RX62N Group

Clock Synchronous Single Master Control Software Using the RSPI

R01AN0323EJ0104
Rev.1.04
Nov 13, 2012

Introduction

This application note describes a clock synchronous single master control method that uses RX62N Group Renesas serial peripheral interface (RSPI) clock synchronous (three-wire method) serial communication and sample code that uses that method.

SPI mode single master control can be implemented by adding SPI slave device selection control using port control.

This sample code implements the single master basic control method that is unique to these microcontrollers. The user should implement the software required to control the slave devices using this sample code.

Note that Renesas provides sample software to control slave devices. We recommend acquiring that software and using it in conjunction with this sample code.

Target Devices

Target microcontroller: RX62N Group microcontrollers

Devices used in verifying operation: Renesas R1EX25xxx Series SPI Serial EEPROM.

When using this application note’s sample code with another microcontroller, the code must be modified to match the specifications of the microcontroller used and tested thoroughly.

Contents

1. Specifications 2
2. Verified Operating Conditions 3
3. Related Application Notes 4
4. Peripheral Functions 4
5. Hardware Description 5
6. Software Description 6
7. Sample Application 37
8. Usage Notes 45

1. Specifications

This sample program uses RX62N Group microcontroller RSPI clock synchronous (three-wire method) serial communications to perform clock synchronous control. SPI mode single master control can be implemented by adding SPI slave device selection control using port control.

Table 1.1 lists the used peripheral functions and their uses and figure 1.1 shows an example of the use of this application.

In the following, we present an overview of these functions.

- An RX62N is used as the master device and the sample program implements a block type device driver for clock synchronous single master communication using the RSPI module.
- The microcontroller's built-in clock synchronous (three-wire method) serial communications function is used. A single channel set up by the user can also be used. Multiple channels cannot be used.
- This sample code does not support chip select control. If an SPI device is controlled, it will be necessary to provide device select control code separately.
- This sample code supports both big endian and little endian byte orders.
- Data is transferred in an MSB first format.
- Only CPU transfers are supported. DMAC, EXDMAC, and DTC transfers are not supported.
- Using interrupts to start transfers is not supported.
- Either normal receive mode or high-speed receive mode can be selected as the reception method.
- Operation in supervisor mode is possible when high-speed receive mode is selected. Operation in user mode is not possible.
  Operation in either supervisor mode or user mode is possible when normal receive mode is selected.
- The NMI interrupt must be disabled in high-speed receive mode.

Table 1.1 Peripheral Devices and Uses

Peripheral Device	Use
RSPI	Clock synchronous (three-wire method) serial communications: 1 channel (required)
Port	Used for SPI slave device selection control A number of ports corresponding to the number of devices used are needed (required). Note, however, that ports are not used in this sample code.

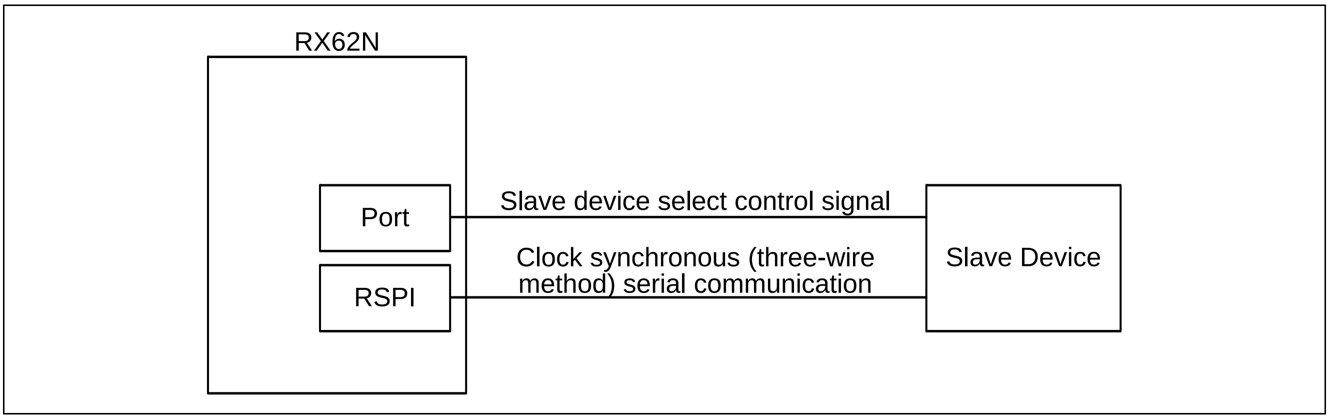


Figure 1.1 Usage Example

2. Verified Operating Conditions

Operation of this application note's sample code has been verified under the following conditions.

Table 2.1 Verified Operating Conditions

Item	Description
Microcontroller	RX62N Group microcontrollers (Program ROM: 512 KB, RAM: 96 KB)
Memory	Renesas Electronics Corporation R1EX25xxx Series SPI Serial EEPROM
Operating frequency	ICLK: 96 MHz, PCLK: 48 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics Corporation High-performance embedded Workshop Version 4.09.00.007
C compiler	Renesas Electronics Corporation RX Family C/C++ Compiler Package (Toolchain 1.0.1.0)
	Compiler options The integrated development environment default settings* are used. Note: * Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version number	Ver. 2.04
Software	Renesas R1EX25xxx Series Serial EEPROM Control Software (R01AN0565EJ), Version 2.01
Board	Renesas Starter Kit for RX62N

Table 2.2 Verified Operating Conditions

Item	Description
Microcontroller	RX62N Group microcontrollers (Program ROM: 512 KB, RAM: 96 KB)
Memory	Micron Technology M25P Series Serial Flash Memory: 64 Mbits
Operating frequency	ICLK: 96 MHz, PCLK: 48 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics Corporation High-performance embedded Workshop Version 4.09.00.007
C compiler	Renesas Electronics Corporation RX Family C/C++ Compiler Package (Toolchain 1.0.1.0)
	Compiler options The integrated development environment default settings* are used. Note: * Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version number	Ver. 2.04
Software	Micron Technology M25P Series Serial Flash Memory Control Software (R01AN0566EJ), Version 2.01
Board	Renesas Starter Kit for RX62N

Table 2.3 Verified Operating Conditions

Item	Description
Microcontroller	RX62N Group microcontrollers (Program ROM: 512 KB, RAM: 96 KB)
Memory	Micron Technology M25P Series Serial Flash Memory: 1 Mbit
Operating frequency	ICLK: 96 MHz, PCLK: 48 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics Corporation High-performance embedded Workshop Version 4.09.00.007
C compiler	Renesas Electronics Corporation RX Family C/C++ Compiler Package (Toolchain 1.0.1.0)
	Compiler options The integrated development environment default settings* are used. Note: * Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version number	Ver. 2.04
Software	Micron Technology M45PE Series Serial Flash Memory Control Software (R01AN0567EJ), Version 2.01
Board	Renesas Starter Kit for RX62N

3. Related Application Notes

Related application notes are listed below. Refer to these when using this application note.

- Renesas R1EX25xxx Series Serial EEPROM Control Software (R01AN0565EJ)
- Micron Technology M25P Series Serial Flash Memory Control Software (R01AN0566EJ)
- Micron Technology M45PE Series Serial Flash Memory Control Software (R01AN0567EJ)

4. Peripheral Functions

The RSPI module supports two types of operation: SPI operation (four-wire method) and clock synchronous operation (three-wire method).

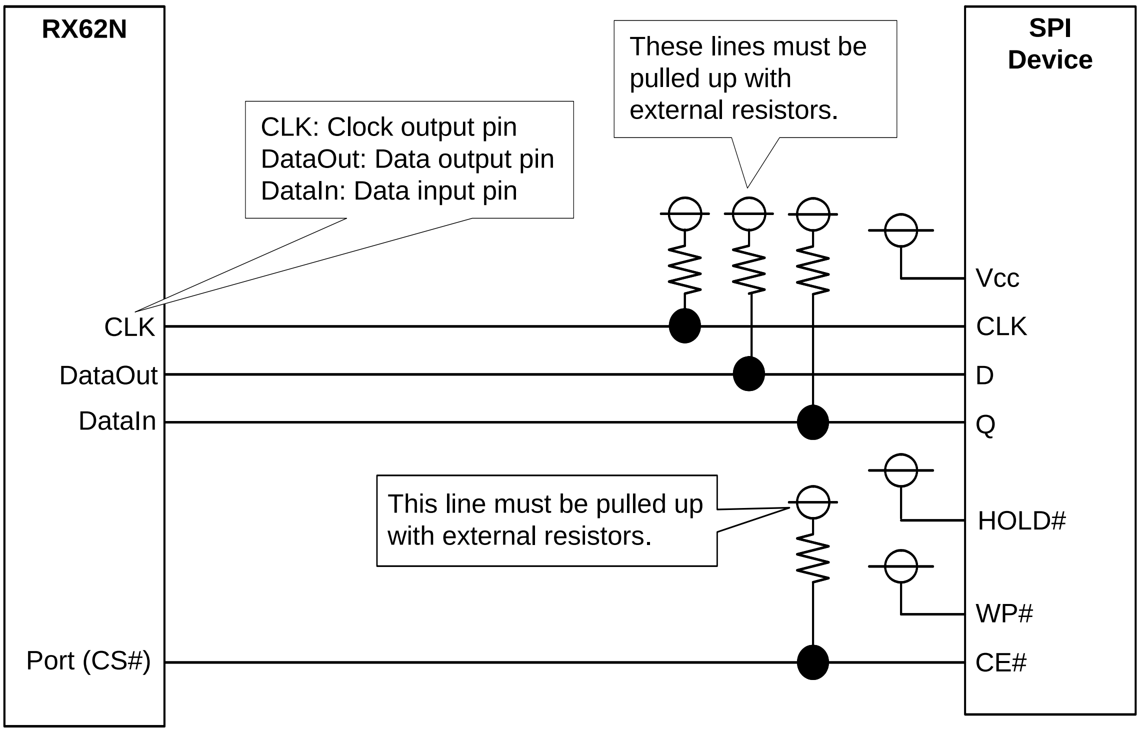
This application note uses clock synchronous operation (three-wire method). In this sample code, a port is allocated as the SPI slave device select pin when an SPI device is controlled.

The SSL pin used with the RSPI four-wire method can be allocated as a CE# pin for port control when three-wire method is used.

5. Hardware Description

5.1 Reference Circuit

Figure 5.1 shows the device connection circuit diagram. Note that if the hardware will be operated at high speed, a damping resistor and capacitor should be added for circuit matching for each signal line.



- The pin names used for serial I/O on the microcontroller depend on the microcontroller used.
- In this application note, the pins used are expressed as the CLK, DataIn, DataOut, and Port (CS#) pins to match the notation used in the sample code.

Figure 5.1 Connection Between RX62N RSPI and SPI Slave Device

5.2 List of Pins

Table 5.1 lists the pins used and their functions.

Table 5.1 Pins and Usage

Pin Name	I/O	Description
RSPCK (CLK in figure 5.1)	Output	Clock output
MOSI (DataOut in figure 5.1)	Output	Master data output
MISO (DataIn in figure 5.1)	Input	Master data input
Port (Port(CS#) in figure 5.1)	Output	Slave device select output Note, however, that this pin is not handled by this sample code.

6. Software Description

6.1 Operation Overview

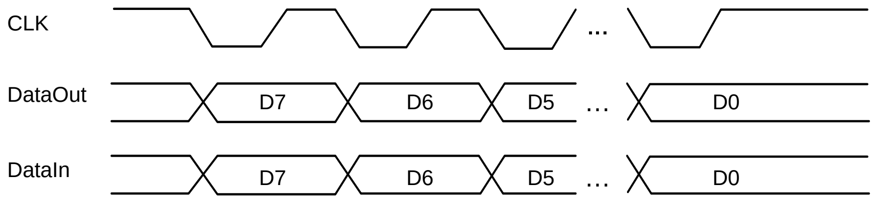
This sample code uses the RSPI module's clock synchronous (three-wire method) serial communication function to implement clock synchronous single master control.

This sample code implements the following control operation.

- Control of data transmit/receive operations in clock synchronous operation (using an internal clock).

6.1.1 Timing Generated in Clock Synchronous Operation

This sample code generates the SPI mode 3 (CPOL = 1, CPHA = 1) timing shown in figure 6.1, which is required for SPI slave device control.



- For microcontroller to SPI slave device transmission: Transmit data output starts on the falling edge of the transfer clock
- For SPI slave device to microcontroller reception: Receive data input is acquired on the rising edge of the transfer clock
- Data is transferred MSB first.

The CLK pin is held at the high level when no transfer is being performed.

Figure 6.1 Timing Settings for Clock Synchronous Operation

Check the microcontroller and SPI slave device datasheets for the serial clock frequencies that can be used.

6.1.2 SPI Slave Device CE# Pin Control

This sample code does not control the SPI slave device CE# pin. To control an SPI device, the user must provide SPI slave device CE# pin control separately.

As the control method, we recommend connecting to a microcontroller port and controlling the SPI device with the microcontroller general-purpose port output.

Also, the application must provide time from the fall of the SPI device CE# (microcontroller port CS#) signal to the fall of the SPI device CLK (the microcontroller CLK) signal.

Similarly, the application must provide time from the rise of the SPI device CLK (the microcontroller CLK) signal to the rise of the SPI device CE# (microcontroller port CS#) signal.

Check the SPI device data sheet, and implement the application with software wait times appropriate for the system.

The SSL pin used in four-wire method with the RSPI module may be allocated to the CE# pin in port control for three-wire method.

6.2 Software Control Outline

6.2.1 Software Structure

This sample code implements a single master basic control method that is unique to the microcontroller.

In particular, this sample code implements control that uses SPI mode 3 (CPOL = 1, CPHA = 1) without control of the SPI slave device CE# pin.

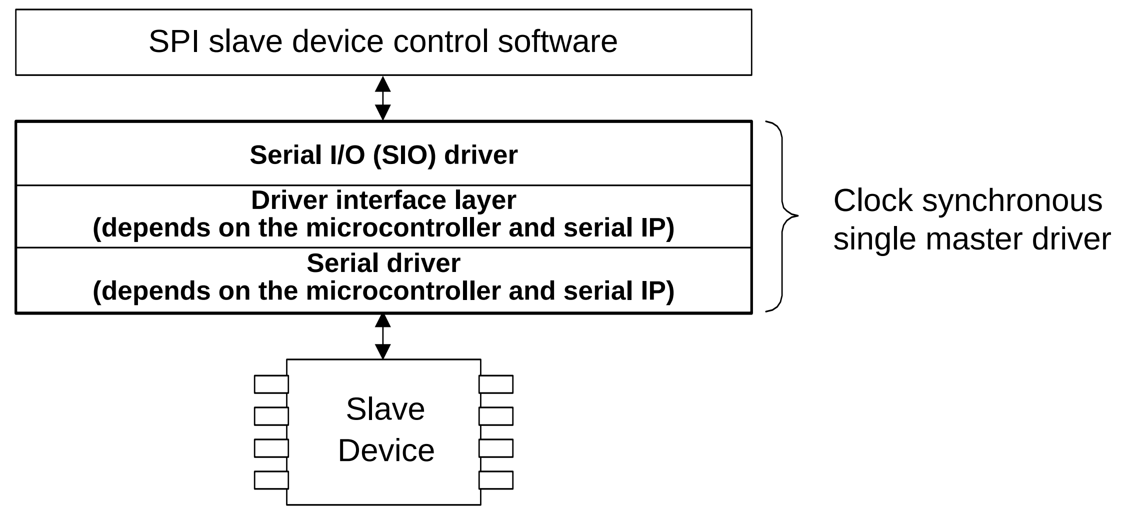


Figure 6.2 Software Structure

The user must implement slave device access by referring to the functions shown in section 6.8, State Transition Diagram, and section 6.9, Function Specifications.

Refer to the previously mentioned section 3, Related Application Notes for specific application examples.

6.2.2 Relationship Between Data Buffers and Transmit/Receive Data

This sample code is a block type device driver and passes the transmit or receive data pointer as an argument. The relationship between the data ordering in the data buffer in RAM and the transmit/receive order is shown below and this sample code both transmits in the order data is stored in the transmit buffer and writes data to the receive data buffer in the order received regardless of the endian order or serial communication function used.

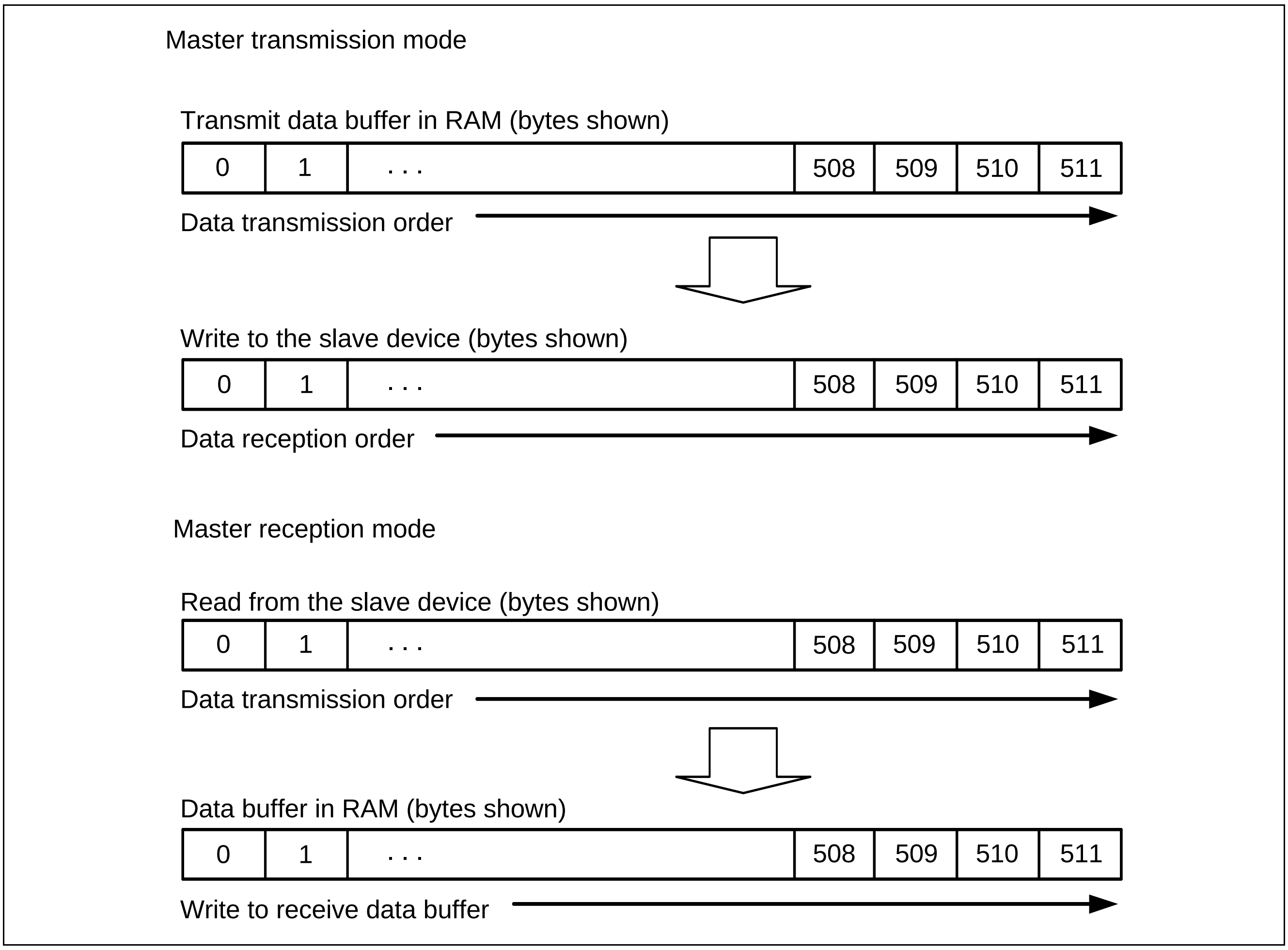


Figure 6.3 Relationship Between Data Buffers and Transmit/Receive Data

6.3 Size of Required Memory

Table 6.1 lists the memory requirements.

The memory sizes listed in table 6.1 apply when SIO_OPTION_4 is selected with the operating mode definition used in section 7.2.2, R_SIO_rspi.h (1). The memory requirements differ depending on the selected definition.

The maximum user stack size applies when the serial EEPROM control software is used and the stack size used by the serial EEPROM control software is included.

Table 6.1 Memory Requirements

Memory Used	Size	Remarks
ROM	1,432 bytes (little endian)	R_SIO_rspi_rx.c
RAM	0 byte (little endian)	R_SIO_rspi_rx.c
Maximum user stack usage	204 bytes	
Maximum interrupt stack usage	—	

The memory requirements may differ with the version of the C compiler used or with the compiler options specified.

The above memory requirements may differ depending on the endian order selected.

6.4 File Configuration

Table 6.2 lists the files used by the sample code. Note that the files automatically generated by the integrated development environment are not included.

Table 6.2 File Configuration

\an_r01an0323ej0104_rx62n_serial <DIR>				Sample code folder
	r01an0323ej0104_rx62n.pdf			Application note
	source <DIR>			Program folder
		\com <DIR>		Common function folder
		Note 1	mtl_com.c	Common function definitions
			mtl_com.h.common	Common header file
			mtl_com.h.RX	Common functions header file
			mtl_endi.c	Common files (endian setting related)
			mtl_mem.c	Common files (Standard library functions)
			mtl_os.c mtl_os.h	Common files (Standard library functions)
			mtl_str.c	Common files (Standard library functions)
			mtl_tim.c mtl_tim.h	Common files (Loop timer related)
			mtl_tim.h.sample	Sample loop timer settings
		\r_sio_rspi_rx <DIR>		Folder for clock synchronous single master control software using the RSPI
			R_SIO.h	Header file
			R_SIO_rspi.h.rx62x	Interface module common definitions
			R_SIO_rspi_rx.c	Interface module

Note: 1. The files held in the com folder are also used by the slave device control software. Use the latest versions of these files.

6.5 List of Constants

6.5.1 Return Values

Table 6.3 lists the return values used in the sample code.

Table 6.3 Return Values

Constant Name	Value	Description
SIO_OK	(error_t)(0)	Successful operation
SIO_ERR_PARAM	(error_t)(-1)	Parameter error
SIO_ERR_HARD	(error_t)(-2)	Hardware error
SIO_ERR_OTHER	(error_t)(-7)	Other error

6.5.2 Definitions

Table 6.4 lists the values for certain definitions used in the sample code.

Table 6.4 Return Values

Constant Name	Value	Description
SIO_LOG_ERR	(uint8_t)0x01	Log type: Error
SIO_TRUE	(uint8_t)0x01	Flag "ON"
SIO_FALSE	(uint8_t)0x00	Flag "OFF"
SIO_HI	(uint8_t)0x01	Port "H"
SIO_LOW	(uint8_t)0x00	Port "L"
SIO_OUT	(uint8_t)0x01	Port output setting
SIO_IN	(uint8_t)0x00	Port input setting
SIO_TX_WAIT	(uint16_t)50000	SIO transmission completion waiting time 50000* 1us = 50m
SIO_RX_WAIT	(uint16_t)50000	SIO reception completion waiting time 50000* 1us = 50ms
SIO_DMA_TX_WAIT	(uint16_t)50000	DMA transmission completion waiting time 50000* 1us = 50ms
SIO_DMA_RX_WAIT	(uint16_t)50000	DMA reception completion waiting time 50000* 1us = 50ms
SIO_T_SIO_WAIT	(uint16_t)MTL_T_1US	SIO transmission&reception completion waiting polling time
SIO_T_DMA_WAIT	(uint16_t)MTL_T_1US	DMA transmission&reception completion waiting polling time
SIO_T_BRR_WAIT	(uint16_t)MTL_T_10US	BRR setting wait time

6.5.3 Other Definitions

Table 6.5 lists the values of certain other definitions used in the sample code.

Table 6.5 Return Values

Constant Name	Value	Description
SIO_TRAN_SIZE	(uint8_t)0x04	4 bytes (This value may not be changed.)

6.6 Structures and Unions

The structures used in the sample code are shown below.

```
/* uint32_t <-> uint8_t conversion */
typedef union {
  uint32_t   ul;
  uint8_t uc[4];
} SIO_EXCHG_LONG;                          /* total 4byte                */


/* uint16_t <-> uint8_t conversion */
typedef union {
  uint16_t   us;
  uint8_t uc[2];
} SIO_EXCHG_SHORT;                         /* total 2byte                */
```

6.7 List of Functions

Table 6.6 lists the functions in the sample code.

Table 6.6 List of Functions

Function Name	Outline
R_SIO_Init_Driver()	Driver initialization
R_SIO_Disable()	Disables serial I/O
R_SIO_Enable()	Enables serial I/O
R_SIO_Open_Port()	Releases serial I/O
R_SIO_Tx_Data()	Transmits serial I/O data
R_SIO_Rx_Data()	Receives serial I/O data

To increase the speed of RSPI control operations, 32-bit access is used for the SPDR registers. When specifying a transmit/receive data buffer pointer, we recommend assuring that the start address falls on a 4-byte boundary to increase the speed of this processing.

6.8 State Transition Diagram

Figure 6.4 shows the state transition diagram for this system.

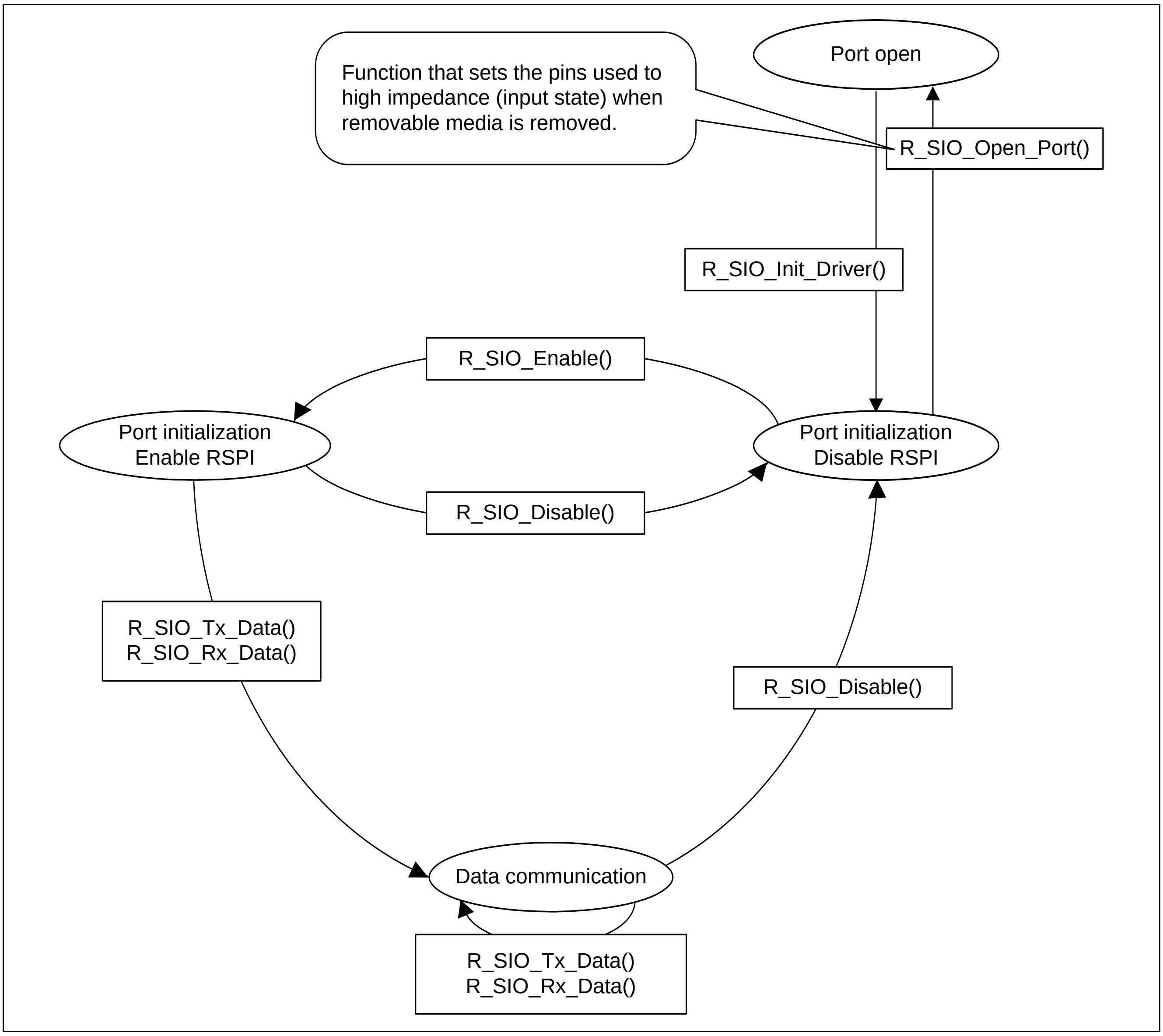


Figure 6.4 State Transition Diagram

6.9 Function Specifications

6.9.1 Driver Initialization

R_SIO_Init_Driver

Synopsis	Driver initialization
Header	R_SIO.h, R_SIO_rspi.h, mtl_com.h
Declaration	error_t R_SIO_Init_Driver(void)
Explanation	• Initializes the driver. Disables the serial I/O function and sets the pins to their port function. • This function must be called exactly once when the system starts. • Set the slave device select control signal to the high level before calling this function.
Arguments	None
Return values	SIO_OK ; Successful operation
Remarks	The following processing, which takes into account the previous state, is performed. • The function R_SIO_Disable() is called.

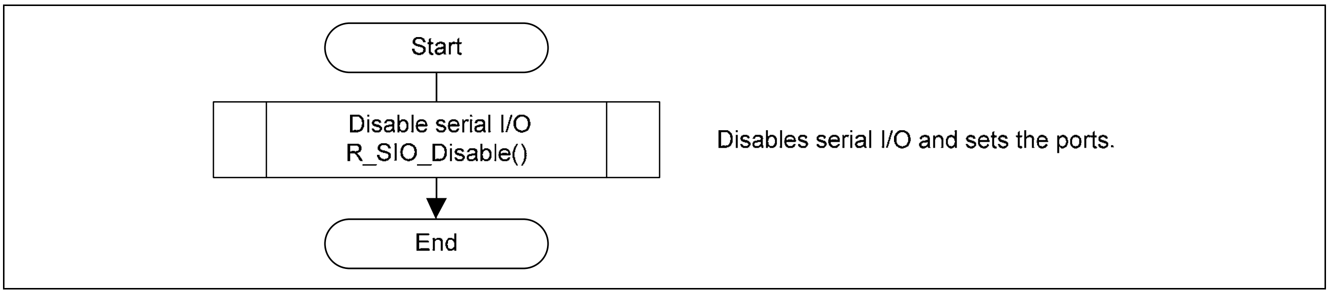


Figure 6.5 Driver Initialization Processing Outline

6.9.2 Serial I/O Disable Setup Processing

R_SIO_Disable

Synopsis Disable serial I/O processing

Header R_SIO.h, R_SIO_rspi.h, mtl_com.h

Declaration error_t R_SIO_Disable(void)

Explanation

- Disables the serial I/O function and sets the pins to their port function.
  Disables serial I/O.
  Sets the pins used for serial I/O to their port function.
- Set the slave device select control signal to the high level before calling this function.

Arguments None

Return values SIO_OK ; Successful operation

Remarks

- The RSPI module stop state is canceled temporarily to write to the RSPI related registers. After setting the RSPI related registers, the module is set back to the module stop state.
- If not used, this function can be called to disable the serial I/O function.

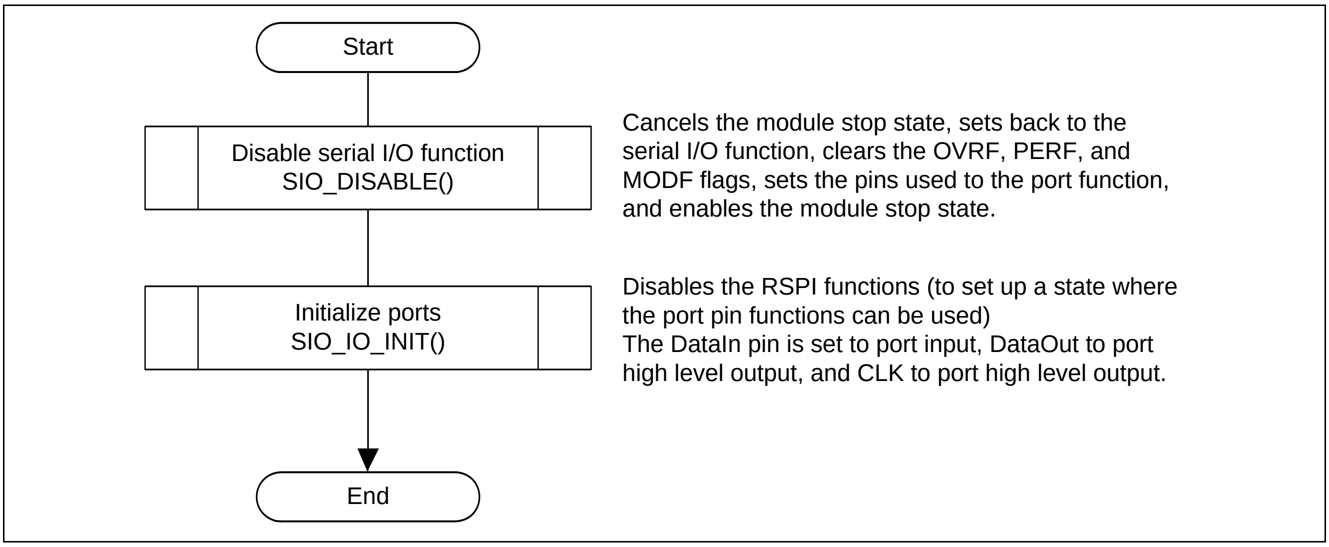


Figure 6.6 Serial I/O Disable Setup Processing Outline

6.9.3 Serial I/O Enable Setup Processing

R_SIO_Enable

Synopsis Enable serial I/O processing

Header R_SIO.h, R_SIO_rspi.h, mtl_com.h

Declaration error_t R_SIO_Enable(uint8_t BrgData)

Explanation
- Enables the serial I/O function and sets the bit rate.
  Sets the pins used by serial I/O to their port function.
  Enables serial I/O and sets the bit rate.
- Call this function only after calling R_SIO_Disable().
- This function must be called before performing either serial I/O data transmission or serial I/O data reception.
- Use this function to change the bit rate. But before doing that, first call the disable serial I/O function.

Arguments uint8_t BrgData ; Bit rate setting

Return values SIO_OK ; Successful operation

Remarks
- This function sets the serial I/O module used to the module stop cleared state.
- The software wait (10 µs) is the wait time required to set the bit rate.

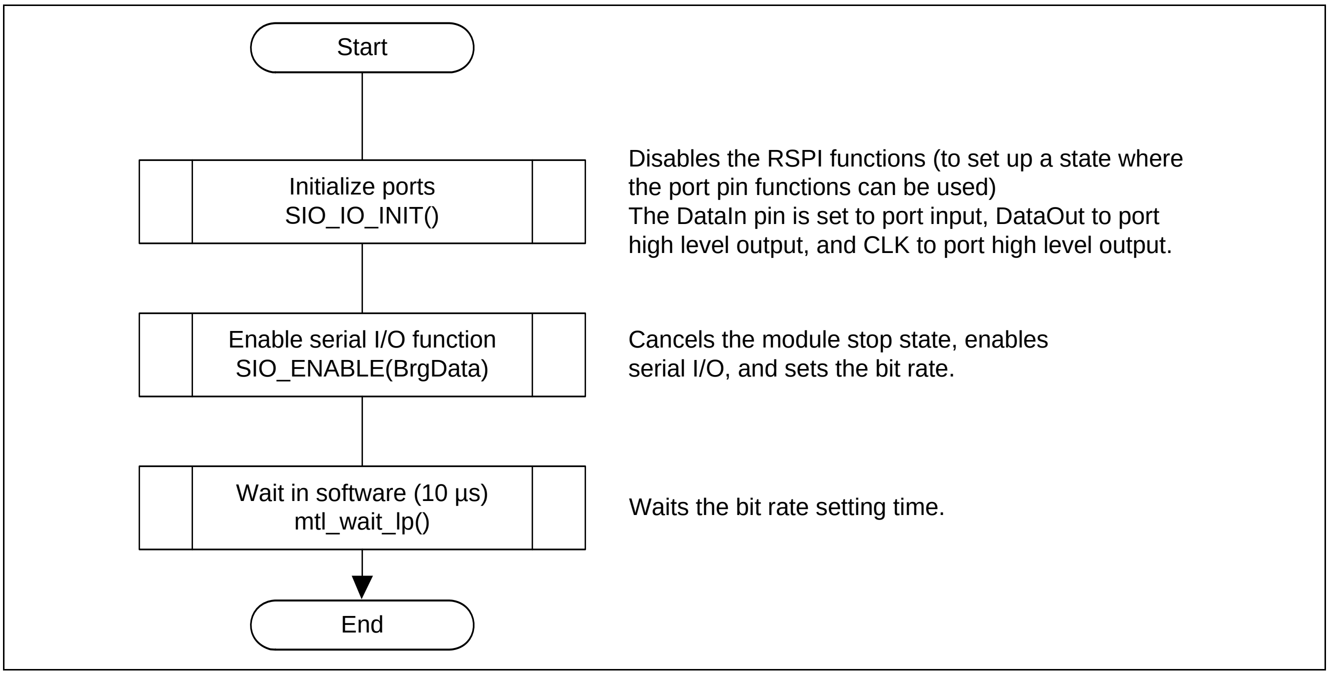


Figure 6.7 Serial I/O Enable Setup Processing Outline

6.9.4 Serial I/O Open Setup Processing

R_SIO_Open_Port

Synopsis	Serial I/O port (DataOut, DataIn, and CLK) open processing
Header	R_SIO.h, R_SIO_rspi.h, mtl_com.h
Declaration	error_t R_SIO_Open_Port(void)
Explanation	• Sets the pins used for serial I/O to open (the input state). • Set the slave device select control signal to the high level before calling this function.
Arguments	None
Return values	SIO_OK ; Successful operation
Remarks	This function is provided for inserting and removing removable media. Use this function before inserting or removing removable media. Perform the serial I/O disable setup processing before removing removable media.

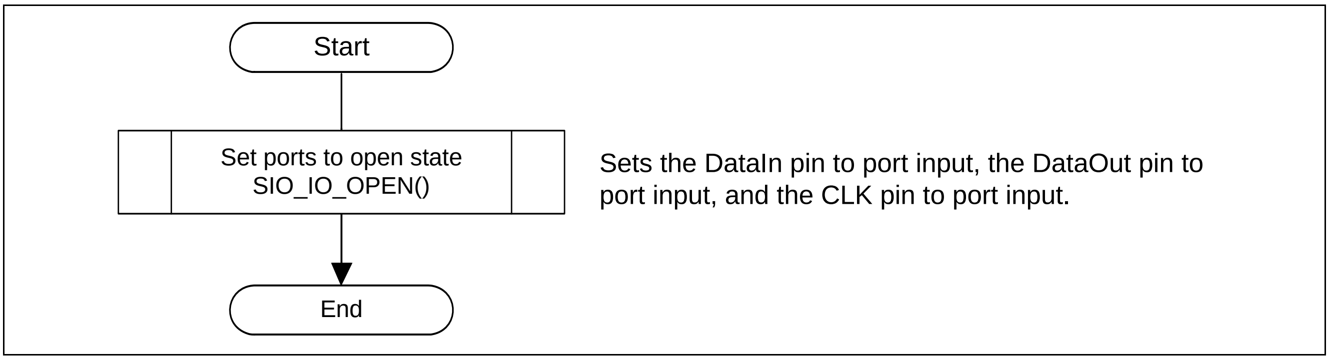


Figure 6.8 Serial I/O Open Setup Processing Outline

6.9.5 Serial I/O Data Transmission Processing

R_SIO_Tx_Data

Synopsis	Transmit serial I/O data
Header	R_SIO.h, R_SIO_rspi.h, mtl_com.h
Declaration	error_t R_SIO_Tx_Data(uint16_t TxCnt, uint8_t FAR* pData)
Explanation	• Transmits the specified number of bytes of data from pData. • The serial I/O enable setup processing must be performed prior to calling this function. • The serial I/O disable setup processing must be performed if the result of this function indicates that an error occurred.
Arguments	uint16_t TxCnt ; Number of bytes to transmit uint8_t FAR* pData ; Pointer to transmit data buffer
Return values	SIO_OK ; Successful operation SIO_ERR_HARD ; Hardware error
Remarks	• Use this function for half-duplex transmission. • The following operations, which follow the initialization flowchart shown in the hardware manual, are performed. (the inline function SIO_TX_ENABLE()) (1) Sets SPCR2 (enables RSPI idle interrupt requests) (2) Sets SPCMD (3) Clears the IR flag (4) Sets the port function register PFxSPI (enables the RSPI pins) (5) Sets SPCR (enables transmission) (6) Reads SPCR • After transmission completes, serial communication is disabled by the reverse of the enable processing shown above. (The inline function SIO_TX_DISABLE()) (1) Sets SPCR (stops transmission and reception) (2) Reads SPCR (3) Sets the port function register PFxSPI (disables the RSPI pins) (4) Sets SPCR2 (disables RSPI idle interrupt requests) • Both the transmit buffer empty IR and the RSPI idle IR are used to verify the completion of data transmission. • We recommend performing the serial I/O disable setup processing if serial I/O is not to be used sequentially.

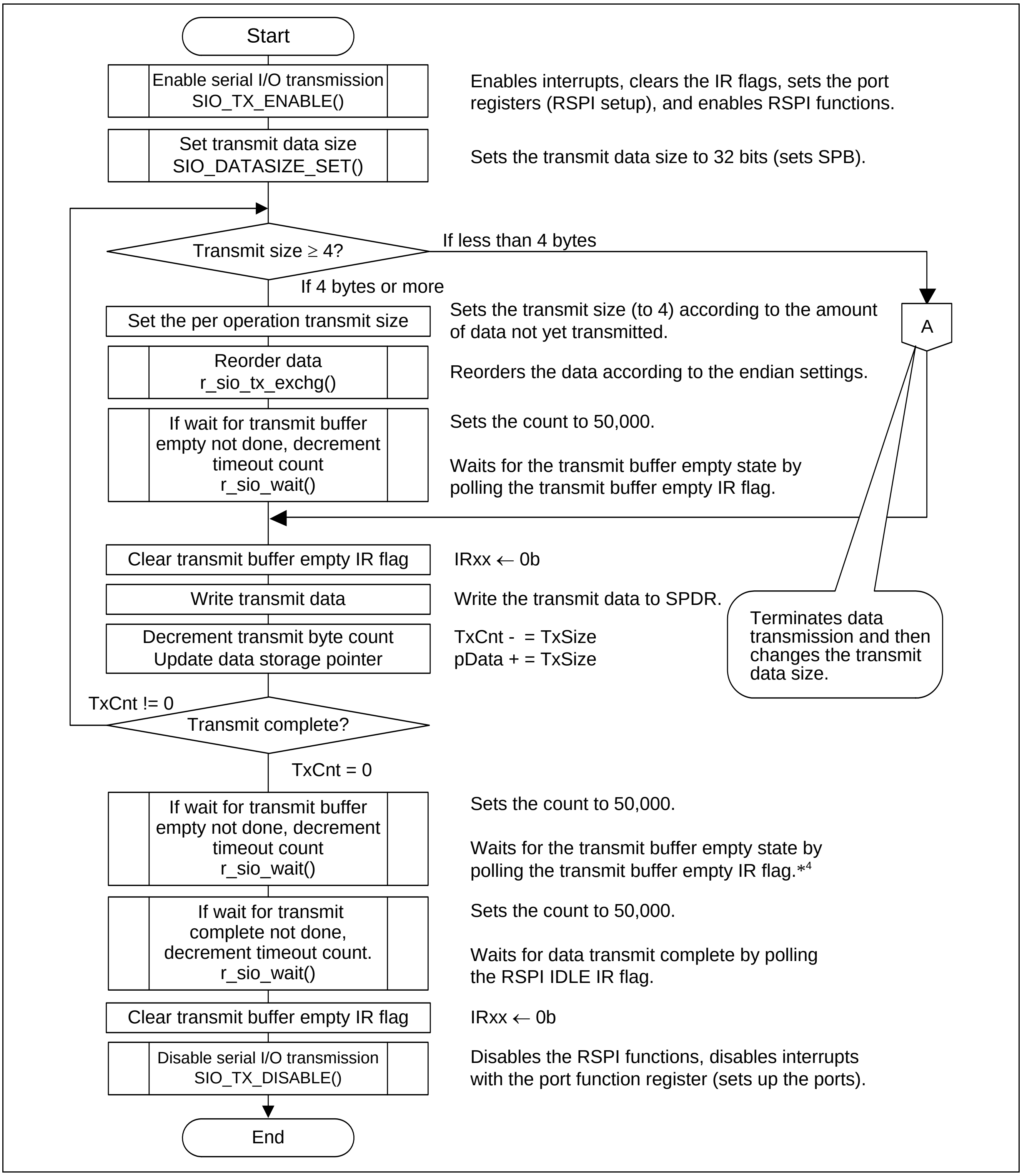


Figure 6.9 Serial I/O Data Transmission Processing Outline — 1

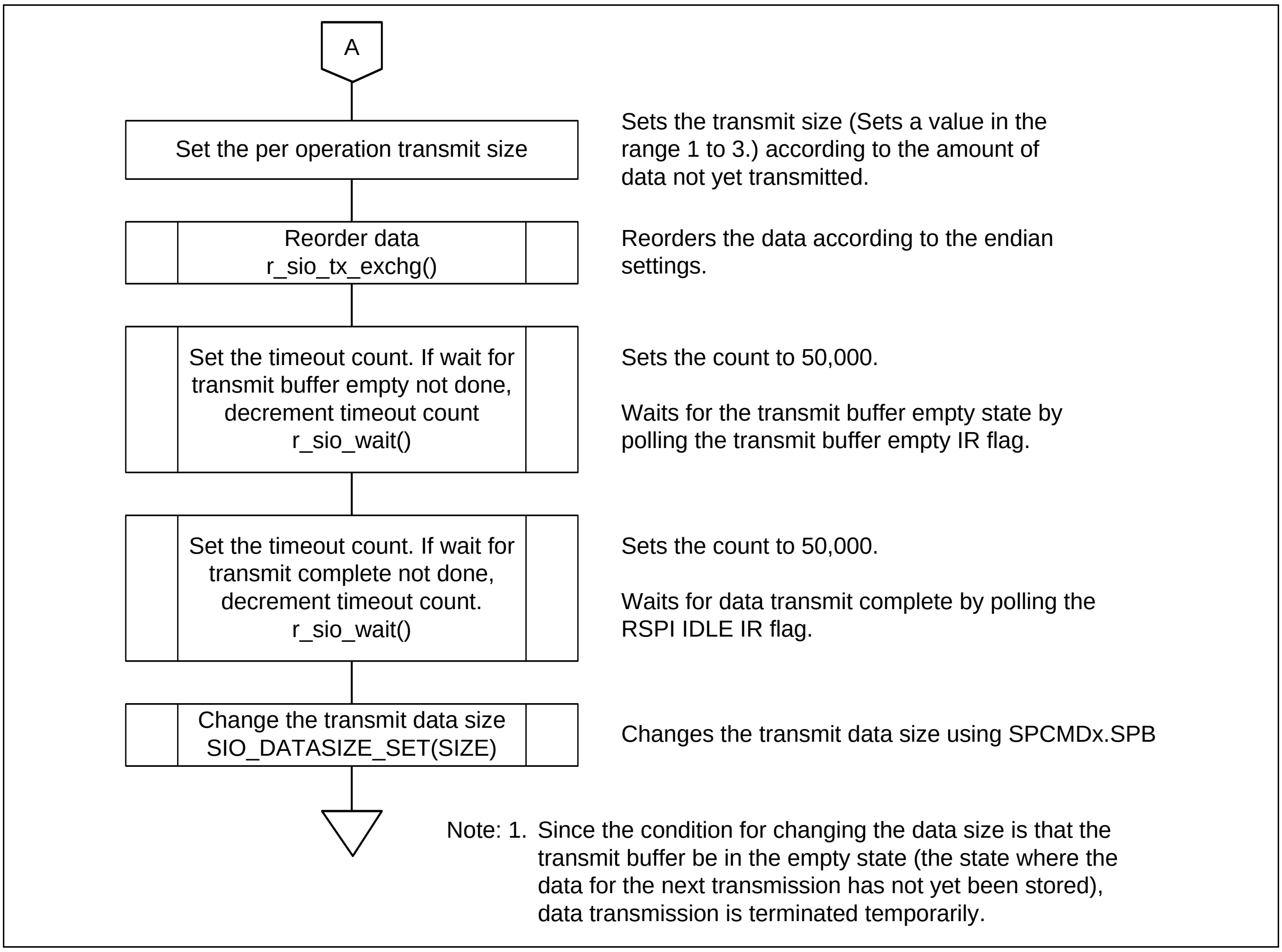


Figure 6.10 Serial I/O Data Transmission Processing Outline — 2

6.9.6 Serial I/O Data Reception Processing

R_SIO_Rx_Data

Synopsis Receive serial I/O data

Header R_SIO.h, R_SIO_rspi.h, mtl_com.h

Declaration error_t R_SIO_Rx_Data(uint16_t RxCnt, uint8_t FAR* pData)

Explanation
- Receives the specified number of bytes of data stores it in pData.
- The serial I/O enable setup processing must be performed prior to calling this function.
- The serial I/O disable setup processing must be performed if the result of this function indicates that an error occurred.
- Either normal reception or high-speed reception may be selected. For the selection method, see section 7.2.2, R_SIO_rspi.h (1) for the definitions of the operation modes used.
- An overview of normal reception is shown in figure 6.11, Serial I/O Data Reception Processing Outline — 1 (Normal Reception) and figure 6.12, Serial I/O Data Reception Processing Outline — 2 (Normal Reception).
- An overview of normal reception is shown in figure 6.13, Serial I/O Data Reception Processing Outline — 3 (High-Speed Reception) and figure 6.14, Serial I/O Data Reception Processing Outline — 4 (High-Speed Reception).

Arguments

uint16_t	RxCnt	; Reception byte count
uint8_t FAR*	pData	; Pointer to receive data storage buffer

Return values

SIO_OK	; Successful operation
SIO_ERR_HARD	; Hardware error

Remarks
- Use this function for half-duplex reception.
- The following operations, which follow the initialization flowchart shown in the hardware manual, are performed. (the inline function SIO_TRX_ENABLE())
  (1) Sets SPCMD
  (2) Clears the IR flag
  (3) Sets the port function register PFxSPI (enables the RSPI pins)
  (4) Sets SPCR (enables transmission and reception)
  (5) Reads SPCR
- After reception completes, serial communication is disabled by the reverse of the enable processing shown above. (The inline function SIO_TRX_DISABLE())
  (1) Sets SPCR (stops transmission and reception)
  (2) Reads SPCR
  (3) Sets the port function register PFxSPI (disables the RSPI pins)
- We recommend performing the disable serial I/O processing if serial I/O is not to be used sequentially.
- The following processing is added for high-speed reception.
  (1) To prevent overrun errors*[1] from occurring during continuous reception, interrupts are disabled from the immediately before the next dummy write to the point where the previous receive data has been acquired. The interrupts disabled state is implemented by setting the processor interrupt priority level (IPL[3:0]) to the highest level.
  (2) The dummy data writes for the third and following continuous reception operations are performed after data reception has completed. This allows other interrupts to be accepted during continuous reception operations.

Note: 1. Overrun errors may occur if there is contention for a shared bus between a DMAC, EXDMAC, or DTC transfer performed by another programs and this reception operation, or if a high-priority NMI interrupt occurs.

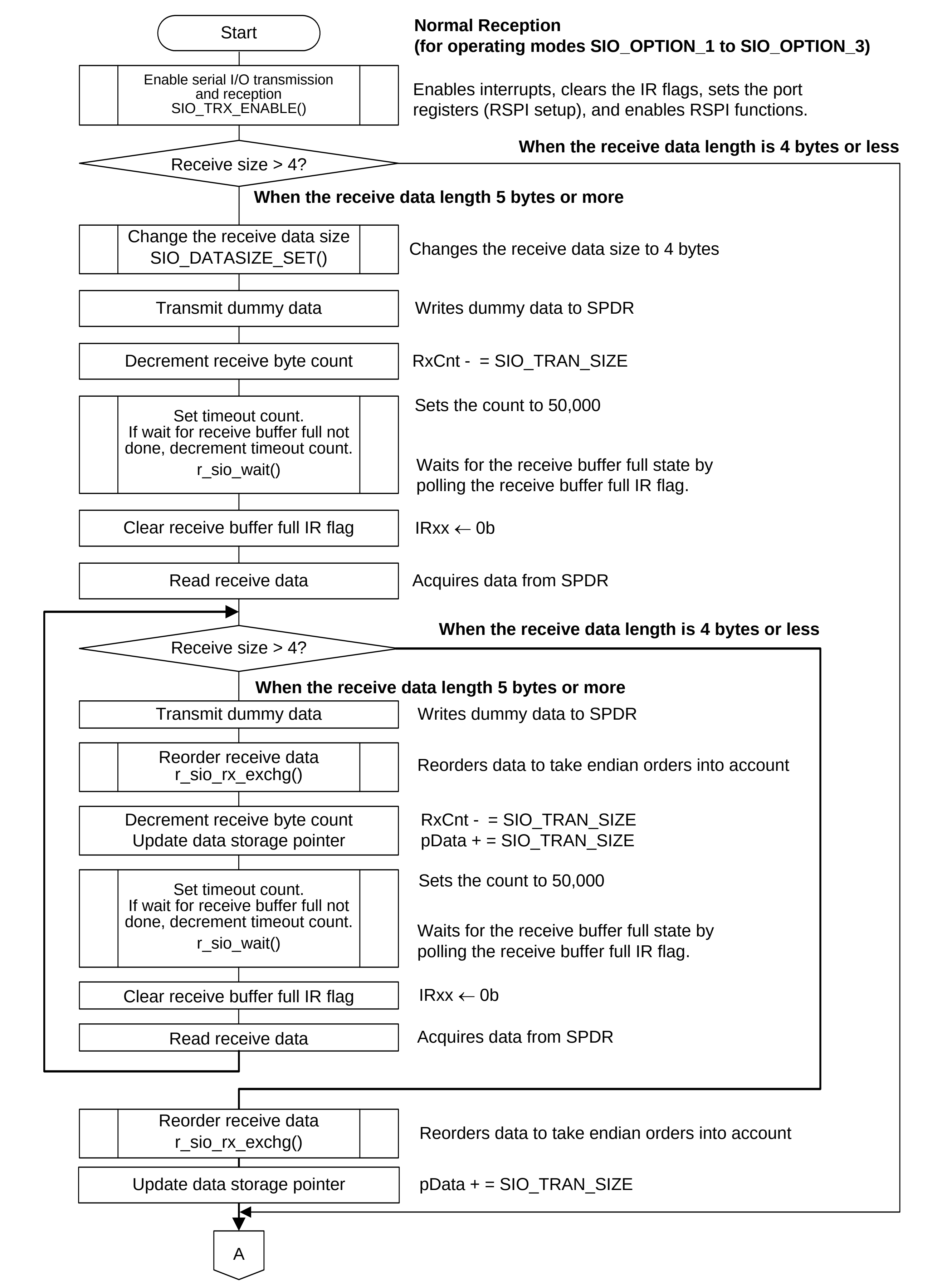


Figure 6.11 Serial I/O Data Reception Processing Outline — 1 (Normal Reception)

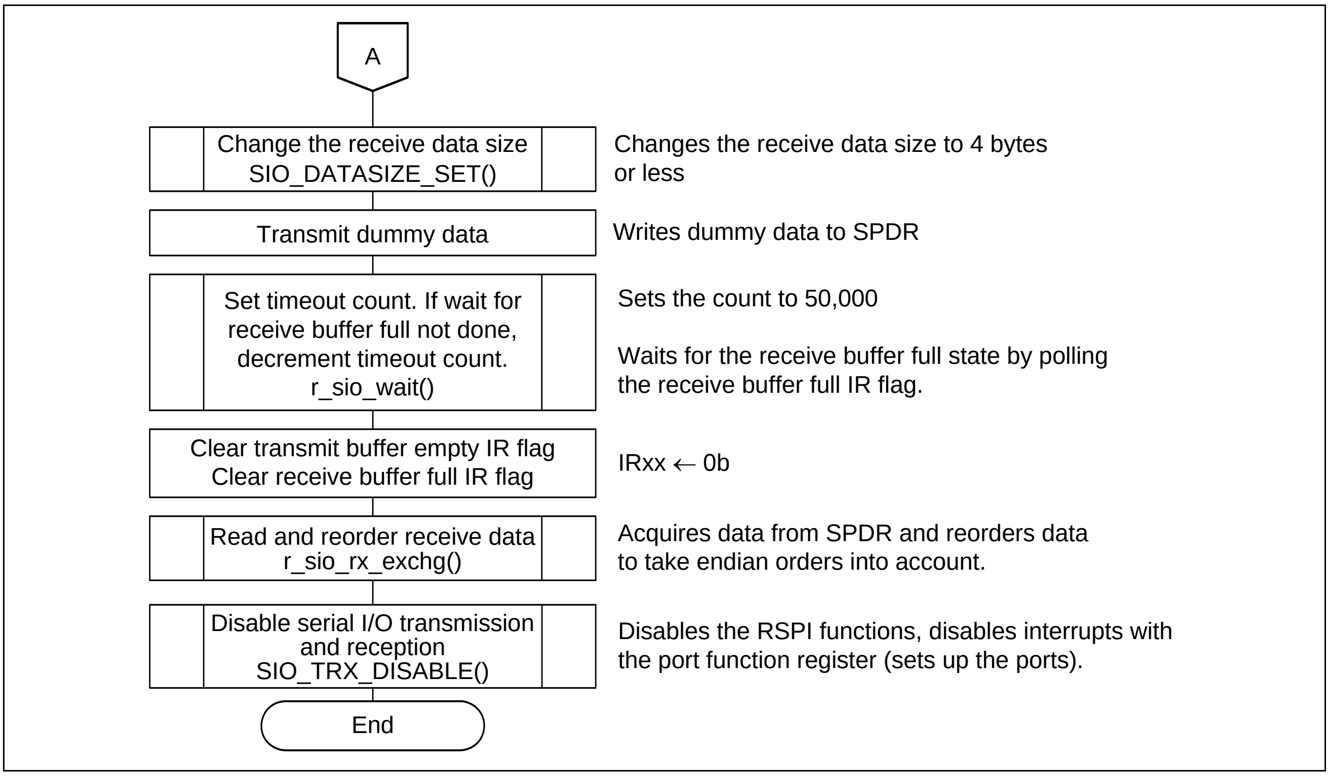


Figure 6.12 Serial I/O Data Reception Processing Outline — 2 (Normal Reception)

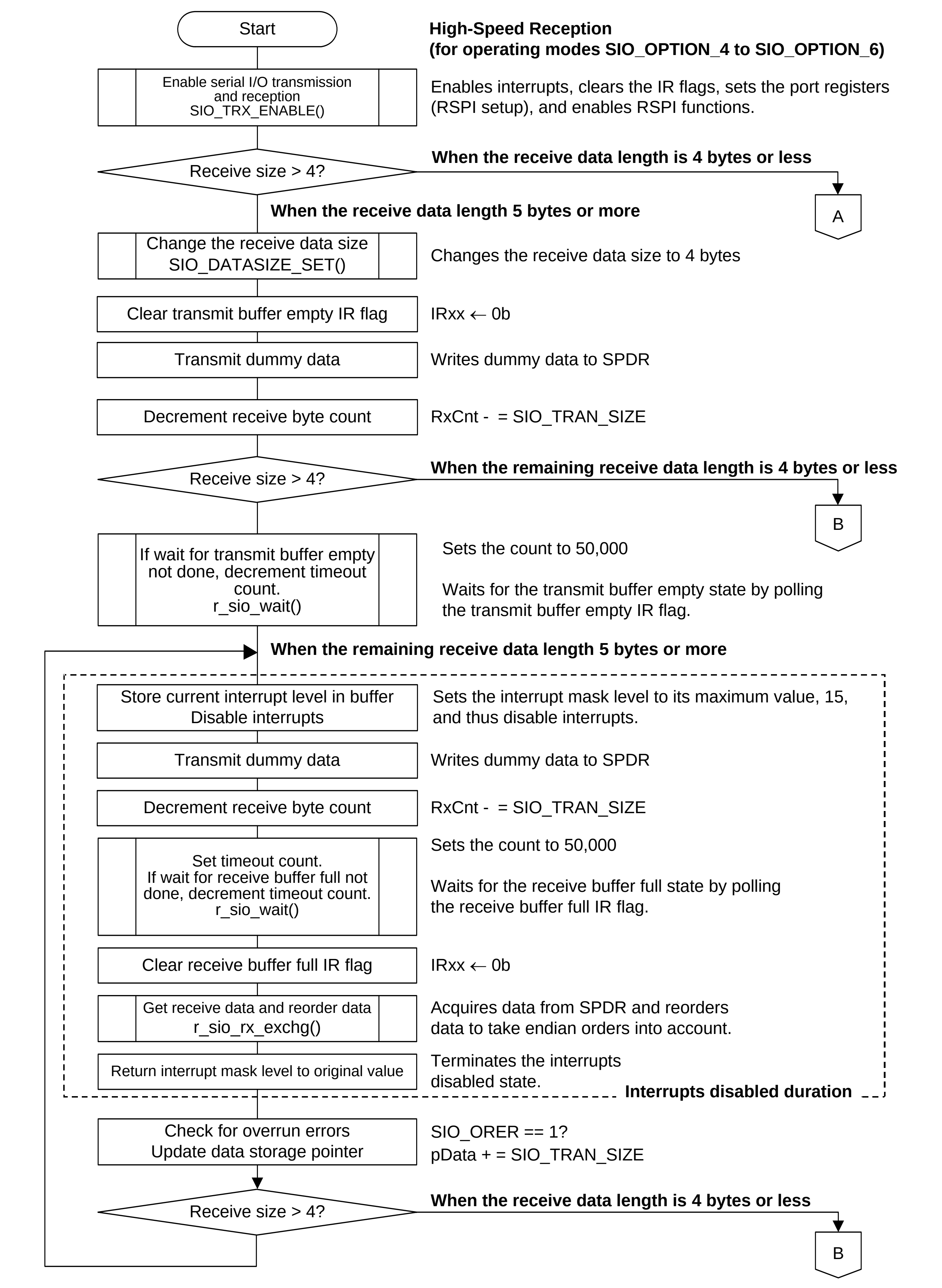


Figure 6.13 Serial I/O Data Reception Processing Outline — 3 (High-Speed Reception)

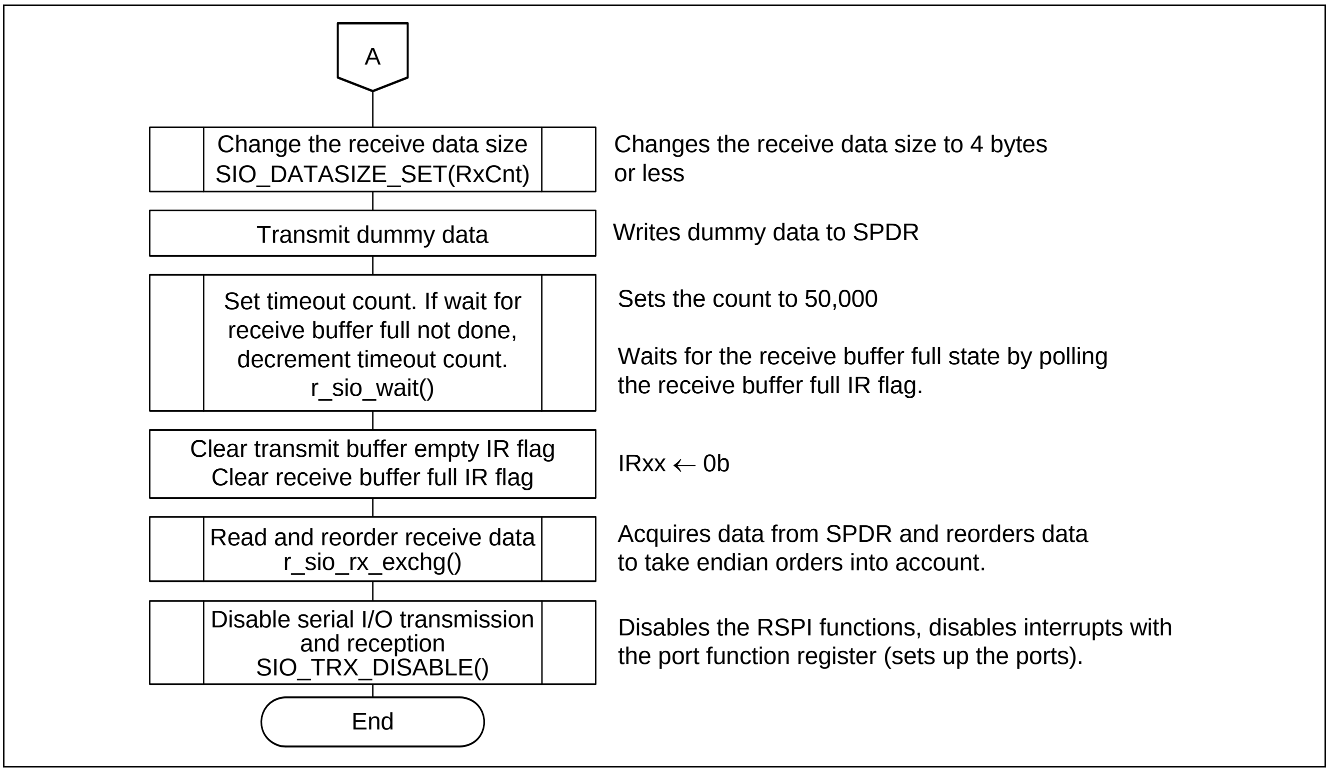


Figure 6.14 Serial I/O Data Reception Processing Outline — 4 (High-Speed Reception)

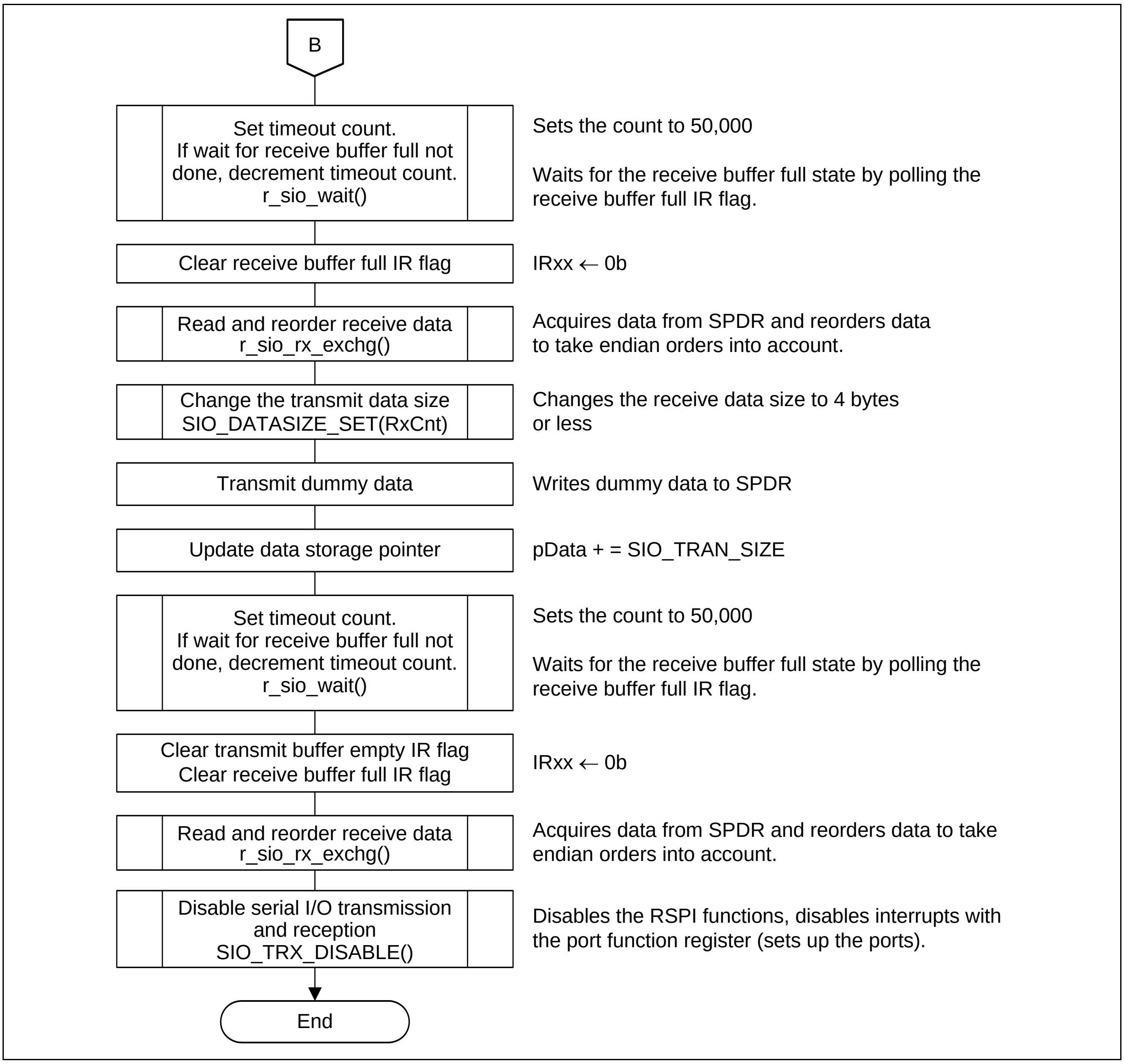


Figure 6.15 Serial I/O Data Reception Processing Outline — 5 (High-Speed Reception)

6.10 Inline Function Specifications

This section describes the inline functions used in this sample code.

6.10.1 SIO_IO_INIT()

(1) Purpose

This function disables the RSPI functions for the corresponding pins, sets input pins to the port input state, and sets output pins to the port output state.

(2) Function

This function disables the RSPI functions for the corresponding pins, sets the DataIn pin to the port input state, and sets the DataOut and CLK pins to the port output state.

The following processing is implemented. If necessary, revise this processing.

1. Disables the RSPI functions for each pin using the port function register (PFxSPI) that controls the RSPI module.
   — PFxSPI. ← 00h: Disables the RSPI functions (port enabled setting)
2. Sets the DataIn pin to port input.
   See the SIO_DATAI_INIT() function.
3. Sets the DataOut pin to port high output.
   See the SIO_DATAO_INIT() function.
4. Sets the DataOut pin to port high output.
   See the SIO_CLK_INIT() function.

(3) Remarks

This inline function changes the pins from their peripheral function to their port function. Applications should first verify that other peripheral functions are not being used before executing this function.

6.10.2 SIO_IO_OPEN()

(1) Purpose

Sets the input pins and output pins to the port input state.

(2) Function

Sets the DataIn pin, the DataOut pin, and the CLK pin to the port input state.

The following processing is implemented. If necessary, revise this processing.

1. Sets the DataIn pin to port input.
   See the SIO_DATAI_INIT() function.
2. Sets the DataOut pin to port input.
   See the SIO_DATAO_OPEN() function.
3. Sets the CLK pin to port input.
   See the SIO_CLK_OPEN() function.

(3) Remarks

Use this function to set all pins to high impedance before removable media is inserted or removed. Execute this function after executing SIO_IO_INIT().

6.10.3 SIO_DATAI_INIT()

(1) Purpose

Sets the DataIn pin to the port input state.

(2) Function

The following processing is implemented. If necessary, revise this processing.

1. Enables the DataIn pin input buffer function using the input buffer control register (ICR).
   — DataIn pin ICR ← 1b: Input buffer enabled
2. Disables the DataIn pin input pull-up resistor with the pull-up resistor control register (PCR).*[1]
   — DataIn pin PCR ← 0b: Input pull-up resistor disabled*[2]
3. Sets the DataIn pin to port input using the data direction register (DDR).
   — DataIn pin DDR ← 0b: Input port

(3) Remarks

Notes: 1. The pins that can be set by the pull-up resistor control register (PCR) are port 9, ports A to E, and port G. If port A, port C, or port E is used, use SIO_PCR_DATAI for this setting.
2. Perform this setting as required.

6.10.4 SIO_DATAO_INIT()

(1) Purpose

Sets the DataOut pin to port high output.

(2) Function

The following processing is implemented. If necessary, revise this processing.

1. Sets the DataOut pin output type to CMOS output using the open drain control register (ODR).*[1]
   — DataOut pin ODR ← 0b: CMOS output*[3]
2. Disables the DataOut pin input buffer function using the input buffer control register (ICR).*[2]
   — DataOut pin ICR ← 0b: Input buffer disabled
3. Sets the DataOut pin to high output using the data register (DR).
   — DataOut pin DR ← 1b: High output
4. Sets the DataOut pin to port output using the data direction register (DDR) and the data register (DR).
   — DataOut pin DDR ← 1b: Output port
   — DataOut pin DR ← 1b: High output

(3) Remarks

Notes: 1. The pins that can be set by the open drain control register (ODR) are ports 0 to 3 and port C. If port 2 or port C is used, use SIO_ODR_DATAO and SIO_ODR_CLK for this setting.
2. When a pin is used as an output pin, if the input buffer function is enabled with the input buffer control register (ICR), it will be possible to acquire the output data as the pin state. Therefore, it is necessary to disable the input buffer function with the ICR setting for pins used as output pins.
3. Perform this setting as required.

6.10.5 SIO_DATAO_OPEN()

(1) Purpose

Sets the DataOut pin to the port input function.

(2) Function

The following processing is implemented. If necessary, revise this processing.

1. Sets the DataOut pin to port input using the data direction register (DDR).
   — DataOut pin DDR ← 0b: Input port (input buffer disabled)

(3) Remarks

None

6.10.6 SIO_CLK_INIT()

(1) Purpose

Sets the CLK pin to port high output.

(2) Function

The following processing is implemented. If necessary, revise this processing.

1. Sets the CLK pin output type to CMOS output using the open drain control register (ODR).*[1]
   — CLK pin ODR ← 0b: CMOS output*[3]
2. Disables the CLK pin input buffer function using the input buffer control register (ICR).*[2]
   — CLK pin ICR ← 0b: Input buffer disabled
3. Sets the CLK pin to high output using the data register (DR).
   — CLK pin DR ← 1b: High output
4. Sets the CLK pin to port output using the data direction register (DDR) and the data register (DR).
   — CLK pin DDR ← 1b: Output port
   — CLK pin DR ← 1b: High output

(3) Remarks

Notes: 1. The pins that can be set by the open drain control register (ODR) are ports 0 to 3 and port C. If port 2 or port C is used, use SIO_ODR_DATAO and SIO_ODR_CLK for this setting.
2. When a pin is used as an output pin, if the input buffer function is enabled with the input buffer control register (ICR), it will be possible to acquire the output data as the pin state. Therefore, it is necessary to disable the input buffer function with the ICR setting for pins used as output pins.
3. Perform this setting as required.

6.10.7 SIO_CLK_OPEN()

(1) Purpose

Sets the CLK pin to the port input function.

(2) Function

The following processing is implemented. If necessary, revise this processing.

1. Sets the CLK pin to port input using the data direction register (DDR).
   — CLK pin DDR ← 0b: Input port (input buffer disabled)

(3) Remarks

None

6.10.8 SIO_ENABLE()

(1) Purpose

Initializes serial I/O and enables its functions. Note that this function performs the common processing through enabling transmission, reception, or transmission and reception. It also sets the bit rate.

(2) Function

Initializes serial I/O as stipulated in the hardware manual. If necessary, revise this processing.
This function performs the following processing when an RX62N microcontroller is used.

1. Sets the module to the module stop canceled state using the module stop control register (MSTPCRB).
   — MSTPCRB MSTPBxx ← 0b: Cancels module stop state and enables reading and writing of the RSPI registers.
   — Reads MSTPCRB MSTPBxx
2. Performs the common processing for enabling transmission and transmission/reception.
   The common processing for enabling transmission and transmission/reception consists of the following operations.
   — SPPCR ← 30h: Sets up normal mode, CMOS output, and a MOSI idle fixed value of 1.
   — Sets the SPBR bit rate.
   — SPDCR ← 20h: Setting 1.1, SSL0 to SSL3 output*[1], read receive buffer, longword access.
   — SPCKD ← 00h: SPCKD delay value setting (initial value)*[2]
   — SSLND ← 00h: SSL negation delay value setting (initial value)*[2]
   — SPND ← 00h: Next access delay value setting (initial value)*[2]
   — SPCR2 ← 00h: Parity function disabled, idle interrupt disabled.
   — Clears the SPSR OVFR, MODF, and PERF flags.
   See the SIO_SPSR_CLEAR() function.
   — SPCR ← 09h: three-wire method, master mode, transmit interrupts disabled, RSPI functions disabled, receive interrupts disabled.
   — SPSCR ← 00h: Sequence length: only SPCMD0 is used.

(3) Remarks

The user should insert wait processing after this inline function completes for serial I/O that requires a wait after setting the bit rate.

This function forms a pair with SIO_DISABLE(). If this function is run, call SIO_DISABLE() to terminate processing.

Call one of SIO_DISABLE(), SIO_TX_DISABLE(), or SIO_TRX_DISABLE() (to disable communication operation using SPCR) to stop communication operation before calling this function.

Notes: 1. Since three-wire method is used, the SSL functions are not used. Since the SSL pins can be allocated to other functions, sets the SSL pins other than SSL0 to I/O and allocates them to the other functions.
2. Not used in this sample code.

6.10.9 SIO_DISABLE()

(1) Purpose

Disables the serial I/O functions.

(2) Function

Disables the serial I/O functions. This function performs the common processing in the procedures for disabling transmission or reception. If necessary, revise this processing.

This function performs the following processing when an RX62N microcontroller is used.

1. Sets the module to the module stop canceled state using the module stop control register (MSTPCRB) so that the RSPI related registers can be set.*[1]
   — MSTPCRB MSTPBxx ← 0b: Cancels module stop state and enables reading and writing of the RSPI registers.
   — Reads MSTPCRB MSTPBxx
2. Disables the RSPI functions.
   — SPCR ← 09h: three-wire method, master mode, transmit interrupts disabled, RSPI functions disabled, receive interrupts disabled.
3. Clears the SPSR OVFR, MODF, and PERF flags.
   See the SIO_SPSR_CLEAR() function.
4. Sets the module to the module stop state using the module stop control register (MSTPCRB).
   — MSTPCRB MSTPBxx ← 1b: Enables module stop state and disables reading or writing the RSPI registers. (The RSPI register states are retained.)
   — Reads MSTPCRB MSTPBxx.

(3) Remarks

This function forms a pair with SIO_ENABLE(). If SIO_ENABLE() is run, call this function to terminate processing.

Note: 1. With the RX62N, registers for a module in the module stop state cannot be read or written. In this inline function, the module stop state is canceled temporarily to use SPCR to disable the RSPI functions. After setting SPCR, this function enables module stop state. Note that register values are retained while a module is in the module stop state.

6.10.10 SIO_DATASIZE_SET()

(1) Purpose

Sets SPB[3:0] in the SPCMD0 to SPCMD7 registers.

(2) Function

Sets the data length (8, 16, 24, or 32 bits).

(3) Remarks

None

6.10.11 SIO_TX_ENABLE()

(1) Purpose

Enables serial I/O transmission.

(2) Function

Enables serial I/O according to the specifications in the hardware manual. After switching the pins from their port functions to their serial I/O functions, it enables serial I/O. If necessary, revise this processing.

This function performs the processing from the initialization procedure following SIO_ENABLE() to the dedicated initialization processing for transmission.

The following processing is performed when an RX62N microcontroller is used.

1. SPCR2 settings (Sets the RSPI idle interrupt request to enabled.)
   — SPCR2 ← 04h: Parity function disabled, idle interrupt enabled (for detection of end of transmission)
2. SPCMD settings
   — SPCMD0 ← 0203h: CPHA = 1, CPOL = 1, base bit rate, SSL0*[1], SSL signal negated on end of transfer*[2], 32 bits, MSB first*[3].
3. Clears the IR flags.
   See the SIO_IR_CLEAR() function.
4. Sets the used pins to their RSPI function.
   The registers are set in the following order according to the setting method*[4] for the port function registers show in the hardware manual.
   1) Sets the pins used by the RSPI module with the PFxSPI (x = G or H) RSPI pin selection bits (RSPIS).
      - PFxSPI.RSPIS ← 0b or 1b: Selects the pins used by the RSPI module.
   2) Enables the PFxSPI (x = G or H) RSPI output pins.
      - PFxSPI | = 0Eh: RSPCK pin enabled, MOSI pin enabled, MISO pin enabled.
5. SPCR settings (Enables transmission)
   Enables transmission by setting TXMD, SPTIE, and SPE in the SPCR register.
   — SPCR ← 6Bh: three-wire method, transmission operation only, master mode, transmit interrupts enabled, RSPI functions enabled.
6. Reads SPCR.

(3) Remarks

This function forms a pair with SIO_TX_DISABLE(). If this function is run, call SIO_TX_DISABLE() to terminate processing.

Notes: 1. Since three-wire method is used, the SSL functions are not used. Since the SSL pins can be allocated to other functions, sets the SSL pins other than SSL0 to I/O and allocates them to the other functions.
2. Since the SSL functions are not used, this setting is ignored.
3. Since SPCKD, SSLND, and SPND are not used, bits b15 to b13 in the SPCMD0 register should be set to 0b.
4. In the RSPI port function registers (PFxSPI), there are both pin selection bits that change the I/O destination and enable bits that enable the pin functions. Therefore the I/O destinations are set with the pin selection bits and then the pin functions are enabled with the enable bits.

6.10.12 SIO_TX_DISABLE()

(1) Purpose

Stops the serial I/O data transmission function.

(2) Function

This function stops the transmission function with the reverse procedure from that used by SIO_TX_ENABLE(). After performing the settings to stop transmission, it switches the pins from their serial I/O functions to their port functions. If necessary, revise this processing.

This function performs the following processing when an RX62N microcontroller is used.

1. SPCR settings (Stops transmission and reception)
   Clears the TXMD, SPTIE, SPE, and SPRIE bits in the SPCR register to stop transmission and reception.
   — SPCR ← 09h: Master mode, transmit interrupts disabled, RSPI functions disabled, receive interrupts disabled.
2. Reads SPCR.
3. Sets the used pins to their port functions.
   Disables the RSPI output enabled pins with the port function register (PFxSPI).
   — PFxSPI ← 00h: RSPCK pin disabled, MOSI pin disabled, MISO pin disabled.
4. SPCR2 settings (Sets the RSPI idle interrupt request to disabled.)
   — SPCR2 ← 00h: Parity function disabled, idle interrupt disabled.

(3) Remarks

This function forms a pair with SIO_TX_ENABLE(). After SIO_TX_ENABLE() is run, call this function to terminate processing.

6.10.13 SIO_TRX_ENABLE()

(1) Purpose

Enables serial I/O transmission and reception.

(2) Function

Enables serial I/O according to the specifications in the hardware manual. After switching the pins from their port functions to their serial I/O functions, it enables serial I/O transmission and reception. If necessary, revise this processing.

This function performs the processing from the initialization procedure following SIO_ENABLE() to the dedicated initialization processing for transmission.

The following processing is performed when an RX62N microcontroller is used.

1. SPCMD settings
   — SPCMD0 ← 0203h: CPHA = 1, CPOL = 1, base bit rate, SSL0*[1], SSL signal negated on end of transfer*[2], 32 bits, MSB first*[3].
2. Clears the IR flags.
   See the SIO_IR_CLEAR() function.
3. Sets the used pins to their RSPI function.
   The registers are set in the following order according to the setting method*[4] for the port function registers (PFxSPI) show in the hardware manual.
   1) Sets the pins used by the RSPI module with the PFxSPI (x = G or H) RSPI pin selection bits (RSPIS).
      - PFxSPI.RSPIS ← 0b or 1b: Selects the pins used by the RSPI module.
   2) Enables the PFxSPI (x = G or H) RSPI output pins.
      - PFxSPI | = 0Eh: RSPCK pin enabled, MOSI pin enabled, MISO pin enabled.
4. SPCR settings (Enables transmission and reception)
   Enables transmission and reception by setting SPTIE, SPE and SPRIE in the SPCR register.
   — SPCR ← E9h: three-wire method, full-duplex operation, master mode, transmit and receive interrupts enabled, RSPI functions enabled.
5. Reads SPCR.

(3) Remarks

This function forms a pair with SIO_TRX_DISABLE(). If this function is run, call SIO_TRX_DISABLE() to terminate processing.

Notes: 1. Since three-wire method is used, the SSL functions are not used. Since the SSL pins can be allocated to other functions, sets the SSL pins other than SSL0 to I/O and allocates them to the other functions.
2. Since the SSL functions are not used, this setting is ignored.
3. Since SPCKD, SSLND, and SPND are not used, bits b15 to b13 in the SPCMD0 register should be set to 0b.
4. In the RSPI port function registers (PFxSPI), there are both pin selection bits that change the I/O destination and enable bits that enable the pin functions. Therefore the I/O destinations are set with the pin selection bits and then the pin functions are enabled with the enable bits.

6.10.14 SIO_TRX_DISABLE()

(1) Purpose

Stops the serial I/O data transmission and reception function.

(2) Function

This function stops the transmission and reception function with the reverse procedure from that used by SIO_TRX_ENABLE(). After performing the settings to stop transmission and reception, it switches the pins from their serial I/O functions to their port functions. If necessary, revise this processing.

This function performs the following processing when an RX62N microcontroller is used.

1. SPCR settings (Stops transmission and reception)
   Clears the TXMD, SPTIE, SPE, and SPRIE bits in the SPCR register to stop transmission and reception.
   — SPCR ← 09h: Master mode, transmit interrupts disabled, RSPI functions disabled, receive interrupts disabled.
2. Reads SPCR.
3. Sets the used pins to their port functions.
   Disables the RSPI output enabled pins with the port function register (PFxSPI).
   — PFxSPI ← 00h: RSPCK pin disabled, MOSI pin disabled, MISO pin disabled.

(3) Remarks

This function forms a pair with SIO_TRX_ENABLE(). After SIO_TRX_ENABLE() is run, call this function to terminate processing.

6.10.15 SIO_SPSR_CLEAR()

(1) Purpose

Clears the SPSR error flags.

(2) Function

Clears the OVRF, MODF, and PERF flags.

1. If a flag is 1, it is cleared to 0.
2. The flag is then read to verify that it is 0.

(3) Remarks

None

6.10.16 SIO_IR_CLEAR()

(1) Purpose

Clears the IR flag.

(2) Function

Clears the flag using the following procedure. If necessary, revise this processing.

This function performs the following processing when an RX62N microcontroller is used.

1. Clears the IR flag.

(3) Remarks

None

7. Sample Application

This section presents a sample application that sets up the serial I/O control block.

The sample settings for actual usage are shown below.

The places in each file that need to be set are marked with the comment "/** SET **/".

7.1 mtl_com.h (Common header file)

Common header file for common functions.

Files (except for mtl_com.h.common) with the filename mtl_com.h.XXX have been created for each microcontroller. Rename one of these to mtl_com.h and use that file. If there is no corresponding file for the microcontroller used, refer to these files and create a file appropriate for the microcontroller used.

(1) OS Header File Definitions

This sample code does not use any settings for OS system calls.

The example below is for the case where no OS is used.

Set these up to be unused settings with this sample code. They depend on other software.

```
/* In order to use wai_sem/sig_sem/dly_tsk for microITRON (Real-Time OS)-compatible,    */
/* include the OS header file that contains the prototype declaration.                  */
/* When not using the OS, put the following 'define' and 'include' as comments.         */
//#define MTL_OS_USE                /* Use OS                                           */
//#include <RTOS.h>                 /* OS header file                                   */
//#include "mtl_os.h"
```

(2) Header File Definitions that Define the Common Access areas

A header file in which the MCU function registers are defined is included.

This file is mainly used by device drivers for port control and must be included.

Include the header file that matches the microcontroller used.

In the example below, the header file for the RX62N is included.

This header file must be included when this sample code is used.

```
/* In order to use definitions of MCU SFR area,                                         */
/* include the header file of MCU SFR definition.                                       */
#include "iodefine.h"             /* definition of MCU SFR                              */
```

(3) Loop Timer Definitions

Include the following header file if the software loop timer is used.

This file is mainly used for device drivers to provide wait times.

Comment out the following include statement if the software loop timer is not used.

The example shown below is for the case where the software loop timer is used.

This header file must be included when this sample code is used.

```
/* When not using the loop timer, put the following 'include' as comments.  */
#include "mtl_tim.h"
```

(4) Endian Order Definition

Either little endian or big endian may be specified.

The example below shows how big endian is specified.

```
/* When using M16C or SuperH for Little Endian setting, define it.         */
/* When using other MCUs, put 'define' as a comment.                       */
//#define MTL_MCU_LITTLE                  /* Little Endian                 */
```

(5) Definition for Fast Endian Processing

High-speed processing can be specified for mtl_end.c. If an M16C microcontroller is used, this will speed up processing.

For RX family microcontrollers, comment out this definition so that the symbol is not defined.

```
/* When using M16C, define it.                                             */
/* It performs the fast processes of 'mtl_endi.c'.                         */
//#define MTL_ENDI_HISPEED             /* Uses the high-speed function.    */
```

(6) Standard Library Type Definition

The type of standard library used must be defined.

If the library included with the compiler will be used for the processing shown below, comment out the following definition.

The example shown below is for the case where the library included with the compiler is used.

```
/* Specify the type of user standard library.                              */
/* When using the compiler-bundled library for the following processes,    */
/* put the following 'define' as comments.                                 */
/* memcmp() / memmove() / memcpy() / memset() / strcat() / strcmp() / strcpy() / strlen() */
//#define MTL_USER_LIB                    /* use optimized library         */
```

(7) Definition of the RAM Area to be Accessed

The RAM area used must be defined.

Highly efficient processing can be applied to standard functions and certain other operations.

For the RX62N, MTL_MEM_NEAR should be defined.

```
/* Define the RAM area to be accessed by the user process.                */
/* Efficient operations for standard functions and processes are applied. */
//#define MTL_MEM_FAR /* Defines 'FAR' as 'far' attribute for RAM area. (For M16C Family)  */
#define MTL_MEM_NEAR       /* No far/near attribute for RAM area.         */
```

7.1.2 mtl_tim.h

This file is included if the loop timer is defined in mtl_com.h.

This file depends on the microcontroller used, the clock, the compiler options, and other items.

In systems in which the instruction cache is enabled, the loop timer should be set up assuming that it is running from the instruction cache.

Measure the loop timer performance and set it up according to the operating environment used.

```
/* Define the counter value for the timer.                                       */
/* Specify according to the user MCU, clock and wait requirements.               */
#if 1
/* Setting for 12 MHz no wait Ix8 = 96 MHz(Compile Option "-optimize=2",com.V406R00) */
#define MTL_T_1US                          10    /* loop Number of   1us      */
#define MTL_T_2US                          20    /* loop Number of   2us      */
#define MTL_T_4US                          40    /* loop Number of   4us      */
#define MTL_T_5US                          50    /* loop Number of   5us      */
#define MTL_T_10US                        100    /* loop Number of  10us      */
#define MTL_T_20US                        200    /* loop Number of  20us      */
#define MTL_T_30US                        300    /* loop Number of  30us      */
#define MTL_T_50US                        500    /* loop Number of  50us      */
#define MTL_T_100US                      1000    /* loop Number of 100us      */
#define MTL_T_200US                      2000    /* loop Number of 200us      */
#define MTL_T_300US                      3000    /* loop Number of 300us      */
#define MTL_T_400US     ( MTL_T_200US  *  2 )    /* loop Number of 400us      */
#define MTL_T_1MS                       10000    /* loop Number of   1ms      */
#endif
```

Note that the values above have not been measured and thus appropriate values have not been determined. Through testing should be performed to determine these values.

7.2 Settings for the Clock Synchronous Single Master Control Software

The places in each file that need to be set are marked with the comment "/** SET **/".

7.2.1 R_SIO.h

(1) Definition of the Wait Following the BRR Setting

After the RSPI SPBR register is set, the application waits in software for the period to transfer 1 bit. This wait time must be set.

A time of 10 µs is set as an initial value.

When a MultiMediaCard is used, a value of 10 µs should be set assuming a communications rate of 100 kHz.

```
#define SIO_T_BRR_WAIT        (uint16_t)MTL_T_10US /* BRR setting wait time */
```

7.2.2 R_SIO_rspi.h

This is the definitions file for the RSPI module.

Files with the filename R_SIO_rspi.h.XXX have been created for each microcontroller. Rename one of these to R_SIO_rspi.h and use that file. If there is no corresponding file for the microcontroller used, refer to these files and create a file appropriate for the microcontroller used.

(1) Operating Mode Definitions

The resources for the microcontroller used can be set up. Select the one required definition. In the example below, SIO_OPTION_4 has been selected. Table 7.1 lists operating modes and their functions.

```
/*----------------------------------------------------------------------- */
/*  Define the combination of the MCU's resources.                        */
/*----------------------------------------------------------------------- */
//#define SIO_OPTION_1     /*    */ /* SI/O                               */
//#define SIO_OPTION_2     /*    */ /* SI/O + CRC                         */
//#define SIO_OPTION_3     /*    */ /* SI/O + S/W CRC                     */
#define SIO_OPTION_4       /*    */ /* SI/O           + High Speed Read   */
//#define SIO_OPTION_5     /*    */ /* SI/O + CRC     + High Speed Read   */
//#define SIO_OPTION_6     /*    */ /* SI/O + S/W CRC + High Speed Read   */
```

Table 7.1 Operating Modes

#define Definition	Operating Mode: SI/O (RSPI)	Operating Mode: CRC Calculation (using microcontroller hardware)	Operating Mode: CRC Calculation (using software)	Operating Mode: Receive Mode
SIO_OPTION_1	○	—	—	Normal receive mode
SIO_OPTION_2	○	○	—	Normal receive mode
SIO_OPTION_3	○	—	○	Normal receive mode
SIO_OPTION_4	○	—	—	High-speed receive mode
SIO_OPTION_5	○	○	—	High-speed receive mode
SIO_OPTION_6	○	—	○	High-speed receive mode

When one of SIO_OPTION_1 to SIO_OPTION_3 is selected, normal receive mode is used. In normal receive mode the next data receive operation is performed only after full data reception has been verified and the data output. Therefore, overrun errors do not occur and reliable reception is possible. This mode is designed to avoid software processing during data reception as much as possible. For example, the endian conversion processing for data during continuous reception is performed only after the dummy write of the following data.

When one of SIO_OPTION_4 to SIO_OPTION_6 is selected, high-speed receive mode is used. In high-speed receive mode, writing the dummy data for the next data reception is performed during the current reception, which allows the next data reception operation to be performed immediately after the received data is acquired. Note, however, that a period in which interrupts are disabled occurs during continuous data reception in this mode. Here, if contention for the bus between this reception and a DMAC, EXDMAC, or DTC transfer by another application, or a high-priority NMI interrupt occurs, an overrun error may occur if this application does not acquire the received data in time. Select one of SIO_OPTION_1 to SIO_OPTION_3 to avoid this problem.

If the microcontroller’s internal CRC unit is used to perform MSB-first CRC CCITT calculations, select either SIO_OPTION_2 or SIO_OPTION_5.

If software processing is used to perform MSB-first CRC CCITT calculations, select either SIO_OPTION_3 or SIO_OPTION_6.

If either serial EEPROM or serial flash memory is controlled, comment this setting so that no CRC CCITT calculation is used.

(2) CRC Calculation Type Definition

The CRC calculation type must be specified.

If either serial EEPROM or serial flash memory is controlled, comment these settings so that no CRC CCITT calculation is used.

One of these must be defined if a MultiMediaCard is used.

```
/*------------------------------------------------------------------------------*/
/*  Define the CRC calculation.                                                 */
/*------------------------------------------------------------------------------*/
#define SIO_CRCCCITT_USED                /* CRC-CCITT used                      */
#define SIO_CRC7_USED                    /* CRC7 used                           */
```

(3) Used RSPI Channel Definition

The RSPI channel used must be defined.

```
/*------------------------------------------------------------------------------*/
/*  Define the RSPI channel.                                                    */
/*------------------------------------------------------------------------------*/
#define SIO_RSPI_CHANNEL   0               /* RSPI Channel Select               */
```

(4) Used Pin Definitions

The definitions of the serial pins used are shown below. Specify the pin numbers for the used pins by referring to table 7.2, Used Pin Definitions.

```
/*--------------------------------------------------------------------------*/
/*  Define the control port.                                                */
/*--------------------------------------------------------------------------*/
/* Set to use port numbers and bit numbers */
#define SIO_DATAI_PORTNO   A              /* SIO DataIn Port No.           */
#define SIO_DATAI_BITNO    7              /* SIO DataIn Bit No.            */
#define SIO_CLK_PORTNO     A              /* SIO CLK Port No.              */
#define SIO_CLK_BITNO      5              /* SIO CLK Bit No.               */
#define SIO_DATAO_PORTNO   A              /* SIO DataOut Port No.          */
#define SIO_DATAO_BITNO    6              /* SIO DataOut Bit No.           */
#define SIO_PFC_SELECT     G              /* RSPI PFC SELECT.( Set 'G'or'H' ) */
```

Table 7.2 Used Pin Definitions

#define Definition	Set Value
SIO_DATAI_PORTNO	DataIn pin port number
SIO_DATAI_BITNO	DataIn pin bit number
SIO_CLK_PORTNO	CLK pin port number
SIO_CLK_BITNO	CLK pin bit number
SIO_DATAO_PORTNO	DataOut pin port number
SIO_DATAO_BITNO	DataOut pin bit number
SIO_PFC_SELECT	Port function register setting "x": PFxSPI, where x = G or H.

(5) RSPIS Bit Definition for the Used Port Function Register (PFxSPI)

This definition sets the port function register RSPIS bit (RSPI pin selection bit) to match the serial pins used.

```
/*--------------------------------------------------------------------------*/
/*  Define the control RSPIS Bit (RSPI Pin Select) of Port Function Control Register.    */
/*--------------------------------------------------------------------------*/
#define SIO_RSPI_SELECT    (uint8_t)(1)/* RSPI pin select set.(Set '0'or'1') */
```

(6) Mask Level Definition

Processing with interrupts disabled occurs during high-speed receive mode operation. To disable interrupts, specify the mask level with the highest priority. Note that this depends on the microcontroller used.

Set the mask level to 15 if the RX62N is used.

```
/*--------------------------------------------------------------------------*/
/*  Define the interrupt mask level.                                        */
/*  Set interrupt mask level due to protect an overrun error by interrupt.  */
/*--------------------------------------------------------------------------*/
#define SIO_INT_MASK_LEVEL    (uint8_t)(15)     /* Interrupt Mask Level     */
```

(7) Software Timer Definition

Set up the software timer that is used only by this sample code.

Set a value of 0.1 μs or larger as the initial value.

```
/*------------------------------------------------------------------------*/
/*  Define the wait time for timeout.                                      */
/*  Time out is occurred after 50000 times loop process of wait time.      */
/*------------------------------------------------------------------------*/
#define SIO_T_RSPI_WAIT      (uint16_t)(1)   /* 0.1us wait When CPU clock = 96MHz */
```

(8) Pull-up Resistor Control Register (PCR) Definitions

The PCR can be defined in the SIO_DATAI_INIT() inline function. If used, remove the commenting that hides this code. Note that only the port 9, ports A to E, and port G pins allow this PCR setting.

```
/*------------------ DataIn control ------------------*/
#pragma inline(SIO_DATAI_INIT)
static void SIO_DATAI_INIT(void)          /* DataIn Initial Setting         */
{
  SIO_ICR_DATAI    = 1;                   /* DataIn Input Buffer: Enable    */
  SIO_PCR_DATAI    = 0;     /** PA7/PC7/PE7 **//* DataIn Input Pull-up: off   */
  SIO_DDR_DATAI    = SIO_IN;              /* DataIn      Input              */
}
```

(9) Open-drain Control Register (ODR) Definitions

The ODR can be defined in the SIO_DATAO_INIT() and SIO_DATAO_CLK() inline functions. If used, remove the commenting that hides this code. Note that only the port 0 to port 3 and port C pins allow this setting.

```
/*----------------- DataOut control ------------------*/
#pragma inline(SIO_DATAO_INIT)
static void SIO_DATAO_INIT(void)                /* DataOut Initial Setting  */
{
/* SIO_ODR_DATAO  = 0; */  /** P26/PC6 **/      /* Open Drain Control: CMOS */


/*------------------- CLK control --------------------*/
#pragma inline(SIO_CLK_INIT)
static void SIO_CLK_INIT(void)                  /* CLK Initial Setting      */
{
/* SIO_ODR_CLK    = 0; */  /** P27/PC5 **/      /* Open Drain Control: CMOS */
```

8. Usage Notes

8.1 Notes on Embedding

When embedding this sample code in an application, include the files R_SIO.h and R_SIO_rspi.h (the renamed R_SIO_rspi.h.XXX).

8.2 Unused Functions

We recommend commenting out unused functions so that they do not consume ROM capacity unnecessarily.

8.3 Using a Different Microcontroller

Other microcontrollers can be handled easily.

Only the following two files need to be provided.

- A common I/O module definitions file corresponding to R_SIO_rspi.h.XXX
- A header definitions file corresponding to mtl_com.h.XXX.

Create these files based on the provided samples.

8.4 CRC Unit Module Stop Setting (option)

While functions that use the CRC calculation unit cancel the module stop state in initialization, there is no function that sets this module stop state. If it is necessary to set up the module stop state, the user must implement code that performs this control.

8.5 Input Buffer Control Register (PORTn.ICR) Setting

Since this sample code does not set any peripheral modules other than RSPI, this sample code must be used with the input functions disabled in other peripheral modules to which the pins are also allocated.

If the PORTn.ICR setting is changed in a state where these input functions are not disabled, edges in these pin states may be generated internally causing unexpected operations to occur.

8.6 Compiler Options

Operation has been verified with optimization level set to 2 and optimization method set to "prioritize size".

Operation has not been verified with optimization level set to 2 and optimization method set to "prioritize speed".

8.7 When Other Applications Use DMAC, EXDMAC, or DTC Transfers

When one of SIO_OPTION_4 to SIO_OPTION_6 is selected as the operating mode and contention for the bus that this sample code uses or a high-priority NMI interrupt occurs when another application uses DMAC, EXDMAC, or DTC transfers, overrun errors may occur due to receive data not being acquired in time.

To avoid this problem, select one of SIO_OPTION_1 to SIO_OPTION_3 as the operating mode.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description: Page	Summary
1.00	Oct.19.11	—	First edition issued
1.01	Dec.14.11	3	"Specification other than optimization option 0" for the C compiler removed from table 2.1.
		3	Sample code version number changed from 2.00 to 2.02 in table 2.1.
		3	Software used for evaluation in table 2.1: Corrected.
		9	6.3 Memory Requirements: Corrected.
		10	6.4 File Structure: Application note number corrected.
		11	6.5.3 Other Definitions: Header file deleted.
		14	6.8.1 Driver Initialization: Remarks corrected.
		15	6.8.2 Serial I/O Disable Setting: Remarks corrected and figure 6.5 corrected.
		16	6.8.3 Serial I/O Enable Setting: Remarks corrected and figure 6.6 corrected.
		18	6.8.5 Serial I/O Data Transmission Processing: Remarks corrected.
		19	Figure 6.8 corrected.
		20	Figure 6.9 corrected.
		21	6.8.6 Serial I/O Data Reception Processing: Remarks corrected.
		22	Figure 6.10 corrected. "Transmit data size modification" changed to "Receive data size modification" and "Receive data reordering" changed to "Receive data reordering and acquisition".
		23	Figure 6.11 corrected. "Transmit data size modification" changed to "Receive data size modification", "Transmit buffer empty" changed to "Transmit buffer full", and "Receive data reordering" changed to "Receive data reordering and acquisition".
		24	Figure 6.12 corrected. "Receive data reordering" changed to "Receive data reordering and acquisition".
		25	6.8.7 Transmit Data Endian Reordering: "static" removed from function declarations.
		26	6.8.8 Receive Data Endian Reordering: "static" removed from function declarations.
		28	6.9.3 Macro Function SIO_DATAI_INIT(): "Disable input buffer function" corrected to "Enable input buffer function" in (2) Function (1).
		33	6.9.12 Macro Function SIO_TX_DISABLE(): "Reads SPCR" added to (2) Function (2).
		34	6.9.14 Macro Function SIO_TRX_ENABLE(): "Clears IR flag" added to (2) Function (2).
		34	6.9.14 Macro Function SIO_TRX_EXABLE(): "SIO_TX_DISABLE()" corrected to " SIO_TRX_DISABLE()" in (3) Remarks.
		35	6.9.14 Macro Function SIO_TRX_DISABLE(): "Reads SPCR" added to (2) Function (2).
		38	7.1 mtl_com.h: (1) OS header file definitions changed.
		40	7.1.2 mtl_tim.h: Timer counter modified.
		41	7.2.1 R_SIO.h added
		46	8.4 Compiler Options: Deleted The following items were moved up due to this deletion.

Rev.	Date	Description Page	Summary
1.02	Dec.19.11	3	Added default settings group in table 2.1 compiler options.
		3	Changed the table 2.1 sample code version from 2.02 to 2.03.
		10	6.4 File Structure: Application note number corrected.
		46	Added section 8.6, Compiler Options
1.03	Mar.31.12	2	1. Specifications: Content modified.
		3	Changed the table 2.1 sample code version from 2.03 to 2.04.
		3	Added tables 2.2 and 2.3.
		5	Interchanged sections 5.1, Hardware Structure and 5.2 Pins
		6	6.1 Operation Overview: Moved part of the content to section 6.2.2, Relationship Between Data Buffers and Transmit/Receive Data.
		7	6.2.1 Software Structure: Added new content
		8	Added section 6.2.2 Relationship Between Data Buffers and Transmit/Receive Data.
		8	Deleted sections 6.2.3 to 6.2.6.
		9	6.3 Memory Requirements: Added content and updated memory sizes.
		10	6.4 File Structure: Application note number corrected.
		11	6.5.1 Return Values: Original Japanese used a different spelling.
		11	Table 6.4 Definition Values: Changed technical terms used in column headers.
		13	Table 6.6 Functions: Deleted R_SIO_Tx_Exchg() and R_SIO_Rx_Exchg().
		14	6.8 State Transition Diagram: Moved from 6.10 to 6.8.
		15	6.9.1 Driver Initialization: Modified the notes.
		16	6.9.2 Disable Serial I/O Processing: Modified notes and processing overview.
		17	6.9.3 Enable Serial I/O Processing: Modified notes and processing overview.
		18	6.9.4 Serial I/O Open Processing: Modified notes and processing overview.
		19 to 21	6.9.5 Transmit Serial I/O Data: Modified notes and processing overview.
		22 to 27	6.9.6 Receive Serial I/O Data: Added "Normal receive mode" and "High-speed receive mode" to the description column. Modified notes and processing overview.
		27	Deleted 6.9.7, Transmit Data Endian Reordering Processing, and 6.8.8, Receive Data Endian Reordering Processing.
		28 to 37	6.10 Inline Function Specifications: Originally called "Macro Functions"
		28	6.10.1 SIO_IO_INIT(): Modified (2) Function.
		29	6.10.2 SIO_IO_OPEN(): Modified (2) Function.
		29	6.10.3 SIO_DATAI_INIT(): Modified (2) Function.
		30	6.10.4 SIO_DATAO_INIT(): Modified (2) Function and (3) Remarks.
		30	6.10.5 SIO_DATAO_OPEN() (2) : Modified (2) Function.
		31	6.10.6 SIO_CLK_INIT() (2) : Modified (2) Function and (3) Remarks.
		31	6.10.7 SIO_CLK_OPEN() (2) : Modified (2) Function.
		32	6.10.8 SIO_ENABLE() (2) : Modified (2) Function and (3) Remarks.
		33	6.10.9 SIO_DISABLE() (2) : Modified (2) Function.

Rev.	Date	Description Page	Summary
1.03	Mar.31.12	34	6.10.11 SIO_TX_ENABLE(): Modified (2) Function and (3) Remarks.
		35	6.10.12 SIO_TX_DISABLE() (2) : Modified (2) Function.
		36	6.10.13 SIO_TRX_ENABLE(): Modified (2) Function and (3) Remarks.
		37	6.10.14 SIO_TRX_DISABLE() (2) : Modified (2) Function.
		42	7.2.1 R_SIO.h (1): Content modified.
		42 to 43	7.2.2 (1) Operating Mode Definitions: Content modified.
		43	7.2.2 (2) CRC Calculation Type Definition: Content modified.
		43	7.2.2 (3) Used RSPI Channel Definition: Content modified.
		44	7.2.2 (4) Used Pin Definitions: Content modified.
		44	7.2.2 (5) RSPIS Bit Definition for the Used Port Function Register (PFxSPI): Content modified.
		45	7.2.2 (7) Software Timer Definition: Content modified.
		45	Added section 7.2.2 (8), Pull-up Resistor Control Register (PCR) Definitions. Deleted the /* SET */ comments to conform to the program.
		45	Added section 7.2.2 (9), Open-drain Control Register (ODR) Definitions. Deleted the /* SET */ comments to conform to the program.
		46	Added section 8.7, When Other Applications Use DMAC, EXDMAC, or DTC Transfers.
1.04	Nov.13.12	1	Overview: Content modified.
		7	6.2.1 Software Overview: Content modified.
		10	6.4 File Structure: Corrected application note numbers.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. Handling of Unused Pins

   Handle unused pins in accord 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 2.2