RX63N Group
RX63N Group Flash Programmer (Boot Mode)
Using the Renesas Starter Kit+ for RX63N

Abstract
This document describes a flash programmer for RX63N Group using the Renesas Starter Kit+ for RX63N (hereinafter referred to as RSK+RX63N).
The rewrite target is the RX63N Group. Boot mode is used for rewriting the user area in the RX63N Group.

Products
Flash programmer: RX63N Group (ROM size: 1 Mbyte to 2 Mbytes)
Target MCU: RX63N Group (ROM size: 256 Kbytes to 1 Mbyte)
When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.
Contents

1. Specifications .......................................................................................................................... 3
   1.1 RSK+RX63N User Area Memory Map .............................................................................. 4
2. Operation Confirmation Conditions .......................................................................................... 5
3. Reference Application Notes .................................................................................................... 5
4. Hardware .................................................................................................................................. 6
   4.1 Hardware Configuration ................................................................................................. 6
   4.2 Pins Used ....................................................................................................................... 6
5. Software .................................................................................................................................. 7
   5.1 Programming the RSK+RX63N ..................................................................................... 7
      5.1.1 Prepare the FDT Workspace .................................................................................. 8
      5.1.2 Merge and Save Data ........................................................................................... 9
      5.1.3 Program the RSK+RX63N User Area ................................................................... 16
   5.2 Operation Overview .......................................................................................................... 17
      5.2.1 Start the MCU in Boot Mode ............................................................................... 18
      5.2.2 Bit Rate Automatic Adjustment ............................................................................ 19
      5.2.3 Fix the Target MCU ............................................................................................. 20
      5.2.4 Check ID Code Protection ..................................................................................... 23
      5.2.5 Rewrite the Target MCU User Area ...................................................................... 26
      5.2.6 Reset the Target MCU ......................................................................................... 30
   5.3 File Composition .................................................................................................................. 31
   5.4 Option-Setting Memory ...................................................................................................... 32
   5.5 Constants ............................................................................................................................ 32
   5.6 Structure/Union List .......................................................................................................... 36
   5.7 Variables ............................................................................................................................ 36
   5.8 Functions ............................................................................................................................ 38
   5.9 Function Specifications ...................................................................................................... 39
   5.10 Flowcharts ......................................................................................................................... 43
      5.10.1 Main Processing and Communication Protocol Control ....................................... 43
      5.10.2 Initialization of the Peripheral Functions ............................................................... 55
      5.10.3 Initialization of the Timer for Wait Time With the CMT ....................................... 56
      5.10.4 Setting Wait Time With the CMT ......................................................................... 57
      5.10.5 Wait Processing With the CMT ............................................................................ 58
      5.10.6 Interrupt Handling for CMIO in CMT0 ................................................................. 58
      5.10.7 Initialization of the SCI ......................................................................................... 59
      5.10.8 Processing to Change the SCI Bit Rate ................................................................. 60
      5.10.9 Processing to Calculate the SUM Data ................................................................. 61
      5.10.10 Processing to Start the Target MCU in Boot Mode ............................................ 62
      5.10.11 Processing to Reset the Target MCU ................................................................. 63
      5.10.12 Processing to Send a Command ...................................................................... 64
      5.10.13 Processing to Receive a Response .................................................................... 65
      5.10.14 Copying Unsigned 4-Byte Data ........................................................................... 69
6. Sample Code ............................................................................................................................ 70
7. Reference Documents ............................................................................................................ 70
1. Specifications

The flash programmer runs on the RSK+RX63N. After starting the target RX63N Group MCU in boot mode, the flash programmer rewrites the user area in the RX63N Group using asynchronous serial communication.

Table 1.1 lists the Peripheral Functions and Their Applications, and Figure 1.1 shows a Flash Programmer Usage Example.

Channel 0 (SCI0) in the serial communications interface is used for asynchronous serial communication.

The communication data format and output format are as follows.

- Start bit: 1 bit
- Transfer data: 8 bits
- Parity bit: None
- Stop bit: 1 bit
- Bit rate: 19,200 bps (until response to the new bit rate selection command)
  - 1.5 Mbps (after the programming/erasure state transition command)
- Output format: CMOS output

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCI0</td>
<td>Asynchronous serial transmission and reception</td>
</tr>
<tr>
<td>CMT0</td>
<td>Timer for wait time</td>
</tr>
<tr>
<td>I/O ports</td>
<td>Control for boot mode, LCD output</td>
</tr>
</tbody>
</table>

Figure 1.1 Flash Programmer Usage Example
1.1 RSK+RX63N User Area Memory Map

The program of the flash programmer and data to be written to the target MCU user area are stored in the RSK+RX63N User Area. Figure 1.2 shows the RSK+RX63N User Area Memory Map.

Refer to 5.1 Programming the RSK+RX63N for details on programming the RSK+RX63N user area.

---

Figure 1.2  RSK+RX63N User Area Memory Map
2. Operation Confirmation Conditions

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

Table 2.1 Operation Confirmation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU used</td>
<td>R5F563NBDDFC (RX63N Group)</td>
</tr>
<tr>
<td>Operating frequencies</td>
<td>Main clock: 12 MHz&lt;br&gt;PLL: 192 MHz (main clock divided by 1 and multiplied by 16)&lt;br&gt;System clock (ICLK): 96 MHz (PLL divided by 2)&lt;br&gt;Peripheral module clock B (PCLKB): 48 MHz (PLL divided by 4)</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3 V</td>
</tr>
<tr>
<td>Integrated development environment</td>
<td>Renesas Electronics&lt;br&gt;High-performance Embedded Workshop Version 4.09.01</td>
</tr>
<tr>
<td>C compiler</td>
<td>Renesas Electronics&lt;br&gt;C/C++ compiler package for RX Family V.1.02 Release 01&lt;br&gt; Compile options&lt;br&gt;-cpu=rx600 -output=obj=&quot;$$(CONFIGDIR)$(FILELEAF).obj&quot; -debug&lt;br&gt;-nologo&lt;br&gt;(default settings of the integrated development environment are used)</td>
</tr>
<tr>
<td>iodelfine.h version</td>
<td>Version 1.6A</td>
</tr>
<tr>
<td>Endian</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>Renesas Starter Kit+ for RX63N (part number: R0K50563NC010BR)</td>
</tr>
</tbody>
</table>

3. Reference Application Notes

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

- RX63N Group, RX631 Group Initial Setting Rev. 1.10 (R01AN1245EJ)
- RX63N Renesas Starter Kit Sample Code for Hi-performance Embedded Workshop Rev.1.00 (R01AN1395EG)

The initial setting functions and debug LCD output functions in the reference application notes are used in the sample code in this application note. The revision numbers of the reference application notes are current as of the publication of this application note. However, the latest version is always recommended. Visit the Renesas Electronics Corporation website to check for and download the latest version.
4. Hardware

4.1 Hardware Configuration

Figure 4.1 shows a Connection Example.

![Connection Example Diagram]

Figure 4.1 Connection Example

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>P87</td>
<td>Output</td>
<td>Debug LCD data 7 output</td>
</tr>
<tr>
<td>P86</td>
<td>Output</td>
<td>Debug LCD data 6/backlight output</td>
</tr>
<tr>
<td>P85</td>
<td>Output</td>
<td>Debug LCD data 5/Y drive output</td>
</tr>
<tr>
<td>P84</td>
<td>Output</td>
<td>Debug LCD data 4/X drive output</td>
</tr>
<tr>
<td>PF5</td>
<td>Output</td>
<td>Debug LCD enable output</td>
</tr>
<tr>
<td>PJ5</td>
<td>Output</td>
<td>Debug LCD register select output</td>
</tr>
<tr>
<td>P33/RXD0</td>
<td>Input</td>
<td>Input pin for SCI0 receive data</td>
</tr>
<tr>
<td>P32/TXD0</td>
<td>Output</td>
<td>Output pin for SCI0 transmit data</td>
</tr>
<tr>
<td>PE0</td>
<td>Output</td>
<td>RES# pin control</td>
</tr>
<tr>
<td>PE1</td>
<td>Output</td>
<td>MD pin control</td>
</tr>
<tr>
<td>PE2</td>
<td>Output</td>
<td>PC7 pin control</td>
</tr>
</tbody>
</table>
5. Software

5.1 Programming the RSK+RX63N

Data to be programmed in the RSK+RX63N user area is as follows:

- User program to be programmed in the target MCU user area
- Flash programmer program

This document describes an example of using the Renesas Flash Development Toolkit (hereinafter referred to as FDT). Data to be programmed in the RSK+RX63N user area are merged using the editor function for S-Record files or hexadecimal files in the FDT. Also, the merged data is programmed in the RSK+RX63N user area using the FDT. Refer to the User’s Manual of the FDT (R20UT0508EJ) for details on using the FDT.

Figure 5.1 shows the Flow of Programming the RSK+RX63N.

(1) Refer to 5.1.1 Prepare the FDT Workspace for details.
(2) Refer to 5.1.2 Merge and Save Data for details.
(3) Refer to 5.1.3 Program the RSK+RX63N User Area for details.
5.1.1 Prepare the FDT Workspace

Create a workspace and project to use the FDT. Set the MCU used for the flash programmer as the target device.

In the example, Workspace Name is FDT, and Project Name is RX63N_RX631_FlashMemory.
5.1.2 Merge and Save Data

Perform steps (1) to (7) to merge and save data.

(1) Add data files to be merged to the project

In the example, folders FlashMemoryPrograma and UserProgram are added to the RX63N_RX631_FlashMemory project.

The main.mot file of the flash programmer’s program is added to the FlashMemoryPrograma folder.

The following data files are added to the UserProgram folder for each size of the user program to be programmed in the target MCU user area:

- um_all00_256KB.mot file when the user program size is 256 Kbytes
- um_all00_384KB.mot file when the user program size is 384 Kbytes
- um_all00_512KB.mot file when the user program size is 512 Kbytes
- um_all00_768KB.mot file when the user program size is 768 Kbytes
- um_all00_1016KB.mot file when the user program size is 1016 Kbytes
(2) Open data files to be merged on the hex editor window and set the endian

In the example, files “main.mot” and “um_all00_256KB.mot” are opened in the hex editor window. Little endian is selected for both files.
(3) Select user program data to be programmed in the target MCU user area to merge

Select the range as follows:

Addresses 0xFFFC 0000 to 0xFFFF FFFF when the user program size is 256 Kbytes
Addresses 0xFFFA 0000 to 0xFFFF FFFF when the user program size is 384 Kbytes
Addresses 0xFFF8 0000 to 0xFFFF FFFF when the user program size is 512 Kbytes
Addresses 0xFFF4 0000 to 0xFFFF FFFF when the user program size is 768 Kbytes
Addresses 0xFFF0 0000 to 0xFFFF FFFF when the user program size is 1016 Kbytes

In the example, addresses 0xFFFC 0000 to 0xFFFF FFFF of the um_all00_256KB.mot file are selected.
(4) Copy the highlighted user program data to the Windows clipboard

In the example, addresses 0xFFF0 0000 to 0xFFFF FFFF of the um_all00_256KB.mot file are copied to the Windows clipboard.
(5) Merge and create data to be programmed to the RSK+RX63N user area

Select the main.mot file in the hex editor window, and paste the data that was copied to the Windows clipboard in step (4) into addresses 0xFFF0 0000 and higher.

In the example, the start address of paste destination in the main.mot file is set to 0xFFF0 0000. After setting the start address, paste the data from the clipboard.
(6) Save data to be programmed to the RSK+RX63N user area

Select the main.mot file in the hex editor window, name the file to save the data that was created in step (5), and add the file to the project.

In the example, the FlashMemoryPrograma.MOT file is saved in the S-Record Files folder.
(7) Confirm data to be programmed to the RSK+RX63N user area

Confirm the allocation of the merged data in the data file that was created in step (6). Select the data file to be programmed to the RSK+RX63N user area in the workspace window, and confirm the address range of the block used.

Confirm the address range as follows:

- Addresses 0xFFF0 0000 to 0xFFF3 FFFF when the user program size is 256 Kbytes
- Addresses 0xFFF0 0000 to 0xFFF5 FFFF when the user program size is 384 Kbytes
- Addresses 0xFFF0 0000 to 0xFFF7 FFFF when the user program size is 512 Kbytes
- Addresses 0xFFF0 0000 to 0xFFF8 FFFF when the user program size is 768 Kbytes
- Addresses 0xFFF0 0000 to 0xFFFF DFFF when the user program size is 1016 Kbytes
- Addresses 0xFFFF E000 to 0xFFFF FFFF for the program of the flash programmer

In the example, the address range of the block used is confirmed when the user program size is 256 Kbytes.
5.1.3 Program the RSK+RX63N User Area

Select and download the data file to be programmed to the RSK+RX63N user area.

In the example, the FlashMemoryPrograma.MOT file in the S-Record Files folder is downloaded.
5.2 Operation Overview

The target MCU is started in boot mode and the bit rate is automatically adjusted to connect to the MCU at 19,200 bps. After connecting, the supported device inquiry command is sent to obtain information of the target MCU, and then the device select command and block information inquiry command are sent. Also, the operating frequency select command is sent to change the bit rate to 1.5 Mbps.

The programming/erasure state transition command is sent to check the ID code protection of the target MCU and the processing for the boot mode ID code protection is performed.

The user program is written in the target MCU according to the information obtained from the target MCU. After programming is completed, the programmed area in the target MCU is read to verify the read data with the programmed data.

Figure 5.2 shows the Flash Programmer State Transition.

![Flash Programmer State Transition Diagram]

Figure 5.2 Flash Programmer State Transition

1. Refer to 5.2.1 Start the MCU in Boot Mode for details
2. Refer to 5.2.2 Bit Rate Automatic Adjustment for details
3. Refer to 5.2.3 Fix the Target MCU for details
4. Refer to 5.2.4 Check ID Code Protection for details
5. Refer to 5.2.5 Rewrite the Target MCU User Area for details
6. Refer to 5.2.6 Reset the Target MCU for details
5.2.1 Start the MCU in Boot Mode

(1) The flash programmer sets the RES# pin of the target MCU to low.
(2) The flash programmer sets the MD pin of the target MCU to low.
(3) The flash programmer sets the PC7 pin of the target MCU to low.
(4) After waiting 2 ms, the flash programmer sets the RES# pin of the target MCU to high.

![Diagram](image-url)

Figure 5.3 Start Procedure in Boot Mode
5.2.2 Bit Rate Automatic Adjustment

The flash programmer starts the target MCU in boot mode, waits 400 ms, and then sends “00h” 30 times to adjust the bit rate to 19,200 bps.

When the flash programmer receives 00h, send 55h to the target MCU. When 00h cannot be received, the flash programmer restarts the target MCU in boot mode and performs bit rate automatic adjustment again.

After sending 55h, the flash programmer completes the bit rate automatic adjustment when it receives E6h. When the flash programmer sends 55h and then receives FFh, it restarts the target MCU in boot mode and performs bit rate automatic adjustment again.

Figure 5.4 shows the Bit Rate Automatic Adjustment Procedure.

---

**Figure 5.4  Bit Rate Automatic Adjustment Procedure**

---

*Note 1. Restart the target MCU in boot mode and send the bit rate automatic adjust command again.*
5.2.3 Fix the Target MCU

To fix the target MCU, the flash programmer performs steps (1) to (4) below.

(1) The flash programmer sends the supported device inquiry command and stores device codes for selecting the endian of data to be programmed in the user area.

When the flash programmer receives a response (data starting with 30h) to the supported device inquiry command, it stores device codes for selecting the endian of data to be programmed in the user area. When the flash programmer receives data other than the response (data starting with 30h), it resets the target MCU to abort.

Figure 5.5 shows the Procedure to Store Device Codes.

```
Flash programmer                        Target MCU

20h (supported device inquiry)          \

30h, 3Eh, 02h, ...                     20h (supported device inquiry)
(response to the supported device inquiry)

Store device codes for selecting the endian of data to be programmed in the user area
```

Figure 5.5 Procedure to Store Device Codes

(2) The flash programmer sends the device select command to select the endian of data to be programmed in the user area.

The flash programmer sends the device select command (10h) to select the endian of data to be programmed in the user area. The flash programmer uses the device code corresponding to the endian of the flash programmer in the device codes that were stored by the support device inquiry command.

When the flash programmer receives a response (06h) after sending the device select command, it completes the endian selection. When the flash programmer receives data other than the response (06h) after sending the device select command, it resets the target MCU to abort.

Figure 5.6 shows the Procedure to Select the Endian.

```
Flash programmer                        Target MCU

10h (device select), 04h (size), XXXXh (stored device code), XXh (SUM)

06h (response to the device select command)

90h, 11h (checksum error)
90h, 21h (device code error)
```

Figure 5.6 Procedure to Select the Endian
(3) The flash programmer sends the clock mode selection command to select the clock mode of the target MCU.

The flash programmer sends the clock mode selection command (11h) to specify the clock mode. When the flash programmer receives data other than 06h (response to the clock mode selection command), it resets the target MCU to abort.

Figure 5.7 shows the Procedure to Select Clock Mode.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>11h (clock mode selection)</td>
<td></td>
</tr>
<tr>
<td>01h (size)</td>
<td></td>
</tr>
<tr>
<td>00h (clock mode)</td>
<td></td>
</tr>
<tr>
<td>EEh (SUM)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>06h (response to the clock mode selection command)</td>
</tr>
<tr>
<td></td>
<td>91h, 11h (checksum error)</td>
</tr>
<tr>
<td></td>
<td>91h, 22h (clock mode error)</td>
</tr>
</tbody>
</table>

Figure 5.7 Procedure to Select Clock Mode
(4) The flash programmer sends the new bit rate selection command to change the bit rate for communication with the target MCU to 1.5 Mbps.

The flash programmer sends the new bit rate selection command (3Fh) to change the bit rate to 1.5 Mbps. With this command, the input frequency of the target MCU is sent. When the flash programmer receives ACK (06h) after sending the new bit rate selection command, it waits 25 ms at the bit rate for sending the new bit rate selection command and changes the bit rate to 1.5 Mbps. After that, when the flash programmer sends confirmation data (06h) at the changed bit rate and receives a response (06h), it completes changing the bit rate.

When the flash programmer receives data other than 06h (response) after sending the new bit rate selection command or confirmation data (06h), it resets the target MCU to abort.

Figure 5.8 shows the Procedure to Change the Bit Rate.

---

**Figure 5.8  Procedure to Change the Bit Rate**

```
Flash programmer       Target MCU

3Fh (new bit rate selection)
07h (size)
3Ah, 98h (bit rate: 1.5 Mbps)
04h, B0h (input frequency)
02h (number of clock types)
08h (multiplication ratio 1)
04h (multiplication ratio 2)
26h (checksum)

06h (ACK)

BFh, 11h (checksum error)
BFh, 24h (bit rate selection error)
BFh, 25h (input frequency error)
BFh, 26h (multiplication ratio error)
BFh, 27h (operating frequency error)

Wait 25 ms
Change to 1.5 Mbps

06h (confirmation data)

06h (response to the confirmation data)
```
5.2.4 Check ID Code Protection

The flash programmer performs steps (1) to (3) below to check the ID code protection.

1. The flash programmer sends the programming/erasure state transition command to check and store the status of the ID code protection for the target MCU.

The flash programmer sends the programming/erasure state transition command (40h) to check the ID code protection for the target MCU.

After the flash programmer sends the programming/erasure state transition command, it determines the status according to the received response and store the status in the ID code protection status buffer.

Table 5.1 lists the Responses and Values Stored in the ID Code Protection Status Buffer.

<table>
<thead>
<tr>
<th>Response</th>
<th>Values Stored in the ID Code Protection Status Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>26h</td>
<td>00h</td>
</tr>
<tr>
<td>16h</td>
<td>01h</td>
</tr>
</tbody>
</table>

When the flash programmer receives data other than values listed in Table 5.1 after sending the programming/erasure state transition command, it resets the target MCU to abort.

Figure 5.9 shows the Procedure to Check ID Code Protection by the Programming/Erasure State Transition Command.
(2) The flash programmer sends the ID code check command to check and store the status of the ID code protection for the target MCU.

The flash programmer performs this step when the value stored in the ID code protection status buffer is 01h.

The flash programmer sends the ID code check command (60h) to determine the state of ID code protection for the target MCU. The control code, and ID code 1 to ID code 15 are set by reading and using data to be programmed in the target MCU user area.

When the flash programmer receives data other than 26h after sending the ID code check command, it resets the target MCU to abort.

Figure 5.10 shows the Procedure to Check ID Code Protection by the ID Code Check Command.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>60h (ID code check)</td>
<td></td>
</tr>
<tr>
<td>10h (size)</td>
<td></td>
</tr>
<tr>
<td>XXh (control code)</td>
<td></td>
</tr>
<tr>
<td>XXh, XXh, XXh (ID code 1 to ID code 3)</td>
<td></td>
</tr>
<tr>
<td>XXh, XXh, XXh (ID code 4 to ID code 6)</td>
<td></td>
</tr>
<tr>
<td>XXh, XXh, XXh (ID code 7 to ID code 9)</td>
<td></td>
</tr>
<tr>
<td>XXh, XXh, XXh (ID code 10 to ID code 12)</td>
<td></td>
</tr>
<tr>
<td>XXh, XXh, XXh (ID code 13 to ID code 15)</td>
<td></td>
</tr>
<tr>
<td>XXh (SUM)</td>
<td></td>
</tr>
<tr>
<td>26h (response to the ID code check: transition to the programming/erasure state)</td>
<td></td>
</tr>
<tr>
<td>E0h, 11h (checksum error)</td>
<td></td>
</tr>
<tr>
<td>E0h, 61h (ID codes do not match)</td>
<td></td>
</tr>
<tr>
<td>E0h, 63h (error in all-block erasure executed when the ID codes do not match)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.10 Procedure to Check ID Code Protection by the ID Code Check Command
The flash programmer erases the flash memory of the target MCU for writing the user program in the target MCU. The flash programmer performs this step when the value stored in the ID code protection status buffer is 01h.

The flash programmer sends the erasure selection command (48h). When the flash programmer receives 06h (response to the erasure selection command) after sending the erasure selection command, it completes erasure selection. When the flash programmer receives data other than 06h (response to the erasure selection command) after sending the erasure selection command, it resets the target MCU to abort.

The flash programmer sends a block erase command (58h) as many times as the number of blocks in the target MCU, and then it sends a block erase command (58h 04h FFh A5h) to end erasing blocks. When receiving 06h (response to the block erase command) after sending a block erase command, it completes a block erase operation. When the flash programmer receives data other than 06h (response to the block erase command) after sending the block erase command, it resets the MCU to abort.

Figure 5.11 shows the Procedure to Erase the Flash Memory of the Target MCU.

---

Figure 5.11  Procedure to Erase the Flash Memory of the Target MCU
5.2.5 Rewrite the Target MCU User Area

To rewrite the target MCU user area, the flash programmer performs steps (1) to (2) below.

(1) The flash programmer writes the user program to the target MCU user area.

The flash programmer sends 43h (user/data area programming selection command). After that, when the flash programmer receives 06h (response to the user/data area programming selection command), it completes user/data area programming selection. When it receives data other than 06h, it resets the target MCU to abort.

After completion of selection, the flash programmer sends a program command (50h) for the size of the user program to be programmed in the target MCU setting the 256-byte aligned addresses for program addresses and setting program data in 256 bytes.

The range of program addresses (destination of the target MCU) is as follows:
- Addresses from 0xFFFC 0000 to 0xFFFF FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFFFA 0000 to 0xFFFF FFFF when the user program size is 384 Kbytes
- Addresses from 0xFFFF8 0000 to 0xFFFF FFFF when the user program size is 512 Kbytes
- Addresses from 0xFFFF4 0000 to 0xFFFF FFFF when the user program size is 768 Kbytes
- Addresses from 0xFFF0 2000 to 0xFFFF FFFF when the user program size is 1016 Kbytes

The range of program data (source data on the RSK+RX63N) is as follows:
- Addresses from 0xFFF0 0000 to 0xFFF3 FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFF5 FFFF when the user program size is 384 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFF7 FFFF when the user program size is 512 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFFB FFFF when the user program size is 768 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFFF DFFF when the user program size is 1016 Kbytes

After sending the program command for the size of the user program to be programmed in the target MCU user area, the flash programmer sends 50h FFh FFh FFh FFh B4h (program command to end programming). When the flash programmer receives 06h (response to the program command), it completes a program operation. When the flash programmer receives data other than 06h, it resets the MCU to abort.
Figure 5.12 shows the Procedure to Program the User Area.

```
<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>43h (user/data area programming selection)</td>
<td></td>
</tr>
<tr>
<td>06h (response to the user/data area programming selection command: ACK)</td>
<td></td>
</tr>
<tr>
<td>50h (program)</td>
<td></td>
</tr>
<tr>
<td>FFh, Fxh, Xxh, 00h (program address)</td>
<td></td>
</tr>
<tr>
<td>XXh, ..., 256 bytes ..., XXh (program data)</td>
<td></td>
</tr>
<tr>
<td>XXh (SUM)</td>
<td></td>
</tr>
<tr>
<td>Repeat until the user program is completely written</td>
<td></td>
</tr>
<tr>
<td>06h (response to the program command: ACK)</td>
<td></td>
</tr>
<tr>
<td>D0h, 11h (checksum error)</td>
<td></td>
</tr>
<tr>
<td>D0h, 2Ah (address error)</td>
<td></td>
</tr>
<tr>
<td>D0h, 53h (programming error)</td>
<td></td>
</tr>
<tr>
<td>50h (program)</td>
<td></td>
</tr>
<tr>
<td>FFh, FFh, FFh (end of program)</td>
<td></td>
</tr>
<tr>
<td>B4h (SUM)</td>
<td></td>
</tr>
<tr>
<td>06h (response to the program command: ACK)</td>
<td></td>
</tr>
<tr>
<td>D0h, 11h (checksum error)</td>
<td></td>
</tr>
<tr>
<td>D0h, 2Ah (address error)</td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 5.12  Procedure to Program the User Area**
(2) The flash programmer confirms that the target MCU has been programmed correctly.

To confirm that the data has been programmed in the target MCU user area successfully, the flash programmer reads the data in the target MCU user area and compares the read data with the written data.

The flash programmer sends 52h (memory read command) for the size of the user program written in the target MCU user area setting 256-byte aligned addresses for the read addresses.

The range of read addresses is as follows:

- Addresses from 0xFFFC 0000 to 0xFFFF FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFFFA 0000 to 0xFFFF FFFF when the user program size is 384 Kbytes
- Addresses from 0xFFFF8 0000 to 0xFFFF FFFF when the user program size is 512 Kbytes
- Addresses from 0xFFFF4 0000 to 0xFFFF FFFF when the user program size is 768 Kbytes
- Addresses from 0xFFFF0 2000 to 0xFFFF FFFF when the user program size is 1016 Kbytes

When the flash programmer receives data starting with 52h (response to the memory read command), it compares the read data with the source data in the RSK+RX63N user area. When the data do not match, or when the flash programmer receives data other than the response (data starting with 52h), it resets the target MCU to abort.

The range of source addresses is as follows:

- Addresses from 0xFFF0 0000 to 0xFFF3 FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFF5 FFFF when the user program size is 384 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFF7 FFFF when the user program size is 512 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFFB FFFF when the user program size is 768 Kbytes
- Addresses from 0xFFF0 0000 to 0xFFFF DFFF when the user program size is 1016 Kbytes
Figure 5.13 shows the Procedure to Confirm Data in the User Area.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat until all data of the user program is compared</td>
<td></td>
</tr>
</tbody>
</table>

- 52h (memory read)
- 09h (size)
- 01h (area)
- FFh, FXh, XXh, 00h (read address)
- 00h, 00h, 01h, 00h (read size)
- XXh (SUM)

- 52h (response to the memory read command: ACK)
- 00h, 00h, 01h, 00h (read size)
- XXh, ... 256 bytes ..., XXh (read data)
- XXh (SUM)

- D2h, 11h (checksum error)
- D2h, 2Ah (address error)
- D2h, 2Bh (data size error)

![Figure 5.13 Procedure to Confirm Data in the User Area](image-url)
5.2.6 Reset the Target MCU

1. The flash programmer drives the MD pin of the target MCU high.
2. The flash programmer drives the RES# pin of the target MCU low.
3. The flash programmer waits 2 ms and then drives the RES# pin of the target MCU high.
4. The flash programmer goes into an infinite loop.

---

**Figure 5.14 Procedure to Reset the Target MCU**
5.3 File Composition

Table 5.2 lists the Files Used in the Sample Code. Table 5.3 lists the Standard Include Files. Table 5.4 and Table 5.5 list the Functions and Setting Values in the Reference Application Note. Files generated by the integrated development environment are not included in these tables.

### Table 5.2 Files Used in the Sample Code

<table>
<thead>
<tr>
<th>File Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Main processing, processing to send a command, processing to receive a response</td>
</tr>
<tr>
<td>cmt_wait.c</td>
<td>Wait processing with the CMT</td>
</tr>
<tr>
<td>cmt_wait.h</td>
<td>Header file for cmt_wait.c</td>
</tr>
</tbody>
</table>

### Table 5.3 Standard Include Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdint.h</td>
<td>Defines macros declaring the integer type having the specified widths</td>
</tr>
<tr>
<td>stdbool.h</td>
<td>Defines macros for the Boolean type and value</td>
</tr>
<tr>
<td>machine.h</td>
<td>Defines formats of intrinsic functions for the RX Family</td>
</tr>
<tr>
<td>string.h</td>
<td>Library for comparing strings, copying, etc.</td>
</tr>
</tbody>
</table>

### Table 5.4 Functions and Setting Values in the Reference Application Note (RX63N Group, RX631 Group Initial Setting)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Function</th>
<th>Setting Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_init_stop_module.c</td>
<td>R_INIT_StopModule()</td>
<td>-</td>
</tr>
<tr>
<td>r_init_stop_module.h</td>
<td>-</td>
<td>The DMAC/DTC or EXDMAC is set to stop.</td>
</tr>
<tr>
<td>r_init_non_existent_port.c</td>
<td>R_INIT_NonExistentPort()</td>
<td>-</td>
</tr>
<tr>
<td>r_init_non_existent_port.h</td>
<td>-</td>
<td>The 176-pin package is selected.</td>
</tr>
<tr>
<td>r_init_clock.c</td>
<td>R_INIT_Clock()</td>
<td>-</td>
</tr>
<tr>
<td>r_init_clock.h</td>
<td>-</td>
<td>The PLL is selected as the system clock. The sub-clock is not used.</td>
</tr>
</tbody>
</table>

### Table 5.5 Functions and Setting Values in the Reference Application Note (RX63N Renesas Starter Kit Sample Code for Hi-performance Embedded Workshop)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Function</th>
<th>Setting Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcd.c</td>
<td>Init_LCD()</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Display_LCD()</td>
<td></td>
</tr>
<tr>
<td>lcd.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rskrx63ndef.h</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
5.4 Option-Setting Memory

Table 5.6 lists the Option-Setting Memory Configured in the Sample Code. When necessary, set a value suited to the user system.

Table 5.6 Option-Setting Memory Configured in the Sample Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
</table>
| OFS0   | FFFF FF8Fh to FFFF FF8Ch | FFFF FFFFh    | After a reset, the IWDT stops.  
After a reset, the WDT stops. |
| OFS1   | FFFF FF8Bh to FFFF FF88h | FFFF FFFFh    | After a reset, voltage monitoring 0 reset is disabled.  
After a reset, the HOCO oscillation is disabled. |
| MDES   | FFFF FF83h to FFFF FF80h | FFFF FFFFh    | Little endian                                  |

5.5 Constants

Table 5.7 to Table 5.14 list the Constants Used in the Sample Code.
Table 5.7 Constants Used in the Sample Code

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROMVOL_256KB</td>
<td>(256 * 1024)</td>
<td>Selected when the user area size of the target MCU is 256 Kbytes.</td>
</tr>
<tr>
<td>ROMVOL_384KB</td>
<td>(384 * 1024)</td>
<td>Selected when the user area size of the target MCU is 384 Kbytes.</td>
</tr>
<tr>
<td>ROMVOL_512KB</td>
<td>(512 * 1024)</td>
<td>Selected when the user area size of the target MCU is 512 Kbytes.</td>
</tr>
<tr>
<td>ROMVOL_768KB</td>
<td>(768 * 1024)</td>
<td>Selected when the user area size of the target MCU is 768 Kbytes.</td>
</tr>
<tr>
<td>ROMVOL_1MB</td>
<td>(1024 * 1024)</td>
<td>Selected when the user area size of the target MCU is 1 Mbytes.</td>
</tr>
<tr>
<td>TARGET_ROMVOL</td>
<td>ROMVOL_256KB</td>
<td>User area size of the target MCU (256 Kbytes selected)</td>
</tr>
<tr>
<td>TARGET_DATA_ADD</td>
<td>0xFFFF00000</td>
<td>Start address for storing data programmed in the target MCU user area</td>
</tr>
<tr>
<td>FLASH_PRGRMA_SIZE</td>
<td>(8 * 1024)</td>
<td>Size of the flash programmer program (8 Kbytes)</td>
</tr>
<tr>
<td>READING_HEAD_ADDR</td>
<td>WRITING_HEAD_ADDR</td>
<td>Start address for reading the target MCU (same as the start address for programming)</td>
</tr>
<tr>
<td>MDES_ADD</td>
<td>0xFFFFFFFF80</td>
<td>MDES Determine Address</td>
</tr>
<tr>
<td>WRITING_TIME</td>
<td>(TARGET_ROMVOL / 256)</td>
<td>Number of times the target MCU is programmed (in 256 byte units)</td>
</tr>
<tr>
<td>READING_TIME</td>
<td>WRITING_TIME</td>
<td>Number of times the target MCU is read (same as the number of times the target MCU is programmed)</td>
</tr>
<tr>
<td>RES_BUF_SIZE</td>
<td>(262)</td>
<td>Size of the received data storage buffer</td>
</tr>
<tr>
<td>OK</td>
<td>(0)</td>
<td>True value</td>
</tr>
<tr>
<td>NG</td>
<td>(1)</td>
<td>False value</td>
</tr>
<tr>
<td>ERRLOOP_ON</td>
<td>(1)</td>
<td>Selected when error processing (infinite loop) is performed if an error is detected during reception.</td>
</tr>
<tr>
<td>ERRLOOP_OFF</td>
<td>(0)</td>
<td>Selected when error processing (infinite loop) is not performed if an error is detected during reception.</td>
</tr>
<tr>
<td>INTERVAL_ON</td>
<td>(1)</td>
<td>Selected when an interval is set during transmission.</td>
</tr>
<tr>
<td>INTERVAL_OFF</td>
<td>(0)</td>
<td>Selected when no interval is set during transmission.</td>
</tr>
<tr>
<td>RES_ACK_NORMAL</td>
<td>(0x06)</td>
<td>Normal ACK is received</td>
</tr>
<tr>
<td>RES_ID_DISABLED</td>
<td>(0x26)</td>
<td>ACK for disabling ID code protection is received.</td>
</tr>
<tr>
<td>RES_ID_ENABLED</td>
<td>(0x16)</td>
<td>ACK for enabling ID code protection is received.</td>
</tr>
<tr>
<td>ARRAY_SIZE_OF(a)</td>
<td>(sizeof(a) / sizeof(a[0]))</td>
<td>Macro function obtaining the number of bytes for data sending commands</td>
</tr>
<tr>
<td>WT_BASE_US</td>
<td>(1000000)</td>
<td>Operand for calculating wait time in 1 μs units</td>
</tr>
<tr>
<td>WT_BASE_MS</td>
<td>(1000L)</td>
<td>Operand for calculating wait time in 1 ms units</td>
</tr>
<tr>
<td>WT_CMT_CLOCK</td>
<td>(48L * WT_BASE_US)</td>
<td>CMT count source frequency (PCLKB: 48 MHz)</td>
</tr>
<tr>
<td>WT_CMT_DIVIDE</td>
<td>(512L)</td>
<td>CMT count source division ratio</td>
</tr>
<tr>
<td>WAIT_1MS</td>
<td>((1. * (WT_CMT_CLOCK/ WT_CMT_DIVIDE)) / WT_BASE_MS +0.5)</td>
<td>Wait time with the CMT (1 ms)</td>
</tr>
<tr>
<td>WAIT_2MS</td>
<td>((2. * (WT_CMT_CLOCK/ WT_CMT_DIVIDE)) / WT_BASE_MS +0.5)</td>
<td>Wait time with the CMT (2 ms)</td>
</tr>
<tr>
<td>WAIT_25MS</td>
<td>((25. * (WT_CMT_CLOCK/ WT_CMT_DIVIDE)) / WT_BASE_MS +0.5)</td>
<td>Wait time with the CMT (25 ms)</td>
</tr>
<tr>
<td>WAIT_100MS</td>
<td>((100. * (WT_CMT_CLOCK/ WT_CMT_DIVIDE)) / WT_BASE_MS +0.5)</td>
<td>Wait time with the CMT (100 ms)</td>
</tr>
<tr>
<td>WAIT_400MS</td>
<td>((400. * (WT_CMT_CLOCK/ WT_CMT_DIVIDE)) / WT_BASE_MS +0.5)</td>
<td>Wait time with the CMT (400 ms)</td>
</tr>
</tbody>
</table>
### Table 5.8 Constants Used in the Sample Code (ROMVOL_256KB is Selected as TARGET_ROMVOL)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_ID1_ADD</td>
<td>0xFFF3FFA0</td>
<td>Reference address for the control code, and ID code 1 to ID code 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID2_ADD</td>
<td>0xFFF3FFA4</td>
<td>Reference address for ID code 4 to ID code 7 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID3_ADD</td>
<td>0xFFF3FFA8</td>
<td>Reference address for ID code 8 to ID code 11 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID4_ADD</td>
<td>0xFFF3FFAC</td>
<td>Reference address for ID code 12 to ID code 15 programmed to the target MCU</td>
</tr>
<tr>
<td>WRITING_HEAD_ADD</td>
<td>0xFFF0000</td>
<td>Start address for programming the target MCU</td>
</tr>
<tr>
<td>MAX_BLK_NUMBER</td>
<td>0x15</td>
<td>Maximum block number of the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.9 Constants Used in the Sample Code (ROMVOL_384KB is Selected as TARGET_ROMVOL)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_ID1_ADD</td>
<td>0xFFF5FFA0</td>
<td>Reference address for the control code, and ID code 1 to ID code 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID2_ADD</td>
<td>0xFFF5FFA4</td>
<td>Reference address for ID code 4 to ID code 7 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID3_ADD</td>
<td>0xFFF5FFA8</td>
<td>Reference address for ID code 8 to ID code 11 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID4_ADD</td>
<td>0xFFF5FFAC</td>
<td>Reference address for ID code 12 to ID code 15 programmed to the target MCU</td>
</tr>
<tr>
<td>WRITING_HEAD_ADD</td>
<td>0xFFF0000</td>
<td>Start address for programming the target MCU</td>
</tr>
<tr>
<td>MAX_BLK_NUMBER</td>
<td>0x1D</td>
<td>Minimum block number of the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.10 Constants Used in the Sample Code (ROMVOL_512KB is Selected as TARGET_ROMVOL)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_ID1_ADD</td>
<td>0xFFF7FFA0</td>
<td>Reference address for the control code, and ID code 1 to ID code 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID2_ADD</td>
<td>0xFFF7FFA4</td>
<td>Reference address for ID code 4 to ID code 7 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID3_ADD</td>
<td>0xFFF7FFA8</td>
<td>Reference address for ID code 8 to ID code 11 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID4_ADD</td>
<td>0xFFF7FFAC</td>
<td>Reference address for ID code 12 to ID code 15 programmed to the target MCU</td>
</tr>
<tr>
<td>WRITING_HEAD_ADD</td>
<td>0xFFF80000</td>
<td>Start address for programming the target MCU</td>
</tr>
<tr>
<td>MAX_BLK_NUMBER</td>
<td>0x25</td>
<td>Minimum block number of the target MCU</td>
</tr>
</tbody>
</table>
### Table 5.11 Constants Used in the Sample Code (ROMVOL_768KB is Selected as TARGET_ROMVOL)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_ID1_ADD</td>
<td>0xFFFFBFFA0</td>
<td>Reference address for the control code, and ID code 1 to ID code 3 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID2_ADD</td>
<td>0xFFFFBFFA4</td>
<td>Reference address for ID code 4 to ID code 7 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID3_ADD</td>
<td>0xFFFFBFFA8</td>
<td>Reference address for ID code 8 to ID code 11 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID4_ADD</td>
<td>0xFFFFBFFAC</td>
<td>Reference address for ID code 12 to ID code 15 programmed to the target MCU</td>
</tr>
<tr>
<td>WRITING_HEAD_ADD</td>
<td>0xFFF40000</td>
<td>Start address for programming the target MCU</td>
</tr>
<tr>
<td>MAX_BLK_NUMBER</td>
<td>0x2D</td>
<td>Maximum block number of the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.12 Constants Used in the Sample Code (ROMVOL_1MB is Selected as TARGET_ROMVOL)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_ID1_ADD</td>
<td>0xFFFFDFA0</td>
<td>Reference address for the control code, and ID code 1 to ID code 3 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID2_ADD</td>
<td>0xFFFFDFA4</td>
<td>Reference address for ID code 4 to ID code 7 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID3_ADD</td>
<td>0xFFFFDFA8</td>
<td>Reference address for ID code 8 to ID code 11 programmed to the target MCU</td>
</tr>
<tr>
<td>TARGET_ID4_ADD</td>
<td>0xFFFFDFAC</td>
<td>Reference address for ID code 12 to ID code 15 programmed to the target MCU</td>
</tr>
<tr>
<td>WRITING_HEAD_ADD</td>
<td>0xFFF02000</td>
<td>Start address for programming the target MCU</td>
</tr>
<tr>
<td>MAX_BLK_NUMBER</td>
<td>0x35</td>
<td>Maximum block number of the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.13 Constants Used in the Sample Code (Definition Used for Entering Boot Mode)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTMD_PMR</td>
<td>(PORTE.PMR.BYTE)</td>
<td>Output pin is assigned to pins PC7, MD, and RES# of the target MCU (port mode register).</td>
</tr>
<tr>
<td>BTMD_PODR</td>
<td>(PORTE.PODR.BYTE)</td>
<td>Output pin is assigned to pins PC7, MD, and RES# of the target MCU (port output data register).</td>
</tr>
<tr>
<td>BTMD_PDR</td>
<td>(PORTE.PDR.BYTE)</td>
<td>Output pin is assigned to pins PC7, MD, and RES# of the target MCU (port direction register).</td>
</tr>
<tr>
<td>UB_PIN</td>
<td>(PORTE.PODR.BIT.B2)</td>
<td>Output is assigned to the PC7 pin of the target MCU.</td>
</tr>
<tr>
<td>MD_PIN</td>
<td>(PORTE.PODR.BIT.B1)</td>
<td>Output is assigned to the MD pin of the target MCU.</td>
</tr>
<tr>
<td>RES_PIN</td>
<td>(PORTE.PODR.BIT.B0)</td>
<td>Output is assigned to the RES# pin of the target MCU.</td>
</tr>
<tr>
<td>BTMD_PDR_INIT</td>
<td>(0x07)</td>
<td>Initial value of the output from pins PC7, MD, and RES# of the target MCU</td>
</tr>
<tr>
<td>BTMD_PODR_INIT</td>
<td>(0x00)</td>
<td>Initial value of high level output from the PC7 pin of the target MCU</td>
</tr>
</tbody>
</table>
Table 5.14  Constants Used in the Sample Code (Definition for Asynchronous Serial Communication)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCIn</td>
<td>SCI0</td>
<td>SCI channel: SCI0</td>
</tr>
<tr>
<td>MSTP_SCIn</td>
<td>MSTP(SCI0)</td>
<td>SCI0 module stop bit</td>
</tr>
<tr>
<td>IR_SCIn_RXIn</td>
<td>IR(SCI0,RXIO)</td>
<td>SCI0.RXIO interrupt status flag</td>
</tr>
<tr>
<td>IR_SCIn_TXIn</td>
<td>IR(SCI0,TXIO)</td>
<td>SCI0.TXIO interrupt status flag</td>
</tr>
<tr>
<td>RXDn_PDR</td>
<td>(PORT3.PDR.BIT.B3)</td>
<td>SCI0.RXIO pin direction control bit</td>
</tr>
<tr>
<td>RXDnP</td>
<td>(PORT3.PMR.BIT.B3)</td>
<td>SCI0.RXIO pin mode control bit</td>
</tr>
<tr>
<td>RXDn_PDR</td>
<td>(PORT3.PDR.BIT.B2)</td>
<td>SCI0.RXIO pin direction control bit</td>
</tr>
<tr>
<td>RXDnPFS</td>
<td>P32PFS</td>
<td>SCI0.RXIO pin function control register</td>
</tr>
<tr>
<td>RXDnPFS_SELECT</td>
<td>(0x0B)</td>
<td>RXD0 pin function select bit setting value</td>
</tr>
<tr>
<td>TXDn_PDR</td>
<td>(PORT3.PDR.BIT.B2)</td>
<td>SCI0.TXIO pin direction control bit</td>
</tr>
<tr>
<td>TXDnP</td>
<td>(PORT3.PMR.BIT.B2)</td>
<td>SCI0.TXIO pin mode control bit</td>
</tr>
<tr>
<td>TXDnPFS</td>
<td>P32PFS</td>
<td>SCI0.TXIO pin function control register</td>
</tr>
<tr>
<td>TXDnPFS_SELECT</td>
<td>(0x0B)</td>
<td>TXD0 pin function select bit setting value</td>
</tr>
<tr>
<td>SSR_ERROR_FLAGS</td>
<td>(0x38)</td>
<td>Bit pattern of error flags in the SCI.SSR register</td>
</tr>
<tr>
<td>BRR_SET(bps)</td>
<td>(WT_CMT_CLOCK/(32*(0.5)*(bps))-1+0.5)</td>
<td>Macro function to calculate the SCI.BRR register setting value</td>
</tr>
</tbody>
</table>

5.6  Structure/Union List

Figure 5.15 shows the Structure/Union Used in the Sample Code.

```c
typedef struct{
    uint32_t TrnSize;  /* expected value of the transmit size of command */
    uint32_t RecSize;  /* expected value of the receive size of response */
    uint8_t  ACKRes;   /* ACK value of response */
    uint8_t *Command; /* boot command sequence data pointer */
} boot_cmd_t;
```

Figure 5.15  Structure/Union Used in the Sample Code

5.7  Variables

Table 5.15 lists the Global Variable, and Table 5.16 lists the static Variables.

Table 5.15  Global Variable

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>volatile uint8_t_t</td>
<td>CMT_INTERRUPTFLAG</td>
<td>Wait time enable flag</td>
<td>CMT_WAITSET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CMT_WAIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Excep_CMT0_CMI0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReceiveResponse</td>
</tr>
</tbody>
</table>
### Table 5.16  static Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>ResponseBuffer[RES_BUF_SIZE]</td>
<td>Receive data storage buffer</td>
<td>main ReceiveResponse</td>
</tr>
<tr>
<td>uint8_t</td>
<td>TransferMode</td>
<td>Transmit mode flag</td>
<td>main TransferCommand</td>
</tr>
<tr>
<td>uint8_t</td>
<td>ReceiveMode</td>
<td>Receive mode flag</td>
<td>main ReceiveResponse</td>
</tr>
<tr>
<td>uint8_t</td>
<td>IDProtectMode</td>
<td>ID code protection status buffer</td>
<td>main ReceiveResponse</td>
</tr>
<tr>
<td>uint32_t</td>
<td>BufferIndex</td>
<td>Index of the receive data storage buffer</td>
<td>ReceiveResponse</td>
</tr>
<tr>
<td>uint32_t</td>
<td>DeviceCode</td>
<td>Device code storage buffer</td>
<td>main ReceiveResponse</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_BitRateAdjustment_1st[]</td>
<td>Bit rate automatic adjust command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_BitRateAdjustment_2nd[]</td>
<td>Bit rate automatic adjustment confirm command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_EnquiryDevice[]</td>
<td>Supported device inquiry command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_SelectDevice[]</td>
<td>Device selection command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_SelectClockMode[]</td>
<td>Clock mode selection command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_OperatingFreqSel_1st[]</td>
<td>New bit rate selection command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_OperatingFreqSel_2nd[]</td>
<td>New bit rate selection confirm command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_PStatusTransition[]</td>
<td>Programming/erasure state transition command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_IDCodeCheck[]</td>
<td>ID code check command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_EraseSelection[]</td>
<td>Erasure selection command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_BlockErase[]</td>
<td>Block erase command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_ProgramSelection[]</td>
<td>User/data area program selection command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_Program[]</td>
<td>Program command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_ProgramTermination[]</td>
<td>Program end command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_MemoryRead[]</td>
<td>Memory read command data</td>
<td>-</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>BitRateAdjustment_1st</td>
<td>Bit rate automatic adjust command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>BitRateAdjustment_2nd</td>
<td>Bit rate automatic adjustment confirm command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>EnquiryDevice</td>
<td>Supported device inquiry command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>SelectDevice</td>
<td>Device selection command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>SelectClockMode</td>
<td>Clock mode selection command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>OperatingFreqSel_1st</td>
<td>New bit rate selection command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>OperatingFreqSel_2nd</td>
<td>New bit rate selection confirm command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>PStatusTransition</td>
<td>programming/erasure state transition command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>IDCodeCheck</td>
<td>ID code check command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>EraseSelection</td>
<td>Erasure selection command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>BlockErase</td>
<td>Block erase command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>ProgramSelection</td>
<td>User/data area programming selection command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>Program</td>
<td>Programming command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>ProgramTermination</td>
<td>Programming end command structure</td>
<td>main</td>
</tr>
<tr>
<td>boot_cmd_t</td>
<td>MemoryRead</td>
<td>Memory read command structure</td>
<td>main</td>
</tr>
</tbody>
</table>
5.8 Functions

Table 5.17 lists the Functions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing and communication protocol control</td>
</tr>
<tr>
<td>peripheral_init</td>
<td>Initialization of the peripheral functions</td>
</tr>
<tr>
<td>CMT_WaitInit</td>
<td>Initialization of the timer for wait time with the CMT</td>
</tr>
<tr>
<td>CMT_WaitSet</td>
<td>Wait time setting with the CMT</td>
</tr>
<tr>
<td>CMT_Wait</td>
<td>Wait time processing with the CMT</td>
</tr>
<tr>
<td>Excep_CMT0_CMI0</td>
<td>Interrupt handling for CMI0 in CMT0</td>
</tr>
<tr>
<td>SCI_Init</td>
<td>Initialization of the SCI</td>
</tr>
<tr>
<td>SCI_change</td>
<td>Processing to change the SCI bit rate</td>
</tr>
<tr>
<td>CalcSumData</td>
<td>Processing to calculate the SUM data</td>
</tr>
<tr>
<td>BootModeEntry</td>
<td>Processing to start the target MCU in boot mode</td>
</tr>
<tr>
<td>BootModeRelease</td>
<td>Processing to reset the target MCU</td>
</tr>
<tr>
<td>TransferCommand</td>
<td>Processing to send a command</td>
</tr>
<tr>
<td>ReceiveResponse</td>
<td>Processing to receive a response</td>
</tr>
<tr>
<td>U4memcpy</td>
<td>Copying unsigned 4-byte data</td>
</tr>
</tbody>
</table>
5.9 Function Specifications

The following tables list the sample code function specifications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Outline</th>
<th>Header</th>
<th>Declaration</th>
<th>Description</th>
<th>Arguments</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
<td>lcd.h, cmt_wait.h</td>
<td>void main(void)</td>
<td>After initialization, starts the target MCU in boot mode and rewrite the user area.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>peripheral_init</td>
<td>Initialization of the peripheral functions</td>
<td>lcd.h, cmt_wait.h</td>
<td>void peripheral_init(void)</td>
<td>Initializes the peripheral functions used.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>CMT_WaitInit</td>
<td>Initialization of the timer for wait time with the CMT</td>
<td>cmt_wait.h</td>
<td>void CMT_WaitInit(void)</td>
<td>Initializes the timer for wait time (CMT0).</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>CMT_WaitSet</td>
<td>Wait time setting with the CMT</td>
<td>cmt_wait.h</td>
<td>void CMT_WaitSet(uint16_t cnt)</td>
<td>Sets the CMCOR register to the time (μs) specified in the argument and starts incrementing the CMCNT register.</td>
<td>uint16_t cnt: Wait time</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>Minimum wait time: ( 1 \div (\text{PCLKB[MHz]} \div 512) \approx 10.67 , \mu\text{s} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### CMT_Wait

**Outline**
Wait time processing with the CMT

**Header**
cmt_wait.h

**Declaration**
void CMT_Wait(uint16_t cnt)

**Description**
Waits the time (μs) specified in the argument.

**Arguments**
uint16_t cnt: Wait time

**Return Value**
None

**Remarks**
Minimum wait time: $\frac{1}{(PCLKB[MHz] \div 512)} \approx 10.67$ μs

---

### Excep_CMT0_CMI0

**Outline**
Interrupt handling for CMI0 in CMT0

**Header**
cmt_wait.h

**Declaration**
void Excep_CMT0_CMI0(void)

**Description**
Interrupt handling for compare match between CMT0.CMCNT and CMT0.CMCOR

**Arguments**
None

**Return Value**
None

---

### SCI_Init

**Outline**
Initialization of the SCI

**Header**
None

**Declaration**
void SCI_Init(void)

**Description**
Initializes the SCI.

**Arguments**
None

**Return Value**
None

---

### SCI_change

**Outline**
Processing to change the SCI bit rate

**Header**
None

**Declaration**
void SCI_change(void)

**Description**
Changes the SCI bit rate from 19,200 bps to 1.5 Mbps.

**Arguments**
None

**Return Value**
None

---

### CalcSumData

**Outline**
Processing to calculate the SUM data

**Header**
None

**Declaration**
uint8_t CalcSumData(uint8_t *pData, uint32_t Length)

**Description**
Calculates the SUM data in the boot communication protocol.

**Arguments**
uint8_t *pData: Data address for SUM
uint32_t Length: Amount of data for SUM

**Return Value**
SUM data
### BootModeEntry

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing to start the target MCU in boot mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void BootModeEntry(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Controls pins MD, PC7, and RES# to start the target MCU in boot mode.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### BootModeRelease

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing to reset the target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void BootModeRelease(uint8_t mode)</td>
</tr>
<tr>
<td>Description</td>
<td>Resets the target MCU.</td>
</tr>
<tr>
<td>Arguments</td>
<td>uint8_t mode: Select the output pattern for the second line of the debug LCD</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### TransferCommand

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing to send a command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void TransferCommand(boot_cmd_t *pCmd)</td>
</tr>
<tr>
<td>Description</td>
<td>Sends command data of the command structure specified in the argument.</td>
</tr>
<tr>
<td>Arguments</td>
<td>boot_cmd_t *pCmd: Address of the command structure to be sent</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>Call CMT_Wait(WAIT_1MS) if the TransferMode variable is INTERVAL_ON.</td>
</tr>
</tbody>
</table>

### ReceiveResponse

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing to receive a response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>uint8_t ReceiveResponse(boot_cmd_t *pCmd)</td>
</tr>
<tr>
<td>Description</td>
<td>Receives a response for the number of bytes of the expected response size in the command structure.</td>
</tr>
<tr>
<td>Arguments</td>
<td>boot_cmd_t *pCmd: Address of the command structure to be received</td>
</tr>
</tbody>
</table>
| Return Value | OK: Reception completed successfully  
 NG: Timeout (5 seconds) or error response received |
<p>| Remarks | When the ReceiveMode variable is ERRLOOP_ON and the return value is NG, call the BootModeRelease(NG) function and do not return from the ReceiveResponse function. |</p>
<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Copying unsigned 4-byte data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void *U4memcpy(void *pS1, const void *pS2)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Copies 4 bytes of data in the source memory area to the destination memory area. If the data arrangement is little endian, reverses bytes of the unsigned 4-byte data in the destination.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>void *pS1: Address of the destination memory area</td>
</tr>
<tr>
<td></td>
<td>const void *pS2: Address of the source memory area</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>pS1 value</td>
</tr>
</tbody>
</table>
5.10 Flowcharts

5.10.1 Main Processing and Communication Protocol Control

Figure 5.16 to Figure 5.28 show the Main Processing and Communication Protocol Control.

![Flowchart](image)

Figure 5.16  Main Processing and Communication Protocol Control (1/13)
Processing to start the target MCU in boot mode

BootModeEntry()

Processing to send a command

TransferCommand()

Processing to receive a response

ReceiveResponse()

Is the bit rate automatic adjustment complete?

Yes

Set modes for transmission and reception

TransferMode ← INTERVAL_OFF
ReceiveMode ← ERRLOOP_ON

No

Is the bit rate automatic adjustment complete?

Yes

Set modes for transmission and reception

TransferMode ← INTERVAL_ON
ReceiveMode ← ERRLOOP_OFF

BitAdjust ← NG

Figure 5.17 Main Processing and Communication Protocol Control (2/13)
Figure 5.18  Main Processing and Communication Protocol Control (3/13)
Device selection

Copying unsigned 4-byte data
U4memcpy()

Processing to calculate the
SUM data
CalcSumData()

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Argument
&CMD_SelectDevice[2]
&DeviceCode

CMD_SelectDevice[6] ← CalcSumData()
Argument
&CMD_SelectDevice[0]
6ul

Argument
&SelectDevice

Argument
&SelectDevice

Figure 5.19 Main Processing and Communication Protocol Control (4/13)

Clock mode selection

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Argument
&SelectClockMode

Argument
&SelectClockMode

Figure 5.20 Main Processing and Communication Protocol Control (5/13)
New bit rate selection

5

Processing to send a command
TransferCommand()

Argument
&OperatingFreqSel_1st

Processing to receive a response
ReceiveResponse()

Argument
&OperatingFreqSel_1st

Processing to change the SCI bit rate
SCI_change()

Processing to send a command
TransferCommand()

Argument
&OperatingFreqSel_2nd

Processing to receive a response
ReceiveResponse()

Argument
&OperatingFreqSel_2nd

6

Figure 5.21 Main Processing and Communication Protocol Control (6/13)
Set receive mode
ReceiveMode ← ERRLOOP_OFF

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

ReceiveMode ← ERRLOOP_OFF

Argument &PEstatusTransition

Was RES_ID_DISABLED received?
No
Yes

Was RES_ID_ENABLED received?
No
Yes

Set the ID code protection status buffer
IDProtectMode ← 0

Set the ID code protection status buffer
IDProtectMode ← 1

Processing to reset the target MCU
BootModeRelease()
Argument NG

(See Note 1)

Note 1. The flash programmer goes into an infinite loop by the BootModeRelease() function and does not return to the caller.

Figure 5.22 Main Processing and Communication Protocol Control (7/13)
Figure 5.23  Main Processing and Communication Protocol Control (8/13)
Erasure selection and block erase

8

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Set the first block number to be erased

Is the block number to be erased equal to or greater than 0?

Yes

Set the block number to be erased

Set the block number to be erased

Processing to calculate the SUM data
CalcSumData()

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Update the block number to be erased

No

Figure 5.24 Main Processing and Communication Protocol Control (9/13)
End of block erase

9

Set the end of block erase
memset()

Processing to calculate the SUM data
CalcSumData()

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Argument
&CMD_BlockErase[2]
0xFF
4ul

CMD_BlockErase[6] ← CalcSumData()
Argument
&CMD_BlockErase[0]
6ul

Argument
&BlockErase

Argument
&BlockErase

10

Figure 5.25 Main Processing and Communication Protocol Control (10/13)
User/data area program preparation and program

10

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Clear the WriteIndex variable
WriteIndex ← 0

Is WriteIndex less than WRITING_TIME?

No

Yes

Set the AddressIndex variable
AddressIndex ← WriteIndex << 8 :256-fold increase

Calculate the program address
WriteAddress ← WRITING_HEAD_ADD + AddressIndex

Set the program data
memcpy()

Copying unsigned 4-byte data
U4memcpy()

Processing to calculate the SUM data
CalcSumData()

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

WriteIndex ← WriteIndex + 1

11

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Argument &ProgramSelection
Argument &ProgramSelection

Argument &CMD_Program[1]
Argument &CMD_Program[0]
261ul

Argument &CMD_Program[5]
TARGET_DATA_ADD + AddressIndex
256ul

CMD_Program[261] ← CalcSumData() 
Argument &CMD_Program[0]
261ul

Argument &Program
Argument &Program

Argument &ProgramTermination
Argument &ProgramTermination

Figure 5.26  Main Processing and Communication Protocol Control (11/13)
Memory read

11

Clear the ReadIndex variable

ReadIndex ← 0

No

Is ReadIndex less than READING_TIME?

Yes

Set the AddressIndex

AddressIndex ← ReadIndex << 8 : 256-fold increase

Calculate the read address

ReadAddress ← READING_HEAD_ADD + AddressIndex

Copying unsigned 4-byte data

U4memcpy()

Set the read address

Argument

&CMD_MemoryRead[3]

&ReadAddress

CMD_MemoryRead[11] ← CalcSumData()

Argument

&CMD_MemoryRead[0]

11ul

Processing to calculate the SUM data

CalcSumData()

Processing to send a command

TransferCommand()

Processing to receive a response

ReceiveResponse()

Comparing the read data

strncmp()

ReadIndex + 1

Argument

&ResponseBuffer[5]

TARGET_DATA_ADD + AddressIndex

256ul

Do the program data and read data match?

Yes

Processing to reset the target MCU

BootModeRelease()

Argument

NG

ReadIndex + 1

ReadIndex ← ReadIndex + 1

No

12

Note 1. The flash programmer goes into an infinite loop by the BootModeRelease() function and does not return to the caller.
Processing to reset the target MCU
BootModeRelease()

Argument
OK

(See Note 1)

Note 1. The flash programmer goes into an infinite loop by the BootModeRelease() function and does not return to the caller.
Figure 5.29 shows the Initialization of the Peripheral Functions.

```
  peripheral_init()

  Disable register protection
  PRCR register ← A502h
  PRC1 bit = 1
 : Enable writing to the associated registers.

  MPC.PWPR register
  B0WI bit ← 0

  MPC.PWPR register
  PFSWE bit ← 1

  Enable writing to the PFS register
  PFSWE bit ← 0
 : Disable writing to the PFS register

  Enable writing to the PFSWE bit
  B0WI bit ← 1

  Disable register protection
  PRCR register ← A500h
  PRC1 bit = 0
 : Disable writing to the associated registers.

  Initialization of the timer for wait
  time with the CMT
  CMT_WaitInit

  Initialization of the SCI
  SCI_Init()

  Set the port for output of debug LCD
  data
  LCD_DATA_PORT ← LCD_DATA_PORT & 0x0F
 : Set output from the port to 0.
  LCD_DATA_DIR_PORT ← LCD_DATA_DIR_PORT | 0xF0
 : Set the port direction register to output.

  Set the control pin for the debug LCD
  RS_PIN ← 0
 : Set output from the port to 0.
  E_PIN ← 0
 : Set output from the port to 0.
  RS_PIN_DIR ← 1
 : Set port direction register to output.
  E_PIN_DIR ← 1
 : Set port direction register to output.

  Disable writing to the PFS register
  MPC.PWPR register
  PFSWE bit ← 0

  Disable writing to the PFSWE bit
  MPC.PWPR register
  B0WI bit ← 1

  Enable register protection
  PRCR register ← A500h
  PRC1 bit = 0
 : Disable writing to the associated registers.

  Debug LCD output
  Display_LCD()

  return
```

Figure 5.29 Initialization of the Peripheral Functions
5.10.3 Initialization of the Timer for Wait Time With the CMT

Figure 5.30 shows the Initialization of the Timer for Wait Time with the CMT.

```
CMT_WaitInit()

Release from the module stop state
MSTPCRA register
MSTPA15 bit ← 0 : The module-stop state is canceled for CMT0 and CMT1.

Stop the CMT0 count
CMSTR0 register
STR0 bit ← 0 : CMT0.CMCNT count is stopped.

Set the CMT0 count source and enable compare match interrupt
CMCR register ← 00C3h
CKS[1:0] bits = 11b : PCLK/512
CMIE bit = 1 : Compare match interrupt (CMI0) enabled.

Clear the CMT0 counter
CMCNT register ← 0000h

Set the CMI0 interrupt priority level
IPR004 register ← 03h
IPR[3:0] bits = 0011b : CMI0 interrupt priority level 3

Enable the CMI0 interrupt request
IER03 register
IEN4 bit ← 1 : CMI0 interrupt request is enabled.

Clear the CMI0 interrupt request
IR028 register
IR flag ← 0 : CMI0 interrupt request is cleared.
```

Figure 5.30 Initialization of the Timer for Wait Time with the CMT
5.10.4 Setting Wait Time With the CMT

Figure 5.31 shows Setting Wait Time With the CMT.

```
CMT_WaitSet()
  Argument
  uint16_t cnt : Wait time
  CMT_InterruptFlag ← 1
  CMSTR0 register
  STR0 bit ← 0 : CMT0.CMCNT count is stopped.
  CMCNT register ← 0000h
  Is the specified wait time other than 0?
    Yes (wait time ≠ 0)
      Decrement the wait time
      cnt ← cnt - 1
    No (wait time = 0)
      Set the wait time in CMT0
      CMCOR register ← cnt
      Clear the CMI0 interrupt request
      IR028 register
      IR flag ← 0 : CMI0 interrupt request is cleared.
      Start the CMT0 count
      CMSTR0 register
      STR0 bit ← 1 : CMT0.CMCNT count is started.
```

Figure 5.31 Setting Wait Time With the CMT
5.10.5 Wait Processing With the CMT

Figure 5.32 shows Wait Processing With the CMT.

```
CMT_Wait()
```

Argument
```
uint16_t cnt : Wait time
```

Wait time setting with the CMT

```
CMT_WaitSet()
```

Argument
```
cnt : Wait time
```

Is the wait time enable flag other than 1?

- No (wait time enable flag = 1)
- Yes (wait time enable flag ≠ 1)

Wait until the CMIO interrupt occurs.

```
return
```

Figure 5.32  Wait Processing With the CMT

5.10.6 Interrupt Handling for CMIO in CMT0

Figure 5.33 shows Interrupt Handling for CMIO in CMT0.

```
Excep_CMT0_CMIO()
```

Clear the wait enable flag

```
CMT_InterruptFlag ← 0
```

Stop the CMT0 count

```
CMSTR0 register
STR0 bit ← 0 : CMT0.CMCNT count is stopped.
```

```
return
```

Figure 5.33  Interrupt Handling for CMIO in CMT0
5.10.7 Initialization of the SCI

Figure 5.34 shows SCI Initialization.

```
| Set port mode | TXDn_PMR ← 0 :TXDn: Used as a general I/O port. |
|               | RXDn_PMR ← 0 :RXDn: Used as a general I/O port. |
| Set port output data | TXDn_PODR ← 1 :TXDn: High level |
| Set the port direction | TXDn_PDR ← 0 :TXDn: Output |
|                    | RXDn_PDR ← 0 :RXDn: Input |
| Release from SCIn module stop | MSTP_SCIn ← 0 :SCIn is released from the module stopped state. |
| Disable transmission/reception and the SCIn interrupt request | SCIn.SCR register ← 00h |
|                     | CKE[1:0] bits = 00b :On-chip baud rate generator |
|                     | TEIE bit = 0 :TEI interrupt request is disabled. |
|                     | RE bit = 0 :Serial reception is disabled. |
|                     | TE bit = 0 :Serial transmission is disabled. |
|                     | RIE bit = 0 :RXI and ERI interrupt requests are disabled. |
|                     | TIE bit = 0 :TXI interrupt request is disabled. |
| Select the pin function | MPC.TXDnPFS register ← TXDnPFS_SELECT :TXDn |
|                     | MPC.RXDnPFS register ← RXDnPFS_SELECT :RXDn |
| Set the format for transmission and reception | SCIn.SMR register ← 00h |
|                     | CKS[1:0] bits = 00b :PCLK clock |
|                     | MP bit = 0 :Multi-processor communications function is disabled. |
|                     | STOP bit = 0 :1 stop bit |
|                     | PE bit = 0 :No parity |
|                     | CHR bit = 0 :Selects 8 bits as the data length. |
|                     | CM bit = 0 :Asynchronous mode |
|                     | SCIn.SCMR register ← F2h |
|                     | SMIF bit = 0 :Serial communications interface mode |
|                     | SINV bit = 0 :TDR contents are transmitted as they are. |
|                     | SDIR bit = 0 :Receive data is stored as it is in RDR. |
|                     | Selects 8 base clock cycles for 1-bit period. |
|                     | NFEN bit = 0 :Noise cancellation function for the RXDn input signal is disabled. |
| Set the bit rate | SCIn.BRR register ← BRR_SET(19200) : (48 MHz/(32 × 2⁻¹ × 19200 bps)) - 1 |
| Clear the SCIn interrupt request | IR_SCIn_RXIn ← 0 :SCIn.RXIn interrupt not requested. |
| Enable transmission/reception and SCIn interrupt request | IR_SCIn_TXIn ← 0 :SCIn.TXIn interrupt not requested. |
|                     | SCIn.SCR register ← F0h |
|                     | RE bit = 1 :Serial reception is enabled. |
|                     | TE bit = 1 :Serial transmission is enabled. |
|                     | RIE bit = 1 :RXI and ERI interrupt requests are enabled. |
|                     | TIE bit = 1 :TXI interrupt request is enabled. |
| Set port mode | TXDn_PMR ← 1 :TXDn: Used for peripheral functions. |
|                     | RXDn_PMR ← 1 :RXDn: Used for peripheral functions. |
```

Figure 5.34 SCI Initialization
5.10.8 Processing to Change the SCI Bit Rate

Figure 5.35 shows Processing to Change the SCI Bit Rate.

```
5.10.8 Processing to Change the SCI Bit Rate
Figure 5.35 shows Processing to Change the SCI Bit Rate.

SCI_change()
  Wait time processing with the CMT
  CMT_Wait()
  Argument
  WAIT_25MS : Wait time 25 ms
  Set port mode
  TXDn_PMR ← 0 : TXDn: Used as a general I/O port.
  Disable transmission/reception and the
  SCIn interrupt request
  SCIn.SCR register ← 00h
    CKE[1:0] bits = 00b : On-chip baud rate generator
    TEIE bit = 0 : TEI interrupt request is disabled.
    RE bit = 0 : Serial reception is disabled.
    TE bit = 0 : Serial transmission is disabled.
    RIE bit = 0 : RXI and ERI interrupt requests are disabled.
    TIE bit = 0 : TXI interrupt request is disabled.
  Set the bit rate
  SCIn.BRR register ← BRR_SET(1500000) : (48 MHz/(32 × 2^1 × 1500000 bps)) - 1
  Enable transmission/reception and the
  SCIn interrupt request
  SCIn.SCR register ← F0h
    RE bit = 1 : Serial reception is enabled.
    TE bit = 1 : Serial transmission is enabled.
    RIE bit = 1 : RXI and ERI interrupt requests are enabled.
    TIE bit = 1 : TXI interrupt request is enabled.
  Clear the SCIn interrupt request
  IR_SCIn_TXIn ← 0 : SCIn.RXIn interrupt not requested.
  Set port mode
  TXDn_PMR ← 1 : TXDn: Used for peripheral functions.
  return
```

Figure 5.35   Processing to Change the SCI Bit Rate
### 5.10.9 Processing to Calculate the SUM Data

Figure 5.36 shows Processing to Calculate the SUM Data.

```
CalcSumData()

Clear the loop counter
loop ← 0

Clear the checksum
CheckSum ← 0

Is the loop counter value less than the amount of data for SUM?

No

CheckSum ← CheckSum + *pData;

pData ← pData + 1

loop ← loop + 1

Yes

Calculate the checksum

CheckSum ← 0 - CheckSum

Calculate the SUM value

return (CheckSum)
```

Figure 5.36 Processing to Calculate the SUM Data
5.10.10 Processing to Start the Target MCU in Boot Mode

Figure 5.37 shows Processing to Start the Target MCU in Boot Mode.

![Diagram of Processing to Start the Target MCU in Boot Mode]

- **BootModeEntry()**
  - Set the output port
  - **BTMD_PMR ← 00h**
  - **BTMD_PODR ← BTMD_PODR_INIT**
  - **BTMD_PDR ← BTMD_PDR_INIT**
  - Used as a general I/O port.
  - Initialize output from pins PC7, MD, RES# of the target MCU.
  - Output from the UB# pin of the target MCU is initialized to the high level.

- **Wait time processing with the CMT**
  - **CMT_Wait()**
  - **Argument WAIT_2MS**
  - Wait time 2 ms for RES# pulse width after power-on

- **Release the target MCU reset**
  - **RES_PIN ← 1**
  - Set the RES# pin of the target MCU to the high level.

- **Wait time processing with the CMT**
  - **CMT_Wait()**
  - **Argument WAIT_400MS**
  - Wait time 400 ms until the start of communication

**Figure 5.37 Processing to Start the Target MCU in Boot Mode**
### 5.10.11 Processing to Reset the Target MCU

Figure 5.38 shows Processing to Reset the Target MCU.

![Diagram of Processing to Reset the Target MCU](image)

- **BootModeRelease()**
  - Argument: `uint8_t mode`
  - MD_PIN ← 1: Set the MD pin of the target MCU to the high level.
  - RES_PIN ← 0: Set the RES# pin of the target MCU to the low level.

- **Set the MD pin to the high level**
  - MD_PIN ← 1

- **Reset the target MCU**
  - RES_PIN ← 1

- **Wait time processing with the CMT**
  - Argument: `WAIT_2MS`
  - WAIT_2MS: Wait time
  - 2 ms for the RES# pulse width after power-on

- **Release the target MCU reset**
  - RES_PIN ← 1

- **Is the normal value output to line 2 of the debug LCD?**
  - Yes
    - Argument of `Display_LCD()`: "...End" is output to LCD_LINE2
  - No
    - Argument of `Display_LCD()`: "...Error" is output to LCD_LINE2

- **Debug LCD output**
  - `Display_LCD()`
5.10.12 Processing to Send a Command

Figure 5.39 shows Processing to Send a Command.

![Flowchart Diagram]

- **TransferCommand()**
  - **Argument**
    - `boot_cmd_t *pCmd` : Address for command structure to be transmitted
  - **Argument**
    - `WAIT_1MS` : Wait time
      - 1 ms (interval for transmit data)

- **Clear the transmit counter**
  - `TransferCount ← 0`

- **Is the transmit counter value less than the command size?**
  - **Yes**
    - **Wait until data has been transmitted**
    - **Set the transmit data**
    - **Wait until SCIn.SSR.BIT.TEND becomes 1.**
  - **No**
    - **Is an interval set for transmission?**
      - **Yes**
        - **Wait time processing with the CMT**
        - **CMT_Wait()**
      - **No**
        - **Is the transmit counter value less than the command size?**
          - **Yes**
          - **No**

- **Transfer counter value + 1**
  - `TransferCount ← TransferCount + 1`

- **return**
5.10.13 Processing to Receive a Response

Figure 5.40 to Figure 5.43 show Processing to Receive a Response.

- ReceiveResponse
  - Argument `boot_cmd_t *pCmd`: Address for the command structure to be received
  - Initialize the timeout counter `TimeOutCount ← 50`
  - Clear the index `BufferIndex ← 0`: Index for the buffer storing receive data
  - Set the return value (OK) `ret ← OK`
  - Clear the receive counter `ReceiveCount ← 0`

- Is the receive counter value less than the response size, and is the return value OK?
  - No
  - Wait time setting with the CMT `CMT_WaitSet()`
    - Argument `WAIT_100MS`: Wait time 100 ms
  - Yes

Figure 5.40 Processing to Receive a Response
Read the SCIn.SSR register
PER flag : 0: No parity error occurred.
1: A parity error has occurred.
FER flag : 0: No framing error occurred.
1: A framing error has occurred.
ORER flag : 0: No overrun error occurred.
1: An overrun error has occurred.
SCIn.SSR register ← (SCIn.SSR register & C7h) | C0h
PER flag = 0 : No parity error occurred.
FER flag = 0 : No framing error occurred.
ORER flag = 0 : No overrun error occurred.
ret ← NG

TimeOutCount ← TimeOutCount - 1
: Timeout
(5 sec. = 100 ms × 50 times)
ret ← NG

Argument
WAIT_100MS : Wait time
100 ms

Note 1. After writing to bits RE and RIE and flags PER and FER and ORER, confirm that the written values can be read.

Figure 5.41 Processing to Receive a Response
Figure 5.42 Processing to Receive a Response

- Store the receive data: \( \text{ResponseBuffer[BufferIndex]} \leftarrow \text{SCIn.RDR register} \)
- Index + 1: \( \text{BufferIndex} \leftarrow \text{BufferIndex + 1} \) : Index of the receive data storage buffer
- Clear the index: \( \text{BufferIndex} \leftarrow 0 \) : Index of the receive data storage buffer
- Set the return value (NG): \( \text{ret} \leftarrow \text{NG} \)
- Receive counter value + 1: \( \text{ReceiveCount} \leftarrow \text{ReceiveCount + 1} \)
Is the error processing (infinite loop) selected, and is the return value NG?

No

Yes

Processing to reset the target MCU
BootModeRelease()

(See Note 1)

Argument NG : Output "...Error" to LCD_LINE2

return (ret)

Note 1. The flash programmer goes into an infinite loop by the BootModeRelease() function and does not return to the caller.

Figure 5.43 Processing to Receive a Response
5.10.14 Copying Unsigned 4-Byte Data

Figure 5.44 shows Copying Unsigned 4-Byte Data.

```
void *memcpy(const void *pS1, const void *pS2, size_t n);
```

- **pS1**: Destination address of the memory area for copying
- **pS2**: Source address of the memory area for copying

```
*(uint32_t *)pS1 ← revl(*(uint32_t *)pS1)
```

**Argument**
- pS1
- pS2
- 4ul

**Return**
- return (pS1)
6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

User’s Manual: Hardware
RX63N Group, RX631 Group User's Manual: Hardware Rev.1.80 (R01UH0041EJ)
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.

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.

Website and Support

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

Inquiries
http://www.renesas.com/contact/
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep. 10, 2014</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 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 LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

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

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

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

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
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 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 products quality grade, as indicated below.

   “Standard” : Computer, 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 implants etc.), or may cause serious property damage (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 specific 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, the control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of micromachine software alone is very difficult, please evaluate the safety of the final product or system 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 products or technology described in this document for any purpose relating to military or space application, or 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.

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

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

(Note 2) “Renesas Electronics products” means any product developed or manufactured by or for Renesas Electronics.