Renesas USB MCU

USB Peripheral Firmware Updater

Overview

This application note explains Flahs ROM rewrite program (hereinafter referred to as "Updater"), which uses USB peripheral controllers.

Target Devices

RL78/G1C, RL78/L1C

When implementing this application note in the user system, conduct an extensive evaluation to ensure compatibility.

Table of Contents

1.   Document Outline

This application note explains the Updater used for USB peripheral controllers. Please use in combination with the documents listed in Section **1.2 Related Documents**.

1.1   Functions

This updater updates the user program using the Communication Device Class of the Universal Serial Bus Specification (referred to as USB herein).

1.2   Related Documents

1.   Universal Serial Bus Revision 2.0 specification

2.   RL78 Family Flash Self Programming Library Type01 application note

3.   User's Hardware Manual corresponding to each MCU

The latest versions of all documents are available for download from the Renesas Electronics website.

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

USB device page
>   http://www.renesas.com/prod/usb/

1.3   Cautions

a.   The operations described in this application note are not guaranteed. When using this application note for your system, conduct an extensive evaluation to ensure compatibility.

b.   When implementing this program into your system, please refer to the contents of section **6   USB Peripheral Firmware Updater and User Program Settings** and **7.3 Cautions** section**.**

1.4   Note

Please do the following works since the Flash Self Library (FSL) is not included in this project.

1.   Get the Flash Self Library for RL78.

2.   Copy the Flash Self Library in the "FSL" folder.

>   FSL¥incrl78

>   FSL¥librl78

Set the include path and the library path for the Flash Self Library to the compiler and linker.

1.5   List of Abbreviations and Acronyms

The following lists terms and abbreviations used in this document.

API	:	Application Program Interface
BSP	:	Renesas Board support package module
CDC	:	Communication Device Class
e^2 studio	:	Eclipse embedded studio
H/W	:	Renesas USB device
MCU	:	Micro control Unit
P/E	:	Program / Erase

RSK   :   Renesas Starter Kit
USB   :   Universal Serial Bus

2. USB Peripheral Firmware Updater Overview

The program transfers a specified user program from the file transfer application on the host machine (referred to as "PC" herein) to the evaluation board via a USB connection. The transferred user program is written to an address in the ROM using the Flash self-programming library

The program is configured as follows:

(1). USB Peripheral Firmware Updater

This is the program to be implemented in the evaluation board; performs serial transmission via USB and self programming.

(2). File transfer application

The application runs on the host machine (PC) and transfers specified files to the evaluation board in USB transmissions.

(3). User program

This file is written by USB Peripheral Firmware Updater for the operation confirming.

Program 1: The string is displayed on RSK LCD.

Program 2: The string is displayed on RSK LCD.

The following shows the program's data flow.
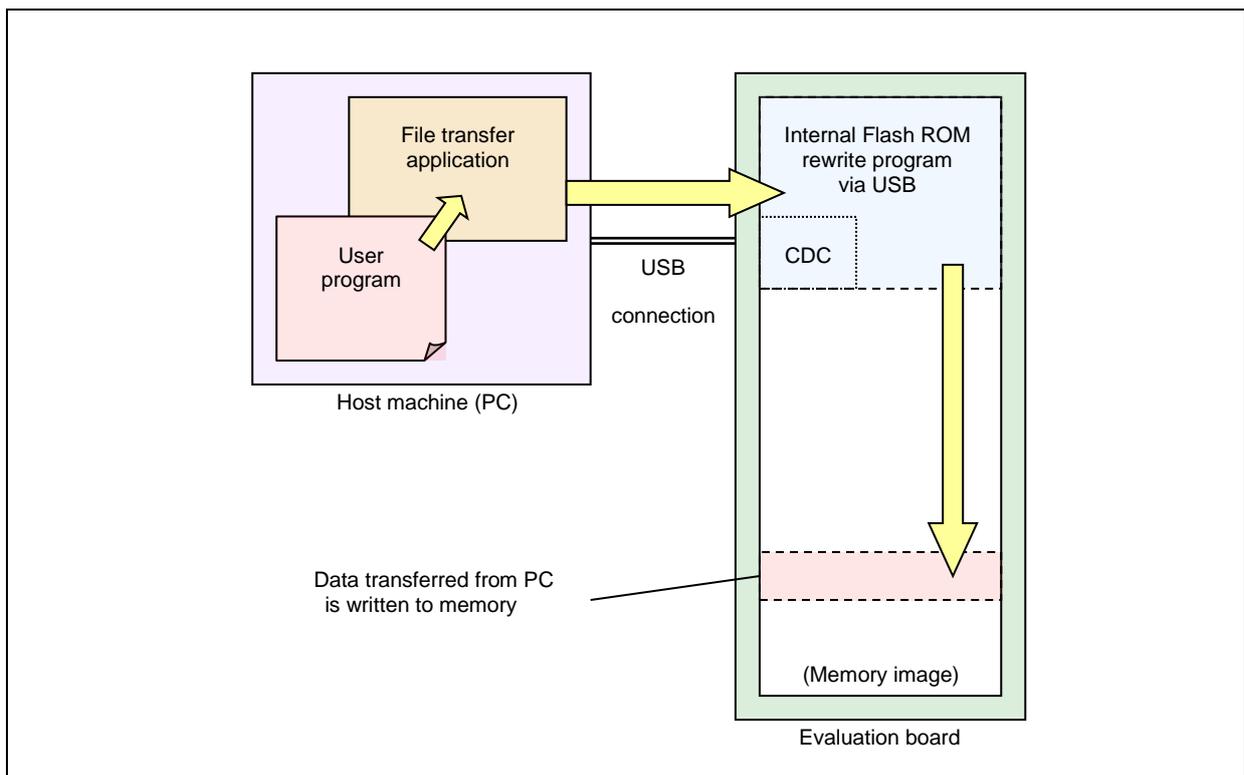
Figure 2-1 USB F/W Update Data Flow

The USB Peripheral Firmware Updater works when the evaluation board is started up in specified conditions, otherwise the user program works.

2.1   Features

This program offers the following features.

1.   The program performs full-speed data transfers between the USB host and the evaluation board using CDC.

2.   The program occupies part of the internal flash memory (See Table 2-1). If your MCU supports user boot area, the Flash ROM rewite program can be assigned to the user boot area.

3.   This program supports the Motorola S and Intel extended file formats as the user program format (.mot/.HEX files).

4.   Programs can be written to any area by specifying an address on ROM.

5.   The user program can use all interrupt functions.

2.2   ROM/RAM Size

ROM/RAM used by this program is listed in Table 2-1.

Table 2-1 Firmware Update Program ROM/RAM Size

Compiler	ROM/RAM Size (Bytes)	
	ROM	RAM
CCRL	6.2K	0.5K
CA78K0R	7.3K	0.7K

*1   The default option is specified in the optimization option .

2.3   **Operation Confirmation Environment**

Operations for this program have been confirmed under the following environment:

1.   Hardware environment

(1).	Evaluation board	Renesas Starter Kit
(2).	MCU	RL78/G1C, RL78/L1C
(3).	Emulator	E1
(4).	USB cable	USB communication between evaluation board and PC
(5).	PC	PC running on Windows® 7/ Window® 8.1/ Window® 10

2.   Software environment

(1).	Compiler	CA78K0R V.1.71 (CS+) and CC-RL V.1.01 (e^2 studio)
(2).	Flash programming tool	Renesas Flash Programmer
(3).	USB F/W Update sample/program set	

USB Peripheral Firmware Updater

File transfer application

Sample user program

2.4 Folder Configuration

The following is the folder configuration for this program.

```
(Top Directory)
 +—reference
 |      +--cdc_inf
 |      |     CDC driver sample inf file (CDC_Demo.inf)
 |      +--FirmupdaterGUI
 |      |     |   File transfer application (UsbfUpdater.exe / UsbfUpdater.ini)
 |      |     +---source
 |      |             File transfer application sources
 |      +--SampleProgram (Sample program for operation confirmation )
 +—workspace (USB Peripheral Firmware Updater Sample projects)
          +-- (CA78K0R)
          +-- (CCRL)
```

The following provides a description of each folder.

(1). **reference¥cdc_inf**

This folder stores the Windows ® CDC driver.

CDC_Demo.inf: Windows ® CDC driver (Windows® 32bit/64bit)

(2). **reference¥FirmupdaterGUI**

This folder stores the file transfer application.

UsbfUpdater.exe: File transfer application execution file

UsbfUpdater.ini: File transfer application setting file

(3). **reference¥FirmupdaterGUI¥source**

This folder stores the file transfer application source program. For more details, refer to section **8   File Transfer**

Application (USB Firmware Updater) Explanation.

(4). **reference¥SampleProgram**

This folder stores the sample user program. This sample program displays the string on RSK LCD.

(5). **workspace**

This file stores USB Peripheral Firmware Updater for each compiler. For more details, refer to section **7 USB Peripheral**

Firmware Updater Explanation.

3. USB Peripheral Firmware Updater Setup

This section explains the setup sequence for this program.

3.1 Project Setup (e² studio)

Select the folder with the name of the MCU you are using from the Workspace folder tab. Set up the project according to the following sequence. This sequence is for setting up with e² studio.

(1).   Start up e² studio.

   *If running e² studio for the first time, the Workspace Launcher dialog box will appear first. Specify the folder which will store the project.

(2).   Select [**File**] → [**Import**]; the import dialog box will appear.

(3).   In the Import dialog box, select [**Existing Projects into Workspace**].
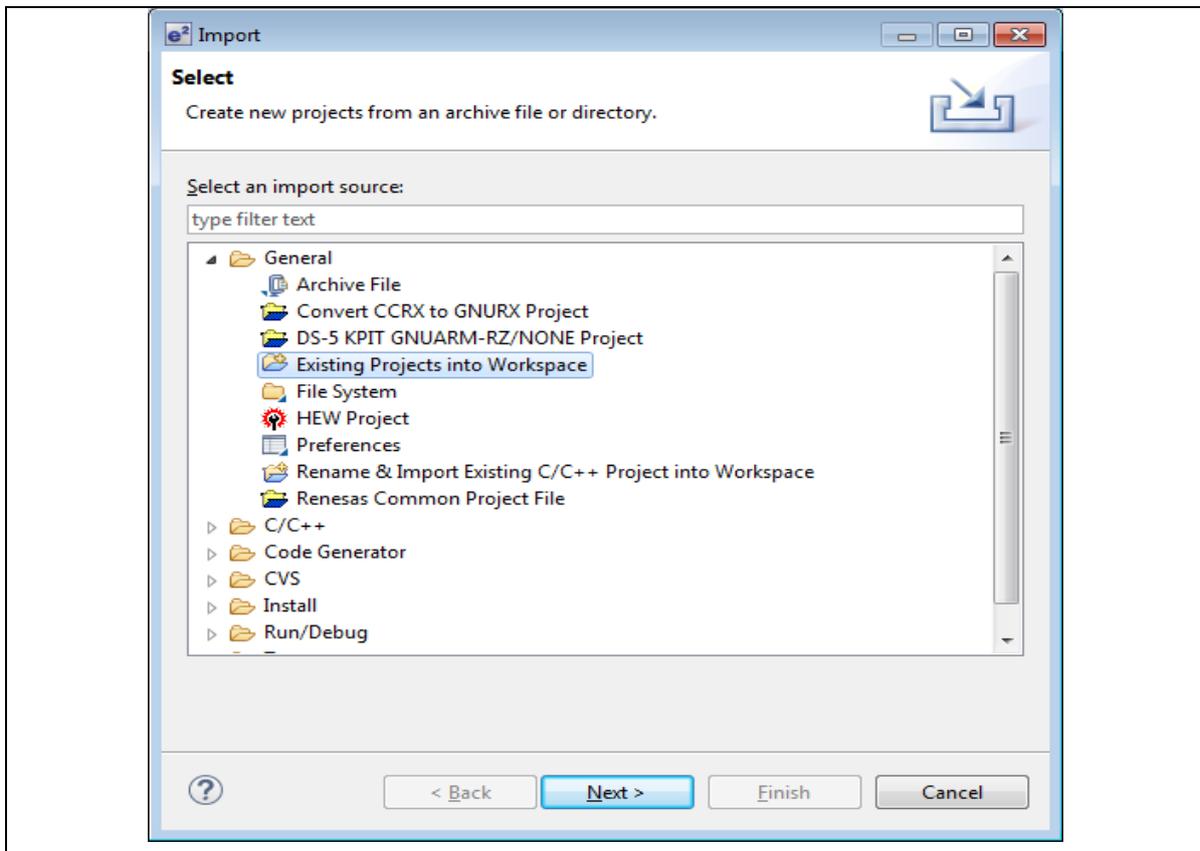
Figure 3-1   Select Import Source

(4).   Press [**Browse**] for [**Select root directory**]. Select the folder in which [.cproject] (project file) is stored.
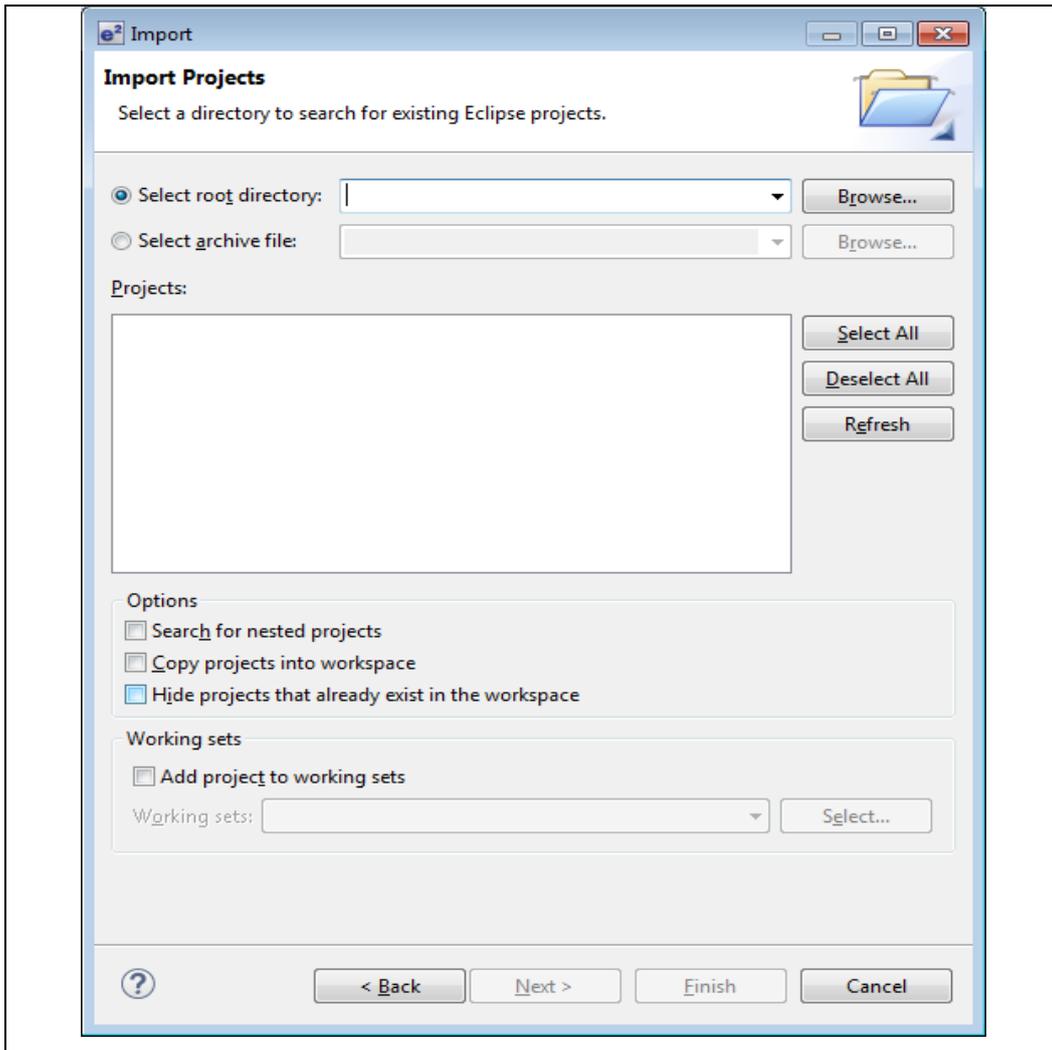
Figure 3-2   Project Import Dialog Box
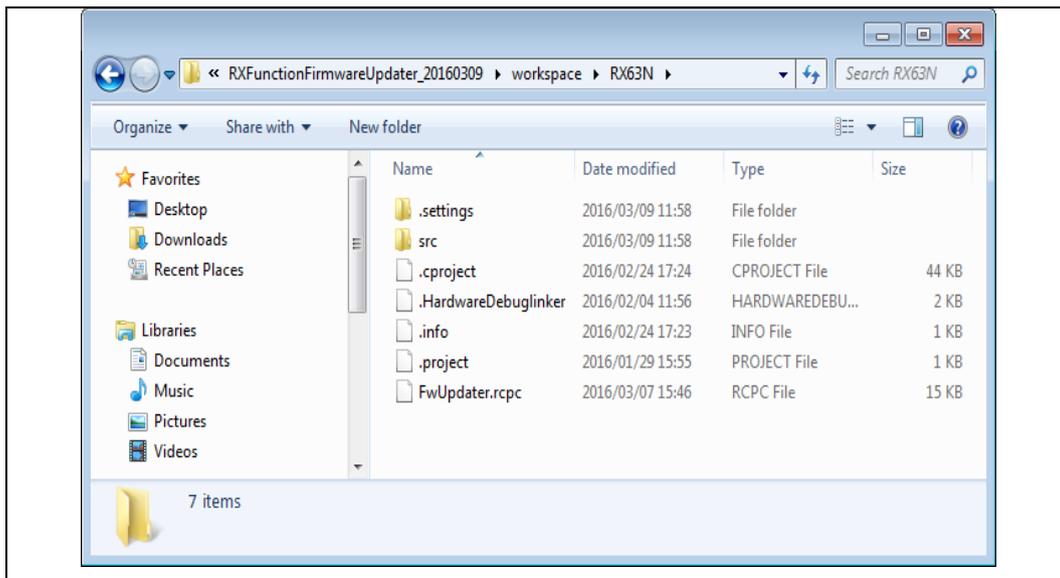
Figure 3-3   Example of Folder with .project File

(5).   Click [**Finish**].

     This completes the step for importing a project to the project workspace.

3.2   **Project Setup (CS+)**

Please double-click the *mtpj* file.

4.   Execute USB Peripheral Firmware Updater

This section describes how to execute this program.

This process uses the RSK board to confirm operations of two different user programs.

4.1   File Transfer Application (USB Function Firmware Updater) Startup

The File Transfer Application which transmits the user program starts up when the UsbfUpdater.exe file in the FirmupdateGUI folder is executed.

Figure 4-1 shows how to set the following file transfer application.

Notes:

If the file transfer application does not start up, make sure the folder that contains the exe file also contains the UsbfUpdater.ini, and then retry the process.
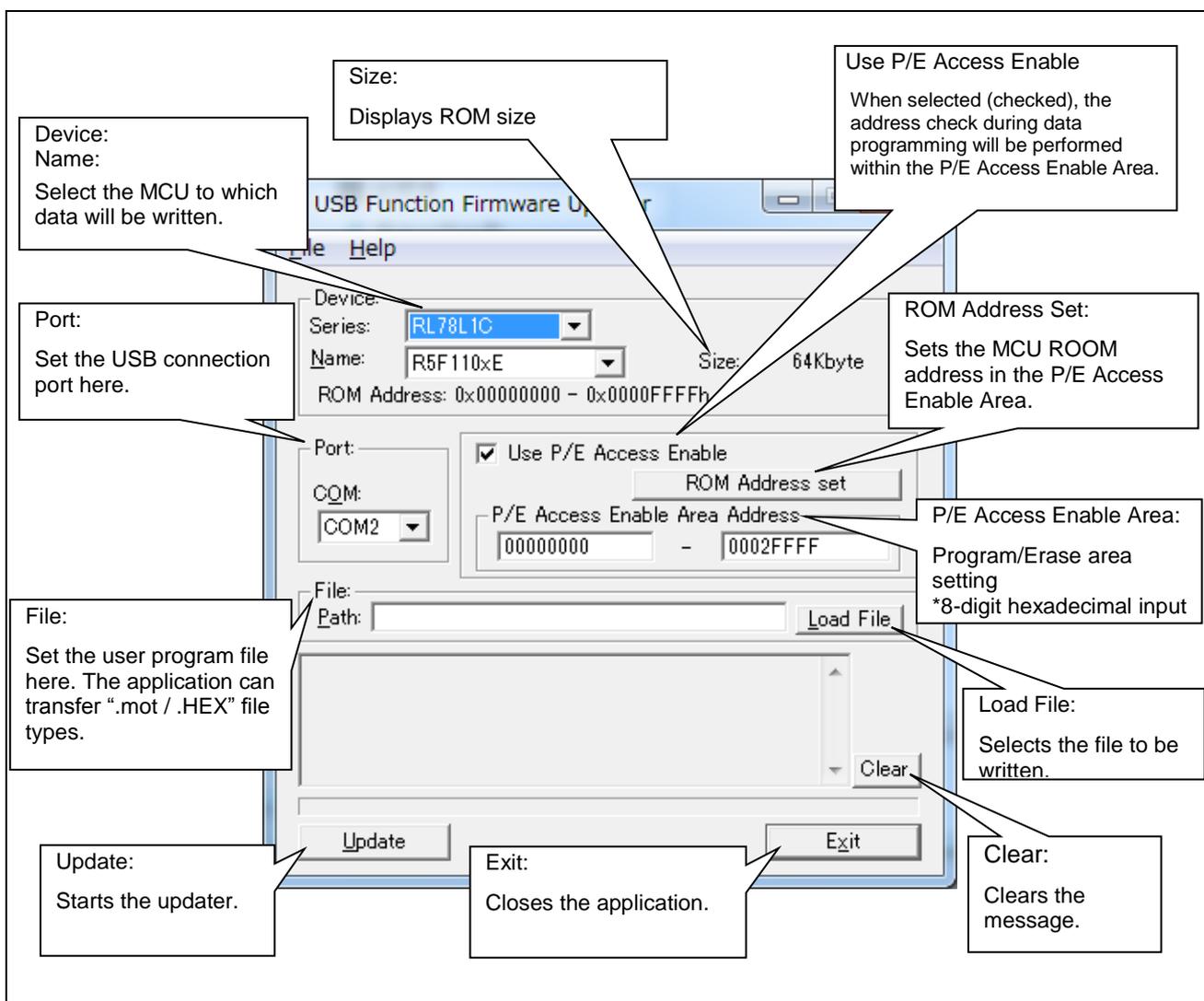
Figure 4-1   USB Firmware Updater GUI Software

4.1.1   P/E Access Enable Area Address

Set the Program/Erase enable area so that the firmware updater program area will not be written over when the user program is written to the MCU. Please use the settings listed in Table 4-1 to set the range for P/E Access Enable Area Address. (Be sure to set "0x00000000" in the start address.)

Table 4-1   P/E Access Enable Area Address Settings

MCU	P/E address Setting		
RL78/G1C	0x00002000	-	Any
RL78/L1C	0x00002000	-	Any

Notes:

1.   The block including the specified address will be erased during an erase operation. Be careful when setting the ROM block size. For more details on ROM block size, refer to the user's hardware manual corresponding to the target MCU.

4.2 Writing USB Peripheral Firmware Updater to Flash ROM write and execution

This section explains the sequence for writing and executing the Internal Flash ROM rewrite program.

4.2.1 Writing USB Peripheral Firmware Updater to ROM

(1) Hardware setup

The following figures show connection diagrams for writing USB Peripheral Firmware Updater to the MCU.

a.   Using an E1/E20 emulator
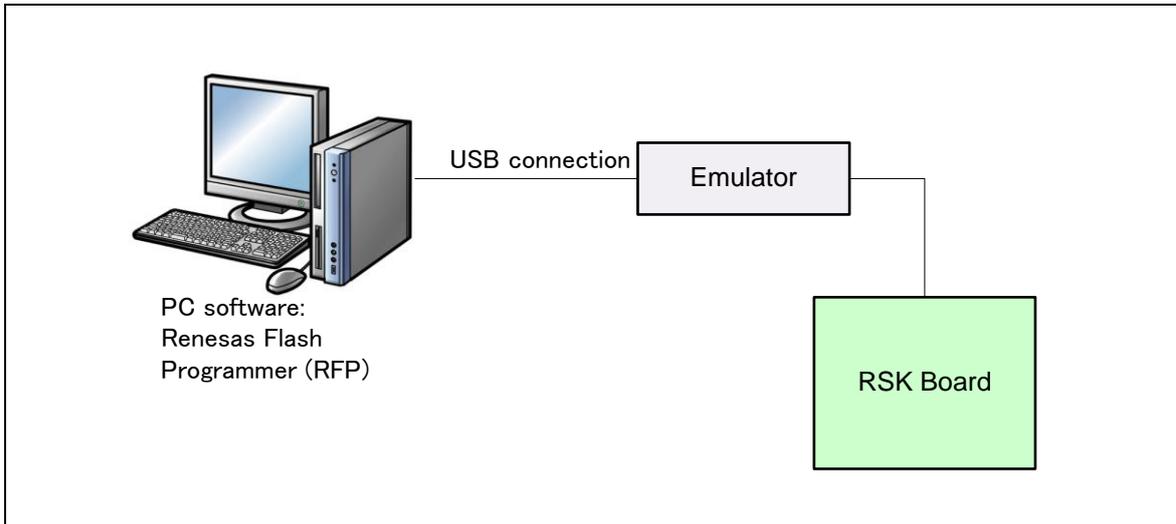
Figure 4-2   Connection Diagram Using an Emulator

b.   Not using an E1/E20 emulator
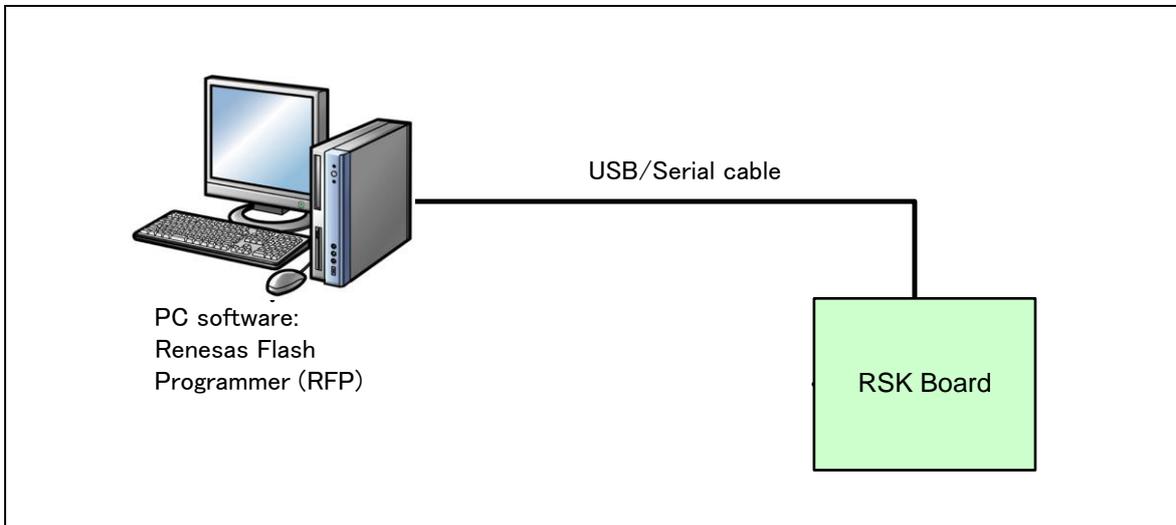
Figure 4-3   Connection Diagram with No Emulator

(2)   Writing the USB Peripheral Firmware Updater

Run the Renesas Flash Programmer (RFP) and, using the [**Browse]** for [User/data area] button, select USB Peripheral Firmware Updater file to be written from the Workspace/(MCU *name*) folder. Press **Start** to download the program to the target board. The write operation is complete when **PASS** is displayed in the output panel and Download Complete appears in the green area at the top of the panel.
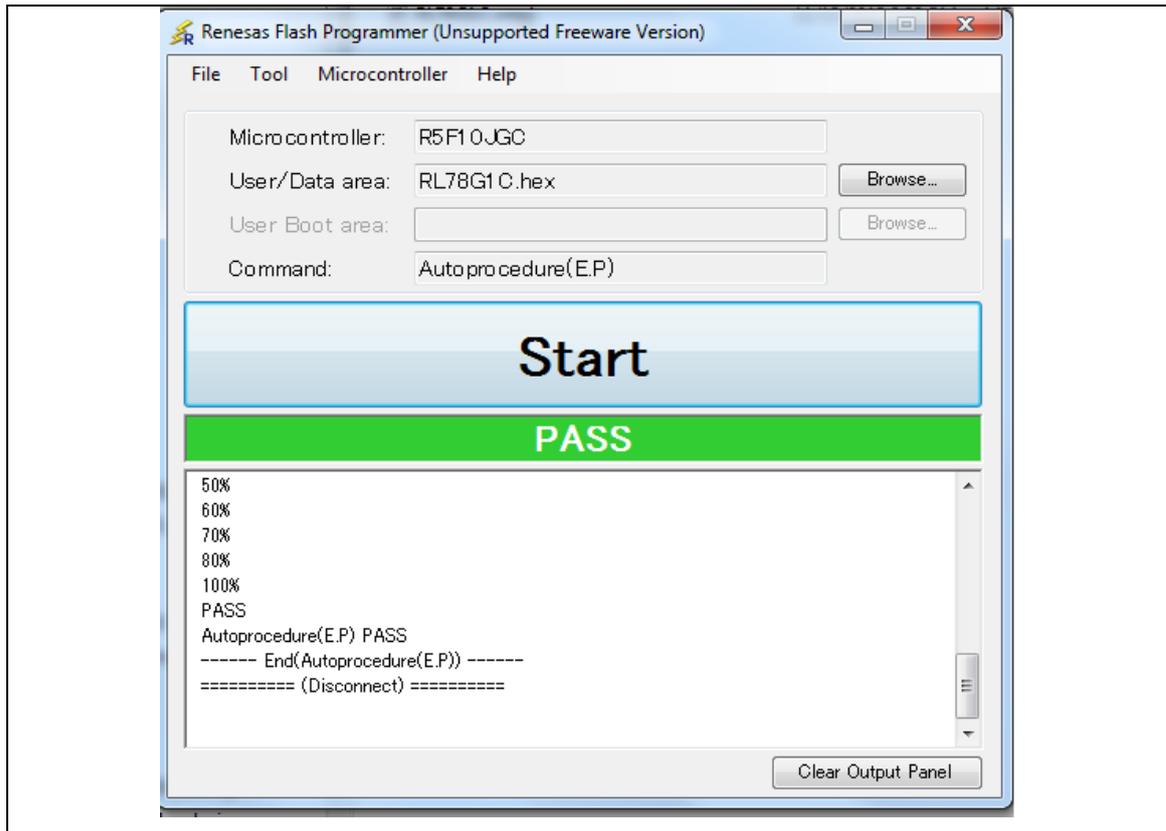
Figure 4-4   File Specification

Notes:

a.   Refer to the following URLs for more details on the Renesas Flash Programmer:

   URL:

   https://www.renesas.com/en-us/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html

b.   Refer to section **4.2.2 USB Peripheral Firmware Updater address assignment** for more details concering positioning of

   USB Peripheral Firmware Updater.

4.2.2   USB Peripheral Firmware Updater address assignment

This section explains the assigned address of this program.

USB Peripheral Firmware Updater can be assigned to any area after the address 0x2000.

The following shows the memory map for RL78/G1C. For more details, refer to the user's hardware manual corresponding to the

target MCU.
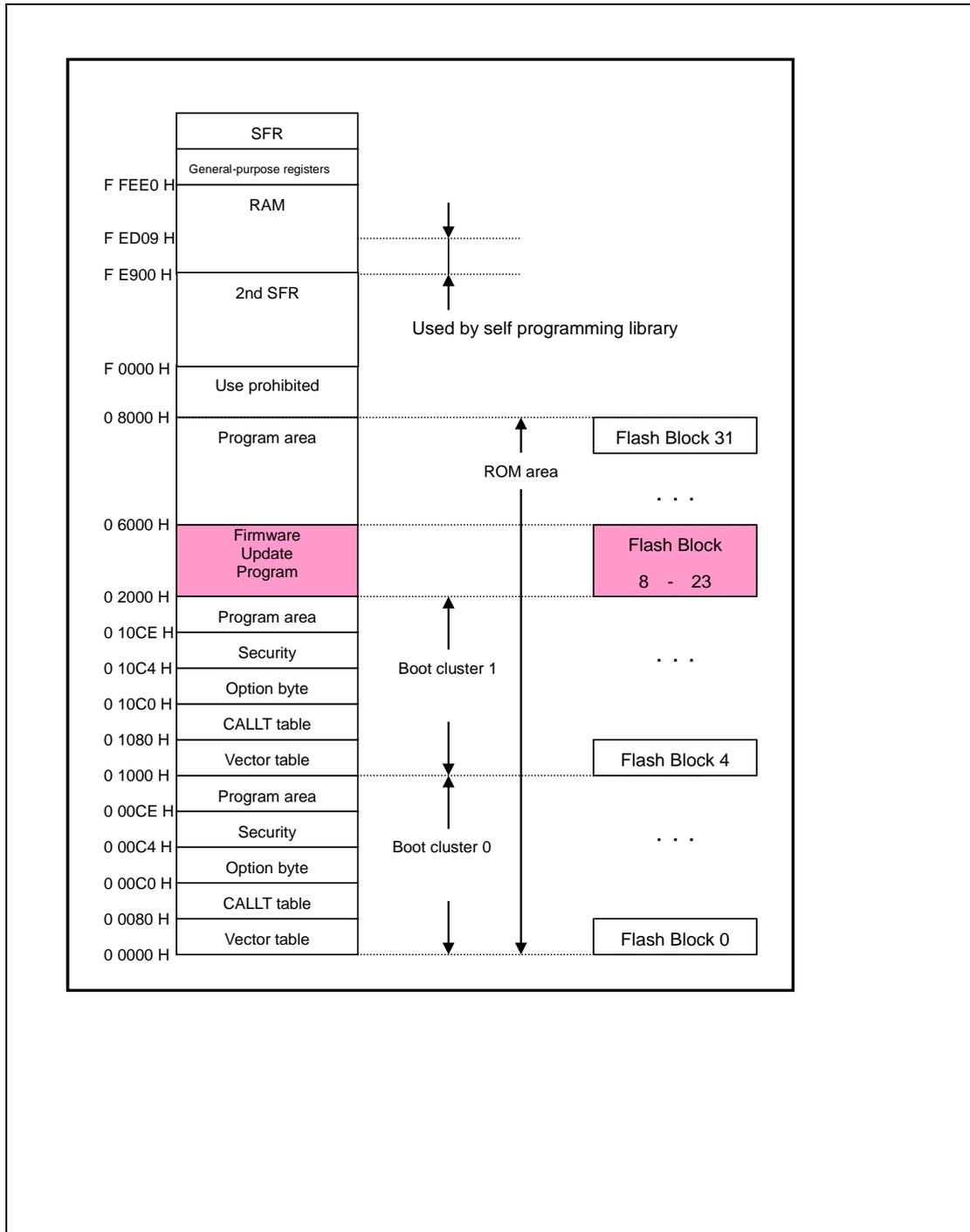
Figure 4-5   Memory Map

4.3   Execution of USB Peripheral Firmware Updater (user program write operation)

This section explains the sequence for **USB Peripheral Firmware Updater** execution and user program write operation.

(1).   Hardware preparation

To execute the write operation, detach the emulator and connect the PC and evaluation board with the USB cable. **Figure 4-6** shows the connection diagram.

USB connection

RSK Board
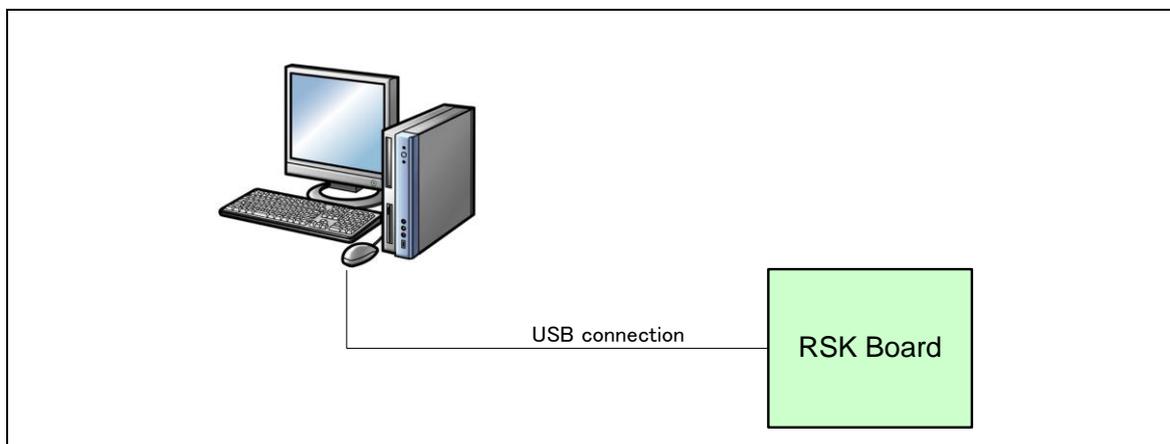
Figure 4-6   PC-Evaluation Board Connection Diagram

(2).   USB Peripheral Firmware Updater startup

Press the RESET button while holding down switch SW3 on the evaluation board. After transitioning to program mode, the board will wait for transfer data from the PC.

Note:

The PC used to run the file transfer application must be installed with a CDC driver. For details, refer to section **4.5 CDC Driver Installation.**

(3).   File transfer preparation

Run the file transfer application (USB Function Firmware Updater: PC-side software). Refer to Figure 4-8 for image.

Confirm the Windows device manager under "COM:" in the updater window, and then select the assigned COM number.

Note:

The COM number varies according to environment. Numbers 1 to 9 can be used the COM number.
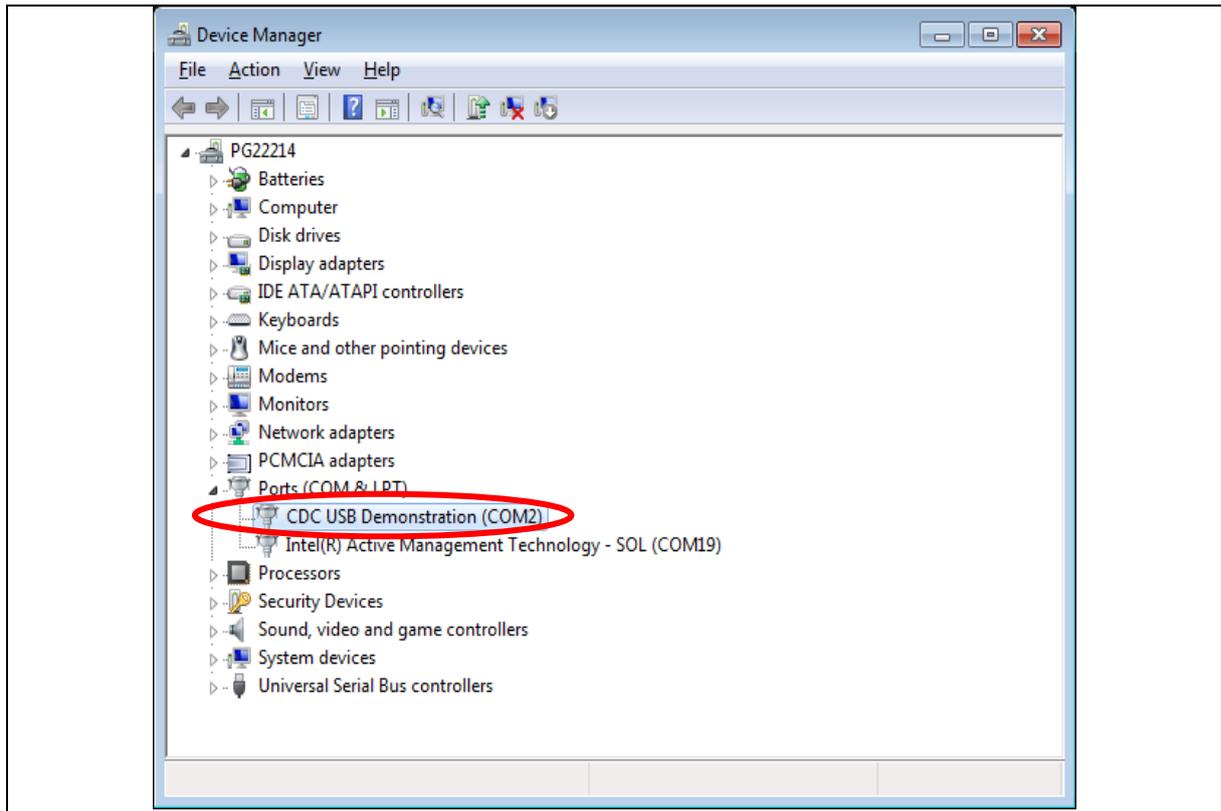
Figure 4-7 Device Manager Port Confirmation

(4). Transfer file selection

Click the **Load File** button in the file transfer application (USB Function Firmware Updater: PC-side software) and select the file to be written to the ROM. Then select the target MCU under **Device:**.
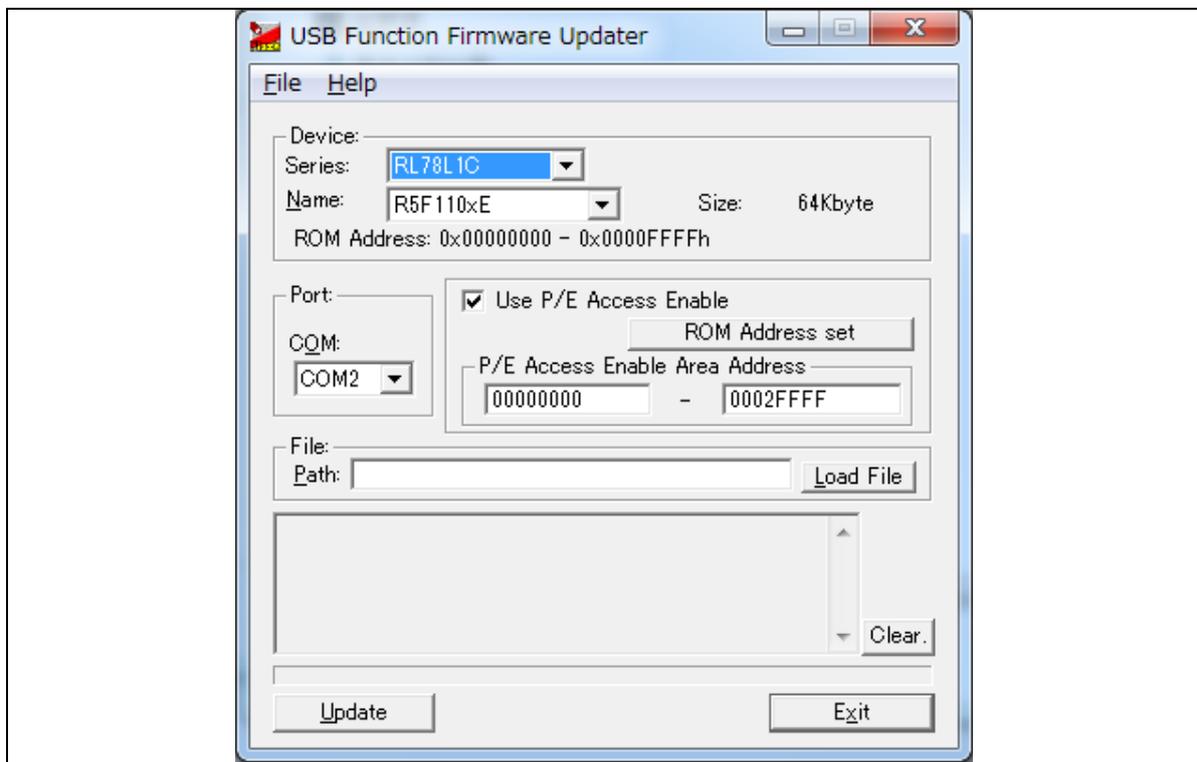
Figure 4-8 USB Firmware Updater GUI Software (file transfer application)

For details on how to use the file transfer application, refer to section **4.1 File Transfer Application (USB Function Firmware Updater) Startup.**

(5).   P/E limited area setting (P/E Enable Address setting)

Next, set the Program/Erase Enable Area within the ROM. For details, refer to section **4.1.1 P/E Access Enable Area Address**.
Sequence:

Check **Use P/E Access Enable** and specify the area.
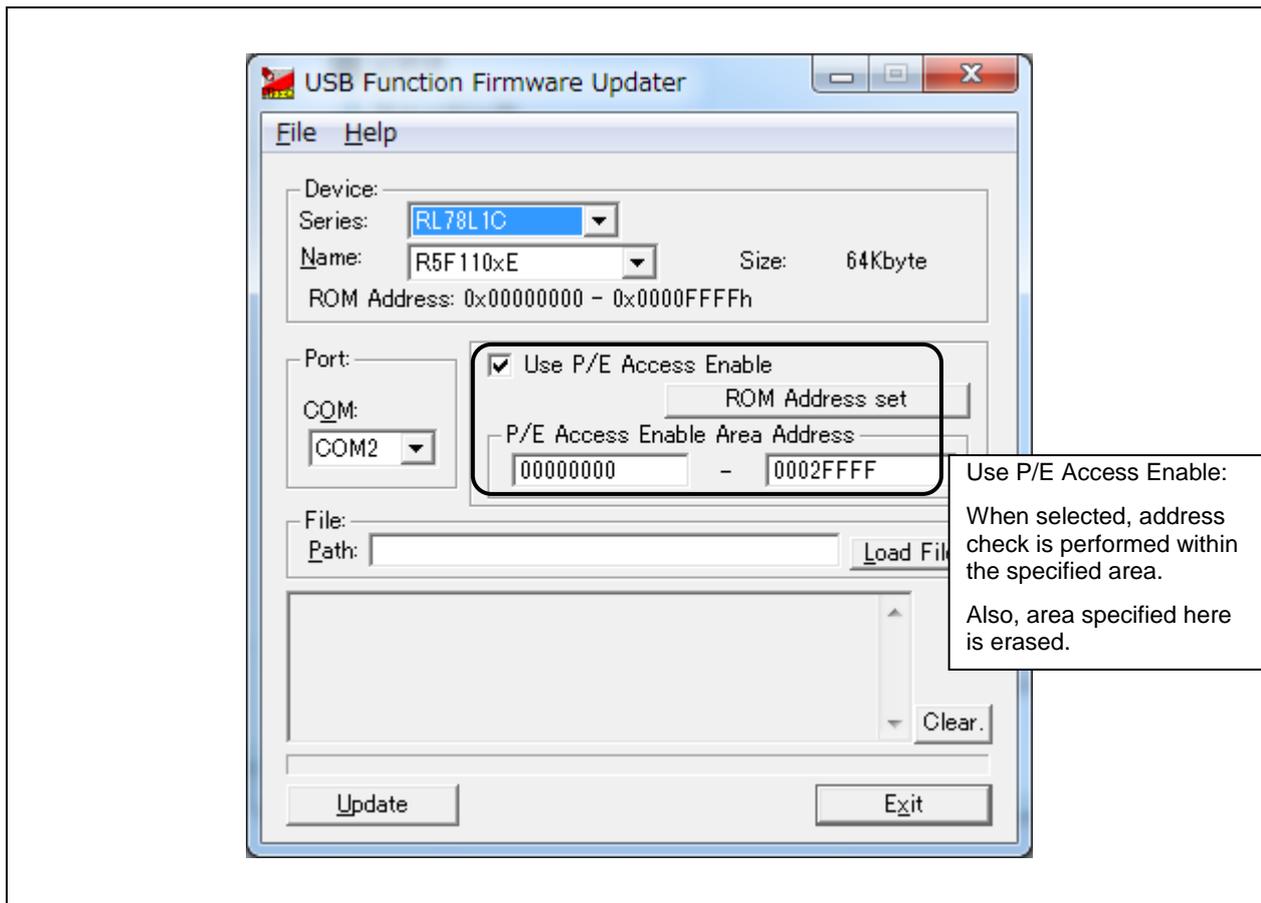
Figure 4-9   P/E Limited Area Setting

(6).   User program transfer execution

Click the **Update** button in the file transfer application GUI window. This will display the start message and start the file transfer or write operation processing.

Note:

If the user program write operation fails, the file transfer application interface will show a corresponding message. See section **8.4   Application Messages** for detailed explanations.

(7).   User program transfer complete

When the file transfer or write operation processing ends, the file transfer application interface will display "Success" to indicate the operation is complete. This ends the full write operation processing.
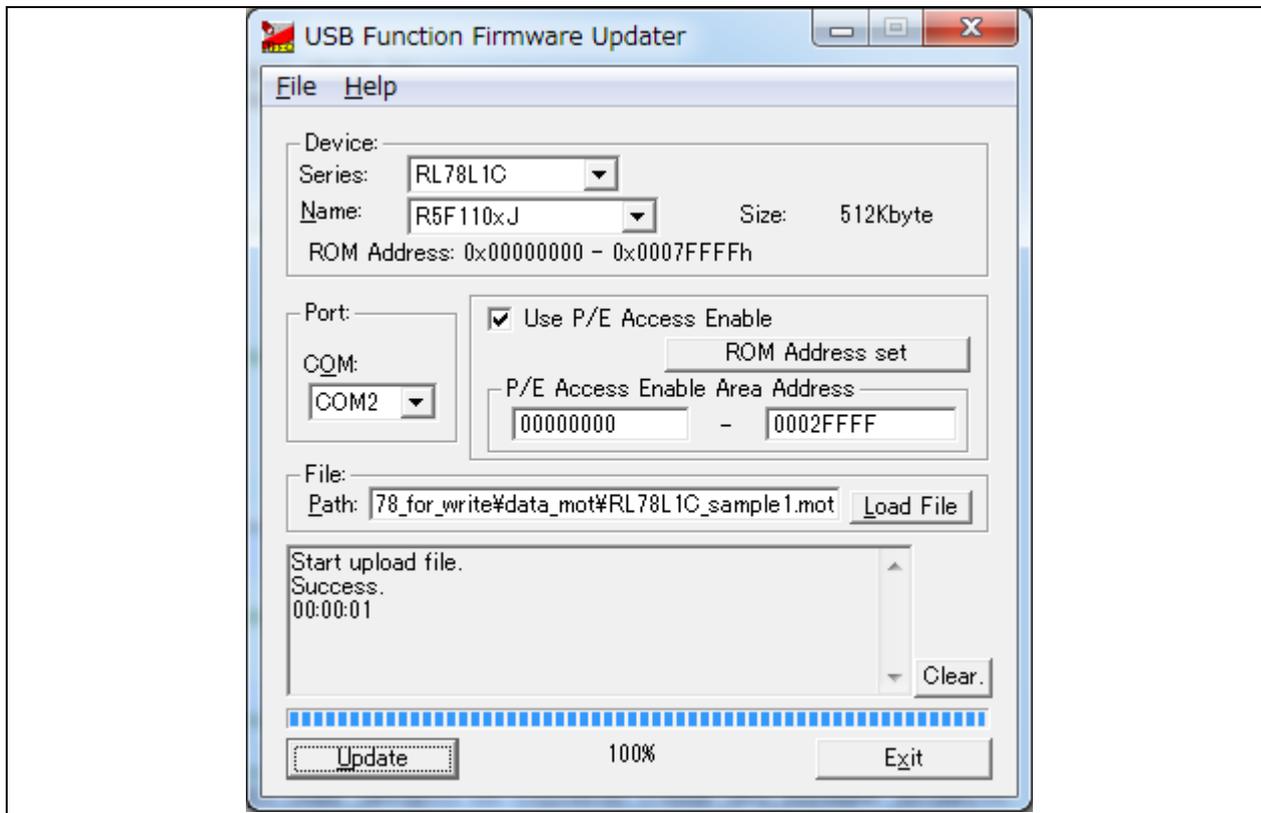
Figure 4-10   Write Processing Complete

(8).   User program startup

When the rewrite operation is completed, a software reset is executed automatically and the written user program is started.

When sample program 1 (user program) has been written to the MCU, the LEDs on the RSK board light up in consecutive order.

(9).   User program rewrite operation

This step rewrites the user program. Prepare sample program 2 (user program), restart **USB Peripheral Firmware Updater**, and repeat the sequence from step (4).

(10).   Rewrite complete

When the rewrite operation is complete, the evaluation board is reset, and the new user program is started. The RSK board LEDs light up if sample program 2 (user program) is written.

4.4   Cautions Regarding User Program Write Operation

If you write the user program to the area which already contains USB Peripheral Firmware Updater, please start over by re-writing USB Peripheral Firmware Updater.

4.5   CDC Driver Installation

   The PC used to run the file transfer application must be installed with a CDC driver. The wizard shown in Figure 4-11 will appear
on your screen and prompt the CDC driver installation when you connect your PC to target board used to write **USB Peripheral
Firmware Updater** to the MCU.

   (1).   Select **Update Driver Software** from the device manager.

   (2).   Select "**Browse my computer for driver software**".
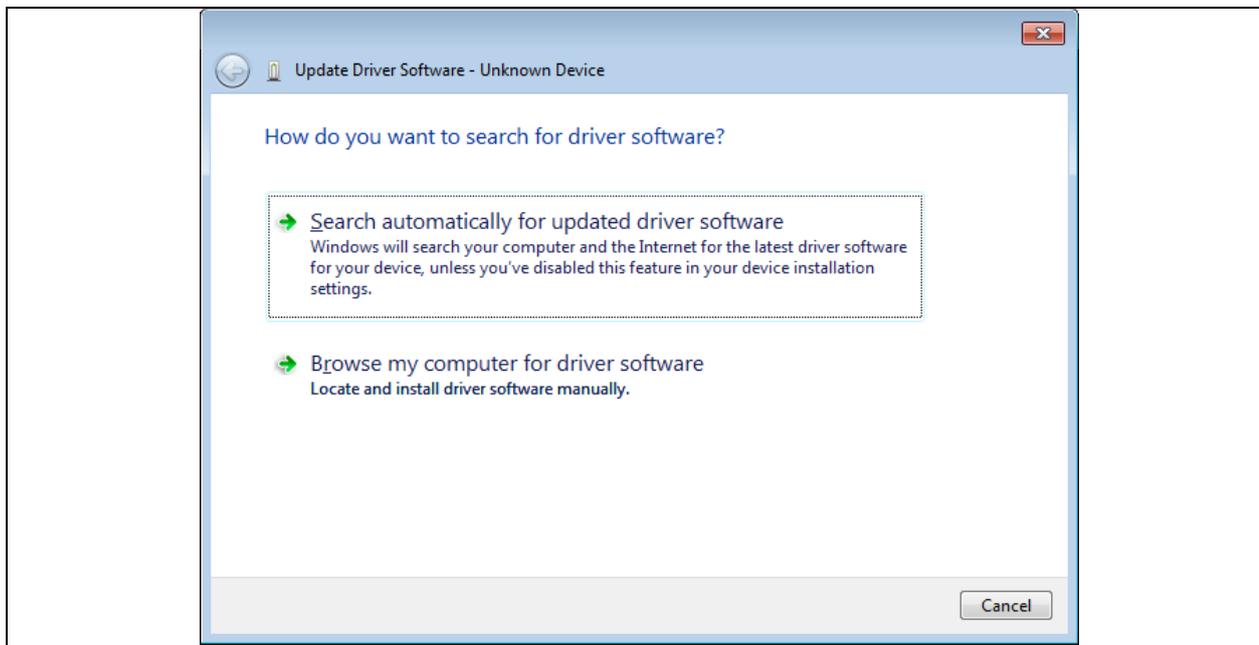
Figure 4-11   New Hardware Search Wizard

   (3).   Select **"Browse for driver software on your computer"**

       Click **Browse**, specify the folder in which the CDC_Demo.inf is stored, then click **"Next"**
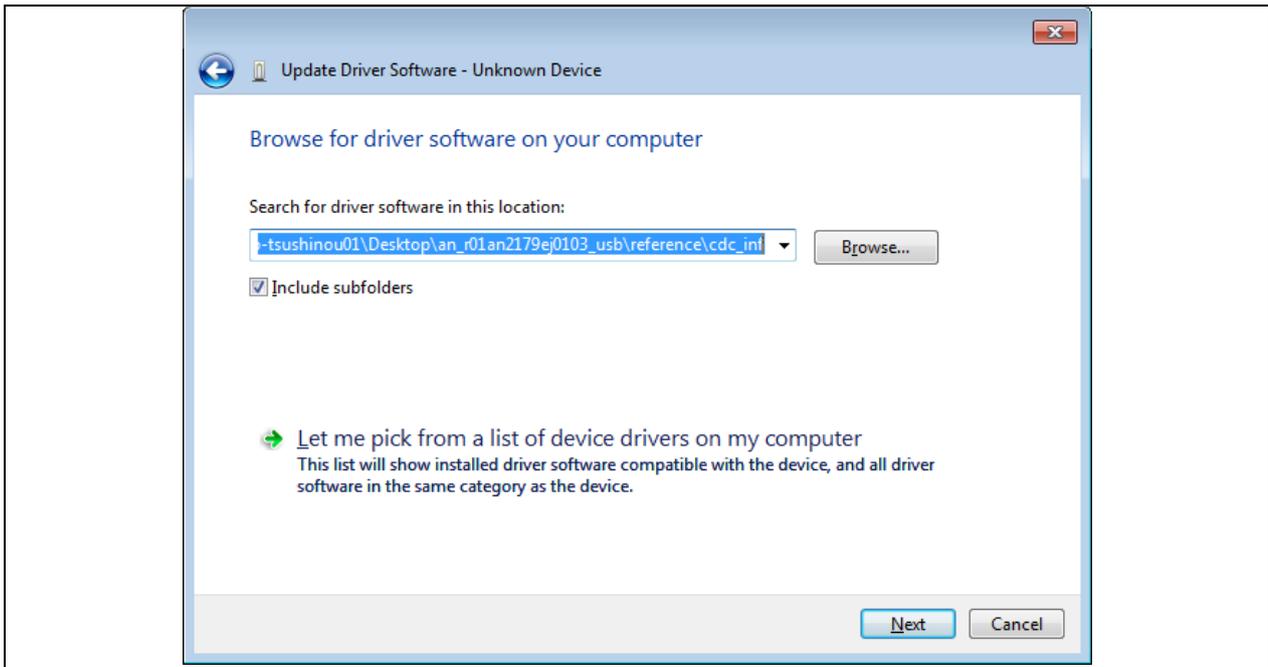
Figure 4-12   Select Driver Location

(4).   If the following installation confirmation screen appears, click **"Browse for driver software on your computer"**
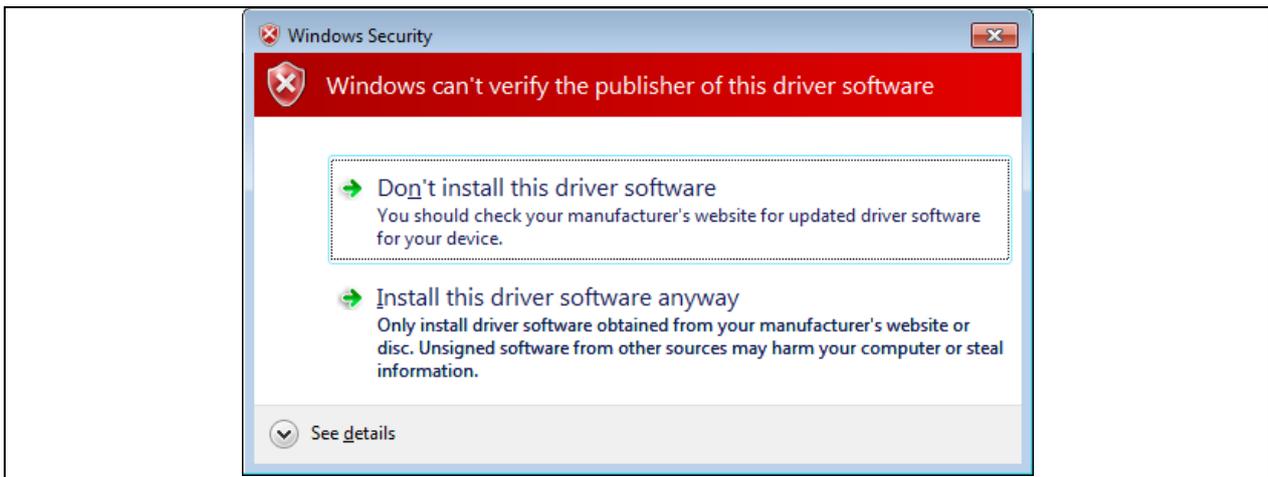
Figure 4-13   Installation Confirmation Screen

(5).   When the following window appears, the CDC driver has been successfully installed. Click **"Close."**
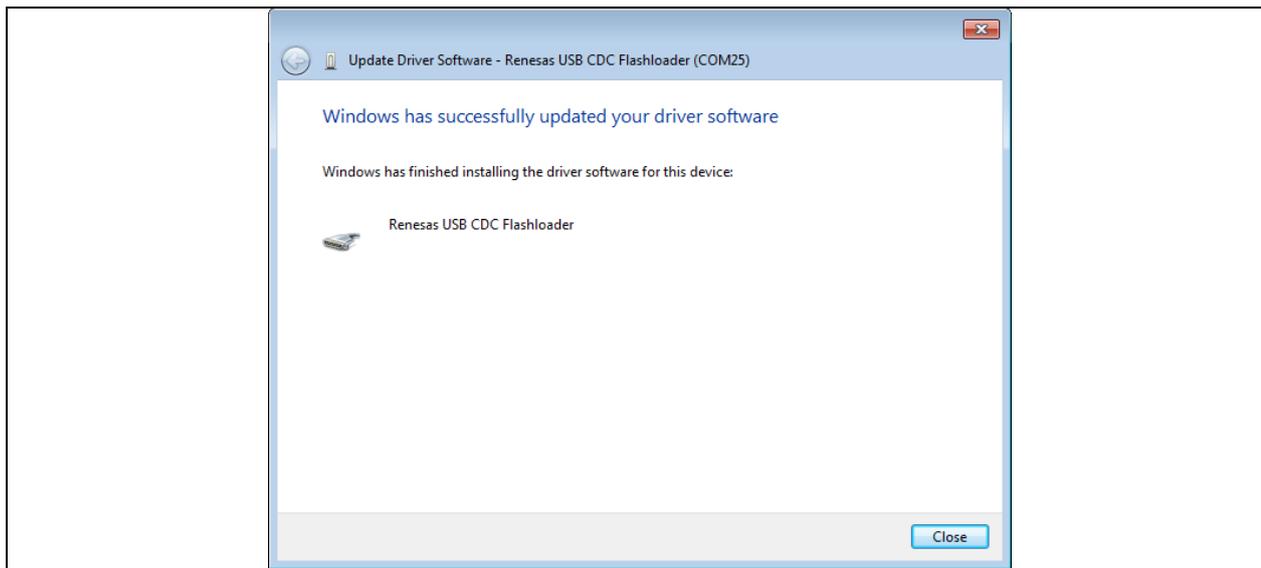
Figure 4-14   Installation Complete

* An error may occur when installing the driver in the Windows 8.1 environment. In this case the installation confirmation screen will

not appear.

5.   Cautions Regarding Creating the User Program

This sections explains cautions that apply when creating the user program

5.1   File Format

The program supports the following file formats.

　　　・Motorola type S (32-bit)

　　　・Motorola type S (standard)

　　　・Intel extended file format

5.2   UserApp Header Area (user application header)

When using this program to write a user program, you must include a UserApp Header (user application header) area in the user program. The size of the UserApp Header area should be a total of 8 bytes: 4 bytes for the user program start address storage area and 4 bytes for the security code storage area (see Figure 5-1).

Refer to section 6.1 **User Program Settings** for details on how to create the UserApp Header area.

UserApp Header
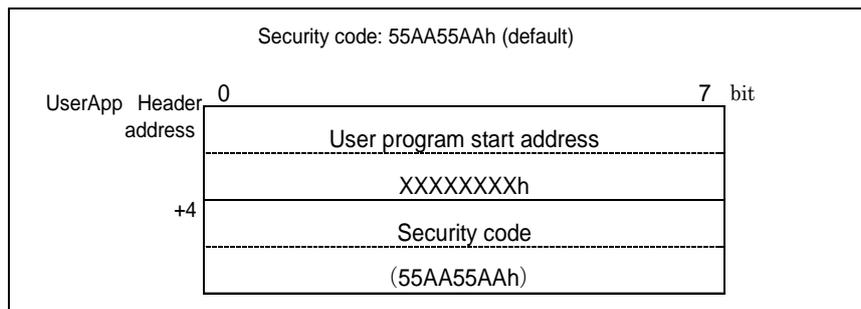
Figure 5-1   UserApp Header Area

This header information is read when **USB Peripheral Firmware Updater** is started up and transitions to the UserApp startup sequence. For details, refer to section **7.2.1 Power On Operation Flow.**

6. USB Peripheral Firmware Updater and User Program Settings

This section provides the setting contents required for **USB Peripheral Firmware Updater** and the user program.

6.1 User Program Settings

1. Setting Content 1

   Create the UserApp Header area in the user program according to the sample in Figure 6-1. For more details about the UserApp Header, see section **5.2　UserApp Header Area (user application header)**.

2. Setting Content 2

   Set the section for the UserApp Header area created in step 1 above, and set an arbitrary address to the section. Make sure the location of the UserApp Header area does not overlap with **USB Peripheral Firmware Updater** area (including fixed vectors).

```
/*****************************************************************************
    APPLICATION INTERFACE HEADER
    The purpose of the header is for an external application to be able to read
    certain values from known addresses.
    - Start address of UserApp.
    - Security code must match what PCDC Flashloader expects.         Section specification
    - For revision purposes of applications etc.
    - Do not change the order of these variables!
*****************************************************************************/
#pragma section C UserApp_Head_Sect

/* START ADDRESS of user application header data    - Appheader address + 0x00. */
const uint32_t  userapp_entry_addr =   (uint32_t)  userprog_start;

/*                                                 - Appheader address + 0x04. */
const uint32_t  userapp_sec_code =     (uint32_t)  USERAPP_SECURITY_CODE;
                                                                     Header
/* Total header area size 12 bytes */                               information
```

Figure 6-1　UserApp Header Code Example

Sequence:

First select [Properties] → [C/C+ Build] → [Settings]. Next, select the Tool setting tab, and select [Linker] → [Section].
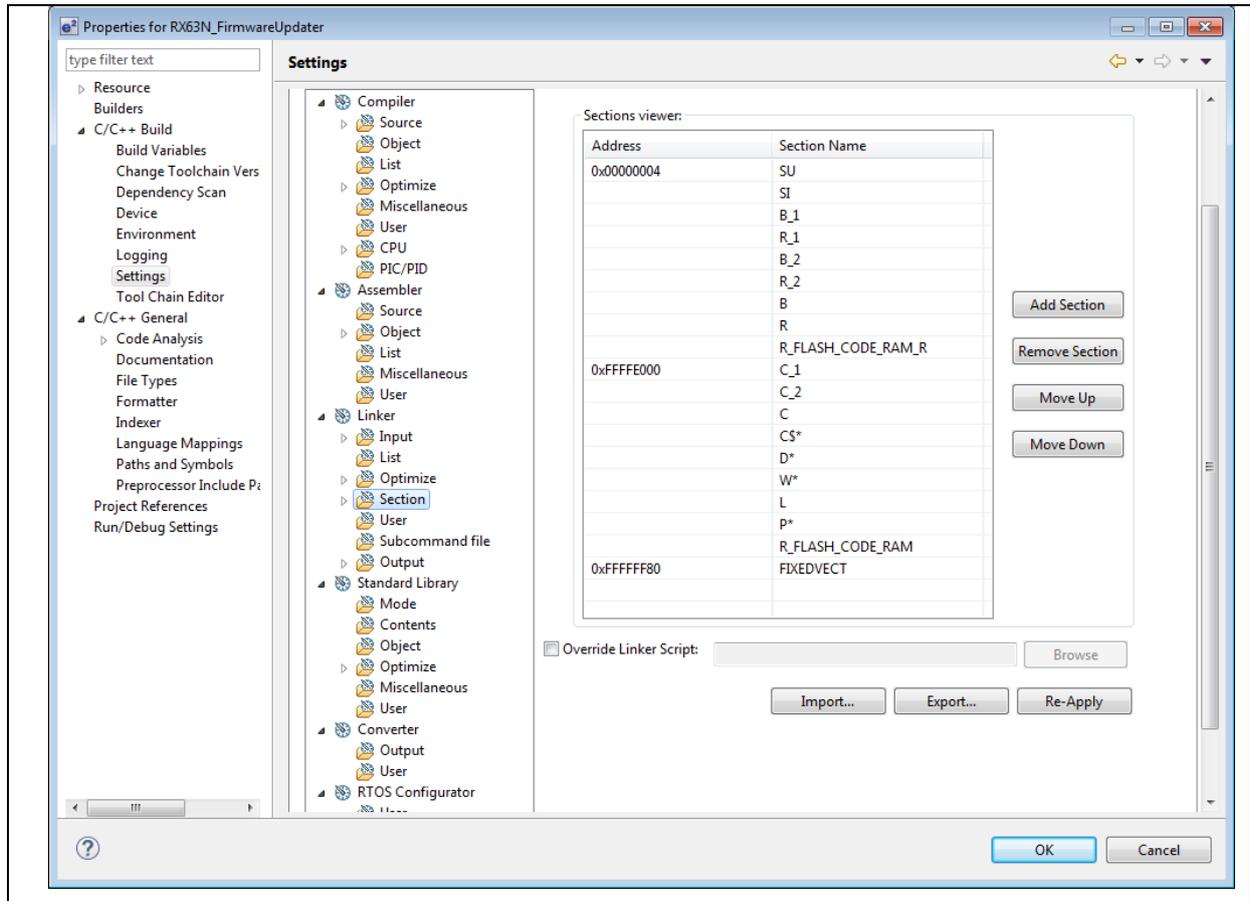
Figure 6-2   Example of Section Settings for Sample Program

6.2   USB Peripheral Firmware Updater Settings

1. Setting Content 1 (r_config¥r_usb_updater_config.h)

   USB Peripheral Firmware Updater references the UserApp Header area in the user program. Therefore, if you change the assigned address of the UserApp Header area, make sure you also change this program to reference the revised UserApp Header area. In the same manner, if you change the security code value, make sure you make the corresponding changes in this program. Refer to section **5.2 UserApp Header Area (user application header)** about UserApp Header area.

   (1).   **USERAPP_HEADER_ADDR** definition setting
   Set the assigned address of the UserApp Header area to the *USERAPP_HEADER_ADDR* definition in the main.c file.

   #define   USB_CFG_USERAPP_HEADER_ADDR   Assigned address of UserApp Header area

   (2).   **USERAPP_SECURITY_CODE** definition setting
   Set the security code specified in the UserApp Header area to the *USERAPP_SECURITY_CODE* definition in the main.c file.

   #define   USB_CFG_USERAPP_SECURITY_CODE   Security code

   (3).   **Firmware Updater program location address setting**
   Set the location address for Firmware Updater program to the following definitions

   #define   UPDATER_SECTION_START_ADDRESS   Firmware Updater program start address

2. Setting Content 2
   This program jumps to **USB Peripheral Firmware Updater** or the user program depending on the state of SW (Switch) on the evaluation board. The process for determining the state of SW depends on the board specifications. Please adjust the determination process to meet the target board specifications. This determination process is performed in the main function.

3. Setting Content 3
   USB Peripheral Firmware Updater does not write the *option byte* area in the user program's option. Because the *option byte* area of this firmware updater is also used by the user program, set the *option byte* in accordance with this firmware updater program.

6.3   User Program Reset Vector

   When **USB Peripheral Firmware Updater** is implemented to the user system, the reset vector of **USB Peripheral Firmware Updater** are used; and the reset vectors of the user program is not written to the MCU.

6.4   User Program Position

   Make sure you assign the user program(except the vector table area) to ROM area which does not overlap with the area written by **USB Peripheral Firmware Updater**. Assign the user program locations according to section settings.

   Note:

   1. Set all sections of the user program (except the vector table area) according to the ROM area assignments shown in the list below.

MCU	Assignable Areas for User Program		
RL78/G1C	0x00002000	-	0x00007FFF

RL78/L1C	0x00002000	-	0x0003FFFF

2. When writing the user program to the ROM, be careful not to write the program to the area containing **USB Peripheral Firmware Updater** as it is protected by P/E access limitations. For more details, refer to section 4.1.1 P/E Access Enable Area Address.

3. Although the Flash self-programming library occupies part of the RAM area, it is only used when executing **USB Peripheral Firmware Updater** and will not affect the user program operations.

7. USB Peripheral Firmware Updater Explanation

This section explains each file used by **USB Peripheral Firmware Updater**.

7.1   File/Folder Configuration

The following shows the source file/folder configuration of this program.

```
(Compiler name)
   +src
      +————APL              [FW Updater application]
      +————config           [FWUpdater configuration file]
      +————FSL              [Flash Self Library]
      +————HwResource       [Hardware Resource Setting]
      +————USB               [USB driver]
                +———— inc                      USB driver common header file
                +———— src                      USB driver
```
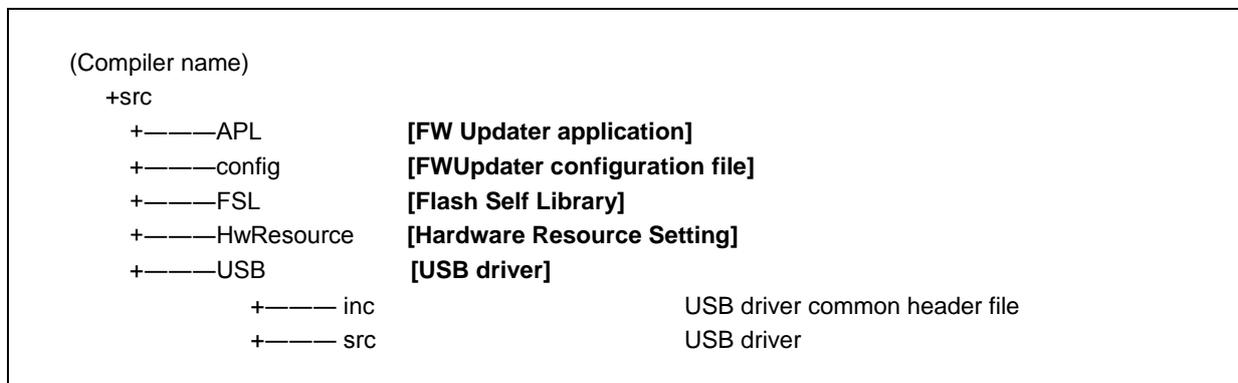
Figure 7-1   USB Peripheral Firmware Updater Folder Configuration

7.1.1   config Folder

This folder stores all the setting files for the target MCU.

Table 7-1   API Header Files

File Name	Description
r_usb_fwupdater_config.h	Flash ROM rewrite program setting file

7.1.1   FSL Folder

Please store the flash self library and the include file.

7.1.2   APL Folder

This folder stores **USB Peripheral Firmware Updater** source files.

Table 7-2   USB Peripheral Firmware Updater Source Files

File Name	Description
src¥main.c	C language main function description file
src¥r_usb_main_flashapi.c	Flash ROM rewrite program processing file
inc¥r_usb_main_flashapi.h	Flash ROM rewrite program header file

7.1.3   USB Folder

This folder stores the CDC (USB) source files and header file.

Table 7-3   USB Peripheral Firmware Updater Source Files

File Name	Description
inc¥r_usb_fwupdater_defreg.h	USB register initialization, setting definitions
inc¥r_usb_fwupdater_defusr.h	USB register setting. To be set by the user.
inc¥r_usb_fwupdater_extern.h	Function Extern
inc¥r_usb_ctypedef.h	Type definition
inc¥r_usb_defvalue.h	USB setting definition
inc¥r_usb_usrconfig.h	USB user setting file
inc¥r_usb_sysdef.h	USB system information
src¥r_usb_fwupdater_api.c	USB transmit/receive, initialization processing file
src¥r_usb_fwupdater_basic.c	USB main processing
src¥r_usb_fwupdater_classcdc.c	USB CDC processing
src¥rl78usbmcu.c	USB interrupt initialization, port setting file
src¥r_usb_fwupdater_usbreg.c	USB register setting, etc.

7.1.4   HwResource Folder

The driver for the peripheral which RSK supports is stored in this folder.

Table 7-4   Peripheral Driver File

File Name	Description
src¥keydriver.c	Driver file for RSK switch
src¥lcddriver.c	Driver file for RSK LCD
src¥rl78usbmcu.c	RL78 setting file

7.2   Boot Processing

Boot processing indicates the processing executed after the MCU is reset and before the main function (C language description: main()) is executed.

In RL78 MCUs, boot processing chiefly performs the following as initialization after reset:

- Allocate stack area and set stack pointer

- Allocate argument area for main function

- Initialize data area and stack area

- Branch to user program and initialize MCU peripheral devices in hdwinit function

- Branch to main function

After reset, processing jumps from **USB Peripheral Firmware Updater** to the user program. Therefore, make sure **USB Peripheral Firmware Updater** is complete and the above-described MCU initializations are executed.

7.2.1   **Power On Operation Flow**

This section explains the operation flow after power is turned on for **USB Peripheral Firmware Updater**.
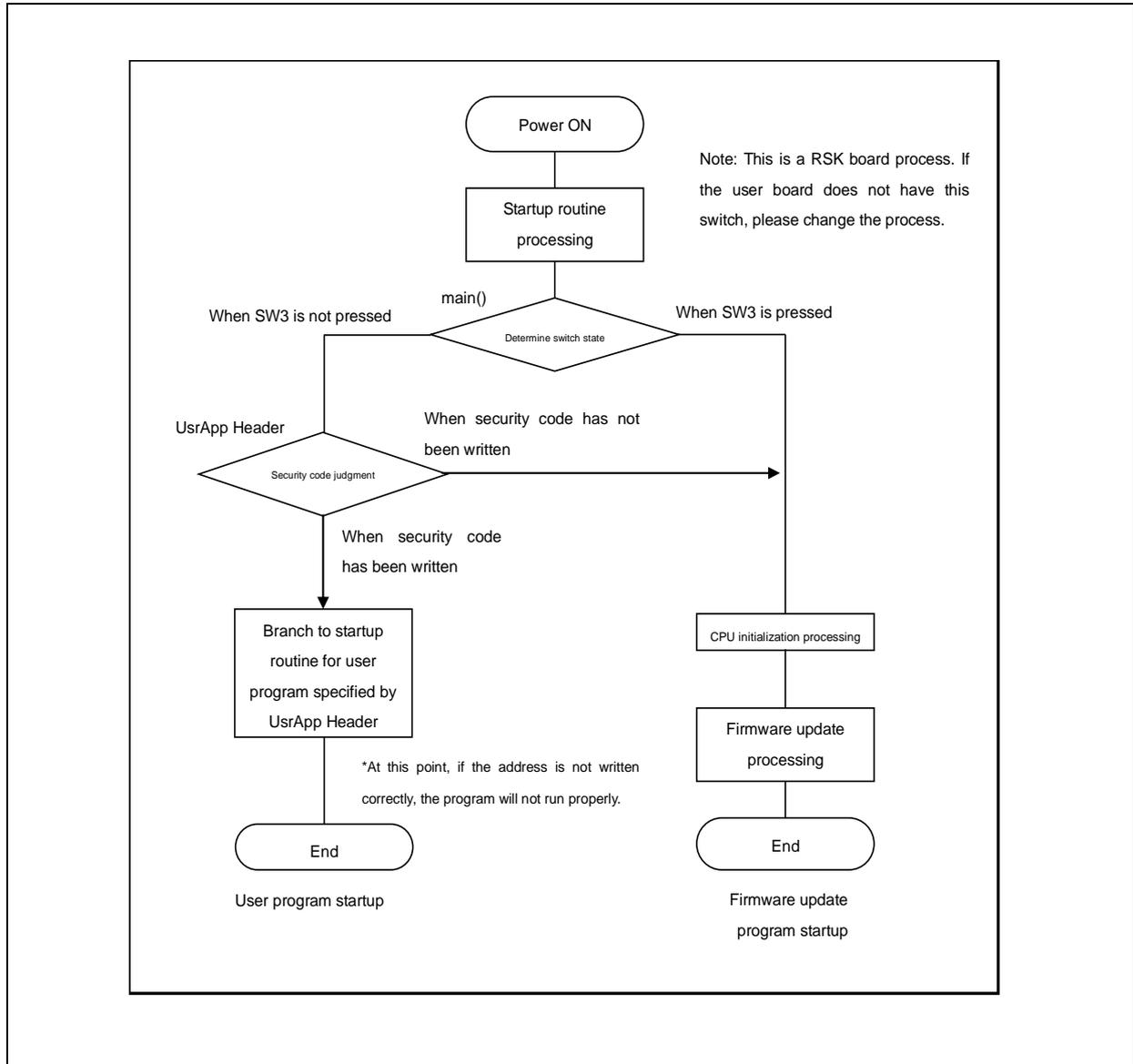
Figure 7-2   Power On Operation Flow

For information regarding branch address to security code and user program, refer to section **5.2   UserApp Header Area (user application header)**

Note that even if the security code in the UsrApp Header area is set correctly, if the start address of the user program is incorrect, the user program will not operate properly.

7.2.2   **User program startup conditions**

The user program set in the UsrApp Header area is started up when all of the following conditions are met:

a.   Correct security code is set

b.   Correct user program start address is set

If the security code does not match (is incorrect), only **USB Peripheral Firmware Updater** will start up; the user program

will not run.

7.2.3   **USB Peripheral Firmware Updater startup conditions**

1.   When user program has been written to ROM:

USB Peripheral Firmware Updater starts up when RESET is executed while SW3 on the evaluation board is pressed.

2.   When user program has not been written to ROM:

USB Peripheral Firmware Updater starts up when power is turned on.

7.3   **Cautions**

USB Peripheral Firmware Updater determines whether to jump to the user program or continue on with the firmware update

program by judging the state of SW on the evaluation board. This judgment process is dependent on the board's specifications.

Please change the judgment process to meet the specifications of your evaluation board. The judgment processing is performed in the

main function of **USB Peripheral Firmware Updater**.

7.4   Functions for USB Peripheral Firmware Updater

This section describes all functions used in the Updater other than simple Flash API-related functions.

7.4.1   Data Type

Data types applicable in **USB Peripheral Firmware Updater** are listed below.

Table 7-5   Data Type

Data Type	Specifier	Valid Range
int8_t	signed char	Signed 8-bit integer
int16_t	signed short	Signed 16-bit integer
int32_t	signed long	Signed 32-bit integer
uint8_t	unsigned char	Unsigned 8-bit integer
uint16_t	unsigned short	Unsigned 16-bit integer
uint32_t	unsigned long	Unsigned 32-bit integer

7.4.2   Structures

Table 7-6   Start Record Definitions

Data Type	Variable Name	Description
uint8_t	device_type	Device type (0x00 is fixed)
uint8_t	rsv1[1]	Reserved
uint8_t	rom_start_addr[4]	ROM erase start address
uint8_t	rom_end_addr[4]	ROM erase end address
uint8_t	rsv2[1]	Reserved
uint8_t	checksum	Check sum

Table 7-7   End Record Definitions

Data Type	Variable Name	Description
uint8_t	record_type	Record type
uint8_t	record_len	Record length
uint8_t	dev_type	Device type (0x00 is fixed)
uint8_t	checksum	Check sum

Table 7-8   Response Record Definitions

Data Type	Variable Name	Description
uint32_t	record_type	Record type
uint8_t	record_len	Record length
uint8_t	response_type	Response type ACK/NAK
uint8_t	err_field	Error code
uint8_t	checksum	Check sum

Table 7-9   ROM Write Strucure Definitions

Data Type	Variable Name	Description
uint8_t	recv_data[ROM_WRITE_SIZE]	ROM write buffer
uint32_t	now_addr	Current write address
uint32_t	next_addr	Next write address
uint32_t	dest_addr	Write base address [dest_addr + wr_count * ROM_WRITE_SIZE] will become destination address
uint16_t	wr_count	Data count (number of writes) from base address
uint16_t	counter	Current number of data <0 to ROM_WRITE_SIZE-1>

Each data type has a fixed Flash write size (ROM_WRITE_SIZE). Data is copied from the receive data buffer to the ROM write buffer until the fixed number of write data is accumulated.

The write base address is maintained while the data write continues. When the write address surpasses the ROM_WRITE_SIZE from the previous address, the base address is re-established and the write data count is cleared.

7.4.3   Flash write main processing functions

Table 7-10 Main Processing Functions

File Name	Function Name	Processing Description
main.c	**main**	Initialization, judgment to jump to user program
main.c	**cdc_read**	CDC read detection
main.c	**cdc_main**	Receive data processing branch (erase/write/end judgment)
main.c	**cf_code_copy**	Copy code to RAM
main.c	**send_response_record**	Send response to application
main.c	**sub_Byte2ddress**	Convert4-byte address to address value in unsigned long
main.c	**subproc_start_record**	Initial process when start record (ROM erase, etc.) is received
main.c	**sub_write_initial**	Flash write variable initialization
main.c	**sub_flashwrite_proc**	Flash write judgment, write processing
main.c	**jump_to_userapp**	Jump code to user program
r_usb_main_flashapi.c	**FSL_Func_Write**	Call function for ROM write API. Processing branches according to type.
r_usb_main_flashapi.c	**FSL_Func_Erase**	Call function for ROM erase API. Processing branch according to type.
r_usb_main_flashapi.c	**FSL_Func_SetAccessWindow**	Call function for ROM access enable API. Type 1 only.

Table 7-11   main Function

Function Name		**main**	
Description Format		void main (void)	
Function		Entry function at start. Executes initialization processing and branching to **USB Peripheral Firmware Updater** or user program.	
I/O	Input	None	
	Output	None	
Remarks		For operation details, refer to section 7.4.5 Branch to firmware update program.	

Table 7-12   cdc_read Function

Function Name		cdc_read
Description Format		static uint16_t cdc_read(void)
Function		CDC read detection
I/O	Input	None
	Output	uint16_t: read results
Remarks		CDC_BLK_OUT_OK: read complete
		CDC_NO_CONFIGUED: CDC not connected
		CDC_DETCH: CDC connection error
		CDC_BLK_OUT_ERR: read error

Table 7-13   cdc_main Function

Function Name		cdc_main
Description Format		void cdc_main (void)
Function		main processing for **USB Peripheral Firmware Updater**
I/O	Input	None
	Output	None
Remarks		For operation details, refer to section 7.4.7 **Firmware update main processing**.

Table 7-14   cf_code_copy Function

Function Name		cf_code_copy
Description Format		static void cf_code_copy (void)
Function		Restore all information of specified Flash memory block to RAM area.
I/O	Input	None
	Output	None
Remarks		Access to ROM in Flash ROM write mode (ROM P/E mode) is restricted. Therefore, codes used for ROM write and ROM erase must be restored to RAM.

Table 7-15   subproc_start_record Function

Function Name		subproc_start_record
Description Format		static flash_err_t subproc_start_record(uint16_t flashseq_check)
Function		Erase specified ROM area when start record is received.
I/O	Input	uint16_t flashseq_check: Flash write transition status.
	Output	None
Remarks		None

Table 7-16 send_response_record **Function**

Function Name	send_response_record
Description Format	static void send_response_record (U8 response_type, U8 response_field)
Function	Transmit data to host side.
I/O Input	None
Output	None
Remarks	For details concerning communication protocol, refer to section 9 **Data Transmission Specification**.

Table 7-17 sub_Byte2num **Function**

Function Name	sub_Byte2num
Description Format	static uint32_t sub_Byte2num(U8 * dat, uint16_t size)
Function	Macro that returns the byte row as an integer in a size-specific communication.
I/O Input	Dat: byte row Size: size to be connected
Output	uint32_t: return calculated results
Remarks	None

Table 7-18 sub_write_initial Function

Function Name	sub_write_initial
Description Format	static void sub_write_initial (void)
Function	Flash write variable initialization
I/O Input	None
Output	None
Remarks	None

Table 7-19 sub_flashwrite_proc Function

Function Name	sub_flashwrite_proc
Description Format	static flash_err_t sub_flashwrite_proc(void)
Function	Flash write processing
I/O Input	None
Output	flash_err_t: Write result judgment
Remarks	None

Table 7-20   jump_to_userapp Function

Function Name		jump_to_userapp
Description Format		static void jump_to_userapp (void)
Function		Jump to user program
I/O	Input	None
	Output	None
Remarks		For more information concerning the jump destination address, refer to section 5.2 **UserApp Header Area (user application header)**.

Table 7-21   FSL_Func_Write Function

Function Name		FSL_Func_Write
Description Format		flash_err_t FSL_Func_Write (void)
Function		Call function for ROM erase API.
I/O	Input	None
	Output	flash_err_t: processing result
Remarks		None

Table 7-22   FSL_Func_Erase Function

Function Name		FSL_Func_Erase
Description Format		flash_err_t FSL_Func_Erase (const uint32_t start_addr, const uint32_t end_addr, uint16_t disable_check)
Function		Call function for ROM erase API. Processing branch according to type.
I/O	Input	start_addr: erase start address (erase block that includes address) end_addr: erase end address (erase block that includes address) disable_check: Disable access to block that includes start/end address, delete one block from erase range (option setting memory support now under consideration)
	Output	flash_err_t: processing result
Remarks		None

Table 7-23   FSL_Func_SetAccessWindow Function

Function Name		FSL_Func_SetAccessWindow
Description Format		flash_err_t FSL_Func_SetAccessWindow (const uint32_t start_addr, const uint32_t end_addr)
Function		Call window function for ROM access area setting API
I/O	Input	start_addr: ROM access enable start address end_addr ROM access enable end address
	Output	flash_err_t: processing result
Remarks		This process is only performed for Flash type 1. The access-enabled

	address is set assuming the end address will be truncated by 10-bits because it is retained after a 10-bit shift. This will become an access enabled area, so there will be no problems in processing a large area.

7.4.4   USB Driver Functions

Table 7-24 lists the USB driver functions.

Table 7-24   USB Module Functions

File Name	Function Name	Processing Description
r_usb_fwupdater_api.c	**bulk_in_start**	USB bulk receive
r_usb_fwupdater_api.c	**bulk_out_start**	USB bulk transfer
r_usb_fwupdater_api.c	**usb_init**	USB initialization processing
r_usb_fwupdater_basic.c	**usbint**	USB interrupt processing
r_usb_fwupdater_basic.c	**save_request**	Get request information
r_usb_fwupdater_basic.c	**ctrl_read_data_stage**	Data processing
r_usb_fwupdater_basic.c	**ctrl_write_nodata_stage**	Non-data processing
r_usb_fwupdater_basic.c	**intr_int_pipe0**	Pipe 0 interrupt initialization
r_usb_fwupdater_basic.c	**bemp_int_pipe0**	Interrupt BEMP pipe0
r_usb_fwupdater_basic.c	**intr_int**	Interrupt initialization
r_usb_fwupdater_basic.c	**intr_int_read**	Receive interrupt initialization
r_usb_fwupdater_basic.c	**intr_int_write**	Transmit interrupt initialization
r_usb_fwupdater_basic.c	**ctr_read_start**	Receive start
r_usb_fwupdater_basic.c	**ctr_write_start**	Transmit start
r_usb_fwupdater_basic.c	**write_fifo**	Write specified data to specified USB FIFO
r_usb_fwupdater_basic.c	**read_fifo**	Read specified data buffer from USB FIFO
r_usb_fwupdater_basic.c	**chg_port**	Read specified buffer size from the USB FIFO
r_usb_fwupdater_basic.c	**req_get_descriptor**	Get descriptor
r_usb_fwupdater_basic.c	**req_set_configuration**	Configuration
r_usb_fwupdater_cdcclass.c	**serial_init**	CDC initialization
r_usb_fwupdater_classcdc.c	**reset_ep**	Pipe Configuration
r_usb_fwupdater_classcdc.c	**cdc_init**	Serial initialize
r_usb_fwupdater_classcdc.c	**class_write_data_stage**	Class request is "get line coding".
r_usb_fwupdater_classcdc.c	**class_read_data_stage**	Class request is "set line coding".
r_usb_fwupdater_classcdc.c	**class_write_nodata_stage**	Class request is "set control line state".
rl78usbmcu.c	**usb_cpu_mcu_initialize**	MCU initialization
rl78usbmcu.c	**usb_int_init**	USB interrupt initialization
rl78usbmcu.c	**delay_xus**	Wait
rl78usbmcu.c	**usb_cpu_int_disable**	USB interrupt disable
rl78usbmcu.c	**usb_cpu_usbint_init**	USB interrupt initialization

7.4.5   Branch to USB Peripheral Firmware Updater

The main() function in **USB Peripheral Firmware Updater** performs branch judgment to determine whether to jump to the user program or to continue with **USB Peripheral Firmware Updater**.

After conditional branching is performed, the CPU build-in functions and peripheral circuits are initialized and **USB Peripheral Firmware Updater** is executed.

```
void main(void)
{
  if (usb_cpu_sw_data() != SW3_PUSH)        ← Judgment for starting up the
  {                                            user application
      if (*(((uint32_t *)USB_CFG_USERAPP_HEADER_ADDR) + SEC_CODE_OFFSET) ==
             ((uint32_t)USB_CFG_USERAPP_SECURITY_CODE))
      {                                        ← Startup user application
          usb_cpu_int_disable();
          jump_to_userapp();
      }
  }
  rr.record_type = REC_TYPE_RESPONSE;
  rr.record_len = NR_RESPONSE_BYTES;

  usb_init();
  FSL_flashInit();

  while(1)
  {
      usb_cpu_usb_interrupt();
      if ( CDC_BLK_OUT == cdc_read())        ← Receive check
      {
              cdc_main();
      }                                        ← Execute firmware update processing
  }                                              in cdc_main() functoin
}
```

Figure 7-3   main() Function

7.4.6   Jump to user application

The processing to jump to the user program is performed in the jump_to_userapp( ) function. Refer to section 5.2 **UserApp Header Area (user application header)** for details on specifying the start address of the user program jump destination.

7.4.7   Firmware update main processing

USB Peripheral Firmware Updater processing is performed by cdc_main() function. This function is called from the main() function.

```
                    ╭─────────────────╮
                    │    cdc_main()   │
                    ╰─────────────────╯
                             │
               Processing branches
               according to receive record
               type
                             │
                          ╱ Start record ╲ ──────No──────┐
                          ╲              ╱                │
                            Yes                           ▼
                             │                      ╱ Data record ╲ ────No────┐
            Call Flash API   │                      ╲            ╱             │
          ┌ ─ ─ ─ ─ ─ ─ ─ ─ ┐                        Yes                      ▼
          │ ┌─────────────┐ │                          │                ╱ End record ╲
          │ │Flash P/E mode│ │                          │                ╲          ╱
          │ └─────────────┘ │                          │                   Yes
          │        │        │              ╱ Confirm number of receive ╲    │
          │ ┌─────────────┐ │              ╲    bytes                  ╱    ▼
          │ │  ROM Erase  │ │         Receive confirmed          ┌──────────────┐
          │ └─────────────┘ │         number of write bytes      │Software reset│
          │        │        │                   │                └──────────────┘
          │ ┌─────────────┐ │         ┌ ─ ─ ─ ─ ─ ─ ─ ─ ┐
          │ │Flash Read mode│ │         │ ┌─────────────┐ │        *If the MCU does not
          │ └─────────────┘ │         │ │Flash P/E mode│ │        support software reset,
          └ ─ ─ ─ ─ ─ ─ ─ ─ ┘         │ └─────────────┘ │        the processing ends
                   │                   │        │        │        without a reset.
                   │                   │ ┌─────────────┐ │        In such a case, reset
                   │                   │ │  ROM Write  │ │        the MCU manually or
                   │                   │ └─────────────┘ │        restart board.
                   │                   │        │        │
                   │                   │ ┌─────────────┐ │
                   │                   │ │Flash Read mode│ │
                   │                   │ └─────────────┘ │
                   │                   └ ─ ─ ─ ─ ─ ─ ─ ─ ┘
                   │                            │
                   │ ◄──────────────────────────┘
             ╭─────────────╮
             │     End     │
             ╰─────────────╯
```

Figure 7-4   Flash Write Operation Flow

ROM erase is performed in block units; write size varies according to ROM specification.

8. File Transfer Application (USB Firmware Updater) Explanation

This section explains how the file transfer application performs on the host PC.

8.1 Development Environment

The file transfer application is configured with the following environment:

OS: Windows 7, Windows 8.1, Windows 10

Development language: Microsoft Visual C++ 6.0 (MFC)

8.2 Operation Overview

The file transfer application transitions to the direct re-write processing when it receives the name (or option) of a target re-write file name as an argument at startup. If a file has not been specified, the setting dialog is displayed.
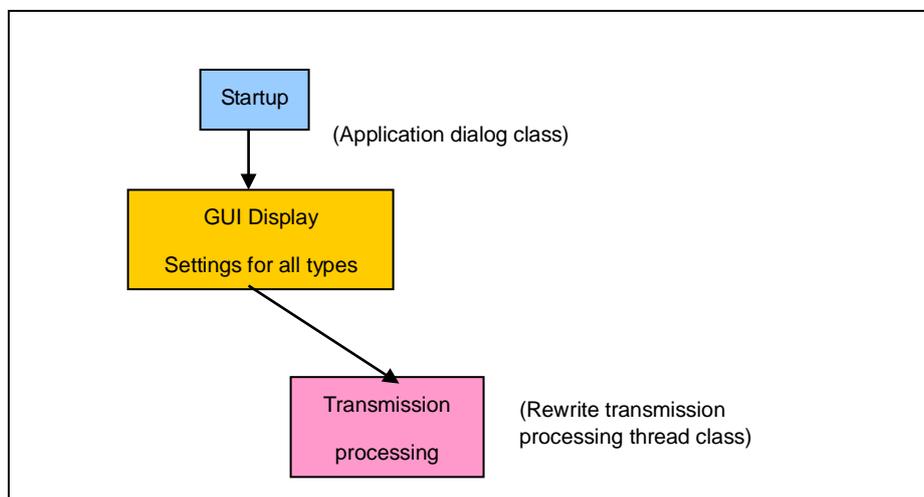
Figure 8-1   File Transfer Application Operation Overview

8.3   File Configuration

The following lists the file transfer application files (only key files are listed).

Table 8-1   File Transfer Application Files

File Name	Description
FlashSelfRewriteGUI.dsw	Workspace file
FlashSelfRewriteGUI.dsp	Project file
FlashSelfRewriteGUI.clw	Class wizard file
FlashSelfRewriteGUI.rc	Resource file
FlashSelfRewriteGUI.cpp	Application class processing file
FlashSelfRewriteGUI.h	Application class definition file
FlashSelfRewriteGUIDlg.cpp	Application dialog class processing file
FlashSelfRewriteGUIDlg.h	Application dialog class definition file
CommandThread.cpp	Rewrite transmission processing thread class processing file
CommandThread.h	Rewrite transmission processing thread class definition file
CommonProc.cpp	Common processing class processing file
CommonProc.h	Common processing class definition file
SerialPort.cpp	Serial COM port transmission class processing file
SerialPort.h	Serial COM port transmission class definition file
Resource.h	Resource header file
UsbfUpdater.ini	Application operation setting file

8.3.1   Application Class (FlashSelfRewriteGUI)

This processing checks the arguments (options) at the initial startup, then calls the dialog class.

The following lists the application startup options.

Table 8-2   Application Startup Options

Option	Description
/S nnnnnn	Specify the write start address as a hexadecimal number
/C nn	Specify the connection COM port number
Filename	Target rewrite file path

8.3.2   **Application Dialog Class (FlashSelfRewriteGUIDlg)**

This processing displays the rewrite specification dialog screen (refer to section 3.2 **Project Setup (CS+)**

Please double-click the mtpj file.

Execute USB Peripheral Firmware Updater for details). This screen allows the user to specify operation mode, rewrite address, rewrite file, and connection COM port. In addition, if these items are already specified when the screen is displayed, the function reads the application operation setting file and reflects the settings as default values.

Click the Update button to call the rewrite transmission processing thread class.

Added member variables are shown below.

Table 8-3   Application Dialog Class Member Variables

Member Variable		Description
Type	Name	
Int	m_nCOM	Number of COM port to be connected
TCHAR	m_tcAppDir[_MAX_PATH]	Application execution directory
CString	m_strCurTargetSeries	Current target series
CString	m_strCurTarget	Current target name
CString	m_strCurDevice	Current device
CStringArray	m_arDeviceSeries	Device series list
CStringArray	m_arDeviceVal	Device list
CStringArray	m_arDeviceText	Device name list
Int	m_nDevSize	Current device ROM size
CWinThrread*	m_pCommandThread	Thread class pointer
BOOL	m_bExistThread	Thread operation status
BOOL	m_bStartUp	Display initial startup
DWORD	m_dwROMStartAddress	ROM area start address
DWORD	m_dwROMEndAddress	ROM area end address
DWORD	m_dwEnROMStartAddress	ROM P/E access enabled start address
DWORD	m_dwEnROMEndAddress	ROM P/E access enabled end address
COleDateTime	m_dtStart	Rewrite processing start date and time
COleDateTime	m_dtEnd	Rewrite processing end date and time

Member functions are described below.

Table 8-4   Read_DeviceInfo Function

Function name		Read_DeviceInfo
Description		bool Read_DeviceInfo (void)
Function		Get information from application operation setting file
I/O	Input	None
	Output	TRUE(成功)／FALSE(失敗)

Table 8-5   Write_DeviceInfo Function

Function Name	Write_DeviceInfo
Description	bool Write_DeviceInfo (void)

Format		
Function		Update application operation setting file
I/O	**Input**	None
	Output	TRUE(success)/FALSE(fail)

Table 8-6   Update_Message Function

Function Name		Update_Message
Description Format		void Update_Message (LPCTSTR)
Function		Display message in message display column
I/O	**Input**	Message character string pointer
	Output	None

Table 8-7   Initialize_Device Function

Function Name		Initialize_Device
Description Format		void Initialize_Device(void)
Function		Initialization processing
I/O	**Input**	None
	Output	None

Table 8-8 DeviceListRefresh Function

Function Name		DeviceListRefresh
Description Format		void DeviceListRefresh (void)
Function		Create Device list
I/O	**Input**	None
	Output	None

Table 8-9 DeviceInfoRefresh Function

Function Name		DeviceInfoRefresh
Description Format		void DeviceInfoRefresh (void)
Function		Update device combo box
I/O	**Input**	None
	Output	None

Table 8-10   AppStatus Function

Function Name		AppStatus
Description Format		void AppStatus(bool stu)
Function		Set status at rewrite operation
I/O	Input	stu: TRUE (enable screen controls)
		FALSE (disable screen controls)
	Output	None

8.3.3   Rewrite Transmission Processing Thread Class (CommandThread)

This processing uses the serial COM port transmission class to send and receive the specified file based on the interface specifications when connected to the target evaluation board. If the file is a HEX file, analysis is also performed.

Added member variables are shown below (files listed under application dialog class are not repeated here).

Table 8-11   Rewrite Transmission Processing Thread Class Member Variables

Member Variable		Description
Type	Member Name	
CDialog*	m_pAppDlg	Dialog class of call origin pointer
CString	m_strAppDir	Directory in application
BOOL*	m_pbExistThread	Thread operation status pointer
CSerialPort	m_Serial	Serial COM port transmission class
int	m_nCOM	Connection COM port number
CString	m_strFileName	Target file path
EnMode	m_enMode	Rewrite mode
DWORD	m_dwStartAddress	Rewrite start address
DWORD	m_dwROMStartAddress	ROM start address
DWORD	m_dwROMEndAddress	ROM end address

Added member functions are listed below.

Table 8-12   Cal_CheckSum Function

Function Name	Cal_CheckSum	
Description Format	BYTE Cal_CheckSum(LPBYTE bytes, LONG size)	
Function	Calculate check sum	
I/O	Input	Bytes: data string pointer Size: data string length
	Output	Calculated check sum value

Table 8-13   Change_strHex2Bibary Function

Function Name	Change_strHex2Binary	
Description Format	VOID Change_strHex2Binary (LPCSTR strHex, LPBYTE pbytes, LONG size)	
Function	Convert string displayed in hex to binary data string	
I/O	Input	strHex: pointer to character string displayed in hexidecimal notation pbyte: data string start pointer size: number of conversion data
	Output	None

Table 8-14   Upsets_DWORD Function

Function Name	Upsets_DWORD	
Description Format	DWORD Upsets_DWORD(DWORD dwVal)	
Function	Invert DWORD type values by byte (ex.) 0xaabbccdd -> 0xddccbbaa	
I/O	Input	dwVal: value of DWROD to be inverted
	Output	Inverted value

Table 8-15   SET_StartRecord Function

Function Name	SET_StartRecord	
Description Format	VOID SET_StartRecord (LPVOID lpRecord)	
Function	Creates rewrite start record	
I/O	Input	lpRecord: record storage pointer
	Output	None

Table 8-16   SET_EndRecord Function

Function Name		SET_EndRecord
Description Format		VOID SET_EndRecord (LPVOID lpRecord)
Function		Creates rewrite end record
I/O	Input	lpRecord: record storage pointer
	Output	None

8.3.4   **Common Processing Class (CommonProc)**

Processes that are shared in the File Transfer Application are defined in this section. Added member functions are described below.

Table 8-17   GetAppDir Function

Function Name		GetAppDir
Description Format		static VOID GetAppDir(LPTSTR path, int sw = 0)
Function		Get the application execution address
I/O	Input	Path: target character string pointer sw: 0 Get path as is 1 Get shortened path
	Output	None

Table 8-18   Change_Hex2Val Function

Function Name		Change_Hex2Val
Description Format		static DWORD Change_Hex2Val(LPCSTR pHex)
Function		Convert character string displayed in 1 byte (2 hex digits) to a numerical value
I/O	Input	pHex: pointer for character string displayed in 2 hex digits
	Output	Converted value

Table 8-19   IsNumeric Function

Function Name		IsNumeric
Description Format		static BOOL IsNumeric(LPCTSTR lpNum, LONG size, int type)
Function		Numerical value check processing
I/O	Input	lpNum: pointer of character string expressed in numerical value size: number of digits of checked value type : 10 Check as a decimal number 16 Check as a hex number
	Output	TRUE (indicates a numerical value) /FALSE (indicates a non-numerical value)

Table 8-20   IsExistFile Function

Function Name		IsExistFile
Description Format		static BOOL IsExistFile(LPCTSTR lpszFileName, BOOL bDirectory = FALSE)
Function		Check for existing file
I/O	Input	lpszFileName: file path to be confirmed bDirectory: FALSE (check file) TRUE (check directory)
	Output	TRUE (file exists) / FALSE (no file)

8.3.5   **Serial COM Port Transmission Class (SerialPort)**

This class is used for serial transmission via the COM port.

Added member variables are list below.

Table 8-21   Serial COM Port Transmission Class Member Variables

Member Variable		Description
Type	**Member Name**	
HANDLE	m_hCom	Handle that is received when connection is made
DCB	m_Dcb	Device control block structure
COMMTIMEOUTS	m_TimeoutSts	Time out setting structure
INT	m_nCOM	Number of port to be connected

Member functions are described below.

Table 8-22   Port_Open Function

Function Name		Port_Open
Description Format		LONG Port_Open(INT com)
Function		Connect to specified COM port.
I/O	**Input**	Com: COM port number
	Output	0 Successful connection −1 Failed connection

Table 8-23   Port_Close Function

Function Name		Port_Close
Description Format		VOID Port_Close(VOID)
Function		Disconnect the connected port.
I/O	**Input**	None
	Output	None

Table 8-24   Port_Write Function

Function Name		Port_Write
Description Format		LONG Port_Write(LPCVOID buf, LONG cnt)
Function		Transmit data in serial transmission
I/O	Input	Buf: transmit data string pointer Cnt: transmit data length (bytes)
	Output	Number of transmitted bytes, "-1" indicates transmit failure.

Table 8-25   Port_Read Function

Function Name		Port_Read
Description Format		LONG Port_Read(LPVOID buf, LONG cnt)
Function		Receive data in serial transmission.
I/O	Input	Buf: pointer of data string that stores receive data cnt: receive data length (bytes)
	Output	Number of received bytes. "-1" indicated receive failure.

Table 8-26   Get_PortNumber Function

Function Name		Get_PortNumber
Description Format		INT Get_PortNumber(VOID)
Function		Get number of connected port.
I/O	Input	None
	Output	Number of currently connected port

Table 8-27   AutoScanCom Function

Function Name		AutoScanCom
Description Format		INT AutoScanCom (LPCTSTR pszService, LPCTSTR pszInterface, INT nNo = 0)
Function		Detect connectable COM ports.
I/O	Input	pszService: Name of service run by COM port pszInterface: interface name nNo: search beyond this number
	Output	Detected COM port number. If not found, return 0.

8.3.6   **Application Operation Setting File (UsbfUpdater.ini)**

The application operation setting file is ini file format and retains setting values and device information. Please keep this file in the folder that stores the exe file. Note that the application will not run normally without the ini file.

Definitions for the ini file are provided below.

Table 8-28   Application Operation Setting File Description (sections)

Section	Description
Application	Display values currently set in the application. This is information to be written by the application.
SS_xxx	Retain previously displayed device information. This is information to be written by the application.
Device. XXXXXXXX	Display device information (multiple settings possible), This is information that can be added by user.

Table 8-29   Application Operation Setting File Contents

Section	Key	Value	Description
Application	Series	XXX	Series of specified target
	COM	1 to 20	The number of the COM port that is currently or will be connected. Note: Can be set but not used in OS versions later than Windows 10.
	EnableStartAddress	FFFFFFFF	Write enabled start address
	EnableEndAddress	FFFFFFFF	Write enabled end address
SS_XXX	Device	XXX	Device specified by target
Device. XXX	TargetSeries	XXX	Series of this device
	Name	XXX	Name of this device
	Size	1 to 999	ROM size (Kbytes) of this device
	StartAddress	FFFFFFFF	ROM start address for this device

Items other than the device information are stored as display information and will be updated automatically when the GUI software is closed.

```
[Application]
Series=RL78L1C
COM=6
EnableStartAddress=00000000
EnableEndAddress=0007FFFF
  [Device.R5F110PJ]
TargetSeries=RL78L1C
Name=R5F110PJ
Size=512
StartAddress=00000000

..

[SS_RL78L1C]
Device=R5F110PJ
```

[Application Information]

Displayed target series: RL78/L1C(Retained information from previous display)

Connected COM port: 6 (Retained information from previous connection)

Write enabled address: Start address to End address

[Device Information]

Device target series: RL78/L1C

Device name: R5F110PJ

Size: 512KK bytes

ROM Address   0x00000000 to 0x0007FFF

(Input address using 8 hex numbers)

The remaining is individual device information

Information for display. Retained information from previously selected device.

Figure 8-2   Application Operation Setting ini File

8.4   Application Messages

The following lists the messages displayed by the application in the message column and the timing in which they are displayed.

Table 8-30   Application Messages

Message	Display Timing
Start upload file.	At start of rewrite processing
Success.	At end of rewrite processing
Please input file.	At rewrite processing when specified file is not specified. Also when specified file is not found.
Please set the address correctly.	When address is not specified correctly
Please set COM port.	When COM port is not specified correctly
ERR: file open error.	Failure in opening file
ERR: file format error.	When a file in other than Motorola S file format or Intel extended file format is specified
ERR: Unable to connect to the COM port n.	Failed connection to COM port n
ERR: Data transmission error.	Failed data transmission
ERR: Data reception error.	Failed data reception (failed for 3 retries)
ERR: Writing process stop.	Received NAK (error) in response record from board side
ERR: Write Enable Area Address is ROM area over, or illegal value.	When specified P/E Access Enable Area exceeds ROM area or is an illegal value (only when Use P/E Access Enable is selected).
ERR: Address is ROM area over. Process stop.	When write address exceeds ROM area
ERR: file size error.	When file size check shows data size exceeds ROM area
ERR: Get ROM Address Error. <Device: xxxx >	When the ini file ROM information is incorrect
ERR: Get ROM Address Error. Update process stop.	When a write operation is executed and the ROM information read in the ini file is incorrect

9. Data Transmission Specification

9.1   Rewrite Transmission Interface Specification

This section describes transmission between the PC which the file transfer application works on and the evaluation board.

9.1.1   Transmission data configuration

The PC transmits the start record and end record. The write data is sent to the Flash memory in data record format.
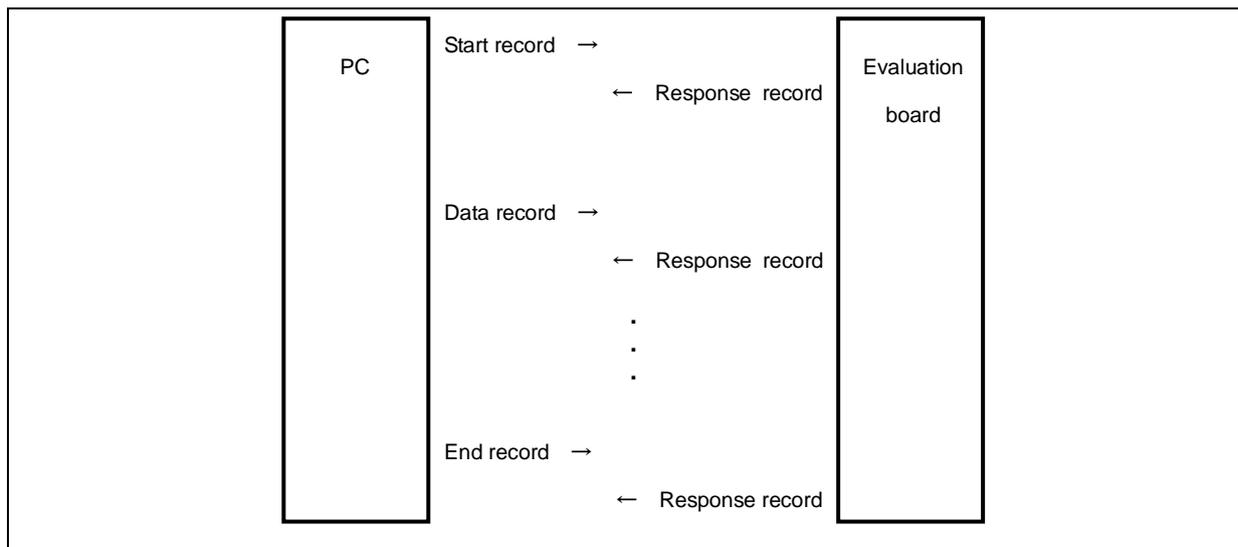
Figure 9-1   Transmission Data Sequence

9.1.2   **PC-side transmission data**

The PC side sends the start record, data record, and end record.

Each record is transmitted one at a time and the next record is not sent until a response for the previously sent record is received.

(1).   Start record

The start record is the first record to be transmitted when executing a rewrite: 14 bytes.
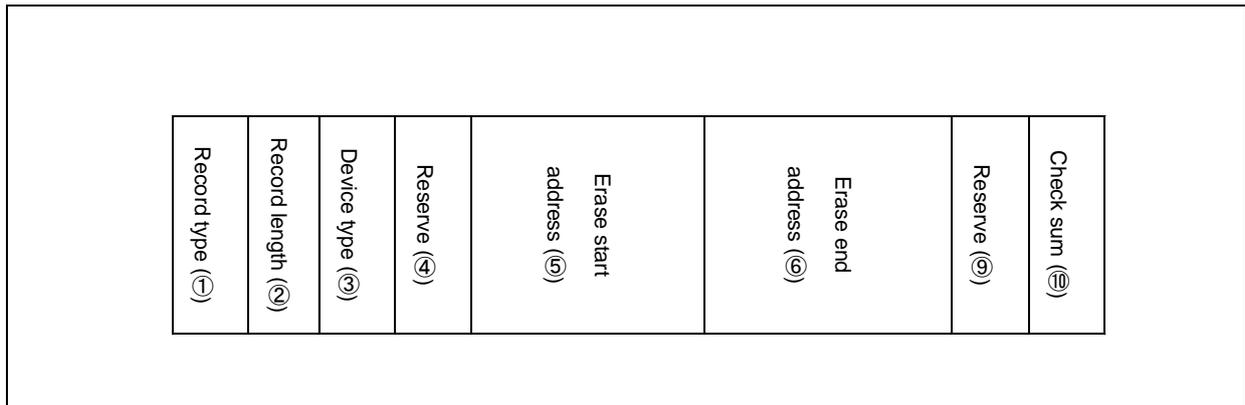
Figure 9-2   Start Record Format

① Record type: 1 byte

   Record type

   The start record record type is 0x00.

② Record length: 1 byte

   Number of bytes after the device type

③ Device type: 1 byte

   Device type (currently unused, therefore fixed as 0x00)

④ Reserve: 1 byte

   0x00 fixed

⑤ Erase start address: 4 bytes

   ROM erase start address setting. The address is a 32-bit numerical value in Little Endian format.

⑥ Erase end address: 4 bytes

   ROM end address specification. The address is a 32-bit numerical value in Little Endian format.

⑦ Reserve:1 byte

   0x00 fixed

⑧ Check sum: 1 byte

   Record check sum.

   Check sum of the record length, device type, and date and time.

   The lower 8 bits of the complement 1 of the sum of all the bytes received.

(2).   **Data record**

Write data record: (7+number of data) bytes (MAX 64 bytes)
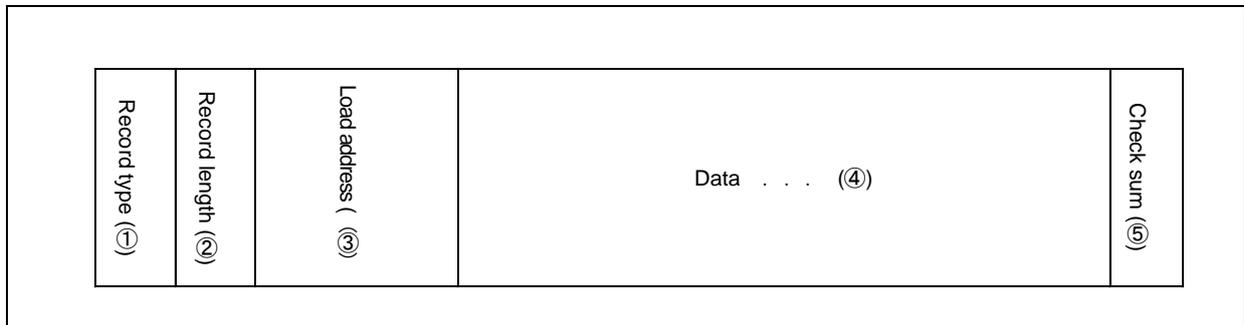
Figure 9-3   Data Record Format

①   Record type: 1 byte

    Record type

    The data record record type is 0x0f.

①   Record length: 1 byte

    Number of bytes after the load address.

②   Load address: 4 bytes

    Flash memory address

    Data is written from this address.

    The load address is a 32-bit numerical value in Little Endian format.

③   Data: 1 to 57 bytes

    Data to be written to the Flash memory

    1 record is a maximum of 57 bytes.

④   Check sum: 1 byte

    Record check sum.

    Check sum of the record length and address data.

    The lower 8 bits of the complement 1 of the sum of all the bytes received.

(3).   **End record**

The end record is sent after all data is transmitted: 4 bytes.

Figure 9-4   End Record Format

① Record type: 1 byte

Record type

The end record record type is 0xf0.

⑨ Record length: 1 byte

Number of bytes after the device type

⑩ Device type: 1 byte

Device type (currently unused, therefore fixed as 0x00)

⑪ Check sum: 1 byte

Record check sum.

Check sum of the record length and device type.

The lower 8 bits of the complement 1 of the sum of all the bytes received.

9.1.3   Evaluation board-side transmission data

The evaluation board sends a record in response to the record received from the PC: 5 to 8 bytes

(1).   Response record
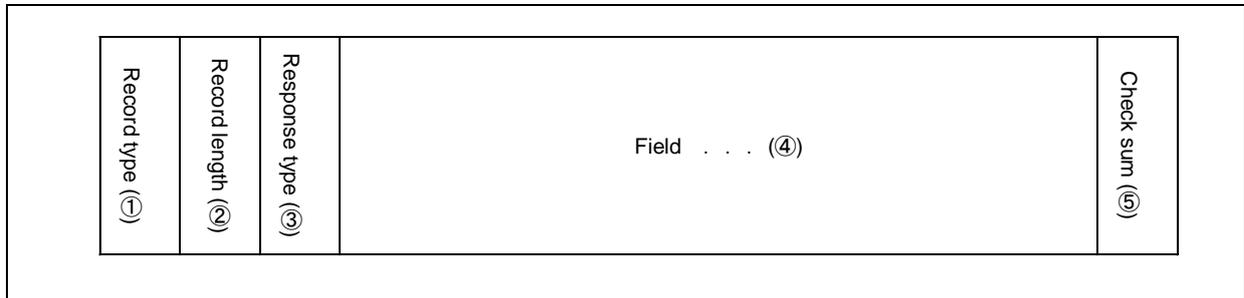
Figure 9-5   Response Record Format

①   Record type: 1 byte

      Record type

      Type of record to which a response is being sent.

      The response record type is 0xFF

②   Record length: 1 byte

      Number of bytes after the response type

③   Response type: 1 byte

      Response type

      One of the following 3 types

           －0x00   : ACK

           －0x0f   : NAK (re-transmit/receive request)

           －0xf0   : NAK (error end)

④   Field: 1 to 4 bytes

      If error, error code is 1 byte

      If not an error, content depends on record type

           －Start record: device type, write/erase access restriction address

           －Data record: load address

           －End record: device type

⑫   Check sum: 1 byte

      Record check sum.

      Check sum of the record length, response type, and field.

      The lower 8 bits of the complement 1 of the sum of all the bytes received.

10. Using the e² studio project with CS+

   This package contains a project only for e² studio. When you use this project with CS+, import the project to CS+ by following procedures.

[Note]

   1.   Uncheck the checkbox *Backup the project composition files after conversion* in *Project Convert Settings* window.
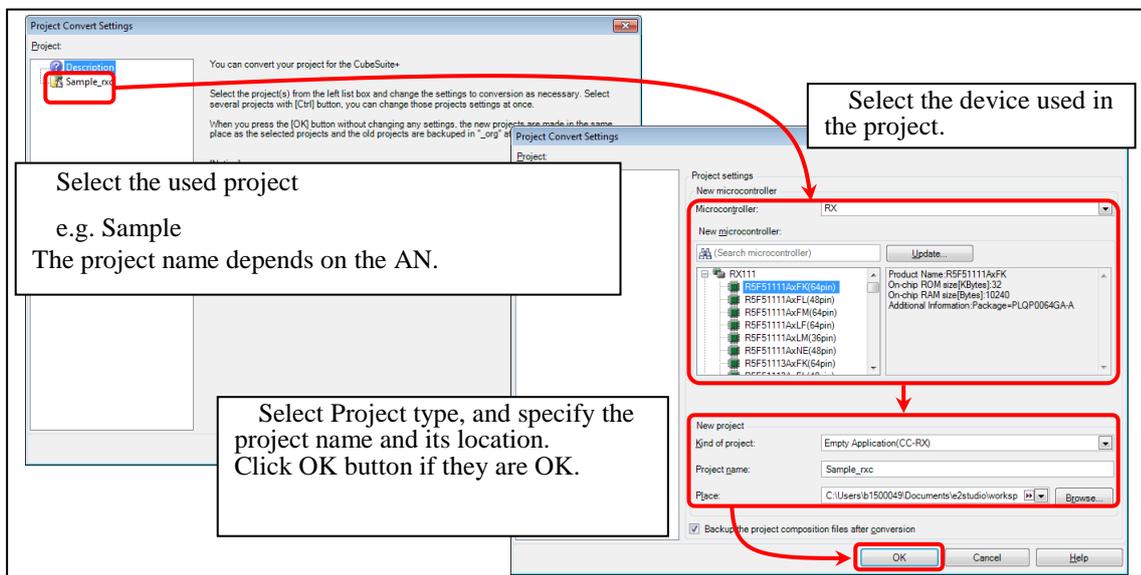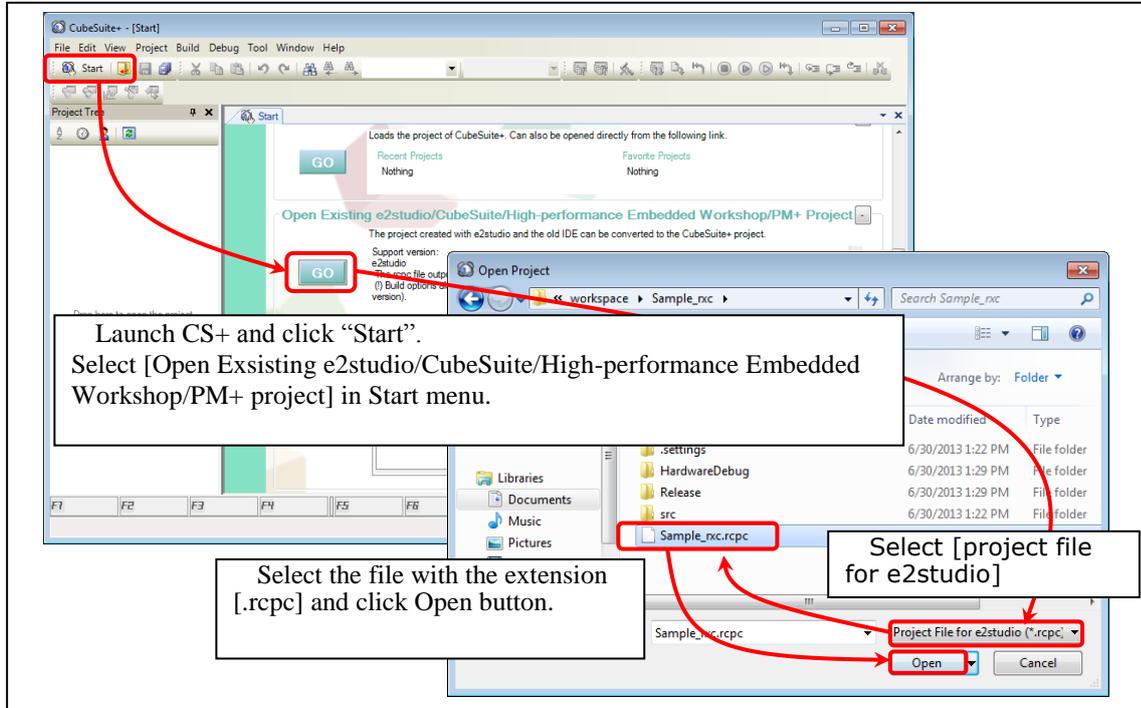
Figure 10-1   Using the e² studio project with CS+

Website and Support

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

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

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

Revision Record

Rev.	Date	Page	Summary
			Description
1.00	Jun 30, 2016	-	First edition issued.
1.10	Sep 30, 2016	-	USB driver part is changed.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins

    Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

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

2. Processing at Power-on

    The state of the product is undefined at the moment when power is supplied.

    — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

    In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

    In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

    Access to reserved addresses is prohibited.

    — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

    After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

    — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

    Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.

    — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

RENESAS

Renesas Electronics Corporation

http://www.renesas.com

SALES OFFICES

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

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141