RX Family

RSPIA Module Using Firmware Integration Technology

Introduction

This document covers the RSPIA Module Using Firmware Integration Technology (FIT) for the supported RX family MCUs. Details are provided that describe the RSPIA driver's architecture, integration of the FIT module into a user's application, and how to use the API.

The RX family MCUs supported by this module have a built-in Enhanced Serial Peripheral Interface (RSPIA) for only one channel. The RSPIA performs synchronous serial communication with full duplex or simplex (transmit-only or receive-only). It has a function that performs serial communication with multiple processors and peripheral.

Target Devices

The following is a list of devices that are currently supported by this API:

- RX671 Group
- RX26T Group (Products with 64 Kbytes 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 Confirmed Operation Environment ".

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833)
- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)

Contents

1. Overview

This software provides an application programing interface (API) to prepare the RSPIA peripheral for operation and for performing data transfers over the SPI bus.

The RSPIA FIT module fits between the user application and the physical hardware to take care of the low-level hardware control tasks that manage the RSPIA peripheral.

It is recommended to review the RSPIA peripheral chapter in the RX MCU hardware user's manual before using this software.

1.1 RSPIA FIT Module

The RSPIA FIT module can be used by being implemented in a project as an API. See section 2.13, Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the RSPIA FIT Module

After adding the RSPIA FIT module to your project you will need to modify the *r_rspia_rx_config.h* file to configure the software for your installation. See Section 2.8 Configuration Overview for details on configuration options.

The RSPIA FIT module does not have a function to initialize a register of the I/O port. The setting of the I/O port must be accomplished other than this module. See Section 4, Pin Setting for setting of the I/O port.

When using an RSPIA channel at run time, the first step is to call the R_RSPIA_Open() function by passing the required settings and parameters. On completion, by setting up the I/O ports, the RSPIA channel will be active and ready to perform all other functions available in this API. SPI Data transfer operations may be used at this time, or various control operations may be performed to change settings (Note 1).

Note 1: When using in clock synchronous operation (3-wire method) and in master mode, follow the procedure below to prepare for data transmission. Otherwise, the synchronization gap of the clock may occur.
(1) Disable the slave for communication (For RSPIA slave, set SPE=0)
(2) Call R_RSPIA_Open(), note that wait for this operation to be completed
(3) Set the pins to peripheral module by I/O ports setting
(4) Enable the slave for communication

Setting of the RSPIA register is executed by calling R_RSPIA_Open(). As it intended to general-purpose use, the register's default value should be set in the RSPIA register. Also, by calling R_RSPIA_Control(), RSPIA register information stored in RSPIA FIT module can be rewritten.

Five commands are provided in the R_RSPIA_Control() function:

- Change the base bit-clock rate.
- Immediately abort a transfer operation.
- Rewrite RSPIA register information.
- Change the transmit FIFO threshold.
- Change the receive FIFO threshold.

When data transfers are performed over the SPI bus the driver informs the user's application of the completion status by calling the user-provided callback function.
Most of the RSPIA API functions will require a 'handle' argument. This is used to identify the RSPIA channel number that is selected for the operation. A handle is obtained by first calling the R_RSPIA_Open() function. You must provide the address of a location where you will store the handle to R_RSPIA_Open(), and on completion the handle will be available for use. Thereafter, simply pass the provided handle value for that RSPIA channel number to the other API functions when calling them. In your application you will need to keep track of which handle belongs to a given channel, as each channel will be assigned its own handle.

1.2.1 Features Supported

This driver supports the following subset of the features available with the RSPIA peripheral.

RSPIA transfer functions:

- Use of MOSI (master out/slave in), MISO (master in/slave out), SSL (slave select), and RSPCK (RSPIA clock) signals allows serial communications through SPI operation (four-wire method) or clock synchronous operation (three-wire method).
- Full-duplex or simplex (transmit-only or receive-only) communications can be selected.
- Capable of serial communications in master/slave mode
- Switching of the polarity of the serial transfer clock
- Switching of the phase of the serial transfer clock
- Three transfer modes are provided: SW (Software), DMAC (Direct Memory Access Controller), and DTC (Data Transfer Controller).

Data format:

- MSB-first/LSB-first selectable
- Transfer bit length can be changed to 4 through 32 bits.
- Transmit buffer size/receive buffer size: 32 bits × 4 stages FIFO.

Bit rate:

- In master mode, the on-chip baud rate generator generates RSPCK by frequency-dividing PCLK (Division ratio: 2 to 4096).
- In slave mode, the externally input clock is used as the serial clock (for maximum frequency, refer to MCU User's manual).

Error detection:

- Mode fault error detection
- Overrun error detection
- Parity error detection
- Under run detection
- Receive data ready detection

SSL control function:

- Four SSL signals (SSL00 to SSL03) for each channel
- In single-master mode: SSL00 to SSL03 signals are output.
- In slave mode: SSLn0 signal for input and SSL01 to SSL03 signals are Hi-Z (not used).
- Controllable delay from SSL output assertion to RSPCK operation (RSPCK delay)
  Range: 1 to 8 RSPCK cycles (set in RSPCK-cycle units)
- Controllable delay from RSPCK stop to SSL/OE output negation (SSL/OE negation delay)
  Range: 1 to 8 RSPCK cycles (set in RSPCK-cycle units)
- Controllable wait for next-access SSL output assertion (next-access delay)
  Range: 1 to 8 RSPCK cycles (set in RSPCK-cycle units)
- Able to change SSL polarity

Communication Protocol:

- Motorola SPI
- TI SSP (Synchronous Serial Protocol)

Control in master transfer:

- For each transfer operation, the following can be set: Slave select number, further division of base bit rate, SPI clock polarity/phase, transfer data bit length, MSB/LSB-first, burst (holding SSL), SPI clock delay, slave select negation delay, and next access Delay

1.2.2 Features Not Supported

- To conserve limited RAM resources of smaller memory MCUs, this driver requires that data buffers are not statically allocated by the driver, but rather must be allocated by the user application at a higher level. This gives the application the control of how to allocate RAM.
- Only single-sequence data transfers are supported. the multi-command-sequence data transfer features of the RSPIA peripheral are not supported by this driver.
- Byte swap for 16-bit type is not supported.

1.3 Using the RSPIA FIT module

1.3.1 Using RSPIA FIT module in C++ project

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

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

1.4 API Overview

Table 1.1 lists the API functions included in this module. Also, section 2.9, Code Size, lists the size of the code sections used by this module.

Table 1.1 API Functions

Function	Function Description
R_RSPIA_Open()	Initializes the associated registers required to prepare the specified RSPIA channel for use, provides the handle for use with other API functions. Takes a callback function pointer for responding to interrupt events.
R_RSPIA_Control()	Handles special hardware or software operations for the RSPIA channel
R_RSPIA_Read()	The Read function receives data from a SPI master or slave device.
R_RSPIA_Write()	The Write function transmits data to a SPI master or slave device.
R_RSPIA_WriteRead()	The Write Read function simultaneously transmits data to a SPI master or slave device while receiving data from that device (full duplex).
R_RSPIA_Close()	Disables the specified RSPIA channel.
R_RSPIA_GetVersion()	Returns the driver version number.
R_RSPIA_IntSptiIerClear()	SPTI transmit interrupt request disable processing
R_RSPIA_IntSpriIerClear()	SPRI receive interrupt request disable processing
R_RSPIA_DisableSpti()	Disables the generation of transmit buffer empty interrupt requests
R_RSPIA_DisableRSPI()	Disables the RSPI function
R_RSPIA_GetBuffRegAddress()	SPDR register address acquisition processing

1.5 Driver Architecture

1.5.1 System Examples

The driver supports single-master/multi-slave mode operation, or slave-mode operation. Each RSPIA channel controls one SPI bus. Multiple-master operation on the same bus is not supported in this driver. An example of a single master connected to multiple slaves on one SPI bus is shown.

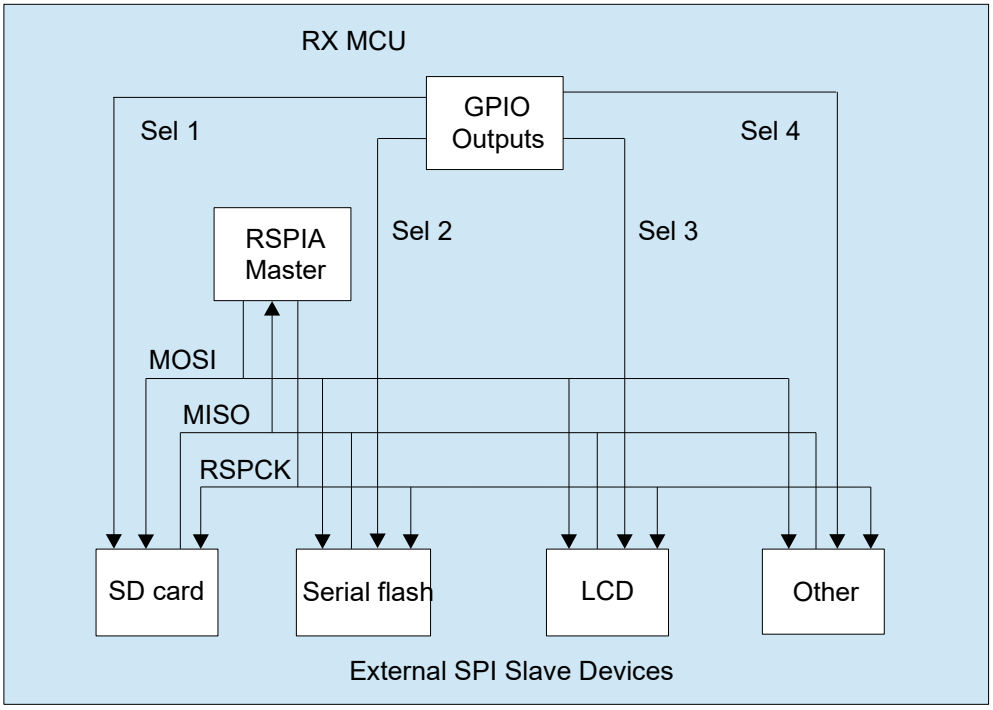

Figure 1.1 : This example shows the use of GPIO ports to serve as the slave select signals (3- Wire-mode)

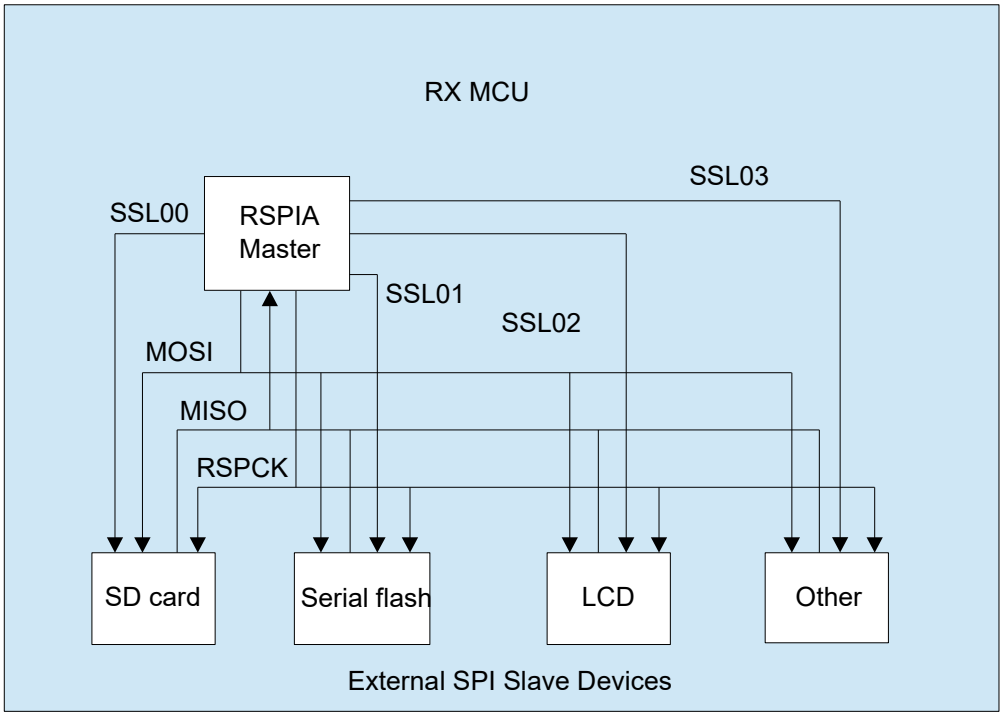

Figure 1.2 : The built-in RSPIA peripheral slave select hardware (SSL) may be used to generate the signals (SPI 4-Wire mode)

1.6 Basic Operation (SW Transfer)

The RSPIA FIT module provides three data transfer functions available for both master and slave mode operation: Write, Read, Write/Read (full duplex). The module uses the non-blocking method, which starts communication by calling these functions, and obtains notification of the communication result using a callback function. The callback function occurs when operation has been successfully initialized or there is an error.

If locking has been enabled by the configuration option, the RSPIA channel will be locked for the duration of the operation. After that, the remainder of the transfer operation is performed within RSPIA interrupt handler routines.

1.7 Basic Operations (In DMAC/DTC)

The RSPIA FIT module can perform transfer of data using DMAC/DTC (write data to the SPDR register or read data from the SPDR register). When using DMAC/DTC, first set RSPIA_TRANS_MODE_DMAC or RSPIA_TRANS_MODE_DTC to the second argument pconfig -> tran_mode of the R_RSPIA_Open() function (Note 1). Also, set DMAC/DTC in advance (Note 2).

The communication start method is the same as SW transfer. The method of notifying the communication result differs between DMAC and DTC.

- Termination of DMAC communication

When communication is completed normally, a DMAC transfer end interrupt occurs, and the callback function registered in the DMAC FIT module is called. In data transfer using DMAC, the callback function registered in the RSPIA FIT module is not called. In case of a communication error, an RSPIA error interrupt occurs and the callback function of the RSPIA FIT module is called.

- Termination of DTC communication

When communication is completed normally, an RSPIA transmit buffer empty interrupt or receive buffer full interrupt occurs and the callback function registered in the RSPIA FIT module is called. In case of a communication error, an RSPIA error interrupt occurs and the callback function of the RSPIA FIT module is called.

Note 1 After calling the R_RSPIA_Open() function, you can change the data transfer method by calling the R_RSPIA_Control() function.

Note 2 For the setting method, see the sample program included in the DMAC/DTC FIT application note or the RSPIA FIT application note. Use the block transfer mode to configure for DMAC/DTC, and the number of data frames transmitted is a multiple of the block size. In which block size will be equal to the value of RSPIA_CFG_CH0_TX_FIFO_THRESH+1 or RSPIA_CFG_CH0_RX_FIFO_THRESH+1, depending on the setting value of those macros in the file "r_rspia_rx_config.h".

2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires your MCU support the following features.

This section details the hardware peripherals that this driver requires. Unless explicitly stated, these resources must be reserved for the driver and the user cannot use them independently.

- One or more available RSPIA peripheral channels.

2.2 Software Requirements

This driver is dependent upon the support from the following software:

- This software depends on a FIT-compliant BSP module Rev.6.10 or higher. The related I/O ports should be correctly initialized elsewhere after calling the R_RSPIA_Open() of this software.
- This software requires that the peripheral clock (PCLK) has been initialized by the BSP prior to calling the APIs of this module. The r_bsp macro ‘BSP_PCLKx_HZ’ is used by the driver for calculating bit-rate register settings. If the user modifies the PCLKx setting outside of the r_bsp module, then calculations on the bit rate will be invalid.

2.3 Limitations

2.3.1 RAM Location Limitations

In FIT, if a value equivalent to NULL is set as the pointer argument of an API function, error might be returned due to parameter check. Therefore, do not pass a NULL equivalent value as pointer argument to an API function.

The NULL value is defined as 0 because of the library function specifications. Therefore, the above phenomenon would occur when the variable or function passed to the API function pointer argument is located at the start address of RAM (address 0x0). In this case, change the section settings or prepare a dummy variable at the top of the RAM so that the variable or function passed to the API function pointer argument is not located at address 0x0.

In the case of the CCRX project (e2 studio V7.5.0), the RAM start address is set as 0x4 to prevent the variable from being located at address 0x0. In the case of the GCC project (e2 studio V7.5.0) and IAR project (EWRX V4.12.1), the start address of RAM is 0x0, so the above measures are necessary.

The default settings of the section may be changed due to the IDE version upgrade. Please check the section settings when using the latest IDE.

2.4 Supported Toolchain

The operation of RSPIA FIT module has been confirmed with the toolchain listed in “6.1 Confirmed Operation Environment”.

2.5 Interrupt Vector

When running the R_RSPIA_Open() function, the interrupt according to the argument channel and the interrupt occurrence factor is enabled.

Table 2.1 lists the interrupt vectors used in the FIT Module.

Table 2.1 Interrupt Vector

Device	Interrupt Vector
RX671 RX26T	SPRI interrupt (vector no.: 48) SPTI interrupt (vector no.: 49)

2.6 Header Files

All API calls are accessed by including a single file "r_rspia_rx_if.h" which is supplied with this software's project code.

Build-time configuration options are selected or defined in the file "r_rspia_rx_config.h"

2.7 Integer Types

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

2.8 Configuration Overview

The configuration option settings of this module are located in r_rspia _rx_config.h. The option names and setting values are listed in the table below:

Configuration options in r_rspia _rx_config.h	
Definition	**Description**
RSPIA_CFG_PARAM_CHECKING_ENABLE 1	1: Parameter checking is included in the build. 0: Parameter checking is omitted from the build. Setting this #define to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting.
RSPIA_CFG_REQUIRE_LOCK 1	If this is set to (1) then the RSPIA driver will attempt to obtain the lock for the channel when performing certain operations to prevent concurrent access conflicts.
RSPIA_CFG_DUMMY_TXDATA 0xFFFFFFFF	The user-specified Dummy Data to be transmitted during receive-only operations.
RSPIA_CFG_USE_CH0 1	Enable the RSPIA channels to use at build-time. (0) = not used. (1) = used. **Note that must enable at least 1 channel for use. Be sure to enable the channels you will be using in the config file.**
RSPIA_CFG_CH0_IR_PRIORITY 3	Sets the shared interrupt priority for the channel. This is provided as a convenience. Priority can still be changed outside of this module at run time after a call to R_RSPIA_Open has been made to a channel. However, the next call to R_RSPIA_Open for that channel will change it back to this configuration value.
RSPIA_CFG_CH0_TX_FIFO_THRESH 1	SET TX FIFO THRESHOLD; (RSPIA supported MCU ONLY) 0 lowest, 3 highest Set the same value for TX FIFO THRESHOLD and RX FIFO THRESHOLD.
RSPIA_CFG_CH0_RX_FIFO_THRESH 1	SET RX FIFO THRESHOLD; (RSPIA supported MCU ONLY) 0 lowest, 3 highest

2.9 Code Size

Typical code sizes associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.8, Configuration Overview. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.4, Supported Toolchain. 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.

ROM, RAM, and Stack Code Sizes							
		Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
Device	Category	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX671	ROM	3056 bytes	2729 bytes	3796 bytes	3308 bytes	5501 bytes	5105 bytes
	RAM	68 bytes	68 bytes	108 bytes	108 bytes	60 bytes	60 bytes
	Maximum stack usage	80 bytes	72 bytes	- bytes	- bytes	160 bytes	156 bytes
RX26T	ROM	3090 bytes	2763 bytes	3804 bytes	3316 bytes	5505 bytes	5105 bytes
	RAM	68 bytes	68 bytes	108 bytes	108 bytes	60 bytes	60 bytes
	Maximum stack usage	92 bytes	84 bytes	- bytes	- bytes	160 bytes	156 bytes

2.10 Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_rspia_rx_if.h as are the prototype declarations of API functions.

Channel Settings structure for Open

The R_RSPIA_Open() function requires a pointer to an initialized instance of this structure to set certain operating modes at the channel open.

```
typedef struct rspia_chnl_settings_s
{
    rspia_interface_mode_t gpio_ssl;                /* RSPIA_IF_MODE_4WIRE:
RSPIA slave selects, RSPIA_IF_MODE_3WIRE: GPIO slave selects. */
    rspia_master_slave_mode_t master_slave_mode;    /* RSPIA_MS_MODE_MASTER or
RSPIA_MS_MODE_SLAVE. */
    rspia_frame_select_t frame_mode;                /* RSPIA_IF_FRAME_TI_SSP or
RSPIA_IF_FRAME_MOTOROLA_SPI. */
    uint32_t    bps_target;                         /* The target bits per
second setting value for the channel. */
    rspia_str_tranmode_t       tran_mode;           /* Data transfer mode. */
} rspia_chan_settings_t;
```

Abstraction of channel handle data structure

User application will use this as a reference to an opened channel.

```
typedef struct rspia_cfg_block_s *rspia_hdl_t;


typedef struct rspia_cfg_block_s
{
    uint8_t chan;
    uint8_t current_slave;                  /* Number of the currently assigned
slave. */
    bool rspia_chan_opened;                 /* This variable determines whether
the peripheral has already been initialized. */
    void   (*p_callback)(void *p_cbdat);    /* pointer to user callback function.
*/
} rspia_cfg_block_t;
```

Typedef enumerations used for the command settings word

This is list contains the enumerated types available for specific settings of the command word for write and read operations. The command word is a 32-bit value that is a collection of bit fields.

This contains one of each of the above types in the correct order to set all the bits of the SPCMD register.

```
typedef union rspia_command_word_s
{
    R_BSP_ATTRIB_STRUCT_BIT_ORDER_RIGHT_14(
        rspia_spcmd_cpha_t         cpha          :1,
        rspia_spcmd_cpol_t         cpol          :1,
        rspia_spcmd_br_div_t       br_div        :2,
        rspia_spcmd_reserve_bit_t  rs0           :3,    /* reserved */
        rspia_spcmd_ssl_negation_t ssl_negate    :1,
        rspia_spcmd_reserve_bit_t  rs1           :4,    /* reserved */
        rspia_spcmd_bit_order_t    bit_order     :1,
        rspia_spcmd_spnden_t       next_delay    :1,
        rspia_spcmd_slnden_t       ssl_neg_delay :1,
        rspia_spcmd_sckden_t       clock_delay   :1,
        rspia_spcmd_bit_length_t   bit_length    :5,
        rspia_spcmd_reserve_bit_t  rs2           :3,    /* reserved */
        rspia_spcmd_ssl_assert_t   ssl_assert    :3,
        rspia_spcmd_reserve_bit_t  rs3           :5     /* reserved */
    );
    uint32_t word[1];
} rspia_command_word_t;
```

Example of command word initialization

```
static const rspia_command_word_t my_command_reg_word = {
    RSPIA_SPCMD_CPHA_SAMPLE_ODD,
    RSPIA_SPCMD_CPOL_IDLE_LO,
    RSPIA_SPCMD_BR_DIV_1,
    RSPIA_SPCMD_RESERVE_BIT,
    RSPIA_SPCMD_SSL_KEEP,
    RSPIA_SPCMD_RESERVE_BIT,
    RSPIA_SPCMD_ORDER_MSB_FIRST,
    RSPIA_SPCMD_NEXT_DLY_SSLND,
    RSPIA_SPCMD_SSL_NEG_DLY_SSLND,
    RSPIA_SPCMD_CLK_DLY_SPCKD,
    RSPIA_SPCMD_BIT_LENGTH_8,
    RSPIA_SPCMD_RESERVE_BIT,
    RSPIA_SPCMD_ASSERT_SSL0,
    RSPIA_SPCMD_RESERVE_BIT,
};
```

Note that ignore these values of reserve bits have defined. User don’t need enter these enumerated types.

Callback function data structure

The channel number and the procedure result code are passed in this data structure to the user defined. callback function.

```
typedef struct rspia_callback_data_s
{
    rspia_hdl_t hdl;    /* The channel handle */
    rspia_evt_t event;  /* The event code */
}rspia_callback_data_t;
```

2.11 Return Values

This section describes return values of API functions. This enumeration is located in r_rspia_rx_if.h as are the prototype declarations of API functions.

```
typedef enum rspia_err_e        /* RSPIA API error codes */
{
    RSPIA_SUCCESS = 0,
    RSPIA_ERR_BAD_CHAN,         /* Invalid channel number. */
    RSPIA_ERR_OMITTED_CHAN,     /* RSPIA_USE_CHx is 0 in config.h */
    RSPIA_ERR_CH_NOT_OPENED,    /* Channel not yet opened. */
    RSPIA_ERR_CH_NOT_CLOSED,    /* Channel still open from previous open. */
    RSPIA_ERR_UNKNOWN_CMD,      /* Control command is not recognized. */
    RSPIA_ERR_INVALID_ARG,      /* Argument is not valid for parameter. */
    RSPIA_ERR_ARG_RANGE,        /* Argument is out of range for parameter. */
    RSPIA_ERR_NULL_PTR,         /* Received null pointer; missing required
                                   argument. */
    RSPIA_ERR_LOCK,             /* The lock procedure failed. */
    RSPIA_ERR_UNDEF,            /* Undefined/unknown error */
} rspia_err_t;
```

2.12 Callback Function

In this module, the callback function specified by the user is called when the SPRI, SPTI, and SPEI interrupt occurs.

The callback function is specified by storing the address of the user function in the “void (*p_callback)(void *p_cbdat)” structure member (see 2.10, Parameters). When the callback function is called, the variable which stores the constant is passed as the argument.

The argument is passed as void type. Thus the argument of the callback function is cast to a void pointer. See examples below as reference.

When using a value in the callback function, type cast the value.

The following shows an example template for the callback function.

```
void my_callback(void *p_args)
{
    rspia_callback_data_t  *args;

    args = (rspia_callback_data_t *)p_args;
    callback_called_flag = true;


    if (args->event == RSPIA_EVT_TRANSFER_COMPLETE)
    {
        /* From SPRI interrupt. */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_TRANSFER_ABORTED)
    {
        /* The data transfer was aborted. */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_MODE_FAULT)
    {
        /* From SPEI interrupt; a mode fault error occur */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_READ_OVF)
    {
        /* From SPEI interrupt; a read overflow error occur */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_PARITY)
    {
        /* From receiver parity error interrupt; Error condition is cleared in
           calling interrupt routine */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_UNDER_RUN)
    {
        /* From underrun error interrupt; Error condition is cleared in calling
           interrupt routine */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_UNDEF)
    {
        /* Undefined/unknown error event. */
        R_BSP_NOP();
    }
}
```

2.13 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), (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 “Renesas e^2 studio Smart Configurator User Guide (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 “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” 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 “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.

2.14 “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 */
```

2.15 Peripheral Functions and Modules Other than RSPIA

In addition to the RSPIA, the RSPIA FIT module can be used in combination with the following peripheral functions and modules.

- DMA controller (DMAC)
- Data transfer controller (DTC)

2.15.1 DMAC/DTC

The control method when using DMAC transfer or DTC transfer is described below.

The RSPIA FIT module sets the ICU.IERm.IENj bit to 1 to start a DMAC transfer or DTC transfer and then waits for the transfer to end. Other settings to DMAC registers or DTC registers can be performed by using the DMAC FIT module or DTC FIT module, or by using a custom processing routine created by the user.

Note that in the case of DMAC transfer settings, clearing of the ICU.IERm.IENj bit and clearing of the transfer-end flag must be performed by the user after the DMAC transfer has finished.

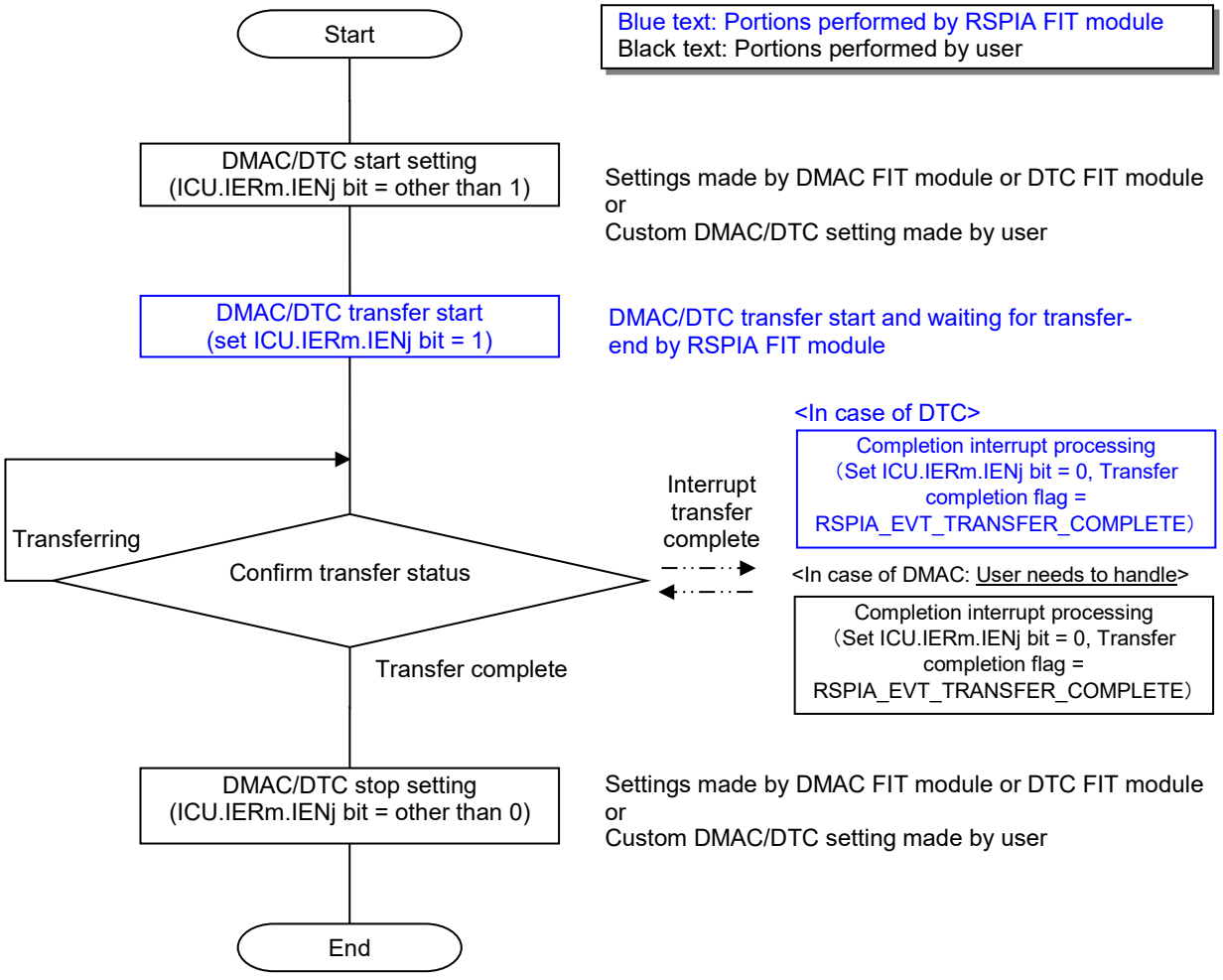

Figure 2-1 Processing for DMAC Transfer and DTC Transfer Settings

3. API Functions

R_RSPIA_Open()

This function applies power to the RSPIA channel, initializes the associated registers, enables interrupts, and provides the channel handle for use with other API functions.

Format

```
rspia_err_t   R_RSPIA_Open(uint8_t                chan,
                           rspia_chan_settings_t *p_cfg,
                           rspia_command_word_t  p_cmd,
                           void                  (*p_callback)(void *p_cbdat),
                           rspia_hdl_t          *p_hdl);
```

Parameters

channel

Number of the RSPIA channel to be initialized.

**p_cfg*

Pointer to RSPI channel configuration data structure.

p_cmd

SPCMD command data structure.

p_callback

Pointer to user defined function called from interrupt.

**p_hdl*

Pointer to a handle for channel. Handle value will be set by this function.

Return Values

RSPIA_SUCCESS	/* Successful: channel initialized. */
RSPIA_ERR_BAD_CHAN	/* Channel number is not available. */
RSPIA_ERR_OMITTED_CHAN	/* RSPIA_CFG_USE_CHx is 0 in config.h. */
RSPIA_ERR_CH_NOT_CLOSED	/* Channel currently in operation; Perform R_RSPIA_Close() first. */
RSPIA_ERR_NULL_PTR	/* *p_cfg pointer or *p_hdl pointer is NULL. */
RSPIA_ERR_ARG_RANGE	/* An element of the *p_cfg structure contains an invalid value. */
RSPIA_ERR_INVALID_ARG	/* Argument is not valid for parameter. */
RSPIA_ERR_LOCK	/* The lock could not be acquired. */

Properties

The declaration is located in r_rspia_rx_if.h.

Description

The Open function is responsible for preparing an RSPIA channel for operation. This function must be called once prior to calling any other RSPIA API functions (except R_RSPIA_GetVersion). Once successfully completed, the status of the selected RSPI will be set to "open". After that, this function should not be called again for the same RSPIA channel without first performing a "close" by calling R_RSPIA_Close(). Communication is not yet available upon completion of this processing. Set MPC and PMR in the I/O ports to peripheral module.

Example

Condition: Channel not yet open

```
    /* Configure the RSPIA settings structure */
    g_rspia_cfg.bps_target = 1000000;                    // Ask for 1Mbps bit-
rate.
    g_rspia_cfg.master_slave_mode = RSPIA_MS_MODE_MASTER; // Configure the RSPIA
as SPI Master.
#if RSPIA_CFG_USE_GPIO_SSL == (0)
    g_rspia_cfg.gpio_ssl = RSPIA_IF_MODE_4WIRE;          // Set interface mode
to 4-wire.
    g_rspia_cfg.frame_mode = RSPIA_IF_FRAME_MOTOROLA_SPI; // Set communication
protocol .
#else
```

```
    g_rspia_cfg.gpio_ssl = RSPIA_IF_MODE_3WIRE;           // Set interface mode
to 3-wire.
#endif /* RSPIA_CFG_USE_GPIO_SSL == (0) */
    g_rspia_cfg.tran_mode = RSPIA_TRANS_MODE_SW; // Data transfer mode.

    /* Open the RSPIA channel using the API function */
    error = R_RSPIA_Open (RSPIA_CH0,&g_rspia_cfg, g_rspia_cmd,
&my_rspia_callback, &g_rspia_hdl);

    /* If there were an error this would demonstrate error detection of API
calls. */
    if (RSPIA_SUCCESS != error)
    {
        return error(); // Your error handling code would go here.
    }

    /* Initialize I/O port pins for use with the RSPIA peripheral.
     * This is specific to the MCU and ports chosen. */
    rspia_rx671_init_ports();
```

Special Notes

Take note of the following points when specifying DMAC transfer or DTC transfer.

- The DMAC FIT module, and DTC FIT module must be obtained separately.

R_RSPIA_Control()

The Control function is responsible for handling special hardware or software operations for the RSPIA channel.

Format

```
rspia_err_t  R_RSPIA_Control(rspia_hdl_t  hdl,
                             rspia_cmd_t     cmd,
                             void           *pcmd_data);
```

Parameters

hdl

Handle for the channel

cmd

Enumerated command code.
Available command codes:
RSPIA_CMD_SET_BAUD: Change the base bit rate setting without reinitializing the RSPIA channel.
RSPIA_CMD_SET_REGS: Set all supported RSPIA registers in one operation. (Expert use only)
RSPIA_CMD_CHANGE_TX_FIFO_THRESH: Change Frame TX FIFO threshold.
RSPIA_CMD_CHANGE_RX_FIFO_THRESH: Change Frame RX FIFO threshold.
RSPIA_CMD_SET_TRANS_MODE: Set the data transfer mode.

**pcmd_data*

Pointer to the command-data structure parameter of type void that is used to reference the location of any data specific to the command needed for its completion. Commands that do not require supporting data must use the FIT_NO_PTR.

The valid *cmd* values are as follows:

```
typedef enum rspia_cmd_e
{
    RSPIA_CMD_SET_BAUD = 1,             /* Change the base bit rate */
    RSPIA_CMD_ABORT,                    /* Stop the current read or write
                                           operation immediately. */
    RSPIA_CMD_SET_REGS,                 /* Set all supported RSPIA register in
                                           one operation. Expert use only! */
    RSPIA_CMD_CHANGE_TX_FIFO_THRESH,    /* change TX FIFO threshold */
    RSPIA_CMD_CHANGE_RX_FIFO_THRESH,    /* change TX FIFO threshold */
    RSPIA_CMD_SET_TRANS_MODE,           /* Set the data transfer mode */
    RSPIA_CMD_UNKNOWN                   /* Not a valid command. */
} rspia_cmd_t;
```

Commands other than the following command do not require arguments and take FIT_NO_PTR for pcmd_data.

The argument for RSPIA_CMD_SET_BAUD is a pointer to the rspia_cmd_baud_t variable containing the new bit rate desired. Data structure for the set baud command is shown below.

```
typedef struct rspia_cmd_baud_s
{
    uint32_t    bps_target;     /* The target bits per second setting value for
                                   the channel. */
} rspia_cmd_baud_t;
```

The argument for RSPIA_CMD_CHANGE_TX_FIFO_THRESH and RSPIA_CMD_CHANGE_RX_FIFO_THRESH (for MCU which can specify different FIFO thresh levels for transmit and receive buffer) is a pointer to a uint8_t variable to hold the thresh level.

The argument for RSPIA_CMD_SET_REGS is a pointer to the rspia_cmd_setregs_t variable containing these values to set the RPSIA registers. To use RSPIA_CMD_SET_REGS command, create the instance with as-needed setting value first, then call R_RSPIA_Control() to pass the pointer as argument.

```
typedef struct rspia_cmd_setregs_s
{
    uint16_t spdcr_val;     /* RSPIA Data Control Register (SPDCR) */
    uint8_t sslp_val;       /* RSPIA Slave Select Polarity Register (SSLP) */
    uint8_t sppcr_val;      /* RSPIA Pin Control Register (SPPCR) */
    uint8_t spckd_val;      /* RSPIA Clock Delay Register (SPCKD) */
    uint8_t sslnd_val;      /* RSPIA Slave Select Negation Delay Register
                               (SSLND) */
    uint8_t spnd_val;       /* RSPIA Next-Access Delay Register (SPND) */
} rspia_cmd_setregs_t;
```

Data structure for RSPIA_CMD_SET_TRANS_MODE command. This command is used to change the setting of data transfer mode. There are three kinds of mode for RSPIA_TRANS_MODE_SW, RSPIA_TRANS_MODE_DMAC and RSPIA_TRANS_MODE_DTC.

```
typedef struct rspia_cmd_trans_mode_s
{
    uint8_t    transfer_mode;   /* The transfer mode setting value for the
channel. */
} rspia_cmd_trans_mode_t;
```

Return Values

RSPIA_SUCCESS	/* Command successfully completed. */
RSPIA_ERR_CH_NOT_OPENED	/* The channel has not been opened. Perform R_RSPIA_Open() first. */
RSPIA_ERR_UNKNOWN_CMD	/* Control command is not recognized. */
RSPIA_ERR_NULL_PTR	/* *pcmd_data pointer or *p_hdl pointer is NULL. */
RSPIA_ERR_ARG_RANGE	/* An element of the *pcmd_data structure contains an invalid value. */
RSPIA_ERR_LOCK	/* The lock could not be acquired. */

Properties

The declaration is located in r_rspia_rx_if.h.

Description

This function is responsible for handling special hardware or software operations for the RSPIA channel. It takes an RSPIA handle to identify the selected RSPIA, an enumerated command value to select the operation to be performed, and a void pointer to a location that contains information or data required to complete the operation. This pointer must point to storage that has been type-cast by the caller for the particular command using the appropriate type provided in "r_rspia_rx_if.h".

Example

```
my_setbaud_struct.bps_target = 4000000; // Set for 4 Mbps
result = R_RSPIA_Control(handle, RSPIA_CMD_SET_BAUD, &my_setbaud_struct);
if (RSPIA_SUCCESS != result)
{
      return result;
}
...
/* This is taking too long, stop the current transfer now! */
result = R_RSPIA_Control(handle, RSPIA_CMD_ABORT, FIT_NO_PTR);
```

Special Notes

None.

R_RSPIA_Read()

The Read function receives data from the selected SPI device.

Format

```
rspia_err_t  R_RSPIA_Read(rspia_hdl_t         hdl,
                          rspia_command_word_t  p_cmd,
                          void                 *p_dst,
                          uint16_t              length);
```

Parameters

hdl

Handle for the channel

p_cmd

Bit field data consisting of all the RSPIA command register settings for SPCMD for this operation. See 2.10 Parameters used for the command settings word.

**p_dst*

Void type pointer to a destination buffer into which data will be copied that has been received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL. Based on the data frame bit-length specified in the p_cmd.bit_length, the *p_dst pointer will be type cast to the corresponding data type during the transfer. So, for example, if the bit-length is set to 16-bits, then the data will be stored in the destination buffer as a 16-bit value, and so on for each bit-length setting. Bit-length settings that are not 8, 16 or 32 will use the smallest data type that they can be contained within. For example, 24-bit frames will be stored in 32-bit storage, 11-bit frames will be stored in 16-bit storage, etc.

length

Transfer length variable to indicate the number of data frames to be transferred. The size of the data word is determined from settings in the p_cmd.bit_length argument. Be sure that the length argument matches the storage type of the source data; this is a count of the number of frames, not the number of bytes.

Return Values

RSPIA_SUCCESS	/* Read operation successfully completed. */
RSPIA_ERR_CH_NOT_OPENED	/* The channel has not been opened. Perform R_RSPIA_Open() first. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */
RSPIA_ERR_LOCK	/* The lock could not be acquired. The channel is busy. */
RSPIA_ERR_INVALID_ARG	/* Argument is not valid for parameter. */

Properties

The declaration is located in r_rspia_rx_if.h.

Description

Starts reception of data from a SPI device. The function returns immediately after the operation begins, and data will continue to be received in the background under interrupt control until the requested length has been received. Received data is stored in the destination buffer. When the transfer is complete the user defined callback function is called.

Operation differs slightly depending on whether the RSPIA is operating as Master or Slave. If the RSPIA is configured as slave, then data will only transfer when clocks are received from the Master. While receiving data, the RSPIA will also transmit the user definable dummy data pattern defined in the configuration file.

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;
result = R_RSPIA_Read(handle, my_command_word, dest, length);
if (RSPIA_SUCCESS != result)
{
      return result;
}
while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
      // Do something useful while waiting for the transfer to complete.
      R_BSP_NOP();
}
```

Special Notes:

Add the following processing when specifying DMAC transfer or DTC transfer.

- For the callback function that occurs when communication ends, see 1.7, Basic Operations (In DMAC/DTC).
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.

R_RSPIA_Write()

The Write function transmits data to the selected SPI device.

Format

```
rspia_err_t  R_RSPIA_Write(rspia_hdl_t         hdl,
                           rspia_command_word_t  p_cmd,
                           void                 *p_src,
                           uint16_t              length);
```

Parameters

hdl

Handle for the channel

p_cmd

Bit field data consisting of all the RSPIA command register settings for SPCMD for this operation. See 2.10 Parameters used for the command settings word.

**p_src*

Void type pointer to a source data buffer from which data will be transmitted to a SPI device. Based on the data frame bit-length specified in the p_cmd.bit_length, the *p_src pointer will be type cast to the corresponding data type during the transfer. So, for example, if the bit-length is set to 16-bits, then the source buffer data will be accessed as a block of 16-bit data, and so on for each bit-length setting. Bit-length settings that are not 8, 16 or 32 will use the data type that they can be contained within. For example, 24-bit frames will be stored in 32-bit storage, 11-bit frames will be stored in 16-bit storage, etc.

length

Transfer length variable to indicate the number of data frames to be transferred. The size of the data word is determined from settings in the p_cmd.bit_length argument. Be sure that the length argument matches the storage type of the source data; this is a count of the number of frames, not the number of bytes.

Return Values

RSPIA_SUCCESS	/* Write operation successfully completed. */
RSPIA_ERR_CH_NOT_OPENED	/* The channel has not been opened. Perform R_RSPIA_Open() first. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */
RSPIA_ERR_LOCK	/* The lock could not be acquired. The channel is busy. */
RSPIA_ERR_INVALID_ARG	/* Argument is not valid for parameter. */

Properties

Prototyped in file “r_rspia_rx_if.h”

Description

Starts transmission of data to a SPI device. The function returns immediately after the transmit operation begins, and data will continue to be transmitted in the background under interrupt control until the requested length has been transmitted. When the transmission is complete the user-defined callback function is called. The callback function should be used to notify the user application that the transfer has completed. This function only perform transfer operations. During the RSPIA transmission, no data is received.

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;
result = R_RSPIA_Write(handle, my_command_word, source, length);
if (RSPIA_SUCCESS != result)
{
      return result;
}
while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
      // Do something useful while waiting for the transfer to complete.
      R_BSP_NOP();
}
```

Special Notes:

Add the following processing when specifying DMAC transfer or DTC transfer.

- For the callback function that occurs when communication ends, see 1.7, Basic Operations (In DMAC/DTC).
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.

R_RSPIA_WriteRead()

The Write Read function simultaneously transmits data to a SPI device while receiving data from a SPI device.

Format

```
rspia_err_t  R_RSPIA_WriteRead(rspia_hdl_t        hdl,
                               rspia_command_word_t  p_cmd,
                               void                *p_src,
                               void                *p_dst,
                               uint16_t             length);
```

Parameters

hdl

Handle for the channel

p_cmd

Bit field data consisting of all the RSPIA command register settings for SPCMD for this operation. See 2.10 Parameters used for the command settings word.

**p_src*

Void type pointer to a source data buffer from which data will be transmitted to a SPI device. Based on the data frame bit-length specified in the p_cmd.bit_length, the *p_src pointer will be type cast to the corresponding data type during the transfer. So, for example, if the bit-length is set to 16-bits, then the source buffer data will be accessed as a block of 16-bit data, and so on for each bit-length setting. Bit-length settings that are not 8, 16 or 32 will use the data type that they can be contained within. For example, 24-bit frames will be stored in 32-bit storage, 11-bit frames will be stored in 16-bit storage, etc.

**p_dst*

Void type pointer to a destination buffer into which data will be copied that has been received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL. Based on the data frame bit-length specified in the p_cmd.bit_length, the *p_dst pointer will be type cast to the corresponding data type during the transfer. So, for example, if the bit-length is set to 16-bits, then the data will be stored in the destination buffer as a 16-bit value, and so on for each bit-length setting. Bit-length settings that are not 8, 16 or 32 will use the smallest data type that they can be contained within. For example, 24-bit frames will be stored in 32-bit storage, 11-bit frames will be stored in 16-bit storage, etc.

length

Transfer length variable to indicate the number of data frames to be transferred. The size of the data word is determined from settings in the p_cmd.bit_length argument. Be sure that the length argument matches the storage type of the source data; this is a count of the number of frames, not the number of bytes. The same number of frames will be both written and read.

Return Values

RSPIA_SUCCESS	/* Read operation successfully completed. */
RSPIA_ERR_CH_NOT_OPENED	/* The channel has not been opened. Perform R_RSPIA_Open() first. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */
RSPIA_ERR_LOCK	/* The lock could not be acquired. The channel is busy. */
RSPIA_ERR_INVALID_ARG	/* Argument is not valid for parameter. */

Properties

Prototyped in file “r_rspia_rx_if.h”

Description

Starts full-duplex transmission and reception of data to and from a SPI device. The function returns immediately after the transfer operation begins, and data transfer will continue in the background under interrupt control until the requested length has been transferred. When the operation is complete the userdefinedcallback function is called. The callback function should be used to notify the user application that the transfer has completed.

Operation differs slightly depending on whether the RSPIA is operating as Master or Slave. If the RSPIA is configured as slave, then data will only transfer when clocks are received from the Master. Data to be

transmitted is obtained from the source buffer, while received data is stored in the destination buffer.

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;
result = R_RSPIA_WriteRead(handle, my_command_word, source, dest, length);
if (RSPIA_SUCCESS != result)
{
      return result;
}
while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
      // Do something useful while waiting for the transfer to complete.
      R_BSP_NOP();
}
```

Special Notes:

Add the following processing when specifying DMAC transfer or DTC transfer.

- For the callback function that occurs when communication ends, see 1.7, Basic Operations (In DMAC/DTC).
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.

R_RSPIA_Close()

Fully disables the RSPIA channel designated by the handle.

Format

```
rspia_err_t R_RSPIA_Close(rspia_hdl_t hdl);
```

Parameters

hdl

Handle for the channel

Return Values

RSPIA_SUCCESS	/* Successful: channel closed. */
RSPIA_ERR_CH_NOT_OPENED	/* The channel has not been opened so closing has no effect. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */

Properties

The declaration is located in r_rspia_rx_if.h.

Description

This disables the RSPIA channel designated by the handle. The RSPIA handle is modified to indicate that it is no longer in the 'open' state. The RSPIA channel cannot be used again until it has been reopened with the R_RSPIA_Open function. If this function is called for an RSPIA that is not in the open state, then an error code is returned.

Reentrant

This function is not reentrant.

Example

```
rspia_err_t result;
result = R_RSPIA_Close(handle);
if (RSPIA_SUCCESS != result)
{
      return result;
}
```

Special Notes:

None

R_RSPIA_GetVersion()

This function returns the driver version number at runtime.

Format

```
uint32_t    R_RSPIA_GetVersion(void);
```

Parameters

None

Return Values

Version number with major and minor version digits packed into a single 32-bit value.

Properties

The declaration is located in r_rspia_rx_if.h.

Description

The function returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

Example

```
/* Retrieve the version number and convert it to a string. */
uint32_t version, version_high, version_low;
char version_str[9];
version = R_RSPIA_GetVersion();
version_high = (version >> 16)&0xf;
version_low = version & 0xff;
sprintf(version_str, "RSPIAv%1.1hu.%2.2hu", version_high, version_low);
```

Special Notes:

None

R_RSPIA_IntSptilerClear()

This function is used to clear the ICU.IERm.IENj bit of the transmit buffer-empty interrupt (SPTI).

Format

```
rspia_err_t R_RSPIA_IntSptiIerClear (rspia_hdl_t hdl)
```

Parameters

hdl

RSPIA handle number.

Return Values

RSPIA_SUCCESS	/* Successful operation. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */

Properties

Prototype declarations are contained in r_rspia_rx_if.h.

Description

Use this function when disabling interrupt from within the callback function generated at DMAC transfer-end or an intentional cancellation of transmission.

Please call this function after calling R_RSPIA_DisableSpti().

Example

```
DMA_Handler_W()
{
      R_RSPIA_DisableSpti(handle);
      R_RSPIA_IntSptiIerClear(handle);
}
```

Special Notes

Do not use this function during transmission other than an intentional cancellation of transmission.

Doing so could disrupt the transfer.

R_RSPIA_IntSprilerClear()

This function is used to clear the ICU.IERm.IENj bit of the receive buffer-full interrupt (SPRI).

Format

```
rspia_err_t R_RSPIA_IntSpriIerClear(rspia_hdl_t hdl)
```

Parameters

hdl

RSPIA handle number.

Return Values

RSPIA_SUCCESS	/* Successful operation. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */

Properties

Prototype declarations are contained in r_rspia_rx_if.h.

Description

Use this function when disabling interrupts from within the callback function generated at DMAC transfer-end or an intentional cancellation of transmission.

Please call this function before calling R_RSPIA_DisableRSPI().

Example

```
DMA_Handler_R()
{
      R_RSPIA_IntSpriIerClear(handle);
      R_RSPIA_DisableRSPI(handle);
}
```

Special Notes

Do not use this function during transmission other than an intentional cancellation of transmission.

Doing so could disrupt the transfer.

R_RSPIA_DisableSpti()

This function disables the generation of transmit buffer empty interrupt request.

Format

```
rspia_err_t R_RSPIA_DisableSpti(rspia_hdl_t hdl)
```

Parameters

hdl

RSPIA handle number.

Return Values

RSPIA_SUCCESS /* Successful operation. */
RSPIA_ERR_NULL_PTR /* A required pointer argument is NULL. */

Properties

Prototype declarations are contained in r_rspia_rx_if.h.

Description

Use this function when disabling interrupts from within the callback function generated at DMAC transfer-end or an intentional cancellation of transmission.

Please call this function before calling R_RSPIA_IntSptilerClear().

Example

```
DMA_Handler_R()
{
      R_RSPIA_DisableSpti(handle);
      R_RSPIA_IntSpriIerClear(handle);
}
```

Special Notes

Do not use this function during transmission other than an intentional cancellation of transmission.

Doing so could disrupt the transfer.

R_RSPIA_DisableRSPI()

This function is set to disable the RSPI function.

Format

```
rspia_err_t R_RSPIA_DisableRSPI(rspia_hdl_t hdl)
```

Parameters

hdl

RSPIA handle number.

Return Values

RSPIA_SUCCESS	/* Successful operation. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */

Properties

Prototype declarations are contained in r_rspia_rx_if.h.

Description

Use this function when disabling RSPI function from within the callback function generated at DMAC transfer-end or an intentional cancellation of transmission.

Please call this function after calling R_RSPIA_IntSpriIerClear().

Example

```
DMA_Handler_R()
{
      R_RSPIA_IntSpriIerClear(handle);
      R_RSPIA_DisableRSPI(handle);
}
```

Special Notes

Do not use this function during transmission other than an intentional cancellation of transmission.

Doing so could disrupt the transfer.

R_RSPIA_GetBuffRegAddress()

This function is used to fetch the address of the RSPIA data register (SPDR).

Format

```
rspia_err_t R_RSPIA_GetBuffRegAddress(rspia_hdl_t hdl, uint32_t *p_spdr_addr)
```

Parameters

hdl

RSPIA handle number.

** p_spdr_addr*

The pointer for storing the address of SPDR. Set this to the address of the storage destination.

Return Values

RSPIA_SUCCESS:	/* Successful operation. */
RSPIA_ERR_INVALID_ARG:	/* Argument is not valid for parameter. */
RSPIA_ERR_NULL_PTR	/* A required pointer argument is NULL. */

Properties

Prototype declarations are contained in r_rspia_rx_if.h.

Description

Use this function when setting the DMAC/DTC transfer destination/transfer source address, etc.

Example

```
uint32_t          reg_buff;
rspia_err_t ret = RSPIA_SUCCESS;
rspia_handle_t    handle;

handle->channel = 0;
ret = R_RSPIA_GetBuffRegAddress(handle, &reg_buff);
```

Special Notes

None

4. Pin Setting

To use the RSPIA 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. Please perform the pin setting after calling the R_RSPIA_Open function.

When performing the pin setting in the e^2 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

Option Selected	Function to be Output
Channel 0	R_RSPIA_PinSet_RSPIA0()

Note that if the 3-wire interface mode is being used then a GPIO port must be configured to handle the Slave Select signal. GPIOs may be configured using the FIT GPIO module API, or through direct register settings.

Setting RSPCK polarity
The setting of the value of the rspia_command_word_t structure rspia_spcmd_cpol_t, which sets the polarity of the RSPCK pin is updated when R_RSPIA_Open() function is called. Also, the output of the RSPCK pin is finalized by executing the functions shown in Table 4.1.

5. Sample Program

This application note includes one or more sample program to demonstrate basic usage of the FIT RSPIA Module. The sample program is intended to provide a quick functional example of common API function calls in use.

The provided sample application simulates a full-duplex transfer (simultaneous transmit and receive) by routing the Master output data to the Master input data with a jumper wire. Data received is tested to confirm that it matches the data sent. The RSPIA module version number is retrieved and can be displayed on the Renesas Virtual Debug Console window if desired.

5.1 Adding the Sample program to a Workspace

Sample programs are found in the FITDemos folder of the distribution file for this application note. Sample programs are MCU and board specific. Locate the sample program that matches the Renesas development board you will be using.

5.2 Running the Sample program

1. Prepare the board by jumpering MOSIx to MISOx depending on the target board:
   - RSKRX671
     i. Connect expansion header JA3 pin 7 to JA3 pin 8, SW4 pin 3 turn off.
2. Build and download the sample application to the RSK board using the e2 studio debugger.
3. Select the Renesas Virtual Debug Console view in e2 studio to view print information.
4. Run the application in the debugger.
5. Observe the version number print in the debug console window.
6. “Success!” is displayed in the debug console window. If transfer fails “Failed.” is displayed in the debug console window.

6. Appendices

6.1 Confirmed Operation Environment

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

Table 6.1 Confirmed Operation Environment (Rev.1.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 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	Little endian
Revision of the module	Rev.1.00
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxx)

Table 6.2 Confirmed Operation Environment (Rev.1.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 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	Little endian
Revision of the module	Rev.1.10
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxx)

Table 6.3 Confirmed Operation Environment (Rev.1.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2022-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202202 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	Little endian
Revision of the module	Rev.1.20
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)

Table 6.4 Confirmed Operation Environment (Rev.1.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2022-10 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	Little endian
Revision of the module	Rev.1.30
Board used	Renesas Flexible Motor Control Kit for RX26T(product No.:RTK0EMXE70S00020BJ)

Table 6.5 Confirmed Operation Environment (Rev.1.40)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2023-04 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	Little endian
Revision of the module	Rev.1.40
Board used	-

Table 6.6 Confirmed Operation Environment (Rev.1.41)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2023-04 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	Little endian
Revision of the module	Rev.1.41
Board used	-

Table 6.7 Confirmed Operation Environment (Rev.1.50)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2023-04 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	Little endian
Revision of the module	Rev.1.50
Board used	Renesas Flexible Motor Control Kit for RX26T(product No.:RTK0EMXE70S00020BJ) Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxx)

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:

- Using CS+:

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

- Using e^2 studio:

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

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. 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_rspia_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(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_rspia_rx_config.h” may be wrong. Check the file “r_rspia_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.8 Configuration Overview for details.

7. Reference Documents

User's Manual: Hardware

Technical Update/Technical News

User's Manual: Development Tools
The latest version can be downloaded from the Renesas Electronics website.

Related Technical Update

Not applicable technical update for this module.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar 31, 2021	--	First edition issued
1.10	Sep 13, 2021	30 31	Updated contents of Section 5 Sample Program. Added RX671 in Section 5.2 Running the Sample program. 6.1 Confirmed Operation Environment: Added Table for Rev.1.10.
1.20	Jul 29, 2022	31 Program	6.1 Confirmed Operation Environment: Added Table for Rev.1.20. Updated demo projects
1.30	Aug 15, 2022	1, 9 11 31 Program	Added support for RX26T. Added code size corresponding to RX26T. 6.1 Confirmed Operation Environment: Added Table for Rev. 1.30. Added support for RX26T
1.40	Jun 30, 2023	15, 28 32 Program	Deleted the description of FIT configurator from "2.13 Adding the FIT Module to Your Project", "4. Pin Settings". 6.1 Confirmed Operation Environment: Added Table for Rev. 1.40. Added support for RX26T-256KB Deleted the description of FIT configurator.
1.41	Nov 13, 2023	16 33 Program	Added 2.14 “for”, “while” and “do while” statements. 6.1 Confirmed Operation Environment: Added Table for Rev. 1.41. Added WAIT_LOOP comments.
1.50	Dec 15, 2023	5, 8, 13, 19 6 21 22-23 25 27 29 32 33 34 35 36 42 Program	Added support RSPIA with DMAC/DTC. Added 1.3 Using the FIT RSPIA module. Added 1.3.1 Using FIT RSPIA module in C++ project. Added new API in Section 1.4 API Overview. Modified R_RSPIA_Open() section. Modified R_RSPIA_Control() section. Modified R_RSPIA_Read() section. Modified R_RSPIA_Write() section. Modified R_RSPIA_WriteRead() section. Added R_RSPIA_IntSptiIerClear() section. Added R_RSPIA_IntSpriIerClear () section. Added R_RSPIA_DisableSpti() section. Added R_RSPIA_DisableRSPI() section. Added R_RSPIA_GetBuffRegAddress() section. 6.1 Confirmed Operation Environment: Added Table for Rev. 1.50. Added support RSPIA with DMAC/DTC.

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

Contact information

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

Trademarks

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