RX220 Group and RX21A Group
Asynchronous SCI Communication Using the DTCa

Introduction
This application note describes methods for performing asynchronous serial communication using the data transfer controller (DTC) and serial communications interface (SCI) peripheral modules provided by the RX220 Group and RX21A Group microcontrollers.

Target Device
- RX220 Group: 100-pin versions with ROM capacities of 32 to 256 KB
- RX21A Group: 100-pin versions with ROM capacities of 256 to 512 KB

When using the code presented in this application note with a different microcontroller, modify the code according to the specifications of that microcontroller and test the code thoroughly.
## Contents

1. Specifications ................................................................. 3
2. Confirmed Operating Condition ........................................ 4
3. Reference Application Notes ............................................ 5
4. Hardware ........................................................................ 6
5. Software ......................................................................... 7
6. Sample Code .................................................................... 27
7. Usage Notes ..................................................................... 27
8. Reference Documents ...................................................... 28
1. Specifications

This system performs serial communication using the SCI. The transmit data is set up in advance in the RAM transmit data storage area and transmitted using the DTC. The receive data is stored in the RAM receive data storage area using the DTC.

Serial communication is started when a falling edge is detected on the interrupt request pin (IRQ1).

- Transfer rate: 38,400 bps
- Data length: 8 bits, LSB first
- Stop bits: 1 bit
- Parity: None
- Hardware flow control: None

Table 1.1 lists the peripheral functions used and their applications and figure 1.1 presents a block diagram.

Table 1.1 Peripheral Functions Used and Their Applications

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCI</td>
<td>Asynchronous serial communication</td>
</tr>
<tr>
<td>DTC</td>
<td>Transfers the SCI1 receive data to RAM</td>
</tr>
<tr>
<td></td>
<td>Transfers the transmit data in RAM to the SCI1</td>
</tr>
<tr>
<td>IRQ1</td>
<td>Serial communication start trigger</td>
</tr>
</tbody>
</table>

![Figure 1.1 Block Diagram](image)
2. **Confirmed Operating Condition**

The sample code accompanying this application note has been run and confirmed under the conditions below.

(1) **For the RX220**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MCU used</strong></td>
<td>R5F52206BDFP (RX220 Group)</td>
</tr>
</tbody>
</table>
| **Operating frequency**     | Main clock: 20.0 MHz  
                             System clock (ICLK): 20 MHz (1 period of the main clock)  
                             Peripheral module clock B (PCLKB): 20 MHz (1 period of the main clock) |
| **Operating voltage**       | 5.0 V: Supplied from the E1 emulator                                       |
| **Integrated development environment** | Renesas Electronics Corporation  
                             High-performance Embedded Workshop Version 4.09.01.007 |
| **C compiler**              | Renesas Electronics Corporation  
                             C/C++ Compiler Package for RX Family V.1.02 Release 01 |
|                             | Compiler option  
                             -cpu=rx200 -output=obj="$(CONFIGDIR)$(FILELEAF).obj" -debug -nologo |
|                             | (The integrated development environment default settings are used.)          |
| **iodefine.h version**      | Version 1.0A                                                                |
| **Endian order**            | Little endian                                                              |
| **Operating mode**          | Single-chip mode                                                           |
| **Processor mode**          | Supervisor mode                                                            |
| **Sample code version**     | Version 1.00                                                               |
| **Board used**              | Renesas Starter Kit for RX220 (Product number: R0K505220S000BE)            |
(2) For the RX21A

Table 2.2 Confirmed Operating Condition

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU used</td>
<td>R5F521A8BDFP (RX21A Group)</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>Main clock: 20.0 MHz</td>
</tr>
<tr>
<td></td>
<td>PLL: 100 MHz (2 period of the main clock and multiplied by 10)</td>
</tr>
<tr>
<td></td>
<td>System clock (ICLK): 50 MHz (2 period of the PLL)</td>
</tr>
<tr>
<td></td>
<td>Peripheral module clock B (PCLKB): 25 MHz (4 period of the PLL)</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3 V: Supplied from the E1 emulator</td>
</tr>
<tr>
<td>Integrated development</td>
<td>Renesas Electronics Corporation</td>
</tr>
<tr>
<td>environment</td>
<td>High-performance Embedded Workshop Version 4.09.01.007</td>
</tr>
<tr>
<td>C compiler</td>
<td>Renesas Electronics Corporation</td>
</tr>
<tr>
<td></td>
<td>C/C++ Compiler Package for RX Family V.1.02 Release 01</td>
</tr>
<tr>
<td></td>
<td>Compiler option</td>
</tr>
<tr>
<td></td>
<td>-cpu=rx200 -output=obj=&quot;$(CONFIGDIR)$(FILELEAF).obj&quot; -debug --nologo</td>
</tr>
<tr>
<td></td>
<td>(The integrated development environment default settings are used.)</td>
</tr>
<tr>
<td>iodefine.h version</td>
<td>Version 1.00</td>
</tr>
<tr>
<td>Endian order</td>
<td>Little endian</td>
</tr>
<tr>
<td>Operating mode</td>
<td>Single-chip mode</td>
</tr>
<tr>
<td>Processor mode</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>Sample code version</td>
<td>Version 1.00</td>
</tr>
<tr>
<td>Board used</td>
<td>HSB Series microcontroller board (Hokuto Denshi Co., Ltd.)</td>
</tr>
<tr>
<td></td>
<td>(Catalog number: HSBZRZ21AP-B)</td>
</tr>
<tr>
<td></td>
<td>Serial connection board</td>
</tr>
<tr>
<td></td>
<td>• RS-232C serial connector (D-sub 9-pin connector)</td>
</tr>
<tr>
<td></td>
<td>• RS-232C transceiver (MAX3232CPE)</td>
</tr>
</tbody>
</table>

3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- RX220 Group Initial Setting Rev.1.00 (R01AN1494EJ0100_RX220)
- RX21A Group Initial Setting Rev.1.00 (R01AN1486EJ0100_RX21A)

The sample code with this application note uses the initialization from the application note noted above. The revision number is the one that was current when this application note was written.

If there is a more recent version, you should replace this code with the most recent version. The most recent version can be downloaded from the Renesas Electronics Corporation web site.
4. Hardware

4.1 Hardware Structure

Figure 4.1 shows a sample RX220 connection and figure 4.2 shows a sample RX21A connection.

![Figure 4.1 Sample RX220 Connection](image1)

![Figure 4.2 Sample RX21A Connection](image2)

4.2 Pins Used

Table 4.1 lists the pins used and their functions.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P31/IRQ1</td>
<td>Input</td>
<td>Communication start switch input</td>
</tr>
<tr>
<td>P15/RXD1</td>
<td>Input</td>
<td>SCI1 receive data input</td>
</tr>
<tr>
<td>P16/TXD1</td>
<td>Output</td>
<td>SCI1 transmit data output</td>
</tr>
</tbody>
</table>
5. Software

This sample code automatically processes SCI1 transmit and receive operations by using the DTC module. The communication start switch is pressed SCI1 transmit and receive operations are started.

When transmission is enabled, a TXI1 interrupt request occurs and this acts as the DTC start factor. The data in the transmit data storage area is transferred to the TDR register by the DTC module and transmitted.

When a reception completes, an RXI1 interrupt request occurs and this acts as the DTC start factor. The receive data is transmitted to the receive data storage area by the DTC module.

When 256 transfers of transmit data complete, a TXI1 interrupt request occurs. At this point the TXI1 interrupt is disabled and the TEI1 interrupt is enabled.

When 256 transfers of receive data complete, an RXI1 interrupt occurs. At this point SCI1 reception and the RXI1 interrupt are disabled and the reception complete flag is set to 1.

When the 256-byte transmission has completed, a TEI1 interrupt occurs. At this point SCI1 transmission and the TEI1 interrupt are disabled and the transmission complete flag is set to 1.

The settings for the peripheral functions used are shown below.

SCI1
- Serial communication method: Asynchronous
- Transfer rate: 38400 bps
- Clock source: PCLKB
- Data length: 8 bits
- Stop bits: 1 bit
- Parity function: No parity
- Data transfer method: LSB first
- Interrupts: 
  - Reception error interrupt (ERI1) enabled
  - Receive data full interrupt (RXI1) enabled
  - Transmit data empty interrupt (TXI1) enabled
  - Transmit complete interrupt (TEI1) enabled

DTC
- Start factor: TXI1 and RXI1 interrupt requests
- DTC addressing mode: Full addressing mode

[DTC Transfer Settings for Activation by a TXI1 Interrupt Request]
- Transfer mode: Normal transfer mode
- Transfer source addressing mode: SAR register incremented after transfer
- Transfer source address: RAM (transmit data storage area start address)
- Transfer target addressing mode: DAR register holds fixed address
- Transfer target address: SCI1.TDR register
- Data transfer unit: 8 bits
- Transfer count: 256 transfers
- Chained transfers: Disabled
- Interrupts: The CPU is interrupted when the specified data transfer completes.
[DTC Transfer Settings for Activation by a RXI1 Interrupt Request]

- Transfer mode : Normal transfer mode
- Transfer source addressing mode : SAR register holds fixed address
- Transfer source address : SCI1.RDR register
- Transfer target addressing mode : DAR register incremented after transfer
- Transfer target address : RAM (receive data storage area start address)
- Data transfer unit : 8 bits
- Transfer count : 256 transfers
- Chained transfers : Disabled
- Interrupts : The CPU is interrupted when the specified data transfer completes.

IRQ1 input pin
- Detection method : Falling edge detection
- Digital filter : Disabled
- Interrupt : Not used

### 5.1 Operation Overview

#### 5.1.1 Transmit Operation

1. **Initialization**
   After initialization, the sample code waits for a communication start switch input.

2. **Communication start switch input detection**
   When a communication start switch input is detected, The IRQ1 IR flag is set to 0. When the sample code has verified that transmission and reception have competed by checking the transmission complete and reception complete flags, the transmission complete flag is set to 0 (transmission in progress). The DTC transfer source address and transfer count are set and DTC activation is enabled. The SCI1.SCR.TEI, TIE, RIE, TE, and RE bits are set to 1 at the same time to enable transmission and reception. The TXI1 interrupt IR flag goes to 1 when the SCI1.SCR.TIE and TE bits are set to 1 at the same time.

3. **Data transfer start**
   When the TXI1 interrupt is enabled, the TXI1 interrupt IR flag goes to 0 when the DTC is activated. The first byte of the transmit data is transferred from the RAM transmit data storage area to the SCI1.TDR register.

4. **Data transmission start**
   The TXI1 interrupt IR flag is set to 1 by the transfer of data from the SCI1.TDR register to the SCI1.TSR register and the first byte of transmit data is output from the TXD1 pin. The DTC module is activated by the TXI1 interrupt and the second byte of data is transferred.

5. **TXI1 interrupt**
   When the 256th data transfer completes, the CPU accepts a TXI1 interrupt request. In TXI1 interrupt handling, the TXI1 interrupt is disabled and the TEI1 interrupt is enabled.

6. **TEI1 interrupt**
   When the last bit of the 256th byte is transmitted, the SCI1.TDR register is not updated and therefore a TEI1 interrupt request is generated. In TEI1 interrupt handling, transmission is disabled and the TEI1 interrupt is disabled. The transmission complete flag is set to 1 (transmission complete).
   After this, this sequence is repeated starting at item (2) above.
Figure 5.1 shows the timing chart for the transmission operation.
5.1.2 Receive Operation

(1) Initialization
After initialization, the sample code waits for a communication start switch input.

(2) Communication start switch input detection
When a communication start switch input is detected, the IRQ1 IR flag is set to 0. When the sample code has verified that transmission and reception have completed by checking the transmission complete and reception complete flags, the reception complete flag is set to 0 (reception in progress). The DTC transfer target address and transfer count are set and DTC activation is enabled. The SCI1.SCR.TEIE, TIE, RIE, TE, and RE bits are set to 1 at the same time to enable transmission and reception and the RXI1 interrupt is enabled.

(3) Data reception termination
When the reception of one byte of data completes, the data is transferred from the SCI1.RSR register to the SCI1.RDR register and the RXI1 interrupt IR flag is set to 1.

(4) Data transfer start
The DTC module is started by an RXI1 interrupt request and the RXI1 interrupt IR flag becomes 0. One byte of receive data is transferred from the SCI1.RDR register to the RAM receive data storage area.

(5) RXI1 interrupt
When 256th data transfer completes, the CPU accepts the RXI1 interrupt request. Reception is disabled and the RXI1 interrupt is disabled in RXI1 interrupt handling. The reception complete flag is set to 1 (reception complete). After this, this sequence is repeated starting at item (2) above.

Figure 5.2 shows the timing chart for the receive operation.

---

**Figure 5.2  Reception Operation Timing Chart**
Notes on Embedding in an Actual System

The following phenomenon requires care when using the sample code provided with this application note in an actual system.

- This sample code may not operate correctly if an interrupt used by this application note's code is delayed for an extended period due to, for example, the handling of other interrupts.

### 5.2 Section Structure

Table 5.1 lists the section information modified in this sample code.

Refer to the latest version of the RX Family C/C++ Compiler Package User's Manual for the methods for adding, modifying, and deleting sections.

**Table 5.1 Section Information Modified by this Sample Code**

<table>
<thead>
<tr>
<th>Section Name</th>
<th>Modification</th>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTC_SECTION</td>
<td>Added</td>
<td>0000 3000h</td>
<td>DTC vector table</td>
</tr>
</tbody>
</table>
5.3 File Composition
Table 5.2 lists the files used for the sample code. Note that the files generated automatically by the integrated development environment are not shown.

Table 5.2 File Composition

<table>
<thead>
<tr>
<th>Target Device</th>
<th>File Name</th>
<th>Overview</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>main.c</td>
<td>Main processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>device_cfg.h</td>
<td>Device configuration header file</td>
<td></td>
</tr>
<tr>
<td>RX220</td>
<td>r_init_stop_module.c</td>
<td>Stops peripheral functions operating after a reset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_stop_module.h</td>
<td>r_init_stop_module_.c header file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_non_existent_port.c</td>
<td>Nonexistent port initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_non_existent_port.h</td>
<td>r_init_non_existent_port.c header file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_clock.c</td>
<td>Clock initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_clock.h</td>
<td>r_init_clock.c header file</td>
<td></td>
</tr>
<tr>
<td>RX21A</td>
<td>r_init_stop_module.c</td>
<td>Stops peripheral functions operating after a reset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_stop_module.h</td>
<td>r_init_stop_module_.c header file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_non_existent_port.c</td>
<td>Nonexistent port initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_non_existent_port.h</td>
<td>r_init_non_existent_port.c header file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_clock.c</td>
<td>Clock initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_init_clock.h</td>
<td>r_init_clock.c header file</td>
<td></td>
</tr>
</tbody>
</table>

5.4 Option-Setting Memory
Table 5.3 lists the states of the option settings memory used by the sample code. Set these locations to appropriate values for your user system as required.

Table 5.3 Option Settings Memory Set by the Sample Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFS0</td>
<td>FFFF FF8Fh to FFFF FF8Ch</td>
<td>FFFF FFFFh</td>
<td>Stops IWDT after a reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stops WDT after a reset</td>
</tr>
<tr>
<td>OFS1</td>
<td>FFFF FF8Bh to FFFF FF88h</td>
<td>FFFF FFFFh</td>
<td>Disables voltage monitoring resets after a reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Disables HOCCO oscillation after a reset</td>
</tr>
<tr>
<td>MDES</td>
<td>FFFF FF83h to FFFF FF80h</td>
<td>FFFF FFFFh</td>
<td>Little endian</td>
</tr>
</tbody>
</table>

5.5 Constants
Table 5.4 lists the constants used in the sample code.

Table 5.4 Constants Used in the Sample Code

<table>
<thead>
<tr>
<th>Constant</th>
<th>Set value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUF_SIZE</td>
<td>256</td>
<td>Transmit and receive data storage area size</td>
</tr>
<tr>
<td>DTC_CNT</td>
<td>BUF_SIZE</td>
<td>DTC transfer count</td>
</tr>
<tr>
<td>SCI_BIT_RATE</td>
<td>15</td>
<td>SCI1.BRR register setting value (when PCLKB is 20 MHz)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCI1.BRR register setting value (when PCLKB is 25 MHz)</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>SCI1.BRR register setting value (when PCLKB is 25 MHz)</td>
</tr>
</tbody>
</table>
5.6 Structures and Unions

Figure 5.3 shows the structures and unions used in the sample code.

/* **** DTC transfer Information **** */
#pragma bit_order left /* Bit field order specification: members are allocated starting with the high order bits. */
#pragma unpack /* Structure member boundary alignment: alignment is according to the member type. */
struct st_dtc_full_t
{
  union
  {
    uint32_t  LONG;
    struct
    {
      uint32_t  MRA_MD     :2;
      uint32_t  MRA_SZ     :2;
      uint32_t  MRA_SM     :2;
      uint32_t               :2;
      uint32_t  MRB_CHNE   :1;
      uint32_t  MRB_CHNS   :1;
      uint32_t  MRB_DISEL  :1;
      uint32_t  MRB_DTS    :1;
      uint32_t  MRB_DM     :2;
      uint32_t               :2;
      uint32_t                     :16;
    } BIT;
  } MR;
  void *  SAR;
  void *  DAR;
  struct
  {
    uint32_t  CRA:16;
    uint32_t  CRB:16;
  } CR;
};
#pragma packoption /* End structure member boundary alignment. */
#pragma bit_order /* End bit field order specification. */
5.7 Variables

Table 5.5 lists the static variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>static uint8_t</td>
<td>trn_end_flag</td>
<td>Transmission complete flag</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Transmission in progress</td>
<td>Excep_SCI1_TEI1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Transmission complete</td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>rcv_end_flag</td>
<td>Reception complete flag</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Reception in progress</td>
<td>Excep_SCI1_RXI1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Reception complete</td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>trnbuf[BUF_SIZE]</td>
<td>Transmit data storage area</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dtc_init sci1_start</td>
</tr>
<tr>
<td>static uint8_t</td>
<td>rcvbuf[BUF_SIZE]</td>
<td>Receive data storage area</td>
<td>dtc_init sci1_start</td>
</tr>
<tr>
<td>static struct</td>
<td>dtc_info_rxi1</td>
<td>RXI1 DTC transfer information</td>
<td>dtc_init sci1_start</td>
</tr>
<tr>
<td>st_dtc_full_t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static struct</td>
<td>dtc_info_txi1</td>
<td>TXI1 DTC transfer information</td>
<td>dtc_init sci1_start</td>
</tr>
<tr>
<td>st_dtc_full_t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static void *</td>
<td>dtc_vect_table [256]</td>
<td>DTC vector table</td>
<td>dtc_init sci1_start</td>
</tr>
</tbody>
</table>

5.8 Functions

Table 5.6 shows the functions used in the sample code.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>port_init</td>
<td>Port initialization</td>
</tr>
<tr>
<td>R_INIT_StopModule</td>
<td>Stops peripheral modules operating after a reset</td>
</tr>
<tr>
<td>R_INIT_NonExistentPort</td>
<td>Nonexistent port initialization</td>
</tr>
<tr>
<td>R_INIT_Clock</td>
<td>Clock initialization</td>
</tr>
<tr>
<td>peripheral_init</td>
<td>Peripheral function initialization</td>
</tr>
<tr>
<td>sci1_init</td>
<td>SCI1 initialization</td>
</tr>
<tr>
<td>dtc_init</td>
<td>DTC initialization</td>
</tr>
<tr>
<td>irq_init</td>
<td>IRQ initialization</td>
</tr>
<tr>
<td>sci1_start</td>
<td>Start SCI1 transmission</td>
</tr>
<tr>
<td>Excep_SCI1_RXI1</td>
<td>SCI1 receive data full interrupt handler</td>
</tr>
<tr>
<td>Excep_SCI1_TXI1</td>
<td>SCI1 transmit data empty interrupt handler</td>
</tr>
<tr>
<td>Excep_SCI1_TEI1</td>
<td>SCI1 transmission complete interrupt handler</td>
</tr>
<tr>
<td>Excep_SCI1_ERI1</td>
<td>SCI1 receive error interrupt handler</td>
</tr>
</tbody>
</table>
5.9 Function Specifications
This section lists the specifications of the functions in the sample code.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>Overview</td>
<td>Main processing</td>
</tr>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>After initialization, starts an SCI1 communication operation when a communication start switch input is detected.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>None</td>
</tr>
</tbody>
</table>

| port_init   | Port initialization                                                         |
| Overview    | Port initialization                                                         |
| Header      | None                                                                        |
| Declaration | static void port_init(void)                                                |
| Description | Initializes the ports.                                                      |
| Arguments   | None                                                                        |
| Return values | None                      |

<p>| R_INIT_StopModule | Stops peripheral modules operating after a reset                           |
| Overview          | Stops peripheral modules operating after a reset                           |
| Header            | r_init_stop_module.h                                                       |
| Declaration        | void R_INIT_StopModule(void)                                              |
| Description        | Performs the settings required to transition to the module stop state.    |
| Arguments          | None                                                                        |
| Return values      | None                                                                        |
| Remarks            | In this sample code, the transition to the module stop state is not performed. See the &quot;RX220 Group Initialization Example, Revision 1.00&quot; and &quot;RX21A Group Initialization Example, Revision 1.00&quot; application notes for details on this function. |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Overview</th>
<th>Header</th>
<th>Declaration</th>
<th>Description</th>
<th>Arguments</th>
<th>Return values</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_INIT_NonExistentPort</td>
<td>Nonexistent port initialization</td>
<td>r_init_non_existent_port.h</td>
<td>void R_INIT_NonExistentPort(void)</td>
<td>Performs port direction register initialization for non-existent port pins for microcontroller versions that have fewer than 100 pins.</td>
<td>None</td>
<td>None</td>
<td>In this sample code, settings for the 100-pin versions (PIN_SIZE = 100) are performed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>After this function is called, if an application writes in byte units to a PDR or PODR register that includes ports that do not exist, the application should set the direction control bit to 1 and the port output data storage bit to 0 for those nonexistent ports.</td>
<td></td>
<td></td>
<td>See the &quot;RX220 Group Initialization Example, Revision 1.00&quot; and &quot;RX21A Group Initialization Example, Revision 1.00&quot; application notes for details on this function.</td>
</tr>
<tr>
<td>R_INIT_Clock</td>
<td>Clock initialization</td>
<td>r_init_clock.h</td>
<td>void R_INIT_Clock(void)</td>
<td>Initializes the clocks.</td>
<td>None</td>
<td>None</td>
<td>In the RX220 sample code, processing in which the main clock is used as the system clock and the subclock is not used is selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In the RX21A sample code, processing in which the PLL circuit is used as the system clock and the subclock is not used is selected.</td>
<td></td>
<td></td>
<td>See the &quot;RX220 Group Initialization Example, Revision 1.00&quot; and &quot;RX21A Group Initialization Example, Revision 1.00&quot; application notes for details on this function.</td>
</tr>
<tr>
<td>peripheral_init</td>
<td>Peripheral function initialization</td>
<td></td>
<td>static void peripheral_init(void)</td>
<td>Initializes the used peripheral modules.</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
sci1_init

Overview: SCI1 initialization
Header: None
Declaration: static void sci1_init(void)
Description: Initializes the SCI1 module.
Arguments: None
Return values: None

dtc_init

Overview: DTC initialization
Header: None
Declaration: static void dtc_init(void)
Description: Initializes the DTC module.
Arguments: None
Return values: None

irq_init

Overview: IRQ initialization
Header: None
Declaration: static void irq_init(void)
Description: Initializes the IRQ1 module.
Arguments: None
Return values: None

sci1_start

Overview: Start SCI1 transmission
Header: None
Declaration: static void sci1_start(void)
Description: Starts an SCI1 communication operation
Arguments: None
Return values: None

Excep_SCI1_RXI1

Overview: SCI1 receive data full interrupt handler
Header: None
Declaration: static void Excep_SCI1_RXI1(void)
Description: Stops reception and disables the RXI1 interrupt. Sets the reception complete flag.
Arguments: None
Return values: None
### Excep_SCI1_TXI1

**Overview**
SCI1 transmit data empty interrupt handler

**Header**
None

**Declaration**
static void Excep_SCI1_TXI1(void)

**Description**
Disables the TXI1 interrupt and enables the TEI1 interrupt.

**Arguments**
None

**Return values**
None

### Excep_SCI1_TEI1

**Overview**
SCI1 transmission complete interrupt handler

**Header**
None

**Declaration**
static void Excep_SCI1_TEI1(void)

**Description**
Disables transmission and disables the TEI1 interrupt. Sets the transmission complete flag.

**Arguments**
None

**Return values**
None

### Excep_SCI1_ERI1

**Overview**
SCI1 receive error interrupt handler

**Header**
None

**Declaration**
static void Excep_SCI1_ERI1(void)

**Description**
Performs SCI1 reception error handling.

**Arguments**
None

**Return values**
None

**Remarks**
In this sample code, SCI1 reception errors are not handled. (The code goes into an infinite loop.)
If required, add the appropriate processing to this function.
5.10 Flowcharts

5.10.1 Main Processing
Figure 5.4 shows the flowchart for the main processing.

![Flowchart of Main Processing]

Figure 5.4 Main Processing
5.10.2 Port Initialization

Figure 5.5 shows the flowchart for port initialization.

```
port_init

Set port output data
PORT1.PODR register
B6 bit ← 1 : P16/TXD1: High output

Set port directions
PORT1.PDR register
B5 bit ← 0 : P15/RXD1: Input
B6 bit ← 1 : P16/TXD1: Output

PORT3.PDR register
B1 bit ← 0 : P31/IRQ1: Input

Set port mode
PORT1.PMR register
B5 bit ← 0 : P15/RXD1: Used as general-purpose I/O port
B6 bit ← 0 : P16/TXD1: Used as general-purpose I/O port

PORT3.PMR register
B1 bit ← 0 : P31/IRQ1: Used as general-purpose I/O port

return
```

Figure 5.5 Port Initialization

5.10.3 Peripheral Function Initialization

Figure 5.6 shows the flowchart for peripheral function initialization.

```
peripheral_init

Enable writing to related registers
PRCR register ← A502h
PRC1 bit = 1

Clears module stop state
MSTPCRB register
MSTPB30 bit ← 0 : Clears the SCI1 module stop state.

Clears module stop state
MSTPCRA register
MSTPA28 bit ← 0 : Clears the DTC module stop state.

Enable writing to related registers
PRCR register ← A500h
PRC1 bit = 0

Initialize SCI1
sci1_init()

Initialize DTC
dtc_init()

Initialize IRQ
irq_init()

return
```

Figure 5.6 Peripheral Function Initialization
5.10.4 SCI1 Initialization

Figure 5.7 shows the flowchart for SCI1 initialization.

```
c11_init

Disable SCI1 interrupt requests
IER1B register ← 0 : Disables SCI1 ERI1 interrupt request
IEN2 bit ← 0 : Disables SCI1.RXI1 interrupt request
IEN4 bit ← 0 : Disables SCI1.TXI1 interrupt request
IEN5 bit ← 0 : Disables SCI1.TEI1 interrupt request

Disable transmission/reception interrupt requests
SC11.SCR register ← 00h
TEIE bit = 0 : Disables TEI1 interrupt request
RE bit = 0 : Disables serial reception operation
TE bit = 0 : Disables serial transmission operation
RIE bit = 0 : Disables RXI and ERI interrupt requests
TIE bit = 0 : Disables TXI interrupt request

Set up I/O port functions
PWPR register
B0WI bit ← 0 : Enables writing the PFSWE bit.
PWSWE bit ← 1 : Enables writing the PFS register
PSEL[3:0] bits = 1010b : Selects P15 pin function: RXD1
P15PFS register ← 15 : 15.27 = (20 MHz / (64 \times 2^{−1} \times 38400 bps) ) − 1

Select clock
SC11.SCR register
CK[1:0] bits ← 00b : Internal baud rate generator

Set transmission/reception formats
SC11.SMR register ← 00h
CKS[1:0] bits = 00b : PCLK clock
MP bit = 0 : Disables multiplexer communication functions
STOP bit = 0 : One stop bit
PE bit = 0 : No parity
CHR bit = 0 : Transmission and reception with 8-bit data length
CM bit = 0 : Operates in asynchronous mode
SC11.SCMR register ← F2h
SMIF bit = 0 : Serial communications interface mode
SINV bit = 0 : Transmits the contents of the TDR register without modification, and stores the receive data in the RDR register without modification.
SDIR bit = 0 : LSB first transmission and reception
SC11.SEMR register ← 00h
ABCS bit = 0 : The one-bit period for the transfer rate is 16 cycles of the base clock.
NFEN bit = 0 : Disables the RXDn input signal noise exclusion function.

Set bit rate
SC11.BRR register ← 15
[When RX21A is selected]
SC11.BRR register ← 19
15.27 = (20 MHz / (64 \times 2^{−1} \times 38400 bps) ) − 1
19.34 = (25 MHz / (64 \times 2^{−1} \times 38400 bps) ) − 1

Set interrupt priority level
IPR[3:0] bits ← 0001b : Sets the SCI1.RX11, TX11, and TE11 interrupt priority levels to 1.
IR219 register
IR flag ← 0 : No SCI1.RX11 interrupts
IR220 register
IR flag ← 0 : No SCI1.TX11 interrupts

Clear interrupt requests

return
```

Note: 1. After writing a value to the SCR register, applications should verify that the written value can be read back.

Figure 5.7 SCI1 Initialization
### 5.10.5 DTC Initialization

Figure 5.8 shows the flowchart for DTC initialization.

![Flowchart for DTC Initialization](image)

- **dtc_init**
  - DTCST register ← 00h
    - DTCST bit = 0: Stops the DTC module.
  - DTCCCR register
    - RRS bit ← 0: Sets the transfer information read skip operation to not be performed.
  - DTCADM register ← 00h
    - SHORT bit = 0: Full addressing mode
  - **Set RXI1 DTC transfer information**
    - dtc_info_rxi1.MR.LONG ← 0000 0000h: Initialized to 0
    - dtc_info_rxi1.MR.BIT.MRA_MD ← 00b: Normal transfer mode
    - dtc_info_rxi1.MR.BIT.MRA_SZ ← 00b: 8-bit (byte) transfers
    - dtc_info_rxi1.MR.BIT.MRA_SM ← 00b: SAR register holds a fixed address
    - dtc_info_rxi1.MR.BIT.MRB_CHNE ← 0: Chain transfers disabled
    - dtc_info_rxi1.MR.BIT.MRB_DISEL ← 0: Interrupt issued to the CPU when the specified data transfer completes
    - dtc_info_rxi1.MR.BIT.MRB_DM ← 10b: DAR register incremented after transfer
    - dtc_info_rxi1.SAR ← Address of SCI1.RDR register
    - dtc_info_rxi1.DAR ← Address of rcvbuf
    - dtc_info_rxi1.CR.CRA ← DTC_CNT
    - dtc_info_rxi1.CR.CRB ← 0000h: Not used for normal transfers
  - **Set TXI1 DTC transfer information**
    - dtc_info_txi1.MR.LONG ← 0000 0000h: Initialized to 0
    - dtc_info_txi1.MR.BIT.MRA_MD ← 00b: Normal transfer mode
    - dtc_info_txi1.MR.BIT.MRA_SZ ← 00b: 8-bit (byte) transfers
    - dtc_info_txi1.MR.BIT.MRA_SM ← 00b: SAR register incremented after transfer
    - dtc_info_txi1.MR.BIT.MRB_CHNE ← 0: Chain transfers disabled
    - dtc_info_txi1.MR.BIT.MRB_DISEL ← 0: Interrupt issued to the CPU when the specified data transfer completes
    - dtc_info_txi1.MR.BIT.MRB_DM ← 00b: DAR register holds a fixed address
    - dtc_info_txi1.SAR ← Address of trnbuf
    - dtc_info_txi1.DAR ← Address of SCI1.TDR register
    - dtc_info_txi1.CR.CRA ← DTC_CNT
    - dtc_info_txi1.CR.CRB ← 0000h: Not used for repeat transfers
  - **Set address of DTC transfer information in DTC vector**
    - dtc_vect_table[219] ← Address of dtc_info_rxi1
    - dtc_vect_table[220] ← Address of dtc_info_txi1
  - **Set base address**
    - DTCVBR register ← Address of dtc_vect_table
  - return

---

**Figure 5.8 DTC Initialization**
5.10.6 IRQ Initialization

Figure 5.9 shows the flowchart for IRQ initialization.

![Flowchart for IRQ Initialization](image)

**Figure 5.9 IRQ Initialization**
5.10.7 Start SCI1 Communication

Figure 5.10 shows the flowchart for starting SCI1 communication.

```
sci1_start

Stop DTC module
DTCSR register ← 00h
DTCSR bit ← 0 : Stops the DTC module.

Disable transfer information read skip
DTCSR register
RRS bit ← 0 : The transfer information read skip operation is not performed.

Set RXI1 DTC transfer information again
dtc_info_rxi1.DAR ← Address of rcvbuf
dtc_info_rxi1.CRA ← DTC_CNT

Set TXI1 DTC transfer information again
dtc_info_txi1.SAR ← Address of trnbuf
dtc_info_txi1.CRA ← DTC_CNT

Enable transfer information read skip
DTCSR register
RRS bit ← 1 : Transfer information read skip is performed when the header numbers match.

Enable DTC activation by RXI1 and TXI1 interrupt requests
DTCE219 register
DTCE bit ← 1
DTCE220 register
DTCE bit ← 1

Set DTC module to operating state
DTCSR register ← 01h
DTCSR bit = 1 : Starts the DTC module.

Start SCI1 transmission/reception operations
SCI1.SCR register ← SCI1.SCR register | F4h
TEIE bit = 1 : Enables TEI interrupt request
RE bit = 1 : Enables serial reception operations
TE bit = 1 : Enables serial transmission operations
RIE bit = 1 : Enables RXI and ERI interrupt requests
TIE bit = 1 : Enables TXI interrupt requests

PORT1.PMR register
B6 bit ← 1 : P16/TXD1: Used as peripheral function

Enable ERI1 interrupt request
IER1B register
IEN2 bit ← 1 : Enables the SCI1.ERI1 interrupt request.

Enable RXI1 interrupt request
IER1B register
IEN3 bit ← 1 : Enables the SCI1.RXI1 interrupt request.

Enable TXI1 interrupt request
IER1B register
IEN4 bit ← 1 : Enables the SCI1.TXI1 interrupt request.

return
```

Figure 5.10 Start SCI1 Communication
5.10.8 SCI1 Receive Data Full Interrupt Handler

Figure 5.11 shows the flowchart for SCI1 receive data full interrupt handling.

```
ExcepSCI1_RXI1

Disable RXI1 interrupt request
IER1B register
IEN3 bit ← 0: Disables SCI1.RXI1 interrupt requests.

Disable ER1 interrupt request
IER1B register
IEN2 bit ← 0: Disables SCI1.ERI1 interrupt requests.

Disable interrupt request*1
SCI1.SCR register
RIE bit ← 0: Disables RXI and ERI interrupt requests.

Clear RXI1 interrupt request
IR219 register
IR flag ← 0: There is now no SCI1.RXI1 interrupt request.

Set reception complete flag
rcv_end_flag ← 1: Terminates reception

return
```

Note: 1. After writing the RE and RIE bits, applications should verify that the written value can be read back.

Figure 5.11 SCI1 Receive Data Full Interrupt Handler

5.10.9 SCI1 Transmit Data Empty Interrupt Handler

Figure 5.12 shows the flowchart for SCI1 transmit data empty interrupt handling.

```
ExcepSCI1_TXI1

Disable TXI1 interrupt request
IER1B register
IEN4 bit ← 0: Disables SCI1.TXI1 interrupt requests.

Disable interrupt request*1
SCI1.SCR register
TIE bit ← 0: Disables TXI interrupt requests.

Clear TXI1 interrupt request
IR220 register
IR flag ← 0: There is now no SCI1.TXI1 interrupt request.

Enable TEI1 interrupt request
IER1B register
IEN5 bit ← 1: Enables SCI1.TE11 interrupt requests.

return
```

Note: 1. After writing the TIE bits, applications should verify that the written value can be read back.

Figure 5.12 SCI1 Transmit Data Empty Interrupt Handler
5.10.10 SCI1 Transmission Complete Interrupt Handler

Figure 5.13 shows the flowchart for SCI1 transmission complete interrupt handling.

![Flowchart for SCI1 Transmission Complete Interrupt Handler](image)

Note: 1. After writing the SCR register, applications should verify that the written value can be read back.

Figure 5.13 SCI1 Transmission Complete Interrupt Handler

5.10.11 SCI1 Reception Error Interrupt Handler

Figure 5.14 shows the flowchart for SCI1 reception error interrupt handling.

![Flowchart for SCI1 Reception Error Interrupt Handler](image)

In this sample code, SCI1 reception errors are not handled. (The code goes into an infinite loop.) If required, add the appropriate processing to this function.

Figure 5.14 SCI1 Reception Error Interrupt Handler
6. Sample Code
The sample code can be downloaded from the Renesas Electronics Corporation web site.

7. Usage Notes

7.1 Sample Code Usage Notes
Either an RX220 Group or an RX21A Group device can be selected in the sample code. Use the following settings when selecting the device.

1. In the project tab in the workspace window in the High-performance Embedded Workshop, set the project for the device to be used to be the active project.
   Refer to the latest version of the High-performance Embedded Workshop user's manual for the procedure for setting the active project.

![Setting the Active Project](image)

2. Select the device used in the device configuration file (device_config.h). Uncomment the code for the device used and comment out the codes for unused devices.

7.2 Board Usage Notes
Keep the following points in mind when verifying the operation of the sample code for the board used as stipulated in this application note.

- Board used: Renesas Starter Kit for RX220 (R0K505220S000BE)
  Although the microcontroller ports P20 and P21 are connected to the RS-232C serial connector through the RS-232C transceiver when the Renesas Starter Kit for RX220 is shipped from the factory, since ports P20 and P21 do not have an SCI function, these connections must be changed to P15/RXD1 and P16/TXD1.

- Board used: Hokuto Denshi Co., Ltd. HSB Series Microcontroller Board (Catalog number: HSBRX21AP-B)
  No RS-232C transceiver or RS-232C serial connector are connected in the Hokuto Denshi Co., Ltd. HSB Series Microcontroller Board. To verify operation, the user must provide an RS-232C transceiver and an RS-232C serial connector.
8. Reference Documents

User’s Manual: Hardware
- RX220 Group User’s Manual: Hardware Rev.1.00 (R01UH0292EJ)
- RX21A Group User’s Manual: Hardware Rev.1.00 (R01UH0251EJ)
  The latest version can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools
- RX Family C/C++ Compiler Package V.1.01 User’s Manual Rev.1.00 (R20UT0570EJ)
  The latest version can be downloaded from the Renesas Electronics website.

- High-performance Embedded Workshop V.4.09 User’s Manual Rev.1.00 (R20UT0372EJ)
  The latest version can be downloaded from the Renesas Electronics website.

Technical Update / Technical News
- The latest information can be downloaded from the Renesas Electronics website.

Website and Support

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

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

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>Feb 14, 2014</td>
<td>—</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

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

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

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

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

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

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different type number, confirm that the change will not lead to problems.
   - The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy, or otherwise misappropriation of Renesas Electronics product.

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

   - "Standard": Computers, office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   - "High-Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor 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 implantation etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain-use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

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

(1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

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

SALES OFFICES

Renesas Electronics Corporation

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

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

Renesas Electronics Canada Limited
1101 Nicholson Road, Nepean, Ontario L3Y 9C3, Canada
Tel: +1-613-998-8441, Fax: +1-613-998-3229

Renesas Electronics Europe Limited
Dukas Meadows, Millboard Road, Bourne End, Buckinghamshire, SL8 5PH, U.K.
Tel: +44-1462-851-700, Fax: +44-1462-851-804

Renesas Electronics (China) Co., Ltd.
702 Floor, Quantum Plaza, No.27 ZhichunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7879

Renesas Electronics (Shanghai) Co., Ltd.
Unit 501, Tower A, Central Towers, 559 Landao Rd., Puduo District, Shanghai, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1901-1913, 19/F, Tower 3, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-3818, Fax: +852-2886-9204

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-8390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Tahansin-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-5179, Fax: +82-2-558-5110

http://www.renesas.com