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

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

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

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

Table 2.2 Verified Operating Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>RX62N Group microcontrollers (Program ROM: 512 KB, RAM: 96 KB)</td>
</tr>
<tr>
<td>Memory</td>
<td>Micron Technology M25P Series Serial Flash Memory: 64 Mbits</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>ICLK: 96 MHz, PCLK: 48 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3 V</td>
</tr>
<tr>
<td>Integrated development environment</td>
<td>Renesas Electronics Corporation</td>
</tr>
<tr>
<td></td>
<td>High-performance embedded Workshop Version 4.09.00.007</td>
</tr>
<tr>
<td>C compiler</td>
<td>Renesas Electronics Corporation</td>
</tr>
<tr>
<td></td>
<td>RX Family C/C++ Compiler Package (Toolchain 1.0.1.0)</td>
</tr>
<tr>
<td></td>
<td>Compiler options</td>
</tr>
<tr>
<td></td>
<td>The integrated development environment default settings* are used.</td>
</tr>
<tr>
<td></td>
<td>Note: * Optimization level: 2, optimization method: Size priority</td>
</tr>
<tr>
<td>Endian order</td>
<td>Big endian / Little endian</td>
</tr>
<tr>
<td>Sample code version number</td>
<td>Ver. 2.04</td>
</tr>
<tr>
<td>Software</td>
<td>Micron Technology M25P Series Serial Flash Memory Control Software</td>
</tr>
<tr>
<td></td>
<td>(R01AN0566EJ), Version 2.01</td>
</tr>
<tr>
<td>Board</td>
<td>Renesas Starter Kit for RX62N</td>
</tr>
</tbody>
</table>
Table 2.3  Verified Operating Conditions

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

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](image)

5.2 List of Pins

Table 5.1 lists the pins used and their functions.

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

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.

Master transmission mode

Transmit data buffer in RAM (bytes shown)

| 0 | 1 | ... | 508 | 509 | 510 | 511 |

Data transmission order

Write to the slave device (bytes shown)

| 0 | 1 | ... | 508 | 509 | 510 | 511 |

Data reception order

Master reception mode

Read from the slave device (bytes shown)

| 0 | 1 | ... | 508 | 509 | 510 | 511 |

Data transmission order

Data buffer in RAM (bytes shown)

| 0 | 1 | ... | 508 | 509 | 510 | 511 |

Write to receive data buffer

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>
<thead>
<tr>
<th>Memory Used</th>
<th>Size</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>1,432 bytes (little endian)</td>
<td>R_SIO_rspi_rx.c</td>
</tr>
<tr>
<td>RAM</td>
<td>0 byte (little endian)</td>
<td>R_SIO_rspi_rx.c</td>
</tr>
<tr>
<td>Maximum user stack usage</td>
<td>204 bytes</td>
<td></td>
</tr>
<tr>
<td>Maximum interrupt stack usage</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_OK</td>
<td>(error_t)(0)</td>
<td>Successful operation</td>
</tr>
<tr>
<td>SIO_ERR_PARAM</td>
<td>(error_t)(-1)</td>
<td>Parameter error</td>
</tr>
<tr>
<td>SIO_ERR_HARD</td>
<td>(error_t)(-2)</td>
<td>Hardware error</td>
</tr>
<tr>
<td>SIO_ERR_OTHER</td>
<td>(error_t)(-7)</td>
<td>Other error</td>
</tr>
</tbody>
</table>

6.5.2 Definitions

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

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_LOG_ERR</td>
<td>(uint8_t)0x01</td>
<td>Log type: Error</td>
</tr>
<tr>
<td>SIO_TRUE</td>
<td>(uint8_t)0x01</td>
<td>Flag &quot;ON&quot;</td>
</tr>
<tr>
<td>SIO_FALSE</td>
<td>(uint8_t)0x00</td>
<td>Flag &quot;OFF&quot;</td>
</tr>
<tr>
<td>SIO_HI</td>
<td>(uint8_t)0x01</td>
<td>Port &quot;H&quot;</td>
</tr>
<tr>
<td>SIO_LOW</td>
<td>(uint8_t)0x00</td>
<td>Port &quot;L&quot;</td>
</tr>
<tr>
<td>SIO_OUT</td>
<td>(uint8_t)0x01</td>
<td>Port output setting</td>
</tr>
<tr>
<td>SIO_IN</td>
<td>(uint8_t)0x00</td>
<td>Port input setting</td>
</tr>
<tr>
<td>SIO_TX_WAIT</td>
<td>(uint16_t)50000</td>
<td>SIO transmission completion waiting time</td>
</tr>
<tr>
<td>SIO_RX_WAIT</td>
<td>(uint16_t)50000</td>
<td>SIO reception completion waiting time</td>
</tr>
<tr>
<td>SIO_DMA_TX_WAIT</td>
<td>(uint16_t)50000</td>
<td>DMA transmission completion waiting time</td>
</tr>
<tr>
<td>SIO_DMA_RX_WAIT</td>
<td>(uint16_t)50000</td>
<td>DMA reception completion waiting time</td>
</tr>
<tr>
<td>SIO_T_SIO_WAIT</td>
<td>(uint16_t)MTL_T_1US</td>
<td>SIO transmission &amp; reception completion waiting polling time</td>
</tr>
<tr>
<td>SIO_T_DMA_WAIT</td>
<td>(uint16_t)MTL_T_1US</td>
<td>DMA transmission &amp; reception completion waiting polling time</td>
</tr>
<tr>
<td>SIO_T_BRR_WAIT</td>
<td>(uint16_t)MTL_T_10US</td>
<td>BRR setting wait time</td>
</tr>
</tbody>
</table>

6.5.3 Other Definitions

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

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_TRAN_SIZE</td>
<td>(uint8_t)0x04</td>
<td>4 bytes (This value may not be changed.)</td>
</tr>
</tbody>
</table>
6.6 Structures and Unions

The structures used in the sample code are shown below.

```c
/* 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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_SIO_Init_Driver()</td>
<td>Driver initialization</td>
</tr>
<tr>
<td>R_SIO_Disable()</td>
<td>Disables serial I/O</td>
</tr>
<tr>
<td>R_SIO_Enable()</td>
<td>Enables serial I/O</td>
</tr>
<tr>
<td>R_SIO_Open_Port()</td>
<td>Releases serial I/O</td>
</tr>
<tr>
<td>R_SIO_Tx_Data()</td>
<td>Transmits serial I/O</td>
</tr>
<tr>
<td>R_SIO_Rx_Data()</td>
<td>Receives serial I/O</td>
</tr>
</tbody>
</table>

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.

![State Transition Diagram](image)

**Figure 6.4 State Transition Diagram**
6.9 Function Specifications

6.9.1 Driver Initialization

**R_SIO_Init_Driver**

<table>
<thead>
<tr>
<th>Synopsis</th>
<th>Driver initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>R_SIO.h, R_SIO_rspi.h, mtl_com.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>error_t R_SIO_Init_Driver(void)</td>
</tr>
</tbody>
</table>
| 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

![Diagram](image-url)
### 6.9.2 Serial I/O Disable Setup Processing

#### R_SIO_Disable

<table>
<thead>
<tr>
<th>Synopsis</th>
<th>Disable serial I/O processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>R_SIO.h, R_SIO_rspi.h, mtl_com.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>error_t R_SIO_Disable(void)</td>
</tr>
<tr>
<td>Explanation</td>
<td>- Enables the serial I/O function and sets the pins to their port function.</td>
</tr>
<tr>
<td></td>
<td>- Sets the pins used for serial I/O to their port function.</td>
</tr>
<tr>
<td></td>
<td>- Sets the slave device select control signal to the high level before calling this function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>SIO_OK ; Successful operation</td>
</tr>
<tr>
<td>Remarks</td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- If not used, this function can be called to disable the serial I/O function.</td>
</tr>
</tbody>
</table>

#### Diagram

```
Start
  `-- Disable serial I/O function
        `-- SIO_DISABLE()
            `-- Initialize ports
                `-- SIO_IO_INIT()

End
```

Cancels the module stop state, sets back to the serial I/O function, clears the OVRF, PERF, and MODF flags, sets the pins used to the port function, and enables the module stop state.

Disables the RSPI functions (to set up a state where the port pin functions can be used)
The DataIn pin is set to port input, DataOut to port high level output, and CLK to port high level output.

![Figure 6.6 Serial I/O Disable Setup Processing Outline](image-url)
6.9.3 Serial I/O Enable Setup Processing

**R_SIO_Enable**

<table>
<thead>
<tr>
<th>Synopsis</th>
<th>Enable serial I/O processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>R_SIO.h, R_SIO_rspi.h, mtl_com.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>error_t R_SIO_Enable(uint8_t BrgData)</td>
</tr>
</tbody>
</table>
| 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. |

```
Start

Initialize ports
SIO_IO_INIT()

Enable serial I/O function
SIO_ENABLE(BrgData)

Wait in software (10 µs)
mtl_wait_LP()

End

Disables the RSPI functions (to set up a state where the port pin functions can be used)  
The DataIn pin is set to port input, DataOut to port high level output, and CLK to port high level output.

Cancels the module stop state, enables serial I/O, and sets the bit rate.

Waits the bit rate setting time.
```

**Figure 6.7 Serial I/O Enable Setup Processing Outline**
6.9.4 Serial I/O Open Setup Processing

R_SIO_Open_Port

<table>
<thead>
<tr>
<th>Synopsis</th>
<th>Serial I/O port (DataOut, DataIn, and CLK) open processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>R_SIO.h, R_SIO_rspi.h, mtl_com.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>error_t R_SIO_Open_Port(void)</td>
</tr>
<tr>
<td>Explanation</td>
<td>• Sets the pins used for serial I/O to open (the input state).</td>
</tr>
<tr>
<td></td>
<td>• Set the slave device select control signal to the high level before calling this function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>SIO_OK ; Successful operation</td>
</tr>
<tr>
<td>Remarks</td>
<td>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.</td>
</tr>
</tbody>
</table>

![Diagram](image)

**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**
```c
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.
Enables interrupts, clears the IR flags, sets the port registers (RSPI setup), and enables RSPI functions.

Sets the transmit data size to 32 bits (sets SPB).

Reorders the data according to the endian settings.

Sets the transmit size (to 4) according to the amount of data not yet transmitted.

Sets the count to 50,000.

Waits for the transmit buffer empty state by polling the transmit buffer empty IR flag.

IRxx ← 0b

Write transmit data

Write the transmit data to SPDR.

TxCnt - = TxSize

pData + = TxSize

Terminates data transmission and then changes the transmit data size.

Start

Enable serial I/O transmission
SIO_TX_ENABLE()

Set transmit data size
SIO_DATASIZE_SET()

Transmit size ≥ 4?

If less than 4 bytes

Set the per operation transmit size

Reorder data
r_sio_tx_exchg()

If wait for transmit buffer empty not done, decrement timeout count
r_sio_wait()

Clear transmit buffer empty IR flag

Write transmit data

Decrement transmit byte count
Update data storage pointer

TxCnt != 0

Transmit complete?

TxCnt = 0

If wait for transmit buffer empty not done, decrement timeout count
r_sio_wait()

If wait for transmit complete not done, decrement timeout count.
r_sio_wait()

Clear transmit buffer empty IR flag

Disable serial I/O transmission
SIO_TX_DISABLE()

End

Figure 6.9 Serial I/O Data Transmission Processing Outline — 1
Sets the transmit size (Sets a value in the range 1 to 3.) according to the amount of data not yet transmitted.

Reorders the data according to the endian settings.

Note: 1. Since the condition for changing the data size is that the transmit buffer be in the empty state (the state where the data for the next transmission has not yet been stored), data transmission is terminated temporarily.

Sets the count to 50,000.

Waits for the transmit buffer empty state by polling the transmit buffer empty IR flag.

Sets the count to 50,000.

Waits for data transmit complete by polling the RSPI IDLE IR flag.

Changes the transmit data size using SPCMDx.SPB

---

**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.
Normal Reception (for operating modes SIO_OPTION_1 to SIO_OPTION_3)

Enables interrupts, clears the IR flags, sets the port registers (RSPI setup), and enables RSPI functions.

When the receive data length is 4 bytes or less

When the receive data length 5 bytes or more

Changes the receive data size to 4 bytes

Writes dummy data to SPDR

Sets the count to 50,000

Waits for the receive buffer full state by polling the receive buffer full IR flag.

IRxx ← 0b

Read receive data

Acquires data from SPDR

Receive size > 4?

When the receive data length 5 bytes or more

Transmit dummy data

Decrement receive byte count

Set timeout count. If wait for receive buffer full not done, decrement timeout count. r_sio_wait()

Clear receive buffer full IR flag

Read receive data

When the receive data length is 4 bytes or less

Transmit dummy data

Reorder receive data r_sio_rx_exchg()

Decrement receive byte count

Update data storage pointer

Set timeout count. If wait for receive buffer full not done, decrement timeout count. r_sio_wait()

Clear receive buffer full IR flag

Read receive data

Reorder receive data r_sio_rx_exchg()

Update data storage pointer

RxCnt - = SIO_TRAN_SIZE

pData + = SIO_TRAN_SIZE

Sets the count to 50,000

Waits for the receive buffer full state by polling the receive buffer full IR flag.

IRxx ← 0b

Read receive data

Acquires data from SPDR

A

Figure 6.11 Serial I/O Data Reception Processing Outline — 1 (Normal Reception)
Changes the receive data size to 4 bytes or less

Write dummy data to SPDR

Sets the count to 50,000

Waits for the receive buffer full state by polling the receive buffer full IR flag.

IRxx ← 0b

Acquires data from SPDR and reorders data to take endian orders into account.

Disables the RSPI functions, disables interrupts with the port function register (sets up the ports).

Figure 6.12   Serial I/O Data Reception Processing Outline — 2 (Normal Reception)
Enables interrupts, clears the IR flags, sets the port registers (RSPI setup), and enables RSPI functions.

When the receive data length is 4 bytes or less

Receive size > 4?

When the receive data length 5 bytes or more

Change the receive data size
SIO_DATASIZE_SET()

Clear transmit buffer empty IR flag
IRxx ← 0b

Transmit dummy data
SIO_TRX_ENABLE()

Decrement receive byte count
RxCnt - = SIO_TRAN_SIZE

Receive size > 4?

When the remaining receive data length is 4 bytes or less

If wait for transmit buffer empty not done, decrement timeout count.
r_sio_wait()

Sets the count to 50,000

Waits for the transmit buffer empty state by polling the transmit buffer empty IR flag.

When the remaining receive data length 5 bytes or more

Store current interrupt level in buffer

Disables interrupts

Transmit dummy data

Decrement receive byte count

Set timeout count. If wait for receive buffer full not done, decrement timeout count.
r_sio_wait()

Clear receive buffer full IR flag
IRxx ← 0b

Get receive data and reorder data
r_sio_rx_exchg()

Return interrupt mask level to original value

Check for overrun errors
SIO_ORER == 1?
Update data storage pointer
pData + = SIO_TRAN_SIZE

Receive size > 4?

When the receive data length is 4 bytes or less

A

B

Interrupts disabled duration

Figure 6.13  Serial I/O Data Reception Processing Outline — 3 (High-Speed Reception)
Changes the receive data size to 4 bytes or less

Transmit dummy data

Sets timeout count. If wait for receive buffer full not done, decrement timeout count.

Clear transmit buffer empty IR flag

Clear receive buffer full IR flag

Read and reorder receive data

Disable serial I/O transmission and reception

End

Figure 6.14 Serial I/O Data Reception Processing Outline — 4 (High-Speed Reception)
Sets the count to 50,000
Waits for the receive buffer full state by polling the receive buffer full IR flag.

IRxx ← 0b

Acquires data from SPDR and reorders data to take endian orders into account.

SIO_DATASIZE_SET(RxCnt)

Changes the receive data size to 4 bytes or less

Writes dummy data to SPDR

pData + = SIO_TRAN_SIZE

Acquires data from SPDR and reorders data to take endian orders into account.

Disables the RSPI functions, disables interrupts with the port function register (sets up the ports).

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. Enables 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, read receive buffer, longword access.
   - SPCKD ← 00h: SPCKD delay value setting (initial value)*1
   - 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)
   - 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.
   - 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 SPX.
   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.

```c
/* 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.

```c
/* 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.

```c
/* 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_endi.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.

```c
#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.

```c
/*-------------------------------------------------------------------------- */
/*     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

<table>
<thead>
<tr>
<th>#define Definition</th>
<th>S/I/O (RSPI)</th>
<th>CRC Calculation (using microcontroller hardware)</th>
<th>CRC Calculation (using software)</th>
<th>Receive Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_OPTION_1</td>
<td>○</td>
<td>―</td>
<td>―</td>
<td>Normal receive mode</td>
</tr>
<tr>
<td>SIO_OPTION_2</td>
<td>○</td>
<td>○</td>
<td>―</td>
<td>Normal receive mode</td>
</tr>
<tr>
<td>SIO_OPTION_3</td>
<td>○</td>
<td>―</td>
<td>○</td>
<td>Normal receive mode</td>
</tr>
<tr>
<td>SIO_OPTION_4</td>
<td>○</td>
<td>―</td>
<td>―</td>
<td>High-speed receive mode</td>
</tr>
<tr>
<td>SIO_OPTION_5</td>
<td>○</td>
<td>○</td>
<td>―</td>
<td>High-speed receive mode</td>
</tr>
<tr>
<td>SIO_OPTION_6</td>
<td>○</td>
<td>―</td>
<td>○</td>
<td>High-speed receive mode</td>
</tr>
</tbody>
</table>
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.

```c
/*---------------------------------------------*/
/* 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.

```c
/*---------------------------------------------*/
/* 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.

```c
/* 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>
<thead>
<tr>
<th>#define Definition</th>
<th>Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_DATAI_PORTNO</td>
<td>DataIn pin port number</td>
</tr>
<tr>
<td>SIO_DATAI_BITNO</td>
<td>DataIn pin bit number</td>
</tr>
<tr>
<td>SIO_CLK_PORTNO</td>
<td>CLK pin port number</td>
</tr>
<tr>
<td>SIO_CLK_BITNO</td>
<td>CLK pin bit number</td>
</tr>
<tr>
<td>SIO_DATAO_PORTNO</td>
<td>DataOut pin port number</td>
</tr>
<tr>
<td>SIO_DATAO_BITNO</td>
<td>DataOut pin bit number</td>
</tr>
<tr>
<td>SIO_PFC_SELECT</td>
<td>Port function register setting &quot;x&quot;: PFxSPI, where x = G or H.</td>
</tr>
</tbody>
</table>

(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.

```c
#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.

```c
#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.

```c
#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.

```c
#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.

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

```c
#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

All trademarks and registered trademarks are the property of their respective owners.
## Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Oct.19.11</td>
<td>First edition issued</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>&quot;Specification other than optimization option 0&quot; for the C compiler removed from table 2.1.</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Sample code version number changed from 2.00 to 2.02 in table 2.1.</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Software used for evaluation in table 2.1: Corrected.</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.3 Memory Requirements: Corrected.</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.4 File Structure: Application note number corrected.</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.5.3 Other Definitions: Header file deleted.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.1 Driver Initialization: Remarks corrected.</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.2 Serial I/O Disable Setting: Remarks corrected and figure 6.5 corrected.</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.3 Serial I/O Enable Setting: Remarks corrected and figure 6.6 corrected.</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.5 Serial I/O Data Transmission Processing: Remarks corrected.</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Figure 6.8 corrected.</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Figure 6.9 corrected.</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.6 Serial I/O Data Reception Processing: Remarks corrected.</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Figure 6.10 corrected. &quot;Transmit data size modification&quot; changed to &quot;Receive data size modification&quot; and &quot;Receive data reordering&quot; changed to &quot;Receive data reordering and acquisition&quot;.</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Figure 6.11 corrected. &quot;Transmit data size modification&quot; changed to &quot;Receive data size modification&quot;, &quot;Transmit buffer empty&quot; changed to &quot;Transmit buffer full&quot;, and &quot;Receive data reordering&quot; changed to &quot;Receive data reordering and acquisition&quot;.</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>Figure 6.12 corrected. &quot;Receive data reordering&quot; changed to &quot;Receive data reordering and acquisition&quot;.</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.7 Transmit Data Endian Reordering: &quot;static&quot; removed from function declarations.</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.8.8 Receive Data Endian Reordering: &quot;static&quot; removed from function declarations.</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.9.3 Macro Function SIO_DATAI_INIT(): &quot;Disable input buffer function&quot; corrected to &quot;Enable input buffer function&quot; in (2) Function (1).</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.9.12 Macro Function SIO_TX_DISABLE(): &quot;Reads SPCR&quot; added to (2) Function (2).</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.9.14 Macro Function SIO_TRX_ENABLE(): &quot;Clears IR flag&quot; added to (2) Function (2).</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.9.14 Macro Function SIO_TRX_EXABLE(): &quot;SIO_TX_DISABLE()&quot; corrected to &quot; SIO_TRX_DISABLE()&quot; in (3) Remarks.</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>6.9.14 Macro Function SIO_TRX_DISABLE(): &quot;Reads SPCR&quot; added to (2) Function (2).</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>7.1 mtl_com.h: (1) OS header file definitions changed.</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>7.1.2 mtl_tim.h: Timer counter modified.</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>7.2.1 R_SIO.h added</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td>Dec.14.11</td>
<td>8.4 Compiler Options: Deleted</td>
<td>46</td>
<td>The following items were moved up due to this deletion.</td>
</tr>
<tr>
<td>Rev.</td>
<td>Date</td>
<td>Page</td>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1.02</td>
<td>Dec.19.11</td>
<td>3</td>
<td>Added default settings group in table 2.1 compiler options.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Changed the table 2.1 sample code version from 2.02 to 2.03.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>6.4 File Structure: Application note number corrected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46</td>
<td>Added section 8.6, Compiler Options</td>
<td></td>
</tr>
<tr>
<td>1.03</td>
<td>Mar.31.12</td>
<td>2</td>
<td>1. Specifications: Content modified.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Changed the table 2.1 sample code version from 2.03 to 2.04.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Added tables 2.2 and 2.3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Interchanged sections 5.1, Hardware Structure and 5.2 Pins</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>6.1 Operation Overview: Moved part of the content to section 6.2.2, Relationship Between Data Buffers and Transmit/Receive Data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>6.2.1 Software Structure: Added new content</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Added section 6.2.2 Relationship Between Data Buffers and Transmit/Receive Data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Deleted sections 6.2.3 to 6.2.6.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>6.3 Memory Requirements: Added content and updated memory sizes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>6.4 File Structure: Application note number corrected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>6.5.1 Return Values: Original Japanese used a different spelling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Table 6.4 Definition Values: Changed technical terms used in column headers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Table 6.6 Functions: Deleted R_SIO_Tx_Exchg() and R_SIO_Rx_Exchg().</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>6.8 State Transition Diagram: Moved from 6.10 to 6.8.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>6.9.1 Driver Initialization: Modified the notes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>6.9.2 Disable Serial I/O Processing: Modified notes and processing overview.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>17</td>
<td>6.9.3 Enable Serial I/O Processing: Modified notes and processing overview.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>18</td>
<td>6.9.4 Serial I/O Open Processing: Modified notes and processing overview.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 to 21</td>
<td>6.9.5 Transmit Serial I/O Data: Modified notes and processing overview.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>22 to 27</td>
<td>6.9.6 Receive Serial I/O Data: Added &quot;Normal receive mode&quot; and &quot;High-speed receive mode&quot; to the description column. Modified notes and processing overview.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>27</td>
<td>Deleted 6.9.7, Transmit Data Endian Reordering Processing, and 6.8.8, Receive Data Endian Reordering Processing.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>28 to 37</td>
<td>6.10 Inline Function Specifications: Originally called &quot;Macro Functions&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
<td>6.10.1 SIO_IO_INIT(): Modified (2) Function.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>29</td>
<td>6.10.2 SIO_IO_OPEN(): Modified (2) Function.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>29</td>
<td>6.10.3 SIO_DATAI_INIT(): Modified (2) Function and (3) Remarks.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>6.10.4 SIO_DATAO_INIT(): Modified (2) Function and (3) Remarks.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>6.10.5 SIO_DATAO_OPEN() (2) : Modified (2) Function.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>31</td>
<td>6.10.6 SIO_CLK_INIT() (2) : Modified (2) Function and (3) Remarks.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>31</td>
<td>6.10.7 SIO_CLK_OPEN() (2) : Modified (2) Function.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>6.10.8 SIO_ENABLE() (2) : Modified (2) Function and (3) Remarks.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>33</td>
<td>6.10.9 SIO_DISABLE() (2) : Modified (2) Function.</td>
<td></td>
</tr>
<tr>
<td>Rev.</td>
<td>Date</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.03</td>
<td>Mar.31.12</td>
<td>34 6.10.11 SIO_TX_ENABLE(): Modified (2) Function and (3) Remarks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>35 6.10.12 SIO_TX_DISABLE() (2) : Modified (2) Function.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>36 6.10.13 SIO_TRX_ENABLE(): Modified (2) Function and (3) Remarks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>37 6.10.14 SIO_TRX_DISABLE() (2) : Modified (2) Function.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>42 7.2.1 R_SIO.h (1): Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>42 to 43 7.2.2 (1) Operating Mode Definitions: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>43 7.2.2 (2) CRC Calculation Type Definition: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>43 7.2.2 (3) Used RSPI Channel Definition: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>44 7.2.2 (4) Used Pin Definitions: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>44 7.2.2 (5) RSPIS Bit Definition for the Used Port Function Register (PFxSPI): Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>45 7.2.2 (7) Software Timer Definition: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>45 Added section 7.2.2 (8), Pull-up Resistor Control Register (PCR) Definitions. Deleted the /* SET */ comments to conform to the program.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>45 Added section 7.2.2 (9), Open-drain Control Register (ODR) Definitions. Deleted the /* SET */ comments to conform to the program.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 Added section 8.7, When Other Applications Use DMAC, EXDMAC, or DTC Transfers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.04</td>
<td>Nov.13.12</td>
<td>1 Overview: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 6.2.1 Software Overview: Content modified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 6.4 File Structure: Corrected application note numbers.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.