RL78/G22,RL78/G23,RL78/G24

Firmware Update Module

Introduction

This application note describes the firmware update module for the RL78/G22 and RL78/G23,RL78/G24. The module is referred to below as the firmware update module.

By using the module, users can easily incorporate firmware update functionality into their applications. This application note explains the specifications of the firmware update module and how to incorporate its API functions into user applications.

The release package associated with this application note includes a demo project. You can confirm the basic operation of the firmware update functionality by following the steps described in section 5, Demo Project, to build an environment to run the demo.

Operation Confirmation Devices

RL78/G22 (R7F102GGE)
RL78/G23 (R7F100GSN)
RL78/G24 (R7F101GLG)

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Application Notes

Application notes related to this application note are listed below. Refer to them in conjunction with this application note.

- RL78 Family Board Support Package Module Using Software Integration System (R01AN5522)
- RL78 Family Renesas Flash Driver RL78 Type01 User's Manual (R20UT4830)
- RL78 Smart Configurator User's Guide: e^2 studio (R20AN0579)
- Smart Configurator User's Guide: RL78 API Reference (R20UT4852)

Target Compliers

- CC-RL V1.14.00 from Renesas Electronics
- IAR C/C++ Compiler for Renesas RL78 version 5.10.1 from IAR Systems
- IAR Assembler for Renesas RL78 version 5.10.1 from IAR Systems
- LLVM for Renesas RL78 17.0.1.202412

For details of the environments on which operation has been confirmed, refer to 6.1, Confirmed Operation Environments.

Contents

1. Overview

1.1 About the Firmware Update Module

A firmware update is a process in which a device overwrites its own firmware, the software that controls the device's hardware, with a new version of the firmware (called the "update image" in this document) obtained through unspecified means. Firmware updates may be applied to fix bugs, add new functions, or improve performance.

The firmware update module is middleware that, when firmware update functionality is added to the user's system, provides the following functionality as its components:

- Functionality for importing the update image to the MCU via a communication interface
- Functionality for validating the update image (ECDSA NIST P-256 and SHA256 are used for validation.)
- Functionality for programming the update image to the on-chip flash memory (self-programming)
- Functionality for activating the update image

Generally, a firmware update system comprises two programs: an application program providing firmware update functionality and a bootloader providing secure boot functionality used to validate the first program.

The bootloader functionality is essential to the proper functioning of the firmware update. It guarantees that the sequence of processing that composes the firmware update, including validation of the update image, is legitimate.

The firmware update module for the RL78 Family provides functionality for the following three firmware update methods.

- Partial update method (buffer plane is internal flash)
- Full update method (without buffer plane)
- Full update method (buffer plane is external flash)

A tool (Renesas Image Generator) for creating firmware images is provided as a utility. Renesas Image Generator can generate the following types of images for use by the firmware update module.

- Initial image: An image file containing the bootloader and application program that is programmed using Flash Writer at the time of initial system configuration (extension: mot).
- Update image: An image file containing the firmware update (extension: rsu).

1.2 Configuration of Firmware Update Module

Figure 1.1 shows the configuration of the modules in the bootloader and application program incorporating the firmware update module, and Table 1.1 lists the modules used in the bootloader and application program.

The update image received by the communication interface is self-programmed to the on-chip flash memory of the target device via the firmware update module and the flash memory driver.

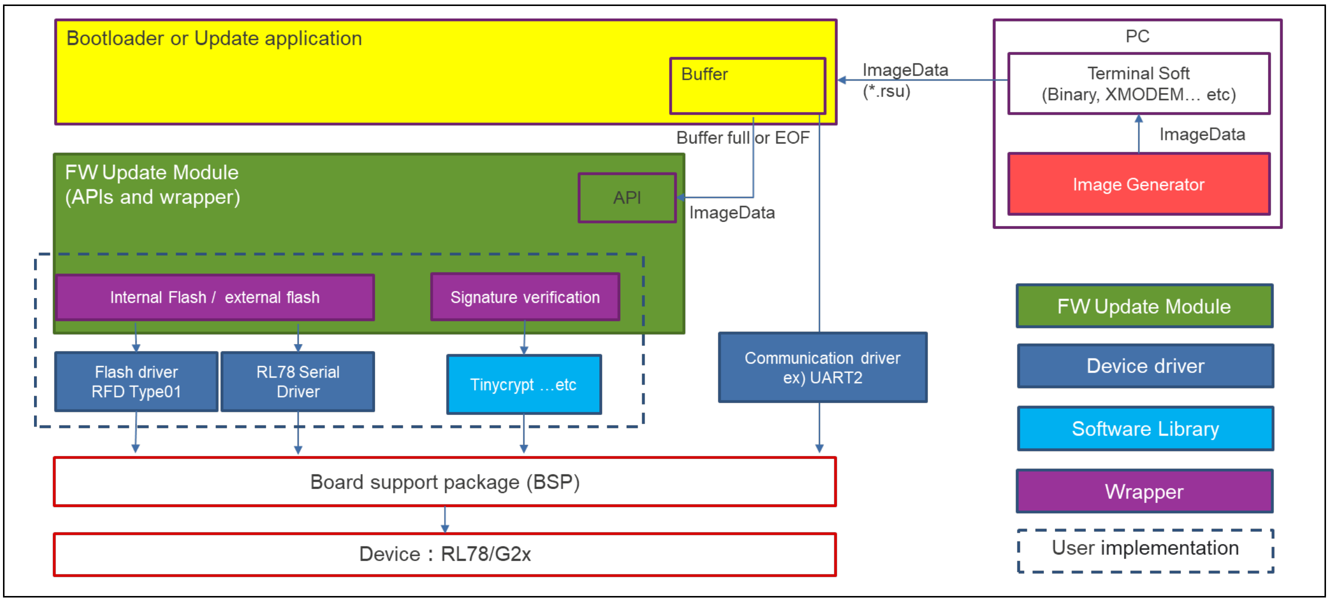


Figure 1.1 Configuration of Modules in Sample Bootloader and Application Program

Table 1.1 List of External Modules Used in Sample Bootloader and Application Program

Function	Module Name	Notes
BSP	r_bsp	Automatic generation by smart configurator
UART	r_Config_UART1 : RL78/G22 r_Config_UART2 : RL78/G23	Automatic generation by smart configurator
PORT	r_Config_PORT	Automatic generation by smart configurator
FLASH	RFD RL78 Type01	Automatic generation by smart configurator
CSI	RL78_Serial	Implemented in wrapper
Serial Flash	r_qspi_flash_mx25l	Implemented in wrapper
Crypt Library	Tinucrypt	Implemented in wrapper

1.3 Firmware Update Operation

The RL78 family firmware update module provides two methods: once storing the firmware to be updated (update image) on the buffer plane and once writing it directly to the main plane. The buffer plane can be set in the internal flash memory or external flash memory.

- Main plane: Area for storing the image used for booting
- Buffer plane: Area for storing the image to be applied as an update

The method of writing the update image directly to the main plane allows all of the internal flash memory to be used as the main plane, but since there is no buffer plane, it is not possible to restore the firmware to its pre-update state in the event of an update failure.

1.3.1 Partial Update Method (buffer plane is internal flash)

The update image is stored temporarily in the buffer plane in the on-chip flash memory, and, after it is validated, it is self-programmed to the main plane. This method allows the application program to contain the firmware update functionality. This means that if the firmware update fails before self-programming to the main plane occurs, the pre-update image in the main plane can be launched to retry the firmware update. The size that can store the application program is half the size of the remaining internal flash memory minus the bootloader.

1.3.1.1 Operation of Partial Update Method (buffer plane is internal flash)

This method divides the on-chip flash memory into a main plane and a buffer plane and then temporarily stores the update image in the buffer plane. Firmware is updated by storing the update image on the buffer plane and copying it from the buffer plane to the main plane.

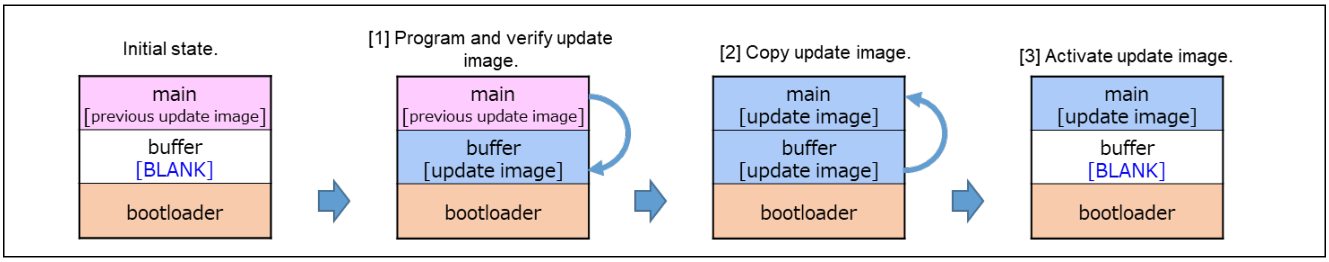


Figure 1.2 Operation of Partial Update Method (buffer plane is internal flash)

[1] Program and verify update image.
The previous update image (application program) stored in the main plane is used to program the update image to the buffer plane and verify it.

[2] Copy update image.
If verification is successful, the system is reset, the main plane is erased by the bootloader, and the updated image is copied from the buffer plane to the main plane.

[3] Activate update image.
The buffer plane is erased by the bootloader.
(The demo program does not erase the buffer plane. If you need to erase the image before updating for rollback measures, please add a process to erase the buffer plane image.)

1.3.2 Full Update Method (without buffer plane)

The update image is self-programmed to the main plane, after which it is validated. This method requires the bootloader to contain the firmware update functionality. This means that if the firmware update fails, the bootloader functionality can be used to retry the firmware update. The functionality of the application program cannot be used until the firmware update succeeds.
The size that can store the application program is the remaining size of the internal flash memory minus the bootloader.

1.3.2.1 Operation of Full Update Method (without buffer plane)

This method of writing the update image directly to the main plane allows all of the internal flash memory to be used as the main plane, but since there is no buffer plane, it is not possible to restore the firmware to its pre-update state in the event of an update failure.

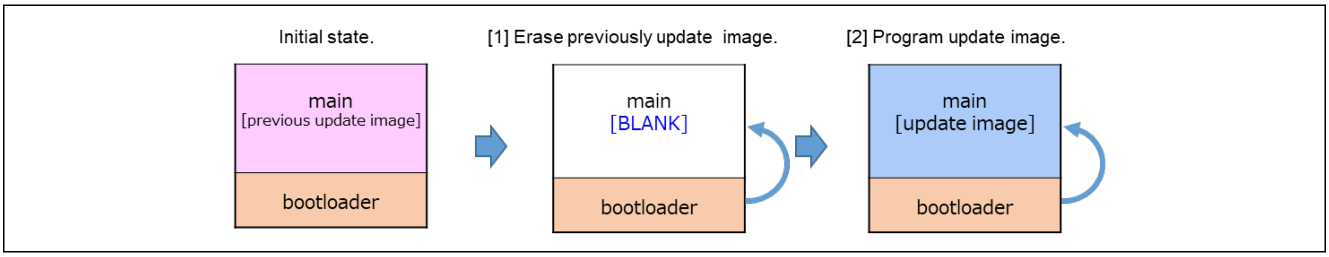


Figure 1.3 Operation of Full Update Method (without buffer plane)

[1] Erase previously update image.
The previous update image (application program) stored in the main plane configures the data indicating updates to the main plane and then applies a reset. After this, the bootloader runs and erases the initial image from the main plane.

[2] Program update image.
The bootloader downloads the update image from an external source and programs it to the main plane. The programmed update image is verified, and if verification is successful, the update image is activated.

1.3.3 Full Update Method (buffer plane is external flash)

The update image is stored temporarily in the buffer plane in the on-chip flash memory, and, after it is validated, it is self-programmed to the main plane. This method allows the application program to contain the firmware update functionality. This means that if the firmware update fails before self-programming to the main plane occurs, the pre-update image in the main plane can be launched to retry the firmware update. The size that can store the application program is the size remaining after subtracting the bootloader from the internal flash memory, since only the main plane is provided in the internal flash memory.

1.3.3.1 Operation of Full Update Method (buffer plane is external flash)

The update image is stored once in the buffer plane, with the main plane set in the internal flash and the buffer plane set in the external flash.

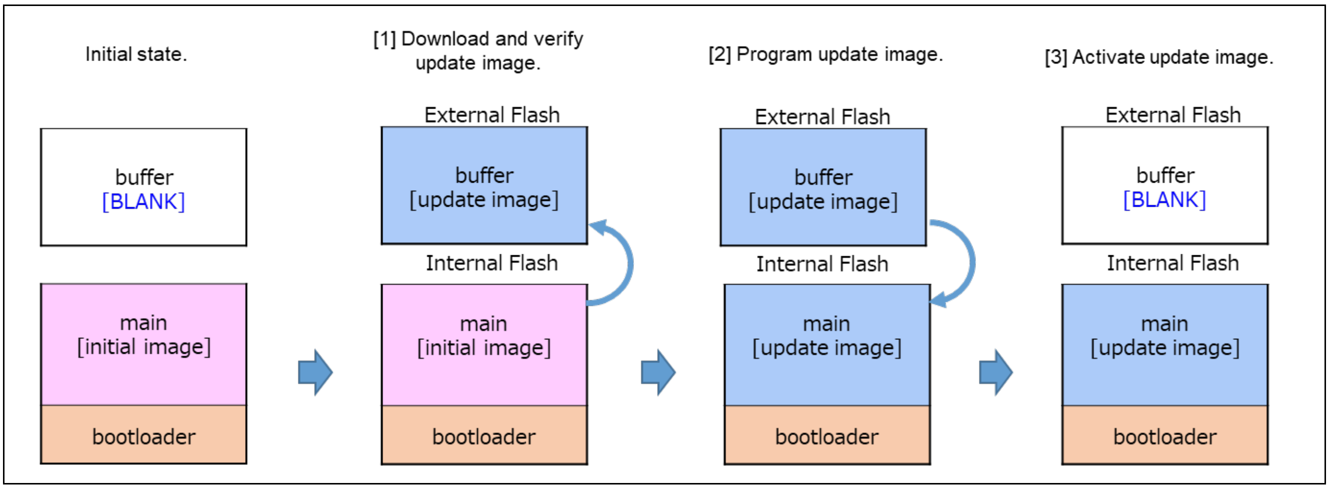


Figure 1.4 Operation of Full Update Method (buffer plane is external flash)

[1] Program and verify update image.
The previous update image (application program) stored in the main plane is used to program the update image to the buffer plane and verify it.

[2] Copy update image.
If verification is successful, the system is reset, the main plane is erased by the bootloader, and the updated image is copied from the buffer plane to the main plane.

[3] Activate update image.
The buffer plane is erased by the bootloader.
(The demo program does not erase the buffer plane. If you need to erase the image before updating for rollback measures, please add a process to erase the buffer plane image.)

1.4 Initial State of Firmware Update

To set the firmware update system using the firmware update module to the initial state, build the system by writing the initial image generated by the Renesas Image Generator to the built-in flash memory with a flash writer or similar device.

As an alternative method, it is also possible to build the system by first writing only the bootloader with a flash writer, etc., and then writing the updated image of the application program with the bootloader function.

1.4.1 Initial State of Partial Update Method Settings Utilizing Renesas Image Generator

The following figure shows the construction of the initial state of the partial update method using the Renesas Image Generator.

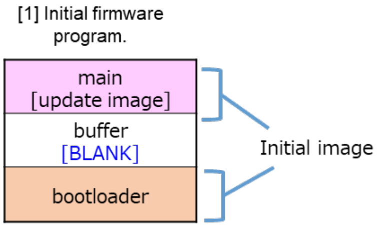


Figure 1.5 Initial Firmware Update Settings Utilizing Renesas Image Generator (Example of Partial Update Method)

[1] Program the initial image

The initial image is programmed to the on-chip flash memory using a tool such as Flash Writer.

1.4.2 Initial State of Full Update Method Settings Utilizing Renesas Image Generator

The following figure shows the construction of the initial state of the full update method using the Renesas Image Generator.

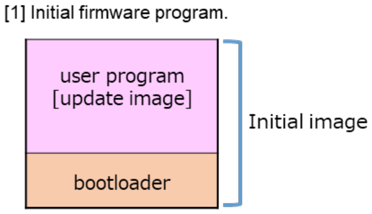


Figure 1.6 Initial Firmware Update Settings Utilizing Renesas Image Generator (Example of Full Update Method)

[1] Program the initial image

The initial image is programmed to the on-chip flash memory using a tool such as Flash Writer.

1.4.3 Initial State of Partial Update Method Settings Utilizing Bootloader

The following figure shows the construction of the initial state of the dual-bank method using the bootloader.

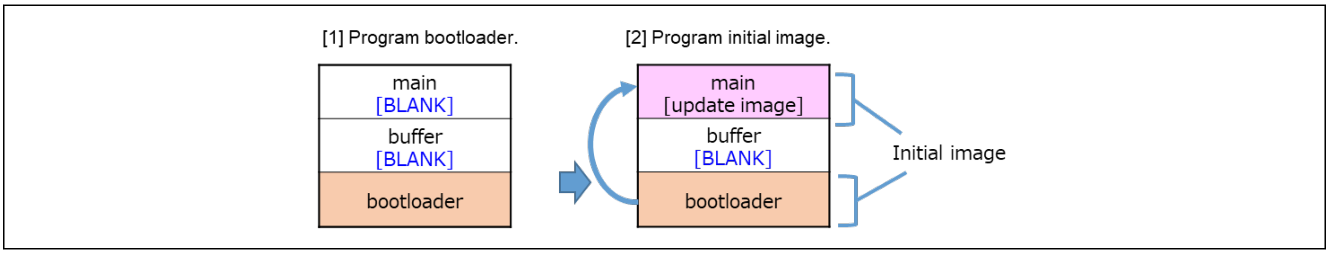


Figure 1.7 Initial Firmware Update Settings Utilizing Bootloader (Example of Partial Update Method)

[1] Program bootloader.
The bootloader is programmed to the on-chip flash memory using a tool such as Flash Writer.

[2] Program initial image.
The initial image is downloaded from an external source and programmed to the main plane using the functionality of the bootloader. The programmed firmware is verified, and if verification is successful, processing ends.

1.4.4 Initial State of Full Update Method Settings Utilizing Bootloader

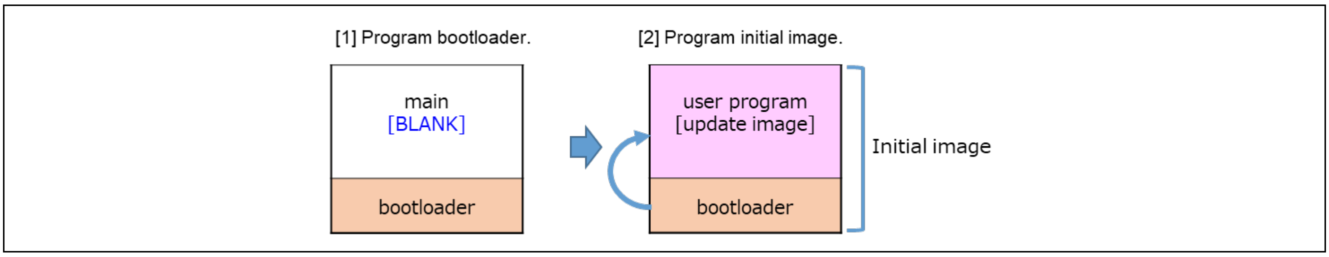


Figure 1.8 Initial Firmware Update Settings Utilizing Bootloader (Example of Full Update Method)

[1] Program bootloader.
The bootloader is programmed to the on-chip flash memory using a tool such as Flash Writer.

[2] Program initial image.
The initial image is downloaded from an external source and programmed to the main plane using the functionality of the bootloader. The programmed firmware is verified, and if verification is successful, processing ends.

1.5 Package Contents

The firmware update module package contains several files, including software and tools. These are listed in the table below.

Table 1.2 Folder Structure of Firmware Update Module Package

Folder Name	Description
r01an6374xx0203-rl78g23-fwupdate.zip\	
├─demos	Demo projects
│ ├─ccrl_iar	
│ │ ├─r_fwup	Firmware update module
│ │ └─rl	
│ │ ├─modules	Drivers and libraries
│ │ │ ├─3rd_party	
│ │ │ │ └─tinycrypt	Crypto library
│ │ │ ├─etc	
│ │ │ │ └─base64	Base64 decode
│ │ │ ├─flash	Flash wrapper
│ │ │ ├─rl78_serial	Serial driver
│ │ │ └─r_qspi_flash_mx25l	MX23L driver
│ │ ├─rl78g22-fpb	RL78/G22-48p FPB
│ │ │ └─linear	
│ │ │ ├─e2_ccrl	CCRL
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ └─fwup_leddemo	LED illumination application
│ │ │ └─iar	IAR
│ │ │ ├─boot_loader	Bootloader
│ │ │ └─fwup_leddemo	LED illumination application
│ │ ├─rl78g23-fpb	RL78/G23-128p FPB
│ │ │ └─linear	
│ │ │ ├─e2_ccrl	CCRL
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ └─iar	IAR
│ │ │ ├─boot_loader	Bootloader
│ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ └─fwup_main	User applications including firmware update
│ │ └─rl78g24-fpb	RL78/G24-64p FPB
│ │ └─linear	
│ │ ├─e2_ccrl	CCRL
│ │ │ ├─boot_loader	Bootloader
│ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ └─fwup_main	User applications including firmware update
│ │ └─iar	IAR
│ │ ├─boot_loader	Bootloader
│ │ ├─fwup_leddemo	LED illumination application
│ │ └─fwup_main	User applications including firmware update
│ └─llvm	
│ ├─r_fwup	Firmware update module
│ └─rl	

Folder Name	**Description**
│　　　　├─modules	Drivers and libraries, wrapper
│　　　　└─rl78g23-fpb	RL78/G23-128p FPB
│　　　　　　└─linear	
│　　　　　　　　└─llvm	LLVM
│　　　　　　　　　　├─boot_loader	Bootloader
│　　　　　　　　　　├─fwup_leddemo	LED illumination application
│　　　　　　　　　　└─fwup_main	User applications including firmware update
└─RenesasImageGenerator	Renesas Image Generator
├─image-gen.py	Python program for Renesas Image Generator
└─RL*78_xxxx*_ImageGenerator_PRM.csv	Parameter file for demo project

Note: The LLVM compiler is currently only available for RL78/G23; for RL78/G22 and G24, please use other compilers (CC-RL, IAR).

1.6 API Overview

Table 1.4 lists the API functions included in the firmware update module.

Table 1.3 API Functions

Function	Function Description
R_FWUP_Open	Opens the module.
R_FWUP_Close	Performs processing to close the module.
R_FWUP_IsExistImage	Confirms the existence of an image in the specified area.
R_FWUP_EraseArea	Erases the specified area.
R_FWUP_GetImageSize	Obtains the size of the image.
R_FWUP_WriteImage	Writes the image (header portion + program portion).
R_FWUP_VerifyImage	Validates the image.
R_FWUP_ActivateImage	Activates a new image.
R_FWUP_ExecImage	Launches a new image.
R_FWUP_SoftwareReset	Applies a software reset.
R_FWUP_SoftwareDelay	Applies a software delay.
R_FWUP_GetVersion	Returns the version number of the module.
R_FWUP_WriteImageHeader	Writes the header portion of the image.
R_FWUP_WriteImageProgram	Writes the program portion of the image.

Note: Special purpose refers to the use of FreeRTOS. If you are considering to use Rev2.0x firmware update module on bare metal without FreeRTOS, please skip this section.

2. API Information

2.1 Hardware Requirements

The MCU used must support the following functions:

- Flash memory

2.2 Software Requirements

The module is dependent upon the following drivers:

- Board support package (r_bsp)
- Renesas Flash Driver RL78 Type01 (RFD)
- UART Driver (r_Config_UART1)
- PORT Driver (r_Config_PORT)
- Macronix International MX25/66L family serial NOR Flash Memory control software (r_qspi_flash_mx25l)
- Clock-synchronized single-master control software using CSI mode for serial array units (rl78_serial)

2.3 Supported Toolchains

The module has been confirmed to work with the toolchains listed in 6.1, Confirmed Operation Environments.

2.4 Header Files

All API calls and their supporting interface definitions are located in r_fwup_if.h.

2.5 Integer Types

The driver uses ANSI C99. These types are defined in stdint.h.

2.6 Compile Settings

The configuration option settings of the module are contained in r_fwup_config.h.

The names of the options and descriptions of their setting values are listed in Table 2.1.

Table 2.1 Configuration Settings

Configuration options in r_fwup _config.h	
FWUP_CFG_UPDATE_MODE	Update method 0: Not available for RL78 1: Partial Update Method Method (buffer plane is internal flash) 2: Full Update Method (without buffer plane) 3: Full Update Method (buffer plane is external flash)
FWUP_CFG_FUNCTION_MODE	Specifies how the module is used. 0: Bootloader 1: Application program
FWUP_CFG_MAIN_AREA_ADDR_L	Specifies the start address of the main plane.
FWUP_CFG_BUF_AREA_ADDR_L	Specifies the start address of the buffer plane (in on-chip flash memory).
FWUP_CFG_AREA_SIZE	Specifies the size of the main plane and buffer plane.
FWUP_CFG_CF_BLK_SIZE	Specifies the block size of the on-chip code flash.
FWUP_CFG_CF_W_UNIT_SIZE	Specifies the writing unit for the on-chip code flash.
FWUP_CFG_EXT_BUF_AREA_ADDR_L	Specifies the start address of the buffer plane in external flash memory.
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	Specifies the block size or sector size of the external flash memory.
FWUP_CFG_DF_ADDR_L	Start address of data flash.
FWUP_CFG_DF_BLK_SIZE	Block size of data flash.
FWUP_CFG_DF_NUM_BLKS	Block count of data flash. Specify 0 if there is no data flash.
FWUP_CFG_FWUPV1_COMPATIBLE	FWUP V1 Compatibility Setting (For Special Purpose) 0: Disable (default) 1: Enable (For Special Purpose)
FWUP_CFG_SIGNATURE_VERIFICATION	Verification method 0: ECDSA + SHA256 (default) 1: SHA256
FWUP_CFG_PRINTF_DISABLE	Log display setting 0: Enable 1: Disable (default)

2.7 Demo Project Code Sizes

The table below shows the ROM size, RAM size, and maximum stack size of this module.

The values in the table below are confirmed under the following conditions.

Module revision:	Firmware update module for RL78 v2.0.0
Compiler version:	Renesas Electronics C Compiler Package for RL78 Family V1.11 IAR C/C++ Compiler for Renesas RL78 version 5.10.1 LLVM for Renesas RL78 17.0.1.202412
Configuration options:	Configuration option settings are listed in each FPB

CC-RL

Optimization level: size & execution speed (-Odefault)

Delete variables/functions that have never been referenced (-optimize=symbol_delete)

IAR

Optimization level: High (balanced)

LLVM
Optimization level:-Os

2.7.1 Demo Project for RL78/G23-128p FPB

Configuration settings for the RL78/G23-128p FPB demo project:

FWUP_CFG_UPDATE_MODE	1 : Single bank with buffer. (default)
FWUP_CFG_SIGNATURE_VERIFICATION	0 : ECDSA. (default)
FWUP_CFG_PRINTF_DISABLE	1 : Disable. (default)

Table 2.2 ROM, RAM, and Stack Code Size for boot_loader

Items	Category	Memory Used (byte)		
		CC-RL	IAR	LLVM
boot_loader	ROM	19027	26103	40811
	RAM	1365	3663	1078
	Stack	514	3146	506

Table 2.3 ROM, RAM, and Stack Code Size for fwup_main

Items	Category	Memory Used (byte)		
		CC-RL	IAR	LLVM
fwup_main	ROM	16080	24227	37144
	RAM	859	3661	412
	Stack	514	2204	504

2.7.2 Demo Project for RL78/G24-64p FPB

Configuration settings for the RL78/G24-64p FPB demo project:

FWUP_CFG_UPDATE_MODE	1 : Single bank with buffer. (default)
FWUP_CFG_SIGNATURE_VERIFICATION	0 : ECDSA. (default)
FWUP_CFG_PRINTF_DISABLE	1 : Disable. (default)

Table 2.4 ROM, RAM, and Stack Code Size for boot_loader

Items	Category	Memory Used (byte)	
		CC-RL	IAR
boot_loader	ROM	19282	26613
	RAM	1365	3688
	Stack	514	3146

Table 2.5 ROM, RAM, and Stack Code Size for fwup_main

Items	Category	Memory Used (byte)	
		CC-RL	IAR
fwup_main	ROM	16843	24533
	RAM	865	3684
	Stack	514	2204

2.7.3 Demo project for RL78/G22-48p FPB

Configuration settings for the RL78/G22-48p FPB demo project:

FWUP_CFG_UPDATE_MODE	2 : Single bank without buffer.
FWUP_CFG_SIGNATURE_VERIFICATION	1 : SHA256
FWUP_CFG_PRINTF_DISABLE	1 : Disable. (default)

Table 2.6 ROM, RAM, and Stack Code Size for boot_loader

Items	Category	Memory Used (byte)	
		CC-RL	IAR
boot_loader	ROM	9066	11408
	RAM	789	2057
	Stack	398	1956

2.8 Arguments

The return values of the API functions are shown below. This enumeration is located in r_fwup_if.h, as are the prototype declarations of the API functions.

```
typedef enum fwup_area
{
    FWUP_AREA_MAIN = 0,
    FWUP_AREA_BUFFER,
    FWUP_AREA_DATA_FLASH
} e_fwup_area_t;

typedef enum e_fwup_delay_units
{
    FWUP_DELAY_MICROSECS = 0,
    FWUP_DELAY_MILLISECS,
    FWUP_DELAY_SECS
} e_fwup_delay_units_t;
```

2.9 Return Values

The return values of the API functions are shown below. This enumeration is located in r_fwup_if.h, as are the prototype declarations of the API functions.

```
typedef enum fwup_err
{
    FWUP_SUCCESS = 0,               // Normally terminated.
    FWUP_PROGRESS,                  // Firmware update is in progress.
    FWUP_ERR_FLASH,                 // Detect error of flash module.
    FWUP_ERR_VERIFY,                // Verify error.
    FWUP_ERR_FAILURE,               // General error.
} e_fwup_err_t;
```

2.10 Implementation Examples of APIs

The following is an example implementation of a bootloader and application program for each firmware update method.
For details, please refer to the source code of the demo project included in this application note package.

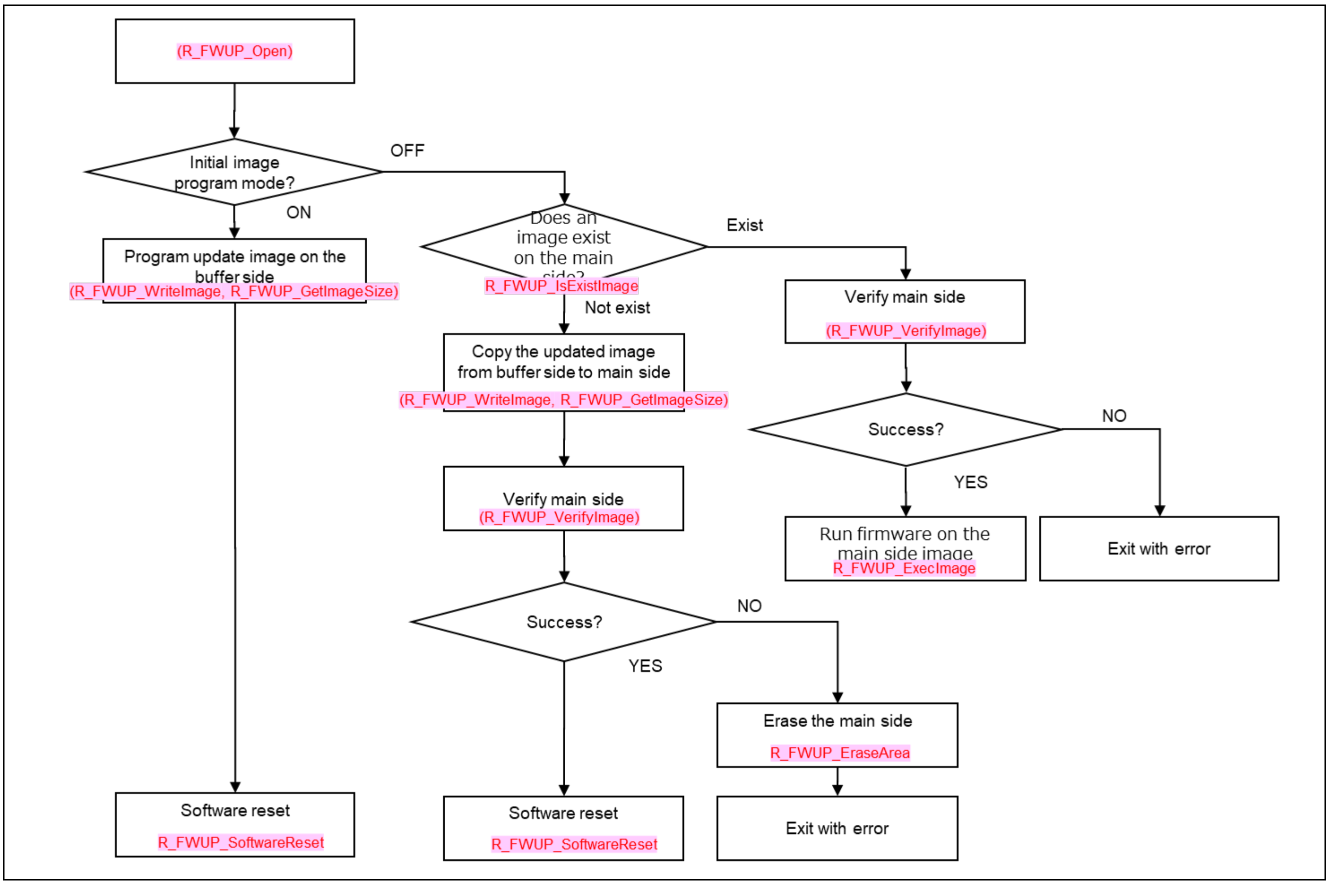


Figure 2.1 Bootloader Implementation Example for Partial/Full Update Method (with buffer plane)

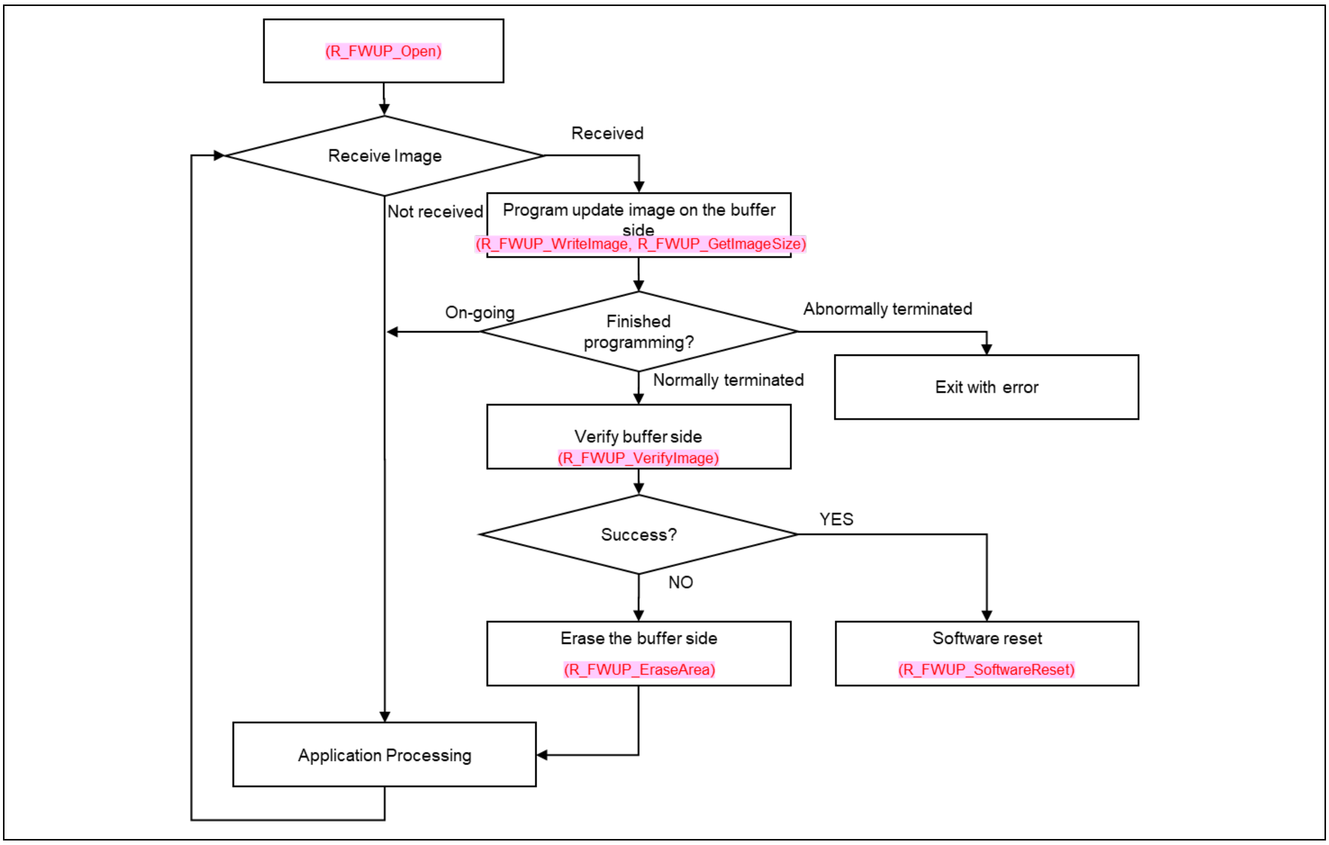


Figure 2.2 Application Program Implementation Example for Partial/Full Update Method (with buffer plane)

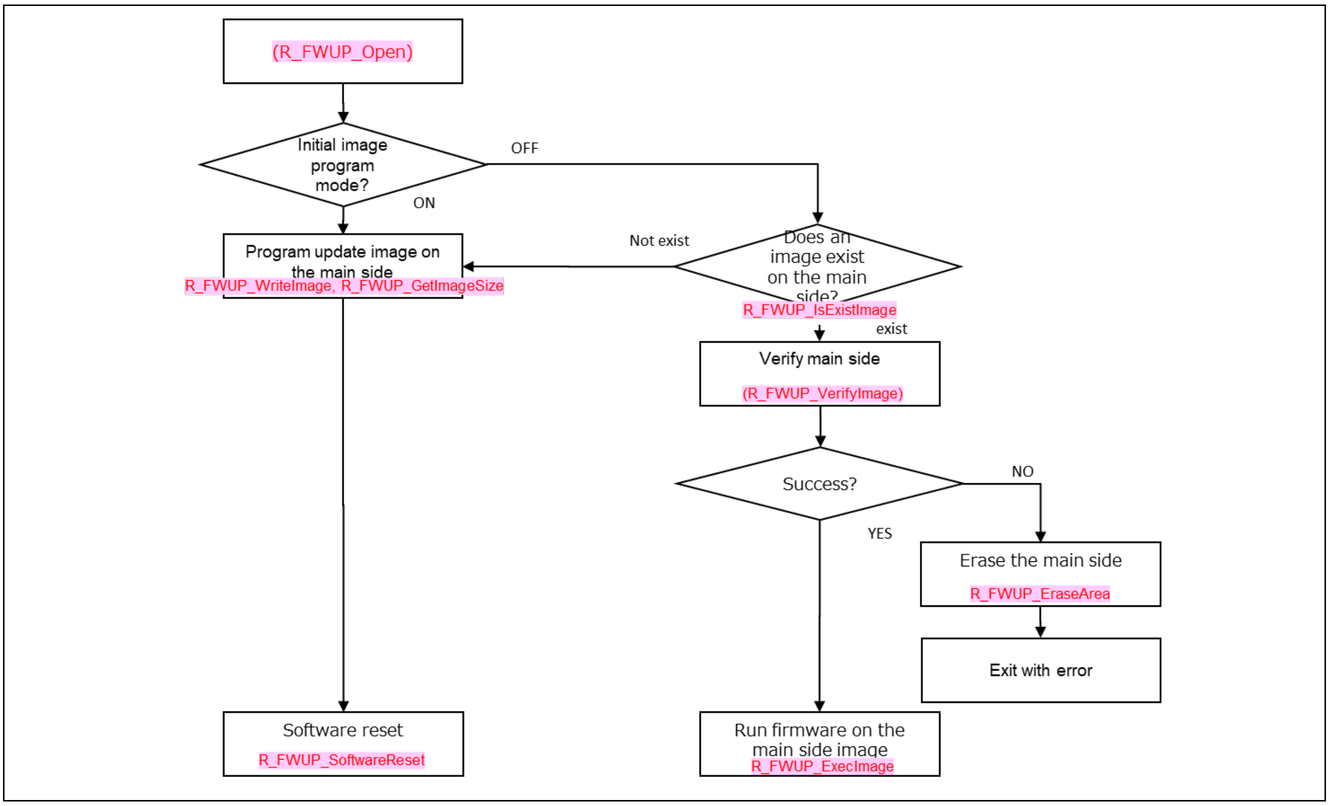


Figure 2.3 Bootloader Implementation Example for Full Update Method(without buffer plane)

3. API Functions

3.1 R_FWUP_Open Function

Table 3.1 R_FWUP_Open Function Specifications

Format	e_fwup_err_t R_FWUP_Open (void)	
Description	Performs processing to open the firmware update module. Implements processing to open the flash module.	
Parameters	None	
Return Values	FWUP_SUCCESS	Normal end
	FWUP_ERR_FLASH	Flash module error
Special Notes	—	

3.2 R_FWUP_Close Function

Table 3.2 R_FWUP_Close Function Specifications

Format	void R_FWUP_Close (void)
Description	Performs processing to close the firmware update module. Implements processing to close the flash module.
Parameters	None
Return Values	None
Special Notes	—

3.3 R_FWUP_IsExistImage Function

Table 3.3 R_FWUP_IsExistImage Function Specifications

Format	bool R_FWUP_IsExistImage(e_fwup_area_t area)	
Description	Confirms the existence of an image in the specified area.	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER)	
Return Values	true	Image exists.
	false	Image does not exist.
Special Notes	Verify that the magic code is written correctly.	

3.4 R_FWUP_EraseArea Function

Table 3.4 R_FWUP_EraseArea Function Specificationss

Format	e_fwup_err_t R_FWUP_EraseArea(e_fwup_area_t area)	
Description	Erases the specified area.	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER), Data Flash (FWUP_AREA_DATA_FLASH)	
Return Values	FWUP_SUCCES	Normal end
	FWUP_ERR_FLASH	Flash module error
Special Notes	Erasure of the main plane can only be performed by the bootloader.	

3.5 R_FWUP_GetImageSize Function

Table 3.5 R_FWUP_GetImageSize Function Specificationss

Format	uint32_t R_FWUP_GetImageSize(void)	
Description	Returns the size of the image in bytes. This function obtains the byte size of the image based on the RSU header address information shown in Figure 4.1. Therefore, first write the RSU header address information to code flash using the R_FWUP_WriteImage function or the R_FWUP_WriteImageProgram function.	
Parameters	None	
Return Values	0	Acquisition in progress
	1 or more	Image size
Special Notes	—	

3.6 R_FWUP_WriteImage Function

Table 3.6 R_FWUP_WriteImage Function Specifications

Format	e_fwup_err_t R_FWUP_WriteImage(e_fwup_area_t area, uint8_t *p_buf, uint32_t buf_size)	
Description	Writes an image (header portion + program portion) to the specified area. Continue calling this function until the total size of the image is reached. The image size is obtained by R_FWUP_GetImageSize().	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER) p_buf: Image (header + program) buffer buf_size: Buffer size*1	
Return Values	FWUP_SUCCES	Writing of all images is completed.
	FWUP_PROGRESS	Writing of all images not completed (Writing of the specified number of images completed)
	FWUP_ERR_FLASH	Flash module error
	FWUP_ERR_FAILURE	Illegal parameter
Special Notes	1. Specify a multiple of the code flash write unit (for example, 64, 128, or 256). This size also applies to data flash. When FWUP_CFG_FWUPV1_COMPATIBLE is enabled, the magic code in the RSU header area used for processing is "Renesas" for FWUP V1.	

3.7 R_FWUP_VerifyImage Function

Table 3.7 R_FWUP_VerifyImage Function Specifications

Format	e_fwup_err_t R_FWUP_VerifyImage(e_fwup_area_t area)	
Description	Verifies an image using the cryptographic library embedded in the module.	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER)	
Return Values	FWUP_SUCCES	Verification successful
	FWUP_ERR_VERIFY	Verification failed
	FWUP_ERR_FAILURE	Illegal parameter
Special Notes	—	

3.8 R_FWUP_ActivateImage Function

Table 3.8 R_FWUP_ActivateImage Function Specifications

Format	e_fwup_err_t R_FWUP_ActivateImage(void)	
Description	Activates a new image. • partial update method — Bootloader: Copies the buffer plane image to the main plane. — User program: Returns FWUP_SUCCESS without doing anything. • full update method — Returns FWUP_SUCCESS without doing anything.	
Parameters	None	
Return Values	FWUP_SUCCESS	Normal end
	FWUP_ERR_FLASH	Flash module error
Special Notes	—	

3.9 R_FWUP_ExecImage Function

Table 3.9 R_FWUP_ExecImage Function Specifications

Format	void R_FWUP_ExecImage(void)
Description	Runs the program in a valid image.
Parameters	None
Return Values	None
Special Notes	Disable interrupts when transitioning from the bootloader to the application.

3.10 R_FWUP_SoftwareReset Function

Table 3.10 R_FWUP_SoftwareReset Function Specifications

Format	void R_FWUP_SoftwareReset(void)
Description	Execute software reset processing.
Parameters	None
Return Values	None
Special Notes	—

3.11 R_FWUP_SoftwareDelay Function

Table 3.11 R_FWUP_SoftwareDelay Function Specifications

Format	uint32_t R_FWUP_SoftwareDelay(uint32_t delay, e_fwup_delay_units_t units)	
Description	Execute software delay processing.	
Parameters	delay: Delay time units: Unit (µs, ms, or sec.)	
Return Values	0	Normal end
	Other	Abnormal end
Special Notes	—	

3.12 R_FWUP_GetVersion Function

Table 3.12 R_FWUP_GetVersion Function Specifications

Format	uint32_t R_FWUP_GetVersion(void)
Description	Returns the version number of the module.
Parameters	None
Return Values	Version number
Special Notes	—

3.13 R_FWUP_WriteImageHeader Function

This function is an API for special use where header information and program information must be written separately. Normally, use the R_FWUP_WriteImage function.

Table 3.13 R_FWUP_WriteImageHeader Function Specifications

Format	e_fwup_err_t R_FWUP_WriteImageHeader (e_fwup_area_t area, uint8_t FWUP_FAR *p_sig_type, uint8_t FWUP_FAR *p_sig, uint32_t sig_size)	
Description	Writes a signature that the bootloader uses for verification to the header of the image in the designated area.	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER) p_sig_type: Signature type character string “hash-sha256” or “sig-sha256-ecdsa” p_sig: Signature sig_size: Length of signature (Should be set to 64.)	
Return Values	FWUP_SUCCES	Write completed
	FWUP_ERR_FLASH	Flash module error
	FWUP_ERR_FAILURE	Illegal parameter
Special Notes	When FWUP_CFG_FWUPV1_COMPATIBLE is enabled, the magic code in the RSU header area used for processing is "Renesas" for FWUP V1.	

3.14 R_FWUP_WriteImageProgram Function

This function is an API for special use where header information and program information must be written separately. Normally, use the R_FWUP_WriteImage function.

Table 3.14 R_FWUP_WriteImageProgram Function Specifications

Format	e_fwup_err_t R_FWUP_WriteImageProgram (e_fwup_area_t area, uint8_t *p_buf, uint32_t offset, uint32_t buf_size)	
Description	Writes the program portion of the image to the specified area. Continue calling this function until the total size of the image is reached. The image size is obtained by R_FWUP_GetImageSize(). This function writes a program by offset based on the address information in the RSU header shown in Figure 4.1. Therefore, be sure to set 0x100 bytes of data from the offset (0x200) in Table 4.3 in the first call to this function. (Specify 0x200 for the offset argument and 0x100 or more for the buf_size argument.)	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER) p_buf: Buffer for program portion of image offset: Offset*[1] buf_size: Buffer size*[2]	
Return Values	FWUP_SUCCES	Writing of all images is completed.
	FWUP_PROGRESS	Writing of all images not completed (Writing of the specified number of images completed)
	FWUP_ERR_FLASH	Flash module error
	FWUP_ERR_FAILURE	Illegal parameter
Special Notes	1. The offset must be 0x200 or greater. 2. Specify a multiple of the code flash write unit (for example, 64, 128, or 256). This size also applies to data flash.	

3.15 Wrapper Functions

This module implements the flash driver and cryptographic operations in a wrapper function. The process is implemented in the following comment section of the source file. Please refer to the demo project for the implementation method.

/**** Start user code ****/

/**** End user code ****/

3.15.1 r_fwup_wrap_com.c, h

3.15.1.1 r_fwup_wrap_disable_interrupt Function

Table 3.15 r_fwup_wrap_disable_interrupt Function Specifications

Format	void r_fwup_wrap_disable_interrupt (void)
Description	Disable Interrupt
Parameters	None
Return Values	None
Special Notes	—

3.15.1.2 r_fwup_wrap_enable_interrupt Function

Table 3.16 r_fwup_wrap_enable_interrupt Function Specifications

Format	void r_fwup_wrap_enable_interrupt (void)
Description	Enable Interrupt
Parameters	None
Return Values	None
Special Notes	—

3.15.1.3 r_fwup_wrap_software_reset Function

Table 3.17 r_fwup_wrap_software_reset Function Specifications

Format	void r_fwup_wrap_software_reset (void)	
Description	Software reset	
Parameters	None	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.1.4 r_fwup_wrap_software_delay Function

Table 3.18 r_fwup_wrap_software_delay Function Specifications

Format	uint32_t r_fwup_wrap_software_delay (uint32_t delay, e_fwup_delay_units_t units)	
Description	Software delay	
Parameters	delay：Delay time uinits: unit (us,ms,sec) FWUP_DELAY_MICROSECS FWUP_DELAY_MILLISECS FWUP_DELAY_SECS	
Return Values	0	: normal end
	Other	: Abnormal end
Special Notes	—	

3.15.2 r_fwup_wrap_flash.c, h

3.15.2.1 r_fwup_wrap_flash_open Function

Table 3.19 r_fwup_wrap_flash_open Function Specifications

Format	e_fwup_err_t r_fwup_wrap_flash_open (void)	
Description	Open the internal flash.	
Parameters	None	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.2 r_fwup_wrap_flash_close Function

Table 3.20 r_fwup_wrap_flash_close Function Specifications

Format	void r_fwup_wrap_flash_close (void)
Description	Close the internal flash.
Parameters	None
Return Values	None
Special Notes	—

3.15.2.3 r_fwup_wrap_flash_erase Function

Table 3.21 r_fwup_wrap_flash_erase Function Specifications

Format	e_fwup_err_t r_fwup_wrap_flash_erase (uint32_t addr, uint32_t num_blocks)	
Description	Erase the internal flash in block units.	
Parameters	addr : erase address num_blocks : erase block	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.4 r_fwup_wrap_flash_write Function

Table 3.22 r_fwup_wrap_flash_write Function Specifications

Format	e_fwup_err_t r_fwup_wrap_flash_write(uint32_t src_addr, uint32_t dest_addr, uint32_t num_bytes)	
Description	Erase the internal flash in block units.	
Parameters	src_addr: Pointer to write data dest_addr: Write address num_bytes: Write size (bytes)	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.5 r_fwup_wrap_flash_read Function

Table 3.23 r_fwup_wrap_flash_read Function Specifications

Format	e_fwup_err_t r_fwup_wrap_flash_read (uint32_t buf_addr, uint32_t src_add, uint32_t size)	
Description	Reads the internal flash.	
Parameters	buf_addr: Address of the buffer to store the read data src_addr: Read address size: Read size	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.6 r_fwup_wrap_bank_swap Function

Table 3.24 r_fwup_wrap_bank_swap Function Specifications

Format	e_fwup_err_t r_fwup_wrap_bank_swap (void)	
Description	Execute a bank swap.	
Parameters	None	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	Only models with dual banks	

3.15.2.7 r_fwup_wrap_ext_flash_open Function

Table 3.25 r_fwup_wrap_ext_flash_open Function Specifications

Format	e_fwup_err_t r_fwup_wrap_ext_flash_open (void)	
Description	Open the external flash.	
Parameters	None	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.8 r_fwup_wrap_ext_flash_close Function

Table 3.26 r_fwup_wrap_ext_flash_close Function Specifications

Format	void r_fwup_wrap_ext_flash_close (void)
Description	Close the external flash.
Parameters	None
Return Values	None
Special Notes	—

3.15.2.9 r_fwup_wrap_ext_flash_erase Function

Table 3.27 r_fwup_wrap_ext_flash_erase Function Specifications

Format	e_fwup_err_t r_fwup_wrap_ext_flash_erase (uint32_t offsetadd, uint32_t num_sectors)	
Description	Erase the external flash in sector units.	
Parameters	offsetadd: Starting address of the sector to be erased num_sectors: Number of sectors	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.10 r_fwup_wrap_ext_flash_write Function

Table 3.28 r_fwup_wrap_ext_flash_write Function Specifications

Format	e_fwup_err_t r_fwup_wrap_ext_flash_write (uint32_t src_addr, uint32_t dest_addr, uint32_t num_bytes);	
Description	Writes data to external flash.	
Parameters	src_addr: Pointer to write data dest_addr: Write address num_bytes: Write size (bytes)	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.2.11 r_fwup_wrap_ext_flash_read Function

Table 3.29 r_fwup_wrap_ext_flash_read Function Specifications

Format	e_fwup_err_t r_fwup_wrap_ext_flash_read (uint32_t buf_addr, uint32_t src_addr, uint32_t size);	
Description	Reads the external flash.	
Parameters	buf_addr: Address of the buffer to store the read data src_addr: Read address size: Read size	
Return Values	FWUP_SUCCES	: Normal end
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.15.3 r_fwup_wrap_verify.c, h

3.15.3.1 r_fwup_wrap_sha256_init Function

Table 3.30 r_fwup_wrap_sha256_init Function Specifications

Format	int32_t r_fwup_wrap_sha256_init (void *vp_ctx);	
Description	Start hash value calculation.	
Parameters	vp_ctx: pointer to the context of the cryptographic library	
Return Values	0	: normal end
	Other	: Abnormal end
Special Notes	—	

3.15.3.2 r_fwup_wrap_sha256_update Function

Table 3.31 r_fwup_wrap_sha256_update Function Specifications

Format	int32_t r_fwup_wrap_sha256_update (void *vp_ctx, const uint8_t *p_data, uint32_t datalen)	
Description	Calculates hash values for a specified range.	
Parameters	vp_ctx: pointer to the context of the cryptographic library p_data: starting address datalen: data length (bytes)	
Return Values	0	: normal end
	Other	: Abnormal end
Special Notes	—	

3.15.3.3 r_fwup_wrap_sha256_final Function

Table 3.32 r_fwup_wrap_sha256_final Function Specifications

Format	int32_t r_fwup_wrap_sha256_final (uint8_t *p_hash, void *vp_ctx)	
Description	Finishes computing the hash value and returns the hash value.	
Parameters	p_hash: Pointer to the buffer to store the calculated hash value vp_ctx: pointer to the context of the cryptographic library	
Return Values	0	: normal end
	Other	: Abnormal end
Special Notes	—	

3.15.3.4 r_fwup_wrap_verify_ecdsa Function

Table 3.33 r_fwup_wrap_verify_ecdsa Function Specifications

Format	int32_t r_fwup_wrap_verify_ecdsa (uint8_t *p_hash, uint8_t *p_sig_type, uint8_t *p_sig, uint32_t sig_size)	
Description	Perform verification with ECDSA.	
Parameters	p_hash: Pointer to the buffer where the hash value is stored p_sig_type: signature type p_sig: signature sig_size: signature size	
Return Values	0	: normal end
	Other	: Abnormal end
Special Notes	—	

3.15.3.5 r_fwup_wrap_get_crypt_context Function

Table 3.34 r_fwup_wrap_get_crypt_context Function Specifications

Format	void * r_fwup_wrap_get_crypt_context (void);	
Description	Returns a pointer to the context of the cryptographic library.	
Parameters	None	
Return Values	Void *	Pointer to cryptographic library context
Special Notes	—	

4. Demo Project

The demo project is a sample program that shows how to implement firmware update functionality using the serial communications interface (SCI).

Note: The RL78/G24 and RL78/G22 demo project does not support the LLVM compiler. Please use other compilers (CC-RL, IAR) for RL78/G24 and RL78/G22.

4.1 Demo project Structure

The demo project comprises the module, modules dependent on it, and a main() function that implements the firmware update demonstration. Versions of the demo project for the devices and compilers listed in 1.5 are provided.

The firmware update demo consists of the following projects.

folder structure： Under □□\ linear \△△\
□□ : Board name
△△ : Compiler (e2_ccrl / iar / llvm)

- boot_loader: Bootloader
  This program runs first after a reset. It verifies that the user program has not been tampered with and then, if verification is successful, launches the user program.
- fwup_main: Application program
  An application program (initial firmware) that downloads updated firmware and performs signature verification.
- fwup_leddemo: Application program (for update)
  This is an application program (for updating) that blinks an LED.

4.2 Operating environment preparation

To run the firmware update demo project, you need to install the tools (see 4.2.1 to 4.2.4) on your Windows PC. Also, use a USB serial conversion board (see 4.2.5) that connects the Windows PC and the target board.

4.2.1 Installing TeraTerm

Used to transfer the firmware update image via serial communication from a Windows PC to the target board. In the demo project, we have checked the operation with TeraTerm 4.105.

After installation, set the serial port communication settings as shown in Table Table 4.1

Table 4.1 Communication Specifications

Item	Description
Communication system	Asynchronous communication
Bit rate	115,200 bps
Data length	8 bits
Parity	None
Stop bit	1 bit
Flow control	CTS/RTS

4.2.2 Installing the Python execution environment

Used by Renesas Image Generator (image-gen.py) to create initial and update images.

Renesas Image Generator uses ECDSA to generate signature data. In the demo project, environment operation is confirmed with Python 3.9.0.

Install Python 3.9.0 or higher.

In addition, since the Python encryption library (pycryptodome) is used, after installing Python, execute the following pip command from the command prompt to install the library.

```
pip install pycryptodome
```

4.2.3 Installing the OpenSSL execution environment

OpenSSL is used to generate the keys needed to generate and verify ECDSA signature data for initial and update images.

Download the OpenSSL installer from the following URL and install it. There is no problem with the Light version.

https://slproweb.com/products/Win32OpenSSL.html

4.2.4 Installing the Flash Writer

A flash writer is required to write the initial image.

The demo project uses Renesas Flash Programmer v3.11.01.

Renesas Flash Programmer (Programming GUI) | Renesas

4.2.5 USB serial conversion board

Used to transfer the firmware update image via serial communication from a Windows PC to the target board.

For details on how to connect with the target board, refer to the operation confirmation environment (6.2) of the relevant target board.

Use Pmod USBUART (manufactured by DIGILENT).

https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

4.3 Execution environment preparation

4.3.1 Generating Keys for Signature Generation and Verification

Use OpenSSL for key generation. Refer to 4.2.3 in advance and install OpenSSL.

Execute the following OpenSSL commands to generate an elliptic curve cryptography (secp256r1) key pair to be used to generate and verify image signatures, and to extract the private and public keys:

```
>openssl ecparam -genkey -name secp256r1 -out secp256r1.keypair
using curve name prime256v1 instead of secp256r1

>openssl ec -in secp256r1.keypair -outform PEM -out secp256r1.privatekey
read EC key
writing EC key

> openssl ec -in secp256r1.keypair -outform PEM -pubout -out
secp256r1.publickey
read EC key
writing EC key
```

4.3.2 Preparing the execution environment for Renesas Image Generator

Unzip ImageGenerator.zip included in the package to any folder on your Windows PC. Make sure the folder name does not contain double-byte characters.

Renesas Image Generator requires a Python execution environment, so refer to 4.2.2 and install Python in advance.

4.4 Demo Project for RL78/G23 and G24

This section describes the demo projects using the RL78/G23-128p FPB.

See this chapter for a demonstration using the RL78/G24-64p FPB, which is the same as the RL78/G23-128p FPB.

Three demo projects for RL78/G23-128p FPB, boot_loader, fwup_leddemo, and fwup_main, are available for CC-RL compiler and IAR compiler, as shown. These demo projects support two firmware update methods by changing configuration settings: partial update method (buffer plane is internal flash) and full update method (buffer plane is external flash).

The execution procedure of this demo project does not assume a debugger connection. Refer to 4.6 for information on how to debug the application through a debugger connection.

4.4.1 Partial Update Method (buffer face is internal flash)

4.4.1.1 Execution Environment

Prepare the RL78/G23 operation check environment (6.2.1).

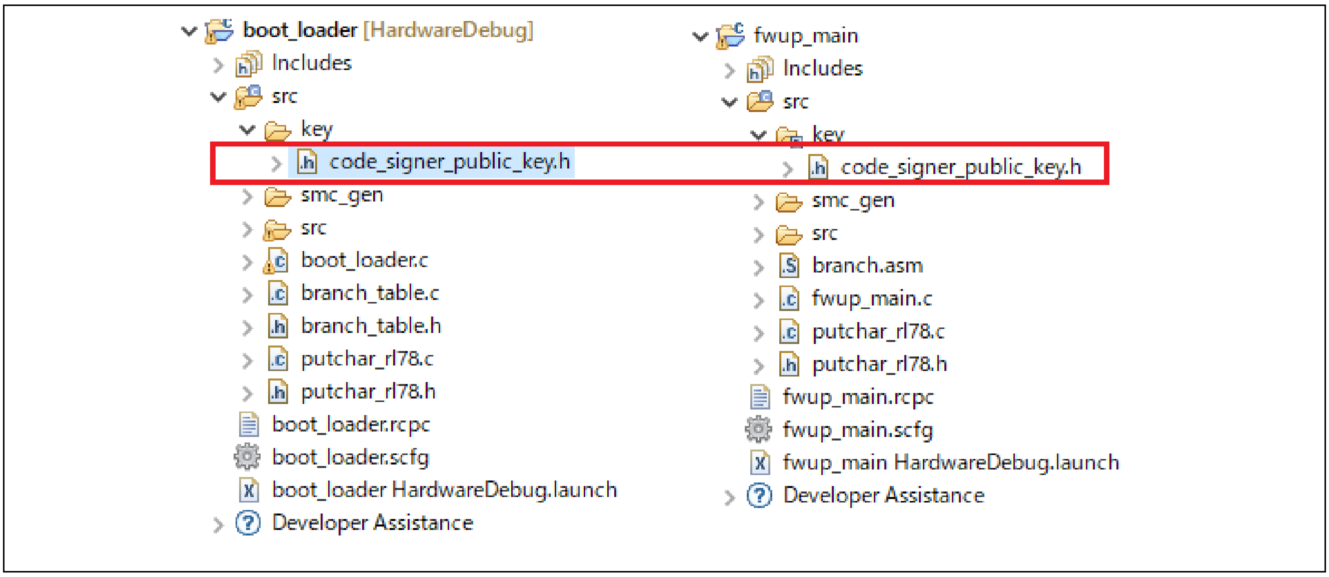
4.4.1.2 Build Demo Project

The following steps are used to build three demo projects for the partial update method (buffer plane is internal flash).

The following procedure is described for the e2 studio environment; when using the IAR environment, please read and follow the procedure for IAR's Integrated Development Environment.

1. Import the boot_loader, fwup_leddemo, and fwup_main demo projects into the integrated development environment.

2. Add the public key used to verify the image to the demo project.
Paste the contents of secp256r1.publickey into code_signer_public_key.h in project boot_loader and fwup_main.



```
boot_loader [HardwareDebug]
  Includes
  src
    key
      code_signer_public_key.h
    smc_gen
    src
    boot_loader.c
    branch_table.c
    branch_table.h
    putchar_rl78.c
    putchar_rl78.h
  boot_loader.rcpc
  boot_loader.scfg
  boot_loader HardwareDebug.launch
  Developer Assistance
```

```
fwup_main
  Includes
  src
    key
      code_signer_public_key.h
    smc_gen
    src
    branch.asm
    fwup_main.c
    putchar_rl78.c
    putchar_rl78.h
  fwup_main.rcpc
  fwup_main.scfg
  fwup_main HardwareDebug.launch
  Developer Assistance
```

Figure 4.1 Location of the code_signer_public_key.h file for the demo project

```
/*
 * PEM-encoded code signer public key.
 *
 * Must include the PEM header and footer:
 * "-----BEGIN CERTIFICATE-----\n"\
 * "...base64 data...\n"\
 * "-----END CERTIFICATE-----"
 */
#define CODE_SIGNER_PUBLIC_KEY_PEM \
"-----BEGIN PUBLIC KEY-----"\
Paste the contents of secp256r1.publickey here.
"-----END PUBLIC KEY-----"\
#endif /* CODE_SIGNER_PUBLIC_KEY_H_ */
```

3. Set the configuration settings for the firmware update module.
   Open r_fwup_config.h in the project and configure as shown in Table 6.4 or Table 6.5 is not used.
   Log output is off by default, so please set the following.

boot_loader, fwup_main	: FWUP_CFG_PRINTF_DISABLE = 0
fwup_leddemo	: FWUP_DEMO_PRINTF_DISABLE = 0

4. Build the demo project.
   Build the three demo projects and verify that the following mot files have been generated:
   boot_loader.mot, fwup_main.mot, fwup_leddemo.mot

   Note: The mot file is renamed srec file in llvm.

4.4.1.3 Create initial and updated images

This section describes the procedure for creating the initial and updated images, assuming that the initial image name is initial_firm.mot and the updated image name is fwup_leddemo.rsu.

1. Store the mot file of the built demo project and the secret key generated in 4.3.1 in the same folder as the Renesas Image Generator.

```
 image-gen.py
RL78_G23_ImageGenerator_PRM.csv
RL78_G24_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
```

2. Execute the following command to create the initial image.

```
> python image-gen.py -iup fwup_main.mot -ip RL78_G23_ImageGenerator_PRM.csv
-o initial_firm -ibp boot_loader.mot -vt ecdsa

Successfully generated the initial_firm.mot file.
```

3. Execute the following command to create the updated image.

```
> python image-gen.py -iup fwup_leddemo.mot -ip
RL78_G23_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa

Successfully generated the fwup_leddemo.rsu file.
```

Initial and updated images are generated in the same folder as the Renesas Image Generator.

```
image-gen.py
RL78_G23_ImageGenerator_PRM.csv
RL78_G24_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
fwup_leddemo.rsu
initial_firm.mot
```

4.4.1.4 Program Initial Image

Write the initial image (initial_firm.mot) to the MCU board using a flash writer. After writing, turn off the power to the board.

4.4.1.5 Update Firmware

Once the initial image firmware is activated, it waits for the transfer of the updated image via the terminal. The received update image is programmed into flash memory, and after the reception is completed, the signature of the update image is verified and the firmware of the update image is activated.

Follow the steps below to try the firmware update.

1. Connect devices with reference to "Figure 6.1 RL78/G23-128p FPB Device Connection Diagram".
2. Start the terminal software on the PC, select the serial COM port, and configure the connection settings.
3. Turn on power to the board. The following message is output.

For CC-RL and IAR.

```
==== RL78G23 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...

==== RL78G23 : Update from User [with buffer] ver 1.0.0 ====
send image(*.rsu) via UART.
```

For LLVM.

```
==== RL78G23 : BootLoader [with buffer] ====
verify install OK
execute new image ...

==== [fwup_main] RL78G23 : Update from User [with buffer] ver 1.0.0 ====
[fwup_main] send image(*.rsu) via UART.
```

4. Send the updated image through the terminal software.

Send file>check binary>fwup_leddemo.rsu

The following message is output during the transfer of the update image, and the software resets after installation and signature verification are complete.

For CC-RL and IAR.

```
W 0x59000, 128 ... OK
W 0x59080, 128 ... OK
...
W 0x5BA00, 128 ... OK
W 0x5BA80, 128 ... OK
verify install area buffer [sig-sha256-ecdsa]...OK
software reset...
```

For LLVM.

```
[fwup_main] Flash Write OK
[fwup_main] Flash Write OK
...
[fwup_main] Flash Write OK
[fwup_main] Flash Write OK
[fwup_main] verify install
[fwup_main] software reset...
```

5. Execute the activation process in the bootloader and perform a software reset again.

   For CC-RL and IAR.

```
==== RL78G23 : BootLoader [with buffer] ====
verify install area buffer [sig-sha256-ecdsa]...OK
copy to main area ... OK
software reset...
```

   For LLVM.

```
==== RL78G23 : BootLoader [with buffer] ====
verify install OK
copy to main area ... OK
software reset...
```

6. When the signature verification is completed in the bootloader, the firmware of the updated image will boot. It is normal if the following message is output and the LED is blinking.

   For CC-RL and IAR.

```
==== RL78G23 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...

----------------------------------------
FWUP demo (ver 0.1.1)
----------------------------------------
Check the LEDs on the board.
```

   For LLVM.

```
==== RL78G23 : BootLoader [with buffer] ====
verify install OK
execute new image ...

----------------------------------------
FWUP demo (ver 0.1.1)
----------------------------------------
Check the LEDs on the board.
```

Note: The demo program does not erase the buffer plane. If you need to erase the image before updating for rollback measures, please add a process to erase the buffer plane image.

4.4.2 Full Update Method (buffer face is external flash)

4.4.2.1 Execution Environment

Prepare the RL78/G23 operation check environment (6.2.1).

4.4.2.2 Build Demo Project

The following steps are used to build three demo projects for the partial update method (buffer plane is external flash).

The following procedure is described for the e2 studio environment; when using the IAR environment, please read and follow the procedure for IAR's Integrated Development Environment.

1. Import the boot_loader, fwup_leddemo, and fwup_main demo projects into the integrated development environment.

2. Add the public key used to verify the image to the demo project.
   Paste the contents of secp256r1.publickey into code_signer_public_key.h in project boot_loader and fwup_main.

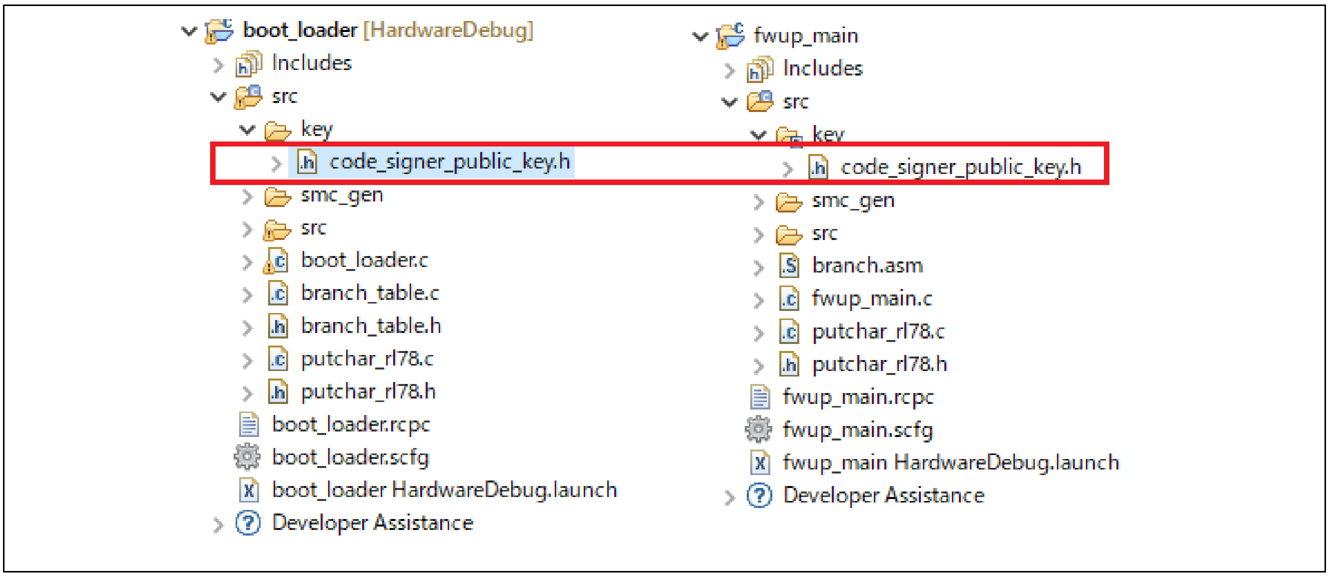


Figure 4.2 Location of the code_signer_public_key.h file for the demo project

```
/*
 * PEM-encoded code signer public key.
 *
 * Must include the PEM header and footer:
 * "-----BEGIN CERTIFICATE-----\n"\
 * "...base64 data...\n"\
 * "-----END CERTIFICATE-----"
 */
#define CODE_SIGNER_PUBLIC_KEY_PEM \
"-----BEGIN PUBLIC KEY-----"\
Paste the contents of secp256r1.publickey here.
"-----END PUBLIC KEY-----"\
#endif /* CODE_SIGNER_PUBLIC_KEY_H_ */
```

3. Set the configuration settings for the firmware update module.
   Open r_fwup_config.h in the project and configure as shown in Table 6.6 or Table 6.7 is not used.
   Log output is off by default, so please set the following.

boot_loader, fwup_main	: FWUP_CFG_PRINTF_DISABLE = 0
fwup_leddemo	: FWUP_DEMO_PRINTF_DISABLE = 0

4. Build the demo project.
   Build the three demo projects and verify that the following mot files have been generated:
   boot_loader.mot, fwup_leddemo.mot, fwup_main.mot

   Note: The mot file is renamed srec file in llvm.

4.4.2.3 Create initial and updated images

This section describes the procedure for creating the initial and updated images, assuming that the initial image name is initial_firm.mot and the updated image name is fwup_leddemo.rsu.

1. Store the mot file of the built demo project and the secret key generated in 4.3.1 in the same folder as the Renesas Image Generator.

```
image-gen.py
RL78_G23_ImageGenerator_PRM.csv
RL78_G24_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
```

2. Execute the following command to create the initial image.

```
> python image-gen.py -iup fwup_main.mot -ip
RL78_G23_ImageGenerator_PRM.csv -o initial_firm -ibp boot_loader.mot -vt
ecdsa -key secp256r1.privatekey

Successfully generated the initial_firm.mot file.
```

3. Execute the following command to create the updated image.

```
> python image-gen.py -iup fwup_leddemo.mot -ip
RL78_G23_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa -key
secp256r1.privatekey

Successfully generated the fwup_leddemo.rsu file.
```

Initial and updated images are generated in the same folder as the Renesas Image Generator.

```
image-gen.py
RL78_G23_ImageGenerator_PRM.csv
RL78_G24_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
fwup_leddemo.rsu
initial_firm.mot
```

4.4.2.4 Program Initial Image

Use Flash Writer to program the initial image (initial_firm.mot) to the MCU board. After programming, turn off the power to the board and disconnect the debugger (E2 Lite).

4.4.2.5 Update Firmware

Once the initial image firmware is activated, it waits for the transfer of the updated image via the terminal. The received update image is programmed into flash memory, and after the reception is completed, the signature of the update image is verified and the firmware of the update image is activated.

Follow the steps below to try the firmware update.

1. Connect devices with reference to "Figure 6.2 RL78/G23-128p FPB Device Connection Pin Information".
2. Start the terminal software on the PC, select the serial COM port, and configure the connection settings.
3. Turn on power to the board. The following message is output.

   For CC-RL and IAR.

```
==== RL78G23 : BootLoader [with ext-buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...

==== RL78G23 : Update from User [with ext-buffer] ver 1.0.0 ====
send image(*.rsu) via UART.
```

   For LLVM.

```
==== RL78G23 : BootLoader [with ext-buffer] ====
verify install OK
execute new image ...

==== [fwup_main] RL78G23 : Update from User [with ext-buffer] ver 1.0.0 ====
[fwup_main] send image(*.rsu) via UART.
```

4. Send the updated image through the terminal software.

   Send file>check binary>fwup_leddemo.rsu

   The following message is output during the transfer of the update image, and the software resets after installation and signature verification are complete.

   For CC-RL and IAR.

```
W 0x0000, 128 ... OK
W 0x0080, 128 ... OK
...
W 0x1A00, 128 ... OK
W 0x1A80, 128 ... OK
verify install area buffer [sig-sha256-ecdsa]...OK
software reset...
```

   For LLVM.

```
[fwup_main] Flash Write OK
[fwup_main] Flash Write OK
...
[fwup_main] Flash Write OK
[fwup_main] Flash Write OK
[fwup_main] verify install OK
[fwup_main] software reset...
```

5. Execute the activation process in the bootloader and perform a software reset again.

For CC-RL and IAR.

```
==== RL78G23 : BootLoader [with ext-buffer] ====
verify install area buffer [sig-sha256-ecdsa]...OK
copy to main area ... OK
software reset...
```

For LLVM.

```
==== RL78G23 : BootLoader [with ext-buffer] ====
verify install OK
copy to main area ... OK
software reset...
```

6. When the signature verification is completed in the bootloader, the firmware of the updated image will boot. It is normal if the following message is output and the LED is blinking.

For CC-RL and IAR.

```
==== RL78G23 : BootLoader [with ext-buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...

----------------------------------------
FWUP demo (ver 0.1.1)
----------------------------------------
Check the LEDs on the board.
```

For LLVM.

```
==== RL78G23 : BootLoader [with ext-buffer] ====
verify install OK
execute new image ...

----------------------------------------
FWUP demo (ver 0.1.1)
----------------------------------------
Check the LEDs on the board.
```

Note: The demo program does not erase the buffer plane. If you need to erase the image before updating for rollback measures, please add a process to erase the buffer plane image.

4.5 Demo Project for RL78/G22

This section describes the demo projects using the RL78/G22-48p FPB.

Three demo projects for RL78/G22-48p FPB, boot_loader and fwup_leddemo, are available for CC-RL compiler and IAR compiler, as shown in the following. These demo projects support the firmware update method, which is a full update method (without buffer plane).

The execution procedure of this demo project does not assume a debugger connection. Refer to 4.6 for information on how to debug the application through a debugger connection.

4.5.1 Full Update Method (without buffer plane)

4.5.1.1 Execution Environment

Prepare the RL78/G22 operation check environment (6.2.3).

4.5.1.2 Build Demo Project

The following steps are used to build two demo projects for the full update method (without buffer plane).

1. Import the boot_loader and fwup_leddemo demo projects into the integrated development environment.

2. Set the configuration settings for the firmware update module.
   Open r_fwup_config.h in the project and configure as shown in Table 6.7 is used.
   Log output is off by default, so please set the following.
   boot_loader : FWUP_CFG_PRINTF_DISABLE = 0
   fwup_leddemo : FWUP_DEMO_PRINTF_DISABLE = 0

3. Build the demo project.
   a. Build the project (boot_loader) and generate boot_loader.mot.
   b. Build the project (fwup_leddemo) and generate fwup_leddemo.mot.
   c. Rename fwup_leddemo.mot to fwup_leddemo_011.mot.
   d. Change the version of the project (fwup_leddemo) as follows, build and generate fwup_leddemo.mot.

```
fwup_leddemo.c
---
#define FWUP_DEMO_VER_MAJOR        (0)
#define FWUP_DEMO_VER_MINOR        (1)
#define FWUP_DEMO_VER_BUILD        (1)★1->2
```

   e. Rename fwup_leddemo.mot to fwup_leddemo_012.mot.

4.5.1.3 Create initial and updated images

This section describes the procedure for creating the initial and updated images, assuming that the initial image name is initial_firm.mot and the updated image name is fwup_leddemo_012.rsu.

1. Store the mot file of the built demo project in the same folder as Renesas Image Generator.

```
image-gen.py
RL78_G22_ImageGenerator_PRM.csv
boot_loader.mot
fwup_leddemo_011.mot
fwup_leddemo_012.mot
```

2. Execute the following command to create the initial image.

```
> python image-gen.py -iup fwup_leddemo_011.mot -ip
RL78_G22_ImageGenerator_PRM.csv -o initial_firm -ibp boot_loader.mot

Successfully generated the initial_firm.mot file.
```

3. Execute the following command to create the updated image.

```
> python image-gen.py -iup fwup_leddemo_012.mot -ip
RL78_G22_ImageGenerator_PRM.csv -o fwup_leddemo_012

Successfully generated the fwup_leddemo_012.rsu file.
```

Initial and updated images are generated in the same folder as the Renesas Image Generator.

```
image-gen.py
RL78_G22_ImageGenerator_PRM.csv
boot_loader.mot
fwup_leddemo_011.mot
fwup_leddemo_012.mot
fwup_leddemo_012.rsu
initial_firm.mot
```

4.5.1.4 Program Initial Image

Write the initial image (initial_firm.mot) to the MCU board using a flash writer. After writing, turn off the board power and disconnect the debugger (E2 Lite).

4.5.1.5 Update Firmware

The LED will blink when the initial image is activated. Enter update mode by pressing RESET_SW while holding down USER_SW on the board and wait for the transfer of the update image via the terminal. Program the received update image into flash memory, verify the update image after the transfer is complete, and then boot the firmware of the update image.

Follow the steps below to try the firmware update.

1. Connect devices with reference to "Figure 6.11 RL78/G22 full update method demo project memory map".
2. Start the terminal software on the PC, select the serial COM port, and configure the connection settings.
3. Power on the board. The following message is output.

```
==== RL78G22 : BootLoader [without buffer] ====
verify install area main [hash-sha256]...OK
execute new image ...

----------------------------------------
FWUP demo (ver 0.1.1)
----------------------------------------
```

4. Press RESET_SW while holding down USER_SW. The initial image programme mode is switched on.

```
==== RL78G22 : Image updater [without buffer] ====
send image(*.rsu) via UART.
```

5. Send the updated image through the terminal software.

   Send file>check binary>fwup_leddemo_012.rsu

   The following message is output during the transfer of the update image, and the software resets after installation and signature verification are complete.

```
W 0x2000, 64 ... OK
W 0x2040, 64 ... OK
...
W 0x4D00, 64 ... OK
W 0x4D40, 64 ... OK
verify install area 0 [hash-sha256]...OK
software reset...
```

6. When the signature verification is completed in the bootloader, the firmware of the updated image will boot. It is normal if the following message is output and the LED is blinking.

```
==== RL78G22 : BootLoader [without buffer] ====
verify install area main [hash-sha256]...OK
execute new image ...

----------------------------------------
FWUP demo (ver 0.1.2)
----------------------------------------
Check the LEDs on the board.
```

4.6 How to debug the demo project

If you wish to debug this project (bootloader + application program) in the e2 studio environment, the following procedure can be used.

This demo project is set to be powered by the emulator in the debugger (E2 Lite). If you want to connect with other debuggers or supply power from the target board, change the debugger settings.

(1) Build the bootloader and application program without optimization.

Build the bootloader (boot_loader) and application program (fwup_main).

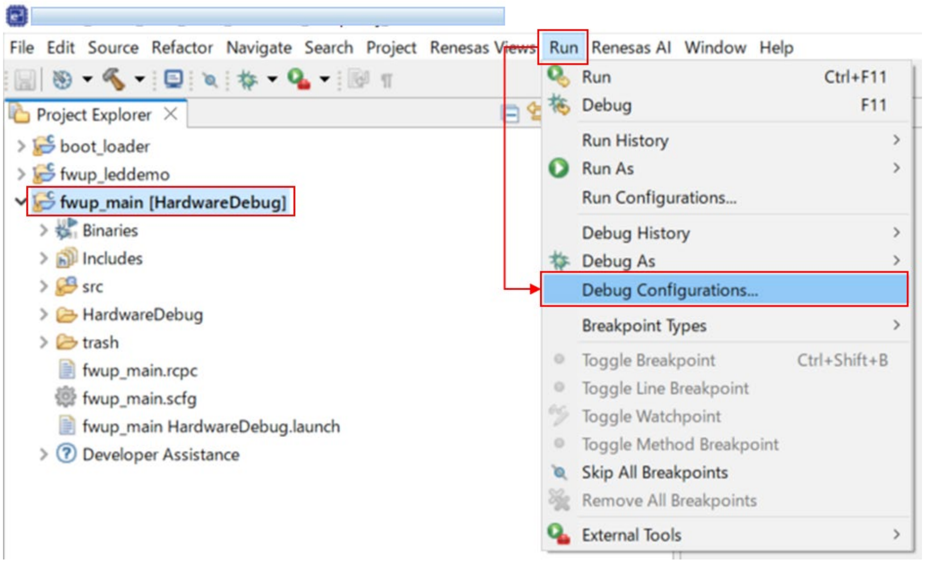
(2) Generate the initial image.

The Renesas Image Generator generates an initial image file (.mot) consisting of a bootloader (boot_loader) and an application program (fwup_main).

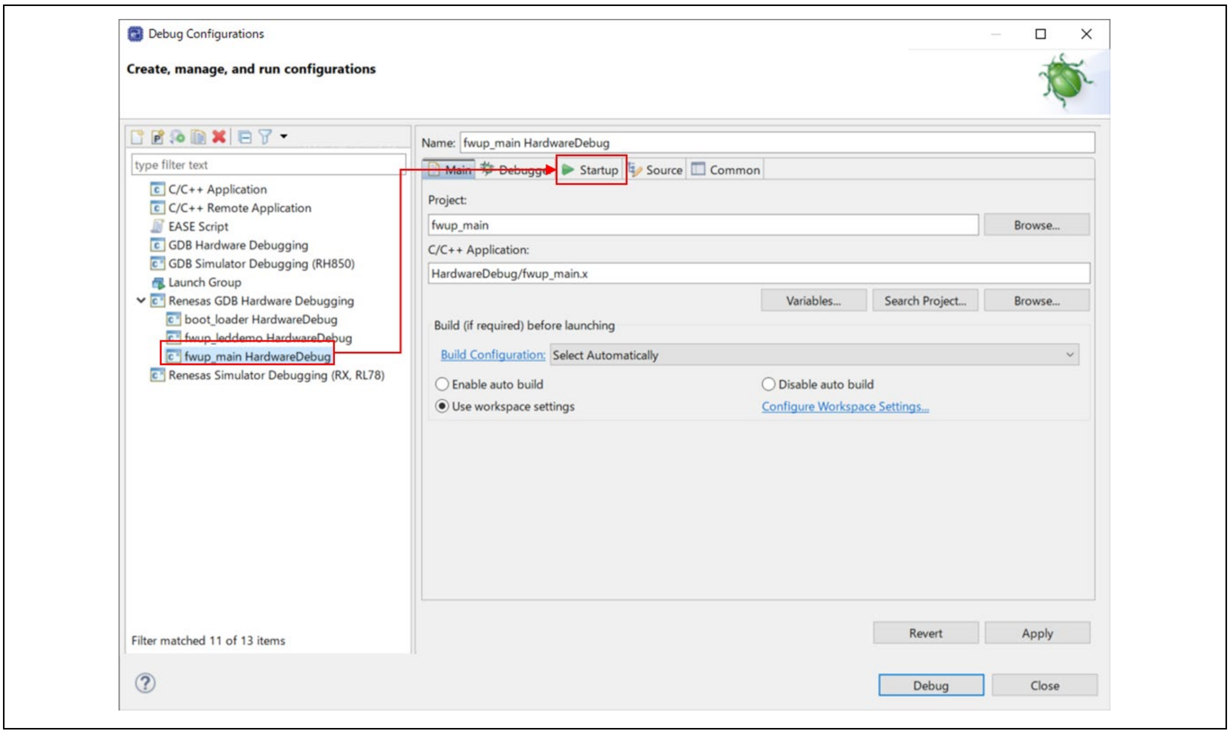
(3) Debug settings for the application program (fwup_main).

Follow the steps below to configure debugging settings for the application program (fwup_main).

a) Open Run->Debug Configuration and select fwup_main_HardwareDebug.



b) Select fwup_main_HardwareDebug and click Startup.



Debug Configurations

Create, manage, and run configurations

Name: fwup_main HardwareDebug

Project: fwup_main

C/C++ Application: HardwareDebug/fwup_main.x

Build (if required) before launching

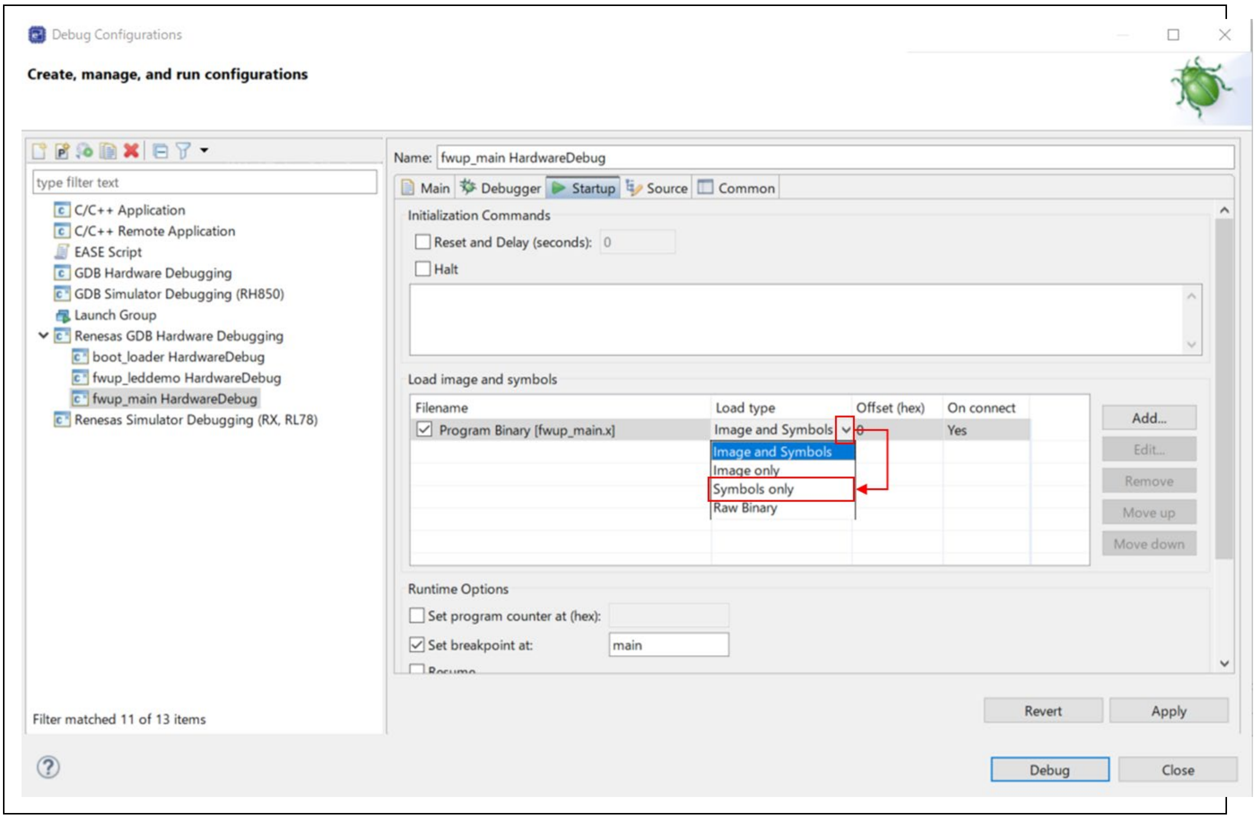
Build Configuration: Select Automatically

○ Enable auto build ○ Disable auto build

● Use workspace settings

Filter matched 11 of 13 items

c) Change the load type of the program binary [fwup_main.x] from "Image and Symbol" to "Symbol Only".



Debug Configurations

Create, manage, and run configurations

Name: fwup_main HardwareDebug

Initialization Commands

☐ Reset and Delay (seconds): 0

☐ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [fwup_main.x]	Image and Symbols	0	Yes

Runtime Options

☐ Set program counter at (hex):

☑ Set breakpoint at: main

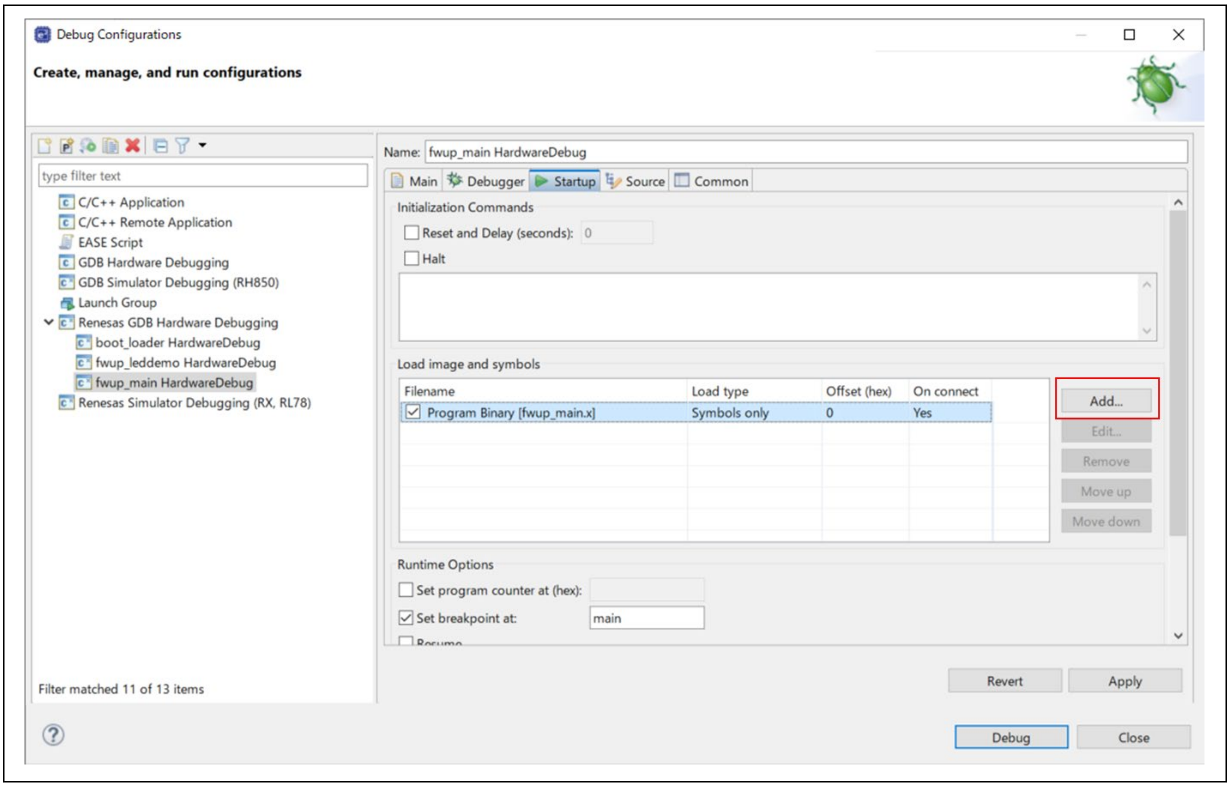
☐ Resume

Filter matched 11 of 13 items

(4) Add the bootloader (boot_loader) symbol.

Follow the procedure below to add the boot loader (boot_loader) symbol built in step (1).

a) Click "Add".



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB Simulator Debugging (RH850)
- Launch Group
- Renesas GDB Hardware Debugging
  - boot_loader HardwareDebug
  - fwup_leddemo HardwareDebug
  - fwup_main HardwareDebug
- Renesas Simulator Debugging (RX, RL78)

Filter matched 11 of 13 items

Name: fwup_main HardwareDebug

Initialization Commands

□ Reset and Delay (seconds): 0

□ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [fwup_main.x]	Symbols only	0	Yes

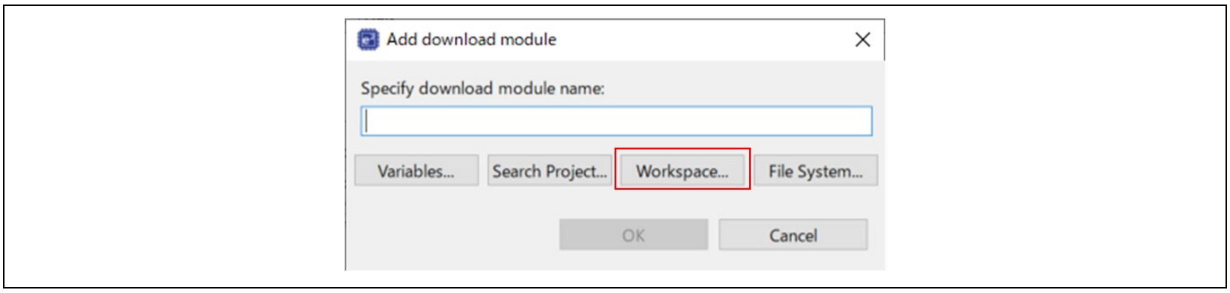
Runtime Options

□ Set program counter at (hex):

☑ Set breakpoint at: main

□ Resume

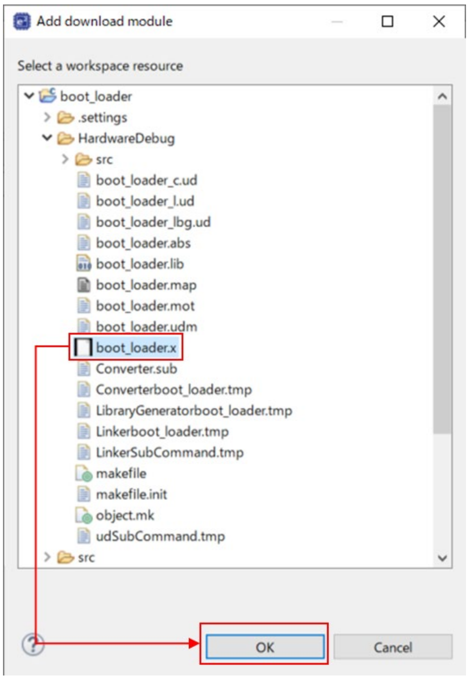
b) Click Workspace.



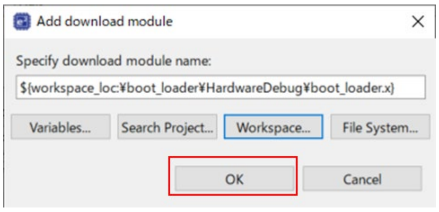
Add download module

Specify download module name:

c) Select the bootloader (boot_loader.x) and click "OK.".

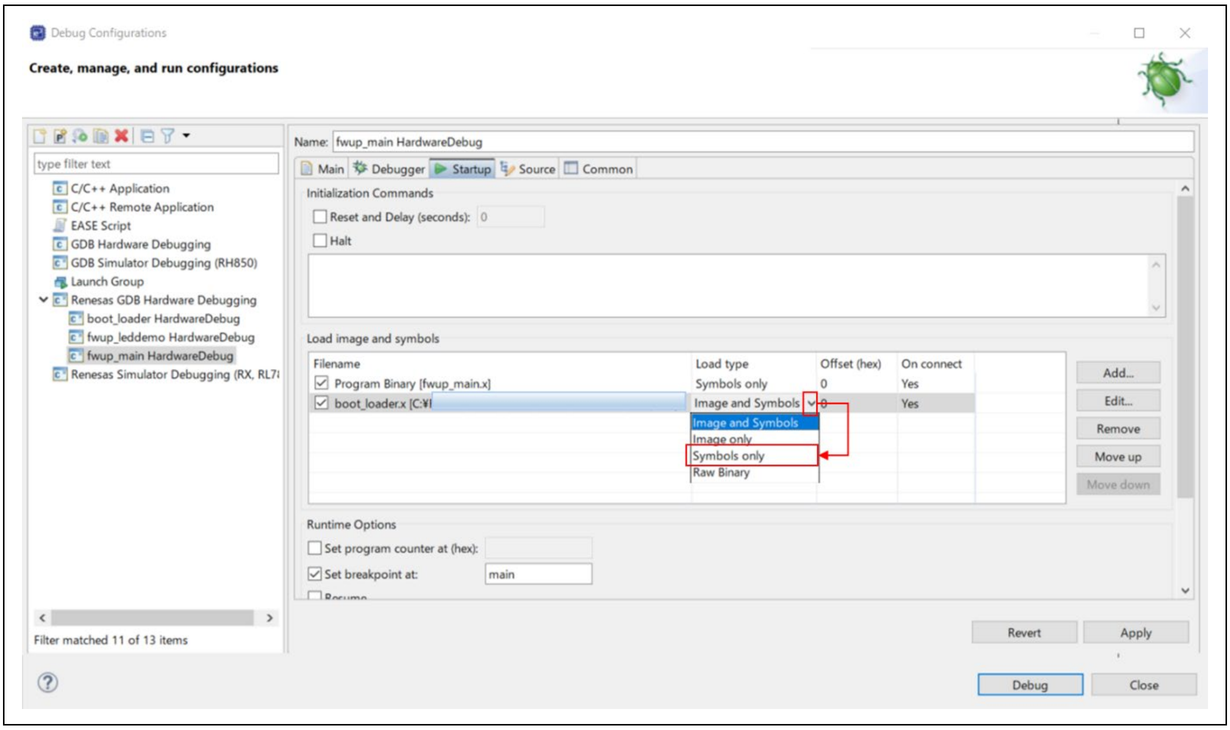


d) Confirm that the download module name is set to bootloader (boot_loader.x) and click "OK.



Specify download module name:
${workspace_loc:¥boot_loader¥HardwareDebug¥boot_loader.x}

e) Change the load type of the bootloader (boot_loader.x) from "Image and Symbol" to "Symbol only".



Name: fwup_main HardwareDebug

Initialization Commands

☐ Reset and Delay (seconds): 0

☐ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [fwup_main.x]	Symbols only	0	Yes
☑ boot_loader.x [C:¥\	Image and Symbols	0	Yes

Runtime Options

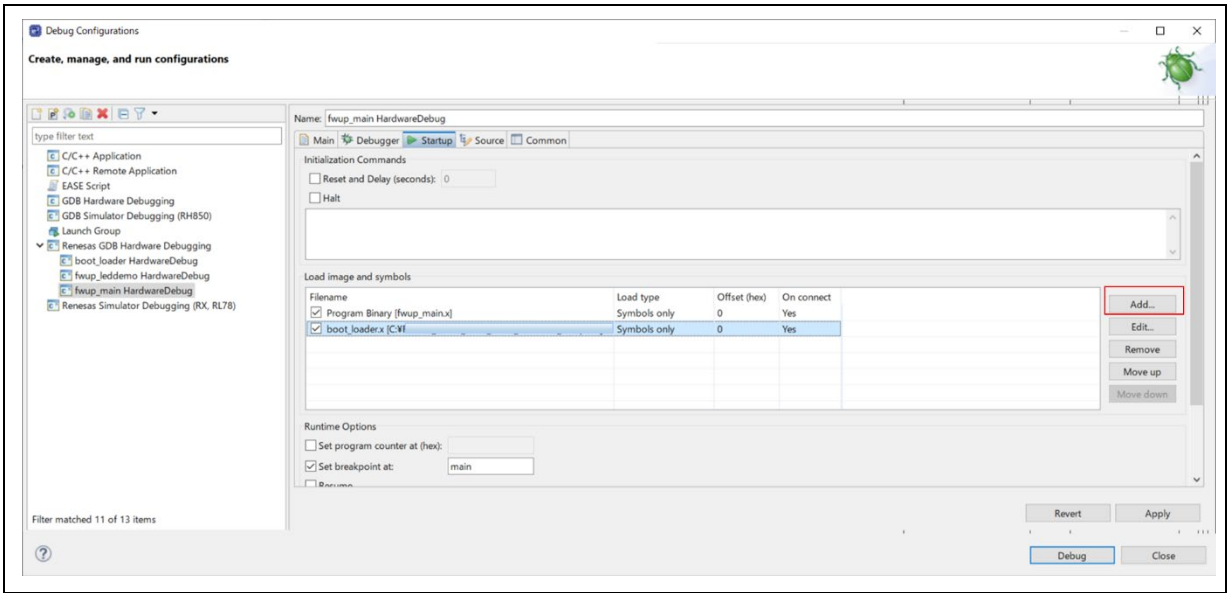
☐ Set program counter at (hex):

☑ Set breakpoint at: main

(5) Add the image of the initial image (initial_firm.mot).

Add the image of the initial image (initial_firm.mot) generated in step (2) according to the following procedure.

a) Click "Add".



Name: fwup_main HardwareDebug

Initialization Commands

☐ Reset and Delay (seconds): 0

☐ Halt

Load image and symbols

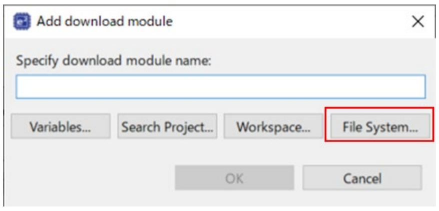
Filename	Load type	Offset (hex)	On connect
☑ Program Binary [fwup_main.x]	Symbols only	0	Yes
☑ boot_loader.x [C:¥\	Symbols only	0	Yes

Runtime Options

☐ Set program counter at (hex):

☑ Set breakpoint at: main

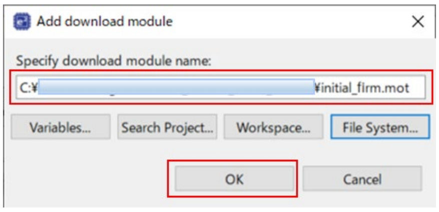
b) Click File System.



Add download module

Specify download module name:

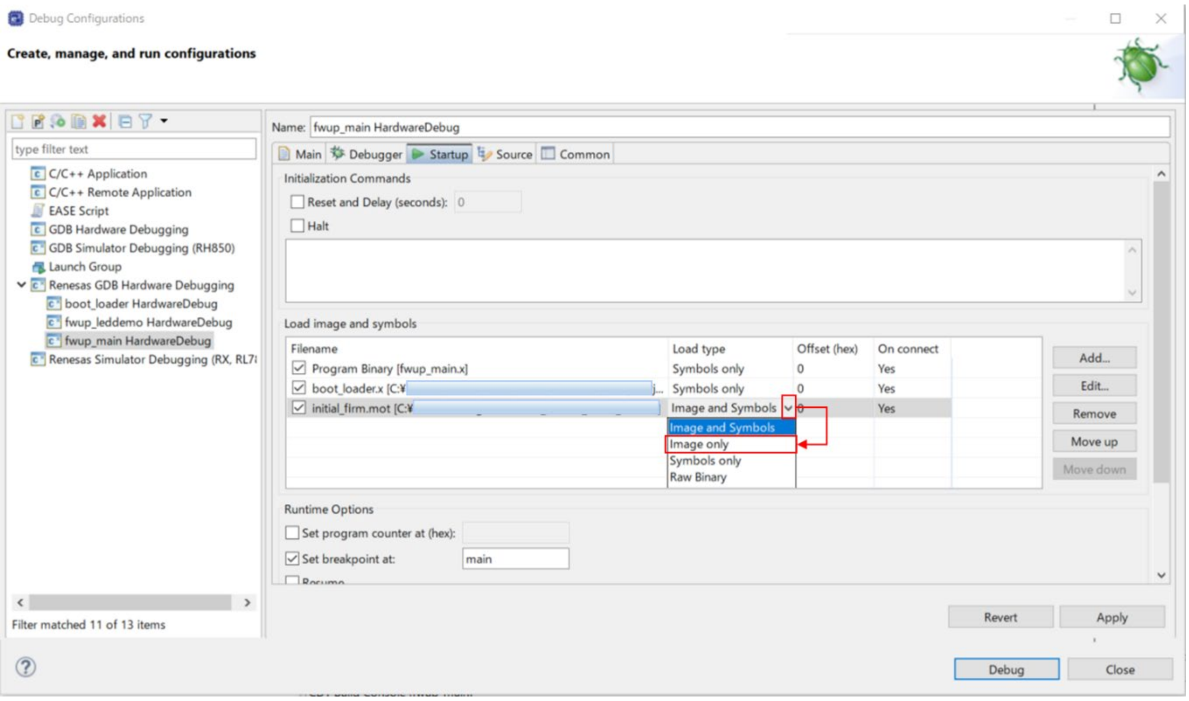
c) Select the initial image (initial_firm.mot) and click "OK”.



Add download module

Specify download module name: C:¥[redacted]¥initial_firm.mot

d) Change the load type of the initial image (initial_firm.mot) from "Image and Symbol" to "Image only" and click "OK”.



Debug Configurations

Create, manage, and run configurations

Name: fwup_main HardwareDebug

Initialization Commands

□ Reset and Delay (seconds): 0

□ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [fwup_main.x]	Symbols only	0	Yes
☑ boot_loader.x [C:¥[redacted]	Symbols only	0	Yes
☑ initial_firm.mot [C:¥[redacted]	Image and Symbols	0	Yes

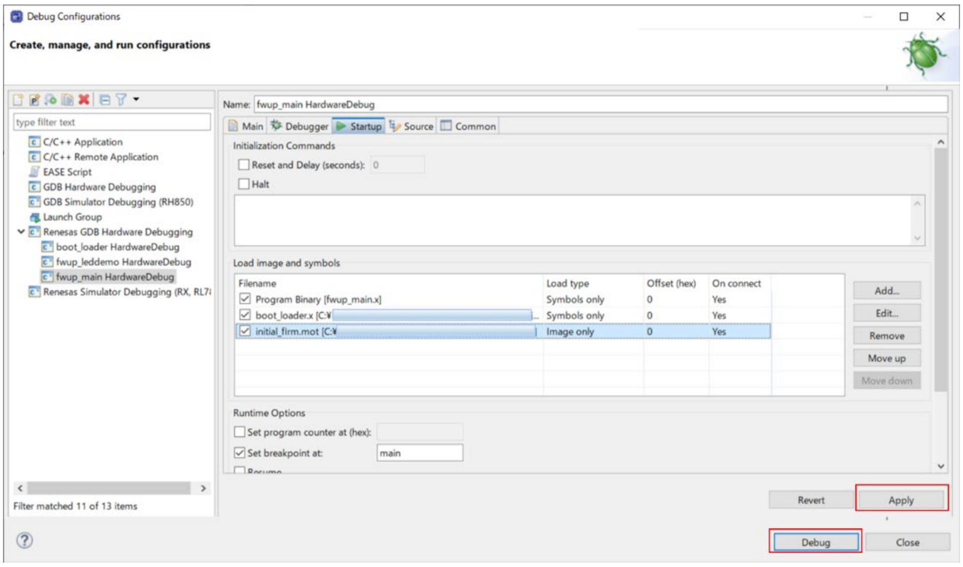
Runtime Options

□ Set program counter at (hex):

☑ Set breakpoint at: main

Filter matched 11 of 13 items

e) Click "Apply" and click "Debug".



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB Simulator Debugging (RH850)
- Launch Group
- Renesas GDB Hardware Debugging
  - boot_loader HardwareDebug
  - fwup_leddemo HardwareDebug
  - fwup_main HardwareDebug
- Renesas Simulator Debugging (RX, RL7

Filter matched 11 of 13 items

Name: fwup_main HardwareDebug

Main | Debugger | Startup | Source | Common

Initialization Commands

☐ Reset and Delay (seconds): 0

☐ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [fwup_main.x]	Symbols only	0	Yes
☑ boot_loader.x [C:¥	Symbols only	0	Yes
☑ initial_firm.mot [C:¥	Image only	0	Yes

Runtime Options

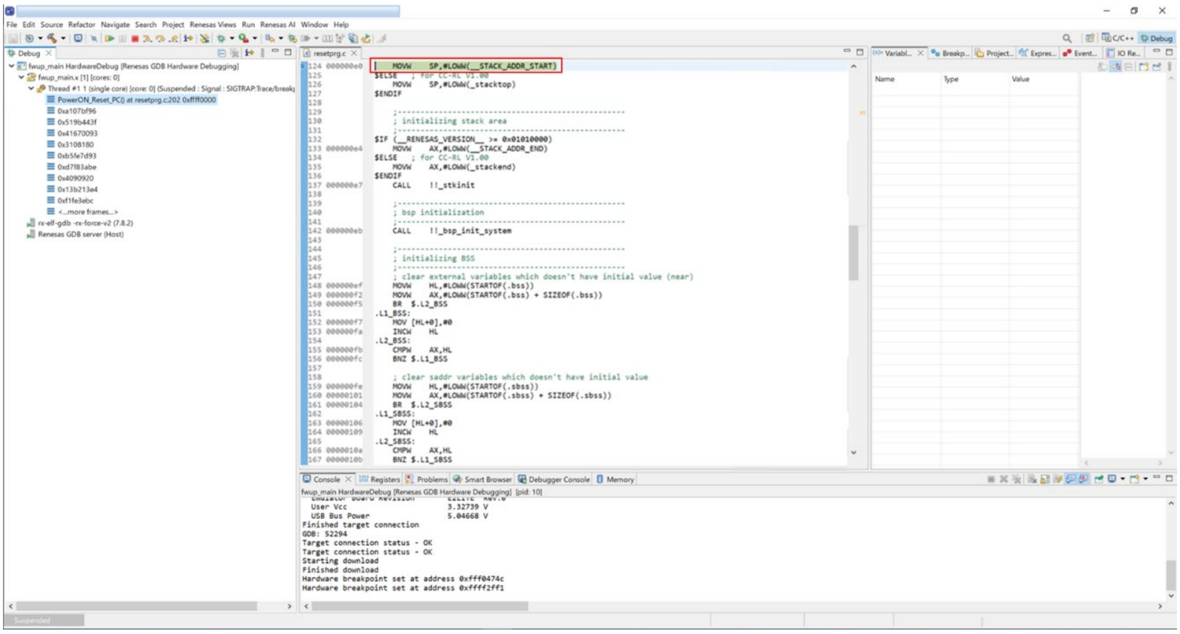
☐ Set program counter at (hex):

☑ Set breakpoint at: main

☐ Resume

(6) Start the debugger.

When the debugger starts, it jumps to cstart.asm in the boot_loader project.



File Edit Source Refactor Navigate Search Project Renesas Views Run Renesas AI Window Help
Debug
resetprg.c
Variabl... Breakp... Project... Expres... Event... IO Re...
Name Type Value
Console Registers Problems Smart Browser Debugger Console Memory
Suspended

- fwup_main HardwareDebug [Renesas GDB Hardware Debugging]
  - fwup_main.x [1] [cores: 0]
    - Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:Trace/breakq
      - PowerON_Reset_PC() at resetprg.c:202 0xffff0000
      - 0xa107bf96
      - 0x519b443f
      - 0x41670093
      - 0x3108180
      - 0xb5fe7d93
      - 0xd7f83abe
      - 0x4090920
      - 0x13b213e4
      - 0xf1fe3ebc
      - <...more frames...>
  - rx-elf-gdb -rx-force-v2 (7.8.2)
  - Renesas GDB server (Host)

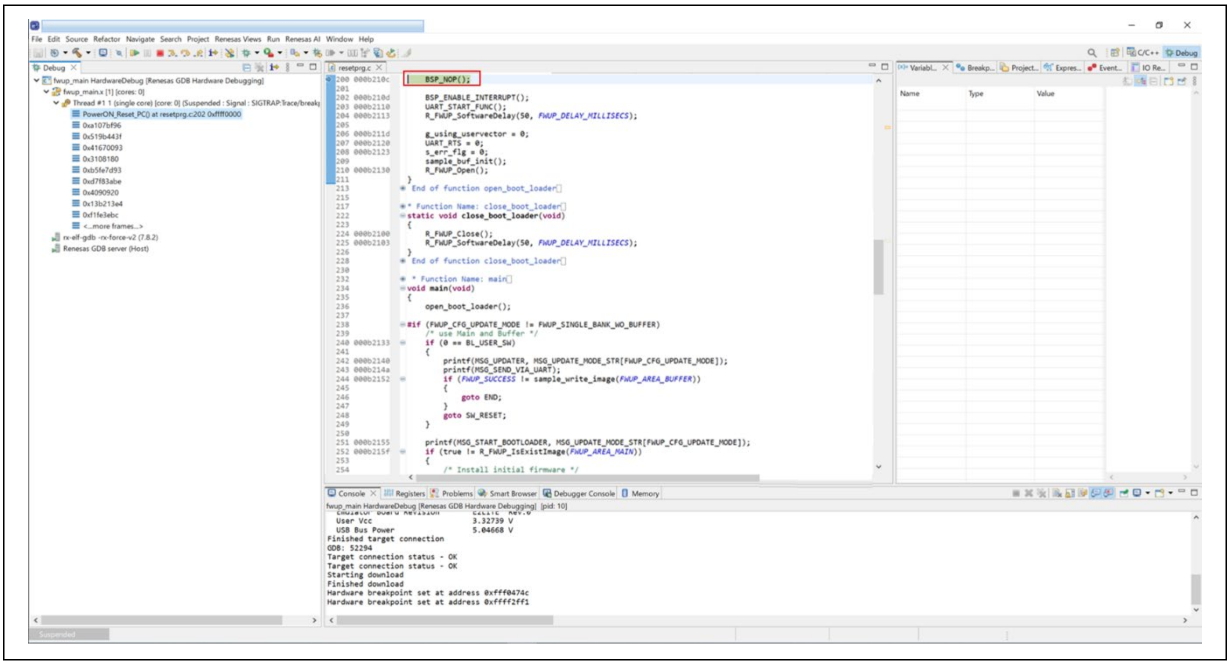
```
000000e0     MOVW    SP,#LOWW(__STACK_ADDR_START)
         $ELSE   ; for CC-RL V1.00
             MOVW    SP,#LOWW(_stacktop)
         $ENDIF

             ;--------------------------------------------------
             ; initializing stack area
             ;--------------------------------------------------
         $IF (__RENESAS_VERSION__ >= 0x01010000)
000000e4     MOVW    AX,#LOWW(__STACK_ADDR_END)
         $ELSE   ; for CC-RL V1.00
             MOVW    AX,#LOWW(_stackend)
         $ENDIF
000000e7     CALL    !!_stkinit

             ;--------------------------------------------------
             ; bsp initialization
             ;--------------------------------------------------
000000eb     CALL    !!_bsp_init_system

             ;--------------------------------------------------
             ; initializing BSS
             ;--------------------------------------------------
             ; clear external variables which doesn't have initial value (near)
000000ef     MOVW    HL,#LOWW(STARTOF(.bss))
000000f2     MOVW    AX,#LOWW(STARTOF(.bss) + SIZEOF(.bss))
000000f5     BR  $.L2_BSS
         .L1_BSS:
000000f7     MOV [HL+0],#0
000000fa     INCW    HL
         .L2_BSS:
000000fb     CMPW    AX,HL
000000fc     BNZ $.L1_BSS

             ; clear saddr variables which doesn't have initial value
000000fe     MOVW    HL,#LOWW(STARTOF(.sbss))
00000101     MOVW    AX,#LOWW(STARTOF(.sbss) + SIZEOF(.sbss))
00000104     BR  $.L2_SBSS
         .L1_SBSS:
00000106     MOV [HL+0],#0
00000109     INCW    HL
         .L2_SBSS:
0000010a     CMPW    AX,HL
0000010b     BNZ $.L1_SBSS
```

fwup_main HardwareDebug [Renesas GDB Hardware Debugging] [pid: 10]

```
  Emulator Board Revision          E2Lite  Rev.0
  User Vcc                         3.32739 V
  USB Bus Power                    5.04668 V
Finished target connection
GDB: 52294
Target connection status - OK
Target connection status - OK
Starting download
Finished download
Hardware breakpoint set at address 0xfff0474c
Hardware breakpoint set at address 0xffff2ff1
```

(7) Resume the program.

When you click Resume, the program stops at the beginning of the main() function in boot_loader.c. project.


File Edit Source Refactor Navigate Search Project Renesas Views Run Renesas AI Window Help
Debug
resetprg.c
Console Registers Problems Smart Browser Debugger Console Memory
Name Type Value
Suspended

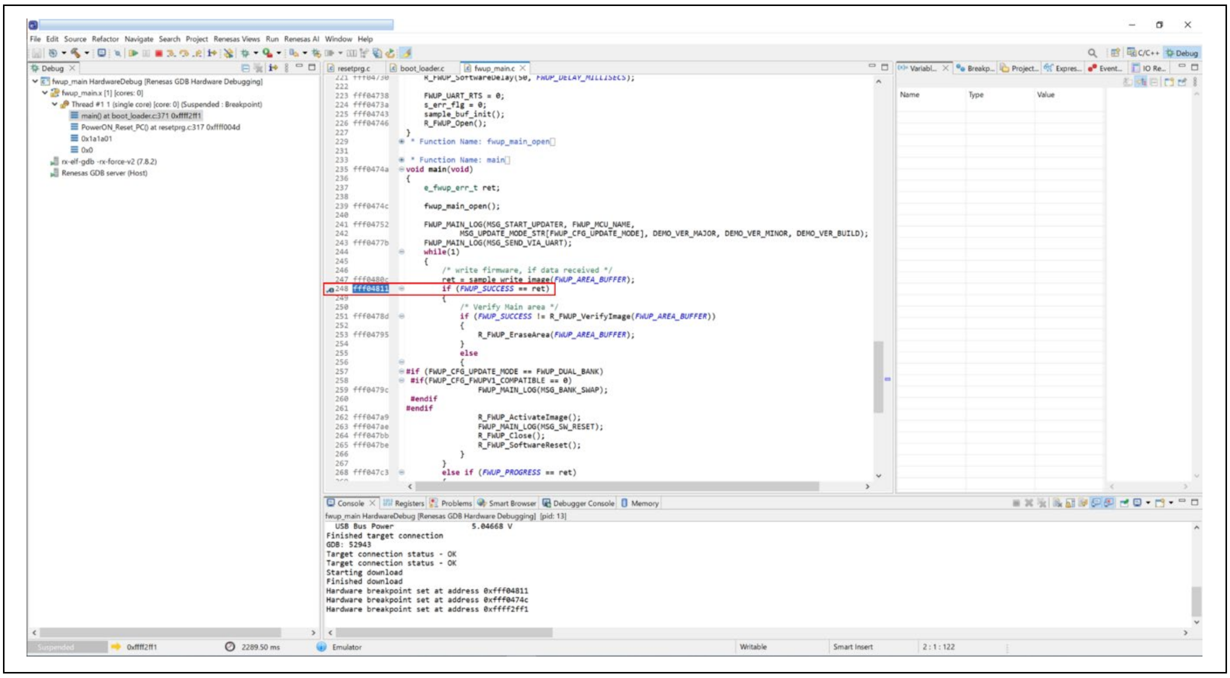
```
fwup_main HardwareDebug [Renesas GDB Hardware Debugging]
  fwup_main.x [1] [cores: 0]
    Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:Trace/breakp
      PowerON_Reset_PC() at resetprg.c:202 0xffff0000
      0xa107bf96
      0x519b443f
      0x41670093
      0x3108180
      0xb5fe7d93
      0xd7f83abe
      0x4090920
      0x13b213e4
      0xf1fe3ebc
      <...more frames...>
  rx-elf-gdb -rx-force-v2 (7.8.2)
  Renesas GDB server (Host)
```

```
000b210c      BSP_NOP();

000b210d      BSP_ENABLE_INTERRUPT();
000b2110      UART_START_FUNC();
000b2113      R_FWUP_SoftwareDelay(50, FWUP_DELAY_MILLISECS);

000b211d      g_using_uservector = 0;
000b2120      UART_RTS = 0;
000b2123      s_err_flg = 0;
              sample_buf_init();
000b2130      R_FWUP_Open();
          }
          End of function open_boot_loader

          * Function Name: close_boot_loader
          static void close_boot_loader(void)
          {
000b2100      R_FWUP_Close();
000b2103      R_FWUP_SoftwareDelay(50, FWUP_DELAY_MILLISECS);
          }
          End of function close_boot_loader

           * Function Name: main
          void main(void)
          {
              open_boot_loader();

          #if (FWUP_CFG_UPDATE_MODE != FWUP_SINGLE_BANK_WO_BUFFER)
              /* use Main and Buffer */
000b2133      if (0 == BL_USER_SW)
              {
000b2140          printf(MSG_UPDATER, MSG_UPDATE_MODE_STR[FWUP_CFG_UPDATE_MODE]);
000b214a          printf(MSG_SEND_VIA_UART);
000b2152          if (FWUP_SUCCESS != sample_write_image(FWUP_AREA_BUFFER))
                  {
                      goto END;
                  }
                  goto SW_RESET;
              }

000b2155      printf(MSG_START_BOOTLOADER, MSG_UPDATE_MODE_STR[FWUP_CFG_UPDATE_MODE]);
000b215f      if (true != R_FWUP_IsExistImage(FWUP_AREA_MAIN))
              {
                  /* Install initial firmware */
```

```
fwup_main HardwareDebug [Renesas GDB Hardware Debugging]  [pid: 10]
  Emulator Board Revision        E2LITE  Rev.0
  User Vcc                       3.32739 V
  USB Bus Power                  5.04668 V
Finished target connection
GDB: 52294
Target connection status - OK
Target connection status - OK
Starting download
Finished download
Hardware breakpoint set at address 0xfff0474c
Hardware breakpoint set at address 0xffff2ff1
```

(8) Set a breakpoint in main() of the fwup_main project.

Set a breakpoint in the following red frame in main() of the fwup_main project.


File Edit Source Refactor Navigate Search Project Renesas Views Run Renesas AI Window Help
Debug
resetprg.c
boot_loader.c
fwup_main.c
Console Registers Problems Smart Browser Debugger Console Memory
Name Type Value
Suspended
0xffff2ff1
2289.50 ms
Emulator
Writable
Smart Insert
2 : 1 : 122

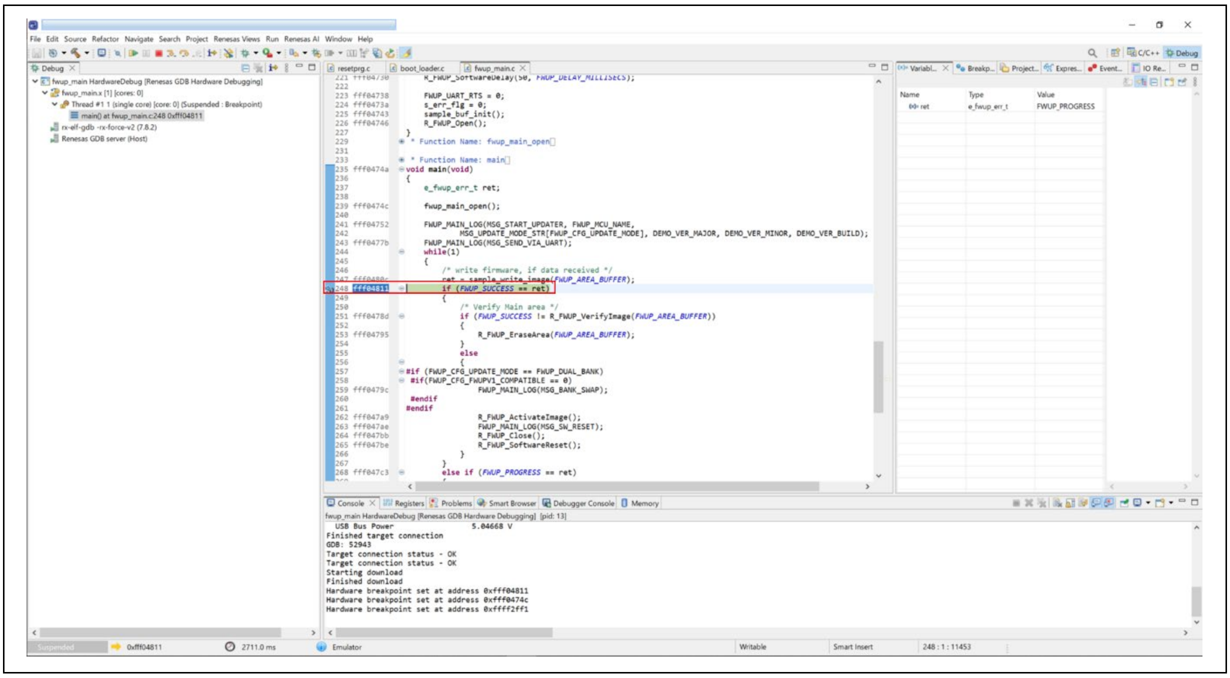
```
fwup_main HardwareDebug [Renesas GDB Hardware Debugging]
  fwup_main.x [1] [cores: 0]
    Thread #1 1 (single core) [core: 0] (Suspended : Breakpoint)
      main() at boot_loader.c:371 0xffff2ff1
      PowerON_Reset_PC() at resetprg.c:317 0xffff004d
      0x1a1a01
      0x0
  rx-elf-gdb -rx-force-v2 (7.8.2)
  Renesas GDB server (Host)
```

```
fff04730      R_FWUP_SoftwareDelay(50, FWUP_DELAY_MILLISECS);

fff04738      FWUP_UART_RTS = 0;
fff0473a      s_err_flg = 0;
fff04743      sample_buf_init();
fff04746      R_FWUP_Open();
          }
           * Function Name: fwup_main_open

           * Function Name: main
fff0474a  void main(void)
          {
              e_fwup_err_t ret;

fff0474c      fwup_main_open();

fff04752      FWUP_MAIN_LOG(MSG_START_UPDATER, FWUP_MCU_NAME,
                      MSG_UPDATE_MODE_STR[FWUP_CFG_UPDATE_MODE], DEMO_VER_MAJOR, DEMO_VER_MINOR, DEMO_VER_BUILD);
fff0477b      FWUP_MAIN_LOG(MSG_SEND_VIA_UART);
              while(1)
              {
                  /* write firmware, if data received */
fff0480c          ret = sample_write_image(FWUP_AREA_BUFFER);
fff04811          if (FWUP_SUCCESS == ret)
                  {
                      /* Verify Main area */
fff0478d              if (FWUP_SUCCESS != R_FWUP_VerifyImage(FWUP_AREA_BUFFER))
                      {
fff04795                  R_FWUP_EraseArea(FWUP_AREA_BUFFER);
                      }
                      else
                      {
          #if (FWUP_CFG_UPDATE_MODE == FWUP_DUAL_BANK)
           #if(FWUP_CFG_FWUPV1_COMPATIBLE == 0)
fff0479c                  FWUP_MAIN_LOG(MSG_BANK_SWAP);
           #endif
          #endif
fff047a9                  R_FWUP_ActivateImage();
fff047ae                  FWUP_MAIN_LOG(MSG_SW_RESET);
fff047bb                  R_FWUP_Close();
fff047be                  R_FWUP_SoftwareReset();
                      }
                  }
fff047c3          else if (FWUP_PROGRESS == ret)
```

```
fwup_main HardwareDebug [Renesas GDB Hardware Debugging]  [pid: 13]
  USB Bus Power                  5.04668 V
Finished target connection
GDB: 52943
Target connection status - OK
Target connection status - OK
Starting download
Finished download
Hardware breakpoint set at address 0xfff04811
Hardware breakpoint set at address 0xfff0474c
Hardware breakpoint set at address 0xffff2ff1
```

(9) Resume the program.

Click restart and stop at the breakpoint set in (8).



```
fwup_main HardwareDebug [Renesas GDB Hardware Debugging]
  fwup_main.x [1] [cores: 0]
    Thread #1 1 (single core) [core: 0] (Suspended : Breakpoint)
      main() at fwup_main.c:248 0xfff04811
  rx-elf-gdb -rx-force-v2 (7.8.2)
  Renesas GDB server (Host)
```

```
fff04730          R_FWUP_SoftwareDelay(50, FWUP_DELAY_MILLISECS);

fff04738          FWUP_UART_RTS = 0;
fff0473a          s_err_flg = 0;
fff04743          sample_buf_init();
fff04746          R_FWUP_Open();
              }
               * Function Name: fwup_main_open

               * Function Name: main
fff0474a      void main(void)
              {
                  e_fwup_err_t ret;

fff0474c          fwup_main_open();

fff04752          FWUP_MAIN_LOG(MSG_START_UPDATER, FWUP_MCU_NAME,
                          MSG_UPDATE_MODE_STR[FWUP_CFG_UPDATE_MODE], DEMO_VER_MAJOR, DEMO_VER_MINOR, DEMO_VER_BUILD);
fff0477b          FWUP_MAIN_LOG(MSG_SEND_VIA_UART);
                  while(1)
                  {
                      /* write firmware, if data received */
fff0480c              ret = sample_write_image(FWUP_AREA_BUFFER);
fff04811              if (FWUP_SUCCESS == ret)
                      {
                          /* Verify Main area */
fff0478d                  if (FWUP_SUCCESS != R_FWUP_VerifyImage(FWUP_AREA_BUFFER))
                          {
fff04795                      R_FWUP_EraseArea(FWUP_AREA_BUFFER);
                          }
                          else
                          {
              #if (FWUP_CFG_UPDATE_MODE == FWUP_DUAL_BANK)
               #if(FWUP_CFG_FWUPV1_COMPATIBLE == 0)
fff0479c                      FWUP_MAIN_LOG(MSG_BANK_SWAP);
               #endif
              #endif
fff047a9                      R_FWUP_ActivateImage();
fff047ae                      FWUP_MAIN_LOG(MSG_SW_RESET);
fff047bb                      R_FWUP_Close();
fff047be                      R_FWUP_SoftwareReset();
                          }
                      }
fff047c3              else if (FWUP_PROGRESS == ret)
```

Name	Type	Value
ret	e_fwup_err_t	FWUP_PROGRESS

fwup_main HardwareDebug [Renesas GDB Hardware Debugging] [pid: 13]

```
  USB Bus Power                5.04668 V
Finished target connection
GDB: 52943
Target connection status - OK
Target connection status - OK
Starting download
Finished download
Hardware breakpoint set at address 0xfff04811
Hardware breakpoint set at address 0xfff0474c
Hardware breakpoint set at address 0xffff2ff1
```

5. Renesas Image Generator

Renesas Image Generator is a utility tool that generates firmware images for use with firmware update modules. The Renesas Image Generator can generate the following images used by the firmware update module.

- Initial image: An image file containing the bootloader and application program that is programmed using Flash Writer at the time of initial system configuration (extension: mot).
- Update image: An image file containing the firmware update (extension: rsu).

See 5.1 for how to generate an image, and 5.2 to 5.3 for details on image configuration and parameter files.

Renesas Image Generator is a program that runs on Python.

5.1 Image Generation Methods

Describes the specifications of Renesas Image Generator (image-gen.py) and how to generate an image file (initial image or update image) using this tool.

See 5.1.1 for how to generate an initial image, and 5.1.2 for how to generate an update image.

The format of the image-gen.py command is as follows:

```
  python image-gen.py < options >
```

Some image-gen.py command options are required and others are optional. Table 5.1 lists the required image-gen.py options, and Table 5.2 lists the optional image-gen.py options.

Table 5.1 Required Options of image-gen.py

Option	Description
-iup <file>	Specifies the application program. For the character string < file >, specify the mot file name (the full path including the file name) of the user application program.
-ip <file>	Specifies a parameter file. For the character string < file >, specify the name of the file (the full path including the file name) containing the parameters to be input.
-o <file>	Specifies the file name of the output image. For the character string < file >, specify the file name (the full path including the file name), excluding the extension, of the firmware update image file to be output. The file extension is .mot because the output image is determined to be the initial image when the bootloader is specified with the -ibp <file> option. If you omit the -ibp <file> specification, the output image is determined to be an update image and becomes .rsu.

Table 5.2 Optional Options of image-gen.py

Option	Description
-ibp <file>	Specifies the bootloader. For the character string < file >, specify mot file name (the full path including the file name) of the bootloader program. Specify this option when generating a mot file.
--key <file>	Specify the name of the key file to be used to sign the image using ECDSA. (This option does not need to be set if sha256 is specified for the -vt option.) Store the file **secp256r1.privatekey** in the command execution folder. If the file name has been changed, specify the full path including the file name.
-vt <VerificationType>[sha256 / ecdsa]	Specifies the image verification method in the firmware update module. The following VerificationType can be specified. sha256: Append a hash of the image. If this option is omitted, "sha256" is specified. ecdsa: Adds an image signature. The key file specified by -key is used to generate signature data. An error will result if the key file is not specified with -key.
-ff <FileFormat>	Specifies the RSU format type. The following FileFormat can be specified BareMetal: Generates an image of the application program data with RSU header signature information. This is the RSU format used in the demo project. If this option is omitted, "BareMetal" is specified. RTOS: Generates an updated image for FreeRTOS OTA. Update images for FreeRTOS OTA do not add RSU header signature information. BareMetal_FWUP_V2_V1_DATA: For special purpose. RTOS_FWUP_V2_V1_DATA: For special purpose.
-h	Output a list of commands. Specify this option to display help information for the tool.

5.1.1 Initial Image Generation Method

Renesas Image Generator has the bootloader file name (.mot) generated by build, application program (.mot), parameter file name (.csv), output file name (no extension), image verification method in firmware update module. Specify (ecdsa/sha256) as a command line option to generate an initial image file (.mot).

Command input example

```
> python image-gen.py -iup fwup_main.mot -ip
RL78_G23_ImageGenerator_PRM.csv -o initial_firm -ibp boot_loader.mot
-vt ecdsa -key secp256r1.privatekey
```

fwup_main.mot: The mot file name of the user application program
RL78_G23_ImageGenerator_PRM.csv: The name of the file containing the parameters to be input
initial_firm: The file name of the initial image file to be output
boot_loader.mot: The mot file name of the bootloader program
ecdsa: Specifies that ECDSA is used to sign the image.
secp256r1.privatekey: Key file name for signing images with ECDSA.

5.1.2 Update Image Generation Method

The Renesas Image Generator uses the update application program (.mot) generated by the build, parameter file name (.csv), output file name (no extension), image verification method (ecdsa/sha256) for the firmware update module. Set the command line options to generate an update image file (.rsu).

Command input example

```
> python image-gen.py -iup fwup_leddemo.mot -ip
RL78_G23_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa -key
secp256r1.privatekey
```

fwup_leddemo.mot: The mot file name of the user application program to be applied as an update
RL78_G23_ImageGenerator_PRM.csv: The name of the file containing the parameters to be input
fwup_leddemo: The file name of the update image file to be output
ecdsa: Specifies that ECDSA is used to sign the image.
secp256r1.privatekey: Key file name for signing images with ECDSA.

5.2 Image File

5.2.1 Update Image File

Figure 5.1 shows the configuration diagram of the update image file generated by Renesas Image Generator.

For the format of the RSU header, see Table 5.3.

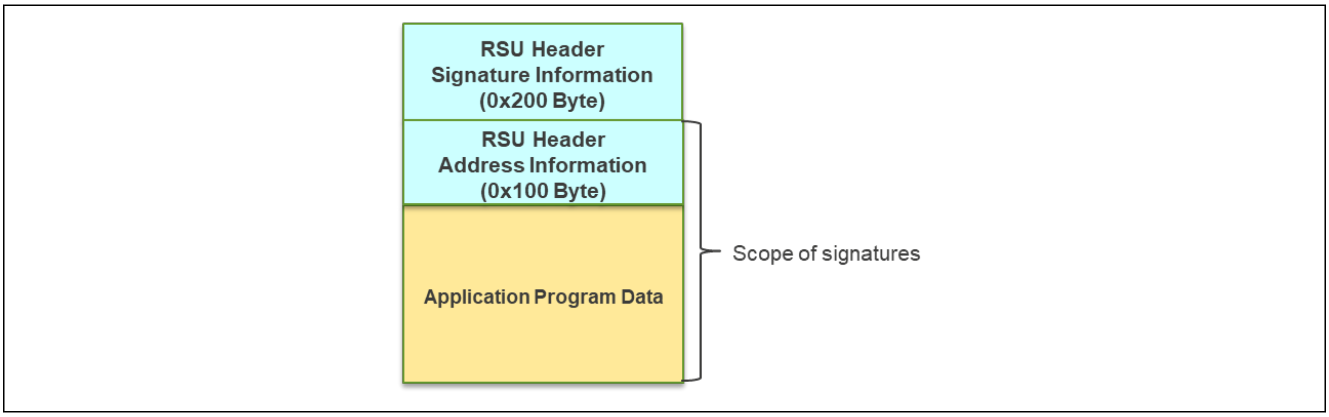


Figure 5.1 Configuring the update image file

The update image file consists of RSU header and application program data. The RSU header stores the application program location information required to verify the validity of the application program, as well as the signature value and hash value of the application program calculated based on the information. Following the RSU header, place the application program data corresponding to the program allocation information stored in the RSU header. The Renesas Image Generator arranges the application program data in the order of the data to be placed in the code flash and the data to be placed in the data flash. Valid code flash data and data flash data are extracted from the user-generated application program file (.mot), converted to binary data, and set.

The update image file has the same configuration for partial-updating method and full-updating method.

Table 5.3 RSU Header Format (1/2)

Offset	Item	Length (Bytes)	Description
0x00000000	Magic Code	7	Magic code (“RELFWV2”)
0x00000007	Reserved	1	Reserved area
0x00000008	Firmware Verification Type	32	Image verification method Set sig-sha256-ecdsa to use ECDSA for image verification, and hash-sha256 to use hash.
0x00000028	Signature size	4	Data size of signature value or hash value stored in Signature Set 0x40 if Firmware Verification Type is sig-sha256-ecdsa, and 0x20 if hash-sha256.
0x0000002C	Signature	64	Signature value used for firmware verification For SHA-256 signature data, bytes 33 to 64 are set to 0x00.
0x0000006C	RSU File Size	4	File size of entire update image file
0x00000070	Reserved	400	Reserved area

Table 5.3 RSU Header Format (2/2)

Offset	Item	Length (Bytes)	Description
0x00000200	Program Data Num	4	Number of subsequent divided application programs or data flashes (maximum 31)
0x00000204	Start Address[0]	4	Start address of the first application program or data flash
0x00000208	Data Size[0]	4	Size of the first application program or data flash
0x0000020C	Start Address[1]	4	Start address of second application program or data flash
0x00000210	Data Size[1]	4	Second application program or data flash size
:	:		
0x000002F4	Start Address[30]	4	Start address of the 31st application program or data flash
0x000002F8	Data Size[30]	4	Size of the 31st application program or data flash
0x000002FC	Reserved	4	Reserved area

See Figure 5.2 for the mechanism of generating the update image file.

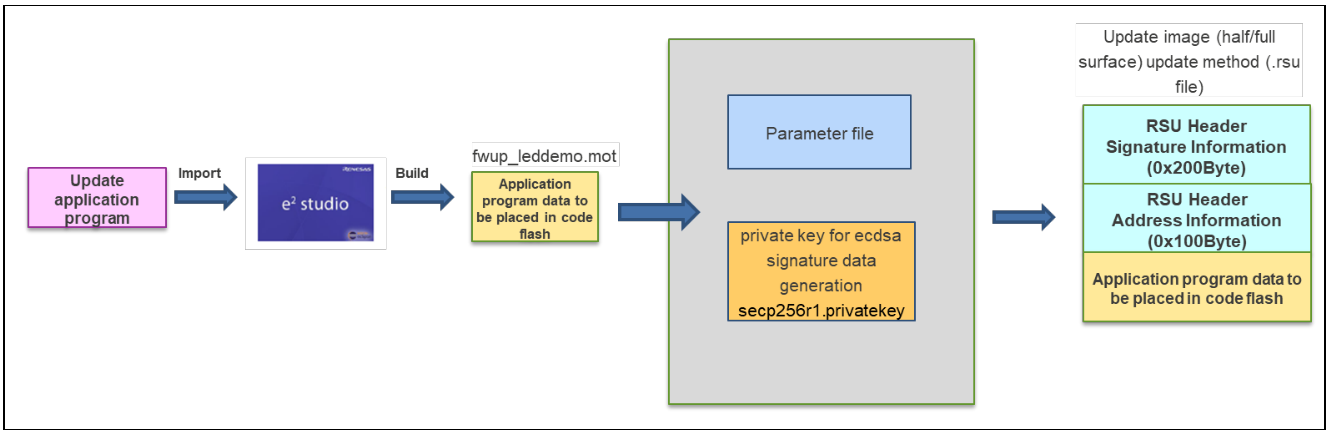

Update application program
Import
e² studio
Build
fwup_leddemo.mot
Application program data to be placed in code flash
Parameter file
private key for ecdsa signature data generation
secp256r1.privatekey
Update image (half/full surface) update method (.rsu file)
RSU Header Signature Information (0x200Byte)
RSU Header Address Information (0x100Byte)
Application program data to be placed in code flash

Figure 5.2 Updating image of partial / full updating method

- The parameter file is a CSV format file that contains the device address information required to generate the image file.
- The private key for generating the ecdsa signature value is used when ecdsa is specified as the image verification method in the firmware update module.

5.2.2 Initial Image File

The initial image file is the RSU header and application program data plus the bootloader program data.

Figure 5.3 also show a diagram of the initial image file (partial / full update method).

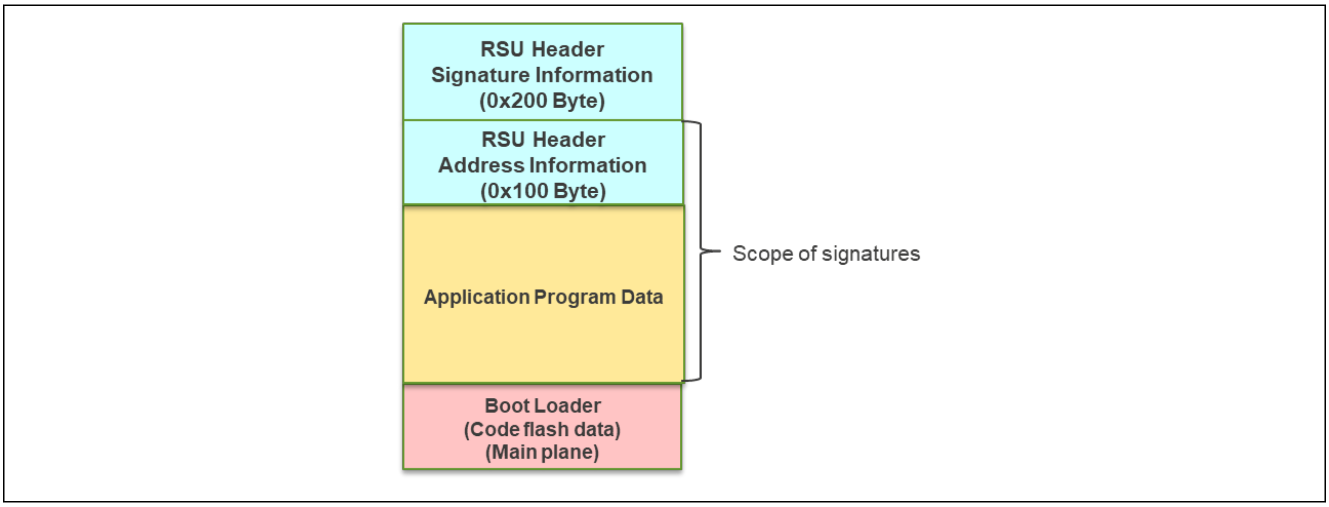


Figure 5.3 Composition of initial image file (partial / full update method)

In the initial image file of partial / full update method, the bootloader data to be placed on the main plane of the code flash uses the data in the user-generated bootloader file (boot_loader.mot) as is.

See Figure 5.4 for the mechanism that generates the initial image file.

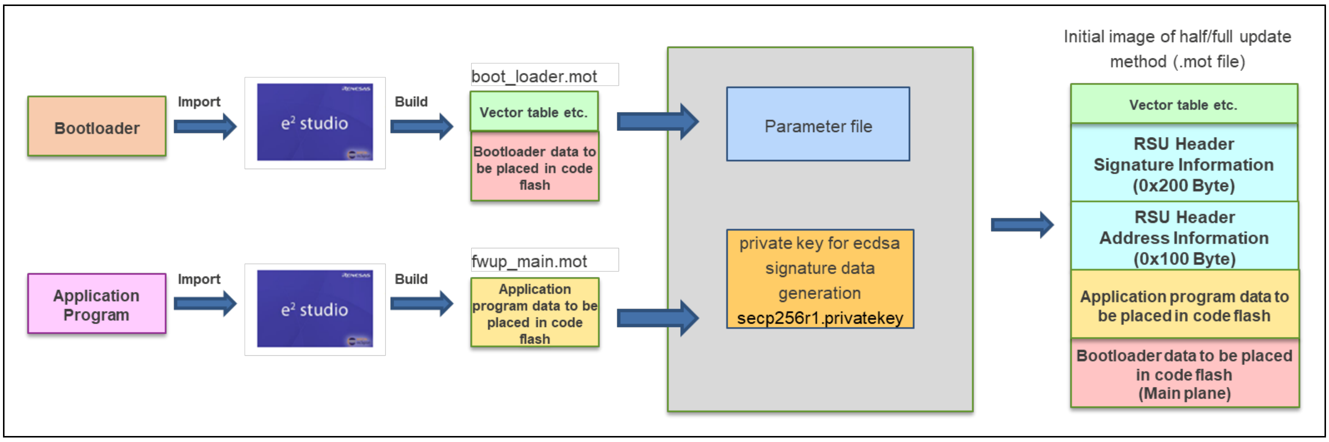

Bootloader
Import
e² studio
Build
boot_loader.mot
Vector table etc.
Bootloader data to be placed in code flash
Application Program
Import
e² studio
Build
fwup_main.mot
Application program data to be placed in code flash
Parameter file
private key for ecdsa signature data generation
secp256r1.privatekey
Initial image of half/full update method (.mot file)
Vector table etc.
RSU Header
Signature Information
(0x200 Byte)
RSU Header
Address Information
(0x100 Byte)
Application program data to be placed in code flash
Bootloader data to be placed in code flash
(Main plane)

Figure 5.4 Initial image of partial/ full update method

- The parameter file is a CSV format file that contains the device address information required to generate the image file.
- The private key for generating the ecdsa signature value is used when ecdsa is specified as the image verification method in the firmware update module.

5.3 Parameter File

The parameter file is the information required for Renesas Image Generator to generate the initial and updated image files for the sample program, and is included in the release package as part of the Renesas Image Generator Python It is included in the release package as part of the Renesas Image Generator Python program set. When a customer generates an initial or updated image for a demo project, there is no need to change the contents of the parameter file.

5.3.1 Contents of Parameter File

The items listed in the parameter file are the same for all devices, but the settings differ for each device. Table 5.4 shows the contents of the parameter file for the RL78/G23 demo project.

Figure 5.5 shows the parameters referenced when generating images for the RL78/G23 partial update method, and Figure 5.6 shows an example of parameters referenced when generating initial images for the RL78/G23 partial update method.

Table 5.4 Contents of parameter file

Parameter name	**Description**	**Example of setting contents RL78/G23**
device Type	Linear Mode：partial / full update method Mot file generation for	Liner Mode
Code Flash Size(Dual Mode Only)	Code Flash Size (For RL78, set 'No Used.')	No Uset
Bootloader Start Address	Bootloader start address	0x000B1000
Bootloader End Address	Bootloader end address	0x000BFFFF
User Program Start Address	Starting address of the application program on the main face	0x00001000
User Program End Address	End address of the application program on the main plane (in dual mode, application program area on main plane)	0x00058FFF
OFS Data Start Address	OFSM data start address (For RL78, set 'No Used.')	No Used.
OFS Data End Address	OFSM data end address (For RL78, set 'No Used.')	No Used.
Data Flash Start Address	Data flush start address (Set 'No Used.' if data flush data is not to be generated)	0x000F1000
Data Flash End Address	Data flash end address (Set 'No Used.' if data flash data is not to be generated)	0x000F2FFF
Near Data Start Address(RL78 Only)	Near bootloader start address for RL78	0x00000000
Near Data End Address(RL78 Only)	Near boot loader start address for RL78	0x00000FFF
Flash Write Size	Flash write size (number of bytes required for one write to the flash in decimal)	128

The value specified for each parameter is specified in decimal for Flash Write Size and in hexadecimal (with 0x added at the beginning) for other parameters.

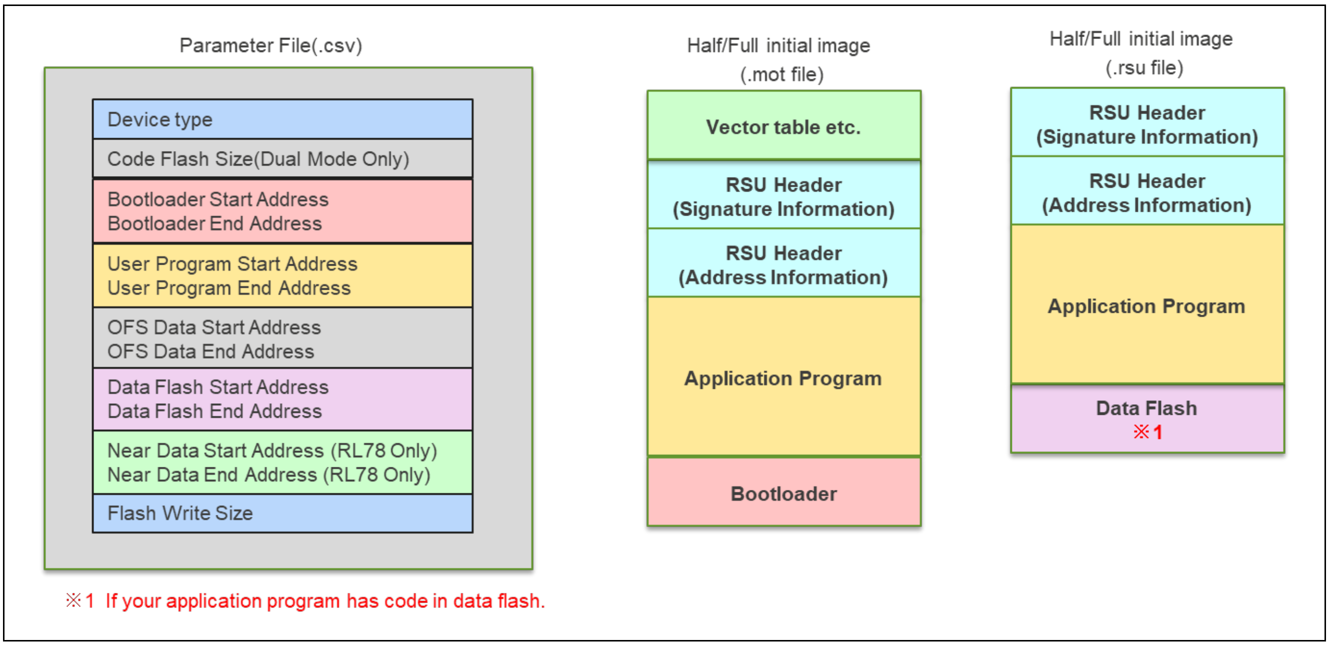
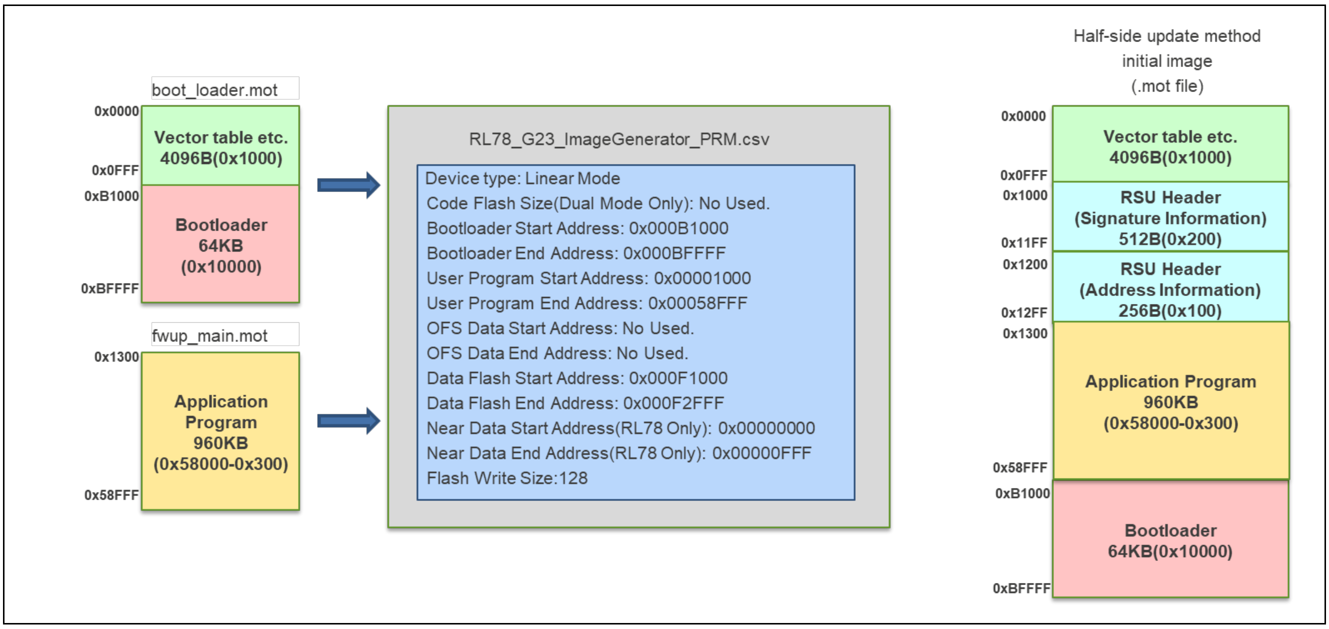

Parameter File(.csv)
Device type
Code Flash Size(Dual Mode Only)
Bootloader Start Address
Bootloader End Address
User Program Start Address
User Program End Address
OFS Data Start Address
OFS Data End Address
Data Flash Start Address
Data Flash End Address
Near Data Start Address (RL78 Only)
Near Data End Address (RL78 Only)
Flash Write Size
Half/Full initial image (.mot file)
Vector table etc.
RSU Header (Signature Information)
RSU Header (Address Information)
Application Program
Bootloader
Half/Full initial image (.rsu file)
RSU Header (Signature Information)
RSU Header (Address Information)
Application Program
Data Flash ※1
※1 If your application program has code in data flash.

Figure 5.5 Parameters referenced when generating image files

- Device type is used to determine how the initial image is generated; for the RL78, set to ‘Linear Mode’.
- Using the bootloader file (boot_loader.mot) as input data, the range from Bootloader Start Address to Bootloader End Address is generated as a code flash for the bootloader (main plane).
- Using the application program file (.mot) as input data, the range from User Program Start Address to User Program End Address is generated as an application program code flash.
- Using the application program file (.mot) as input data, the range from Data Flash Start Address to Data Flash End Address is generated as a data flash. (This demonstration project does not use data flash.)
- Using the bootloader file (boot_loader.mot) as input data, the range from Near Data Flash Start Address (RL78 Only) to Near Data Flash End Address (RL78 Only) is generated using the bootloader vector table etc.
- Flash Write Size is used to set the data size of the RSU header (address information) as the minimum unit when writing to the flash.



Device type: Linear Mode
Code Flash Size(Dual Mode Only): No Used.
Bootloader Start Address: 0x000B1000
Bootloader End Address: 0x000BFFFF
User Program Start Address: 0x00001000
User Program End Address: 0x00058FFF
OFS Data Start Address: No Used.
OFS Data End Address: No Used.
Data Flash Start Address: 0x000F1000
Data Flash End Address: 0x000F2FFF
Near Data Start Address(RL78 Only): 0x00000000
Near Data End Address(RL78 Only): 0x00000FFF
Flash Write Size:128

Figure 5.6 Example of parameters referred to when generating the initial image of RL78/G23 partial update method

6. Appendices

6.1 Confirmed Operation Environments

This section describes confirmed operation environment for the module.

Table 6.1 Confirmed Operation Environment (CC-RL)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio 2025-01
C compiler	Renesas Electronics CC-RL V1.14.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Little endian
Revision of the module	Rev.2.03
Board used	RL78/G23-128p Fast Prototyping Board 128-pin (R7F100GSN CF 768KB) RL78/G22-48p Fast Prototyping Board 48-pin (R7F102GGE2DFB CF 64KB) RL78/G24-64p Fast Prototyping Board 64-pin (RTK7RLG240C00000BJ)

Table 6.2 Confirmed Operation Environment (IAR)

Item	Description
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RL78 5.10.1
C compiler	IAR Systems IAR C/C++ Compiler for Renesas RL78 version 5.10.1 IAR Assembler for Renesas RL78 version 5.10.1 Compiler option: Default settings of the integrated development environment.
Endian order	Little endian
Revision of the module	Rev.2.03
Board used	RL78/G23-128p Fast Prototyping Board 128-pin (R7F100GSN CF 768KB) RL78/G22-48p Fast Prototyping Board 48-pin (R7F102GGE2DFB CF 64KB) RL78/G24-64p Fast Prototyping Board 64-pin (RTK7RLG240C00000BJ)

Table 6.3 Confirmed Operation Environment (LLVM)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio 2025-01
C compiler	LLVM for Renesas RL78 17.0.1.202412
Endian order	Little endian
Revision of the module	Rev.2.03
Board used	RL78/G23-128p Fast Prototyping Board 128-pin (R7F100GSN CF 768KB)

6.2 Operating Environment for Demo Project

This module supports multiple compilers. When using this module, the different settings for each compiler are shown below.

6.2.1 Operation Confirmation Environment for RL78/G23

The execution environment and connection diagram are shown below.

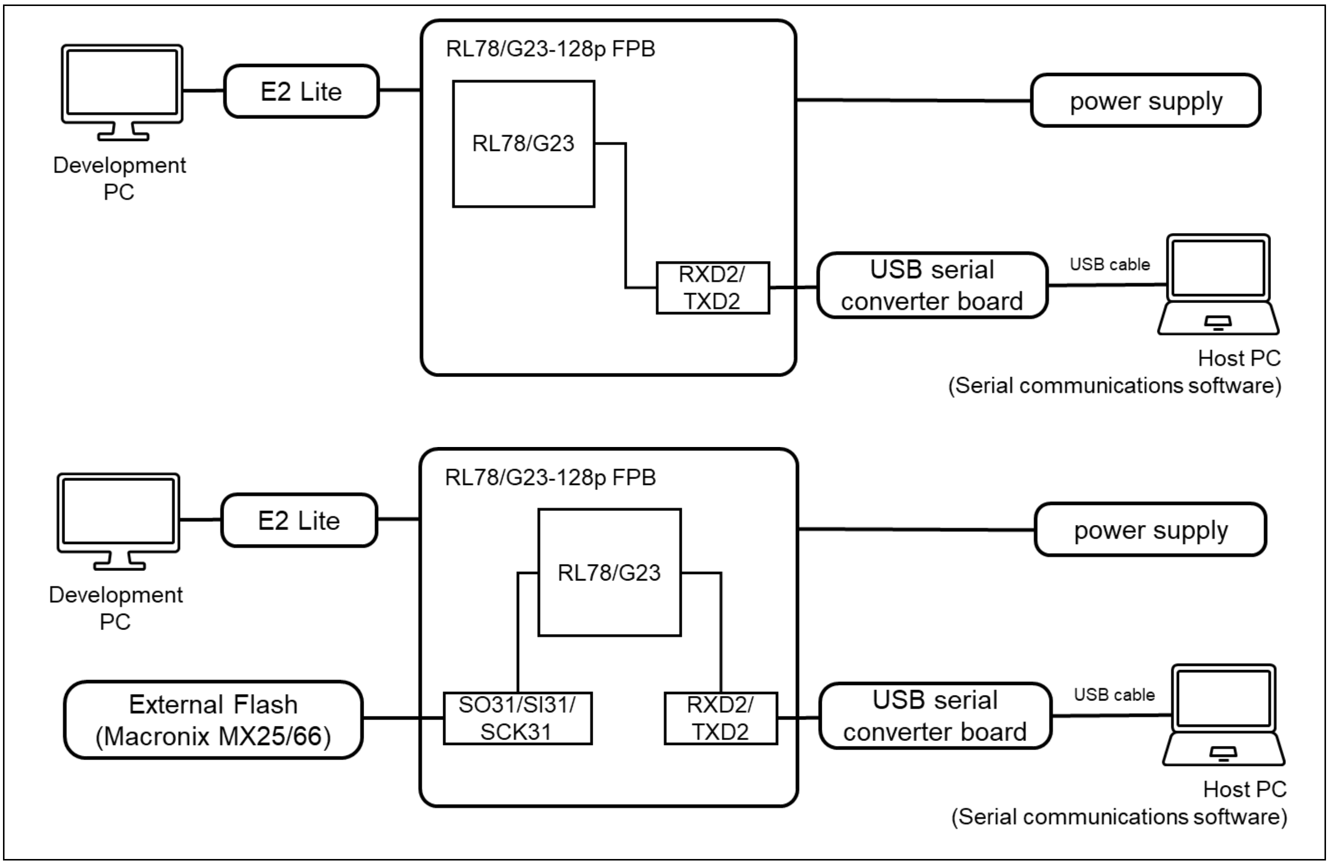
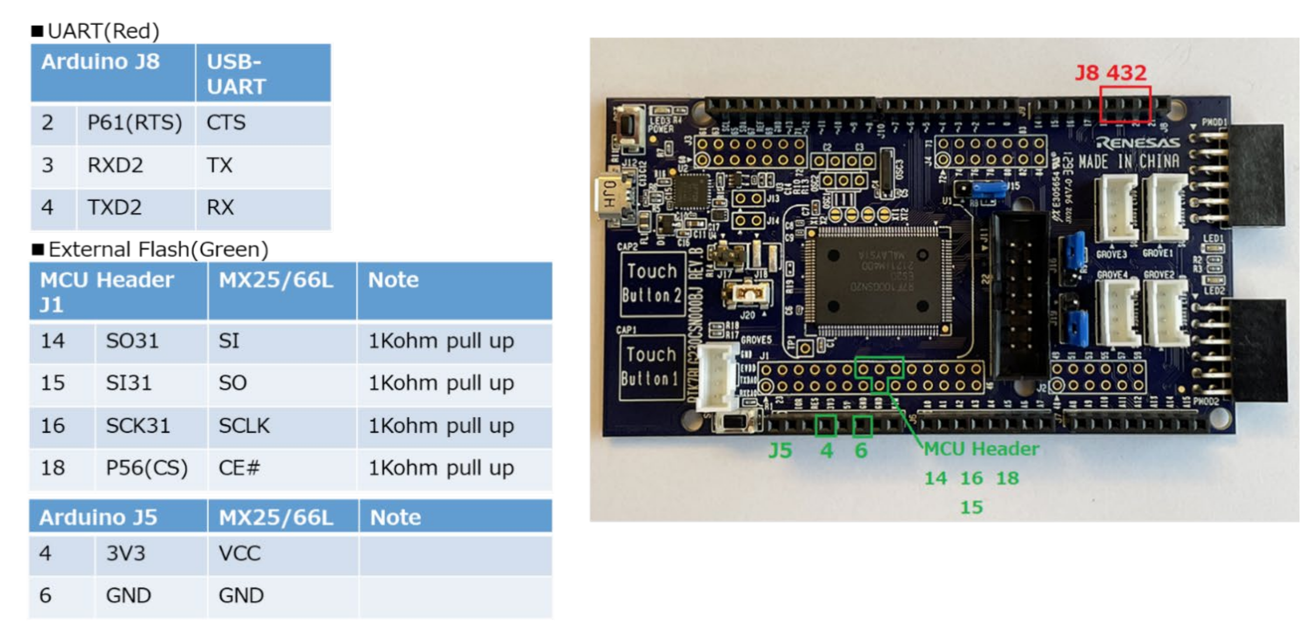


Figure 6.1 RL78/G23-128p FPB Device Connection Diagram

The pin assignment is shown in the figure below.



■UART(Red)

Arduino J8		USB-UART
2	P61(RTS)	CTS
3	RXD2	TX
4	TXD2	RX

■External Flash(Green)

MCU Header J1		MX25/66L	Note
14	SO31	SI	1Kohm pull up
15	SI31	SO	1Kohm pull up
16	SCK31	SCLK	1Kohm pull up
18	P56(CS)	CE#	1Kohm pull up

Arduino J5		MX25/66L	Note
4	3V3	VCC	
6	GND	GND	

Figure 6.2 RL78/G23-128p FPB Device Connection Pin Information

6.2.1.1 Memory map of demo project for partial update method

Shown below are the memory map of the RL78/G23 partial update method demo project and the memory map of the configuration settings.

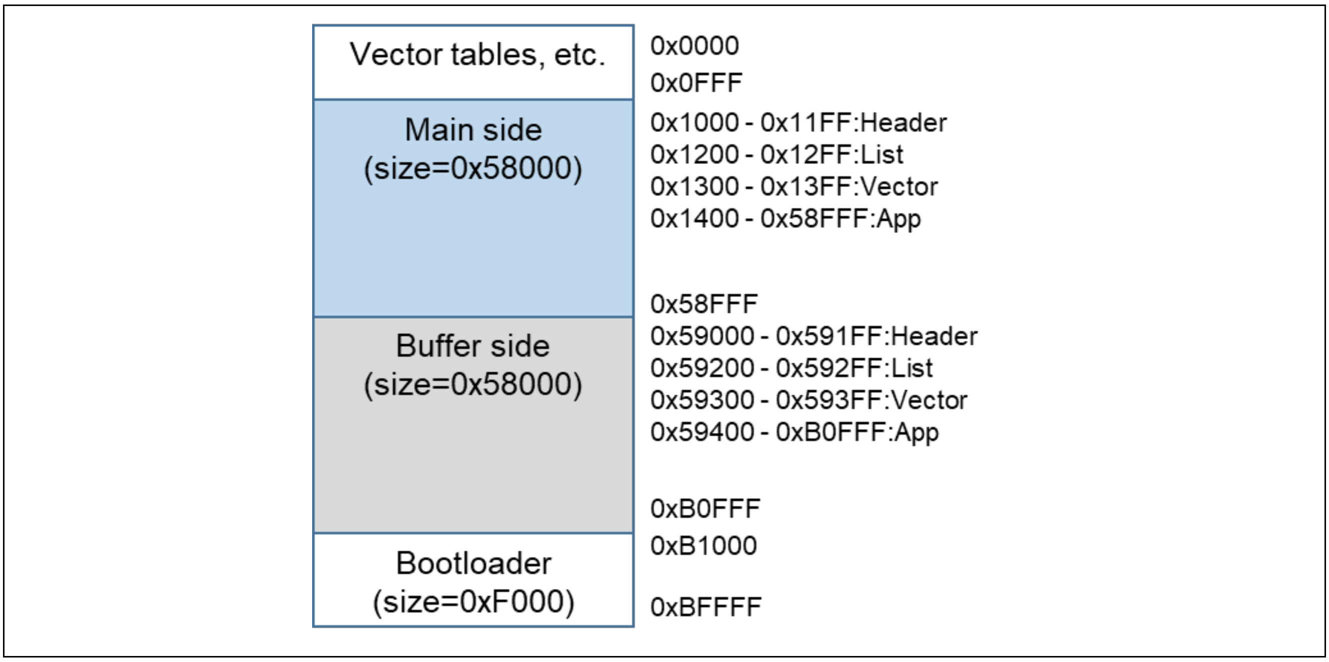


Figure 6.3 RL78/G23 partial update method demo project memory map (For CC-RL and IAR)

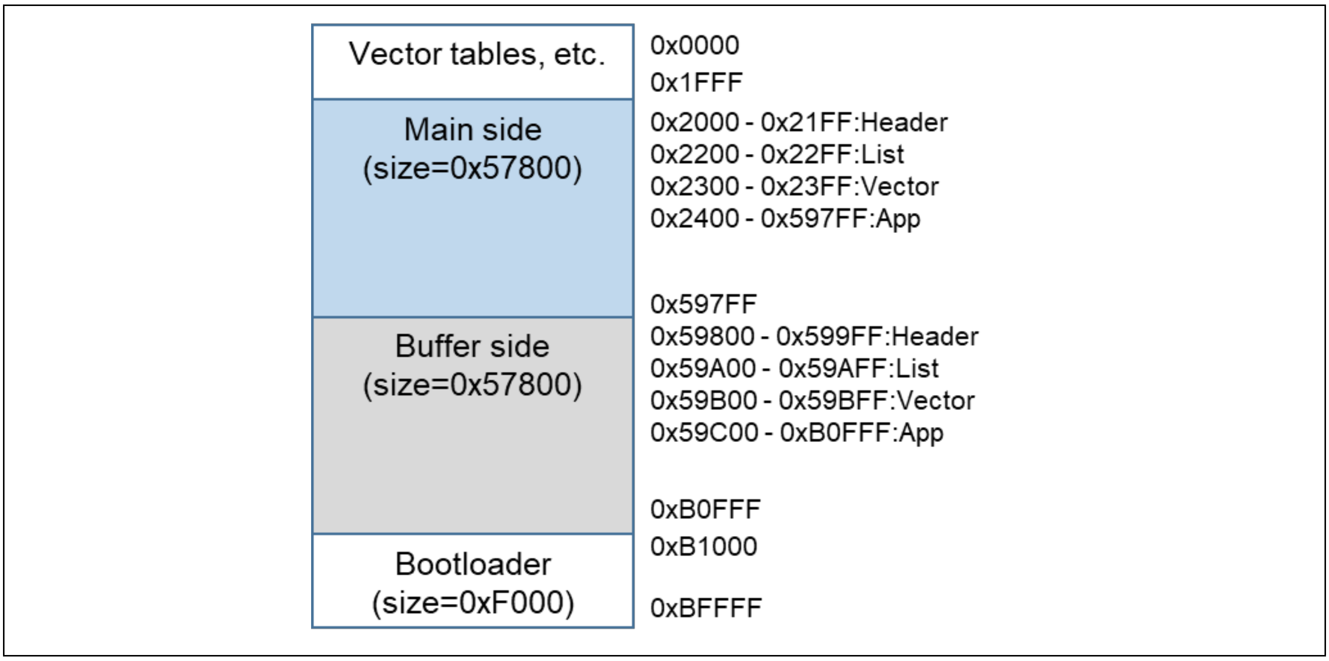


Figure 6.4 RL78/G23 partial update method demo project memory map (For LLVM)

Table 6.4 RL78/G23 partial update method configuration setting (For CC-RL and IAR)

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0x1000	0x1000
FWUP_CFG_BUF_AREA_ADDR_L	0x59000	0x59000
FWUP_CFG_AREA_SIZE	0x58000	0x58000
FWUP_CFG_CF_BLK_SIZE	2048	2048
FWUP_CFG_CF_W_UNIT_SIZE	128	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	4096	4096
FWUP_CFG_DF_ADDR_L	0xF1000	0xF1000
FWUP_CFG_DF_BLK_SIZE	256	256
FWUP_CFG_DF_NUM_BLKS	32	32
FWUP_CFG_FWUPV1_COMPATIBLE	0	0
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	1	1

Table 6.5 RL78/G23 partial update method configuration setting (For LLVM)

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0x2000	0x2000
FWUP_CFG_BUF_AREA_ADDR_L	0x59800	0x59800
FWUP_CFG_AREA_SIZE	0x57800	0x57800
FWUP_CFG_CF_BLK_SIZE	2048	2048
FWUP_CFG_CF_W_UNIT_SIZE	128	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	4096	4096
FWUP_CFG_DF_ADDR_L	0xF1000	0xF1000
FWUP_CFG_DF_BLK_SIZE	256	256
FWUP_CFG_DF_NUM_BLKS	32	32
FWUP_CFG_FWUPV1_COMPATIBLE	0	0
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	1	1

6.2.1.2 Memory map of demo project for full update method

The memory map of the RL78/G23 full update method demo project and the memory map of the configuration settings are shown below.

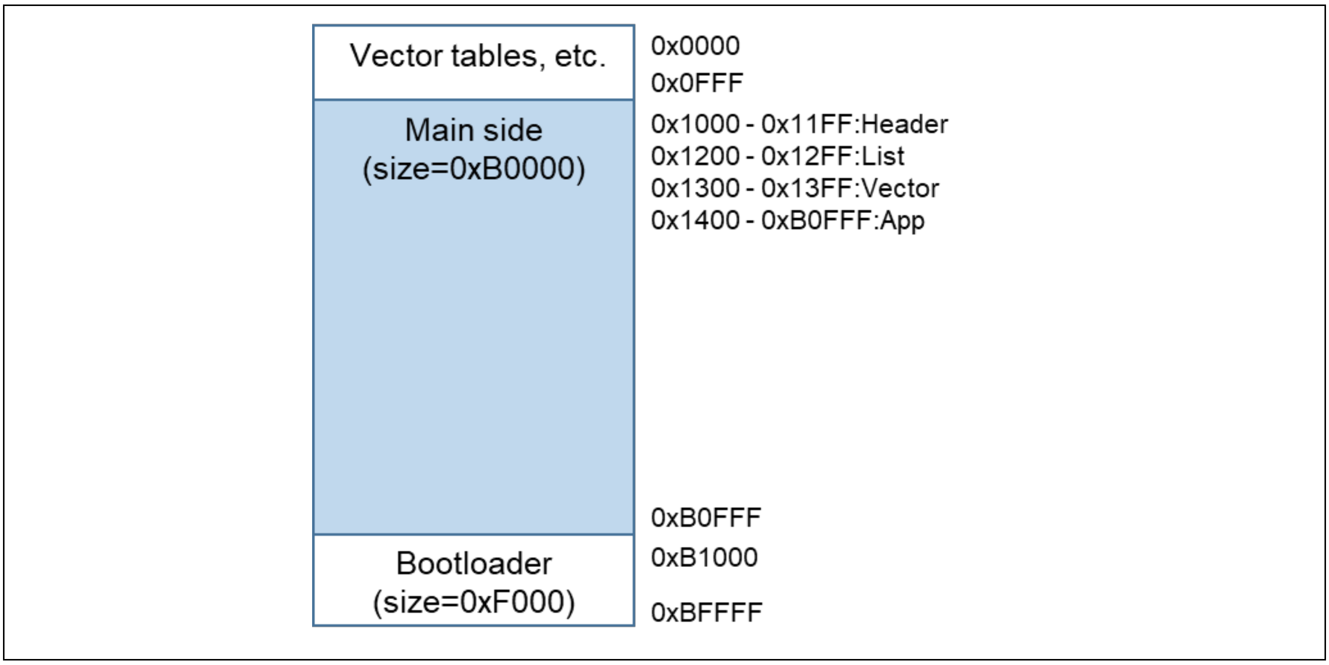


Figure 6.5 RL78/G23 full update method demo project memory map (For CC-RL and IAR)

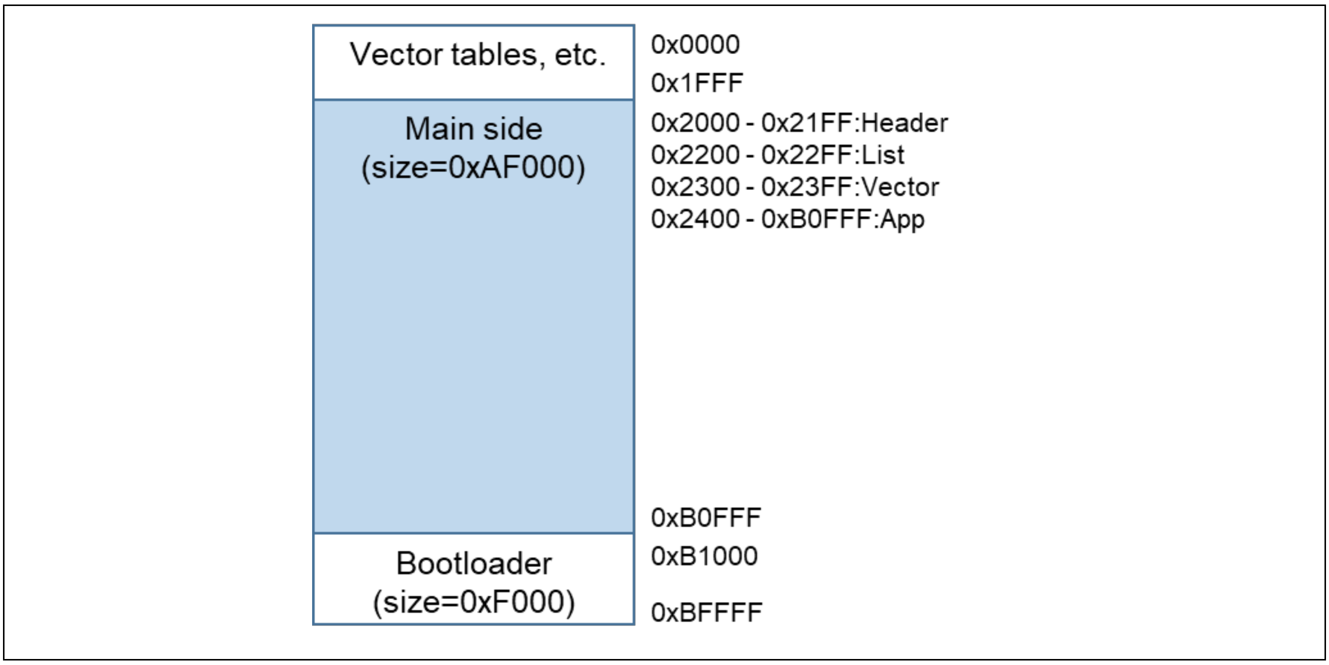


Figure 6.6 RL78/G23 full update method demo project memory map (For LLVM)

Table 6.6 RL78/G23 full update method configuration setting (For CC-RL and IAR)

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	3	3
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0x1000	0x1000
FWUP_CFG_BUF_AREA_ADDR_L	0x59000	0x59000
FWUP_CFG_AREA_SIZE	0xB0000	0xB0000
FWUP_CFG_CF_BLK_SIZE	2048	2048
FWUP_CFG_CF_W_UNIT_SIZE	128	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	4096	4096
FWUP_CFG_DF_ADDR_L	0xF1000	0xF1000
FWUP_CFG_DF_BLK_SIZE	256	256
FWUP_CFG_DF_NUM_BLKS	32	32
FWUP_CFG_FWUPV1_COMPATIBLE	0	0
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	1	1

Table 6.7 RL78/G23 full update method configuration setting (For LLVM)

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	3	3
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0x2000	0x2000
FWUP_CFG_BUF_AREA_ADDR_L	0x2000	0x2000
FWUP_CFG_AREA_SIZE	0xAF000	0xAF000
FWUP_CFG_CF_BLK_SIZE	2048	2048
FWUP_CFG_CF_W_UNIT_SIZE	128	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	4096	4096
FWUP_CFG_DF_ADDR_L	0xF1000	0xF1000
FWUP_CFG_DF_BLK_SIZE	256	256
FWUP_CFG_DF_NUM_BLKS	32	32
FWUP_CFG_FWUPV1_COMPATIBLE	0	0
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	1	1

6.2.2 Operation Confirmation Environment for RL78/G24

The execution environment and connection diagram are shown below.

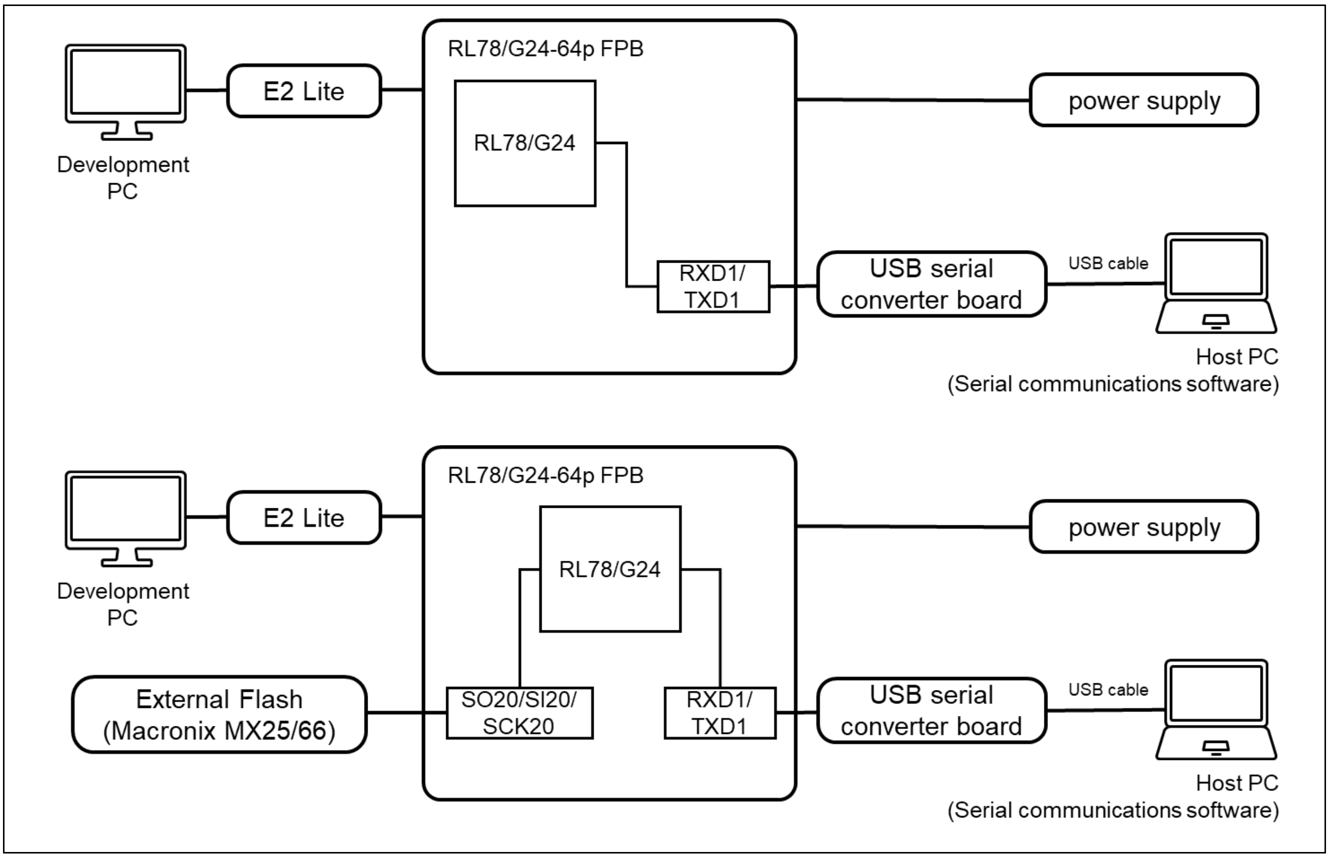


Figure 6.7 RL78/G24-64p FPB Device Connection Diagram

The pin assignment is shown in the figure below.

■UART(Red)

Arduino J5		USB-UART	Note
1	RXD1	TX	
2	TXD1	RX	
3	P140(RTS)	CTS	

■External Flash(Green)

Arduino J5,J6		MX25 /66L	Note
J6 3	SO20	SI	1Kohm pull up
J6 2	SI20	SO	1Kohm pull up
J5 7	SCK20	SCLK	1Kohm pull up
J5 6	P16(CS)	CE#	1Kohm pull up

Arduino J3		MX25/66L	Note
4	3V3	VCC	
6	GND	GND	

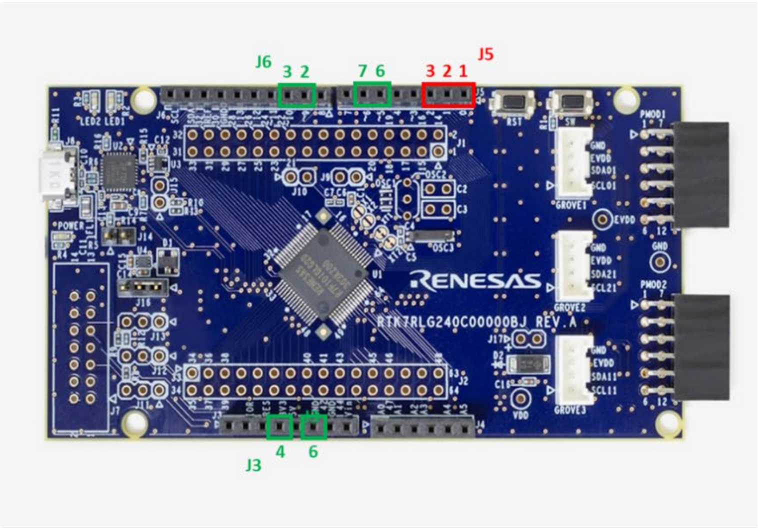


Figure 6.8 RL78/G24-64p FPB Pin Information

6.2.2.1 Memory map of demo project for partial update method

Shown below are the memory map of the RL78/G24 partial update method demo project and the memory map of the configuration settings.

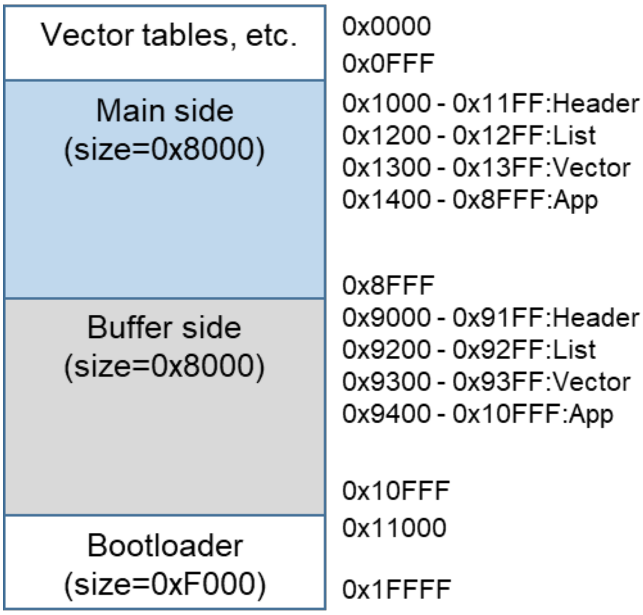


Figure 6.9 RL78/G24 partial update method demo project memory map

Table 6.8 RL78/G24 partial update method configuration setting

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0x1000	0x1000
FWUP_CFG_BUF_AREA_ADDR_L	0x9000	0x9000
FWUP_CFG_AREA_SIZE	0x8000	0x8000
FWUP_CFG_CF_BLK_SIZE	2048	2048
FWUP_CFG_CF_W_UNIT_SIZE	128	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	4096	4096
FWUP_CFG_DF_ADDR_L	0xF1000	0xF1000
FWUP_CFG_DF_BLK_SIZE	256	256
FWUP_CFG_DF_NUM_BLKS	16	16
FWUP_CFG_FWUPV1_COMPATIBLE	0	0
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	1	1

6.2.2.2 Memory map of demo project for full update method

The memory map of the RL78/G24 full update method demo project and the memory map of the configuration settings are shown below.

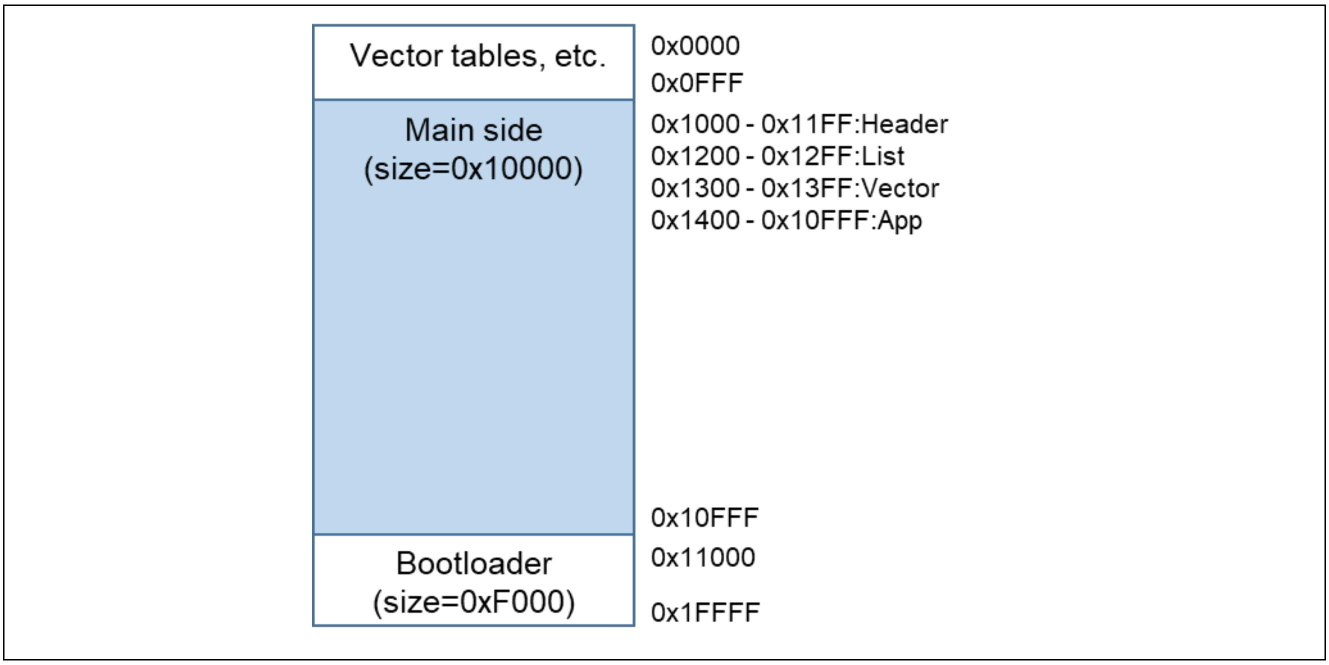


Figure 6.10 RL78/G24 full update method demo project memory map

Table 6.9 RL78/G24 full update method configuration setting

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	3	3
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0x1000	0x1000
FWUP_CFG_BUF_AREA_ADDR_L	0x9000	0x9000
FWUP_CFG_AREA_SIZE	0x10000	0x10000
FWUP_CFG_CF_BLK_SIZE	2048	2048
FWUP_CFG_CF_W_UNIT_SIZE	128	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	4096	4096
FWUP_CFG_DF_ADDR_L	0xF1000	0xF1000
FWUP_CFG_DF_BLK_SIZE	256	256
FWUP_CFG_DF_NUM_BLKS	16	16
FWUP_CFG_FWUPV1_COMPATIBLE	0	0
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	1	1

6.2.3 Operation Confirmation Environment for RL78/G22

The execution environment and connection diagram are shown below.

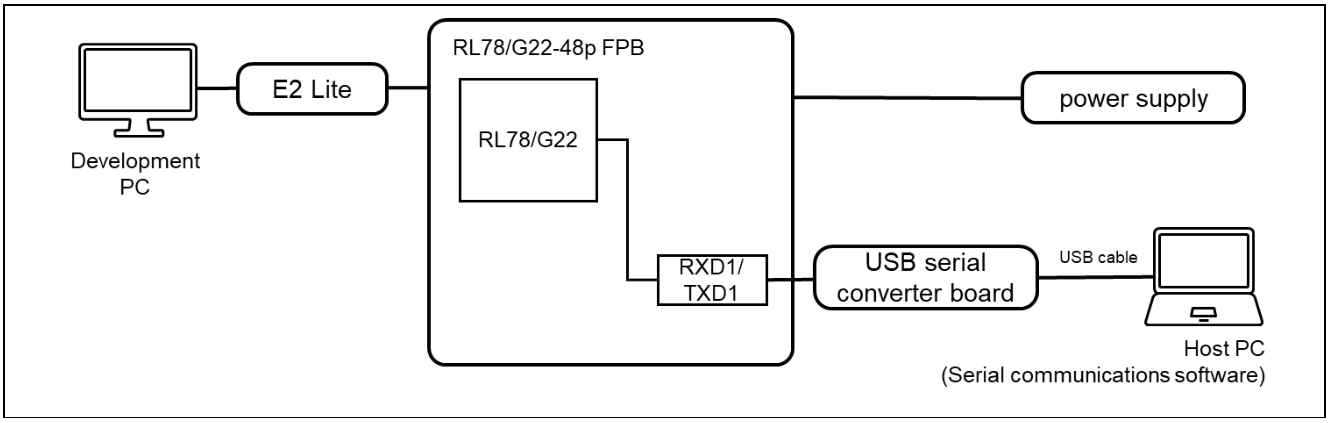


Figure 6.11 RL78/G22-48p FPB Device Connection Diagram

The pin assignment is shown in the figure below.

Arduino J7		USB-UART
1	RXD1	TX
2	TXD1	RX
3	P140	CTS

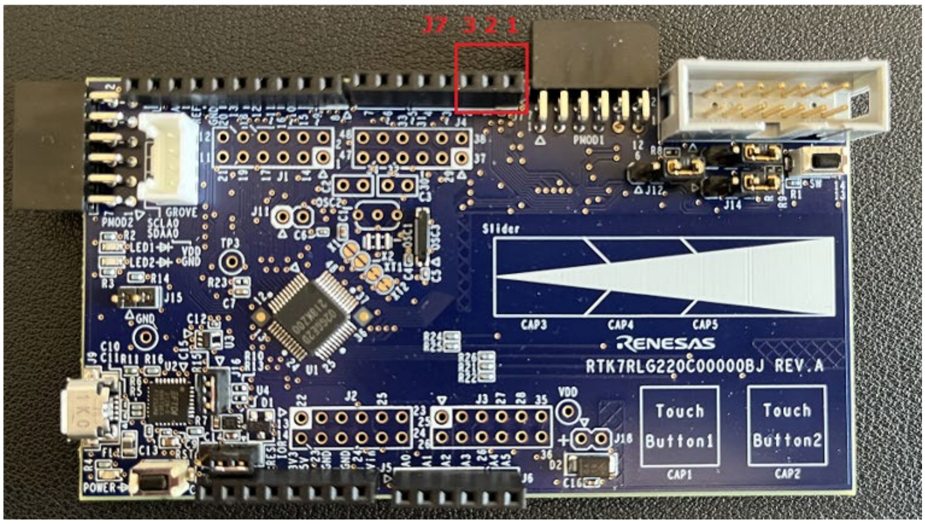


Figure 6.12 RL78/G22-48p FPB Pin Information

6.2.3.1 Memory map of demo project for full update method

The memory map of the RL78/G22 full update method demo project and the memory map of the configuration settings are shown below.

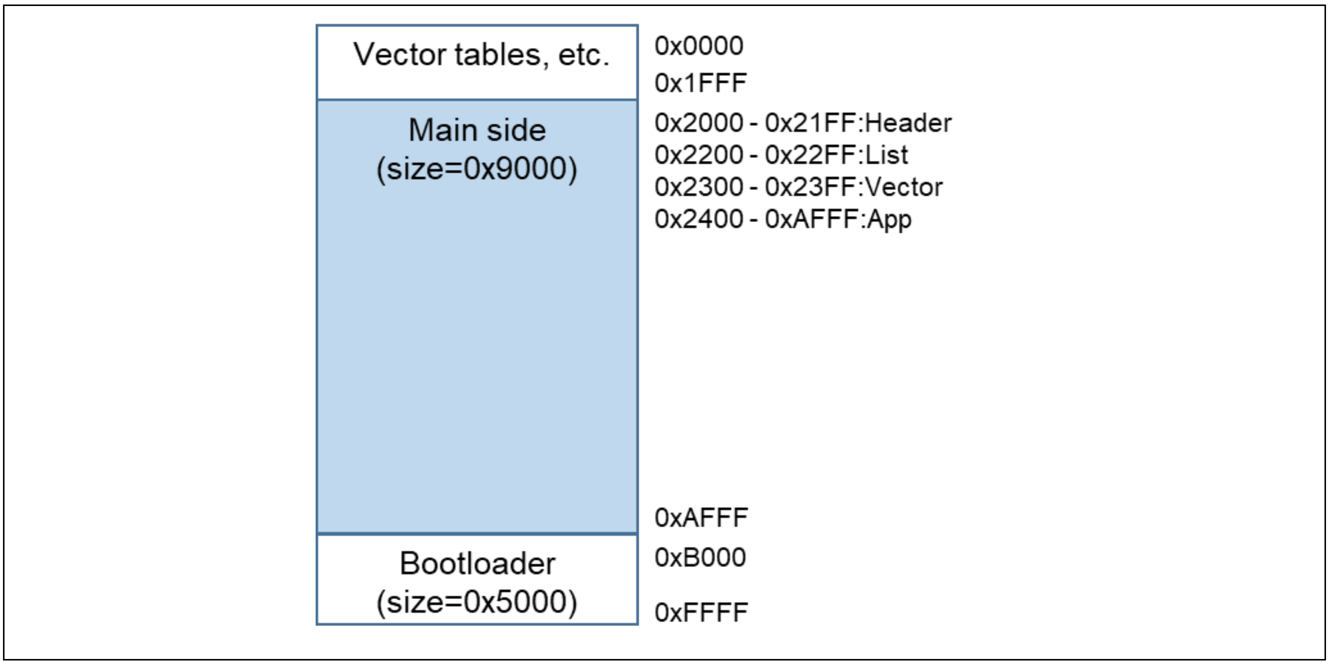


Figure 6.13 RL78/G22 full update method demo project memory map

Table 6.10 RL78/G22 full update method configuration setting

Configuration options in r_fwup _config.h	
parameter name	**boot_loader**
FWUP_CFG_UPDATE_MODE	2
FWUP_CFG_FUNCTION_MODE	0
FWUP_CFG_MAIN_AREA_ADDR_L	0x2000
FWUP_CFG_BUF_AREA_ADDR_L	0x2000
FWUP_CFG_AREA_SIZE	0x9000
FWUP_CFG_CF_BLK_SIZE	2048
FWUP_CFG_CF_W_UNIT_SIZE	128
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096
FWUP_CFG_DF_ADDR_L	0xF1000
FWUP_CFG_DF_BLK_SIZE	256
FWUP_CFG_DF_NUM_BLKS	8
FWUP_CFG_FWUPV1_COMPATIBLE	0
FWUP_CFG_SIGNATURE_VERIFICATION	1
FWUP_CFG_PRINTF_DISABLE	1

6.3 Open source license information used in the demo project

The demo project for this product uses the open source TinyCrypt. If you use TinyCrypto for your cryptographic library, you must comply with the terms of use set forth in TinyCrypt's license terms.

Check out the TinyCrypt license terms below.

URL：https://github.com/intel/tinycrypt

license：https://github.com/intel/tinycrypt/blob/master/LICENSE

7. Notes

7.1 Notes on Transition from Bootloader to Application.

When transitioning from the sample bootloader program to the application, the settings of the bootloader's peripheral functions will be taken over by the application.

For the peripheral functions used in the sample bootloader, the API functions of each module are closed at the end of the bootloader. Other settings are default values when the smart configurator is used.

If the customer modifies the bootloader sample program for use, the settings of the peripheral functions set in the bootloader will be inherited by the application side. Therefore, it is recommended to initialize the settings of the peripheral functions before moving from the bootloader to the application, or to share the settings of the peripheral functions with the application.

When creating an application, please take the implementation of the bootloader into consideration.

Table 7.1 Notes on peripheral functions used in the bootloader

Peripheral Functions	Settings and Notes on the Boot Loader
Board Functions	These are the default values when the module is embedded in the Smart Configurator. The settings are not changed in the bootloader. Note) For the demo project for RL78/G24, the PLL setting is set to 32 MHz, which is different from the default value.
Functions of Flash Memory	The Flash API performs Close for peripheral functions related to flash memory and transitions to the application.
Serial Communication Functions	For peripheral functions related to serial communication, Close is performed by the SCI API and the transition is made to the application. For the SCI channels used in the bootloader, refer to the device connection diagram for each product in 6.2 Operating Environment for Demo Project.
Option Setting Memory	For the option setting memory, set the same value in the bootloader and the application program.
Other Functions	As for the settings of other functions, these are the default values when using the Smart Configurator. The interrupt enable flag is set to interrupt disabled to transition to the application.

7.2 Security measures for the bootloader area

When the firmware update module is commercialized by the customer, it is recommended to protect the area of the code flash where the bootloader (boot_loader) is deployed.

Revision History

Rev.	Date	Description	
		Page	Summary
2.00	Jul. 20, 2023	—	First edition issued
2.01	Nov. 22, 2023	1	Added RL78/G24 to Target Devices
		13-14	Added device to folder structure
		18	Added FWUP_CFG_CF_W_UNIT_SIZE and FWUP_CFG_FWUPV1_COMPATIBLE to configuration settings
		20	Added device in ROM/RAM/Stack
		23	Added parameter to R_FWUP_EraseArea function
		23	Added description to R_FWUP_GetImageSize function
		23	Added description to R_FWUP_GetImageSize function
		24	Added parameter to R_FWUP_WriteImageProgram function
		24	Added return value to R_FWUP_WriteImage function
		25	Added return value to R_FWUP_VerifyImage function
		58	Added board used for operation check environment
		59-67	Added device to Operation check environment
		69	Added note
2.02	Dec.13.2024	12-13	Modified chapter 1.4.
		22-23	Moved figures from chapter 1.6 to chapter 2.10.
		28	Moved the R_FWUP_WriteImageHeader function to chapter 3.13.
		28	Moved the R_FWUP_WriteImageProgram function to chapter 3.14.
		-	Replaced chapters 4 and 5.
		37-52	Modified the description of 4.demo project.
		53-61	Added chapter 4.6 Debugging demo projects.
		62-70	Modified chapter 5 Renesas Image Generator.
		83	Added chapter 7.2 Security measures for the bootloader area
2.03	Apr.18.2025	1	Executive summary LLVM is added to the target compiler.
		14,15	1.5 LLVM content added to the package configuration table.
		18	2.6 Default log display setting changed to Disable.
		19	2.7 Added LLVM to the code size of the demo project.
		22,23	2.10 Changed the flow of the example bootloader implementation.
		41	4.4 Added LLVM contents.
		74	6.1 Added LLVM testing environment.
		77,78	6.2.1.1 LLVM figures and tables added.
		79,80	6.2.1.2 LLVM figures and tables added.

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 product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall 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, copying 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 implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
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, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
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 for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
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.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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