_CFG_MAX_CHANNELS = 16
RM_RAI_DATA_COLLECTOR_CFG_DC0_FRAME_BUF_LEN = 50
RM_RAI_DATA_COLLECTOR_CFG_DC0_SNAPSHOT_CHANNEL0_DATA_TYPE = Unsigned 32-bit
RM_RAI_DATA_COLLECTOR_CFG_DC0_SNAPSHOT_CHANNELS = 1
RM_RAI_DATA_COLLECTOR_CFG_DC0_SNAPSHOT_COUNT = 1
RM_RAI_DATA_COLLECTOR_CFG_DC0_DATA_FEED_CHANNELS = 0
RM_RAI_DATA_SHIPPER_CFG_DC_NUM_MAX = 1
RM_RAI_DATA_SHIPPER_CFG_CRC_ENABLE = 1

2.9 Parameters

The API function arguments are shown below.

The structures of “configuration structure” and “control structure” are used as parameters type. These structures are described along with the API function prototype declaration.

The configuration structure is used for the initial configuration of Data Shipper SIS module, Data Collector SIS module, and COMMS UART SIS module during the module open API call. The configuration structure is used purely as an input into each module.

The control structure is used as a unique identifier for each module instance of Data Shipper SIS module, Data Collector SIS module, and COMMS UART SIS module. It contains memory required by the module. Elements in the control structure are owned by the associated module and must not be modified by the application. The user allocates storage for a control structure, often as a global variable, then sends a pointer to it into the module open API call for a module.

2.9.1 Configuration Structure and Control Structure of Data Shipper SIS Module

(1) Configuration Struct rai_data_shipper_cfg_t

This structure is located in “rm_data_shipper_api.h” file.

```
/** RAI Data Shipper general configuration  */
typedef struct st_rai_data_shipper_cfg
{
    uint8_t        divider;                  ///< Send data on every (divider + 1) requests in
case the interface bandwidth is not sufficient

    crc_instance_t const          * p_crc;    ///< Pointer to CRC instance
    rm_comms_uart_instance_t const        * p_comms;  ///< Pointer to COMMS API instance

    void const     * p_context;               ///< Pointer to the user-provided context
    void           (* p_callback)(rai_data_shipper_callback_args_t * p_args);   ///< Pointer to the
callback function on data sent or error
} rai_data_shipper_cfg_t;
```

(2) Control Struct rm_rai_data_shipper_ctrl_t

This is Data Shipper SIS module control block and allocates an instance specific control block to pass into the Data Shipper API calls. This structure is implemented as “rai_data_shipper_instance_ctrl_t” located in “rm_rai_data_shipper.h” file.

```
/** RAI_DATA_SHIPPER instance control block. Initialization occurs when RM_RAI_DATA_SHIPPER_Open()
is called. */
typedef struct st_rai_data_shipper_instance_ctrl
{
    uint32_t       opened;                   // Flag to determine if the module is open or not
    rai_data_shipper_cfg_t const  * p_cfg;    // Pointer to configuration structure
    rai_data_shipper_tx_info_t    tx_info;    // Tx info
} rai_data_shipper_instance_ctrl_t;
```

2.9.2 Configuration Structure and Control Structure of Data Collector SIS Module

(1) Configuration Struct rai_data_collector_cfg_t

This structure is located in “rm_data_collector_api.h” file.

```
/** RAI Data Collector general configuration  */
typedef struct st_rai_data_collector_cfg
{
    uint32_t   channels      : 8;            ///< Total number of channels
    uint32_t   instance_id   : 8;            ///< Instance id
    uint32_t   virt_channels : 8;            ///< Virtual channels
    uint32_t   reserved      : 8;            ///< Reserved

    uint32_t   channel_ready_mask;           ///< Bitmask of configured channels
    uint32_t   required_frame_len;           ///< Length of each frame buffer
    rai_data_collector_snapshot_cfg_t     const  * p_snapshot_cfg;   ///< Pointer to snapshot mode
configuration structure
    rai_data_collector_data_feed_cfg_t    const * p_data_feed_cfg;   ///< Pointer to data feed mode
configuration structure
    void       * p_extend;                   ///< Pointer to extended configuration structure

    void (* p_callback)(rai_data_collector_callback_args_t const * p_args);     ///< Pointer to the
callback function when data is collected
    void (* p_error_callback)(rai_data_collector_error_callback_args_t const * p_args);    ///<
Pointer to the callback function when there is an error
    void const * p_context;                  ///< Pointer to the user-provided context
} rai_data_collector_cfg_t;
```

(2) Control Struct rm_rai_data_collector_ctrl_t

This is Data Collector SIS module control block and allocates an instance specific control block to pass into the Data Collector API calls. This structure is implemented as “rai_data_collector_instance_ctrl_t” located in “rm_rai_data_collector.h” file.

```
/** RAI_DATA_COLLECTOR instance control block. Initialization occurs when
RM_RAI_DATA_COLLECTOR_Open() is called. */
typedef struct st_rai_data_collector_instance_ctrl
{
    uint32_t           opened;               // Flag to determine if the module is open or not

    volatile uint32_t  channel_ready;        // Bit mask of channels that have frame buffers ready to submit
    volatile rai_data_collector_buffer_status_t  buf_status; // PING-PONG buffer status

    rai_data_collector_cfg_t const       * p_cfg;          // Pointer to configuration structure
    rai_data_collector_extended_cfg_t    * p_extend;       // Pointer to extended configuration structure
} rai_data_collector_instance_ctrl_t;
```

(3) Data Type e_rai_data_collector_data_type

```
/** Data types */
typedef enum e_rai_data_collector_data_type
{
    RAI_DATA_COLLECTOR_DATA_TYPE_INT8_T   = 0x01, ///< Signed 8-bit
    RAI_DATA_COLLECTOR_DATA_TYPE_UINT8_T  = 0x11, ///< Unsigned 8-bit
    RAI_DATA_COLLECTOR_DATA_TYPE_INT16_T  = 0x22, ///< Signed 16-bit
    RAI_DATA_COLLECTOR_DATA_TYPE_UINT16_T = 0x32, ///< Unsigned 16-bit
    RAI_DATA_COLLECTOR_DATA_TYPE_INT32_T  = 0x44, ///< Signed 32-bit
    RAI_DATA_COLLECTOR_DATA_TYPE_UINT32_T = 0x54, ///< Unsigned 32-bit
    RAI_DATA_COLLECTOR_DATA_TYPE_FLOAT    = 0x64, ///< Float
    RAI_DATA_COLLECTOR_DATA_TYPE_DOUBLE   = 0x78, ///< Double
} rai_data_collector_data_type_t;
```

2.9.3 Configuration Structure and Control Structure of COMMS UART SIS Module

(1) Configuration Struct rm_comms_uart_cfg_t

This structure is located in “rm_comms_api.h” file.

```
/** Communications middleware configuration block */
typedef struct st_rm_comms_uart_cfg
{
    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 const * p_context;               ///< Pointer to the user-provided context
    void    (* p_callback)(rm_comms_uart_callback_args_t * p_args); ///< Pointer to callback
function, mostly used if using non-blocking functionality.
} rm_comms_uart_cfg_t;
```

(2) Control Struct rm_comms_uart_ctrl_t

This is COMMS UART SIS module control block and allocates an instance specific control block to pass into the COMMS API calls. This structure is implemented as “rm_comms_uart_instance_ctrl_t” located in “rm_comms_uart.h” file.

```
/** Communications middleware control structure. */
typedef struct st_rm_comms_uart_instance_ctrl
{
    uint32_t              open;          ///< Open flag.
    rm_comms_uart_cfg_t const * p_cfg;   ///< Middleware configuration.
    rm_comms_uart_extended_cfg_t const    * p_extend; ///< Pointer to extended configuration structure
    void    (* p_callback)(rm_comms_uart_callback_args_t * p_args); ///< Pointer to callback that
is called when a uart_event_t occurs.
    void const            * p_context;   ///< Pointer to context passed into callback function
} rm_comms_uart_instance_ctrl_t;
```

2.10 Return Values

The API function return values are shown below.

```
This enumeration is listed in fsp_common_api.h which is included in RL78 BSP (Board Support Package Module)
typedef enum e_fsp_err
{
    FSP_SUCCESS = 0,

    FSP_ERR_ASSERTION            = 1,    ///< A critical assertion has failed
    FSP_ERR_INVALID_POINTER      = 2,    ///< Pointer points to invalid memory location
    FSP_ERR_INVALID_ARGUMENT     = 3,    ///< Invalid input parameter
    FSP_ERR_INVALID_CHANNEL      = 4,    ///< Selected channel does not exist
    FSP_ERR_INVALID_MODE         = 5,    ///< Unsupported or incorrect mode
    FSP_ERR_UNSUPPORTED          = 6,    ///< Selected mode not supported by this API
    FSP_ERR_NOT_OPEN             = 7,    ///< Requested channel is not configured or API not open
    FSP_ERR_IN_USE               = 8,    ///< Channel/peripheral is running/busy
    FSP_ERR_OUT_OF_MEMORY        = 9,    ///< Allocate more memory in the driver's cfg.h
    FSP_ERR_HW_LOCKED            = 10,   ///< Hardware is locked
    FSP_ERR_IRQ_BSP_DISABLED     = 11,   ///< IRQ not enabled in BSP
    FSP_ERR_OVERFLOW             = 12,   ///< Hardware overflow
    FSP_ERR_UNDERFLOW            = 13,   ///< Hardware underflow
    FSP_ERR_ALREADY_OPEN         = 14,   ///< Requested channel is already open in a different configuration
    FSP_ERR_APPROXIMATION        = 15,   ///< Could not set value to exact result
    FSP_ERR_CLAMPED              = 16,   ///< Value had to be limited for some reason
    FSP_ERR_INVALID_RATE         = 17,   ///< Selected rate could not be met
    FSP_ERR_ABORTED              = 18,   ///< An operation was aborted
    FSP_ERR_NOT_ENABLED          = 19,   ///< Requested operation is not enabled
    FSP_ERR_TIMEOUT              = 20,   ///< Timeout error
    FSP_ERR_INVALID_BLOCKS       = 21,   ///< Invalid number of blocks supplied
    FSP_ERR_INVALID_ADDRESS      = 22,   ///< Invalid address supplied
    FSP_ERR_INVALID_SIZE         = 23,   ///< Invalid size/length supplied for operation
    FSP_ERR_WRITE_FAILED         = 24,   ///< Write operation failed
    FSP_ERR_ERASE_FAILED         = 25,   ///< Erase operation failed
    FSP_ERR_INVALID_CALL         = 26,   ///< Invalid function call is made
    FSP_ERR_INVALID_HW_CONDITION = 27,   ///< Detected hardware is in invalid condition
    FSP_ERR_INVALID_FACTORY_FLASH = 28,  ///< Factory flash is not available on this MCU
    FSP_ERR_INVALID_STATE        = 30,   ///< API or command not valid in the current state
    FSP_ERR_NOT_ERASED           = 31,   ///< Erase verification failed
    FSP_ERR_SECTOR_RELEASE_FAILED = 32,  ///< Sector release failed
    FSP_ERR_NOT_INITIALIZED      = 33,   ///< Required initialization not complete
    FSP_ERR_NOT_FOUND            = 34,   ///< The requested item could not be found
    FSP_ERR_NO_CALLBACK_MEMORY   = 35,   ///< Non-secure callback memory not provided for non-secure callback
    FSP_ERR_BUFFER_EMPTY         = 36,   ///< No data available in buffer

    /* Start of RTOS only error codes */
    FSP_ERR_INTERNAL             = 100,  ///< Internal error
    FSP_ERR_WAIT_ABORTED         = 101,  ///< Wait aborted

    /* Start of Sensor specific */
    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.

    /* Start of CTSU Driver specific */
    FSP_ERR_CTSU_SCANNING                = 6000,    ///< Scanning.
    FSP_ERR_CTSU_NOT_GET_DATA            = 6001,    ///< Not processed previous scan data.
    FSP_ERR_CTSU_INCOMPLETE_TUNING       = 6002,    ///< Incomplete initial offset tuning.
    FSP_ERR_CTSU_DIAG_NOT_YET            = 6003,    ///< Diagnosis of data collected no yet.
    FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE   = 6004,    ///< Diagnosis of LDO over voltage failed.
    FSP_ERR_CTSU_DIAG_CCO_HIGH           = 6005,    ///< Diagnosis of CCO into 19.2uA failed.
    FSP_ERR_CTSU_DIAG_CCO_LOW            = 6006,    ///< Diagnosis of CCO into 2.4uA failed.
    FSP_ERR_CTSU_DIAG_SSCG               = 6007,    ///< Diagnosis of SSCG frequency failed.
    FSP_ERR_CTSU_DIAG_DAC                = 6008,    ///< Diagnosis of non-touch count value failed.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE     = 6009,    ///< Diagnosis of LDO output voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE       = 6010,    ///< Diagnosis of over voltage detection circuit failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT       = 6011,    ///< Diagnosis of over current detection circuit failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE    = 6012,    ///< Diagnosis of LDO internal resistance value failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE     = 6013,    ///< Diagnosis of Current source value failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN       = 6014,    ///< Diagnosis of SENSCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN         = 6015,    ///< Diagnosis of SUCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY     = 6016,    ///< Diagnosis of SUCLK clock recovery function failed.
    FSP_ERR_CTSU_DIAG_CFC_GAIN           = 6017,    ///< Diagnosis of CFC oscillator gain failed.

    /* Start of COMMS specific */
    FSP_ERR_COMMS_BUS_NOT_OPEN,                     ///< Bus is not open.
} fsp_err_t;
```

2.11 Adding the SIS Module to Your Project

This module must be added to each project in which it is used. Renesas recommends using “Smart Configurator” described in (1) or (2). However, “Smart Configurator” only supports some RL78 devices.

(1) Adding the SIS module to your project using “Smart Configurator” in e² studio

By using the “Smart Configurator” in e² studio, the SIS module is automatically added to your project. Refer to “RL78 Smart Configurator User’s Guide: e² studio(R20AN0579)” for details.

(2) Adding the SIS module to your project using “Smart Configurator” on CS+

By using the “Smart Configurator Standalone version” in CS+, the SIS module is automatically added to your project. Refer to “RL78 Smart Configurator User’s Guide: CS+ (R20AN0580)” for details.

3. Data Shipper API Functions

3.1 RM_RAI_DATA_SHIPPER_Open()

This function opens and configures the Data Shipper module.

This function must be called before calling any other Data Shipper API functions.

Format

```
fsp_err_t        RM_RAI_DATA_SHIPPER_Open(
        rai_data_shipper_ctrl_t * const              p_api_ctrl,
        rai_data_shipper_cfg_t const * const         p_cfg
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.1(2)Control Struct rm_rai_data_shipper_ctrl_t.

p_cfg

Pointer to configuration structure

The members of this structure are shown in 2.9.1(1)Configuration Struct rai_data_shipper_cfg_t.

Return Values

FSP_SUCCESS	Data Shipper successfully configured.
FSP_ERR_ALREADY_OPEN	Module already open.
FSP_ERR_ASSERTION	One or more pointers point to NULL or callback is NULL.

Properties

Prototyped in rm_rai_data_shipper.h

Description

This function opens and configures the Data Shipper SIS module.

This function opens CRC Calculator operation if CRC operation is enabled, also.

This function set header data of TX information to TX buffer.

This function configures as follows:

- Sets related instance of COMMS UART SIS module.
- Sets callback and context.
- Sets “open” flag.

This function calls open API of COMMS UART SIS module to open communication module after all above initializations are done.

Special Notes

None

3.2 RM_RAI_DATA_SHIPPER_Close()

This function closes the Data Shipper module instance.

Format

```
fsp_err_t        RM_RAI_DATA_SHIPPER_Close(
        rai_data_shipper_ctrl_t * const              p_api_ctrl
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.1(2)Control Struct rm_rai_data_shipper_ctrl_t.

Return Values

FSP_SUCCESS	Data Shipper module closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_shipper.h

Description

This function calls close API of COMMS UART SIS module to close communication module.

This function clears the “open” flag after all above are done.

Special Notes

None

3.3 RM_RAI_DATA_SHIPPER_Read()

This function reads data.

This function is not supported.

Format

```
fsp_err_t        RM_RAI_DATA_SHIPPER_Read(
        rai_data_shipper_ctrl_t * const          p_api_ctrl,
        void * const                             p_buf,
        uint32_t * const                         buf_len
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.1(2)Control Struct rm_rai_data_shipper_ctrl_t.

p_buf

Pointer to the buffer to store read data

buf_len

Number of bytes to read

Return Values

FSP_ERR_UNSUPPORTED	Data Shipper module read not supported.

Properties

Prototyped in rm_rai_data_shipper.h

Description

This function is not supported.

Special Notes

None

3.4 RM_RAI_DATA_SHIPPER_Write()

This function writes data.

This function sends the data over UART bus.

Format

```
fsp_err_t        RM_RAI_DATA_SHIPPER_Write(
-       rai_data_shipper_ctrl_t * const                 p_api_ctrl,
        rai_data_shipper_write_params_t const *         p_write_params
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.1(2)Control Struct rm_rai_data_shipper_ctrl_t.

p_write_params

Pointer to write parameters structure

```
typedef struct st_rai_data_shipper_write_params
{
    uint16_t    events;                                 ///< Events
    uint16_t    diagnostic_data_len;                    ///< Diagnostic data length
    uint8_t *   p_diagnostic_data;                      ///< Pointer to diagnostic data
    rai_data_collector_callback_args_t *  p_sensor_data;    ///< Pointer to sensor data info
} rai_data_shipper_write_params_t;
```

Return Values

FSP_SUCCESS	Tx buf list created and transmission starts, or write request skipped.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_shipper.h

Description

This function starts sending the data over UART bus.

The write API of COMMS UART SIS module is called in this function to send the data.

Special Notes

This function may be called in ISR.

3.5 rai_data_shipper_write_callback()

This is a write callback function for Data Shipper module.

Format

```
void            rai_data_shipper_write_callback(
    rm_comms_uart_callback_args_t *     p_args
)
```

Parameters

p_args

Pointer to callback parameter definition

```
typedef struct st_rm_comms_uart_callback_args
{
    void const *                p_context;
    rm_comms_uart_event_t       event;
} rm_comms_uart_callback_args_t;
```

Return Values

None

Properties

Prototyped in rm_rai_data_shipper_instance_rl.c

Description

This callback function is a write callback function called in UART Communication module callback function.

This callback function determines the next channel write execution using the RM_RAI_DATA_SHIPPER_Write() function, successful end and error end by the event.

If it is the successful end or the error end, the user RM_RAI_DATA_SHIPPER_CFG_CALLBACK function that user configured in rm_rai_data_shipper_rl_config.h file is called.

For the function prototypes of the RM_RAI_DATA_SHIPPER_CFG_CALLBACK function, refer to “rm_rai_data_shipper_instance_rl.c” file.

And the events of COMMS UART SIS module are as follow.

```
typedef enum e_rm_comms_uart_event
{
    RM_COMMS_UART_EVENT_OPERATION_COMPLETE = 0,
    RM_COMMS_UART_EVENT_TX_OPERATION_COMPLETE,      : Transmission end
    RM_COMMS_UART_EVENT_RX_OPERATION_COMPLETE,      : Reception end
    RM_COMMS_UART_EVENT_ERROR,                      :=Receive error
} rm_comms_uart_event_t;
```

Special Notes

None

4. Data Collector API Functions

4.1 RM_RAI_DATA_COLLECTOR_Open()

This function opens and configures the Data Collector module.

This function must be called before calling any other Data Collector API functions.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_Open(
        rai_data_collector_ctrl_t * const          p_api_ctrl,
        rai_data_collector_cfg_t const * const     p_cfg
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

p_cfg

Pointer to configuration structure

The members of this structure are shown in 2.9.2(1)Configuration Struct rai_data_collector_cfg_t.

Return Values

FSP_SUCCESS	: Data Collector successfully configured.
FSP_ERR_ALREADY_OPEN	: Module already open.
FSP_ERR_ASSERTION	: One or more pointers point to NULL or callback is NULL.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function opens and configures the Data Collector SIS module.

This creates the DTC control data.

This function configures as follows:

- Sets “channel_ready” to 0 and “buf_status” to idle.
- Sets internal buffers to initial state.
- Sets “open” flag.

Special Notes

None

4.2 RM_RAI_DATA_COLLECTOR_Close()

This function closes Data Collector module instance.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_Close(
        rai_data_collector_ctrl_t * const            p_api_ctrl
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

Return Values

FSP_SUCCESS	Data Collector module closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function clears the “open” flag after all above are done.

Special Notes

None

4.3 RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister()

This function configures transfer source address for snapshot mode channel.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister(
        rai_data_collector_ctrl_t * const        p_api_ctrl,
        uint8_t                                  channel,
        void const *                             p_src
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

channel

Which snapshot mode channel

p_src

Pointer to transfer source address

Return Values

FSP_SUCCESS	Source addresses are set.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function sets source address for DTC transfer information.

Special Notes

None

4.4 RM_RAI_DATA_COLLECTOR_BufferReset()

This function resets to discard accumulated data and start with internal buffers.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_BufferReset(
        rai_data_collector_ctrl_t * const            p_api_ctrl,
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

Return Values

FSP_SUCCESS	Data Collector module internal buffers reset.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function resets to discard accumulated data and start with internal buffers.

Special Notes

Application must stop data transfer on all channels first.

4.5 RM_RAI_DATA_COLLECTOR_BufferRelease()

This function releases frame buffers.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_BufferRelease(
        rai_data_collector_ctrl_t * const            p_api_ctrl,
)
```

Parameters

p_api_ctrl

Pointer to control structure
The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

Return Values

FSP_SUCCESS	Buffer released.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function releases frame buffers.

“buf_status” is set to idle.

Special Notes

None

4.6 RM_RAI_DATA_COLLECTOR_ChannelBufferGet()

This function obtains channel destination buffer address for asynchronous data transfer.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_ChannelBufferGet(
        rai_data_collector_ctrl_t * const        p_api_ctrl,
        uint8_t                                  channel,
        void **                                  pp_buf
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

channel

Which snapshot mode channel

pp_buf

Returned buffer address

Return Values

FSP_SUCCESS	Buffer available.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function obtains channel destination buffer address for asynchronous data transfer.

Special Notes

None

4.7 RM_RAI_DATA_COLLECTOR_ChannelWrite()

This function performs synchronous data transfer using CPU copy.
For data feed mode only.

Format

```
fsp_err_t       RM_RAI_DATA_COLLECTOR_ChannelWrite(
        rai_data_collector_ctrl_t * const       p_api_ctrl,
        uint8_t                                 channel,
        const void *                            p_buf,
        uint32_t                                len
)
```

Parameters

p_api_ctrl
Pointer to control structure
The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.
channel
Which data feed mode channel
p_buf
Data buffer
len
Length of data buffer in data samples

Return Values

FSP_SUCCESS	Data copy completed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function writes data to frame(internal) buffers using CPU copy.

Special Notes

For data feed mode only.

4.8 RM_RAI_DATA_COLLECTOR_SnapshotStart()

This function starts snapshot mode channels.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_SnapshotStart(
        rai_data_collector_ctrl_t * const            p_api_ctrl,
)
```

Parameters

p_api_ctrl

Pointer to control structure
The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

Return Values

FSP_SUCCESS	Snapshot mode started.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.
FSP_ERR_UNSUPPORTED	No snapshot mode channel.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function executes as below.

- Updates the DTC transfer information and sets the related DTCEN to “1”.
- Starts the count of the TAU.

Special Notes

None

4.9 RM_RAI_DATA_COLLECTOR_SnapshotStop()

This function stops snapshot mode channels.

Format

```
fsp_err_t        RM_RAI_DATA_COLLECTOR_SnapshotStop(
        rai_data_collector_ctrl_t * const            p_api_ctrl
)
```

Parameters

p_api_ctrl

Pointer to control structure
The members of this structure are shown in 2.9.2(2)Control Struct rm_rai_data_collector_ctrl_t.

Return Values

FSP_SUCCESS	Snapshot mode stopped.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.
FSP_ERR_UNSUPPORTED	No snapshot mode channel.

Properties

Prototyped in rm_rai_data_collector.h

Description

This function executes as below.

- Stops the count of the TAU.

Special Notes

None

5. UART Communication API Functions

5.1 RM_COMMS_UART_Open()

This function opens and configures the UART Communication module.

This function must be called before calling any other UART Communication API functions.

Format

```
fsp_err_t       RM_COMMS_UART_Open(
        rm_comms_uart_ctrl_t * const            p_api_ctrl,
        rm_comms_uart_cfg_t const * const       p_cfg
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

p_cfg

Pointer to configuration structure

The members of this structure are shown in 2.9.3(1)Configuration Struct rm_comms_uart_cfg_t.

Return Values

FSP_SUCCESS	Communications Middle 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.

Properties

Prototyped in rm_comms_uart.h

Description

This function opens and configures the COMMS UART SIS module.

“R_Config_UARTx_Start()” (x: Channel Number) are executed.

This function configures as follows:

- Sets bus configuration
- Sets lower-level driver configuration
- Sets callback and context
- Sets open flag

Special Notes

None

5.2 RM_COMMS_UART_Close()

This function disables the specified UART communication module.

Format

```
fsp_err_t        RM_COMMS_UART_Close(
        rm_comms_uart_ctrl_t * const              p_api_ctrl
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

Return Values

FSP_SUCCESS	Communications Middle module successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

Properties

Prototyped in rm_comms_uart.h

Description

This function executes “R_Config_UARTx_Stop()” (x: Channel Number).

This function clears current device on bus and the “open” flag.

Special Notes

None

5.3 RM_COMMS_UART_CallbackSet()

This function updates the user callback.

Format

```
fsp_err_t       RM_COMMS_UART_CallbackSet(
        rm_comms_uart_ctrl_t * const            p_api_ctrl,
        void (* p_callback)                     (rm_comms_uart_callback_args_t *),
        void const * const                      p_context
)
```

Parameters

p_api_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

p_callbackt

Callback function

p_context

Pointer to send to callback function

Return Values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

Properties

Prototyped in rm_comms_uart.h

Description

This function reconfigures callback function and optional context pointer.

Special Notes

None

5.4 RM_COMMS_UART_Read()

This function performs a read from the UART device.

Format

```
fsp_err_t        RM_COMMS_UART_Read(
        rm_comms_uart_ctrl_t * const        p_ctrl,
        uint8_t * const                     p_dest,
        uint32_t const                      bytes
)
```

Parameters

p_ctrl

Pointer to control structure
The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

p_dest

Pointer to the buffer to store read data

bytes

Number of bytes to read

Return Values

FSP_SUCCESS	: Communications Middle module successfully configured.
FSP_ERR_ASSERTION	: Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	: Module is not open.
FSP_ERR_INVALID_ARGUMENT	: Invalid argument.

Properties

Prototyped in rm_comms_uart.h

Description

This function calls internal function “rm_comms_uart_bus_read()” to start read operation from UART bus which is SAU bus depending on the device connection.

Special Notes

None

5.5 RM_COMMS_UART_Write()

This function performs a write to the UART device.

Format

```
fsp_err_t       RM_COMMS_UART_Write(
        rm_comms_uart_ctrl_t * const        p_ctrl,
        uint8_t * const                     p_src,
        uint32_t const                      bytes
)
```

Parameters

p_ctrl

Pointer to control structure
The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

p_src

Pointer to the buffer to store writing data

bytes

Number of bytes to write

Return Values

FSP_SUCCESS	: Communications Middle module successfully configured.
FSP_ERR_ASSERTION	: Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	: Module is not open.
FSP_ERR_INVALID_ARGUMENT	: Invalid argument.

Properties

Prototyped in rm_comms_uart.h

Description

This function calls internal function “rm_comms_uart_bus_write()” to start write operation to UART bus which is SAU bus depending on device connection.

Special Notes

None

5.6 RM_COMMS_UART_WriteRead()

This function performs a write to, then a read from the UART device.

This function is not supported.

Format

```
fsp_err_t       RM_COMMS_UART_WriteRead(
        rm_comms_uart_ctrl_t * const                p_ctrl,
        rm_comms_uart_write_read_params_t const     write_read_params
)
```

Parameters

p_ctrl

Pointer to control structure

The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

write_read_params

Parameters structure for writeRead API

```
/** Struct to pack params for writeRead */
typedef struct st_rm_comms_uart_write_read_params
{
    uint8_t * p_src;            ///< pointer to buffer for storing write data
    uint8_t * p_dest;           ///< pointer to buffer for storing read data
    uint8_t   src_bytes;        ///< number of write data
    uint8_t   dest_bytes;       ///< number of read data
} rm_comms_uart_write_read_params_t;
```

Return Values

FSP_ERR_UNSUPPORTED : Not supported.

Properties

Prototyped in rm_comms_uart.h

Description

This function is not supported. So that this returns the error.

Special Notes

None.

5.7 rm_comms_uart_callback

This is a common callback function for COMMS UART SIS module called in UART driver callback function.

Format

```
void        rm_comms_uart_callback(
    rm_comms_uart_ctrl_t const *    p_api_ctrl,
    rm_comms_uart_event_t           event
)
```

Parameters

p_ctrl

Pointer to control structure
The members of this structure are shown in 2.9.3(2)Control Struct rm_comms_uart_ctrl_t.

event

Event

```
typedef enum e_rm_comms_uart_event
{
    RM_COMMS_UART_EVENT_OPERATION_COMPLETE = 0,
    RM_COMMS_UART_EVENT_TX_OPERATION_COMPLETE,      : Transmission end
    RM_COMMS_UART_EVENT_RX_OPERATION_COMPLETE,      : Reception end
    RM_COMMS_UART _EVENT_ERROR,                     : Receive error
} rm_comms_uart_event_t;
```

Return Values

None

Properties

Prototyped in rm_comms_uart_driver_rl.c

Description

This callback function is a common callback function called in UART driver callback function.

A callback function (rm_comms_uart_busN_callback() function (“N”: 0 -)) for each UART bus is provided in rm_comms_uart_instance_rl.c and calls this common callback function.

Therefore, user should add the rm_comms_uart_busN_callback() function to the UART driver of each bus. Refer to the follwing ‘Example’.

This callback function calls the user COMMS_UART_CFG_DEVICEx_CALLBACK (“x”: 0 -) function that user configured in rm_comms_uart_rl_config.h.

Therefore, user can notify the event to upper module (ex. Data Shipper module).

For the function prototypes of the COMMS_UART_CFG_DEVICEx_CALLBACK (“x”: 0 -) function, refer to “rm_comms_uart_instance_rl.c” file.

Example

A sample for “Config_UART3_user.c” is shown as follows. The rm_comms_uart_bus0_callback() is called rm_comms_uart_callback() function.

```
/***********************************************************************************************
Pragma directive
***********************************************************************************************/
```

```
#pragma interrupt r_Config_UART3_interrupt_send(vect=INTST3)
#pragma interrupt r_Config_UART3_interrupt_receive(vect=INTSR3)
#pragma interrupt r_Config_UART3_interrupt_error(vect=INTSRE3)
/* Start user code for pragma. Do not edit comment generated here */
extern void rm_comms_uart_bus0_callback(rm_comms_uart_event_t event);
/* End user code. Do not edit comment generated here */

/***********************************************************************************************************
Global variables and functions
***********************************************************************************************************/
extern volatile uint8_t * gp_uart3_tx_address;    /* uart3 transmit buffer address */
extern volatile uint16_t g_uart3_tx_count;        /* uart3 transmit data number */
extern volatile uint8_t * gp_uart3_rx_address;    /* uart3 receive buffer address */
extern volatile uint16_t g_uart3_rx_count;        /* uart3 receive data number */
extern uint16_t g_uart3_rx_length;                /* uart3 receive data length */
/* Start user code for global. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

/***********************************************************************************************************
* Function Name: R_Config_UART3_Create_UserInit
* Description  : This function adds user code after initializing UART3.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
void R_Config_UART3_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
* Function Name: r_Config_UART3_callback_sendend
* Description  : This function is a callback function when UART3 finishes transmission.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
static void r_Config_UART3_callback_sendend(void)
{
    /* Start user code for r_Config_UART3_callback_sendend. Do not edit comment generated here */
    rm_comms_uart_bus0_callback(RM_COMMS_UART_EVENT_TX_OPERATION_COMPLETE);
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
* Function Name: r_Config_UART3_callback_receiveend
* Description  : This function is a callback function when UART3 finishes reception.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
static void r_Config_UART3_callback_receiveend(void)
{
    /* Start user code for r_Config_UART3_callback_receiveend. Do not edit comment generated here */
    rm_comms_uart_bus0_callback(RM_COMMS_UART_EVENT_RX_OPERATION_COMPLETE);
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
* Function Name: r_Config_UART3_callback_error
* Description  : This function is a callback function when UART3 reception error occurs.
* Arguments    : err_type -
*                    error type info
* Return Value : None
***********************************************************************************************************/
static void r_Config_UART3_callback_error(uint8_t err_type)
{
    /* Start user code for r_Config_UART3_callback_error. Do not edit comment generated here */
    rm_comms_uart_bus0_callback(RM_COMMS_UART_EVENT_ERROR);
    /* End user code. Do not edit comment generated here */
}
```

Special Notes

None

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug 25, 2023	-	First Release
1.10	Oct 20, 2023	9	1.4 Operating Test Environment / Table 1-5 Operation Test Environment / C Compiler Added “-far_rom (configurable)”.
		9	1.5 Notes/ Restrictions Added Warning when Configuring Compiler “-far_rom” Option Setting
		12	2.7.2 Data Collector SIS Module Configuration (rm_rai_data_collector_rl_config.h) RM_RAI_DATA_COLLECTOR_CFG_DC0 _SNAPSHOT_COUNT Changed Selection value.
		15	2.8 Code Size Updated optimization level. Added “-far_rom (configurable)”.
		15	2.8 Code Size / Table 2-1 ROM/RAM Size Updated size.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

© 2023 Renesas Electronics Corporation. All rights reserved.