RX100 Series
RX100 Series Flash Programmer (SCI)
Using the Renesas Starter Kit+ for RX63N

Abstract
This document describes a flash programmer for RX100 Series using the Renesas Starter Kit+ for RX63N (hereinafter referred to as RSK+RX63N).

The target for rewriting is the RX100 Series. Boot mode (SCI) is used for rewriting the user area in the RX100 Series.

Products
RX100 Series

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 ................................................................................................................................. 4
   1.1 RSK+RX63N User Area Memory Map................................................................................ 5

2. Operation Confirmation Conditions .............................................................................................. 6

3. Reference Application Notes ......................................................................................................... 6

4. Hardware ........................................................................................................................................ 7
   4.1 Hardware Configuration ........................................................................................................ 7
   4.2 Pins Used .............................................................................................................................. 7

5. Software ......................................................................................................................................... 8
   5.1 Programming the RSK+RX63N .......................................................................................... 8
      5.1.1 Prepare the FDT Workspace ..................................................................................... 9
      5.1.2 Merge and Save Data ............................................................................................... 10
      5.1.3 Program the RSK+RX63N User Area ....................................................................... 17
   5.2 Operation Overview .................................................................................................................. 18
      5.2.1 Start the MCU in Boot Mode (SCI) ........................................................................ 19
      5.2.2 Bit Rate Automatic Adjustment .................................................................................. 20
      5.2.3 Fix the Target MCU .................................................................................................... 21
      5.2.4 Check ID Code Protection ......................................................................................... 24
      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 .................................................................................................................................. 37
   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 ................................................................. 57
      5.10.3 Initialization of the Timer for Wait Time with the CMT .............................................. 58
      5.10.4 Setting Wait Time with the CMT .............................................................................. 59
      5.10.5 Wait Processing with the CMT ................................................................................... 60
      5.10.6 Interrupt Handling for CMIO in CMT0 ..................................................................... 61
      5.10.7 Initialization of the SCI .............................................................................................. 62
      5.10.8 Processing to Change the SCI Bit Rate ..................................................................... 63
      5.10.9 Processing to Calculate the SUM Data ..................................................................... 64
      5.10.10 Processing to Start the Target MCU in Boot Mode ............................................... 65
      5.10.11 Processing to Reset the Target MCU ...................................................................... 66
      5.10.12 Processing to Send a Command .............................................................................. 67
      5.10.13 Processing to Receive a Response ........................................................................... 68
      5.10.14 Copying Unsigned 4-Byte Data ............................................................................... 72
6. Sample Code....................................................................................................................................... 73

7. Reference Documents....................................................................................................................... 73
1. Specifications

The flash programmer runs on the RSK+RX63N. After starting the target RX100 Series MCU in boot mode (SCI), the flash programmer rewrites the user area in the RX100 Series 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 operating frequency select command)
  - 1 Mbps (after the program/erase status 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>

Table 1.1 Peripheral Functions and Their Applications

![Figure 1.1 Flash Programmer Usage Example](image-url)
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](image-url)
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</td>
</tr>
<tr>
<td></td>
<td>PLL: 192 MHz (main clock divided by 1 and multiplied by 16)</td>
</tr>
<tr>
<td></td>
<td>System clock (ICLK): 96 MHz (PLL divided by 2)</td>
</tr>
<tr>
<td></td>
<td>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</td>
<td>Renesas Electronics e² 2020-04</td>
</tr>
<tr>
<td>environment</td>
<td></td>
</tr>
<tr>
<td>C compiler</td>
<td>Renesas Electronics C/C++ Compiler Package for RX Family V.3.02.00</td>
</tr>
<tr>
<td></td>
<td>Compile option</td>
</tr>
<tr>
<td></td>
<td>(default settings of the integrated development environment are used)</td>
</tr>
<tr>
<td>iodefine.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 1.10</td>
</tr>
<tr>
<td>Board used</td>
<td>Renesas Starter Kit+ for RX63N (part number: R0K50563NC000BE)</td>
</tr>
</tbody>
</table>

Notes:
If the same version of the toolchain (C compiler) specified in the original project is not in the import destination, the toolchain will not be selected and an error will occur.
Check the selected status of the toolchain on the project configuration dialog.

For the setting method, refer to FAQ 3000404.

FAQ 3000404 : "Program "make" not found in PATH" error when attempting to build an imported project (e² studio)"

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]

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>UB# 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 (R20UT0508EJ1200) for details on using the FDT.

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

![Flow of Programming the RSK+RX63N](image)

**Figure 5.1 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_128KB.mot file when the user program size is 128 Kbytes
- um_all00_256KB.mot file when the user program size is 256 Kbytes
- um_all00_512KB.mot file when the user program size is 512 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_128KB.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 0xFFFE 0000 to 0xFFFF FFFF when the user program size is 128 Kbytes
- Addresses 0xFFFC 0000 to 0xFFFF FFFF when the user program size is 256 Kbytes
- Addresses 0xFFF8 0000 to 0xFFFF FFFF when the user program size is 512 Kbytes

In the example, addresses 0xFFFE 0000 to 0xFFFF FFFF of the um_all00_128KB.mot file are selected.
(4) Copy the highlighted user program data to the Windows clipboard

In the example, addresses 0xFFFE 0000 to 0xFFFF FFFF of the um_all00_128KB.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 0xFFF4 0000 and higher.

In the example, the start address of paste destination in the main.mot file is set to 0xFFF4 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 0xFFFF 0000 to 0xFFFF FFFF when the user program size is 128 Kbytes
Addresses 0xFFFF 0000 to 0xFFFF7 FFFF when the user program size is 256 Kbytes
Addresses 0xFFFF 0000 to 0xFFFB FFFF when the user program size is 512 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 128 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 (SCI) and the bit rate is automatically adjusted to connect to the MCU at 19,200 bps.

After connecting, the supported device inquiry command, device select command, and block information inquiry command are sent to obtain information of the target MCU, and the operating frequency select command is sent to change the bit rate to 1 Mbps.

The program/erase state transition command is sent to check the ID code protection of the target MCU and perform the processing for the boot mode ID code protection.

The target MCU user area is erased and then programmed according to the obtained information of the target MCU. After the user area has been programmed, 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.

---

**Figure 5.2 Flash Programmer State Transition**

1. Refer to 5.2.1 Start the MCU in Boot Mode (SCI) 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 (SCI)

(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 UB# pin of the target MCU to high.

(4) After waiting 3 ms, the flash programmer sets the RES# pin of the target MCU to high.

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

The flash programmer starts the target MCU in boot mode (SCI), 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 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.

Note 1. Restart the target MCU in boot mode (SCI) and send the bit rate automatic adjust command again.

Figure 5.4 Bit Rate Automatic Adjustment Procedure
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 the identification code for selecting the endian of data to be programmed in the user area.

The flash programmer receives a response (data starting with 30h) to the supported device inquiry command to store identification 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 Identification Codes.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>20h (supported device inquiry)</td>
<td>30h, 3Eh, 02h, ... (response to the supported device inquiry)</td>
</tr>
<tr>
<td>Store identification codes for selecting the endian of data to be programmed in the user area</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.5 Procedure to Store Identification 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 identification code corresponding to the endian of the flash programmer in the identification codes that were stored by the support device inquiry command.

After the flash programmer sends the device select command, the endian selection is completed when a response (46h) is received. When the flash programmer receives a response (46h) after sending the device select command, it completes the endian selection. When the flash programmer receives data other than the response (46h) after sending the device select command, it resets the target MCU to abort.

Figure 5.6 shows the Procedure to Select the Endian.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>10h (device select), 04h (size), XXXXh (stored identification code), XXXh (SUM)</td>
<td>46h (response to the device select command)</td>
</tr>
<tr>
<td>90h, 11h (SUM error)</td>
<td></td>
</tr>
<tr>
<td>90h, 21h (device code error)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.6 Procedure to Select the Endian
(3) The flash programmer sends the block information inquiry command to store the block configuration of the target MCU.

When the flash programmer receives a response (data starting with 36h) to the block information inquiry command, it stores the block configuration of the target MCU in the block information storage buffer. When the flash programmer receives data other than the response (data starting with 36h) to the block information inquiry command, it resets the target MCU to abort.

Figure 5.7 shows the Procedure to Store the Block Information.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>26h (block information inquiry)</td>
<td>36h, 00h, 19h, DDh, ... (response to the block information inquiry command)</td>
</tr>
<tr>
<td>Store the block information of the target MCU</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.7 Procedure to Store the Block Information
(4) The flash programmer sends the operating frequency select command to change the bit rate with the target MCU to 1 Mbps.

The flash programmer sends the operating frequency select command (3Fh) to change the bit rate to 1 Mbps. When the flash programmer receives ACK (06h) after sending the operating frequency select command, it waits the 1-bit period at the bit rate for sending the operating frequency select command and then changes the bit rate to 1 Mbps. After that, the flash programmer sends a communication confirmation data (06h) at the changed bit rate. When the flash programmer receives 06h (response to the confirmation data), it completes changing the bit rate.

When the flash programmer receives data other than the response (06h) after sending the operating frequency select command, or when it receives data other than 06h (response to the confirmation data) after sending the communication confirmation data (06h), it resets the target MCU to abort.

Figure 5.8 shows the Procedure to Change the Bit Rate.

<table>
<thead>
<tr>
<th>Flash programmer</th>
<th>Target MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>3Fh (operating frequency select)</td>
<td></td>
</tr>
<tr>
<td>07h (size)</td>
<td></td>
</tr>
<tr>
<td>27h, 10h (bit rate: 1 Mbps)</td>
<td></td>
</tr>
<tr>
<td>06h, 40h (input frequency)</td>
<td></td>
</tr>
<tr>
<td>02h (number of multipliers)</td>
<td></td>
</tr>
<tr>
<td>01h (multiplier 1)</td>
<td></td>
</tr>
<tr>
<td>01h (multiplier 2)</td>
<td></td>
</tr>
<tr>
<td>39h (SUM)</td>
<td></td>
</tr>
<tr>
<td>06h (ACK)</td>
<td></td>
</tr>
<tr>
<td>BFh, 11h (SUM error)</td>
<td></td>
</tr>
<tr>
<td>BFh, 24h (bit rate selection error)</td>
<td></td>
</tr>
<tr>
<td>Wait 1-bit period at 19,200 bps</td>
<td></td>
</tr>
<tr>
<td>Change to 1Mbps</td>
<td></td>
</tr>
<tr>
<td>06h (communication confirmation data)</td>
<td></td>
</tr>
<tr>
<td>06h (response to the confirmation data)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.8 Procedure to Change the Bit Rate
5.2.4 Check ID Code Protection

The flash programmer performs steps (1) and (2) below to check the ID code protection.

(1) The flash programmer sends the program/erase state transition command to check and store the status of the ID code protection for the target MCU.

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

After the flash programmer sends the program/erase 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 5.1 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>06h</td>
<td>00h</td>
</tr>
<tr>
<td>16h</td>
<td>01h</td>
</tr>
<tr>
<td>56h</td>
<td>02h</td>
</tr>
</tbody>
</table>

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

Figure 5.9 shows the Procedure to Check ID Code Protection by the Program/Erase 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.

The flash programmer sends the ID code check command (60h) to check the response received after sending the ID code check command and store the corresponding value in the ID code protection status buffer.

Table 5.2 lists the Responses and Values Stored in the ID Code Protection Status Buffer.

**Table 5.2 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>06h</td>
<td>00h</td>
</tr>
<tr>
<td>56h</td>
<td>02h</td>
</tr>
</tbody>
</table>

The flash programmer receives data other than values listed in Table 5.2 after sending the program/erase state transition command, reset the target MCU to abort.

Figure 5.4 shows the Bit Rate Automatic Adjustment Procedure.

![Figure 5.4 Bit Rate Automatic Adjustment Procedure](image)

**Figure 5.10 Procedure to Check ID Code Protection by the ID Code Check Command**
5.2.5 Rewrite the Target MCU User Area

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

(1) The flash programmer erases the flash memory of the target MCU for rewriting the user program in the target MCU user area.

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

The flash programmer sends 59h (block erase command) as many times as the number of blocks to erase. The number of blocks to erase is calculated from the value stored in the ID code protection status buffer as follows:

- When the stored value is 02h, the sum of values in block information storage buffer 3 and block information storage buffer 6
- When the stored value is a value other than 02h, the value in block information storage buffer 3

After sending the block erase command as many times as the number of blocks to erase, the flash programmer sends 59h 04h FFh FFh FFh FFh A7h (block erase command to end block erase). When the flash programmer receives 06h (response to the block erase command) after sending the block erase command, it completes the block erase operation. When the flash programmer receives data other than 06h (response to the block erase command), it resets the target 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](image-url)
(2) The flash programmer confirms that the target MCU has erased the flash memory successfully.

The flash programmer sends the boot mode status inquiry command (4Fh).

When the flash programmer receives a response (data starting with 5Fh) after sending the boot mode status inquiry command, the flash programmer confirms that the target MCU has erased the flash memory successfully.

When the flash programmer receives data other than the response (data starting with 5Fh), it resets the target MCU to abort.

Figure 5.12 shows the Procedure to Complete Preparation for Programming the Target MCU.

![Figure 5.12 Procedure to Complete Preparation for Programming the Target MCU](image-url)

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

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

After completion of preparation, the flash programmer sends 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 0xFFFF 0000 to 0xFFFF FFFF when the user program size is 128 Kbytes
- Addresses from 0xFFFC 0000 to 0xFFFF FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFF8 0000 to 0xFFFF FFFF when the user program size is 512 Kbytes

The range of program data (source data on the RSK+RX63N) is as follows:
- Addresses from 0xFFF4 0000 to 0xFFF5 FFFF when the user program size is 128 Kbytes
- Addresses from 0xFFF4 0000 to 0xFFF7 FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFF4 0000 to 0xFFF8 FFFF when the user program size is 512 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 the program operation. When the flash programmer receives data other than 06h, it resets the MCU to abort.
Figure 5.13 shows the Procedure to Program the User Area.

Figure 5.13   Procedure to Program the User Area
(4) 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 0xFFFE 0000 to 0xFFFF FFFF when the user program size is 128 Kbytes
- Addresses from 0xFFFC 0000 to 0xFFFF FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFF8 0000 to 0xFFFF FFFF when the user program size is 512 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 0xFFF4 0000 to 0xFFF5 FFFF when the user program size is 128 Kbytes
- Addresses from 0xFFF4 0000 to 0xFFF7 FFFF when the user program size is 256 Kbytes
- Addresses from 0xFFF4 0000 to 0xFFFB FFFF when the user program size is 512 Kbytes

Figure 5.14 shows the Procedure to Confirm Data in the User Area.
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 3 ms and then drives the RES# pin of the target MCU high.
4. The flash programmer goes into an infinite loop.

Figure 5.15 Procedure to Reset the Target MCU
5.3 File Composition

Table 5.3 lists the Files Used in the Sample Code. Files generated by the integrated development environment are not included in this table.

<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 5.4 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 5.5 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 5.6 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 5.7 lists the Option-Setting Memory Configured in the Sample Code. When necessary, set a value suited to the user system.

Table 5.7 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>
<tbody>
<tr>
<td>OFS0</td>
<td>FFFF FF8Fh to FFFF FF8Ch</td>
<td>FFFF FFFFh</td>
<td>After a reset, the IWDT stops. After a reset, the WDT stops.</td>
</tr>
<tr>
<td>OFS1</td>
<td>FFFF FF88h to FFFF FF88h</td>
<td>FFFF FFFFh</td>
<td>After a reset, voltage monitoring 0 reset is disabled. After a reset, the HOCO oscillation is disabled.</td>
</tr>
<tr>
<td>MDES</td>
<td>FFFF FF83h to FFFF FF80h</td>
<td>FFFF FFFFh</td>
<td>Little endian</td>
</tr>
</tbody>
</table>

5.5 Constants
Table 5.8 to Table 5.13 list the Constants Used in the Sample Code.
### Table 5.8 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_128KB</td>
<td>(128 * 1024)</td>
<td>Selected when the user area size of the target MCU is 128 Kbytes</td>
</tr>
<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_512KB</td>
<td>(512 * 1024)</td>
<td>Selected when the user area size of the target MCU is 512 Kbytes</td>
</tr>
<tr>
<td>TARGET_ROMVOL</td>
<td>ROMVOL_128KB</td>
<td>User area size of the target MCU (128 Kbytes selected)</td>
</tr>
<tr>
<td>TARGET_DATA_ADD</td>
<td>0xFFF400000</td>
<td>Start address for storing data programmed in the target MCU user area</td>
</tr>
<tr>
<td>READING_HEAD_ADD</td>
<td>WRITING_HEAD_ADD</td>
<td>Start address for reading the target MCU (same as the start address for programming)</td>
</tr>
<tr>
<td>MDES_ADD</td>
<td>0xFFFFFFF80</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_ACK_ID</td>
<td>(0x16)</td>
<td>ACK for enabling ID code protection is received.</td>
</tr>
<tr>
<td>RES_ACK_BERS_EXSPC</td>
<td>(0x46)</td>
<td>ACK for block erase extended specification is received.</td>
</tr>
<tr>
<td>RES_ACK_MERSMD</td>
<td>(0x56)</td>
<td>ACK for erase ready operation 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>(1000000L)</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_52US</td>
<td>((( 52. * (WT_CMT_CLOCK / WT_CMT_DIVIDE) ) / WT_BASE_US +0.5)</td>
<td>Wait time with the CMT (52 μs)</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_3MS</td>
<td>((( 3. * (WT_CMT_CLOCK / WT_CMT_DIVIDE) ) / WT_BASE_MS +0.5)</td>
<td>Wait time with the CMT (3 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 5.9  Constants Used in the Sample Code (ROMVOL_128KB 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>0xFFF80000</td>
<td>Start address for programming the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.10  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>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>0xFFFCC0000</td>
<td>Start address for programming the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.11  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>0xFFFBBFFA0</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>0xFFFBBFFA4</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>0xFFFBBFFA8</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>0xFFFBBFFAC</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>
</tbody>
</table>
### Table 5.12 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 UB#, 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 UB#, 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 UB#, 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 UB# 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 UB#, MD, and RES# of the target MCU</td>
</tr>
<tr>
<td>BTMD_PODR_INIT</td>
<td>(0x04)</td>
<td>Initial value of high level output from the UB# pin of the target MCU</td>
</tr>
</tbody>
</table>

### Table 5.13 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,RXI0)</td>
<td>SCI0.RXI0 interrupt status flag</td>
</tr>
<tr>
<td>IR_SCIn_TXIn</td>
<td>IR(SCI0,TXI0)</td>
<td>SCI0.TXI0 interrupt status flag</td>
</tr>
<tr>
<td>RXDn_PDR</td>
<td>(PORT3.PDR.BIT.B3)</td>
<td>SCI0.RXI0 pin direction control bit</td>
</tr>
<tr>
<td>RXDn_PMR</td>
<td>(PORT3.PMR.BIT.B3)</td>
<td>SCI0.RXI0 pin mode control bit</td>
</tr>
<tr>
<td>RXDn_PFS</td>
<td>P33PFS</td>
<td>SCI0.RXI0 pin function control register</td>
</tr>
<tr>
<td>RXDn_PFS_SELECT</td>
<td>(0x0B)</td>
<td>RXD0 pin function select bit setting value</td>
</tr>
<tr>
<td>TXDn_PODR</td>
<td>(PORT3.PODR.BIT.B2)</td>
<td>SCI0.TXI0 pin output data store bit</td>
</tr>
<tr>
<td>TXDn_PDR</td>
<td>(PORT3.PDR.BIT.B2)</td>
<td>SCI0.TXI0 pin direction control bit</td>
</tr>
<tr>
<td>TXDn_PMR</td>
<td>(PORT3.PMR.BIT.B2)</td>
<td>SCI0.TXI0 pin mode control bit</td>
</tr>
<tr>
<td>TXDn_PFS</td>
<td>P32PFS</td>
<td>SCI0.TXI0 pin function control register</td>
</tr>
<tr>
<td>TXDn_PFS_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<em>0.5</em>(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 5.16 shows the Structure/Union Used in the Sample Code.

typedef struct BOOT_CMD_s
{
    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 5.16 Structure/Union Used in the Sample Code
### 5.7 Variables

Table 5.14 lists the Global Variable, and Table 5.15 lists the static Variables.

#### Table 5.14 Global Variable

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>volatile uint8_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 5.15 static Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>ResponseBuffer[RES_BUF_SIZE]</td>
<td>Receive data storage buffer</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReceiveResponse</td>
</tr>
<tr>
<td>uint8_t</td>
<td>TransferMode</td>
<td>Transmit mode flag</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TransferCommand</td>
</tr>
<tr>
<td>uint8_t</td>
<td>ReceiveMode</td>
<td>Receive mode flag</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReceiveResponse</td>
</tr>
<tr>
<td>uint8_t</td>
<td>IDProtectMode</td>
<td>ID code protection status buffer</td>
<td>main</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</td>
</tr>
<tr>
<td>uint32_t</td>
<td>BlockInfoData[6]</td>
<td>Block information storage buffer</td>
<td>main</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&gt;SelectDevice[]</td>
<td>Device select command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_BlockInfo[]</td>
<td>Block information inquiry command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_OperatingFreqSel_1st[]</td>
<td>Operating frequency select command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_OperatingFreqSel_2nd[]</td>
<td>Operating frequency selection confirm command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_PStatusTransition[]</td>
<td>Program/erase 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_ErasePreparation[]</td>
<td>Erase prepare command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_BlockErase[]</td>
<td>Block erase (extended specification) command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_BootModeStatusInquiry[]</td>
<td>Boot mode state inquiry command data</td>
<td>-</td>
</tr>
<tr>
<td>uint8_t</td>
<td>CMD_ProgramPreparation[]</td>
<td>User/data area program preparation 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>
</tbody>
</table>
### Type | Variable Name | Contents | Function Used
--- | --- | --- | ---
```
| uint8_t | CMD_MemoryRead[] | Memory read command data | - |
| BOOT_CMD_t | BitRateAdjustment_1st | Bit rate automatic adjust command structure | main |
| BOOT_CMD_t | BitRateAdjustment_2nd | Bit rate automatic adjustment confirm command structure | main |
| BOOT_CMD_t | EnquiryDevice | Supported device inquiry command structure | main |
| BOOT_CMD_t | SelectDevice | Device select command structure | main |
| BOOT_CMD_t | BlockInfo | Block information inquiry command structure | main |
| BOOT_CMD_t | OperatingFreqSel_1st | Operating frequency select command structure | main |
| BOOT_CMD_t | OperatingFreqSel_2nd | Operating frequency selection confirm command structure | main |
| BOOT_CMD_t | PEstatusTransition | Program/erase state transition command structure | main |
| BOOT_CMD_t | IDCod | ID code check command structure | main |
| BOOT_CMD_t | ErasePreparation | Erase prepare command structure | main |
| BOOT_CMD_t | BlockErase | Block erase (extended specification) command structure | main |
| BOOT_CMD_t | BootModeStatusInquiry | Boot mode state inquiry command structure | main |
| BOOT_CMD_t | ProgramPreparation | User/data area program preparation command structure | main |
| BOOT_CMD_t | Program | Program command structure | main |
| BOOT_CMD_t | ProgramTermination | Program end command structure | main |
| BOOT_CMD_t | MemoryRead | Memory read command structure | main |

### 5.8 Functions

Table 5.16 lists the Functions.

#### Table 5.16 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.

#### main

<table>
<thead>
<tr>
<th>Outline</th>
<th>Main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>lcd.h, cmt_wait.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>After initialization, start the target MCU in boot mode (SCI) and rewrite the user area.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

#### peripheral_init

<table>
<thead>
<tr>
<th>Outline</th>
<th>Initialization of the peripheral functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>lcd.h, cmt_wait.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void peripheral_init(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Initialize the peripheral functions used.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

#### CMT_WaitInit

<table>
<thead>
<tr>
<th>Outline</th>
<th>Initialization of the timer for wait time with the CMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>cmt_wait.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void CMT_WaitInit(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Initialize the timer for wait time (CMT0).</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

#### CMT_WaitSet

<table>
<thead>
<tr>
<th>Outline</th>
<th>Wait time setting with the CMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>cmt_wait.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void CMT_WaitSet(uint16_t cnt)</td>
</tr>
<tr>
<td>Description</td>
<td>Set the CMCOR register to the time (μs) specified in the argument and start incrementing the CMCNT register.</td>
</tr>
<tr>
<td>Arguments</td>
<td>uint16_t cnt: Wait time</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>The minimum wait time: ( \frac{1}{(PCLKB[MHz] ÷ 512)} \approx 10.67 , \mu s )</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 Wait the time (μs) specified in the argument
Arguments uint16_t cnt Wait time
Return Value None
Remarks The minimum wait time is 1/(PCLKB[MHz]/512) ≈ 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 Initialize 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 Change the SCI bit rate from 19,200 bps to 1 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 Calculate 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>Control pins MD, UB#, and RES# to start the target MCU in boot mode (SCI).</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>Reset 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>Send 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>Receive 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>
<tr>
<td>Return Value</td>
<td>OK: Reception completed successfully NG: Timeout (1 second) or error response received</td>
</tr>
<tr>
<td>Remarks</td>
<td>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</td>
</tr>
<tr>
<td>Outline</td>
<td>Copying unsigned 4-byte data</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void *U4memcpy(void *pS1, const void *pS2)</td>
</tr>
<tr>
<td>Description</td>
<td>Copy 4 bytes of data in the source memory area to the destination memory area. If the data arrangement is little endian, reverse bytes of the unsigned 4-byte data in the destination.</td>
</tr>
</tbody>
</table>
| Arguments    |void *pS1 Address of the destination memory area  
const void *pS2 Address of the source memory area |
| Return Value |pS1 value                     |
5.10 Flowcharts

5.10.1 Main Processing and Communication Protocol Control

Figure 5.17 to Figure 5.30 show the Main Processing and Communication Protocol Control.

![Diagram of Main Processing and Communication Protocol Control](image)

Figure 5.17 Main Processing and Communication Protocol Control
Figure 5.18  Main Processing and Communication Protocol Control

1. Start up in boot mode and bit rate automatic adjustment
   - TransferMode ← INTERVAL_ON
   - ReceiveMode ← ERRLOOP_OFF
   - BitAdjust ← NG

2. Processing to start the target MCU in boot mode
   - BootModeEntry()

3. Processing to send a command
   - TransferCommand()

4. Processing to receive a response
   - ReceiveResponse()

5. Is the bit rate automatic adjustment complete?
   - No

6. Processing to send a command
   - TransferCommand()

7. Processing to receive a response
   - ReceiveResponse()

8. Set modes for transmission and reception
   - TransferMode ← INTERVAL_OFF
   - ReceiveMode ← ERRLOOP_ON

9. Processing to send a command
   - TransferCommand()

10. Processing to receive a response
    - ReceiveResponse()

11. Set modes for transmission and reception
    - TransferMode ← INTERVAL_OFF
    - ReceiveMode ← ERRLOOP_ON

12. Is the bit rate automatic adjustment complete?
    - Yes

13. Initialize the flag for checking the bit rate automatic adjustment
    - BitAdjust ← NG
Supported device inquiry

Set the device code start index of little endian

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Big endian operation? (Check MDES)
No
DataIndex ← 4
Argument
&EnquiryDevice

Yes

Set the device code start index of big endian

Copy unsigned 4-byte data
U4memcpy()

Argument
&DeviceCode &ResponseBuffer[DataIndex]

DataIndex ← DataIndex + ResponseBuffer[3] + 1

Figure 5.19  Main Processing and Communication Protocol Control
Device selection

3

Copy unsigned 4-byte data
U4memcpy()

Argument
&CMD_SelectDevice[2]
&DeviceCode

Processing to calculate the
SUM data
CalcSumData()

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

Processing to send a command
TransferCommand()

Argument
&SelectDevice

Processing to receive a response
ReceiveResponse()

Argument
&SelectDevice

4

Figure 5.20 Main Processing and Communication Protocol Control
Copy unsigned 4-byte data
U4memcpy()

Block information reference start index + 4

Block information storage index + 1

Is the block information storage index less than 6?

Yes

Store block information
Argument
&BlockInfoData[BlockInfoIndex]
&ResponseBuffer[DataIndex]

DataIndex ← DataIndex + 4

BlockInfoIndex ← BlockInfoIndex + 1

No

Set the block information reference start index
DataIndex ← 4

Clear the block information storage index

BlockInfoIndex ← 0

Processing to send a command
TransferCommand()

Argument
&BlockInfo

Processing to receive a response
ReceiveResponse()

Argument
&BlockInfo

Figure 5.21 Main Processing and Communication Protocol Control
Operating frequency 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

Figure 5.22 Main Processing and Communication Protocol Control
### Program/erase state transition

1. **Set receive mode**
   - ReceiveMode ← ERRLOOP_OFF
   - Argument: &PEstatusTransition

2. **Processing to send a command**
   - TransferCommand()

3. **Processing to receive a response**
   - ReceiveResponse()

4. **Has reception been successfully completed?**
   - No
   - **Was RES_ACK_MERSID received?**
     - Yes
       - **Set the ID code protection status buffer**
         - IDProtectMode ← 0
         - **Set the ID code protection status buffer**
           - IDProtectMode ← 1
           - **Set the ID code protection status buffer**
             - IDProtectMode ← 2
             - **Processing to reset the target MCU**
               - BootModeRelease()
               - Argument: NG (See Note 1)

5. **Was RES_ACK_ID received?**
   - Yes
   - **Set the ID code protection status buffer**
     - IDProtectMode ← 0

6. **Has reception been successfully completed?**
   - Yes
   - **Was RES_ACK_MERSID received?**
     - Yes
     - **Set the ID code protection status buffer**
       - IDProtectMode ← 0
     - **Set the ID code protection status buffer**
       - IDProtectMode ← 1
     - **Set the ID code protection status buffer**
       - IDProtectMode ← 2
     - **Processing to reset the target MCU**
       - BootModeRelease()
       - Argument: NG (See Note 1)

7. **Processing to reset the target MCU**
   - BootModeRelease()
   - Argument: NG

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

---

**Figure 5.23   Main Processing and Communication Protocol Control**
Note 1. The flash programmer goes into an infinite loop by the BootModeRelease() function and does not return to the caller.

Figure 5.24 Main Processing and Communication Protocol Control
Erase preparation and block erase

Processing to send a command
TransferCommand()

Argument &ErasePreparation

Processing to receive a response
ReceiveResponse()

Argument &ErasePreparation

Set the EraseIndexMax variable
EraseIndexMax ← 1

IDProtectMode = 2

Is the ID code protection status buffer 2?

No

Yes

Set the EraseIndexMax variable
EraseIndexMax ← 2

Clear the EraseIndex variable
EraseIndex ← 0

Figure 5.25 Main Processing and Communication Protocol Control
Figure 5.26  Main Processing and Communication Protocol Control
Erase preparation and block erase

10

Set the end of block erase
memset()

Argument
&CMD_BlockErase[2]
0xFF
4ul

Processing to calculate the SUM data
CalcSumData()

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

Processing to send a command
TransferCommand()

Argument
&BlockErase

Processing to receive a response
ReceiveResponse()

Argument
&BlockErase

Boot mode status inquiry

Processing to send a command
TransferCommand()

Argument
&BootModeStatusInquiry

Processing to receive a response
ReceiveResponse()

Argument
&BootModeStatusInquiry

Figure 5.27  Main Processing and Communication Protocol Control
User/data area program preparation and program

11  
Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

Clear the WriteIndex variable

Is WriteIndex less than WRITING_TIME?

Yes

Set the AddressIndex variable

Calculate the program address

Copy unsigned 4-byte data
U4memcpy()

Set the program data
memcpy()

Processing to calculate the SUM data
CalcSumData()

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

WriteIndex ← WriteIndex + 1

No

Set the AddressIndex variable

Calculate the program address

Copy unsigned 4-byte data
U4memcpy()

Set the program data
memcpy()

Processing to calculate the SUM data
CalcSumData()

Processing to send a command
TransferCommand()

Processing to receive a response
ReceiveResponse()

WriteIndex ← WriteIndex + 1

Argument &ProgramPreparation

Argument &ProgramPreparation

WriteIndex ← 0

AddressIndex ← WriteIndex << 8 :256-fold increase

WriteAddress ← WRITING_HEAD_ADD + AddressIndex

Set the program address
Argument &CMD_Program[1]
&WriteAddress

Argument &CMD_Program[5]
TARGET_DATA_ADD + AddressIndex
256ul

CMD_Program[261] ← CalcSumData()

Argument &CMD_Program[0]
261ul

Argument &CMD_Program[1]
&WriteAddress

Argument &CMD_Program[0]
256ul

Argument &CMD_Program[0]
261ul

Argument &ProgramTermination

Argument &Program

Argument &Program

Argument &Program

Argument &Program

Argument &Program

Figure 5.28 Main Processing and Communication Protocol Control
Memory read

12

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

Copy unsigned 4-byte data

U4memcpy()

Set the read address

Argument

&CMD_MemoryRead[3]

&ReadAddress

Processing to calculate the SUM data

CalcSumData()

Argument

CMD_MemoryRead[11] ← CalcSumData()

11ul

Processing to send a command

TransferCommand()

Argument

&MemoryRead

Processing to receive a response

ReceiveResponse()

Argument

&MemoryRead

Compare the read data

strncmp()

Argument

&ResponseBuffer[5]

TARGET_DATA_ADD + AddressIndex

256ul

Do the program data and read data match?

No

Yes

Processing to reset the target MCU

BootModeRelease()

(see Note 1)

Argument

NG

ReadIndex + 1

ReadIndex ← ReadIndex + 1

13

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

Figure 5.29  Main Processing and Communication Protocol Control
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 5.30 Main Processing and Communication Protocol Control
5.10.2 Initialization of the Peripheral Functions

Figure 5.31 shows the Initialization of the Peripheral Function.

```
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 PFSWE bit

Enable writing to the PFS register

Set the port for output of debug LCD data

Initialize the debug LCD
init_LCD()

Initialize the timer for wait time with the CMT
CMT_WaitInit

Initialize the SCI
SCI_Init()

Disable writing to the PFS register

Disable writing to the PFSWE bit

Enable register protection

Debug LCD output
Display_LCD()

return
```

Set the pin 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

Argument
Output "Start" to LCD_LINE1
Output "... " to LCD_LINE2

Figure 5.31 Initialization of the Peripheral Functions
5.10.3 Initialization of the Timer for Wait Time with the CMT

Figure 5.32 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 : CMT0 and CMT1 module clocks are enabled.

Stop counting of the CMT0
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] bit = 11b : PCLK/512
CMI bit = 1 : Compare match interrupt (CMI0) enabled.

Clear CMT0 count
CMCNT register ← 0000h

Set the CMT0 interrupt priority level
IPR004 register ← 03h
IPR[3:0] bit = 0011b : CMT0.CMI0 interrupt priority level 3

Enable the CMT0 interrupt request
IER03 register
IEN4 bit ← 1 : CMT0.CMI0 interrupt request is enabled.

Clear the CMT0 interrupt request
IR028 register
IR flag ← 0 : CMT0.CMI0 interrupt request is cleared.

return
```

Figure 5.32 Initialization of the Timer for Wait Time with the CMT
5.10.4 Setting Wait Time with the CMT

Figure 5.33 shows Setting Wait Time with the CMT.

![Setting Wait Time with the CMT Diagram](image)

### Argument
- **uint16_t cnt**: Wait time

### CMT_WaitSet()
- **CMT_INTERRUPT_FLAG**: 1
- **CMSTR0 register**: STR0 bit ← 0 : CMT0 count is stopped.
- **CMCNT register**: ← 0000h
- **CMT_InterruptFlag**: ← 1
- **Wait time enable flag**: Set
- **cnt**: Decrement
- **CMT0**: Set
- **IR028 register**: IR flag ← 0 : CMT0.CMI0 interrupt request is cleared.
- **CMSTR0 register**: STR0 bit ← 1 : CMT0 count is started.

---

Figure 5.33 Setting Wait Time with the CMT
5.10.5  Wait Processing with the CMT

Figure 5.34 shows Wait Processing with the CMT.

Argument

\[ \text{CMT\_Wait()} \]

\[ \text{cnt} : \text{Wait time} \]

Argument

\[ \text{CMT\_WaitSet()} \]

\[ \text{cnt} : \text{Wait time} \]

Wait until the CMI0 interrupt occurs

Is the wait time enable flag other than 1?

Yes

return

No

Figure 5.34  Wait Processing with the CMT
5.10.6 Interrupt Handling for CMI0 in CMT0

Figure 5.35 shows Interrupt Handling for CMI0 in CMT0.

```
Excep_CMT0_CMI0()

Clear the wait enable flag

Stop CMT0 count

return

CMT_InterruptFlag ← 0

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

Figure 5.35  Interrupt Handling for CMI0 in CMT0
5.10.7 Initialization of the SCI

Figure 5.36 shows SCI Initialization.

```
SCI_Init

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 SCI module stop
MSTP_SCI ← 0 : SCI is released from the module stopped state

Disable transmission/reception and the SCI interrupt request
SCIn.SCR register ← 00h
CKE[1:0] = 00b : On-chip baud rate generator
TEIE = 0 : TEI interrupt request is disabled
RE = 0 : Serial reception is disabled
TE = 0 : Serial transmission is disabled
RIE = 0 : RXI and ERI interrupt requests are disabled
TIE = 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] = 00b : PCLK clock
MP = 0 : Multi-processor communications function is disabled
STOP = 0 : 1 stop bit
PE = 0 : No parity
CHR = 0 : Selects 8 bits as the data length
CM = 0 : Asynchronous mode
SCIn.SCMR register ← F2h
SMIF = 0 : Serial communications interface mode
SINV = 0 : TDR contents are transmitted as they are. Receive data is stored as it is in RDR
SDIR = 0 : Transfer with LSB first

Set the bit rate
SCIn.BRR register ← BRR_SET(19200) : (48 MHz/(32×2¹×19200 bps)) - 1

Clear the SCI interrupt request
IR_SCI_RXIn ← 0 : SCI RXI interrupt not requested
IR_SCI_TXIn ← 0 : SCI TXI interrupt not requested

Enable transmission/reception and SCI interrupt request
SCIn.SCR register ← F0h
RE = 1 : Serial reception is enabled
TE = 1 : Serial transmission is enabled
RIE = 1 : RXI and ERI interrupt requests are enabled
TIE = 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

return
```

Figure 5.36 SCI Initialization
5.10.8 Processing to Change the SCI Bit Rate

Figure 5.37 shows Processing to Change the SCI Bit Rate.

```
SCI_change()

Wait Processing with the CMT
CMT_Wait()

Set port mode

Disable transmission/reception and
the SCIn interrupt request

Set the bit rate

Enable transmission/reception and
the SCIn interrupt request

Clear the SCIn interrupt request

Set port mode

return
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT_52US</td>
<td>Wait time 1 bit period at 19200bps ≈ 52 µs</td>
</tr>
<tr>
<td>TXDn_PMR ← 0</td>
<td>TXDn: Used as a general I/O port</td>
</tr>
</tbody>
</table>

SCIn, SCR register ← 00h
- CKE[1:0] = 00b: On-chip baud rate generator
- TEIE = 0: TEI interrupt request is disabled
- RE = 0: Serial reception is disabled
- TE = 0: Serial transmission is disabled
- RIE = 0: RXI and ERI interrupt requests are disabled
- TIE = 0: TXI interrupt request is disabled

SCIn, BRR register ← BRR_SET(1000000) : (48MHz/(32×2⁻¹×1000000 bps)) - 1

SCIn, SCR register ← F0h
- RE = 1: Serial reception is enabled
- TE = 1: Serial transmission is enabled
- RIE = 1: RXI and ERI interrupt requests are enabled
- TIE = 1: TXI interrupt request is enabled

IR_SCIn_TXIn ← 0: SCIn.RXIn interrupt not requested

TXDn_PMR ← 1: TXDn: Used for peripheral functions

Figure 5.37 Processing to Change the SCI Bit Rate
5.10.9 Processing to Calculate the SUM Data

Figure 5.37 shows Processing to Calculate the SUM Data.

```
CalcSumData()
Argument
  uint8_t *pData : Data address for SUM
  uint32_t Length : Amount of data for SUM

Clear the loop counter
loop ← 0
Clear the checksum
Checksum ← 0

Is the loop counter value less than the amount of data for SUM?
Yes
Calculate the checksum
Checksum ← CheckSum + *pData;
pData ← pData + 1
Loop counter value + 1
loop ← loop + 1

No
Calculate the SUM value
Checksum ← 0 - CheckSum
```

Figure 5.38 Processing to Calculate the SUM Data
5.10.10 Processing to Start the Target MCU in Boot Mode

Figure 5.39 shows 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
    Argument
    WAIT_3MS : Wait time
                3 ms for RES# pulse width after power-on
  Release the target MCU reset
    RES_PIN ← 1
    Argument
    WAIT_400MS : Wait time
                 400 ms for wait time until the start of communication
  Wait processing with the CMT
    CMT_Wait()
  Wait processing with the CMT
    CMT_Wait()
  return
```

Figure 5.39 Processing to Start the Target MCU in Boot Mode
5.10.11 Processing to Reset the Target MCU

Figure 5.40 shows Processing to Reset the Target MCU.

```
BootModeRelease()
Set the MD pin to the high level
Reset the target MCU
Wait processing with the CMT
Release the target MCU reset

Is the normal value output to the debug LCD line 2?
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()
```

Argument
uint8_t mode : Selects the output pattern of debug LCD line 2
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
Argument
WAIT_3MS : Wait time
3 ms for the RES# pulse width after power-on
RES_PIN ← 1 : Set the RES# pin of the target MCU to the high level
5.10.12 Processing to Send a Command

Figure 5.41 shows Processing to Send a Command.

```
TransferCommand()

Clear the transmit counter
TransferCount ← 0

Is the transmit counter value less than the command size?

Yes
Wait until data has been transmitted
IR_SCIn_TXIn ← 0
SCIn.TDR register ← pCmd->Command[TransferCount]
Wait until SCIn.SSR.BIT.TEND becomes 1

No
Wait SCIn.TXIn interrupt request
Clear the SCIn.TXIn interrupt request
Set the transmit data
Wait until data has been transmitted

Is an interval set for transmission?

Yes
Wait processing with the CMT
CMT_Wait()

No
Transmit counter value + 1
TransferCount ← TransferCount + 1

Argument
BOOT_CMD_t *pCmd : Address for command structure to be transmitted

Argument
WAIT_1MS : Wait time
1 ms (interval for transmit data)
```
5.10.13 Processing to Receive a Response

Figure 5.42 to Figure 5.45 show Processing to Receive a Response.

![Flowchart of Processing to Receive a Response]

- **ReceiveResponse**
- **Argument**
  - `BOOT_CMD_t *pCmd` : Address for the command structure to be received
- **Initialize the timeout counter**
  - `TimeOutCount ← 10`
- **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`

**3**

**Is the receive counter value less than the response size, and is the return value OK?**

- **Yes**
  - **Set the wait time with the CMT**
  - `CMT_WaitSet()`
  - **Argument**
    - `WAIT_100MS` : Wait time 100 ms

- **4**
- **1**
1

No
Has a receive error occurred?

Yes

Clear the error flag *1

Set the return value (NG)

No
Has the wait time elapsed?

Yes

Timeout counter - 1

No
Is the timeout counter value equal to or greater than 0?

Yes

Set the return value (NG)

Set the wait time with the CMT
CMT_WaitSet()

No
Are the conditions below satisfied?
- SCIn.RXIn interrupt is not requested
- Return value is OK?

Yes

Clear the SCIn.RXIn interrupt request

2

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 = 0 : No parity error occurred
FER = 0 : No framing error occurred
ORER = 0 : No overrun error occurred

ret ← NG

TimeOutCount ← TimeOutCount - 1

: Timeout (1s = 100 ms × 10 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 5.43  Processing to Receive a Command
Figure 5.44  Processing to Receive a Command
Is the error processing (infinite loop) selected, and is the return value NG?

Yes

Processing to reset the target MCU

BootModeRelease()

(See Note 1)

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

No

return (ret)

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

Figure 5.45  Processing to Receive a Command
5.10.14 Copying Unsigned 4-Byte Data

Figure 5.46 show Copying Unsigned 4-Byte Data.

```
void *pS1 Destination address of the memory area for copying
const void *pS2 Source address of the memory area for copying

Copy 4-byte data
memcpy()

Little endian operation?
(Check MDES)

No

Yes

Byte reverse of 4-byte data
revl()

*(uint32_t *)MDES_ADD == 0xFFFFFFFF
Little endian operation?
Yes
No

*(uint32_t *)pS1 ← revl(*(uint32_t *)pS1)
Argument
*(uint32_t *)pS1

return (pS1)
```

Figure 5.46 Copying Unsigned 4-Byte Data
6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

User’s Manual: Hardware
- RX110 Group User's Manual: Hardware Rev.1.00 (R01UH0421EJ)
- RX111 Group User’s Manual: Hardware Rev.1.10 (R01UH0365EJ)
- RX63N Group, RX631 Group User's Manual: Hardware Rev.1.80 (R01UH0041EJ)
The latest versions 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 (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/
# REVISION HISTORY
RX100 Series Application Note  
RX100 Series Flash Programmer (SCI)  
Using the Renesas Starter Kit+ for RX63N

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>June 02, 2014</td>
<td>—</td>
<td>—</td>
<td>First edition issued</td>
</tr>
<tr>
<td>1.10</td>
<td>Aug, 20, 2020</td>
<td>—</td>
<td>—</td>
<td>Update the toolchain version</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

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

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

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

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

6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (Max.) and VIH (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (Max.) and VIH (Min.).

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

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

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

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

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

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

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depend on the product's quality grade, as indicated below.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

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

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

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

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

9. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

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

11. This document shall not be reprinted, 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.

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

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

(Rev.4.0-1 November 2017)

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

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

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