RX Family

RSPI Module Using Firmware Integration Technology

Introduction

This document covers the RSPI Module Using Firmware Integration Technology (FIT) for the supported RX family MCUs. Details are provided that describe the RSPI 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 Serial Peripheral Interface (RSPI) for up to three channels. The RSPI performs synchronous serial communication with full duplex or only transmission. It has a function that performs high-speed serial communication with multiple processors and peripheral devices.

Target Device

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

- **RX110 Group**
- **RX111 Group**
- **RX113 Group**
- **RX130 Group**
- **RX140 Group**
- **RX231, RX230 Group**
- **RX23E-A Group**
- **RX23E-B Group**
- **RX23T Group**
- **RX23W Group**
- **RX24T Group**
- **RX24U Group**
- **RX260, RX261 Group**
- **RX26T Group**
- **RX64M Group**
- **RX65N, RX651 Group**
- **RX660 Group**
- **RX66T Group**
- **RX66N Group**
- **RX671 Group**
- **RX71M Group**
- **RX72T Group**
- **RX72M Group**
- **RX72N Group**

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 Operation Confirmation 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 RSPI peripheral for operation and for performing data transfers over the SPI bus.

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

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

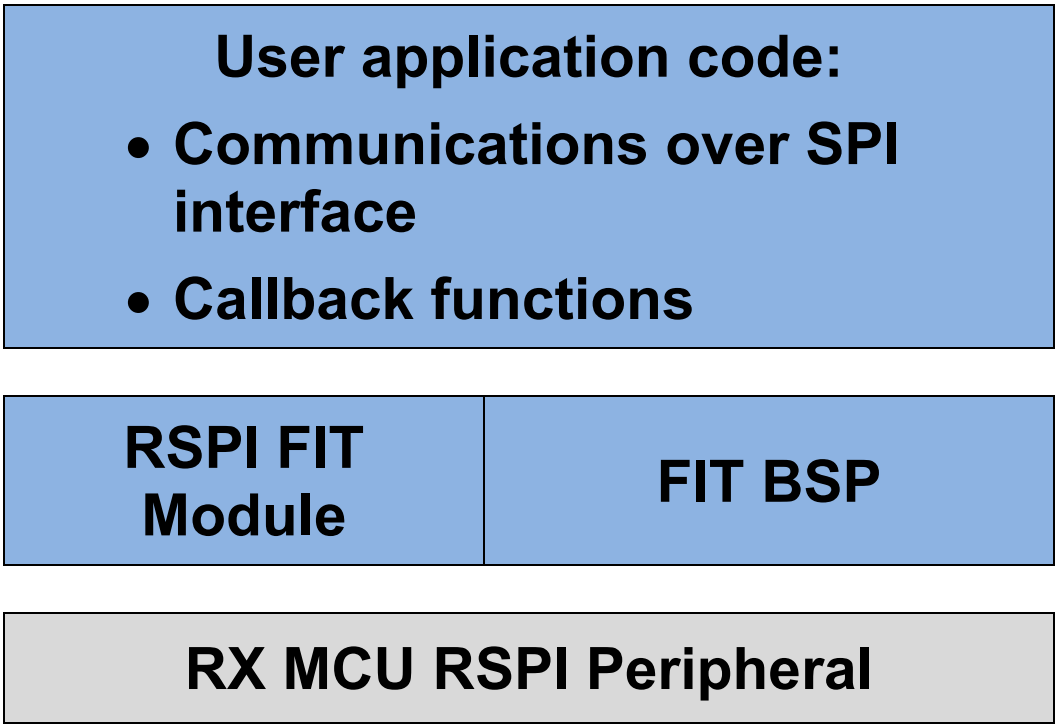

Figure 1-1 : Example Figure Showing Project Layers

1.1 RSPI FIT Module

The functions of the RSPI FIT module can be incorporated into software programs by means of APIs. For information on incorporating the RSPI FIT module into projects, see 2.12 Adding the FIT Module to Your Project.

1.2 Overview of RSPI FIT Module

After adding the RSPI FIT module to your project you will need to modify the *r_rspi_rx_config.h* file to configure the software for your installation. See Section 2.7 Compile Option for details on configuration options.

The RSPI 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 RSPI channel at run time, the first step is to call the R_RSPI_Open() function by passing the required settings and parameters. On completion, by setting up the I/O ports, the RSPI 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 RSPI slave, set SPE=0)
(2) Call R_RSPI_Open()
(3) Set the pins to peripheral module by I/O ports setting
(4) Enable the slave for communication

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

Three commands are provided in the R_RSPI_Control() function:

- Change the base bit-clock rate.
- Immediately abort a transfer operation.
- Rewrite RSPI register information.

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 RSPI API functions will require a 'handle' argument. This is used to identify the RSPI channel number that is selected for the operation. A handle is obtained by first calling the R_RSPI_Open() function. You must provide the address of a location where you will store the handle to R_RSPI_Open(), and on completion the handle will be available for use. Thereafter, simply pass the provided handle value for that RSPI 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 RSPI peripheral.

RSPI transfer functions:

- Use of MOSI (master out/slave in), MISO (master in/slave out), SSL (slave select), and RSPCK (RSPI clock) signals allows serial communications through SPI operation (four-wire method) or clock synchronous operation (three-wire method).
- 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 is selectable as 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, or 32 bits.

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

SSL control function:

- Four SSL signals (SSLn0 to SSLn3) for each channel

- In single-master mode: SSLn0 to SSLn3 signals are output.
- In slave mode: SSLn0 signal for input, selects the RSPI slave. SSLn1 to SSLn3 signals are unused.
- 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 output negation (SSL 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 + 2 PCLK units)
- Able to change SSL polarity

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

Interrupt sources:

- RSPI receive interrupt (receive buffer full)
- RSPI transmit interrupt (transmit buffer empty)
- RSPI error interrupt (mode fault, overrun, parity error, under run)
- Idle interrupt
- Communication end interrupt (RX671 only)

1.2.2 Features Not Supported

- To conserve limited RAM resources of smaller memory MCUs, like the RX111, 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 RSPI peripheral are not supported by this driver.
- Only single-frame data transfers are supported by this driver. The multi-frame features of the RSPI peripheral are not supported. This means that the maximum supported data frame size is 32 bits.
- Multi-master mode is not supported.
- Byte swap for 16-bit type is not supported.

1.3 Using the RSPI FIT module

1.3.1 Using RSPI FIT module in C++ project

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

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

1.4 Overview of APIs

The following functions are included in this design:

Table 1.1: List of RSPI API functions

Function	Description
R_RSPI_Open()	Initializes the associated registers required to prepare the specified RSPI channel for use, provides the handle for use with other API functions. Takes a callback function pointer for responding to interrupt events.
R_RSPI_Close()	Disables the specified RSPI channel.
R_RSPI_Control()	Handles special hardware or software operations for the RSPI channel.
R_RSPI_Write()*1	The Write function transmits data to a SPI master or slave device.
R_RSPI_Read()*1	The Read function receives data from a SPI master or slave device.
R_RSPI_WriteRead()*1	The Write Read function simultaneously transmits data to a SPI master or slave device while receiving data from that device (full duplex).
R_RSPI_GetVersion()	Returns the driver version number.
R_RSPI_GetBuffRegAddress()	SPDR register address acquisition processing
R_RSPI_IntSptiIerClear()	SPTI transmit interrupt request disable processing
R_RSPI_IntSpriIerClear()	SPRI receive interrupt request disable processing
R_RSPI_DisableSpti()	Disables the generation of transmit buffer empty interrupt requests
R_RSPI_DisableRSPI()	Disables the RSPI function
R_RSPI_SetLogHdlAddress()	LONGQ FIT module handler address setting processing

Note 1: To speed up RSPI control, 32-bit access is used for the SPDR register. Align the start address with a 4-byte boundary when specifying transmit and receive data storage buffer pointers.

1.5 Driver Architecture

1.5.1 System Examples

The driver supports single-master/multi-slave mode operation, or slave-mode operation. Each RSPI 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.

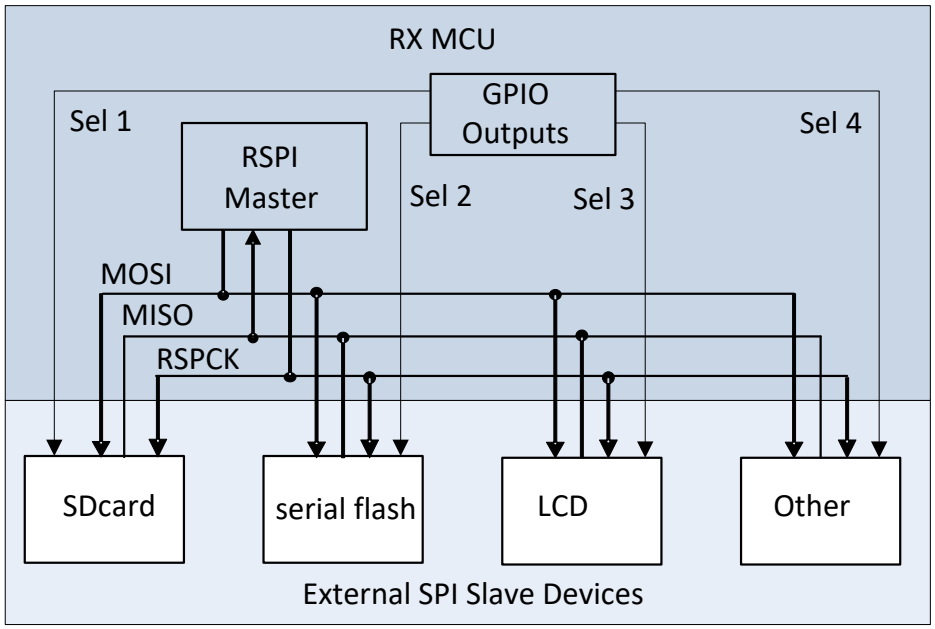

Figure 1-2 : This example shows the use of GPIO ports to serve as the slave select signals (3-Wire mode)

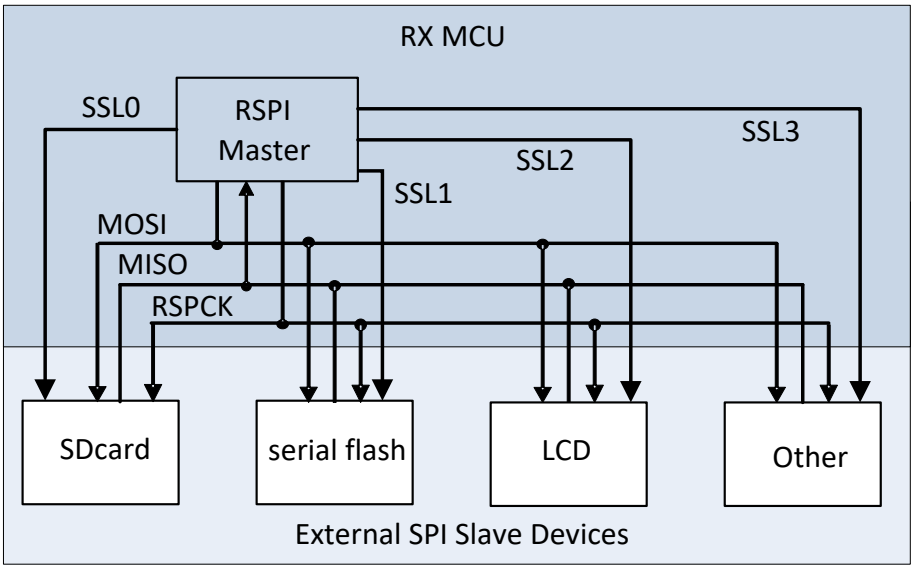

Figure 1-3 : The built-in RSPI peripheral slave select hardware (SSL) may be used to generate the signals (SPI 4-Wire mode)

1.5.2 Multi-Channel RSPI Support

For supported RX family MCUs that have multiple channels of RSPI available, this driver will operate all available channels on an individually selectable basis with the same body of code. Each channel can be configured with its own setup independently of the other channels in use.

1.6 Basic Operation (SW Transfer)

The RSPI 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 RSPI channel will be locked for the duration of the operation. After that, the remainder of the transfer operation is performed within RSPI interrupt handler routines.

1.6.1 Master Transmission

In master mode, data is written by the RSPI Master on the MOSI (master out, slave in) line. In slave mode, data is written by the RSPI Slave on the MISO (master in, slave out) line. The RSPI FIT module can perform continuous transmission since it writes the next transmission data to the SPDR register during data transmission. Data to be transmitted is read from a buffer location pointed to by the user application, and it is copied to the RSPI transmit data register after being type-casted for the data type specified by the current operation, and then written to the SPDR register.

1.6.2 Master Reception / Master Transmission and Reception

Data is received by the RSPI Master on the MISO (master in, slave out) line. With the RSPI peripheral configured as SPI bus Master, it is set for full-duplex operation in order to receive data from a slave device on the SPI bus. This requires the RSPI Master to output clocks to the Slave. Clocks are output only when the RSPI Master is sending data. Therefore, in order to read data from the SPI bus, the master must also simultaneously write data. This can either be actual data that needs to be transmitted (if the slave is capable of full-duplex communication), or it can be dummy data ignored by the slave. In this RSPI FIT module, data is received by clock by sending dummy data for which the user can set data pattern.

Master reception/Master transmission of RSPI FIT module also includes a normal mode and high speed mode.

- Normal mode

In normal mode, after the first data is received, unless that data is read from the SPDR register, the next data reception will not start. If reading from the SPDR register is performed by CPU processing, communication will be stopped during the period from reading to the start of next data reception, so a gap will be generated between communication frames. To receive data without a gap between communication frames, use high speed reception mode or receive with a combination of DMAC/DTC.

- High speed mode

In high speed mode, after the first data is received, the next data reception starts immediately. Therefore, there is no gap between communication frames. (Note1)This control requires reading the first data from the SPDR register before reception of the next data is completed. In the RSPI FIT module, this is achieved using the RSPCK auto-stop function (Note2) installed in the RSPI.

Note1: Depending on the mcu's system clock, peripheral clock, and RSPI communication speed, gaps may occur between the communication frames.

Note2: Since this function is not installed in some RX family MCU RSPI, an overrun error may therefore occur because data reading cannot be completed in time. When using an MCU that does not have this function, we recommend disabling other interrupts during RSPI communication or using normal receive mode.

1.6.3 Slave Transmission

The slave mode write operation is nearly the same as the master mode except that, after setting up for transmission, the slave waits for clocks from a master SPI device. Additionally, in the slave mode, double buffers are used for slave’s data transmission so that back-to-back frames clocked by the master will not starve the slave’s transmit shift register.

If not reading while writing, every time a frame is transmitted the SPDR register is read and the data is discarded. The transmit operation will terminate when the requested number of frames has been transmitted or until aborted by a user command.

1.6.4 Slave Reception / Slave Transmission and Reception

The slave mode read operation is exactly the same as the master mode except that, after setting up for reception, the slave waits for clocks from a master SPI device. If not also transmitting valid data while receiving, the shift register is filled with a dummy data pattern. The read operation will terminate when the requested number of frames has been received or until aborted by a user command.

1.7 Basic Operations (In DMAC/DTC)

The RSPI 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 RSPI_TRANS_MODE_DMAC or RSPI_TRANS_MODE_DTC to the second argument pconfig -> tran_mode of the R_RSPI_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 RSPI FIT module is not called. In case of a communication error, an RSPI error interrupt occurs and the callback function of the RSPI FIT module is called.

- Termination of DTC communication

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

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

Note 2 For the setting method, see the sample program included in the DMAC/DTC FIT application note or the RSPI FIT application note.

1.8 Interrupts

1.8.1 Data transfer interrupts

The RSPI driver transmit and receive operations are performed in a non-blocking fashion. When using SW transfer, communication operations are carried out with interrupt service routines on an event-driven basis. The RSPI Transmit Buffer Empty (SPTI) is used to call a write function that performs a single frame transmit procedure, and Receive Buffer Full (SPRI) interrupts is used to call a read function that performs a single frame receive procedure.

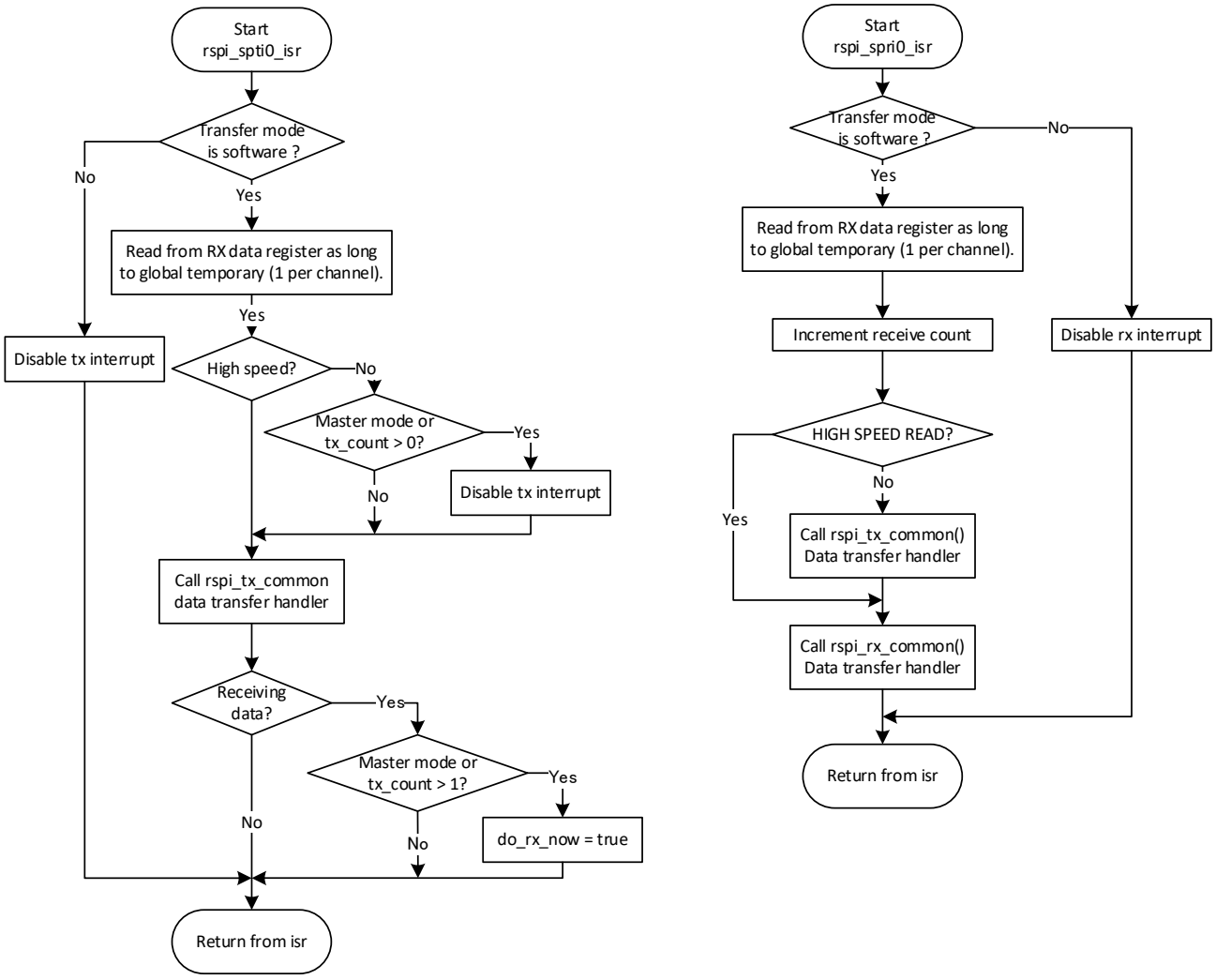

Figure 1-4. Common interrupt handler algorithm.

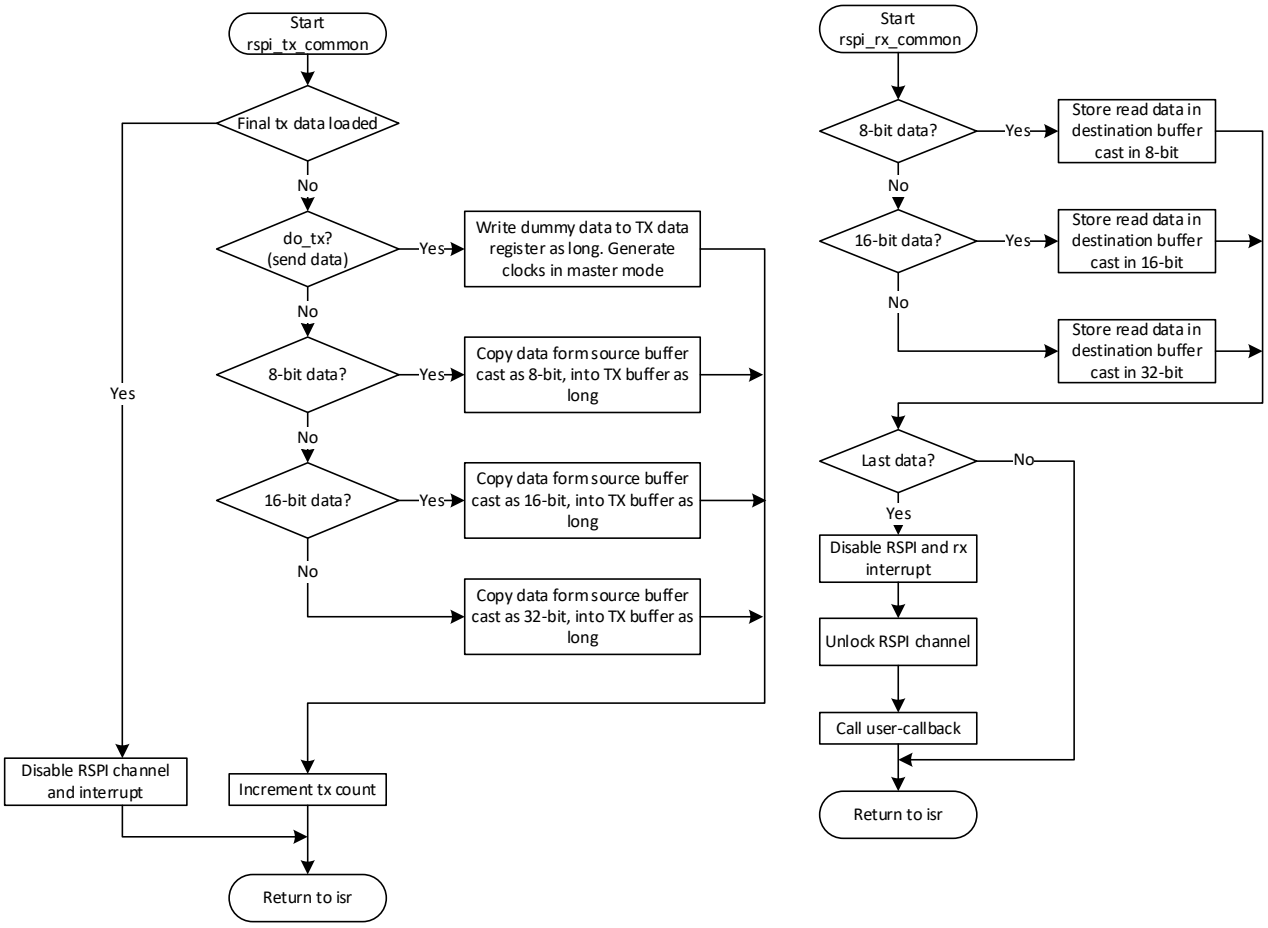

Figure 1-5 common data transmit and receive handle

1.8.2 Error interrupts

The RSPI Error interrupts (SPEI) are used to call a common handler function that reads the status register to determine the interrupt cause. Further processing of the data transfer operation is halted and the callback function is called.
In the error interrupt handler processing, check each flag state for SPSR register in the order of OVRF→MODF→UDRF→PERF.
Set the first-detected error flag state to argument event of callback function.

1.9 Relations of Data Output and RAM

Note that the data is not output in the order of the data stored in RAM when the data is 16-bit type or 32-bit type, and is Little endian. Perform byte swap processing as necessary. IP version RSPIc or later includes byte swap function.

1.9.1 Transmitting Data

1.9.1.1 16-bit type[Little endian]

As shown in Figure 1-6, when 1 frame data is16-bit type, the data is inverted at the timing of writing RAM data to SPDR register. Therefore, the order of data output is as Byte1, Byte0, Byte3, Byte2…

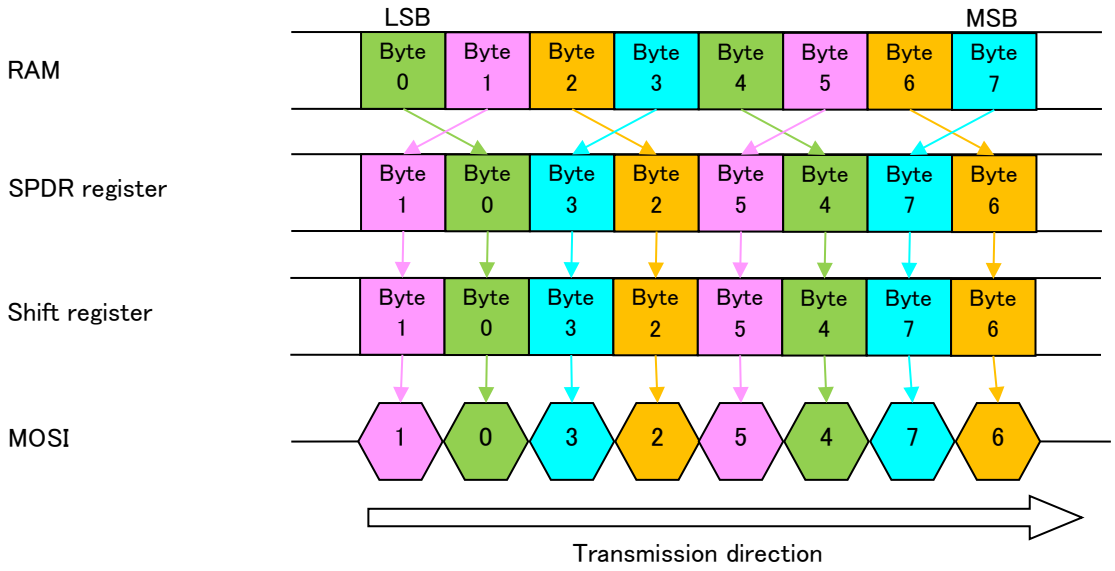

Figure 1-6 : Transmitting Data 16-bit type[Little endian] No byte swapping

IP version RSPIc or later includes byte swap function, and byte swap is available on the hardware. However, RSPI driver does not support 16-bit hardware byte swap.

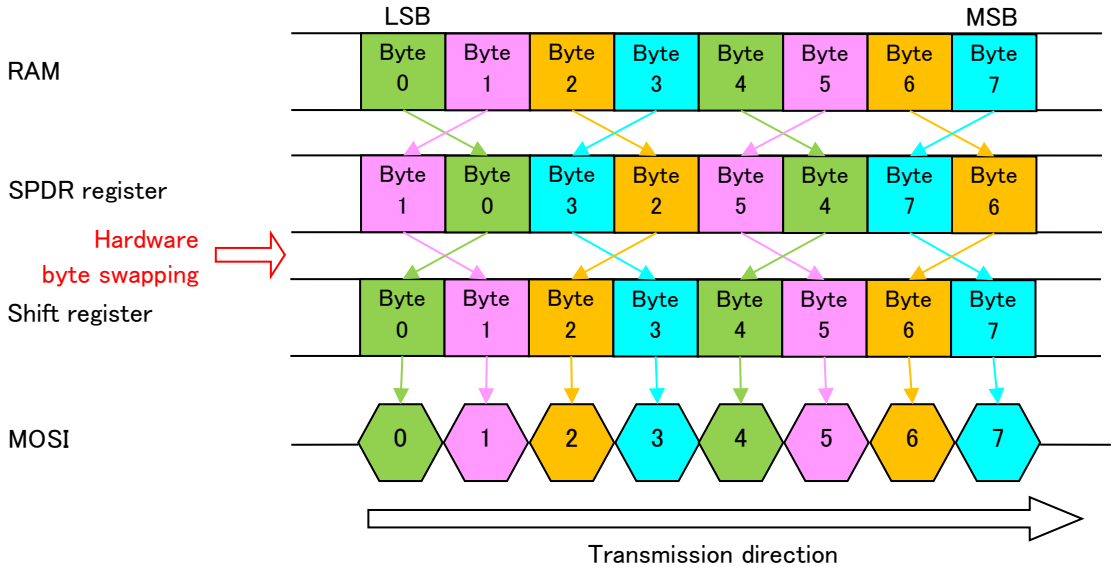

Figure 1-7 : Transmitting Data 16-bit type[Little endian] hardware byte swapping

1.9.1.2 32-bit type[Little endian]

As shown in Figure 1-8, when 1 frame data is 32-bit type, the data is inverted at the timing of writing RAM data to SPDR register. Therefore, the order of data output is as Byte3, Byte2, Byte1, Byte0….

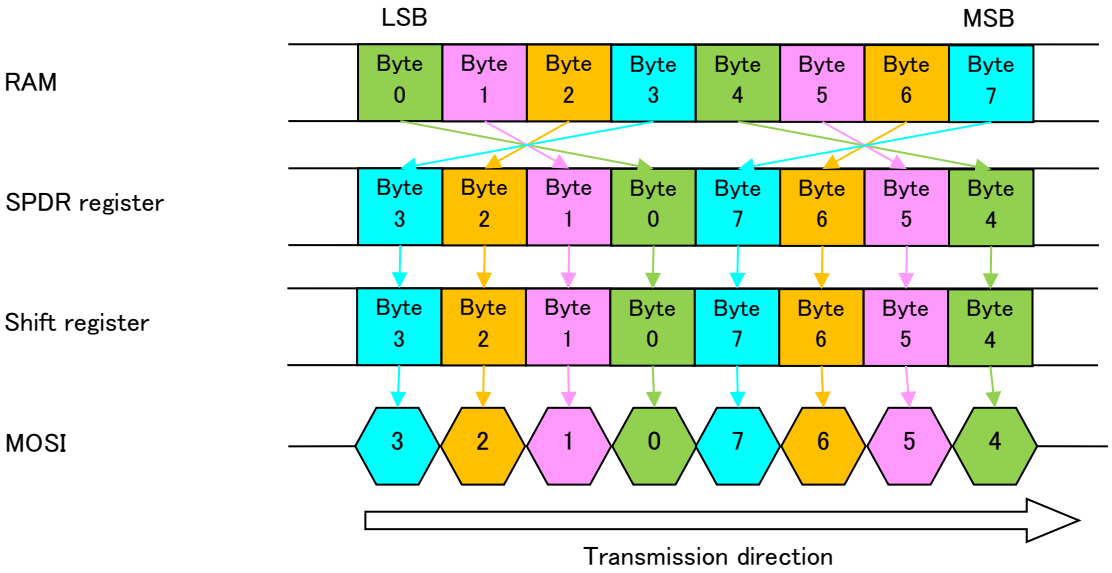

Figure 1-8 : Transmitting Data 32-bit type[Little endian]No byte swapping

IP version RSPIc or later includes byte swap function, and byte swap is available on the hardware. RSPI driver supports 32-bit hardware byte swap.

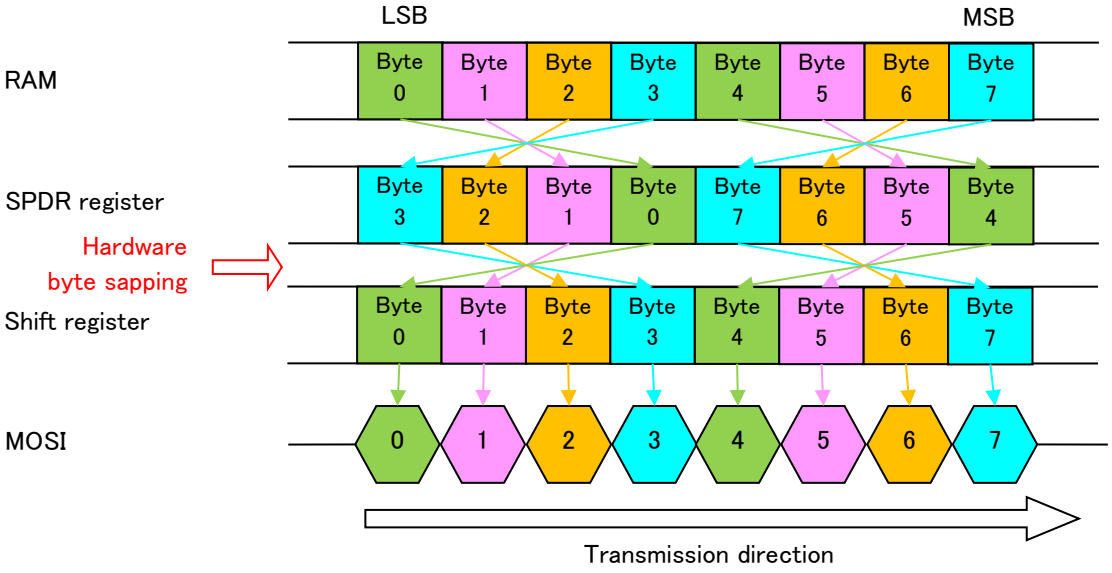

Figure 1-9 : Transmitting Data 32-bit type[Little endian]hardware byte swapping

1.9.1.3 Other data type and Endian

For the data type and endian shown below, the data is output in the order of the data stored in RAM.

- 8-bit type[Little endian/ Big endian]
- 16-bit type[Big endian]
- 32-bit type[Big endian]

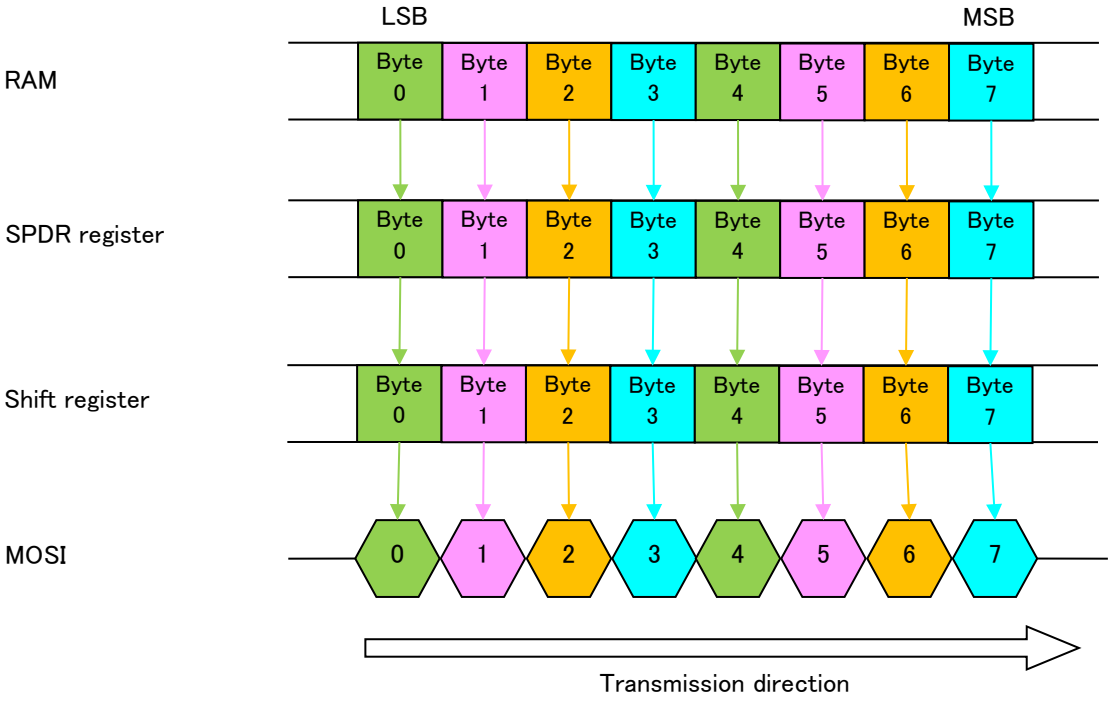

Figure 1-10 : Transmitting Data Other data type and Endian

1.9.2 Receiving Data

1.9.2.1 16-bit type[Little Endian]

As shown in Figure 1-11, when 1 frame data is 16-bit type, the data is inverted at the timing of reading the data from SPDR register to RAM. Therefore, the order of data stored in RAM is as Byte1, Byte0, Byte3, Byte2….

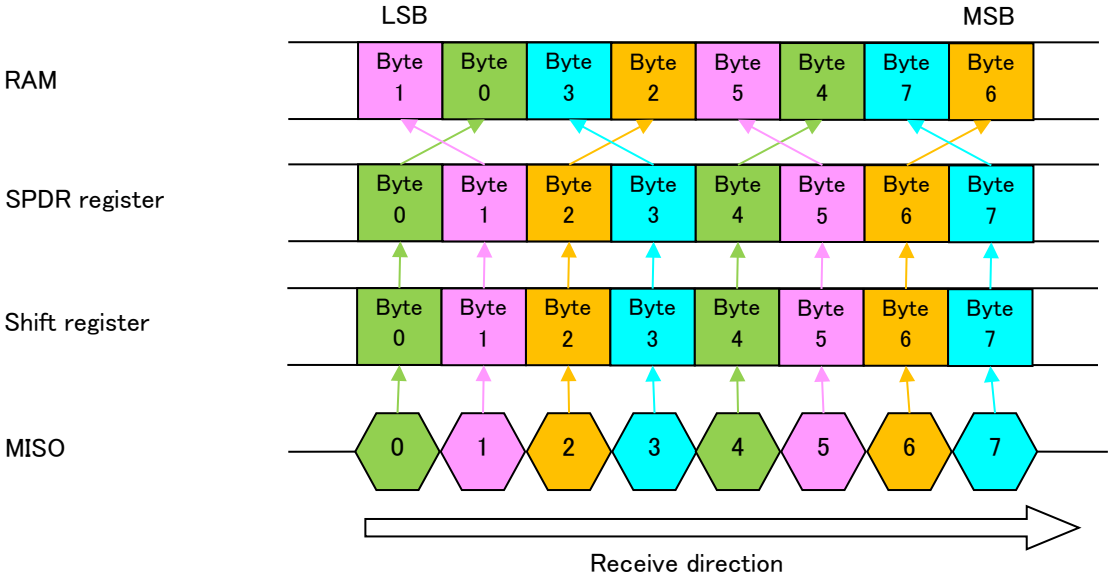

Figure 1-11 : Receiving Data 16-bit type[Little endian]No byte swapping

IP version RSPIc or later includes byte swap function, and byte swap is available on the hardware.

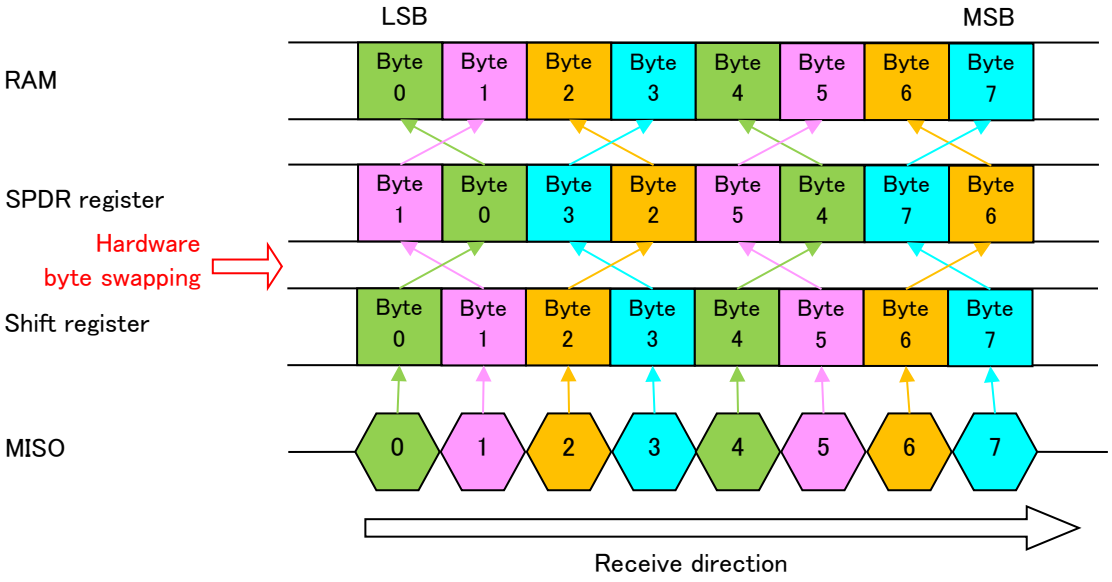

Figure 1-12 : Receiving Data 16-bit type[Little endian]hardware byte swapping

1.9.2.2 32-bit type[Little endian]

As shown in Figure 1-13, when 1 frame data is 32-bit type, the data is inverted at the timing of reading the data from SPDR register to RAM. Therefore, the order of the data stored in RAM is as Byte3, Byte2, Byte1, Byte0….

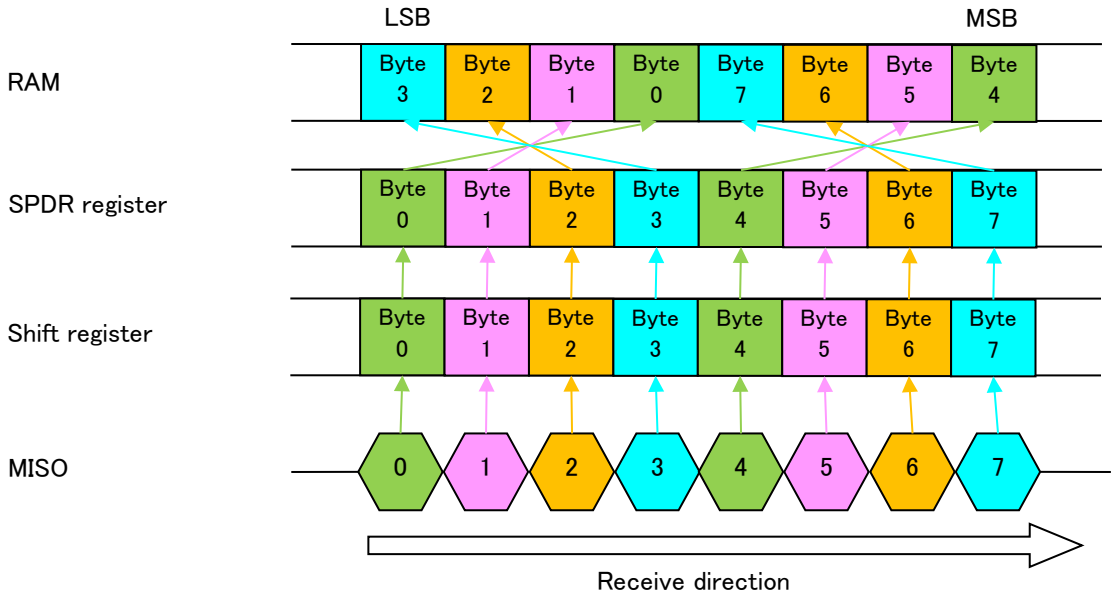

Figure 1-13 : Receiving Data 32-bit type[Little endian]No byte swapping

IP version RSPIc or later includes byte swap function, and byte swap is available on the hardware.

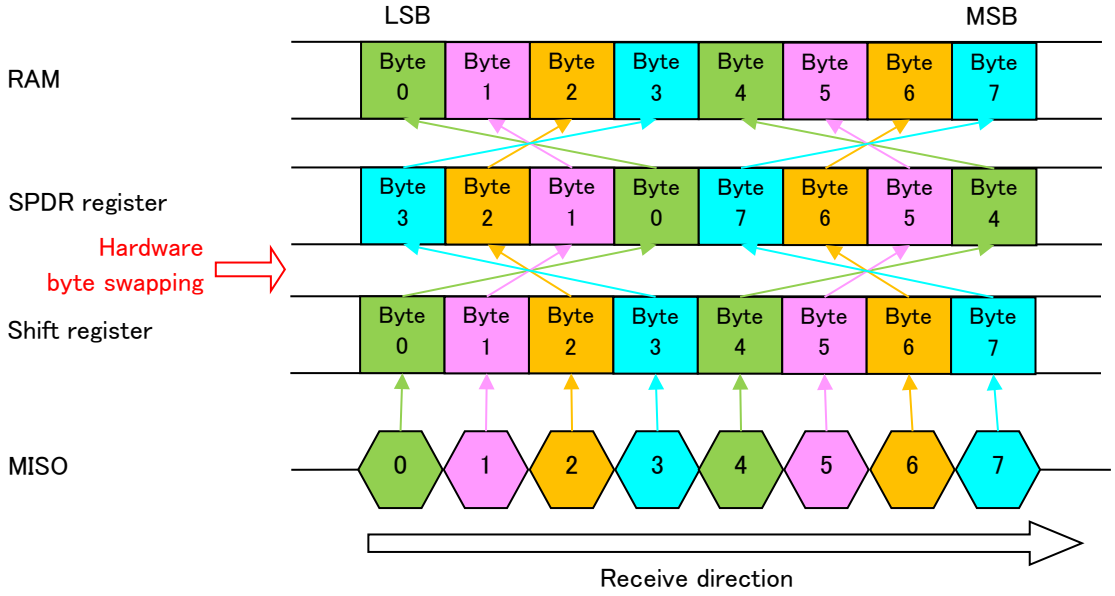

Figure 1-14 : Receiving Data 32-bit type[Little endian]Hardware byte swapping

1.9.2.3 Other data type and Endian

For the data type and endian shown below, the data is stored in the RAM in the order of data output.

- 8-bit type[Little endian/ Big endian]
- 16-bit type[Big endian]
- 32-bit type[Big endian]

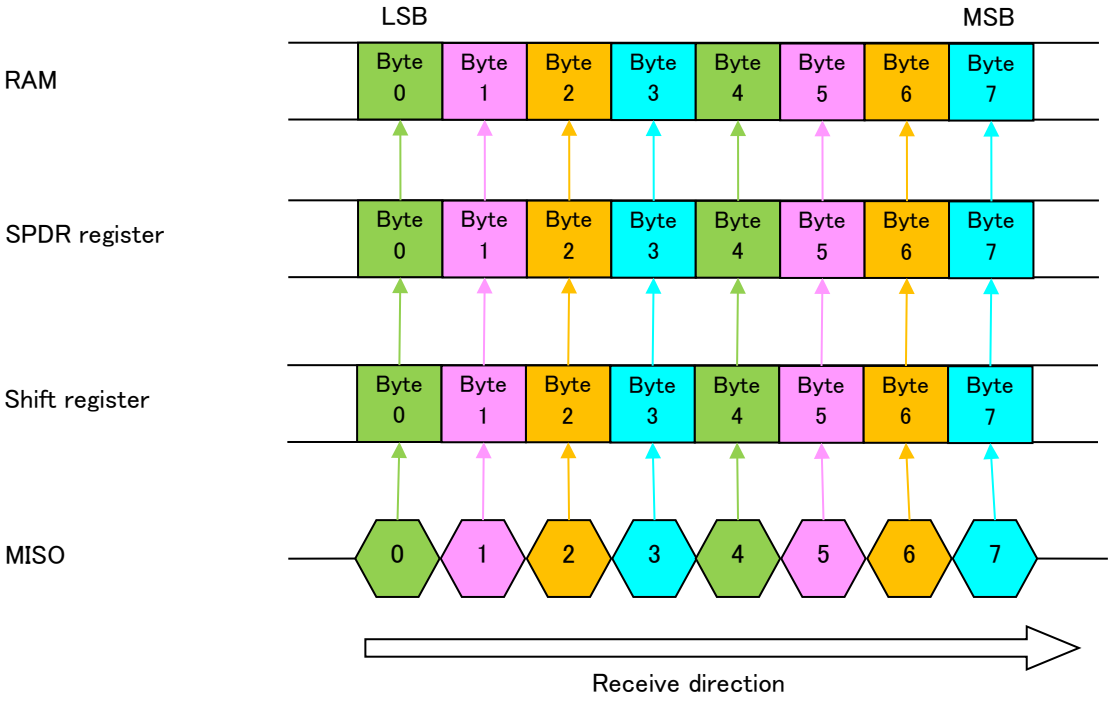

Figure 1-15 : Receiving Data Other Data type and Endian

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 RSPI 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.5.20 or higher. The related I/O ports should be correctly initialized elsewhere after calling the R_RSPI_Open() of this software.
- This software requires that the peripheral clock (PCLKB) 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 Supported Toolchains

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

2.4 Interrupt vector

When running the R_RSPI_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
RX110 RX111 RX113 RX130 RX140 RX230 RX231 RX23E-A RX23E-B RX23T RX23W RX24T RX24U RX260 RX261	SPRI0 interrupt[channel0](vector no.: 45) SPTI0 interrupt[channel0](vector no.: 46) SPII0 interrupt[channel0](vector no.: 47)
RX64M RX660 RX66T RX72T	SPRI0 interrupt[channel0](vector no.: 38) SPTI0 interrupt[channel0](vector no.: 39) SPII0 interrupt[channel0](vector no.: 112)
RX65N RX651 RX66N RX72M RX72N	SPRI0 interrupt[channel0](vector no.: 38) SPTI0 interrupt[channel0](vector no.: 39) SPII0 interrupt[channel0](vector no.: 112) SPRI1 interrupt[channel1](vector no.: 40) SPTI1 interrupt[channel1](vector no.: 41) SPII1 interrupt[channel1](vector no.: 112) SPRI2 interrupt[channel2](vector no.: 108) SPTI2 interrupt[channel2](vector no.: 109) SPII2 interrupt[channel2](vector no.: 112)
RX671	SPRI0 interrupt[channel0](vector no.: 38) SPTI0 interrupt[channel0](vector no.: 39) SPII0 interrupt[channel0](vector no.: 112) SPCI0 interrupt[channel0](vector no.: 252) SPRI1 interrupt[channel1](vector no.: 40) SPTI1 interrupt[channel1](vector no.: 41) SPII1 interrupt[channel1](vector no.: 112) SPCI1 interrupt[channel1](vector no.: 253) SPRI2 interrupt[channel2](vector no.: 108) SPTI2 interrupt[channel2](vector no.: 109) SPII2 interrupt[channel2](vector no.: 112) SPCI2 interrupt[channel2](vector no.: 254)

Device	Interrupt Vector
RX71M	SPRI0 interrupt[channel0](vector no.: 38) SPTI0 interrupt[channel0](vector no.: 39) SPII0 interrupt[channel0](vector no.: 112) SPRI1 interrupt[channel1](vector no.: 40) SPTI1 interrupt[channel1](vector no.: 41) SPII1 interrupt[channel1](vector no.: 112)
RX26T	SPRI0 interrupt[channel0](vector no.: 38) SPTI0 interrupt[channel0](vector no.: 39) SPII0 interrupt[channel0](vector no.: 112) SPCI0 interrupt[channel0](vector no.: 252)

2.5 Header Files

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

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

2.6 Integer Types

If your toolchain supports C99 then *stdint.h* should be described as shown below. If not, then there should be *typedefs.h* file that is included with your project as defined by the Renesas Coding Standards document.

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in *stdint.h*.

2.7 Compile Option

Some features or behavior of the software are determined at build-time by configuration options that the user must select.

Table 2.2 : List of RSPI driver module configuration options

Configuration options in *r_rspi_rx_config.h*	
RSPI_CFG_PARAM_CHECKING_ENABLE	Checking of arguments passed to RSPI API functions can be enabled or disabled. Disabling argument checking is provided for systems that absolutely require faster and smaller code. By default the module is configured to use the setting of the system-wide BSP_CFG_PARAM_CHECKING_ENABLE macro. This can be locally overridden for the RSPI module by redefining RSPI_CFG_PARAM_CHECKING_ENABLE. To control parameter checking locally, set RSPI_CFG_PARAM_CHECKING_ENABLE to 1 to enable it, otherwise set to 0 skip checking.
RSPI_CFG_REQUIRE_LOCK	If this is set to (1) then the RSPI driver will attempt to obtain the lock for the channel when performing certain operations to prevent concurrent access conflicts.
RSPI_CFG_DUMMY_TXDATA	The user-specified Dummy Data to be transmitted during receive-only operations.
RSPI_CFG_USE_CHANn	Enable the RSPI channels to use at build-time. (0) = not used. (1) = used.
RSPI_CFG_IR_PRIORITY_CHANn	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_RSPI_Open has been made to a channel. However, the next call to R_RSPI_Open for that channel will change it back to this configuration value.
RSPI_CFG_USE_RX63_ERROR_INTERRUPT	For RX63 group MCUs the RSPI error interrupt is a group interrupt shared with the SCI peripheral. So the error interrupt is disabled by default for RX63 group to prevent conflict with SCI FIT module. However, if not using the SCI FIT module, this may be enabled by setting RSPI_CFG_USE_RX63_ERROR_INTERRUPT to (1).
RSPI_CFG_MASK_UNUSED_BITS	When reading the RSPI received data register for data frame bit lengths that are not 8, 16, or 32-bits the unused upper bits will contain residual values from the transmit data. As a convenience, these unused upper bits may be optionally masked off (cleared to 0) by the driver when the data is transferred to the user-data buffer. Since this takes additional processing time in the data transfer interrupt handler it will cause a reduction in performance for the highest bit rate settings. This is not needed for 8, 16, or 32-bit transfers. So only enable this option when using data frame bit lengths that are not 8, 16, or 32-bits. (0) = do not clear. (1) = clear unused upper bits.

Configuration options in *r_rspi_rx_config.h*	
RSPI_CFG_HIGH_SPEED_READ	Selects master transmission/master reception and transmission mode. When this option is set to “disabled”, reception/transmission and reception operate in normal mode. When this option is set to “enabled”, reception/transmission and reception operate in high speed mode.
RSPI_CFG_LONGQ_ENABLE	Selects whether or not debug error log acquisition processing is used. When this option is set to “disabled”, code for the relevant processing is omitted. When this option is set to “enabled”, code for the relevant processing is included. To use this functionality, the LONGQ FIT module is also required.

2.8 Code Size

Typical code sizes associated with this module are listed below. Information is listed for a single representative device of the RX100 Series, RX200 Series, RX600 Series and RX700 Series respectively.

The code size is based on optimization level 2 for the RXC toolchain. The ROM (code and constants) and permanently allocated RAM sizes vary based on the build-time configuration options set in the module configuration header file.

Sizes listed here are given for a minimum build configuration and a maximum build configuration. The minimum build includes one RSPI channel configured for use and all other optional features disabled. The maximum build sizes include all available RSPI channels for the given MCU configured to be used, and the parameter checking and access locking options turned on. Stack usage is not listed, and should be determined by the user.

The values in the table below are confirmed under the following conditions.

Module Revision: r_rspi_rx rev3.50

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00

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

GCC for Renesas RX 8.03.00.202405

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

IAR C/C++ Compiler for Renesas RX version 5.10.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

Table 2.3 Code Size

ROM, RAM and Stack Code Sizes (Note1, 2, 3, 4)							
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
RX111	ROM	2,351 bytes	2,214 bytes	3,040 bytes	2,840 bytes	4,112 bytes	3,951 bytes
	RAM	63 bytes		128 bytes		55 bytes	
	Max. user stack	48 bytes		-		92 bytes	
	Max. interrupt stack	44 bytes		-		52 bytes	
RX261	ROM	2,410 bytes	2,271 bytes	3,112 bytes	2,920 bytes	4,239 bytes	4,085 bytes
	RAM	63 bytes		64 bytes		55 bytes	
	Max. user stack	48 bytes		-		92 bytes	
	Max. interrupt stack	44 bytes		-		52 bytes	

ROM, RAM and Stack Code Sizes (Note1, 2, 3, 4)							
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
RX65N	ROM	2,538 bytes	2,399 bytes	3,416 bytes	3,224 bytes	4,447 bytes	4,288 bytes
	RAM	63 bytes		128 bytes		55 bytes	
	Max. user stack	48 bytes		-		100 bytes	
	Max. interrupt stack	44 bytes		-		52 bytes	
RX71M	ROM	2,502 bytes	2,364 bytes	3,376 bytes	3,176 bytes	4,376 bytes	4,209 bytes
	RAM	63 bytes		0 bytes		55 bytes	
	Max. user stack	48 bytes		-		100 bytes	
	Max. interrupt stack	44 bytes		-		52 bytes	

Note 1 The memory sizes listed apply when the default settings listed in, “2.7 Compile Option”, are used. The memory sizes differ according to the definitions selected.

Note 2 Under confirmation conditions listed the following

- r_dmaca_rx.c
- r_dmaca_rx_target.c

Note 3 The required memory sizes differ according to the C compiler version and the compile conditions.

Note 4 The memory sizes listed apply when the little endian. The above memory sizes also differ according to endian mode.

2.9 Argument

It shows the structure for the argument of the API functions. This structure is listed in r_rspi_rx_if.h along with the prototype declarations of the API functions.

See “2.13 API Data Structures” for the details.

2.10 Return values

The different values API functions can return.

Return Type: rspi_err_t

Table 2.4 Return values

Values:	Cause
RSPI_SUCCESS	Function completed without errors
RSPI_ERR_BAD_CHAN	Invalid channel number
RSPI_ERR_CH_NOT_OPENED	Channel not yet opened. Function cannot be completed.
RSPI_ERR_CH_NOT_CLOSED	Channel still open from previous open.
RSPI_ERR_UNKNOWN_CMD	Control command is not recognized.
RSPI_ERR_INVALID_ARG	Argument is not valid for parameter.
RSPI_ERR_ARG_RANGE	Argument is out of range for parameter.
RSPI_ERR_NULL_PTR	Received null pointer; missing required argument.
RSPI_ERR_LOCK	A lock procedure failed
RSPI_ERR_UNDEF	Undefined/unknown error

2.11 Callback Functions

The definition of callbacks follows the FIT 1.0 specification rules:

a. Callback functions take one argument. This argument is ‘void * pdata’.

b. Before calling a callback function the function pointer is checked to be valid. At a minimum the pointer is checked to be:

i. Non-null

ii. Not equal to FIT_NO_FUNC macro.

2.11.1 Example callback function prototype declaration.

```
void callback(void * pdata)
```

2.11.2 Invocation of Callback functions

At the conclusion of every transfer operation the user defined callback will be called. This will occur within the context of the interrupt handler that processed the transfer operation. Any error condition that generates an interrupt, most typically the receive-overrun error, will also call the callback. A pointer to a data structure containing the channel number and result code of the interrupt that calls the callback are be passed as the only argument. It is up to the user application to process the provided information appropriately. Since callbacks are being processed within the context of the interrupt, and interrupts are disabled at this time, it is strongly recommended that the user-defined callback function complete as quickly as possible to avoid missing further system interrupts.

The most typical use of the callback function is to inform the application that the data transfer has completed. This may be done by setting a “busy” flag just before starting the transfer, and then clearing the busy flag within the callback. When used in RTOS environments, then a semaphore or other flag or message service provided by the OS may be used within the callback.

Example transfer start:

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;
rspi_result = R_RSPI_WriteRead(handle, my_command_word, source, dest, length);
if (RSPI_SUCCESS != rspi_result)
{
    return error;
}

while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
    // Do something useful while waiting for the transfer to complete.
    R_BSP_NOP();
}
```

Example callback function:

```
void my_callback(void * pdata)
{
    /* Examine the event to check for abnormal termination of transfer. */
    g_test_callback_event = (*(rspi_callback_data_t *)pdata).event_code;

    g_transfer_complete = true;
}
```

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 API Data Structures

This section details the data structures that are used with the driver's API functions.

2.13.1 Special Data Types

To provide strong type checking and reduce errors, many parameters used in API functions require arguments to be passed using the provided type definitions. Allowable values are defined in the public interface file *r_rspi_rx_if.h*.
The following special types have been defined:

Enumeration of SPI bus interface modes

Type: rspi_interface_mode_t

Values: RSPI_IF_MODE_3WIRE // Use GPIO for slave select.
RSPI_IF_MODE_4WIRE // Use slave select signals controlled by RSPI.

Enumeration of master or slave operating mode configuration settings

Type: rspi_master_slave_mode_t

Values: RSPI_MS_MODE_MASTER // Channel operates as SPI master.
RSPI_MS_MODE_SLAVE // Channel operates as SPI slave.

RSPI control command codes

Type: rspi_cmd_t

Values: RSPI_CMD_SET_BAUD // Use to set the base SPI clock bit-rate.
RSPI_CMD_ABORT // Stop the current read or write operation immediately.
RSPI_CMD_SETREGS // Set additional RSPI regs in one operation. (Expert use only)
RSPI_CMD_SET_TRANS_MODE // Set the data transfer mode.
RSPI_CMD_UNKNOWN, // Not a valid command.

RSPI control command data structures

See R_RSPI_Control() chapter.

Handle

Type: rspi_handle_t

Values: User allocates storage for this type for a handle. The address of this location must be passed in the call to the R_RSPI_Open() function. The handle value is automatically assigned by R_RSPI_Open() function and returned in the location specified.

Channel Settings structure for Open

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

Type: rspi_chnl_settings_t

Members:
```
rspi_interface_mode_t   gpio_ssl;          // Specify the interface mode.
rspi_master_slave_mode_t master_slave_mode; // Specify master or slave
                                            mode operation.
uint32_t    bps_target;     // The target bits per second setting for the channel.
rspi_str_tranmode_t   tran_mode;     // Data transfer mode.
```

Callback function data structure

The channel number and the procedure result code are passed in this data structure to the user defined callback function. For the event code, see “2.13.2 Event Codes”

Type: rspi_callback_data_t

Members: rspi_handle_t handle; // The channel handle.

rspi_evt_t event_code; // The event code.

Enumeration of SPI data transfer modes

Type: rspi_str_tranmode_t

Values: RSPI_TRANS_MODE_SW // Data transfer mode is software.

RSPI_TRANS_MODE_DMAC // Data transfer mode is DMAC.

RSPI_TRANS_MODE_DTC // Data transfer mode is DTC.

Enumeration of DMAC/DTC transfer flag

Type: rspi_trans_flg_t

Values: RSPI_SET_TRANS_STOP // Data transfer start flag.

RSPI_SET_TRANS_START // Data transfer end flag.

2.13.2 Event Codes

The different codes returned by API events.

Return Type: rspi_evt_t

Table 2.5 Event Codes

Values:	Cause
RSPI_EVT_TRANSFER_COMPLETE	The data transfer completed.
RSPI_EVT_TRANSFER_ABORTED	The data transfer was aborted.
RSPI_EVT_ERR_MODE_FAULT	Mode fault error.
RSPI_EVT_ERR_READ_OVF	Read overflow.
RSPI_EVT_ERR_PARITY	Parity error.
RSPI_EVT_ERR_UNDER_RUN	Underrun error
RSPI_EVT_ERR_UNDEF	Undefined/unknown error event.

2.14 Typedef enumerations used for the command settings word

This 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. Note that the valid data is lower 16 bits. The lower 16-bit data will get copied to the SPCMD register for each call to one of the read or write functions. To build a complete command lower 16-bit data select one and only one member from each type and assign it to the corresponding member in the rspi_command_word_t structure. For lower 16 bits, set the dummy data (RSPI_SPCMD_DUMMY).

Clock phase

The combination of the CPHA (clock phase) and CPOL (clock resting polarity) determine the “SPI mode setting”

Note: For slave-mode operation RSPI only supports sampling on even edge. This corresponds to what is sometimes referred to as SPI Mode-1, or Mode-3.

Type: rspi_spcmd_cpha_t

Members: RSPI_SPCMD_CPHA_SAMPLE_ODD // Data sampling on odd edge, data variation on even edge.
RSPI_SPCMD_CPHA_SAMPLE_EVEN // Data variation on odd edge, data sampling on even edge.

Clock polarity

Type: rspi_spcmd_cpol_t
Members: RSPI_SPCMD_CPOL_IDLE_LO // RSPCK is low when idle.
RSPI_SPCMD_CPOL_IDLE_HI // RSPCK is high when idle.

Clock base rate division

The SPI clock base bit rate setting will be further divided by this. (Note 1)

Type: rspi_spcmd_br_div_t
Members: RSPI_SPCMD_BR_DIV_1 // Select the base bit rate
RSPI_SPCMD_BR_DIV_2 // Select the base bit rate divided by 2
RSPI_SPCMD_BR_DIV_4 // Select the base bit rate divided by 4
RSPI_SPCMD_BR_DIV_8 // Select the base bit rate divided by 8

Note 1 : The bit rate specified in R_RSPI_Open() or R_RSPI_Control() is based on no frequency division (RSPI_SPCMD_BR_DIV_1). To divide the selected bit rate, change the setting of this bit.

Slave select to be asserted during transfer operation.

Type: rspi_spcmd_ssl_assert_t
Members: RSPI_SPCMD_ASSERT_SSL0 // Select SSL0
RSPI_SPCMD_ASSERT_SSL1 // Select SSL1
RSPI_SPCMD_ASSERT_SSL2 // Select SSL2
RSPI_SPCMD_ASSERT_SSL3 // Select SSL3

Slave select negation.

This bit determines whether the RSPI will deassert the slave select signal after each frame, or keep it asserted.

Type: rspi_spcmd_ssl_negation_t
Members: RSPI_SPCMD_SSL_NEGATE // Negates all SSL signals upon completion of transfer.
RSPI_SPCMD_SSL_KEEP // Keep SSL signal level from end of transfer until start of next.

Frame data length

The number of bits in each SPI data frame.

Type: rspi_spcmd_bit_length_t

Members:

RSPI_SPCMD_BIT_LENGTH_8	// 8 bits data length
RSPI_SPCMD_BIT_LENGTH_9	// 9 bits data length
RSPI_SPCMD_BIT_LENGTH_10	// 10 bits data length
RSPI_SPCMD_BIT_LENGTH_11	// 11 bits data length
RSPI_SPCMD_BIT_LENGTH_12	// 12 bits data length
RSPI_SPCMD_BIT_LENGTH_13	// 13 bits data length
RSPI_SPCMD_BIT_LENGTH_14	// 14 bits data length
RSPI_SPCMD_BIT_LENGTH_15	// 15 bits data length
RSPI_SPCMD_BIT_LENGTH_16	// 16 bits data length
RSPI_SPCMD_BIT_LENGTH_20	// 20 bits data length
RSPI_SPCMD_BIT_LENGTH_24	// 24 bits data length
RSPI_SPCMD_BIT_LENGTH_32	// 32 bits data length

Data transfer bit order.

Type: rspi_spcmd_bit_order_t

Members:

RSPI_SPCMD_ORDER_MSB_FIRST	// MSB first.
RSPI_SPCMD_ORDER_LSB_FIRST	// LSB first.

RSPI signal delays

Type: rspi_spcmd_spnden_t

Members:

RSPI_SPCMD_NEXT_DLY_1	// A next-access delay of 1 RSPCK +2 PCLK.
RSPI_SPCMD_NEXT_DLY_SSLND	// Next-access delay = next access delay register (SPND)

Type: rspi_spcmd_slnden_t

Members:

RSPI_SPCMD_SSL_NEG_DLY_1	// An SSL negation delay of 1 RSPCK.
RSPI_SPCMD_SSL_NEG_DLY_SSLND	// Delay = SSL negation delay register (SSLND)

Type: rspi_spcmd_sckden_t

Members:

RSPI_SPCMD_CLK_DLY_1	// An RSPCK delay of 1 RSPCK.
RSPI_SPCMD_CLK_DLY_SPCKD	// Delay = setting of RSPI clock delay register (SPCKD).

Dummy data

Type: rspi_spcmd_dummy_t

Members:

RSPI_SPCMD_DUMMY	// upper 16-bit dummy data

2.14.1 Complete command word data structure.

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

```
typedef union rspi_command_word_s
{
    R_BSP_ATTRIB_STRUCT_BIT_ORDER_RIGHT_11(
        rspi_spcmd_cpha_t           cpha           :1,
        rspi_spcmd_cpol_t           cpol           :1,
        rspi_spcmd_br_div_t         br_div         :2,
        rspi_spcmd_ssl_assert_t     ssl_assert     :3,
        rspi_spcmd_ssl_negation_t   ssl_negate     :1,
        rspi_spcmd_bit_length_t     bit_length     :4,
        rspi_spcmd_bit_order_t      bit_order      :1,
        rspi_spcmd_spnden_t         next_delay     :1,
        rspi_spcmd_slnden_t         ssl_neg_delay  :1,
        rspi_spcmd_sckden_t         clock_delay    :1,
        rspi_spcmd_dummy_t          dummy          :16
    );
    uint16_t word[2];
} rspi_command_word_t;
```

Example of command word initialization

```
static const rspi_command_word_t my_command_reg_word = {
        RSPI_SPCMD_CPHA_SAMPLE_ODD,
        RSPI_SPCMD_CPOL_IDLE_LO,
        RSPI_SPCMD_BR_DIV_1,
        RSPI_SPCMD_ASSERT_SSL0,
        RSPI_SPCMD_SSL_KEEP,
        RSPI_SPCMD_BIT_LENGTH_8,
        RSPI_SPCMD_ORDER_MSB_FIRST,
        RSPI_SPCMD_NEXT_DLY_SSLND,
        RSPI_SPCMD_SSL_NEG_DLY_SSLND,
        RSPI_SPCMD_CLK_DLY_SPCKD,
        RSPI_SPCMD_DUMMY,
};
```

2.15 “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 */
```

2.16 Peripheral Functions and Modules Other than RSPI

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

- DMA controller (DMAC)
- Data transfer controller (DTC)
- Long queue (LONGQ) software module

2.16.1 DMAC/DTC

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

The RSPI 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.

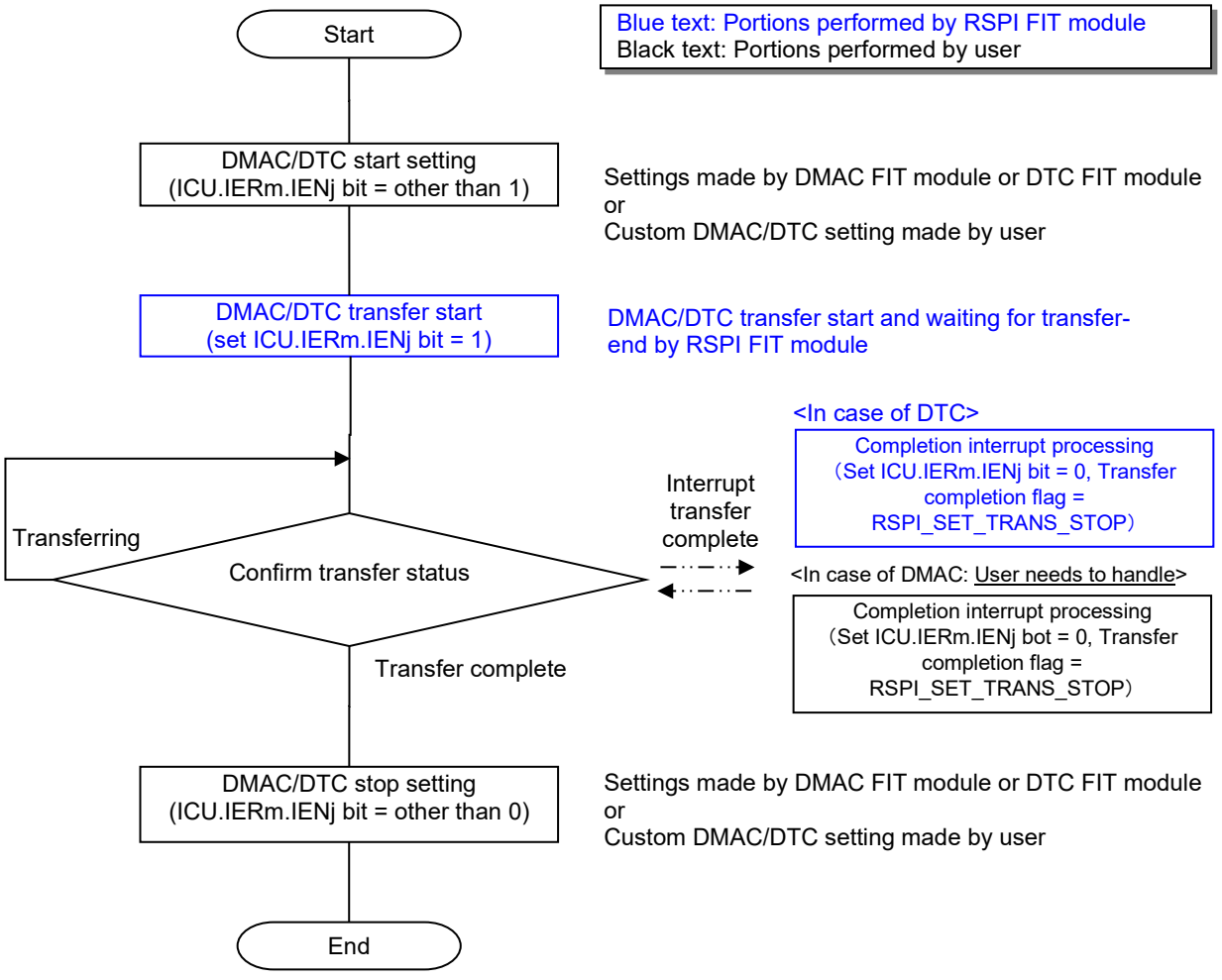

Figure 2-1 Processing for DMAC Transfer and DTC Transfer Settings

2.16.2 LONGQ

The LONGQ FIT module is used by the functionality that fetches the error log.

An example of control utilizing the LONGQ FIT module is included in the RSPI FIT module. The default setting of the relevant configuration option of the RSPI FIT module disables the error log fetching functionality. See 2.7, “Compile Option”.

2.16.2.1 R_LONGQ_Open() setting

Set to 1 ignore_overflow, the third argument of the R_LONGQ_Open() function of LONGQ FIT module. This allows the error log buffer to be used as a ring buffer.

2.16.2.2 Control procedure

Before calling R_RSPI_Open(), call the following functions in the order shown.

1. R_LONGQ_Open()
2. R_RSPI_SetLogHdlAddress()

3. API Functions

R_RSPI_Open()

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

Format

```
rspi_err_t   R_RSPI_Open(uint8_t                 channel,
                         rspi_chnl_settings_t  *pconfig,
                         rspi_command_word_t   spcmd_command_word,
                         void                  (*pcallback)(void *pcbdat),
                         rspi_handle_t         *phandle);
```

Parameters

channel
: Number of the RSPI channel to be initialized

*pconfig
: Pointer to RSPI channel configuration data structure.

spcmd_command_word
: SPCMD command data structure.

(*pcallback)(void * pcbdat)
: Pointer to user defined function called from interrupt.

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

Return Values

RSPI_SUCCESS:	Successful; channel initialized
RSPI_ERR_BAD_CHAN:	Channel number is not available
RSPI_ERR_CH_NOT_CLOSED:	Channel currently in operation; Perform R_RSPI_Close() first
RSPI_ERR_NULL_PTR:	*pconfig pointer or *phandle pointer is NULL
RSPI_ERR_ARG_RANGE:	An element of the *pconfig structure contains an invalid value.
RSPI_ERR_LOCK:	The lock could not be acquired.

Properties

Prototyped in file “r_rspi_rx_if.h”

Description

The Open function is responsible for preparing an RSPI channel for operation. This function must be called once prior to calling any other RSPI API functions (except R_RSPI_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 RSPI channel without first performing a "close" by calling R_RSPI_Close().

Communication is not yet available upon completion of this processing. Set MPC and PMR in the I/O ports to peripheral module.

Example

```
   /* Initialize demo command word settings.
    * This can be constant if you don't need to change the settings. */
   static const rspi_command_word_t my_rspi_command = {
           RSPI_SPCMD_CPHA_SAMPLE_EVEN,
           RSPI_SPCMD_CPOL_IDLE_LO,
           RSPI_SPCMD_BR_DIV_1,
           RSPI_SPCMD_ASSERT_SSL0,
           RSPI_SPCMD_SSL_KEEP,
           RSPI_SPCMD_BIT_LENGTH_32,
           RSPI_SPCMD_ORDER_MSB_FIRST,
           RSPI_SPCMD_NEXT_DLY_SSLND,
           RSPI_SPCMD_SSL_NEG_DLY_SSLND,
           RSPI_SPCMD_CLK_DLY_SPCKD,
           RSPI_SPCMD_DUMMY,
   };

   /* Conditions: Channel not yet open. */
   uint8_t chan = 0;
   rspi_handle_t handle;
   rspi_chnl_settings_t my_config;
   rspi_cmd_baud_t my_setbaud_struct;
   rspi_err_t rspi_result;

   my_config.gpio_ssl          = RSPI_IF_MODE_4WIRE;
   my_config.master_slave_mode = RSPI_MS_MODE_MASTER;
   my_config.bps_target        = 4000000; // Bit rate in bits-per-second.
   my_config.tran_mode         = RSPI_TRANS_MODE_SW;
   rspi_result = R_RSPI_Open(chan, &my_config, my_rspi_command, &test_callback,
&handle );

   if (RSPI_SUCCESS != rspi_result)
   {
       return rspi_result;
   }

   /* Initialize I/O port pins for use with the RSPI peripheral.
    * This is specific to the MCU and ports chosen. */
    rspi_64M_init_ports();
```

Special Notes

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

- The DMAC FIT module, DTC FIT module, and timer module (CMT FIT module, for example) must be obtained separately.

R_RSPI_Control()

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

Format

```
rspi_err_t     R_RSPI_Control(rspi_handle_t  handle,
                              rspi_cmd_t     cmd,
                              void          *pcmd_data);
```

Parameters

handle

Handle for the channel

cmd

Enumerated command code.

Available command codes:

RSPI_CMD_SET_BAUD // Change the base bit rate setting without reinitializing the RSPI channel.
RSPI_CMD_ABORT, // Stop the current read or write operation immediately.
RSPI_CMD_SETREGS, // Set all supported RSPI regs in one operation. (Expert use only)
RSPI_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

Return Values

RSPI_SUCCESS:	Command successfully completed.
RSPI_ERR_CH_NOT_OPENED:	The channel has not been opened. Perform R_RSPI_Open() first
RSPI_ERR_UNKNOWN_CMD:	Control command is not recognized.
RSPI_ERR_NULL_PTR:	*pcmd_data pointer or *phandle pointer is NULL.
RSPI_ERR_ARG_RANGE:	An element of the *pcmd_data structure contains an invalid value.
RSPI_ERR_LOCK:	The lock could not be acquired.

Properties

Prototyped in file “r_rspi_rx_if.h”

Description

This function is responsible for handling special hardware or software operations for the RSPI channel. It takes an RSPI handle to identify the selected RSPI, 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_rspi_rx_if.h".

Table 3.1 R_RSPI_Control()

Command	argument pcmd_data	Description
RSPI_CMD_SET_BAUD	rspi_cmd_baud_t *	Change the bit rate setting without reinitializing the RSPI channel.
RSPI_CMD_ABORT	FIT_NO_PTR	Stop the current read or write operation immediately.
RSPI_CMD_SETREGS	rspi_cmd_setregs_t *	Set all supported RSPI regs in one operation. Expert use only.
RSPI_CMD_SET_TRANS_MODE	rspi_cmd_trans_mode_t *	Set the SW/DMAC/DTC transfer mode.

Example

```
my_setbaud_struct.bps_target = 4000000; // Set for 4 Mbps
rspi_result = R_RSPI_Control(handle, RSPI_CMD_SET_BAUD,  &my_setbaud_struct);
if (RSPI_SUCCESS != rspi_result)
{
    return error;
}

...
/* This is taking too long, stop the current transfer now! */
rspi_result = R_RSPI_Control(handle, RSPI_CMD_ABORT,  FIT_NO_PTR);
```

Special Notes:

Control function command codes.

```
typedef enum rspi_cmd_e
{
    RSPI_CMD_SET_BAUD = 1,
    RSPI_CMD_ABORT,     // Stop the current read or write operation
immediately.
    RSPI_CMD_SETREGS,   // Set all supported RSPI regs in one operation.
    RSPI_CMD_SET_TRANS_MODE, // Set the data transfer mode.
    RSPI_CMD_UNKNOWN    // Not a valid command.
} rspi_cmd_t;
```

Data structure for the Set Baud command. This command sets the base-bit rate for the specified channel. The value specified in ‘bps_target’ may not be what actually gets set. The function will try to find a setting to match, but if the requested bit rate is not possible based on the divisor ratios available, then the function will set the next lower available bit-rate. SPCMD.BRDV[1:0] bit is based on zero (no frequency division).

```
typedef struct rspi_cmd_baud_s
{
    uint32_t    bps_target;   // The target bits-per-second setting for the
channel.
} rspi_cmd_baud_t;
```

Using the RSPI_CMD_SETREGS command, the RSPI register setting information can be changed. To use RSPI_CMD_SETREGS command, create the instance with as-needed setting value first, then call R_RSPI_Control() to pass the pointer as argument.

```
typedef struct rspi_cmd_setregs_s
{
    uint8_t sslp_val;   /* RSPI Slave Select Polarity Register (SSLP) */
    uint8_t sppcr_val;  /* RSPI Pin Control Register (SPPCR) */
    uint8_t spckd_val;  /* RSPI Clock Delay Register (SPCKD) */
    uint8_t sslnd_val;  /* RSPI Slave Select Negation Delay Register (SSLND)
*/
    uint8_t spnd_val;   /* RSPI Next-Access Delay Register (SPND) */
    uint8_t spcr2_val;  /* RSPI Control Register 2 (SPCR2) */
    uint8_t spdcr2_val; /* RSPI Data Control Register 2 (SPDCR2) */
#if defined(BSP_MCU_RX671)
    uint8_t spcr3_val;  /* RSPI Control Register 3 (SPCR3) */
#endif
} rspi_cmd_setregs_t;
```

Data structure for the Set Transfer mode command. This command is used to change the setting of data transfer mode. There are three kinds of mode for RSPI_TRANS_MODE_SW, RSPI_TRANS_MODE_DMAC and RSPI_TRANS_MODE_DTC.

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

R_RSPI_Close()

Fully disables the RSPI channel designated by the handle.

Format

```
RSPI_err_t      R_RSPI_Close(rspi_handle_t handle);
```

Parameters

handle

Handle for the channel

Return Values

RSPI_SUCCESS:	Successful; channel closed
RSPI_ERR_CH_NOT_OPENED:	The channel has not been opened so closing has no effect.
RSPI_ERR_NULL_PTR:	A required pointer argument is NULL

Properties

Prototyped in file “r_rspi_rx_if.h”

Description

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

Example

```
RSPI_err_t  rspi_result;

rspi_result = R_RSPI_Close(handle);

if (RSPI_SUCCESS != rspi_result)
{
    return rspi_result;
}
```

R_RSPI_Write()

The Write function transmits data to the selected SPI device

Format

```
rspi_err_t     R_RSPI_Write(rspi_handle_t       handle,
                            rspi_command_word_t spcmd_command_word,
                            void               *psrc,
                            uint16_t            length);
```

Parameters

handle

Handle for the channel

spcmd_command_word

Bit field data consisting of all the RSPI command register settings for SPCMD for this operation. See 2.14 Typedef enumerations used for the command settings word.

*psrc

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 spcmd_command_word.bit_length, the *psrc 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 spcmd_command_word.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

RSPI_SUCCESS:	Write operation successfully completed.
RSPI_ERR_CH_NOT_OPENED:	The channel has not been opened. Perform R_RSPI_Open() first.
RSPI_ERR_NULL_PTR:	A required pointer argument is NULL.
RSPI_ERR_LOCK:	The lock could not be acquired. The channel is busy.
RSPI_ERR_INVALID_ARG:	Argument is not valid for parameter.

Properties

Prototyped in file “r_rspi_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.

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

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;

rspi_result = R_RSPI_Write(handle, my_command_word, source, length);
if (RSPI_SUCCESS != rspi_result)
{
   if (RSPI_ERR_LOCK == rspi_result)
   {
       // Channel must be busy. Try again later.
   }
   return error;
}

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

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

- For the callback function that occurs when communication ends, see 1.6, Basic Operations (In DMAC/DTC).
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.

R_RSPI_Read()

The Read function receives data from the selected SPI device.

Format

```
rspi_err_t     R_RSPI_Read(rspi_handle_t         handle,
                           rspi_command_word_t   spcmd_command_word,
                           void                 *pdest,
                           uint16_t              length);
```

Parameters

handle

Handle for the channel

spcmd_command_word

Bit field data consisting of all the RSPI command register settings for SPCMD for this operation. See 2.14 Typedef enumerations used for the command settings word.

*pdest

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 spcmd_command_word.bit_length, the *pdest 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 spcmd_command_word.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

RSPI_SUCCESS:	Read operation successfully completed.
RSPI_ERR_CH_NOT_OPENED:	The channel has not been opened. Perform R_RSPI_Open() first.
RSPI_ERR_NULL_PTR:	A required pointer argument is NULL.
RSPI_ERR_LOCK:	The lock could not be acquired. The channel is busy.
RSPI_ERR_INVALID_ARG:	Argument is not valid for parameter.

Properties

Prototyped in file “r_rspi_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 RSPI is operating as Master or Slave. If the RSPI is configured as slave, then data will only transfer when clocks are received from the Master. While receiving data, the RSPI will also transmit the user definable Dummy data pattern defined in the configuration file.

Example

```
/* Conditions: Channel currently open. */
...
g_transfer_complete = false; // state flag variable defined elsewhere.

rspi_result = R_RSPI_Read(handle, my_command_word, dest, length);
if (RSPI_SUCCESS != rspi_result)
{
   return error;
}

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

When using as a master reception, select normal mode or high speed mode in advance. For details of the setting method, see 2.7, Compile Option.

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

- For the callback function that occurs when communication ends, see 1.6, Basic Operations (In DMAC/DTC).
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.

R_RSPI_WriteRead()

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

Format

```
rspi_err_t     R_RSPI_WriteRead(rspi_handle_t         handle,
                                rspi_command_word_t   spcmd_command_word,
                                void                 *psrc,
                                void                 *pdest,
                                uint16_t              length);
```

Parameters

handle

Handle for the channel

spcmd_command_word

Bit field data consisting of all the RSPI settings for the command register (SPCMD) for this operation. See 2.14 Typedef enumerations used for the command settings word.

*psrc

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 spcmd_command_word.bit_length, the *psrc 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.

*pdest

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 spcmd_command_word.bit_length, the *pdest 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 spcmd_command_word.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

RSPI_SUCCESS:	Read operation successfully completed.
RSPI_ERR_CH_NOT_OPENED:	The channel has not been opened. Perform R_RSPI_Open() first.
RSPI_ERR_NULL_PTR:	A required pointer argument is NULL.
RSPI_ERR_LOCK:	The lock could not be acquired. The channel is busy.
RSPI_ERR_INVALID_ARG:	Argument is not valid for parameter.

Properties

Prototyped in file “r_rspi_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 user-defined callback 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 RSPI is operating as Master or Slave. If the RSPI 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;
rspi_result = R_RSPI_WriteRead(handle, my_command_word, source, dest, length);
if (RSPI_SUCCESS != rspi_result)
{
    return error;
}

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

When using as a master transmission and reception, select normal mode or high speed mode in advance. For details of the setting method, see 2.7, Compile Option.

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

- For the callback function that occurs when communication ends, see 1.6, Basic Operations (In DMAC/DTC).
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.

R_RSPI_GetVersion()

This function returns the driver version number at runtime.

Format

```
uint32_t  R_RSPI_GetVersion(void);
```

Parameters

None

Return Values

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

Properties

Prototyped in file “r_rspi_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_RSPI_GetVersion();

version_high = (version >> 16)&0xf;
version_low  =  version & 0xff;

sprintf(version_str, "RSPIv%1.1hu.%2.2hu", version_high, version_low);
```

R_RSPI_GetBuffRegAddress()

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

Format

```
rspi_err_t R_RSPI_GetBuffRegAddress(
rspi_handle_t handle,
uint32_t * p_spdr_adr
)
```

Parameters

handle
RSPI handle number
* p_spdr_adr
The pointer for storing the address of SPDR. Set this to the address of the storage destination.

Return Values

RSPI_SUCCESS:	Successful operation.
RSPI_ERR_INVALID_ARG:	Argument is not valid for parameter.
RSPI_ERR_NULL_PTR	A required pointer argument is NULL

Properties

Prototype declarations are contained in r_rspi_rx_if.h.

Description

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

Example

```
uint32_t       reg_buff;
rspi_err_t     ret = RSPI_SUCCESS;
rspi_handle_t  handle;

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

Special Notes

None

R_RSPI_IntSptilerClear()

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

Format

```
rspi_err_t R_RSPI_IntSptiIerClear(
rspi_handle_t handle
)
```

Parameters

handle

RSPI handle number

Return Values

RSPI_SUCCESS:	Successful operation.
RSPI_ERR_NULL_PTR	A required pointer argument is NULL

Properties

Prototype declarations are contained in r_rspi_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_RSPI_DisableSpti().

Example

```
DMA_Handler_W()
{
   R_RSPI_DisableSpti(my_rspi_handle);
   R_RSPI_IntSptiIerClear(my_rspi_handle);
}
```

Special Notes

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

Doing so could disrupt the transfer.

R_RSPI_IntSprilerClear()

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

Format

```
rspi_err_t R_RSPI_IntSpriIerClear(
rspi_handle_t handle
)
```

Parameters

handle

RSPI handle number

Return Values

RSPI_SUCCESS:	Successful operation.
RSPI_ERR_NULL_PTR	A required pointer argument is NULL

Properties

Prototype declarations are contained in r_rspi_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_RSPI_DisableRSPI().

Example

```
DMA_Handler_R()
{
   R_RSPI_IntSpriIerClear(my_rspi_handle);
   R_RSPI_DisableRSPI(my_rspi_handle);
}
```

Special Notes

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

Doing so could disrupt the transfer.

R_RSPI_DisableSpti()

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

Format

```
rspi_err_t R_RSPI_DisableSpti(
rspi_handle_t handle
)
```

Parameters

handle

RSPI handle number

Return Values

RSPI_SUCCESS:	Successful operation.
RSPI_ERR_NULL_PTR	A required pointer argument is NULL

Properties

Prototype declarations are contained in r_rspi_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_RSPI_IntSptiIerClear().

Example

```
DMA_Handler_R()
{
   R_RSPI_DisableSpti(my_rspi_handle);
   R_RSPI_IntSpriIerClear(my_rspi_handle);
}
```

Special Notes

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

Doing so could disrupt the transfer.

R_RSPI_DisableRSPI()

This function is set to disable the RSPI function.

Format

```
rspi_err_t R_RSPI_DisableRSPI(
rspi_handle_t handle
)
```

Parameters

handle

RSPI handle number

Return Values

RSPI_SUCCESS:	Successful operation.
RSPI_ERR_NULL_PTR	A required pointer argument is NULL

Properties

Prototype declarations are contained in r_rspi_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_RSPI_IntSprilerClear().

Example

```
DMA_Handler_R()
{
   R_RSPI_IntSpriIerClear(my_rspi_handle);
   R_RSPI_DisableRSPI(my_rspi_handle);
}
```

Special Notes

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

Doing so could disrupt the transfer.

R_RSPI_SetLogHdlAddress()

This function specifies the handler address for the LONGQ FIT module. Call this function when using error log acquisition processing.

Format

```
rspi_err_t R_RSPI_SetLogHdlAddress(
uint32_t user_long_que
)
```

Parameters

user_long_que

Specify the handler address of the LONGQ FIT module.

Return Values

RSPI_SUCCESS: Successful operation

Properties

Prototype declarations are contained in r_rspi_rx_if.h.

Description

The handler address of the LONGQ FIT module is set in the RSPI FIT module.

It uses the LONGQ FIT module to perform preparatory processing for fetching the error log.

Run this processing before calling R_RSPI_Open().

Example

```
#define ERR_LOG_SIZE (16)
#define RSPI_USER_LONGQ_IGN_OVERFLOW    (1)

rspi_err_t                  ret = RSPI_SUCCESS;
uint32_t                        MtlLogTbl[ERR_LOG_SIZE];
longq_err_t                     err;
longq_hdl_t                     p_rspi_user_long_que;
uint32_t                        long_que_hndl_address;


/* Open LONGQ module. */
err = R_LONGQ_Open(&MtlLogTbl[0],
                        ERR_LOG_SIZE,
                        RSPI_USER_LONGQ_IGN_OVERFLOW,
                        &p_rspi_user_long_que
);

long_que_hndl_address = (uint32_t)p_rspi_user_long_que;
ret = R_RSPI_SetLogHdlAddress(long_que_hndl_address);
```

Special Notes

Incorporate the LONGQ FIT module separately. Also, enable the macro RSPI_CFG_LONGQ_ENABLE in r_rspi_rx_config.h.

If RSPI_CFG_LONGQ_ENABLE == 0 and this function is called, this function does nothing.

4. Pin Setting

To use the RSPI FIT module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). This pin allocation is referred to as “pin setting” in this document. Please perform the pin setting after calling the R_RSPI_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 output according to the option selected in the Pin Setting window in the Smart configurator. Pins are configured by calling the function defined in the source file. Refer to Table 4.1 for details.

Table 4.1 Function Output by the Smart Configurator

Option Selected	Function to be Output
Channel 0	R_RSPI_PinSet_RSPI0()
Channel 1	R_RSPI_PinSet_RSPI1()
Channel 2	R_RSPI_PinSet_RSPI2()

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 rspi_command_word_t structure rspi_spcmd_cpol_t, which sets the polarity of the RSPCK pin is updated when R_RSPI_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 contains three sample programs that demonstrate basic usage of the FIT RSPI Module. The sample program is intended to provide a quick functional example of common API function calls in use.

The provided sample program rx65n_rsk_rspi_sample 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 RSPI module version number is retrieved and can be displayed on the Renesas Virtual Debug Console window if desired.

The provided sample programs rx65n_rsk_rspi_master_sample and rx231_rsk_rspi_slave_sample use RSKRX65N as a Master and RSKRX231 as a Slave to realize Master-Slave transmission and reception. Data received is tested to confirm that it matches the data sent. The RSPI 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

5.2.1 rx65n_rsk_rspi_sample

1. Prepare the board by jumper connecting the MOSIA terminal of RSKRX65N to the MISOA terminal. In this sample program, connect the expansion header J13 pin 2 to J11 pin 2.
2. Build and download the sample application to the RSK board using the e^2 studio debugger.
3. Select the Renesas Virtual Debug Console view in e^2 studio to view print information.
4. Run the application in the debugger.
5. Observe the version number print in the debug console window.
6. Transfer multiple times. "Success!" is displayed in the debug console window if the transfer is successful each time, and "Failed." if it fails.

Note: When using the source code of this demo project with other devices, the pins to be connected differ depending on the target board. Also, refer to the user's manual of other devices and the circuit diagram of the target board and change the pin settings.

a) RSKRX113
   i) Connect expansion header J3 pin 24 to J3 pin 23.

b) RSKRX64M and RSKRX71M
   i) Remove any jumper plugs from board jumpers J14 and J12.
   ii) Connect J14 pin 2 to J12 pin 2.

c) RSKRX231
   i) Connect expansion header J3 pin 14 to J3 pin 13.

d) RSSKRX23E-A
   i) Connect expansion header J3 pin 10 to J3 pin 9.

e) RSSKRX23W
   i) Connect U3 pin 5 to U3 pin 6.

f) RSKRX65N-2MB
   i) Connect expansion header JA3 pin 7 to JA3 pin 8.
   ii) Turn off SW4 pin 3 and SW4 pin4.

g) RSKRX660
   i) Connect expansion header J3 pin 17 to JA3 pin 8.

h) RSKRX66T
   i) Connect expansion header J1 pin 23 to J1 pin 24.

i) RSKRX671
   i) Connect expansion header JA2 pin 17 to JA2 pin 18.

j) RSKRX72T
   i) Connect expansion header J1 pin 28 to J1 pin 29.

k) RSKRX72M
   i) Connect expansion header PMOD1 pin 3 to J12 pin 2.

l) RSKRX72N
   i) Connect expansion header JA3 pin 6 to JA3 pin 7.

m) Target board for RX140
   i) Connect expansion header CN2 pin 17 to CN2 pin 18.

5.2.2 rx660_rsk_rspi_sample, rx660_rsk_rspi_sample_gcc

1. Prepare the board by jumper connecting the MOSIA terminal of RSKRX660 to the MISOA terminal. In this sample program, connect the expansion header J3 pin 17 to J3 pin 16.
2. Build and download the sample application to the RSK board using the e^2 studio debugger.
3. Select the Renesas Virtual Debug Console view in e^2 studio to view print information.
4. Run the application in the debugger.
5. Observe the version number print in the debug console window.
6. Transfer multiple times. "Success!" is displayed in the debug console window if the transfer is successful each time, and "Failed." if it fails.

5.2.3 rx65n_rsk_rspi_master_sample and rx231_rsk_rspi_slave_sample

1. Connect RSPCKA, MOSIA, MISOA and SSLA0 of RSKRX65N to RSPCKA, MOSIA, MISOA and SSLA0 of RSKRX231 respectively. See the table below for detailed connection information.

Table 5.1 Expansion header information connecting RSKRX65N and RSKRX231

Pin name	RSKRX65N	RSKRX231
RSPCKA	JA3-Pin6	J3-Pin15
MOSIA	JA3-Pin7	J3-Pin14
MISOA	JA3-Pin8	J3-Pin13
SSLA0	JA3-Pin5	J3-Pin16

2. Build and download the Master sample program and Slave sample program to the RSK board using the e^2 studio debugger.
3. Select the Renesas Virtual Debug Console view in e^2 studio to view print information.
4. Run the Slave sample program in the debugger, then the Master sample program.

5. Observe the version number print in the debug console window.
6. Send and receive multiple times, and first display the number of processing (starting from 0) in the debug console window. "Success!" will be displayed in the debug console window of the master and slave if the received data each time matches the transmitted data, and "Failed." if they do not match.

6. Appendix

6.1 Operation Confirmation Environment

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

Table 6.1: Operating Environment (Ver.2.00)

Item	Contents
Integrated development environment	Renesas Electronics e² studio V7.3.0
C compiler	Renesas Electronics C/C++ compiler for RX Family V.3.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big-endian/Little-endian
Version of the module	Ver.2.00
Board used	Renesas Starter Kit for RX130 (RTK5005130xxxxxxxx) Renesas Starter Kit for RX130-512KB (RTK5051308xxxxxxxx) Renesas Starter Kit for RX24T (RTK500524Txxxxxxxx) Renesas Starter Kit for RX24U (RTK500524Uxxxxxxxx) Renesas Starter Kit+ for RX64M (R0K50564Mxxxxxxxx) Renesas Starter Kit+ for RX65N (RTK500565Nxxxxxxxx) Renesas Starter Kit+ for RX65N-2MB (RTK50565N2xxxxxxxx) Renesas Starter Kit for RX66T (RTK50566Txxxxxxxxx) Renesas Starter Kit for RX72T (RTK5572Txxxxxxxxxx)

Table 6.2: Operating Environment (Ver.2.01)

Item	Contents
Integrated development environment	Renesas Electronics e² studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.2.01
Board used	Renesas Starter Kit+ for RX65N (RTK500565Nxxxxxx)

Table 6.3: Operating Environment (Ver.2.02)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endianness	Big-endian/Little-endian
Version of the module	Ver.2.02
Board used	Renesas Solution Starter Kit for RX23W (RTK5523Wxxxxxxxxx)

Table 6.4: Operating Environment (Ver.2.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 IAR Embedded Workbench for Renesas 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.2.03
Board used	Renesas Starter Kit+ for RX72M (RTK5572Mxxxxxxxxx)

Table 6.5: Operating Environment (Ver.2.04)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 IAR Embedded Workbench for Renesas 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.2.04
Board used	Renesas Starter Kit+ for RX72N (RTK5572Nxxxxxxxxx)

Table 6.6: Operating Environment (Ver.2.05)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.7.0 IAR Embedded Workbench for Renesas 4.13.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.201904 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.13.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.2.05
Board used	Renesas Solution Starter Kit for RX23E-A (RTK0ESXB10C00001BJ)

Table 6.7: Operating Environment (Ver.3.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2020-07 IAR Embedded Workbench for Renesas 4.14.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202002 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.00
Board used	Renesas Starter Kit+ for RX72N (RTK5572Nxxxxxxxxxx)

Table 6.8: Operating Environment (Ver.3.01)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2021-01 (21.1.0) IAR Embedded Workbench for Renesas 4.14.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202002 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.01
Board used	Renesas Starter Kit+ for RX671 (RTK55671xxxxxxxxx)

Table 6.9: Operating Environment (Ver.3.02)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2021-07 (21.7.0) IAR Embedded Workbench for Renesas 4.20.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202102 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.02
Board used	Target board for RX140 (RTK5RX140xxxxxxxxx)

Table 6.10: Operating Environment (Ver.3.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2021-07 (21.7.0) IAR Embedded Workbench for Renesas 4.20.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202102 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.03
Board used	Renesas Starter Kit+ for RX65N (RTK500565Nxxxxxx) Renesas Starter Kit for RX24T (RTK500524Txxxxxxxx) Renesas Starter Kit for RX231 (R0K505231xxxxxx)

Table 6.11: Operating Environment (Ver.3.04)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2022-04 (22.4.0) IAR Embedded Workbench for Renesas 4.20.03
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.04.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202104 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.03 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.04
Board used	Renesas Starter Kit for RX660 (RTK556609HCxxxxxBJ)

Table 6.12: Operating Environment (Ver.3.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2022-10 (22.10.0) IAR Embedded Workbench for Renesas 4.20.03
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202204 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.03 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.10
Board used	Renesas Starter Kit for RX660 (RTK556609HCxxxxxBJ) Renesas Flexible Motor Control Kit for RX26T (Part Number: RTK0EMXE70S00020BJ)

Table 6.13: Operating Environment (Ver.3.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2023-04 (23.04.0) IAR Embedded Workbench for Renesas 4.20.03
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202204 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.03 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.20
Board used	Renesas Solution Starter Kit for RX23E-B (product No.: RTK0ES1001C00001BJ)

Table 6.14: Operating Environment (Ver.3.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2023-04 (23.04.0) IAR Embedded Workbench for Renesas 4.20.03
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202204 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.03 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.30
Board used	Renesas Solution Starter Kit for RX23E-B (product No.: RTK0ES1001C00001BJ)

Table 6.15: Operating Environment (Ver.3.40)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2023-07 (23.07.0) IAR Embedded Workbench for Renesas 4.20.03
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202305 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.03 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.40
Board used	Renesas Flexible Motor Control Kit for RX26T (Part Number: RTK0EMXE70S00020BJ)

Table 6.16: Operating Environment (Ver.3.50)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2024-07 (24.07.0) IAR Embedded Workbench for Renesas 5.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.06.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202405 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 5.10.1 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.50
Board used	Evaluation Kit for RX261 (product No.: RTK5EK2610S00011BJ)

Table 6.17: Operating Environment (Ver.3.51)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2025-01 (25.01.0) IAR Embedded Workbench for Renesas 5.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.07.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202411 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 5.10.1 Compiler option: The default settings of the integrated development environment.
Endianness	Big-endian/Little-endian
Version of the module	Ver.3.51
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_rspi_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: How to set RSPCK polarity before Read/Write APIs(R_RSPI_Write(), R_RSPI_Read(), R_RSPI_WriteRead()) ?

A: Set CPOL bit of SPCMD register directly before pin setting.

(4) Q: I have used the API function of R_RSPI_IntSptiDmacdtcFlagSet and R_RSPI_IntSpriDmacdtcFlagSet when upgrading from Rev.2.03 to Rev.2.04 or later. Then I got the error: the function is not defined in the current r_rspi_rx module.

A: The FIT module Rev.2.04 or later you added does not support the two function. You can operate normally just by deleting the above function call.

(5) Q: API functions (R_RSPI_Write(), R_RSPI_Read(), and R_RSPI_WriteRead()) send and receive more data than expected when updated from Rev.2.03 or earlier to Rev.2.04 or later?

A: The specification of the argument length of API functions (R_RSPI_Write(), R_RSPI_Read(), R_RSPI_WriteRead()) have changed in Rev.2.04. Before Rev.2.03, the specification was to specify a length value that was doubled or quadrupled to be byte length depending on the length of data per frame to communicate. Since Rev.2.04, length has been a specification that specifies the number of frames to communicate as it is. Therefore, if the data length of one frame is 32 bits and 16 bits, modify the length value of the argument.

If the data length of one frame is 32 bits before Rev.2.03, length will be four times the value, but in Rev.2.04 or later, please correct the length value to 1/4. If the data length of one frame is 16 bits before Rev.2.03, length will specify a value that is doubled, but in Rev.2.04 or later, please correct the length value to 1/2. If the data length of one frame is 8 bits, no modification is required.

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.

Technical Update Information

The following technical update applies to this module.

- TN-RX*-A147A/E
  The Technical Update describes how to check all data transmission completion without using interrupt.
  The contents of the Technical Update do not apply to the RSPI driver as it uses interrupt on completion of transmission.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov 15, 2013	--	First edition issued
1.20	April 4, 2014	--	Updated list of supported/tested MCUs
1.30	Jan 20, 2015	--	Updated list of supported/tested MCUs
		8	Added Section 2.9 Code Size and RAM usage
		32	Added Demo Project section Corrected fonts
1.40	Jun 29, 2015	1,3,9,33	Updated to include support for RX231
1.50	Sep 30, 2016	1	Updated to include support for RX65N, RX130, RX230, RX23T and RX24T.
		16	Changed the chapter number of API Functions to 3 from 6.
		34	Added Section 6.4 Relations of Data Output and RAM
1.60	Mar 31, 2017	--	Released the application note of the Japanese edition.
		1	Support for RX24U.
		6	Changed contents of Section 2.3 Operating Environment.
		8	Changed contents of Section 2.7 Support Toolchains.
		8	Updated size of Section 2.10 Code Size and RAM usage.
		40	Changed titles and contents of Section 7 Sample program, 7.1 Adding the Sample program to a Workspace and 7.2 Running the Sample program.
		41	Added Section 8 Appendix.
1.70	Jul 31, 2017	-	Added the following chapter. -Added 2.4 Interrupt vector -Added 2.7 Compile Option -Added 2.9 Argument -Added 2.10 Return values -Added 2.11 Callback Functions -Added 2.12 Adding the FIT Module to Your Project -Added 4. Ping Setting -Added 6.2 Troubleshooting Moved the following chapter. -Moved form 3.1 Overview to 1.3 Overview of APIs. -Moved from 6. Data transfer operations to 1.5 Data transfer operations. -Moved form 2.5 RSPI features Supported by Driver to 1.2.1 RSPI features Supported by Driver. -Moved from 2.6 RSPI features not supported to 1.2.3 RSPI features not supported. Moved the following chapter. -Moved from 2.10 Code size and RAM usage to 2.8 Code size. Changed the following chapter contents. -Changed 5.2 Running the Sample program. -Changed 6.1 Operation Confirmation Environment. Deleted the following description -2.2 "Software Requirements" description on “cgc".
		1	Support for RX651.

Rev.	Date	Description	
		Page	Summary
1.80	Sep 20, 2018	1	Updated Related Documents. Support for RX66T.
		18	Added RX66T 2.4 Interrupt Vector.
		21	Changed contents of Section 2.8 Code Size.
		31	Added 2.15 “for”, “while” and “do while” statements
		44	Added description Setting RSPCK polarity.
		45	Changed RX66T 5.2 Running the Sample program
		46	Changed RX66T 6.1 Operation Confirmation Environment
		47	Added Troubleshooting (3).
2.00	Feb 20, 2019	1	Support for RX72T.
		7	Added new API in Section 1.3 Overview of APIs.
		9	Added DMAC/DTC transmission mode in Section 1.5.
		11-12	Changed the flowcharts in Section 1.6.1 Data transfer interrupts.
		21	Added RX72T in Section 2.4 Interrupt Vector.
		24	Added the macros in Section 2.7 Compile Option.
		25	Changed Section 2.8 Code Size.
		29	Changed Section 2.13 API data structures.
		35-36	Added Section 2.16 Peripheral Functions and Modules Other than RSPI.
		37-38	Changed Section 3.1 R_RSPI_Open().
		39-41	Changed Section 3.2 R_RSPI_Control().
		42	Changed Section 3.3 R_RSPI_Close().
		43-44	Changed Section 3.4 R_RSPI_Write().
		45-46	Changed Section 3.5 R_RSPI_Read().
		47-48	Changed Section 3.6 R_RSPI_WriteRead().
		50	Added Section 3.8 _RSPI_GetBuffRegAddress().
		51	Added Section 3.9 R_RSPI_IntSptilerClear().
		52	Added Section 3.10 R_RSPI_IntSprilerClear().
		53	Added Section 3.11 R_RSPI_IntSptiDmacdtcFlagSet().
		54	Added Section 3.12 R_RSPI_IntSpriDmacdtcFlagSet().
		55	Added Section 3.13 R_RSPI_SetLogHdlAddress().
		58	Added RX72T in Section 5.2 Running the Sample program.
		59	Changed Section 6.1 Operation Confirmation Environment.
2.01	May. 20. 2019	-	Update the following compilers: GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Added Target Compilers.
		1	Deleted R01AN1723 and R01AN1826 from Related Documents.
		3	1 Overview, fixed.
		6	Added IAR compiler restrictions in Section 1.2.2 Features Not Supported.
		20	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		25	2.8 Code Size, amended.

Rev.	Date	Description	
		Page	Summary
		28,45,47,49	Changed nop to BSP's built in function.
		59	Changed Section 6.1 Operation Confirmation Environment (Ver. 2.01).
		-	Deleted RX210, RX62N, RX62T, RX631, RX63N related note for these boards are not supported in the following versions.
		program	Deleted RX210, RX62N, RX62T, RX631, RX63N from Target Devices. Deleted RX210 and RX63N from Table 2.1. Deleted RX210, RX62N, RX62T, RX631, RX63N from 2.15.
2.02	Jun 20, 2019	1	Support for RX23W.
		21	Added RX23W in Section 2.4 Interrupt Vector.
		25	2.8 Code Size, amended.
		58	Added RX23W in Section 5.2 Running the Sample program.
		60	Changed Section 6.1 Operation Confirmation Environment (Ver. 2.02).
2.03	Jul 30, 2019	1	Support for RX72M.
		6	Deleted IAR compiler restrictions in Section 1.2.2 Features Not Supported.
		21	Added RX72M in Section 2.4 Interrupt Vector.
		25	2.8 Code Size, amended.
		38-55	Delete "Reentrant" item on the API description page.
		57	Added RX72M in Section 5.2 Running the Sample program.
		59	Changed Section 6.1 Operation Confirmation Environment (Ver. 2.03).
2.04	Nov 22, 2019	1	Support for RX72N, RX66N.
		8	Deleted R_RSPI_IntSptiDmacdtcFlagSet(),R_RSPI_IntSpriDmacdtcFlagSet() from 1.3.Overview of APIs.
		22	Added RX72M, RX66N in Section 2.4 Interrupt Vector.
		26	2.8 Code Size, amended.
		44-49	Changed "Special Notes" in 3.4,3.5,3.6.
		-	Deleted R_RSPI_IntSptiDmacdtcFlagSet(),R_RSPI_IntSpriDmacdtcFlagSet() from 3.API Function.
		54	Changed "Special Notes" in 3.11 R_RSPI_SetLogHdlAddress.
		56	Added RX72N, RX66N in Section 5.2 Running the Sample program.
		60	Changed Section 6.1 Operation Confirmation Environment (Ver. 2.04).
2.05	Mar 10, 2020	1	Support for RX23E-A.
		11	Updated contents of termination of DMAC communication in 1.6 Basic Operations(In DMAC/DTC).
		22	Added RX23E-A in Section 2.4 Interrupt Vector.
		24	Added RSPI_CFG_REQUIRE_LOCK restriction in 2.7 Compile Option.
		26	2.8 Code Size, amended.
		30	Changed Section 2.12 Adding the FIT Module to Your Project.
		56	Added RX23E-A in Section 5.2 Running the Sample program.
		60	Changed Section 6.1 Operation Confirmation Environment (Ver. 2.05).

Rev.	Date	Description	
		Page	Summary
3.00	Sep 10, 2020	8	Added R_RSPI_DisableSpti(), R_RSPI_DisableRSPI() from 1.3.Overview of APIs
		24	Deleted RSPI_CFG_REQUIRE_LOCK restriction in 2.7 Compile Option.
		26	2.8 Code Size, amended.
		30	Changed Section 2.12 Adding the FIT Module to Your Project.
		52-53	Changed “Description” and “Example” in 3.9 and 3.10.
		54-55	Added 3.11 R_RSPI_DisableSpti() and 3.12 R_RSPI_DisableRSPI() from 3.API Function.
		58	Changed 5 Sample program.
		63	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.00).
		program	RSPI FIT module fixed due to software failure. Description: The second communication cannot be started with the R_RSPI_WriteRead function in combination with the DMAC/DTC (the same phenomenon occurs with the R_RSPI_Write function and R_RSPI_Read function). Conditions: When the R_RSPI_WriteRead function is repeated in the DMAC/DTC transfer by referring to the example code in the application note and the sample code in the demo project, it always occurs after the second execution of R_RSPI_WriteRead. Corrective action: Please use RSPI FIT module Rev3.00. The following functions are added by this modification. R_RSPI_DisableSpti R_RSPI_DisableRSPI function The following functions are changed by this modification. rspi_spriX_isr functions (X = 0, 1, 2) rspi_sptiX_isr functions (X = 0, 1, 2) Corresponding tool news number: R20TS0590

Rev.	Date	Description	
		Page	Summary
3.00	Sep 10, 2020	program	RSPI FIT module fixed due to software failure. Description: When RSPI_CFG_REQUIRE_LOCK==1 and the second data transfer in DMAC/DTC transfer, RSPI_ERR_LOCK is returned and communication is not possible. Conditions: It will always occur when the second data transfer is performed in the DMAC/DTC transfer, referring to the example in the application note and the sample code in the demo project. Corrective action: Please use RSPI FIT module Rev3.00. The following functions are added by this modification. R_RSPI_DisableRSPI function
3.01	Jun 30,2021	1	Support for RX671.
		7	Added Idle interrupt and Communication end interrupt(RX671 only) in Section 1.2.1 Features Supported
		22	Added RX671 in Section 2.4 Interrupt Vector.
		26	2.8 Code Size, amended.
		59	Added RX671 in Section 5.2 Running the Sample program.
		63	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.01).
		64	Added Troubleshooting (4) and (5).
		program	RSPI FIT module fixed due to software failure. Description and Condition: 1. Buffer overflow occurs. In high-speed reception, if the timing of reading data from the data register is delayed due to other peripheral interrupts, buffer overflow will occur. 2. Received data not be received correctly. In high-speed reception, the reception buffer full interrupt is delayed due to other peripheral interrupts, the SPDR register is dummy-read by the transmission empty interrupt, which may result in the loss of received data. Corrective action: Please use rev.3.01 or a later version of the RSPI FIT module. Corresponding tool news number: R20TS0667
3.02	Jul 31,2021	1	Support for RX140.
		22	Added RX140 in Section 2.4 Interrupt Vector.
		26	2.8 Code Size, amended.
		60	Added RX140 in Section 5.2 Running the Sample program.
		65	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.02).
		66	Add Q(4) and Q(5) in section 6.2 Troubleshooting.

Rev.	Date	Description	
		Page	Summary
3.03	Oct 31,2021	7	Added IDLE interrupt application range(other than RX671) in Section 1.2.1 Features Supported.
		10	Added description of high speed mode.
		22	Added IDLE interrupt(other than RX671) in Section 2.4 Interrupt Vector.
		26	2.8 Code Size, amended.
		65	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.03).
		program	1. RSPI FIT module fixed due to software failure. Description and Condition: When the "R_RSPI_Write" function, "R_RSPI_Read" function, or "R_RSPI_WriteRead" function is called with the RSPI FIT setting set to master mode operation, the SSL negate delay and next access delay are waited after receiving the final data of the serial transfer. There is a problem that the RSPI function stops without. Corrective action: Please use rev.3.03 or a later version of the RSPI FIT module. Corresponding tool news number: R20TS0720 2. Remove unnecessary code.
3.04	Dec 31,2021	1	Added support for RX660.
		24	Added RX660 in Section 2.4 Interrupt Vector.
		28	2.8 Code Size, amended.
		62	Added RX660 in Section 5.2 Running the Sample program.
		67	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.04).
		program	Added support for RX660.
3.10	Mar 31,2023	1	Added support for RX26T.
		24, 25	Added RX26T in Section 2.4 Interrupt Vector.
		28, 29	2.8 Code Size, amended.
		62	Added RSKRX660 in Section 5.2 Running the Sample program.
		69	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.10).
		program	Added support for RX26T. Added idle interrupt for RX671. Added new demo projects.
3.20	May 29, 2023	1	Added support for RX23E-B.
		24	Added RX23E-B in Section 2.4 Interrupt Vector.
		28, 29	2.8 Code Size, amended.
		32, 60	Deleted the description of FIT configurator from "2.12 Adding the FIT Module to Your Project", "4. Pin Settings".
		69	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.20).
		program	Added support for RX23E-B. Deleted the description of FIT configurator.

Rev.	Date	Description	
		Page	Summary
3.30	Aug 18, 2023	55-58	Modified “Description” and “Special Notes” of the following API functions: R_RSPI_IntSptilerClear, R_RSPI_IntSprilerClear, R_RSPI_DisableSpti, R_RSPI_DisableRSPI.
		70	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.30).
		program	Modified comments of the following API functions: R_RSPI_IntSptilerClear, R_RSPI_IntSprilerClear, R_RSPI_DisableSpti, R_RSPI_DisableRSPI.
3.40	Oct 05, 2023	70	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.40).
		program	Implemented code to clear the SPRF flag at the start of SPI communication. Modified the order of disabling the error interrupt and canceling the error handler registration when disabling interrupts. Modified the order of disabling interrupts between SPTI and SPRI when disabling interrupts. Added include header to fix missing #include platform.h issue in "r_rspi_rx_private.h".
3.50	Jun 28, 2024	1	Added support for RX260, RX261.
		24	Added RX260, RX261 in Section 2.4 Interrupt Vector.
		28	2.8 Code Size, amended.
		71	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.50).
		program	Added support for RX260, RX261. Fixed the default value of RSPI_CFG_DUMMY_TXDATA in MDF file to match the value in file r_rspi_rx_config.h.
3.51	Mar 15, 2025	71	Changed Section 6.1 Operation Confirmation Environment (Ver. 3.51).
		program	Updated FIT Disclaimer and Copyright.

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.