RX610 Group

Clock Synchronous Single Master Control Software Using the SCI

R01AN0534EJ0100
Rev.1.00
May 19, 2011

Introduction

This application note explains how to control a single master in clock synchronous (three-wire method) serial communications through the RX610 group's serial communications interface and how to use the sample code for this application.

The SPI mode single master can be controlled by adding control of SPI slave device selection through port control.

This sample code lies in a lower-level layer of the software for controlling a SPI device as a slave device.

Software in the upper-level layer for controlling the slave device is separately available, so please obtain this as well.

Target Device

Corresponding MCU: RX610 group

Device used for checking the operation of the sample code: Renesas Electronics R1EX25xxx series
SPI Serial EEPROM

When applying the contents of this application note to other series of microcomputers, make necessary modifications to and make extensive evaluations of the sample code according to the specifications for the microcomputer to be used.

Contents

1. Specifications

This software program controls a single master for clock synchronous (three-wire method) serial communications through the SCI of RX610 group products. The SPI mode single master can be controlled by adding control of SPI slave device selection through port control.

Table 1 summarizes the peripheral devices to be used and their uses. Figure 1 illustrates a sample configuration.

The major functions are summarized below.

- This software is a block-type device driver for using the SCI of an RX610 as the master device in clock synchronous single master communications.
- It can only be used with a single user-configured channel; that is, it cannot be used with multiple channels.
- The sample code does not support chip-select control. To control the SPI device, the chip-select control must be separately embedded.
- Both big endian and little endian modes are supported
- This software supports MSB-first transfer.
- The software supports transfer by the CPU but not by the DMAC.
- It does not support using an interrupt to start the transfer.

Table 1 Peripheral Devices Used and their Uses

Peripheral Device	Use
SCI	Clock synchronous (three-wire method) serial 1 channel (required)
Port	For SPI slave device select control signals. As many ports as there are SPI slave devices in use are necessary (required). Not used by this sample code.

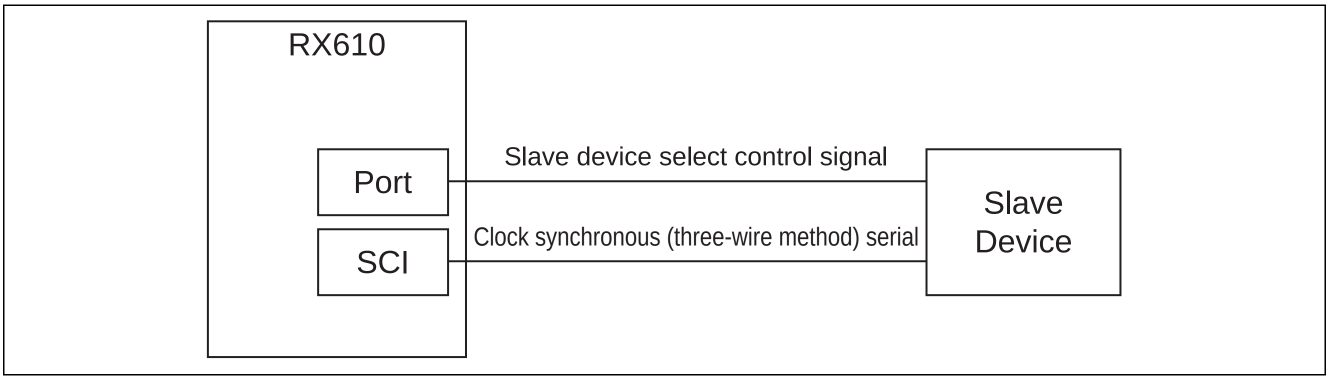


Figure 1 Sample Configuration

2. Conditions of Checking the Operation of the Software

The sample code described in this application note has been confirmed to run normally under the operating conditions given below.

Table 2 Operating Conditions

Item	Description
Microcomputer used for evaluation	RX610 group (program ROM 2 MB/RAM 128 KB)
Memory used for evaluation	Renesas Electronics R1EX25xxx series SPI Serial EEPROM
Operating frequency	ICLK: 100 MHz, PCLK: 50 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance embedded Workshop Version 4.07.00.007
C compiler	Renesas Electronics RX family C/C++ compiler package (Toolchain 1.0.0.0)
	Compiler options: The default settings for the integrated development environment are used.
Endian	Big endian/Little endian
Version of the sample code	Ver.2.00
Software used for evaluation	Renesas Electronics The R1EX25xxx series' SPI serial EEPROM control software, Ver.2.00
Evaluation board used	Renesas Starter Kit for the RX610

3. Related Application Notes

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

- Renesas R1EX25xxx Series Serial EEPROM Control Software (R01AN0565EJ)

4. Hardware Description

4.1 List of Pins

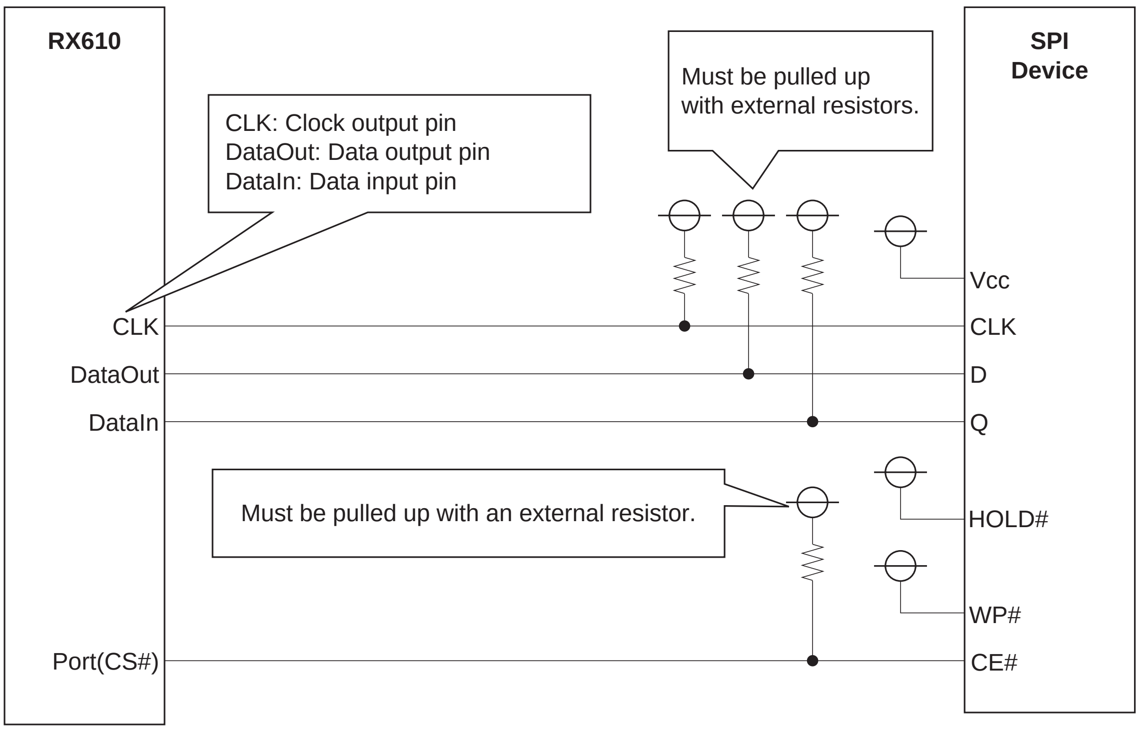
Table 3 lists the pins that are used and their uses.

Table 3 List of Pins Used

Pin Name	I/O	Description
SCK (CLK of figure 2)	Output	Clock output
TxD (DataOut of figure 2)	Output	Master data output
RxD (DataIn of figure 2)	Input	Master data input
Port (Port(CS#) of figure 2)	Output	Storage device select output Not used by this sample code.

4.2 Reference Circuit

Figure 2 shows a sample wiring configuration.



- The names of the MCU pins used for serial I/O are dependent on the individual MCU.
- In this application note, pin names CLK, DataIn, DataOut, and Port (CS#) are used in accordance with the sample code.

Figure 2 Sample Wiring Diagram for an RX610 SCI and an SPI Slave Device

5. Software Description

5.1 Operation Outline

The SCI's clock synchronous (three-wire method) serial communication function is used to realize the clock synchronous single master control.

The sample code explained in this section provides the following control functions:

- Controls the input/output of the data in the clock synchronous mode (using an internal clock).

In this sample code, the byte offset value of the data on the device is made equal to the byte offset value in the source or destination memory as illustrated in the figure below.

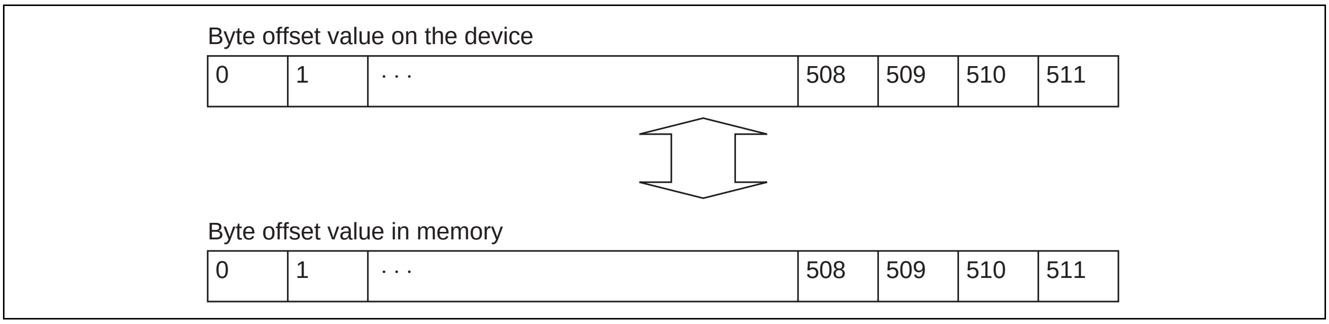
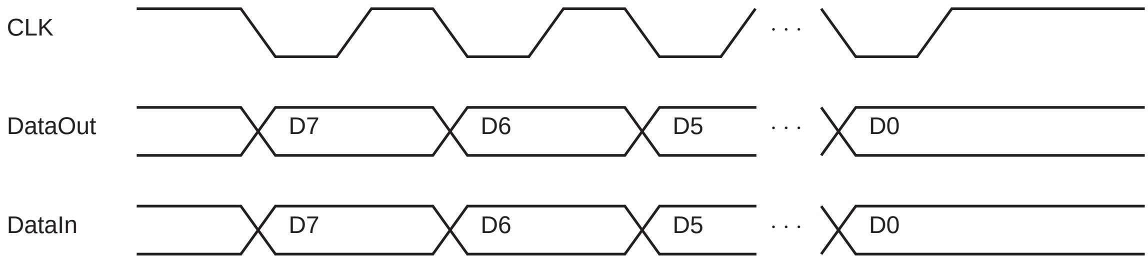


Figure 3 Storage Format of the Transferred Data

5.1.1 Clock Synchronous Mode Timing

The SPI mode 3 (CPOL=1, CPHA=1) timing shown in Figure 4 is used to control the SPI slave device.



- MCU->SPI slave device transmission: Transmission of transmit data is started on the falling edge of the transfer clock.
- SPI slave device->MCU reception: The receive data is taken in on the rising edge of the transfer clock.
- MSB first mode transfer

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

Figure 4 Clock Synchronous Mode Timing Setup

For available serial clock frequencies, see the datasheets for the individual MCUs and SPI slave devices.

5.1.2 SPI Slave Device CE# Pin Control

The CE# pin of the SPI slave device is not controlled in this sample code. To control the SPI device, the CE# pin control must be embedded in the SPI slave device.

The recommended method is connecting the CE# pin of the SPI slave device to is recommended to the port pin of the MCU and control it as an MCU general port output.

Secure the time between the falling edge of the CE# signal of the SPI device (the Port signal of the MCU (CS#)) and that of the CLK signal of the SPI device (the clock signal of the MCU) as the setup time of the CE# pin of the SPI device.

Secure the time between the rising edge of the CLK signal of the SPI device (the CLK signal of the MCU) and that of the CE# signal of the SPI device (the Port signal of the MCU (CS#)) as the hold time of the CE# pin of the SPI device.

Check the datasheet for the SPI device in use and set up the software wait times that are appropriate to your system.

5.2 Software Control Outline

5.2.1 Software Configuration

The sample code ranks in the lower-level layer of the SPI device control software as a slave device.

The sample code realizes the control the clock synchronous single master by using SPI mode 3 (CPOL = 1 and CPHA = 1) without controlling the CE# pin of the SPI slave device.

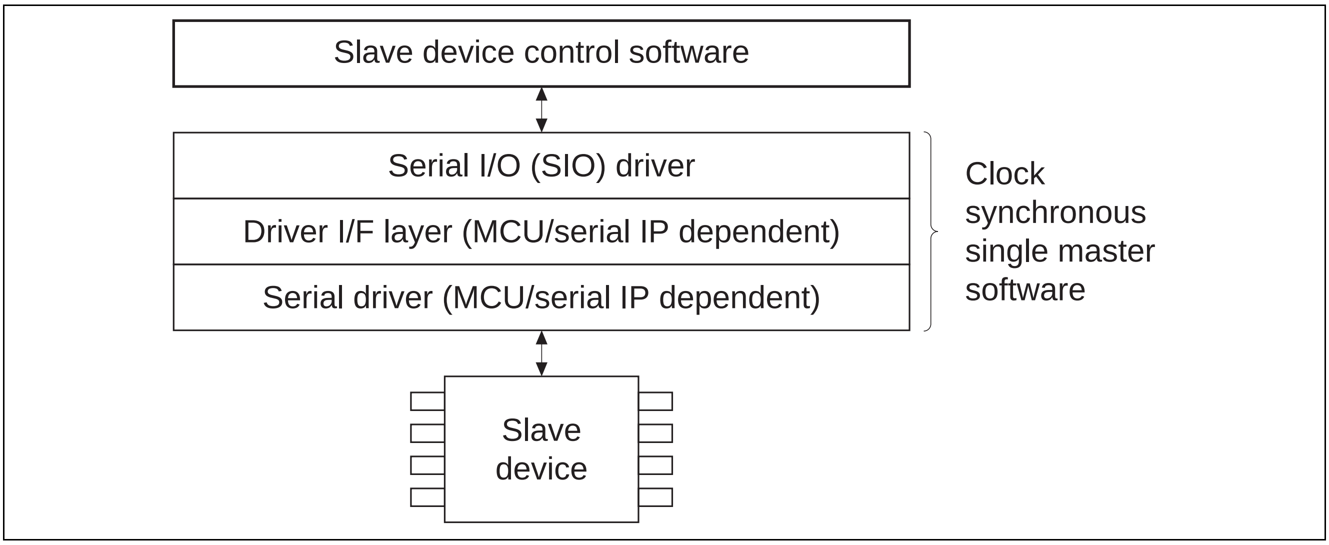


Figure 5 Software Configuration

The following transmission and reception are realized.

(1) Send data using the clock synchronous single master software.
(2) Receive data using the clock synchronous single master software.

This sample code is made up of the following five basic routines:

- Serial enabling
  Set the DataIn pin for port input, set the DataOut and CLK pins high, Enable serial I/O and set the bit rate.
- Serial disabling
  Disable serial I/O, set the DataIn pin for port input, set the DataOut and CLK pins high.
- Serial opening
  Disable serial I/O, set the DataIn pin for port input, set the DataOut and CLK pins for port input.
- Data transmission
  Send data to the SPI device.
- Data reception
  Receive data from the SPI device.

5.2.2 Serial Enabling (R_SIO_Enable())

Sets the DataIn pin to be used for serial I/O for port input and set the DataOut and CLK pins high.

Enables the serial I/O function and switches the DataIn pin for data input, the DataOut pin for data output, and the CLK pin for clock output.

Sets the baud rate (bit rate) to be used for serial I/O.

5.2.3 Serial Disabling (R_SIO_Disable())

This routine switches the pin to be used for serial I/O to a port pin and sets the DataIn pin for port input and sets the DataOut and CLK pins high.

5.2.4 Serial Opening (R_SIO_Open_Port())

This routine switches the pin to be used for serial I/O to a port pin and sets the DataIn, DataOut, and CLK pins for port input.

5.2.5 Data Transmission (R_SIO_Tx_Data())

This routine sends data using the serial I/O function.

This routine sends data according to the transmission setting.

5.2.6 Data Reception (R_SIO_Rx_Data())

This routine receives data using the serial I/O function.

This routine receives data according to the transmission/reception settings.

5.3 Sizes of Required Memory

Table 4 lists the sizes of the required memory areas.

Table 4 Sizes of Required Memory

Memory Used	Size	Remarks
ROM	838 bytes (little endian)	R_SIO_sci_rx.c
RAM	0 bytes (little endian)	R_SIO_sci_rx.c
Maximum user stack size	168 bytes	
Maximum interrupt stack size	—	

Note: The sizes of required memory areas vary with the version and compiler options of the C compiler. The above-mentioned memory sizes vary with the endian mode adopted.

5.4 File Configuration

Table 5 lists the files that are used for the sample code. The table excludes the files that are automatically generated by the integrated development environment.

Table 5 File Configuration

\an_r01an0534ej_rx610 <DIR>	Folder for the sample code
r01an0534ej0100_rx610.pdf	Application note
\r01an0534ej_rx610_src <DIR>	Folder for storing the programs
\com *1 <DIR>	Folder for storing the common functions
mtl_com.c	Miscellaneous common function definitions
mtl_com.h.common	Common header file
mtl_com.h.RX600	Common function header file
mtl_endi.c	Common file (related to endian setting)
mtl_mem.c	Common file (standard library function)
mtl_os.c mtl_os.h	Common file (standard library function)
mtl_str.c	Common file (standard library function)
mtl_tim.c mtl_tim.h	Common file (related to loop timer)
mtl_tim.h.sample	Sample for setting the value in the loop timer
\r_sio_sci_rx <DIR>	Folder for clock synchronous single master control software using the SCI for the RX610
R_SIO.h	Header file
R_SIO_sci.h.rx610	I/F module common definitions
R_SIO_sci_rx.c	I/F module

Note: *1 The file in the com folder is used in the slave device control software, too. Use the latest file.

5.5 List of Constants

5.5.1 Return Values

Table 6 lists the return values that are returned by the sample code.

Table 6 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

5.5.2 Miscellaneous Definitions

Table 7 lists miscellaneous definitions that are used in the sample code.

Table 7 Miscellaneous Definitions

Constant Name	Value	Description
SIO_LOG_ERR	1	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 × 1 μs = 50 ms
SIO_RX_WAIT	(uint16_t)50000	SIO receive completion waiting time 50000 × 1 μs = 50 ms
SIO_DMA_TX_WAIT	(uint16_t)50000	DMA transmission completion waiting time 50000 × 1 μs = 50 ms
SIO_DMA_RX_WAIT	(uint16_t)50000	DMA receive completion waiting time 50000 × 1 μs = 50 ms
SIO_T_SIO_WAIT	(uint16_t)MTL_T_1US	SIO transmit&receive completion waiting polling time
SIO_T_DMA_WAIT	(uint16_t)MTL_T_1US	DMA transmit&receive completion waiting polling time
SIO_T_BRR_WAIT	(uint16_t)MTL_T_10US	BRR setting wait time

5.6 Structures and Unions

Shown below are the structures that are used in the sample code.

```
/* 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                */
```

5.7 List of Functions

Table 8 lists the functions that are used in the sample code.

Table 8 List of Functions

Function Name	Outline
R_SIO_Init_Driver()	Initialize driver.
R_SIO_Disable()	Disable serial I/O.
R_SIO_Enable()	Enable serial I/O.
R_SIO_Open_Port()	Open serial I/O.
R_SIO_Tx_Data()	Send serial I/O data.
R_SIO_Rx_Data()	Receive serial I/O data.

5.8 Function Details

5.8.1 Driver Initialization

R_SIO_Init_Driver

Synopsis	Initializes driver.
Headers	R_SIO.h, R_SIO_sci.h, mtl_com.h
Declaration	error_t R_SIO_Init_Driver(void)
Explanation	• Initializes the driver. Disables the serial I/O function and set the pin in the port. • This function must be called only once at system start time. • Set the slave device select signal high before calling this function.
Arguments	None
Return value	SIO_OK ; Successful operation
Remarks	Performs the following processing, considering the previous use conditions. • Stops transmission/reception. • Clears the PER, FER, and OERE flags of the SSR.

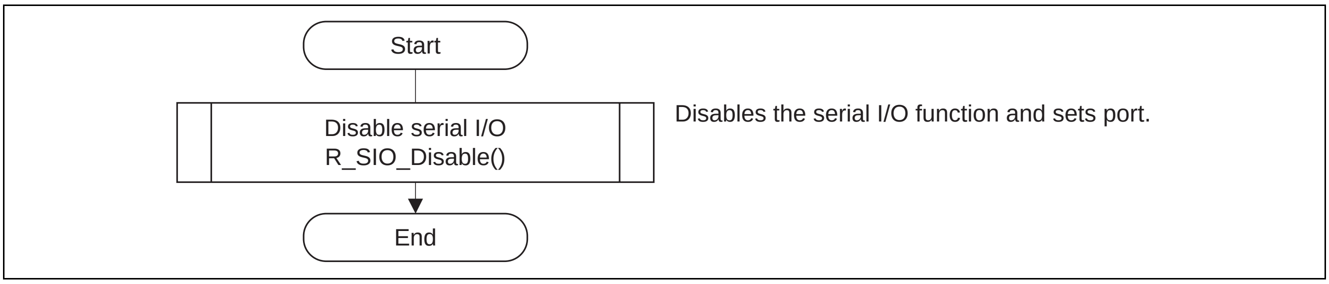


Figure 6 Driver Initialization Processing Outline

5.8.2 Serial I/O Disable Setup Processing

R_SIO_Disable

Synopsis	Performs serial I/O disable setup processing.
Headers	R_SIO.h, R_SIO_sci.h, mtl_com.h
Declaration	error_t R_SIO_Disable(void)
Explanation	• Disables the serial I/O function and sets the pin in the port. Disables serial I/O. Sets the pin to be used for serial I/O in the port. • Set the slave device select signal high before calling this function.
Arguments	None
Return value	SIO_OK ; Successful operation
Remarks	• Writes 00h to SMR and SCR to initialize the driver. (Writes 00h to SCR according to the initialization procedure in the hardware manual) • For transmission and reception, reads the PER, FER, and OERE flags of SSR and then clears them to 0. • Sets serial I/O to be used to the module stop state. • Can be called to disable the serial I/O function when serial I/O is not used.

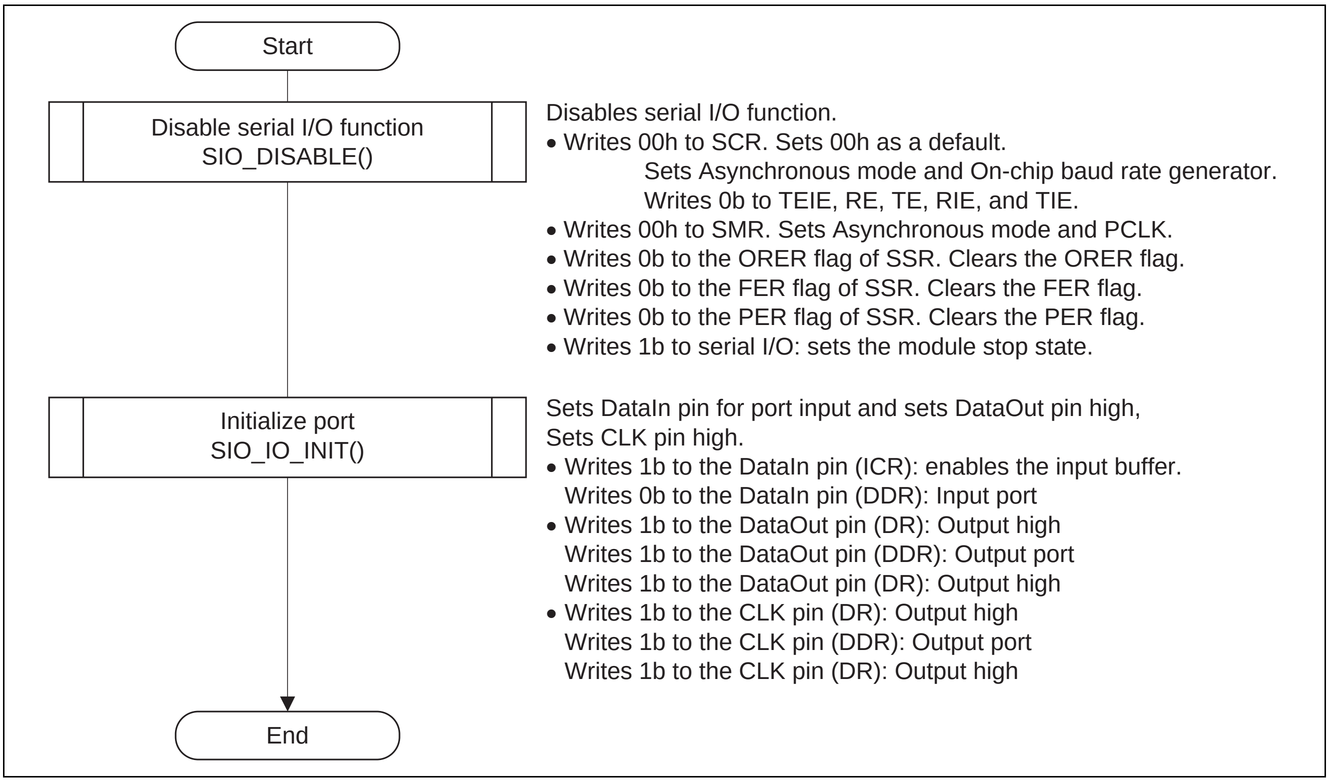
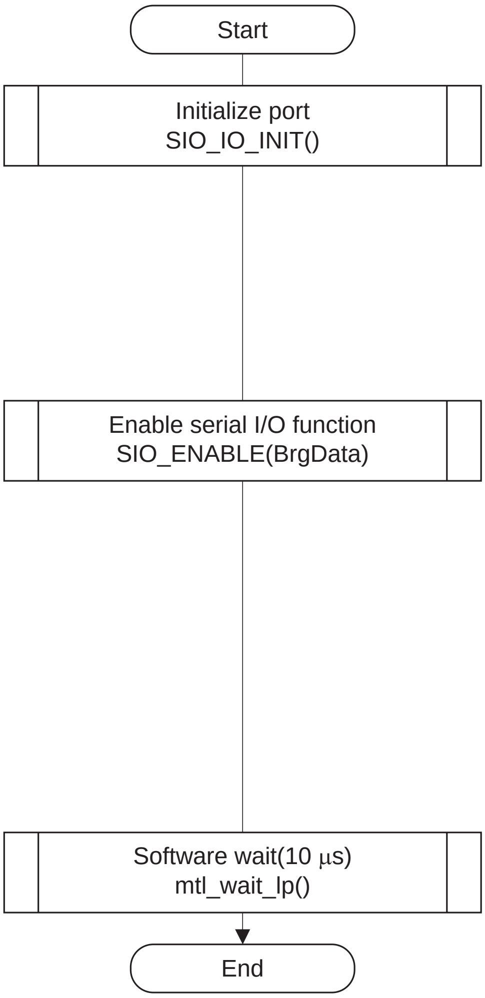


Figure 7 Serial I/O Disable Setup Processing Outline

5.8.3 Serial I/O Enable Setup Processing

R_SIO_Enable	
Synopsis	Performs serial I/O enable setup processing.
Headers	R_SIO.h, R_SIO_sci.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 pin to be used for serial I/O in the port. Enables the serial I/O and sets the bit rate. • Call this function after calling R_SIO_Disable() • Call this function before performing serial I/O data transmission processing and serial I/O data reception processing. • To change the bit rate, disable serial I/O setting, and then, use this function.
Arguments	uint8_t BrgData ; Bit rate setting value
Return value	SIO_OK ; Successful operation
Remarks	• Sets the serial I/O to be used to the module stop off state. • Executes the following processing according to the initialization flow chart in the hardware manual. (Assumes that a default value 00h is written to SCR by calling R_SIO_Disable()) (1) SCR TIE=RIE=TE=RE=TEIE=0b (A default value, 00h, has been written to SCR) (2) Set CKE of SCR (by this function) (3) Set SMR (by this function) (4) Set SCMR (by this function) (5) Set BBR (by this function) (6) Software wait time: 10 μs: assumes that the bit rate is 0.1 Mbps or more (by this function) Reconsider the wait time if wait time is not enough.



Sets DataIn pin for port input and sets DataOut pin high,
Sets CLK pin high.

- Writes 1b to the DataIn pin (ICR): enables the input buffer.
  Writes 0b to the DataIn pin (DDR): Input port
- Writes 1b to the DataOut pin (DR): Output high
  Writes 1b to the DataOut pin (DDR): Output port
  Writes 1b to the DataOut pin (DR): Output high
- Writes 1b to the CLK pin (DR): Output high
  Writes 1b to the CLK pin (DDR): Output port
  Writes 1b to the CLK pin (DR): Output high

Enables the serial I/O function and sets the bit rate.

- Writes 0b to serial I/O: Turns off the module stop state.
- Writes 01h to SCR. Sets CKE.
  Writes 01b to CKE. (Sets the SCK pin of the internal clock to clock output at clock synchronous setup)
  Writes 0b to TEIE, RE, TE, RIE, and TIE.
- Writes 80h to SMR. Selects PCLK and clock synchronous mode.
- Writes 72h to SCMR. LSB first.
- Writes 0b to the ORER flag of SSR. Clears the ORER flag.
- Writes 0b to the FER flag of SSR. Clears the FER flag.
- Writes 0b to the PER flag of SSR. Clears the PER flag.
- Writes 00h to SEMR. (Not required to do because it is the setup in asynchronous mode. Sets the default.)
- Sets the bit rate in BRR.

Figure 8 Serial I/O Enable Setup Processing Outline

5.8.4 Serial I/O Open Setup Processing

R_SIO_Open_Port	
Synopsis	Performs SIO port (DataOut, DataIn, and CLK) open setup processing.
Headers	R_SIO.h, R_SIO_sci.h, mtl_com.h
Declaration	error_t R_SIO_Open_Port(void)
Explanation	• Sets the pin used for serial I/O to "open" (input state). • Set the slave device select signal high before calling this function.
Arguments	None
Return value	SIO_OK ; Successful operation
Remarks	• Prepared to connect and disconnect removable media. Use this function before connecting and disconnecting the removable media. Perform serial I/O disable setup processing before disconnecting the removable media.

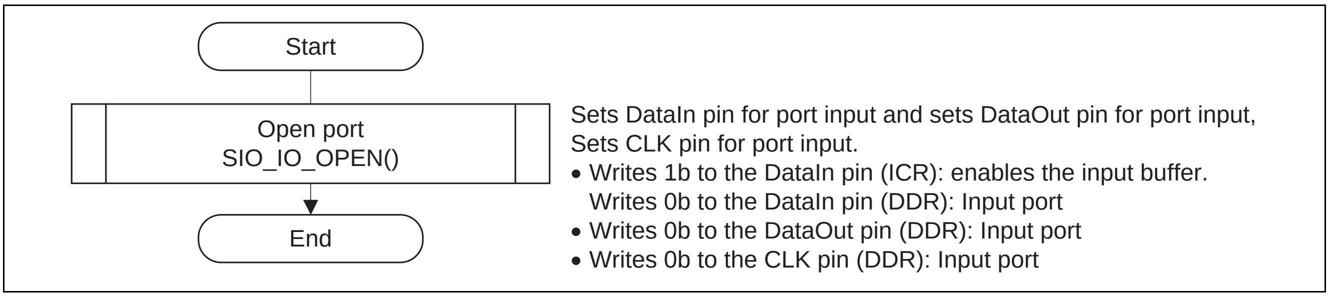


Figure 9 Serial I/O Open Setup Processing Outline

5.8.5 Serial I/O Data Transmission Processing

R_SIO_Tx_Data

Synopsis	Performs serial I/O data transmission processing.
Headers	R_SIO.h, R_SIO_sci.h, mtl_com.h
Declaration	error_t R_SIO_Tx_Data(uint16_t TxCnt, uint8_t FAR* pData)
Explanation	• Transmits a specified number of bytes of pData. • Perform serial I/O enable setup processing before calling this function. • Perform serial I/O disable setup processing in case of unsuccessful operation after calling this function.
Arguments	uint16_t TxCnt ; Number of transmitted bytes uint8_t FAR* pData ; Transmit data storage buffer pointer
Return value	SIO_OK ; Successful operation SIO_ERR_HARD ; Hardware error
Remarks	• Executes the following processing according to the initialization flow chart in the hardware manual. (1) Sets TE, RE, TIE, RIE, and TEIE of SCR. • After completion of transmission, set TE, RE, TIE, RIE, and TEIE to 0b according to the serial transmission flow chart in the hardware manual. • Recommended to perform serial I/O disable setup processing if this function is not continuously used.

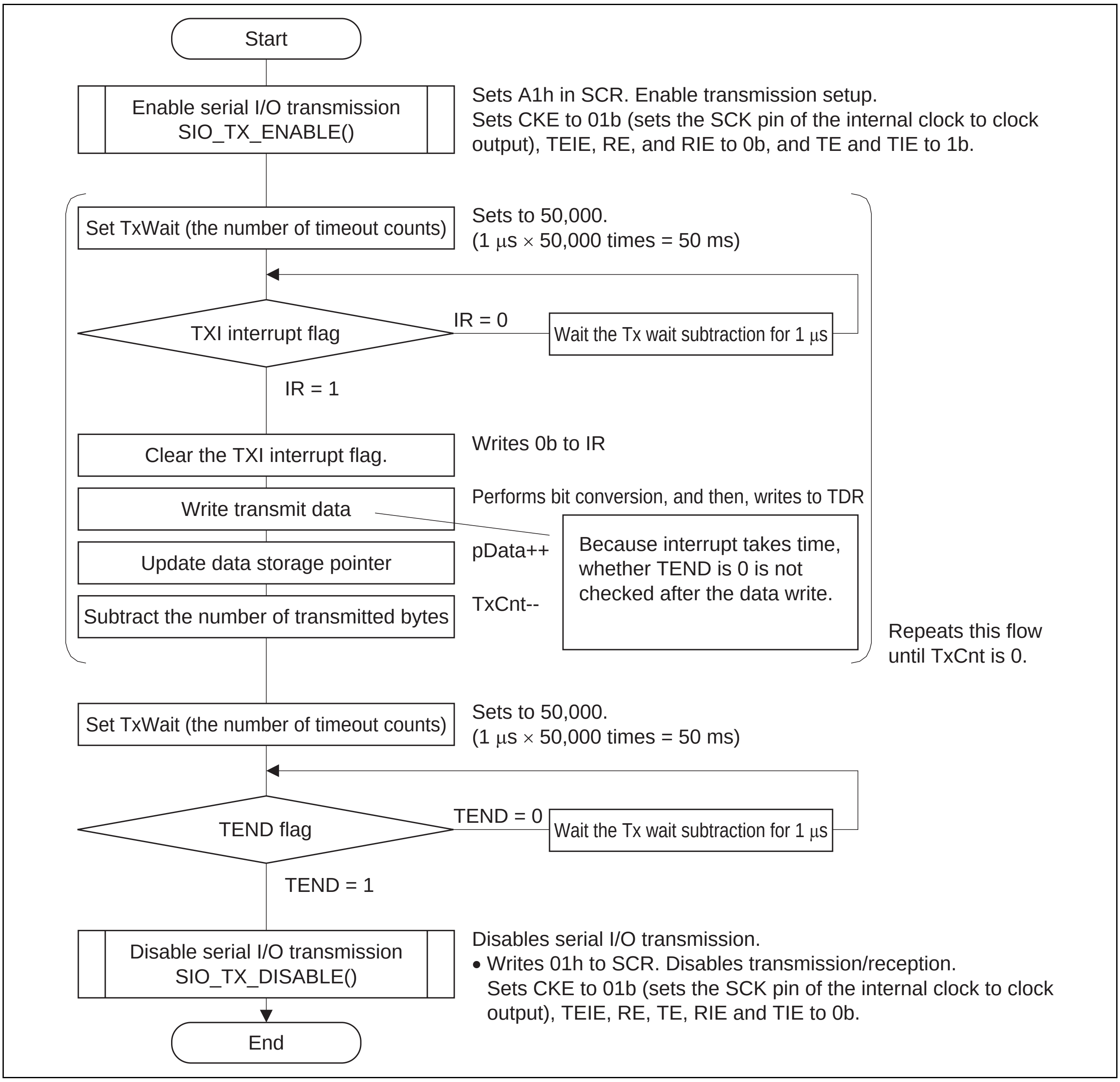


Figure 10 Serial I/O Data Transmission Processing Outline

5.8.6 Serial I/O Data Reception Processing

R_SIO_Rx_Data

Synopsis	Performs serial I/O data reception processing.
Headers	R_SIO.h, R_SIO_sci.h, mtl_com.h
Declaration	error_t R_SIO_Rx_Data(uint16_t RxCnt, uint8_t FAR* pData)
Explanation	• Receives a specified number of data and stores it in pData. • Perform serial I/O enable setup processing before calling this function. • Perform serial I/O disable setup processing in case of unsuccessful operation after calling this function.
Arguments	uint16_t RxCnt ; Number of received bytes uint8_t FAR* pData ; Receive data storage buffer pointer
Return value	SIO_OK ; Successful operation SIO_ERR_HARD ; Hardware error
Remarks	• Executes the following processing according to the initialization flow chart in the hardware manual. (1) Sets TE, RE, TIE, RIE, and TEIE of SCR. • After completion of reception, set TE, RE, TIE, RIE, and TEIE to 0b according to the serial transmission flow chart in the hardware manual. • Not cause overrun errors because clock is generated by one-byte dummy data transmission and one-byte master reception is performed. Therefore, overflow confirmation processing is skipped in the flow chart. • Recommended to perform serial I/O disable setup processing if this function is not continuously used.

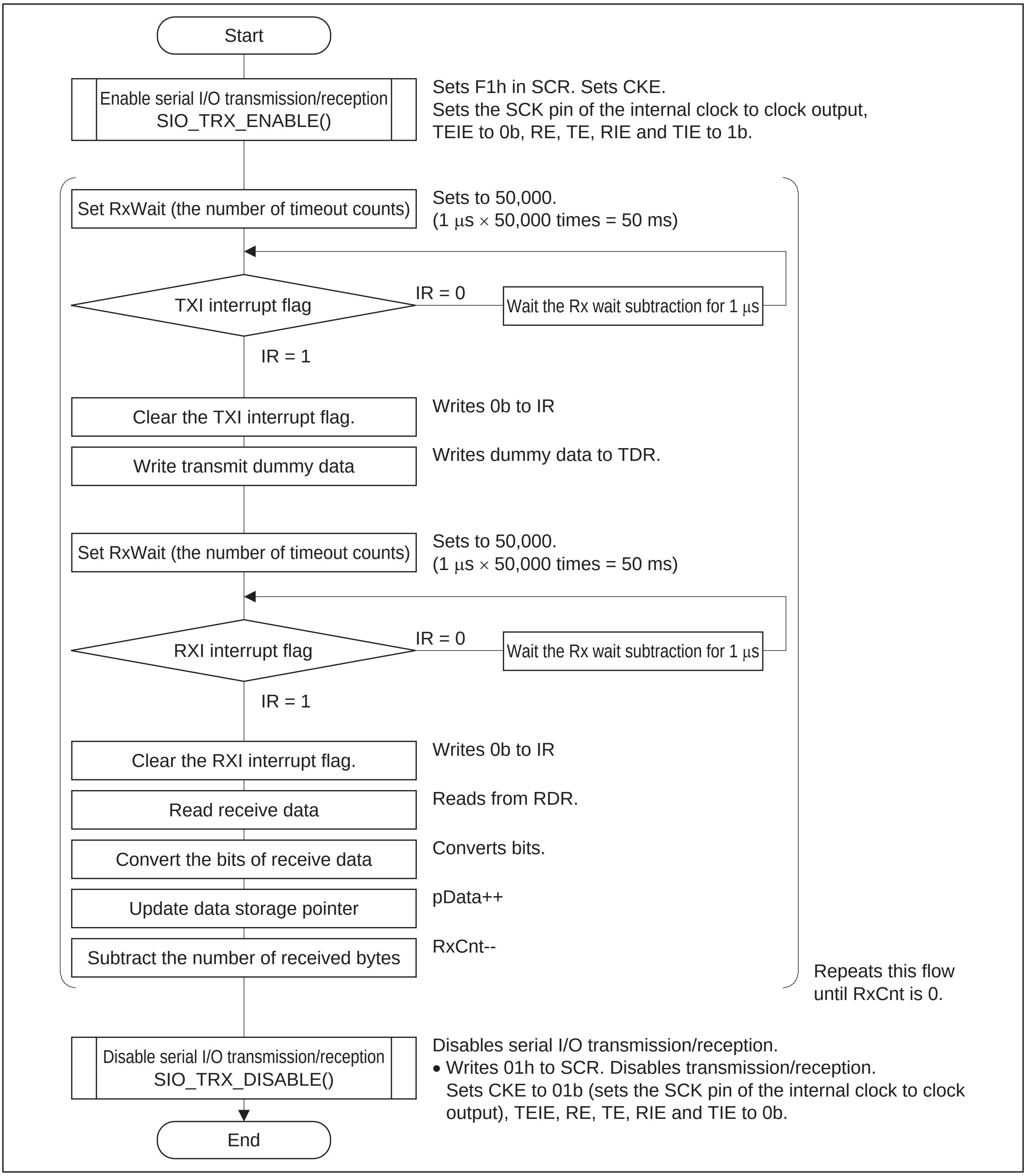


Figure 11 Serial I/O Data Reception Processing Outline

5.9 Macro Function Specifications

The macro function used in this sample code is explained below.

5.9.1 Macro Function SIO_IO_INIT()

1. Purpose
   Sets the input pin to the port input state and the output pin to the port output state.
2. Function
   Sets the DataIn pin to the port input state and the DataOut and CLK pins to the port output state.
   Performs the following processing. Review the processing as necessary.
   (1) Sets the DataIn pin to the port input.
   (2) Sets the DataOut pin to the port "H" output.
   (3) Sets the CLK pin to the port "H" output.

5.9.2 Macro Function SIO_IO_OPEN()

1. Purpose
   Sets the input and output pins to the port input state.
2. Function
   Sets the DataIn, DataOut, and CLK input pins to the port input state.
   Performs the following processing. Review the processing as necessary.
   (1) Sets the DataIn pin to the port input.
   (2) Sets the DataOut pin to the port input.
   (3) Sets the CLK pin to the port input.
3. Remarks
   Use this function to put all the pins in the Hi-z state before connecting and after disconnecting the removable media.

5.9.3 Macro Function SIO_DATAI_INIT()

1. Purpose
   Sets the DataIn pin to the port input state.
2. Function
   Performs the following processing. Review the processing as necessary.
   (1) Sets the DataIn pin to the port input.

5.9.4 Macro Function SIO_DATAO_INIT()

1. Purpose
   Sets the DataOut pin to the port "H" output.
2. Function
   Performs the following processing. Review the processing as necessary.
   (1) Sets the DataOut pin to the port "H" output.

5.9.5 Macro Function SIO_DATAO_OPEN()

1. Purpose
   Sets the DataOut pin to the port input state.
2. Function
   Performs the following processing. Review the processing as necessary.
   (1) Sets the DataOutn pin to the port input.

5.9.6 Macro Function SIO_CLK_INIT()

1. Purpose
   Sets the CLK pin to the port "H" output.
2. Function
   Performs the following processing. Review the processing as necessary.
   (1) Sets the CLK pin to the port "H" output.

5.9.7 Macro Function SIO_CLK_OPEN()

1. Purpose
   Sets the CLK pin to the port input state.
2. Function
   Performs the following processing. Review the processing as necessary.
   (1) Sets the CLK pin to the port input.

5.9.8 Macro Function SIO_ENABLE()

1. Purpose

   Initializes serial I/O and enables the function. Performs the common processing to enable transmission, reception, or transmission/reception. Furthermore, sets the bit rate.

2. Function

   Enables serial I/O according to the hardware manual. Reconsider the processing as necessary.

   Performs the following processing in the RX610.

   (1) Sets the module stop off state by using the module stop control register.

   (2) Performs the common processing to enable transmission and transmission/reception setups.

   Sets the following common parts of transmission and transmission/reception setups.

   - Sets TIE, RIE, TE, RE, and TEIE of SCR to 0.
   - Sets the CKE[1:0] bits of SCR.
   - Sets transmission/reception format in the SMR and SCMR.
   - Reads the ORER, FER, and PER flags of SSR, clear them to 0, and check whether these flags have been cleared.
   - Sets SEMR.
   - Writes a value in BRR.

3. Remarks

   Perform wait processing, after the bit rate is set and then the macro function is completed, in case of serial I/O requiring wait processing.

   Paired with SIO_DISABLE(). Perform SIO_DISABLE() and then finish the processing, if SIO_ENABLE() is performed.

5.9.9 Macro Function SIO_DISABLE()

1. Purpose

   Disables the serial I/O function.

2. Function

   Disables the serial I/O function. Performs the common processing to disable transmission and transmission/reception setups. Reconsider the processing as necessary.

   Performs the following processing in the RX610.

   (1) Sets 00h as a default value in SCR and stops transmission/reception.

   (2) Sets 00h as a default value in SMR.

   (3) Reads the ORER, FER, and PER flags of SSR, clear them to 0, and check whether these flags have been cleared.

   (4) Sets the module stop state by using the module stop control register.

3. Remarks

   Paired with SIO_ENABLE(). Perform SIO_DISABLE() and then finish the processing, if SIO_ENABLE is performed.

5.9.10 Macro Function SIO_TX_ENABLE()

1. Purpose
   Enables serial I/O transmission.
2. Function
   Enables serial I/O transmission according to the hardware manual. Enables the transmission after switching the pin from the port function to serial I/O function. Reconsider the processing as necessary.
   Performs the initialization procedure for the rest after SIO_ENABLE() and for transmission setting only.
   Performs the following processing in the RX610.
   (1) Enables transmission.
   Sets TE and TIE of SCR to 1b and then enables transmission.
3. Remarks
   Paired with SIO_TX_DISABLE(). Perform SIO_TX_DISABLE() and then finish the processing, if SIO_TX_ENABLE is performed.

5.9.11 Macro Function SIO_TX_DISABLE()

1. Purpose
   Disables the serial I/O transmission function.
2. Function
   Disables transmission according to the inverse processing of SIO_TX_ENABLE(). Switches the pin from the serial I/O function to the port function after disabling transmission. Reconsider the processing as necessary.
   Performs the following processing in the RX610.
   (1) Disables transmission.
   Sets TE, RE, TIE, RIE, and TEIE of SCR to 0b and then disables transmission.
3. Remarks
   Paired with SIO_TX_ENABLE(). Perform SIO_TX_DISABLE() and then finish the processing, if SIO_TX_ENABLE is performed.

5.9.12 Macro Function SIO_TRX_ENABLE()

1. Purpose

   Enables serial I/O transmission/reception.

2. Function

   Enables serial I/O transmission/reception according to the hardware manual. Enables the transmission/reception after switching the pin from the port function to serial I/O function. Reconsider the processing as necessary.

   Performs the initialization procedure for the rest after SIO_ENABLE() and for transmission/reception setting only.

   Performs the following processing in the RX610.

   (1) Enables transmission/reception.

   Sets TE, RE, TIE and RIE of SCR to 1b and then enables transmission/reception.

3. Remarks

   Paired with SIO_TRX_DISABLE(). Perform SIO_TRX_DISABLE() and then finish the processing, if SIO_TRX_ENABLE is performed.

5.9.13 Macro Function SIO_TRX_DISABLE()

1. Purpose

   Disables the serial I/O transmission/reception function.

2. Function

   Disables transmission/reception according to the inverse processing of SIO_TRX_ENABLE(). Switches the pin from the serial I/O function to the port function after disabling transmission/reception. Reconsider the processing as necessary.

   Performs the following processing in the RX610.

   (1) Disables transmission/reception.

   Sets TE, RE, TIE, RIE, and TEIE of SCR to 0b and then disables transmission/reception.

3. Remarks

   Paired with SIO_TRX_ENABLE(). Perform SIO_TRX_DISABLE() and then finish the processing, if SIO_TRX_ENABLE is performed.

5.9.14 Macro Function SIO_SSR_CLEAR()

1. Purpose

   Clears the error flag in SSR.

2. Function

   Clears the ORER, FER, and PER flags.

   Performs the following processing on each of the flags in case of RX610.

   (1) Clears the flags to 0 if they are set to 1.

   (2) Reads the flags to confirm that they are set to 0.

5.10 State Transition Diagram

Figure 12 shows the state transition diagram.

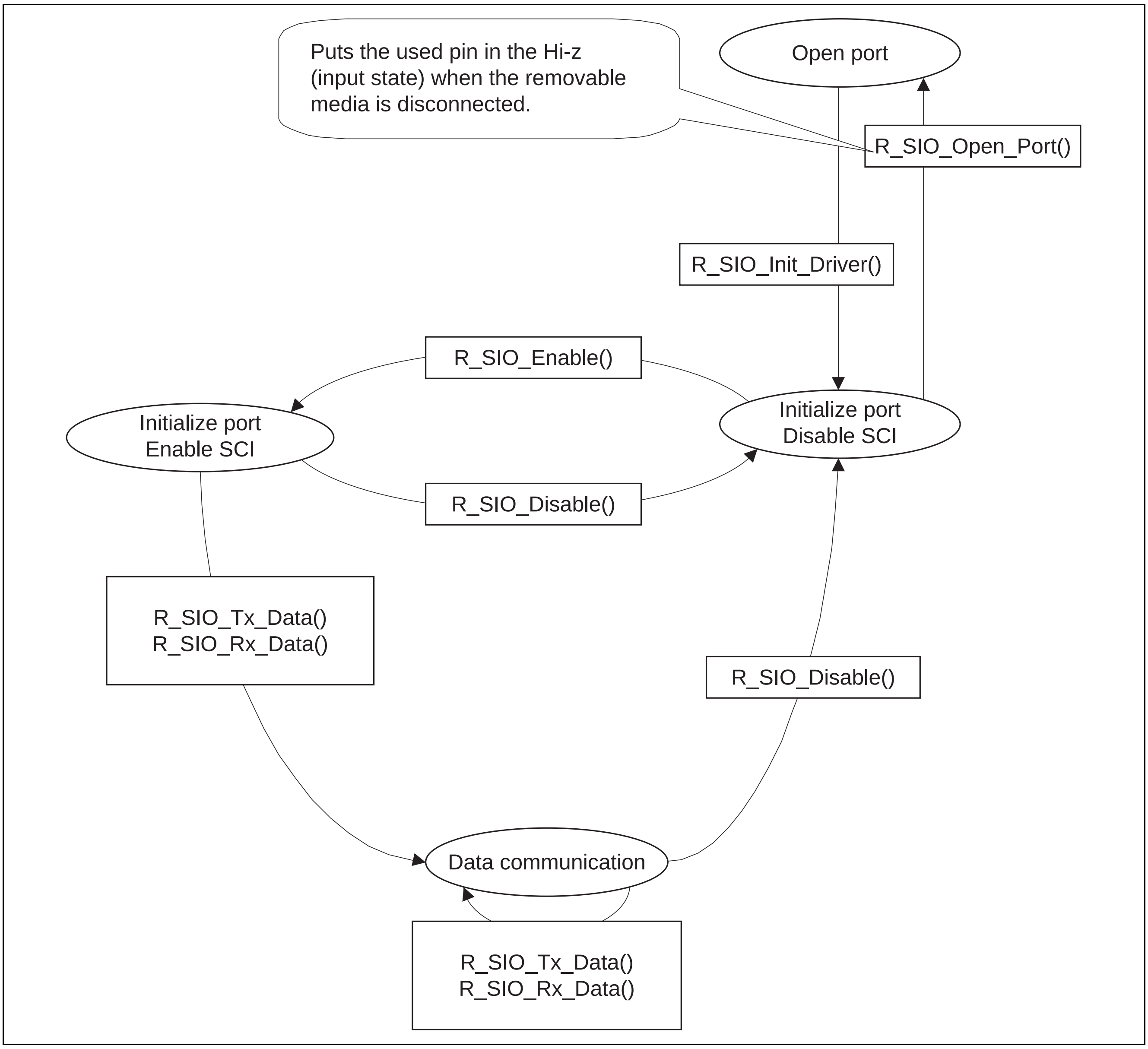


Figure 12 State Transition Diagram

6. Application Example

This section gives an example of settings for the serial I/O control section.

Examples of the settings for usage are given below.

The locations where settings are made are identified by the comments header "/** SET **/" in the defining file.

6.1 mtl_com.h (common header file)

This is the header file for functions to be in common use.

Each mtl_com.h.XXX (excluding mtl_com.h.common) is made for the evaluation of a given MCU. Use the appropriate header file after renaming it mtl_com.h. If there is no header file for the MCU to be evaluated, make mtl_com.h with reference to mtl_com.h.XXX.

1. Defining the header files for the OS
   This sample code is independent of the OS.
   In the example given below, the OS is not to be used.
   That is, the settings in the sample code are for when the OS is not to be used, so the code is independent of the OS.
   This sample code does, however 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. Defining the header file with the common access area defined
   The header file of MCU function register definitions is included.
   The main reason for including this header file is for the device driver to use the port pins.
   Include the header file that corresponds to the MCU.
   The header file for the RX610 is included in the example below.
   This header file must be included if the sample code is to be 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. Defining the loop timer
   The following header file is included so that the software loop timer is available for use.
   This is used to secure waiting time for the device driver.
   Comment out the "#include" directive if the software loop timer is not to be used.
   The software loop timer is to be used in this example.
   This header file must be included if the sample code is to be used.

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

4. Defining the endian
   Either little endian or big endian can be specified.
   The setting below is for big endian.

```
/* 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. Defining high-speed endian processing
   High-speed processing by mtl_end.c can be specified. Processing becomes high-speed if the M16C is in use.
   In the case of the RX family, leave this commented out so that the definition is not made.

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

6. Defining the standard library to be used
   Define the type of standard library to be used.
   Leave the "#define" below commented out if the library attached to the compiler is to handle the indicated processing.
   The library attached to the compiler is to be used in the example below.

```
/* 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. Defining the RAM area to be accessed
   Define the RAM area to be accessed.
   This obtains more efficient processing by standard functions and some other processes.
   Define MTL_MEM_NEAR in the case of the RX family.

```
/* Define the RAM area to be accessed by the user process.                 */
/* Efficient operations for standard functions and processes are applied.  */
//#define MTL_MEM_FAR         /* Supports Far RAM area of M16C/60          */
#define MTL_MEM_NEAR          /* Supports Near RAM area.       (Others)    */
```

6.1.1 mtl_tim.h

This is included by the include directive for the loop timer in mtl_com.h.

The effects of the settings depend on the MCU, clock, and compiler options in use.

If the system is cache-equipped, make settings on the assumption that the instruction cache is enabled and that the code for loop-timer processing is stored in the cache.

Repeat measurement and adjust the settings according to the conditions of usage.

```
/* Define the counter value for the timer.                                   */
/* Specify according to the user MCU, clock and wait requirements.           */
#if 1
/* Setting for 12.5MHz no wait Ix8 = 100MHz(Compile Option "-optimize=1" or "-
optimize=1 -speed")*/
#define MTL_T_1US             30    /* loop Number of   1us          */
#define MTL_T_2US             60    /* loop Number of   2us          */
#define MTL_T_4US            120    /* loop Number of   4us          */
#define MTL_T_5US            150    /* loop Number of   5us          */
#define MTL_T_10US           300    /* loop Number of  10us          */
#define MTL_T_20US           600    /* loop Number of  20us          */
#define MTL_T_30US           900    /* loop Number of  30us          */
#define MTL_T_50US          1500    /* loop Number of  50us          */
#define MTL_T_100US         3000    /* loop Number of 100us          */
#define MTL_T_200US         6000    /* loop Number of 200us          */
#define MTL_T_300US         9000    /* loop Number of 300us          */
#define MTL_T_400US  ( MTL_T_200US  *  2 )   /* loop Number of 400us */
#define MTL_T_1MS          30000    /* loop Number of   1ms          */
#endif
```

Times for the above values have not been measured, so the settings are not necessarily appropriate. Perform evaluation as required.

6.2 Setting up the Control Software for Clock Synchronous Single Master Operation

The locations where settings are made are identified by the comments header "/** SET **/" in the defining file.

6.2.1 R_SIO.h

1. Defining the wait time after setting up the BRR
   Setting the BRR of the SCI is followed by a software wait until one bit of data is transferred. Set this wait time as required.
   The default setting is for 10 µs.
   Supposing transfer at 100 kHz and usage with Multimedia Cards, make the setting for 10 µs.

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

6.2.2 R_SIO_sci.h

This is the definition file for the SCI.

Each R_SIO_sci.h.XXX is made for the evaluation of a given MCU. Use the appropriate header file after renaming it R_SIO_sci.h. If there is no header file for the MCU to be evaluated, make R_SIO_sci.h with reference to the R_SIO_sci.h.XXX files.

1. Defining the operating mode to be used
   The resources of the MCU to be used can be set.
   If processing is to be of MSB-first CRC-CCITT calculations, specify SIO_OPTION_2 as in the following example.
   CRC-CCITT calculations are unnecessary when control is of serial EEPROM or serial Flash memory. In such cases, comment the definition out.
   The separate R_SIO_sci_rx_mmc.c file is needed to perform CRC-CCITT calculations for controlling Multimedia Cards.

```
/*-------------------------------------------------------------------------*/
/*  Define the combination of the MCU's resources.                         */
/*-------------------------------------------------------------------------*/
//#define SIO_OPTION_1     /* Low speed*/ /* SI/O                          */
#define SIO_OPTION_2       /*           */ /* SI/O       + CRC calculation  */
```

2. Defining the form of CRC calculation to be used
   Define the form of CRC calculation to be used.
   CRC-CCITT calculation is not used when control is of serial EEPROM or serial Flash memory. In such cases, comment the definition out.
   To control multimedia cards, define both CRC-CCITT calculation and CRC-CCITT calculation at the same time.

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

3. Defining the pins to be used
   Define the pins to be used.

```
/*-------------------------------------------------------------------------*/
/*  Define the control port.                                               */
/*  Delete comment of a related macrodefinition, and please validate setting. */
/*-------------------------------------------------------------------------*/
#define SIO_DR_DATAO     PORT2.DR.BIT.B6         /* SIO DataOut         */
#define SIO_PORT_DATAI   PORT2.PORT.BIT.B5       /* SIO DataIn          */
#define SIO_DR_CLK       PORT2.DR.BIT.B7         /* SIO CLK             */
#define SIO_DDR_DATAO    PORT2.DDR.BIT.B6        /* SIO DataOut         */
#define SIO_DDR_DATAI    PORT2.DDR.BIT.B5        /* SIO DataIn          */
#define SIO_DDR_CLK      PORT2.DDR.BIT.B7        /* SIO CLK             */
#define SIO_ICR_DATAI    PORT2.ICR.BIT.B5        /* SIO DataIn          */
```

4. Defining the module stop register
   Specify the module stop register that contains the stop bit for the SCIF to be used.

```
#define SIO_MSTPCR_SCI   SYSTEM.MSTPCRB.BIT.MSTPB30  /* SCI Module stop setting*/
```

5. Defining the SCI channel to be used
   Specify the SCI channel to be used.
   Channel 1 is used in the example below.

```
/*----------------- SIO definitions -----------------*/

#define SIO_SMR          SCI1.SMR.BYTE        /* Serial mode register        */
#define SIO_SCR          SCI1.SCR.BYTE        /* Serial control register     */
#define SIO_SSR          SCI1.SSR.BYTE        /* Serial status register      */
#define SIO_SCMR         SCI1.SCMR.BYTE       /* Smart card mode register    */
#define SIO_BRR          SCI1.BRR             /* Bit rate register           */
#define SIO_SEMR         SCI1.SEMR.BYTE       /* Serial extend mode register*/
#define SIO_TXBUF        SCI1.TDR       /* SCI Transmit FIFO data register   */
#define SIO_RXBUF        SCI1.RDR       /* SCI Receive FIFO data register    */

#define SIO_ORER         SCI1.SSR.BIT.ORER    /* SCI Overrun error flag      */
#define SIO_FER          SCI1.SSR.BIT.FER     /* SCI Framing error flag      */
#define SIO_PER          SCI1.SSR.BIT.PER     /* SCI Parity error flag       */
#define SIO_TXEND        SCI1.SSR.BIT.TEND    /* SCI Transmit end flag       */
#define SIO_TXNEXT       ICU.IR[220].BIT.IR   /* SCI Transmit data empty     */
#define SIO_RXNEXT       ICU.IR[219].BIT.IR   /* SCI Receive data full       */
```

7. Usage Notes

7.1 Usage Notes to be Observed when Building the Sample Code

To incorporate the sample code, include R_SIO.h and R_SIO_sci.h (after renaming R_SIO_sci.h.XXX).

7.2 Unnecessary Functions

Unused functions waste ROM capacity, so we recommend excluding them by commenting them out and so on.

7.3 Using Other MCUs

Other MCUs can easily be used.

The files to be prepared are as follows:

- A common I/O module definition file corresponding to R_SIO_sci.h.XXX
- A header definition file corresponding to mtl_com.h.XXX

Make them by referring the attachment.

Website and Support

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

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

Revision Record

Rev.	Date	Description: Page	Description: Summary
1.00	May 19, 2011	—	First edition issued

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.

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: +44-1628-585-100, Fax: +44-1628-585-900

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.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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 1.1