RX Family

HS300x Relative Humidity and Temperature Sensor Control Module Firmware Integration Technology

Introduction

This application note explains the I2C sensor control module for HS300x relative humidity and temperature sensor manufactured by Renesas Electronics using Firmware Integration Technology (FIT).

This control module acquires the sensor data using the I2C bus control FIT module (IIC FIT Module) and calculate both relative humidity value [%RH] and temperature value in degrees Celsius [°C].

Hereinafter, this control module is abbreviated as HS3000 FIT module.

This HS300x Relative Humidity and Temperature Sensor Control Module has been integrated into the "Sensor Software" shown at the following URL.

Therefore, please use the control program of "Sensor Software".

This application note will be discontinued at the end of December 2021.

https://www.renesas.com/sensor-software

Target Device

- **Sensor:**
  - Renesas Electronics HS300x 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:**
  - RX23W (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

For details of the confirmed operation contents of each compiler, refer to “5.1 Operating Test Environment".

Reference Documents

- Renesas Electronics HS300x Datasheet (April22, 2020)
- RX Family I2C Bus Interface (RIIC) Module Using Firmware Integration Technology (R01AN1692)
- RX Family Simple I2C Module Using Firmware Integration Technology (R01AN1691)

Contents

1. Overview

1.1 Overview of HS300x FIT Module

The HS300x FIT module provides a method to receive data of the HS300x Sensors on the I2C bus of RX family MCUs.

Table 1-1 shows the available HS300x Sensors. Table 1-2 shows the available IIC FIT modules.

Table 1-1 Available HS300x Sensors

Available HS300x Sensors	Reference Datasheet
HS3001 HS3002 HS3003 HS3004	HS300x Datasheet (April22, 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)

1.2 Terminology/Abbreviation

Table 1-3 Terminology/Abbreviation Lists

Terminology/Abbreviation	Description
HS3000 FIT Module	Indicates HS300x Relative Humidity and Temperature Sensor Control Module.
HS300x Sensor	Indicates HS300x Relative Humidity and Temperature Sensor.
I2C Bus Control FIT Module IIC FIT Module	Indicates RIIC FIT Module and SCI_IIC FIT Module.
ReST	Repeated Start Condition
SP	Stop Condition
ST	Start Condition

1.3 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.
- Up to 2 HS300x sensors can be controlled. If you want to control 3 or more, you need to add processing. Refer to "5.3 How to Change When Increasing Number of HS300x Sensors to Be Controlled to 3 or More".
- This FIT module supports up to channel No. 2 for RIIC and up to channel No. 12 for SCI_IIC. If you add a channel No, refer to "5.4 How to Add Channel Number".

For the notes and restrictions of the IIC FIT modules, refer to each application note.

1.4 How to Combine HS3000 FIT Module and IIC FIT Module

This FIT module can control simultaneously control multiple I2C sensors on any channel of any I2C bus.

However, since the HS300x Sensor supports one device address, only one HS300x Sensor can be connected on one channel of the I2C bus.

Figure 1-1 shows the relationship between the HS300x FIT module, the IIC FIT module and the I2C devices.

This FIT module has a Driver I/F function layer to absorb the difference between the IIC FIT modules.

The initialization processing of this FIT module attaches the initialized channel of the IIC FIT module to the HS300x Sensor. Therefore, it is possible to access multiple HS300x Sensors on any channel of any I2C bus.

Before initializing this FIT module, complete the initialization of the IIC FIT module to be used.

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

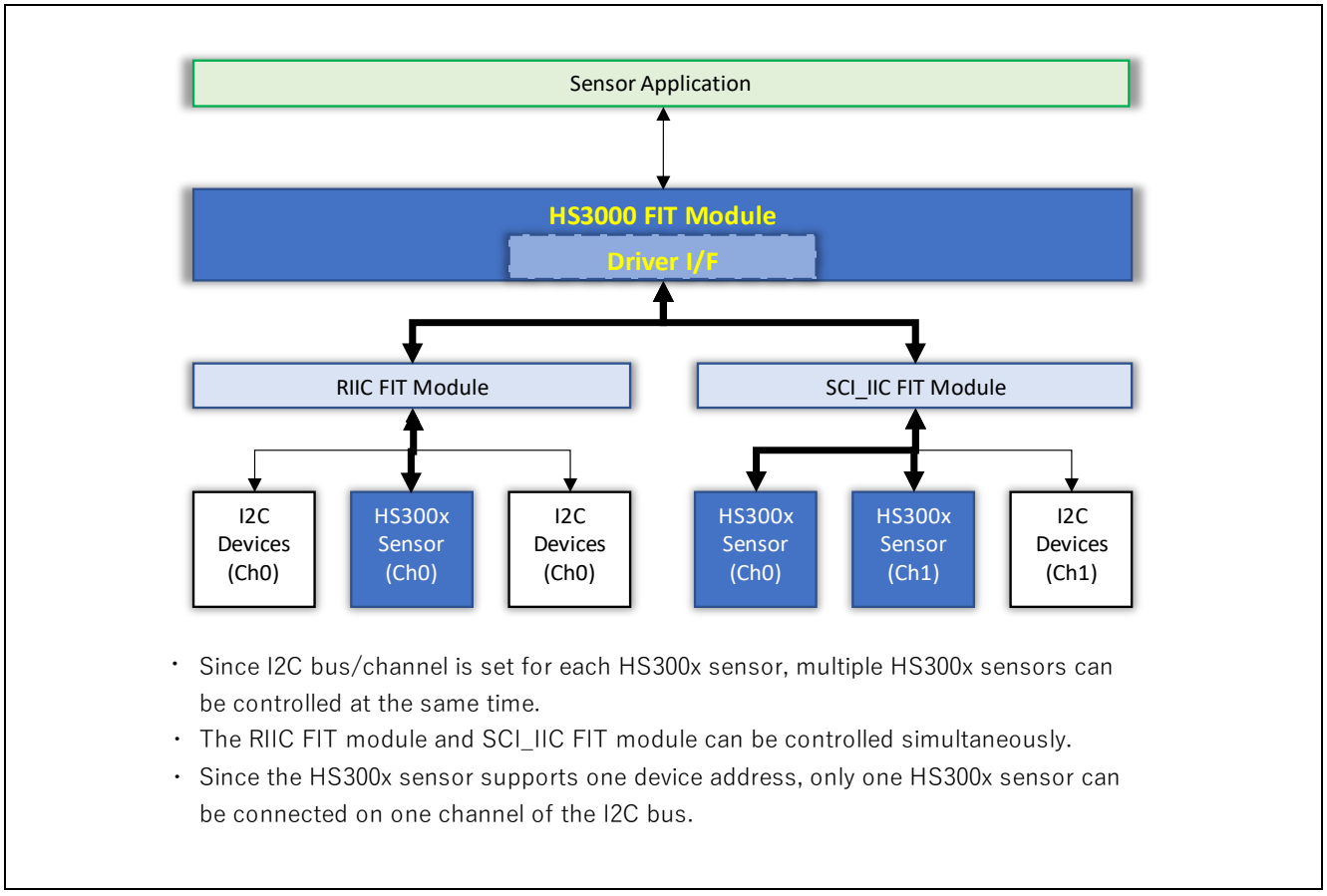


Figure 1-1 Example of Combination of HS3000 FIT module and IIC FIT Module When Controlling Multiple HS300x Sensors

1.5 Outline of the API

Table 1-4 lists the API functions.

Table 1-4 API Functions

Item	Contents
R_HS3000_Open()	The function initializes the HS3000 FIT module. This function must be called before calling any other API functions.
R_HS3000_Close()	This function completes the sensor control and releases the resource to be used.
R_HS3000_Start()	The function starts measuring.
R_HS3000_Get()	The function acquires the measurement data.
R_HS3000_Polling()	The function checks the communication end and the acquisition of the measurement data.
R_HS3000_Calculate()	This function calculates the relative humidity value [%RH] and temperature value in degrees Celsius [°C].
R_HS3000_Control()	The function executed the control command operation.
R_HS3000_GetVersion()	This function returns the current version of this module.

1.6 Recovery Method When Communication Error Occurs

It may be necessary to control error a clearing operation at the I2C bus channel level when a communication error occurs.

When the arbitration lost error generating or NACK response generating, the HS3000 FIT module clears with internal processing.

For errors other than those, use the R_RIIC_Control() function or R_SCI_IIC_Control() function of the IIC FIT module to clear the communication error.

1.7 Bit Rate Setting When Standard Mode I2C Device Is Mixed on Same Bus

The HS300x Sensors support the standard mode and the fast mode.

If the standard mode I2C device is connected on the same channel of the I2C bus, the bit rate must be set to standard mode (up to 100kbps).

1.8 Software State Transition

This FIT module uses API function calls as events, and Software State changes. The Software State transition diagram is shown below. Available API functions vary depending on the Software State.

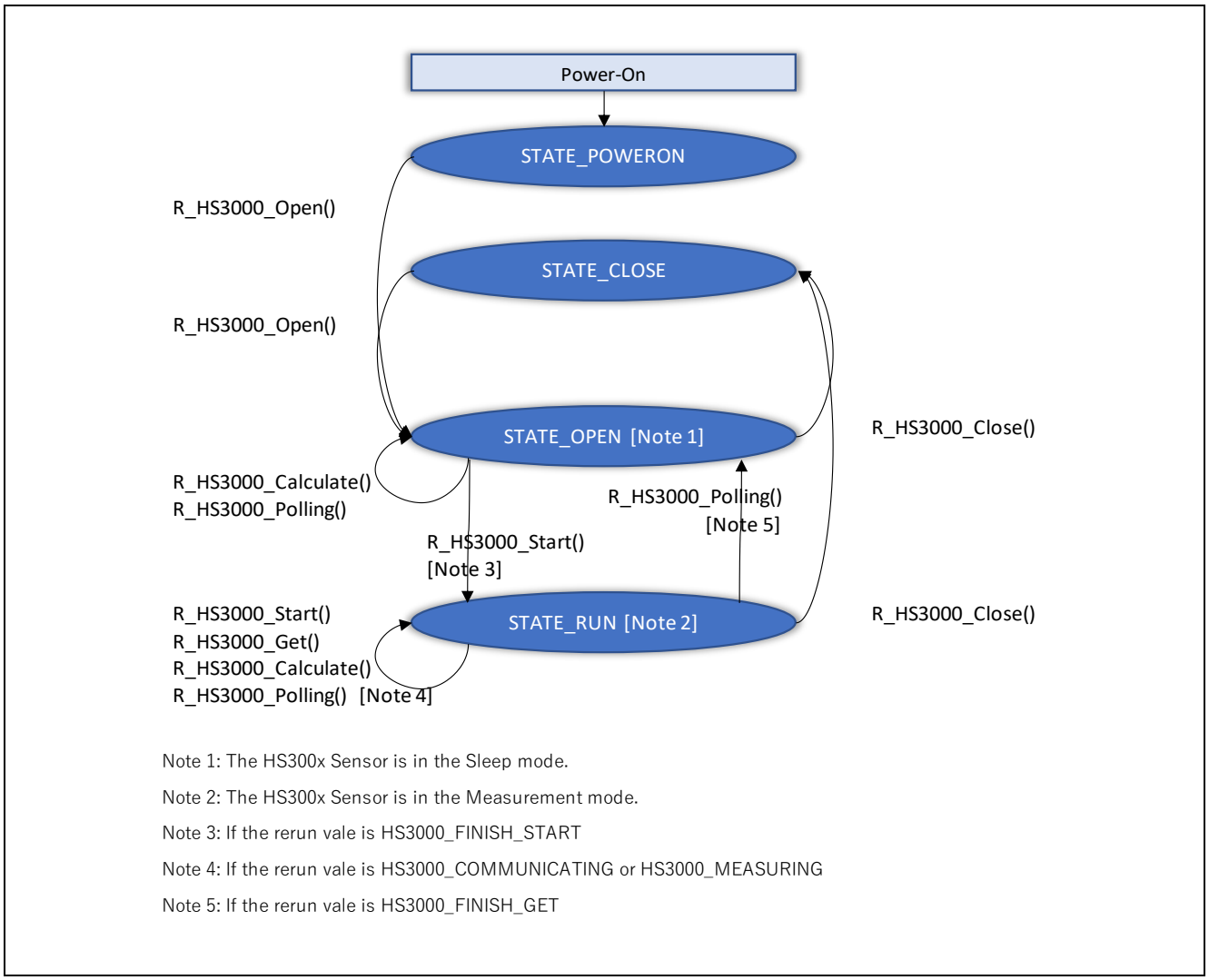


Figure 1-2 Software State Transition of HS3000 FIT Module

1.9 API Call Sequence during Measurement and I2C Bus State

The API call sequence during measurement is shown below.

1. Call R_HS3000_Start() Function.

This function transmits the MR command and starts Measuring.

Return Value	Descriptions
HS3000_FINISH_START	Communication completed state and HS300x sensor measuring state. You can access other devices on the same I2C bus.

2. Call R_HS3000_Get() Function.

This function transmits the DF command and starts acquiring the measurement data.

Return Value	Descriptions
HS3000_ COMMUNICATING	Start Condition (ST) is detected state and communicating state.

3. Call R_HS3000_Polling() Function.

It is possible to checks the acquisition of the measurement data.

Return Value	Descriptions
HS3000_COMMUNICATING	Communicating state
HS3000_MEASURING	Communication completed state and HS300x sensor measuring state. Call the R_HS3000_Get() function again. You can access other devices on the same I2C bus.
HS3000_FINISH_GET	Communication completed state and HS300x sensor measurement completed state You can access other devices on the same I2C bus.

4. Call R_HS3000_Calculate() Function.

This function calculates the relative humidity value [%RH] and temperature value in degrees Celsius [°C].

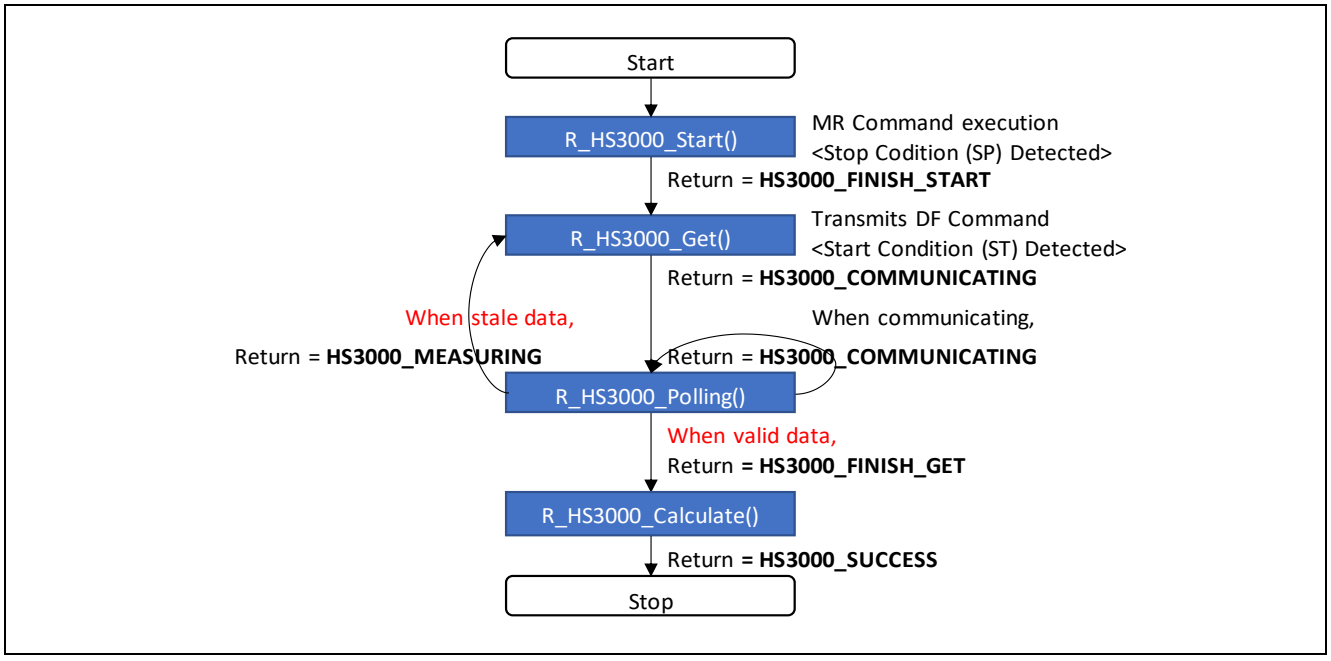
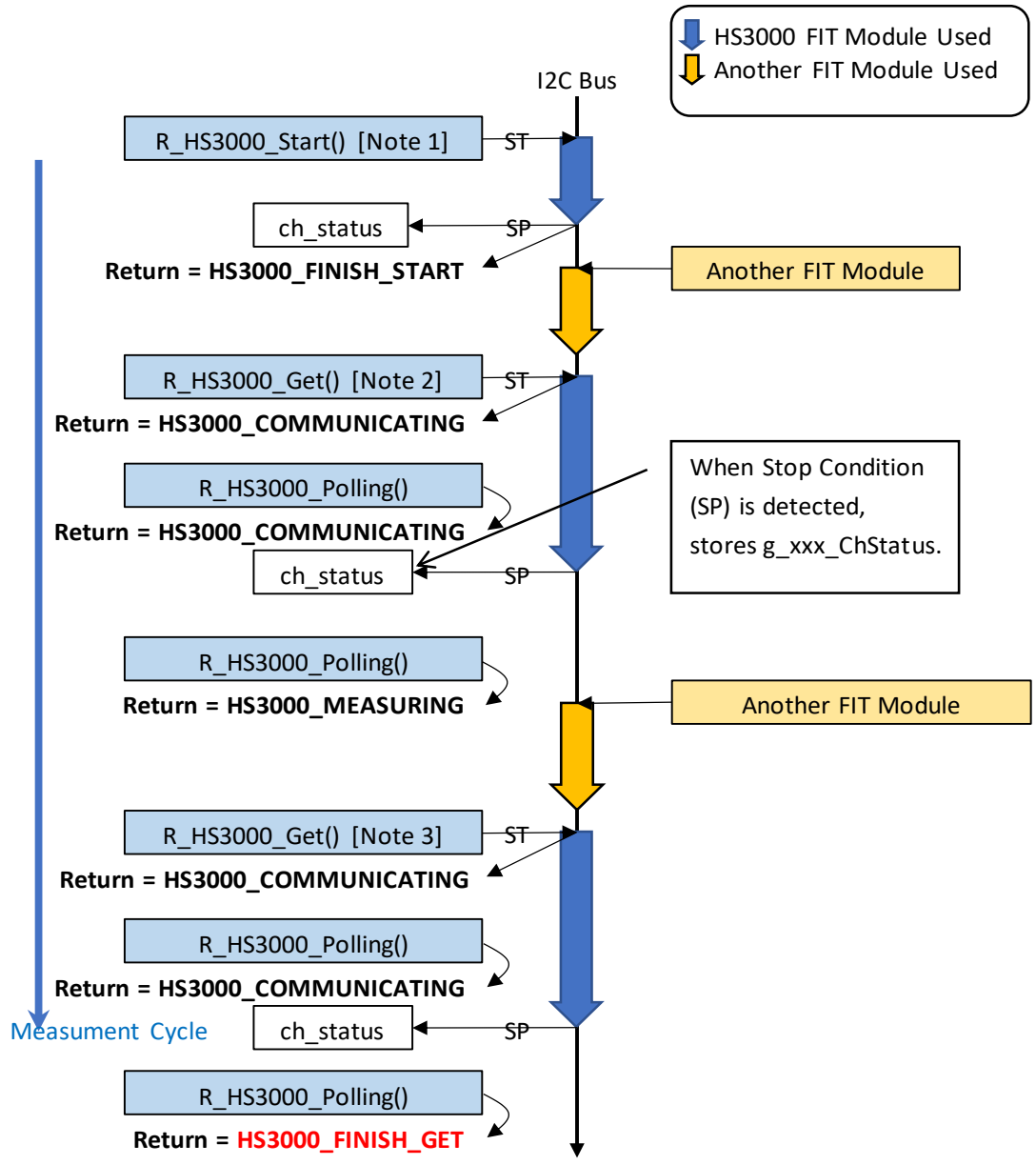


Figure 1-3 API Control Sequence during Measurement

The state of the I2C bus during HS300x Sensor measurement is shown below.

If the return value is HS3000_COMMUNICATING, it shows that communication is in progress. Do not communicate with other devices.

After the communication has finished, it is possible to start the new communication to other devices on the same I2C bus channel.



Note 1: MR command is exectued.

Note 2: DF command is exectued.

Note 3: Because data is stale, DF command is exectued again.

Figure 1-4 API Calling and I2C Bus State Transition during Measurement

2. API Information

2.1 Hardware Requirements

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

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

2.2 Software Requirements

This FIT module is dependent upon the following packages:

- Board Support Package Module (r_bsp) Ver.5.50 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

This FIT module is tested and works with the following toolchain:

- Renesas RX Toolchain v.3.02.00

2.4 Usage of Interrupt Vector

This FIT module does not use interrupts. However, the IIC FIT module to be used uses interrupts. Refer to each application note.

2.5 Header Files

All API calls and their supporting interface definitions are located in r_hs3000_rx_if.h.

2.6 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7 Configuration Overview

The configuration options in this FIT module are specified in r_hs3000_rx_config.h.

It is also necessary to set the IIC FIT module to be used. Refer to each application note.

(1) r_hs3000_rx_config.h

The following explains the option names and setting values of this FIT module.

The default value setting is an example of setting when controlling two sensors simultaneously.

<Default Value Setting Conditions>

- Device No: 0 Connect HS300x Sensor to RIIC channel 0
- Device No: 1 Connect HS300x Sensor to SCI_IIC channel 1

For the settings of the IIC FIT module, refer to "5.2 Compile Configuration of IIC FIT Module".

Configuration Options in r_hs3000_rx_config.h	
HS3000_CFG_PARAM_CHECKING_ENABLE Default value = “(BSP_CFG_PARAM_CHECKING_ENABLE)”	Selectable whether to include parameter checking in the code. - If set to “0”, parameter checking is omitted. - If set to “1”, parameter checking is included.
HS3000_CFG_DEVICE_NUM_MAX Default value = “2”	Set the number of HS300x Sensors controlled by the FIT module
HS3000_CFG_DEVICEx_DRIVER_TYPE x=0- [Note 1] Default value: x=0: “1”, x=1: “2”	Set the driver type (see below) to be used for each Device No. - Use RIIC: “1” - Use SCI_IIC: “2”
HS3000_CFG_DEVICEx_CH x=0- [Note 1,2] Default value: x=0: “0”, x=1: “1”	Set the Channel No. of the I2C bus to which HS300x Sensor is connected for each Device No. - When unused: “0”
HS3000_CFG_DEVICEx_SLAVE_ADDRESS x=0- [Note 1] Default value: x=0: “0x44”, x=1: “0x44”	Set the slave address for each Device No. For HS300x Sensor, only “0x44” is available.
HS3000_CFG_CHANNEL_NUM_MAX_RIIC Default value = 3 [Note 2]	Set the value obtained by adding 1 to the maximum RIIC Channel No. of the MCU to be used.
HS3000_CFG_CHANNEL_NUM_MAX_SCI_IIC Default value = 13 [Note 2]	Set the value obtained by adding 1 to the maximum SCI_IIC Channel No. of the MCU to be used.

Note 1: “x” indicates the Device No. Please set from 0.

Note 2: Used to secure internal variables for the number of channels. Make the following settings. Also, set the channel number and the number of channels as follows. If not used, the setting value will be ignored.

Use RIIC: HS3000_CFG_DEVICEx_CH < HS3000_CFG_CHANNEL_NUM_MAX_RIIC

Use SCI_IIC: HS3000_CFG_DEVICEx_CH < HS3000_CFG_CHANNEL_NUM_MAX_SCI_IIC

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

MCU	Category		With Parameter Checking	With Parameter Checking
RX23W	ROM	RIIC 1 channel and SCI 1 channel used	1,798 bytes	1,610 bytes
	RAM		24 bytes (Handle pointer for 2 devices and channel status for RIIC 3 channels and SCI 13 channels for a total of 16 channels)	
	Stack		140 bytes callback functions: 16 bytes [Note1]	

Note1: The callback function for IIC FIT module provided by this FIT module is shown.

2.9 Parameters

The API function arguments are shown below.

The structure is described in r_hs3000_rx_if.h along with the API function prototype declaration.

The contents of the Sensor Information structure are referenced and updated during communication.

In addition, the Sensor Handle Information structure is managed by this FIT module, so do not rewrite the contents of the structure after setting by open processing.

(1) Device No. device_no

(2) Sensor Information Structure st_hs3000_info_t

```
typedef struct  /* HS3000 Information */
{
    uint8_t     control_cmd;        /**< Control command */
    uint8_t     operation;          /**< Operation */
    uint8_t     data_status;        /**< Data status */
    uint8_t     sw_state;           /**< Software State */
    uint8_t     * p_ch_status;      /**< Pointer to channel status */
    uint16_t    address_cnt;        /**< Size of register address */
    uint16_t    data_cnt;           /**< Size of data */
    uint8_t     * p_access_addr;    /**< Pointer to register address */
    uint8_t     * p_write_data;     /**< Pointer to write data buffer */
    uint8_t     * p_read_data;      /**< Pointer to read data buffer */
    uint16_t    humd_data;          /**< Sensor humidity data */
    uint16_t    temp_data;          /**< Sensor temperature data */
    float       humd_value;         /**< Humidity [%RH] */
    float       temp_value;         /**< Temperature [Degree C] */
} st_hs3000_info_t;                 /* 36-byte */
```

(3) Sensor Handle Information Structure st_hs3000_handle_t

```
typedef struct  /* HS3000 Handle */
{
    uint8_t     rsv0;               /**< reserved */
    uint8_t     driver_type;        /**< IIC driver type */
    uint8_t     ch_no;              /**< IIC channel No. */
    uint8_t     slave_address;      /**< Slave address */
    void        * p_drv_iic_info;   /**< Pointer to IIC Information for IIC FIT module */
    void        * p_drv_callback;   /**< Pointer to callback function for Stop Condition complete
                                         detection */
    void        * p_sensor_info;    /**< Pointer to Sensor Information */
} st_hs3000_handle_t;               /* 16-byte */
```

2.10 Return Values

The API function return values are shown below.

This enumeration is listed in r_hs3000_rx_if.h, along with the prototype declarations for API functions.

```
typedef enum    /* HS3000 API Error Codes */
{
    HS3000_SUCCESS        = 0U,        /**< Successful operation */
    HS3000_ERR_INVALID_ARG,            /**< Invalid parameter */
    HS3000_ERR_INVALID_STATE,          /**< Invalid state */
    HS3000_ERR_IIC_LOCK_FUNC,          /**< IIC: Lock has already been acquired by another task */
    HS3000_ERR_IIC_INVALID_CHAN,       /**< IIC: None existent channel No. */
    HS3000_ERR_IIC_INVALID_ARG,        /**< IIC: Parameter error */
    HS3000_ERR_IIC_UNINIT,             /**< IIC: Uninitialized state */
    HS3000_ERR_IIC_BUS_BUSY,           /**< IIC: Channel is on communication */
    HS3000_ERR_IIC_AL,                 /**< IIC: Arbitration lost error (for RIIC FIT module) */
    HS3000_ERR_IIC_TMO,                /**< IIC: Timeout error (for RIIC FIT module) */
    HS3000_ERR_IIC_NACK,               /**< IIC: NACK detected status */
    HS3000_ERR_IIC_OTHER,              /**< IIC: Other error */
    HS3000_NOP,                        /**< Not operating state */
    HS3000_FINISH_START,               /**< Result: Measurement start state */
    HS3000_COMMUNICATING,              /**< Result: Communicating state */
    HS3000_MEASURING,                  /**< Result: Measuring state */
    HS3000_FINISH_GET,                 /**< Result: Latest data acquisition state */
    HS3000_ERR_OTHER                   /**< Other error */
} e_hs3000_return_t;
```

2.11 Callback Functions for IIC FIT Module

In this FIT module, the callback function for IIC FIT module is set in the I2C communication information structure member ‘callbackfunc’ of the IIC FIT module each time communication functions of IIC FIT module are called.

Therefore, when controlling other devices, set the dedicated callback function in the I2C communication information structure member ‘callbackfunc’ of the IIC FIT module each time communication functions of IIC FIT module are called.

When one of the following conditions is met and the IIC FIT module interrupt request occurs, the callback function is called.

The callback function has the function of acquiring the channel status flag (g_riic_ChStatus[]/g_sci_iic_ChStatus[]) of the IIC FIT module.

- The communication operation is finished and the Stop Condition (SP) is detected.
- A timeout was detected during communication operation. [Note1]

Note1: When using the RIIC FIT module and when the timeout detection function is enabled.
For details, refer to the RIIC FIT module application note.

2.12 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 e2 studio

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

(2) Adding the FIT module to your project using “FIT Configurator” in e2 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 e2 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.

2.13 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described.

Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

while statement example:

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

for statement example:

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

do statement example:

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET));
/* WAIT_LOOP */
```

3. API Functions

3.1 R_HS3000_Open()

The function initializes the HS3000 FIT module. This function must be called before calling any other API functions. Initialize the IIC FIT module to be used in advance.

Format

```
e_hs3000_return_t     R_HS3000_Open(
    uint8_t                   device_no,          /* Device No. */
    st_hs3000_handle_t        * p_hs3000_handle,  /* Structure data */
    st_hs3000_info_t          * p_hs3000_info     /* Structure data */
)
```

Parameters

device_no *Device No.*

** p_hs3000_handle* *Pointer to Sensor Handle Structure (16-byte)*

The members used are shown below.

- driver_type; : No settings required; HS3000_CFG_DEVICEx_DRIVER_TYPE is set.
- ch_no : No settings required; HS3000_CFG_DEVICEx_CH is set.
- slave_address : No settings required; HS3000_CFG_DEVICEx_SLAVE_ADDRESS is set.
- * p_drv_iic_info; : Set the pointer to IIC Information of IIC FIT module to be used.
- * p_drv_callback; : No settings required; The callback function is set by HS3000_CFG_DEVICEx_DRIVER_TYPE and HS3000_CFG_DEVICEx_CH.
- * p_sensor_info; : No settings required; The third argument p_hs3000_info is set.

As this FIT module controls them, do not modify the contents of this structure in the subsequent processing.

** p_hs3000_info* *Pointer to Sensor Information Structure (36-byte)*

The members used are shown below.

- ch_state : No settings required; This is updated.
- operation : No settings required; This is updated.
- sw_state : No settings required; This is updated.

As this FIT module controls it, do not modify the content of this member in the subsequent processing.

Return Values

HS3000_SUCCESS	: Processing finished successfully
HS3000_ERR_INVALID_ARG	: Review the argument.

Properties

Prototyped in r_hs3000_rx_if.h

Description

This function initializes the HS3000 FIT module.

This function sets all members of 'p_hs3000_handle', registers 'p_hs3000_handle' in 'gp_hs3000_handle[device_no]', and attaches the IIC FIT module to be used to 'device_no'.

This function sets the address of 'g_hs3000_ch_status_riic[]/g_hs3000_ch_status_sci_iic[]' for storing the channel status flag ('g_riic_ChStatus[]/g_sci_iic_ChStatus[]') of the IIC FIT module to 'p_hs3000_info->p_ch_status'.

The settings for '* p_ch_status', 'operation', and 'sw_state' of 'p_hs3000_info' when HS3000_SUCCESS is returned are shown below.

Table 3-1 Setting for * p_ch_status, operation, sw_state of p_hs3000_info (R_HS3000_Open())

Return Value	Setting for * p_ch_status, operation, sw_state of p_hs3000_info		
	*** p_ch_status**	**operation**	**sw_state**
HS3000 _SUCCESS	channel state flag value of IIC FIT module	HS3000 _OPERATION_NOP	HS3000 _STATE_OPEN

This function does not include the initialization processing of the IIC FIT module. Complete the initialization of the IIC FIT module in advance. Also, set the pointer of the I2C communication information structure set in the initialization processing of the IIC FIT module to ‘p_drv_iic_info’.

Example

```
#include "platform.h"
#include "r_hs3000_rx_if.h"
#include "r_hs3000_rx_config.h"

#include "r_riic_rx_config.h"
#include "r_riic_rx_if.h"

riic_info_t                 riic_info_ch0;    /* for RIIC Ch0 */

st_hs3000_handle_t          hs3000_handle[HS3000_CFG_DEVICE_NUM_MAX];
st_hs3000_info_t            hs3000_info[HS3000_CFG_DEVICE_NUM_MAX];

void main(void);
void main(void)
{
  volatile riic_return_t    riic_ret;

  e_hs3000_return_t         ret;
  uint8_t                   device_no;

  /* IIC FIT Module Open Operation */
  riic_info_ch0.ch_no       = 0;
  riic_info_ch0.dev_sts     = RIIC_NO_INIT;
  riic_ret = R_RIIC_Open(&riic_info_ch0);
  ...

  /* HS3000 FIT Module Open Operation */
  device_no = 0;
  hs3000_handle.p_drv_iic_info = (void *)&riic_info_ch0;
  ret = R_HS3000_Open(device_no, &hs3000_handle[device_no],
&hs3000_info[device_no]);
  ...

}
```

Special Notes
None

3.2 R_HS3000_Close()

This function completes the sensor control and releases the resource to be used.

Format

```
e_hs3000_return_t    R_HS3000_Close(
    uint8_t                 device_no,          /* Device No. */
    st_hs3000_info_t        * p_hs3000_info     /* Structure data */
)
```

Parameters

device_no *Device No.*

Return Values

HS3000_SUCCESS	: Processing finished successfully
HS3000_ERR_INVALID_ARG	: Review the argument.
HS3000_ERR_INVALID_STATE	: Control according to the Software State transition diagram.

Properties

Prototyped in r_hs3000_rx_if.h

Description

This function completes the sensor control and releases the resource to be used.

This function clears the 'gp_hs3000_handle[device_no]' and detaches the IIC FIT module.

When HS3000_SUCCESS is returned are shown below.

— Set Software State (‘p_hs3000_info->sw_state’) to HS3000_STATE_CLOSE.

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Open operation, see the example of R_HS3000_Open()
  ...

  /* HS3000 FIT Module Close Operation */
  device_no = 0;
  ret = R_HS3000_Close(device_no);
  ...
}
```

Special Notes

None

3.3 R_HS3000_Start()

The function transmits an MR command and starts measurement.

Format

```
e_hs3000_return_t    R_HS3000_Start(
    uint8_t                 device_no,          /* Device No. */
    st_hs3000_info_t        * p_hs3000_info     /* Structure data */
)
```

Parameters

device_no *Device No.*

** p_hs3000_info* *Pointer to Sensor Information Structure (36-byte)*

The members used are shown below.

* p_ch_status	: No settings required; This is updated when ST and SP are detected.
operation	: No settings required; This is updated when ST is detected.
sw_state	: No settings required; This is updated when SP is detected.
address_cnt	: No settings required; This is updated to 0.
data_cnt	: No settings required; This is updated to 0.
* p_access_addr	: Set pointer of 1-byte data.
* p_write_data	: Set pointer of 1-byte data.
* p_read_data	: Set pointer of 4-byte data.

As this FIT module controls them, do not modify the contents of this structure in the subsequent processing.

Return Values

HS3000_FINISH_START	: Call the R_HS3000_Get() function.
HS3000_ERR_INVALID_ARG	: Review the argument and the settings in r_hs3000_rx_config.h.
HS3000_ERR_INVALID_STATE	: Control according to the Software State transition diagram.
HS3000_ERR_IIC_UNINIT	: Initialize the IIC FIT module.
HS3000_ERR_IIC_BUS_BUSY	: Call again because I2C bus communication is in progress.
HS3000_ERR_IIC_AL	: Call again. [Note1]
HS3000_ERR_IIC_TMO	: Clear the error. [Note2]
HS3000_ERR_IIC_NACK	: Check the sensor connection and the slave address. [Note3]
Other than the above	: Unexpected error

Note1: This error occurs only when using the RIIC FIT module. The error has been cleared by calling the R_RIIC_GetStatus () function in the internal processing.

Note2: This error occurs only when using the RIIC FIT module. Refer to the documentation for the RIIC FIT module and clear the error.

Note3: The error has been cleared by calling the R_RIIC_GetStatus () function in the internal processing.

Properties

Prototyped in r_hs3000_rx_if.h

Description

The function transmits the MR command in order to start measurement.

When the return value is HS3000_FINISH_START, the I2C communication is finished successfully. Therefore, it is possible to start the new communication to other devices on the same I2C bus channel.

The settings for '* p_ch_status', 'operation', and 'sw_state' of 'p_hs3000_info' when the return value HS3000_FINISH_START is returned are shown below.

Table 3-2 Setting for * p_ch_status, operation, sw_state of p_hs3000_info (R_HS3000_Start())

Return Value	Setting for * p_ch_status, operation, sw_state of p_hs3000_info		
	*** p_ch_status**	**operation**	**sw_state**
HS3000 _FINISH_START	HS3000 _CH_FINISH	HS3000 _OPERATION_START	HS3000 _STATE_RUN

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Open operation, see the example of R_HS3000_Open()
  ...
  uint8_t               access_addr
  uint8_t               w_buf[1];
  uint8_t               r_buf[4];
  ...

  /* HS3000 FIT Module Start Operation */
  device_no = 0;
  hs3000_info.p_access_addr    = &access_addr;
  hs3000_info.p_write_data     = &w_buf[0];
  hs3000_info.p_read_data      = &r_buf[0];

  ret = R_HS3000_Start(device_no, &hs3000_info[device_no]);
  ...
}
```

Special Notes

For API call sequence, refer to “1.9 API Call Sequence during Measurement and I2C Bus State”.

It is possible to call the R_HS3000_Start() function again even after the R_HS3000_Start() function has finished successfully.

This function uses the Pattern 3 (only slave address transmission) of the Master Transmission of the IIC FIT module.

If an error occurs in the IIC FIT module, HS3000_ERR_IIC_XXX is returned.

Before rewriting the I2C communication information structure of the IIC FIT module, this function checks whether it is in the communication status using the channel status flag (g_riic_ChStatus[]/g_sci_iic_ChStatus[]) of the IIC FIT module.

The callback function for the IIC FIT module is set in the I2C communication information structure member ‘callbackfunc’ of the IIC FIT module each time this function is called. Refer to “2.11 Callback Functions for IIC FIT Module”.

3.4 R_HS3000_Get()

The function starts transmitting the DF command for acquisition of the measurement data.

Format

```
e_hs3000_return_t    R_HS3000_Get(
    uint8_t                  device_no,          /* Device No. */
)
```

Parameters

device_no *Device No.*

Return Values

HS3000_COMMUNICATING : Call the R_HS3000_Polling() function. [Note4]
HS3000_ERR_INVALID_ARG : Review the argument and the settings in r_hs3000_rx_config.h.
HS3000_ERR_INVALID_STATE : Control according to the Software State transition diagram.
HS3000_ERR_IIC_UNINIT : Initialize the IIC FIT module.
HS3000_ERR_IIC_BUS_BUSY : Call again because I2C bus communication is in progress.
HS3000_ERR_IIC_AL : Call again. [Note1]
HS3000_ERR_IIC_TMO : Clear the error. [Note2]
HS3000_ERR_IIC_NACK : Check the sensor connection and the slave address. [Note3]
Other than the above : Unexpected error

Note1: This error occurs only when using the RIIC FIT module. The error has been cleared by calling the R_RIIC_GetStatus () function in the internal processing.
Note2: This error occurs only when using the RIIC FIT module. Refer to the documentation for the RIIC FIT module and clear the error.
Note3: The error has been cleared by calling the R_RIIC_GetStatus () function in the internal processing.
Note4: After the communication has finished, it is possible to start the new communication to other devices on the same I2C bus channel.

Properties

Prototyped in r_hs3000_rx_if.h

Description

The function starts transmitting the DF command for acquisition of the measurement data. It does not indicate the completion of transmission or data acquisition.

When the return value is HS3000_COMMUNICATING, the I2C communication is the communicating state. Therefore, do not communicate with other devices on the same I2C bus channel.

When all communication has finished successfully, the data is acquired from the HS300x Sensor and stored the received data in ‘p_read_data’. However, the update data (latest data) can be acquired after the measurement time of the HS300x Sensor has elapsed.

The received data of the ‘p_read_data’ is shown below.

Table 3-3 Received Data Format of ‘p_read_data’

Received Data Offset	Data Content
0x00	b7-b6：Status bits; In case of 00b, it indicates the updated data (latest data). b5-b0：Humidity data[13:8]
0x01	b7-b0：Humidity data[7:0]
0x02	b7-b0：Temperature data[13:6]
0x03	b7-b2：Temperature data[5:0] b1-b0：Mask data[1:0]

In addition, it stores the acquired data as follows.

Table 3-4 Data Format for data_status, humm_data and temp_data

Members	Data Content
data_status	b7-b6 : Status bits b5-b0 : 000000b When the data_status is 0x00, it indicates the updated data (latest data).
humd_data	b15-b14 : 00b b13-b0 : Humidity data
temp_data	b15-b14 : 00b b13-b0 : Temperature data

It is necessary to convert the acquired data to relative humidity [%RH] and temperature [°C]. Use the R_HS3000_Calculate() function.

The settings for '* p_ch_status', 'operation', and 'sw_state' of 'p_hs3000_info' when the return value HS3000_COMMUNICATING is returned are shown below.

Table 3-5 Setting for * p_ch_status, operation, sw_state of p_hs3000_info (R_HS3000_Get())

Return Value	Setting for * p_ch_status, operation, sw_state of p_hs3000_info		
	*** p_ch_status**	**operation**	**sw_state**
HS3000 _COMMUNICATING	HS3000 _CH_COMMUNICATING	HS3000 _OPERATION_GET	HS3000 _STATE_RUN

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Start operation, see the example of R_HS3000_Start()
  ...
  device_no = 0;
  ret = R_HS3000_Get(device_no);
  ...
}
```

Special Notes

For API call sequence, refer to "1.9 API Call Sequence during Measurement and I2C Bus State".

This function uses the Master Transmission without Repeated Start Condition (ReST) of the IIC FIT module.

If an error occurs in the IIC FIT module, HS3000_ERR_IIC_XXX is returned.

Before rewriting the I2C communication information structure of the IIC FIT module, this function checks whether it is in the communication status using the channel status flag (g_riic_ChStatus[]/g_sci_iic_ChStatus[]) of the IIC FIT module.

This function uses the DF (Data Fetch) command communication that acquires both humidity and temperature data (4-byte in total) from the HS300x Sensor.

For the calculation method of measurement time, refer to the HS300x Datasheet.

The callback function for the IIC FIT module is set in the I2C communication information structure member 'callbackfunc' of the IIC FIT module each time this function is called. Refer to "2.11 Callback Functions for IIC FIT Module".

3.5 R_HS3000_Polling()

The function checks the communication end and the acquisition of the measurement data.

Format

```
e_hs3000_return_t    R_HS3000_Polling(
    uint8_t                  device_no,          /* Device No. */
)
```

Parameters

device_no *Device No.*

Return Values

HS3000_NOP	: Call the R_HS3000_Start() function. [Note6]
HS3000_FINISH_START	: Call the R_HS3000_Get() function.
HS3000_COMMUNICATING	: Call the R_HS3000_Polling() function again. [Note4]
HS3000_MEASURING	: Call the R_HS3000_Get() function again. [Note5]
HS3000_FINISH_GET	: Call the R_HS3000_Calculate() function.
HS3000_ERR_INVALID_ARG	: Review the argument and the settings in r_hs3000_rx_config.h.
HS3000_ERR_INVALID_STATE	: Control according to the Software State transition diagram.
HS3000_ERR_IIC_UNINIT	: Initialize the IIC FIT module.
HS3000_ERR_IIC_BUS_BUSY	: Call again because I2C bus communication is in progress.
HS3000_ERR_IIC_AL	: Call again. [Note1]
HS3000_ERR_IIC_TMO	: Clear the error. [Note2]
HS3000_ERR_IIC_NACK	: Check the sensor connection and the slave address. [Note3]
Other than the above	: Unexpected error

Note1: This error occurs only when using the RIIC FIT module. The error has been cleared by calling the R_RIIC_GetStatus () function in the internal processing.

Note2: This error occurs only when using the RIIC FIT module. Refer to the documentation for the RIIC FIT module and clear the error.

Note3: The error has been cleared by calling the R_RIIC_GetStatus () function in the internal processing.

Note4: After the communication has finished, it is possible to start the new communication to other devices on the same I2C bus channel.

Note5: Since the communication is finished, it is possible to start the new communication to other devices on the same I2C bus channel before calling the R_HS3000_Get() function.

Note6: After the operation fails, this return value may be returned. In that case, the error will be returned when calling R_HS3000_Start() function.

Properties

Prototyped in r_hs3000_rx_if.h

Description

The communication end and the measurement result are judged from the latest information of ‘p_ch_status’, ‘operation’, ‘sw_state’ of the ‘p_hs3000_info’ member set at the time of calling the R_HS3000_Start() function. The communication processing is not included.

When the return value is HS3000_COMMUNICATING, the I2C communication is the communicating state. Therefore, do not communicate with other devices on the same I2C bus channel.

When the return value is HS3000_MEASURING, the I2C communication has finished successfully but the data are stale. Therefore, call the R_HS3000_Get() function again.

When the return value is HS3000_FINISH_GET, the I2C communication has finished successfully and the data are the latest.

In addition, the return value HS3000_NOP is returned after the initialization is completed or the latest measurement data was acquired. After the R_HS3000_Get() function has finished successfully, the return value HS3000_FINISH_START is returned.

Note that the '* p_ch_status' is updated when Stop Condition (SP) is detected.

The settings for '* p_ch_status', 'operation', and 'sw_state' of 'p_hs3000_info' when the return value HS3000_COMMINICATING, HS3000_MEASURING or HS3000_FINISH_GET is returned are shown below.

Table 3-6 Setting for * p_ch_status, operation, sw_state of p_hs3000_info (R_HS3000_Polling())

Return Value	Setting for * p_ch_status, operation, sw_state of p_hs3000_info		
	*** p_ch_status**	**operation**	**sw_state**
HS3000 _COMMUNICATING	HS3000 _CH_COMMUNICATING	No change (HS3000 _OPERATION_GET)	No change (HS3000 _STATE_RUN)
HS3000 _MEASURING	HS3000_CH_FINISH		
HS3000 _FINISH_GET		HS3000 _OPERATION_NOP	HS3000 _STATE_OPEN

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Start operation, see the example of R_HS3000_Start()
  ...
  device_no = 0;
  ret = R_HS3000_Polling(device_no);
  ...
}
```

Special Notes

For API call sequence, refer to "1.9 API Call Sequence during Measurement and I2C Bus State".

This function uses the Data Fetch communication to acquire both humidity and temperature data (4 bytes in total) from the HS300x Sensor.

Refer to the HS300x Datasheet for how to calculate the measurement time.

This function acquires the channel status during communication with the specified device and during measurement. After that, the information is retained until the next communication of the specified device starts.

3.6 R_HS3000_Calculate()

This function calculates the relative humidity value [%RH] and temperature value in degrees Celsius [°C].

Format

```
e_hs3000_return_t   R_HS3000_Calculate(
    uint8_t                 device_no,          /* Device No. */
)
```

Parameters

device_no *Device No.*

Return Values

HS3000_SUCCESS	: Processing finished successfully
HS3000_ERR_INVALID_ARG	: Review the argument.

Properties

Prototyped in r_hs3000_rx_if.h

Description

This function calculates the relative humidity value [%RH] and temperature value in degrees Celsius [°C] from the acquired data and stores to ‘humd_value’ and ‘temp_value’ of the ‘p_hs3000_info’ member set at the time of calling the R_HS3000_Start() function.

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

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

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

This function converts without depending on the status of data status (‘data_status’). Call this function when the result of data acquisition is valid data status.

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Start operation, see the example of R_HS3000_Start()
  ...
  device_no = 0;
  ret = R_HS3000_Calculate(device_no);
  ...
}
```

Special Notes

None

3.7 R_HS3000_Control()

This function executes various controls by commands.

Format

```
e_hs3000_return_t    R_HS3000_Control(
    uint8_t                 device_no,          /* Device No. */
    st_hs3000_info_t        * p_hs3000_info     /* Structure data */
)
```

Parameters

device_no *Device No.*

** p_hs3000_info* *Pointer to Sensor Information Structure (36-byte)*

The members used are shown below.

control_cmd : Set a control command.

Members used other than the above depend on the control command. Please refer to the details of the control command after “3.7.1”.

Return Values

It depends on the control command. Refer to the details of the control command after “3.7.1”.

Properties

Prototyped in r_hs3000_rx_if.h

Description

Various controls can be performed using control commands.

The control commands and control contents are shown below.

Table 3-7 Control Commands and Control Contents

control_cmd	Control Contents
HS3000_CTRL_CMD _SEND_SLAVE_ADDRESS	Transmits only the Slave address to check the connection of the device.
HS3000_CTRL_CMD _IIC_GET_STATUS	Calls R_RIIC_GetStatus()/R_SCI_IIC_GetStatus() function on the IIC FIT module.

Special Notes

None

3.7.1 control_cmd = HS3000_CTRL_CMD_SEND_SLAVE_ADDRESS

Parameters

** p_hs3000_info* *Pointer to Sensor Information Structure (36-byte)*

The members used are shown below.

control_cmd	: Set the HS3000_CTRL_CMD_SEND_SLAVE_ADDRESS.
* p_ch_status	: No settings required; This is updated when ST and SP are detected.
operation	: No settings required; This is updated when ST is detected.
sw_state	: No settings required; This is updated when SP is detected.
address_cnt	: No settings required; This is updated to 0.
data_cnt	: No settings required; This is updated to 0.
* p_access_addr	: Set pointer of 1-byte data.
* p_write_data	: Set pointer of 1-byte data.

Return Values

HS3000_SUCCESS	: ACK Received
HS3000_ERR_IIC_NACK	: NACK Received
Other than the above	: Unexpected error

Description

This control command transmits the slave address only.

If the ‘sw_state’ is HS3000_STATE_OPEN or HS3000_STATE_RUN, only the slave address will be transmitted. Use for connection confirmation only.

Transmitting only the slave address is the same process as sending the MR command for the HS300x Sensor. When HS3000_SUCCESS is returned, the HS300x sensor will be in the measurement start state.

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Open operation, see the example of R_HS3000_Open()
  ...
  uint8_t               access_addr
  uint8_t               w_buf[1];
  ...

  /* HS3000 FIT Module Control Operation */
  device_no = 0;
  hs3000_info.p_access_addr    = &access_addr;
  hs3000_info.p_write_data     = &w_buf[0];
  hs3000_info.control_cmd      = HS3000_CTRL_CMD_SEND_SLAVE_ADDRESS;

  ret = R_HS3000_Control(device_no, &hs3000_info[device_no]);
  ...
}
```

Special Notes

This function uses the Pattern 3 (only slave address transmission) of the Master Transmission of the IIC FIT module.

Before rewriting the I2C communication information structure of the IIC FIT module, this function checks whether it is in the communication status using the channel status flag (g_riic_ChStatus[]/g_sci_iic_ChStatus[]) of the IIC FIT module.

3.7.2 control_cmd = HS3000_CTRL_CMD_IIC_GET_STATUS

Parameters

** p_hs3000_info* *Pointer to Sensor Information Structure (36-byte)*

The members used are shown below.

control_cmd : Set the HS3000_CTRL_CMD_IIC_GET_STATUS.
* p_read_data : Set pointer of 4-byte data.

Return Values

HS3000_SUCCESS	: Successful operation
HS3000_ERR_INVALID_ARG	: Review the argument and the settings in r_hs3000_rx_config.h.
HS3000_ERR_INVALID_STATE	: Control according to the Software State transition diagram.
HS3000_ERR_IIC_BUS_BUSY	: Call again because I2C bus communication is in progress.
Other than the above	: Unexpected error

Description

This control command calls the R_RIIC_GetStatus() or R_SCI_IIC_GetStatus() function.

When it has finished successfully, the data is stored the status flag in ‘p_read_data’.

The status flag is stored as follows.

Table 3-8 Status Flag Format of p_read_data

Status Flag Offset	Data Contents
0x00	b7-b0: Acquired status flag b7-b0
0x01	b7-b0: Acquired status flag b15-b8
0x02	b7-b0: Acquired status flag b23-b16
0x03	b7-b0: Acquired status flag b31-b24

Example

```
As for #include and variables, see the example of R_HS3000_Open()

void main(void)
{
  As for Open operation, see the example of R_HS3000_Open()
  ...
  uint8_t               access_addr
  uint8_t               r_buf[4];
  ...

  /* HS3000 FIT Module Control Operation */
  device_no = 0;
  hs3000_info.p_access_addr    = &access_addr;
  hs3000_info.p_read_data      = &r_buf[0];
  hs3000_info.control_cmd      = HS3000_CTRL_CMD_IIC_GET_STATUS;

  ret = R_HS3000_Control(device_no, &hs3000_info[device_no]);
  ...
}
```

Special Notes

Before rewriting the I2C communication information structure of the IIC FIT module, this function checks whether it is in the communication status using the channel status flag (g_riic_ChStatus[]/g_sci_iic_ChStatus[]) of the IIC FIT module.

3.8 R_HS3000_GetVersion()

This function returns the current version of this module.

Format

```
uint32_t        R_HS3000_GetVersion(
        void
)
```

Parameters

None

Return Values

Version number

Properties

Prototyped in r_hs3000_rx_if.h

Description

This function will return the version of the currently installed RIIC FIT module.

The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

For example, Version 4.25 would be returned as 0x00040019.

Example

```
uint32_t version;
version = R_HS3000_GetVersion();
```

Special Notes

It is possible to execute regardless of the Software State.

4. Demo Projects

The demo projects whose operation has been confirmed when controlling the HS3001 using RX23W is attached.

Demo projects include function main() that utilizes the module and its dependent modules (e.g. r_bsp).

It explains about GUI operation when you use e2 studio.

4.1 Project for RX23W

(1) Description

This is a sample code to acquire temperature and humidity data from the HS3001 sensor by single master control using Target Board for RX23W.

The configuration option settings are the initial values of "2.7 Configuration Overview".

You can check the operation by selecting either device 0 or device 1.

Please modify the value of ‘g_test_device_no’ on line 44 in the HS300_Test.c file to select the target device.

In this sample code, it supports either of the following:

- HS3000_CFG_DEVICE0_DRIVER_TYPE is RIIC
- HS3000_CFG_DEVICE1_DRIVER_TYPE is SCI_IIC.

If you change the driver type, you cannot check the operation. You can change the channel No.

(2) Connection between Board and HS3001

The RIIC setting uses the 1-pin and 2-pin for RIIC Ch0 of J5 connector.

The SCI_IIC setting is a setting that uses the 2-pin and 3-pin for SCI_IIC Ch1 of CN2 connector.

(3) Setup and Execution

1. Compile and download the sample code.

   The processing flow of the sample code is shown below. Repeat the measurement.

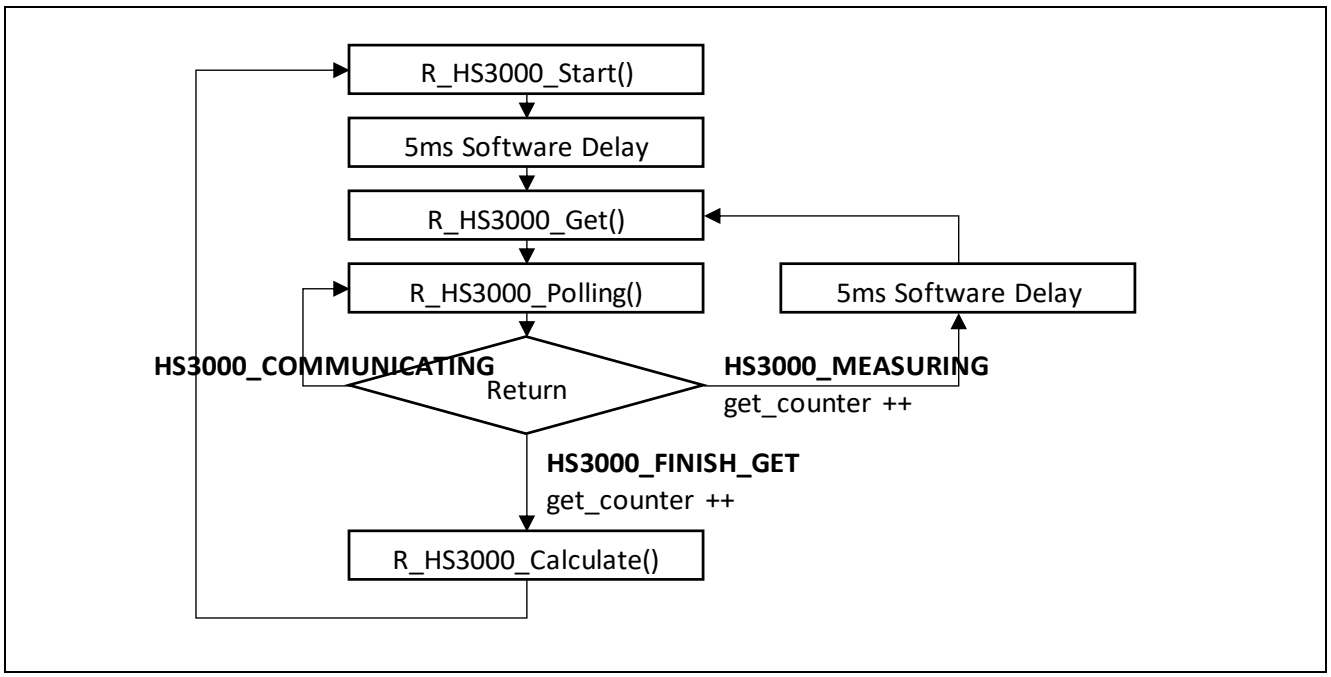


Figure 4-1 Processing Flow of Sample Code

2. Click ‘Reset Go’ to start the software. If PC stops at Main, press F8 to resume.
3. The following log is displayed in "Renesas Debug Virtual Console [Note]".

   The ‘Get Counter’ value indicates the number of calling the R_HS3000_Get() function until the return value HS3000_FINISH_GET is returned.

   In addition, on the display, the log is output with the unit of temperature in degrees Celsius as [*C].

Note: You can display by ‘Renesas Views’-> ‘Debug’-> ‘Renesas Debug Virtual Console’.

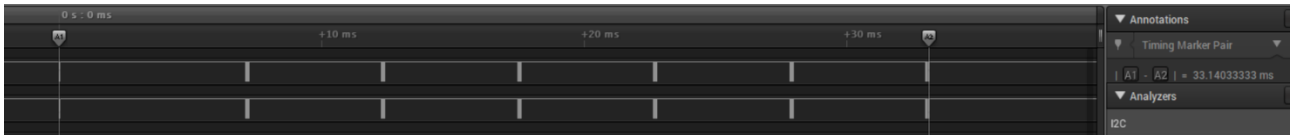
```
HS3000 FIT Module Version VX.XX
RIIC FIT Module Version VX.XX
SCI_IIC FIT Module Version VX.XX
Open(): OK
Start(): OK,
Get(): OK, Get Counter: 6
 Humidity: 49.31 [%RH]
 Temperature: 23.72 [*C]

Start(): OK,
Get(): OK, Get Counter: 6
 Humidity: 49.23 [%RH]
 Temperature: 23.74 [*C]
```

Figure 4-2 Example of Log Output during Demo Operation

4. I2C Bus Measurement Wave from Measurement Start to Measurement End

   The overall waveform of the I2C bus from the start of measurement to the end of measurement in the above demo operation is shown below.

Measurement Condition: RIIC, Bitrate = 400kbps, ICLK = 32MHz, PCLKB = 32MHz



Start() Get() Get() Get() Get() Get() Get()

Figure 4-3 I2C Bus Measurement Wave from Measurement Start to Measurement End

5. Appendices

5.1 Operating Test Environment

This section describes for detailed the operating test environments of this module.

Table 5-1 Operation Test Environment

Item	Contents
Integrated Development Environment	Renesas Electronics e2 studio 2020-07
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	Target Board for RX23W (RTK5RX23W0C00000BJ)

5.2 Compile Configuration of IIC FIT Module

The compile configuration of each IIC FIT module at the operation test are shown below.

(1) r_riic_rx

Configurations	
Set parameter checking enable	Include
MCU supported channels for CH0	Supported
MCU supported channels for CH1	Not supported
MCU supported channels for CH2	Not supported
CH0 RIIC bps(kbps)	400
CH1 RIIC bps(kbps)	400
CH2 RIIC bps(kbps)	400
Digital filter for CH0	Two IIC phi
Digital filter for CH1	Two IIC phi
Digital filter for CH2	Two IIC phi
Setting port setting processing	Include port setting
Master arbitration lost detection function for CH0	Used
Master arbitration lost detection function for CH1	Used
Master arbitration lost detection function for CH2	Used
Address 0 format for CH0	7 bit address format
Address 1 format for CH0	Not
Address 2 format for CH0	Not
Slave address 0 for CH0	0x0025
Slave address 1 for CH0	0x0000
Slave address 2 for CH0	0x0000
Address 0 format for CH1	7 bit address format
Address 1 format for CH1	Not
Address 2 format for CH1	Not
Slave address 0 for CH1	0x0025
Slave address 1 for CH1	0x0000
Slave address 2 for CH1	0x0000
Address 0 format for CH2	7 bit address format
Address 1 format for CH2	Not
Address 2 format for CH2	Not
Slave address 0 for CH2	0x0025
Slave address 1 for CH2	0x0000
Slave address 2 for CH2	0x0000
General call address for CH0	Unused
General call address for CH1	Unused
General call address for CH2	Unused
CH0 RXI INT Priority Level	Level 1
CH0 TXI INT Priority Level	Level 1
CH0 EEI INT Priority Level	Level 1
CH0 TEI INT Priority Level	Level 1
CH1 RXI INT Priority Level	Level 1
CH1 TXI INT Priority Level	Level 1
CH1 EEI INT Priority Level	Level 1
CH1 TEI INT Priority Level	Level 1
CH2 RXI INT Priority Level	Level 1
CH2 TXI INT Priority Level	Level 1
CH2 EEI INT Priority Level	Level 1
CH2 TEI INT Priority Level	Level 1
Timeout function for CH0	Used
Timeout function for CH1	Used
Timeout function for CH2	Used
Timeout detection time for CH0	Long mode
Timeout detection time for CH1	Long mode
Timeout detection time for CH2	Long mode
Count up during low period of timeout detection	Used
Count up during low period of timeout detection	Used
Count up during low period of timeout detection	Used
Count up during high period of timeout detectior	Used
Count up during high period of timeout detectior	Used
Count up during high period of timeout detectior	Used
Set Counter of checking bus busy	1000
Resources	
RIIC	
RIIC0	☑
SCL Pin	☑ Used
SDA Pin	☑ Used

Figure 5-1 Smart Configurator settings (r_riic_rx)

(2) r_sci_iic_rx

Property	Value
Configurations	
Set parameter checking enable	Include
MCU supported channels for CH0	Not supported
MCU supported channels for CH1	Supported
MCU supported channels for CH2	Not supported
MCU supported channels for CH3	Not supported
MCU supported channels for CH4	Not supported
MCU supported channels for CH5	Not supported
MCU supported channels for CH6	Not supported
MCU supported channels for CH7	Not supported
MCU supported channels for CH8	Not supported
MCU supported channels for CH9	Not supported
MCU supported channels for CH10	Not supported
MCU supported channels for CH11	Not supported
MCU supported channels for CH12	Not supported
SCI IIC bitrate (bps) for CH0	384000
SCI IIC bitrate (bps) for CH1	384000
SCI IIC bitrate (bps) for CH2	384000
SCI IIC bitrate (bps) for CH3	384000
SCI IIC bitrate (bps) for CH4	384000
SCI IIC bitrate (bps) for CH5	384000
SCI IIC bitrate (bps) for CH6	384000
SCI IIC bitrate (bps) for CH7	384000
SCI IIC bitrate (bps) for CH8	384000
SCI IIC bitrate (bps) for CH9	384000
SCI IIC bitrate (bps) for CH10	384000
SCI IIC bitrate (bps) for CH11	384000
SCI IIC bitrate (bps) for CH12	384000
Interrupt Priority for CH0	Level 2
Interrupt Priority for CH1	Level 2
Interrupt Priority for CH2	Level 2
Interrupt Priority for CH3	Level 2
Interrupt Priority for CH4	Level 2
Interrupt Priority for CH5	Level 2
Interrupt Priority for CH6	Level 2
Interrupt Priority for CH7	Level 2
Interrupt Priority for CH8	Level 2
Interrupt Priority for CH9	Level 2
Interrupt Priority for CH10	Level 2
Interrupt Priority for CH11	Level 2
Interrupt Priority for CH12	Level 2
Digital noise filter (NFEN bit) for CH0	Enable
Digital noise filter (NFEN bit) for CH1	Enable
Digital noise filter (NFEN bit) for CH2	Enable
Digital noise filter (NFEN bit) for CH3	Enable
Digital noise filter (NFEN bit) for CH4	Enable
Digital noise filter (NFEN bit) for CH5	Enable
Digital noise filter (NFEN bit) for CH6	Enable
Digital noise filter (NFEN bit) for CH7	Enable
Digital noise filter (NFEN bit) for CH8	Enable
Digital noise filter (NFEN bit) for CH9	Enable
Digital noise filter (NFEN bit) for CH10	Enable
Digital noise filter (NFEN bit) for CH11	Enable
Digital noise filter (NFEN bit) for CH12	Enable
Noise Filter Setting Register (NFCS bit) for CH1	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH2	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH3	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH4	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH5	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH6	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH7	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH8	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH9	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH10	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH11	The clock divided by 1
Noise Filter Setting Register (NFCS bit) for CH12	The clock divided by 1

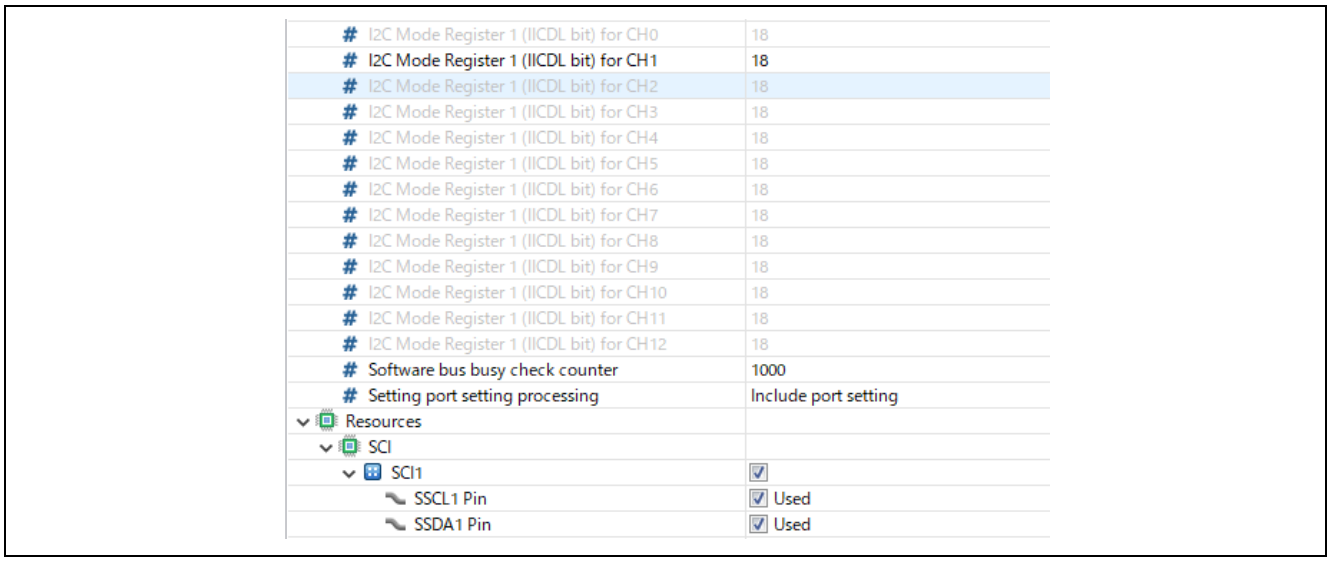


Figure 5-2 Smart Configurator settings (r_sci_iic_rx)

5.3 How to Change When Increasing Number of HS300x Sensors to Be Controlled to 3 or More

The parts that need to be changed are shown below. [ADD_DEVICE] is described on the source as a keyword when adding a device.

(1) r_hs3000_rx_config.h

Replace “N” and “n” with the specified number and add the definition.

```
/* For Device No.N */
/* #define HS3000_CFG_DEVICEn_DRIVER_TYPE        (2)     */   /* 1: RIIC, 2: SCI_IIC */
/* #define HS3000_CFG_DEVICEn_CH                 (5)     */   /* Channel No. */
/* #define HS3000_CFG_DEVICEn_SLAVE_ADDRESS      (0x44)  */   /* 0x88(Write)/0x89(Read) */
```

Figure 5-3 Description on Source When Adding Device (r_hs3000_rx_config.h)

(2) r_hs3000_set_handle() Function

Replace “n” with the specified number and add the code.

```
    /* Add when device_no is missing [Add Device] */
  /*  case n: */  /* Device No.n */
  /*      retval = r_hs3000_drvif_check_channel(HS3000_CFG_DEVICEn_CH,
                                    HS3000_CFG_DEVICEn_DRIVER_TYPE);
          if (HS3000_SUCCESS == retval)
          {
             p_hs3000_handle->driver_type    = HS3000_CFG_DEVICEn_DRIVER_TYPE;
             p_hs3000_handle->ch_no          = HS3000_CFG_DEVICEn_CH;
             p_hs3000_handle->slave_address  = HS3000_CFG_DEVICEn_SLAVE_ADDRESS;     */

             /* Casting to void type is valid */
  /*         p_hs3000_handle->p_sensor_info      = (void *)p_hs3000_info;
             retval = HS3000_SUCCESS;
          }
          break;                                                    */
```

Figure 5-4 Description on Source When Adding Device (r_hs3000_rx_if.h)

(3) r_hs3000_rx_if.h

This is the process of defining and checking the definitions of "HS3000_DRIVER_RIIC_ENABLE" and "HS3000_DRIVER_SCI_IIC_ENABLE". If you add it, please change it.

```
* Define HS3000_DRIVER_RIIC_ENABLE when using the specified FIT module [ADD_DEVICE] */
#if ((HS3000_CFG_DEVICE_NUM_MAX >= 1) && (HS3000_CFG_DEVICE0_DRIVER_TYPE == 1) \
||   (HS3000_CFG_DEVICE_NUM_MAX >= 2) && (HS3000_CFG_DEVICE1_DRIVER_TYPE == 1))
   #define HS3000_DRIVER_RIIC_ENABLE          (1)          /* for RIIC FIT module */
#else
   #define HS3000_DRIVER_RIIC_ENABLE          (0)
#endif

/* Define HS3000_DRIVER_SCI_IIC_ENABLE when using the specified FIT module [Add Device] */
#if ((HS3000_CFG_DEVICE_NUM_MAX >= 1) && (HS3000_CFG_DEVICE0_DRIVER_TYPE == 2) \
||   (HS3000_CFG_DEVICE_NUM_MAX >= 2) && (HS3000_CFG_DEVICE1_DRIVER_TYPE == 2))
   #define HS3000_DRIVER_SCI_IIC_ENABLE       (1)          /* for SCI_IIC FIT module */
#else
   #define HS3000_DRIVER_SCI_IIC_ENABLE       (0)
#endif

/* Check the Channel No. and the Number (Max.) of the Channel [Add Device] */
#if (HS3000_CFG_DEVICE_NUM_MAX >= 1)
   #if (HS3000_CFG_DEVICE0_DRIVER_TYPE == 1)
      #if (HS3000_CFG_DEVICE0_CH >= HS3000_CFG_CHANNEL_NUM_MAX_RIIC)
         #error "Review HS3000_CFG_DEVICE0_CH and HS3000_CFG_CHANNEL_NUM_MAX_RIIC."
      #endif
   #elif (HS3000_CFG_DEVICE0_DRIVER_TYPE == 2)
      #if (HS3000_CFG_DEVICE0_CH >= HS3000_CFG_CHANNEL_NUM_MAX_SCI_IIC)
         #error "Review HS3000_CFG_DEVICE0_CH and HS3000_CFG_CHANNEL_NUM_MAX_SCI_IIC."
      #endif
   #else
   #endif
#endif

#if (HS3000_CFG_DEVICE_NUM_MAX >= 2)
   #if (HS3000_CFG_DEVICE1_DRIVER_TYPE == 1)
      #if (HS3000_CFG_DEVICE1_CH >= HS3000_CFG_CHANNEL_NUM_MAX_RIIC)
         #error "Review HS3000_CFG_DEVICE1_CH and HS3000_CFG_CHANNEL_NUM_MAX_RIIC."
      #endif
   #elif (HS3000_CFG_DEVICE1_DRIVER_TYPE == 2)
      #if (HS3000_CFG_DEVICE1_CH >= HS3000_CFG_CHANNEL_NUM_MAX_SCI_IIC)
         #error "Review HS3000_CFG_DEVICE1_CH and HS3000_CFG_CHANNEL_NUM_MAX_SCI_IIC."
      #endif
   #else
   #endif
#endif
```

Figure 5-5 Description on Source When Adding Device (r_hs3000_rx_if.h)

5.4 How to Add Channel Number

The parts that need to be changed are shown below when the number of channels that can be supported increases are shown below. [ADD_CHANNEL] is described on the source as a keyword when adding a channel.

(1) r_hs3000_drvif_attach() Function

(a) When Adding RIIC Channel No

```
    /* Add when channel is missing [ADD_CHANNEL] */
/*  case N:
        if (1 == RIIC_CFG_CHn_INCLUDED)
        {                                                   */
            /* Casting to void type is valid */
/*          p_hs3000_handle->p_drv_callback = (void *)&r_hs3000_drvif_callback_riic_chN;
            p_hs3000_info->p_ch_status = &g_hs3000_ch_status_riic[p_hs3000_handle->ch_no];
            retval = HS3000_SUCCESS;
        }
        break;                                              */
```

Figure 5-6 Description on Source When Adding RIIC Channel

(b) When Adding SCI_IIC Channel No

```
    /* Add when channel is missing [ADD_CHANNEL] */
/*  case N:
        if (1 == SCI_IIC_CFG_CHn_INCLUDED)
        {                                                   */
            /* Casting to void type is valid */
/*          p_hs3000_handle->p_drv_callback = (void *)&r_hs3000_drvif_callback_sci_iic_chN;
            p_hs3000_info->p_ch_status = &g_hs3000_ch_status_sci_iic[p_hs3000_handle->ch_no];
            retval = HS3000_SUCCESS;
        }
        break;                                              */
```

Figure 5-7 Description on Source When Adding SCI_IIC Channel

(2) Callback Functions

(a) When Adding RIIC Channel No

Add by referring to the r_hs3000_drvif_callback_riic_ch2() function.

(b) When Adding SCI_IIC Channel No

Add by referring to the r_hs3000_drvif_callback_sci_iic_ch12() function.

5.5 Precautions when IIC Control Drivers other than IIC FIT Driver are mixed

If the R_HS3000_Start()/R_HS3000_Get()/R_HS3000_Control()function are called during communication using the IIC FIT module, the I2C communication information structure of the IIC FIT module is rewritten during communication, which corresponds to the prohibited items for using the IIC FIT module.

When calling the IIC FIT module, the HS3000 FIT module first refers to the channel status flag (g_riic_ChStatus[]/g_sci_iic_ChStatus[]) of the IIC FIT module to check if it is in communicating status, as shown in the figure below. The HS3000 FIT module rewrites the I2C communication information structure of the IIC FIT module when communication is not in progress.

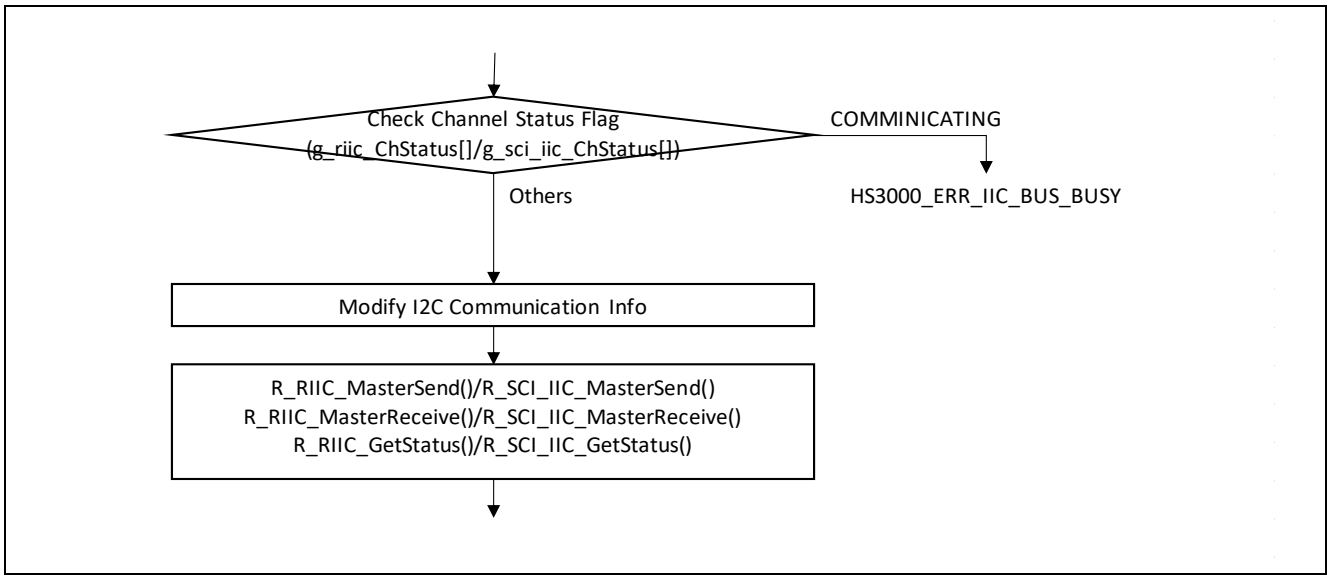


Figure 5-8 I2C Bus Busy Check Process before Rewriting I2C Communication Information Structure

If IIC control modules other than the IIC FIT module are mixed, the above bus busy check process will not work effectively. Therefore, control by paying attention to the following.

— Call the HS3000 FIT module after communication with other devices is finished

6. Reference Documents

User’s Manual: Hardware
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.

User’s Manual: Development Tools
RX Family Compiler CC-RX User's Manual (R20UT3248)
The latest versions can be downloaded from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	Summary
1.01	Jan.12.2021	-	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

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/.