RX Family

Clock Synchronous Control Module for Serial NOR Flash Memory Access

Introduction

This application note explains how to control and use serial flash memory with microcontrollers manufactured by Renesas Electronics. Refer to “Target Devices” below for a list of the supported serial flash memory products.

The control software accompanying this application note is upper-layer software that controls the serial flash memory 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:

Serial Flash Memory Driver
http://www.renesas.com/driver/spi_serial_flash

The control software uses Firmware Integration Technology (FIT). It is referred to as the serial flash memory 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 Devices

Device on which operation has been confirmed:

Serial NOR flash memory
Macronix International Co., Ltd.
- MX25/66L family serial NOR flash memory 32Mbit - 1Gbit
- MX25R family serial NOR flash memory 64Mbit

Dialog Semiconductor Plc.
- AT25QF family serial NOR flash memory 64Mbit

RX Family microcontrollers

Microcontrollers on which operation has been confirmed:

RX111, RX110, RX113 and RX130 Group (RSPI)
RX230, RX231, RX23T and RX24T Group (RSPI)
RX64M and RX71M Group (RSPI, QSPI, SCI)
RX72T and RX72N Group (RSPI, SCI)
RX671 Group (QSPIX, RSCI)

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.

The following abbreviations are used in this application note:

- Single-SPI (communication in single-SPI mode)
- Dual-SPI (communication in dual-SPI mode)
- Quad SPI (communication in quad-SPI mode)

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 “5.1 Confirmed Operation Environment".

FIT 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 controls serial flash memory, 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 serial flash memory 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
  - QSPIX FIT module
  - RSCI FIT module
- Ability to control up to two serial flash memory devices
- Ability to make serial flash memory settings on a per-device basis
- Support for both big-endian and little-endian byte order

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).

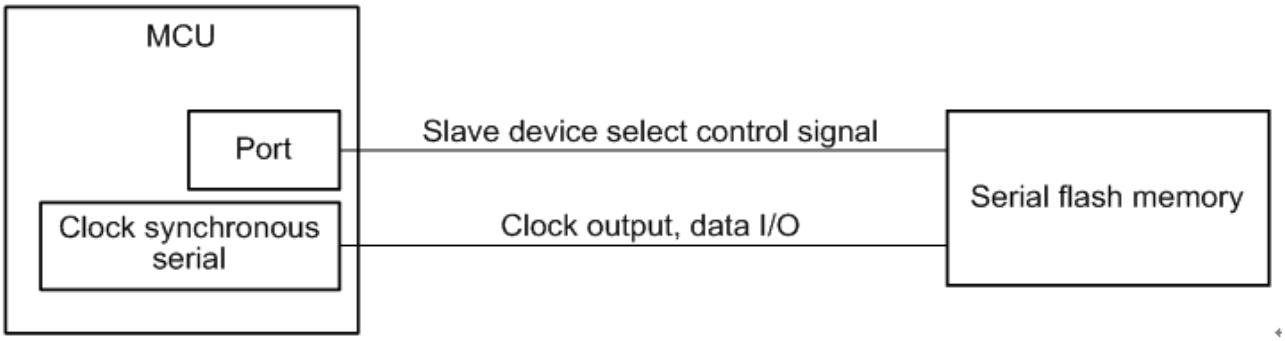


Figure 1.1 Sample Configuration

1.1 FIT Support of Serial flash Memory Control Software

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

The serial flash memory control software can also be integrated into your project as an API. For information on adding the serial flash memory 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 flash memory control software.

Table 1.2 API Functions

Function Name	Description
R_FLASH_SPI_Open()	Control software initialization processing
R_FLASH_SPI_Close()	Control software end processing
R_FLASH_SPI_Read_Status()	Status Register read processing
R_FLASH_SPI_Read_Status2()	Status Register 2 read processing
R_FLASH_SPI_Read_Status3()	Status Register 3 read processing
R_FLASH_SPI_Set_Write_Protect()	Write protect setting processing
R_FLASH_SPI_Write_Di()	WRDI command processing
R_FLASH_SPI_Read_Data()*1	Data read processing
R_FLASH_SPI_Write_Data_Page()*1	Data write (Single-page write) processing
R_FLASH_SPI_Erase()	Erase processing
R_FLASH_SPI_Polling()	Polling processing
R_FLASH_SPI_Read_ID()	ID read processing
R_FLASH_SPI_GetMemoryInfo()	Memory size acquisition processing
R_FLASH_SPI_Read_Configuration()	Configuration Register read processing
R_FLASH_SPI_Write_Configuration()	Configuration Register write processing
R_FLASH_SPI_Write_Status()	Status Register 1 write processing
R_FLASH_SPI_Write_Status2()	Status Register 2 write processing
R_FLASH_SPI_Write_Status3()	Status Register 3 write processing
R_FLASH_SPI_Set_4byte_Address_Mode()	4-byte address mode setting processing
R_FLASH_SPI_Read_Security()	Security register read processing
R_FLASH_SPI_Read_Data_Security_Page()	Data Security Page read processing
R_FLASH_SPI_Write_Data_Security_Page()	Data Security Page write processing
R_FLASH_SPI_Quad_Enable()	Quad mode enable setting processing
R_FLASH_SPI_Quad_Disable()	Quad mode disable setting processing
R_FLASH_SPI_GetVersion()	Control software version information acquisition processing
R_FLASH_SPI_Set_LogHdlAddress()	LONGQ FIT module handler address setting processing
R_FLASH_SPI_Log()	Error log acquisition processing using LONGQ FIT module
R_FLASH_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 flash memory control software are listed below. Refer to them alongside this application note.

1.3.1 FIT Module–Related Application Notes

- RX Family RSPI Module Using Firmware Integration Technology(R01AN1827)
- RX Family QSPI Clock Synchronous Single Master Control Module Using Firmware Integration Technology(R01AN1940)
- RX Family Application Note SCI Multi-Mode Module Using Firmware Integration Technology(R01AN1815)
- RX Family QSPIX Module Using Firmware Integration Technology(R01AN5685)
- RX Family RSCI Module Using Firmware Integration Technology(R01AN5759)
- 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 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 Clock Synchronous Control Module for EEPROM Access Firmware Integration Technology(R01AN2325)
- RX Family Memory Access Driver Interface Module Using Firmware Integration Technology(R01AN4548)

1.4 Using Serial Flash Memory Module

1.4.1 Using Serial Flash Memory Module in C++ project

For C++ project, add Serial Flash Memory Module header file within extern “C”{}:

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

1.5 Hardware Settings

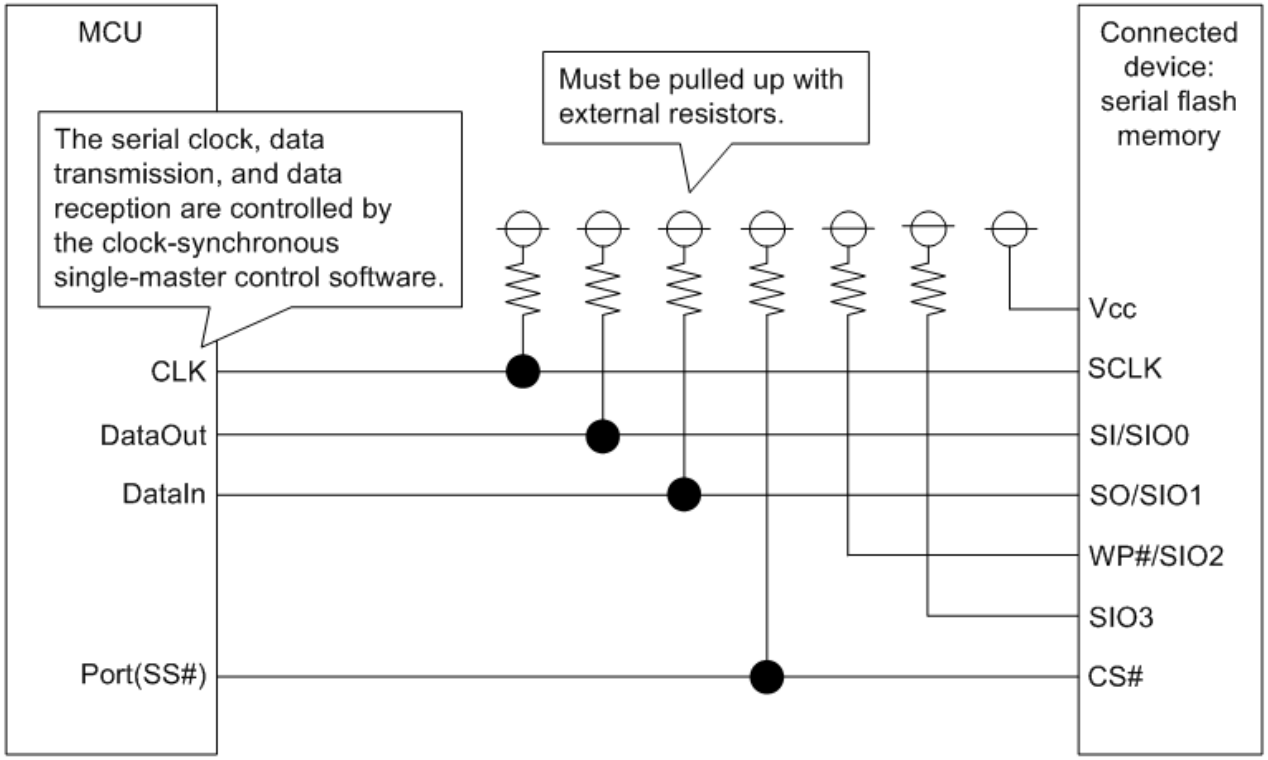
1.5.1 Hardware Configuration Example

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.

(1) Single-SPI Configuration Example

An example wiring diagram when using single-SPI is shown below.



- The names of the pins used by the microcontroller for serial I/O depend on the microcontroller version.
- In the example WP# and RESET# are not used. When using WP# and RESET#, be sure to check the specifications of the device to be used.

Figure 1.2 Sample Wiring Diagram for MCU and SPI Slave Device Using Single-SPI

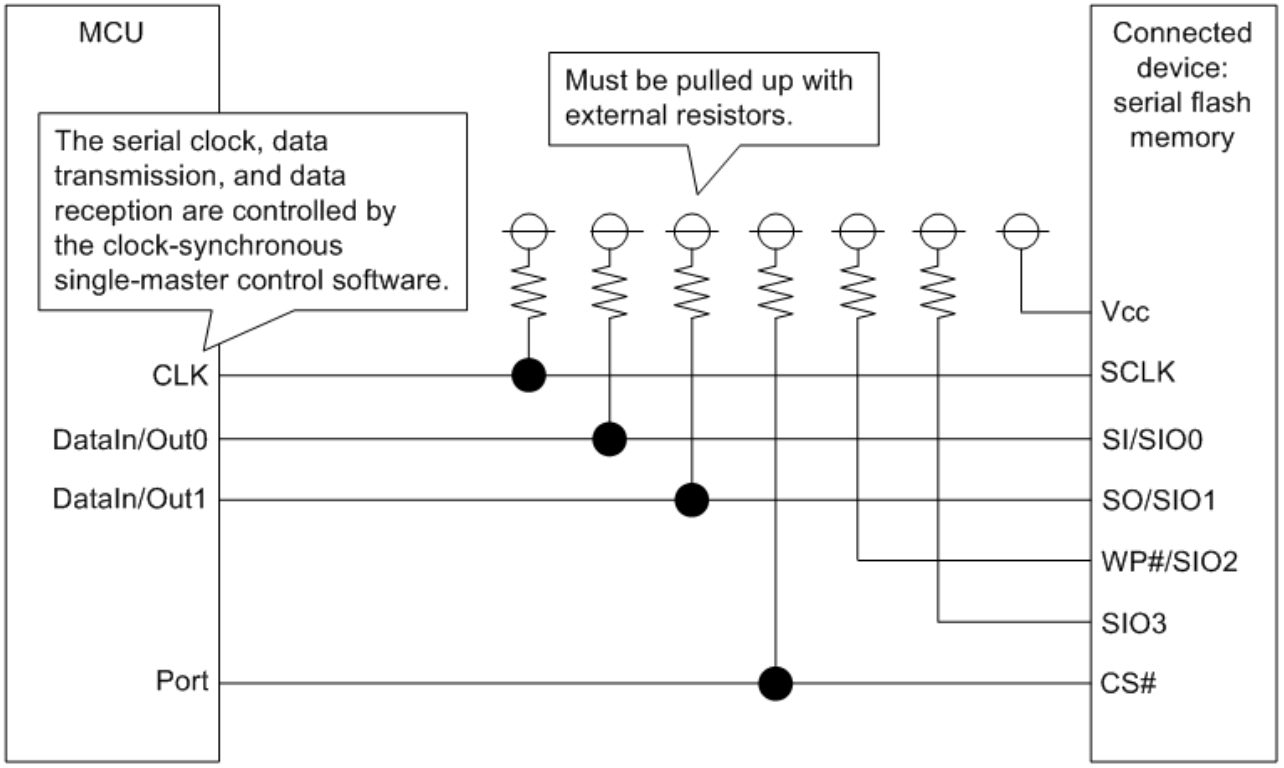
Table 1.3 Single-SPI Pins Used and Functions

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 device select (SS#) output

(2) Dual-SPI Configuration Example

An example wiring diagram when using dual-SPI is shown below.

In order to use dual-SPI the target microcontroller must be equipped with quad serial peripheral interface functionality.



- The names of the pins used by the microcontroller for serial I/O depend on the microcontroller version.
- In the example WP# and RESET# are not used. When using WP# and RESET#, be sure to check the specifications of the device to be used.

Figure 1.3 Sample Wiring Diagram for MCU and SPI Slave Device Using Dual-SPI

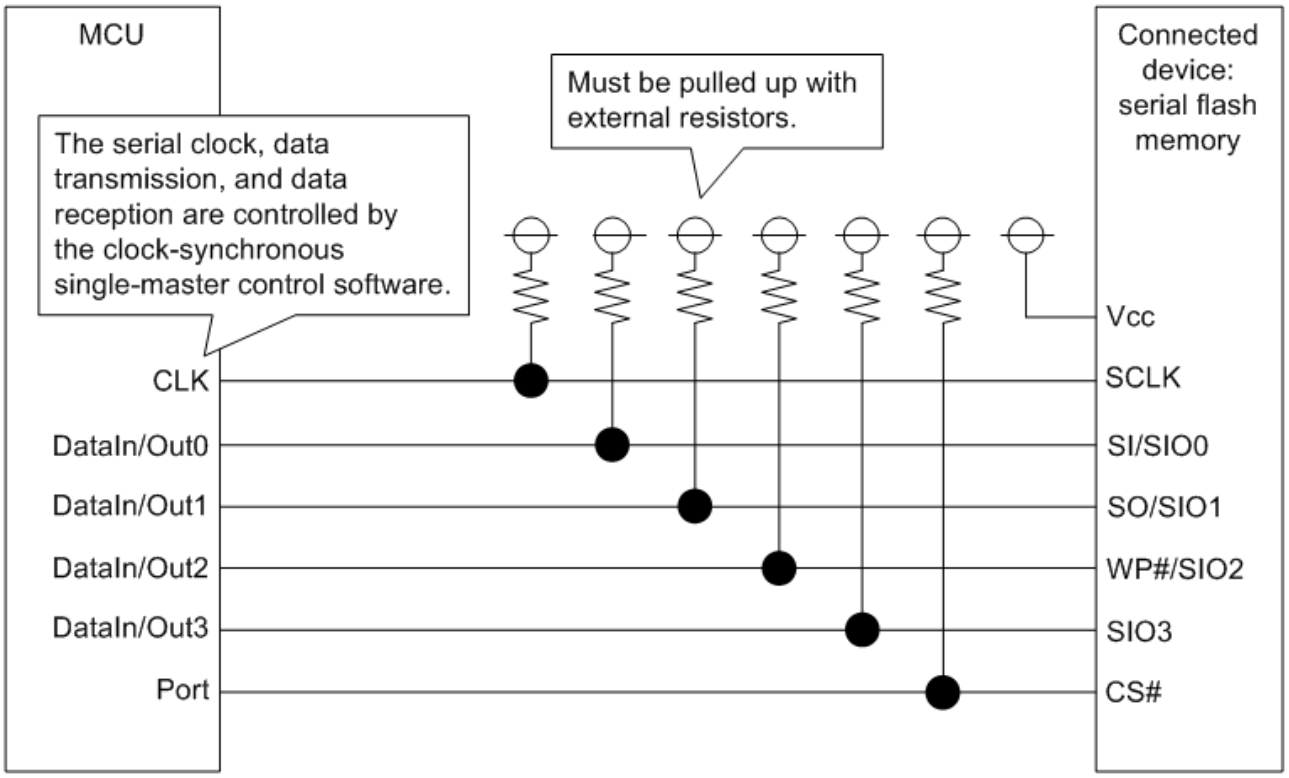
Table 1.4 Dual-SPI Pins Used and Functions

Pin Name	I/O	Description
CLK	Output	Clock output
DataIn/Out0	I/O	Master data I/O 0
DataIn/Out1	I/O	Master data I/O 1
Port (Port (SS#) of figure 1.3)	Output	Slave device select (SS#) output

(3) Quad-SPI Configuration Example

An example wiring diagram when using quad-SPI is shown below.

In order to use quad-SPI the target microcontroller must be equipped with quad serial peripheral interface functionality.



- The names of the pins used by the microcontroller for serial I/O depend on the microcontroller version.
- In the example WP# and RESET# are not used. When using WP# and RESET#, be sure to check the specifications of the device to be used.

Figure 1.4 Sample Wiring Diagram for MCU and SPI Slave Device Using Quad-SPI

Table 1.5 Quad-SPI Pins Used and Functions

Pin Name	I/O	Description
CLK	Output	Clock output
DataIn/Out0	I/O	Master data I/O 0
DataIn/Out1	I/O	Master data I/O 1
DataIn/Out2	I/O	Master data I/O 2
DataIn/Out3	I/O	Master data I/O 3
Port (Port (SS#) of figure 1.4)	Output	Slave device select (SS#) output

1.6 Software

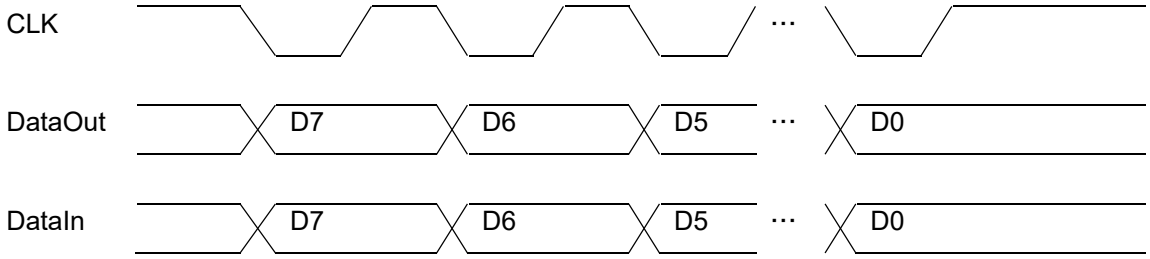
1.6.1 Operation Overview

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

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.

(4) Single-SPI Control

Control is performed in SPI mode 3 (CPOL = 1, CPHA = 1), as shown in figure 1.5.



・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.

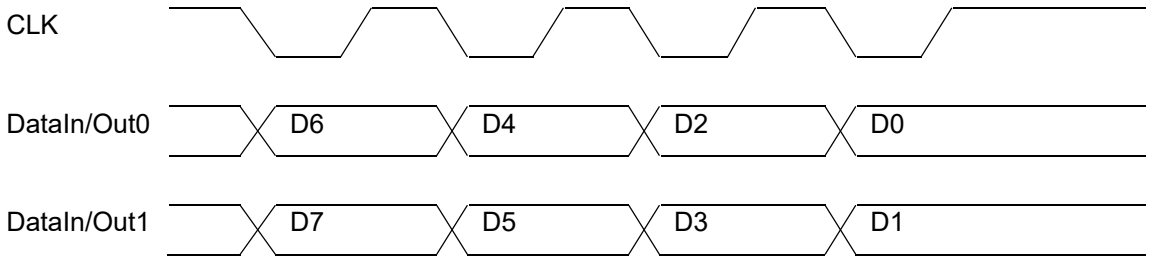
・MSB-first mode transfer.

・The level of the CLK pin is held high when no transfer processing is in progress.

Figure 1.5 Timing of Controllable Slave Devices for Single-SPI

(5) Dual-SPI Control

Control is performed in SPI mode 3 (CPOL = 1, CPHA = 1), as shown in figure 1.6.



・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.

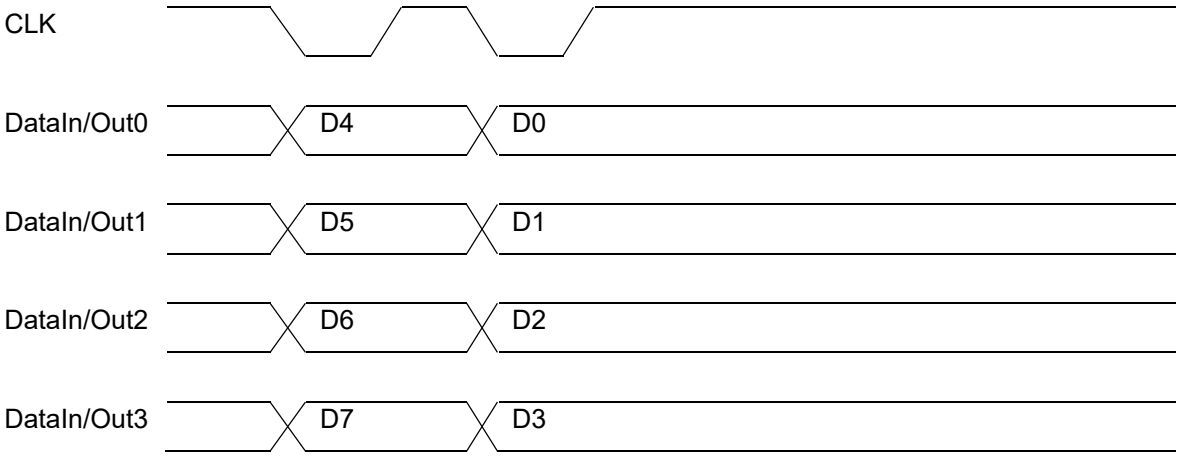
・MSB-first mode transfer.

・The level of the CLK pin is held high when no transfer processing is in progress.

Figure 1.6 Timing of Controllable Slave Devices for Dual-SPI

(6) Quad-SPI Control

Control is performed in SPI mode 3 (CPOL = 1, CPHA = 1), as shown in figure 1.7.



・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 1.7 Timing of Controllable Slave Devices for Quad-SPI

1.6.2 Serial Flash Memory Chip Select Pin Control

The chip select pin of the serial flash memory 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 chip select setup time of the serial flash memory, which is the time interval from the falling edge of the serial flash memory's chip select (microcontroller port (SS#)) signal to the falling edge of the serial flash memory's clock (microcontroller CLK) signal.

In like manner, control is performed in the software to wait during the chip select hold time of the serial flash memory, which is the time interval from the rising edge of the serial flash memory's clock (microcontroller CLK) signal to the rising edge of the serial flash memory'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.

1.6.3 Software Structure

Figure 1.8 shows the software structure.

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

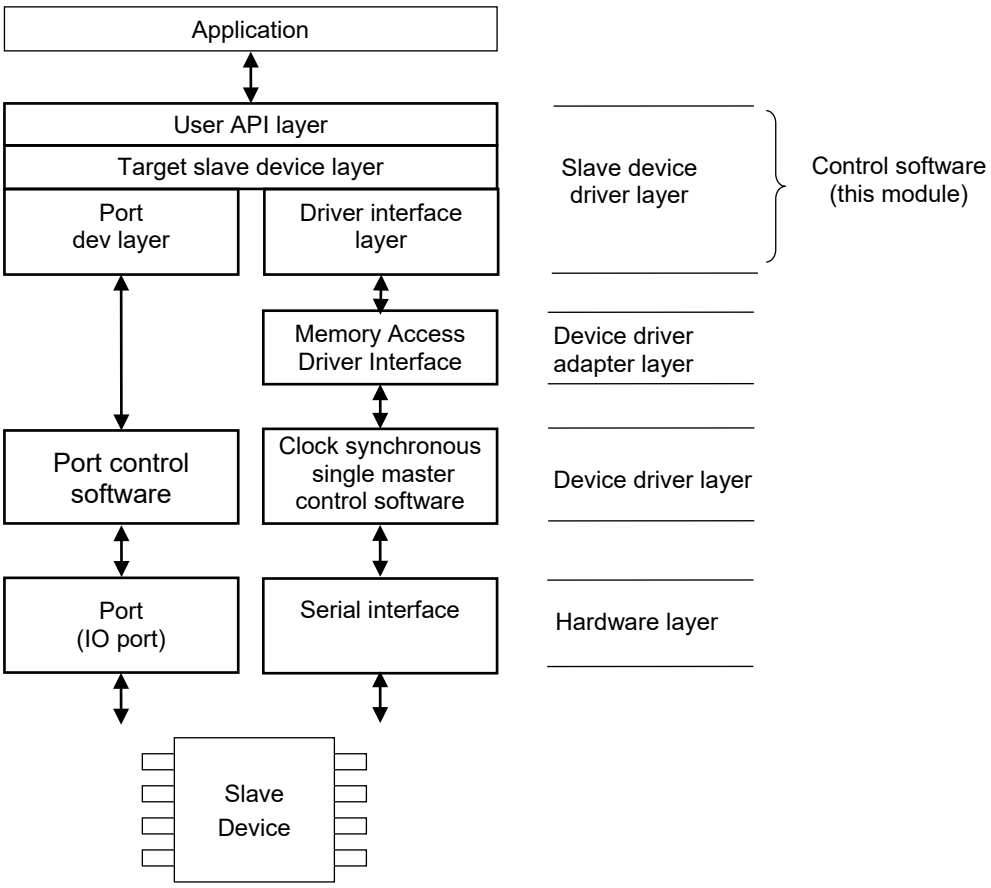


Figure 1.8 Software Structure

(a) User API layer (r_flash_spi.c)
The user interface, this portion of the software is not dependent on lower-layer device drivers.
(b) Target slave device layer (r_flash_spi_type.c)
The serial flash memory control module, this portion of the software is not dependent on lower-layer device drivers.
(c) Driver interface (I/F) layer (r_flash_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_flash_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 MX25L, MX66L, or MX25R family serial NOR flash memory, manufactured by Macronix International Co., Ltd., is provided for reference.
Sample code for controlling AT25QF family serial NOR flash memory, manufactured by Dialog Semiconductor Plc., is provided for reference.

1.6.4 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.

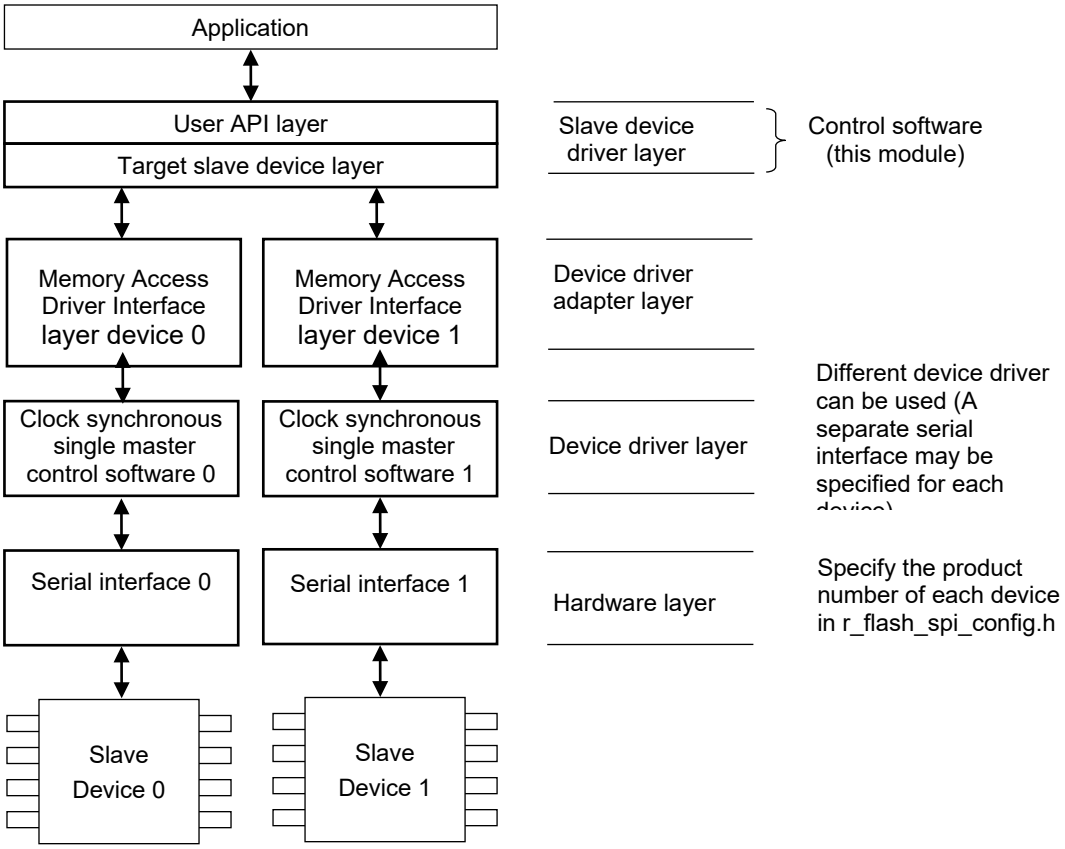


Figure 1.9 Software Configuration with Clock Synchronous Single Master Control Modules

1.6.5 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.

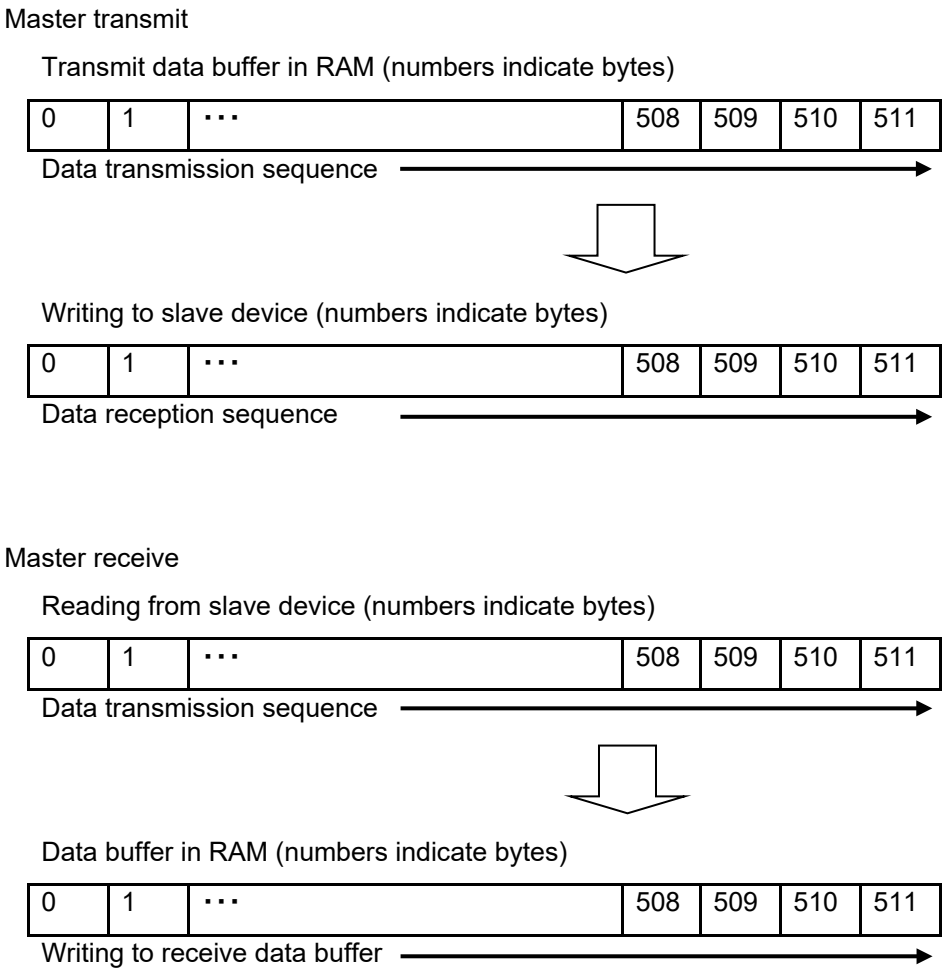


Figure 1.10 Data Buffers and Transmit/Receive Data

1.6.6 State Transition Diagram

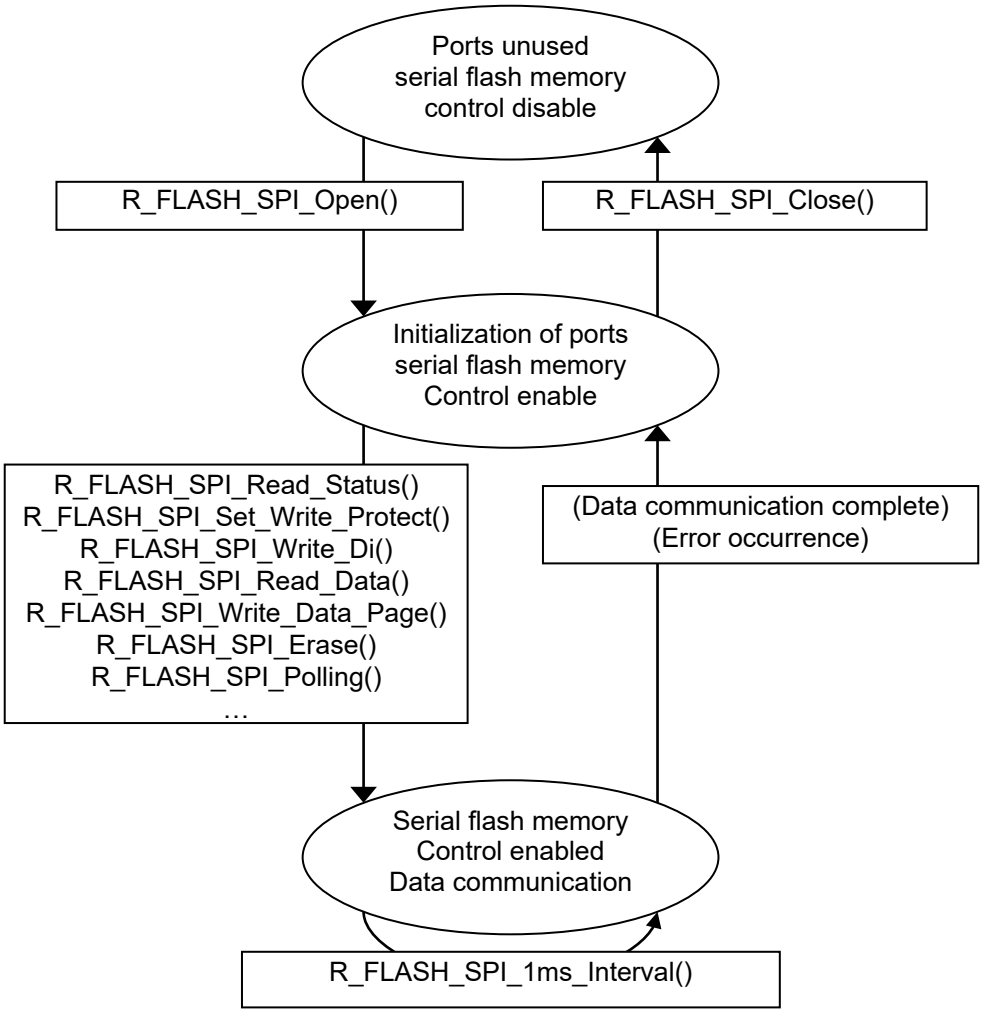


Figure 1.11 State Transition Diagram

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 Rev.1.04 or higher
- r_rspi_rx (when using the RSPI FIT module)
- r_qspix_rx (when using the QSPIX FIT module)
- r_rsci_rx (when using the RSCI FIT module)
- r_qspi_smstr_rx (when using the QSPI FIT module for clock synchronous single master control)
- r_scifa_smstr_rx (when using the SCIFA FIT module for clock synchronous single master 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 5.1,Confirmed Operation Environment.

2.4 Header Files

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

Configuration options for individual builds are selected in r_flash_spi_config.h. The included statements should be in the following order.

```
#include "r_flash_spi_if.h"
#include "r_flash_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_flash_spi_config.h.

The option names and setting values are described below.

Configuration options in *r_flash_spi_config.h*	
#define FLASH_SPI_CFG_WEL_CHK Note: The default value is “1 (enabled)”.	Selects whether or not the WEL bit is checked after the WREN command is issued. (1: enabled, 0: disabled)
#define FLASH_SPI_CFG_LONGQ_ENABLE Note: The default value is “0 (disabled)”.	Selects whether or not error log acquisition processing is performed for debugging, when using the BSP environment of a FIT module. (1: enabled, 0: disabled) 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. In addition, enable #define xxx_LONGQ_ENABLE in the clock synchronous single master control software of the specified device.
#define FLASH_SPI_CFG_USE_GPIO_MPC_FIT Note: The default value is “0 (disabled)”.	Selects whether the GPIO FIT module or MPC FIT module is used to control the SS# pin. (1: enabled, 0: disabled) When this option is set to “disabled”, neither the GPIO FIT module nor the MPC FIT module controls the SS# pin. When this option is set to “enabled”, the GPIO FIT module or MPC FIT module controls the SS# pin. To use this functionality, the GPIO FIT module or MPC FIT module is also required.
#define FLASH_SPI_CFG_DEVx_INCLUDED Note: The default value for device 0 is “1 (enabled)”. The “x” in DEVx represents the device number (x = 0 or 1).	This definition is related to device x. (1: enabled, 0: disabled) This option must be set to “enabled” for at least one device.
#define FLASH_SPI_CFG_DEVx_MX25L #define FLASH_SPI_CFG_DEVx_MX66L #define FLASH_SPI_CFG_DEVx_MX25R #define FLASH_SPI_CFG_DEVx_AT25QF Note: The default values for device 0 are FLASH_SPI_CFG_DEVx_MX25L: 1, other: 0. The “x” in DEVx represents the device number (x = 0 or 1).	Select only one serial flash memory device to be controlled for device x (1: control target, 0: not control target).

Configuration options in *r_flash_spi_config.h*	
#define FLASH_SPI_CFG_DEVx_SIZE_512K #define FLASH_SPI_CFG_DEVx_SIZE_2M #define FLASH_SPI_CFG_DEVx_SIZE_4M #define FLASH_SPI_CFG_DEVx_SIZE_8M #define FLASH_SPI_CFG_DEVx_SIZE_16M #define FLASH_SPI_CFG_DEVx_SIZE_32M #define FLASH_SPI_CFG_DEVx_SIZE_64M #define FLASH_SPI_CFG_DEVx_SIZE_128M #define FLASH_SPI_CFG_DEVx_SIZE_256M #define FLASH_SPI_CFG_DEVx_SIZE_512M #define FLASH_SPI_CFG_DEVx_SIZE_1G Note: The default values for device 0 are FLASH_SPI_CFG_DEVx_SIZE_64M: 1, other: 0. The “x” in DEVx represents the device number (x = 0 or 1).	Select only one serial flash memory capacity to be controlled for device x (1: control target, 0: not control target).
#define FLASH_SPI_CS_DEVx_CFG_PORTNO Note: The default value for device 0 is “'X'”. The “x” in DEVx represents the device number (x = 0 or 1).	Specifies the port number assigned to SS# for device x. Enclose the setting value in single quotation marks (‘ ’). Configure Device x Port Number with 0 - 9, A - X.
#define FLASH_SPI_CS_DEVx_CFG_BITNO Note: The default value for device 0 is “'0'”. The “x” in DEVx represents the device number (x = 0 or 1).	Specifies the bit number assigned to SS# for device x. Enclose the setting value in single quotation marks (‘ ’). 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_flash_spi_if.h, along with the prototype declarations of the API functions.

```
/* FLASH Memory 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         */
   flash_spi_opmode_t op_mode;      /* SPI operating mode; ignore it when
                          using write/read data in a Security register page */
} flash_spi_info_t;                 /* 20 bytes                            */

/* FLASH Memory size information */
typedef struct
{
   uint32_t    mem_size;            /* Max memory size                     */
   uint32_t    wpag_size;           /* Write page size                     */
} flash_spi_mem_info_t;             /* 8 bytes                             */

/* FLASH Memory erase information */
typedef struct
{
   uint32_t    addr;                /* Address to issue a command          */
   flash_spi_erase_mode_t  mode;    /* Mode of erase                       */
} flash_spi_erase_info_t;           /* 8 bytes                             */

/* FLASH Memory register information */
typedef struct
{
   uint8_t     status;        /* Status register                          */
   uint8_t     config1;       /* Configuration or Configuration-1 register */
   uint8_t     config2;       /* Configuration-2 register                 */
   uint8_t     rsv[1];
} flash_spi_reg_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_flash_spi rev.3.20

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 8.3.0.202202

(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: 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_flash_spi.c, r_flash_spi_dev_port_iodefine.c, r_flash_spi_drvif.c, r_flash_spi_type.c, r_flash_spi_type_sub.c

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
RX113	ROM	**4,491 bytes**	**11,396 bytes**	**7,825 bytes**
	RAM	**8 bytes**	**8 bytes**	**8 bytes**
	STACK	**140 bytes**	**-**	**200 bytes**
RX231	ROM	**4,495 bytes**	**11,436 bytes**	**7,860 bytes**
	RAM	**8 bytes**	**8 bytes**	**8 bytes**
	STACK	**140 bytes**	**-**	**200 bytes**

ROM, RAM and Stack Code Sizes				
Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
RX64M	ROM	**4,535 bytes**	**11,436 bytes**	**7,902 bytes**
	RAM	**8 bytes**	**8 bytes**	**8 bytes**
	STACK	**140 bytes**	**-**	**232 bytes**
RX71M	ROM	**4,535 bytes**	**11,436 bytes**	**7,823 bytes**
	RAM	**8 bytes**	**8 bytes**	**8 bytes**
	STACK	**140 bytes**	**-**	**232 bytes**

2.9 Return Values

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

```
typedef enum e_flash_status
{
   FLASH_SPI_SUCCESS_BUSY  =  1,    /* Successful operation (EERPOM is busy) */
   FLASH_SPI_SUCCESS       =  0,    /* Successful operation                  */
   FLASH_SPI_ERR_PARAM     =  -1,   /* Parameter error                       */
   FLASH_SPI_ERR_HARD      =  -2,   /* Hardware error                        */
   FLASH_SPI_ERR_WP        =  -4,   /* Write-protection error                */
   FLASH_SPI_ERR_TIMEOUT   =  -6    /* time out error                        */
   FLASH_SPI_ERR_OTHER     =  -7    /* Other error                           */
} flash_spi_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 (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.

2.11 Using the Serial Flash Memory Control Software in Other Than an FIT Module Environment

To use the serial flash memory 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_flash_spi_if.h.

Include the following header files in #r_flash_spi_if.h.

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

Disable the option #define FLASH_SPI_CFG_USE_FIT in #r_flash_spi_if.h.

Add the definition #define FLASH_SPI_CFG_xxx (replacing xxx with the microcontroller name using all capital letters) to #r_flash_spi_if.h. For example, for the RX64M microcontroller use the string FLASH_SPI_CFG_RX64M.

In # r_flash_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 1.5.1 (1), Single-SPI Control, 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_FLASH_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_FLASH_SPI_Open()	GPIO high-output state Set on user side
After R_FLASH_SPI_Open()	GPIO high-output state Set by this module
After R_FLASH_SPI_Close()	GPIO high-output state Set by this module

Note: * Use an external resistor to pull up the slave device select pin. See 1.5.1, 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

R_FLASH_SPI_Open()

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

Format

```
flash_spi_status_t R_FLASH_SPI_Open(
    uint8_t devno
)
```

Parameters

devno

Device number (0, 1)

Return Values

FLASH_SPI_SUCCESS */* Successful operation */*

FLASH_SPI_ERR_PARAM */* Parameter error */*

Properties

Prototype declarations are contained in r_flash_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Open(FLASH_SPI_DEV0);
```

Special Notes

After calling this user API function, it is recommended that R_FLASH_SPI_Polling() be used to confirm that the serial flash memory write cycle has completed. The next read or write processing will not be accepted while the serial flash memory write cycle is in progress.
However, it is possible to access the serial flash memory during the write cycle by, for example, issuing a system reset while the serial flash memory write cycle is in progress and restarting serial flash memory control from the beginning.

R_FLASH_SPI_Close()

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

Format

```
flash_spi_status_t R_FLASH_SPI_Close(
    uint8_t devno
)
```

Parameters

devno

Device number (0, 1)

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_flash_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Close(FLASH_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_FLASH_SPI_Polling() be used to confirm that the serial flash memory write cycle has completed. This makes it possible to restart serial flash memory control because the serial flash memory has not transitioned to the write cycle.

R_FLASH_SPI_Read_Status()

This function is used to read the status register.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Status(
    uint8_t devno,
    uint8_t * p_status
)
```

Parameters

devno

Device number (0, 1)

** p_status*

Status register storage buffer (size: 1 byte)

Return Values

```
FLASH_SPI_SUCCESS     /* Successful operation */
FLASH_SPI_ERR_PARAM   /* Parameter error */
FLASH_SPI_ERR_HARD    /* Hardware error */
FLASH_SPI_ERR_OTHER   /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the status register and stores the contents in p_status. The following information is stored in p_status: Refer to the data sheet for information on protected areas and protected bits.

- <MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>

Table 3.1 MX25L, MX66L, or MX25R family serial NOR flash memory

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRWD (status register write protect)	QE (quad enable)	BP3 (level of protected block)	BP2 (level of protected block)	BP1 (level of protected block)	BP0 (level of protected block)	WEL (write enable latch)	WIP (write in progress bit)
1 = status register write disable	1 = quad enable 0 = not quad enable	*1	*1	*1	*1	1 = write enable 0 = not write enable	1 = write operation 0 = not in write operation
Non-volatile bit	Non-volatile bit	Non-volatile bit	Non-volatile bit	Non-volatile bit	Non-volatile bit	Volatile bit	Volatile bit

Note 1: Set to 1, a designated memory area is protected from PROGRAM and ERASE operations.

- <AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.>

Table 3.2 AT25QF family serial NOR flash memory

Bit	Mnemonic	Name	Type	Description	
7	SRP0	Status Register Protection bit 0	R/W		*4
6	SEC	Block Protection	R/W	*3	*4
5	TB	Top or Bottom Protection	R/W	*2	*4
4	BP2	Block Protection bit 2	R/W	*1	*4
3	BP1	Block Protection bit 1	R/W	*1	*4
2	BP1	Block Protection bit 0	R/W	*1	*4
1	WEL	Write Enable Latch Status	R	0	Device is not Write Enable (default).
				1	Device is Write Enable.
0	$\overline{\text{RDY}}$ /BSY	Ready/Busy Status	R	0	Device is ready.
				1	Device is busy with an internal operation.

Note 1: Set to 0 or 1, determine how much of the array is protected.
Note 2: Set to 0 or 1, selects between top of the array or bottom of the array protection.
Note 3: Set to 0 or 1, selects between large and small block size protection.
Note 4: See the specification of Flash memory.

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint32_t              stat = 0;

ret = R_FLASH_SPI_Read_Status(FLASH_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.

R_FLASH_SPI_Read_Status2()

This function is used to read the status register 2. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Status2(
    uint8_t devno,
    uint8_t * p_status
)
```

Parameters

devno

Device number (0, 1)

** p_status*

Status register storage buffer (size: 1 byte)

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the status register 2 and stores the contents in p_status. The following information is stored in p_status:

- **<AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.>**

Table 3.3 AT25QF family serial NOR flash memory

Bit	Mnemonic	Name	Type	Description	
7	E_SUS	Erase Suspend Status	R	0	Erase operation is not suspended (default).
				1	Erase operation is suspended.
6	CMP	Complement Block Protection	R/W	*1	*2
5	LB3	Lock Security Register 3	R/W	0	Security Register page-3 is not locked (default).
				1	Security Register page-3 cannot be erased/programmed. *3
4	LB2	Lock Security Register 2	R/W	0	Security Register page-2 is not locked (default).
				1	Security Register page-2 cannot be erased/programmed. *3
3	BP1	Lock Security Register 1	R/W	0	Security Register page-1 is not locked (default).
				1	Security Register page-1 cannot be erased/programmed. *3
2	P_SUS	Program Suspend Status	R/	0	Program operation is not suspended (default).
				1	Program operation is suspended.
1	QE	Quad Enable	R/W	0	$\overline{\text{HOLD}}$ and $\overline{\text{WP}}$ function normally.
				1	$\overline{\text{HOLD}}$ and $\overline{\text{WP}}$ are I/O pins (default).
0	SRP1	Status Register Protect bit 1	R/W		*2

Note 1: Set to 0 or 1, complements the effect of the other bits.
Note 2: See the specification of Flash memory.
Note 3: Once a Lock Bit is set to 1, the corresponding Security Register is permanently locked. The Erase Security Register Page instruction is ignored for Security Registers with their Lock Bit set.

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint32_t              stat = 0;

ret = R_FLASH_SPI_Read_Status2(FLASH_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.

R_FLASH_SPI_Read_Status3()

This function is used to read the status register 3. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Status3(
    uint8_t devno,
    uint8_t * p_status
)
```

Parameters

devno

Device number (0, 1)

** p_status*

Status register storage buffer (size: 1 byte)

Return Values

```
FLASH_SPI_SUCCESS      /* Successful operation */
FLASH_SPI_ERR_PARAM    /* Parameter error */
FLASH_SPI_ERR_HARD     /* Hardware error */
FLASH_SPI_ERR_OTHER    /* Other task has acquired clock synchronous single
                          master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the status register 3 and stores the contents in p_status. The following information is stored in p_status:

- **<AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.>**

Table 3.4 AT25QF family serial NOR flash memory

Bit	Mnemonic	Name	Type	Description	
7	Res	Reserved	R	0	Reserved bit.
6:5	DRV[1:0]	Drive Strength	R/W	11	Drive level. Th DRV1 and DRV0 bits are used to determine the output drive strength during read operations. One of the setting of below allows the drive strength to be set by hardware based on the VCC level. Four drive settings are supported. This field is encoded as follows: 11: Auto (7pF base on VCC level. 10: 50% (15pF). 01: 75% (22pF). 00: 100% (30pF).
4:0	Res	Reserved	R	0	Reserved bit.

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint32_t              stat = 0;

ret = R_FLASH_SPI_Read_Status3(FLASH_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.

R_FLASH_SPI_Set_Write_Protect()

This function is used to make write protect settings.

Format

```
flash_spi_status_t R_FLASH_SPI_Set_Write_Protect(
    uint8_t devno,
    uint8_t wpsts
)
```

Parameters

devno

Device number (0, 1)

wpsts

Write protect setting data

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Make write protect settings. SRWD is cleared to 0.
Specify the write protect setting data (wpsts) as indicated below.
Refer to the data sheet for information on protected areas and protected bits.

- **<MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd>**

Make top and bottom settings during configuration write processing.

wpsts	BP3	BP2	BP1	BP0
0x00	0	0	0	0
0x01	0	0	0	1
0x02	0	0	1	0
0x03	0	0	1	1
0x04	0	1	0	0
0x05	0	1	0	1
0x06	0	1	1	0
0x07	0	1	1	1
0x08	1	0	0	0
0x09	1	0	0	1
0x0a	1	0	1	0
0x0b	1	0	1	1
0x0c	1	1	0	0
0x0d	1	1	0	1
0x0e	1	1	1	0
0x0f	1	1	1	1

- **<AT25QF family serial NOR flash memory of Dialog Semiconductor Plc>**

Make large and small block size, top, and bottom settings during status register 1 write processing.

Make compliment settings during status register 2 write processing.

wpsts	BP2	BP1	BP0
0x00	0	0	0
0x01	0	0	1
0x02	0	1	0
0x03	0	1	1
0x04	1	0	0
0x05	1	0	1
0x06	1	1	0
0x07	1	1	1

When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing. R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.
See figure 3.1 for details.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t      ret    = FLASH_SPI_SUCCESS;
uint32_t                loop_cnt = 0;
flash_spi_poll_mode_t   mode;

ret = R_FLASH_SPI_Set_Write_Protect(FLASH_SPI_DEV0, 0);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);      /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

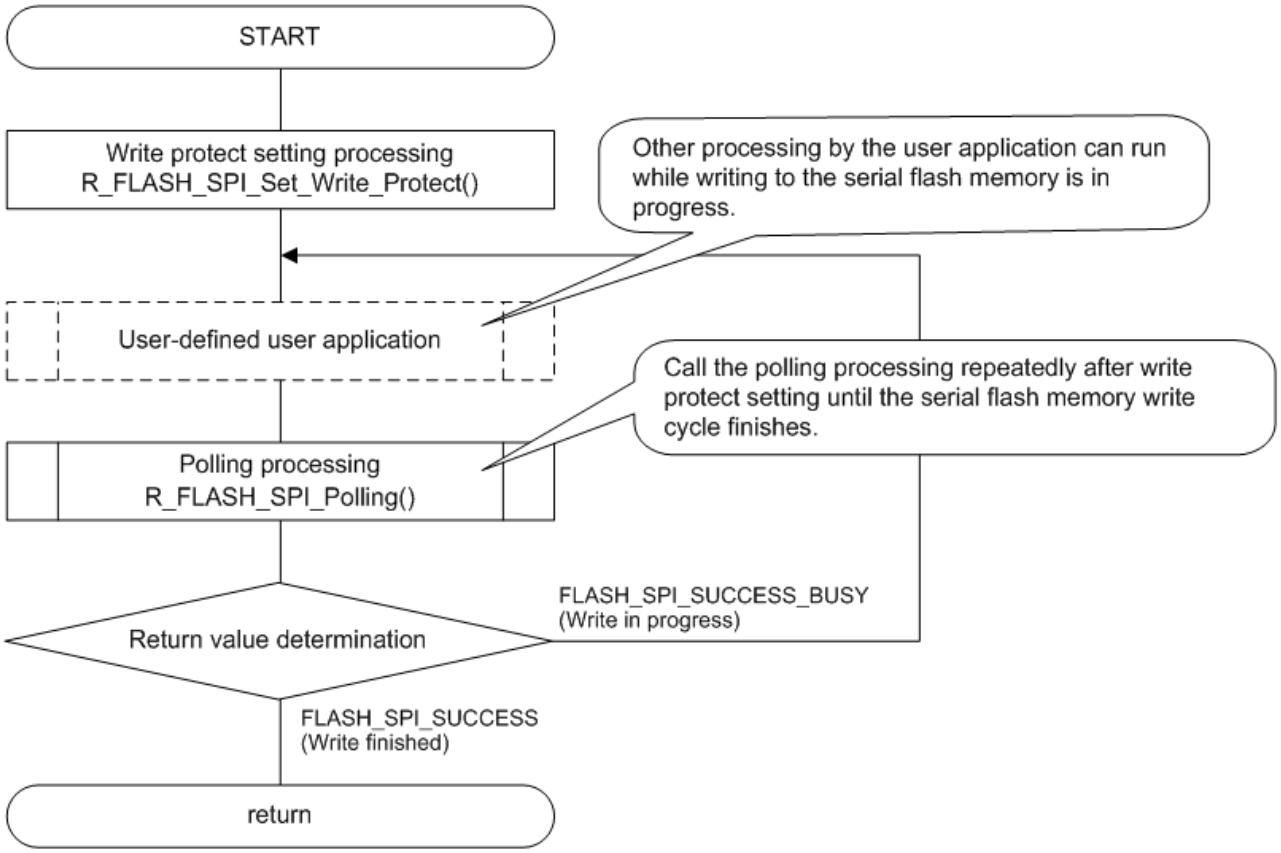


Figure 3.1 R_FLASH_SPI_Set_Write_Protect() Processing Example

R_FLASH_SPI_Write_Di()

This function is used to disable write operation.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Di(
    uint8_t devno
)
```

Parameters

devno

Device number (0, 1)

Return Values

```
FLASH_SPI_SUCCESS       /* Successful operation */
FLASH_SPI_ERR_PARAM     /* Parameter error */
FLASH_SPI_ERR_HARD      /* Hardware error */
FLASH_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                           master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

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

Example

```
flash_spi_status_t     ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Write_Di(FLASH_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.

R_FLASH_SPI_Read_Data()

This function is used to read data from the serial flash memory.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Data(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

Device number (0, 1)

** p_flash_spi_info*

Serial flash memory information structure. Use a 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.

op_mode

SPI mode setting.

Please select one from the following:

```
FLASH_SPI_SINGLE /* Single-SPI (full-duplex communication) */
FLASH_SPI_DUAL /* Dual-SPI (half-duplex communication) */
FLASH_SPI_QUAD /* Quad-SPI (half-duplex communication) */
```

Return Values

```
FLASH_SPI_SUCCESS       /* Successful operation */
FLASH_SPI_ERR_PARAM     /* Parameter error */
FLASH_SPI_ERR_HARD      /* Hardware error */
FLASH_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                           master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the specified number of bytes of data from the specified address in the serial flash memory and stores the data in p_data.

The maximum read address is the serial flash memory capacity – 1.

Rollover read operations are not supported. After the end address is read, processing ends. It is then necessary to reset the address and call this API function again.

FLASH_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_R;
uint32_t             buf2[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Flash_Info_R.addr    = 0;
Flash_Info_R.cnt     = 32;
Flash_Info_R.p_data  = (uint8_t *)&buf2[0];
Flash_Info_R.op_mode = FLASH_SPI_QUAD;
ret = R_FLASH_SPI_Read_Data(FLASH_SPI_DEV0, &Flash_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.

When reading data in QSPIX memory-mapped mode, data can only be read from addresses in a specified bank. A bank is a 64MB sliding access window into the QSPI device flash memory space. The code is not designed to read data from flash memory across banks. If the requested amount of data is read consecutively from the address area of two consecutive bank areas, this function will return the error "FLASH_SPI_ERR_OTHER".

R_FLASH_SPI_Write_Data_Page()

This function is used to write data to the serial flash memory in single-page units.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Data_Page(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

Device number (0, 1)

** p_flash_spi_info*

Serial flash memory information structure. Use a 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.

op_mode

SPI mode setting.

Please select one from the following:

FLASH_SPI_SINGLE /* Single-SPI (full-duplex communication) */

FLASH_SPI_QUAD /* Quad-SPI (half-duplex communication) */

※: Setting of FLASH_SPI_DUAL is prohibited.

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_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 serial flash memory, 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 serial flash memory 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 FLASH_SPI_ERR_WP.
The maximum write address is the serial flash memory capacity – 1.
The maximum write byte count (cnt) setting value is the capacity of the serial flash memory.
FLASH_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 serial flash memory information stricture (p_flash_info) after processing of a single page write

finishes. It is possible to write the remaining bytes by specifying p_flash_info unmodified in this API function again.
After this user API function finishes successfully, the serial flash memory transitions to the write cycle. Do not fail to confirm that the write has finished with R_FLASH_SPI_Polling(). If an attempt is made to perform the next read or write processing while a write cycle is in progress, the serial flash memory will not accept that processing.
R_FLASH_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

```
#define FLASH_PP_BUSY_WAIT (uint32_t)(3)      /* 3 * 1ms = 3ms */

flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_W;
uint32_t             buf1[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */
uint32_t             loop_cnt = 0;

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

do
{
   ret = R_FLASH_SPI_Write_Data_Page(FLASH_SPI_DEV0, &Flash_Info_W);
   if (FLASH_SPI_SUCCESS > ret)
   {
      /* Error */
   }

   loop_cnt = FLASH_PP_BUSY_WAIT;
   mode = FLASH_SPI_MODE_PROG_POLL;
   do
   {
      /* FLASH is busy.
         User application can perform other processing while flash is busy. */

         ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
         if (FLASH_SPI_SUCCESS_BUSY != ret)
         {
            /* FLASH is ready or error. */
            break;
         }
         loop_cnt--;
         wait_timer(0, 1);      /* 1ms */
   }
   while (0 != loop_cnt);
}
while (0 != Flash_Info_W.cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

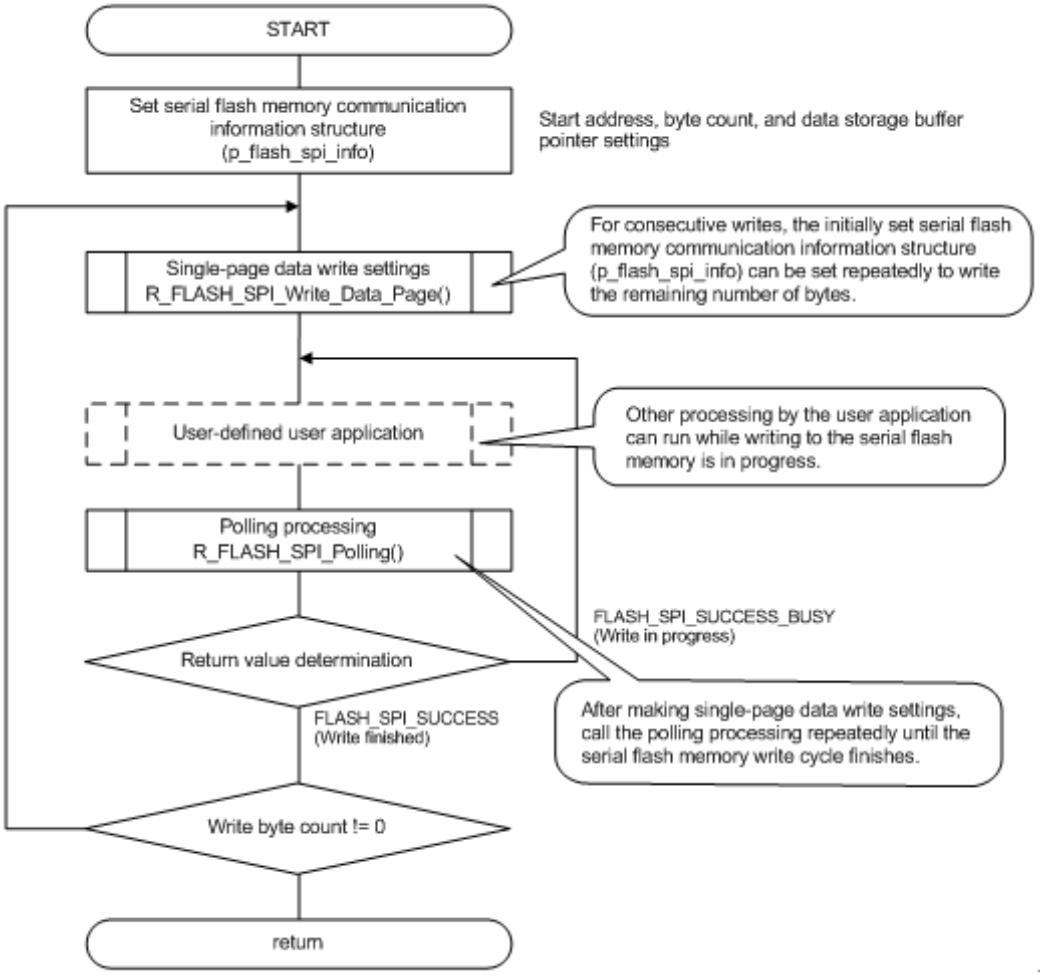


Figure 3.2 R_FLASH_SPI_Write_Data_Page() Processing Example

R_FLASH_SPI_Erase()

Based on the mode setting, this function erases all the data in the specified sector (sector erase), all the data in the specified block (block erase: 32 KB block or 64 KB block), or all the data on the specified chip (chip erase).

Format

```
flash_spi_status_t R_FLASH_SPI_Erase(
    uint8_t devno,
    flash_spi_erase_info_t * p_flash_spi_erase_info
)
```

Parameters

devno

Device number (0, 1)

** p_flash_spi_erase_info*

Serial flash memory erase information structure. Use a structure address aligned with a 4-byte boundary.

addr

Specify the start address of the memory write.

mode

Erase mode setting

Select one of the following:

- <MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>

```
FLASH_SPI_MODE_C_ERASE      /* Erases all the data on the chip (chip erase) */
FLASH_SPI_MODE_S_ERASE      /* Erases all the data on the sector (sector erase) */
FLASH_SPI_MODE_B32K_ERASE   /* Erases all the data in the block (block erase: 32 KB) */
FLASH_SPI_MODE_B64K_ERASE   /* Erases all the data in the block (block erase: 64 KB) */
```

- <AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.>

```
FLASH_SPI_MODE_C_ERASE      /* Erases all the data on the chip (chip erase) */
FLASH_SPI_MODE_B4K_ERASE    /* Erases all the data in the block (block erase: 4 KB) */
FLASH_SPI_MODE_B32K_ERASE   /* Erases all the data in the block (block erase: 32 KB) */
FLASH_SPI_MODE_B64K_ERASE   /* Erases all the data in the block (block erase: 64 KB) */
FLASH_SPI_MODE_SCUR_ERASE   /* Erases the security register pages */
```

Return Values

```
FLASH_SPI_SUCCESS     /* Successful operation */
FLASH_SPI_ERR_PARAM   /* Parameter error */
FLASH_SPI_ERR_HARD    /* Hardware error */
FLASH_SPI_ERR_OTHER   /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

For sector erase, specify the start address of the sector in addr.
For block erase, specify the start address of the block in addr.
For security erase, specify the start address of the security register pages in addr.
For chip erase, set addr to 0x00000000.
Erasing data in serial flash memory is only possible when write protect has been canceled.
It is not possible to erase data in a protected area. Attempting to do so returns the error FLASH_SPI_ERR_OTHER.
When this user API function completes successfully, the serial flash memory transitions to an erase cycle. Do not fail to confirm erase completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous erase cycle is in progress, the serial flash memory will not accept the new processing.
R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while an erase cycle is in progress.

See figure 3-3 for details.

Example

```
#define FLASH_SE_BUSY_WAIT (uint32_t)(200)    /* 200 * 1ms = 200ms */

flash_spi_status_t      ret = FLASH_SPI_SUCCESS;
flash_spi_erase_info_t  Flash_Info_E;
uint32_t                loop_cnt = 0;

Flash_Info_E.addr    = 0;
Flash_Info_E.mode    = FLASH_SPI_MODE_S_ERASE;

ret = R_FLASH_SPI_Erase(FLASH_SPI_DEV0, &Flash_Info_E);
if (FLASH_SPI_SUCCESS > ret)
{
  /* Error */
}

loop_cnt = FLASH_SE_BUSY_WAIT;
mode = FLASH_SPI_MODE_ERASE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

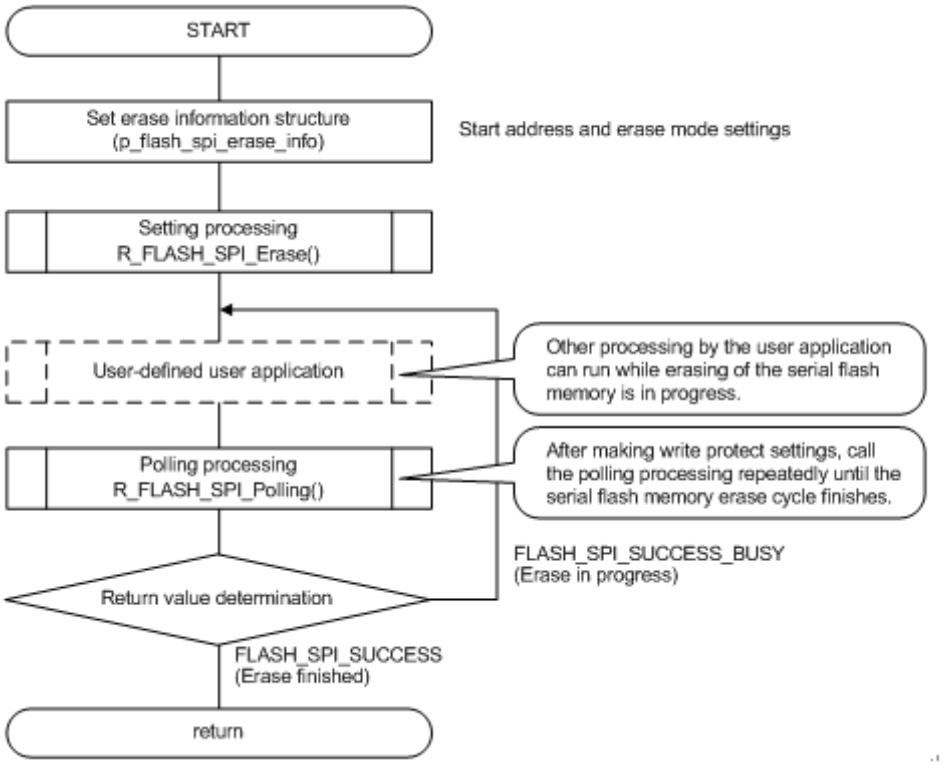


Figure 3.3 R_FLASH_SPI_Erase() Processing Example

R_FLASH_SPI_Polling()

This function is used to perform polling to determine if a write or erase operation has finished.

Format

```
flash_spi_status_t R_FLASH_SPI_Polling(
    uint8_t devno,
    flash_spi_poll_mode_t mode
)
```

Parameters

devno

Device number (0, 1)

mode

Completion wait processing setting

Select one of the following:

```
FLASH_SPI_MODE_REG_WRITE_POLL   /* Waits for register write to complete */
FLASH_SPI_MODE_PROG_POLL        /* Waits for data write to complete */
FLASH_SPI_MODE_ERASE_POLL       /* Waits for erase to complete */
```

Return Values

```
FLASH_SPI_SUCCESS       /* Normal end, and write finished */
FLASH_SPI_SUCCESS_BUSY  /* Normal end, and write in progress */
FLASH_SPI_ERR_PARAM     /* Parameter error */
FLASH_SPI_ERR_HARD      /* Hardware error */
FLASH_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                           master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Determines whether or not a write or erase operation has finished.

Example

Refer to figure 3.1 or figure 3.2.

Special Notes

R_FLASH_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.

R_FLASH_SPI_Read_ID()

This function is used to read ID information.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_ID(
    uint8_t devno,
    uint8_t * p_data
)
```

Parameters

devno

Device number (0, 1)

** p_data*

ID information storage buffer. The size differs depending on the serial flash memory product used. Refer to the following and check the contents of the read buffer.

- <MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>
  The function reads the manufacturer ID and device ID. Specify 3 bytes as a read buffer.
  (1) Manufacturer ID (1 byte)
  (2) Device ID (2 bytes)
- <AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.>
  The function reads the manufacturer ID and device ID. Specify 3 bytes as a read buffer.
  (1) Manufacturer ID (1 byte)
  (2) Device ID (2 bytes)

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Stores ID information for the serial flash memory in p_data.

Example

```
flash_spi_status_t     ret = FLASH_SPI_SUCCESS;
uint8_t     gID[4];

ret = R_FLASH_SPI_Read_ID(FLASH_SPI_DEV0, &gID[0]);
```

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.

R_FLASH_SPI_GetMemoryInfo()

This function is used to fetch the serial flash memory size information.

Format

```
flash_spi_status_t R_FLASH_SPI_GetMemoryInfo(
    uint8_t devno,
    flash_spi_mem_info_t * p_flash_spi_mem_info
)
```

Parameters

devno

Device number (0, 1)

** p_flash_spi_mem_info*

Serial flash memory size information structure. Use a structure address aligned with a 4-byte boundary.

mem_size

Maximum memory size

wpag_size

Page size

Return Values

```
FLASH_SPI_SUCCESS       /* Successful operation */
FLASH_SPI_ERR_PARAM     /* Parameter error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Fetches serial flash memory size information for the device number specified by the argument devno.

Example

```
flash_spi_status_t     ret = FLASH_SPI_SUCCESS;
flash_spi_mem_info_t   Flash_MemInfo;

ret = R_FLASH_SPI_GetMemoryInfo(FLASH_SPI_DEV0, &Flash_MemInfo);
```

Special Notes

None

R_FLASH_SPI_Read_Configuration()

This function is used to read the configuration register(s). It is a dedicated API function for MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Configuration (
    uint8_t devno,
    uint8_t * p_config
)
```

Parameters

devno

Device number (0, 1)

** p_config*

Configuration register storage buffer. The size differs depending on the serial NOR flash memory product used. Refer to the following and check the contents of the read buffer.

- **<MX25L, MX66L or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>**
  The function reads the configuration register. Specify 1 byte as a read buffer.
- **<MX25L, MX66L or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>**
  The function reads configuration register 1 and configuration register 2. Specify 2 bytes as a read buffer.

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the configuration register(s) of the serial NOR Flash memory and stores the contents in p_config. The data stored in p_config is listed below. Note that, depending on the serial NOR flash memory product used, there may be function allocations or reserved bits. For details, refer to the data sheet of the serial NOR flash memory product used.

- **<MX25L, MX66L or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>**

Table 3.5 Configuration Register Listing

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DC1 (dummy cycle 1)	DC0 (dummy cycle 0)	4 BYTE	PBE (preamble bit enable)	TB (top/bottom selected)	ODS 2 (output driver strength)	ODS 1 (output driver strength)	ODS 0 (output driver strength)
*	*	0 = 3-byte address mode 1 = 4-byte address mode (default = 0)	0 = disable 1 = enable	0 = top area protect 1 = bottom area protect (default = 0)	*	*	*
volatile bit	volatile bit	volatile bit	volatile bit	OTP	volatile bit	volatile bit	volatile bit

Note: * See the specification of the Flash memory.

- **<MX25R family serial NOR flash memory of Macronix International Co., Ltd.>**

Table 3.6 Configuration Register 2 Listing

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	L/H switch	Reserved
x	x	x	x	x	x	0 = low power mode (default) 1 = high performance mode	x
x	x	x	x	x	x	volatile bit	x

Example

```
flash_spi_status_t     ret = FLASH_SPI_SUCCESS;
uint8_t     gConfig[4];    /* the buffer boundary (4-byte unit) */

ret = R_FLASH_SPI_Read_Configuration(FLASH_SPI_DEV0, &gConfig[0]);
```

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.

R_FLASH_SPI_Write_Configuration()

This function is used to write the configuration register(s). It is a dedicated API function for MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Configration(
    uint8_t devno,
    flash_spi_reg_info_t * p_reg
)
```

Parameters

devno

Device number (0, 1)

** p_reg*

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

status

Status register (Used by the control software, so setting by the user is prohibited.)

config1

Configuration register setting data

config2

Configuration register 2 setting data

Note that the configuration of the structure differs depending on the serial NOR flash memory product used. Refer to the following when making settings. Also, refer to “Description” for setting values.

- **<MX25L or MX66L family serial NOR flash memory of Macronix International Co., Ltd.>**
  The value set in p_reg ->config1 is written to the configuration register.
  The setting of p_reg->config2 is ignored.
- **<MX25R family serial NOR flash memory of Macronix International Co., Ltd.>**
  The value set in p_reg->config1 is written to the configuration register 1.
  The value set in p_reg->config2 is written to the configuration register 2.

Return Values

```
FLASH_SPI_SUCCESS     /* Successful operation */
FLASH_SPI_ERR_PARAM   /* Parameter error */
FLASH_SPI_ERR_HARD    /* Hardware error */
FLASH_SPI_ERR_OTHER   /* Other task has acquired clock synchronous single
                         master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

The values set in p_reg->config1 and p_reg->config2 are written to the configuration register.
Refer to the information below when making settings to p_reg->config1 and p_reg->config2. Note that, depending on the serial NOR flash memory product used, there may be function allocations or reserved bits. For details, refer to the data sheet of the serial NOR flash memory product used.

Configuration register
Bits 7 to 6: DC1-DC0 (Dummy cycle)
See the specification of the Flash memory.
Bit 5: 4BYTE (4BYTE Indicator)
1: 4-byte address mode
0: 3-byte address mode
Bit 4: PBE (Preamble bit Enable)
1: Enable

0: Disable
Bit 3: TB (Top/Bottom)
1: Bottom area protect
0: Top area protect
Bits 2 to 0: ODS2-ODS0 (Output driver strength)
See the specification of the Flash memory.

Configuration register 2
Bits 7 to 2: Reserved
Bit 1: L/H Switch
1: High performance mode
0: Low power mode
Bit 0: Reserved

Before calling this user API function, read the value of the configuration registers, change the values of only the bits that need to be overwritten, and then make settings to p_reg->config1 and p_reg->config2.
After processing finishes, read the configuration registers to confirm that the written values are correct.
The 4BYTE bit is read-only, and its setting is ignored. This bit can be set to 1 by using R_FLASH_SPI_Set_4byte_Address_Mode().
When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing.
R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.
See figure 3.4 for details.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
flash_spi_reg_info_t Reg;
uint8_t              gConfig[4];    /* the buffer boundary (4-byte unit) */

ret = R_FLASH_SPI_Read_Configuration(FLASH_SPI_DEV0, &gConfig[0]);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg.config1 = (gConfig[0] | 0x10);      /* Set Preamble bit Enable */
ret = R_FLASH_SPI_Write_Configuration(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);      /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

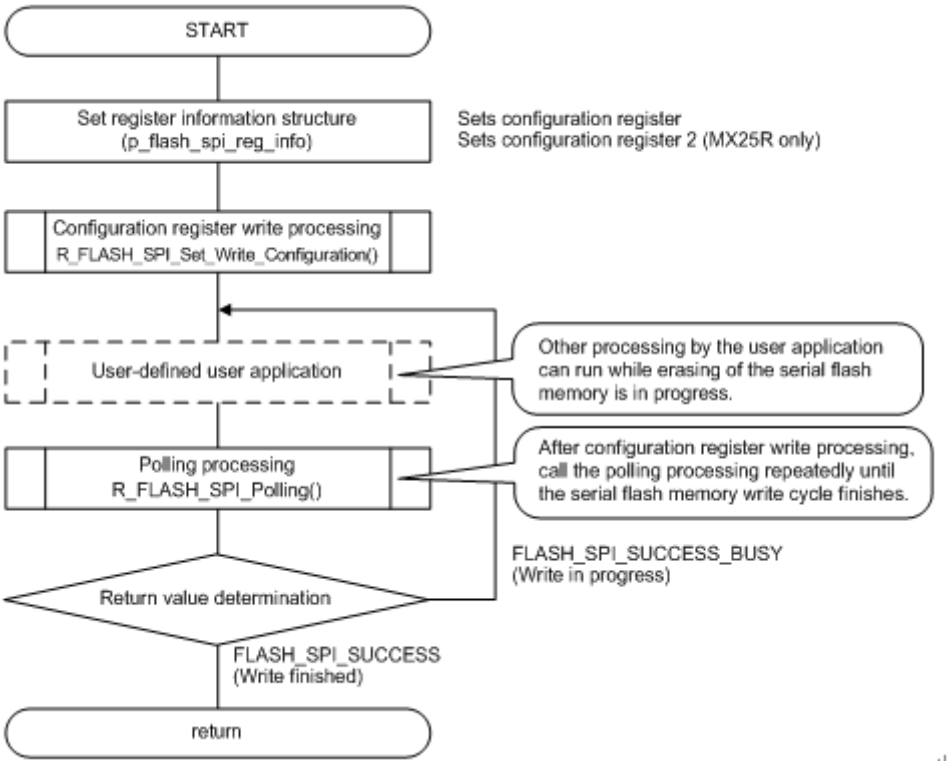


Figure 3.4 R_FLASH_SPI_Write_Configuration() Processing Example

R_FLASH_SPI_Write_Status()

This function is used to write the status register 1. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Status(
    uint8_t devno,
    uint8_t * p_reg
)
```

Parameters

devno
Device number (0, 1)
** p_reg*
Status register setting data buffer

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

The values set in p_reg is written to the status register 1. Note that, depending on the serial NOR flash memory product used, there may be function allocations or reserved bits. For details, refer to the data sheet of the serial NOR flash memory product used.

Status Register 1
Bit 7: Status Register Protection bit 0
See the specification of the Flash memory.
Bit 6: Block Protection
See the specification of the Flash memory.
Bit 5: TB (Top/Bottom)
See the specification of the Flash memory.
Bits 4 to 2: BP2-BP0 (Block Protection bit)
See the specification of the Flash memory.

Before calling this user API function, read the value of the status register 1, change the values of only the bits that need to be overwritten. After processing finishes, read the status register 1 to confirm that the written values are correct.
When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing. R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
uint8_t              gStat;
uint8_t              Reg;

ret = R_FLASH_SPI_Read_Status(FLASH_SPI_DEV0, &gStat);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg = (gStat | 0x10);
ret = R_FLASH_SPI_Write_Status(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

R_FLASH_SPI_Write_Status2()

This function is used to write the status register 2. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Status2(
    uint8_t devno,
    uint8_t * p_reg
)
```

Parameters

devno

Device number (0, 1)

** p_reg*

Status register setting data buffer

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

The values set in p_reg is written to the status register 2. Note that, depending on the serial NOR flash memory product used, there may be function allocations or reserved bits. For details, refer to the data sheet of the serial NOR flash memory product used.

Status Register 2
Bits 7: Complement Block Protection
See the specification of the Flash memory.
Bit 6: Lock Security Register 3
1: Security Register page-3 cannot be erased/programmed.
0: Security Register page-3 is not locked
Bit 5: Lock Security Register 2
1: Security Register page-2 cannot be erased/programmed.
0: Security Register page-2 is not locked
Bit 4: Lock Security Register 1
1: Security Register page-1 cannot be erased/programmed.
0: Security Register page-1 is not locked
Bits 1: Quad Enable
1: HOLD and WP are I/O pins
0: HOLD and WP function normally.

Before calling this user API function, read the value of the status register 2, change the values of only the bits that need to be overwritten. After processing finishes, read the status register 2 to confirm that the written values are correct.
When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing. R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)    /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
uint8_t              gStat;
uint8_t              Reg;

ret = R_FLASH_SPI_Read_Status2(FLASH_SPI_DEV0, & gStat);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg = (gStat | 0x10);
ret = R_FLASH_SPI_Write_Status2(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

R_FLASH_SPI_Write_Status3()

This function is used to write the status register 3. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Status2(
    uint8_t devno,
    uint8_t * p_reg
)
```

Parameters

devno

Device number (0, 1)

** p_reg*

Status register setting data buffer

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

The values set in p_reg is written to the status register 3. Note that, depending on the serial NOR flash memory product used, there may be function allocations or reserved bits. For details, refer to the data sheet of the serial NOR flash memory product used.

Status Register 3
Bit 7: Reserved
Bits 6 to 5: DRV[1:0] (Drive strength)
11: Auto (7 pF based on VCC level)
10: 50% (15 pF)
01: 75% (22 pF)
00: 100% (30 pF)
Bits 4 to 0: Reserved

Before calling this user API function, read the value of the status register 3, change the values of only the bits that need to be overwritten. After processing finishes, read the status register 3 to confirm that the written values are correct.
When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing. R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
uint8_t              gStat;
uint8_t              Reg;

ret = R_FLASH_SPI_Read_Status3(FLASH_SPI_DEV0, & gStat);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg = (gStat | 0x60);       /* Set auto drive strength */
ret = R_FLASH_SPI_Write_Status3(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

R_FLASH_SPI_Set_4byte_Address_Mode()

This function is used to set the address mode to 4-byte address mode. It is a dedicated API function for MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.

Format

```
flash_spi_status_t R_FLASH_SPI_Set_4byte_Address_Mode(
    uint8_t devno
)
```

Parameters

devno

Device number (0, 1)

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

- **<MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.>**

Issues the EN4B (0xb7) command to set the 4BYTE bit in the configuration register to 1.

Example

```
flash_spi_status_t     ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Set_4byte_Address_Mode(FLASH_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.

R_FLASH_SPI_Read_Security()

This function is used to read the security register. It is a dedicated API function for MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd.

Format

```
eepr_status_t R_FLASH_SPI_Read_Security
    uint8_t devno,
    uint8_t * p_scur
)
```

Parameters

devno

Device number (0, 1)

** p_scur*

Security register storage buffer (size: 1 byte)

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the security register and stores the contents in p_scur.
The information stored in p_scur is listed below. Note that, depending on the serial NOR flash memory product used, there may be function allocations or reserved bits. For details, refer to the data sheet of the serial NOR flash memory product used.

Bit 7: WPSEL
- 1: Individual mode
- 0: Normal WP mode

Bit 6: E_FAIL
- 1: Erase failed
- 0: Erase succeed

Bit 5: P_FAIL
- 1: Program failed
- 0: Program succeed

Bit 4: Reserved

Bit 3: ESB (Erase Suspend Bit)
- 1: Erase Suspended
- 0: Erase is not suspended

Bit 2: PSB (Program Suspend Bit)
- 1: Program Suspended
- 0: Program is not suspended

Bit 1: LDSO (Indicate if lock-down)
- 1: Lock-down (Cannot program/erase OTP)
- 0: Not lock-down

Bit 0: Secured OTP indicator
- 1: Factory lock
- 0: Non-factory lock.

If P_FAIL is set to 1, it is cleared to 0 the next time a programming operation succeeds.
If E_FAIL is set to 1, it is cleared to 0 the next time an erase operation succeeds.

Example

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
uint8_t              scur = 0;

ret = R_FLASH_SPI_Read_Security(FLASH_SPI_DEV0, &dcur);
```

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.

R_FLASH_SPI_Read_Data_Security_Page()

This function is used to read data from the security registers. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Data_Security_Page(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

Device number (0, 1)

** p_flash_spi_info*

Serial flash memory information structure. Use a structure address aligned with a 4-byte boundary.

addr

Specify the start address of the security register.

cnt

Specify the read byte count. The allowable setting range is 1 to 256. 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.

Return Values

```
FLASH_SPI_SUCCESS       /* Successful operation */
FLASH_SPI_ERR_PARAM     /* Parameter error */
FLASH_SPI_ERR_HARD      /* Hardware error */
FLASH_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                           master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Reads the specified number of bytes of data from the specified address in the security register and stores the data in p_data.
The maximum read address is the page size – 1.
Rollover read operations are not supported. After the end address is read, processing ends. It is then necessary to reset the address and call this API function again.
FLASH_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_R;
uint32_t             buf2[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Flash_Info_R.addr    = 0x1000;/* Security Register 1 Address */
Flash_Info_R.cnt     = 32;
Flash_Info_R.p_data  = (uint8_t *)&buf2[0];
ret = R_FLASH_SPI_Read_Data_Security_Page(FLASH_SPI_DEV0, &Flash_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.
This API function is omitted in QSPIX Memory Mapped Mode.

R_FLASH_SPI_Write_Data_Security_Page()

This function is used to write data to the security register pages in single-page units. It is a dedicated API function for AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Data_Security_Page(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

Device number (0, 1)

** p_flash_spi_info*

Serial flash memory information structure. Use a structure address aligned with a 4-byte boundary.

addr

Specify the start address of the security register.

cnt

Specify the write byte count. The allowable setting range is 1 to 256. 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

```
FLASH_SPI_SUCCESS       /* Successful operation */
FLASH_SPI_ERR_PARAM     /* Parameter error */
FLASH_SPI_ERR_HARD      /* Hardware error */
FLASH_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                           master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_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 security register pages, 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 security register pages is only possible when they are not locked. It is not possible to write to a locked page. Attempting to do so returns the error FLASH_SPI_ERR_WP.
The maximum write byte count (cnt) setting value is the capacity of the security register page size.
FLASH_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.

Example

```
#define FLASH_PP_BUSY_WAIT (uint32_t)(3)      /* 3 * 1ms = 3ms */

flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_W;
uint32_t             buf1[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */
uint32_t             loop_cnt = 0;

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

do
{
   ret = R_FLASH_SPI_Write_Data_Security_Page(FLASH_SPI_DEV0, &Flash_Info_W);
   if (FLASH_SPI_SUCCESS > ret)
   {
      /* Error */
   }

   loop_cnt = FLASH_PP_BUSY_WAIT;
   mode = FLASH_SPI_MODE_PROG_POLL;
   do
   {
      /* FLASH is busy.
         User application can perform other processing while flash is busy. */

         ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
         if (FLASH_SPI_SUCCESS_BUSY != ret)
         {
            /* FLASH is ready or error. */
            break;
         }
         loop_cnt--;
         wait_timer(0, 1);      /* 1ms */
   }
   while (0 != loop_cnt);
}
while (0 != Flash_Info_W.cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

R_FLASH_SPI_Quad_Enable()

This function is used to enable quad mode. It is a dedicated API function for MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd and AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Quad_Enable(
    uint8_t devno
)
```

Parameters

devno

Device number (0, 1)

Return Values

FLASH_SPI_SUCCESS	*/* Successful operation */*
FLASH_SPI_ERR_PARAM	*/* Parameter error */*
FLASH_SPI_ERR_HARD	*/* Hardware error */*
FLASH_SPI_ERR_OTHER	*/* Other task has acquired clock synchronous single master control software resources, or other error */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Sets the quad enable (QE) bit in the status register to 1 to enable quad mode.
To use quad mode, first call this function.
After processing finishes, read the status register to confirm that the value of the QE bit is 1.
The quad enable (QE) bit is a non-volatile bit. Once quad mode has been enabled, it is necessary to run R_FLASH_SPI_Quad_Disable() to disable quad mode.
When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing.
R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.
See figure 3.5 for details.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t      ret    = FLASH_SPI_SUCCESS;
uint32_t                loop_cnt = 0;
flash_spi_poll_mode_t   mode;

ret = R_FLASH_SPI_Quad_Enable(FLASH_SPI_DEV0);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
  if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);      /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

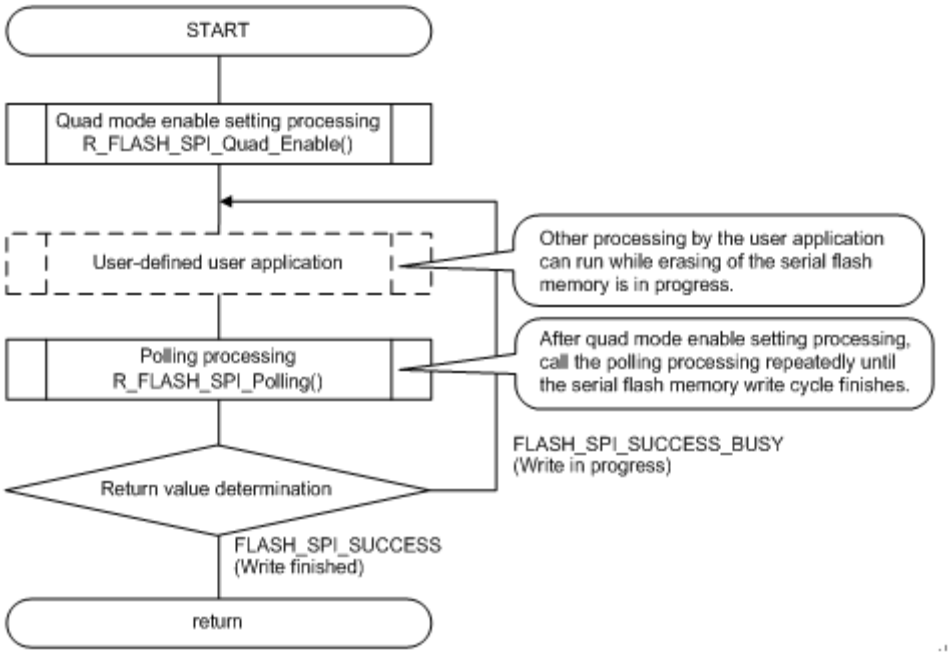


Figure 3.5 R_FLASH_SPI_Quad_Enable() Processing Example

R_FLASH_SPI_Quad_Disable()

This function is used to disable quad mode. It is a dedicated API function for MX25L, MX66L, or MX25R family serial NOR flash memory of Macronix International Co., Ltd and AT25QF family serial NOR flash memory of Dialog Semiconductor Plc.

Format

```
flash_spi_status_t R_FLASH_SPI_Quad_Disable(
    uint8_t devno
)
```

Parameters

devno

Device number (0, 1)

Return Values

```
FLASH_SPI_SUCCESS       /* Successful operation */
FLASH_SPI_ERR_PARAM     /* Parameter error */
FLASH_SPI_ERR_HARD      /* Hardware error */
FLASH_SPI_ERR_OTHER     /* Other task has acquired clock synchronous single
                           master control software resources, or other error */
```

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Clears the quad enable (QE) bit in the status register to 0 to cancel quad mode.
After processing finishes, read the status register to confirm that the value of the QE bit is 0.
The quad enable (QE) bit is a non-volatile bit. Once quad mode has been enabled, it is necessary to run this user API function to disable quad mode.
When this user API function completes successfully, the serial flash memory transitions to a write cycle. Do not fail to confirm write completion with R_FLASH_SPI_Polling(). If the next read or write processing starts when a previous write cycle is in progress, the serial flash memory will not accept the new processing.
R_FLASH_SPI_Polling() can be called at any time specified by the user. This allows a user application to perform other processing while a write cycle is in progress.
See figure 3.6 for details.

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t      ret    = FLASH_SPI_SUCCESS;
uint32_t                loop_cnt = 0;
flash_spi_poll_mode_t   mode;

ret = R_FLASH_SPI_Quad_Disable(FLASH_SPI_DEV0);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
  if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);      /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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.

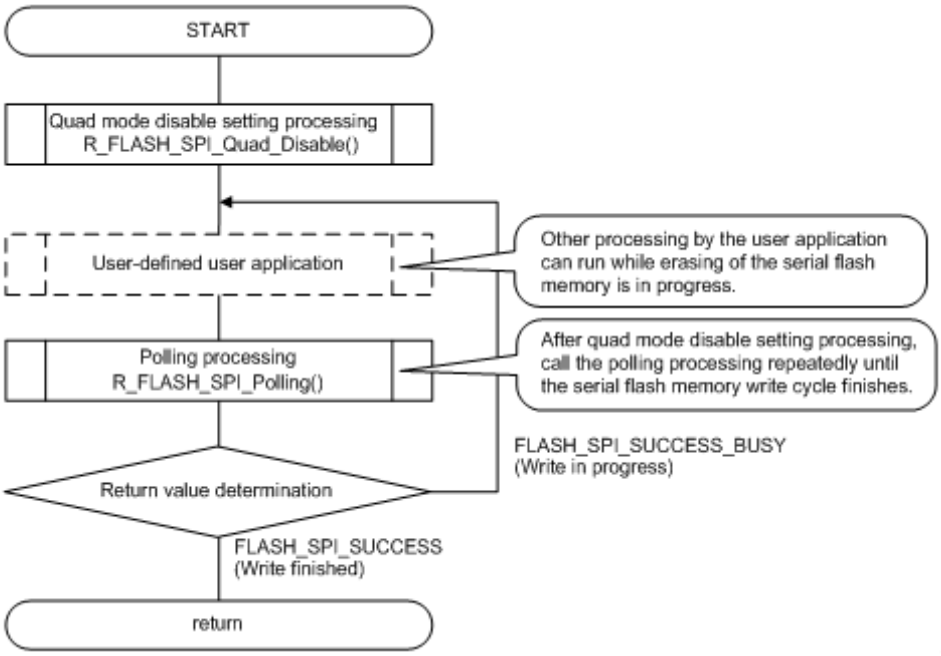


Figure 3.6 R_FLASH_SPI_Quad_Disable() Processing Example

R_FLASH_SPI_GetVersion()

This function is used to fetch the serial flash memory version information.

Format
uint32_t R_FLASH_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_flash_spi_if.h.

Description
Returns the version information.

Example
```
uint32_t version;
version = R_FLASH_SPI_GetVersion();
```

Special Notes
None

R_FLASH_SPI_Set_LogHdlAddress()

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

Format

flash_spi_status_t R_FLASH_SPI_Set_LogHdlAddress(
    uint32_t user_long_que
)

Parameters

user_long_que
Specify the handler address of the LONGQ FIT module.

Return Values

FLASH_SPI_SUCCESS */* Successful operation */*

Properties

Prototype declarations are contained in r_flash_spi_if.h.

Description

Sets the handler address of the LONGQ FIT module in the serial flash memory 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_FLASH_SPI_Open().

Example

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

flash_spi_status_t ret = FLASH_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_FLASH_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 FLASH_SPI_CFG_LONGQ_ENABLE in r_flash_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.

R_FLASH_SPI_Log()

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

Format

```
uint32_t R_FLASH_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_flash_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)

flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_W;
uint32_t             buf1[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Flash_Info_W.addr    = 0;
Flash_Info_W.cnt     = 32;
Flash_Info_W.p_data  = (uint8_t *)&buf1[0];
ret = R_FLASH_SPI_Write_Data_Page(FLASH_SPI_DEV0, &Flash_Info_W);
if (FLASH_SPI_SUCCESS != ret)
{
    /* Set last error log to buffer. */
    R_FLASH_SPI_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
    R_FLASH_SPI_Close(FLASH_SPI_DEV0);
}
```

Special Notes

Incorporate the LONGQ FIT module separately.
Enable the option #define FLASH_SPI_CFG_LONGQ_ENABLE in r_flash_spi_config.h. Also, enable #define xxx_LONGQ_ENABLE in the clock synchronous single master control software used by the specified device.

R_FLASH_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_FLASH_SPI_1ms_Interval(void)

Parameters
None

Return Values
None

Properties
Prototype declarations are contained in r_flash_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_FLASH_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. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

4.1 rx65n_rsk_flash_spi_sample, rx65n_rsk_flash_spi_sample_gcc

This is the sample application of FLASH SPI FIT module written for the Renesas RSKRX65N board. The program demonstrates how to use the APIs to control and use MX25L serial flash memory as a slave through clock synchronous single master control software is QSPI FIT module, operation as a master device.

Setup and Execution

1. Ensure driver support for channel 0 is enabled in r_qspi_smstr_rx_config.h:
   #define QSPI_SMSTR_CFG_CH0_INCLUDED
2. Build and download this sample application to the RSK board using the e2studio debugger.
3. Select the Renesas Virtual Debug Console view in e2studio to view printf information.
4. Run the application in the debugger.
5. Observe the version number of FLASH SPI module and ID of the serial flash memory print in the debug console window.
6. "Success!" is displayed in the debug console window.
7. If one of the operations fails "Failed." is displayed in the debug console window

Boards Supported

RSKRX65N

4.2 rx671_ek_flash_spi_sample, rx671_ek_flash_spi_sample_gcc

This is the sample application of FLASH SPI FIT module written for the Renesas EKRX671 board. The program demonstrates how to use the APIs to control and use AT25QF64 serial flash memory as a slave through clock synchronous single master control software is QSPIX memory mapped mode, operation as a master device.

Setup and Execution

1. Ensure driver support for channel 0 is enabled in r_qspix_rx_config.h:
   #define QSPIX_CFG_USE_CH0 (1)
2. Build and download this sample application to the EK board using the e2studio debugger.
3. Select the Renesas Virtual Debug Console view in e2studio to view printf information.
4. Run the application in the debugger.
5. Observe the version number of FLASH SPI module and ID of the serial flash memory print in the debug console window.
6. "Success!" is displayed in the debug console window.
7. If one of the operations fails "Failed." is displayed in the debug console window.

Boards Supported

EKRX671

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the Flash SPI FIT module.

Table 5.1 Confirmed Operation Environment (Rev.3.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.3.0
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
Endian	Big endian/little endian
Revision of the module	Rev.3.00
Board used	Renesas Starter Kit for RX113 (product No.: R0K505113xxxxxx) Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx) Renesas Starter Kit+ for RX64M (product No.: R0K50564Mxxxxxx) Renesas Starter Kit for RX71M (product No.: R0K50571Mxxxxxx)

Table 5.2 Confirmed Operation Environment (Rev.3.01)

Item	Contents
Integrated development environment	Renesas Electronics e^2 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.
Endian	Big endian/little endian
Revision of the module	Rev.3.01
Board used	Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx)

Table 5.3 Confirmed Operation Environment (Rev.3.02)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2020-07 IAR Embedded Workbench for Renesas RX 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.
Endian	Big endian/little endian
Revision of the module	Rev.3.02
Board used	Renesas Starter Kit+ for RX72N (product number.RTK5572Nxxxxxxxxxx)

Table 5.4 Confirmed Operation Environment (Rev.3.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2021-07 IAR Embedded Workbench for Renesas RX 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.
Endian	Big endian/little endian
Revision of the module	Rev.3.03
Board used	Renesas Starter Kit+ for RX671 (product number.RTK55671xxxxxxxxxx)

Table 5.5 Confirmed Operation Environment (Rev.3.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2022-04 IAR Embedded Workbench for Renesas RX 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 8.03.00.202204 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.
Endian	Big endian/little endian
Revision of the module	Rev.3.10
Board used	Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx)

Table 5.6 Confirmed Operation Environment (Rev.3.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2023-01 IAR Embedded Workbench for Renesas RX 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 8.03.00.202202 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.
Endian	Big endian/little endian
Revision of the module	Rev.3.20
Board used	Evaluation Kit+ for RX671 (product number.RTK5EK671xxxxxxxxxx)

Table 5.7 Confirmed Operation Environment (Rev.3.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2023-04 IAR Embedded Workbench for Renesas RX 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.
Endian	Big endian/little endian
Revision of the module	Rev.3.30
Board used	Evaluation Kit+ for RX671 (product number.RTK5EK671xxxxxxxxxx) Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx)

6. Reference Documents

User’s Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools
RX Family 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
2.32	Jan 18, 2016	—	First edition issued
2.33	Feb 02, 2016	1	In Target Device. Added “RX Family microcontrollers”. Added “RX130”, “RX23T” and “RX24T”.
		26	Updated contents in 2.9 Adding the Driver to Your Project. Deleted “Using” in title of application notes.
2.34	Jul 31, 2017	19	Deleted r_cgc_rx in 2.2 Software Requirements.
3.00	Feb.20,2019	1	Added “RX72T”.
		6-9	Updated contents in 1.2.2 Operating Environment and Memory Sizes.
		10	In 1.3.1FIT Module–Related Application Notes. Deleted R01AN1914EJ Deleted R01AN2280EJ Added R01AN1827EJ Added R01AN1815EJ Added R01AN4548EJ
		16	Update contents in 1.5.3 Software Structure.
		17	Update contents in 1.5.4 Relationship Between Control Software and Clock Synchronous.
		20	2.2 Software Requirements Added r_memdrv_rx. Changed r_rspi_smstr_rx to r_rspi_rx.
		21-22	2.6 Compile Settings Deleted Marco. FLASH_SPI_CFG_DEVx_DRVIF_CH_NO FLASH_SPI_CFG_DEVx_MODE FLASH_SPI_CFG_DEVx_DMAC_CH_NO_Tx FLASH_SPI_CFG_DEVx_DMAC_CH_NO_Rx FLASH_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Tx FLASH_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Rx FLASH_SPI_CFG_DEVx_BR FLASH_SPI_CFG_DEVx_BR_WRITE_DATA FLASH_SPI_CFG_DEVx_BR_READ_DATA
		25	Update contents in 2.9 Adding the Driver to Your Project.
3.01	May.20,2019	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		2	Added Target Compilers.
		2	Deleted R01AN1723 and R01AN1826 from Related Documents.
		6	Changed 1.2 Overview and Memory Size of APIs to 1.2 Overview of APIs. 1.2.2 Operating Environment and Memory Sizes, deleted.
		16	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		20-21	2.8 Code Size, added.
		26	2.13 “for”, “while” and “do while” statements: added
		33	3.4 R_FLASH_SPI_Set_Write_Protect(), fixed Example.

Rev.	Date	Description	
		Page	Summary
3.01	May.20,2019	39	3.7 R_FLASH_SPI_Write_Data_Page(), fixed Example.
		42	3.8 R_FLASH_SPI_Erase(), fixed Example.
		51	3.13 R_FLASH_SPI_Write_Configuration(), fixed Example.
		57	3.16 R_FLASH_SPI_Quad_Enable(), fixed Example.
		60	3.17 R_FLASH_SPI_Quad_Disable(), fixed Example.
		66	4.1 Confirmed Operation Environment, added.
3.02	Dec.10,2020	1	Added “RX72N”.
		21	2.8 Code Size, amended.
		24	Changed Section 2.10 Adding the Driver to Your Project
		28-66	Deleted "Reentrancy" item on the API description page.
		68	Added Table 4.3 Confirmed Operation Environment (Rev.3.02).
		69	Changed Section 5 Reference Documents.
		Program	FLASH_SPI FIT module fixed due to software failure. Description: A warning and linkage errors arise during building when using GPIO module firmware integration technology and MPC module firmware integration technology. Conditions: 1. Use the integrated development environment CS+. 2. Serial Flash memory FIT module general-purpose I/O port control is performed by both of the following FIT modules. GPIO module Firmware Integration Technology MPC module Firmware Integration Technology Corrective action: Please use FLASH_SPI FIT module Rev3.02. Corresponding tool news number: R20TS0609
		Program	FLASH_SPI FIT module fixed due to software failure. Description: When setting the device capacity of r_flash_spi to 1G-bit in the SC component, a build error occurs. Conditions: Set the device capacity of r_flash_spi to 1G-bit in the SC component and build. Corrective action: Please use FLASH_SPI FIT module Rev3.02.

Rev.	Date	Description	
		Page	Summary
3.02	Dec.10,2020	Program	FLASH_SPI FIT module fixed due to software failure. Description: On RX72M/RX72N/RX66N, if the device port of r_flash_spi_pin_config.h is set to "H", "K", "M", "N", or "Q", a build error will occur. Conditions: Set FLASH_SPI_CS_DEV0_CFG_PORTNO or FLASH_SPI_CS_DEV1_CFG_PORTNO to one of "H", "K", "M", "N", and "Q" to build. Corrective action: Please use FLASH_SPI FIT module Rev3.02.
3.03	Nov.30,2021	1	Added “RX671 Group (QSPIX)”.
		5	Added “QSPIX FIT module” in Overview.
		7	Update contents in 1.3.1 FIT Module–Related Application Notes.
		7	1.4 Using Serial Flash Memory Module, added.
		11-16	Modified the picture format.
		17	Added r_qspix_rx in 2.2 Software Requirements.
		21-22	2.8 Code Size, amended.
		68	Added Table 4.4 Confirmed Operation Environment (Rev.3.03).
3.10	Jun.30,2022	19	2.6 Compile Settings Added new macros #define FLASH_SPI_CS_DEVx_CFG_PORTNO #define FLASH_SPI_CS_DEVx_CFG_BITNO
		21	2.8 Updated FIT module version, and compilers’ version
		69	4.1 Confirmed Operation Environment: Added Table for Rev.3.10.
		Program	Added new macros to specify the ports used for SS# Fixed issues of wrong conditional expression in the if statement. Set PORTX as the default port assigned to SS#.
3.20	Mar.16,2023	1, 6, 20, 31-50, 71, 74	Added support for AT25QF641B-SHB.
		1, 5, 7, 17	Added RSCI FIT module.
		18	2.6 Compile Settings: Added new macros #define FLASH_SPI_CFG_DEVx_AT25QF
		21	2.8 Updated FIT module version, and compilers’ version.
		32-35, 58-63, 67-70	3. API Functions: Added new API functions R_FLASH_SPI_Read_Status2() R_FLASH_SPI_Read_Status3() R_FLASH_SPI_Write_Status() R_FLASH_SPI_Write_Status2() R_FLASH_SPI_Write_Status3() R_FLASH_SPI_Read_Data_Security_Page() R_FLASH_SPI_Write_Data_Security_Page()

Rev.	Date	Description	
		Page	Summary
3.20	Mar.16,2023	42	3. API Functions: amended Added notes for R_FLASH_SPI_Read_Data
		83	4.1 Confirmed Operation Environment: Added Table for Rev.3.20.
		Program	Added support for AT25QF641B-SHB with Indirect Access Mode of QSPIX Added support for RSCI and QSPIX Memory Mapped Mode. Removed the processing related to other unsupported flash devices.
3.30	Jun.15,2023	13	Added " AT25QF family" in 1.6.3 Structure Software
		24	Deleted the description of FIT configurator from "2.10 Adding the Driver to Your Project"
		81	Added "4. Demo Project"
		85	5.1 Confirmed Operation Environment: Added Table for Rev.3.30.
		Program	Deleted the description of FIT configurator. Updated and added new demo project

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.