RX Family

Renesas Sensor Control Modules Firmware Integration Technology

Introduction

This application note explains the sensor control modules for HS300x (Renesas high performance relative humidity and temperature sensor) and sensor communication middleware for Renesas sensors using Firmware Integration Technology (FIT).

These control modules acquire the sensor data using the I2C bus control FIT module (IIC FIT Module). And calculate relative humidity value [%RH] and temperature value [°C] for HS300x sensor.

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

- The sensor control module for HS300x: HS300x FIT module

- The sensor communication middleware module: COMMS FIT module

Target Device

- **Sensors:**
  - Renesas Electronics HS300x High Performance Relative Humidity and Temperature Sensor (HS300x sensor)

- **RX Family MCUs:**
  MCUs supported the following IIC FIT module
  - I2C Bus Interface (RIIC) Module (RIIC FIT Module)
  - Simple I2C Module (SCI_IIC FIT Module) using Serial Communication Interface (SCI)

- **Operation confirmed MCU:**
  - RX65N (RIIC FIT Module, SCI_IIC FIT Module)

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 RX Family

Reference Documents

- Renesas Electronics HS300x Datasheet (April 22, 2020)
- RX Family I2C Bus Interface (RIIC) Module Using Firmware Integration Technology (R01AN1692)
- RX Family Simple I2C Module Using Firmware Integration Technology (R01AN1691)
- RX65N 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.
- RX Family Compiler CC-RX User's Manual (R20UT3248)
  The latest versions can be downloaded from the Renesas Electronics website.

Contents

1. Overview of Renesas Sensor Control Modules

The Renesas sensor control modules described in this application note is a hardware abstraction layer of Renesas sensors. This hardware abstraction layer includes sensor API and communication middleware for various Renesas sensors. The software architecture of Renesas sensor hardware abstraction layer is shown below “Figure 1-1 Renesas sensor software architecture”.

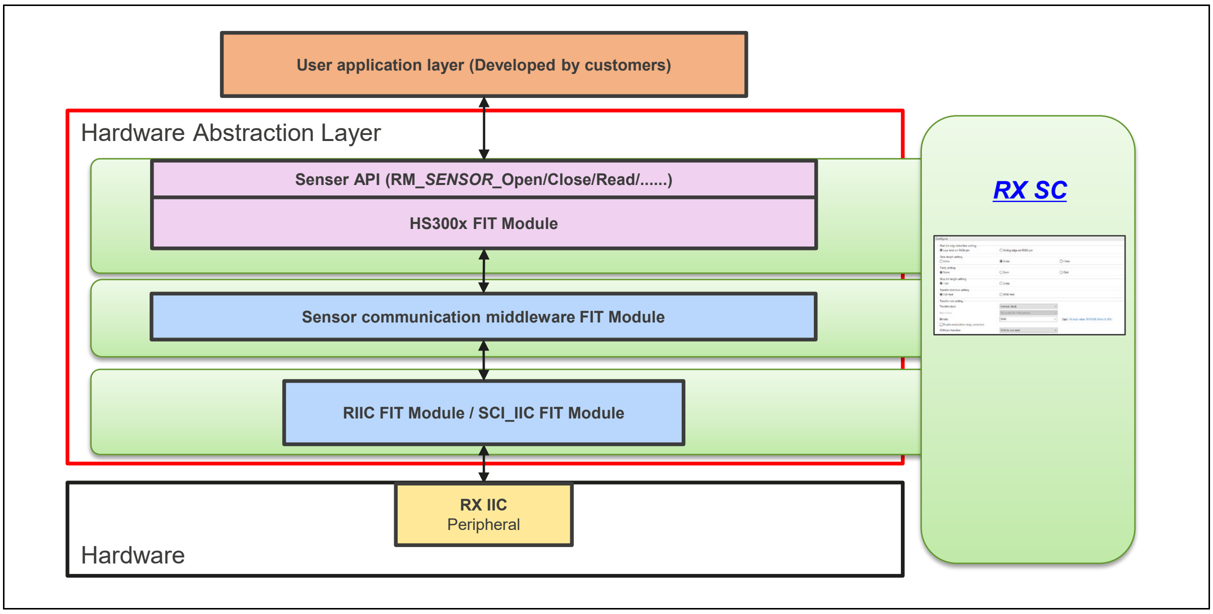

User application layer (Developed by customers)
Hardware Abstraction Layer
Senser API (RM_*SENSOR*_Open/Close/Read/......)
HS300x FIT Module
Sensor communication middleware FIT Module
RIIC FIT Module / SCI_IIC FIT Module
RX IIC
Peripheral
Hardware
RX SC

Figure 1-1 Renesas sensor software architecture

The hardware abstraction layer has three layers, “Sensor API”, “Sensor communication middleware” and “RX IIC FIT module (RIIC FIT Module and SCI_IIC FIT Module).

The sensor APIs of HS300x sensor are provided as “HS300x FIT module”, and the APIs of sensor communication middleware are provided as “Sensor communication middleware FIT module”.

The “HS300x FIT module” provides a method to receive sensor data of the HS300x sensor connected to the I2C bus of RX family MCUs via “Sensor communication middleware FIT module”.

Table 1-1 shows the available Sensors. Table 1-2 shows the available IIC FIT modules.

Table 1-1 Available Sensor

Available Sensors	Reference Datasheet
HS300x High Performance Relative Humidity and Temperature Sensor	HS300x Datasheet (April 22, 2020)

Table 1-2 Available IIC FIT Modules

Available IIC FIT Modules	Reference Application Notes
RIIC FIT Module	I2C Bus Interface (RIIC) Module Using Firmware Integration Technology (R01AN1692)
SCI_IIC FIT Module	Simple I2C Module Using Firmware Integration Technology (R01AN1691)

1.1 Outline of HS300x FIT Module

“Table 1-3 HS300x FIT module API Functions” lists the HS300x FIT module API functions.

Table 1-3 HS300x FIT module API Functions

Function	Description
RM_HS300X_Open ()	This function opens and configures the HS300x FIT module.
RM_HS300X_Close ()	This function disables specified HS300x control block.
RM_HS300X_MeasurementStart ()	This function starts a measurement.
RM_HS300X_Read ()	This function reads ADC data from HS300x sensor.
RM_HS300X_DataCalculate ()	This function calculates humidity [RH] and temperature [Celsius] from ADC data.
rm_hs300x_callback ()	This function is callback function for HS300x FIT module.

1.2 Outline of COMMS (sensor communication middleware) FIT Module

“Table 1-4 Senser communication middleware FIT module API Functions” lists the API functions.

Table 1-4 Senser communication middleware FIT module API Functions

Function	Description
RM_COMMS_I2C_Open ()	The function opens and configures the COMMS FIT module.
RM_COMMS_I2C_Close ()	This function disables specified COMMS FIT module.
RM_COMMS_I2C_Read ()	The function performs a read from I2C device.
RM_COMMS_I2C_Write ()	The function performs a write from the I2C device.
RM_COMMS_I2C_WriteRead ()	The function performs a write to, then a read from the I2C device.

1.3 How to combine sensor control modules and RX IIC FIT modules

HS300x FIT module and COMMS FIT module can control simultaneously control multiple sensors on any channel of any I2C bus.

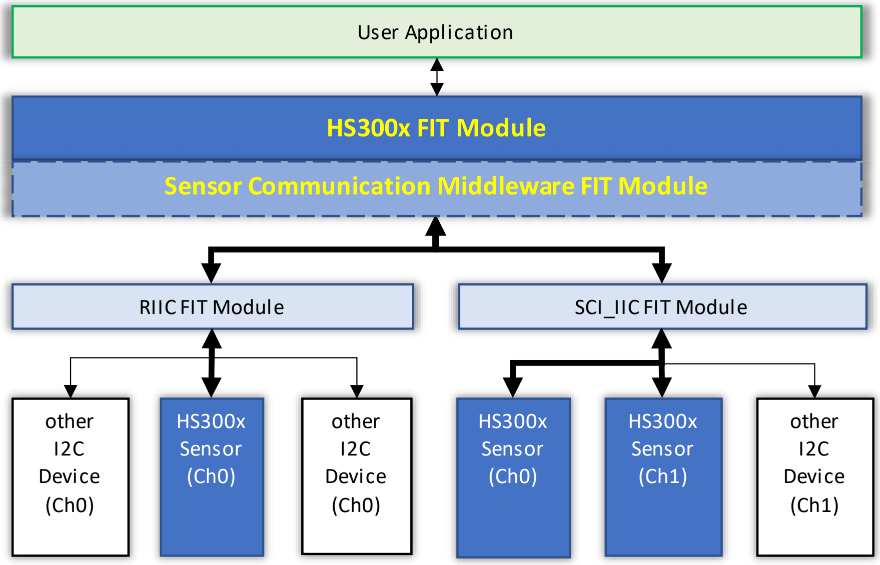
However, the sensors using same slave address cannot be connected to a same channel of I2C bus. Therefore, only one HS300x sensor can be connected to a same channel of the I2C bus.

Figure 1-2 shows the relationship of HS300x FIT module, sensor communication middleware FIT module, RX IIC FIT modules and the I2C devices.

The sensor communication middleware FIT module is a driver interface function layer to absorb the difference between the HS300x FIT module and RX IIC FIT modules.

The initialization processing of these FIT modules opens the module and sets control structure values according to configurations set by user. The initialization of I2C bus need to be done in user application in advanced of above initialization. Depending on sensor connection to IIC bus in user system, the R_RIIC_Open() of RIIC FIT module or R_SCI_IIC_Open() of SCI_IIC FIT module is used for initialization of I2C bus.

For the configuration related to this FIT module, refer to "2.7 Configuration Overview".



Since each I2C bus/channel is configured for each HS300x sensor, multiple HS300x sensors can be controlled simultaneously.

The RIIC FIT module and SCI_IIC FIT module can be controlled simultaneously.

However, since only a slave address is used for HS300x sensor, only one HS300x sensor can be connected on a same channel of the I2C bus.

Figure 1-2 Example of Combination of Sensor (HS300x) FIT Module and IIC FIT Modules

1.4 Terminology/Abbreviation

Table 1-5 Terminology/Abbreviation Lists

Terminology/Abbreviation	Description
HS300x FIT Module	Indicates HS300x Relative Humidity and Temperature Sensor Control Module.
HS300x Sensor	Indicates HS300x Relative Humidity and Temperature Sensor.
Sensor Communication Middleware (COMMS) FIT Module	Indicates communication driver interface function layer module.
I2C Bus Control FIT Module IIC FIT Module	Indicates RIIC FIT Module or/and SCI_IIC FIT Module.
ReST	Repeated Start Condition
SP	Stop Condition
ST	Start Condition

1.5 Operating Test Environment

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

Table 1-6 Operation Test Environment

Item	Contents
Integrated Development Environment	Renesas Electronics e2 studio 2021-04
C Compiler	Renesas Electronics C/C++ compiler for RX family V.3.02.00 Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian Order	Little-endian
Module Version	r_riic_rx Ver.2.46 r_sci_iic_rx Ver.2.46
Board Used	RX65N Envision Kit (RTK5RX65N2C00000BR) PMOD Daughter Card for HS3001 Temperature/humidity sensor (US082-HS3001EVZ) Interposer Board to convert Type2/3 to Type 6A PMOD standard (US082-INTERPEVZ)

1.6 Notes/Restrictions

- The operation by single master control has been confirmed. The operation by multi-master control is unconfirmed. When using it in multi-master control, evaluate it sufficiently.
- Does not support the Programming mode control function to access the Non-volatile Memory of the HS3000x sensor.
- Operation has been confirmed only when the data endian is little endian.
- For the notes and restrictions of the IIC FIT modules, refer to each application note.

2. API Information

2.1 Hardware Requirements

The MCU used must support one or both of the following functions.

- I2C Bus Interface (RIIC)
- Serial Communication Interface (SCI)；Simple I2C bus mode

2.2 Software Requirements

The FIT modules are dependent upon the following packages:

- Board Support Package Module (r_bsp) Ver.5.66 or higher
- RIIC FIT Module (r_riic_rx) Ver.2.46 or higher
- SCI_IIC FIT Module (r_sci_iic_rx) Ver.2.46 or higher

2.3 Supported Toolchains

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

- Renesas RX Toolchain v.3.02.00 or higher

2.4 Usage of Interrupt Vector

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

2.5 Header Files

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

- HS300x FIT Module
  r_hs300x_if.h

  rm_hs300x_api.h

  rm_hs300x.h
- Sensor Communication Middleware FIT Module
  r_comms_i2c_if.h

  rm_comms_api.h

  rm_comms_i2c.h

2.6 Integer Types

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

2.7 Configuration Overview

The configuration options in these FIT modules are specified in r_hs300x_rx_config.h and rm_hs300x_instance.c for HS300x FIT module, r_comms_i2c_rx_config.h and rm_comms_i2c_rx_instance.c.

It is also necessary to set the IIC FIT modules to be used. Refer to each application note for detail information.

2.7.1 HS300x FIT module configuration (r_hs3000_rx_config.h)

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

Configuration options	Description (Smart Configurator display)
RM_HS300X_CFG_PARAM_CHECKING_ENABLE	Specify whether to include code for API parameter checking. Selection: BSP Enabled Disabled Default: BSP
RM_HS300X_CFG_DEVICE_NUM_MAX	Specify maximum numbers of HS300x sensors. Selection: 1-2 Default: 1
RM_HS300X_CFG_DATA_BOTH_HUMIDITY_TEMPERATURE	Specify HS300x sensor data type. Selection: Humidity only Both humidity and temperature Default: Both humidity and temperature
RM_HS300X_CFG_DEVICE0_COMMS_INSTANCE	Specify using communication line instance for device0, g_comms_i2c_bus0_instance is used when “Comms0”. (Note 1) Selection: Comms0-Comms4 Default: Comms0
RM_HS300X_CFG_DEVICE0_CALLBACK_ENABLE	Enable user callback for HS300x Sensor Device0 Selection: Enabled Disabled Default: Disabled
RM_HS300X_CFG_DEVICE0_CALLBACK	Specify user callback function name. Selection: None Default: hs300x_user_callback0 (Need user to input.)
RM_HS300X_CFG_DEVICE1_COMMS_INSTANCE	Specify using communication line instance for device1, g_comms_i2c_bus1_instance is used when “Comms1”. (Note 1) Selection: Comms0-Comms4 Default: Comms0
RM_HS300X_CFG_DEVICE1_CALLBACK_ENABLE	Enable user callback for HS300x Sensor Device0 Selection: Enabled Disabled Default: Disabled
RM_HS300X_CFG_DEVICE1_CALLBACK	Specify user callback function name. Selection: None Default: hs300x_user_callback1 (Need user to input.)

Note 1: Do not set same “Comms(x)” number for sensor device 0 and sensor device 1.

2.7.2 Sensor Communication Middleware FIT Module Configuration (r_comms_i2c_rx_config.h)

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

Configuration	Description (Smart Configurator display)
COMMS_I2C_CFG_PARAM_CHECKING_ENABLE	Specify whether to include code for API parameter checking. Selection: BSP Enabled Disabled Default: BSP
COMMS_I2C_CFG_DEVICE_NUM_MAX	Set the numbers (max.) of I2C devices. Selection: Unuse, 1-5 Default: 1
COMMS_I2C_CFG_RTOS_BLOCKING_SUPPORT_ENABLE	Specify blocking operation of RTOS project. Selection: Enabled Disabled Default: Disabled
COMMS_I2C_CFG_RTOS_BUS_LOCK_SUPPORT_ENABLE	Specify bus locked operation of RTOS project. Selection: Enabled Disabled Default: Disabled
COMMS_I2C_CFG_BUS(x)_DRIVER_TYPE (“x” = 0-4)	Specify the driver type of IIC bus. Selection: Not selected RX FIT RIIC RX FIT SCI IIC Default: Not selected
COMMS_I2C_CFG_BUS(x)_DRIVER_CH (“x” = 0-4)	Specify the channel number of the IIC bus. Selection: None Default: 0 (Need user to input)
COMMS_I2C_CFG_BUS(x)_SLAVE_ADDR (“x” = 0-4)	Specify the slave address of the IIC bus. Selection: None Default: 0x00 (Need user to input)
COMMS_I2C_CFG_BUS(x)_ADDR_MODE (“x” = 0-4)	Specify the slave address mode of the IIC bus. Only support 7bit address mode. Selection: 7 bit address mode Default: 7 bit address mode
COMMS_I2C_CFG_BUS(x)_CALLBACK_ENABLE (“x” = 0-4)	Specify the enable callback function of the IIC bus. Selection: Enabled Disabled Default: Disabled
COMMS_I2C_CFG_BUS(x)_CALLBACK (“x” = 0-4)	Specify Callback function of the IIC bus. Selection: None Default: comms_i2c_user_callbackx (Need user to input)
COMMS_I2C_CFG_BUS(x)_BLOCKING_TIMEOUT (“x” = 0-4)	Specify the blocking timeout of RTOS project. Selection: None Default: 0xFFFFFFFF (Need user to input)
COMMS_I2C_CFG_BUS(x)_TIMEOUT (“x” = 0-4)	Specify the bus timeout of RTOS project. Selection: None Default: 0xFFFFFFFF (Need user to input)

2.8 Code Size

Typical code sizes associated with this FIT 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 compile option default values;

- optimization level: 2,
- optimization type: for size
- 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: r_riic_rx Ver.2.46 and r_sci_iic_rx Ver.2.46
- Compiler Version:
  Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00
  (The option of “-lang = c99” is added to the default settings of the integrated development environment.)
- Configuration Options: Default settings

ROM, RAM and Stack Memory Usage

OS supporting	MCU	Sensor	Category	Numbers
Non	RX656N	HS300x	ROM	438 bytes
			RAM	20 bytes
			Stack	36 bytes
		COMMS	ROM	717 bytes
			RAM	73 bytes
			Stack	60 bytes
FreeRTOS	RX656N	HS300x	ROM	432 bytes
			RAM	20 bytes
			Stack	28 bytes
		COMMS	ROM	952 bytes
			RAM	81 bytes
			Stack	80 bytes

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 HS300x FIT module and COMMS FIT 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 HS300x FIT module and COMMS FIT 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 HS300x FIT Module

(1) Configuration Struct rm_hs300x_cfg_t

This structure is located in “rm_hs300x_api.h” file.

```
/** HS300X Configuration */
typedef struct st_rm_hs300x_cfg
{
    rm_comms_instance_t const * p_instance;   ///< Pointer to Communications Middleware instance.
    void const            * p_context;                ///< Pointer to the user-provided context.
    void const            * p_extend;             ///< Pointer to extended configuration by instance of interface.
    void (* p_callback)(rm_hs300x_callback_args_t * p_args);      ///< Pointer to callback function.
} rm_hs300x_cfg_t;
```

(2) Control Struct rm_hs300x_ctrl_t

This is HS300x FIT module control block and allocates an instance specific control block to pass into the HS300x API calls. This structure is implemented as “rm_hs300x_instance_ctrl_t” located in “rm_hs300x.h” file.

```
/** HS300x Control Block */
typedef struct rm_hs300x_instance_ctrl
{
    uint32_t                        open;              ///< Open flag
    rm_hs300x_cfg_t const     * p_cfg;             ///< Pointer to HS300X Configuration
    rm_comms_instance_t const * p_comms_i2c_instance;    ///< Pointer of I2C Communications
Middleware instance structure
    void const                     * p_context;        ///< Pointer to the user-provided context

    /* Pointer to callback and optional working memory */
    void (* p_callback)(rm_hs300x_callback_args_t * p_args);
} rm_hs300x_instance_ctrl_t;
```

2.9.2 Configuration Structure and Control Structure of COMMS FIT Module

(1) Configuration Struct rm_comms_cfg_t

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

```
/** Communications middleware configuration block */
typedef struct st_rm_comms_cfg
{
    uint32_t            semaphore_timeout;        ///< timeout for callback.
    void (* p_callback)(rm_comms_callback_args_t * p_args);      ///< Pointer to callback function, mostly
used if using non-blocking functionality.
    void const          * p_lower_level_cfg;      ///< Pointer to lower level driver configuration structure.
    void const          * p_extend;               ///< Pointer to extended configuration by instance of
interface.
    void const          * p_context;              ///< Pointer to the user-provided context
} rm_comms_cfg_t;
```

(2) Control Struct rm_comms_ctrl_t

This is COMMS FIT module control block and allocates an instance specific control block to pass into the COMMS API calls. This structure is implemented as “rm_comms_i2c_instance_ctrl_t” located in “rm_comms_i2c.h” file.

```
/** Communications middleware control structure. */
typedef struct st_rm_comms_i2c_instance_ctrl
{
    rm_comms_cfg_t const                * p_cfg;                ///< middleware configuration.
    rm_comms_i2c_bus_extended_cfg_t     * p_bus;                ///< Bus using this device;
    void                                * p_lower_level_cfg;    ///< Used to reconfigure I2C driver
    uint32_t                            open;                   ///< Open flag.
    uint32_t                            transfer_data_bytes;    ///< Size of transfer data.
    uint8_t                             * p_transfer_data;      ///< Pointer to transfer data buffer.

    /* Pointer to callback and optional working memory */
    void (* p_callback)(rm_comms_callback_args_t * p_args);

    void const                          * p_context;            ///< Pointer to the user-provided context
} rm_comms_i2c_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 RX BSP (Board Support Package Module) Ver.5.66 or higher.

```
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            = 0x30000,    ///< Data is invalid.
   FSP_ERR_SENSOR_IN_STABILIZATION        = 0x30001,    ///< Sensor is stabilizing.
   FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED = 0x30002,  ///< Measurement is not finished.

   /* Start of COMMS specific */
   FSP_ERR_COMMS_BUS_NOT_OPEN             = 0x40000,    ///< Bus is not open.
} fsp_err_t;
```

2.11 Adding the FIT 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 (3). However, “Smart Configurator” only supports some RX devices. Please use the methods of (2) or (4) for unsupported RX devices.

(1) Adding the FIT module to your project using “Smart Configurator” in e^2 studio

By using the “Smart Configurator” in e2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using “FIT Configurator” in e^2 studio

By using the “FIT Configurator” in e2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

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

By using the “Smart Configurator Standalone version” in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(4) Adding the FIT module to your project in CS+

In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

If you use Smart Configurator, both RIIC FIT module and SCI_IIC FIT module will be added. Manually remove the unnecessary FIT module.

3. HS300x API Functions

3.1 RM_HS300X_Open

This function opens and configures the HS300x FIT module. This function must be called before calling any other HS300x API functions. The RIIC FIT module or / and SCI_IIC FIT module be used must be initialized in advance.

Format

```
fsp_err_t        RM_HS300X_Open(
        rm_hs300x_ctrl_t * const p_ctrl,
        rm_hs300x_cfg_t const * const p_cfg
);
```

Parameters

p_ctrl

Pointer to control structure.
The members of this structure are shown in 2.9.1(2) Control Struct rm_hs300x_ctrl_t.

p_cfg

Pointer to configuration structure.
The members of this structure are shown in 2.9.1(1) Configuration Struct rm_hs300x_cfg_t

Return Values

FSP_SUCCESS	HS300x 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_hs300x.h

Description

This function opens and configures the HS300x FIT module.

This function copies the contents in “p_cfg” structure to the member “p_ctrl->p_cfg” in “p_cfg” structure.

This function does configurations by setting the members of “p_ctrl” structure as following:

- Sets related instance of COMMS FIT module
- Sets callback and context
- Sets open flag

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

Special Notes

None

3.2 RM_HS300X_Close()

This function disables specified HS300x control block.

Format

fsp_err_t RM_HS300X_Close (rm_hs300x_ctrl_t * const p_ctrl)

Parameters

p_ctrl

Pointer to control structure.
The members of this structure are shown in 2.9.1(2) Control Struct rm_hs300x_ctrl_t.

Return Values

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

Properties

Prototyped in rm_hs300x.h

Description

This function calls close API of COMMS FIT module to close communication middleware.

This function clears open flag after all above are done.

Special Notes

None

3.3 RM_HS300X_MeasurementStart

This function starts a measurement.

Format

fsp_err_t RM_HS300X_MeasurementStart (rm_hs300x_ctrl_t * const p_ctrl)

Parameters

p_ctrl

Pointer to control structure.
The members of this structure are shown in 2.9.1(2) Control Struct rm_hs300x_ctrl_t.

Return Values

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

Properties

Prototyped in rm_hs300x.h

Description

This function sends the slave address to HS300x sensor and start a measurement.

The function should be called when start a measurement and when measurement data is stale data.

The write API of COMMS FIT module is called in this function to send the slave address to HS300x sensor.

Special Notes

None

3.4 RM_HS300X_Read()

This function reads ADC data from HS300x sensor.

Format

```
fsp_err_t RM_HS300X_Read (
        rm_hs300x_ctrl_t * const p_ctrl,
        rm_hs300x_raw_data_t * const p_raw_data
)
```

Parameters

p_ctrl

Pointer to control structure.

The members of this structure are shown in 2.9.1(2) Control Struct rm_hs300x_ctrl_t.

p_raw_data

Pointer to raw data structure for storing the read ADC data from HS300x sensor.

```
/** HS300X raw data */
typedef struct st_rm_hs300x_raw_data
{
    uint8_t humidity[2];            ///< Upper 2 bits of 0th element are data status
    uint8_t temperature[2];           ///< Lower 2 bits of 1st element are mask
} rm_hs300x_raw_data_t;
```

Return Values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options are invalid.
FSP_ERR_NOT_OPEN	Module is not open.

Properties

Prototyped in rm_hs300x.h

Description

This function reads ADC data from HS300x sensor.

The read API of COMMS FIT module is called in this function.

The ADC data read from HS300x sensor is stored in “p_raw_data” structure. The read data length is defined according to GUI configuration setting as 4 bytes (both humidity and temperature) or 2 bytes (humidity only).

Special Notes

None

3.5 RM_HS300X_DataCalculate

This function calculates humidity [RH] and temperature [Celsius] from ADC data.

Format

```
    fsp_err_t RM_HS300X_DataCalculate (
            rm_hs300x_ctrl_t * const     p_ctrl,
            rm_hs300x_raw_data_t * const p_raw_data,
            rm_hs300x_data_t * const     p_hs300x_data
    )
```

Parameters

p_ctrl

Pointer to control structure.

The members of this structure are shown in 2.9.1(2) Control Struct rm_hs300x_ctrl_t.

p_raw_data

Pointer to raw data structure for storing the read ADC data from HS300x sensor.

```
    /** HS300X raw data */
    typedef struct st_rm_hs300x_raw_data
    {
        uint8_t humidity[2];            ///< Upper 2 bits of 0th element are data status
        uint8_t temperature[2];         ///< Lower 2 bits of 1st element are mask
    } rm_hs300x_raw_data_t;
```

p_hs300x_data

Pointer to HS300x sensor measurement results data structure.

Return Values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_SENSOR_INVALID_DATA	Data is invalid.

Properties

Prototyped in rm_hs300x.h

Description

This function calculates the relative humidity value [%RH] and temperature value in degrees Celsius [°C] from the ADC data stored in “p_raw_data” and stores the calculated results to “p_hs300x_data” structure.

The status of raw data is shown in the upper 2 bits of p_raw_data-> humidity[0]. The raw data is invalid (e.g., stale data) if the status bits do not equal “0b00”. This function checks the status calculating. This function will skip calculation if the raw data is invalid.

The calculation method is based on the following formula given in the HS300x Datasheet. The temperature [°C] range is -40 to +125.

$$Humidity\,[\%RH] = \left(\frac{Humidity\,[13:0]}{2^{14}-1}\right) * 100$$

$$Temperature\,[{}^{o}C] = \left(\frac{Temperature\,[15:2]}{2^{14}-1}\right) * 165 - 40$$

The “p_hs300x_data” structure is defined as following.

```
/** HS300X sensor data block */
typedef struct st_rm_hs300x_sensor_data
{
   int16_t integer_part;
   int16_t decimal_part;             ///< To two decimal places
} rm_hs300x_sensor_data_t;

/** HS300X data block */
typedef struct st_rm_hs300x_data
{
   rm_hs300x_sensor_data_t humidity;
   rm_hs300x_sensor_data_t temperature;
} rm_hs300x_data_t;
```

Therefore, user application needs to combine the integer_part and decimal_part to a float number for humidity and temperature usage.

Special Notes

None

3.6 rm_hs300x_callback

This is callback function for HS300x FIT module.

Format

```
void rm_hs300x_callback (rm_comms_callback_args_t * p_args)
```

Parameters

p_args

Pointer to callback parameter definition.

```
/** Communications middleware callback parameter definition */
typedef struct st_rm_comms_callback_args
{
    void const     * p_context;
    rm_comms_event_t event;
} rm_comms_callback_args_t;
```

Return Values

None

Properties

Prototyped in rm_hs300x.h

Description

This callback function is called in COMMS FIT module callback function.

The member “event” in “rm_hs300x_callback_args_t” structure which is a member of “rm_hs300x_cfg_t” structure is set according to COMMS FIT module events status “p_args->event”.

The events of HS300x FIT module are

```
typedef enum e_rm_hs300x_event
{
    RM_HS300X_EVENT_SUCCESS = 0,
    RM_HS300X_EVENT_ERROR,
} rm_hs300x_event_t;
```

And the events of COMMS FIT module are

```
typedef enum e_rm_comms_event
{
    RM_COMMS_EVENT_OPERATION_COMPLETE = 0,
    RM_COMMS_EVENT_ERROR,
} rm_comms_event_t;
```

The “event” of “rm_fsxxxx_callback_args_t” structure is set to “RM_HS300X_EVENT_SUCCESS” when the COMMS FIT module events status is “RM_COMMS_EVENT_OPERATION_COMPLETE” otherwise set to “RM_HS300X_EVENT_ERROR”.

Special Notes

None.

3.7 Usage Example of HS300x FIT Module

```
#include "r_smc_entry.h"
#include "r_hs300x_if.h"
#include "r_comms_i2c_if.h"
#if COMMS_I2C_CFG_DRIVER_I2C
#include "r_riic_rx_if.h"
#endif
#if COMMS_I2C_CFG_DRIVER_SCI_I2C
#include "r_sci_iic_rx_if.h"
#endif

/* Sequence */
typedef enum e_demo_sequence
{
    DEMO_SEQUENCE_1 = (1),
    DEMO_SEQUENCE_2,
    DEMO_SEQUENCE_3,
    DEMO_SEQUENCE_4,
    DEMO_SEQUENCE_5,
    DEMO_SEQUENCE_6,
} demo_sequence_t;

/* Callback status */
typedef enum e_demo_callback_status
{
    DEMO_CALLBACK_STATUS_WAIT = (0),
    DEMO_CALLBACK_STATUS_SUCCESS,
    DEMO_CALLBACK_STATUS_REPEAT,
} demo_callback_status_t;

/* See Developer Assistance in the project */
void g_comms_i2c_bus0_quick_setup(void);
void g_hs300x_sensor0_quick_setup(void);

void        start_demo(void);
static void demo_err(void);

static volatile demo_callback_status_t  gs_demo_callback_status;
static volatile float                   gs_demo_humidity;
static volatile float                   gs_demo_temperature;

void start_demo(void)
{
    fsp_err_t err;
    rm_hs300x_raw_data_t raw_data;
    rm_hs300x_data_t hs300x_data;
    demo_sequence_t sequence = DEMO_SEQUENCE_1;

    /* Open the Bus */
    g_comms_i2c_bus0_quick_setup();

    /* Open HS300X */
    g_hs300x_sensor0_quick_setup();
```

```
while (1)
{
    switch(sequence)
    {
        case DEMO_SEQUENCE_1 :
        {
            /* Clear status */
            gs_demo_callback_status = DEMO_CALLBACK_STATUS_WAIT;

            /* Start the measurement */
            err = g_hs300x_sensor0.p_api->measurementStart(g_hs300x_sensor0.p_ctrl);
            if (FSP_SUCCESS == err)
            {
                sequence = DEMO_SEQUENCE_2;
            }
            else
            {
                demo_err();
            }
        }
        break;

        case DEMO_SEQUENCE_2 :
        {
            switch(gs_demo_callback_status)
            {
                case DEMO_CALLBACK_STATUS_WAIT :
                    break;
                case DEMO_CALLBACK_STATUS_SUCCESS :
                    sequence = DEMO_SEQUENCE_3;
                    break;
                case DEMO_CALLBACK_STATUS_REPEAT :
                    sequence = DEMO_SEQUENCE_1;
                    break;
                default :
                    demo_err();
                    break;
            }
        }
        break;

        case DEMO_SEQUENCE_3 :
        {
            /* Wait 4 seconds. See table 4 on the page 6 of the datasheet. */
            R_BSP_SoftwareDelay(4, BSP_DELAY_SECS);
            sequence = DEMO_SEQUENCE_4;
        }
        break;

        case DEMO_SEQUENCE_4 :
        {
            /* Clear status */
            gs_demo_callback_status = DEMO_CALLBACK_STATUS_WAIT;
            /* Read data */
            err = g_hs300x_sensor0.p_api->read(g_hs300x_sensor0.p_ctrl, &raw_data);
            if (FSP_SUCCESS == err)
            {
                sequence = DEMO_SEQUENCE_5;
            }
            else
            {
                demo_err();
            }
        }
        break;

        case DEMO_SEQUENCE_5 :
        {
            switch(gs_demo_callback_status)
            {
                case DEMO_CALLBACK_STATUS_WAIT :
                    break;
                case DEMO_CALLBACK_STATUS_SUCCESS :
                    sequence = DEMO_SEQUENCE_6;
                    break;
                case DEMO_CALLBACK_STATUS_REPEAT :
                    sequence = DEMO_SEQUENCE_4;
                    break;
                default :
                    demo_err();
                    break;
            }
        }
        break;

        case DEMO_SEQUENCE_6 :
        {
            /* Calculate data */
            err = g_hs300x_sensor0.p_api->dataCalculate(g_hs300x_sensor0.p_ctrl, &raw_data, &hs300x_data);
            if (FSP_SUCCESS == err)
            {
                sequence = DEMO_SEQUENCE_1;

                /* Set data */
                gs_demo_humidity    =
                      (float)hs300x_data.humidity.integer_part + (float)hs300x_data.humidity.decimal_part * 0.01F;
                gs_demo_temperature  =
                      (float)hs300x_data.temperature.integer_part + (float)hs300x_data.temperature.decimal_part * 0.01F;
            }
            else if (FSP_ERR_SENSOR_INVALID_DATA == err)
            {
                sequence = DEMO_SEQUENCE_4;
            }
            else
            {
                demo_err();
```

```
                }
            }
            break;

            default :
                demo_err();
                break;
        }
    }
}

/* Quick setup for g_comms_i2c_bus0. */
void g_comms_i2c_bus0_quick_setup(void)
{
    rm_comms_instance_t                * p_comms_i2c_instance =
                                 (rm_comms_instance_t *)g_hs300x_sensor0.p_cfg->p_instance;
    rm_comms_i2c_bus_extended_cfg_t    * p_comms_i2c_extend   =
                                 (rm_comms_i2c_bus_extended_cfg_t *) p_comms_i2c_instance->p_cfg->p_extend;
    i2c_driver_instance_t              * p_driver_instance    =
                                 (i2c_driver_instance_t *) p_comms_i2c_extend->p_driver_instance;

    /* Open i2c driver */
    if(COMMS_DRIVER_I2C == p_driver_instance->driver_type)
    {
#if COMMS_I2C_CFG_DRIVER_I2C
        riic_return_t i2c_ret;
        riic_info_t * p_i2c_info = (riic_info_t *)p_driver_instance->p_info;

        p_i2c_info->ch_no = (uint8_t) p_driver_instance->driver_channel;
        i2c_ret = R_RIIC_Open(p_i2c_info);
        switch (i2c_ret)
        {
            case RIIC_ERR_LOCK_FUNC:
            case RIIC_ERR_INVALID_CHAN:
            case RIIC_ERR_INVALID_ARG:
            case RIIC_ERR_OTHER:
                demo_err();
                break;
            default:
            break;
        }
#endif
    }
    else if(COMMS_DRIVER_SCI_I2C == p_driver_instance->driver_type)
    {
#if COMMS_I2C_CFG_DRIVER_SCI_I2C
        sci_iic_return_t i2c_ret;
        sci_iic_info_t * p_i2c_info = (sci_iic_info_t *) p_driver_instance->p_info;

        p_i2c_info->ch_no = (uint8_t) p_driver_instance->driver_channel;
        i2c_ret = R_SCI_IIC_Open(p_i2c_info);
        switch (i2c_ret)
        {
            case SCI_IIC_ERR_LOCK_FUNC:
```

```
            case SCI_IIC_ERR_INVALID_CHAN:
            case SCI_IIC_ERR_INVALID_ARG:
            case SCI_IIC_ERR_OTHER:
                demo_err();
                break;
            default:
                break;
        }
#endif
    }
}

void hs300x_callback(rm_hs300x_callback_args_t * p_args)
{
    if (RM_HS300X_EVENT_SUCCESS == p_args->event)
    {
        gs_demo_callback_status = DEMO_CALLBACK_STATUS_SUCCESS;
    }
    else
    {
        gs_demo_callback_status = DEMO_CALLBACK_STATUS_REPEAT;
    }
}

/* Quick setup for g_hs300x_sensor0. */
void g_hs300x_sensor0_quick_setup(void)
{
    fsp_err_t err;

    /* Open HS300X sensor instance, this must be done before calling any HS300X API */
    err = g_hs300x_sensor0.p_api->open(g_hs300x_sensor0.p_ctrl, g_hs300x_sensor0.p_cfg);
    if (FSP_SUCCESS != err)
    {
        demo_err();
    }
}

static void demo_err(void)
{
    while(1)
    {
        // nothing
    }
}
```

4. COMMS (Sensor Communication Middleware) API Functions

4.1 RM_COMMS_I2C_Open()

This function opens and configures the COMMS (sensor communication middleware) FIT module.

Format

```
    fsp_err_t RM_COMMS_I2C_Open (
            rm_comms_ctrl_t * const p_ctrl,
            rm_comms_cfg_t const * const p_cfg
    )
```

Parameters

p_ctrl

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

p_cfg

Pointer to configuration structure.
The members of this structure are shown in 2.9.2(1)Configuration Struct rm_comms_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.
FSP_ERR_COMMS_BUS_NOT_OPEN	: I2C driver is not open.

Properties

Prototyped in rm_comms_i2c.h

Description

This function opens and configures the COMMS FIT module.

This function copies the contents in “p_cfg” structure to the member “p_ctrl->p_cfg” in “p_cfg” structure.

This function does configurations by setting the members of “p_ctrl” structure as following:

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

Special Notes

None

4.2 RM_COMMS_I2C_Close()

This function disables specified COMMS FIT module.

Format

fsp_err_t RM_COMMS_I2C_Close (rm_comms_ctrl_t * const p_ctrl)

Parameters

p_ctrl

Pointer to control structure.
The members of this structure are shown in 2.9.2(2)Control Struct rm_comms_ctrl_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_NOT_OPEN	: Module is not open.

Properties

Prototyped in rm_comms_i2c.h

Description

This function clears current device on bus and open flag.

Special Notes

None

4.3 RM_COMMS_I2C_Read()

This function performs a read from I2C device.

Format

```
fsp_err_t RM_COMMS_I2C_Read (
        rm_comms_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 2.9.2(2)Control Struct rm_comms_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_CHANNEL	: Invalid channel.
FSP_ERR_INVALID_ARGUMENT	: Invalid argument.
FSP_ERR_IN_USE	: Bus is busy.

Properties

Prototyped in rm_comms_i2c.h

Description

This function calls internal function “rm_comms_i2c_bus_read()” to start read operation from I2C bus which is RIIC bus or SCI bus depending on the device (sensor) connection.

The internal function “rm_comms_i2c_bus_read()” does bus re-configuration according to contents in “p_ctrl”. Then it calls “R_RIIC_MasterReceive()” API of RIIC FIT module when the device (sensor) is connected to RIIC bus, calls “R_SCI_IIC_MasterReceive()” API of SCI_IIC FIT module when the device (sensor) is connected to SCI bus.

The receive pattern of “R_RIIC_MasterReceive()” and “R_SCI_IIC_MasterReceive()” is set as master reception. In this pattern, the master (RX MCU) receives data from the slave.

Please refer to following documents for detail of “R_RIIC_MasterReceive()” API and “R_SCI_IIC_MasterReceive()” API:

- RX Family I2C Bus Interface (RIIC) Module Using Firmware Integration Technology (R01AN1692)
- RX Family Simple I2C Module Using Firmware Integration Technology (R01AN1691)

Special Notes

None

4.4 RM_COMMS_I2C_Write()

This function performs a write from the I2C device.

Format

```
fsp_err_t RM_COMMS_I2C_Write (
        rm_comms_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 2.9.2(2)Control Struct rm_comms_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_CHANNEL	: Invalid channel.
FSP_ERR_INVALID_ARGUMENT	: Invalid argument.
FSP_ERR_IN_USE	: Bus is busy.

Properties

Prototyped in rm_comms_i2c.h

Description

This function calls internal function “rm_comms_i2c_bus_write()” to start write operation to I2C bus which is RIIC bus or SCI bus depending on device (sensor) connection.

The internal function “rm_comms_i2c_bus_write()” does bus re-configuration according to contents in “p_ctrl”. Then it calls “R_RIIC_MasterSend()” API of RIIC FIT module when the device (sensor) is connected to RIIC bus, calls “R_SCI_IIC_MasterSend()” API of SCI_IIC FIT module when the device (sensor) is connected to SCI bus.

Please refer to following documents for detail of “R_RIIC_MasterSend()” API and “R_SCI_IIC_MasterSend()” API:

- RX Family I2C Bus Interface (RIIC) Module Using Firmware Integration Technology (R01AN1692)
- RX Family Simple I2C Module Using Firmware Integration Technology (R01AN1691)

Special Notes

None

4.5 RM_COMMS_I2C_WriteRead()

This function performs a write to, then a read from the I2C device.

Format

```
fsp_err_t RM_COMMS_I2C_WriteRead (
        rm_comms_ctrl_t * const           p_ctrl,
        rm_comms_write_read_params_t const write_read_params
)
```

Parameters

p_ctrl

Pointer to control structure.

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

write_read_params

Parameters structure for writeRead API.

```
/** Struct to pack params for writeRead */
typedef struct st_rm_comms_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_write_read_params_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_NOT_OPEN	: Module is not open.
FSP_ERR_INVALID_CHANNEL	: Invalid channel.
FSP_ERR_INVALID_ARGUMENT	: Invalid argument.
FSP_ERR_IN_USE	: Bus is busy.

Properties

Prototyped in rm_comms_i2c.h

Description

This function calls internal function “rm_comms_i2c_bus_write_read ()” to start writing to I2C bus, then reading from I2C bus with re-start. The I2C bus is RIIC bus or SCI bus depending on device (sensor) connection.

The internal function “rm_comms_i2c_bus_write_read ()” does bus re-configuration according to contents in “p_ctrl”. Then it calls “R_RIIC_MasterReceive()” API of RIIC FIT module when the device (sensor) is connected to RIIC bus, calls “R_SCI_IIC_MasterReceive()” API of SCI_IIC FIT module when the device (sensor) is connected to SCI bus. The receive pattern of “R_RIIC_MasterReceive()” and “R_SCI_IIC_MasterReceive()” is set as master transmit/receive. In this pattern, the master (RX MCU) transmits data to the slave. After the transmission completes, a restart condition is generated, and the master receives data from the slave.

Please refer to following documents for detail of “R_RIIC_MasterReceive()” API and “R_SCI_IIC_MasterReceive()” API:

- RX Family I2C Bus Interface (RIIC) Module Using Firmware Integration Technology (R01AN1692)
- RX Family Simple I2C Module Using Firmware Integration Technology (R01AN1691)

Special Notes

None.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	June 30, 2021	-	First Release

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

Corporate Headquarters

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

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/.

Trademarks

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