RX Family

RI3C Module Using Firmware Integration Technology

Introduction

This application note describes the Renesas I3C module using firmware integration technology (FIT) for communications between devices using the Improved Inter-Integrated Circuit communications interface.

Target Device

- RX26T Groups (products with 64 Kbytes of RAM)

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

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.1 Operating Test Environment”.

Contents

1. Overview

The Renesas I3C module using firmware integration technology (RI3C FIT module) provides a method to transmit and receive data between the controller and target devices using the RI3C. The RI3C complies with MIPI I3C.

Limitations

- This module does not support HDR (I3C high data rate) mode.

1.1 Renesas I3C FIT Module

This module is implemented in a project and used as the API. Refer to 2.12 Adding the FIT Module to Your Project for details on implementing the module to the project.

1.2 Outline of the API

Table 1.1 lists the API Functions.

Table 1.1 API Functions

Item	Contents
R_RI3C_Open()	Configure an RI3C instance.
R_RI3C_Enable()	Enable the RI3C device.
R_RI3C_DeviceCfgSet()	Set the configuration for this device.
R_RI3C_ControllerDeviceTableSet()	Configure an entry in the controller device table.
R_RI3C_TargetStatusSet()	Set the status returned to the controller in response to a GETSTATUS command.
R_RI3C_DeviceSelect()	In controller mode, select the device for the next transfer.
R_RI3C_DynamicAddressAsignmentStart()	Start the Dynamic Address Assignment Process.
R_RI3C_CommandSend()	Send a broadcast or direct command to target devices on the bus.
R_RI3C_Write()	Set the write buffer for the transfer. In controller mode, start the transfer. When the transfer is completed, send a stop condition or a repeated-start.
R_RI3C_Read()	Set the read buffer for the transfer. In controller mode, start the transfer. When the transfer is completed, send a stop condition or a repeated-start.
R_RI3C_IbiWrite()	Initiate an IBI write operation.
R_RI3C_IbiRead()	Set the read buffer for storing received IBI data.
R_RI3C_Close()	Close the RI3C instance.

1.3 Overview of RI3C FIT Module

1.3.1 Specifications of RI3C FIT Module

1. This module supports controller transmission, controller reception, target transmission and target reception.
2. This module supports SDR (I3C single data rate) mode with FIFO transfer.
3. RI3C autonomously starts transfer when data and command are written.
4. An interrupt occurs in one of the following contexts: Respond queue full (RESPI), command queue empty (CMDI), IBI queue empty/full (IBII), receive status queue full (RCVI), receive data full (RXI), transmit data empty (TXI), Communication error or communication event (EEI).
5. Controller device can communicate with multiple target devices by using 7-bit address.

1.4 Using RI3C FIT Module

1.4.1 Using RI3C FIT Module in C++ project

For C++ project, add RI3C FIT module interface header file within extern “C”{}:

```
extern “C”
{
#include “r_smc_entry.h”
#include “r_ri3c_rx_if.h”
}
```

2. API Information

This driver API adheres to the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires your MCU supports the following feature:

- RI3C Module.

2.2 Software Requirements

This driver is dependent upon the following packages:

- Board Support Package Module (r_bsp) Rev.7.30 or higher

2.3 Supported Toolchains

This driver has been confirmed to work with the toolchain listed in 6.1 Operating Test Environment

2.4 Interrupt Vector

The In-band interrupts are enabled by execution of R_RI3C_Enable API function.

Table 2.1 shows the interrupt vectors used by the RI3C FIT module.

Table 2.1 List of Usage of Interrupt Vectors.

Device	Contents
RX26T	RESPI interrupt (vector no.: 40) CMDI interrupt (vector no.: 41) IBII interrupt (vector no.: 42) RCVI interrupt (vector no.: 43) RXI interrupt (vector no.: 118) TXI interrupt (vector no.: 119) EEI interrupt (vector no.: 113)

2.5 Header Files

All API calls and their supporting interface definitions are located in r_ri3c_rx_if.h and r_ri3c_rx_api.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 module are specified in r_ri3c_rx_config.h. The option names and setting values are listed in the table below.

Configuration options in *r_ri3c_rx_config.h*	
RI3C_CFG_PARAM_CHECKING_ENABLE **- Default value =** **BSP_CFG_PARAM_CHECKING_ENABLE**	Specify whether to include code for API parameter checking Set to 1 includes parameter checking; 0 compiles out parameter checking Use this option with caution.
RI3C_CFG_UNALIGNED_SUPPORT **- Default value = 1**	Selects whether to support Unaligned Buffer in this device. 1 = Enabled (System default) 0 = Disabled
RI3C_CFG_DEVICE_TYPE **- Default value = RI3C_DEVICE_TYPE_TARGET**	Device operation mode. RI3C_DEVICE_TYPE_PRIMARY_CONTROLLER RI3C_DEVICE_TYPE_TARGET (System default)
RI3C_CFG_CONTROLLER_SUPPORT **- Default value = 1**	Select enable/disable Controller mode of this device. 1 = Enabled (System default) 0 = Disabled If only Controller mode is required, disable Target support to decrease code size.
RI3C_CFG_TARGET_SUPPORT **- Default value = 1**	Select enable/disable Target mode of this device 1 = Enabled (System default) 0 = Disabled If only Target mode is required, disable Controller support to decrease code size.
RI3C_CFG_PCLKA_REF_VALUE **- Default value = 48000000**	Peripheral Clock A Reference Value. 48 MHz (System default) 64 MHz FIT RI3C Modules works properly on 48 MHz or 64 MHz.
RI3C_CFG_STANDARD_OPEN_DRAIN_LOGIC_HIGH_PERIOD **- Default value = 167**	The Logic High period of SCL during Standard Mode Open Drain transfers.
RI3C_CFG_STANDARD_OPEN_DRAIN_FREQUENCY **- Default value = 1000000**	The Frequency of SCL during Standard Mode Open Drain transfers.
RI3C_CFG_STANDARD_PUSH_PULL_LOGIC_HIGH_PERIOD **- Default value = 167**	The Logic High period of SCL during Standard Mode Push Pull transfers.
RI3C_CFG_STANDARD_PUSH_PULL_FREQUENCY **- Default value = 3400000**	The Frequency of SCL during Standard Mode Push Pull transfers.
RI3C_CFG_EXTENDED_OPEN_DRAIN_LOGIC_HIGH_PERIOD **- Default value = 167**	The Logic High period of SCL during Extended Mode Open Drain transfers.
RI3C_CFG_EXTENDED_OPEN_DRAIN_FREQUENCY **- Default value = 1000000**	The Frequency of SCL during Extended Mode Open Drain transfers.
RI3C_CFG_EXTENDED_PUSH_PULL_LOGIC_HIGH_PERIOD **- Default value = 167**	The Logic High period of SCL during Extended Mode Push Pull transfers.
RI3C_CFG_EXTENDED_PUSH_PULL_FREQUENCY **- Default value = 3400000**	The Frequency of SCL during Extended Mode Push Pull transfers.
RI3C_CFG_OPEN_DRAIN_RISING_TIME **- Default value = 0**	The Open Drain rising time in nanoseconds. Value must be greater than or equal to 0

Configuration options in *r_ri3c_rx_config.h*	
RI3C_CFG_OPEN_DRAIN_FALLING_TIME **- Default value = 0**	The Open Drain falling time in nanoseconds. Value must be greater than or equal to 0
RI3C_CFG_PUSH_PULL_RISING_TIME **- Default value = 0**	The Push Pull rising time in nanoseconds. Value must be greater than or equal to 0
RI3C_CFG_PUSH_PULL_FALLING_TIME **- Default value = 0**	The Push Pull falling time in nanoseconds. Value must be greater than or equal to 0
RI3C_CFG_ADDRESS_ASSIGNMENT_PHASE **- Default value = 0**	Enable clock stalling during the Address Assignment Phase of ENTDAA. 0 = Disable (System default) 1 = Enable
RI3C_CFG_TRANSITION_PHASE **- Default value = 0**	Enable clock stalling during the Transition Bit of a read transfer. 0 = Disable (System default) 1 = Enable
RI3C_CFG_PARITY_PHASE **- Default value = 0**	Enable clock stalling during the Parity Bit of a write transfer. 0 = Disable (System default) 1 = Enable
RI3C_CFG_ACK_PHASE **- Default value = 0**	Enable clock stalling during the ACK phase of a transfer. 0 = Disable (System default) 1 = Enable
RI3C_CFG_CLOCK_STALLING_TIME **- Default value = 0**	The amount of time to stall the clock during the Address Assignment Phase, Transition Phase, Parity Phase, and ACK Phase. Value must be an integer Value must be great than or equal 0 and less than PCLKA
RI3C_CFG_CONTROLLER_ACK_HOTJOIN_REQ **- Default value = 0**	If enabled, the RI3C instance will ACK Hot-Join Requests and notify the application. 0 = Disable (System default) 1 = Enable
RI3C_CFG_CONTROLLER_NOTIFY_REJECTED_HOTJOIN_REQ **- Default value = 0**	If enabled, the application will get a callback when an IBI Hot-Join Request is rejected. 0 = Disable (System default) 1 = Enable
RI3C_CFG_CONTROLLER_NOTIFY_REJECTED_CONTROLLERROLE_REQ **- Default value = 0**	If enabled, the application will get a callback when an IBI Controller Role Request is rejected. 0 = Disable (System default) 1 = Enable
RI3C_CFG_CONTROLLER_NOTIFY_REJECTED_INTERRUPT_REQ **- Default value = 0**	If enabled, the application will get a callback when an IBI Interrupt Request is rejected. 0 = Disable (System default) 1 = Enable
RI3C_CFG_TARGET_IBI_REQ **- Default value = 0**	Configure whether the target can issue IBI requests. 0 = Disable (System default) 1 = Enable

Configuration options in *r_ri3c_rx_config.h*	
RI3C_CFG_TARGET_HOTJOIN_REQ **- Default value = 0**	Configure whether the target can issue Hot-Join requests. 0 = Disable (System default) 1 = Enable
RI3C_CFG_TARGET_CONTROLLERROLE_REQ **- Default value = 0**	Configure whether the target can issue Controller Role requests. 0 = Disable (System default) 1 = Enable
RI3C_CFG_TARGET_INCLUDE_MAX_READ_TURNAROUND_TIME **- Default value = 0**	Configure whether the Max Read Turnaround time will be transmitted. 0 = Disable (System default) 1 = Enable
RI3C_CFG_TARGET_ENTER_ACTIVITY_STATE **- Default value = RI3C_ACTIVITY_STATE_ENTAS0**	Configure the starting activity state of the target. RI3C_ACTIVITY_STATE_ENTAS0 = 1 nanosec: Latency-free operation (Default value) RI3C_ACTIVITY_STATE_ENTAS1 = 100 nanosec RI3C_ACTIVITY_STATE_ENTAS2 = 2 microsec RI3C_ACTIVITY_STATE_ENTAS3 = 50 microsec: Lowest-activity operation
RI3C_CFG_TARGET_MAX_WRITE_LENGTH **- Default value = 65535**	Set the max write length in Target Mode. Write length must be in range of [8, 65535]
RI3C_CFG_TARGET_MAX_READ_LENGTH **- Default value = 65535**	Set the max read length in Target Mode. Read length must be in range of [8, 65535]
RI3C_CFG_TARGET_MAX_IBI_PAYLOAD_LENGTH **- Default value = 0**	Set the max IBI payload length, or set it to 0 for unlimited. Read length must be in range of [0, 255]
RI3C_CFG_TARGET_WRITE_DATA_RATE **- Default value = RI3C_DATA_RATE_SETTING_2MHZ**	Set the max write data rate in Target Mode. RI3C_DATA_RATE_SETTING_FSCL_MAX = FSCL_MAX RI3C_DATA_RATE_SETTING_8MHZ = 8 MHz RI3C_DATA_RATE_SETTING_6MHZ = 6 MHz RI3C_DATA_RATE_SETTING_4MHZ = 4 MHz RI3C_DATA_RATE_SETTING_2MHZ = 2 MHz (Default)
RI3C_CFG_TARGET_READ_DATA_RATE **- Default value = RI3C_DATA_RATE_SETTING_2MHZ**	Set the max read data rate in Target Mode. RI3C_DATA_RATE_SETTING_FSCL_MAX = FSCL_MAX RI3C_DATA_RATE_SETTING_8MHZ = 8 MHz RI3C_DATA_RATE_SETTING_6MHZ = 6 MHz RI3C_DATA_RATE_SETTING_4MHZ = 4 MHz RI3C_DATA_RATE_SETTING_2MHZ = 2 MHz (Default)

Configuration options in *r_ri3c_rx_config.h*	
RI3C_CFG_TARGET_CLK_TURNAROUND_RATE **- Default value =** **RI3C_CLOCK_DATA_TURNAROUND_8NS**	Set the max clock to data turnaround time in Target Mode. RI3C_CLOCK_DATA_TURNAROUND_8NS = 8 nanoseconds (Default) RI3C_CLOCK_DATA_TURNAROUND_9NS = 9 nanoseconds RI3C_CLOCK_DATA_TURNAROUND_10NS = 10 nanoseconds RI3C_CLOCK_DATA_TURNAROUND_11NS = 11 nanoseconds RI3C_CLOCK_DATA_TURNAROUND_12NS = 12 nanoseconds RI3C_CLOCK_DATA_TURNAROUND_EXTENDED = Greater than 12 nanoseconds
RI3C_CFG_TARGET_INCLUDE_MAX_READ_TURNAROUND_TIME **- Default value = 0**	Configure whether the Max Read Turnaround Time will be transmitted in Target Mode. 0 = Disable (System default) 1 = Enable
RI3C_CFG_TARGET_MAX_READ_TURNAROUND_TIME **- Default value = 0**	Set max read turnaround time in Target Mode. Value must be in the range of [0, 255]
RI3C_CFG_TARGET_FREQUENCY_BYTE **- Default value = 0**	Set the internal oscillator frequency in increments of 0.5 MHz in Target Mode. Value must be in the range of [0, 255]
RI3C_CFG_TARGET_INACCURACY_BYTE **- Default value = 0**	Set the oscillator inaccuracy byte in increments of 0.5% in Target Mode. Value must be in the range of [0, 255]
RI3C_CFG_BUS_FREE_DETECT_TIME **- Default value = 38.4**	The minimum period occurring after a STOP and before a START. Must be greater than or equal to 38.4 nanoseconds.
RI3C_CFG_BUS_AVAILABLE_CONDITION_DETECT_TIME **- Default value = 1**	The minimum period occurring after the Bus Free Condition when Targets can initiate IBI requests. Must be greater than or equal to 1 microsecond.
RI3C_CFG_BUS_IDLE_CONDITION_DETECT_TIME **- Default value = 1000**	The minimum period occurring after the Bus Available Condition when Targets can initiate Hot-Join requests. Must be greater than or equal to 1000 microsecond.
RI3C_CFG_TIMEOUT_DETECTION **- Default value = 0**	If enabled, the application will get a callback if SCL is stuck at a logic high or logic low level for more than 65535 cycles of the RI3C source clock. 0 = Disable (System default) 1 = Enable
RI3C_CFG_INTERRUPT_PRIORITY_LEVEL **- Default value = 2**	Set the interrupt priority level of the RI3C Module. Value must be in the range of [0, 15]

Table 2.2 List of RI3C module configuration options.

2.8 Code Size

Typical code sizes associated with this module are listed below. Information is listed for a single representative device of the RX26T Group.

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 are optimization level: 2, optimization type: for size, and 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 Revision: r_ri3c_rx rev1.00

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00

(The option of "-lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 8.03.00.202204

(The option of "-std=gnu99" is added to the default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Memory Usage

Device	Category		Memory Used					
			Renesas Compiler		GCC		IAR Compiler	
			With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX26T	ROM	Controller device	**6569 bytes**	**5788 bytes**	**7508 bytes**	**6804 bytes**	**7071 bytes**	**6615 bytes**
		Target device	**5931 bytes**	**5474 bytes**	**6876 bytes**	**6428 bytes**	**6615 bytes**	**6465 bytes**
	RAM	Controller device	**292 bytes**		**384 bytes**		**192 bytes**	
		Target device	**292 bytes**		**384 bytes**		**192 bytes**	
	STACK *1		**268 bytes**	**300 bytes**	**-**		**300 bytes**	**300 bytes**

Note 1. The size of maximum usage stack of Interrupts functions is included.

2.9 Parameters

This section describes the structure whose members are API parameters. These structures are located in r_ri3c_rx_if.h and r_ri3c_rx_api.h

```
typedef struct st_ri3c_instance_ctrl
{
    uint32_t open;

#if((__CCRX__) || (__GNUC__))
    volatile struct RI3C0_Type R_BSP_EVENACCESS * p_reg;
#elif(__ICCRX__)
    struct RI3C0_Type R_BSP_VOLATILE_SFR * p_reg;
#endif

    volatile uint32_t              internal_state;
    uint8_t                        current_command_code;
    uint32_t                       device_index;
    ri3c_bitrate_mode_t            device_bitrate_mode;
    ri3c_target_info_t             current_target_info;
    uint32_t                       next_word;
    uint32_t                       ibi_next_word;
    ri3c_write_buffer_descriptor_t write_buffer_descriptor;
    ri3c_read_buffer_descriptor_t  read_buffer_descriptor;
    ri3c_read_buffer_descriptor_t  ibi_buffer_descriptor;
    volatile uint32_t              read_transfer_count_final;
    volatile uint32_t              ibi_transfer_count_final;
    ri3c_cfg_t const             * p_cfg;
} ri3c_instance_ctrl_t
```

```
typedef struct s_ri3c_extended_cfg
{
    ri3c_bitrate_settings_t              bitrate_settings;
    ri3c_ibi_control_t                   ibi_control
    uint32_t                             bus_free_detection_time;
    uint32_t                             bus_available_detection_time;
    uint32_t                             bus_idle_detection_time;
    bool                                 timeout_detection_enable;
    ri3c_target_command_response_info_t target_command_response_info;
    uint8_t                              ipl;
    uint8_t                              eei_ipl;
} ri3c_extended_cfg_t;
```

```
typedef struct s_target_command_response_info
{
    bool                          inband_interrupt_enable;
    bool                          controllerrole_request_enable;
    bool                          hotjoin_request_enable;
    ri3c_activity_state_t         activity_state;
    uint16_t                      write_length;
    uint16_t                      read_length;
    uint8_t                       ibi_payload_length;
    ri3c_data_rate_setting_t      write_data_rate;
    ri3c_data_rate_setting_t      read_data_rate;
    ri3c_clock_data_turnaround_t clock_data_turnaround;
    bool                          read_turnaround_time_enable;
    uint32_t                      read_turnaround_time;
    uint8_t                       oscillator_frequency;
    uint8_t                       oscillator_inaccuracy;
} ri3c_target_command_response_info_t;
```

```
typedef struct s_ri3c_clock_stalling
{
    uint32_t assigned_address_phase_enable : 1;
    uint32_t transition_phase_enable       : 1;
    uint32_t parity_phase_enable           : 1;
    uint32_t ack_phase_enable              : 1;
    uint16_t clock_stalling_time;
} ri3c_clock_stalling_t;
```

```
typedef struct s_ri3c_bitrate_settings
{
    uint32_t              icsbr; ///< The standard bitrate settings.
    uint32_t              icebr; ///< The extended bitrate settings.
    ri3c_clock_stalling_t clock_stalling;
} ri3c_bitrate_settings_t;
```

```
typedef struct s_ri3c_ibi_control
{
    uint32_t hot_join_acknowledge                    : 1;
    uint32_t notify_rejected_hot_join_requests       : 1;
    uint32_t notify_rejected_controllerrole_requests : 1;
    uint32_t notify_rejected_interrupt_requests      : 1;
} ri3c_ibi_control_t;
```

```
typedef struct s_ri3c_read_buffer_descriptor
{
    uint8_t * p_buffer;
    uint32_t  count;
    uint32_t  buffer_size;
    bool      read_request_issued;
} ri3c_read_buffer_descriptor_t;
```

```
typedef struct s_ri3c_write_buffer_descriptor
{
    uint8_t * p_buffer;
    uint32_t  count;
    uint32_t  buffer_size;
} ri3c_write_buffer_descriptor_t;
```

```
typedef enum e_ri3c_common_command_code
{
    /* Broadcast Common Command Codes */
    I3C_CCC_BROADCAST_ENEC     = (0x00), ///< Enable Target initiated events.
    I3C_CCC_BROADCAST_DISEC    = (0x01), ///< Disable Target initiated events.
    I3C_CCC_BROADCAST_ENTAS0   = (0x02), ///< Enter Activity State 0.
    I3C_CCC_BROADCAST_ENTAS1   = (0x03), ///< Enter Activity State 1.
    I3C_CCC_BROADCAST_ENTAS2   = (0x04), ///< Enter Activity State 2.
    I3C_CCC_BROADCAST_ENTAS3   = (0x05), ///< Enter Activity State 3.
    I3C_CCC_BROADCAST_RSTDAA   = (0x06), ///< Reset Dynamic Address Assignment.
    I3C_CCC_BROADCAST_ENTDAA   = (0x07), ///< Enter Dynamic Address Assignment.
    I3C_CCC_BROADCAST_DEFSVLS  = (0x08), ///< Define List of Targets.
    I3C_CCC_BROADCAST_SETMWL   = (0x09), ///< Set Max Write Length.
    I3C_CCC_BROADCAST_SETMRL   = (0x0A), ///< Set Max Read Length.
    I3C_CCC_BROADCAST_ENTTM    = (0x0B), ///< Enter Test Mode.
    I3C_CCC_BROADCAST_ENTHDR0  = (0x20), ///< Enter HDR Mode 0.
    I3C_CCC_BROADCAST_ENTHDR1  = (0x21), ///< Enter HDR Mode 1.
    I3C_CCC_BROADCAST_ENTHDR2  = (0x22), ///< Enter HDR Mode 2.
    I3C_CCC_BROADCAST_ENTHDR3  = (0x23), ///< Enter HDR Mode 3.
    I3C_CCC_BROADCAST_ENTHDR4  = (0x24), ///< Enter HDR Mode 4.
    I3C_CCC_BROADCAST_ENTHDR5  = (0x25), ///< Enter HDR Mode 5.
    I3C_CCC_BROADCAST_ENTHDR6  = (0x26), ///< Enter HDR Mode 6.
    I3C_CCC_BROADCAST_ENTHDR7  = (0x27), ///< Enter HDR Mode 7.
    I3C_CCC_BROADCAST_SETXTIME = (0x28), ///< Set Exchange Timing Info.
    I3C_CCC_BROADCAST_SETAASA  = (0x29), ///< Set All Addresses to Static Address.

    /* Direct Common Command Codes */
    I3C_CCC_DIRECT_ENEC      = (0x80),   ///< Enable Target initiated events.
    I3C_CCC_DIRECT_DISEC     = (0x81),   ///< Disable Target initiated events.
    I3C_CCC_DIRECT_ENTAS0    = (0x82),   ///< Enter Activity State 0.
    I3C_CCC_DIRECT_ENTAS1    = (0x83),   ///< Enter Activity State 1.
    I3C_CCC_DIRECT_ENTAS2    = (0x84),   ///< Enter Activity State 2.
    I3C_CCC_DIRECT_ENTAS3    = (0x85),   ///< Enter Activity State 3.
    I3C_CCC_DIRECT_RSTDAA    = (0x86),   ///< Reset Dynamic Address Assignment.
    I3C_CCC_DIRECT_SETDASA   = (0x87),   ///< Set Dynamic Address from Static Address.
    I3C_CCC_DIRECT_SETNEWDA  = (0x88),   ///< Set New Dynamic Address.
    I3C_CCC_DIRECT_SETMWL    = (0x89),   ///< Set Max Write Length.
    I3C_CCC_DIRECT_SETMRL    = (0x8A),   ///< Set Max Read Length.
    I3C_CCC_DIRECT_GETMWL    = (0x8B),   ///< Get Max Write Length.
    I3C_CCC_DIRECT_GETMRL    = (0x8C),   ///< Get Max Read Length.
    I3C_CCC_DIRECT_GETPID    = (0x8D),   ///< Get Provisional ID.
    I3C_CCC_DIRECT_GETBCR    = (0x8E),   ///< Get Bus Characteristic Register.
    I3C_CCC_DIRECT_GETDCR    = (0x8F),   ///< Get Device Characteristic Register.
    I3C_CCC_DIRECT_GETSTATUS = (0x90),   ///< Get Device Status.
    I3C_CCC_DIRECT_GETACCMST = (0x91),   ///< Get Accept Controller Role.
    I3C_CCC_DIRECT_GETMXDS   = (0x94),   ///< Get Max Data Speed.
    I3C_CCC_DIRECT_SETXTIME  = (0x98),   ///< Set Exchange Timing Information.
    I3C_CCC_DIRECT_GETXTIME  = (0x99),   ///< Get Exchange Timing Information.
} ri3c_common_command_code_t;
```

```
typedef struct s_ri3c_target_device_cfg
{
    uint8_t            static_address;
    uint8_t            dynamic_address;
    ri3c_target_info_t target_info;
} ri3c_device_cfg_t;
```

```
typedef enum e_ri3c_event
{
    RI3C_EVENT_ENTDAA_ADDRESS_PHASE,
    RI3C_EVENT_IBI_READ_COMPLETE,
    RI3C_EVENT_IBI_READ_BUFFER_FULL,
    RI3C_EVENT_READ_BUFFER_FULL,
    RI3C_EVENT_IBI_WRITE_COMPLETE,
    RI3C_EVENT_HDR_EXIT_PATTERN_DETECTED,
    RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE,
    RI3C_EVENT_COMMAND_COMPLETE,
    RI3C_EVENT_WRITE_COMPLETE,
    RI3C_EVENT_READ_COMPLETE,
    RI3C_EVENT_TIMEOUT_DETECTED,
    RI3C_EVENT_INTERNAL_ERROR,
} ri3c_event_t;
```

```
typedef enum e_ri3c_type
{
    RI3C_DEVICE_TYPE_PRIMARY_CONTROLLER,
    RI3C_DEVICE_TYPE_TARGET,
} ri3c_device_type_t;
```

```
typedef enum e_ri3c_device_protocol
{
    RI3C_DEVICE_PROTOCOL_I2C,
    RI3C_DEVICE_PROTOCOL_I3C,
} ri3c_device_protocol_t;
```

```
typedef enum e_ri3c_address_assignment_mode
{
    RI3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA  = I3C_CCC_BROADCAST_ENTDAA,
    RI3C_ADDRESS_ASSIGNMENT_MODE_SETDASA = I3C_CCC_DIRECT_SETDASA,
} ri3c_address_assignment_mode_t;
```

```
typedef enum e_ri3c_ibi_type
{
    RI3C_IBI_TYPE_INTERRUPT,
    RI3C_IBI_TYPE_HOT_JOIN,
    RI3C_IBI_TYPE_CONTROLLERROLE_REQUEST
} ri3c_ibi_type_t;
```

```
typedef struct s_ri3c_device_status
{
    uint8_t pending_interrupt;
    uint8_t vendor_status;
} ri3c_device_status_t;
```

```
typedef struct s_ri3c_device_table_cfg
{
    uint8_t static_address;            ///< I3C Static address / I2C address for
this device.

    /** Dynamic address for the device. This address will be assigned during
Dynamic Address Assignment. */
    uint8_t dynamic_address;

    ri3c_device_protocol_t device_protocol;     ///< The protocol used to
communicate with this device (I3C / I2C Legacy).
    bool ibi_accept;                            ///< Accept or reject IBI
requests from this device.
    bool controllerrole_request_accept;         ///< Accept controller role
requests from this device.

    /**
     * IBI requests from this device have a data payload.
     *
     * Note: When the device is configured using ENTDAA, the ibi_payload will
automatically be updated
     *       based on the value of BCR.
     */
    bool ibi_payload;
} ri3c_device_table_cfg_t;
```

```
typedef struct s_ri3c_target_info
{
    uint8_t pid[6];
    union
    {
        uint8_t bcr;
        struct
        {
            uint8_t max_data_speed_limitation : 1;
            uint8_t ibi_request_capable       : 1;
            uint8_t ibi_payload               : 1;
            uint8_t offline_capable           : 1;
            uint8_t                           : 2;
            uint8_t device_role               : 2;
        } bcr_b;
    };

   uint8_t dcr;
} ri3c_target_info_t;
```

```
typedef struct s_ri3c_command_descriptor
{
    uint8_t   command_code;
    uint8_t * p_buffer;
    uint32_t  length;
    bool      restart;
    bool      rnw;
} ri3c_command_descriptor_t;
```

```
typedef struct s_ri3c_callback_args
{
    ri3c_event_t               event;
    uint32_t                   event_status;
    uint32_t                   transfer_size;
    ri3c_target_info_t const * p_target_info;
    uint8_t                    dynamic_address;
    ri3c_ibi_type_t            ibi_type;
    uint8_t                    ibi_address;
    uint8_t                    command_code;
    void const               * p_context;
} ri3c_callback_args_t;
```

```
typedef struct st_ri3c_cfg
{
    uint32_t            channel;
    ri3c_device_type_t  device_type;
    void (* p_callback) (ri3c_callback_args_t const * const p_args);
    void const        * p_context;
    void const        * p_extend;
} ri3c_cfg_t;
```

```
typedef void ri3c_ctrl_t;

typedef struct st_ri3c_instance
{
    ri3c_ctrl_t * p_ctrl;
    ri3c_cfg_t  * p_cfg;
    ri3c_api_t  * p_api;
} ri3c_instance_t;
```

```
typedef struct st_ri3c_api
{
    fsp_err_t (* open)
              (ri3c_ctrl_t * const p_ctrl, ri3c_cfg_t const * const p_cfg);

    fsp_err_t (* enable)
              (ri3c_ctrl_t * const p_ctrl);

    fsp_err_t (* deviceCfgSet)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_device_cfg_t const * const p_device_cfg);

    fsp_err_t (* controllerDeviceTableSet)
              (ri3c_ctrl_t * const p_ctrl,
               uint32_t device_index,
               ri3c_device_table_cfg_t const * const p_device_table_cfg);

    fsp_err_t (* deviceSelect)
              (ri3c_ctrl_t * const p_ctrl,
               uint32_t device_index,
               uint32_t bitrate_mode);

    fsp_err_t (* dynamicAddressAssignmentStart)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_address_assignment_mode_t address_assignment_mode,
               uint32_t starting_device_index,
               uint32_t device_count);

    fsp_err_t (* targetStatusSet)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_device_status_t device_status);

    fsp_err_t (* commandSend)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_command_descriptor_t * p_command_descriptor);

    fsp_err_t (* write)
              (ri3c_ctrl_t * const p_ctrl,
               uint8_t const * const p_data,
               uint32_t length,
               bool restart);

    /**
     * In controller mode: Start a read transfer. When the transfer is
     * completed, send a stop condition or a repeated-start.
     * In target mode:  Set the read buffer for storing data read during the
     * transfer. When the buffer is full, the application
     * will receive a callback requesting a new read buffer. If no buffer
     * is provided by the application, the driver will discard
     * any remaining bytes read during the transfer.
     *
     * @param[in]   p_ctrl     Control block set this instance.
     * @param[in]   p_data     Pointer to a buffer to store the bytes read
     *                         during the transfer.
     * @param[in]   length     Number of bytes to transfer.
     * @param[in]   restart    If true, issue a repeated-start after the
     *                         transfer is completed (Controller only).
     */
    fsp_err_t (* read)(ri3c_ctrl_t * const p_ctrl, uint8_t * const p_data,
uint32_t length, bool restart);
```

```
    /**
     * Initiate an IBI write operation.
     *
     * Note: This function is not used in controller mode.
     *
     * @param[in]   p_ctrl     Control block set this instance.
     * @param[in]   p_data     Pointer to a buffer to start the bytes read
     *                         during the transfer.
     * @param[in]   length     Number of bytes to transfer.
     */
    fsp_err_t (* ibiWrite)(ri3c_ctrl_t * const p_ctrl,
                           ri3c_ibi_type_t ibi_type,
                           uint8_t const * const p_data,
                           uint32_t length);

    /**
     * Set the read buffer for storing received IBI data (This function is not
used in target mode).
     *
     * @param[in]   p_ctrl     Control block set this instance.
     * @param[in]   p_data     Pointer to a buffer to store the bytes read
     *                         during the transfer.
     * @param[in]   length     Number of bytes to transfer.
     */
    fsp_err_t (* ibiRead)(ri3c_ctrl_t * const p_ctrl, uint8_t * const p_data,
uint32_t length);

    /** Allows driver to be reconfigured and may reduce power consumption.
     *
     * @param[in]   p_ctrl     Control block set this instance.
     */
    fsp_err_t (* close)(ri3c_ctrl_t * const p_ctrl);
} ri3c_api_t;
```

2.10 Return Values

This section describes return values of API functions. This enumeration is located in fsp_common_api.h.

```
/** Common error codes */
typedef enum e_fsp_err
{
    FSP_SUCCESS                  = 0,

    FSP_ERR_ASSERTION            = 1,
    FSP_ERR_INVALID_POINTER      = 2,
    FSP_ERR_INVALID_ARGUMENT     = 3,
    FSP_ERR_INVALID_CHANNEL      = 4,
    FSP_ERR_INVALID_MODE         = 5,
    FSP_ERR_UNSUPPORTED          = 6,
    FSP_ERR_NOT_OPEN             = 7,
    FSP_ERR_IN_USE               = 8,
    FSP_ERR_OUT_OF_MEMORY        = 9,
    FSP_ERR_HW_LOCKED            = 10,
    FSP_ERR_IRQ_BSP_DISABLED     = 11,
    FSP_ERR_OVERFLOW             = 12,
    FSP_ERR_UNDERFLOW            = 13,
    FSP_ERR_ALREADY_OPEN         = 14,
    FSP_ERR_APPROXIMATION        = 15,
    FSP_ERR_CLAMPED              = 16,
    FSP_ERR_INVALID_RATE         = 17,
    FSP_ERR_ABORTED              = 18,
    FSP_ERR_NOT_ENABLED          = 19,
    FSP_ERR_TIMEOUT              = 20,
    FSP_ERR_INVALID_BLOCKS       = 21,
    FSP_ERR_INVALID_ADDRESS      = 22,
    FSP_ERR_INVALID_SIZE         = 23,
    FSP_ERR_WRITE_FAILED         = 24,
    FSP_ERR_ERASE_FAILED         = 25,
    FSP_ERR_INVALID_CALL         = 26,
    FSP_ERR_INVALID_HW_CONDITION  = 27,
    FSP_ERR_INVALID_FACTORY_FLASH = 28,
    FSP_ERR_INVALID_STATE        = 30,
    FSP_ERR_NOT_ERASED           = 31,
    FSP_ERR_SECTOR_RELEASE_FAILED = 32,
    FSP_ERR_NOT_INITIALIZED      = 33,
    FSP_ERR_NOT_FOUND            = 34,
    FSP_ERR_NO_CALLBACK_MEMORY   = 35,
    FSP_ERR_BUFFER_EMPTY         = 36,
    ...
} fsp_err_t;
```

2.11 Callback Functions

In this module, a callback function set up by the user is called when either of the following conditions is met.

(1) Transmit data empty

(2) Receive data full

(3) IBI queue empty/full

(4) Command queue empty

(5) Response queue full

(6) Receive status queue full

(7) Communication error/communication event

The callback function is set up by storing the address of the user function in the p_callback argument of ri3c_cfg_t structure. The default value of the callback function is a function pointer. User can change it into the user function by assigning g_ri3c0_cfg.p_callback to any function has “ri3c_callback_args_t const *const p_args” as argument.

Example

```
/* Function prototype */
void g_i3c0_user_callback(ri3c_callback_args_t const *const p_args);

void main(void)
{
    /* Assign callback function to RI3C FIT Module */
    g_ri3c0_cfg.p_callback = &g_i3c0_user_callback;
    R_RI3C_Open(&g_ri3c0_ctrl, &g_ri3c0_cfg);
}

void g_i3c0_user_callback(ri3c_callback_args_t const *const p_args)
{
    user_program();
}
```

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (2) or (4) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (3) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

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

(4) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

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 while 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

R_RI3C_Open()

Configure an RI3C instance.

Format

```
fsp_err_t R_RI3C_Open(
        ri3c_ctrl_t *const p_api_ctrl
        ri3c_cfg_t const *const p_cfg
)
```

Parameters

*ri3c_ctrl_t *const p_api_ctrl*

Pointer to RI3C control block. All elements of this structure are initialized by calling R_RI3C_Open()

ri3c_cfg_t const *const p_cfg*

This is the pointer to RI3C configuration structure. All elements of this structure have been predefined. However, user can change before calling R_RI3C_Open().

Refer to 2.9 Parameters for details on the structures.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was invalid. */*
FSP_ERR_ALREADY_OPEN / Open has already called for this instance */*
FSP_ERR_UNSUPPORTED / A selected feature is not supported with the current configuration. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Configure an RI3C instance.

- Initialize ri3c_instance_ctrl_t.
- Initialize RI3C FIT Module registers allocation.
- Evaluate the bitrate setting.
- Release the RI3C reset bit.
- Generate start condition.

Example

```
fsp_err_t err;

err = R_RI3C_Open(&g_api_ctrl, &g_ri3c0_cfg);
```

Special Notes

None.

R_RI3C_Enable()

Enable the RI3C device.

Format
fsp_err_t R_RI3C_Enable(ri3c_ctrl_t *const p_api_ctrl)

Parameters
*ri3c_ctrl_t *const p_api_ctrl*

Pointer to RI3C control block.

Refer to 2.9. Parameters for details on the structure.

Return Values
FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_INVALID_MODE / This instance is already enabled. */*

Properties
Prototyped in r_ri3c_rx_if.h.

Description
Enable the RI3C device.

- Set the RI3C state flag and status flags.
- Initialize bitrate settings.
- Enable the RI3C interrupts.

Example
```
fsp_err_t err;

err = R_RI3C_Enable(&g_api_ctrl);
```

Special Notes
None

R_RI3C_DeviceCfgSet()

Set the configuration for this device.

Format

fsp_err_t R_RI3C_DeviceCfgSet(

ri3c_ctrl_t *const p_api_ctrl,

ri3c_device_cfg_t const *const p_device_cfg

)

Parameters

p_api_ctrl

Pointer to RI3C control block.

p_device_cfg

Structure for configuring a target address when the driver is in Target mode.

Refer to 2.9. Parameters for details on the structure.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_UNSUPPORTED / The device cannot be a secondary controller if controller support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Set the configuration for this device.
If the device type is controller:
- Set the Dynamic Address of RI3C.

If the device type is target:
- Configure the device static address.
- Configure the device dynamic address.
- Configure the IBI payload setting based on the BCR register.
- Write settings to the Target Device Address Table Register.
- Write the BCR and DCR settings to the Target Device Characteristic Table Register.
- Write the PID setting to the Target Device Characteristic Table PID Register.
- Set the target address to valid.

Example

```
fsp_err_t err;
ri3c_device_cfg_t controller_device_cfg =
{
    /* This is the Static I3C / I2C Legacy address defined by the device
manufacturer. */
    .static_address  = RI3C_CONTROLLER_DEVICE_STATIC_ADDRESS,
    /* The dynamic address will be automatically updated when the controller
configures this device using CCC ENTDAA. */
    .dynamic_address = RI3C_CONTROLLER_DEVICE_DYNAMIC_ADDRESS
};

err = R_RI3C_DeviceCfgSet(
&g_api_ctrl,
&controller_device_cfg
);
```

Special Notes

None.

R_RI3C_ControllerDeviceTableSet()

Configure an entry in the controller device table.

Format

fsp_err_t R_RI3C_ControllerDeviceTableSet(

ri3c_ctrl_t *const p_api_ctrl,

uint32_t device_index,

ri3c_device_table_cfg_t const *const p_device_table_cfg

)

Parameters

p_api_ctrl

Pointer to RI3C control block.

device_index

Index into the device table.

p_device_table_cfg

Pointer to the table settings for the entry in the controller device table.

Refer to 2.9. Parameters for details on the structure.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_UNSUPPORTED / Controller Role requests must be rejected if target support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Configure an entry in the controller device table.

- Configure the IBI settings for this device.
- Configure the device static address.
- Set the device type so that transfers with this device use the legacy I2C protocol.

Example

```
fsp_err_t err;

err = R_RI3C_ControllerDeviceTableSet(
&g_ri3c0_ctrl,
0,
&g_device_table_cfg
);
```

Special Notes

None.

R_RI3C_TargetStatusSet()

Set the status returned to the controller in response to a GETSTATUS command.

Format

fsp_err_t R_RI3C_TargetStatusSet(

ri3c_ctrl_t *const p_api_ctrl,

ri3c_device_status_t status

)

Parameters

p_api_ctrl

Pointer to RI3C control block.

status

The current status of the target device

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_INVALID_MODE / The instance is not in target mode. */*
FSP_ERR_UNSUPPORTED / Target support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Set the status returned to the controller in response to a GETSTATUS command.

- Clear the Pending Interrupt and Vendor Reserved fields in ICDSR register.
- Write the new Pending Interrupt and Vendor Reserved fields in ICDSR register.

Example

```
ri3c_device_status_t g_target_device_status =
{
     .pending_interrupt = 0,
     .vendor_status = 0,
};
fsp_err_t err;

err = R_RI3C_TargetStatusSet(&g_ri3c0_ctrl, g_target_device_status);
```

Special Notes

None.

R_RI3C_DeviceSelect()

In controller mode, select the device for the next transfer.

Format

fsp_err_t R_RI3C_DeviceSelect(
ri3c_ctrl_t *const p_api_ctrl,
uint32_t device_index,
uint32_t bitrate_mode
)

Parameters

p_api_ctrl

Pointer to RI3C control block.

device_index

Index into the device table.

bitrate_mode

The bitrate settings for the selected device.

Refer to “ri3c_bitrate_mode_t” in 2.9 Parameters

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_INVALID_MODE / This operation is prohibited in target mode. */*
FSP_ERR_UNSUPPORTED / Controller support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

In controller mode, select the device for the next transfer.

Example

```
fsp_err_t err;

ret = R_RI3C_DeviceSelect(
&g_ri3c0_ctrl,
0,
RI3C_BITRATE_MODE_RI3C_SDR4_ICEBR_X4
);
```

Special Notes

None.

R_RI3C_DynamicAddressAssignmentStart()

Start the Dynamic Address Assignment Process.

Format

```
fsp_err_t R_RI3C_DynamicAddressAssignmentStart(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_address_assignment_mode_t address_assignment_mode,
        uint32_t starting_device_index,
        uint32_t device_count
)
```

Parameters

p_api_ctrl

Pointer to RI3C control block.

address_assignment_mode

Index into the device table.

starting_device_index

The bitrate settings for the selected device.

device_count

The number of devices to assign (Only used with I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA).

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_INVALID_MODE / This operation is prohibited in target mode. */*
FSP_ERR_IN_USE / The operation could not be completed because the driver is busy. */*
FSP_ERR_UNSUPPORTED / Controller support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Start the Dynamic Address Assignment Process.

Example

```
fsp_err_t err;

err = R_RI3C_DynamicAddressAssignmentStart(
&g_ri3c0_ctrl,
RI3C_ADDRESS_ASSIGNMENT_MODE_SETDASA,
1,
0
);
```

Special Notes

None.

R_RI3C_CommandSend()

Send a broadcast or direct command to target devices on the bus.

Format

```
fsp_err_t R_RI3C_CommandSend(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_command_descriptor_t *p_command_descriptor
)
```

Parameters

p_api_ctrl

Pointer to RI3C control block.

p_command_descriptor

A descriptor for executing the command.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_IN_USE / The operation could not be completed because the driver is busy. */*
FSP_ERR_INVALID_MODE / This operation is prohibited in target mode. */*
FSP_ERR_INVALID_ALIGNMENT / The buffer must be aligned to 4 bytes. If it is a read operation, the length also be a multiple of 4 bytes. */*
FSP_ERR_UNSUPPORTED / Controller support must be enabled to call this function. Target support must be enabled when sending the GETACCMST command. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Send a broadcast or direct command to target devices on the bus bases on contents of Command descriptor. For more information about “Command descriptor”, refer to the application note “RX26T Group User’s Manual: Hardware” (R01UH0979).

Example

```
/* Send Reset Dynamic Address Assignment to all Target devices. */
fsp_err_t err;

ri3c_command_descriptor_t command_descriptor;
command_descriptor.command_code = I3C_CCC_BROADCAST_RSTDAA;
command_descriptor.p_buffer     = NULL;
command_descriptor.length       = 0;
command_descriptor.restart      = false;
command_descriptor.rnw          = false;
err = R_RI3C_CommandSend(
    &g_ri3c0_ctrl,
    &command_descriptor
);
```

Special Notes

Refer to “ri3c_common_command_code_t” in 2.9 Parameters for more information of “command_code” field.

Command buffer must be aligned to 4 bytes.

R_RI3C_Write()

Set the write buffer for the transfer. In controller mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start.

Format

```
fsp_err_t R_RI3C_Write(
        ri3c_ctrl_t *const p_api_ctrl,
        uint8_t const *const p_data,
        uint32_t length,
        bool restart
)
```

Parameters

p_api_ctrl

Pointer to RI3C control block.

p_data

Pointer to a buffer to write.

length

Number of bytes to transfer.

restart

If true, issue a repeated-start after the transfer is completed (Controller only).

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_IN_USE / The operation could not be completed because the driver is busy. */*
FSP_ERR_INVALID_MODE / This driver is disabled. */*
FSP_ERR_INVALID_ALIGNMENT / The buffer must be aligned to 4 bytes. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Set the write buffer for the transfer. In controller mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start.

-.R_RI3C_Write() uses FIFO to transfer data.

Example

```
uint8_t g_write_data[MAX_WRITE_DATA_LEN];

fsp_err_t err;

err = R_RI3C_Write(
   &g_ri3c0_ctrl, /* Pointer to RI3C instance */
    g_write_data, /* Allocated memory for write transfer */
    sizeof(g_write_data), /* Size of allocated transfer memory */
false /* bool value of repeated-start flag */
);
```

Special Notes

None.

R_RI3C_Read()

Set the read buffer for the transfer. In controller mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start.

Format

```
fsp_err_t R_RI3C_Read(
        ri3c_ctrl_t *const p_api_ctrl,
        uint8_t const *const p_data,
        uint32_t length,
        bool restart
)
```

Parameters

p_api_ctrl

Pointer to RI3C control block.

p_data

Pointer to a buffer to store the bytes read during the transfer.

length

Number of bytes to transfer.

restart

If true, issue a repeated-start after the transfer is completed (Controller only).

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_IN_USE / The operation could not be completed because the driver is busy. */*
FSP_ERR_INVALID_MODE / This driver is disabled. */*
FSP_ERR_INVALID_ALIGNMENT / The buffer must be aligned to 4 bytes. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Set the read buffer for the transfer. In controller mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start.

-.R_RI3C_Read() uses FIFO to transfer data.

Example

```
uint8_t g_read_data[MAX_READ_DATA_LEN];

fsp_err_t err;

err = R_RI3C_Read(
   &g_ri3c0_ctrl, /* Pointer to RI3C instance */
    g_read_data,  /* Allocated memory for read transfer */
    sizeof(g_read_data), /* Size of allocated transfer memory */
    false /* bool value of repeated-start flag */
);
```

Special Notes

None.

R_RI3C_IbiWrite()

Initiate an IBI write operation.

Format

```
fsp_err_t R_RI3C_IbiWrite(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_ibi_type_t ibi_type,
        uint8_t const *const p_data,
        uint32_t length
)
```

Parameters

p_api_ctrl

Pointer to RI3C control block.

ibi_type

The type of In-Band Interrupt.

p_data

Pointer to a buffer to start the bytes read during the transfer.

length

Number of bytes to transfer.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_IN_USE / The operation could not be completed because the driver is busy. */*
FSP_ERR_INVALID_MODE / This function is only called in target mode. */*
FSP_ERR_INVALID_ALIGNMENT / The buffer must be aligned to 4 bytes. */*
FSP_ERR_UNSUPPORTED / Target support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Initiate an IBI write operation.

- If the device has an IBI payload, then a buffer is setup for writing the data portion of the IBI.
- Calculate the command descriptor for starting an IBI.
- Write the descriptor to the command queue.

Example

```
fsp_err_t err;

err = R_RI3C_IbiWrite(
   &g_ri3c0_ctrl,
    RI3C_IBI_TYPE_HOT_JOIN,
    NULL,
    0
);
```

Special Notes

None.

R_RI3C_IbiRead()

Set the read buffer for storing received IBI data.

Format

```
fsp_err_t R_RI3C_IbiRead(
        ri3c_ctrl_t *const p_api_ctrl,
        uint8_t *const p_data,
        uint32_t length
)
```

Parameters

p_api_ctrl

Pointer to RI3C control block.

p_data

Pointer to a buffer to start the bytes read during the transfer.

length

Number of bytes to transfer.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*
FSP_ERR_IN_USE / The operation could not be completed because the driver is busy. */*
FSP_ERR_INVALID_MODE / This function is only called in target mode. */*
FSP_ERR_INVALID_ALIGNMENT / The buffer must be aligned to 4 bytes. */*
FSP_ERR_UNSUPPORTED / Target support is disabled. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Set the read buffer for storing received IBI data.

Example

```
uint8_t g_ibi_read_data[MAX_IBI_PAYLOAD_SIZE];

fsp_err_t err;

err = R_RI3C_IbiRead(
   &g_ri3c0_ctrl,
    g_ibi_read_data,
    MAX_IBI_PAYLOAD_SIZE
);
```

Special Notes

This function is only used in controller mode.

R_RI3C_Close()

Close the RI3C instance.

Format

fsp_err_t R_RI3C_Close(
ri3c_ctrl_t *const p_api_ctrl,
)

Parameters

p_api_ctrl

Pointer to RI3C control block.

Return Values

FSP_SUCCESS / Open successful. */*
FSP_ERR_ASSERTION / An argument was NULL. */*
FSP_ERR_NOT_OPEN / This instance has not been opened yet. */*

Properties

Prototyped in r_ri3c_rx_if.h.

Description

Close the RI3C instance.

- Disable RI3C IRQs.
- Disable the I3C bus operation.

Example

```
fsp_err_t err;

err = R_RI3C_Close(&g_api_ctrl);
```

Special Notes

None.

4. Pin Settings

To use the RI3C FIT module, assign input/output signals of the peripheral function to pins with the multi-function pin controller (MPC). The pin assignment is referred to as the “Pin Setting” in this document.

Note: Perform the pin setting after calling R_RI3C_Open().

When performing the pin setting in the e2 studio, the Pin Setting feature of the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the Smart Configurator. Then pins are configured by calling the function defined in the source file. Refer to Table 4.1 Function Output by the Smart Configurator for details.

Table 4.1 Function Output by the Smart Configurator.

MCU Used	Function to be Output	Remarks
RX26T	R_RI3C_PinSet_RI3C0()	

5. Sample Code

5.1 RI3C Controller Basic Example

This is a basic example of minimal use of the RI3C Controller in an application.

The example implements controller device initialization, data transfers and provides a prototype of callback function. The procedure is including allocates memory for module registers, sets the device role as primary controller with bitrate, selects I2C or I3C protocols, assigns dynamic addresses for I3C protocol, enables module interrupt requests. The data is sent to target device from write buffer and receive from read buffer.

```
void ri3c_controller_basic_example (void)
{
    /* Initializes the module. */
    fsp_err_t status = R_RI3C_Open(&g_ri3c_ctrl, &g_ri3c_cfg);
    assert(FSP_SUCCESS == status);
    static ri3c_device_cfg_t controller_device_cfg =
    {
        /* This is the Static I3C / I2C Legacy address defined by the device manufacturer. */
        .static_address  = EXAMPLE_CONTROLLER_STATIC_ADDRESS,
        /* If the device is a main controller, it must configure its own dynamic address. */
        .dynamic_address = EXAMPLE_CONTROLLER_DYNAMIC_ADDRESS,
    };
    status = R_RI3C_DeviceCfgSet(&g_ri3c_ctrl, &controller_device_cfg);
    assert(FSP_SUCCESS == status);
    static ri3c_device_table_cfg_t device_table_cfg =
    {
        /* This is the Static I3C / I2C Legacy address defined by the device manufacturer. */
        .static_address        = EXAMPLE_STATIC_ADDRESS,
        /* Dynamic address is not used in I2C. */
        .dynamic_address       = EXAMPLE_DYNAMIC_ADDRESS,
        /* This is the type of device. It may be either an I2C device or an I3C device. */
        .device_protocol       = I3C_DEVICE_PROTOCOL_I3C,
        .ibi_accept            = false,
        /* Depending on the device the IBI requests may have a data payload.
         * Note that this field will be automatically updated if the device is configured
         * using ENTDAA.
         */
        .ibi_payload           = false,
        /* Controller requests cannot be accepted because Secondary Controller is not supported. */
        .controllerrole_request_accept = false,
    };
    /* Set the device configuration in the controller device table. */
    status = R_RI3C_ControllerDeviceTableSet(&g_ri3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
    /* Enable the RI3C device. */
    status = R_RI3C_Enable(&g_ri3c_ctrl);
    assert(FSP_SUCCESS == status);
    /* Start assigning dynamic addresses to devices on the bus using the ENTDAA command. */
    status = R_RI3C_DynamicAddressAssignmentStart(&g_ri3c_ctrl,
                                                  I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA,
                                                  0,
                                                  1);
    assert(FSP_SUCCESS == status);
    /* Wait for dynamic address assignment to complete. */
    ri3c_app_event_wait(RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
    /* Select the configured device and bitrate mode for the following operations. */
    status = R_RI3C_DeviceSelect(&g_ri3c_ctrl, 0, I3C_BITRATE_MODE_I3C_SDR0_STDBR);
    assert(FSP_SUCCESS == status);
    /* Start a write transfer. */
    static uint8_t p_write_buffer[] = {1, 2, 3, 4, 5};
    status = R_RI3C_Write(&g_ri3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
    /* Wait for the write transfer to complete. */
    ri3c_app_event_wait(RI3C_EVENT_WRITE_COMPLETE);
    /* Start a read transfer. */
    static uint8_t p_read_buffer[16];
    status = R_RI3C_Read(&g_ri3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
    /* Wait for the read transfer to complete. */
    ri3c_app_event_wait(RI3C_EVENT_READ_COMPLETE);
}
```

```
/* This function is called by the I3C driver from ISRs in order to notify the application of I3C
events. */
void ri3c_controller_basic_example_callback (ri3c_callback_args_t const * const p_args)
{
    switch (p_args->event)
    {
        case RI3C_EVENT_ENTDAA_ADDRESS_PHASE:
        {
            /* The device PID, DCR, and BCR registers will be available in
             * ri3c_callback_args_t::p_target_info. */
            break;
        }
        case RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
        {
            ri3c_app_event_notify(RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
            break;
        }
        case RI3C_EVENT_WRITE_COMPLETE:
        {
            ri3c_app_event_notify(RI3C_EVENT_WRITE_COMPLETE);
            break;
        }
        case RI3C_EVENT_READ_COMPLETE:
        {
            /* The number of bytes read from the target will be available in
             * ri3c_callback_args_t::transfer_size. */
            ri3c_app_event_notify(RI3C_EVENT_READ_COMPLETE);
            break;
        }
        default:
        {
            break;
        }
    }
}
```

5.2 RI3C Target Basic Example

This is a basic example of minimal use of the RI3C Target in an application.

The example implements target device initialization and provides a prototype of callback function. The procedure is including allocates memory for module registers, sets the device role as target with a static address and target device information, allocates memory for data transfers.

```
void ri3c_target_basic_example (void)
{
    /* Initializes the module. */
    fsp_err_t status = R_RI3C_Open(&g_ri3c_ctrl, &g_ri3c_cfg);
    assert(FSP_SUCCESS == status);
    static ri3c_device_cfg_t target_device_cfg =
    {
        /* This is the Static I3C / I2C Legacy address defined by the device manufacturer. */
        .static_address  = EXAMPLE_STATIC_ADDRESS,
        /* The dynamic address will be automatically updated when the controller configures
         * this device using ENTDAA. */
        .dynamic_address = 0,
        /* Device Registers that are read by the controller. */
        .target_info     =
        {
            .bcr = EXAMPLE_BCR_SETTING,
            .dcr = EXAMPLE_DCR_SETTING,
            .pid =
            {
                0, 1, 2, 3, 4, 5
            }
        }
    };
    /* Set the device configuration for this device. */
    status = R_RI3C_DeviceCfgSet(&g_ri3c_ctrl, &target_device_cfg);
    assert(FSP_SUCCESS == status);
    /* Enable Target Mode. */
    status = R_RI3C_Enable(&g_ri3c_ctrl);
    assert(FSP_SUCCESS == status);
    static uint8_t p_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
    static uint8_t p_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE];
    /* Set the buffer for storing data received during a read transfer. */
    status = R_RI3C_Read(&g_ri3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
    /* Wait for the controller to complete a read transfer. */
    ri3c_app_event_wait(RI3C_EVENT_READ_COMPLETE);
    /* Set the write buffer that will be transmitted during a write transfer. */
    status = R_RI3C_Write(&g_ri3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
    /* Wait for the controller to complete a write transfer. */
    ri3c_app_event_wait(RI3C_EVENT_WRITE_COMPLETE);
}
```

```
void ri3c_target_basic_example_callback (ri3c_callback_args_t const * const p_args)
{
    switch (p_args->event)
    {
        case RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
        {
            ri3c_app_event_notify(RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
            break;
        }
        case RI3C_EVENT_READ_BUFFER_FULL:
        {
            /* If there is no user provided read buffer, or if the user provided read buffer
             * has been filled, the driver will notify the application that the buffer is full.
             * The application may provide a new read buffer by calling R_RI3C_READ().
             * If no read buffer is provided, then any remaining bytes in the transfer
             * will be dropped. */
            uint8_t * p_read_buffer = ri3c_app_next_read_buffer_get();
            R_RI3C_Read(&g_ri3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE, false);
            break;
        }
        case RI3C_EVENT_READ_COMPLETE:
        {
            /* The number of bytes read by the target will be available in
             * ri3c_callback_args_t::transfer_size. */
            ri3c_app_event_notify(RI3C_EVENT_READ_COMPLETE);
            /* Note that the application may also call R_RI3C_READ() or R_RI3C_WRITE()
             * from this event. In order to set the transfer buffers for the next transfer. */
            break;
        }
        case RI3C_EVENT_WRITE_COMPLETE:
        {
            /* The number of bytes written by the target will be available in
             * ri3c_callback_args_t::transfer_size. */
            ri3c_app_event_notify(RI3C_EVENT_WRITE_COMPLETE);
            /* Note that the application may also call R_RI3C_READ() or R_RI3C_WRITE()
             * from this event. In order to set the transfer buffers for the next transfer.
             */
            break;
        }
        default:
        {
            break;
        }
    }
}
    /* Wait for the read transfer to complete. */
    ri3c_app_event_wait(RI3C_EVENT_READ_COMPLETE);
}
```

6. Appendices

6.1 Operating Test Environment

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

Table 6.1 Operation Test Environment for Rev.1.00

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2022-10 (22.10.0) IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202204 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl, --no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module.
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian order	Little-endian
Module version	Rev.1.00
Board used	Renesas Flexible Motor Control Kit for RX26T (Part Number: RTK0EMXE70S00020BJ)

Table 6.2 Operation Test Environment for Rev.1.10

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2023-04 (23.4.0) IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202204 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl, --no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module.
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian order	Little-endian
Module version	Rev.1.10
Board used	-

Table 6.3 Operation Test Environment for Rev.1.11

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2023-10 (23.10.0) IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202305 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl, --no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module.
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian order	Little-endian
Module version	Rev.1.11
Board used	-

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- When using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”

- When using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. For this, refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_ri3c_rx module.

A: The FIT module you added may not support the target device chosen in the user project. Check if the FIT module supports the target device for the project used.

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_ri3c_rx_config.h” may be wrong. Check the file “r_ri3c_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7 Configuration Overview details.

7. 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 C/C++ Compiler CC-RX User's Manual (R20UT3248)
The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

None.

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	Aug 15, 2022	—	Initial release.
1.10	Jun 30, 2023	23, 46	Deleted the description of FIT configurator from "2.12 Adding the FIT Module to Your Project" and "4. Pin Settings".
		53	6.1 Operating Test Environment: Added Table for Rev.1.10
		Program	Added support for RX26T-256KB (products with 64 Kbytes of RAM)
1.11	Dec 13, 2023	23	2.12. Adding the FIT Module to Your Project: Revised.
		53	6.1 Operating Test Environment: Added Table for Rev.1.11
		Program	Added WAIT_LOOP comments.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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

© 2023 Renesas Electronics Corporation. All rights reserved.