RL78/G23

Self-Programming Using Boot Swapping via UART communications

Introduction

This application note gives the outline of self-programming via UART communications.

This application note explains how the flash self-programming code (Renesas Flash Driver RL78 Type01) is used to rewrite the boot area in flash memory and perform boot swapping.

Target Device

RL78/G23

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

1. Specifications ................................................................................................................................. 4
  1.1 Outline ........................................................................................................................................ 4
    1.1.1 Outline of the Flash Self-Programming Code (Renesas Flash Driver RL78 Type01) .......... 4
    1.1.2 Code Flash Memory ............................................................................................................... 5
    1.1.3 Flash Memory Self-Programming .......................................................................................... 7
    1.1.4 Boot Swap Function .............................................................................................................. 7
    1.1.5 Flash Memory Reprogramming ............................................................................................. 9
    1.1.6 Flash Shield window ............................................................................................................ 10
    1.1.7 Communication Specifications ............................................................................................ 10
    1.1.8 How to obtain the flash self-programming code ................................................................. 11
  1.2 Operation Outline ....................................................................................................................... 12

2. Operation Check Conditions ......................................................................................................... 14

3. Description of the Hardware ......................................................................................................... 15
  3.1 Hardware Configuration Example ............................................................................................. 15
  3.2 List of Pins to be Used ............................................................................................................. 16

4. Software Explanation ..................................................................................................................... 17
  4.1 List of Option Byte Settings ...................................................................................................... 17
  4.2 Startup routine settings ............................................................................................................ 18
  4.2.1 Definition of the section for the stack area (.stack_bss) ..................................................... 18
  4.2.2 Deploying the Rewrite Programs in the RAM Area ............................................................. 19
  4.3 On-chip Debug Security ID ....................................................................................................... 20
  4.4 Resources Used by the Sample Program .................................................................................. 20
    4.4.1 List of Sections in the ROM Area ......................................................................................... 20
    4.4.2 List of Sections in the RAM Area ....................................................................................... 20
  4.5 List of Constants ....................................................................................................................... 21
  4.6 Enumerated type ....................................................................................................................... 22
  4.7 List of Variables ....................................................................................................................... 22
  4.8 List of Functions ....................................................................................................................... 23
  4.9 Function Specifications ............................................................................................................. 25
  4.10 Flowcharts ............................................................................................................................... 32
    4.10.1 Main Processing .................................................................................................................. 32
    4.10.2 Initialization Processing for RFD RL78 Type01 ................................................................. 34
    4.10.3 START Command Processing ............................................................................................ 35
    4.10.4 WRITE Command Processing ........................................................................................... 36
    4.10.5 END Command Processing ............................................................................................... 37
    4.10.6 Range Erase Processing for the Code Flash Memory ....................................................... 38
    4.10.7 Block Erase Processing for the Code Flash Memory ......................................................... 39
    4.10.8 Write-and-verify Processing for the Code Flash Memory ................................................. 40
    4.10.9 Write Processing for the Code Flash Memory ................................................................... 41
    4.10.10 Verify Processing for the Code Flash Memory ............................................................... 42
    4.10.11 Sequence End Processing for the Code Flash Memory .................................................. 43
    4.10.12 Sequence End Processing for the Extra Area ................................................................. 45

How to obtain the flash self-programming code ........................................................................... 11
Communication Specifications ......................................................................................................... 10
Flash Memory Reprogramming ....................................................................................................... 9
Startup routine settings .................................................................................................................... 18
Enumerated type ............................................................................................................................. 22
List of Variables .............................................................................................................................. 22
List of Functions ............................................................................................................................. 23
Function Specifications .................................................................................................................. 25
Flowcharts ................................................................................................................................------- 32
Main Processing ............................................................................................................................ 32
Initialization Processing for RFD RL78 Type01 ........................................................................... 34
4.10.13 Boot Swapping Execution Processing ................................................................. 47
4.10.14 Callback Processing at a Reception Completion Interrupt for UART0 ....................... 49
4.10.15 Callback Processing at a Sending Completion Interrupt for UART0 .......................... 50
4.10.16 Command Reception Processing by UART0 ......................................................... 51
4.10.17 Command Analysis Processing by UART0 .......................................................... 52
4.10.18 Data Reception Processing by UART0 ............................................................... 53
4.10.19 Data Sending Processing by UART0 ................................................................. 54
4.10.20 Normal Response Sending Processing by UART0 .................................................. 55
4.10.21 Callback Processing at a Sending Completion Interrupt for IICA0 ......................... 56
4.10.22 Callback Processing at a Sending Error Interrupt for IICA0 ................................. 57
4.10.23 Processing to Initialize the LCD Module .............................................................. 58
4.10.24 Processing to Clear Display for the LCD Module .................................................. 59
4.10.25 Processing to Send Strings to the LCD Module ...................................................... 60
4.10.26 Command Sending Processing for the LCD Module ............................................ 61
4.10.27 Processing to Send Data to the LCD Module ......................................................... 62
4.10.28 Communication End Flag Setting for the LCD Module ......................................... 63
4.10.29 Communication End Wait Processing for the LCD Module ..................................... 64

5. Sample code ................................................................................................................. 65

6. Reference Documents .................................................................................................. 65

Revision History ................................................................................................................ 66
1. Specifications

Outline

First, the sample program displays the current program version on the LCD module. Then, when it receives the START command via UART communications, it turns LED1 on (flash memory is being accessed) and enters the code flash programming mode. After that, the sample program erases the data that has been written to the code flash memory's boot cluster 1 (04000H to 07FFFH) and waits for the WRITE command.

When the sample program receives the WRITE command together with the rewrite data, it rewrites the contents of boot cluster 1. When the sample program completes rewriting and receives the END command, it turns LED1 off. If all processing performed before this point in time has terminated normally, the sample program generates an internal reset and performs boot swamping. After the sample program restarts, it displays the version of the new (rewritten) program on the LCD module.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial array unit UART0</td>
<td>Obtain of rewrite data</td>
</tr>
<tr>
<td>Serial interface IICAO</td>
<td>Communication with LCD module</td>
</tr>
</tbody>
</table>
1.1.2 Code Flash Memory

The configuration of the RL78/G23 (R7F100GLG) code flash memory is shown below.

Caution: When the boot swap function is used, the option byte area (000C0H to 000C3H) in boot cluster 0 is swapped with the option byte area (040C0H to 040C3H) in boot cluster 1. Accordingly, place the same values in the area (040C0H to 040C3H) as those in the area (000C0H to 000C3H) when using the boot swap function.
The features of the RL78/G23 code flash memory are summarized below.

### Table 1-2  Features of the Code Flash Memory

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum unit of erasure and verification</td>
<td>1 block (2048 bytes)</td>
</tr>
<tr>
<td>Minimum unit of programming</td>
<td>1 word (4 bytes)</td>
</tr>
<tr>
<td>Security functions</td>
<td>Block erasure, programming, and boot cluster 0 reprogramming are supported.</td>
</tr>
<tr>
<td></td>
<td>(They are enabled at shipment)</td>
</tr>
<tr>
<td></td>
<td>It is possible to disable reprogramming and erasure outside the specified window only at flash memory self-programming time using the flash shield window.</td>
</tr>
<tr>
<td></td>
<td>Security settings programmable using the flash self-programming code (Renesas Flash Driver RL78 Type01)</td>
</tr>
</tbody>
</table>

Caution: The boot cluster 0 reprogramming protection setting and the security settings for outside the flash shield window are disabled during flash memory self-programming.
1.1.3 Flash Memory Self-Programming

The RL78/G23 is provided with the flash self-programming code for flash memory self-programming. Flash memory self-programming is accomplished by calling functions of the flash self-programming code from the reprogramming program.

The flash self-programming code for the RL78/G23 controls flash memory reprogramming using a sequencer (a dedicated circuit for controlling flash memory). The code flash memory cannot be referenced while control by the sequencer is in progress. When the user program needs to be run while the sequencer control is in progress, therefore, it is necessary to relocate part of the segments for the flash self-programming code and the reprogramming program in RAM when erasing or reprogramming the code flash memory or making settings for the security flags. If there is no need to run the user program while the sequencer control is in progress, it is possible to keep the flash self-programming code and reprogramming program on ROM (code flash memory) for execution.

1.1.4 Boot Swap Function

When reprogramming of the area where vector table data, the basic functions of the program, and flash self-programming code are allocated fails due to a temporary power blackout or a reset caused by an external factor, the data that is being reprogrammed will be corrupted, as the result of which the restarting of the user program or reprogramming cannot be accomplished when a reset is subsequently performed. This problem is be avoided by the introduction of the boot swap function.

The boot swap function swaps between boot cluster 0 which is the boot program area and boot cluster 1 which is the target of boot swapping. A new program is written into boot cluster 1 before reprogramming is attempted. This boot cluster 1 is swapped with boot cluster 0 and boot cluster 1 is designated as the boot program area. In this configuration, even when a temporary power blackout occurs while the boot program area is being reprogrammed, the system boot will start at boot cluster 1 on the next reset start, thus ensuring the normal execution of the programs.

The outline image of boot swapping is shown in the figure below.
Below is an image of boot swapping.

Figure 1-3  Outline of Boot Swapping

(1) Erasing boot cluster 1
Call the r_CF_EraseBlock function to erase boot cluster 1 (blocks 8 to 15).

(2) Writing the new boot program into boot cluster 1
Call the r_CF_WriteData function to write the new boot program into boot cluster 1 and call the r_CF_VerifyData function to verify boot cluster 1.

The steps that have been performed up to here ensure that the programs will run normally even when the programming of the new boot program fails due to a temporary power blackout or reset because the system boot is started by the old boot program.

(3) Setting the boot swap bit
Call the r_RequestBootSwap function to invert the state of the boot flag.
When a temporary power blackout or reset occurs after the state of the boot flag is inverted, the programs will run normally because the system boot is started by the new boot program whose reprogramming has been completed.

(4) When a reset occurs
When a reset occurs, boot clusters 0 and 1 are swapped.

(5) Boot swapping completed
1.1.5 Flash Memory Reprogramming

This subsection describes the outline image of reprogramming using the flash memory self-programming technique. The flash memory self-programming program is located in boot cluster 0.

In this application note, the rewrite target is limited to the boot area.

Figure 1-4 Outline of Flash Memory Reprogramming
1.1.6 Flash Shield window

The flash shield window is one of security mechanisms used for flash memory self-programming. It disables the write and erase operations on the areas outside the designated window only during flash memory self-programming.

The figure below shows the outline image of the flash shield window on the area of which the start block is 08H and the end block is 0FH.

![Outline of the Flash Shield Window](image)

1.1.7 Communication Specifications

This application note explains how to perform self-programming via UART communications. The sample program performs the processing corresponding to the received command (START, WRITE, or END). If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates abnormally, the sample program displays "ERROR!" on the LCD module and terminates processing without sending data. The following shows the UART communication settings and the specifications of each command.

<table>
<thead>
<tr>
<th>UART Communication Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data bit length [bit]</td>
</tr>
<tr>
<td>Data transfer direction</td>
</tr>
<tr>
<td>Parity setting</td>
</tr>
<tr>
<td>Transfer rate [bps]</td>
</tr>
</tbody>
</table>

- **START command**

When the sample program receives the START command, it initializes the self-programming settings. If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates abnormally, the sample program does not send data.
WRITE command

When the sample program receives the WRITE command, it writes the received data to the flash memory. At this time, the sample program verifies the written data every 256 bytes. If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates normally, the sample program does not send data.

<table>
<thead>
<tr>
<th>START code (01H)</th>
<th>Date length (0102H)</th>
<th>Command (03H)</th>
<th>Date (256 byte)</th>
<th>Checksum (1 byte)</th>
</tr>
</thead>
</table>

END command

When the sample program receives the END command, it sends 01H as notification of response. The sample program then reverses the boot flag. If the processing terminates normally, the sample program generates a reset and performs boot swapping. If the processing terminates abnormally, it does not perform boot swapping.

<table>
<thead>
<tr>
<th>START code (01H)</th>
<th>Date length (0002H)</th>
<th>Command (04H)</th>
<th>Date (None)</th>
<th>Checksum (1 byte)</th>
</tr>
</thead>
</table>

Abnormal termination

The sample program displays "ERROR!" on the LCD module and terminates processing.

Checksum calculation method

The checksum is calculated by using the "32-bit addition calculation method".

The low-order 8 bits of the results of sequentially adding a value by one byte from 00000000H is used as the checksum for the command or data.

1.1.8 How to obtain the flash self-programming code

Before starting compilation, download the latest version of the flash self-programming code (Renesas Flash Driver RL78 Type01) and copy the file to the RFD_RL78_TYPE1 folder.

You can obtain the flash self-programming code from the following URL:
### Operation Outline

This application note explains how to perform self-programming via UART communications.

(1) Initial settings

Initial port settings
- Set P53 as the output port (initial value: high level, LED1 turned off).

Initial settings of the serial array unit:
- Use channels 0 and 1 for a UART.
- Use the P12/TxD0 pin for data output. Use the P11/RxD0 pin for data input.
- Set the operation clock for CK00. Set the clock source for fCLK/2.
- Set an interrupt source for the transfer completion interrupt.
- Specify the following settings: No parity bit, transfer order = LSB first, stop bit length = 1 bit, data length = 8 bits
- Set non-reverse (standard) sending.
- Set the baud rate to 115,200 bps.

Initial settings of the IICA serial interface:
- Use the IICA0 (P60/SCLA0 and P61/SDAA0 pins).
- Set the operation clock of the IICA0 for fCLK/2.
- Set the local address for 10H.
- Set the operation mode to "standard".
- Set the transfer click to 80,000 bps.
- Permit the INTIICA0 interrupt.

Initial settings of the LCD module and display of the current program version:
- Display the string of the LCD_STRING constant on the LCD module.

Initialization of Renesas Flash Driver RL78 Type01

(2) Processing of the START command

- Set the P53 pin to low output level and turn LED1 (flash memory being accessed) on.
- Use the r Cf_EraseBlock function to erase the data of boot cluster 1 (04000H to 07FFFH). If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates abnormally, the sample program does not send data.
(3) Processing of the WRITE command

- Receive the data to be written (256 bytes).
- Use the r_CF_WriteData function to write the received data to the write-destination address. Increase the write-destination address by the size of written data.
- Use the r_CF_VerifyData function to verify the written data against the received data every 256 bytes.
- If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates abnormally, the sample program does not send data.

(4) Processing of the END command

- Set the P53 pin to high level output and turn LED1 (flash memory being accessed) off.
- Send 01H, which indicates normal response.
- Use the r_RequestBootSwap function to reverse the value of the boot flag. If ret_value is normal, the sample program generates an internal reset. The generated internal reset will exchange boot clusters 0 and 1. If the processing terminates normally, the sample program reverses the boot flag to generate a reset and performs boot swapping. If the processing terminates abnormally, the sample program does not perform boot swapping.

(5) Handling of abnormal termination

- The sample program displays “ERROR!” on the LCD module and terminates processing.

Note 1: If data has already been completely written up to the last address (07FFFH) of boot cluster 1, the sample program writes no more data even when a new WRITE command is received.

Note 2: When the sample program receives the END command (04H), it always sends 01H, which indicates normal response, and sets the P52 to high level output (LED1 turned off). The r_RequestBootSwap function is run to perform boot swapping.

Note 3: If self-programming does not terminate normally, the sample program displays "ERROR!" on the LCD module and performs no subsequent processing.
2. Operation Check Conditions

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

Table 2-1  Operation Check Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller used</td>
<td>RL78/G23 (R7F100GLG)</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>High-speed on-chip oscillator (fIH): 32MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3 V (can be operated at 3.1 V to 5.5 V)</td>
</tr>
<tr>
<td></td>
<td>LVD operations ($V_{LVD}$): Reset mode</td>
</tr>
<tr>
<td></td>
<td>At rising edge TYP. 1.90 V</td>
</tr>
<tr>
<td></td>
<td>At falling edge TYP. 1.86 V</td>
</tr>
<tr>
<td>Integrated development environment (CS+)</td>
<td>CS+ for CC V8.05.00 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>C compiler (CS+)</td>
<td>CC-RL V1.10.00 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>Integrated development environment (e2 studio)</td>
<td>e2studio V2021-04(21.4.0) from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>C compiler (e2 studio)</td>
<td>CC-RL V1.10.00 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>Integrated development environment (IAR)</td>
<td>IAR Embedded Workbench for Renesas RL78 V4.21.1 from IAR Systems Corp.</td>
</tr>
<tr>
<td>C compiler (IAR)</td>
<td>IAR C/C++ Compiler for Renesas RL78 V4.21.1 from IAR Systems Corp.</td>
</tr>
<tr>
<td>Board support package (BSP)</td>
<td>V1.0.1 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td>Board to be used</td>
<td>RL78/G23-64p Fast Prototyping Board, RTK7RLG230CLG000BJ</td>
</tr>
</tbody>
</table>
3. Description of the Hardware

Hardware Configuration Example

Figure 3-1 shows an example of the hardware configuration used for this application note.

Figure 3-1   Hardware Configuration

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

2. VDD must be held at not lower than the reset release voltage (VLVD0) that is specified as LVD0.
List of Pins to be Used

Table 3-1 lists pins to be used and their functions.

<table>
<thead>
<tr>
<th>Pin name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P12/TxD0</td>
<td>Output</td>
<td>Pin for sending UART serial data</td>
</tr>
<tr>
<td>P11/RxD0</td>
<td>Input</td>
<td>Pin for receiving UART serial data</td>
</tr>
<tr>
<td>P53</td>
<td>Output</td>
<td>Pin used to turn on or off LED1, which indicates the access status of flash memory</td>
</tr>
<tr>
<td>P60/SCLA0、P61/SDAA0</td>
<td>Input/Output</td>
<td>Pin used for I2C communication with the LCD module</td>
</tr>
</tbody>
</table>

Caution: In this application note, only the pins used are processed. When actually creating a circuit, perform pin processing appropriately and design it so that it satisfies the electrical characteristics.
4. Software Explanation

List of Option Byte Settings

Table 4-1 summarizes the settings of the option bytes.

<table>
<thead>
<tr>
<th>Address</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000C0H/040C0H</td>
<td>11101111B</td>
<td>Disables the watchdog timer. (Stops counting after the release from the reset status.)</td>
</tr>
<tr>
<td>000C1H/040C1H</td>
<td>11111110B</td>
<td>LVD operations ($V_{LVD}$): Reset mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>At rising edge TYP. 1.90 V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>At falling edge TYP. 1.86 V</td>
</tr>
<tr>
<td>000C2H/040C2H</td>
<td>11101000B</td>
<td>HS mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High-speed on-chip oscillator clock: 32MHz</td>
</tr>
<tr>
<td>000C3H/040C3H</td>
<td>10000101B</td>
<td>Enables the on-chip debugger</td>
</tr>
</tbody>
</table>

The option bytes of the RL78/G23 comprise the user option bytes (000C0H to 000C2H) and on-chip debug option byte (000C3H).

The option bytes are automatically referenced and the specified settings are configured at power-on time or the reset is released. When using the boot swap function for self-programming, it is necessary to set the same values that are set in 000C0H to 000C3H also in 040C0H to 040C3H because the bytes in 000C0H to 000C3H are swapped with the bytes in 040C0H to 040C3H.
Startup routine settings

4.2.1 Definition of the section for the stack area (stack_bss)

In the sample program, the data to be written to boot cluster 1 is saved in a local variable. Because local variables are placed in stack areas, you need to modify "cstart.asm" so that any stack area of your choice is secured and the stack area is initialized.

```assembly
;SIF (__RENESAS_VERSION__ < 0x01010000) Add ';' to the first line and comment out
;-----------------------------------------------------
; stack area
;-----------------------------------------------------
; !!! [CAUTION] !!!
; Set up stack size suitable for a project.
.SECTION .stack_bss, BSS
_stackend:
- .DS 0x200
_stacktop:
;SENDIF Add ';' to the first line and comment out
  ;
  ;
  ;
  ; setting the stack pointer
  ;-----------------------------------------------------
;SIF (__RENESAS_VERSION__ >= 0x01010000) Add ';' to the first line and comment out
;  MOVW SP,#LOWW(__STACK_ADDR_START) Add ';' to the first line and comment out
;SELSE ; for CC-RL V1.00 Add ';' to the first line and comment out
;  MOVW SP,#LOWW(_stacktop)
;SENDIF Add ';' to the first line and comment out

;-----------------------------------------------------
; initializing stack area
;-----------------------------------------------------
;SIF (__RENESAS_VERSION__ >= 0x01010000) Add ';' to the first line and comment out
;  MOVW AX,#LOWW(__STACK_ADDR_END) Add ';' to the first line and comment out
;SELSE ; for CC-RL V1.00 Add ';' to the first line and comment out
;  MOVW AX,#LOWW(_stackend)
;SENDIF Add ';' to the first line and comment out
  CALL !!_stkinit
```
4.2.2 Deploying the Rewrite Programs in the RAM Area

Deploy the programs that will be used to rewrite boot cluster 1 in the RAM area. These programs are deployed in the sections listed in Table 4-2.

Table 4-2 Section Information

<table>
<thead>
<tr>
<th>Section Name</th>
<th>Deployment-destination section name</th>
<th>Item to Be Deployed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFD_CMN_f</td>
<td>RFD_CMN_fR</td>
<td>Program section for the common flash memory control API function</td>
</tr>
<tr>
<td>RFD_CF_f</td>
<td>RFD_CF_fR</td>
<td>Program section for the code flash memory API function</td>
</tr>
<tr>
<td>RFD_EX_f</td>
<td>RFD_EX_fR</td>
<td>Program section for the extra area control API function</td>
</tr>
<tr>
<td>SMP_CMN_f</td>
<td>SMP_CMN_fR</td>
<td>Program section for the common flash memory control sample function</td>
</tr>
<tr>
<td>SMP_CF_f</td>
<td>SMP_CF_fR</td>
<td>Program section for the code flash memory control sample function</td>
</tr>
</tbody>
</table>

To deploy the preceding sections in the RAM area, you need to add processing to "cstart.asm".

In "cstart.asm", add code for the processing after the following lines:

```
;--------------------------------------------------
; ROM data copy
;--------------------------------------------------
```

The code to be added is as follows:

```
; copy .text to RAM (section-name)
MOV  C,#HIGHW(STARTOF(section-name))
MOVW HL,#LOWW(STARTOF(section-name))
MOVW DE,#LOWW(STARTOF(Placement section name))
BR $.L12_TEXT
.Lm1_TEXT:
  MOV  A,C
  MOV  ES,A
  MOV  A,ES;[HL]
  MOV  [DE],A
  INCW DE
  INCW HL
  CLRW AX
  CMPW AX,HL
  SKNZ
  INC
.Lm2_TEXT:
  MOVW AX,HL
  CMPW AX,#LOWW(STARTOF(section-name) + SIZEOF(section-name))
  BNZ $.L11_TEXT
```

- In section-name, specify the name of the section to be deployed.
- Add the preceding code for each section that needs to be deployed.
- For m, set any number of your choice. Specify a different number for each section.
On-chip Debug Security ID

The RL78/G23 has the on-chip debug security ID area allocated to addresses 000C4H to 000CDH of flash memory to preclude the memory contents from being sneaked by the unauthorized third party.

When using the boot swap function for self-programming, it is necessary to set the same values that are set in 000C4H to 000CDH also in 040C4H to 040CDH because bytes in 000C4H to 000CDH are swapped with the bytes in 040C4H to 040CDH.

Resources Used by the Sample Program

4.4.1 List of Sections in the ROM Area

Table 4-3 lists the sections that are deployed in the ROM area and used by the sample program.

<table>
<thead>
<tr>
<th>Section Name</th>
<th>Item to Be Deployed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFD_CMN_f</td>
<td>Program section for the common flash memory control API function</td>
</tr>
<tr>
<td>RFD_CF_f</td>
<td>Program section for the code flash memory control API function</td>
</tr>
<tr>
<td>RFD_EX_f</td>
<td>Program section for the extra area control API function</td>
</tr>
<tr>
<td>RFD_DF_f</td>
<td>Program section for the data flash memory control API function</td>
</tr>
<tr>
<td>SMP_CMN_f</td>
<td>Program section for the common flash memory control sample function</td>
</tr>
<tr>
<td>SMP_CF_f</td>
<td>Program section for the code flash memory control sample function</td>
</tr>
</tbody>
</table>

4.4.2 List of Sections in the RAM Area

Table 4-4 lists the sections that are deployed in the RAM area and used by the sample program.

<table>
<thead>
<tr>
<th>Section Name</th>
<th>Items to Be Deployed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFD_DATA_n</td>
<td>Data section for RFD RL78 Type01</td>
</tr>
<tr>
<td>RFD_CMN_fR</td>
<td>Program section for the common flash memory control API function</td>
</tr>
<tr>
<td>RFD_CF_fR</td>
<td>Program section for the code flash memory control API function</td>
</tr>
<tr>
<td>RFD_EX_fR</td>
<td>Program section for the extra area control API function</td>
</tr>
<tr>
<td>SMP_CMN_fR</td>
<td>Program section for the common flash memory control sample function</td>
</tr>
<tr>
<td>SMP_CF_fR</td>
<td>Program section for the code flash memory control sample function</td>
</tr>
</tbody>
</table>
List of Constants

Table 4-5 lists the constants for the sample program.

Table 4-5  Constants for the Sample Program

<table>
<thead>
<tr>
<th>Constant</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED_ON</td>
<td>00H</td>
<td>LED ON</td>
</tr>
<tr>
<td>LED_OFF</td>
<td>01H</td>
<td>LED OFF</td>
</tr>
<tr>
<td>START_WRITE_ADDRESS</td>
<td>00004000H</td>
<td>Write start address</td>
</tr>
<tr>
<td>END_WRITE_ADDRESS</td>
<td>00007FFFH</td>
<td>Write end address</td>
</tr>
<tr>
<td>WRITE_DATA_SIZE</td>
<td>0100H</td>
<td>Size of data to be written to the code flash memory (256 bytes)</td>
</tr>
<tr>
<td>CF_BLOCK_SIZE</td>
<td>0800H</td>
<td>Block size of the code flash memory (2,048 bytes)</td>
</tr>
<tr>
<td>BT1_START_ADDRESS</td>
<td>00004000H</td>
<td>Start address of boot cluster 1</td>
</tr>
<tr>
<td>BT1_END_ADDRESS</td>
<td>00007FFFH</td>
<td>End address of boot cluster 1</td>
</tr>
<tr>
<td>CPU_FREQUENCY</td>
<td>32</td>
<td>CPU operating frequency</td>
</tr>
<tr>
<td>COMMAND_START</td>
<td>02H</td>
<td>Command code: START</td>
</tr>
<tr>
<td>COMMAND_WRITE</td>
<td>03H</td>
<td>Command code: WRITE</td>
</tr>
<tr>
<td>COMMAND_END</td>
<td>04H</td>
<td>Command code: END</td>
</tr>
<tr>
<td>COMMAND_ERROR</td>
<td>FFH</td>
<td>Command code: ERROR</td>
</tr>
<tr>
<td>VALUE_U08_MASK1_FSQ_STATUS_ERR_ERASE</td>
<td>01H</td>
<td>Error status mask value for the execution result of the flash memory sequencer bit0: Erase command error</td>
</tr>
<tr>
<td>VALUE_U08_MASK1_FSQ_STATUS_ERR_WRITE</td>
<td>02H</td>
<td>Error status mask value for the execution result of the flash memory sequencer bit1: Write command error</td>
</tr>
<tr>
<td>VALUE_U08_MASK1_FSQ_STATUS_ERR_BLANKCHECK</td>
<td>08H</td>
<td>Error status mask value for the execution result of the flash memory sequencer bit3: Blank check command error</td>
</tr>
<tr>
<td>VALUE_U08_MASK1_FSQ_STATUS_ERR_CDFSEQUENCER</td>
<td>10H</td>
<td>Error status mask value for the execution result of the flash memory sequencer bit4: Code/data flash area sequencer error</td>
</tr>
<tr>
<td>VALUE_U08_MASK1_FSQ_STATUS_ERR_EXTRASEQUENCER</td>
<td>20H</td>
<td>Error status mask value for the execution result of the flash memory sequencer bit5: Extra area sequencer error</td>
</tr>
<tr>
<td>VALUE_U08_SHIFT_ADDR_TO_BLOCK_CF</td>
<td>11</td>
<td>Constant used for bit shifting performed to calculate the block number of Code Flash</td>
</tr>
<tr>
<td>VALUE_U08_SHIFT_ADDR_TO_BLOCK_DF</td>
<td>8</td>
<td>Constant used for bit shifting performed to calculate the block number of Data Flash</td>
</tr>
<tr>
<td>VALUE_U01_MASK0_1BIT</td>
<td>0</td>
<td>1-bit mask value</td>
</tr>
<tr>
<td>VALUE_U01_MASK1_1BIT</td>
<td>1</td>
<td>1-bit mask value</td>
</tr>
<tr>
<td>VALUE_U08_MASK0_8BIT</td>
<td>00H</td>
<td>8-bit mask value</td>
</tr>
<tr>
<td>VALUE_U08_MASK1_8BIT</td>
<td>FFH</td>
<td>8-bit mask value</td>
</tr>
</tbody>
</table>
**Enumerated type**

Table 4-6 shows the definition of the enumeration used in the sample program.

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENUM_RET_STS_OK</td>
<td>00H</td>
<td>Normal status</td>
</tr>
<tr>
<td>ENUM_RET.Err_CFDF_SEQUENCER</td>
<td>10H</td>
<td>Code/data flash area sequencer error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_EXTRA_SEQUENCER</td>
<td>11H</td>
<td>Extra area sequencer error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_ERASE</td>
<td>12H</td>
<td>Erase error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_WRITE</td>
<td>13H</td>
<td>Write error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_BLANKCHECK</td>
<td>14H</td>
<td>Blank error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_CHECK_WRITE_DATA</td>
<td>15H</td>
<td>Error in comparison between the written and read values</td>
</tr>
<tr>
<td>ENUM_RET.ERR_MODE_MISMATCHED</td>
<td>16H</td>
<td>Mode mismatch error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_PARAMETER</td>
<td>17H</td>
<td>Parameter error</td>
</tr>
<tr>
<td>ENUM_RET.ERR_CONFIGURATION</td>
<td>18H</td>
<td>Device configuration error</td>
</tr>
</tbody>
</table>

**List of Variables**

Table 4-7 shows the definition of the global variables used in the sample program.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Description</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>f_UART0_sendend</td>
<td>Flag indicating that data sending by the UART0 was completed</td>
<td>r_Send_nByte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_Config_UART0_callback_sendend</td>
</tr>
<tr>
<td></td>
<td>f_UART0_receiveend</td>
<td>Flag indicating that data reception by the UART0 was completed</td>
<td>r_Receive_nByte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_Config_UART0_callback_receiveend</td>
</tr>
</tbody>
</table>
List of Functions

Table 4-8 and Table 4-9 lists the functions that are used in this sample program.

Table 4-8  List of Functions (1/2)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_rfd_initialize</td>
<td>Initialization processing for RFD RL78 Type01</td>
</tr>
<tr>
<td>r_cmd_start</td>
<td>START command processing</td>
</tr>
<tr>
<td>r_cmd_write</td>
<td>WRITE command processing</td>
</tr>
<tr>
<td>r_cmd_end</td>
<td>END command processing</td>
</tr>
<tr>
<td>r_CF_RangeErase</td>
<td>Range erase processing for the code flash memory</td>
</tr>
<tr>
<td>r_CF_EraseBlock</td>
<td>Block erase processing for the code flash memory</td>
</tr>
<tr>
<td>r_CF_WriteVerifySequence</td>
<td>Write-and-verify processing for the code flash memory</td>
</tr>
<tr>
<td>r_CF_WriteData</td>
<td>Write processing for the code flash memory</td>
</tr>
<tr>
<td>r_CF_VerifyData</td>
<td>Verify processing for the code flash memory</td>
</tr>
<tr>
<td>r_CheckCFDFSequencerEnd</td>
<td>Sequence end processing for the code flash memory</td>
</tr>
<tr>
<td>r_CheckExtraSequencerEnd</td>
<td>Sequence end processing for the extra area</td>
</tr>
<tr>
<td>r_RequestBootSwap</td>
<td>Boot swapping execution processing</td>
</tr>
<tr>
<td>r_Config_UART0_callback_sendend</td>
<td>Callback processing at a sending completion interrupt for UART0</td>
</tr>
<tr>
<td>r_Config_UART0_callback_receiveend</td>
<td>Callback processing at a reception completion interrupt for UART0</td>
</tr>
<tr>
<td>r_RecvPacket</td>
<td>Command reception processing by UART0</td>
</tr>
<tr>
<td>r_ReceivePacketAnalyze</td>
<td>Command analysis processing by UART0</td>
</tr>
<tr>
<td>r_Receive_nByte</td>
<td>Data reception processing by UART0</td>
</tr>
<tr>
<td>r_Send_nByte</td>
<td>Data sending processing by UART0</td>
</tr>
<tr>
<td>r_SendACK</td>
<td>Normal response sending processing by UART0</td>
</tr>
<tr>
<td>r_Config_IICA0_callback_master_sendend</td>
<td>Callback processing at a sending completion interrupt for IICA0</td>
</tr>
<tr>
<td>r_Config_IICA0_callback_master_error</td>
<td>Callback processing at a sending error interrupt for IICA0</td>
</tr>
<tr>
<td>r_LCM_init</td>
<td>Processing to initialize the LCD module</td>
</tr>
<tr>
<td>r_LCM_clear</td>
<td>Processing to clear display for the LCD module</td>
</tr>
<tr>
<td>r_LCM_send_string</td>
<td>Processing to send strings to the LCD module</td>
</tr>
<tr>
<td>r_LCM_send_command</td>
<td>Command sending processing for the LCD module</td>
</tr>
<tr>
<td>r_LCM_send_data</td>
<td>Processing to send data to the LCD module</td>
</tr>
<tr>
<td>r_LCM_turn_sendend_on</td>
<td>Communication end flag setting for the LCD module</td>
</tr>
<tr>
<td>r_LCM_wait_sendend</td>
<td>Communication end wait processing for the LCD module</td>
</tr>
</tbody>
</table>
Table 4-9  List of Functions (2/2)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_RFD_Init</td>
<td>Initialization processing for RFD RL78 Type01</td>
</tr>
<tr>
<td>R_RFD_SetFlashMemoryModeNote</td>
<td>Flash memory control mode change processing</td>
</tr>
<tr>
<td>R_RFD_EraseCodeFlashReqNote</td>
<td>Code flash memory erase processing</td>
</tr>
<tr>
<td>R_RFD_WriteCodeFlashReqNote</td>
<td>Code flash memory write processing</td>
</tr>
<tr>
<td>R_RFD_CheckCFDFSeqEndStep1Note</td>
<td>Processing to check whether the code/data flash area sequencer has terminated</td>
</tr>
<tr>
<td>R_RFD_CheckCFDFSeqEndStep2Note</td>
<td>Processing to check whether the command was terminated by clearing the flash memory sequencer control register</td>
</tr>
<tr>
<td>R_RFD_GetSeqErrorStatusNote</td>
<td>Processing to obtain error information generated by the code/data flash area sequencer command or extra area sequencer command</td>
</tr>
<tr>
<td>R_RFD_ClearSeqRegisterNote</td>
<td>Processing to clear the register that controls the code/data flash area sequencer or extra area sequencer</td>
</tr>
<tr>
<td>R_RFD_CheckExtraSeqEndStep1Note</td>
<td>Processing to confirm that the extra area sequencer has terminated</td>
</tr>
<tr>
<td>R_RFD_CheckExtraSeqEndStep2Note</td>
<td>Processing to check whether the command was terminated by clearing the extra area sequencer control register</td>
</tr>
<tr>
<td>R_RFD_GetSecurityAndBootFlagsNote</td>
<td>Processing to obtain the security flag and boot area switching flag</td>
</tr>
<tr>
<td>R_RFD_SetDataFlashAccessModeNote</td>
<td>Processing to set whether to permit or prohibit access to the data flash memory</td>
</tr>
<tr>
<td>R_RFD_SetExtraBootAreaReqNote</td>
<td>Boot area switching flag write processing</td>
</tr>
<tr>
<td>R_RFD_ForceResetNote</td>
<td>Internal CPU reset request</td>
</tr>
</tbody>
</table>

Note: This is an API function defined for the flash self-programming code. For details about the API function, see the "RL78 Family Renesas Flash Driver RL78 Type01 User's Manual".
### Function Specifications

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Synopsis</th>
<th>Header</th>
<th>Declaration</th>
<th>Explanation</th>
<th>Arguments</th>
<th>Return value</th>
</tr>
</thead>
</table>
| r_rfd_initialize | Initialization processing for RFD RL78 Type01 | r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h | R_RFD_FAR_FUNC e_ret_t r_rfd_initialize(void); | This function initializes RFD RL78 Type01. | None | ENUM RET_STS_OK: Normal status  
ENUM RET_ERR_CONFIGURATION: Device configuration error  
ENUM RET_ERR_PARAMETER: Parameter error |
| r_cmd_start | START command processing | r_rfd_common_api.h, r_cg_userdefine.h | R_RFD_FAR_FUNC e_ret_t r_cmd_start(void); | This function performs processing required when the START command is received. | None | ENUM RET_STS_OK: Normal status  
ENUM RET_ERR_MODE_MISMATCHED: Mode mismatch error  
ENUM RET_ERR_ERASE: Erase error |
| r_cmd_write | WRITE command processing | r_rfd_common_api.h, r_cg_userdefine.h | R_RFD_FAR_FUNC e_ret_t r_cmd_write(uint32_t * write_start_addr, uint8_t __near * write_data); | This function performs processing required when the WRITE command is received. | uint32_t i_u32_start_addr: Write start address  
uint8_t __near * inp_u08_write_data: Write data | ENUM RET_STS_OK: Normal status  
ENUM RET_ERR_MODE_MISMATCHED: Mode mismatch error  
ENUM RET_ERR_ERASE: Erase error |
| r_cmd_end | END command processing | r_rfd_common_api.h, r_cg_userdefine.h | R_RFD_FAR_FUNC e_ret_t r_cmd_end(void); | This function performs processing required when the END command is received. If this function terminates normally, an internal reset occurs and the CPU is restarted. | None | ENUM RET_ERR_MODE_MISMATCHED: Mode mismatch error |
**r_CF_RangeErase**

**Synopsis**
Range erase processing for the code flash memory

**Header**
r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h

**Declaration**
R_RFD_FAR_FUNC e_ret_t r_CF_RangeErase(uint32_t start_addr, uint32_t end_addr);

**Explanation**
This function erases data in the code flash memory. Data is erased in blocks. The blocks in the range of addresses specified for arguments will be erased.

**Arguments**
- uint32_t start_addr: Erase start address
- uint32_t end_addr: Erase end address

**Return value**
- ENUM_RET_STS_OK: Normal status
- ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error
- ENUM_RET_ERR_ERASE: Erase error

---

**r_CF_EraseBlock**

**Synopsis**
Block erase processing for the code flash memory

**Header**
r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h

**Declaration**
R_RFD_FAR_FUNC e_ret_t r_CF_EraseBlock(uint32_t start_addr);

**Explanation**
This function erases data in the code flash memory. A block of data is erased. The block that includes the address specified for an argument will be erased.

**Arguments**
- uint32_t start_addr: Erase start address

**Return value**
- ENUM_RET_STS_OK: Normal status
- ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error
- ENUM_RET_ERR_ERASE: Erase error

---

**r_CF_WriteVerifySequence**

**Synopsis**
Write-and-verify processing for the code flash memory

**Header**
r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h

**Declaration**
R_RFD_FAR_FUNC e_ret_t r_CF_WriteVerifySequence(uint32_t start_addr, uint16_t write_data_length, uint8_t __near * write_data);

**Explanation**
This function writes data to the code flash memory and verifies the written data.

**Arguments**
- uint32_t start_addr: Write start address
- uint16_t write_data_length: Write size
- uint8_t __near * write_data: Write data

**Return value**
- ENUM_RET_STS_OK: Normal status
- ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error
- ENUM_RET_ERR_WRITE: Write error
- ENUM_RET_ERR_CHECK_WRITE_DATA: Error in comparison between the written and read values
### r_CF_WriteData

**Synopsis**  
Write processing for the code flash memory

**Header**  
r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h

**Declaration**  
R_RFD_FAR_FUNC e_ret_t r_CF_WriteData(uint32_t i_u32_start_addr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);

**Explanation**  
This function writes data to the code flash memory.

**Arguments**  
- uint32_t i_u32_start_addr: Write start address
- uint16_t i_u16_write_data_length: Write size
- uint8_t __near * inp_u08_write_data: Write data

**Return value**  
- ENUM_RET_STS_OK: Normal status
- ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error
- ENUM_RET_ERR_WRITE: Write error

---

### r_CF_VerifyData

**Synopsis**  
Verify processing for the code flash memory

**Header**  
r_cg_userdefine.h

**Declaration**  
R_RFD_FAR_FUNC e_ret_t r_CF_VerifyData(uint32_t start_addr, uint16_t data_length, uint8_t __near * write_data);

**Explanation**  
This function verifies the data written to the code flash memory.

**Arguments**  
- uint32_t start_addr: Verify start address
- uint16_t data_length: Data size
- uint8_t __near * write_data: Comparison data

**Return value**  
- ENUM_RET_STS_OK: Normal status (match)
- ENUM_RET_ERR_CHECK_WRITE_DATA: Error in comparison between the written and read values (Mismatch)

---

### r_CheckCFDFSequencerEnd

**Synopsis**  
Sequence end processing for the code flash memory

**Header**  
r_rfd_common_api.h, r_cg_userdefine.h

**Declaration**  
R_RFD_FAR_FUNC e_ret_t r_CheckCFDFSequencerEnd(void);

**Explanation**  
This function confirms that the code flash memory sequence has terminated.

**Arguments**  
None

**Return value**  
- ENUM_RET_STS_OK: Normal status
- ENUM_RET_ERR_CFDF_SEQUENCER: Code/data flash area sequencer error
- ENUM_RET_ERR_ERASE: Erase error
- ENUM_RET_ERR_WRITE: Write error
- ENUM_RET_ERR_BLANKCHECK: Blank error
### r_CheckExtraSequencerEnd

**Synopsis**
Sequence end processing for the extra area

**Header**
r_rfd_common_api.h, r_cg_userdefine.h

**Declaration**
R_RFD_FAR_FUNC e_ret_t r_CheckExtraSequencerEnd (void);

**Explanation**
This function confirms that the extra memory sequence has terminated.

**Arguments**
None

**Return value**
ENUM_RET_STS_OK: Normal status
ENUM_RET_ERR_EXTRA_SEQUENCER: Code/data flash area sequencer error
ENUM_RET_ERR_ERASE: Erase error
ENUM_RET_ERR_WRITE: Write error
ENUM_RET_ERR_BLANKCHECK: Blank error

### r_RequestBootSwap

**Synopsis**
Boot swapping execution processing

**Header**
r_rfd_common_api.h, r_rfd_extra_area_api.h, r_cg_userdefine.h

**Declaration**
e_ret_t r_RequestBootSwap(void);

**Explanation**
After a reset is performed, this function enables the boot swapping settings, and then generates an internal reset to restart the CPU.

**Arguments**
None

**Return value**
ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error

### r_Config_UART0_callback_sendend

**Synopsis**
Callback processing at a sending completion interrupt for UART0

**Header**
r_cg_macrodriver.h, Config_IICA0.h, LCM_driver.h

**Declaration**
static void r_Config_UART0_callback_sendend(void);

**Explanation**
This is a callback function that is called at a sending completion interrupt for UART0.

**Arguments**
None

**Return value**
None

### r_Config_UART0_callback_receiveend

**Synopsis**
Callback processing at a reception completion interrupt for UART0

**Header**
r_cg_macrodriver.h, Config_IICA0.h, LCM_driver.h

**Declaration**
static void r_Config_UART0_callback_receiveend(void);

**Explanation**
This is a callback function that is called at a reception completion interrupt for UART0.

**Arguments**
MD_STATUS flag: Error type

**Return value**
None
r_RecvPacket

Synopsis  Command reception processing by UART0
Header    r_cg_macrodriver.h, r_cg_userdefine.h
Declaration MD_STATUS r_RecvPacket(uint8_t *data, uint16_t *length);
Explanation This function uses UART0 to perform command reception processing.
This function waits until reception of a packet of data is completed.
Arguments  
uint8_t *data: Address of receive data buffer
uint16_t *length: Address of area storing receive data length
Return value  MD_OK: Normal status [reception completion]
              COMMAND_ERROR: Parameter error

r_ReceivePacketAnalyze

Synopsis  Command analysis processing by UART0
Header    r_cg_userdefine.h
Declaration uint8_t r_ReceivePacketAnalyze(uint8_t *rxbuf, uint16_t rxlength);
Explanation This function checks the checksum of received data.
If the checksum matches, the function obtains the command code in the received data.
Arguments  
uint8_t *rxbuf: Address of receive data buffer
uint16_t rxlength: Address of area storing receive data length
Return value  COMMAND_START: Receive START command
              COMMAND_WRITE: Receive WRITE command
              COMMAND_END: Receive END command
              COMMAND_ERROR: Checksum error or command code error

r_Receive_nByte

Synopsis  Data reception processing by UART0
Header    Config_UART0.h, Config_WDT.h
Declaration MD_STATUS r_Receive_nByte(uint8_t *rx_buff, const uint16_t rx_num);
Explanation This function performs reception processing by UART0.
This function waits until reception of the number of characters specified for an argument is completed.
Arguments  
uint8_t *rx_buff: Address of receive data buffer
const uint16_t rx_num: Number of characters received
Return value  MD_OK: Normal status [reception completion]
              MD_ARGERROR: Parameter error

r_Send_nByte

Synopsis  Data sending processing by UART0
Header    Config_UART0.h, Config_WDT.h
Declaration MD_STATUS r_Send_nByte(uint8_t *tx_buff, const uint16_t tx_num);
Explanation This function performs sending processing by UART0.
This function waits until sending of the number of characters specified for an argument is completed.
Arguments  
uint8_t *tx_buff: Address of send data buffer
const uint16_t tx_num: Number of characters to send
Return value  MD_OK: Normal status [reception completion]
              MD_ARGERROR: Parameter error

r_SendACK

Synopsis  Normal response sending processing by UART0
<table>
<thead>
<tr>
<th>Function</th>
<th>Synopsis</th>
<th>Header</th>
<th>Declaration</th>
<th>Explanation</th>
<th>Arguments</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_SendACK</td>
<td>This function uses UART0 to perform sending processing for normal response (01H).</td>
<td>MD_STATUS r_SendACK (void);</td>
<td>None</td>
<td>None</td>
<td>MD_OK: Normal status [sending completion] MD_ARGERROR: Parameter error</td>
<td></td>
</tr>
<tr>
<td>r_Config_IICA0_callback_master_sendend</td>
<td>Callback processing at a sending completion interrupt for IICA0</td>
<td>r_cg_macrodriver.h, Config_IICA0.h, LCM_driver.h</td>
<td>static void r_Config_IICA0_callback_master_sendend(void);</td>
<td>This is a callback function that is called at a sending completion interrupt for IICA0.</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>r_Config_IICA0_callback_master_error</td>
<td>Callback processing at a sending error interrupt for IICA0</td>
<td>r_cg_macrodriver.h, Config_IICA0.h, LCM_driver.h</td>
<td>static void r_Config_IICA0_callback_master_error(MD_STATUS flag);</td>
<td>This is a callback function that is called at a sending error interrupt for IICA0.</td>
<td>MD_STATUS flag: Error type</td>
<td></td>
</tr>
<tr>
<td>r_LCM_init</td>
<td>Processing to initialize the LCD module</td>
<td>LCM_driver.h, Config_IICA0.h</td>
<td>void r_LCM_init(void);</td>
<td>This function initializes the LCD module.</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>r_LCM_clear</td>
<td>Processing to clear display for the LCD module</td>
<td>LCM_driver.h, Config_IICA0.h</td>
<td>void r_LCM_clear(void);</td>
<td>This function sends the Clear Display command to the LCD module.</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
### r_LCM_send_string

**Synopsis**
Processing to send strings to the LCD module

**Header**
LCM_driver.h, Config_IICA0.h

**Declaration**
void r_LCM_send_string(uint8_t * const str, lcm_position_t pos);

**Explanation**
This function displays the character string passed by using the "str" argument on the LCD module.
A line can also be displayed by using the "pos" argument.

**Arguments**
- uint8_t * const str: Character string to be displayed
- lcm_position_t pos: Displayed at the top with LCM_POSITION_TOP
  Displayed at the bottom with LCM_POSITION_BOTTOM.

**Return value**
None

### r_LCM_send_command

**Synopsis**
Command sending processing for the LCD module

**Header**
LCM_driver.h, Config_IICA0.h

**Declaration**
void r_LCM_send_command(uint8_t command);

**Explanation**
This function sends the command passed by using the "command" argument to the LCD module.

**Arguments**
- uint8_t command: Command to send to LCD module

**Return value**
None

### r_LCM_send_data

**Synopsis**
Processing to send data to the LCD module

**Header**
LCM_driver.h, Config_IICA0.h

**Declaration**
void r_LCM_send_data(uint8_t data);

**Explanation**
This function sends the data passed by using the "data" argument to the LCD module.

**Arguments**
- uint8_t data: Data to be sent to the LCD module

**Return value**
None

### r_LCM_turn_sendend_on

**Synopsis**
Communication end flag setting for the LCD module

**Header**
LCM_driver.h, Config_IICA0.h

**Declaration**
void r_LCM_turn_sendend_on(void);

**Explanation**
This function sets (for g_LCM_is_sendend) the flag that indicates the end of IIC communication with the LCD module.

**Arguments**
None

**Return value**
None

### r_LCM_wait_sendend

**Synopsis**
Communication end wait processing for the LCD module

**Header**
LCM_driver.h, Config_IICA0.h

**Declaration**
static void r_LCM_wait_sendend(void);

**Explanation**
This function waits until IIC communication with the LCD module ends, and then waits for the command execution wait time (5 ms).

**Arguments**
None

**Return value**
None
Flowcharts

4.10.1 Main Processing

Figure 4-1 to Figure 4-2 shows the flowchart for main processing.

Figure 4-1  Main Processing (1/2)
Figure 4-2  Main Processing (2/2)

Was processing ended normally?

YES

Was a packet received?

YES

Analyze packet data:
r_ReceivePacketAnalyze
(recv_buff, recv_length)

Which command was received?

COMMAND_START

Check the operation mode

Normal mode

START command processing:
r_cmd_start()

COMMAND_WRITE

Operation mode

Normal mode

Operation mode

Write mode

WRITE command processing:
r_cmd_write(&write_start_address, &recv_buff[4])

COMMAND_END

Operation mode

Normal mode

Operation mode

Write mode

END command processing:
r_cmd_end()

LED1 (P53): Turned off

Error display on LCD:
r_LCM_clear
r_LCM_send_string

: While(1) loop
4.10.2 Initialization Processing for RFD RL78 Type01
Figure 4-3 shows the flowchart for initialization processing for RFD RL78 Type01.

Figure 4-3  Initialization Processing for RFD RL78 Type01

```
<table>
<thead>
<tr>
<th>r_rfd_initialize()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is HOCO active?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Initialize RL78 RFD Type01:</td>
</tr>
<tr>
<td>R_RFD_Init(CPU_FREQUENCY)</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Was initialization ended normally?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Initialization of RFD failed</td>
</tr>
<tr>
<td>Set the error status for &quot;ret_value&quot;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>return</td>
</tr>
</tbody>
</table>
```
4.10.3 START Command Processing

Figure 4-4 shows the flowchart for START command processing.

Figure 4-4  START Command Processing

```
r_cmd_start()

LED1 (P53): Turned on

Prohibit interrupts:
DI()

Range erase processing for the code flash memory:
  r_CF_RangeErase(BT1_START_ADDRESS,
                 BT1_END_ADDRESS)

Permit interrupts:
EI()

Was erase processing ended normally?

YES
  Normal response sending processing:
  r_SendACK

NO

IE = 0

IE = 1

return
```
4.10.4 WRITE Command Processing

Figure 4-5 shows the flowchart for WRITE command processing.

Figure 4-5  WRITE Command Processing

```
<table>
<thead>
<tr>
<th>r_cmd_write(write_start_addr, *write_data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write-destination address is...</td>
</tr>
<tr>
<td>Inside boot cluster 1</td>
</tr>
<tr>
<td>Prohibit interrupts:</td>
</tr>
<tr>
<td>D()</td>
</tr>
<tr>
<td>Write-and-verify processing for the code flash memory:</td>
</tr>
<tr>
<td>r_CF_WriteVerifySequence(*write_Start_addr,WRITE_DATA_SIZE,write_data)</td>
</tr>
<tr>
<td>Permit interrupts:</td>
</tr>
<tr>
<td>EI()</td>
</tr>
<tr>
<td>Was write-and-verify processing ended normally?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Normal response sending processing:</td>
</tr>
<tr>
<td>r_SendACK</td>
</tr>
<tr>
<td>return</td>
</tr>
</tbody>
</table>

IE = 0

IE = 1

Outside boot cluster 1

Write-destination address is...

Inside boot cluster 1

Permit interrupts:

EI()

IE = 0

IE = 1

Was write-and-verify processing ended normally?

YES

Normal response sending processing:

r_SendACK

return
```
4.10.5 END Command Processing

Figure 4-6 shows the flowchart for END command processing.

Figure 4-6  END Command Processing

- `r_cmd_end()`
- LED1 (P53): Turned off
- Normal response sending processing: `r_SendACK`
- Boot swapping execution processing: `r_RequestBootswap()`
- `return`
4.10.6 Range Erase Processing for the Code Flash Memory

Figure 4-7 shows the flowchart for range erase processing for the code flash memory.

**Figure 4-7  Range Erase Processing for the Code Flash Memory**

```

is erasure of the blocks in the specified range incomplete?

YES

Block erasure for the code flash memory:
r_CF_EraseBlock(erase_addr)

Was block erasure ended normally?

NO

YES

return
```

```
4.10.7 Block Erase Processing for the Code Flash Memory

Figure 4-8 shows the flowchart for block erase processing for the code flash memory.

Figure 4-8  Block Erase Processing for the Code Flash Memory

```
if (Was the mode change ended normally?)
  if (Was the status set for "ret_value" normal?)
    Set the error status for "ret_value"
  else
    Set the error status for "ret_value"
else
  Processing to end the code flash memory sequencer:
  r_CheckCFDFSequencerEnd
  if (Was block erasure ended normally?)
    Set the error status for "ret_value"
  else
    Set the error status for "ret_value"
```

```cpp
r_CF_EraseBlock(start_addr)
```

Obtain the erase-target block numbers

Place the flash memory sequencer in flash programming mode:

```cpp
R_RFD_SetFlashMemoryMode(R_RFD_ENUM_FLASH_MODE_CODE_PROGRAMMING)
```

```cpp
Was the mode change ended normally?
```

```cpp
YES
```

```cpp
Is the status set for "ret_value" normal?
```

```cpp
NO
```

```cpp
Set the error status for "ret_value"
```

```cpp
YES
```

```cpp
Start erasing blocks:
R_RFD_EraseCodeFlashReq(block_number)
```

```cpp
Processing to end the code flash memory sequencer:
r_CheckCFDFSequencerEnd
```

```cpp
Was block erasure ended normally?
```

```cpp
NO
```

```cpp
Set the error status for "ret_value"
```

```cpp
YES
```

```cpp
Place the flash memory sequencer in normal operation mode:
R_RFD_SetFlashMemoryMode(R_RFD_ENUM_FLASH_MODE_UNPROGRAMMABLE)
```

```cpp
Was the mode change ended normally?
```

```cpp
NO
```

```cpp
Set the error status for "ret_value"
```

```cpp
YES
```

```cpp
return
```
4.10.8 Write-and-verify Processing for the Code Flash Memory

Figure 4-9 shows the flowchart for write-and-verify processing for the code flash memory.

![Flowchart for write-and-verify processing for the code flash memory.](Image)

- **Code flash memory write processing:**
  - `r_CF_WriteData(write_start_addr, write_data_length, write_data)`

- **Code flash memory verification processing:**
  - `r_CF_VerifyData(write_start_addr, write_data_length, write_data)`

**Flowchart Description:**

1. **r_CF_WriteVerifySequens**:
   - `write_start_addr`, `write_data_length`, `write_data`

2. Code flash memory write processing:
   - `r_CF_WriteData(write_start_addr, write_data_length, write_data)`

3. **Was the write ended normally?**
   - **YES**
     - Code flash memory verification processing:
       - `r_CF_VerifyData(write_start_addr, write_data_length, write_data)`

4. **return**

**ret_value:** Error status
4.10.9 Write Processing for the Code Flash Memory

Figure 4-10 shows the flowchart for write processing for the code flash memory.

Figure 4-10  Write processing for the Code Flash Memory

```
r_CF_WriteData(start_addr, write_data_length, write_data)

Place the flash memory sequencer in flash programming mode:
R_RFD_SetFlashMemoryMode(R_RFD_ENUM_FLASH_MODE_CODE_PROGRAMMING)

Was the mode change ended normally?

YES

Set the error status for "ret_value"

Is the status set for "ret_value" normal?

YES

ret_value: Error status

NO

Is there data that has not been written yet?

YES

Start writing 4-byte data:
R_RFD_WriteCodeFlashReq(start_addr + count, &write_data[count])

Processing to end the code flash memory sequencer:
    r_CheckCFDFSequencerEnd()

Was the data write ended normally?

YES

Set the error status for "ret_value"

NO

Place the flash memory sequencer in normal operation mode:
R_RFD_SetFlashMemoryMode(R_RFD_ENUM_FLASH_MODE_UNPROGRAMMABLE)

Was the mode change ended normally?

YES

Set the error status for "ret_value"

NO

return
```
4.10.10 Verify Processing for the Code Flash Memory

Figure 4-11 shows the flowchart for verify processing for the code flash memory.

Figure 4-11  Verify Processing for the Code Flash Memory
4.10.11 Sequence End Processing for the Code Flash Memory

Figure 4-12 to Figure 4-13 shows the flowchart for sequence end processing for the code flash memory.

Figure 4-12  Sequence End Processing for the Code Flash Memory (1/2)

```
<table>
<thead>
<tr>
<th>r_CheckCFDFSequencerEnd()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Wait for completion of processing:</td>
</tr>
<tr>
<td>R_RFD_CheckCFDFSeqEndStep1()</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Wait for completion of processing:</td>
</tr>
<tr>
<td>R_RFD_CheckCFDFSeqEndStep2()</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Obtain the error status:</td>
</tr>
<tr>
<td>R_RFD_GetSeqErrorStatus(&amp;status_flag)</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```
What is the value of the error status (FSASTL)?

VALUE_U08_MASK1_FSQ_STATUS_ERR_CFDF_SEQUENCER
Set ENUM_RET_ERR_CFDF_SEQUENCER (0x10) for "ret_value"

VALUE_U08_MASK1_FSQ_STATUS_ERR_ERASE
Set ENUM_RET_ERR_ERASE (0x12) for "ret_value"

VALUE_U08_MASK1_FSQ_STATUS_ERR_WRITE
Set ENUM_RET_ERR_WRITE (0x13) for "ret_value"

VALUE_U08_MASK1_FSQ_STATUS_ERR_BLANKCHECK
Set ENUM_RET_ERR_BLANKCHECK (0x14) for "ret_value"

Set ENUM_RET_STS_OK (0x00) for "ret_value"

Initialize the register that controls the sequencer:
R_RFD_ClearSeqRegister()

return
4.10.12 Sequence End Processing for the Extra Area
Figure 4-14 to Figure 4-15 shows the flowchart for sequence end processing for the extra area.

Figure 4-14 Sequence End Processing for the Extra Area (1/2)
What is the value of the error status (FSASTL)?

- Set ENUM_RET_ERR_EXTRASEQUENCER (0x10) for "ret_value"
- Set ENUM_RET_ERR_ERASE (0x12) for "ret_value"
- Set ENUM_RET_ERR_WRITE (0x13) for "ret_value"
- Set ENUM_RET_ERR_BLANKCHECK (0x14) for "ret_value"
- Set ENUM_RET_STS_OK (0x00) for "ret_value"

Initialize the register that controls the sequencer: R_RFD_ClearSeqRegister()
### 4.10.13 Boot Swapping Execution Processing

Figure 4-16 to Figure 4-17 shows the flowchart for boot swapping execution processing.

**Figure 4-16 Boot Swapping Execution Processing (1/2)**

1. **r_RequestBootSwap()**
2. Obtain the information about the security flag and boot flag: 
   ```c
   R_RFD_GetSecurityAndBootFlags(&f_security_boot)
   ```
3. What is the boot area that is currently set?
   - **BTFLG = 0**: Boot cluster 0 is set.
   - **BTFLG = 1**: Boot cluster 1 is set.
4. Select boot cluster 1 at the next startup:
   ```c
   next_boot_cluster = R_RFD_ENUM_BOOT_CLUSTER_1
   ```
5. Select boot cluster 0 at the next startup:
   ```c
   next_boot_cluster = R_RFD_ENUM_BOOT_CLUSTER_0
   ```
6. Set the access permission status of the data flash memory:
   ```c
   R_RFD_SetDataFlashAccessMode(R_RFD_ENUM_DF_ACCESS_ENABLE)
   ```
7. Place the flash memory sequencer in flash programming mode:
   ```c
   R_RFD_SetFlashMemoryMode(R_RFD_ENUM_FLASH_MODE_CODE_PROGRAMMING)
   ```
8. Was the mode change ended normally?
   - **YES**
   - **NO**
9. Set the error status for "ret_value"
Request boot swapping after a reset:
R_RFD_SetExtraBootAreaReq(next_boot_cluster)

Sequence end processing for the extra area:
r_CheckExtraSequencerEnd()

Place the flash memory sequencer in normal operation mode:
R_RFD_SetFlashMemoryMode(R_RFD_ENUM_FLASH_MODE_UNPROGRAMMABLE)

Was the mode change ended normally?

Set the access prohibition status for the data flash memory:
R_RFD_SetDataFlashAccessMode(R_RFD_ENUM_DF_ACCESS_MODE_DISABLE)

Is the status set for "ret_value" normal?

Internal reset:
R_RFD_ForceReset

return

ret_value : Error status
4.10.14 Callback Processing at a Reception Completion Interrupt for UART0

Figure 4-18 shows the flowchart for callback processing at a reception completion interrupt for UART0.

**Figure 4-18** Callback Processing at a Reception Completion Interrupt for UART0

```
set 1 for the reception completion flag
```

```
return
```

Set 1 for the reception completion flag: `f_UART0_receiveend = 1`
4.10.15 Callback Processing at a Sending Completion Interrupt for UART0

Figure 4-19 shows the flowchart for callback processing at a sending completion interrupt for UART0.

Figure 4-19 Callback Processing at a Sending Completion Interrupt for UART0

```
r_Config_UART0_callback_sendend()

Set 1 for the sending completion flag
f_UART0_sendend = 1

return
```
4.10.16 Command Reception Processing by UART0

Figure 4-20 shows the flowchart for command reception processing by UART0.

Figure 4-20  Command Reception Processing by UART0

```
r_RecvPacket(*data, *length)

Receive the start code:
r_Receive_nByte(data, 1)

Was reception ended normally or was the received data 0x01?
YES

Was reception ended normally?
YES

Receive the length data:
r_Receive_nByte(data + 1, 2)

Was reception ended normally?
YES

Assign the COMMAND_ERROR value to "ret"

NO

Assign the COMMAND_ERROR value to "ret"

YES

Was reception ended normally?

NO

Assign the COMMAND_ERROR value to "ret"

YES

Was reception ended normally?

NO

Assign the COMMAND_ERROR value to "ret"

YES

Receive "command + data + checksum":
r_Receive_nByte(data + 3, *length)

Was reception ended normally?

NO

YES

Assign the COMMAND_ERROR value to "ret"

return
```
4.10.17 Command Analysis Processing by UART0

Figure 4-21 shows the flowchart for command analysis processing by UART0.

Figure 4-21 Command Analysis Processing by UART0

```
4.10.17 Command Analysis Processing by UART0
Figure 4-21 shows the flowchart for command analysis processing by UART0.

4.10.17 Command Analysis Processing by UART0

Figure 4-21 Command Analysis Processing by UART0

r_ReceivePacketAnalyze(rxbuf, rxlen)

Calculate the checksum

Are the checksum calculation result and the received data identical?

YES

Checksum error
Assign the error status to "ret"

COMMAND_START

Assign the command code to "ret"

COMMAND_WRITE

Assign the command code to "ret"

COMMAND_END

Assign the command code to "ret"

Receive a code other than START, WRITE, and END
Assign the error status to "ret"

return
```
4.10.18 Data Reception Processing by UART0

Figure 4-22 shows the flowchart for data reception processing by UART0.

**Figure 4-22 Data Reception Processing by UART0**

```
<table>
<thead>
<tr>
<th>r_Receive_nByte(*rx_buff, rx_num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear the reception completion flag</td>
</tr>
<tr>
<td>f_UART0_receiveend = 0</td>
</tr>
<tr>
<td>Receive data:</td>
</tr>
<tr>
<td>R_Config_UART0_Receive(rx_buff, rx_num)</td>
</tr>
<tr>
<td>Was sending ended normally with the sending completion flag set to 0?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>return</td>
</tr>
</tbody>
</table>
```
4.10.19 Data Sending Processing by UART0

Figure 4-23 shows the flowchart for data sending processing by UART0.

Figure 4-23 Data Sending Processing by UART0

- `r_Send_nByte(*tx_buff, tx_num)`
- Clear the sending completion flag
- `f_UART0_sendend = 0`
- Send data:
  - `R_Config_UART0_Send(tx_buff, tx_num)`
- Was sending ended normally with the sending completion flag set to 0?
  - YES: return
  - NO: Was sending ended normally with the sending completion flag set to 0?
- return
4.10.20 Normal Response Sending Processing by UART0

Figure 4-23 shows the flowchart for normal response sending processing by UART0.

Figure 4-24 Normal Response Sending Processing by UART0

- `r_SendACK()`
- Create a response notification: `send_buff`
- Send data: `r_Send_nByte(&send_buff, 1)`
- Return
4.10.21 Callback Processing at a Sending Completion Interrupt for IICA0

Figure 4-25 shows the flowchart for callback processing at a sending completion interrupt for IICA0.

Figure 4-25 Callback Processing at a Sending Completion Interrupt for IICA0

```
<table>
<thead>
<tr>
<th>r_Config_IICA0_callback_master_sendend()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create stop conditions</td>
</tr>
<tr>
<td>SPT0 ← 1</td>
</tr>
<tr>
<td>r_LCM_turn_sendend_on()</td>
</tr>
<tr>
<td>return</td>
</tr>
</tbody>
</table>
```
4.10.22 Callback Processing at a Sending Error Interrupt for IICA0

Figure 4-26 shows the flowchart for callback processing at a sending error interrupt for IICA0.

![Flowchart](image)
4.10.23 Processing to Initialize the LCD Module

Figure 4-27 shows the flowchart for processing to initialize the LCD module.

Figure 4-27 Processing to Initialize the LCD Module

- `r_LCM_init()`
  - Set 0 for `g_LCM_is_sendend`

- `r_LCM_send_command(_0x20_LCM_COMMAND_FUNCTION_SET | LCM_CONFIG_FUNCTION_SET_PARAMS)`
  - Set the Function Set parameters

- `r_LCM_send_command(_0x08_LCM_COMMAND_DISPLAY_ONOFF | LCM_CONFIG_DISPLAY_ONOFF_PARAMS)`
  - Set the Display ON/OFF Control parameters

- `r_LCM_send_command(_0x04_LCM_COMMAND_ENTRY_MODE_SET | LCM_CONFIG_ENTRY_MODE_SET_PARAMS)`
  - Set the Entry Mode Set parameters
4.10.24  Processing to Clear Display for the LCD Module

Figure 4-28 shows the flowchart for processing to clear display for the LCD module.

Figure 4-28  Processing to Clear Display for the LCD Module

```
r_LCM_clear()

r_LCM_send_command(_0x00_LCM_COMMAND_CLEAR_DISPLAY)

return
```

Send the Clear Display command
4.10.25 Processing to Send Strings to the LCD Module

Figure 4-29 shows the flowchart for processing to send strings to the LCD module.

r_LCM_send_string(str, pos)

R_LCM_send_command(0x80_LCM_COMMAND_SET_DDRAM_ADDRESS | pos)

Set 0 for control variable "i"

Is "i" smaller than the number of characters specified for "str"?

YES

Is "i" smaller than the number of characters that can be displayed?

YES

r_LCM_send_data(i-th character of the string in "str")

Increment the "i" value

Send the "Set DDRAM Address" command and set the position specified for "pos" as the drawing start point.

NO

NO

return
4.10.26 Command Sending Processing for the LCD Module

Figure 4-30 shows the flowchart for command sending processing for the LCD module.

Figure 4-30 Command Sending Processing for the LCD Module

- `r_LCM_send_command(command)`
- Prepare a temporary buffer, and then sequentially deploy "0x00_LCM_CONTROL_BYTE_RS_LOW" and a command
- Send the command to LCM: `R_Config_IICA0_Master_Send(LCM_slave-address, temporary-buffer, temporary-buffer-size, LCM_CONFIG_WAIT_COUNT)`
- `r_LCM_wait_sendend()`
- `return`

In the LCD module specifications, 0x00 indicates that a command follows.
4.10.27 Processing to Send Data to the LCD Module

Figure 4-31 shows the flowchart for processing to send data to the LCD module.

**Figure 4-31 ** Processing to Send Data to the LCD Module

```
r_LCM_send_data(data)

Prepare a temporary buffer, and then sequentially deploy "_0x00_LCM_CONTROL_BYTE_RS_HIGH"
and data

Send the data to LCM:
R_Config_IICA0_Master_Send(
  LCM-slave-address, temporary-buffer, temporary-buffer-size, LCM_CONFIG_WAIT_COUNT)

r_LCM_wait_sendend()

return
```

In the LCD module specifications, 0x80 indicates that data follows.
4.10.28 Communication End Flag Setting for the LCD Module

Figure 4-32 shows the flowchart for communication end flag setting for the LCD module.

Figure 4-32 Communication End Flag Setting for the LCD Module

```
r_LCM_turn_sendend_on()
```

Turn on g_LCM_is_sendend

```
return
```
4.10.29 Communication End Wait Processing for the LCD Module

Figure 4-33 shows the flowchart for communication end wait processing for the LCD module.

Figure 4-33 Communication End Wait Processing for the LCD Module

```
<table>
<thead>
<tr>
<th>r_LCM_wait_sendend(str, pos)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Is &quot;g_LCM_is_sendend&quot; on?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Clear &quot;g_LCM_is_sendend&quot;</td>
</tr>
<tr>
<td>Wait while repeating NOP()</td>
</tr>
<tr>
<td>return</td>
</tr>
</tbody>
</table>
```

According to the LCD module specifications, a wait of at least 5 ms is required after the sending.
5. Sample code

Sample code can be downloaded from the Renesas Electronics website.

6. Reference Documents

RL78/G23 User’s Manual: Hardware (R01UH0896J)
RL78 family user's manual software (R01US0015J)
The latest versions can be downloaded from the Renesas Electronics website.

Technical update
The latest versions can be downloaded from the Renesas Electronics website.

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Jun. 18. 21</td>
<td>—</td>
<td></td>
<td>First Edition</td>
</tr>
</tbody>
</table>

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   *Standard*: Computers; office equipment; 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 implantsations; etc.), or may cause serious property damage (space system; underwater 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.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

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

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

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

11. Renesas Electronics products and technologies shall not be used 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.

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

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

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

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

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