# RX Family

## Clock Synchronous Control Module for EEPROM Access Firmware Integration Technology

### Introduction

This application note explains how to control and use R1EX25xxx and HN58X25xxx Series SPI serial EEPROM, manufactured by Renesas Electronics, with Renesas microcontrollers.

The control software accompanying this application note is upper-layer software that controls the serial EEPROM as a slave device.

Lower-layer software (clock synchronous single master control software) for controlling the SPI mode on the individual microcontroller, operating as a master device, is available separately; it can be obtained from the webpage below. Note that although the clock synchronous single master control software may support newer microcontrollers, there may be cases where the control software presented in this application note has not yet been updated to match. For information on the latest supported microcontrollers and matching control software releases, see the “Clock Synchronous Single Master Control Software (Lower-level layer of the software)” section of the following webpage:

SPI Serial EEPROM Driver
http://www.renesas.com/driver/spi_serial_eeprom

The control software uses Firmware Integration Technology (FIT). It is referred to as the EEPROM FIT module in the documentation of development tools with FIT support. Other similar function control modules using FIT are referred to as FIT modules or as “function name” FIT modules.

When using development tools that do not support FIT, the software code can be imported with the FIT functionality disabled.

### Target Device

Serial EEPROM Renesas Electronics R1EX25xxx Series SPI Serial EEPROM
RX Family microcontrollers

Microcontrollers on which operation has been confirmed:
RX111 Group, RX110 Group, RX113 Group, RX130 Group (RSPI)
RX230 Group, RX231 Group, RX23T Group, RX24T Group (RSPI)
RX64M Group, RX71M Group (RSPI, QSPI, SCI)
RX72T and RX72N Group (RSPI, SCI)

When applying the information in this application note to a microcontroller other than the above, modifications should be made as appropriate to match the specification of the microcontroller and careful evaluation performed.

### 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 “4.1 Confirmed Operation Environment".

## Contents

# 1. Overview

This software controls R1EX25xxx or HN58X25xxx Series SPI serial EEPROM, manufactured by Renesas Electronics, using a Renesas microcontroller.

A clock synchronous single master control software specific to the microcontroller model used (available separately) is required.

Table 1.1 lists the peripheral devices used and their applications, and figure 1.1 shows a usage example.

The functions of the module are described briefly below.

- Block type device driver using the Renesas microcontroller as the master device and the R1EX25xxx or HN58X25xxx Series SPI serial EEPROM as the slave device
- Control in SPI mode of target serial communication FIT module, using the microcontroller's built-in serial communication functionality (clock synchronous mode) (See 1.3.1 FIT Module–Related Application Notes.)
  - RSPI FIT module
  - QSPI FIT module
  - SCI FIT module
- Ability to control up to two SPI serial EEPROM devices
- Ability to make SPI serial EEPROM settings on a per-device basis
- Support for both big-endian and little-endian byte order (dependent on specified device)

**Table 1.1 Peripheral Devices Used and Their Uses**

| Peripheral Device | Use |
|---|---|
| Microcontroller's on-chip serial communication function (clock synchronous mode) | Communication with SPI slave device using serial communication functionality (clock synchronous mode): Single or multiple channels (required) |
| Port | For slave device selection control signals: A number of ports equal to the number of devices used are necessary (required). |

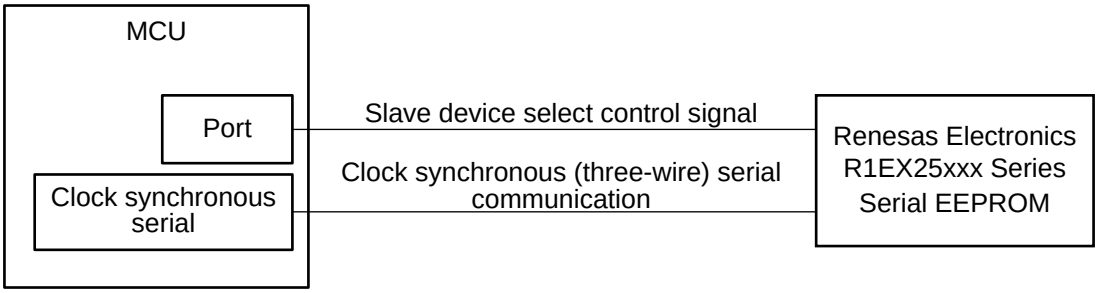



**Figure 1.1 Sample Configuration**

Note:

The RSPI module does not work properly with the big endian of the IAR compiler. In this case, do not use the RSPI module.

## 1.1 FIT Support of Serial EEPROM Control Software

The serial EEPROM control software can be combined with other FIT modules, allowing easy integration into your project.

The serial EEPROM control software can also be integrated into your project as an API. For information on adding the serial EEPROM control software, see 2.10 Adding the Driver to Your Project.

## 1.2 Overview of APIs

Table 1.2 lists the API functions of the serial EEPROM control software.

**Table 1.2 API Functions**

| Function Name | Description |
|---|---|
| R_EEPROM_SPI_Open() | Control software initialization processing |
| R_EEPROM_SPI_Close() | Control software end processing |
| R_EEPROM_SPI_Read_Status() | Status register read processing |
| R_EEPROM_SPI_Set_Write_Protect() | Write protect setting processing |
| R_EEPROM_SPI_Write_Di() | WRDI command processing |
| R_EEPROM_SPI_Read_Data()*1 | Data read processing |
| R_EEPROM_SPI_Write_Data_Page()*1 | Data write (single-page write) processing |
| R_EEPROM_SPI_Polling() | Polling processing |
| R_EEPROM_SPI_GetMemoryInfo() | Memory size acquisition processing |
| R_EEPROM_SPI_GetVersion() | Control software version information acquisition processing |
| R_EEPROM_SPI_Set_LogHdlAddress() | LONGQ FIT module handler address setting processing |
| R_EEPROM_SPI_Log() | Error log acquisition processing using LONGQ FIT module |
| R_EEPROM_SPI_1ms_Interval()*2 | Clock synchronous single master control software interval timer counter processing |

Notes: 1. To speed up data transfers, align the start address with a 4-byte boundary when specifying transmit and receive data storage buffer pointers. There is a limitation on the data size when using DMAC transfer or DTC transfer. Refer to the documentation of the clock synchronous single master control software for the microcontroller used regarding the allowable data size setting range.

2. This function must be called at 1 ms intervals, using a hardware or software timer, in order to implement timeout detection when using DMAC transfer or DTC transfer.

## 1.3 Related Application Notes

Application notes related to the serial EEPROM control software are listed below. Refer to them alongside this application note.

### 1.3.1 FIT Module–Related Application Notes

- RX Family Module Using Firmware Integration Technology (R01AN1827)
- RX Family QSPI Clock Synchronous Single Master Control Module Using Firmware Integration Technology (R01AN1940)
- RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
- RX Family DMAC Module Using Firmware Integration Technology (R01AN2063)
- RX Family DTC Module Using Firmware Integration Technology (R01AN1819)
- RX Family CMT Module Using Firmware Integration Technology (R01AN1856)
- RX Family GPIO Driver Module Using Firmware Integration Technology (R01AN1721)
- RX Family MPC Module Using Firmware Integration Technology (R01AN1724)
- RX Family LONGQ Module Using Firmware Integration Technology (R01AN1889)
- RX Family Memory Access Driver Interface Module Using Firmware Integration Technology(R01AN4548)

## 1.4 Hardware Settings

### 1.4.1 Hardware Configuration Example

Figure 1.2 is a connection diagram. The pin names differ according to the microcontroller and serial interface used. Refer to the listing of pins and functions in table 1.3 and assign pins on the specific microcontroller used.

To achieve high-speed operation, consider adding damping resistors or capacitors to improve the circuit matching of the various signal lines.

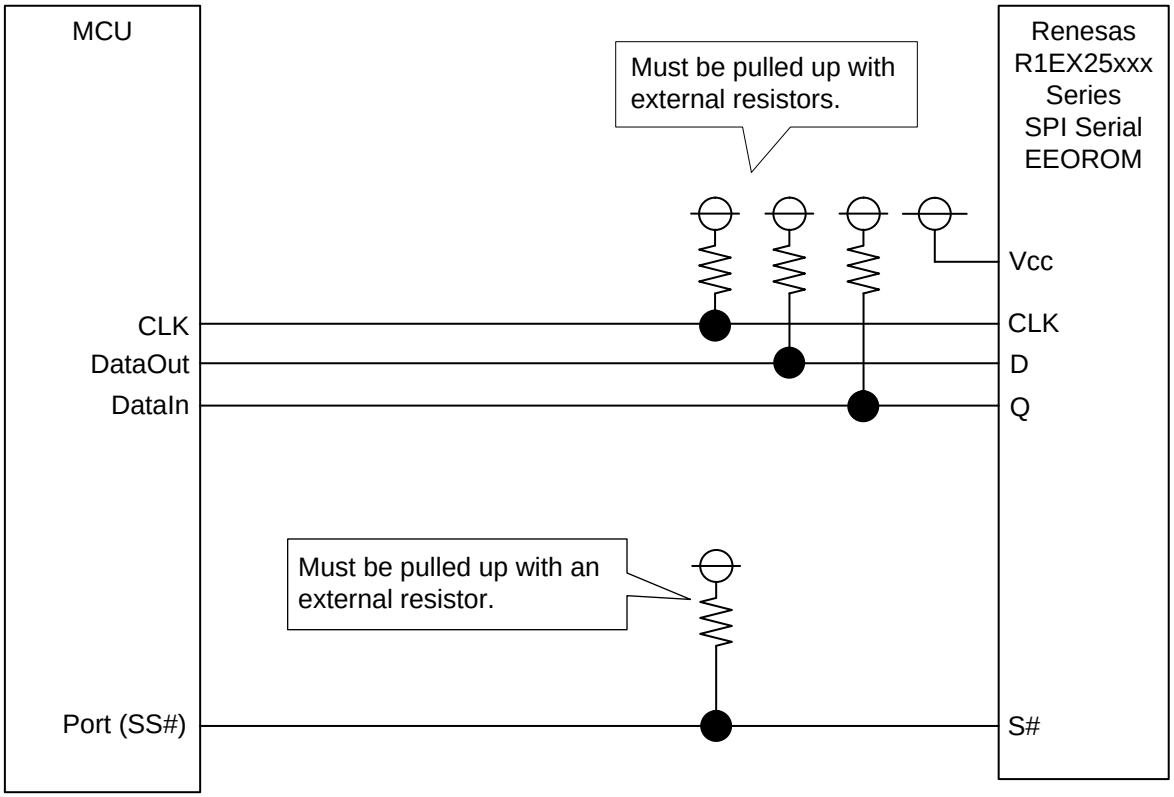



**Figure 1.2 Sample Wiring Diagram for a MCUI and a SPI Slave Device**

**Table 1.3 List of Pins Used**

| Pin Name | I/O | Description |
|---|---|---|
| CLK | Output | Clock output |
| DataOut | Output | Master data output |
| DataIn | Input | Master data input |
| Port (Port (SS#) of figure 1.2) | Output | Slave dev<br>ice select (SS#) output |

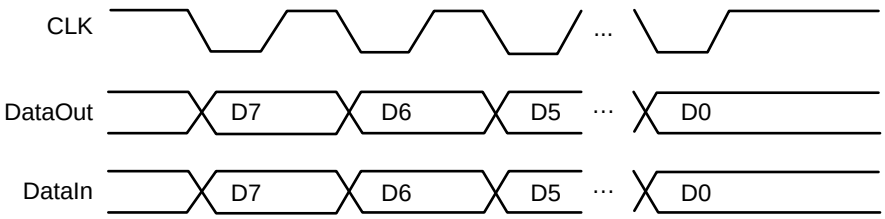
## 1.5 Software

### 1.5.1 Operation Overview

Utilizing the clock synchronous serial communication functionality of the MCU, clock synchronous single master control is implemented using the internal clock.

### 1.5.2 Control in SPI Mode

Control is performed in SPI mode 3 (CPOL = 1, CPHA = 1), as shown in figure 1.3.




- MCU → Slave device transmission: Transmission of transmit data is started on the falling edge of the transfer clock.
- Slave device → MCU reception: The receive data is taken in on the rising edge of the transfer clock.
- MSB-first mode transfer
- The level of the CLK pin is held high when no transfer processing is in progress.

**Figure 1.3 Timing of Controllable Slave Devices**

Refer to the User's Manual: Hardware of the microcontroller and the data sheet of the slave device to determine the usable serial clock frequencies.

### 1.5.3 Control of Chip Select Pin of Serial EEPROM

The Chip Select pin of the serial EEPROM is connected to a port of the microcontroller and controlled by general port output from the microcontroller.

Control is performed in the software to wait during the serial EEPROM Chip Select setup time, which is the time interval from the falling edge of the serial EEPROM's Chip Select (microcontroller port (SS#)) signal to the falling edge of the serial EEPROM's C (microcontroller CLK) signal.

In like manner, control is performed in the software to wait during the serial EEPROM Chip Select hold time, which is the time interval from the rising edge of the serial EEPROM's C (microcontroller CLK) signal to the rising edge of the serial EEPROM's Chip Select (microcontroller port (SS#)) signal.

In this module the wait intervals for the Chip Select setup time and Chip Select hold time are each approximately 1 µs.

### 1.5.4 Software Structure

Figure 1.4 shows the software structure.

Use the control software to create software for controlling slave devices.

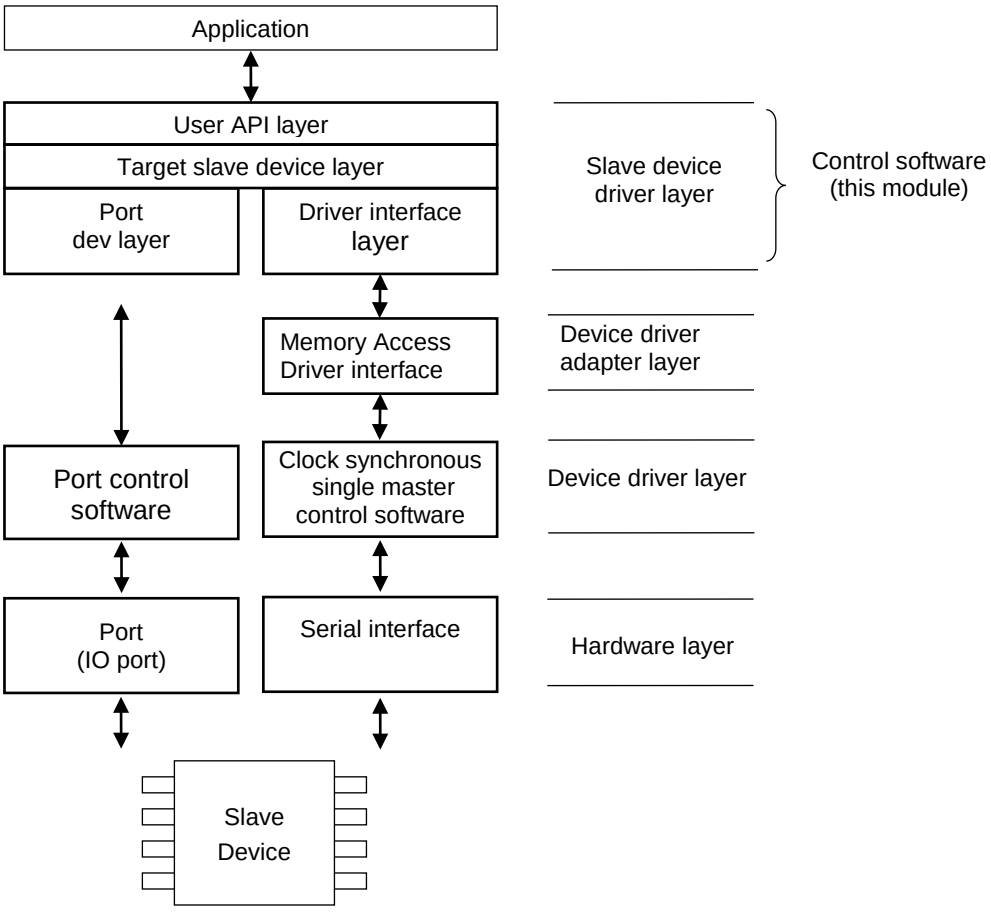



**Figure 1.4 Software Structure**

(a) User API layer (r_eeprom_spi.c)
The EEPROM control module, this portion of the software is not dependent on lower-layer device drivers.
(b) Sub-module layer (r_eeprom_spi_sub.c)
The EEPROM control sub-module, this portion of the software is not dependent on lower-layer device drivers.
(c) Driver interface (I/F) layer (r_eeprom_spi_drvif.c)
The common module for connecting to lower-layer device drivers.
A separate driver interface function is required to match the clock synchronous single master control module for each microcontroller model.
(d) Port dev layer (r_eeprom_spi_dev_port.c)
The control module for controlling the slave device select signal (SS#) with a microcontroller port.
The GPIO FIT module and MPC FIT module can be used.
(e) Application
Sample code for controlling R1EX25xxx Series serial EEPROM, manufactured by Renesas Electronics, is provided for reference.

### 1.5.5 Relationship Between Control Software and Clock Synchronous Single Master Control Software

The method whereby the control software and the clock synchronous single master control software are combined is described below.

Control of up to two slave devices, using up to two clock synchronous single master control modules, is supported. Register the clock synchronous single master control module (or modules) used as the driver interface function (or functions).

As shown below, it is possible to specify a separate driver for each device. For each device number, create processing code using the device driver API in the driver interface function that constitutes the driver interface layer.

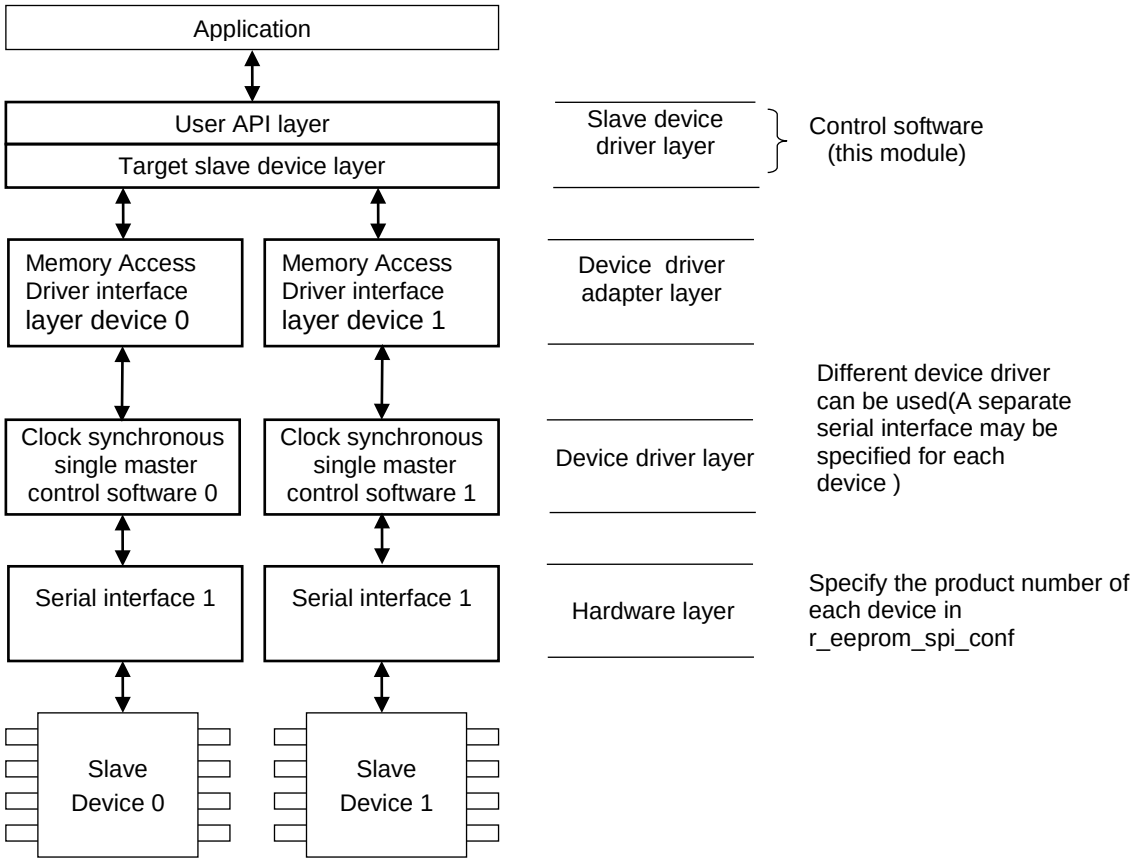



**Figure 1.5 Software Configuration with Clock Synchronous Single Master Control Modules**

### 1.5.6 Data Buffers and Transmit/Receive Data

The control software is a block type device driver that sets transmit and receive data pointers as arguments. The arrangement of data in the data buffer in RAM and the transmit and receive sequences are illustrated below. Regardless of the endian mode and the serial communication function, data is transmitted in the order in which it is arranged in the transmit data buffer, and it is written to the receive data buffer in the order in which it is received.

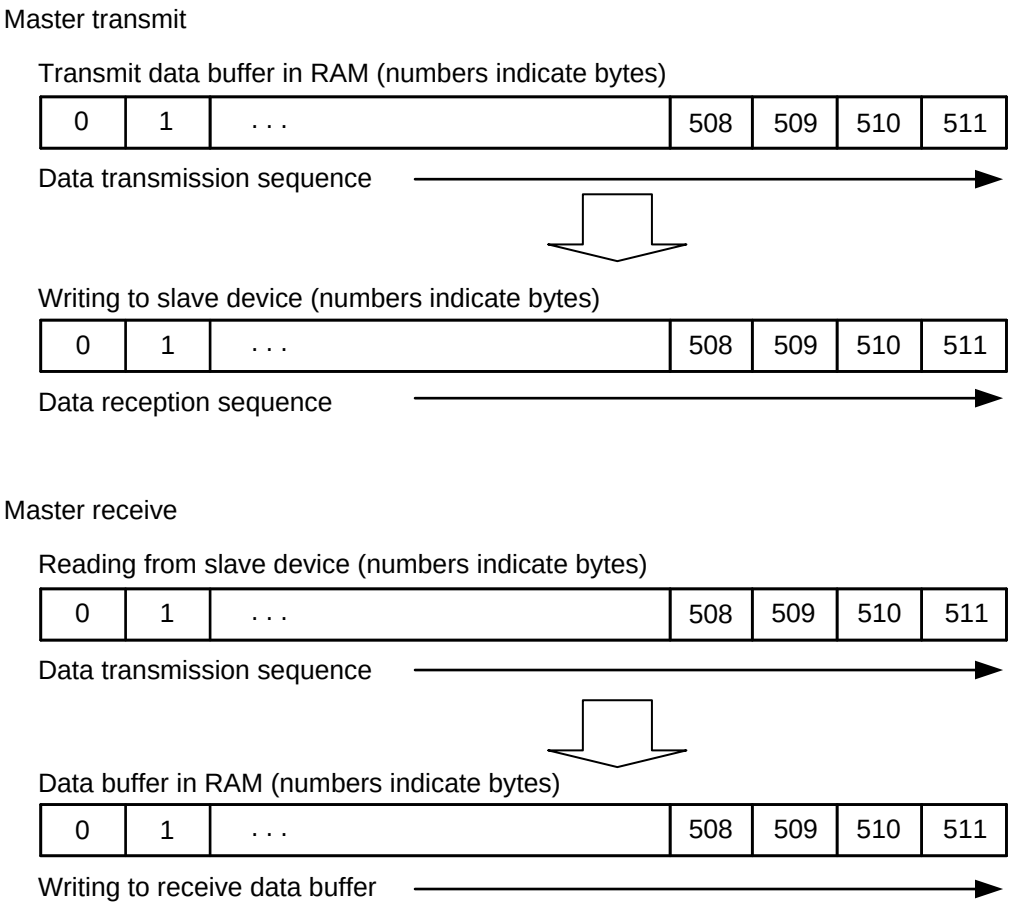



**Figure 1.6 Data Buffers and Transmit/Receive Data**

### 1.5.7 State Transition Diagram

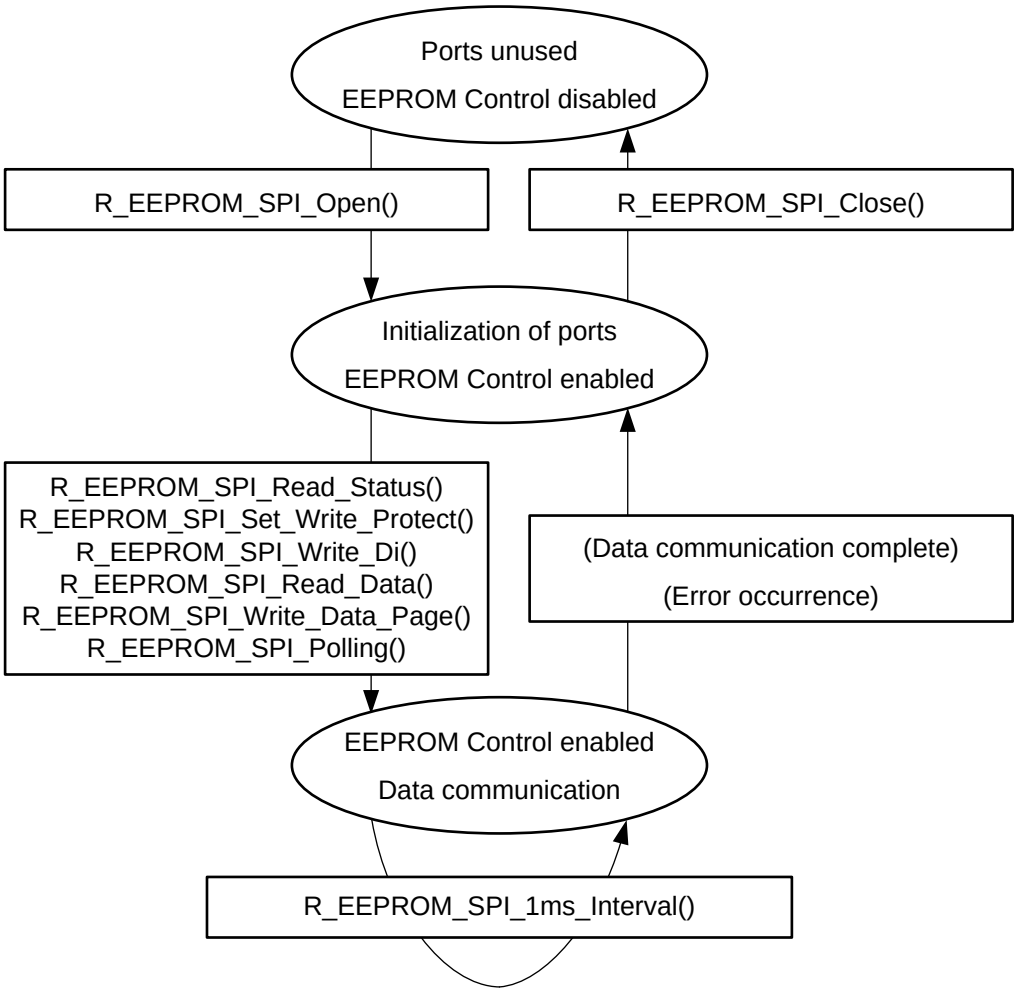



**Figure 1.7 State Transition Diagram**

# 2. API Information

The names of the APIs of the control software follow the Renesas API naming standard.

## 2.1 Hardware Requirements

The microcontroller used must support the following functionality. Note that separate clock synchronous single master control software is required.

- I/O port

## 2.2 Software Requirements

When used with FIT support enabled, the control software is dependent on the following packages.

- r_bsp (Rev.5.00 or higher)
- r_memdrv_rx
- r_rspi_rx (when using the RSPI FIT module for clock synchronous control)
- r_qspi_smstr_rx (when using the QSPI FIT module for clock synchronous single master control)
- r_sci_rx (when using the SCI FIT module for clock synchronous control)
- r_dmaca_rx (only when using the DMACA FIT module for DMAC transfers)
- r_dtc_rx (only when using the DTC FIT module for DTC transfers)
- r_cmt_rx (only when using DMAC transfer or DTC transfer and the compare match timer (CMT) FIT module)
  Another timer or a software timer can be used instead.
- r_gpio_rx (only when using the GPIO and MPC FIT modules to control the GPIO)
- r_mpc_rx (only when using the GPIO and MPC FIT modules to control the MPC)

## 2.3 Supported Toolchain

The operation of the control software has been confirmed with the toolchain listed in 4.1 Confirmed Operation Environment.

## 2.4 Header Files

All the API calls and interface definitions used are listed in r_eeprom_spi_if.h.

Configuration options for individual builds are selected in r_eeprom_spi_config.h and r_eeprom_spi_pin_config.h. The include statements should be in the following order. Note that it is not necessary to include r_eeprom_spi_pin_config.h.

```
#include "r_eeprom_spi_if.h"
#include "r_eeprom_spi_config.h"
```

## 2.5 Integer Types

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

## 2.6 Compile Settings

The configuration option settings for the control software are specified in r_eeprom_spi_config.h and r_eeprom_spi_pin_config.h.

The option names and setting values are described below.

| Configuration options in *r_eeprom_spi_config.h* | |
|---|---|
| #define EEPROM_SPI_CFG_WEL_CHK<br>Note: The default value is “enabled”. | Selects whether or not the WEL bit is checked after the WREN command is issued. |
| #define EEPROM_SPI_CFG_LONGQ_ENABLE<br>Note: The default value is “disabled”. | Selects whether or not error log acquisition processing is performed for debugging, when using the BSP environment of a FIT module.<br>When this option is set to “disabled”, code for the relevant processing is omitted.<br>When this option is set to “enabled”, code for the relevant processing is included.<br>To use this functionality, the LONGQ FIT module is also required.<br>In addition, enable #define xxx_LONGQ_ENABLE in the clock synchronous single master control software of the specified device. |
| #define EEPROM_SPI_CFG_USE_GPIO_MPC_FIT | Selects whether the GPIO FIT module or MPC FIT module is used to control the SS# pin.<br>When this option is set to “disabled”, neither the GPIO FIT module nor the MPC FIT module controls the SS# pin.<br>When this option is set to “enabled”, the GPIO FIT module or MPC FIT module controls the SS# pin.<br>To use this functionality, the GPIO FIT module or MPC FIT module is also required. |
| #define EEPROM_SPI_CFG_DEVx_INCLUDED<br>Note: The default value for device 0 is “enabled”. The “x” in DEVx represents the device number (x = 0 or 1). | This definition is related to device x.<br>This option must be set to “enabled” for device x. |
| #define EEPROM_SPI_CFG_DEVx_SIZE_002K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_004K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_008K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_016K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_032K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_064K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_128K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_256K<br>#define EEPROM_SPI_CFG_DEVx_SIZE_512K<br>Note: The default value for device 0 is “EEPROM_SPI_CFG_DEVx_SIZE_008K”. The “x” in DEVx represents the device number (x = 0 or 1). | Specify only one EEPROM size for device x. |

| Configuration options in *r_eeprom_spi_config.h* | |
|---|---|
| #define EEPROM_SPI_CS_DEVx_CFG_PORTNO<br>Note: The default value for device 0 is "'X'".<br>The "x" in DEVx represents the device number (x = 0 or 1). | Specifies the port number assigned to SS# for device x.<br>Enclose the setting value in single quotation marks (' ').<br>Configure Device x Port Number with 0 - 9, A - X. |
| #define EEPROM_SPI_CS_DEVx_CFG_BITNO<br>Note: The default value for device 0 is "'0'".<br>The "x" in DEVx represents the device number (x = 0 or 1). | Specifies the bit number assigned to SS# for device x.<br>Enclose the setting value in single quotation marks (' ').<br>Configure Device x Bit Number with 0 – 7. |

## 2.7 Arguments

The structure for the arguments of the API functions is shown below. This structure is listed in r_eeprom_spi_if.h, along with the prototype declarations of the API functions.

```
/* EEPROM information */
typedef struct
{
   uint32_t    addr;               /* Address to issue a command          */
   uint32_t    cnt;                /* Number of bytes to be read/written  */
   uint32_t    data_cnt;
/* Temporary counter or Number of bytes to be written in a page    */
   uint8_t     * p_data;           /* Data storage buffer pointer         */
} eeprom_info_t;                   /* 16 bytes                            */

/* EEPROM size information */
typedef struct
{
   uint32_t    mem_size;           /* Max memory size                     */
   uint32_t    wpag_size;          /* Write page size                     */
} eeprom_mem_info_t;               /* 8 bytes                             */
```

## 2.8 Code Size

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.6 Compile Settings.

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

Module Revision: r_eeprom_spi rev.3.10

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00

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

GCC for Renesas RX 8.3.0.202104

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

IAR C/C++ Compiler for Renesas RX version 4.20.3

(The default settings of the integrated development environment.)

Configuration Options: Default settings

Operating frequency: RX111, RX113 ICLK: 32 MHz, PCLKB: 32 MHz

RX231 ICLK: 54 MHz, PCLKB: 27 MHz

RX64M ICLK: 120MHz, PCLKA: 120MHz, PCLKB: 60MHz

RX71M ICLK: 240MHz, PCLKA: 120MHz, PCLKB: 60MHz

Operating voltage: 3.3V

Endian: Little endian

The clock synchronous single master control software: RSPI

Data transfer mode: Software

Confirmation conditions: r_eeprom_spi.c, r_eeprom_spi_sub.c, r_eeprom_spi_dev_port_iodefine.c, r_eeprom_spi_drvif.c

| ROM, RAM and Stack Code Sizes | | | | |
|---|---|---|---|---|
| Device | Category | Memory Used | | |
| | | Renesas Compiler | GCC | IAR Compiler |
| RX111 | ROM | **2,208 bytes** | **5,068 bytes** | **8,877 bytes** |
| | RAM | **4 bytes** | **4 bytes** | **37 bytes** |
| | STACK | **176 bytes** | **-** | **152 bytes** |
| RX113 | ROM | **2,205 bytes** | **5,088bytes** | **8,877 bytes** |
| | RAM | **4 bytes** | **4 bytes** | **37 bytes** |
| | STACK | **176 bytes** | **-** | **152 bytes** |
| RX231 | ROM | **2,207 bytes** | **5,104 bytes** | **8,875 bytes** |
| | RAM | **4 bytes** | **4 bytes** | **37 bytes** |
| | STACK | **176 bytes** | **-** | **152 bytes** |
| RX64M | ROM | **2,207 bytes** | **5,124 bytes** | **9,335 bytes** |
| | RAM | **4 bytes** | **4 bytes** | **37 bytes** |
| | STACK | **188 bytes** | **-** | **156 bytes** |
| RX71M | ROM | **2,207 bytes** | **5,104 bytes** | **9,331 bytes** |
| | RAM | **4 bytes** | **4 bytes** | **49 bytes** |
| | STACK | **188 bytes** | **-** | **156 bytes** |

## 2.9 Return Values

The API function return values are shown below. This enumerated type is listed in r_eeprom_spi_if.h, along with the prototype declarations of the API functions.

```
typedef enum e_eeprom_status
{
   EEPROM_SPI_SUCCESS_BUSY =  1,    /* Successful operation (EERPOM is busy) */
   EEPROM_SPI_SUCCESS      =  0,    /* Successful operation                  */
   EEPROM_SPI_ERR_PARAM    =  -1,   /* Parameter error                       */
   EEPROM_SPI_ERR_HARD     =  -2,   /* Hardware error                        */
   EEPROM_SPI_ERR_WP       =  -4,   /* Write-protection error                */
   EEPROM_SPI_ERR_OTHER    =  -7    /* Other error                           */
} eeprom_status_t;
```

## 2.10 Adding the Driver 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 (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) 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 FIT Configurator in e$^2$ studio
By using the FIT Configurator in e$^2$ studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) 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.

(4) 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.

(5) 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.

## 2.11 Using the Serial EEPROM Control Software in Other Than an FIT Module Environment

To use the serial EEPROM control software in an environment in which FIT modules such as r_bsp are not used, perform the following.

Comment out the line #include “platform.h” in #r_eeprom_spi_if.h.

Include the following header files in #r_eeprom_spi_if.h.

```
#include “iodefine.h”
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <machine.h>
```

Disable the option #define EEPROM_SPI_CFG_USE_FIT in #r_eeprom_spi_if.h.

Add the definition #define EEPROM_SPI_CFG_xxx (replacing xxx with the microcontroller name using all capital letters) to #r_eeprom_spi_if.h. For example, for the RX64M microcontroller use the string EEPROM_SPI_CFG_RX64M.

In # r_eeprom_spi_if.h add the enum definitions shown below. Also add the #define definitions shown below. Set the system clock (ICLK) value in BSP_ICLK_HZ. Note that it is possible that some of these definitions may duplicate other FIT module definitions. Insert the lines #ifndef SMSTR_WAIT and #define SMSTR_WAIT at the beginning of the definitions, and insert #endif as the last line.

```
#ifndef SMSTR_WAIT
#define SMSTR_WAIT
typedef enum
{
    BSP_DELAY_MICROSECS = 1000000,
    BSP_DELAY_MILLISECS = 1000,
    BSP_DELAY_SECS = 1
} bsp_delay_units_t;

#define BSP_ICLK_HZ  (120000000)    /* ICLK = 120MHz */
#endif      /* #ifndef SMSTR_WAIT */
```

## 2.12 Pin States

Table 2.1 lists the pin states after a power on reset and after execution of various API functions.

As shown in 1.5.2 Control in SPI Mode, this module supports SPI mode 3 (CPOL = 1, CPHA = 1). Regardless of the hardware configuration, **after a power on reset, control the GPIO from the user side and put the select pin into the high-output state to use this mode**.

Also, the slave device select pin is in the GPIO high-output state after R_EEPROM_SPI_Close() runs. Review the pin settings if necessary.

**Table 2.1 Pin States after Function Execution**

| Function Name | Slave Device Select Pin* |
|---|---|
| (After power on reset) | GPIO input state |
| Before R_EEPROM_SPI_Open() | GPIO high-output state<br>Set on user side |
| After R_EEPROM_SPI_Open() | GPIO high-output state<br>Set by this module |
| After R_EEPROM_SPI_Close() | GPIO high-output state<br>Set by this module |

Note: * Use an external resistor to pull up the slave device select pin. See 1.4.1 Hardware Configuration Example.

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

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

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

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

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

# 3. API Functions

## 3.1 R_EEPROM_SPI_Open()

This function is run first when using the APIs of the serial EEPROM control software.

### Format

```
eeprom_status_t R_EEPROM_SPI_Open(
  uint8_t devno
)
```

### Parameters

*devno*

Device number (0, 1)

### Return Values

```
EEPROM_SPI_SUCCESS      /* Successful operation */
EEPROM_SPI_ERR_PARAM    /* Parameter error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Initializes the slave device select pin of the device number specified by the argument devno. After initialization the pin is in the general output port high-output state.

Do not call this function when communication is in progress. Communication cannot be guaranteed if the function is called when communication is in progress.

### Example

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;

ret = R_EEPROM_SPI_Open(EEPROM_SPI_DEV0);
```

### Special Notes

After calling this user API function, it is recommended that R_EEPROM_SPI_Polling() be used to confirm that the EEPROM write cycle has completed. The next read or write processing will not be accepted while the EEPROM write cycle is in progress.

However, it is possible to access the EEPROM during the write cycle by, for example, issuing a system reset while the EEPROM write cycle is in progress and restarting EEPROM control from the beginning.

## 3.2 R_EEPROM_SPI_Close()

This function is used to close the serial EEPROM control software when it is in use.

### Format

```
eeprom_status_t R_EEPROM_SPI_Close(
  uint8_t devno
)
```

### Parameters

*devno*

Device number (0, 1)

### Return Values

```
EEPROM_SPI_SUCCESS      /* Successful operation */
EEPROM_SPI_ERR_PARAM    /* Parameter error */
EEPROM_SPI_ERR_OTHER    /* Other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Sets the slave device select pin of the device number specified by the argument devno to function as a general I/O port. After the function runs, the pin is in the general output port high-output state.

Do not call this function when communication is in progress. Communication cannot be guaranteed if the function is called when communication is in progress.

### Example

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;

ret = R_EEPROM_SPI_Close(EEPROM_SPI_DEV0);
```

### Special Notes

The state of the slave device select pin after this function is called is different from its state after a reset (general input port state). Review the pin settings if necessary.

Before calling this user API function, it is recommended that R_EEPROM_SPI_Polling() be used to confirm that the EEPROM write cycle has completed. This makes it possible to restart EEPROM control because the EEPROM has not transitioned to the write cycle.

## 3.3 R_EEPROM_SPI_Read_Status()

This function is used to read the status register.

### Format

```
eepr_status_t R_EEPROM_SPI_Read_Status
  uint8_t devno,
  uint8_t * p_status
)
```

### Parameters

*devno*

Device number (0, 1)

* *p_status*

Status register storage buffer

### Return Values

```
EEPROM_SPI_SUCCESS    /* Successful operation */
EEPROM_SPI_ERR_PARAM  /* Parameter error */
EEPROM_SPI_ERR_HARD   /* Hardware error */
EEPROM_SPI_ERR_OTHER  /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Reads the status register and stores the contents in p_status. Specify 1 byte as a read buffer. The following information is stored in p_status:

< 4 Kb or less >

| | |
|---|---|
| Bit 7 to 4:Reserved (All ”1”) | |
| Bit 3 to 2:BP1, BP0 | 00: No protection |
| | 01: Upper-quarter protection |
| | 10: Upper-half protection |
| | 11: Whole memory protection |
| Bit 1:WEL | 0: Write disabled |
| | 1: Write enabled |
| Bit 0:WIP | 1: During write operation |

< More than 4 Kb >

| | |
|---|---|
| Bit 7:SRWD | 0: Status register can be changed. |
| | 1: Status register cannot be changed. |
| Bit 6 to 4:Reserved (All ”0”) | |
| Bit3 to 2:BP1, BP0 | 00: No protection |
| | 01: Upper-quarter protection |
| | 10: Upper-half protection |
| | 11: Whole memory protection |
| Bit 1:WEL | 0: Write disabled |
| | 1: Write enabled |
| Bit 0:WIP | 1: During write operation |

**Example**

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
uint32_t          stat = 0;

ret = R_EEPROM_SPI_Read_Status(EEPROM_SPI_DEV0, &stat);
```

**Special Notes**

The clock synchronous single master control software resources are acquired at the start of the processing, and the resources are released and the end of the processing.

## 3.4 R_EEPROM_SPI_Set_Write_Protect()

This function is used to make write protect settings.

### Format

```
eeprom_status_t R_EEPROM_SPI_Set_Write_Protect(
  uint8_t devno,
  uint8_t wpsts
)
```

### Parameters

*devno*

Device number (0, 1)

*wpsts*

Write protect setting data

Specify one of the following as the setting:

```
EEPROM_SPI_WP_NONE            /* No protection */
EEPROM_SPI_WP_UPPER_QUART     /* Upper-quarter protection setting */
EEPROM_SPI_WP_UPPER_HALF      /* Upper-half protection setting */
EEPROM_SPI_WP_WHOLE_MEM       /* Whole memory protection setting */
```

### Return Values

```
EEPROM_SPI_SUCCESS    /* Successful operation */
EEPROM_SPI_ERR_PARAM  /* Parameter error */
EEPROM_SPI_ERR_HARD   /* Hardware error */
EEPROM_SPI_ERR_OTHER  /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Make write protect settings. SRWD is cleared to 0.

After this user API function finishes successfully, the EEPROM transitions to the write cycle. Do not fail to confirm that the write has finished with R_EEPROM_SPI_Polling(). If an attempt is made to perform the next read or write processing while a write cycle is in progress, the EEPROM will not accept that processing.

R_EEPROM_SPI_Polling() can be called at any time specified by the user. This makes it possible for the user application to perform other processing while a write cycle is in progress.

See figure 3.1 for details.

**Example**

```
 eeprom_status_t   ret   = EEPROM_SPI_SUCCESS;
 uint32_t          wait  = 0;
 uint32_t          loop_cnt = 0;

  Ret = R_EEPROM_SPI_Set_Write_Protect(EEPROM_SPI_DEV0, EEPROM_SPI_WP_NONE);
  if (EEPROM_SPI_SUCCESS > Ret)
  {
      /* Error */
  }

  loop_cnt = 50000;
  do
  {
   Ret = R_EEPROM_SPI_Polling(EEPROM_SPI_DEV0);
   if (EEPROM_SPI_SUCCESS == Ret)
   {
      /* EEPROM is ready. */
      break;
   }
   else if (EEPROM_SPI_SUCCESS_BUSY == Ret)
   {
      /* EEPROM is busy.
         User application can perform other processing while eeprom is busy.*/
      for (wait = 0; wait < 10; wait++)
      {
         /* Do nothing */
      }
   }
   else
   {
      /* Error */
   }
   loop_cnt--;
  }
  while (0 != loop_cnt);

  if (0 == loop_cnt)
  {
      /* Error */
  }
```

**Special Notes**

The clock synchronous single master control software resources are acquired at the start of the processing, and the resources are released and the end of the processing.

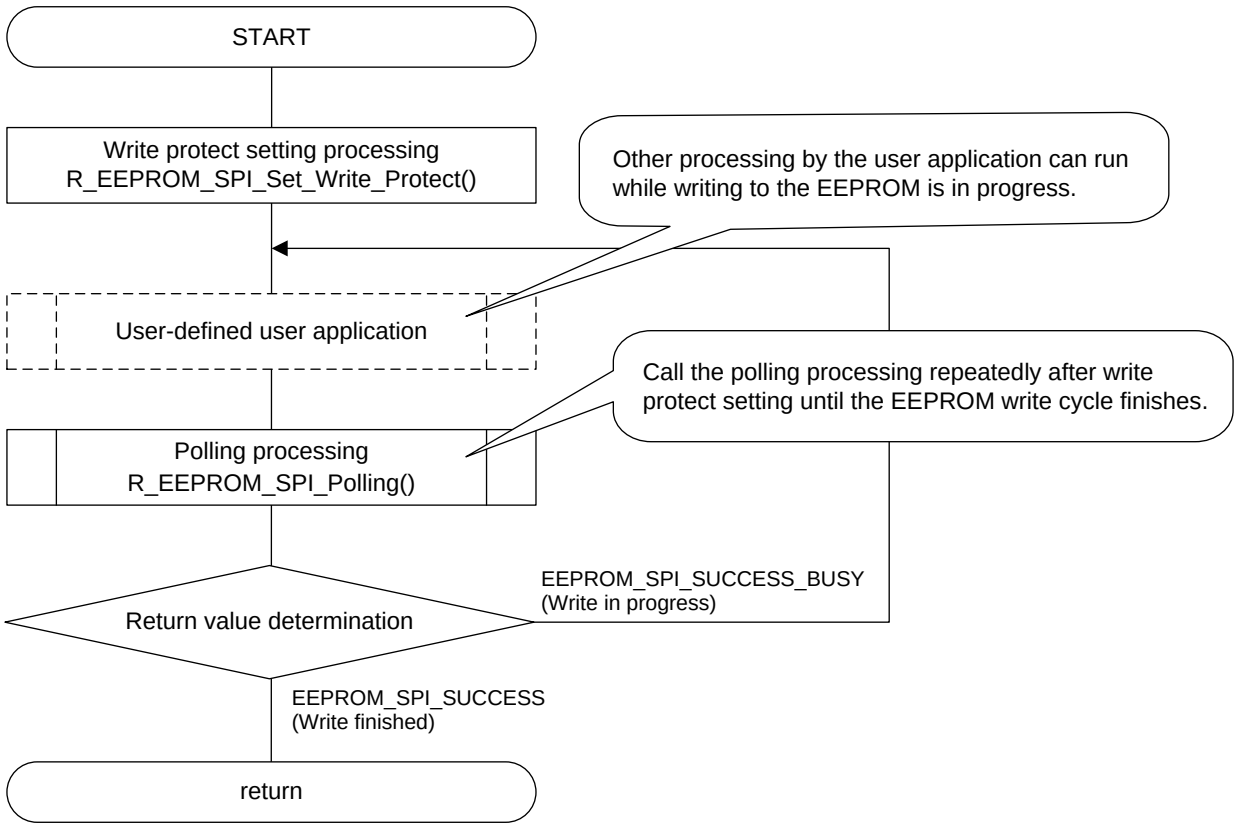



**Figure 3.1 R_EEPROM_SPI_Set_Write_Protect() Processing Example**

## 3.5 R_EEPROM_SPI_Write_Di()

This function is used to disable write operation.

### Format

```
eeprom_status_t R_EEPROM_SPI_Write_Di(
  uint8_t devno
)
```

### Parameters

*devno*

Device number (0, 1)

### Return Values

```
EEPROM_SPI_SUCCESS    /* Successful operation */
EEPROM_SPI_ERR_PARAM  /* Parameter error */
EEPROM_SPI_ERR_HARD   /* Hardware error */
EEPROM_SPI_ERR_OTHER  /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Transmits the WRDI command and clears the WEL bit in the status register.

### Example

```
eeprom_status_t     ret = EEPROM_SPI_SUCCESS;

ret = R_EEPROM_SPI_Write_Di(EEPROM_SPI_DEV0);
```

### Special Notes

The clock synchronous single master control software resources are acquired at the start of the processing, and the resources are released and the end of the processing.

## 3.6 R_EEPROM_SPI_Read_Data()

This function is used to read data from the EEPROM.

### Format

```
eeprom_status_t R_EEPROM_SPI_Read_Data(
  uint8_t devno,
  eeprom_info_t * p_eeprom_info
)
```

### Parameters

*devno*

Device number (0, 1)

* *p_eeprom_info*

EEPROM information structure. Use the structure address aligned with a 4-byte boundary.

addr

Specify the start address of the memory read.

cnt

Specify the read byte count. The allowable setting range is 1 to 4,294,967,295. A setting of 0 causes an error to be returned.

data_cnt

Read byte count (Used by the control software, so setting by the user is prohibited.)

*p_data

Specify the address of the read data storage buffer. Use a buffer address aligned with a 4-byte boundary when specifying DMAC transfer or DTC transfer.

### Return Values

```
EEPROM_SPI_SUCCESS    /* Successful operation */
EEPROM_SPI_ERR_PARAM  /* Parameter error */
EEPROM_SPI_ERR_HARD   /* Hardware error */
EEPROM_SPI_ERR_OTHER  /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Reads the specified number of bytes of data from the specified address in the EEPROM and stores the data in p_data.

The maximum read address is the EEPROM capacity – 1.

EEPROM_SPI_ERR_PARAM is returned if the total value of the read byte count, cnt, and specified address, addr, exceeds the maximum read address.

DMAC transfer or DTC transfer occurs when the transfer size conditions of the clock synchronous single master control software are matched. Otherwise, operation switches to software transfer.

**Example**

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
eeprom_info_t     Eep_Info_R;
uint32_t          buf2[EEPROM_SPI_DEV0_WPAG_SIZE/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Eep_Info_R.addr   = 0;
Eep_Info_R.cnt    = 32;
Eep_Info_R.p_data = (uint8_t *)&buf2[0];
ret = R_EEPROM_SPI_Read_Data(EEPROM_SPI_DEV0, &Eep_Info_R);
```

**Special Notes**

To speed up data transfers, align the start address with a 4-byte boundary when specifying data storage buffer pointers.

The clock synchronous single master control software resources are acquired at the start of the processing, and the resources are released and the end of the processing.

## 3.7 R_EEPROM_SPI_Write_Data_Page()

This function is used to write data to the EEPROM in single-page units.

### Format

```
eeprom_status_t R_EEPROM_SPI_Write_Data_Page(
  uint8_t devno,
  eeprom_info_t * p_eeprom_info
)
```

### Parameters

*devno*

Device number (0, 1)

* *p_eeprom_info*

EEPROM information structure. Use the structure address aligned with a 4-byte boundary.

addr

Specify the start address of the memory write.

cnt

Specify the write byte count. The allowable setting range is 1 to 4,294,967,295. A setting of 0 causes an error to be returned.

data_cnt

Write byte count (Used by the control software, so setting by the user is prohibited.)

*p_data

Specify the address of the write data storage buffer.

### Return Values

```
EEPROM_SPI_SUCCESS    /* Successful operation */
EEPROM_SPI_ERR_PARAM  /* Parameter error */
EEPROM_SPI_ERR_HARD   /* Hardware error */
EEPROM_SPI_ERR_WP     /* Write protect error */
EEPROM_SPI_ERR_OTHER  /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Writes the specified number of bytes of data (up to a maximum size of 1 page) in p_data to the EEPROM, starting from the specified address.

When writing a large volume of data, communication is divided into page units. This prevents a situation in which other processing is not possible while communication is in progress.

Writing to the EEPROM is only possible when write protect has been canceled.

It is not possible to write to a protected page. Attempting to do so returns the error EEPROM_SPI_ERR_WP.

The maximum write address is the EEPROM capacity – 1.

EEPROM_SPI_ERR_PARAM is returned if the total value of the write byte count, cnt, and specified address, addr, exceeds the maximum write address.

DMAC transfer or DTC transfer occurs when the transfer size conditions of the clock synchronous single master control software are matched. Otherwise, operation switches to software transfer.

When a byte count exceeding 1 page is specified, the remaining byte count and next address information remain in the EEPROM information stricture (p_eeprom_info) after processing of a single page write finishes. It is possible to write the remaining bytes by specifying p_eeprom_info unmodified in this API function again.

After this user API function finishes successfully, the EEPROM transitions to the write cycle. Do not fail to confirm that the write has finished with R_EEPROM_SPI_Polling(). If an attempt is made to perform the next read or write processing while a write cycle is in progress, the EEPROM will not accept that processing.

R_EEPROM_SPI_Polling() can be called at any time specified by the user. This makes it possible for the user application to perform other processing while a write cycle is in progress.

See figure 3.2 for details.

**Example**

```
 eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
 eeprom_info_t     Eep_Info_W;
 uint32_t          buf1[EEPROM_SPI_DEV0_WPAG_SIZE/sizeof(uint32_t)];
                                 /* the buffer boundary (4-byte unit) */
 uint32_t          wait  = 0;
 uint32_t          loop_cnt = 0;

 Eep_Info_W.addr   = 0;
 Eep_Info_W.cnt    = 128;
 Eep_Info_W.p_data = (uint8_t *)&buf1[0];

  do
  {
   Ret = R_EEPROM_SPI_Write_Data_Page(gDevNo, &Eep_Info_W);
   if (EEPROM_SPI_SUCCESS > Ret)
   {
      /* Error */
   }
   loop_cnt = 50000;
   do
   {
      Ret = R_EEPROM_SPI_Polling(gDevNo);
      if (EEPROM_SPI_SUCCESS == Ret)
      {
         /* EEPROM is ready. */
         break;
      }
      else if (EEPROM_SPI_SUCCESS_BUSY == Ret)
      {
      /* EEPROM is busy.
         User application can perform other processing while eeprom is busy.*/
         for (wait = 0; wait < 10; wait++)
         {
            /* Do nothing */
         }
      }
      else
      {
         /* Error */
      }
      loop_cnt--;
   }
   while (0 != loop_cnt);
  }
  while (0 != Eep_Info_W.cnt);

  if (0 == loop_cnt)
  {
      /* Error */
  }
```

**Special Notes**

To speed up data transfers, align the start address with a 4-byte boundary when specifying data storage buffer pointers.

The clock synchronous single master control software resources are acquired at the start of the processing, and the resources are released and the end of the processing.

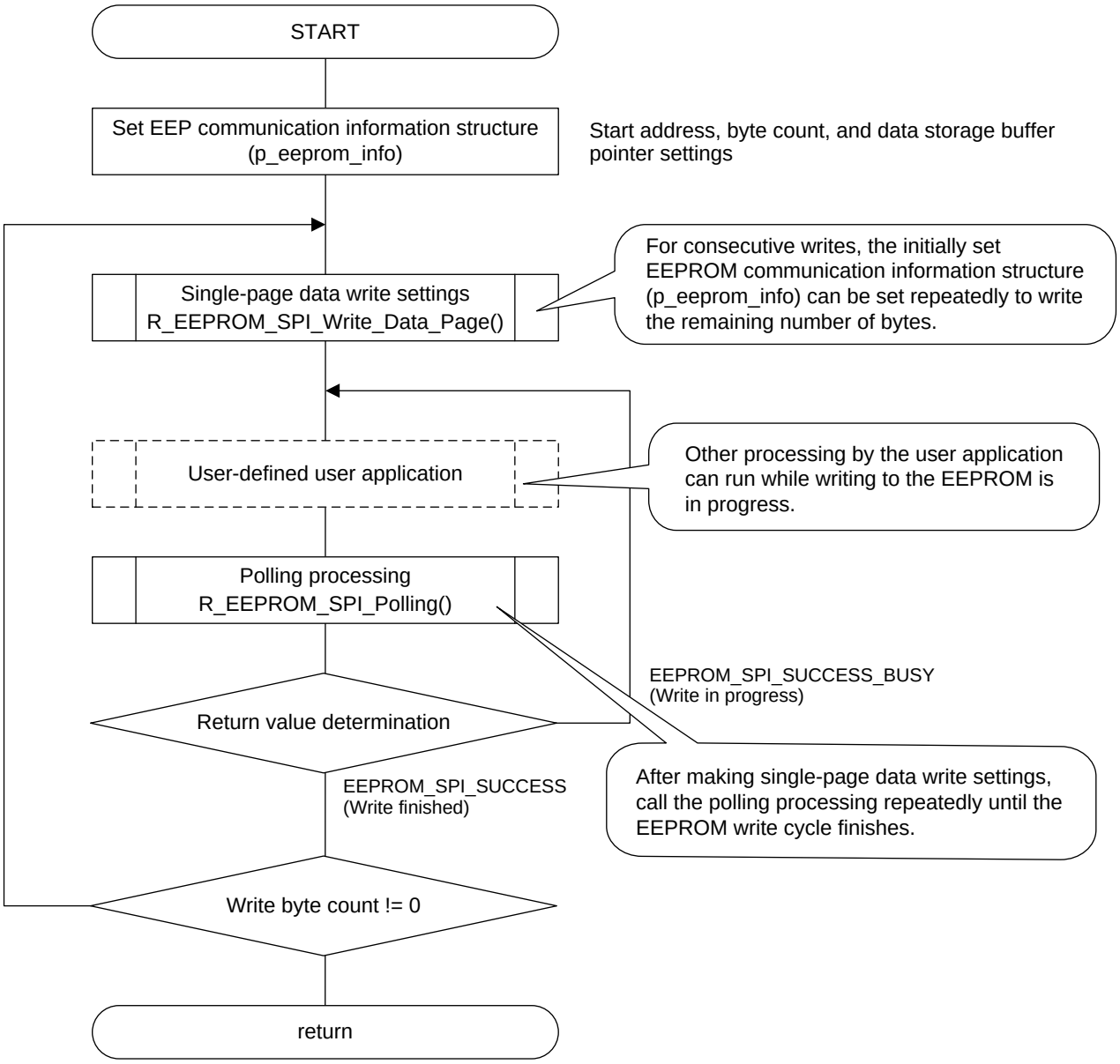



**Figure 3.2 R_EEPROM_SPI_Write_Data_Page() Processing Example**

## 3.8 R_EEPROM_SPI_Polling()

This function is used to perform polling to determine if the write operation has finished.

### Format

```
eeprom_status_t R_EEPROM_SPI_Polling(
  uint8_t devno
)
```

### Parameters

*devno*

Device number (0, 1)

### Return Values

```
EEPROM_SPI_SUCCESS       /* Normal end, and write finished */
EEPROM_SPI_SUCCESS_BUSY  /* Normal end, and write in progress */
EEPROM_SPI_ERR_PARAM     /* Parameter error */
EEPROM_SPI_ERR_HARD      /* Hardware error */
EEPROM_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                            master control software resources, or other error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Determines whether or not the write cycle has finished.

### Example

Refer to figure 3.1 or figure 3.2.

### Special Notes

R_EEPROM_SPI_Polling() can be called at any time specified by the user. This makes it possible for the user application to perform other processing while a write cycle is in progress.

The clock synchronous single master control software resources are acquired at the start of the processing, and the resources are released and the end of the processing.

## 3.9 R_EEPROM_SPI_GetMemoryInfo()

This function is used to fetch the serial EEPROM size information.

### Format

```
eeprom_status_t R_EEPROM_SPI_GetMemoryInfo(
  uint8_t devno,
  eeprom_mem_info_t * p_eeprom_mem_info
)
```

### Parameters

*devno*

Device number (0, 1)

* *p_eeprom_mem_info*

EEPROM size information structure. Use the structure address aligned with a 4-byte boundary.

mem_size

Maximum memory size

wpag_size

Page size

### Return Values

```
EEPROM_SPI_SUCCESS      /* Successful operation */
EEPROM_SPI_ERR_PARAM    /* Parameter error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Fetches EEPROM size information for the device number specified by the argument devno.

### Example

```
eeprom_status_t     ret = EEPROM_SPI_SUCCESS;
eeprom_mem_info_t   Eep_MemInfo;

ret = R_EEPROM_SPI_GetMemoryInfo(EEPROM_SPI_DEV0, &Eep_MemInfo);
```

### Special Notes

None

## 3.10 R_EEPROM_SPI_GetVersion()

This function is used to fetch the serial EEPROM version information.

### Format

```
uint32_t R_EEPROM_SPI_GetVersion(void)
```

### Parameters

None

### Return Values

*Version number*

*Upper 2 bytes: major version, lower 2 bytes: minor version*

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Returns the version information.

### Example

```
uint32_t version;
version = R_EEPROM_SPI_GetVersion();
```

### Special Notes

None

## 3.11 R_EEPROM_SPI_Set_LogHdlAddress()

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

### Format

```
eeprom_status_t R_EEPROM_SPI_Set_LogHdlAddress(
  uint32_t user_long_que
)
```

### Parameters

*user_long_que*

Specify the handler address of the LONGQ FIT module.

### Return Values

```
EEPROM_SPI_SUCCESS      /* Successful operation */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Sets the handler address of the LONGQ FIT module in the serial EEPROM control software and the clock synchronous single master control software used by the specified device.

This is preparatory processing to enable fetching of error logs using the LONGQ FIT module. Run this function before calling R_EEPROM_SPI_Open().

### Example

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

eeprom_status_t ret = EEPROM_SPI_SUCCESS;
uint32_t          MtlLogTbl[ERR_LOG_SIZE];
longq_err_t       ret_longq = LONGQ_SUCCESS;
longq_hdl_t       p_user_long_que;
uint32_t          long_que_hndl_address = 0;

/* Open LONGQ module. */
ret_longq = R_LONGQ_Open(&MtlLogTbl[0],
                         ERR_LOG_SIZE,
                         USER_LONGQ_IGN_OVERFLOW,
                         &p_user_long_que
);

long_que_hndl_address = (uint32_t)p_user_long_que;
R_EEPROM_SPI_Set_LogHdlAddress(long_que_hndl_address);
```

**Special Notes**

Add the LONGQ FIT module, which is available separately, to your project.

Enable the option #define EEPROM_SPI_CFG_LONGQ_ENABLE in r_eeprom_spi_config.h. Also, enable #define xxx_LONGQ_ENABLE in the clock synchronous single master control software used by the specified device.

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

## 3.12 R_EEPROM_SPI_Log()

This function fetches the error log. When an error occurs, call this function immediately before user processing ends.

### Format

```
uint32_t R_EEPROM_SPI_Log(
  uint32_t flg,
  uint32_t fid,
  uint32_t line
)
```

### Parameters

*flg*

Set this to 0x00000001 (fixed value).

*fid*

Set this to 0x0000003f (fixed value).

*line*

Set this to 0x0001ffff (fixed value).

### Return Values

```
0                   /* Successful operation */
1                   /* Error */
```

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

This function fetches the error log. When an error occurs, call this function immediately before user processing ends.

**Example**

```
#define  USER_DRIVER_ID    (0x00000001)
#define  USER_LOG_MAX      (0x0000003f)
#define  USER_LOG_ADR_MAX  (0x00001fff)

eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
eeprom_info_t     Eep_Info_W;
uint32_t          buf1[EEPROM_SPI_DEV0_WPAG_SIZE/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Eep_Info_W.addr   = 0;
Eep_Info_W.cnt    = 32;
Eep_Info_W.p_data = (uint8_t *)&buf1[0];
ret = R_EEPROM_SPI_Write_Data_Page(EEPROM_SPI_DEV0, &Eep_Info_W);
if (EEPROM_SPI_SUCCESS != ret)
{
    /* Set last error log to buffer. */
    R_EEPROM_SPI_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
    R_EEPROM_SPI_Close(EEPROM_SPI_DEV0);
}
```

**Special Notes**

Incorporate the LONGQ FIT module separately.

Enable the option #define EEPROM_SPI_CFG_LONGQ_ENABLE in r_eeprom_spi_config.h. Also, enable #define xxx_LONGQ_ENABLE in the clock synchronous single master control software used by the specified device.

## 3.13 R_EEPROM_SPI_1ms_Interval()

This function calls the interval timer counter function of the clock synchronous single master control software. When using the DMAC or DTC, use a timer to call this function at 1 ms intervals.

### Format

```
void R_EEPROM_SPI_1ms_Interval(void)
```

### Parameters

None

### Return Values

None

### Properties

Prototype declarations are contained in r_eeprom_spi_if.h.

### Description

Increments the internal timer counter of the clock synchronous single master control software while waiting for the DMAC transfer or DTC transfer to finish.

### Example

```
void cmt_callback (void * pdata)
{
    uint32_t channel;

    channel = (uint32_t)pdata;

    if (channel == gs_cmt_channel)
    {
        R_EEPROM_SPI_1ms_Interval();
    }
}
```

### Special Notes

User a timer or the like to call this function at 1 ms intervals.

In the example above, this function is called by a callback function that runs at 1 ms intervals.

# 4. Appendices

## 4.1 Confirmed Operation Environment

This section describes confirmed operation environment for the EEPROM FIT module.

**Table 4.1 Confirmed Operation Environment (Rev.3.00)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics<br>e$^2$ studio V7.3.0 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| Endian | Big endian/little endian |
| Revision of the module | Rev.3.00 |
| Board used | Renesas Starter Kit for RX111 (product No.: R0K505111xxxxxx)<br>Renesas Starter Kit for RX113 (product No.: R0K505113xxxxxx)<br>Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx)<br>Renesas Starter Kit + for RX64M (product No.: R0K50564Mxxxxxx)<br>Renesas Starter Kit for RX71M (product No.: R0K50571Mxxxxxx) |

**Table 4.2 Confirmed Operation Environment (Rev.3.01)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e2 studio V7.3.0<br>IAR Embedded Workbench for Renesas RX 4.10.01 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 4.08.04.201803<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.10.01<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.3.01 |
| Board used | Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx) |

**Table 4.3 Confirmed Operation Environment (Rev.3.02)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio 2020-07<br>IAR Embedded Workbench for Renesas RX 4.14.01 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V.3.02.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.03.00.202002<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.14.01<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.3.02 |
| Board used | Renesas Starter Kit+ for RX72N (product number.RTK5572Nxxxxxxxxxx) |

**Table 4.4 Confirmed Operation Environment (Rev.3.10)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio 2022-04<br>IAR Embedded Workbench for Renesas RX 4.20.03 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V.3.04.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.03.00.202104<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.20.03<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.3.10 |
| Board used | Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx) |

## 5. Reference Documents

Data sheet

The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools

RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)

The latest version can be downloaded from the Renesas Electronics website.

## Related Technical Updates

Not applicable technical update for this module.

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 2.31 | Dec 26, 2014 | — | First edition issued |
| 2.32 | Jun 15, 2015 | 1 | Added RX230 and RX231 in Target Device. |
| | | 5 | Changed “r_eeprom_spi_target_dev_port.c” to “r_eeprom_spi_dev_port.c”. at Table 1.4 in 1.2.2 Operating Environment and Memory Sizes. |
| | | 6 | Changed “r_eeprom_spi_target_dev_port.c” to “r_eeprom_spi_dev_port.c”. at Table 1.6 in 1.2.2 Operating Environment and Memory Sizes.<br>Added “r_eeprom_spi_drvif_dev0t.c” at Table 1.6 in 1.2.2 Operating Environment and Memory Sizes. |
| | | 7 | Changed “r_eeprom_spi_target_dev_port.c” to “r_eeprom_spi_dev_port.c”. at Table 1.8 in 1.2.2 Operating Environment and Memory Sizes.<br>Added “r_eeprom_spi_drvif_dev0t.c” at Table 1.8 in 1.2.2 Operating Environment and Memory Sizes. |
| | | 8 | Changed “r_eeprom_spi_target_dev_port.c” to “r_eeprom_spi_dev_port.c”. at Table 1.10 in 1.2.2 Operating Environment and Memory Sizes.<br>Added “r_eeprom_spi_drvif_dev0t.c” at Table 1.10 in 1.2.2 Operating Environment and Memory Sizes. |
| | | 9 | Added (5) RX231 at Table 1.10 in 1.2.2 Operating Environment and Memory Sizes. |
| | | 10 | Removed “RX64M Group RX Driver Package User’s Manual (R01AN2144EJ)”. |
| | | 12 | Changed “S#” to “Chip Select”. |
| | | 13 | Changed “Target microcontroller dev layer” to “Port Dev layer” in Figure 1.4 Software Structure. |
| | | 18 | Added “#define EEPROM_SPI_CFG_DEVx_DRVIF_CH_NO”. at Configuration options in r_eeprom_spi_config.h in 2.6 Compile Setting. |
| | | 33 | Added “Use the structure address aligned with a 4-byte boundary.” at Parameter *p_eeprom_info in 3.6 R_EEPROM_SPI_Read_Data(). |
| | | 35 | Added “Use the structure address aligned with a 4-byte boundary.” at Parameter *p_eeprom_info in 3.7 R_EEPROM_SPI_Write_Data_Page(). |
| | | 40 | Added “Use the structure address aligned with a 4-byte boundary.” at Parameter *p_eeprom_info in 3.9 R_EEPROM_SPI_GetMemoryInfo(). |
| 2.33 | Dec 29, 2015 | 1 | In Target Device.<br>Deleted “Device on which operation has been confirmed”.<br>Added “RX Family microcontrollers”.<br>Added “RX130”, “RX23T” and “RX24T”. |
| | | 22 | Updated contents in 2.9 Adding the Driver to Your Project. |
| | | — | Deleted “Using” of “Using FIT”. |
| 2.34 | Jul 31, 2017 | 17 | Deleted r_cgc_rx in 2.2 Software Requirements. |

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 3.00 | Feb 20.2019 | 1 | Change “SCIFA” to “SCI” in Target Device.<br>Added RX72T in Target Device.<br>Add “Using” in title of R01AN1685EJ In FIT Related Documents. |
| | | 3 | Change “SCIFA” to “SCI” in Overview. |
| | | 5-9 | Updated contents in 1.2.2 Operating Environment and Memory Sizes. |
| | | 10 | In 1.3.1 FIT Module–Related Application Notes<br>Deleted R01AN1914EJ.<br>Deleted R01AN2280EJ.<br>Added R01AN1827EJ.<br>Added R01AN1815EJ.<br>Added R01AN4548EJ.<br>Update R01AN1940EJ.<br>Update R01AN2280EJ.<br>Update R01AN2063EJ.<br>Update R01AN1819EJ.<br>Update R01AN1856EU.<br>Update R01AN1721EU.<br>Update R01AN1724EU.<br>Update R01AN1889EU. |
| | | 11 | Change table number from 1.9 to 1.13. |
| | | 13 | Update contents in 1.5.4 Software Structure. |
| | | 14 | Update contents in 1.5.5 Relationship Between Control Software and Clock Synchronous. |
| | | 17 | 2.2 Software Requirements<br>Added r_memdrv_rx.<br>Changed r_rspi_smstr_rx to r_rspi_rx.<br>Changed r_scifa_smatr_rx to r_sci_rx. |
| | | 18 | 2.6 Compile Settings<br>Deleted Marco<br>EEPROM_SPI_CFG_DEVx_DRVIF_CH_NO.<br>EEPROM_SPI_CFG_DEVx_MODE.<br>EEPROM_SPI_CFG_DEVx_DMAC_CH_NO_Tx.<br>EEPROM_SPI_CFG_DEVx_DMAC_CH_NO_Tx.<br>EEPROM_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Tx.<br>EEPROM_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Rx.<br>EEPROM_SPI_CFG_DEVx_BR.<br>EEPROM_SPI_CFG_DEVx_BR_WRITE_DATA.<br>EEPROM_SPI_CFG_DEVx_BR_READ_DATA. |
| | | 21 | Update contents in 2.9 Adding the Driver to Your Project |
| 3.01 | May 20.2019 | - | Update the following compilers<br>GCC for Renesas RX<br>IAR C/C++ Compiler for Renesas RX |
| | | 1 | Deleted Related Documents. |
| | | 1 | Added Target Compilers. |
| | | 3 | Added restrictions of IAR compiler. |
| | | 5 | Changed 1.2 Overview and Memory Size of APIs to 1.2 Overview of APIs.<br>1.2.2 Operating Environment and Memory Sizes, deleted. |

| Rev. | Date | Description | |
|---|---|---|---|
| | | **Page** | **Summary** |
| 3.01 | May 20.2019 | 12 | Added revision of dependent r_bsp module in 2.2 Software Requirements. |
| | | 16-17 | 2.8 Code Size, added. |
| | | 21 | 2.13 “for”, “while” and “do while” statements, added. |
| | | 44 | 4.1 Confirmed Operation Environment, added. |
| 3.02 | Dec 10.2020 | 1 | Added “RX72N”. |
| | | 17 | 2.8 Code Size, amended. |
| | | 19 | Changed Section 2.10 Adding the Driver to Your Project |
| | | 23～44 | Deleted "Reentrancy" item on the API description page. |
| | | 46 | Added Table 4.3 Confirmed Operation Environment (Rev.3.02). |
| | | 47 | Changed Section 5 Reference Documents. |
| | | Program | EEPROM_SPI FIT module fixed due to software failure.<br><br>Description:<br>A warning and linkage errors arise during building when using GPIO module firmware integration technology and MPC module firmware integration technology.<br><br>Conditions:<br>1. Use the integrated development environment CS+.<br>2. EEPROM FIT module general-purpose I/O port control is performed by both of the following FIT modules.<br>GPIO module Firmware Integration Technology<br>MPC module Firmware Integration Technology<br><br>Corrective action:<br>Please use EEPROM_SPI FIT module Rev3.02.<br><br>Corresponding tool news number: R20TS0609 |
| | | Program | EEPROM_SPI FIT module fixed due to software failure.<br><br>Description:<br>On RX72M/RX72N/RX66N, if the device port of r_eeprom_spi_pin_config.h is set to "H", "K", "M", "N", or "Q", a build error will occur.<br><br>Conditions:<br>Set EEPROM_SPI_CS_DEV0_CFG_PORTNO or EEPROM_SPI_CS_DEV1_CFG_PORTNO to one of "H", "K", "M", "N", and "Q" to build.<br><br>Corrective action:<br>Please use EEPROM_SPI FIT module Rev3.02. |

| Rev. | Date | Description | |
|---|---|---|---|
| | | **Page** | **Summary** |
| 3.10 | Jun 30.2022 | 15 | 2.6 Compile Settings<br>Added new macros<br>#define EEPROM_SPI_CS_DEVx_CFG_PORTNO<br>#define EEPROM_SPI_CS_DEVx_CFG_BITNO |
| | | 17 | 2.8 Updated FIT module version, and compilers' version |
| | | 46 | 4.1 Confirmed Operation Environment:<br>Added Table for Rev.3.10. |
| | | Program | Added new macros to specify the ports used for SS#<br>Fixed issues of wrong conditional expression in the if statement.<br>Set PORTX as the default port assigned to SS#. |

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

# Notice

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

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

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

## Corporate Headquarters

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

## Trademarks

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

## Contact information

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