RL78/G10
Serial Array Unit (CSI Master Communication)

Introduction
This application note describes how the serial array unit (SAU) performs communication tasks using the CSI master communication feature. As CSI applications, the SAU selects one of two slaves with the CS signal which is issued through a port and performs single transmission/reception, continuous transmission, continuous reception, and continuous transmission/reception operations. To ensure reliable communication, it adopts a simple protocol and a command set plus its compatible format. Since RL78/G10 units running in CSI slave mode are used as slave devices, it performs handshake processing using the BUSY signal.

Target Device
RL78/G10

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.
## Contents

1. Specifications ................................................................................................................. 4  
   1.1 Outline of CSI Communication .................................................................................. 4  
   1.2 Outline of Communication ....................................................................................... 5  
   1.3 Communication Format ............................................................................................. 8  
   1.4 Communication Protocol (Hardware Handshake) ....................................................... 8  

2. Operation Check Conditions ......................................................................................... 9  

3. Related Application Notes .......................................................................................... 9  

4. Description of the Hardware ....................................................................................... 10  
   4.1 Hardware Configuration Example ........................................................................... 10  
   4.2 List of Pins to be Used ............................................................................................ 11  

5. Description of the Software ....................................................................................... 12  
   5.1 Operation Outline ..................................................................................................... 12  
   5.2 List of Option Byte Settings .................................................................................... 14  
   5.3 List of Constants ...................................................................................................... 14  
   5.4 List of Variables ....................................................................................................... 16  
   5.5 List of Functions (Subroutines) .............................................................................. 17  
   5.6 Function (Subroutine) Specifications ..................................................................... 18  
   5.7 Flowcharts ............................................................................................................... 29  
      5.7.1 CPU Initialization Function .............................................................................. 30  
      5.7.2 I/O Port Setup .................................................................................................. 31  
      5.7.3 Clock Generation Circuit Setup ...................................................................... 32  
      5.7.4 SAU Setup ...................................................................................................... 33  
      5.7.5 Timer Array Unit Setup .................................................................................. 42  
      5.7.6 Main Processing ............................................................................................... 43  
      5.7.7 1-ms Interval Timer Startup Processing ......................................................... 46  
      5.7.8 1-ms Interval Wait Function ............................................................................. 48  
      5.7.9 Slave Selection Processing .............................................................................. 49  
      5.7.10 Slave Response Wait Processing .................................................................... 50  
      5.7.11 Slave Status Check Processing ...................................................................... 53  
      5.7.12 Continuous Data Transmission Processing .................................................. 56  
      5.7.13 Continuous Data Reception Processing ......................................................... 58  
      5.7.14 Continuous Data Transmission/Reception Processing .................................. 60  
      5.7.15 Data Update Processing .................................................................................. 62  
      5.7.16 Receive Data Check Processing ...................................................................... 63  
      5.7.17 Data Pointer Setup Processing ........................................................................ 64  
      5.7.18 1-character Transmission Start Processing .................................................... 65
5.7.19 1-character Transmission End Wait Processing ................................................................. 65
5.7.20 1-character Reception Processing ......................................................................................... 66
5.7.21 1-character Reception End Wait Processing ....................................................................... 66
5.7.22 1-character Transfer State Check Processing .................................................................... 67
5.7.23 Continuous Transmission Start Processing ........................................................................ 68
5.7.24 Continuous Reception Startup Processing .......................................................................... 70
5.7.25 Continuous Transmission/Reception Startup Processing .................................................... 71
5.7.26 Continuous Transfer End Wait Processing .......................................................................... 72
5.7.27 Transfer End Interrupt Setup Processing ............................................................................ 73
5.7.28 Buffer Empty Interrupt Setup Processing .......................................................................... 74
5.7.29 Transmission Mode Setup Processing ................................................................................. 75
5.7.30 Reception Mode Setup Processing ....................................................................................... 75
5.7.31 Transmission/Reception Mode Setup Processing ................................................................ 75
5.7.32 Common Processing for Mode Setup .................................................................................. 76
5.7.33 CSIp Communication Enable Processing ............................................................................ 79
5.7.34 CSIp Communication Termination Processing ..................................................................... 81
5.7.35 CSIp Interrupt Startup Processing ..................................................................................... 82
5.7.36 1-character Transfer End Interrupt Processing .................................................................... 83
5.7.37 1-character Transfer End Interrupt Processing in Continuous Reception Mode .................. 83
5.7.38 Buffer Empty Interrupt Processing in Continuous Transmission Mode ................................. 84
5.7.39 Transmission End Interrupt Processing in Continuous Transmission Mode ......................... 84
5.7.40 Buffer Empty Interrupt Processing in Continuous Transmission Mode ................................. 85
5.7.41 Transfer End Interrupt Processing in Continuous Transmission/Reception Mode ................ 85

6. Changing the Channel to be Used .............................................................................................. 86
6.1 Definition File ........................................................................................................................... 86
6.2 Major Items of the Definition File ............................................................................................ 86
6.3 Changing the Transfer Rate ...................................................................................................... 86
6.4 Changing the Microcontroller to be Used ............................................................................... 86
6.5 Changing the Channel to be Used ........................................................................................... 87
6.6 Reference .................................................................................................................................. 88

7. Sample Code .................................................................................................................................. 89

8. Documents for Reference ............................................................................................................ 89
1. Specifications

The serial array unit (SAU) described in this application note performs CSI master communication using the serial array unit (SAU). The SAU outputs SPI CS signals through ports to a maximum of 2 slaves that are attached to select the target of communication and performs single transmission/reception, continuous transmission, continuous reception, or continuous transmission/reception while performing handshaking using the BUSY signal. (Although CS is a negative logic signal, the bar that should normally appear over the signal name is omitted in this document.)

1.1 Outline of CSI Communication

CSI is a protocol for clock synchronous serial communication using three signal lines, namely, serial clock (SCK), serial input data (SI), and serial output data (SO). SPI (Serial Peripheral Interface) uses an additional signal, CS (Chip Select), which is used to select the slave device. The relationship among these signals is shown in figure 1.1.

![CSI signals diagram]

- **SCK signal**: Clock signal for indicating the communication timing, output by the master and input to the slaves.
- **SO signal**: Serial output data signal. Connected to the SI signal pin of the target device.
- **SI signal**: Serial input data signal. Connected to the SO signal pin of the target device.
- **CS signal**: Used by the master to select the target slave device.
- **BUSY signal**: Handshake signal that is introduced by this application note.

**Figure 1.1 Outline of CSI Communication**

The CSI communication master first selects the slave with which it wants to communicate with the CS signal (this is an SPI operation). The master outputs the SCK signals and place data on the SO signal line and inputs data from the SI signal line in synchronization with the SCK signals. In CSI communication, the slave needs to become ready for communication by the time the master starts communication (sending the SCK signals). In this application note, the BUSY signal is introduced as the signal for indicating the slave (RL78/G10 in the example in this application note) is ready for communication. The master verifies this BUSY signal before initiating a communication session.
1.2 Outline of Communication

Communication is performed in 1-ms slot units. In each slot, command transmission from the master and communication processing according to the command are processed. Figure 1.2 shows the outline of slot processing and table 1.1 lists the commands to be used.

![Figure 1.2 Outline of Slots](image)

<table>
<thead>
<tr>
<th>Slot</th>
<th>Command</th>
<th>Communication corresponding to the command</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>1 ms</td>
<td>1 ms</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1 Commands to be Used

<table>
<thead>
<tr>
<th>Command</th>
<th>Outline of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status check</td>
<td>Checks the number of data characters that the slave can transmit or receive.</td>
</tr>
<tr>
<td>Receive</td>
<td>Receives data from the slave in continuous mode.</td>
</tr>
<tr>
<td>Transmit</td>
<td>Transmits data to the slave in continuous mode.</td>
</tr>
<tr>
<td>Transmit/receive</td>
<td>Transmits and receives data to and from the slave in continuous mode.</td>
</tr>
</tbody>
</table>

The slave is designed to transmit the complement of the data it received in the next communication operation, so that the master can determine whether the data received by the slave is correct. The master prepares increment pattern data, e.g., 00, 01, 02, ..., as transmit data and updates the transmit data on each transmission operation.

The CSI channel to be used can be changed easily by editing a header file (however, the CSI channel can only be changed in 16-pin products).

Table 1.2 lists the peripheral functions that are used and their uses. Figures 1.3 to 1.6 show the CSI communication operations. Unless specifically noted, CSIp is represented by CSI00.

Table 1.2 Peripheral Functions to Be Used and Their Uses

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial array unit m</td>
<td>Performs CSI master communication using the SCKp signal (clock output), SIp signal (receive data), and SOp signal (transmit data). p: 00/01 Note</td>
</tr>
<tr>
<td>Port</td>
<td>P03 (CS1 signal output), P04 (CS2 signal output), P137 (BUSY signal input)</td>
</tr>
</tbody>
</table>

Note: 16-pin products only
Figure 1.3  Timing Chart of Status Check Command

Figure 1.4  Timing Chart of Receive Command
Selection of slave (start of slot)

CS1/2 signal

BUSY signal

SCKp signal

Slp signal

SOp signal

TSFmn

INTCSIp

SIOp write

Figure 1.5  Timing Chart of Transmit Command

Selection of slave (start of slot)

CS1/2 signal

BUSY signal

SCKp signal

Slp signal

SOp signal

TSFmn

INTCSIp

SIOp write

Figure 1.6  Timing Chart of Transmit/Receive Command
1.3 Communication Format

The characteristics of the CSI communication format that is used by the sample code are listed in table 1.3.

Table 1.3 Communication Format

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication speed</td>
<td>1 Mbps</td>
<td>About 200 kbps at minimum</td>
</tr>
<tr>
<td>Data bit length</td>
<td>8 bits/character</td>
<td></td>
</tr>
<tr>
<td>Transfer order</td>
<td>MSB first</td>
<td></td>
</tr>
<tr>
<td>Communication type</td>
<td>Type 1</td>
<td></td>
</tr>
<tr>
<td>Communication mode</td>
<td>Single transfer/continuous transfer</td>
<td>Continuous mode is used for data transfer.</td>
</tr>
<tr>
<td>Communication direction</td>
<td>Receive/transmit/transmit and receive</td>
<td></td>
</tr>
<tr>
<td>Maximum number of characters transferred</td>
<td>63 characters/slot</td>
<td>8 characters by default</td>
</tr>
</tbody>
</table>

1.4 Communication Protocol (Hardware Handshake)

The communication target is set to the RL78/G10 running in CSI slave mode and handshaking using the BUSY signal is adopted to have the setup time that is required for the communication operation on the slave side.

The BUSY signal is used to verify that the slave becomes ready for communication when selecting it with the CS signal or when sending a command. A timeout time of 10 µs is set up so that the master does not enter an unnecessary deadlock state when no slave is connected. If no response is returned from the slave within this period, the master terminates processing immediately, considering that the slave is in the busy state in which it is taking some action or that there is no slave available.

Figure 1.7 shows an example of handshaking processing for the status check command. To select the slave, the master waits until the BUSY signal goes low while measuring the time so as to detect a timeout condition after the falling edge of the CS signal. When the BUSY signal goes low before a timeout, the master sends the command. Upon completion of the command transmission, the master waits until the BUSY signal goes low again to start status receive processing. In this way, the master performs handshaking to get synchronized with the slave by checking the BUSY signal before initiating a new communication operation.

Figure 1.7 Handshaking Example

The BUSY signal is not available for dedicated SPI slave devices such as EEPROM, A/D, and D/A. This is because these devices are always ready for communication. A hardware measure to be taken when connecting these dedicated slave devices is to connect the BUSY signal input to VSS. Timeout checking is accomplished by a dedicated subroutine (SWAITRDY). As software countermeasures, it is possible to dispense with the checking for the BUSY signal by modifying the subroutine so that it simply returns after clearing the CY flag. Subsequently, perform communication according to the commands that are defined for the individual devices and their communication protocol.
2. Operation Check Conditions

The sample code contained in this application note has been checked under the conditions listed in the table below.

Table 2.1 Operation Check Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller used</td>
<td>RL78/G10 (R5F10Y16ASP)</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>• High-speed on-chip oscillator (HOCO) clock: 20 MHz</td>
</tr>
<tr>
<td></td>
<td>• CPU/peripheral hardware clock: 20 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>5.0 V (can run at a voltage range of 2.9 V to 5.5 V.)</td>
</tr>
<tr>
<td></td>
<td>SPOR detection voltage:</td>
</tr>
<tr>
<td></td>
<td>• When reset occurs: $V_{DD} &lt; 2.84$ V</td>
</tr>
<tr>
<td></td>
<td>• When reset is released: $V_{DD} \geq 2.90$ V</td>
</tr>
<tr>
<td>Integrated development environment</td>
<td>CubeSuite + E1.03.00k01_RL78_G10 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>(CubeSuite+)</td>
<td></td>
</tr>
<tr>
<td>Assembler (CubeSuite+)</td>
<td>RA78K0R V1.70 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>Integrated development environment</td>
<td>e2studio V2.0.0.16 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>(e2studio)</td>
<td></td>
</tr>
<tr>
<td>Assembler (e2studio)</td>
<td>KPIT GNURL78-ELF Toolchain V13.02 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>Board to be used</td>
<td>RL78/G10 target board (QB-R5F10Y16-TB)</td>
</tr>
</tbody>
</table>

3. Related Application Notes

The application notes that are related to this application note are listed below for reference.

- RL78/G10 Initialization (R01AN1454E) Application Note
- RL78/G10 Serial Array Unit (CSI Slave Communication) (R01AN1461E) Application Note
- RL78 Family CubeSuite+ Startup Guide (R01AN1232E) Application Note
4. Description of the Hardware

4.1 Hardware Configuration Example

Figure 4.1 shows an example of hardware configuration that is used for this application note.

Cautions:

1. The purpose of this circuit is only to provide the connection outline and the circuit is simplified accordingly. When designing and implementing an actual circuit, provide proper pin treatment and make sure that the hardware's electrical specifications are met (connect the input-only ports separately to VDD or VSS via a resistor).

2. VDD must be held at not lower than the reset release voltage (VSPOR) that is specified as SPOR.

3. In the example of wiring in the hardware configuration for the sample code from Renesas (R01AN1461), RL78/G10 devices are mounted as slaves 1 and 2. The sample code operates so that signal output is only by the slave selected by the CS signal (the non-selected pin is placed in the high impedance state). Accordingly, the P04 pins of slaves 1 and 2 are directly connectable (this also applies to the P00/SO00 pins on the slave side).
4.2 List of Pins to be Used

Table 4.1 lists the pins to be used and their function.

**Table 4.1 Pins to be Used and their Functions**

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P02/ANI1/SCK00/SCL00/PCLBUZ0/KR3</td>
<td>Output</td>
<td>Serial clock output pin</td>
</tr>
<tr>
<td>P01/ANI0/SI00/RXD0/SDA00/KR2</td>
<td>Input</td>
<td>Data receive pin</td>
</tr>
<tr>
<td>P00/SO00/TXD0/INTP1</td>
<td>Output</td>
<td>Data transmit pin</td>
</tr>
<tr>
<td>P137/TI00/INTP0</td>
<td>Input</td>
<td>BUSY signal input from the slave</td>
</tr>
<tr>
<td>P03/ANI2/TO00/KR4/(INTP1) (CS1)</td>
<td>Output</td>
<td>Slave 1 select signal</td>
</tr>
<tr>
<td>P04/ANI3/TI01/TO01/KR5 (CS2)</td>
<td>Output</td>
<td>Slave 2 select signal</td>
</tr>
</tbody>
</table>

Note: The channel to be used must be specified in an include file (DEV&CSI_CH.inc). The default value is CSI00. The pins and interrupt to be used are automatically set according to the channel to be used.
5. Description of the Software

5.1 Operation Outline

This sample code, after completion of initialization, selects a slave and performs communication operations such as status check, data transmission, data transmission/reception, and data reception in that order on the selected slave.

(1) Initialize the CSI.

<Conditions for setting the CSI>

- Use SAU0 channel 0 as CSI00 <Note>.
- Use CK00 as the transfer clock.
- Assign the clock output to the P02/SCK00 pin <Note>, the data input to the P01/SI00 pin <Note>, and the data output to the P00/SO00 pin <Note>.
- Set the data length to 8 bits.
- Set the phase between the data and clock to type 1.
- Set the order of data transfer mode to MSB first.
- Set the transfer rate to 1M bps.
- Use transmission end interrupt (INTCSI00) <Note>.
- Set the priority of the interrupt (INTCSI00) <Note> to the lowest (level 3 (default)).

Note: Two channels are available for use in 16-pin products. The channel to be used must be specified in an include file (DEV&CSI_CH.inc). The default value is CSI00. The pins and interrupt to be used are automatically set according to the channel to be used.

(2) Initialize the timer.

<Conditions for setting the timer>

- Run channels 1 and 2 as 8-bit interval timers.
- Set the operating clock frequency to 156 kHz which is derived by dividing fCLK by 128.
- Use the upper TM01H as a 1-ms interval timer.
- Use the lower TM01 as a 10-µs interval timer.

(3) After initialization is completed, the master initializes memory and performs communication with the slave according to the steps given below.

1) Waits in the HALT state for 1ms interval interrupts (INTTM01H).
2) When the master is released from the HALT state on an INTTM01H interrupt, it selects (by issuing the CS signal) the slave that is selected in the flag (RCSFLAG) and waits for a response from the slave.
3) Proceeds to step 4) when the BUSY signal from the slave goes low. When a timeout is detected, the master deselects the slave and proceed with step 10).
4) Transmits a status check command and receives the status from the slave. When a timeout is detected, the master deselects the slave and proceed with step 10).
5) When an INTTM01H occurs, the master transmits the number of data characters specified in step 4) and generates the next transmit data and the expected value of the receive data. If the value of received data does not match the expected value, the master enters the HALT mode and subsequent processing does not proceed. When a timeout is detected, the master deselects the slave and proceed with step 10).
6) When an INTTM01H occurs, the master transmits and receives the number of characters specified in step 4). It also checks whether the received data matches the complement of the data that is transmitted in step 5). When a timeout is detected, the master deselects the slave and proceed with step 10).
7) The master generates the next transmit data and the expected value of the receive data, then waits for an INTTM01H.
8) When an INTTM01H occurs, the master receives the number of data characters specified in step 4). When a timeout is detected, the master deselects the slave and proceed with step 10).
9) The master checks whether the received data matches the expected value.
10) The master changes the state of the flag (RCSFLAG) to switch the target slave. Subsequently, the master repeats the steps starting at 1).
(4) Commands
Each communication operation begins with the transmission of a 1-byte command. The command formats are listed in table 5.1. The master transmits a status check command and receives the response from the slave in the first slot of a communication sequence. The number of received data characters or the size of buffer, whichever is smaller, is used as the data count for the next communication processing. The master then performs the next communication using this data count.

Table 5.1 Command Formats

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Command Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status check</td>
<td>Checks the number of data characters that the slave can transmit or receive. The following responses can be made by the slave: 01xxxxxB: The number of characters that the slave can transmit is xxxxxB 00xxxxxB: The number of characters that the slave can receive is xxxxxB</td>
</tr>
<tr>
<td>Reception</td>
<td>01xxxxxB: The master receives xxxxxB bytes of data.</td>
</tr>
<tr>
<td>Transmission</td>
<td>10xxxxxB: The master transmits xxxxxB bytes of data.</td>
</tr>
<tr>
<td>Transmission/reception</td>
<td>11xxxxxB: Transmits and receives xxxxxB bytes of data.</td>
</tr>
</tbody>
</table>
5.2 List of Option Byte Settings

Table 5.2 summarizes the settings of the option bytes.

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000C0H</td>
<td>11101110B</td>
<td>Disables the watchdog timer. (Stops counting after the release from the reset state.)</td>
</tr>
<tr>
<td>000C1H</td>
<td>11110111B</td>
<td>SPOR detection voltage: When Reset occurs: VDD &lt; 2.84V When Reset is released: VDD &gt;= 2.90V</td>
</tr>
<tr>
<td>000C2H</td>
<td>11111001B</td>
<td>HS mode, HOCO: 20 MHz</td>
</tr>
<tr>
<td>000C3H</td>
<td>10000101B</td>
<td>Enables the on-chip debugger.</td>
</tr>
</tbody>
</table>

5.3 List of Constants

Tables 5.3 and 5.4 list the constants that are used in this sample program.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Defined in</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKFREQ</td>
<td>DEV&amp;CSI_CH.inc</td>
<td>20000</td>
<td>RL78/G10 operating clock frequency in kHz (20 MHz)</td>
</tr>
<tr>
<td>BAUDRATE</td>
<td>↑</td>
<td>1000</td>
<td>Communication speed in kbps (1 Mbps)</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>↑</td>
<td>CLKFREQ / BAUDRATE</td>
<td>Frequency division ratio necessary for attain the specified communication speed</td>
</tr>
<tr>
<td>SDRDATA</td>
<td>↑</td>
<td>(DIVIDE/2-1)*2</td>
<td>Value to be set in SDR to specify the communication speed</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>↑</td>
<td>1</td>
<td>Slot interval in ms units (1 ms)</td>
</tr>
<tr>
<td>TDRDATA</td>
<td>↑</td>
<td>(CLKFREQ/128) * INTERVAL - 1</td>
<td>Value to be set in TDR01H</td>
</tr>
<tr>
<td>SMR0nH</td>
<td>↑</td>
<td>SMR00H</td>
<td>Channel mode setting register (Higher bits)</td>
</tr>
<tr>
<td>SMR0nL</td>
<td>↑</td>
<td>SMR00L</td>
<td>Channel mode setting register (Lower bits)</td>
</tr>
<tr>
<td>SCR0nH</td>
<td>↑</td>
<td>SCR00H</td>
<td>Channel communication operation setting register (Higher bits)</td>
</tr>
<tr>
<td>SCR0nL</td>
<td>↑</td>
<td>SCR00L</td>
<td>Channel communication operation setting register (Lower bits)</td>
</tr>
<tr>
<td>SDR0nH</td>
<td>↑</td>
<td>SDR00H</td>
<td>Higher 8 bits of channel serial data</td>
</tr>
<tr>
<td>SIOp</td>
<td>↑</td>
<td>SIO00</td>
<td>Lower 8 bits of channel serial data</td>
</tr>
<tr>
<td>SSR0n</td>
<td>↑</td>
<td>SSR00</td>
<td>Channel status register</td>
</tr>
<tr>
<td>SIR0n</td>
<td>↑</td>
<td>SIR00</td>
<td>Channel flag clear trigger register</td>
</tr>
<tr>
<td>TRGONn</td>
<td>↑</td>
<td>00000001B</td>
<td>Value for SS0 and ST0</td>
</tr>
<tr>
<td>SOEON</td>
<td>↑</td>
<td>TRGONn</td>
<td>For setting in channel output enable register (enable)</td>
</tr>
<tr>
<td>SOEOFF</td>
<td>↑</td>
<td>11111110B</td>
<td>For setting in channel output enable register (disable)</td>
</tr>
<tr>
<td>SOHIGH</td>
<td>↑</td>
<td>TRGONn</td>
<td>For setting value in channel output register</td>
</tr>
<tr>
<td>PM_SCKp</td>
<td>↑</td>
<td>PM02</td>
<td>SCK signal port mode register</td>
</tr>
<tr>
<td>PM_SIp</td>
<td>↑</td>
<td>PM01</td>
<td>SI signal port mode register</td>
</tr>
<tr>
<td>PM_SOp</td>
<td>↑</td>
<td>PM02</td>
<td>SO signal port mode register</td>
</tr>
</tbody>
</table>
### Table 5.4  Constants for the Sample Program (2/2)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Defined In</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_SCKp</td>
<td>↑</td>
<td>P02</td>
<td>SCK signal port</td>
</tr>
<tr>
<td>P_SIp</td>
<td>↑</td>
<td>P01</td>
<td>SI signal port</td>
</tr>
<tr>
<td>P_SOp</td>
<td>↑</td>
<td>P00</td>
<td>SO signal port</td>
</tr>
<tr>
<td>CSIIFp</td>
<td>↑</td>
<td>CSIIF0</td>
<td>Channel interrupt request flag</td>
</tr>
<tr>
<td>CSIMKp</td>
<td>↑</td>
<td>CSIMK0</td>
<td>Channel interrupt master register</td>
</tr>
<tr>
<td>CRXMODE</td>
<td>↑</td>
<td>0100000000000111B</td>
<td>Value to be loaded in SCR register in receive mode</td>
</tr>
<tr>
<td>CTXMODE</td>
<td>↑</td>
<td>1000000000000111B</td>
<td>Value to be loaded in SCR register in transmit mode</td>
</tr>
<tr>
<td>CTRXMODE</td>
<td>↑</td>
<td>1100000000000111B</td>
<td>Value to be loaded in SCR register in transmit/receive mode</td>
</tr>
<tr>
<td>CSMRDATA</td>
<td>↑</td>
<td>000000000010000B</td>
<td>Initial value for SMR register</td>
</tr>
<tr>
<td>BUSYSIG</td>
<td>r_main. asm</td>
<td>P137</td>
<td>Port for checking the BUSY signal</td>
</tr>
<tr>
<td>CS1SIG</td>
<td>↑</td>
<td>P03</td>
<td>CS1 output port</td>
</tr>
<tr>
<td>CS2SIG</td>
<td>↑</td>
<td>P04</td>
<td>CS2 output port</td>
</tr>
<tr>
<td>CRXDTNO</td>
<td>↑</td>
<td>8</td>
<td>Size of receive data buffer (in bytes)</td>
</tr>
<tr>
<td>CTXDTNO</td>
<td>↑</td>
<td>8</td>
<td>Size of transmit data buffer (in bytes)</td>
</tr>
<tr>
<td>STSCHKCMD</td>
<td>↑</td>
<td>00000000B</td>
<td>Status check command</td>
</tr>
<tr>
<td>MSTRDCMD</td>
<td>↑</td>
<td>01000000B</td>
<td>Master receive command</td>
</tr>
<tr>
<td>MSTWT CMD</td>
<td>↑</td>
<td>10000000B</td>
<td>Master transmit command</td>
</tr>
<tr>
<td>MSTRW CMD</td>
<td>↑</td>
<td>11000000B</td>
<td>Transmit/receive command</td>
</tr>
<tr>
<td>SELOFFSIG</td>
<td>↑</td>
<td>00001100B</td>
<td>Data for terminating slave selection</td>
</tr>
</tbody>
</table>

*Note: Values vary when CSI01 is used.*
5.4 List of Variables

Table 5.5 lists the global variables that are used in this sample program.

**Table 5.5 Global Variables for the Sample Program**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>RCSISUBADDR</td>
<td>Address of the program that performs actual processing when an INTCSIp interrupt occurs.</td>
<td>main, STXDATAST, SRXDATAST, SSEQRXSUB, SSEQTXSUB, SSEQTRXSUB, INTCSIp, STXNEXT, STRXLAST</td>
</tr>
<tr>
<td>8-bit array</td>
<td>RSNDBUF1</td>
<td>Buffer for transmit data to slave 1</td>
<td>main, (SETTRXPNTR), SCHANGEDATA, SSEQTXSUB, SSEQTRXSUB, STXNEXT, STRXNEXT.</td>
</tr>
<tr>
<td>8-bit array</td>
<td>RRCVBUF1</td>
<td>Buffer for receive data from slave 1</td>
<td>main, SRXNEXT, STRXNEXT, STRXEND</td>
</tr>
<tr>
<td>16 bits</td>
<td>RSTTS1</td>
<td>Number of characters that can be sent to or received from slave 1</td>
<td>main, (SETTRXPNTR), STXCMD, SSTSCHK, SRXCMD, STRXCMD</td>
</tr>
<tr>
<td>8-bit array</td>
<td>RSNDBUF2</td>
<td>Buffer for transmit data to slave 2</td>
<td>main, (SETTRXPNTR), SCHANGEDATA, SSEQTXSUB, SSEQTRXSUB, STXNEXT, STRXNEXT.</td>
</tr>
<tr>
<td>8-bit array</td>
<td>RRCVBUF2</td>
<td>Buffer for receive data from slave 2</td>
<td>main, SRXNEXT, STRXNEXT, STRXEND</td>
</tr>
<tr>
<td>16 bits</td>
<td>RSTTS2</td>
<td>Number of characters that can be sent to or received from slave 2</td>
<td>main, (SETTRXPNTR), STXCMD, SSTSCHK, SRXCMD, STRXCMD</td>
</tr>
<tr>
<td>8 bits</td>
<td>RCSFLAG</td>
<td>Slave to which LSB is to access</td>
<td>main, SETTRXPNTR, SSLAVSEL, SSTSCHK</td>
</tr>
<tr>
<td>8 bits</td>
<td>RRCVBUF</td>
<td>Used to store receive data transmitted in single transfer mode.</td>
<td>SWAITRXEND, CSITXEND</td>
</tr>
<tr>
<td>8 bits</td>
<td>CSISTS</td>
<td>Number of remaining characters to be transferred</td>
<td>STXDATAST, SWAITTXEND, SRXDATAST, SWAITRXEND, SSEQRXSUB, SWAITSTREND, SSEQTXSUB, SSEQTRXSUB, CSITXEND, SRXNEXT, STXNEXT, STXEND, STRXNEXT, STRXEND</td>
</tr>
<tr>
<td>8-bit array</td>
<td>RCMPDATA</td>
<td>Area for storing the expected value of the receive data</td>
<td>SCHANGEDATA, SCHKDATSUB</td>
</tr>
</tbody>
</table>
5.5 List of Functions (Subroutines)

Table 5.6 summarizes the functions (subroutines) that are used in this sample program.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET_START</td>
<td>Makes initial settings of hardware and calls the main function.</td>
</tr>
<tr>
<td>SINIPORT</td>
<td>Sets up the I/O ports.</td>
</tr>
<tr>
<td>SINICLK</td>
<td>Sets up the clock generation circuit.</td>
</tr>
<tr>
<td>SINISAU</td>
<td>Initialize CSIp.</td>
</tr>
<tr>
<td>SINITAU</td>
<td>Initialize TM0 and TM01H.</td>
</tr>
<tr>
<td>SSTARTINTV</td>
<td>Start 1-ms interval timer.</td>
</tr>
<tr>
<td>SSLAVSEL</td>
<td>Output CS signal to slave identified by slave number.</td>
</tr>
<tr>
<td>SWAIT1MS</td>
<td>Wait for 1ms interval timing in HALT state.</td>
</tr>
<tr>
<td>SWAITRDY</td>
<td>Wait until BUSY signal goes low, until timeout.</td>
</tr>
<tr>
<td>STSTCHK</td>
<td>Check slave status.</td>
</tr>
<tr>
<td>STXCMD</td>
<td>Transmit data to slave in continuous transmission mode.</td>
</tr>
<tr>
<td>SRXCMD</td>
<td>Receive data from slave in continuous reception mode.</td>
</tr>
<tr>
<td>STRXCMD</td>
<td>Transmit/receive data to and from slave in continuous transmission/reception mode.</td>
</tr>
<tr>
<td>SCHANGEDATA</td>
<td>Generate expected value from transmit data and next transmit data.</td>
</tr>
<tr>
<td>SETTRXPNTR</td>
<td>Set up pointer to data buffer that is determined by slave number.</td>
</tr>
<tr>
<td>SCHKDTXSUB</td>
<td>Compare receive data with expected value.</td>
</tr>
<tr>
<td>STXDATAST</td>
<td>Start 1-character transmission processing (send data from A register).</td>
</tr>
<tr>
<td>SWAITTXEND</td>
<td>Wait for end of 1-character transmission.</td>
</tr>
<tr>
<td>SRXDATAST</td>
<td>Start 1-character reception processing.</td>
</tr>
<tr>
<td>SWAITRXEND</td>
<td>Wait for end of 1-character reception (load receive data in A register).</td>
</tr>
<tr>
<td>STRXREADY</td>
<td>Check 1-character transfer state, set Z flag to 1 if end of transfer.</td>
</tr>
<tr>
<td>SSEQTXSUB</td>
<td>Start continuous transmission processing (send data designated by HL number of times specified in A register.</td>
</tr>
<tr>
<td>SSEQRXSUB</td>
<td>Start continuous reception processing (receive number of characters specified in A register into buffer designated by HL).</td>
</tr>
<tr>
<td>SSEQTRXSUB</td>
<td>Start continuous transmission/reception processing (HL: transmit pointer, DE: receive pointer, A: number of characters).</td>
</tr>
<tr>
<td>SWAITSTREND</td>
<td>Wait for end of continuous transfer.</td>
</tr>
<tr>
<td>SSETENDINT</td>
<td>Set up transfer end interrupts.</td>
</tr>
<tr>
<td>SSETEMPTYINT</td>
<td>Set up buffer empty interrupts.</td>
</tr>
<tr>
<td>SCHNG2TX</td>
<td>Stop operation temporarily and enable transmission mode (transfer end interrupt).</td>
</tr>
<tr>
<td>SCHNG2RX</td>
<td>Stop operation temporarily and enable reception mode (transfer end interrupt).</td>
</tr>
<tr>
<td>SCHNG2TRX</td>
<td>Stop operation temporarily and enable transmission/reception mode (transfer end interrupt).</td>
</tr>
<tr>
<td>SCHNG2TXS</td>
<td>Common processing for mode setup.</td>
</tr>
<tr>
<td>STARTCSIp</td>
<td>Enables CSI.</td>
</tr>
<tr>
<td>STOPCSIp</td>
<td>Disables CSI.</td>
</tr>
<tr>
<td>IINTCSIp</td>
<td>Start INTCSIp interrupt processing (branch to processing block).</td>
</tr>
<tr>
<td>CSITXEND</td>
<td>Process 1-character transfer end interrupts (load receive data into RRCVBUF).</td>
</tr>
<tr>
<td>CSRXNEXT</td>
<td>Process 1-character transfer end interrupts in continuous reception mode.</td>
</tr>
<tr>
<td>STXNEXT</td>
<td>Process buffer empty interrupt in continuous transmission mode.</td>
</tr>
<tr>
<td>STXDEND</td>
<td>Process transmit end interrupts in continuous transmission mode (set variable CSISTS to 0).</td>
</tr>
<tr>
<td>STRXNEXT</td>
<td>Process buffer empty interrupts in continuous transmission/reception mode.</td>
</tr>
<tr>
<td>STRXEND</td>
<td>Process transfer end interrupts in continuous transmission/reception mode.</td>
</tr>
</tbody>
</table>
5.6 Function (Subroutine) Specifications

This section describes the specifications for the functions that are used in the sample program.

<table>
<thead>
<tr>
<th>[Function Name]</th>
<th>RESET_START</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td>Make initial settings of the CPU by starting up from a reset.</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td>This function calls the main routine after setting the stack pointer and making initial settings for the hardware.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Function Name]</th>
<th>SINIPTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td>Set up the I/O ports</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td>This function sets the bits of port registers other than those for CSI-related pins and the CS signal to 0. This function sets unused pins as outputs where possible.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Function Name]</th>
<th>SINICLK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td>Set up the clock generation circuit</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td>This function makes initial settings of the registers related to the clock generation circuit.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Function Name]</th>
<th>SINISAU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td>Initialize CSIp.</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td>This function sets up the CSIp for type 1, 8 bits, MSB first, and transmission/reception on transfer end interrupts.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Function Name]</th>
<th>SINITAU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td>Initialize TM01.</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td>This function sets up the TM01 as two 8-bit interval timers.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
[Function Name] SSTARTINTV

Synopsis: Start TM01H.
Explanation: This function starts the TM01H (1-ms interval timer).
Arguments: None
Return value: None
Remarks: None

[Function Name] SSLAVSEL

Synopsis: Perform slave selection processing.
Explanation: This function generates the CS signal to the slave designated by RCSFLAG.0 and waits for a response from the slave. When a timeout is detected, the function turns off the CS signal.
Arguments: None
Return value: CY flag : [1: No slave response, 0: Slave response present]
Remarks: None

[Function Name] SWAIT1MS

Synopsis: Wait for 1-ms interval timing.
Explanation: This function disables vector interrupts and waits for the occurrence of a TM01H interrupt in HALT mode.
Arguments: None
Return value: None
Remarks: None

[Function Name] SWAITRDY

Synopsis: Wait response from slave.
Explanation: This function starts the TM01 (for timeout measurement) and waits for a response from the slave (BUSY signal going low). When a timeout is detected before the timer startup, the function turns off the CS signal and terminates processing.
Arguments: None
Return value: CY flag : [1: No slave response, 0: Slave response present]
Remarks: None

[Function Name] SSTSCHK

Synopsis: Check slave status.
Explanation: This function sends the Check Status command to the selected slave and stores the number of transmittable or receivable characters, which is received from the selected slave, in a work area. The function signals an error if a timeout is detected or the status (data count) that is received proves invalid.
Arguments: None
Return value: CY flag : [1: Invalid slave response, 0: Normal slave response]
Remarks: On normal execution, the function loads RSTTS1 or RSTTS2 with the numbers of transmittable and receivable characters to and from the slave.
### Function STXCMD

**Synopsis**
Transmit data to slave.

**Explanation**
This function sends a master transmit command to the slave, places the CSIp in transmission mode, and transmits the number of characters that the slave can receive, which is stored in RSTTS1 or RSTTS2, from the transmit data buffer.

**Arguments**
None

**Return value**
CY flag : [1: Invalid slave response, 0: Normal slave response]

**Remarks**
None

### Function SRXCMD

**Synopsis**
Receive data from slave.

**Explanation**
This function sends a master receive command to the slave, places the CSIp in reception mode, and receives and stores the number of characters that the slave can transmit, which is stored in RSTTS1 or RSTTS2, into the receive data buffer.

**Arguments**
None

**Return value**
CY flag : [1: Invalid slave response, 0: Normal slave response]

**Remarks**
None

### Function STRXCMD

**Synopsis**
Transmit/receive data to and from slave.

**Explanation**
This function sends a master transmit/receive command to the slave, places the CSIp in transmission/reception mode, and transmits the number of characters that the slave can transmit and receive, which is stored in RSTTS1 or RSTTS2, from the transmit data buffer and receives and stores received data in the receive data buffer.

**Arguments**
None

**Return value**
CY flag : [1: Invalid slave response, 0: Normal slave response]

**Remarks**
This function is called only when the numbers of the characters that the slave can transmit and receive are the same.

### Function SCHANGEDATA

**Synopsis**
Generate expected value from transmit data and next transmit data.

**Explanation**
This function generates the expected value of the data to be received next from the slave from the data that has been transmitted and places the value in a variable area (RCMPDATA), then update the contents of the transmit buffer.

**Arguments**
None

**Return value**
None

**Remarks**
None

### Function SETTRXPNTR

**Synopsis**
Set up buffer pointer associated with selected slave.

**Explanation**
This function loads the address of the area containing the transmit data associated with the slave designated by RCSFLAG.0 into the HL register and the address of the buffer for storing the receive data into the DE register.

**Arguments**
None

**Return value**
HL register : Address for storing transmit data
DE register : Address for storing receive data

**Remarks**
None
### Function Name: SCHKDTSUB

| **Synopsis** | Compare receive data with expected value. |
| **Explanation** | This function compares the data that is received with the expected value. The result is returned with the CY flag. |
| **Arguments** | None |
| **Return value** | CY flag: [1: Error detected in comparison result, 0: Comparison result is normal.] |
| **Remarks** | None |
The functions (subroutines) given below are available as general-purpose functions.

### Function Name: STXDATAST

**Synopsis**
Start 1-character transmission processing.

**Explanation**
This function writes data from the A register into the SIOp and starts communication processing. The function loads the address of CSITXEND into RCSISUBADDR as an INTCSIp processing routine and sets the number of work-in-progress characters to 1 before returning.

**Arguments**
- A register
- Transmit data

**Return value**
None (However, CSISTS is set to 1)

**Remarks**
The CSIp need be configured for transmission or transmission/reception.

### Function Name: SWAITTXEND

**Synopsis**
Wait for end of 1-character transmission.

**Explanation**
This function waits for the end of transmission processing (CSISTS = 0) that is started by the STXDATAST function.

**Arguments**
None

**Return value**
None

**Remarks**
The transmission end interrupts are processed by CSITXEND (CSISTS is set to 0).

### Function Name: SRXDATAST

**Synopsis**
Start 1-character reception processing.

**Explanation**
This function writes dummy data (0FFH) into the SIOp and starts receive processing. The function loads the address of CSITXEND into RCSISUBADDR as an INTCSIp processing routine and sets the number of work-in-progress characters to 1 before returning.

**Arguments**
None

**Return value**
None

**Remarks**
The CSIp needs to be configured for reception or transmission/reception.

### Function Name: SWAITRXEND

**Synopsis**
Wait for end of 1-character reception.

**Explanation**
This function waits for the end of reception processing (CSISTS = 0) that is started by the SRXDATAST function. Upon end of reception processing, the function reads the received characters from RRCVBUF.

**Arguments**
None

**Return value**
A register

**Remarks**
The receive data is stored in RRCVBUF by CSITXEND.
[Function Name] STRXREADY

**Synopsis**
Check 1-character transfer state.

**Explanation**
This function checks CSISTS to examine the transmission or reception state. The function returns with the Z flag set to 0 if the communication is not yet completed and with the Z flag set to 1 if the communication is completed.

**Arguments**
None

**Return value**
- Z flag: [1: Communication complete, 0: Communication in progress]
- A register: Receive data (contents of RRCVBUF) if communication is completed

**Remarks**
None

[Function Name] SSEQTXSUB

**Synopsis**
Start continuous transmission processing.

**Explanation**
This function places the CSIp in transmission mode and starts transmission processing to send the number of characters specified in the A register from the buffer designated by the HL register. The function verifies the initiation of data transmission by testing the TSF bit. If the number of characters to be transmitted is 2 characters or more, the function changes the interrupt timing to that for buffer empty interrupts and loads RCSISUBADDR with the address of STXNEXT as an INTCSIp processing routine.

If the number of characters to be transmitted is 1 character, the function loads RCSISUBADDR with the address of STXEND.

The function returns after setting the value of the A register to the in-communication data count (CSISTS). It returns with a Z flag value of 1 if the number of receive characters in the A register is 0.

**Arguments**
- HL register: Address of area for storing the transmit data
- A register: Number of characters to be transmitted

**Return value**
- Z flag: [0: Normal startup, 1: Data count is 0.]

(CSISTS is set to the number of characters at normal startup time.)

**Remarks**
None

[Function Name] SSEQRXSUB

**Synopsis**
Start continuous reception processing.

**Explanation**
This function places the CSIp in receive mode, loads the buffer designated by the HL register with the number of characters designated by the A register, and starts reception processing. The function loads the address of SRXNEXT into RCSISUBADDR as an INTCSIp processing routine and sets the A register to the number of work-in-progress characters (CSISTS) before returning. The function returns with the Z flag set to 1 if the number of received characters in the A register is 0.

**Arguments**
- HL register: Address of area for storing receive data
- A register: No of receive characters

**Return value**
- Z flag: [0: Normal startup, 1: Number of characters is 0.]

(CSISTS is set to the number of characters at normal startup time.)

**Remarks**
None
Function Name: SSEQTRXSUB

Synopsis: Start continuous transmission/reception processing.

Explanation: This function places the CSIp in transmission/reception mode and starts the function to transmit and receive the number of data designated by the A register from the buffer designated by the HL register. The function verifies the initiation of data transmission/reception processing by testing the TSF bit.

If the number of characters to be transmitted is 2 characters or more, the function changes the interrupt timing to that for buffer empty interrupts and loads RCSISUBADDR with the address of STRXNEXT as an INTCSIp processing routine. If the number of characters to be transmitted is 1 character, the function loads RCSISUBADDR with the address of STRXEND.

The function returns after setting the value of the A register to the in-communication data count (CSISTS). It returns with a Z flag value of 1 if the number of receive characters in the A register is 0.

Arguments:
- HL register: Address of area storing the transmit data
- DE register: Address of area for storing receive data
- A register: Number of transfer characters

Return value:
- Z flag: [ 0: Normal startup, 1: Data count is 0.]
  (CSISTS is set to the number of characters at normal startup time.)

Remarks: None
[Function Name] SWAITSTREND

<table>
<thead>
<tr>
<th><strong>Synopsis</strong></th>
<th>Wait for end of continuous transfer.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation</strong></td>
<td>This function waits until the number of work-in-progress characters (CSISTS) reaches 0 during end of wait processing that is common to continuous reception, transmission, and transmission/reception processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

[Function Name] SSETENDINT

<table>
<thead>
<tr>
<th><strong>Synopsis</strong></th>
<th>Set up transfer end interrupts.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation</strong></td>
<td>Sets the CSIp interrupt timing to end of transfer.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

[Function Name] SSETEMPTYINT

<table>
<thead>
<tr>
<th><strong>Synopsis</strong></th>
<th>Set up buffer empty interrupts.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation</strong></td>
<td>Sets the CSIp interrupt timing to buffer empty interrupts.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

[Function Name] SCHNG2TX

<table>
<thead>
<tr>
<th><strong>Synopsis</strong></th>
<th>Set CSIp in transmission mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation</strong></td>
<td>This function stops the CSI temporarily and enables the transmission mode. The interrupt timing is set to end of transfer.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

[Function Name] SCHNG2RX

<table>
<thead>
<tr>
<th><strong>Synopsis</strong></th>
<th>Set CSIp in reception mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation</strong></td>
<td>This function stops the CSI temporarily and enables the reception mode. The interrupt timing is set to end of transfer.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
### SCHNG2TRX

**Synopsis**  
Set CSIp in transmission/reception mode

**Explanation**  
This function stops the CSIp temporarily and enables the transmission/reception mode. The interrupt timing is set to end of transfer.

**Arguments**  
None

**Return value**  
None

**Remarks**  
None

### SCHNG2TXS

**Synopsis**  
Common processing for mode setup

**Explanation**  
This function stops the CSIp temporarily and changes the setting for the mode to that set in the AX register so that operations can proceed. The interrupt timing is set for the end of transfer.

**Arguments**  
None

**Return value**  
None

**Remarks**  
None

### STARTCSIp

**Synopsis**  
Enable CSIp.

**Explanation**  
This function enables the CSIp.

**Arguments**  
None

**Return value**  
None

**Remarks**  
None

### STOPCSIp

**Synopsis**  
Disable CSIp.

**Explanation**  
This function disables the CSIp.

**Arguments**  
None

**Return value**  
None

**Remarks**  
None

### IINTCSIp

**Synopsis**  
Start INTCSIp interrupt processing.

**Explanation**  
This function is activated on an INTCSIp and causes a branch to the address that is stored in the variable RCSISUBADDR.

**Arguments**  
None

**Return value**  
None

**Remarks**  
None
### Function Name: CSITXEND

| **Synopsis** | Perform 1-character transfer end interrupt processing. |
| **Explanation** | This function reads receive data from the CSIp into RRCVBUF and sets the number of work-in-progress characters (CSISTS) to 0. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

### Function Name: SRXNEXT

| **Synopsis** | Perform 1-character transfer end interrupt processing in continuous reception mode. |
| **Explanation** | This function reads receive data from the CSIp into the buffer area and decrements the number of characters (CSISTS) by 1. If the number of remaining characters is 2 or more, the function writes dummy data into the SIOp to start the receive function. If the number of remaining characters is 1, the function switches the interrupt timing to transfer end interrupt. The function terminates processing when the number of remaining characters is 0. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

### Function Name: STXNEXT

| **Synopsis** | Perform buffer empty interrupt processing in continuous transmission mode. |
| **Explanation** | If the number of remaining characters is 1, this function switches the interrupt timing to transfer end interrupt and changes the value of RCSISUBADDR to the address of STXEND as an INTCSIp processing routine. If the number of remaining characters is 2 or more, the function decrements the number of work-in-progress characters (CSISTS) by 1 and writes the data from the transmit data buffer into the SIOp. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

### Function Name: STXEND

| **Synopsis** | Perform transmit end interrupt processing in continuous transmission mode. |
| **Explanation** | This function performs transmit end interrupt processing in continuous transmission mode. The function sets the number of work-in-progress characters (CSISTS) to 0 to signals the end of communication. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |
### Function Name: STRXNEXT

**Synopsis**
Perform buffer empty interrupt processing in continuous transmission/reception mode.

**Explanation**
This function stores the receive data in the receive data buffer. If the number of remaining characters is 2 or more, this function decrements the number of work-in-progress characters (CSISTS) by 1 and writes the data from the transmit data buffer into the SIOp. If the number of remaining characters is 1, the function switches the interrupt timing to transfer end interrupt and changes the value of RCSISUBADDR to the address of STRXEND as an INTCSip processing routine.

**Arguments**
None

**Return value**
None

**Remarks**
None

### Function Name: STRXEND

**Synopsis**
Perform transfer end interrupt processing in continuous transmission/reception mode.

**Explanation**
This function performs transfer end interrupt processing in continuous transmission/reception mode. The function stores the receive data in the receive data buffer and sets the number of work-in-progress characters (CSISTS) to 0 to signal the end of communication.

**Arguments**
None

**Return value**
None

**Remarks**
None
5.7 Flowcharts

Figure 5.1 shows the overall flow of the sample program described in this application note.

![Flowchart]

The option bytes are referenced before the CPU initialization function is called.

Figure 5.1 Overall Flow
5.7.1 CPU Initialization Function

Figure 5.2 shows the flowchart for the CPU initialization function.

![Flowchart for CPU Initialization Function](image)

- **RESET_START**
- Set up ES register: ES ← 00H (for table reference)
- Set up stack pointer
- Set up redirection
- Set up I/O ports SINIPORT
- Set up clock generation circuit SINICLK
- Set up SAU SINISAU
- Set up TAU SINITAU
- Call main routine routine
- HALT

**Figure 5.2 CPU Initialization Function**

- PIOR ← 00H
- P0 (CSI00 pin) ← 0001111B
- Leave P00 to P02 as input.

Select HOCO (20 MHz) as an operation clock.

Set CSI mode to type 1, 1 Mbps, transmission/reception, 8 bits and MSB first.

Set TM01 to 8-bit interval timer.
5.7.2 I/O Port Setup

Figure 5.3 shows the flowchart for I/O port setup.

![Flowchart for I/O Port Setup]

*Figure 5.3  I/O Port Setup*

**Note:** Refer to the section entitled "Flowcharts" in RL78/G10 Initialization Application Note (R01AN1454E) for the configuration of the unused ports.

**Caution:** Provide proper treatment for unused pins so that their electrical specifications are observed. Connect each of any unused input-only ports to VDD or VSS via separate resistors.
5.7.3 Clock Generation Circuit Setup

Figure 5.4 shows the flowchart for clock generation circuit setup.

- SINICLK
- Set up high-speed system clock
  - CMC register ← 00000000B: Do not use high-speed system clock.
  - MSTOP bit ← 1
- Set up operation speed mode control register
- Select CPU/peripheral hardware clock (fCLK) Note
  - WUTMMCK0 bit ← 0: Stop interval timer clock.
  - MCM0 bit ← 0: Select HOCO clock (fHOCO) as main system clock (fMAIN).
- Select frequency of high-speed on-chip oscillator
  - HOCODIV2 to HOCODIV0 bits ← 001: Set HOCO frequency to 20 MHz.
- return

Figure 5.4 Clock Generator Circuit Setup

Note: 16-pin products only
Caution: For details on the procedure for setting up the clock generation circuit (SINICLK), refer to the section entitled "Flowcharts" in RL78/G10 Initialization Application Note (R01AN1454E).
5.7.4 SAU Setup

Figure 5.5 shows the flowchart for SAU setup.

Figure 5.5  SAU Setup

- **SINISAU**
  - Supply clock signals to SAU
    - **SAU0EN** bit ← 1: Start supplying clock to the SAU.
  - Select serial clock
    - **SPS0L** register ← 00H
      - CK00: fCLK (20 MHz)
      - CK01: fCLK (20 MHz)
  - Stop channel operation
  - **ST0** register ← #TRGONn
  - **SMRmn** register ← 0020H
    - Operation clock: CK00 (20 MHz)
    - Start trigger source: Software trigger
    - Operation mode: CSI
    - Interrupt: Transfer end interrupt
  - **SCRmn** register ← 0C007H
    - Communication mode: Enable transmission/reception
    - Data length: 8 bits
    - Parity: None
    - Transfer order: MSB first
    - Stop bits: None
    - Clock/data phase: Type 1
  - **SDR00** register ← #SDRDATA
    - (Divide the operation clock frequency (20 MHz) by 20.)
  - **SOm** register ← #SOHIGH*101H (Initial level is high)
  - **SOEmL** register ← #SOEON (Enable channel output)
  - **SIRmnL** register ← 07H
  - **PM_SCKp** bit ← 0: SCKp output
  - **PM_SOp** bit ← 0: SOp output
  - **PM_Sip** bit ← 1: Sip input

return
Starting clock signal supply to SAU
- Peripheral enable register 0 (PER0)
  Start supplying clock signals.

Symbol: PER0

<table>
<thead>
<tr>
<th>TMKAEN&lt;sup&gt;Note&lt;/sup&gt;</th>
<th>CMPEN&lt;sup&gt;Note&lt;/sup&gt;</th>
<th>ADCEN</th>
<th>IIICA0EN&lt;sup&gt;Note&lt;/sup&gt;</th>
<th>0</th>
<th>SAU0EN</th>
<th>0</th>
<th>TA0U0EN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

Bits 2

<table>
<thead>
<tr>
<th>SAU&lt;sup&gt;m&lt;/sup&gt;EN</th>
<th>Control of serial array unit n input clock supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stops supply of input clock.</td>
</tr>
<tr>
<td>1</td>
<td>Enables supply of input clock.</td>
</tr>
</tbody>
</table>

Selecting a serial clock
- Serial clock select register m (SPS0)
  Select an operation clock for SAU.

Symbol: SPS0

<table>
<thead>
<tr>
<th>PRS013</th>
<th>PRS012</th>
<th>PRS011</th>
<th>PRS010</th>
<th>PRS003</th>
<th>PRS002</th>
<th>PRS001</th>
<th>PRS000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 7 to 0

<table>
<thead>
<tr>
<th>PRS 0n3</th>
<th>PRS 0n2</th>
<th>PRS 0n1</th>
<th>PRS 0n0</th>
<th>Selection of operation clock (CK0n) (n = 0 to 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;&lt;sup&gt;=&lt;/sup&gt; 1.25MHz</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;&lt;sup&gt;=&lt;/sup&gt; 2.5MHz</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;&lt;sup&gt;=&lt;/sup&gt; 5MHz</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;&lt;sup&gt;=&lt;/sup&gt; 10MHz</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;&lt;sup&gt;=&lt;/sup&gt; 20MHz</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2</td>
<td>625 kHz</td>
<td>1.25 MHz</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;2&lt;/sup&gt;</td>
<td>313 kHz</td>
<td>625 kHz</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;3&lt;/sup&gt;</td>
<td>156 kHz</td>
<td>313 kHz</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;4&lt;/sup&gt;</td>
<td>78 kHz</td>
<td>156 kHz</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;5&lt;/sup&gt;</td>
<td>39 kHz</td>
<td>78 kHz</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;6&lt;/sup&gt;</td>
<td>19.5 kHz</td>
<td>39 kHz</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;7&lt;/sup&gt;</td>
<td>9.8 kHz</td>
<td>19.5 kHz</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;8&lt;/sup&gt;</td>
<td>4.9 kHz</td>
<td>9.8 kHz</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;9&lt;/sup&gt;</td>
<td>2.5 kHz</td>
<td>4.9 kHz</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;10&lt;/sup&gt;</td>
<td>1.22 kHz</td>
<td>2.5 kHz</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;11&lt;/sup&gt;</td>
<td>625 Hz</td>
<td>1.22 kHz</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;12&lt;/sup&gt;</td>
<td>313 Hz</td>
<td>625 Hz</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;13&lt;/sup&gt;</td>
<td>152 Hz</td>
<td>313 Hz</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;14&lt;/sup&gt;</td>
<td>78 Hz</td>
<td>152 Hz</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>f&lt;sub&gt;CLK&lt;/sub&gt;/2&lt;sup&gt;15&lt;/sup&gt;</td>
<td>39 Hz</td>
<td>78 Hz</td>
</tr>
</tbody>
</table>

Transiting to communication stopped state

- Serial channel stop register 0 (ST0)
  Stop communication

Symbol: ST0

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Bit n

- **ST0n** Operation stop trigger of channel n
  - 0 No trigger operation
  - 1 Sets the SE0n bit to 0 and stops the communication operation.

Setting up channel operation mode

- Serial mode register mn (SMRmn)
  - Interrupt source
  - Operation mode
  - Select transfer clock.
  - Select f_{MCK}.

Symbol: SMR0nH, SMR0nL

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKS0n</td>
<td>CCS0n</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Bit 15

- **CKSmn** Selection of operation clock (f_{MCK}) of channel n
  - 0 Prescaler output clock CK00 set by the SPSm register
  - 1 Prescaler output clock CK01 set by the SPSm register

Bit 14

- **CCSmn** Selection of transfer clock (T_{CLK}) of channel n
  - 0 Divided operation clock f_{MCK} set by the CKSmn bit
  - 1 Clock input from the SCK pin.

Bit 8

- **STSmn** Selection of start trigger source
  - 0 Only software trigger is valid
  - 1 Valid edge of the RxD pin (selected for UART reception)

Remark  n: Channel number (n = 0, 1)
Note    Provided in the SMR01H and SMR01L registers only
Symbol: SMR0nH, SMR0nL

<table>
<thead>
<tr>
<th>Bit</th>
<th>CKS 0n</th>
<th>CCS 0n</th>
<th>SIS 0n0</th>
<th>STS 0nNote</th>
<th>SIS 0n1</th>
<th>MD 0n2</th>
<th>MD 0n1</th>
<th>MD 0n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bit 6

- **SIS 0n0**: Controls inversion of level of receive data of channel n in UART mode
  - 0: Falling edge is detected as the start bit.
  - 1: Rising edge is detected as the start bit.

Bits 2 and 1

- **MDmn2**: Setting of operation mode of channel n
  - 0 0: CSI mode
  - 0 1: UART mode
  - 1 0: Simplified I2C mode
  - 1 1: Setting prohibited

Bit 0

- **MDmn0**: Selection of interrupt source of channel n
  - 0: Transfer end interrupt
  - 1: Buffer empty interrupt

Remark
n: Channel number (n = 0, 1)

Note
Provided in the SMR01H and SMR01L registers only

Caution
Setting up channel communication mode

- Serial communication operation register mn (SCRmn)
  Setup data length, data transfer order, and operation mode.

Symbol: SCR0nH, SCR0nL

<table>
<thead>
<tr>
<th>Symbol</th>
<th>TXE0n</th>
<th>RXE0n</th>
<th>DAP0n</th>
<th>CKP0n</th>
<th>EOC0n</th>
<th>PTC0n</th>
<th>PTC0n</th>
<th>DIR0n</th>
<th>SLC0n1</th>
<th>SLC0n0</th>
<th>DLS0n1</th>
<th>DLS0n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Bits 15 and 14

- **TXE0n, RXE0n** Setting of operation mode of channel n
  - 0 0: Disable communication.
  - 0 1: Reception only
  - 1 0: Transmission only
  - 1 1: Transmission/reception

Bit 10

- **EOC0n** Selection of masking of error interrupt signal (INTSREn)
  - 0: Masks error interrupt INTSRE0.
  - 1: Enables generation of error interrupt INTSREx.

Bits 9 and 8

- **PTCmn** Setting of parity bit in UART mode
  - Transmission
  - Reception
  - 0 0: Does not output the parity bit. Receives without parity.
  - 0 1: Outputs 0 parity. No parity judgment
  - 1 0: Outputs even parity. Judged as even parity
  - 1 1: Outputs odd parity. Judged as odd parity

Bit 7

- **DIRmn** Selection of data transfer sequence in CSI and UART modes
  - 0: Inputs/outputs data with MSB first.
  - 1: Inputs/outputs data with LSB first.

Bits 5 and 4

- **SLCmn** Setting of stop bit in UART mode
  - 0 0: No stop bit
  - 0 1: Stop bit length = 1 bit
  - 1 0: Stop bit length = 2 bits
  - 1 1: Setting prohibited

Remark: **n**: channel number (n=0, 1)

Note: SCR00L register only

Symbol: SCR0nH, SCR0nL

<table>
<thead>
<tr>
<th>TXE 0n</th>
<th>RXE 0n</th>
<th>DAP 0n</th>
<th>CKP 0n</th>
<th>EOC 0n</th>
<th>PTC 0n1</th>
<th>PTC 0n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIR 0n</th>
<th>SCR0n1</th>
<th>SLC 0n1</th>
<th>SCR0nL</th>
<th>SLC 0n0</th>
<th>1</th>
<th>DLS 0n1</th>
<th>DLS 0n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Remarks:
- n: channel number (n=0, 1)
- Note: SCR00L register only


Bits 1 and 0

<table>
<thead>
<tr>
<th>DLSmn1</th>
<th>DLSmn0</th>
<th>Setting of data length in CSI mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>7-bit data length</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8-bit data length</td>
</tr>
</tbody>
</table>

Remarks:
- n: channel number (n=0, 1)
- Note: SCR00L register only

Setting up channel transfer clock
- Serial data register mn (SDR0nH, SDR0nL)
  - Transfer clock frequency: \( f_{MCK}/20 = 1 \text{ MHz} \)

Symbol: SDR0nH, SDR0nL

<table>
<thead>
<tr>
<th>SDRmn[15:9]</th>
<th>Transfer clock setting by dividing operation clock ( f_{MCK} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 0</td>
<td>( f_{MCK}/20 )</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 1 0</td>
<td>( f_{MCK}/20 )</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 0 0</td>
<td>( f_{MCK}/20 )</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 1 0 0</td>
<td>( f_{MCK}/20 )</td>
</tr>
</tbody>
</table>

Remarks:
- n: channel number (n=0, 1)
Setting initial output level

- Serial output register m (SOm)
  Initial output: 1

Symbol: SO0

+---------------------------------+--------+--------+--------+--------+--------+--------+--------+--------+
| Bit n                           |   7    |   6    |   5    |   4    |   3    |   2    |   1    |   0    |
|---------------------------------+--------+--------+--------+--------+--------+--------+--------+--------|
| Serial clock output of channel n|        |        |        |        |        |        |        |        |
| 0                               | 0      | 0      | 0      | 0      | 0      | 0      | SO01   | SO00   |
| 1                               | 0/1    | 0/1    |        |        |        |        |        |        |

Remark: n: channel number (n=0, 1)

Enabling target channel data output

- Serial output enable register m (SOEm/SOEmL)
  Enable output

Symbol: SOE0

+---------------------------------+--------+--------+--------+--------+--------+--------+--------+--------+
| Bit n                           |   7    |   6    |   5    |   4    |   3    |   2    |   1    |   0    |
|---------------------------------+--------+--------+--------+--------+--------+--------+--------+--------|
| Serial output enable/disable of channel n|        |        |        |        |        |        |        |        |
| 0                               | 0      | 0      | 0      | 0      | 0      | 0      | SOE01  | SOE00  |
| 1                               | 0/1    | 0/1    |        |        |        |        |        |        |

Clearing the error flags

- Serial flag clear trigger register mn (SIRmn)
  - Clear the error flags

Symbol: SIR0n

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>FECT0n</th>
<th>Clear trigger of framing error of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Not cleared</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Clears the FEFmn bit of the SSRmn registers to 0.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>PECT0n</th>
<th>Clear trigger of parity error of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Not cleared</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Clears the PEFmn bit of the SSRmn registers to 0.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>OVCT0n</th>
<th>Clear trigger of overrun error of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Not cleared</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Clears the OVFmn bit of the SSRmn registers to 0.</td>
</tr>
</tbody>
</table>

Remark: n: channel number (n=0, 1)
Note: SIR01 register only
Port setting (For CSI00)
- Port register 0 (P0)
- Port mode register 0 (PM0)

Port setting for each of serial clock, transmit data and receive data.

Symbol: P0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Bit 2-0

<table>
<thead>
<tr>
<th>Bit</th>
<th>P0n</th>
<th>Output data control (in output mode) (n=0-2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 is output</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1 is output</td>
</tr>
</tbody>
</table>

Symbol: PM0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bit 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>PM02</th>
<th>P02 pin I/O mode selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Output mode (output buffer on)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Input mode (output buffer off)</td>
</tr>
</tbody>
</table>

Bit 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>PM01</th>
<th>P01 pin I/O mode selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Output mode (output buffer on)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Input mode (output buffer off)</td>
</tr>
</tbody>
</table>

Bit 0

<table>
<thead>
<tr>
<th>Bit</th>
<th>PM00</th>
<th>P00 pin I/O mode selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Output mode (output buffer on)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Input mode (output buffer off)</td>
</tr>
</tbody>
</table>

Note: 16-pin product only

5.7.5 Timer Array Unit Setup

Figure 5.6 shows the flowchart for setting up the timer array unit.

![Flowchart for Timer Array Unit Setup]

- **SINITAU**: Supply clock signals to timer array unit
- **Stop channel operation**
- **Clock select**
- **Set up operation mode**
- **Set interval time**
- **Disable channel output**
- **Set initial output values**
- **Set up interrupts**
- **return**

### Details:

- **TAU0EN** bit ← 1: Starts clock supply.
- **TT0L** register ← #00000010B
- **TPS0** register ← 0007H
  - CK00: \( f_{\text{CLK}} / 2^{7} \) (156 kHz)
  - CK01: \( f_{\text{CLK}} \) (20 MHz)
- **TMR01H** register ← 08H
- **TMR01L** register ← 00H
  - Operation clock: CK00 (156 kHz)
  - 8/16 bit operation: 8 bits \( \times \) 2 channels
  - Operation mode: Interval timer
  - Trigger: Software
- **TDR01H** register ← #TDRDATA (155): 1 ms
- **TDR01L** register ← 01: 10 \( \mu \)s
- **TOE0.1** bit ← 0 (Channel output disabled.)
- **TOL0** register ← 00H
- **TO0** register ← 00H

### Interrupts:

- **TMMK01** bit ← 1: Mask interrupt.
- **TMMK01H** bit ← 1: Mask interrupt mask.
- **TMIF01** bit ← 0: Clear the interrupt request.
- **TMIF01H** bit ← 0: Clear the interrupt request.

---

*Figure 5.6 Timer Array Unit Setup*
5.7.6 Main Processing

Figures 5.7 to 5.9 show the flowcharts for the main processing.

Figure 5.7 Main Processing (1/3)

1. **Initialize data buffers.**
   - Initialize data buffers.
   - RSNDBUF1/2 (transmit buffer) ← Decrement data
   - RRCVBUF1/2 (receive buffer) ← 0
   - RSTTS1/2 (slave status) ← 0

2. **Start 1-ms timer.**
   - Start 1-ms interval timer.

3. **Enable CSIp and unmask interrupts.**
   - IE ← 1: Enable vector interrupts

4. **Wait for 1 ms.**
   - Disable the TM01H for 1-ms interval vector interrupts and wait in the HALT state.

5. **Select slave.**
   - Output the CS signal designated by the RCSFLAG.0 bit to the slave and wait until the BUSY signal goes low.

6. **Ready response?**
   - Terminate processing and cause a branch if a timeout occurs.

7. **Status check command.**
   - Send a status check command to the selected slave and receive the number of transmittable/receivable characters.

8. **Normal response?**
   - Terminate processing and cause a branch if an abnormal response is received from the slave.

9. **Transmit command.**
   - Perform transmit command processing.

10. **Normal end?**
    - Terminate processing and cause a branch if an abnormal response is received from the slave.
Check transmit/receive data count

Data count match?

Yes

Data transmit/receive command

STRXCMD

Normal end?

No

Check receive data

SCHKDTSUB

Data mismatch?

Yes

HALT

No

Update data

SCHANGEDATA

Data receive command

SRXCMDC

No

Check transmit/receive data count

Data count match?

Yes

Data transmit/receive command

STRXCMD

Normal end?

No

Check receive data

SCHKDTSUB

Data mismatch?

Yes

HALT

No

Update data

SCHANGEDATA

Data receive command

SRXCMDC

RCMPDATA ← Expected value of response for previously transmitted data
RSNDBUF1/2 ← Next transmit data (continued from preceding data)

Get the pointer to the target slave.

Check if the slave’s transmittable data count and receivable data count match.

Cause a branch to receive command if the data counts disagree.

Perform transmit/receive command processing.

Check if the value of received data matches the expected value.

Enter the HALT mode if the value of the received data does not match the expected value.

RCMPDATA ← Expected value of response for the previously transmitted data
RSNDBUF1/2 ← Next transmit data (continued from preceding data)

Perform receive command processing.

Figure 5.8  Main Processing (2/3)
Check if the value of received data matches the expected value.

Data mismatch?
- Yes: HALT
- No: Change slave

Normal end?
- Yes
- No: Check receive data SCHKDTSUB

Enter the HALT mode if the value of the received data does not match the expected value.

Invert the RCSFLAG.0 bit identifying the slave and continue processing starting at the status check step.

Figure 5.9 Main Processing (3/3)
5.7.7 1-ms Interval Timer Startup Processing

Figure 5.10 shows the flowchart for the 1-ms interval timer startup function.

![Flowchart](image)

Transiting to timer operating state

- Timer channel startup register 0 (TSH0)
  
  Symbol: TSH0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSH03 Note</td>
<td>0</td>
<td>TSH01</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

  Bit 11

<table>
<thead>
<tr>
<th>TSH01</th>
<th>Operation start trigger of channel 01H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No trigger operation</td>
</tr>
<tr>
<td>1</td>
<td>The TEH03 bit is set to 1 and the count operation becomes enabled.</td>
</tr>
</tbody>
</table>


Note: 16-pin products only
5.7.8 1-ms Interval Wait Function

Figure 5.11 shows the flowchart for the 1ms interval wait function.

![Flowchart for 1ms interval wait function]

Figure 5.11 1ms interval wait function

Interrupt setting
- Interrupt request flag register (IF0L)
  Clear the interrupt request flag

Symbol: IF0L (10-pin products)

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMIF00</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMIF01H</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SREIF0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRIF0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STIF0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSIIF00</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IICIF00</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIF1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIF0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDTIIF</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bit 6

<table>
<thead>
<tr>
<th>TMIF01H</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No interrupt request signal is generated</td>
</tr>
<tr>
<td>1</td>
<td>Interrupt request is generated, interrupt request status</td>
</tr>
</tbody>
</table>

### 5.7.9 Slave Selection Processing

Figure 5.12 shows the flowchart for the slave selection processing.

Read the number of the slave to select from the variable RCSFLAG. CY bit ← RCSFLAG.0 bit

Count up the data count (RDATACNT) (+1).

Drive the CS2 signal to the low level. CLR CS2SIG

Drive the CS1 signal to the low level. CLR CS1SIG

Subsequently wait for a response from the slave.
5.7.10 Slave Response Wait Processing

Figure 5.13 shows the flowchart for the slave response wait processing.

- **SWAITRDY**
  - Clear TM01 interrupt request
  - Start TM01

- **Read BUSY signal**
  - No timeout occurred?
    - Yes
    - Is slave READY?
      - Yes
      - Is slave BUSY?
        - Yes
        - Turn off CS signal
        - Stop TM01
        - Clear TM01 interrupt request
        - return
        - No
        - No timeout occurred?
          - Yes
          - Is slave READY?
            - Yes
            - No
            - Read the BUSY signal (P137) from the slave.
            - CY flag ← BUSYSIG: Place content of P137 in CY.
            - Exit the loop when a timeout occurs (TMIF01 = 1).
            - Wait within the loop until a READY response (BUSY signal low) is received.
            - When timeout is encountered (CY = 1), set the CS signal, which selects the slave, to the high level.
            - OR P0, #SELOFFSIG (00011000B)
            - Stop the timeout measuring TM01.
            - TT0.1 bit ← 1: Stop TM01.
            - Clear the TM01 interrupt request.
            - TMIF01 bit ← 0: Clear the TM01 interrupt request.

- **Figure 5.13 Slave Response Wait Processing**
Transiting to timer operating state

- Timer channel startup register 0 (TS0)
  Start counting.

Symbol: TS0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TS03</td>
<td>TS02</td>
<td>TS01</td>
<td>TS00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>TS01</th>
<th>Operation enable (start) trigger of channel 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No trigger operation</td>
</tr>
<tr>
<td>1</td>
<td>TE01 is set to 1 and the count operation becomes enabled.</td>
</tr>
</tbody>
</table>

Transiting to timer stopped state

- Timer channel stop register 0 (TT0)
  Stop counting.

Symbol: TT0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TT03</td>
<td>TT02</td>
<td>TT01</td>
<td>TT00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>TT01</th>
<th>Operation stop trigger of channel 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No trigger operation</td>
</tr>
<tr>
<td>1</td>
<td>TE01 is cleared to 0. Operation is stopped (stop trigger is generated).</td>
</tr>
</tbody>
</table>

Interrupt setting
- Interrupt request flag register (IF0H)
  Clear the interrupt request flag
- Interrupt mask flag register (MK0H)
  Cancel interrupt mask

Symbol: IF0H (10-pin products)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>KRIF</td>
<td>ADIF</td>
<td>TMIF01</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>

Bit 0

<table>
<thead>
<tr>
<th>TMIF01</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No interrupt request signal is generated</td>
</tr>
<tr>
<td>1</td>
<td>Interrupt request is generated, interrupt request status</td>
</tr>
</tbody>
</table>

Symbol: MK0H (10-pin products)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>KRMK</td>
<td>ADMK</td>
<td>TMMK01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>

Bit 1

<table>
<thead>
<tr>
<th>TMMK01</th>
<th>Interrupt processing control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enables interrupt processing.</td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
</tr>
</tbody>
</table>

5.7.11 Slave Status Check Processing
Figures 5.14 to 5.16 show the flowcharts for slave status check processing.

![Flowchart of Slave Status Check Processing]

Stop the CSIp temporarily, place it into transmission/reception mode, then restart.

Load status check command into the A register.
A register ← #STSCHKCMD

Send the command from the A register to the slave.

Set up the pointer to the area for storing the response message from slave 1.
HL register ← #LOWW RSTTS1

Set up the pointer to the area for storing the response message from slave 2.
HL register ← #LOWW RSTTS2

Clear the area for storing the response message.
AX register ← 0000H
[HL] ← AX register

Wait for the completion of command transmission.

Wait for the response from the slave to the command that is sent.

Terminate processing if a timeout is detected.

Figure 5.14 Status Check Processing (1/3)
Start the function to receive the response to the status check command.

Wait for the end of data transmission and get the status.

Read bit 7 of the received status.
CY bit ← A.7 bit (CY = 1 if error)

Terminate processing if bit 7 of the received status is 1.

Go to receivable data count processing of bit 6 of the received status is 0.

Extract the slave transmittable data count from the response message.
A register ← Slave transmittable data count

Compare the slave transmittable data count with the size of the receive buffer and place the smaller value in the status area.

CMP A, #CRXDTNO
SKC
MOV A, #CRXDTNO
MOV [HL], A

Wait for the end of data transmission and get the status.

Wait for slave response SWAITRDY

Normal response?

Yes

Terminate processing if a timeout is detected.

No

Start response data reception SRXDATAST

Wait for end of reception SWAITRXEND

Check receive status

Bit 7 is 0?

No

D

Yes

Bit 6 is 1?

No

C

Yes

Extract data count

Compare with receive buffer size

Buffer smaller?

No

Set data count to buffer size

Store in status area

Wait for slave response SWAITRDY

Normal response?

Yes

Figure 5.15  Status Check Processing (2/3)
Start receive processing for the second status.

Wait for the end of data reception and get the received status.

Read bit 7 of the received status.
CY bit ← A.7 bit (CY = 1 if error)
CY bit ← CY bit | A.6 bit

Terminate processing if bits 6 and 7 of the received status are not 0.

Compare the slave receivable data count with the size of the transmit buffer and place the smaller value in the status area.

Store in status area

Set the CS signal, which selects the slave to the high level.
OR P0, #SELOFFSIG (00011000B)
5.7.12 Continuous Data Transmission Processing

Figures 5.17 and 5.18 show the flowcharts for the processing for transmitting data continuously to the slave.

---

**Figure 5.17  Continuous Data Transmission Processing (1/2)**

- **STXCMD**
  - Set up pointer
    - SETTRXPNTR
  - Check transmit data count
    - Transmit data count > 0?
      - Yes
        - Put CSIp into transmission/reception mode
          - SCHNG2TRX
        - Stop the CSIp temporarily, place it into transmission/reception mode, then restart.
      - No
        - Generate transmit command
          - Send command
            - STXDATAST
  - If the transmit data count is 0, set the CY flag and terminate processing without doing anything.
    - MOV A, [HL + CRXDTNO + CTXDTNO + 1]
    - CMP0 A
    - SET1 CY
    - BZ $STXCMDEND
  - Set up pointer associated with the slave to be accessed.
    - HL register ← Pointer for transmit data
  - Transmit data count > 0?
    - Yes
      - Set up pointer
        - SETTRXPNTR
      - Transmit data count
        - Transmit data count > 0?
          - Yes
            - Generate a transmit command while setting bit 7 of the data count.
            - Send the command in the A register to the slave.
            - Generate a transmit command while setting bit 7 of the data count.
            - Send the command in the A register to the slave.
            - Wait for end of command transmission
              - SWAITTXEND
            - Wait for slave response
              - SWAITRDY
            - Normal response?
              - Yes
                - B
              - No
                - A
                - Sending command
                  - STXDATAST
                  - Select slave
                    - !SSLAVSEL
                    - READY response?
                      - Yes
                        - B
                      - No
                        - A
  - Disable the TM01H for 1ms interval vector interrupts and wait in the HALT state.
  - Output the CS signal designated by the RCSFLAG.0 bit to the slave and wait until the BUSY signal goes low.
  - Terminate processing and cause a branch if a timeout occurs.

---

**Figure 5.17  Continuous Data Transmission Processing (2/2)**
Load the A register with the transmit data count. (The HL register points to the beginning of the transmit data buffer.)

A register ← [HL + CRXDTNO + CTXDTNO + 1]

Start CSIp continuous data transmission processing.

Wait for end of continuous data transmission processing.

Set the CS signal selecting the slave to the high level.
P0 register ← #SELOFFSIG (00011000B)
5.7.13 Continuous Data Reception Processing

Figures 5.19 and 5.20 show the flowcharts for the processing for receiving data continuously from the slave.
Load the A register with the receive data count. (The HL register points to the beginning of the receive data buffer.)

A register ← [HL+CRXDTNO]

Start CSIp continuous data reception processing.

Continuous reception startup processing
SSEQRXSUB

Wait for end of continuous reception
SWAITSTREND

Set the CS signal selecting the slave to the high level.
P0 register ← #SELOFFSIG (00011000B)

Wait for end of continuous data reception processing.

Turn off CS signal

Figure 5.20  Continuous Data Reception Processing (2/2)
5.7.14 Continuous Data Transmission/Reception Processing

Figures 5.21 and 5.22 show the flowcharts for the processing for transmitting and receiving data continuously to and from the slave.

```
Set up pointer associated with the slave to be accessed.
HL register ← Pointer for transmit data
DE register ← Pointer for receive data
If the transmit/receive data count is 0, set the CY flag and terminate processing without doing anything.
MOV A, [HL + CRXDTNO + CTXDTNO + 1]
CMP0 A
SET1 CY
BZ $STRXCMDEND

Stop the CSIp temporarily, place it into transmission/reception mode, then restart.

Disable the TM01H for 1ms interval vector interrupts and wait in the HALT state.

Output the CS signal designated by the RCSFLAG.0 bit to the slave and wait until the BUSY signal goes low.

Terminate processing and cause a branch if a timeout occurs.

Generate a transmit/receive command while setting bits 7 and 6 of the data count.

Send the command in the A register to the slave.

Wait for the end of command transmission.

Wait for a response from the slave to the command that is sent.
```

**Figure 5.21 Continuous Data Transmission/Reception Processing (1/2)**
Figure 5.22  Continuous Data Transmission/Reception Processing (2/2)

A

Set transmit/receive data count

Continuous data transmission/reception startup processing
SSEQTRXSUB

B

Load the A register with the transmit/receive data count.
(The HL register points to the beginning of the transmit data buffer.)
A register ← [HL + CRXDTNO +CTXDTNO + 1]

Start CS Ip continuous data transmission/reception processing.

Wait for end of continuous transmission/reception processing.

Wait for end of continuous data transmission/reception processing.

Turn off CS signal

P0 register ← #SELOFFSIG (00011000B)

return
5.7.15 Data Update Processing

Figure 5.23 shows the flowchart for the data update processing.

![Flowchart for Data Update Processing](image)

- **SCHANGEDATA**: Set up pointer associated with the slave to be accessed.
- **SETTRXPNT**: HL register ← Pointer for transmit data
- **Set up data count**: A register ← [HL + CRXDTNO + CTXDTNO + 1]
- **Set up expected value data pointer**: Load the DE register with the pointer to the area for storing the expected value data.
  DE register ← #LOWW RCMPDATA
- **Set up loop counter**: Load the B register with the loop count (data count).
  B register ← A register
- **Set up data offset**: Load the X register with the offset to the next data.
  X register ← A register
- **Calculate next data**: Add offset to the transmit data to calculate the next data.
  MOV A, X
  ADD A, [HL]
- **Calculate and store expected value data**: Take the complement of transmit data and select the expected value data.
  XCH A, [HL] : Store next data and read old data.
  XOR A, #0FFH : Calculate expected value data.
  MOV [DE], A : Store expected value data.
- **Count loop count**: Decrement loop counter by 1.
  DEC B
- **Update pointer**: Update the data pointer.
  INCW HL
  INCW DE

All data processed?  
Yes: return

*Figure 5.23 Data Update Processing*
5.7.16 Receive Data Check Processing

Figure 5.24 shows the flowchart for receive data check processing.

```
SCHKDTSUB

Set up pointer
SETTRXPNTR

Read data count

Set up loop counter

Set up expected value pointer

Read receive data

Compare with expected value data

Data match?

Yes

Update pointer and counter

No

All data processed?

Yes

No

return

SCHKEND

- Set up pointer associated with the slave to be accessed.
  - HL register ← Pointer for transmit data
  - DE register ← Pointer for receive data

- Load the A register with the transmit data count and the X register with the receive data count.
  - AX register ← [HL + CRXDTNO + CTXDTNO]

- Set the loop count (X register) to either transmit data count or receive data count, whichever is smaller.
  - CMP A, X
  - SKNC
  - MOV X, A

- Load the HL register with the pointer for reading the expected value data.
  - HL register ← #LOWW RCMPDATA

- Read the transmit data into the A register.
  - MOV A, [HL]

- Comparing the received data with the expected value data.
  - XOR A, [HL] : Compare data.
  - ADD A, #0FFH : Set CY flag if mismatch.

- Terminate processing if no data match occurs.
  - BC $CHKEND

- Update the data pointer and loop counter.
  - INCW HL
  - DEC X
  - INCW DE

Figure 5.24 Receive Data Check Processing
```
5.7.17 Data Pointer Setup Processing

Figure 5.25 shows the flowchart for data pointer setup processing.

```
SETTRXPNTR

Set up slave 1 pointer

Slave 2?
  Yes
    Set up slave 2 pointer
  No
    Compute receive pointer
    Setup receive pointer

return

Load the HL register with the address of the transmit buffer for slave 1.
HL register ← #LOWW RSNDBUF1

Skip if slave 1.
MOV1 CY, RCSFLAG.0
SKNC

Load the HL register with the address of the transmit buffer for slave 2.
HL register ← #LOWW RSNDBUF2

Load the AX register with the pointer to the area for storing the receive data.
AX register ← HL register + #CTXDTNO

Load the DE register with the pointer to the area for storing the receive data.
DE register ← AX register
```

Figure 5.25 Data Pointer Setup Processing
Given below is a collection of subroutines that are used to perform basic character-based communication processing. Two functions, for start and wait processing, are used in pair. Data is exchanged through the A register. It is necessary to establish the direction of communication (SCHNG2TX: master transmission, SCHNG2RX: master reception, SCHNG2TRX: master transmission and reception) before using the CSIp.

### 5.7.18 1-character Transmission Start Processing

Figure 5.26 shows the flowchart for 1-character transmission start processing.

![Figure 5.26 1-character Transmission Start Processing](image)

- STXDATAST
  - Load the SIOp register with the transmit data (transmission start).
    - SIOp register ← A register
  - Write transmit data
  - Set up address of interrupt processing
    - Variable RCSISUBADDR ← #LOWW CSITXEND
  - Set up data count
    - ONEB CSISTS
  - return

### 5.7.19 1-character Transmission End Wait Processing

Figure 5.27 shows the flowchart for 1-character transmission end wait processing.

![Figure 5.27 1-character Transmission End Wait Processing](image)

- SWAITTXEND
  - Check data count
    - CMP0 CSISTS
  - Is data count 0?
    - No
    - Yes
  - Wait until the data count in the variable CSISTS reaches 0.
  - return
5.7.20 1-character Reception Processing
Figure 5.28 shows the flowchart for 1-character reception processing.

![Flowchart for 1-character Reception Processing](image)

- **SRXDATAST**
  - Write dummy data
  - Set up address of interrupt
  - Set data count
  - return
  
  Write dummy data into the SIOp register (reception start).
  SIOp register ← 0FFH

  Set the variable RCSISUBADDR to the address of the INTCSIp interrupt processing routine.
  Variable RCSISUBADDR ← #LOWW CSITXEND

  Set the variable CSISTS to the data count (1).
  ONEB CSISTS

---

5.7.21 1-character Reception End Wait Processing
Figure 5.29 shows the flowchart for 1-character reception end wait processing.

![Flowchart for 1-character Reception End Wait Processing](image)

- **SWAITRXEND**
  - Check data count
  - Is data count 0?
    - No
    - Yes
      - Read receive data count
  
  Check if the data count in the variable CSISTS is 0.
  CMP0 CSISTS

  Wait until the data count in the variable CSISTS reaches 0.

  Read the receive data count from the variable RRCVBUF.
  A register ← RRCVBUF

---
5.7.22 1-character Transfer State Check Processing

Figure 5.30 shows the flowchart for 1-character transfer state check processing.

Given below is a collection of subroutines that are used for basic continuous data communication processing. Two functions, for start and wait processing, are used in pair. Set up the parameters given below when invoking startup processing. The CSIp communication mode is automatically set up.

Continuous transmission processing
- HL register = Address of the transmit buffer
- A register = Number of data count (1 to 255)

Continuous reception processing
- HL register = Address of the buffer for storing the received data
- A register = Receive data count (1 to 255)

Continuous transmission/reception processing
- HL register = Address of the transmit buffer
- DE register = Address of the buffer for storing the received data
- A register = Transmit data count (1 to 255)
### 5.7.23 Continuous Transmission Start Processing

Figure 5.31 shows the flowchart for continuous transmission start processing.

![Flowchart](image)
Checking communication status

- Serial status register 0n (SSR0n)
  
  Reads CSIp communication status.

Symbol: SSR0n

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TSF0n</td>
<td>BFF0n</td>
<td>0</td>
<td>0</td>
<td>FEF0n</td>
<td>PEF0n</td>
<td>OVF0n</td>
</tr>
<tr>
<td>0</td>
<td>0/1</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit 6

<table>
<thead>
<tr>
<th>TSFmn</th>
<th>Communication status indication flag of channel mn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Communication is stopped or suspended.</td>
</tr>
<tr>
<td>1</td>
<td>Communication is in progress.</td>
</tr>
</tbody>
</table>

Remark:  n : channel number (n=0, 1)

Note:    SSR01 register only

5.7.24 Continuous Reception Startup Processing

Figure 5.32 shows the flowchart for continuous reception startup processing.

![Flowchart of Continuous Reception Startup Processing](image)

- **For safety countermeasures (exit if data count is 0)**
  - Variable CSISTS ← A register: Receive data count
  - Stack area ← PSW register (flag)
    - PUSH PSW

- **Put CSI in receive mode and set up end of transfer interrupts.**
  - Start reception processing by writing dummy data into the SIOp register.
    - SIOp register ← Dummy data (0FFH)
    - RCSISUBADDR ← #LOWW SRXNEXT.
  - HL register ← Stack area (data pointer)
    - POP HL

- **Wait for initiation of reception of first data.**
  - Cause a branch and exit if transfer data count (CSISTS) is 1.

- **Change to buffer empty interrupt.**
  - SIOp register ← Dummy data(0FFH)
  - PSW register ← Stack area (flag)
5.7.25 Continuous Transmission/Reception Startup Processing

Figures 5.33 and 5.34 show the flowcharts for continuous transmission/reception startup processing.

![Flowchart for Continuous Transmission/Reception Startup Processing]

SSEQTRXSUB

Transfer data count > 0?

No

Set transfer data count

Save CPU flag

Set up CSI transmission/reception mode SCHNG2TRX

Start data transmission/reception for 1st character

Set up address of interrupt processing

Yes

B

For safety countermeasures (exit if data count is 0)

Variable CSISTS ← A register: Transfer data counter

Stack area ← PSW register (flag)

PUSH PSW

Put CSI into transmission/reception mode and set up end of transfer interrupts.

A register ← [HL]: Read transmit data

SIOp register ← A register

RCSISUBADDR ← #Set to LOWW STRXEND

A

Figure 5.33 Continuous Transmission/Reception Startup Processing (1/2)
5.7.26 Continuous Transfer End Wait Processing

Figure 5.35 shows the flowchart for continuous transfer end wait processing.

**Figure 5.34 Continuous Transmission/Reception Startup Processing (2/2)**

**Figure 5.35 Continuous Transfer End Wait Processing**

Check if the data count in the variable CSISTS is 0.

- **CMP0 CSISTS**
- **Wait until the data count in the variable CSISTS reaches 0.**
### 5.7.27 Transfer End Interrupt Setup Processing

Figure 5.36 shows the flowchart for transfer end interrupt setup processing.

![Figure 5.36  Transfer End Interrupt Setup Processing](image)

Setting up the channel operating mode
- Serial mode register mn (SMR0nH, SMR0nL)
  - Interrupt source and end of transfer interrupt

#### Symbol

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKS0n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCS0n</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STS0n</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS0n0 Note</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MD0n2</td>
<td>MD0n1</td>
<td>MD0n0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bit 0

<table>
<thead>
<tr>
<th>MDmn0</th>
<th>Selection of interrupt source of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transfer end interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Buffer empty interrupt</td>
</tr>
</tbody>
</table>

Remark: n: channel number (n=0, 1)
Note: SMR01H, SMR01L only
### 5.7.28 Buffer Empty Interrupt Setup Processing

Figure 5.37 shows the flowchart for buffer empty interrupt setup processing.

![Flowchart](image)

**Figure 5.37 Buffer Empty Interrupt Setup Processing**

Setting up the channel operating mode
- Serial mode register mn (SMR0nH, SMR0nL)
  - Interrupt source and buffer empty interrupt

<table>
<thead>
<tr>
<th>Symbol: SMR0nH, SMR0nL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
</tr>
<tr>
<td><strong>MDmn0</strong></td>
</tr>
<tr>
<td>Channel n Interrupt Source Select</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Remark: n: channel number (n=0, 1)
Note: SMR01H, SMR01L registers only
5.7.29 Transmission Mode Setup Processing
Figure 5.38 shows the flowchart for transmission mode setup processing.

![Flowchart for Transmission Mode Setup Processing](image)

Save AX register

AX ← SCRmn register value

SCHNG2TXS

Save data into stack because this routine uses the AX register.
PUSH AX

Load the AX register with the value to be placed in the SCRmn register.
MOVW AX, #CTXMODE

Common processing for mode setup

5.7.30 Reception Mode Setup Processing
Figure 5.39 shows the flowchart for reception mode setup processing.

![Flowchart for Reception Mode Setup Processing](image)

Save AX register

AX ← SCRmn register value

SCHNG2TXS

Save data into stack because this routine uses the AX register.
PUSH AX

Load the AX register with the value to be placed in the SCRmn register.
MOVW AX, #CRXMODE

Go to common mode setup processing.

5.7.31 Transmission/Reception Mode Setup Processing
Figure 5.40 shows the flowchart for transmission/reception mode setup processing.

![Flowchart for Transmission/Reception Mode Setup Processing](image)

Save AX register

AX ← SCRmn register value

SCHNG2TXS

Save data into stack because this routine uses the AX register.
PUSH AX

Load the AX register with the value to be placed in the SCRmn register.
MOVW AX, #CTRXMODE

Go to common mode setup processing.
5.7.32 Common Processing for Mode Setup

Figure 5.41 shows the flowchart for common processing for mode setup.

```
SCHNG2TXS

Stop CSIp communication
STOPCSIp

Set up SCR0nH/L registers

Set up SMR0nH/L registers

Restore AX register

STARTCSIp

Stop CSIp communication.

Load the AX register with the value from the SCR0nH/L registers.

MOV !SCR0nH, A
SHLW AX, 8
MOV !SCR0nL, A

Set the parameter of the transfer end interrupt (CSMRDATA)

MOVW AX, #CSMRDATA
MOV !SMR0nH, A
SHLW AX, 8
MOV !SMR0nL, A

Restore the AX register from the stack.

POP AX

Go to CSIp communication enable processing.
```
Interrupt setting

- Interrupt mask flag register (MK0L)
  Interrupt mask setting
  Symbol: MK0L (10-pin products)

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMMK00</td>
<td>TMMK01H</td>
<td>SREMK0</td>
<td>SRMK0</td>
<td>SRMK0</td>
<td>CSIMK0</td>
<td>PMK1</td>
<td>PMK0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIMK00</td>
<td>Interrupt processing control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Enables interrupt processing.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transiting to communication stopped state

- Serial channel stop register 0 (ST0)
  Stop communication.
  Symbol: ST0

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ST01</td>
<td>ST00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0/1</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Bit n

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST0n</td>
<td>Operation stop trigger of channel n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No trigger operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Sets the SE0n bit to 0 and stop the communication operation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Setting up channel communication operation

- Serial communication operation setting register 0n (SCR0nH, SCR0nL)
  Operation mode
  Symbol: SCR0n

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXE0n</td>
<td>RXE0n</td>
<td>DAP0n</td>
<td>CKP0n</td>
<td>0</td>
<td>EOC0n</td>
<td>PTC0n1</td>
<td>PTC0n0</td>
<td>DIR0n</td>
<td>0</td>
<td>SLC0n1</td>
<td>Note</td>
<td>SLC0n0</td>
<td>0</td>
<td>1</td>
<td>DLS0n1</td>
</tr>
<tr>
<td>0/1</td>
<td>0/1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Bits 15 and 14

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Setting of operation mode of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disable communication</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reception only</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Transmission only</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Transmission/reception</td>
</tr>
</tbody>
</table>

Remark  n: Channel number (n = 0, 1)
Setting up the channel operating mode

- Serial mode register 0n (SMR0nH, SMR0nL)
  - Interrupt source and end of transmit interrupt

Symbol: SMR0nH

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKS 0n</td>
<td>CCS 0n</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STS 0n</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Symbol: SMR0nL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SIS 0n</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MD 0n2</td>
<td>MD 0n1</td>
<td>MD 0n0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bit 0

<table>
<thead>
<tr>
<th>MD0n0</th>
<th>Selection of interrupt source of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transfer end interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Buffer empty interrupt</td>
</tr>
</tbody>
</table>

Remark  n: Channel number (n = 0, 1)

5.7.33 CSIp Communication Enable Processing

Figure 5.41 shows the flowchart for CSIp communication enable processing.

STARTCSIp

CSIp interrupt clear

Clear the CSIp interrupt request.
CSIIp bit ← 0: Clear interrupt request.

Enable CSIp interrupts

Mask off CSIp interrupts (enable interrupts).
CSIMKp bit ← 0: Mask off interrupts

Put CSIp in communication standby state

Set the SSmn bit to put the CSIp into communication standby state.
SSmn bit ← 1: Set SEmn bit to 1 for standby state.

return

Figure 5.41 CSIp Communication Enable Processing

Transiting to communication standby state
- Serial channel startup register 0 (SS0)
  Start operation.

Symbol: SS0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SS01</td>
<td>SS00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0/1</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Bits 3 to 0

<table>
<thead>
<tr>
<th>SSmn</th>
<th>Operation start trigger of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No trigger operation</td>
</tr>
<tr>
<td>1</td>
<td>Sets the SEmn bit to 1 and enters the communication wait status.</td>
</tr>
</tbody>
</table>

Remark: n: channel number (n=0, 1)

Interrupt setting
- Interrupt request flag register (IF0L)
  Clear the interrupt request flag
- Interrupt mask flag register (MK0L)
  Clear the interrupt mask

Symbol: IF0L (10-pin products)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMIF00</td>
<td>TMIF01H</td>
<td>SREIF0</td>
<td>SRIF0</td>
<td>STIF0</td>
<td>CSIIF00</td>
<td>IICIF00</td>
<td>PIF1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>CSIIF00</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No interrupt request signal is generated</td>
</tr>
<tr>
<td>1</td>
<td>Interrupt request is generated, interrupt request status</td>
</tr>
</tbody>
</table>

Symbol MK0L (10-pin products)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMMK00</td>
<td>TMMK01H</td>
<td>SREMK0</td>
<td>SRMK0</td>
<td>SRMK0</td>
<td>CSIMK00</td>
<td>IICMK00</td>
<td>PMK1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>CSIMK00</th>
<th>Interrupt processing control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enables interrupt processing.</td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
</tr>
</tbody>
</table>

5.7.34 CSIp Communication Termination Processing

Figure 5.42 shows the flowchart for CSIp communication termination processing.

![Flowchart of CSIp Communication Termination Processing]

- **STOPCSIp**: Set the SSmn bit to put the CSIp into communication stopped state. STmn bit ← 1: Set SEmn bit to 0 to stop the CSIp.
- **Disable CSIp interrupts**: Set the CSIp interrupt mask (disable interrupts). CSIMKp bit ← 1: Set interrupt mask.
- **Clear CSIp interrupt**: Clear the CSIp interrupt request. CSIIFp bit ← 0: Clear interrupt request.
- **return**: Put CSIp in communication stopped state. STmn bit ← 1: Set SEmn to 0 and enters the communication wait status.

*Remark: n: channel number (n=0, 1)*


```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation start trigger of channel n</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST0n</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No trigger operation</td>
</tr>
<tr>
<td>1</td>
<td>Sets the SEmn to 0 and enters the communication wait status.</td>
</tr>
</tbody>
</table>
```
5.7.35 CSIp Interrupt Startup Processing

Figure 5.43 shows the flowchart for CSIp interrupt startup processing.

![Flowchart](image)

**Caution:** For details on the register setup procedures, refer to RL78/G10 User's Manual: Hardware.
5.7.36 1-character Transfer End Interrupt Processing

Figure 5.44 shows the flowchart for 1-character transfer end interrupt processing.

```
           CSITXEND
                |
  Read receive data
                |
  Store receive data
                |
  Data count ← 0
                |
            RETI
```

Read the data received through CSIp into the A register (has no meaning in transmission mode).

```
MOV A, SIOp
```

Store the received data in the buffer area (has no meaning in transmission mode).

```
MOV RRCVBUF, A
```

Set the data count in variable CSISTS to 0 (end of communication).

```
CLRB CSISTS
```

Figure 5.44 1-character Transfer End Interrupt Processing

5.7.37 1-character Transfer End Interrupt Processing in Continuous Reception Mode

Figure 5.45 shows the flowchart for 1-character transfer end interrupt processing in continuous reception mode.

```
           SRXNEXT
                |
  Store receive data
                |
  Update parameters
                |
          No
                |
        Remaining data present?
                |
          Yes
                |
          No
                |
        Remaining data count >1?
                |
          Yes
                |
        Start reception for next data
                |
          No
                |
  Set up transfer end interrupt SSETENDINT
                |
            RETI
```

Store the data received through CSIp in the buffer.

```
MOV A, SIOp
MOV [HL], A
```

Update the pointer and counter.

```
DEC CSISTS
INCW HL
```

Terminate processing if data count (variable CSISTS) is 0.

If the data count (variable CSISTS) > 1, write dummy data (0FFH) into SIOp and start the reception processing for the next data following the data that is being received.

```
MOV SIOp, 0FFH
```

If the data count (variable CSISTS) = 1, set the interrupt type to end of reception interrupt for the data that is being received.

```
SSETENDINT
```

Figure 5.45 1-character Transfer End Interrupt Processing (Continuous Reception Mode)
5.7.38 Buffer Empty Interrupt Processing in Continuous Transmission Mode

Figure 5.46 shows the flowchart for buffer empty interrupt processing in continuous transmission mode.

![Flowchart for Buffer Empty Interrupt Processing](image)

- **STXNEXT**
  - Remaining data count > 1? (Yes/No)
  - **Yes**: Update parameters
  - **No**: Update the pointer and counter.
    - INCW HL
    - DEC CSISTS

- **Read next transmit data**
  - MOV A, [HL]

- **Write transmit data**
  - MOV SIOp, A

- **RETI**
  - If the data count (variable CSISTS) = 1, set the interrupt type to transfer end interrupt.
  - Place the address of the transmission completion processing routine into the variable RCSISUBADDR.
    - Variable RCSISUBADDR ← #LOWW STXEND

**Figure 5.46 Buffer Empty Interrupt Processing (Continuous Transmission Mode)**

5.7.39 Transmission End Interrupt Processing in Continuous Transmission Mode

Figure 5.47 shows the flowchart for transmission end interrupt processing in continuous transmission mode.

![Flowchart for Transmission End Interrupt Processing](image)

- **STXEND**
  - Data count ← 0
    - Set the data count in variable CSISTS to 0 (communication complete).
    - CLR B CSISTS

- **RETI**
  - **Set up transfer end interrupt SSETENDINT**

**Figure 5.47 Transmission End Interrupt Processing (Continuous Transmission Mode)**
5.7.40 Buffer Empty Interrupt Processing in Continuous Transmission Mode

Figure 5.48 shows the flowchart for buffer empty interrupt processing in continuous transmission mode.

![Flowchart for Buffer Empty Interrupt Processing](image)

Figure 5.48 Buffer Empty Interrupt Processing (Continuous Transmission Mode)

- **STRXNEXT**
- Store receive data
- Update receive data pointer
- Remaining data count >1?
  - Yes
    - Update parameters
    - Transmit next transmit data
    - RETI
  - No
    - Cause a branch if data count (variable CSISTS) = 1.
- End of transfer interrupt setup SSETENDINT
- Change address of interrupt processing
- RETI

- Store the data received through CSIp in the buffer.
  - MOV A, SIOp
  - MOV [DE], A

- Update receive data pointer (DE register).
  - INCW DE

- If the data count (variable CSISTS) = 1, set the interrupt type to end of transfer interrupt.
- Place the address of the transmission/reception completion processing routine into the variable RCSISUBADDR.
  - Variable RCSISUBADDR ← #LOWW STRXEND

5.7.41 Transfer End Interrupt Processing in Continuous Transmission/Reception Mode

Figure 5.49 shows the flowchart for transfer end interrupt processing in continuous transmission/reception mode.

![Flowchart for Transfer End Interrupt Processing](image)

Figure 5.49 Transfer End Interrupt Processing (Continuous Transmission/Reception Mode)

- **STRXEND**
- Store receive data
- Data count ← 0
- RETI

- Store the data received through CSIp in the buffer.
  - MOV A, SIOp
  - MOV [DE], A

- Set the data count in variable CSISTS to 0 (end of communication).
  - CLR B CSISTS
6. Changing the Channel to be Used

6.1 Definition File

The channel to be used for CSI master communication is defined in an include file (DEV&CSI_CH.inc). Note that the available channels vary with the device.

6.2 Major Items of the Definition File

The include file defines the following constants that the user can modify. Never modify the value of the other constants. The CPU clock frequency is defined to refer to the clock frequency of the CPU that is actually used in the user system. The user cannot use this definition to change the clock frequency of the CPU.

- **CPU clock frequency (CLKFREQ) in kHz**: The initial value is 20000 (20 MHz).
- **CSI communication speed (BAUDRATE) in kbps**: The initial value is 1000 (1 Mbps).
- **Microcontroller to be used**: The initial value is R5F10Y16ASP.
- **CSI channel to be used**: The initial value is CSI00.

6.3 Changing the Transfer Rate

The transfer rate is defined as shown below. For a CPU clock frequency of 20 MHz, the user can change the transfer rate between 200 kbps and 2000 kbps by changing "1000" to a desired value between 200 and 2000. It is necessary to modify the program to use a transfer rate outside this value range.

```plaintext
;************************************************************************
; Communication definitions
;************************************************************************
CLKFREQ  EQU   20000   ; kHz
BAUDRATE  EQU   1000   ; kbps
```

6.4 Changing the Microcontroller to be Used

When changing the microcontroller to be used, create a new project with CubeSuite+ and specify the desired device in the project. For details, refer to RL78 Family CubeSuite+ Startup Guide (R01AN1232E) Application Note.

The microcontroller to be used is defined as shown below. Only the line that has no leading semicolon (';') is valid. To change the device to be used, append a semicolon to the beginning of the currently valid line and delete the leading semicolon at the beginning of the line defining the desired device.

```plaintext
;************************************************************************
; device select
;************************************************************************
$SET( R5F10Y16 )   ; 10 pins
$SET( R5F10Y14 )   ; 10 pins
$SET( R5F10Y44 )   ; 16 pins
$SET( R5F10Y46 )   ; 16 pins
$SET( R5F10Y47 )   ; 16 pins
```

Defines the microcontroller that is in use.
6.5 Changing the Channel to be Used

The channel to be used is defined as shown below. Select the desired channel from the channels that are available for the microcontroller to be used and delete the leading semicolon (‘;’) from the line defining the desired channel. At the same time, append a semicolon at the beginning of the line for the channel that had been selected until now. **The program will not run normally if two or more channels are selected.**

<table>
<thead>
<tr>
<th>Definition for 10-pin products.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{IF( R5F10Y16 : R5F10Y14 )}$</td>
</tr>
<tr>
<td>$\text{SET( CS100 )} \quad \text{CS100 is selected}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition for 16-pin products.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ELSE}$</td>
</tr>
<tr>
<td>$\text{SET( CS100 )} \quad \text{CS100 is selected}$</td>
</tr>
<tr>
<td>$\text{SET( CS101 )} \quad \text{CS101 is not selected now}$</td>
</tr>
</tbody>
</table>

$\text{ENDIF}$
6.6 Reference

Once the channel to be used is defined, the constants to be used by the program are set to the values that conform to
the newly defined channel by the definitions given below, so that the user need not be aware of the channel he or she
is to use.

Port initialization is accomplished by directly referencing the microcontroller and channel definitions that are
provided separately from these definitions.

$IFDEF (CSI00 )
SMR0nH EQU SMR00H  ; Serial mode register (High)
SMR0nL EQU SMR00L  ; Serial mode register (Low)
SCR0nH EQU SCR00H  ; Serial communication operation setting register (High)
SCR0nL EQU SCR00L  ; Serial communication operation setting register (Low)
SDR0nH EQU SDR00H  ; Serial data register (High)
SIOp EQU SIO00  ; Serial data register (Low)
SSROn EQU SSRO0  ; Serial status register
SIR0n EQU SIR00  ; Serial flag clear trigger register
TRGONn EQU 00000001B  ; for trigger SS00/ST00
SOEON EQU TRGONn  ; for turn on SOE00
SOEOFF EQU 11111110B  ; for turn off SOE00
SOHIGH EQU TRGONn  ; for set SO bit
PM_SCKp EQU PM0.2  ; port mode register bit for SCK
PM_SI p EQU PM0.1  ; port mode register bit for SI
PM_SO p EQU PM0.0  ; port mode register bit for SO
P_SCKp EQU PO.2  ; port register for SCK
P_SI p EQU PO.1  ; port register for SI
P_SO p EQU PO.0  ; port register for SO
CSIIFp EQU CSIIF00  ; interrupt request flag
CSIMKp EQU CSIMK00  ; interrupt mask register
$ENDIF
7. Sample Code

The sample code is available on the Renesas Electronics Website.

8. Documents for Reference

User’s Manual:
- RL78/G10 User's Manual: Hardware (R01UH0384E)
- RL78 Family CubeSuite+ Startup Guide (R01AN1232E)
- RL78/G10 Serial Array Unit (CSI Slave Communication) (R01AN1461E)

(The latest versions of the documents are available on the Renesas Electronics Website.)

Technical Updates/Technical Brochures

(The latest versions of the documents are available on the Renesas Electronics Website.)

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>2013.03.11</td>
<td>-</td>
<td>First Edition</td>
</tr>
<tr>
<td>2.00</td>
<td>2014.09.30</td>
<td>9</td>
<td>e2studio and IAR information added in Table 2.1</td>
</tr>
<tr>
<td>2.10</td>
<td>2022.09.30</td>
<td>9</td>
<td>Delete IAR information from Table 2.1</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.

Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, or other disposal of any products incorporating Renesas Electronics products, if required.

5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; telecommunications; audio and visual equipment; home electronic appliances; machine tools; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implants; etc.), or may cause serious property damage (space system; undersea reefs; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RenESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INUNVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL LIABILITY OR RESPONSIBILITY FOR ANY SECURITY BREACH OR ANY OTHER LOSS ARISING FROM OR RELATING TO ANY VULNERABILITY ISSUE.

8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

11. Renesas Electronics products and technologies shall not be used or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Corporate Headquarters
TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information
For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks
Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

© 2022 Renesas Electronics Corporation. All rights reserved