RX Family

QSPI Clock Synchronous Single Master Control Module Using Firmware Integration Technology

Introduction

This application note describes a clock synchronous single master control module, which uses clock synchronous serial communication over the quad serial peripheral interface (QSPI) on RX Family microcontrollers, and explains its use. The module is a clock synchronous single master control module using Firmware Integration Technology (FIT). It is referred to below as the QSPI FIT module. Other similar function control modules using FIT are referred to as FIT modules or as “function name” FIT modules.

SPI/QSPI mode single master control can be enabled by adding slave device selection control by means of port control.

The QSPI FIT module implements single master basic control. Use the QSPI FIT module to create software for controlling slave devices.

Target Devices

Supported microcontroller

RX64M Group, RX65N Group, RX66N Group,
RX71M Group, RX72M Group, RX72N Group

Device on which operation has been confirmed

Renesas Electronics R1EX25xxx Series Serial EEPROM, 16 Kbit
Macronix International MX25/66L family serial NOR flash memory, 32 Mbit

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.

Note that the expression “RX Family microcontroller” is used in the discussion that follows for convenience as the target devices span multiple product groups.

Target Compilers

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

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

Related Documents

The applications notes that are related to this application note are listed below. Reference should also be made to those application notes.

- Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family LONGQ Module Using Firmware Integration Technology (R01AN1880)
- RX Family DMA Controller DMACA Control Module Using Firmware Integration Technology(R01AN2063)
- RX Family DTC Module Using Firmware Integration Technology(R01AN1819)
- RX Family Compare Match Timer Module Using Firmware Integration Technology (R01AN1856)
- RX Family EEPROM Access Clock Synchronous control module Using Firmware Integration Technology (R01AN2325)
- RX Family General Purpose Input/Output Driver Module Using Firmware Integration Technology (R01AN1721)
- RX Family Multi-Function Pin Controller Module Using Firmware Integration Technology (R01AN1724)

Contents

1. Overview

1.1 QSPI FIT Module

The QSPI FIT module can be combined with other FIT modules for easy integration into the target system.

The functions of the QSPI FIT module can be incorporated into software programs by means of APIs. For information on incorporating the QSPI FIT module into projects, see 2.12 Adding the FIT Module to Your Project, and 2.13 Peripheral Functions and Modules Other than QSPI.

1.2 Overview of QSPI FIT Module

The QSPI built into the RX Family microcontroller is used to implement clock synchronous control. QSPI mode single master control can be enabled by adding slave device selection control by means of port control.

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 for clock synchronous single master using the QSPI, with the RX Family microcontroller as the master device
- Support for Clock Synchronous Single-SPI, Dual-SPI, and Quad-SPI Modes
- The QSPI operates in clock synchronous serial communication mode. It can control one or more channels specified by the user.
- Reentrancy from a different channel is possible.
- Slave device selection control is unsupported.
  Slave device selection control must be implemented separately by means of port control.
- Operation with both big-endian and little-endian data order is supported.
- Data is transferred in MSB-first format.
- Only software transfers are supported.
  A separate DMAC or DTC transfer program is required to perform DMAC or DTC transfers.

Table 1-1 Peripheral Devices Used and Their Uses

Peripheral Device	Use
QSPI	Clock synchronous serial: 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). Not used by QSPI FIT module.

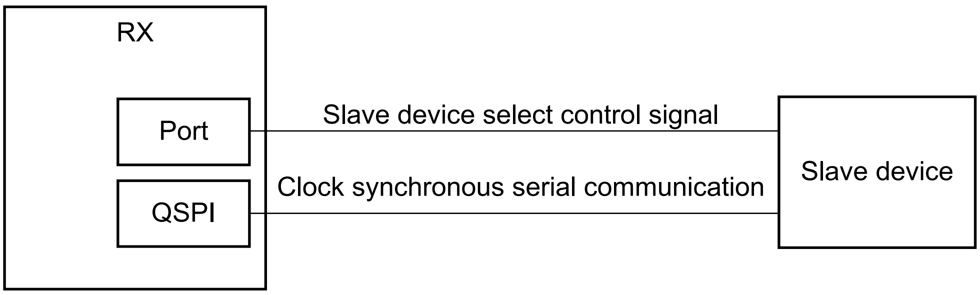


Figure 1-1 Sample Configuration

1.3 Overview of APIs

Table 1-2 lists the API functions of the QSPI FIT module.

Table 1-2 API Functions

Function Name	Description
R_QSPI_SMstr_Open()	Driver initialization processing
R_QSPI_SMstr_Close()	Driver end processing
R_QSPI_SMstr_Control()	Driver control (SPI mode and bit rate) setting processing
R_QSPI_SMstr_Write() *1	Single master transmit (single-SPI/dual-SPI/quad-SPI) processing
R_QSPI_SMstr_Read() *1	Single master receive (single-SPI/dual-SPI/quad-SPI) processing
R_QSPI_SMstr_Get_BuffRegAddress()	SPDR register address acquisition processing
R_QSPI_SMstr_Int_Spti_Ier_Clear()	SPTI transmit interrupt request disable processing
R_QSPI_SMstr_Int_Spri_Ier_Clear()	SPRI receive interrupt request disable processing
R_QSPI_SMstr_Int_Spti_Dmacdtc_Flag_Set()	DMAC/DTC transmit-end flag setting processing
R_QSPI_SMstr_Int_Spri_Dmacdtc_Flag_Set()	DMAC/DTC receive-end flag setting processing
R_QSPI_SMstr_GetVersion()	Driver version information acquisition processing
R_QSPI_SMstr_Set_LogHdlAddress()	LONGQ FIT module handler address setting processing
R_QSPI_SMstr_Log()	Error log acquisition processing using LONGQ FIT module
R_QSPI_SMstr_1ms_Interval() *2	Interval count processing

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

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

1.4 Example

1.4.1 Hardware Settings

(1) Hardware Configuration Example

Figure 1-2 is a connection diagram. To achieve high-speed operation, consider adding damping resistors or capacitors to improve the circuit matching of the various signal lines.

Do not fail to perform pull-up processing. Without pull-up processing, the slave device may enter the write protect state or hold state when a data line is in the Hi-Z state.

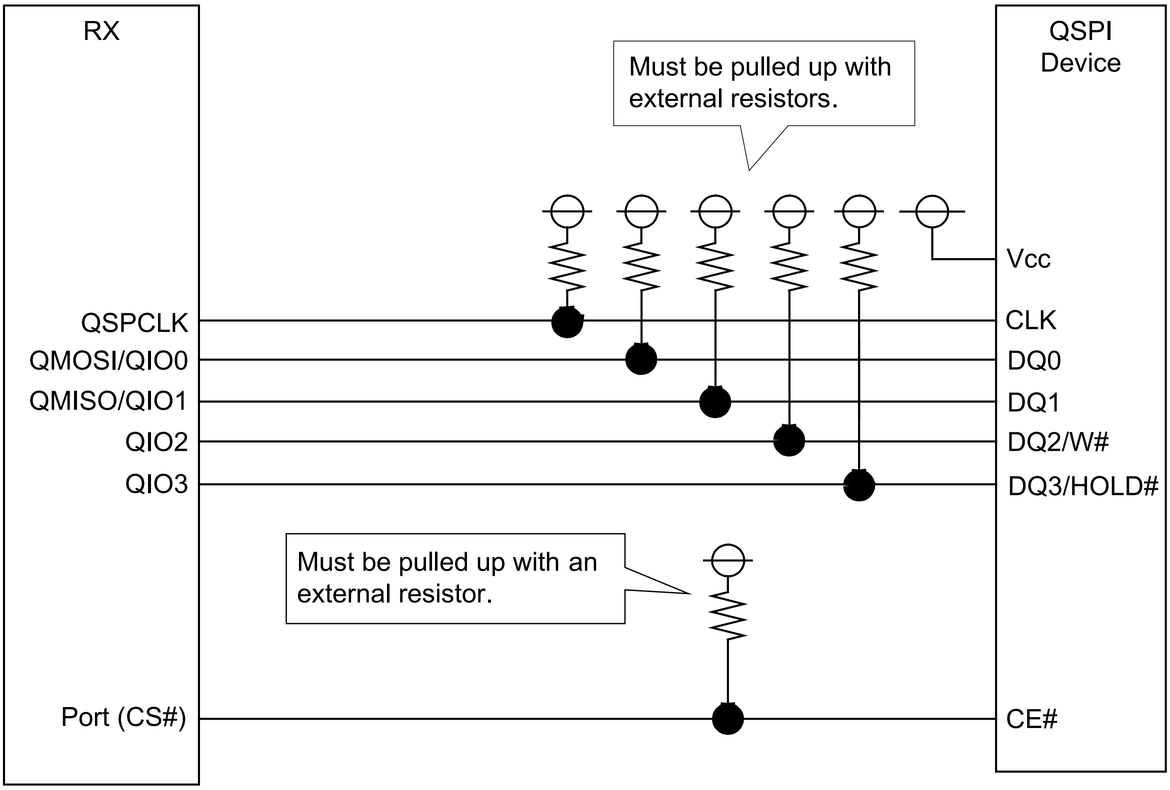


Figure 1-2 Sample Wiring Diagram for a RX Family MCU QSPI and a QSPI Slave Device

(2) List of Pins

Table 1-3 lists the pins that are used and their uses.

Table 1-3 List of Pins Used

Pin Name	I/O	Description
QSPCLK	Output	Clock output
QMOSI/QIO0*1	I/O	Master data output/Data 0
QMISO/QIO1*2	I/O	Master data input/Data 1
QIO2*3	I/O	Data 2
QIO3*3	I/O	Data 3
Figure2-1 Port(CS#)	Output	Slave device select output Not used by QSPI FIT module.

Notes: 1. Abbreviated below as QIO0.
2. Abbreviated below as QIO1.
3. You still need to assign pins when you are not using them. The pin cannot be used for other peripheral functions.

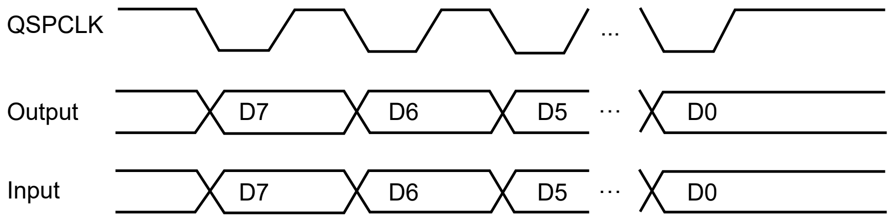
1.4.2 Software

(1) Operation Overview

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

(2) Controllable Slave Devices

Slave devices that support SPI mode 3 (CPOL = 1, CPHA = 1), illustrated in Figure 1-3, can be controlled by the module. The figure shows the polarity and phase of the input and output signals. It does not indicate the bus width.



- 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 QSPCLK pin is held high when no transfer processing is in progress.

Figure 1-3 Timing of Controllable Slave Devices

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

(3) Slave Device CE# Pin Control

The QSPI FIT module does not control the CE# pin of the slave device. To control a slave device, functionality to control the CE# pin of the slave device must be added separately.

Control is implemented by establishing a connection to the ports of the microcontroller and using the microcontroller's general port output for control.

In addition, it is necessary to provide a sufficient time interval (slave device CE# setup time) from the falling edge of the slave device's CE# (microcontroller port (CS#)) signal to the falling edge of the slave device's CLK (microcontroller QSPCLK) signal.

In like manner, it is necessary to provide a sufficient time interval (slave device CE# hold time) from the rising edge of the slave device's CLK (microcontroller QSPCLK) signal to the rising edge of the slave device's CE# (microcontroller port (CS#)).

Check the data sheet of the slave device and set the software wait time to match the system characteristics.

(4) Software Structure

Figure 1-4 shows the software structure.

Use the QSPI FIT module to create software for controlling slave devices.

Note that sample software for controlling slave devices is available for download.

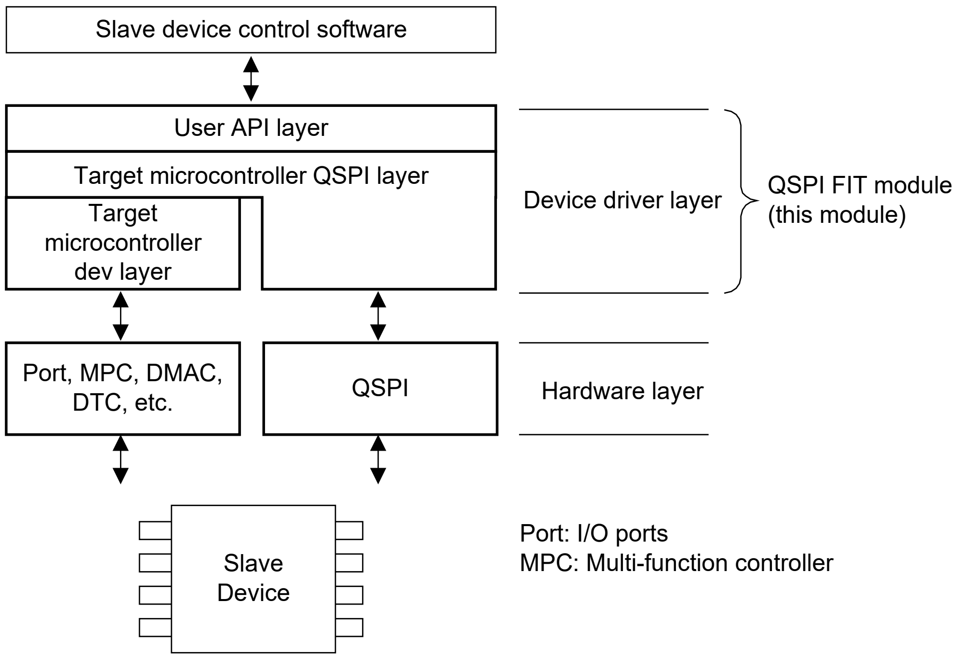


Figure 1-4 Software Structure

(a) User API layer (r_qspi_smstr.c)

This is the QSPI clock synchronous single master control segment, which is not dependent on the specifications of the microcontroller or the QSPI.

It also includes transfer start setting processing required for DMAC control or DTC control. It can be used in combination with the DMAC FIT module or DTC FIT module.

(b) Target microcontroller QSPI layer (r_qspi_smstr_target.c)

This is the QSPI resource control segment.

Separate versions are provided to accommodate different channel counts or QSPI specifications.

(c) Target microcontroller dev layer (r_qspi_smstr_target_dev_port.c)

This segment controls resources other than the QSPI.

This segment controls functions such as the I/O ports and multi-function pin controller. It must be incorporated into the system as necessary by using additional modules, etc.

(d) Control software for slave device

Sample code for controlling Renesas Electronics R1EX25xxx series serial EEPROM (R01AN2325) is provided as an example for reference. The SPI serial EEPROM Control Software has driver interface functions (r_eeprom_spi_drvif_devX.c: X=0 or 1) to incorporate the QSPI FIT module.

(5) Data Buffers and Transmit/Receive Data

The QSPI FIT module 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.

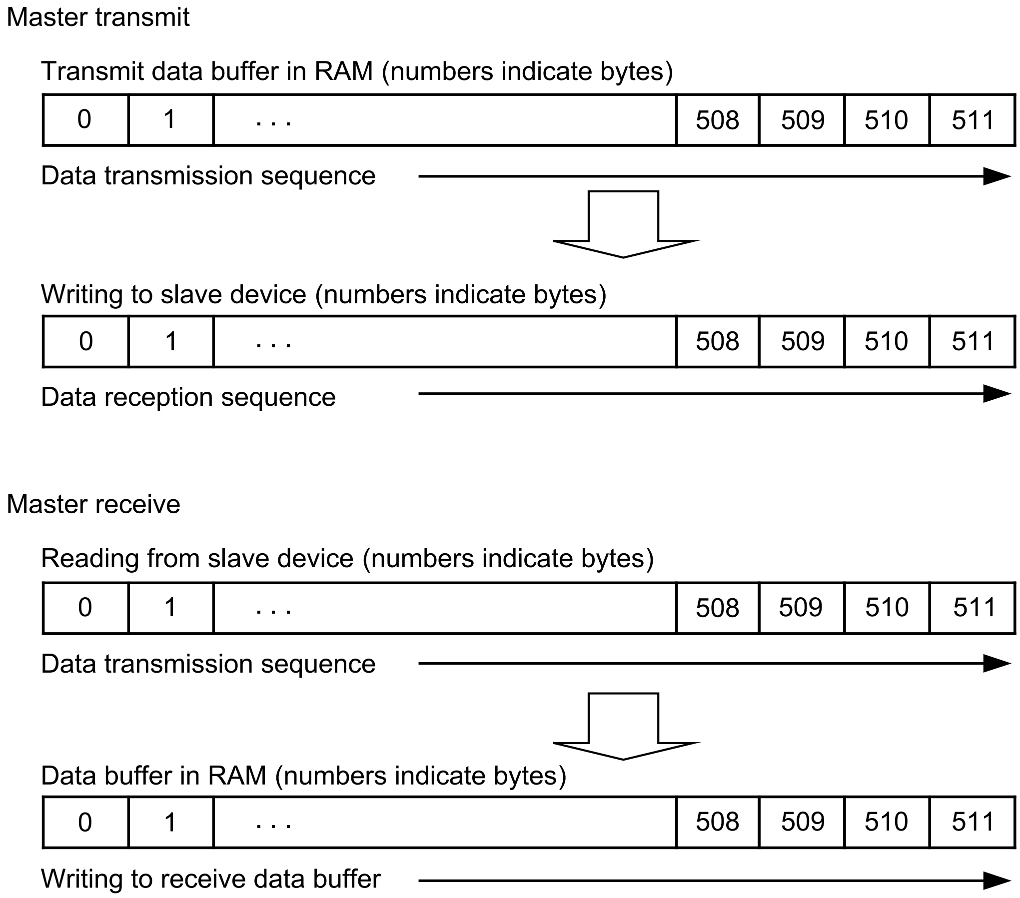


Figure 1-5 Data Buffers and Transmit/Receive Data

(6) Single Master Transmit Processing

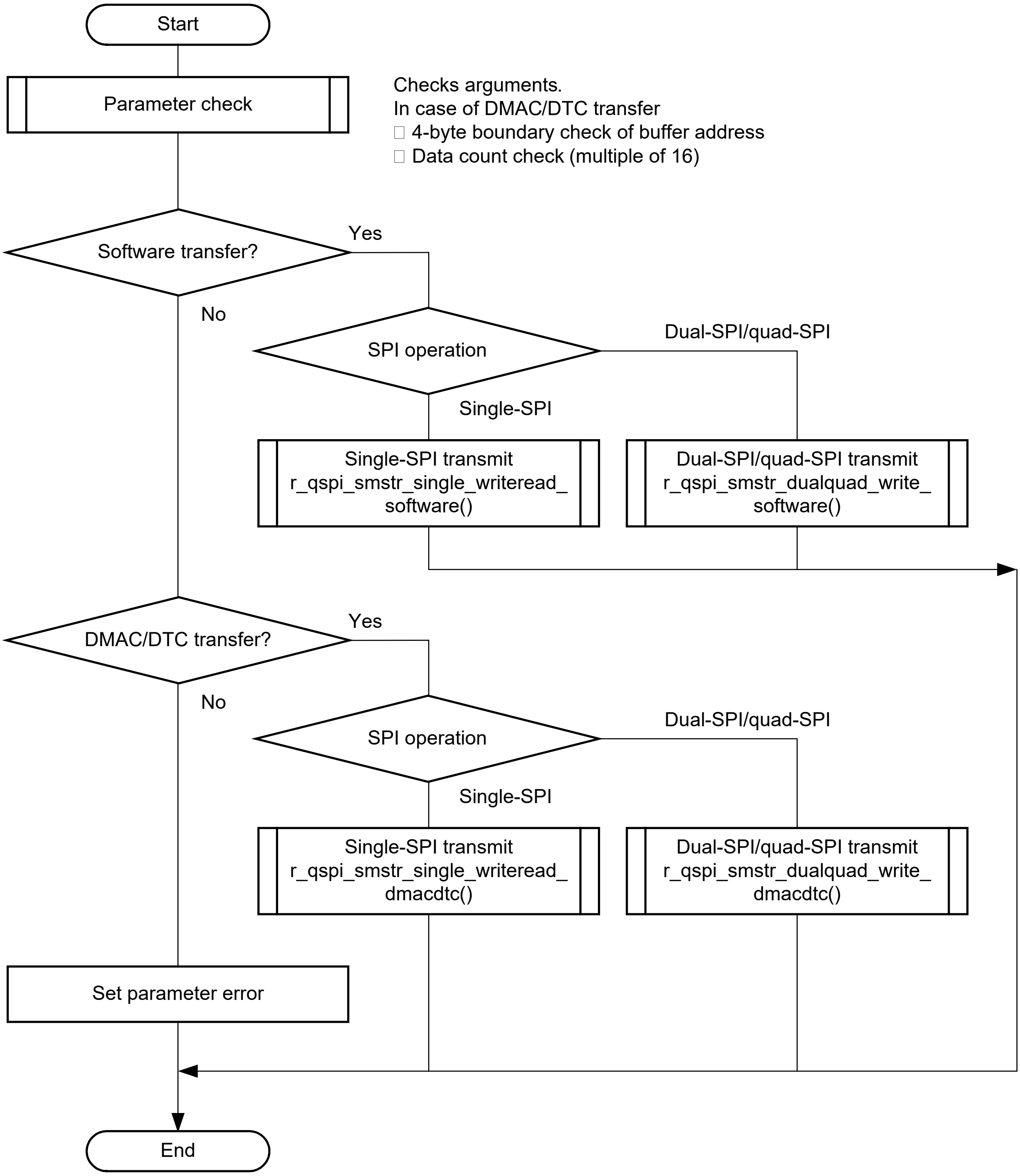


Figure 1-6 Data Transmit Processing

(e) Single-SPI Transmit Processing (Software)

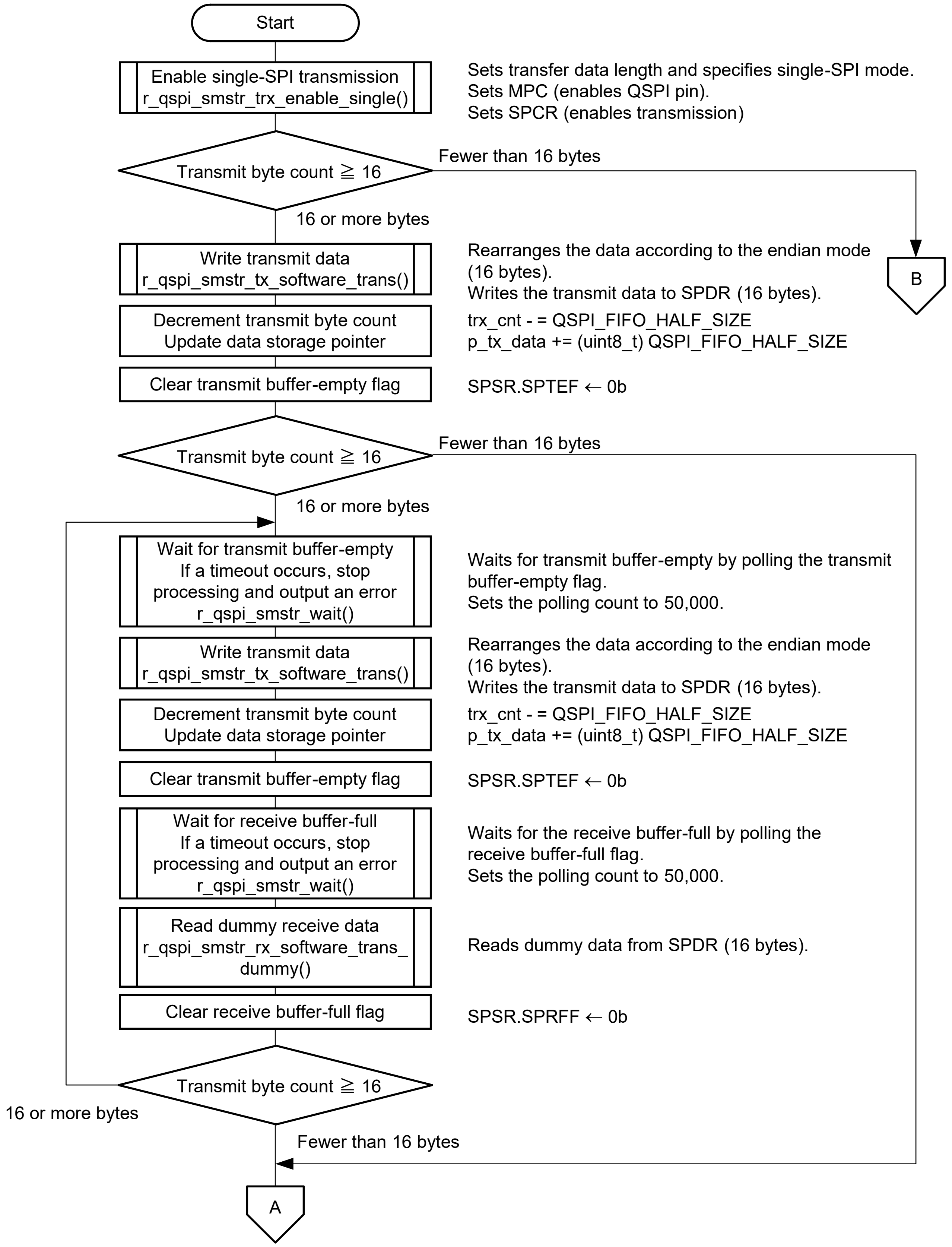


Figure 1-7 Single-SPI Transmit Processing 1 (Software)

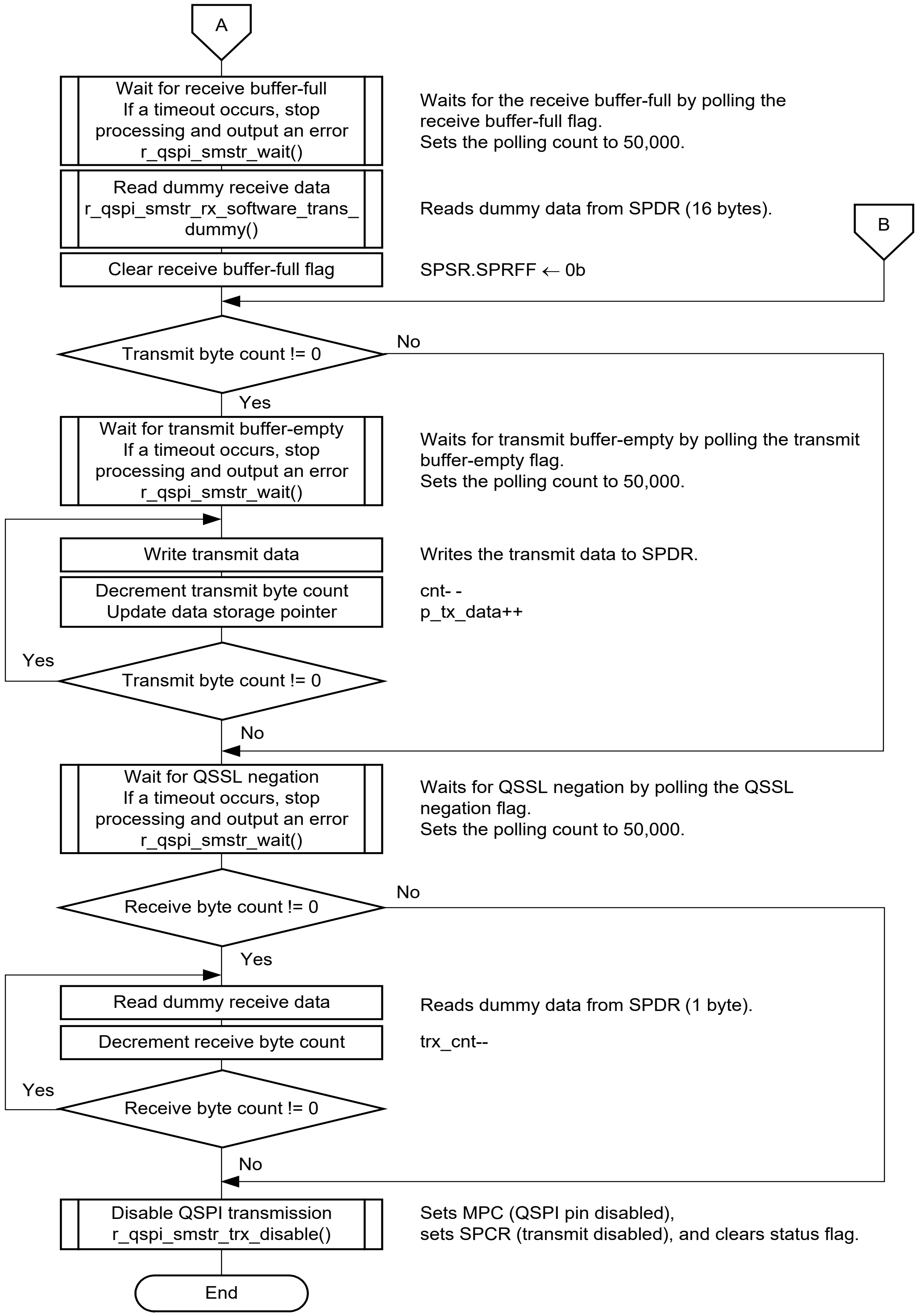


Figure 1-8 Single-SPI Transmit Processing 2 (Software)

(f) Dual-SPI/Quad-SPI Transmit Processing (Software)

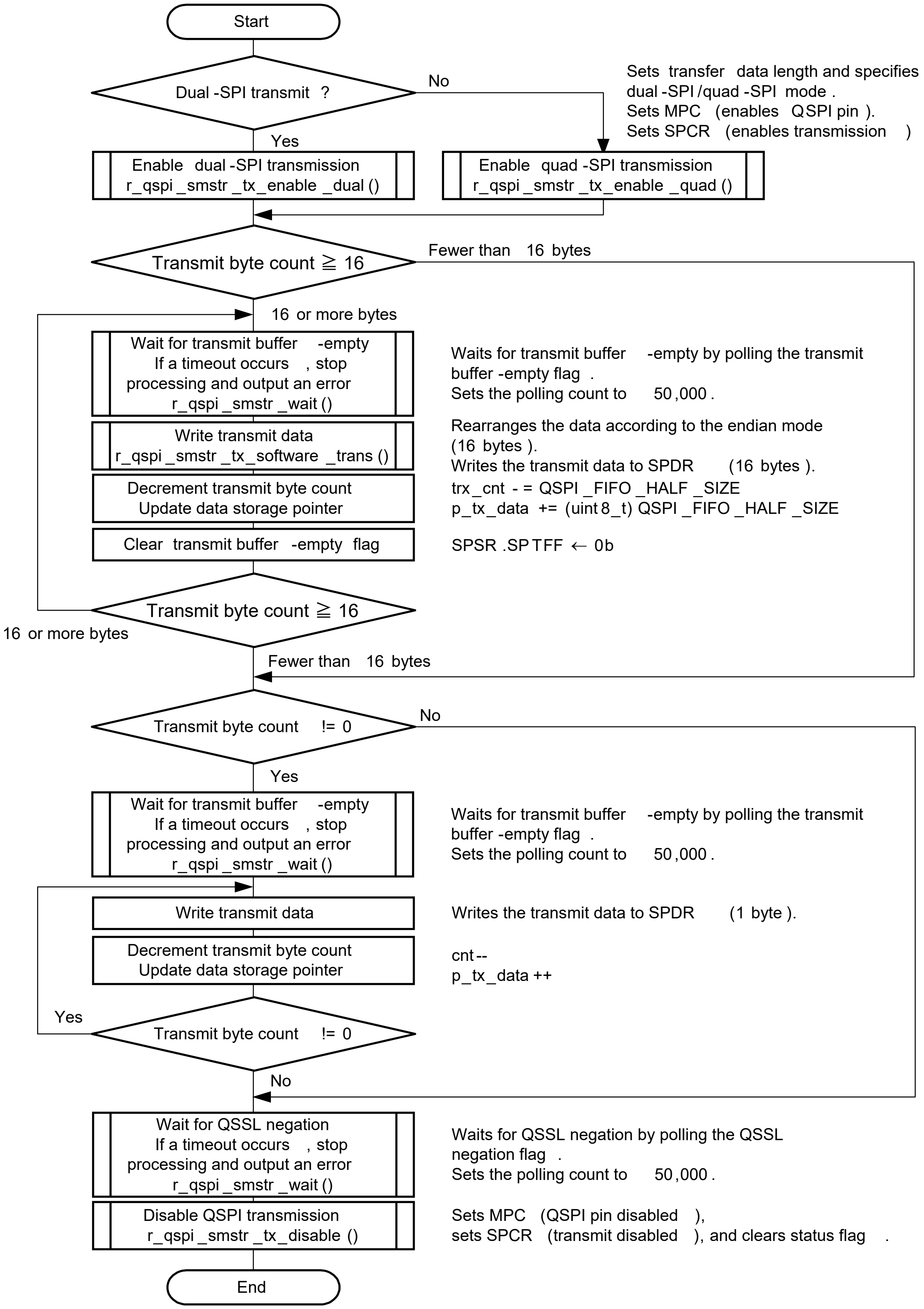


Figure 1-9 Dual-SPI Transmit Processing (Software)

(g) Single-SPI Transmit Processing (DMAC/DTC)

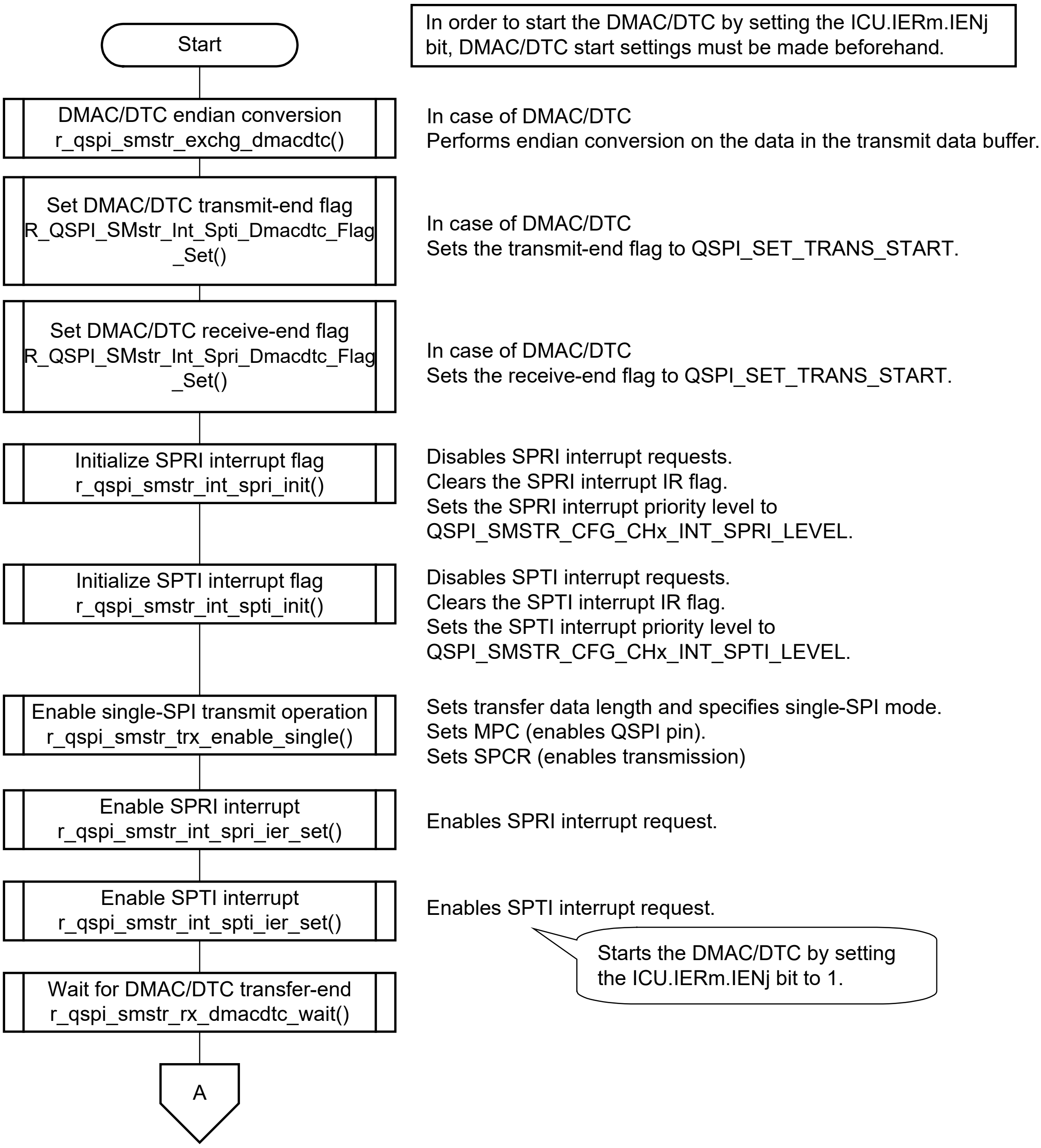


Figure 1-10 Single-SPI Transmit Processing 1 (DMAC/DTC)

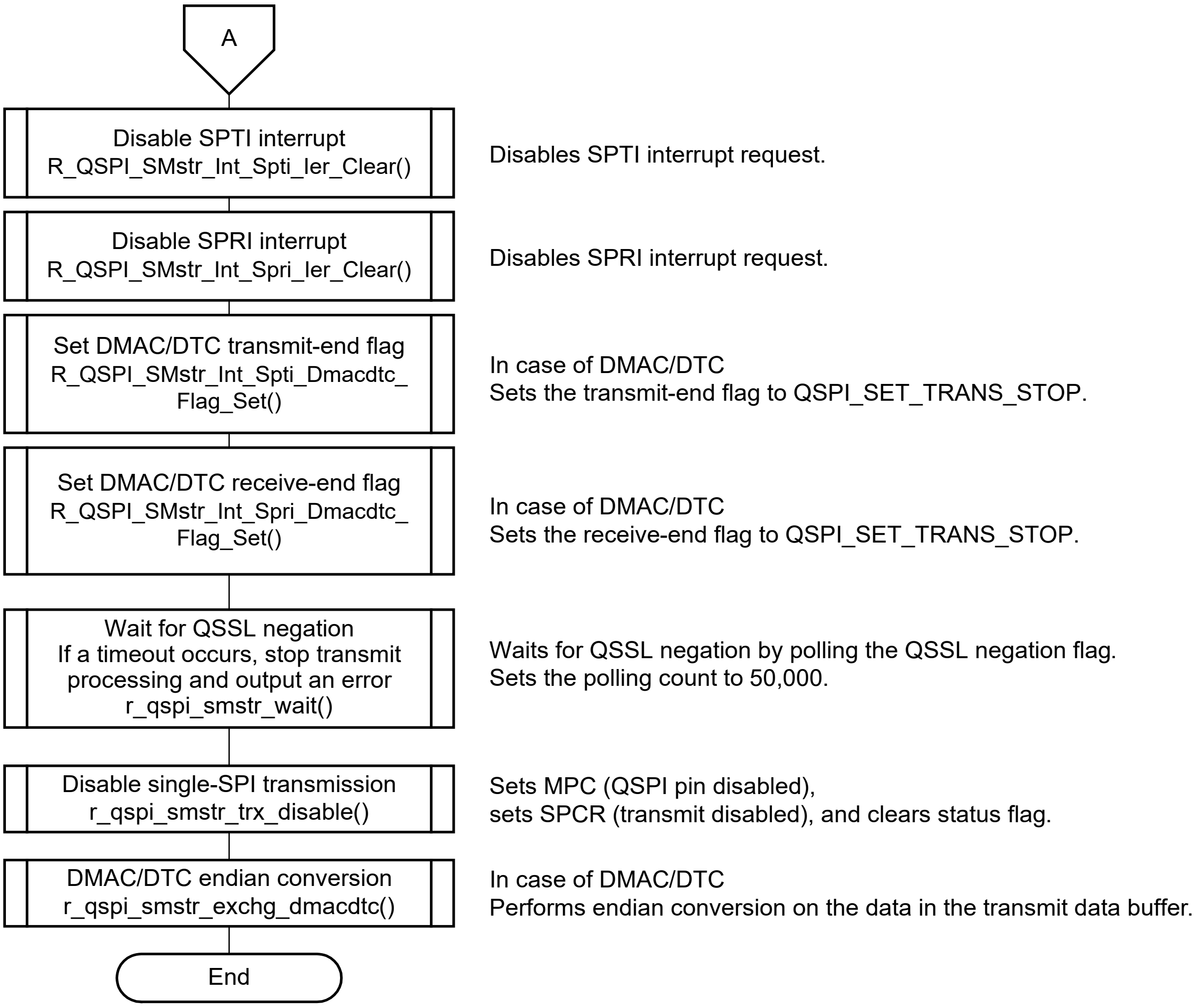


Figure 1-11 Single-SPI Transmit Processing 2 (DMAC/DTC)

(h) Dual-SPI/Quad-SPI Transmit Processing (DMAC/DTC)

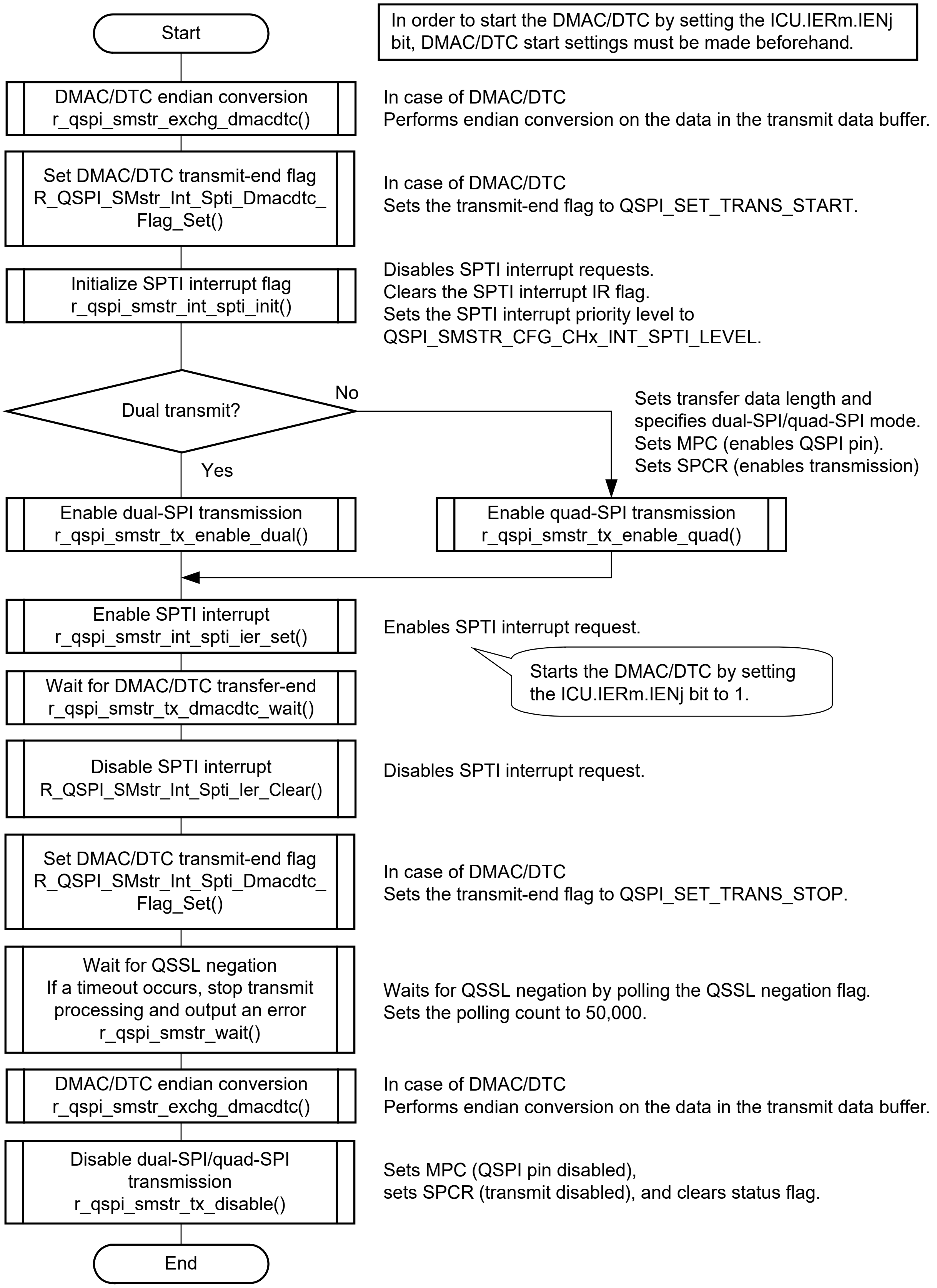


Figure 1-12 Dual-SPI/Quad-SPI Transmit Processing (DMAC/DTC)

(7) Single Master Receive Processing

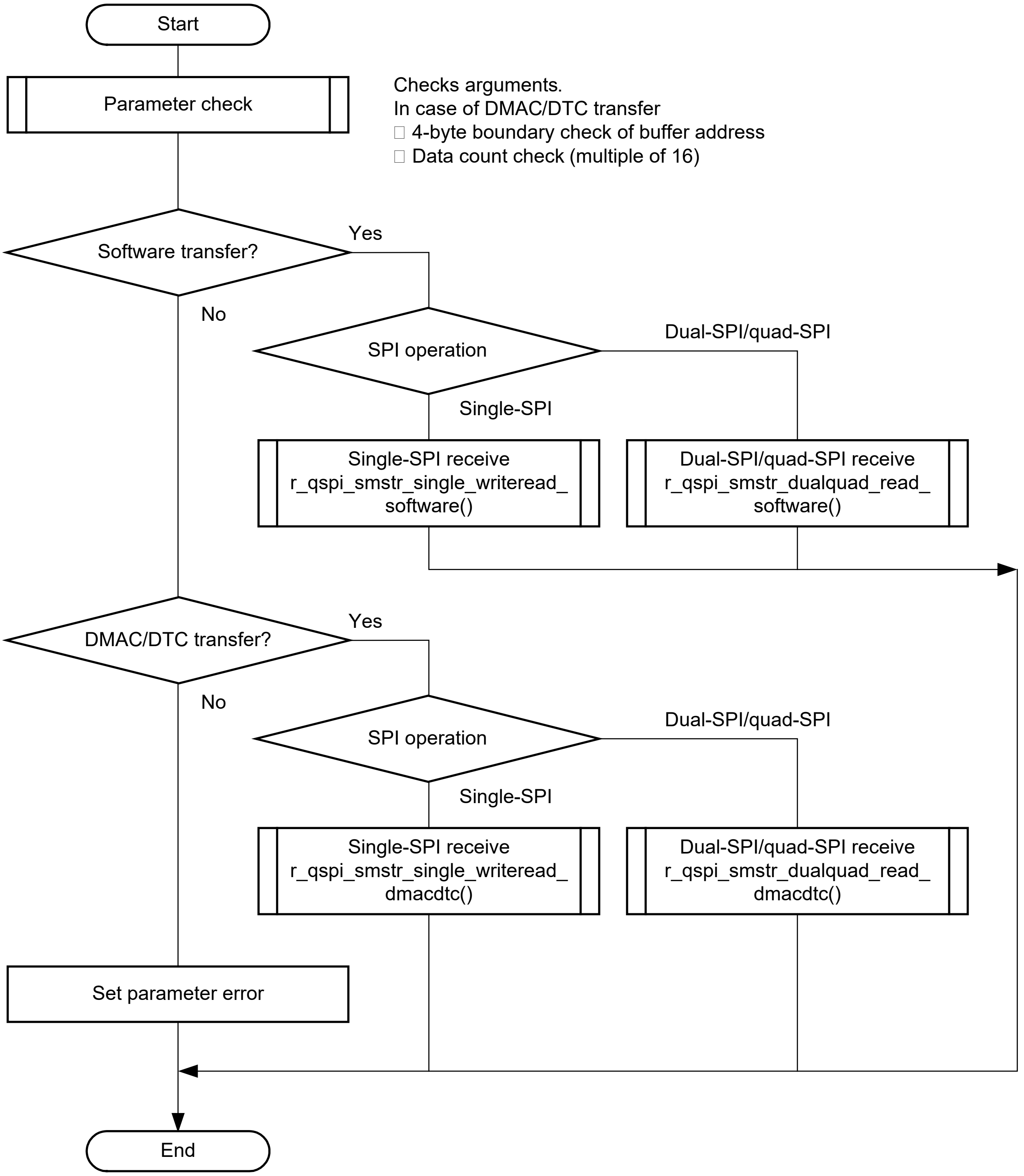


Figure 1-13 Data Receive Processing

(i) Single-SPI Receive Processing (Software)

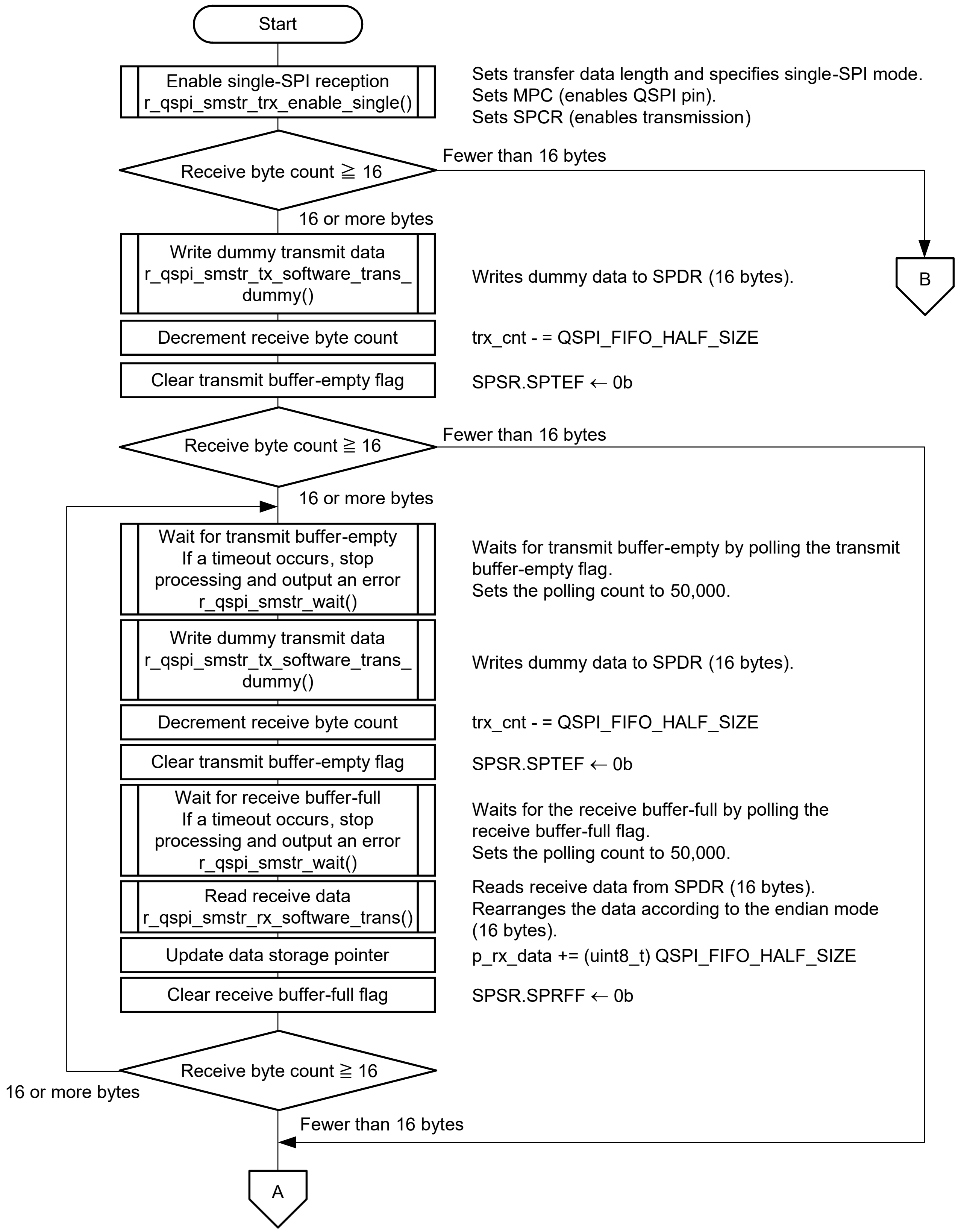


Figure 1-14 Single-SPI Receive Processing 1 (Software)

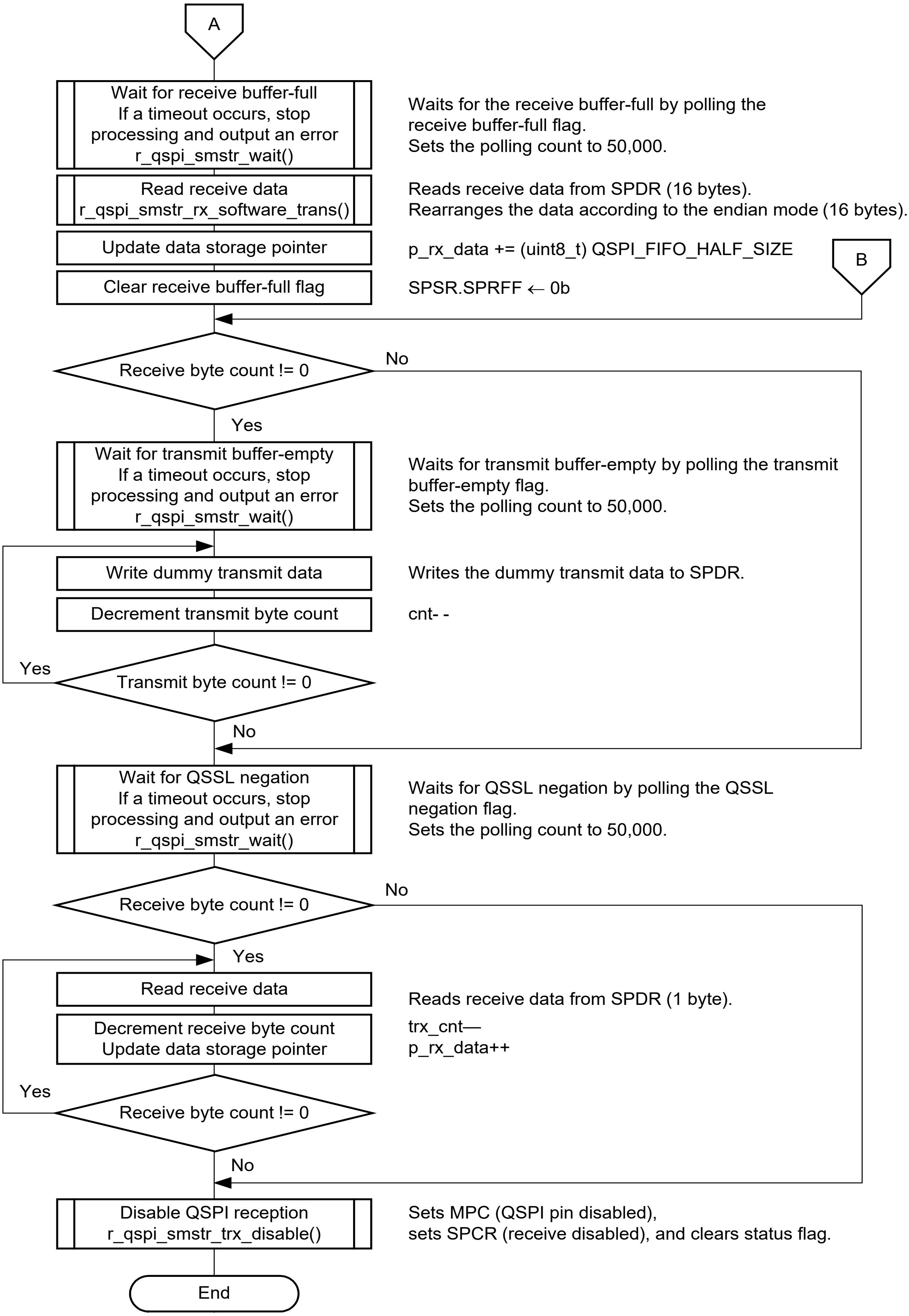


Figure 1-15 Single-SPI Receive Processing 2 (Software)

(j) Dual-SPI/Quad-SPI Receive Processing (Software)

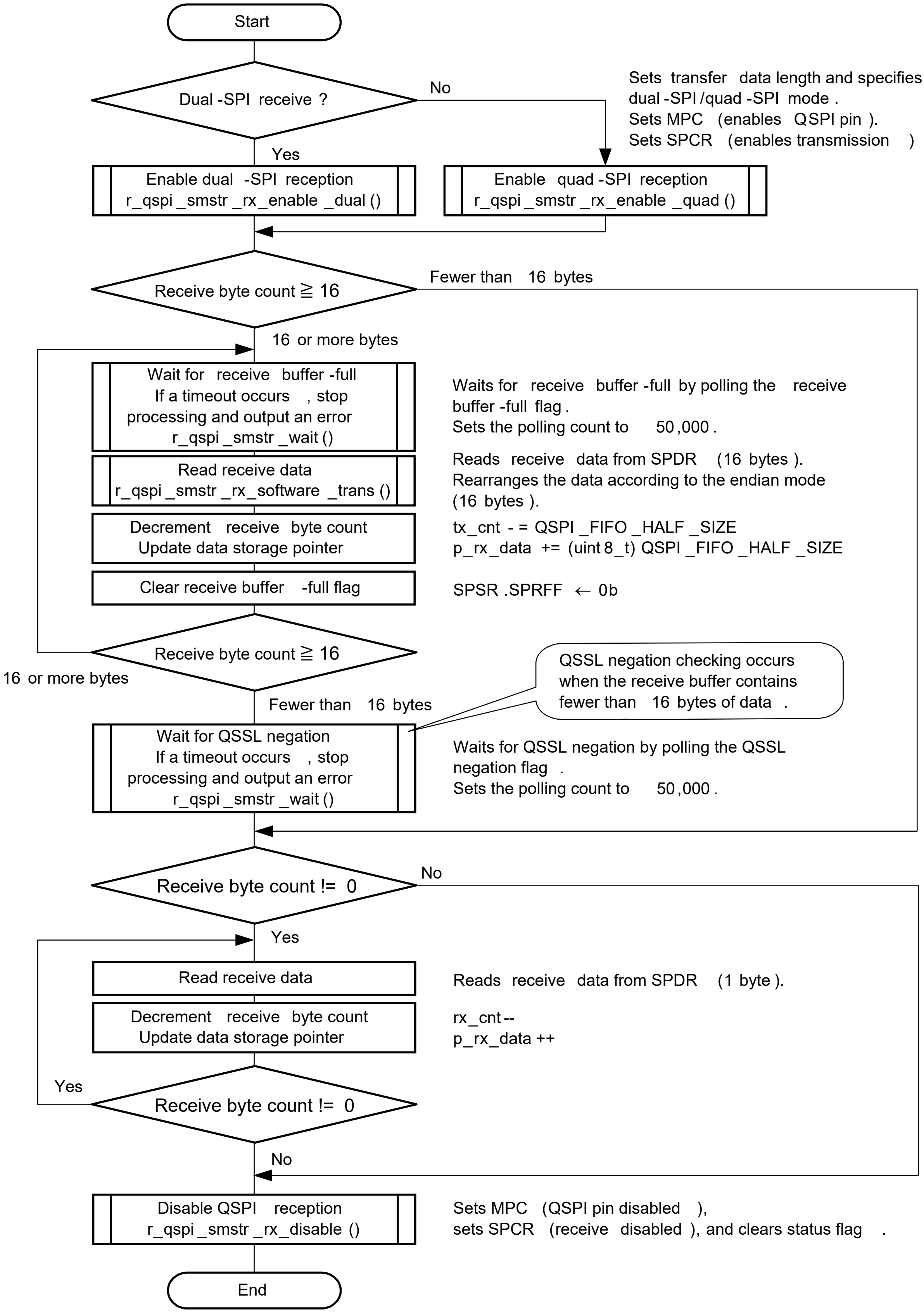


Figure 1-16 Dual-SPI/Quad-SPI Receive Processing (Software)

(k) Single-SPI Receive Processing (DMAC/DTC)

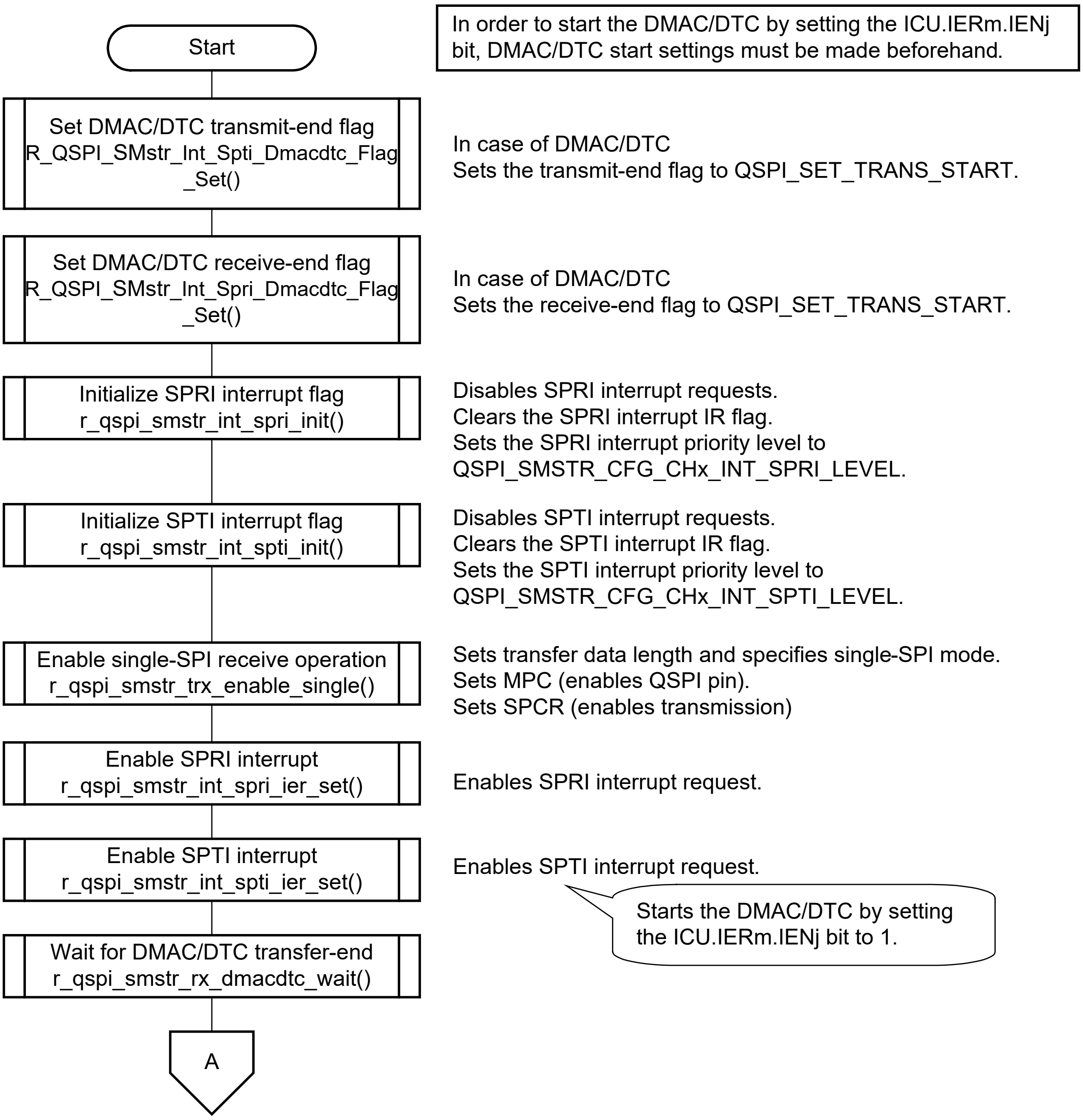


Figure 1-17 Single-SPI Receive Processing 1 (DMAC/DTC)

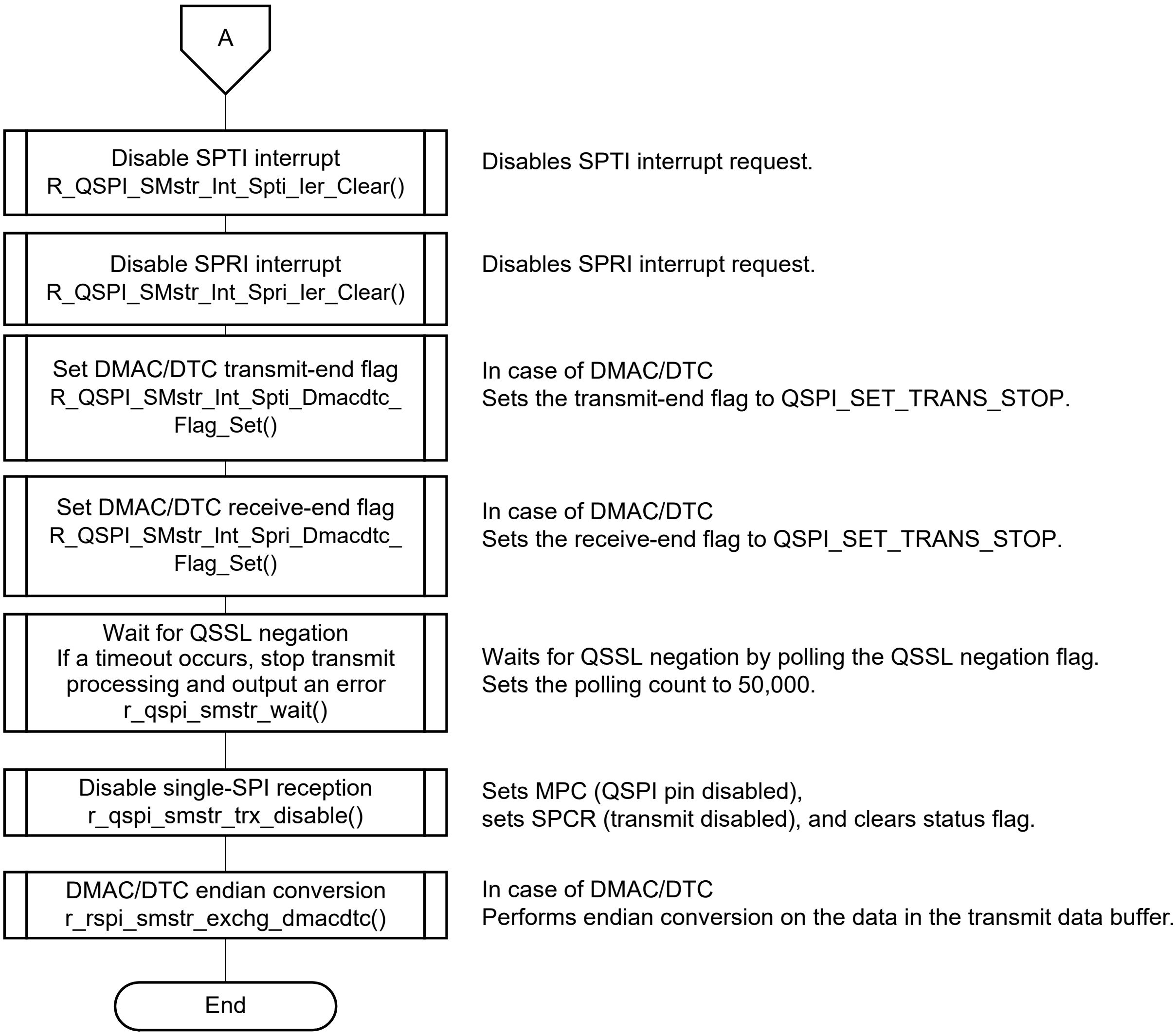


Figure 1-18 Single-SPI Receive Processing 2 (DMAC/DTC)

(l) Dual-SPI/Quad-SPI Receive Processing (DMAC/DTC)

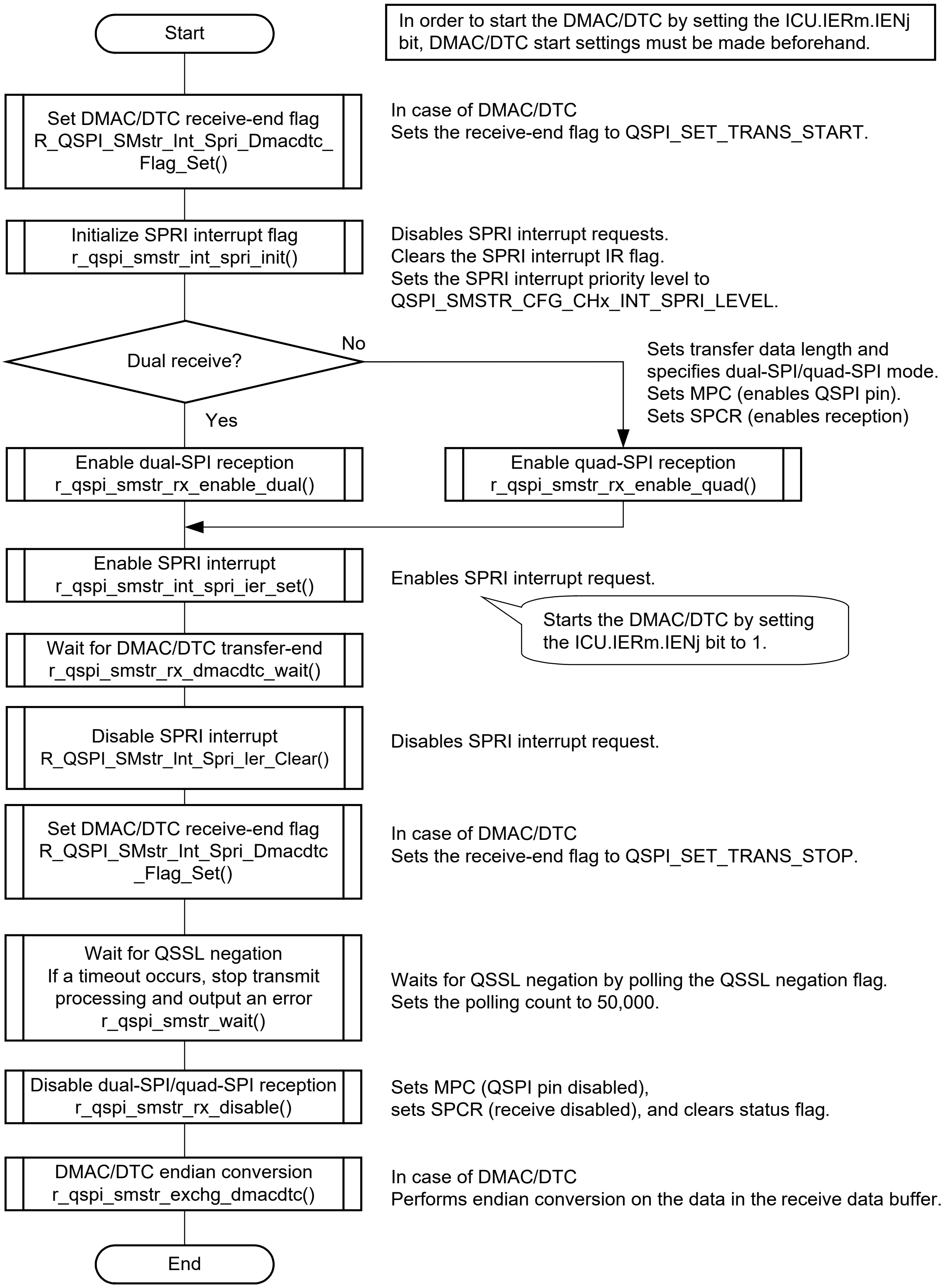


Figure 1-19 Dual-SPI/Quad-SPI Receive Processing (DMAC/DTC)

1.5 State Transition Diagram

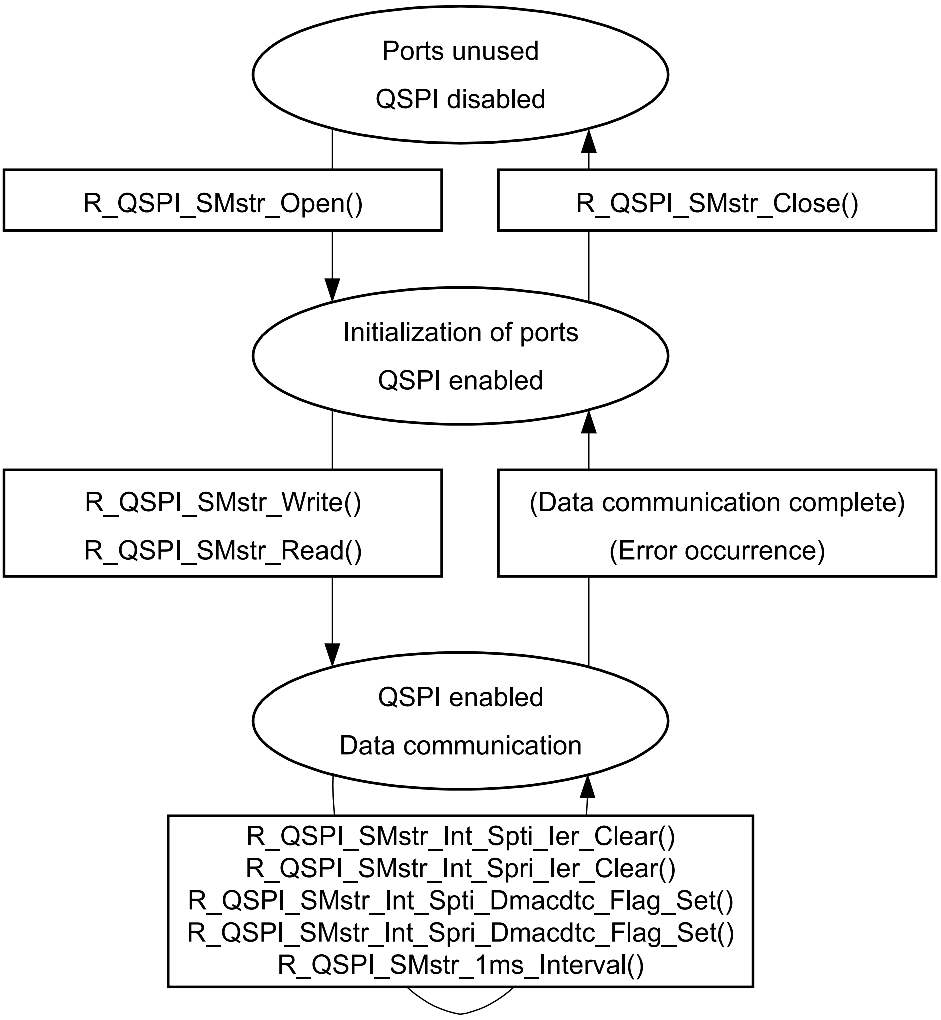


Figure 1-20 State Transition Diagram

2. API Information

The names of the APIs of the QSPI FIT module follow the Renesas API naming standard.

2.1 Hardware Requirements

The microcontroller used must support the following functionality.

- QSPI

2.2 Software Requirements

This driver is dependent on the following packages.

- r_bsp Rev.5.20 or higher
- r_dmaca_rx (When using DMAC transfer using DMACA FIT module)
- r_dtc_rx (When using DTC transfer using DTC FIT module)
- r_cmt_rx (When using Compare Match Timer CMT FIT module and DMAC transfer or DTC transfer)
  It is enable to replace to other timer or software timer.
- 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 QSPI FIT module has been confirmed with the toolchain listed in 6.1 Operating Confirmation Environment.

2.4 Interrupt vector

When the condition listed in the Table 2-1 is reached, the target interrupt is enabled within the API function. When using the DMAC, it is necessary to disable the interrupt by using the R_QSPI_SMstr_Int_Spti_Ier_Clear() function, and the R_QSPI_SMstr_Int_Spri_Ier_Clear() function after the communication ended. See “2.13.3 DMAC/DTC” for the detail.

Table 2-2 lists the interrupt vector used in the QSPI FIT Module.

Table 2-1 Condition for enabling interrupts

Condition			**Interrupt enabled**
API	**Arguments (op_mode)**	**Arguments (tran_mode)**	
R_QSPI_SMstr_Write()	QSPI_SMSTR_SINGLE_SPI_WRITE	QSPI_SMSTR_DMAC	transmit data buffer empty(SPTI) receiving buffer-full(SPRI)
		QSPI_SMSTR_DTC	transmit data buffer empty(SPTI) receiving buffer-full(SPRI)
	QSPI_SMSTR_DUAL_SPI	QSPI_SMSTR_DMAC	transmit data buffer empty(SPTI)
		QSPI_SMSTR_DTC	transmit data buffer empty(SPTI)
	QSPI_SMSTR_QUAD_SPI	QSPI_SMSTR_DMAC	transmit data buffer empty(SPTI)
		QSPI_SMSTR_DTC	transmit data buffer empty(SPTI)
R_QSPI_SMstr_Read()	QSPI_SMSTR_SINGLE_SPI_WRITE	QSPI_SMSTR_DMAC	transmit data buffer empty(SPTI) receiving buffer-full(SPRI)
		QSPI_SMSTR_DTC	transmit data buffer empty(SPTI) receiving buffer-full(SPRI)
	QSPI_SMSTR_DUAL_SPI	QSPI_SMSTR_DMAC	receiving buffer-full(SPRI)
		QSPI_SMSTR_DTC	receiving buffer-full(SPRI)
	QSPI_SMSTR_QUAD_SPI	QSPI_SMSTR_DMAC	receiving buffer-full(SPRI)
		QSPI_SMSTR_DTC	receiving buffer-full(SPRI)

Table 2-2 Interrupt vector list to be used

Device	Interrupt Vector
RX64M	SPRI interrupt(vector no: 42)
	SPTI interrupt(vector no: 43)
RX65N/RX651	SPRI interrupt(vector no: 42)
	SPTI interrupt(vector no: 43)
RX66N	SPRI interrupt(vector no: 42)
	SPTI interrupt(vector no: 43)
RX71M	SPRI interrupt(vector no: 42)
	SPTI interrupt(vector no: 43)
RX72M	SPRI interrupt(vector no: 42)
	SPTI interrupt(vector no: 43)
RX72N	SPRI interrupt(vector no: 42)
	SPTI interrupt(vector no: 43)

2.5 Header Files

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

Configuration options for individual builds are selected in r_qspi_smstr_rx_config.h.

```
#include "r_qspi_smstr_rx_if.h"
```

2.6 Integer Types

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

2.7 Compile Settings

The configuration option settings for the QSPI FIT module are specified in r_qspi_smstr_rx_config.h and r_qspi_smstr_rx_pin_config.h.

The option names and setting values of RX64M RSK are described below.

Configuration options in *r_qspi_smstr_rx_config.h*	
#define QSPI_SMSTR_CFG_USE_FIT Note: The default value is “enabled”.	Selects whether or not the QSPI FIT module is used in a BSP environment. When this option is set to “disabled”, control of FIT modules such as r_bsp is disabled. Also, the equivalent processing must be incorporated separately. For details, see 2.14 Using the Module in Other Than an FIT Module Environment. When this option is set to “enabled”, control of FIT modules such as r_bsp is enabled.
#define QSPI_SMSTR_CFG_CHx_INCLUDED Note: The default value is “enabled”. The channel number is represented by “x”.	Selects whether or not the specified channel is used. When this option is set to “disabled”, code for processing the specified channel is omitted. When this option is set to “enabled”, code for processing the specified channel is included.
#define QSPI_SMSTR_CFG_LONGQ_ENABLE Note: The default value is “disabled”.	Selects whether or not debug error log acquisition processing is used. When this option is set to “disabled”, code for the relevant processing is omitted. When this option is set to “enabled”, code for the relevant processing is included. To use this functionality, the LONGQ FIT module is also required.

Configuration options in *r_qspi_smstr_rx_pin_config.h*	
#define R_QSPI_SMSTR_CFG_QSPI_QSPCLK_PORT Note: The default value for RX64M QSPI channel 0 is “ ‘7’ ”. The channel number is represented by “x”.	Sets the port number assigned to the QSPI’s QSPCLK pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QSPCLK_BIT Note: The default value for RX64M QSPI channel 0 is “ ‘7’ ”. The channel number is represented by “x”.	Sets the bit number assigned to the QSPI’s QSPCLK pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO0_PORT Note: The default value for RX64M QSPI channel 0 is “ ‘C’ ”. The channel number is represented by “x”.	Sets the port number assigned to the QSPI’s QIO0 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO0_BIT Note: The default value for RX64M QSPI channel 0 is “ ‘3’ ”. The channel number is represented by “x”.	Sets the bit number assigned to the QSPI’s QIO0 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO1_PORT Note: The default value for RX64M QSPI channel 0 is “ ‘C’ ”. The channel number is represented by “x”.	Sets the port number assigned to the QSPI’s QIO1 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO1_BIT Note: The default value for RX64M QSPI channel 0 is “ ‘4’ ”. The channel number is represented by “x”.	Sets the bit number assigned to the QSPI’s QIO1 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO2_PORT Note: The default value for RX64M QSPI channel 0 is “ ‘8’ ”. The channel number is represented by “x”.	Sets the port number assigned to the QSPI’s QIO2 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO2_BIT Note: The default value for RX64M QSPI channel 0 is “ ‘0’ ”. The channel number is represented by “x”.	Sets the bit number assigned to the QSPI’s QIO2 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO3_PORT Note: The default value for RX64M QSPI channel 0 is “ ‘8’ ”. The channel number is represented by “x”.	Sets the port number assigned to the QSPI’s QIO3 pin. Enclose the setting value in single quotation marks (‘ ’).
#define R_QSPI_SMSTR_CFG_QSPI_QIO3_BIT Note: The default value for RX64M QSPI channel 0 is “ ‘1’ ”. The channel number is represented by “x”.	Sets the bit number assigned to the QSPI’s QIO3 pin. Enclose the setting value in single quotation marks (‘ ’).

2.8 Code Size

Table 2-3 lists the code sizes of QSPI FIT module.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview.

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

Module Revision: r_qspi_rx rev1.14

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

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

GCC for Renesas RX 4.8.4.201902

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

IAR C/C++ Compiler for Renesas RX version 4.12.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

Table 2-3 Code Size

ROM, RAM and Stack Code Sizes (Note1, 2, 3)				
Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
RX65N	ROM	5,254 bytes	10,584 bytes	7,862 bytes
	RAM	22 bytes	16 bytes	26 bytes
	Max. user stack	160 bytes	-	196 bytes
	Max. interrupt stack	48 bytes	-	68 bytes
RX71M	ROM	5,254 bytes	10,584 bytes	7,862 bytes
	RAM	22 bytes	16 bytes	26 bytes
	Max. user stack	160 bytes	-	196 bytes
	Max. interrupt stack	48 bytes	-	68 bytes

Note 1 Under confirmation conditions listed the following

- r_qspi_smstr_rx.c
- r_qspi_smstr_rx_target.c

Note 2 The required memory sizes differ according to the C compiler version and the compile conditions.

Note 3 The memory sizes listed apply when the little endian. The above memory sizes also differ according to endian mode.

2.9 Arguments

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

```
typedef volatile struct
{
    uint32_t  data_cnt;              /* Number of data (byte unit)          */
    uint8_t * p_tx_data;             /* Pointer to transmit data buffer     */
    uint8_t * p_rx_data;             /* Pointer to receive data buffer      */
    qspi_smstr_opmode_t   op_mode;   /* SPI operation mode                  */
    qspi_smstr_tranmode_t tran_mode;/* Data transfer mode                  */
} qspi_smstr_info_t;
```

2.10 Return Values

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

```
typedef enum e_qspi_smstr_status
{
    QSPI_SMSTR_SUCCESS     = 0,      /* Successful operation                        */
    QSPI_SMSTR_ERR_PARAM   = -1,     /* Parameter error                             */
    QSPI_SMSTR_ERR_HARD    = -2,     /* Hardware error                              */
    QSPI_SMSTR_ERR_OTHER   = -7      /* Other error                                 */
} qspi_smstr_status_t;
```

2.11 Callback function

QSPI has no callback function.

2.12 Adding the FIT Module to Your Project

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

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

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

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

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

2.13 Peripheral Functions and Modules Other than QSPI

In addition to the QSPI, the QSPI FIT module controls the following peripheral functions.

- I/O ports (GPIO)
- Multi-function pin controller (MPC)
- DMA controller (DMAC)
- Data transfer controller (DTC)
- Long queue (LONGQ) software module

Other than LONGQ, FIT modules are not used for resource control. When using the module described in this document in an environment using FIT modules, it is recommended that the control processing of peripheral functions other than the QSPI be replaced by equivalent FIT modules.

The target source code is contained in the file r_qspi_smstr_rx64m_dev_port.c. See 5.4 Details of Functions of Target Microcontroller Dev Layer.

2.13.1 GPIO

No GPIO FIT module is used.

The GPIO control functions in the QSPI FIT module and the control target registers are listed below. Refer to the User’s Manual: Hardware for the register names.

Table 2-4 Control Functions and Control Target Registers

Function Name	Register
r_qspi_smstr_mpc_enable()	PMR
r_qspi_smstr_mpc_disable()	PMR
r_qspi_smstr_datao0_init()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao0_reset()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao1_init()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao1_reset()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao2_init()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao2_reset()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao3_init()	ODR, DSCR, PODR, PDR
r_qspi_smstr_datao3_reset()	ODR, DSCR, PODR, PDR
r_qspi_smstr_clk_init()	ODR, DSCR, PODR, PDR
r_qspi_smstr_clk_reset()	ODR, DSCR, PODR, PDR

2.13.2 MPC

No MPC FIT module is used.

The MPC control functions in the QSPI FIT module and the control target registers are listed below. Refer to the User’s Manual: Hardware for the register names.

Table 2-5 Control Functions and Control Target Registers

Function Name	Register
r_qspi_smstr_mpc_enable()	MPC
r_qspi_smstr_mpc_disable()	MPC

2.13.3 DMAC/DTC

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

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

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

Use the control functions listed in Table 2-6 to perform the various processing tasks.

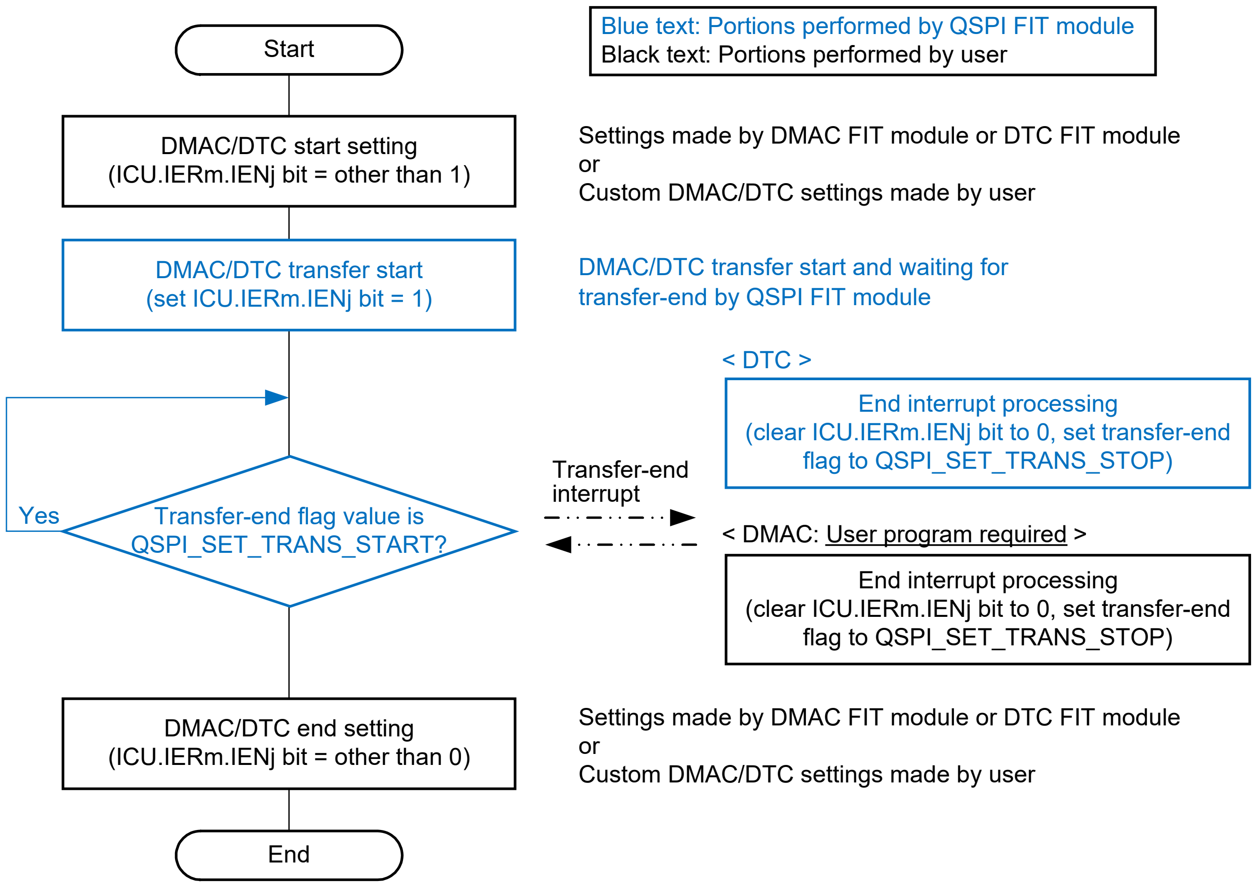


Figure 2-1 Processing for DMAC Transfer and DTC Transfer Settings

Table 2-6 lists the control functions and processing details related to DMAC/DTC control.

The data transmit-end wait processing function r_qspi_smstr_tx_dmacdtc_wait() and data receive-end wait processing function r_qspi_smstr_rx_dmacdtc_wait() wait for transmission or reception to end by running a 1 millisecond (ms) timer. It is therefore necessary to activate a 1 millisecond (ms) timer using the CMT, or the like, on the user system beforehand. Use a callback function, or the like, to call R_QSPI_SMstr_1ms_Interval() at 1 millisecond (ms) intervals.

Table 2-6 Control Functions and Processing Details

Function Name	Processing Details
r_qspi_smstr_spti_isrX()	QSPI channel “X” SPTI interrupt handler processing (X represents the channel number.)
r_qspi_smstr_spri_isrX()	QSPI channel “X” SPRI interrupt handler processing (X represents the channel number.)
r_qspi_smstr_tx_dmacdtc_wait()	DMAC/DTC transmit-end processing
r_qspi_smstr_rx_dmacdtc_wait()	DMAC/DTC receive-end processing
r_qspi_smstr_int_spti_init()	SPTI interrupt initialization processing
r_qspi_smstr_int_spri_init()	SPRI interrupt initialization processing
r_qspi_smstr_int_spti_ier_set()	Sets the SPTI interrupt ICU.IERm.IENj bit to 1.
r_qspi_smstr_int_spri_ier_set ()	Sets the SPRI interrupt ICU.IERm.IENj bit to 1.
R_QSPI_SMstr_Int_Spti_Ier_Clear()	Clears the SPTI interrupt ICU.IERm.IENj bit to 0.
R_QSPI_SMstr_Int_Spri_Ier_Clear()	Clears the SPRI interrupt ICU.IERm.IENj bit to 0.
R_QSPI_SMstr_Int_Spti_Dmacdtc_flag_Set()	Sets the DMAC/DTC transfer-end flag for transmission operations.
R_QSPI_SMstr_Int_Spri_Dmacdtc_flag_Set()	Sets the DMAC/DTC transfer-end flag for reception operations.

2.13.4 CMT

Required when using DMAC transfer or DTC transfer. Used to detect transfer timeouts.

It is possible to use other timer or software timer in place of CMT timer.

2.13.5 LONGQ

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

An example of control utilizing the LONGQ FIT module is included in the QSPI FIT module. The default setting of the relevant configuration option of the QSPI FIT module disables the error log fetching functionality. See 2.7 Compile Settings.

(1) R_LONGQ_Open() setting

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

(2) Control procedure

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

1. R_LONGQ_Open()
2. R_QSPI_SMstr_Set_LogHdlAddress()

2.14 Using the Module in Other Than an FIT Module Environment

To operate the module in an environment in which FIT modules such as r_bsp are not used, perform the following.

- Set #define QSPI_SMSTR_CFG_USE_FIT in #r_qspi_smstr_rx_config.h to "disabled".
- Comment out the line #include "platform.h" in #r_qspi_smstr_rx_if.h.
- Include the following header files in #r_qspi_smstr_rx_if.h.
  #include "iodefine.h"
  #include <stdint.h>
  #include <stdbool.h>
  #include <stddef.h>
  #include <machine.h>
- Add the definition #define BSP_MCU_RXxxx (replacing xxx with the microcontroller name using all capital letters) to #r_qspi_smstr_rx_if.h. For example, for the RX64M microcontroller use the string BSP_MCU_RX64M.

2.15 "for", "while" and "do while" statements

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

The following shows example of description.

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

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

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

3. API Functions

R_QSPI_SMstr_Open()

This function is run first when using the APIs of the QSPI clock synchronous single master control module.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Open(
  uint8_t channel,
  uint8_t spbr_data
)
```

Parameters

channel

QSPI channel number

spbr_data

QSPI bit rate register (SPBR) setting value

Return Values

```
QSPI_SMSTR_SUCCESS       /* Successful operation */
QSPI_SMSTR_ERR_PARAM     /* Parameter error */
QSPI_SMSTR_ERR_OTHER     /* QSPI resource has been acquired by other task. */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Initializes the QSPI registers of the channel number specified by the argument channel.

Sets the value specified by the argument spbr_data in the QSPI bit rate register (SPBR). In the QSPI FIT module the setting value of the bit rate division setting bits (SPCMD0.BRDV[1:0]) is 0. Refer to the User's Manual: Hardware of the microcontroller and set spbr_data as appropriate for the operating environment.

Sets QSPCLK polarity to CPOL = 1 and phase to CPHA = 1.

When the function completes successfully, the QSPI module stop state is canceled, and QSPCLK pin is set as general output port in the high-output state, and the QIO0 to QIO3 pins are set as a general input port.

Note that this function monopolizes the QSPI resource for the channel number specified by the argument channel. To release this resource, call R_QSPI_SMstr_Close().

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

```
qspi_smstr_status_t  ret = QSPI_SMSTR_SUCCESS;
uint8_t              channel;
uint8_t              spbr_data;

channel     = 0;
spbr_data   = 1;
ret = R_QSPI_SMstr_Open(channel, spbr_data);
```

Special Notes

This function controls the GPIO and MPC to set each pin as a general I/O port. Confirm that no other peripheral function is using any of the affected pins before calling this function.

R_QSPI_SMstr_Close()

This function is used to release the resources of the QSPI FIT module currently in use.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Close(
  uint8_t channel
)
```

Parameters

channel

QSPI channel number

Return Values

```
QSPI_SMSTR_SUCCESS       /* Successful operation */
QSPI_SMSTR_ERR_PARAM     /* Parameter error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Sets the QSPI of the channel number specified by the argument channel to the module stop state.

When the function completes successfully, the QSPCLK pin is set as general output port in the high-output state, and the QIO0 to QIO3 pins are set as a general input port..

Note that this function releases the QSPI resource for the channel number specified by the argument channel. To restart communication, call R_QSPI_SMstr_Open().

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

```
qspi_smstr_status_t  ret = QSPI_SMSTR_SUCCESS;
uint8_t              channel;

channel = 0;
ret = R_QSPI_SMstr_Close(channel);
```

Special Notes

This function controls the GPIO and MPC to set each pin as a general I/O port. Confirm that no other peripheral function is using any of the affected pins before calling this function.

After this function is called the states of the QSPCLK pin differ from that after a reset (general input port). Review the pin settings if necessary.

R_QSPI_SMstr_Control()

This function is used to change settings such as the bit rate and QSPCLK phase/polarity.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Control(
  uint8_t channel,
  uint8_t clk_mode,
  uint8_t spbr_data
)
```

Parameters

channel

QSPI channel number

clk_mode

QSPCLK mode. Make the following settings.

clk_mode = 0: CPOL = 0, CPHA = 0
clk_mode = 1: CPOL = 0, CPHA = 1
clk_mode = 2: CPOL = 1, CPHA = 0
clk_mode = 3: CPOL = 1, CPHA = 1

spbr_data

QSPI bit rate register (SPBR) setting value

Return Values

```
QSPI_SMSTR_SUCCESS      /* Successful operation */
QSPI_SMSTR_ERR_PARAM    /* Parameter error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Changes settings such as the QSPI bit rate and QSPCLK phase/polarity for the channel number specified by the argument channel.

Sets the value specified by the argument spbr_data in the QSPI bit rate register. In the QSPI FIT module the setting value of the bit rate division setting bits (SPCMD0.BRDV[1:0]) is 0. Refer to the User's Manual: Hardware of the microcontroller and set spbr_data as appropriate for the operating environment.

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

Example

```
qspi_smstr_status_t  ret = QSPI_SMSTR_SUCCESS;
uint8_t              channel;
uint8_t              spbr_data;
uint8_t              clk_mode;

channel      = 0;
spbr_data    = 1;
ret = R_QSPI_SMstr_Open(channel, spbr_data);     /* Set CPOL=1 and CPHA=1.*/

/* Change SPI clock mode and QSPI bit rate. */
clk_mode     = 1;      /* Set CPOL=0 and CPHA=1. */
spbr_data    = 3;
ret = R_QSPI_SMstr_Control(channel, clk_mode, spbr_data);
```

Special Notes

None

R_QSPI_SMstr_Write()

This function is used to transmit data.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Write(
  uint8_t channel,
  qspi_smstr_info_t * p_qspi_smstr_info
)
```

Parameters

channel

QSPI channel number

** p_qspi_smstr_info*

QSPI information structure

data_cnt

The allowable setting range is 1 to 4,294,967,295. A setting of 0 causes an error to be returned. Also, use a setting value that is a multiple of 16 when specifying DMAC transfer or DTC transfer.

*p_tx_data

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

*p_rx_data

Not used

op_mode

Specify the SPI mode.

QSPI_SMSTR_SINGLE_SPI_WRITE : Single-SPI mode for writing
QSPI_SMSTR_DUAL_SPI : Dual-SPI mode
QSPI_SMSTR_QUAD_SPI : Quad-SPI mode

tran_mode

Specify the transmit mode. Note that a separate DMAC or DTC transfer program is required in order to specify DMAC transfer or DTC transfer.

QSPI_SMSTR_SW : Software transfer
QSPI_SMSTR_DMAC : DMAC transfer
QSPI_SMSTR_DTC : DTC transfer

Return Values

```
QSPI_SMSTR_SUCCESS        /* Successful operation */
QSPI_SMSTR_ERR_PARAM      /* Parameter error */
QSPI_SMSTR_ERR_HARD       /* Hardware error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Uses the QSPI of the channel number specified by the argument channel to transmit data.

When DMAC transfer or DTC transfer is specified by the argument tran_mode, the transferrable byte counts are multiples of 16. If the value is not a multiple of 16, the function ends with an error and no transfer takes place.

Example

```
#define DATA_CNT (uint32_t)(256)

uint8_t             buf[DATA_CNT];
qspi_smstr_info_t   tx_info;
qspi_smstr_status_t ret = QSPI_SMSTR_SUCCESS;
uint8_t             channel;

channel = 0;
tx_info.data_cnt     = DATA_CNT;
tx_info.p_tx_data    = &buf[0];
tx_info.op_mode      = QSPI_SMSTR_SINGLE_SPI_WRITE;
tx_info.tran_mode    = QSPI_SMSTR_SW;
ret = R_QSPI_SMstr_Write(channel, &tx_info);
```

Special Notes

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

- The DMAC FIT module, DTC FIT module, and timer module (CMT FIT module, for example) must be obtained separately.
- Use a buffer address aligned with a 4-byte boundary.
- Specify a transfer data count that is a multiple of 16 when calling this function. If the transfer data count results in a final transfer with a data count of 1 to 15, specify software transfer instead when calling this function.
- Make settings such that block transfer is selected as the transfer mode and 16 bytes of data are transferred for each activation source. For example, is the data transfer size is 4 bytes, specify a block transfer count of 4.
- The data transmit-end wait processing function r_qspi_smstr_tx_dmacdtc_wait() uses a timer. Before calling this function, activate a 1 millisecond (ms) timer using the CMT, or the like. Then call R_QSPI_SMstr_1ms_Interval() at 1 millisecond (ms) intervals.
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.
- If this function is called by setting tran_mode to QSPI_SMSTR_DMAC before the DMAC is ready to be activated, no DMAC transfer will take place. The return value in this case is QSPI_SMSTR_ERR_HARD.
- If this function is called by setting tran_mode to QSPI_SMSTR_DTC before the DTC is ready to be activated, no DTC transfer will take place. The return value in this case is QSPI_SMSTR_ERR_HARD.
- When calling this function in the setting of DMAC transfer or DTC transfer and Little endian, change the data sequence of transmitted data stored in the buffer, and then send the data. After the transmission is complete, original data will be returned.

R_QSPI_SMstr_Read()

This function is used to receive data.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Read(
  uint8_t channel,
  qspi_smstr_info_t * p_qspi_smstr_info
)
```

Parameters

channel

QSPI channel number

** p_qspi_smstr_info*

QSPI information structure

data_cnt

The allowable setting range is 1 to 4,294,967,295. A setting of 0 causes an error to be returned. Also, use a setting value that is a multiple of 16 when specifying DMAC transfer or DTC transfer.

*p_tx_data

Not used

*p_rx_data

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

op_mode

Specify the SPI mode.

QSPI_SMSTR_SINGLE_SPI_READ : Single-SPI mode for reading
QSPI_SMSTR_DUAL_SPI : Dual-SPI mode
QSPI_SMSTR_QUAD_SPI : Quad-SPI mode

tran_mode

Specify the transmit mode. Note that a separate DMAC or DTC transfer program is required in order to specify DMAC transfer or DTC transfer.

QSPI_SMSTR_SW : Software transfer
QSPI_SMSTR_DMAC : DMAC transfer
QSPI_SMSTR_DTC : DTC transfer

Return Values

```
QSPI_SMSTR_SUCCESS        /* Successful operation */
QSPI_SMSTR_ERR_PARAM      /* Parameter error */
QSPI_SMSTR_ERR_HARD       /* Hardware error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Uses the QSPI of the channel number specified by the argument channel to receive data.

When DMAC transfer or DTC transfer is specified by the argument tran_mode, the transferrable byte counts are multiples of 16. If the value is not a multiple of 16, the function ends with an error and no transfer takes place.

Example

```
#define DATA_CNT (uint32_t)(256)

uint8_t               buf[DATA_CNT];
qspi_smstr_info_t     rx_info;
qspi_smstr_status_t   ret = QSPI_SMSTR_SUCCESS;
uint8_t               channel;

channel = 0;
rx_info.data_cnt      = DATA_CNT;
rx_info.p_rx_data     = &buf[0];
rx_info.op_mode       = QSPI_SMSTR_SINGLE_SPI_READ;
rx_info.tran_mode     = QSPI_SMSTR_SW;
ret = R_QSPI_SMstr_Read(channel, &rx_info);
```

Special Notes

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

- The DMAC FIT module, DTC FIT module, and timer module (CMT FIT module, for example) must be obtained separately.
- Use a buffer address aligned with a 4-byte boundary.
- Specify a transfer data count that is a multiple of 16 when calling this function. If the transfer data count results in a final transfer with a data count of 1 to 15, specify software transfer instead when calling this function.
- Make settings such that block transfer is selected as the transfer mode and 16 bytes of data are transferred for each activation source. For example, is the data transfer size is 4 bytes, specify a block transfer count of 4.
- The data transmit-end wait processing function r_qspi_smstr_tx_dmacdtc_wait() uses a timer. Before calling this function, activate a 1 millisecond (ms) timer using the CMT, or the like. Then call R_QSPI_SMstr_1ms_Interval() at 1 millisecond (ms) intervals.
- Make the necessary settings to make the DMAC or DTC ready to start before calling this function.
- If this function is called by setting tran_mode to QSPI_SMSTR_DMAC before the DMAC is ready to be activated, no DMAC transfer will take place. The return value in this case is QSPI_SMSTR_ERR_HARD.
- If this function is called by setting tran_mode to QSPI_SMSTR_DTC before the DTC is ready to be activated, no DTC transfer will take place. The return value in this case is QSPI_SMSTR_ERR_HARD.

R_QSPI_SMstr_Get_BuffRegAddress()

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

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Get_BuffRegAddress (
  uint8_t channel,
  uint32_t * p_spdr_adr
)
```

Parameters

channel

QSPI channel number

** p_spdr_adr*

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

Return Values

```
QSPI_SMSTR_SUCCESS       /* Successful operation */
QSPI_SMSTR_ERR_PARAM     /* Parameter error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

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

Example

```
uint32_t              reg_buff;
qspi_smstr_status_t   ret = QSPI_SMSTR_SUCCESS;
uint8_t               channel;

channel = 0;
ret = R_QSPI_SMstr_Get_BuffRegAddress(channel, &reg_buff);
```

Special Notes

None

R_QSPI_SMstr_Int_Spti_Ier_Clear()

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

Format

```
void R_QSPI_SMstr_Int_Spti_Ier_Clear (
  uint8_t channel
)
```

Parameters

channel

QSPI channel number

Return Values

None

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Use this function when disabling interrupts from within the handler of the SPTI interrupt generated at DMAC transfer-end.

Example

```
DMA_Handler_W()
{
    R_QSPI_SMstr_Int_Spti_Ier_Clear(0);
    R_QSPI_SMstr_Int_Spti_Dmacdtc_Flag_Set(0, QSPI_SET_TRANS_STOP);
}
```

Special Notes

Do not use this function for software transfers or DTC transfers. Doing so could disrupt the transfer.

R_QSPI_SMstr_Int_Spri_Ier_Clear()

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

Format

```
void R_QSPI_SMstr_Int_Spri_Ier_Clear (
  uint8_t channel
)
```

Parameters

channel

QSPI channel number

Return Values

None

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Use this function when disabling interrupts from within the handler of the SPRI interrupt generated at DMAC transfer-end.

Example

```
DMA_Handler_R()
{
    R_QSPI_SMstr_Int_Spri_Ier_Clear(0);
    R_QSPI_SMstr_Int_Spri_Dmacdtc_Flag_Set(0, QSPI_SET_TRANS_STOP);
}
```

Special Notes

Do not use this function for software transfers or DTC transfers. Doing so could disrupt the transfer.

R_QSPI_SMstr_Int_Spti_Dmacdtc_flag_Set()

This function is used to set the DMAC or DTC transfer-end flag for data transmission.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Int_Spti_Dmacdtc_flag_Set (
  uint8_t channel,
  qspi_smstr_trans_flg_t flg
)
```

Parameters

channel

QSPI channel number

flg

Flag. The settings are as follows.

QSPI_SET_TRANS_STOP : DMAC or DTC transfer-end

(QSPI_SET_TRANS_START : DMAC or DTC transfer-start: Setting by the user is prohibited.)

Return Values

```
QSPI_SMSTR_SUCCESS       /* Successful operation */
QSPI_SMSTR_ERR_PARAM     /* Parameter error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Set QSPI_SET_TRANS_STOP from within the handler of the SPTI interrupt generated at DMAC transfer-end.

Example

```
DMA_Handler_W()
{
   R_QSPI_SMstr_Int_Spti_Ier_Clear(0);
   R_QSPI_SMstr_Int_Spti_Dmacdtc_Flag_Set(0, QSPI_SET_TRANS_STOP);
}
```

Special Notes

Do not use this function for software transfers or DTC transfers. Doing so could disrupt the transfer.

R_QSPI_SMstr_Int_Spri_Dmacdtc_flag_Set()

This function is used to set the DMAC or DTC transfer-end flag for data reception.

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Int_Spri_Dmacdtc_flag_Set (
  uint8_t channel,
  qspi_smstr_trans_flg_t flg
)
```

Parameters

channel

QSPI channel number

flg

Flag. The settings are as follows.

QSPI_SET_TRANS_STOP : DMAC or DTC transfer-end

(QSPI_SET_TRANS_START : DMAC or DTC transfer-start: Setting by the user is prohibited.)

Return Values

```
QSPI_SMSTR_SUCCESS        /* Successful operation */
QSPI_SMSTR_ERR_PARAM      /* Parameter error */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Set QSPI_SET_TRANS_STOP from within the handler of the SPRI interrupt generated at DMAC transfer-end.

Example

```
DMA_Handler_R()
{
   R_QSPI_SMstr_Int_Spri_Ier_Clear(0);
   R_QSPI_SMstr_Int_Spri_Dmacdtc_Flag_Set(0, QSPI_SET_TRANS_STOP);
}
```

Special Notes

Do not use this function for software transfers or DTC transfers. Doing so could disrupt the transfer.

R_QSPI_SMstr_GetVersion()

This function is used to fetch the driver version information.

Format

```
uint32_t R_QSPI_SMstr_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_qspi_smstr_rx_if.h.

Description

Returns the version information.

Example

```
uint32_t version;
version = R_QSPI_SMstr_GetVersion();
```

Special Notes

None

R_QSPI_SMstr_Set_LogHdlAddress()

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

Format

```
qspi_smstr_status_t R_QSPI_SMstr_Set_LogHdlAddress(
  uint32_t user_long_que
)
```

Parameters

user_long_que

Specify the handler address of the LONGQ FIT module.

Return Values

```
QSPI_SMSTR_SUCCESS       /* Successful operation */
```

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

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

Uses the LONGQ FIT module perform preparatory processing for fetching the error log.

Run this processing before calling R_QSPI_SMstr_Open().

Example

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

qspi_smstr_status_t  ret = QSPI_SMSTR_SUCCESS;
uint32_t             MtlLogTbl[ERR_LOG_SIZE];   /*Err log buffer           */
longq_err_t          err;
longq_hdl_t          p_qspi_user_long_que;      /* Pointer of LONGQ hndler */
uint32_t             long_que_hndl_address;     /* Address of LONGQ hndler */


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

long_que_hndl_address = (uint32_t)p_qspi_user_long_que;
ret = R_QSPI_SMstr_Set_LogHdlAddress(long_que_hndl_address);
```

Special Notes

Incorporate the LONGQ FIT module separately. Also, enable the line #define QSPI_SMSTR_CFG_LONGQ_ENABLE in r_qspi_smstr_rx_config.h.

If the QSPI_SMSTR_CFG_LONGQ_ENABLE disable and this function is called, this function does nothing.

R_QSPI_SMstr_Log()

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

Format

```
uint32_t R_QSPI_SMstr_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_qspi_smstr_rx_if.h.

Description

This function fetches the error log. Executes the end processing of fetching the error log using LONGQ FIT module. When an error occurs, call this function immediately before user processing ends.

For the setting of the error log size, refer to the LONGQ FIT module.

Example

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

uint8_t                buf[DATA_CNT];
qspi_smstr_info_t      tx_info;
qspi_smstr_status_t    ret = QSPI_SMSTR_SUCCESS;
uint8_t                channel;

channel = 0;
tx_info.data_cnt       = DATA_CNT;
tx_info.p_tx_data      = &buf[0];
tx_info.op_mode        = QSPI_SMSTR_QUAD_SPI;
tx_info.tran_mode      = QSPI_SMSTR_SW;

ret = R_QSPI_SMstr_Write(channel, &tx_info);
if (QSPI_SMSTR_SUCCESS != ret)
{
    /* Set last error log to buffer. */
    R_QSPI_SMstr_Log(
        USER_DRIVER_ID,
        USER_LOG_MAX,
        USER_LOG_ADR_MAX
    );
    R_QSPI_SMstr_Close(channel);
}
```

Special Notes

Incorporate the LONGQ FIT module separately. Also, enable the line #define QSPI_SMSTR_CFG_LONGQ_ENABLE in r_qspi_smstr_rx_config.h.

If the QSPI_SMSTR_CFG_LONGQ_ENABLE disable and this function is called, this function does nothing.

R_QSPI_SMstr_1ms_Interval()

This function increments the internal timer counter each time it is called.

Format

```
void R_QSPI_SMstr_1ms_Interval(void)
```

Parameters

None

Return Values

None

Properties

Prototype declarations are contained in r_qspi_smstr_rx_if.h.

Description

Increments the internal timer counter while waiting for the DMAC transfer or DTC transfer to finish.

Example

```
void r_cmt_callback (void * pdata)
{
    uint32_t channel;
    channel = (uint32_t)pdata;
    if (channel == gs_cmt_channel)
    {
    R_QSPI_SMstr_1ms_Interval();
}
```

Special Notes

Use a timer or the like to call this function at 1 millisecond (ms) intervals.

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

4. Pin Setting

To use the QSPI FIT module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). But user processing is unnecessary because pin setting is executed within the R_QSPI_SMstr_Write/ R_QSPI_SMstr_Read/ R_QSPI_SMstr_WriteRead function.

When performing the pin setting in the e^2 studio, the pin setting feature of the Smart configurator can be used. When using the pin setting feature, can use the pin selected on the terminal setting screen in the Pin Setting window in the Smart configurator.

The selected pin information is reflected in r_qspi_smstr_rx_pin_config.h.

It is overwritten on the macro definition written on Table 4-1 Pin Setting macro definition.

Table 4-1 Pin Setting macro definition

Selected option	Macro definition
channel0	R_QSPI_SMSTR_CFG_QSPI_QSPCLK_PORT
	R_QSPI_SMSTR_CFG_QSPI_QSPCLK_BIT
	R_QSPI_SMSTR_CFG_QSPI_QIO0_PORT
	R_QSPI_SMSTR_CFG_QSPI_QIO0_BIT
	R_QSPI_SMSTR_CFG_QSPI_QIO1_PORT
	R_QSPI_SMSTR_CFG_QSPI_QIO1_BIT
	R_QSPI_SMSTR_CFG_QSPI_QIO2_PORT
	R_QSPI_SMSTR_CFG_QSPI_QIO2_BIT
	R_QSPI_SMSTR_CFG_QSPI_QIO3_PORT
	R_QSPI_SMSTR_CFG_QSPI_QIO3_BIT

Table 4-2 lists the pin states after a power on reset and by execution of various API functions. As shown in " (2) Controllable Slave Devices", this module supports SPI mode 3 (CPOL = 1, CPHA = 1). Also, the QSPCLK pin is in the GPIO high-output state after R_QSPI_SMstr_Close() runs. Review the pin settings if necessary.

Table 4-2 Pin States after Function Execution

Function Name	QSPCLK	QIO0-QIO3 (Note 1)
(After power on reset)	GPIO input state	GPIO input state
After R_QSPI_SMstr_Open()	GPIO high-output state Set by this module	GPIO input state Set by this module
Durring R_QSPI_SMstr_Write() R_QSPI_SMstr_Read() R_QSPI_SMstr_WriteRead()	Peripheral high-output or low-output state	Peripheral high-output or low-output state
After R_QSPI_SMstr_Write() R_QSPI_SMstr_Read() R_QSPI_SMstr_WriteRead()	GPIO high-output state	GPIO high-output state
After R_QSPI_SMstr_Close()	GPIO high-output state Set by this module	GPIO input state Set by this module

Note 1: Use an external resistor to pull up the QIO0, QIO1, QIO2 and QIO3 pins. See, "(1)Hardware Configuration Example".

5. 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.

5.1 qspi_demo_rskrx64m, qspi_demo_rskrx65n_2m, qspi_demo_rskrx72n, qspi_demo_rskrx64m_gcc, qspi_demo_rskrx65n_2m_gcc, qspi_demo_rskrx72n_gcc

These QSPI FIT module demo programs are designed for the Renesas RSKRX64M, RSKRX65N-2M, RSKRX72N demo boards. The program demonstrates how to use the APIs of QSPI FIT module with a RX MCU operating as the master device to read and write to the Serial Flash memory (MX25L3233F serial NOR Flash) operating as the slave device.
Once the code is compiled and downloaded to the target board and is running, LED0 will turn ON after initialization. After the QSPI module is successfully opened, LED1 will turn ON. After data has been successfully written to the slave device, LED2 will turn ON. After data has been successfully read from the slave device, LED3 will turn ON. After the QSPI module is successfully closed, all LEDs will turn OFF.

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. Selection of data transfer module:
When DMACA FIT module is used, macro definition "USE_DMAC_FIT" in rskrx64m.h (for RSKRX64M), rskrx65n_2m.h (for RSKRX65N-2M), and rskrx72n.h (for RSKRX72N) is made effective by changing condition:
```
#if 0
#define USE_DMAC_FIT
#endif
```
Change to:
```
#if 1
#define USE_DMAC_FIT
#endif
```
When DTC FIT module is used, macro definition USE_DTC_FIT in rskrx64m.h (for RSKRX64M), rskrx65n_2m.h (for RSKRX65N-2M), and rskrx72n.h (for RSKRX72N) is made effective by changing condition:
```
#if 0
#define USE_DTC_FIT
#endif
```
Change to:
```
#if 1
#define USE_DTC_FIT
#endif
```
By default, the transmit mode is Software Transfer.

3. Pin setting: check pin settings in r_qspi_smstr_rx_pin_config.h

4. Connect the RSK board to the PC (using Renesas E1 Emulator). Build this sample application, download it to the board.

5. In Renesas e2 studio IDE, click the Renesas Views tab -> Move mouse over the Debug item, and select the Renesas Debug Virtual Console to show it.

6. By checking the log and LEDs, confirm that the application writes and reads data to the slave device (MX25L3233F serial NOR Flash), and validate the 256-byte data sent and received in Renesas Debug Virtual Console view.

Boards Supported
RSKRX64M, RXKRX65N-2M, RSKRX72N

5.2 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects into Workspace*, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.3 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Browser* >> *Application Notes* tab.

6. Appendix

6.1 Operating Confirmation Environment

This section describes operation confirmation environment for the QSPI FIT module.

Table 6-1 Operation Confirmation Conditions(Ver.1.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V6.0.0
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.07.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big-endian/Little-endian
Version of the module	Ver. 1.10
Board used	Renesas Starter Kit for RX64M (product No.: R0K50564MsxxxBE) Renesas Starter Kit for RX65N (product No.: RTK500565NSxxxxxBE) Renesas Starter Kit for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX71M (product No.: R0K50571MsxxxBE)

Table 6-2 Operation Confirmation Conditions(Ver.1.11)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.3.0
C compiler	Renesas Electronics C/C++ compiler for RX Family V.3.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big-endian/Little-endian
Version of the module	Ver. 1.11

Table 6-3 Operation Confirmation Conditions(Ver.1.12)

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 order	Big-endian/Little-endian
Version of the module	Ver. 1.12
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxx)

Table 6-4 Operation Confirmation Conditions(Ver.1.13)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Version of the module	Ver. 1.13
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 6-5 Operation Confirmation Conditions(Ver.1.14)

Item	Contents
Integrated development environment	Renesas Electronics e[2] studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Version of the module	Ver. 1.14
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx)

Table 6-6 Operation Confirmation Conditions(Ver.1.20)

Item	Contents
Integrated development environment	Renesas Electronics e[2] studio Version 2022-07 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202202 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian order	Big endian/little endian
Version of the module	Ver. 1.20
Board used	Renesas Starter Kit for RX64M (product No.: R0K50564Mxxxxxx) Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2Cxxxxxxx) Renesas Starter Kit+ for RX72N (product No.: RTK5572NNDCxxxxxx)

Table 6-7 Operation Confirmation Conditions(Ver.1.21)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2023-04 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202305 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian order	Big endian/little endian
Version of the module	Ver. 1.21

6.2 Trouble Shooting

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

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

- When using CS+:

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

- When using e^2 studio:

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

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

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

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

6.3 Details of Functions of Target Microcontroller QSPI Layer

The names of registers occurring in the descriptions that follow and their abbreviations are listed below. For details of the registers occurring in the descriptions that follow and the names of their bit fields, refer to the User's Manual: Hardware of the microprocessor.

Details of the target microcontroller QSPI layer functions used by the QSPI FIT module are listed below.

Table 6-7 Register Names and Abbreviations

Abbreviation	Register Name
SPCR	QSPI control register
SSLP	QSPI slave select polarity register
SPPCR	QSPI pin control register
SPSR	QSPI status register
SPDR	QSPI data register
SPSCR	QSPI sequence control register
SPSSR	QSPI sequence status register
SPBR	QSPI bit rate register
SPDCR	QSPI data control register
SPCKD	QSPI clock delay register
SSLND	QSPI slave select negate delay register
SPND	QSPI next access delay register
SPCMD0 to SPCMD3	QSPI command registers 0 to 3
SPBFCR	QSPI buffer control register
SPBDCR	QSPI buffer data count setting register
SPBMUL0 to SPBMUL3	QSPI data transfer length multiple setting registers 0 to 3

6.3.1 r_qspi_smstr_ch_check()

(1) Purpose

Confirms whether or not the specified channel has been defined.

(2) Functionality

Ends normally if the specified channel has been defined.

(3) Note

Designate the channel to be used by enabling the line #define QSPI_SMSTR_CFG_CHx_INCLUDED (x = channel number) in r_qspi_smstr_rx_config.h.

6.3.2 r_qspi_smstr_enable()

(1) Purpose

Initializes the QSPI and enables its functionality. The function performs common processing up to the point at which transmission or transmission/reception are enabled. It also sets the bit rate.

(2) Functionality

Common processing is run until transmission or reception is enabled. Also sets the bit rate.

1. Cancels the module stop state.
   Refer to the description of r_qspi_smstr_module_enable().
2. Common processing of procedure for enabling transmit settings, receive settings, and transmit/receive settings.
   - SPCR ← 08h: QSPI functionality disabled, some QSPI functionality initialized, master mode (QSPCLK pin output enabled)
   - SSLP ← 00h: QSSL signal low-active*[1]
     SPPCR ← 36h: QIO2 and QIO3 fixed at 1 in single and dual modes, data output idle value fixed at 1
   - Set SPBR bit rate.
   - SPCKD ← 00h: SPCKD delay value setting (default value)
   - SSLND ← 00h: QSSL negate delay value (default value)
   - SPND ← 00h: Next access delay value (default value)
   - SPDCR ← 00h: Dummy data transmit disabled
   - Clear SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
   - SPSCR ← 00h: Sequence length, use SPCMD0
   - SPBFCR ← 30h:Transmit buffer trigger count 0 bytes, receive buffer trigger count 1 byte
   - SPCMD0 ← E203h:
     CPHA = 1, CPOL = 1, bit rate, single-SPI, negate QSSL signal at transfer end, 32-bit, MSB-first, next access delay enabled, QSSL negate delay enabled, clock delay enabled
   - SPCMD1 ← E203h:
     CPHA = 1, CPOL = 1, bit rate, single-SPI, negate QSSL signal at transfer end, 32-bit, MSB-first, next access delay enabled, QSSL negate delay enabled, clock delay enabled
   - SPCMD2 ← E203h:
     CPHA = 1, CPOL = 1, bit rate, single-SPI, negate QSSL signal at transfer end, 32-bit, MSB-first, next access delay enabled, QSSL negate delay enabled, clock delay enabled

(3) Notes

This function forms a pair with r_qspi_smstr_disable().

6.3.3 r_qspi_smstr_disable()

(1) Purpose

Disables the QSPI.

(2) Functionality

Common processing of procedure for disabling transmit settings, receive settings, and transmit/receive settings.

1. Cancels the module stop state.
   Refer to the description of r_qspi_smstr_module_enable().
2. Clears SPSSLF, SPTEF, and SPRFF in SPSR.
   Refer to the description of r_qspi_smstr_spsr_clear().
3. Enables the module stop state.
   Refer to the description of r_qspi_smstr_module_disablef().

(3) Notes

This function forms a pair with r_qspi_smstr_enable().

6.3.4 r_qspi_smstr_change()

(1) Purpose

Changes the bit rate and QSPCLK phase/polarity.

(2) Functionality

Sets the bit rate in SPBR and set the QSPCLK phase/polarity in SPCMD.

1. Set bit rate in SPBR.
2. Set CPHA (phase) and CPOL (polarity) in SPCMD0 to SPCMD2.

(3) Notes

None

6.3.5 r_qspi_smstr_data_set_long()

(1) Purpose

Specifies 32-bit transmit data.

(2) Functionality

Writes data to SPDR.

(3) Notes

None

6.3.6 r_qspi_smstr_data_set_byte()

(1) Purpose

Specifies 8-bit transmit data.

(2) Functionality

Writes data to SPDR.

(3) Notes

None

6.3.7 r_qspi_smstr_data_get_long()

(1) Purpose

Fetches 32-bit receive data.

(2) Functionality

Fetches the data from SPDR and sets that value as the return value.

(3) Notes

None

6.3.8 r_qspi_smstr_data_get_byte()

(1) Purpose

Fetches 8-bit receive data.

(2) Functionality

Fetches the data from SPDR and sets that value as the return value.

(3) Notes

None

6.3.9 r_qspi_smstr_spsr_clear()

(1) Purpose

Clears the SPSSLF, SPTEF, and SPRFF flags in SPSR.

(2) Functionality

Clears the SPSSLF, SPTEF, and SPRFF flags in SPSR.

1. If any flag is set to 1, clears it to 0.

(3) Notes

None

6.3.10 r_qspi_smstr_sptef_clear()

(1) Purpose

Clears the SPTEF flag in SPSR.

(2) Functionality

Clears the SPTEF flag in SPSR.

1. If the flag is set to 1, clears it to 0.

(3) Notes

None

6.3.11 r_qspi_smstr_sprff_clear()

(1) Purpose

Clears the SPREF flag in SPSR.

(2) Functionality

Clears the SPREF flag in SPSR.

1. If the flag is set to 1, clears it to 0.

(3) Notes

None

6.3.12 r_qspi_smstr_spsslf_clear()

(1) Purpose

Clears the SPSSLF flag in SPSR.

(2) Functionality

Clears the SPSSLF flag in SPSR.

1. If the flag is set to 1, clears it to 0.

(3) Notes

None

6.3.13 r_qspi_smstr_spsr_addr()

(1) Purpose

Fetches the address of SPSR.

(2) Functionality

Sets the SPSR address as the return value.

(3) Notes

None

6.3.14 r_qspi_smstr_trx_enable_single()

(1) Purpose

Enables transmit/receive in single-SPI mode.

(2) Functionality

After switching pin functions from general I/O ports to QSPI functions, enables single-SPI mode QSPI functionality.

This function performs dedicated initialization processing of single-SPI mode transmit/receive settings as an initialization procedure following r_qspi_smstr_enable().

1. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
2. Resets the transmit buffer and receive buffer. (Refer to the description of r_qspi_smstr_buffer_reset().)
3. Sets the transmit/receive buffer data count trigger and data count.
   (Refer to the description of r_qspi_smstr_datasize_set().)
4. Specifies the SPSCR reference sequence number, SPI operation mode, and QSSL signal control method.*1
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*2 and is not a multiple of 16 >
   — SPSCR ← 01h: SPCMD0 → SPCMD1
   — SPCMD0.SPIMOD ← 00b: Single-SPI
   — SPCMD0.SSLKP ← 1b: Maintain QSSL signal level after transfer-end to start of next access
   — SPCMD1.SPIMOD ← 00b: Single-SPI
   — SPCMD1.SSLKP ← 0b: Negate QSSL signal after transfer-end
   < When the data count is other than the above >
   — SPSCR ← 00b: SPCMD0
   — SPCMD0.SPIMOD ← 00b: Single-SPI
   — SPCMD0.SSLKP ← 0b: Negate QSSL signal after transfer-end
5. Sets the pins to be used to QSPI functions.
   Refer to the description of r_qspi_smstr_mpc_enable()
6. Sets SPE in SPCR (functionality enabled).
   Sets SPE to enable QSPI functionality.
   — SPCR ← 48h: QSPI functionality enabled, master mode (QSPCLK pin output enabled)

(3) Notes

This function forms a pair with r_qspi_smstr_trx_disable().

1. QSSL control functionality is not used. The QSSL pin can be reassigned to another function, so it is possible to set the QSSL pin as a general output port and assign it as the slave device select output.
2. The RX64M, RX65N, RX71M, RX72M, RX72N and RX66N have 32 8-bit buffers each for transmission and reception, so half of the buffer count is 16.

6.3.15 r_qspi_smstr_trx_disable()

(1) Purpose

Disables transmit/receive on the QSPI.

(2) Functionality

Disables transmit/receive operation. After switching pin functions from QSPI functions to general I/O ports, makes settings to disable transmit/receive.

1. Disables peripheral functions of pins. (Refer to the description of r_qspi_smstr_mpc_disable().)
2. Clears SPE in SPCR (functionality disabled).
   Clears SPE to disable QSPI functionality.
   — SPCR ← 08h: QSPI functionality disabled, some QSPI functionality initialized, master mode (QSPCLK pin output enabled)
3. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
4. Sets SPSCR to value before reset.
   — SPSCR ← 00b: SPCMD0

(3) Notes

This function forms a pair with r_qspi_smstr_trx_enable().

6.3.16 r_qspi_smstr_tx_enable_dual()

(1) Purpose

Enables transmit in dual-SPI mode.

(2) Functionality

After switching pin functions from general I/O ports to QSPI functions, enables dual-SPI mode QSPI functionality.

This function performs dedicated initialization processing of dual-SPI mode transmit settings as an initialization procedure following r_qspi_smstr_enable().

1. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
2. Resets the transmit buffer and receive buffer. (Refer to the description of r_qspi_smstr_buffer_reset().)
3. Sets the transmit/receive buffer data count trigger and data count.
   (Refer to the description of r_qspi_smstr_datasize_set().)
4. Specifies the SPSCR reference sequence number, SPI read/write access setting, SPI operation mode, and QSSL signal control method.*1
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*2 and is not a multiple of 16 >
   — SPSCR ← 01h: SPCMD0 → SPCMD1
   — SPCMD0.SPRW ← 0b: Write operation
   — SPCMD0.SPIMOD ← 01b: Dual-SPI
   — SPCMD0.SSLKP ← 1b: Maintain QSSL signal level after transfer-end to start of next access
   — SPCMD1.SPRW ← 0b: Write operation
   — SPCMD1.SPIMOD ← 01b: Dual-SPI
   — SPCMD1.SSLKP ← 0b: Negate QSSL signal after transfer-end
   < When the data count is other than the above >
   — SPSCR ← 00b: SPCMD0
   — SPCMD0.SPRW ← 0b: Write operation
   — SPCMD0.SPIMOD ← 01b: Dual-SPI
   — SPCMD0.SSLKP ← 0b: Negate QSSL signal after transfer-end
5. Sets the pins to be used to QSPI functions.
   Refer to the description of r_qspi_smstr_mpc_enable()
6. Sets SPE in SPCR (functionality enabled).
   Sets SPE to enable QSPI functionality.
   — SPCR ← 48h: QSPI functionality enabled, master mode (QSPCLK pin output enabled)

(3) Notes

This function forms a pair with r_qspi_smstr_tx_disable().

1. QSSL control functionality is not used. The QSSL pin can be reassigned to another function, so it is possible to set the QSSL pin as a general output port and assign it as the slave device select output.
2. The RX64M, RX65N, RX71M, RX72M, RX72N and RX66N have 32 8-bit buffers each for transmission and reception, so half of the buffer count is 16.

6.3.17 r_qspi_smstr_tx_enable_quad()

(1) Purpose

Enables transmit in quad-SPI mode.

(2) Functionality

After switching pin functions from general I/O ports to QSPI functions, enables quad-SPI mode QSPI functionality.

This function performs dedicated initialization processing of quad-SPI mode transmit settings as an initialization procedure following r_qspi_smstr_enable().

1. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
2. Resets the transmit buffer and receive buffer. (Refer to the description of r_qspi_smstr_buffer_reset().)
3. Sets the transmit/receive buffer data count trigger and data count.
   (Refer to the description of r_qspi_smstr_datasize_set().)
4. Specifies the SPSCR reference sequence number, SPI read/write access setting, SPI operation mode, and QSSL signal control method.*1
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*2 and is not a multiple of 16 >
   — SPSCR ← 01h: SPCMD0 → SPCMD1
   — SPCMD0.SPRW ← 0b: Write operation
   — SPCMD0.SPIMOD ← 10b: Quad-SPI
   — SPCMD0.SSLKP ← 1b: Maintain QSSL signal level after transfer-end to start of next access
   — SPCMD1.SPRW ← 0b: Write operation
   — SPCMD1.SPIMOD ← 10b: Quad-SPI
   — SPCMD1.SSLKP ← 0b: Negate QSSL signal after transfer-end
   < When the data count is other than the above >
   — SPSCR ← 00b: SPCMD0
   — SPCMD0.SPRW ← 0b: Write operation
   — SPCMD0.SPIMOD ← 10b: Quad-SPI
   — SPCMD0.SSLKP ← 00b: Negate QSSL signal after transfer-end
5. Sets the pins to be used to QSPI functions.
   Refer to the description of r_qspi_smstr_mpc_enable()
6. Sets SPE in SPCR (functionality enabled).
   Sets SPE to enable QSPI functionality.
   — SPCR ← 48h: QSPI functionality enabled, master mode (QSPCLK pin output enabled)

(3) Notes

This function forms a pair with r_qspi_smstr_tx_disable().

1. QSSL control functionality is not used. The QSSL pin can be reassigned to another function, so it is possible to set the QSSL pin as a general output port and assign it as the slave device select output.
2. The RX64M, RX65N, RX71M, RX72M, RX72N and RX66N have 32 8-bit buffers each for transmission and reception, so half of the buffer count is 16.

6.3.18 r_qspi_smstr_tx_disable()

(1) Purpose

Disables transmit/receive on the QSPI.

(2) Functionality

Disables transmit/receive operation. After switching pin functions from QSPI functions to general I/O ports, makes settings to disable transmit/receive.

1. Disables peripheral functions of pins. (Refer to the description of r_qspi_smstr_mpc_disable().)
2. Clears SPE in SPCR (functionality disabled).
   Clears SPE to disable QSPI functionality.
   — SPCR ← 08h: QSPI functionality disabled, some QSPI functionality initialized, master mode (QSPCLK pin output enabled)
3. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
4. Sets SPSCR to value before reset.
   — SPSCR ← 00b: SPCMD0

(3) Notes

This function forms a pair with r_qspi_smstr_tx_enable_dual() or r_qspi_smstr_tx_enable_quad().

6.3.19 r_qspi_smstr_rx_enable_dual()

(1) Purpose

Enables transmit in dual-SPI mode.

(2) Functionality

After switching pin functions from general I/O ports to QSPI functions, enables dual-SPI mode QSPI functionality.

This function performs dedicated initialization processing of dual-SPI mode receive settings as an initialization procedure following r_qspi_smstr_enable().

1. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
2. Resets the transmit buffer and receive buffer. (Refer to the description of r_qspi_smstr_buffer_reset().)
3. Sets the transmit/receive buffer data count trigger and data count.
   (Refer to the description of r_qspi_smstr_datasize_set().)
4. Specifies the SPSCR reference sequence number, SPI read/write access setting, SPI operation mode, and QSSL signal control method.*1
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*2 and is not a multiple of 16 >
   — SPSCR ← 02h: SPCMD0 → SPCMD1 → SPCMD2
   — SPCMD0.SPRW ← 1b: Read operation
   — SPCMD0.SPIMOD ← 01b: Dual-SPI
   — SPCMD0.SSLKP ← 1b: Maintain QSSL signal level after transfer-end to start of next access
   — SPCMD1.SPRW ← 1b: Read operation
   — SPCMD1.SPIMOD ← 01b: Dual-SPI
   — SPCMD1.SSLKP ← 0b: Negate QSSL signal after transfer-end
   — SPCMD2.SPRW ← 0b: Write operation (dummy write sequence setting for transfer-end)
   — SPCMD2.SPIMOD ← 10b: Quad-SPI
   < When the data count is other than the above >
   — SPSCR ← 01b: SPCMD0 → SPCMD1
   — SPCMD0.SPRW ← 1b: Read operation
   — SPCMD0.SPIMOD ← 01b: Dual-SPI
   — SPCMD0.SSLKP ← 0b: Negate QSSL signal after transfer-end
   — SPCMD1.SPRW ← 0b: Write operation (dummy write sequence setting for transfer-end)
   — SPCMD1.SPIMOD ← 10b: Quad-SPI
5. Sets the pins to be used to QSPI functions.
   Refer to the description of r_qspi_smstr_mpc_enable()
6. Sets SPE in SPCR (functionality enabled).
   Sets SPE to enable QSPI functionality.
   — SPCR ← 48h: QSPI functionality enabled, master mode (QSPCLK pin output enabled)

(3) Notes

This function forms a pair with r_qspi_smstr_rx_disable().

1. QSSL control functionality is not used. The QSSL pin can be reassigned to another function, so it is possible to set the QSSL pin as a general output port and assign it as the slave device select output.
2. The RX64M, RX65N, RX71M, RX72M, RX72N and RX66N have 32 8-bit buffers each for transmission and reception, so half of the buffer count is 16.

6.3.20 r_qspi_smstr_rx_enable_quad()

(1) Purpose

Enables transmit in quad-SPI mode.

(2) Functionality

After switching pin functions from general I/O ports to QSPI functions, enables quad-SPI mode QSPI functionality.

This function performs dedicated initialization processing of quad-SPI mode receive settings as an initialization procedure following r_qspi_smstr_enable().

1. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
2. Resets the transmit buffer and receive buffer. (Refer to the description of r_qspi_smstr_buffer_reset().)
3. Sets the transmit/receive buffer data count trigger and data count.
   (Refer to the description of r_qspi_smstr_datasize_set().)
4. Specifies the SPSCR reference sequence number, SPI read/write access setting, SPI operation mode, and QSSL signal control method.*[1]
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*[2] and is not a multiple of 16 >
   — SPSCR ← 02h: SPCMD0 → SPCMD1 → SPCMD2
   — SPCMD0.SPRW ← 1b: Read operation
   — SPCMD0.SPIMOD ← 10b: Quad-SPI
   — SPCMD0.SSLKP ← 1b: Maintain QSSL signal level after transfer-end to start of next access
   — SPCMD1.SPRW ← 1b: Read operation
   — SPCMD1.SPIMOD ← 10b: Quad-SPI
   — SPCMD1.SSLKP ← 0b: Negate QSSL signal after transfer-end
   — SPCMD2.SPRW ← 0b: Write operation (dummy write sequence setting for transfer-end)
   — SPCMD2.SPIMOD ← 10b: Quad-SPI
   < When the data count is other than the above >
   — SPSCR ← 01b: SPCMD0 → SPCMD1
   — SPCMD0.SPRW ← 1b: Read operation
   — SPCMD0.SPIMOD ← 10b: Quad-SPI
   — SPCMD0.SSLKP ← 0b: Negate QSSL signal after transfer-end
   — SPCMD1.SPRW ← 0b: Write operation (dummy write sequence setting for transfer-end)
   — SPCMD2.SPIMOD ← 10b: Quad-SPI
5. Sets the pins to be used to QSPI functions.
   Refer to the description of r_qspi_smstr_mpc_enable()
6. Sets SPE in SPCR (functionality enabled).
   Sets SPE to enable QSPI functionality.
   — SPCR ← 48h: QSPI functionality enabled, master mode (QSPCLK pin output enabled)

(3) Notes

This function forms a pair with r_qspi_smstr_rx_disable().

1. QSSL control functionality is not used. The QSSL pin can be reassigned to another function, so it is possible to set the QSSL pin as a general output port and assign it as the slave device select output.
2. The RX64M, RX65N, RX71M, RX72M, RX72N and RX66N have 32 8-bit buffers each for transmission and reception, so half of the buffer count is 16.

6.3.21 r_qspi_smstr_rx_disable()

(1) Purpose

Disables transmit/receive on the QSPI.

(2) Functionality

Disables transmit/receive operation. After switching pin functions from QSPI functions to general I/O ports, makes settings to disable transmit/receive.

1. Disables peripheral functions of pins. (Refer to the description of r_qspi_smstr_mpc_disable().)
2. Clears SPE in SPCR (functionality disabled).
   Clears SPE to disable QSPI functionality.
   — SPCR ← 08h: QSPI functionality disabled, some QSPI functionality initialized, master mode (QSPCLK pin output enabled)
3. Clears SPSSLF/SPTEF/SPRFF in SPSR. (Refer to the description of r_qspi_smstr_spsr_clear().)
4. Sets SPSCR to value before reset.
   — SPSCR ← 00b: SPCMD0

(3) Notes

This function forms a pair with r_qspi_smstr_rx_enable_dual() or r_qspi_smstr_rx_enable_quad().

6.3.22 r_qspi_smstr_buffer_reset()

(1) Purpose

Resets the transmit buffer and receive buffer.

(2) Functionality

Uses SPBFCR to reset the transmit buffer and receive buffer, then sets both to the normal operating state.

(3) Notes

None

6.3.23 r_qspi_smstr_datasize_set()

(1) Purpose

Sets the transmit buffer data count trigger, receive buffer data count trigger, and transmit/receive data count.

(2) Functionality

Uses SPBFCR to set the transmit buffer data count trigger and receive buffer data count trigger. Then sets the transmit/receive data count by setting the transfer data length in SPB of SPCMDn (n = 0 to 2) and the transfer data length multiple in SPBMULn (n = 0 to 2).

1. Sets the transmit/receive buffer data count trigger, transfer data length, and transfer data length multiple.
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*[1] and is not a multiple of 16 >
   — SPBFCR ← 1Dh: Transmit buffer data count trigger 16 bytes, receive buffer data count trigger 16 bytes
   — SPCMD0.SPB ← 0010b: Sequence 0 transfer data length 32 bits (4 bytes)
   — SPBMUL0 sequence 0 transfer data length multiple setting:
     Equal to (total data count / half of QSPI buffer count) * (half of QSPI buffer count / transfer data length)
   — SPCMD1.SPB ← 0000b: Sequence 1 transfer data length 8 bits (1 byte)
   — SPBMUL1 sequence 1 transfer data length multiple setting:
     Equal to remainder of (total data count / half of QSPI buffer count)
   < When the data count is equal to half or more the QSPI transmit/receive buffer count*[1] and is a multiple of 16 >
   — SPBFCR ← 1Dh: Transmit buffer data count trigger 16 bytes, receive buffer data count trigger 16 bytes
   — SPCMD0.SPB ← 0010b: Sequence 0 transfer data length 32 bits (4 bytes)
   — SPBMUL0 sequence 0 transfer data length multiple setting:
     Equal to (total data count / half of QSPI buffer count) * (half of QSPI buffer count / transfer data length)
   < When the data count is equal to less than half the QSPI transmit/receive buffer count*[1] >
   — SPBFCR ← 30h: Transmit buffer data count trigger 0 bytes, receive buffer data count trigger 1 byte
   — SPCMD0.SPB ← 0000b: Sequence 0 transfer data length 8 bits (1 byte)
   — SPBMUL0 sequence 0 transfer data length multiple setting: Equal to total data count

(3) Notes

1. The RX64M, RX65N, RX71M, RX72M, RX72N and RX66N have 32 8-bit buffers each for transmission and reception, so half of the buffer count is 16.

6.4 Details of Functions of Target Microcontroller Dev Layer

Details of the target microcontroller dev layer functions used by the QSPI FIT module are listed below.

6.4.1 r_qspi_smstr_io_init()

(1) Purpose

Sets the pins to be used as general I/O ports, then sets the QIO0 to QIO3 pins to the port input state and QSPCLK pin to the port output state.

(2) Functionality

1. Sets the pins to be used as general I/O ports.
   Refer to the description of r_qspi_smstr_mpc_disable()
2. Sets the QSPCLK pin to port "H" output.
   Refer to the description of r_qspi_smstr_clk_init()
3. Sets the QIO0 to QIO3 pins to port input.
   Refer to the description of r_qspi_smstr_dataio0_init(), r_qspi_smstr_dataio1_init(), r_qspi_smstr_dataio2_init(), and r_qspi_smstr_dataio3_init()

(3) Notes

Changes pin settings from peripheral functions to general I/O functions. Confirm that other peripheral functions are not in use before calling this function.

6.4.2 r_qspi_smstr_io_reset()

(1) Purpose

Sets the pins to be used as general I/O ports, then sets the input and output pins to the port input state.

(2) Functionality

Sets the pins to be used as general I/O ports, then sets the QIO0 to QIO3 pins and QSPCLK pin to the port input state.

1. Sets the pins to be used as general I/O ports.
   Refer to the description of r_qspi_smstr_mpc_disable()
2. Sets the QSPCLK pin to port input.
   Refer to the description of r_qspi_smstr_clk_reset()
3. Sets the QIO0 to QIO3pins to port input.
   Refer to the description of r_qspi_smstr_dataio0_reset(), r_qspi_smstr_dataio1_reset(), r_qspi_smstr_dataio2_reset(), and r_qspi_smstr_dataio3_reset()

(3) Notes

This function forms a pair with r_qspi_smstr_io_init().

6.4.3 r_qspi_smstr_mpc_enable()

(1) Purpose

Makes QSPI function settings for the pins to be used.

(2) Functionality

As described in the Procedure for Specifying Input/Output Pin Functions item in the Multi-Function Pin Controller (MPC) section of the microcontroller User's Manual: Hardware, this function performs the following steps to make register settings.

1. Clears the corresponding bit in the port mode register (PMR) to 0 to set a pin as a general I/O port.
   — PMR bits for QIO0 to QIO3 and QSPCLK pins ← 0b: Used as general I/O port
2. Sets the write-protect register (PWPR) to enable writing to the Pxn pin function control register (PxnPFS).
   — PWPR.B0WI ← 0b: Writing to PFSWE bit enabled
   — PWPR.PFSWE ← 1b: Writing to PFS register enabled
3. Specifies QSPI pin functions by setting the PxnPFS.PSEL[4:0] bit field.
   — PxnPFS for QIO0 to QIO3 pins ← 1Bh: Usable as QIO0 to QIO3 pins*[1]
   — PxnPFS for QSPCLK pin ← 1Bh: Usable as QSPCLK pin*[1]
4. Clears the PWPR.PFSWE bit to 0 to disable writing to the PxnPFS register.
   — PWPR.PFSWE ← 0b: Writing to PFS register disabled
   — PWPR.B0WI ← 1b: Writing to PFSWE bit disabled
5. Change the PMR setting value to 1 for each pin to switch to QSPI pin functions.
   — PMR bits for QIO0 to QIO3 and QSPCLK pins ← 1b: Used for QSPI functions

(3) Notes

Settings are made to the Pxn pin function control register (PxnPFS) while the PMR register bits corresponding to the target pins are cleared to 0. Making settings to the PxnPFS register while the PMR register bit corresponding to the target pin is set to 1 can cause unintended edge input if the pin is set to input or unintended pulse output if the pin is set to output.

1. The setting value may differ according to the microcontroller used.

6.4.4 r_qspi_smstr_mpc_disable()

(1) Purpose

Makes QSPI function settings for the pins to be used.

(2) Functionality

As described in the Procedure for Specifying Input/Output Pin Functions item in the Multi-Function Pin Controller (MPC) section of the microcontroller User's Manual: Hardware, this function performs the following steps to make register settings.

1. Clears the corresponding bit in the port mode register (PMR) to 0 to set a pin as a general I/O port.
   — PMR bits for QIO0 to QIO3 and QSPCLK pins ← 0b: Used as general I/O port
2. Sets the write-protect register (PWPR) to enable writing to the Pxn pin function control register (PxnPFS).
   — PWPR.B0WI ← 0b: Writing to PFSWE bit enabled
   — PWPR.PFSWE ← 1b: Writing to PFS register enabled
3. Specifies port pin functions by setting the PxnPFS.PSEL[4:0] bit field.
   — PxnPFS for QIO0 to QIO3 and QSPCLK pins ← 00h: Hi-Z (value after reset)
4. Clears the PWPR.PFSWE bit to 0 to disable writing to the PxnPFS register.
   — PWPR.PFSWE ← 0b: Writing to PFS register disabled
   — PWPR.B0WI ← 1b: Writing to PFSWE bit disabled

(3) Notes

Settings are made to the Pxn pin function control register (PxnPFS) while the PMR register bits corresponding to the target pins are cleared to 0. Making settings to the PxnPFS register while the PMR register bit corresponding to the target pin is set to 1 can cause unintended edge input if the pin is set to input or unintended pulse output if the pin is set to output.

6.4.5 r_qspi_smstr_dataio0_init()

6.4.6 r_qspi_smstr_dataio1_init()

6.4.7 r_qspi_smstr_dataio2_init()

6.4.8 r_qspi_smstr_dataio3_init()

(1) Purpose

Sets the QIOn (n = 0 to 3) pin to the port input state.

(2) Functionality

1. Uses the open line control register (ODRn) to set the output state of the QIOn pin to CMOS output.
   — QIOn pin ODR ← 0b: CMOS output
2. Uses the pull-up control register (PCR) to disable input pull-up for the QIOn pin.
   — QIOn pin PCR ← 0b: Input pull-up disabled
3. Uses the drive capacity control register (DSCR) to specify the port drive capacity of the QIOn pin.
   The following settings are recommended, according to the conditions imposed by the AC timing characteristics of the microcontroller used.*1*2
   RX64M, RX71M, RX65N, RX72M, RX72N, RX66N
   Sets the port drive capacity of the QIOn pin to "high-drive output."
   — QIOn pin DSCR ← 1b: High-drive output
4. Uses the port direction register (PDR) to set the QIOn pin to port input.
   — QIOn pin PDR ← 0b: Input port

(3) Notes

1. Depending on the microcontroller used, the output low-level allowable current value (I_{OL}) and output low-level (V_{OL}) characteristics may differ for normal output and high-drive output. Use the setting that is appropriate for the connected device.
2. The pins that can be used with the drive capacity control register (DSCR) are limited.

6.4.9 r_qspi_smstr_dataio0_reset()
6.4.10 r_qspi_smstr_dataio1_reset()
6.4.11 r_qspi_smstr_dataio2_reset()
6.4.12 r_qspi_smstr_dataio3_reset()

(1) Purpose

Sets the QIOn (n = 0 to 3) pin to the port input state.

(2) Functionality

1. Uses the port direction register (PDR) to set the QIOn pin to port output.
   — QIOn pin ODR ← 0b: Input port
2. Uses the drive capacity control register (DSCR) to set the port drive capacity of the QIOn pin to normal output (value after a reset).
   RX64M, RX71M, RX65N, RX72M, RX72N, RX66N
   Sets the port drive capacity of the QIOn pin to "normal output."
   — QIOn pin DSCR ← 0b: Normal output
3. Uses the pull-up control register (PCR) to disable input pull-up for the QIOn pin.
   — QIOn pin PCR ← 0b: Input pull-up disabled
4. Uses the open drain control register (ODRn) to set the output state of the QIOn pin to CMOS output (value after a reset).
   — QIOn pin ODR ← 0b: CMOS output

(3) Notes

None

6.4.13 r_qspi_smstr_clk_init()

(1) Purpose

Sets the QSPCLK pin to port "H" output.

(2) Functionality

1. Uses the open drain control register (ODRn) to set the output state of the QSPCLK pin to CMOS output.
   — QSPCLK pin ODR ← 0b: CMOS output
2. Uses the drive capacity control register (DSCR) to specify the port drive capacity of the QSPCLK pin.
   The following settings are recommended, according to the conditions imposed by the AC timing characteristics of the microcontroller used.*1*2
   RX64M, RX71M, RX65N, RX72M, RX72N, RX66N
   Sets the port drive capacity of the QSPCLK pin to "high-drive output."
   — QSPCLK pin DSCR ← 1b: High-drive output
3. Uses the port output data register (PODR) to set the QSPCLK pin to high-level output.
   — QSPCLK pin PODR ← 1b: High-level output
4. Uses the port direction register (PDR) to set the QSPCLK pin to port output.
   — QSPCLK pin PDR ← 1b: Output port

(3) Notes

1. Depending on the microcontroller used, the output low-level allowable current value (I_{OL}) and output low-level (V_{OL}) characteristics may differ for normal output and high-drive output. Use the setting that is appropriate for the connected device.
2. The pins that can be used with the drive capacity control register (DSCR) are limited.

6.4.14 r_qspi_smstr_clk_reset()

(1) Purpose

Sets the QSPCLK pin to the port input state.

(2) Functionality

1. Uses the port direction register (PDR) to set the QSPCLK pin to port input.
   — QSPCLK pin PDR ← 0b: Input port
2. Uses the port output data register (PODR) to set the QSPCLK pin to low-level output (value after a reset).
   — QSPCLK pin PODR ← 0b: Low-level output
3. Uses the drive capacity control register (DSCR) to set the port drive capacity of the QSPCLK pin to normal output (value after a reset).
   RX64M, RX71M, RX65N, RX72M, RX72N, RX66N
   Sets the port drive capacity of the QSPCLK pin to "normal output."
   — QSPCLK pin DSCR ← 0b: Normal output
4. Uses the open drain control register (ODRn) to set the output state of the QSPCLK pin to CMOS output (value after a reset).
   — QSPCLK pin ODR ← 0b: CMOS output

(3) Notes

None

6.4.15 r_qspi_smstr_module_enable()

(1) Purpose

Cancels the module stop state.

(2) Functionality

Cancels the module stop state.

1. Uses the protect register (PRCR) and module stop control register (MSTPCRB) to cancel the module stop state.
   — PRCR ← A502h: Module stop control register protect canceled
   — MSTPCRB.MSTPBxx ← 0b: Module stop canceled, reading/writing to QSPI registers enabled
   — PRCR ← A500h: Module stop control register protect enabled

(3) Notes

It is not possible to read from or write to the registers of a module that is in the module stop state. After this function is run, the QSPI registers can be accessed. Note that register values are maintained when in the module stop state.

6.4.16 r_qspi_smstr_module_disable()

(1) Purpose

Enables the module stop state.

(2) Functionality

1. Uses the protect register (PRCR) and module stop control register (MSTPCRB) to cancel the module stop state, allowing QSPI functions to make register settings.
   — PRCR ← A502h: Module stop control register protect canceled
   — MSTPCRB.MSTPBxx ← 1b: Module stopped, reading/writing to QSPI registers disabled
   — PRCR ← A500h: Module stop control register protect enabled

(3) Notes

None

6.4.17 r_qspi_smstr_tx_dmacdtc_wait()

(1) Purpose

Performs DMAC/DTC transmit-end wait processing.

(2) Functionality

A software loop is used to wait for the end of DMAC/DTC transmission. The loop is run 50,000 times. If the amount of data to be sent at once is large, the loop count may exceed the upper limit before a successful transfer-end. If necessary, modify the processing code by increasing the loop count, etc., as appropriate.

(3) Note

The QSPI FIT module does not include the DMAC FIT module or DTC FIT module. These must be obtained separately.

When modifying the processing code, ensure that a value of QSPI_SMSTR_ERR_HARD is returned in case of error.

6.4.18 r_qspi_smstr_rx_dmacdtc_wait()

(1) Purpose

Performs DMAC/DTC receive-end wait processing.

(2) Functionality

A software loop is used to wait for the end of DMAC/DTC transmission. The loop is run 50,000 times. If the amount of data to be sent at once is large, the loop count may exceed the upper limit before a successful transfer-end. If necessary, modify the processing code by increasing the loop count, etc., as appropriate.

(3) Note

The QSPI FIT module does not include the DMAC FIT module or DTC FIT module. These must be obtained separately.

When modifying the processing code, ensure that a value of QSPI_SMSTR_ERR_HARD is returned in case of error.

6.4.19 r_qspi_smstr_int_spti_init()

(1) Purpose

Initializes the SPTI interrupt.

(2) Functionality

1. IENx setting
   SPTI interrupt request enable bit: Disable
2. IR setting
   SPTI interrupt status flag: Clear

(3) Note

None

6.4.20 r_qspi_smstr_int_spri_init()

(1) Purpose

Initializes the SPRI interrupt.

(2) Functionality

1. IENx setting
   SPRI interrupt request enable bit: Disable
2. IR setting
   SPRI interrupt status flag: Clear

(3) Note

None

6.4.21 r_qspi_smstr_int_spti_ier_set()

(1) Purpose

Sets the ICU.IERm.IENj bit of the SPTI interrupt.

(2) Functionality

1. IENx setting
   SPTI interrupt request enable bit: Enable

(3) Note

None

6.4.22 r_qspi_smstr_int_spri_ier_set()

(1) Purpose

Sets the SPRI interrupt request ICU.IERm.IENj bit.

(2) Functionality

1. IENx setting
   SPRI interrupt request enable bit: Enable

(3) Note

None

6.5 Note on Drive Capacity Control Register (DSCR) Setting

Depending on the microcontroller used, the output low-level allowable current value (I_{OL}) and output low-level (V_{OL}) characteristics may differ for normal output and high-drive output. Use the setting that is appropriate for the connected device.

6.6 Port Functions of QSPI Pins on Each Microcontroller

Depending on the microcontroller used, the method of controlling the open drain control register (ODR) and drive capacity control register (DSCR) differs. Table 6-8 lists the port functions of the QSPI pins on each microcontroller.

Table 6-8 Port Functions of QSPI Pins on Each Microcontroller

MCU	Channel	Pin	Port Number	Open Drain Control Register (ODR)	Drive Capacity Control Register (DSCR)
RX64M RX65N RX71M	QSPI0	QSPCLK	P77	CMOS/Open drain	Fixed at high-drive
			PD5	CMOS/Open drain	Normal/High-drive
		QMO/QIO0	PC3	CMOS/Open drain	Normal/High-drive
			PD6	CMOS/Open drain	Normal/High-drive
		QMO/QIO1	PC4	CMOS/Open drain	Normal/High-drive
			PD7	CMOS/Open drain	Normal/High-drive
		QIO2	P80	CMOS/Open drain	Fixed at high-drive
			PD2	CMOS/Open drain	Normal/High-drive
		QIO3	P81	CMOS/Open drain	Fixed at high-drive
			PD3	CMOS/Open drain	Normal/High-drive
RX72M RX72N RX66N	QSPI0	QSPCLK	P77	CMOS/Open drain	Normal/High-drive
			PD5	CMOS/Open drain	Normal/High-drive
			PM0	CMOS/Open drain	Normal/High-drive
			PN4	CMOS/Open drain	Normal/High-drive
		QMO/QIO0	PC3	CMOS/Open drain	Normal/High-drive
			PD6	CMOS/Open drain	Normal/High-drive
			PJ3	CMOS/Open drain	Fixed at high-drive
			PM2	CMOS/Open drain	Normal/High-drive
		QMO/QIO1	PC4	CMOS/Open drain	Normal/High-drive
			PD7	CMOS/Open drain	Normal/High-drive
			PJ5	CMOS/Open drain	Fixed at high-drive
			PM3	CMOS/Open drain	Normal/High-drive
		QIO2	P00	CMOS/Open drain	Normal/High-drive
			P80	CMOS/Open drain	Normal/High-drive
			PD2	CMOS/Open drain	Normal/High-drive
			PM4	CMOS/Open drain	Normal/High-drive
		QIO3	P01	CMOS/Open drain	Normal/High-drive
			P81	CMOS/Open drain	Normal/High-drive
			PD3	CMOS/Open drain	Normal/High-drive
			PM5	CMOS/Open drain	Normal/High-drive

7. Reference Documents

User's Manual: Hardware

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

Technical Update/Technical News

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

User's Manual: Development Tools
[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User's Manual: RX Build Rev.1.00 (R20UT02747)
[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User's Manual: RX Coding Rev.1.00 (R20UT02748)
[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User's Manual: Message Rev.1.00 (R20UT02749)

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

Technical Update

The following technical update applies to this module.

- TN-RX*-A145A/E
  It has already been met from Rev.1.00.

Revision History

Rev.	Date	Description Page	Summary
1.00	Jul 31, 2014	—	First edition issued
1.06	Aug 29, 2014	—	Matched Rev. of the application note with Ver. of the source code.
		—	Revised r_qspi_smstr_private.h of the src directory.
		—	Revised r_qspi_smstr.c of the src directory.
		—	Added demo source for DTC in demo directory.
		5	1.4 Related Application Note: Added new content
		75	5. Reference Documents: Added new item
1.08	Dec 26, 2014	-	Changed DMAC/DTC to DMAC or DTC.
		-	Changed folder/file name from ‘r_qspi_smstr’ to ‘r_qspi_smstr_rx’.
		1	Added RX71M Group in Target Devices.
		1	Added an application note (R01AN1826EU) in Related Documents.
		3	Added R_QSPI_SMstr_1ms_Interval() in Table 2.1 and Note 2 in 1.2.1 Overview of APIs.
		4	Changed type name of ‘Board used’ in (1)RX64M, 1.2.2 Operating Environment and Memory Sizes.
		4	Changed type size of ‘Max. interrupt stack’ in (1)RX64M, 1.2.2 Operating Environment and Memory Sizes.
		5	Added (2)RX71M, 1.2.2 Operating Environment and Memory Sizes.
		6	Added the following application notes (4)RX71M, 1.3 Related Application Note. -Compare Match Timer Module Using Firmware Integration Technology (R01AN1856EU) -EEPROM Access Clock Synchronous control module Using Firmware Integration Technology (R01AN2325EJ) -General Purpose Input/Output Driver Module Using Firmware Integration Technology (R01AN1721EU) -Multi-Function Pin Controller Module Using Firmware Integration Technology (R01AN1724EU)
		9	Changed file name in 1.6.4 Software Structure (b).
		9	Changed file name in 1.6.4 Software Structure (c).
		9	Added ‘(R01AN2325EJ)’ and ‘The SPI serial EEPROM Control Software has driver interface functions (r_eeprom_spi_drvif_devX.c: X=0 or 1) to incorporate the QSPI FIT module.’ in 1.6.4 Software Structure (d).
		25	Changed state in 1.6.8 State Transition Diagram.
		26	Changed r_cgc_rx in 2.2 Software Requirements.
		26	Added r_dmaca_rx, r_dtc_rx, r_cmt_rx, r_gpio_rx and r_mpc_rx in 2.2 Software Requirements.
		26	Changed header file in 2.4 Header Files
		27	Separated the contents of r_qspi_smstr_rx_config.h to r_qspi_smstr_rx_config.h and r_qspi_smstr_rx_pin_config.h in 2.6 Compile Settings.
		27	Changed name of definition in 2.6 Compile Settings.
		29	Changed contents of 7.a and 7.b in 2.9.1 Adding the QSPI FIT module (when not using the plug-in).

		29	Added r_qspi_smstr_rx_pin_config_reference.h in 2.9.1 Adding the QSPI FIT module (when not using the plug-in).
		32	Changed explanation in 2.10.3 DMAC/DTC.
		32	Added 2.10.4 CMT.
		33	Changed explanation in 2.11 Using the Module in Other Than an FIT Module Environment.
		33	Added 2.12 Pin States.
		34	Changed “general output ports” to “general output ports in the high-output state” at description in 3.1 R_QSPI_SMster_Open().
		36	Changed “the QSPCLK pin is set to the same state as before the reset (general I/O port)” to “the QSPCLK is set as general output ports in the high-output state” at description in 3.2 R_QSPI_SMster_Close().
		40	Added contents at Special Notes in 3.4 R_QSPI_SMster_Write().
		42	Added contents at Special Notes in 3.5 R_QSPI_SMster_Read().
		51	Added “Executes the end processing of fetching the error log using LONGQ FIT module.” and “For the setting of the error log size, refer to the LONGQ FIT module.” at Description in 3.13 R_QSPI_SMster_Log().
		53	Added 3.15 R_QSPI_SMster_1ms_Interval().
		59 61 62 64 65 67	Added RX71M in (3)Notes, 4.1.14 r_qspi_smstr_trx_enable_single() 4.1.16 r_qspi_smstr_tx_enable_dual() 4.1.17 r_qspi_smstr_tx_enable_quad() 4.1.19 r_qspi_smstr_rx_enable_dual() 4.1.20 r_qspi_smstr_rx_enable_quad() 4.1.23 r_qspi_smstr_datasize_set().
		71	Added RX71M in (2)Functionality, 4.2.5 r_qspi_smstr_datao0_init() 4.2.6 r_qspi_smstr_datao1_init() 4.2.7 r_qspi_smstr_datao2_init() 4.2.8 r_qspi_smstr_datao3_init().
		72	Added RX71M in (2)Functionality, 4.2.9 r_qspi_smstr_datao0_reset() 4.2.10 r_qspi_smstr_datao1_reset() 4.2.11 r_qspi_smstr_datao2_reset() 4.2.12 r_qspi_smstr_datao3_reset().
		72	Added RX71M in (2)Functionality, 4.2.13 r_qspi_smstr_clk_init().
		73	Added RX71M in (2)Functionality, 4.2.14 r_qspi_smstr_clk_reset().
		76	Added RX71M in 4.4 Port Functions of QSPI Pins on Each Microcontroller.
		76	Moved content for SPI serial EEPROM from 4.5 Sample program in Demo Folder to 1.6.4 Software Structure. Removed content for QSPI serial Flash memory.
1.09	Sep 30, 2016	1	Added RX65N Group in Target Devices.
		6	Added (3) RX65N, 1.2.2 Operating Environment and Memory Sizes.
		28	Changed #define name in 2.6 Compile Settings.

		28-29	Added “RX64M QSPI” in 2.6 Compile Settings.
		30	Updated explanation in 2.9 Adding Driver to Your Project.
		55	Moved 2.12 Pin States to 4.Pin Setting.
		61-69	Added RX65N in (3)Notes, 5.1.14 r_qspi_smstr_trx_enable_single() 5.1.16 r_qspi_smstr_tx_enable_dual() 5.1.17 r_qspi_smstr_tx_enable_quad() 5.1.19 r_qspi_smstr_rx_enable_dual() 5.1.20 r_qspi_smstr_rx_enable_quad() 5.1.23 r_qspi_smstr_datasize_set().
		73-75	Added RX65N in (2)Functionality, 5.2.5 r_qspi_smstr_dataio0_init() 5.2.6 r_qspi_smstr_dataio1_init() 5.2.7 r_qspi_smstr_dataio2_init() 5.2.8 r_qspi_smstr_dataio3_init() 5.2.9 r_qspi_smstr_dataio0_reset() 5.2.10 r_qspi_smstr_dataio1_reset() 5.2.11 r_qspi_smstr_dataio2_reset() 5.2.12 r_qspi_smstr_dataio3_reset() 5.2.13 r_qspi_smstr_clk_init() 5.2.14 r_qspi_smstr_clk_reset()
		78	Added RX65N in 5.4 Port Functions of QSPI Pins on Each Microcontroller.
1.10	Jul 31, 2017	1	Added RX651 Group in Target Devices.
		-	Changed the following chapter. - Changed 1.1 QSPI FIT Module. - Changed 1.2 Overview of QSPI FIT Module. - Changed 1.4 Example. - Changed 1.5 State Transition Diagram. -Changed 2. API Information. - Moved from 1.2 Overview and Memory Size of APIs to 5.1 Operating Confirmation Environment. - Moved form 5. Related Application Notes to 6. Related Application Notes. Added the following chapter. - Added 2.4 Interrupt vector. - Added 2.8 Code Size. - Added 2.11 Callback function. - Added 2.12 Adding FIT Module to your Project. - Added 5.2 Trouble Shooting.
		26	Deleted r_cgc_rx in 2.2 Software Requirements.
1.11	Feb 01, 2019	57	Added Table 5-1 Operation Confirmation Conditions(Ver.1.11).
		-	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
1.12	May 20, 2019	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Deleted R01AN1723 and R01AN1826 from Related

			Documents.
		1	Added Target Compilers.
		26	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		30	2.8 Code Size, amended.
		56	Changed Pin Setting macro definition in 4.Pin Setting.
		58	Added Table 5-3 Operation Confirmation Conditions(Ver.1.12).
1.13	Jul 30, 2019	1	Added RX72M Group in Target Devices.
		2	Deleted R01AN1833 from Related Documents.
		27	Added RX72M's Interrupt Vector in Table 2-2.
		30	2.8 Code Size, amended.
		37	Added Section 2.15 "for", "while" and "do while" statements.
		38-55	Deleted the Reentrancy for each API in 3 API Functions.
		59	Added Table 5-4 Operation Confirmation Conditions(Ver.1.13).
		66-74	Added RX72M in (3) Notes, 5.3.14 r_qspi_smstr_trx_enable_single() 5.3.16 r_qspi_smstr_tx_enable_dual() 5.3.17 r_qspi_smstr_tx_enable_quad() 5.3.19 r_qspi_smstr_rx_enable_dual() 5.3.20 r_qspi_smstr_rx_enable_quad() 5.3.23 r_qspi_smstr_datasize_set().
		78-80	Added RX72M in (2) Functionality, 5.4.5 r_qspi_smstr_dataio0_init() 5.4.6 r_qspi_smstr_dataio1_init() 5.4.7 r_qspi_smstr_dataio2_init() 5.4.8 r_qspi_smstr_dataio3_init() 5.4.9 r_qspi_smstr_dataio0_reset() 5.4.10 r_qspi_smstr_dataio1_reset() 5.4.11 r_qspi_smstr_dataio2_reset() 5.4.12 r_qspi_smstr_dataio3_reset() 5.4.13 r_qspi_smstr_clk_init() 5.4.14 r_qspi_smstr_clk_reset().
		83	Added RX72M in Table 5-6 Port Functions of QSPI Pins on Each Microcontroller.
1.14	Nov 22, 2019	1	Added RX72N and RX66N Group in Target Devices.
		28	Added RX72N and RX66N's Interrupt Vector in Table 2-2.
		31	2.8 Code Size, amended.
		53-55	Changed "Special Notes" in 3.12 R_QSPI_SMstr_Set_LogHdlAddress and 3.13 R_QSPI_SMstr_Log.
		61	Added Table 5-5 Operation Confirmation Conditions(Ver.1.14).
		68-76	Added RX72N and RX66N in (3) Notes, 5.3.14 r_qspi_smstr_trx_enable_single() 5.3.16 r_qspi_smstr_tx_enable_dual() 5.3.17 r_qspi_smstr_tx_enable_quad() 5.3.19 r_qspi_smstr_rx_enable_dual() 5.3.20 r_qspi_smstr_rx_enable_quad() 5.3.23 r_qspi_smstr_datasize_set().
		80-82	Added RX72N and RX66N in (2) Functionality, 5.4.5 r_qspi_smstr_dataio0_init() 5.4.6 r_qspi_smstr_dataio1_init()

			5.4.7 r_qspi_smstr_dataio2_init() 5.4.8 r_qspi_smstr_dataio3_init() 5.4.9 r_qspi_smstr_dataio0_reset() 5.4.10 r_qspi_smstr_dataio1_reset() 5.4.11 r_qspi_smstr_dataio2_reset() 5.4.12 r_qspi_smstr_dataio3_reset() 5.4.13 r_qspi_smstr_clk_init() 5.4.14 r_qspi_smstr_clk_reset().
		85	Added RX72N and RX66N in Table 5-7 Port Functions of QSPI Pins on Each Microcontroller.
1.15	Sep 30, 2021	8	Added restrictions on QSPI FIT pins in Table 1-3 List of Pins Used.
1.20	Sep 30, 2022	61	Added “5. Demo Projects”.
		61-62	Added RSKRX64M, RSKRX65N-2M, RSKRX72N to “5. Demo Projects”.
		65	6.1 Operating Confirmation Environment: Added Table for Rev. 1.20
		Program	Added new demo projects
1.21	Jun 15, 2023	35, 59	Deleted the description of FIT configurator from "2.12 Adding the FIT Module to Your Project", "4. Pin Settings"
		66	6.1 Operating Confirmation Environment: Added Table for Rev. 1.21
		88	Remove QSPI_SMSTR_ERR_PARAM in 6.4.17 r_qspi_smstr_tx_dmacdtc_wait() Remove QSPI_SMSTR_ERR_PARAM in 6.4.18 r_qspi_smstr_rx_dmacdtc_wait()
		Program	Fixed typo in the file r_qspi_smstr_target_dev_port.c for all devices Deleted the description of FIT configurator.

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.

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/.