RX Family

Firmware Update Module Using Firmware Integration Technology

Introduction

This application note describes the firmware update module using Firmware Integration Technology (FIT). The module is referred to below as the firmware update FIT module.

This application note is based on Renesas MCU Firmware Update Design Policy (R01AN5548). It is recommended that the reader read that document before consulting this application note.

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

Target Devices

RX130 Group

RX140 Group

RX230, RX231, RX23E-A, RX23W Group

RX65N, RX651 Group

RX66N Group

RX66T Group

RX660 Group

RX671 Group

RX72M Group

RX72N Group

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.

- Renesas MCU Firmware Update Design Policy (R01AN5548)
- RX Family How to implement FreeRTOS OTA by using Amazon Web Services on RX65N (R01AN5549)
- Firmware Integration Technology User's Manual (R01AN1833)
- RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)
- RX Smart Configurator User's Guide: IAREW (R20AN0535)
- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family Flash Module Using Firmware Integration Technology (R01AN2184)
- RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
- RX Family Ethernet Module Using Firmware Integration Technology (R01AN2009)
- RX Family CMT Module Using Firmware Integration Technology (R01AN1856)
- RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)
- RX Family System Timer Module Firmware Integration Technology (R20AN0431)

Target Compilers

- C/C++ Compiler Package for RX Family from Renesas Electronics
- GCC for Renesas RX
- IAR C/C++ Compiler for RX

For compiler details related to the environment on which operation has been confirmed, refer to 5.1, Confirmed Operation Environment.

Contents

1. Overview

1.1 About the Firmware Update Module

A firmware update is a process in which the firmware, the software that controls the device's hardware, is overwritten with a new version of the firmware. Firmware updates may be applied to fix bugs, add new functions, or improve performance.

On RX Family MCUs the firmware is written (programmed) to the on-chip flash memory. Therefore, in the case of the RX Family, the term firmware update refers to the operations and processing for overwriting the contents of the MCU's on-chip flash memory.

Generally, one of the following two methods is used to overwrite the contents of the MCU's on-chip flash memory.

- Off-board programming
  A method in which the MCU is connected to an external flash programming device such as a PC running Flash Programmer and the flash memory is overwritten
- On-board programming (self-programming)
  A method in which the MCU is made to overwrite its own on-chip flash memory

The latter self-programming function is used for firmware updates; the MCU programs its own on-chip flash memory.

To perform self-programming of the on-chip flash memory, it is necessary first to copy to the RAM the program that will program the flash memory and then to execute flash memory programming commands from the RAM. Since users need to obtain new firmware versions via a variety of interfaces, it used to be very difficult to build firmware update functionality into the customer's system.

However, using the firmware update FIT module makes it easy to integrate firmware update functionality into the customer's system.

The firmware update module can be incorporated into user projects as an API. For instructions on adding the module, refer to 2.10, Adding the FIT Module to Your Project.

1.2 Configuration of Firmware Update Module

The firmware update module is middleware for the purpose of updating the firmware of the MCU.

The firmware update module has functions for use on OS-less systems, functions for use on OS-less systems with module-external communication control, and functions for use on systems using FreeRTOS over-the-air (OTA) updates. For details of FreeRTOS over-the-air (OTA) updates, refer to the following webpage:
https://docs.aws.amazon.com/freertos/latest/userguide/freertos-ota-dev.html

Figure 1.1 shows a system configuration incorporating the firmware update module on an OS-less system, Figure 1.2 shows a system configuration incorporating the firmware update module on an OS-less systems with module-external communication control, and Figure 1.3 shows a system configuration incorporating the firmware update module on a system using FreeRTOS over-the-air (OTA) updates.

The bootloader module runs first after the system is reset and verifies that the user program (the program that runs after the bootloader) has not been tampered with.

The firmware update module is incorporated into the user program and performs the actual firmware update.

Table 1.1 lists the FIT modules used for firmware updates.

The firmware to be applied as an update is received via a communication interface and then programmed to the code flash memory of the target device via the firmware update module and flash FIT module.

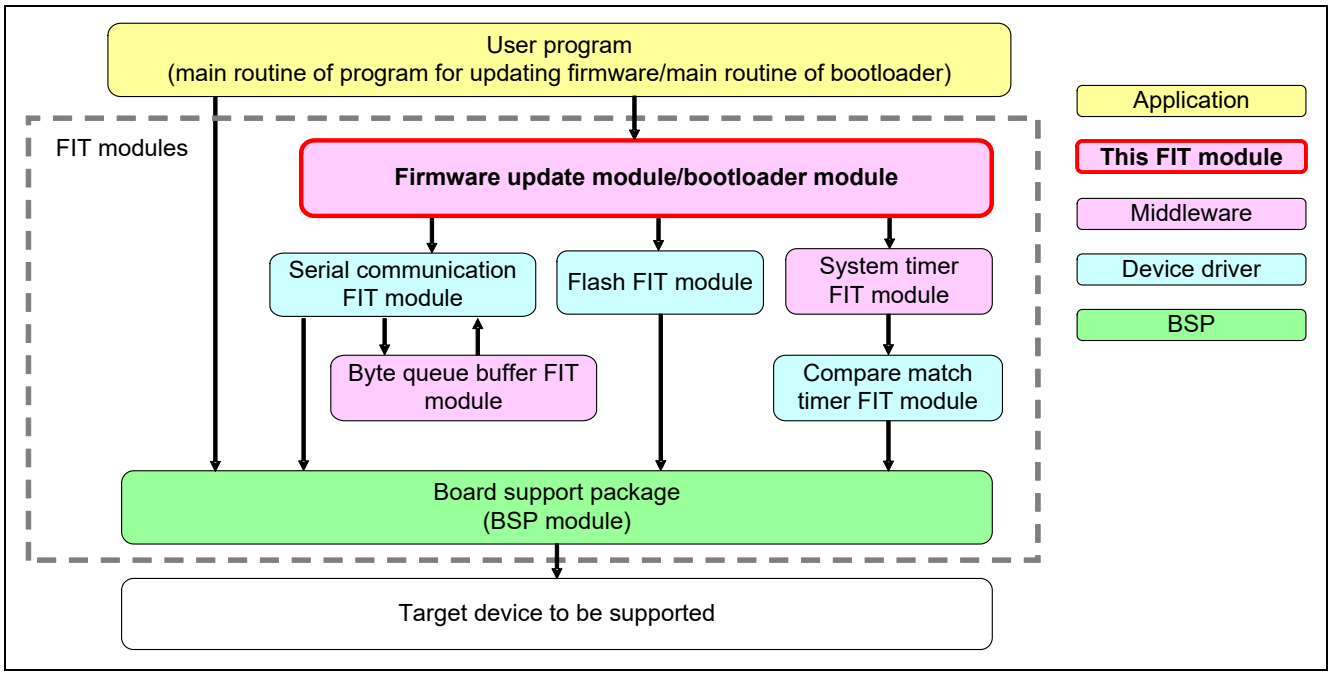


Figure 1.1 System Configuration of Firmware Update Module on OS-less System

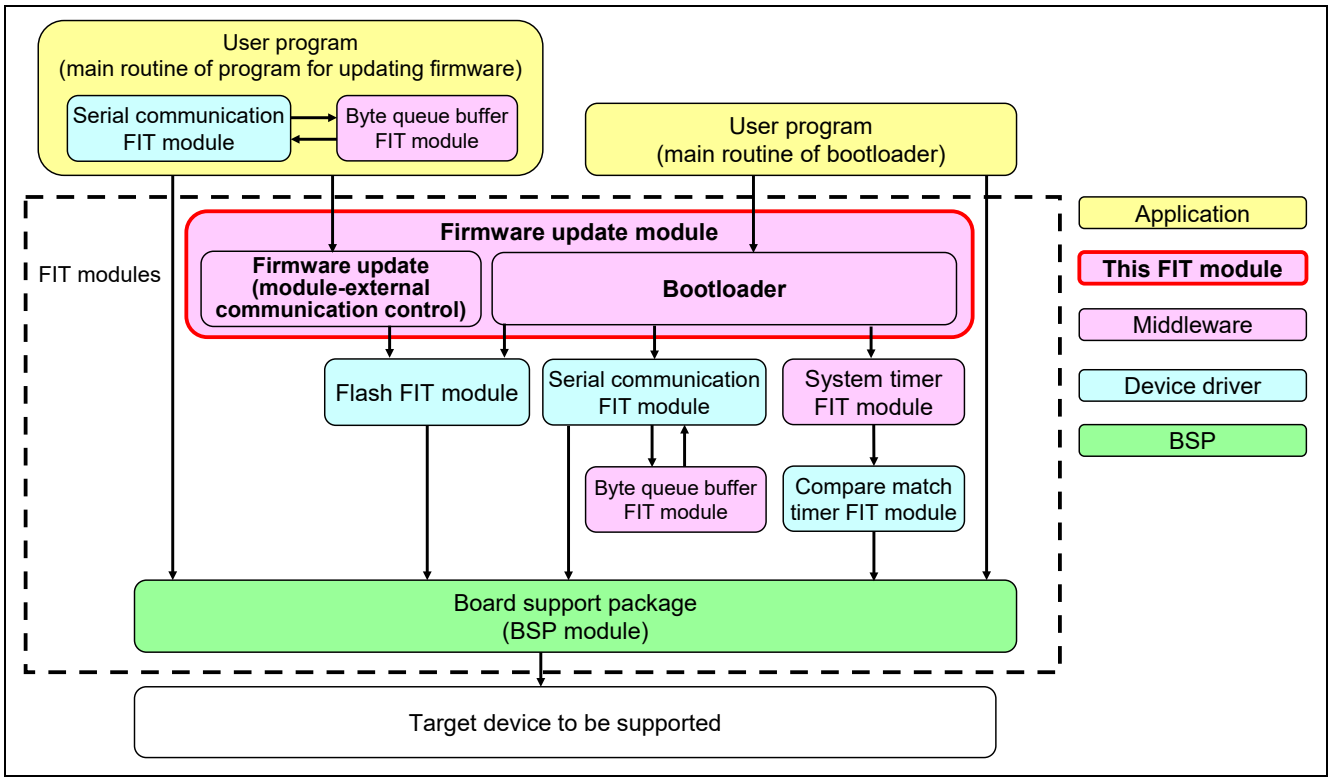


Figure 1.2 System Configuration of Firmware Update Module on OS-less Systems with Module-External Communication Control

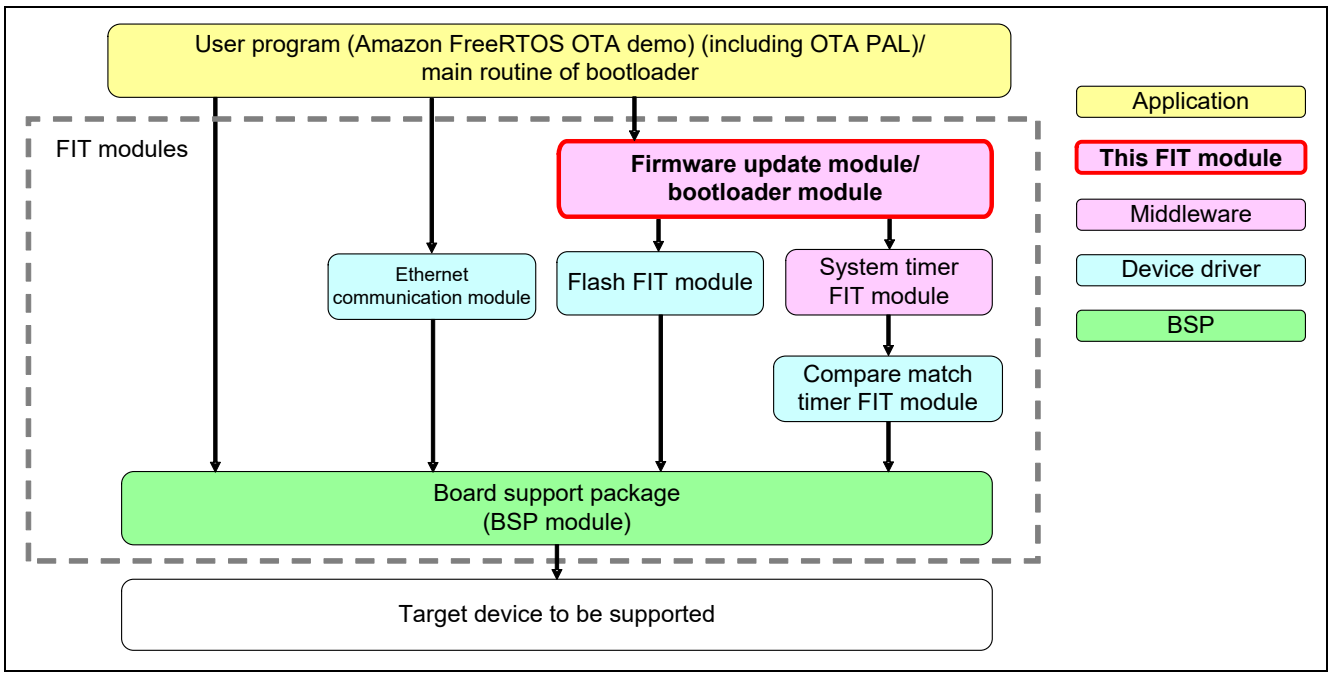


Figure 1.3 System Configuration of Firmware Update Module on System Using FreeRTOS Over-the-Air (OTA) Updates

Table 1.1 List of Modules

Type	Application Note (Document No.)	FIT Module
BSP	RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)	r_bsp
Device driver	RX Family Flash Module Using Firmware Integration Technology (R01AN2184)	r_flash_rx
	RX Family SCI Module Using Firmware Integration Technology (R01AN1815)	r_sci_rx
	RX Family CMT Module Using Firmware Integration Technology (R01AN1856)	r_cmt_rx
Middleware	RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)	r_byteq
	RX Family System Timer Module Firmware Integration Technology (R20AN0431)	r_sys_time_rx

1.3 Firmware Update Operation

On some products in the RX Family the MCU's on-chip flash memory supports dual-bank functionality.

To program the flash memory on a product without dual-bank functionality or when using a product with dual-bank functionality in linear mode, it is necessary first to copy to the RAM the program that will program the flash memory and then to execute flash memory programming commands from the RAM.

When using a product with dual-bank functionality in dual mode, so long as the area of flash memory to be programmed and the area from which the program performing the programming runs are different areas, it is not necessary to run the program from the RAM. This makes it a simple matter to maintain system operation while programming the flash memory.

The firmware update module is capable of applying firmware updates in both linear mode and dual mode.

Table 1.2 Linear Mode and Dual Mode Support on Specific Devices

Device	Linear Mode	Dual Mode
RX130 Group	○	—
RX140 Group	○	—
RX231 Group	○	—
RX65N Group	—	○
RX66T Group	○	—
RX660 Group	○	—
RX671 Group	—	○
RX72N Group	—	○

1.3.1 Firmware Update Operation Using Dual Mode

Firmware update operation when using the flash memory in dual mode is described below.

Firmware update operation is divided into two parts: initial settings to the on-chip flash memory to prepare for the firmware update and applying the firmware update.

Figure 1.4 shows the initial settings for firmware update operation in dual mode.

A tool (Renesas Image Generator) for creating the initial firmware to be written to the on-chip flash memory is provided together with the FIT module. This tool can be used to create two types of initial firmware: initial firmware containing the bootloader only or initial firmware containing both the bootloader and the user program.

By using Flash Programmer or the like to program either of these types of initial firmware, the state shown in step [1] or step [4] of Figure 1.4 can be achieved. You can start from either step [1] or step [4], depending on the characteristics of the customer's system. For a detailed description of the initial firmware, refer to sections 5.4 and 5.5.

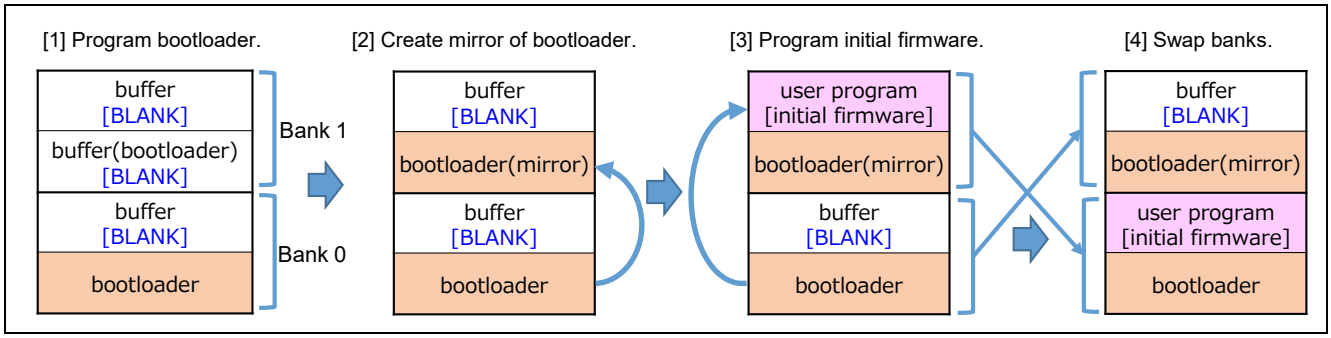


Figure 1.4 Dual Mode Firmware Update Initial Settings

Starting initial settings from step [1]

[1] Use Flash Programmer or the like to program the bootloader to the on-chip flash memory.
[2] Run the bootloader to create a mirror of the bootloader in bank 1.
[3] Use the bootloader to program the initial firmware (must be input externally) and to verify the firmware.
[4] If the verification completes successfully, swap the banks.

Starting initial settings from step [4]

[4] Use Flash Programmer or the like to program the initial firmware containing the bootloader and the user program to the on-chip flash memory.

Figure 1.5 shows dual mode firmware update operation. (Note that "[1] Initial state" below refers to the state after the bootloader has run at initial startup and a mirror of the bootloader has been created in bank 1.)

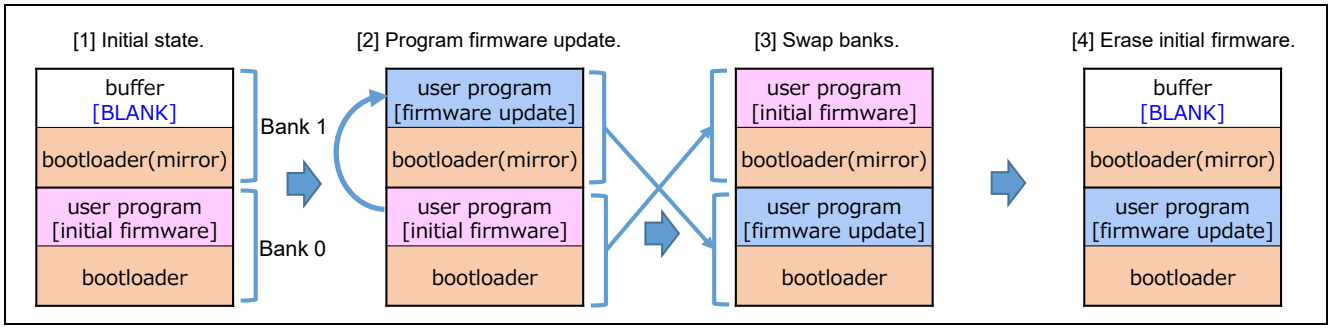


Figure 1.5 Dual Mode Firmware Update Operation

[1] Initial state.
[2] Use the firmware update module incorporated in the user program to program the firmware update (must be input externally) and to verify the firmware after it has been programmed.
[3] If the verification completes successfully, swap the banks.
[4] Erase the initial firmware from bank 1.

1.3.2 Firmware Update Operation Using Linear Mode

There are two methods of performing a firmware update using the flash memory in linear mode: partial overwrite and full overwrite. These are described below.

Note: Unlike the other two methods (dual mode and partial overwrite in linear mode), if an update fails when performing a full overwrite in linear mode, because power is interrupted for example, it is not possible to restore the old version of the firmware.

1.3.2.1 Firmware Update Using Partial Overwrite

Figure 1.6 shows the initial settings for the partial overwrite method.

A tool (Renesas Image Generator) for creating the initial firmware to be written to the on-chip flash memory is provided together with the FIT module. This tool can be used to create initial firmware containing the bootloader only or to create initial firmware containing both the bootloader and the user program.

By using Flash Programmer or the like to program either of these two types of initial firmware, the state shown in step [1] or step [2] of Figure 1.6 can be achieved. You can start from either step [1] or step [2], depending on the characteristics of the customer's system. For a detailed description of the initial firmware, refer to sections 5.4 and 5.5.

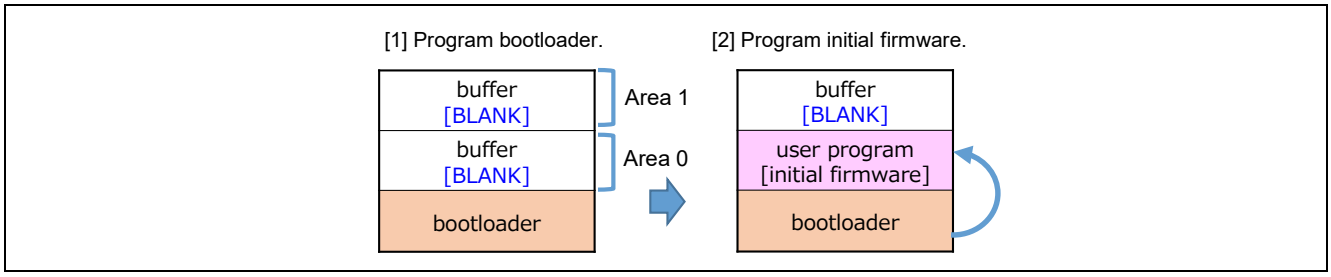


Figure 1.6 Partial Overwrite Firmware Update Initial Settings

Starting initial settings from step [1]

[1] Use Flash Programmer or the like to program the bootloader to the on-chip flash memory.
[2] Use the bootloader to program the initial firmware (must be input externally) and to verify the firmware after it has been programmed to the on-chip flash memory. If the verification completes successfully, the operation is complete.

Starting initial settings from step [2]

[2] Use Flash Programmer or the like to program the initial firmware containing the bootloader and the user program to the on-chip flash memory.

Figure 1.7 shows partial overwrite firmware update operation.

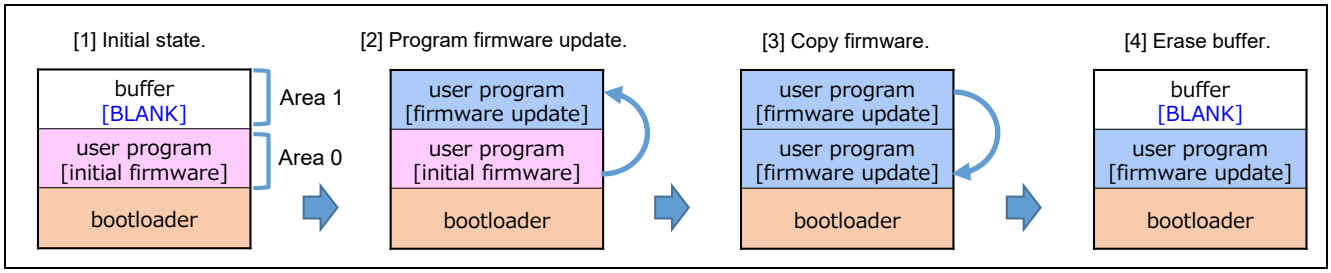


Figure 1.7 Partial Overwrite Firmware Update Initial Settings

[1] Initial state.
[2] Use the user program (area 0) to program the firmware update (must be input externally) to the buffer area (area 1) and to verify the firmware after it has been programmed.
[3] If the verification completes successfully, copy the firmware from the buffer area (area 1) to the user program area (area 0).
[4] Erase the buffer area (area 1).

1.3.2.2 Firmware Update Using Full Overwrite

Figure 1.8 shows initial settings for full overwrite firmware update operation.

A tool (Renesas Image Generator) for creating the initial firmware to be written to the on-chip flash memory is provided together with the FIT module. This tool can be used to create initial firmware containing the bootloader only or to create initial firmware containing both the bootloader and the user program.

By using Flash Programmer or the like to program either of these two types of initial firmware, the state shown in step [1] or step [2] of Figure 1.8 can be achieved. You can start from either step [1] or step [2], depending on the characteristics of the customer's system. For a detailed description of the initial firmware, refer to sections 5.4 and 5.5.

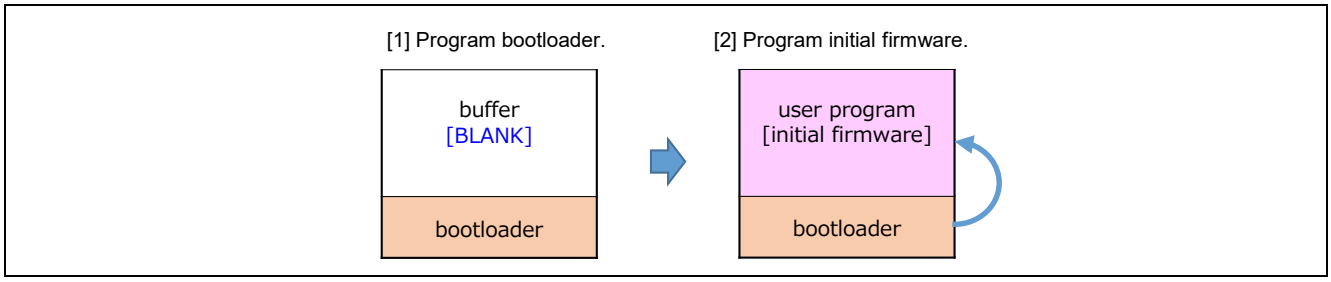


Figure 1.8 Full Overwrite Firmware Update Initial Settings

Starting initial settings from step [1]

[1] Use Flash Programmer or the like to program the bootloader to the on-chip flash memory.
[2] Use the bootloader to program the initial firmware (must be input externally) and to verify the firmware after it has been programmed to the on-chip flash memory. If the verification completes successfully, the operation is complete.

Starting initial settings from step [2]

[2] Use Flash Programmer or the like to program the initial firmware containing the bootloader and the user program to the on-chip flash memory.

Figure 1.9 shows full overwrite firmware update operation.

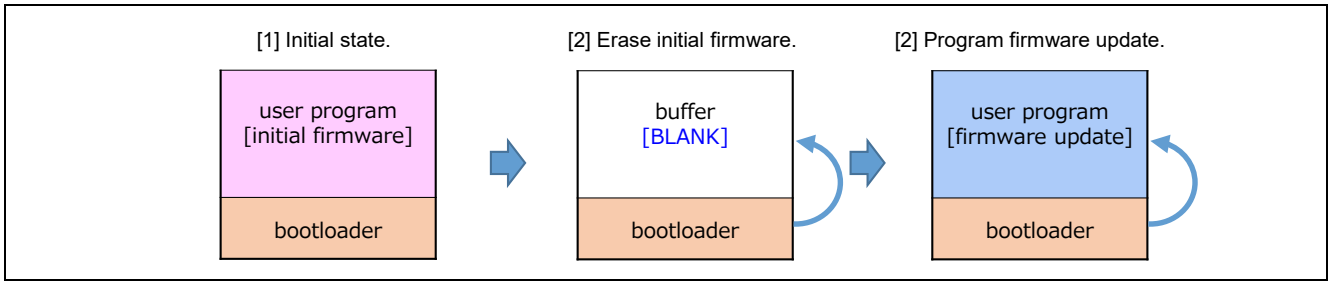


Figure 1.9 Full Overwrite Firmware Update Initial Settings

[1] 0Initial state.
[2] Use the user program to configure settings to disable use of the initial firmware and apply a reset. After this, the bootloader launches and erases the initial firmware.
[3] Use the bootloader to program the initial firmware (must be input externally) and to verify the firmware after it has been programmed to the on-chip flash memory. If the verification completes successfully, the operation is complete.

1.4 Firmware Update Communication Control on OS-Less System

1.4.1 Firmware Update with Module-Internal Communication Control

In this type of system, processing from firmware reception to signature verification is implemented internally by APIs of the firmware update module (via the R_FWUP_Operation function).

An overview of the firmware update operation incorporated into the user program is shown below.

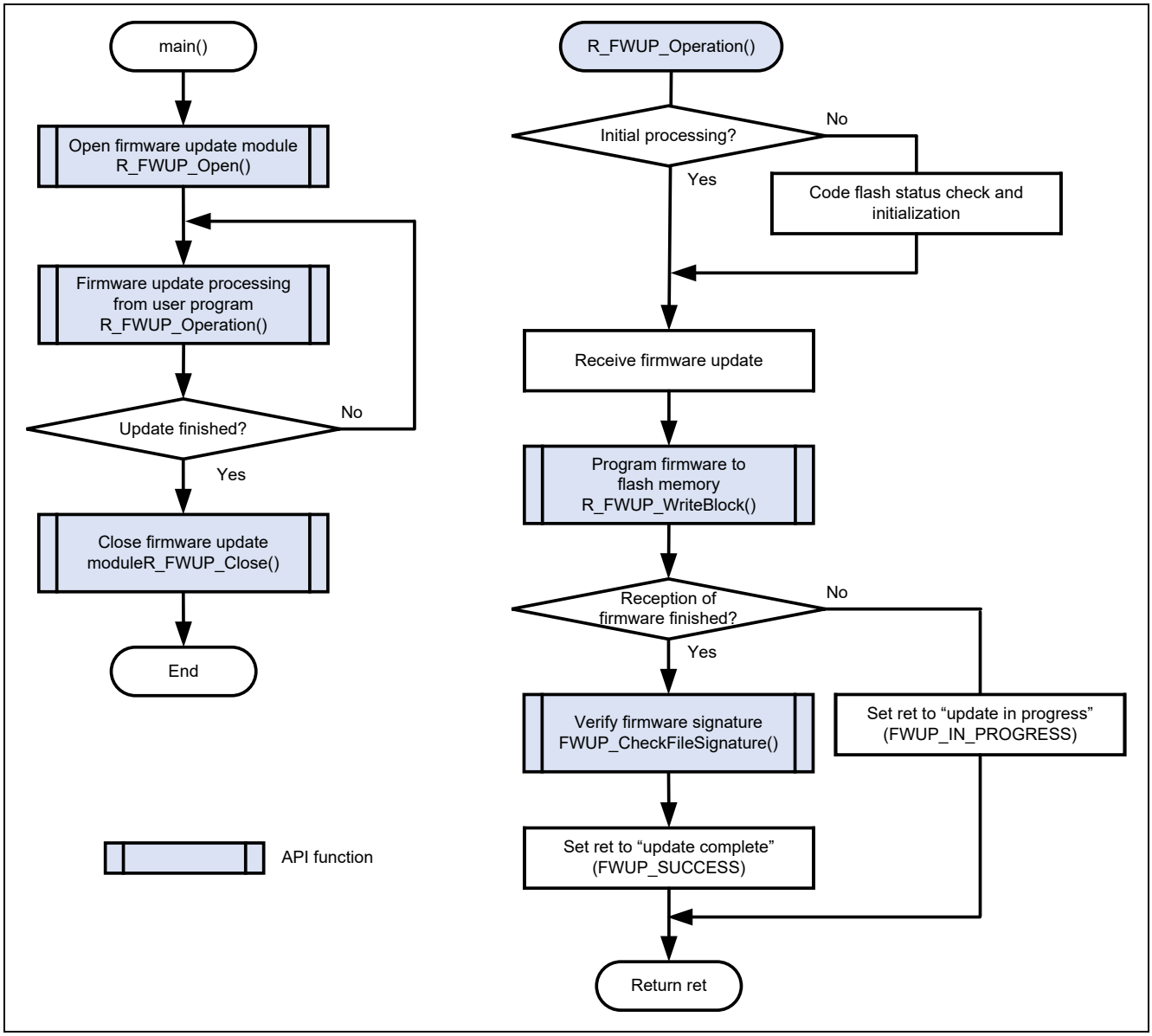


Figure 1.10 Firmware Update Operation on OS-Less System

The system comprises processing for opening the firmware update module (R_FWUP_Open), firmware update processing (R_FWUP_Operation), and closing the firmware update module (R_FWUP_Close).

In addition to communication control, the R_FWUP_Operation function handles a sequence of firmware update processing, including initialization of the flash memory, programming of the flash memory, and verification of the signature of the firmware.

1.4.2 Firmware Update with Module-External Communication Control

In this type of system, firmware reception processing is handled by the user program and the received data is programmed to the flash memory by APIs of the firmware update module.

An overview of firmware update operation on an OS-less system (with module-external communication control) is shown below.

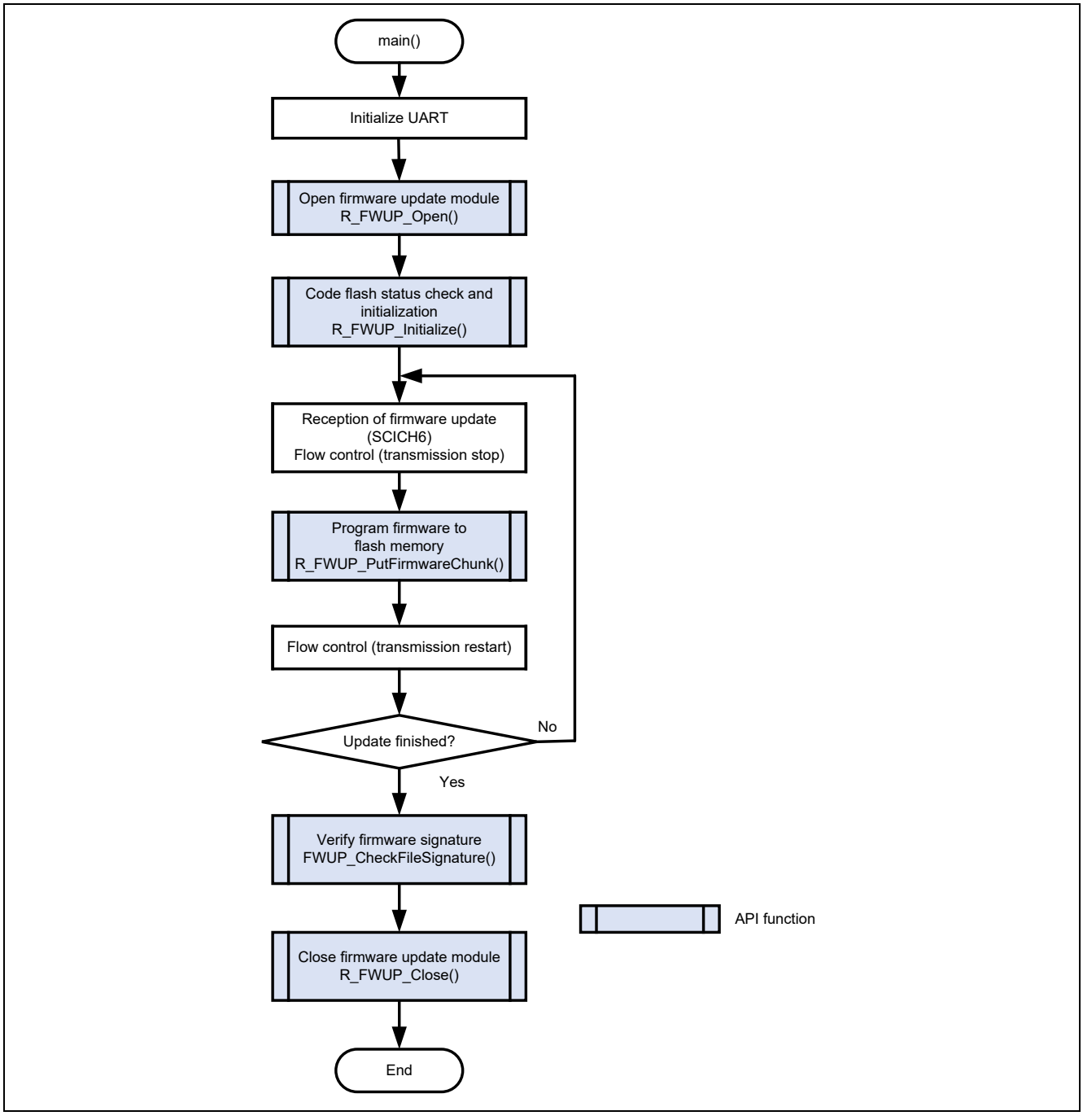


Figure 1.11 Firmware Update Operation on OS-Less System (with Module-External Communication Control)

The OS-less (module-external communication control) system comprises processing for opening the firmware update module (R_FWUP_Open), checking the status of and initializing the code flash (R_FWUP_Initialize), programming the flash memory (R_FWUP_PutFirmwareChunk), closing the firmware update module (R_FWUP_Close), and verifying the signature of the firmware (R_FWUP_CheckFileSigunature).

Processing such as initializing communication settings and receiving the firmware data (including flow control) are handled by the user program, which must control communication.

Regarding flow control, refer to the following code from the demo project (fwup_main_owSciDrv).

Flow control (transmission stop) is implemented within the UART receive callback function (uart_receive_fileblock).

```
static void uart_receive_fileblock(void *pArgs)
{
    sci_cb_args_t * p_args;

    p_args = (sci_cb_args_t *)pArgs;

    switch (p_args->event)
    {
        case SCI_EVT_RX_CHAR:

            /* From RXI interrupt; received character data is in p_args->byte */
            if ((s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size <
                  (sizeof(s_sci_receive_control_block.p_sci_buffer_control->buffer))) &&
                (USR_SCI_RECEIVE_BUFFER_EMPTY == s_sci_receive_control_block.p_sci_buffer_control->buffer_full_flag))
            {
                R_SCI_Receive(p_args->hdl, &s_sci_receive_control_block.p_sci_buffer_control->
                                buffer[s_sci_receive_control_block.p_sci_buffer_control->
                                buffer_occupied_byte_size++ ],
                                1);
                if ((sizeof(s_sci_receive_control_block.p_sci_buffer_control->buffer)) == s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size)
                {
                /* RTS HIGH */
                    USR_CFG_CSI_PORT_SYMBOL.PODR.BIT.USR_CFG_CSI_BIT_SYMBOL = 1;  /* Set RTS to HIGH */
                    USR_CFG_CSI_PORT_SYMBOL.PDR.BIT.USR_CFG_CSI_BIT_SYMBOL  = 1;
                    USR_CFG_CSI_PORT_SYMBOL.PMR.BIT.USR_CFG_CSI_BIT_SYMBOL  = 0;  /* Change to general I/O port */

                    s_sci_receive_control_block.total_byte_size +=
                    s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size;
                    s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size = 0;
                    s_sci_receive_control_block.p_sci_buffer_control->buffer_full_flag          = USR_SCI_RECEIVE_BUFFER_FULL;
```

Figure 1.12 Flow Control (Transmission Stop)

Immediately before flash memory programming (R_FWUP_PutFirmwareChunk), the processing waits for data to be received (R_BSP_SoftwareDelay). The standby duration is specified by FWUP_CFG_SCI_RECEIVE_WAIT, which is defined in r_fwup_config.h.

Flow control (transmission restart) takes place immediately after flash memory programming (R_FWUP_PutFirmwareChunk).

```
    R_BSP_SoftwareDelay(FWUP_CFG_SCI_RECEIVE_WAIT, BSP_DELAY_MICROSECS);  /* SCI receive wait */

    result_fwup = R_FWUP_PutFirmwareChunk (s_sci_receive_control_block.offset,
                                           firm_data, FWUP_WRITE_BLOCK_SIZE);

    USR_CFG_CSI_PORT_SYMBOL.PMR.BIT.USR_CFG_CSI_BIT_SYMBOL = 1;  /* Return port to UART flow control */
    printf("Flash Write: Address = 0x%X, length = %dbyte\n",
            TEMPORARY_AREA_LOW_ADDRESS + s_sci_receive_control_block.offset, FWUP_WRITE_BLOCK_SIZE);
    s_sci_receive_control_block.offset += FWUP_WRITE_BLOCK_SIZE;
```

Figure 1.13 Flow Control (Transmission Restart)

1.5 API Overview

Table 1.3 lists the API functions included in the firmware update module.

Table 1.3 API Functions

Function	Function Description	Firmware Update Module				Bootloader Module
		OS present		OS-less		
		Free RTOS (OTA)	Azure (ADU)	Module-External Communication Control*1	Module-Internal Communication Control	
R_FWUP_Open	Performs processing to open the module.	—	○	○	○	○
R_FWUP_Close	Performs processing to close the module.	—	○	○	○	○
R_FWUP_Initialize	Checks status of and erases code flash.	—	○	○	—	—
R_FWUP_Operation	Performs firmware update processing from the user program.	—	—	—	○	—
R_FWUP_PutFirmwareChunk	Writes a data block at the specified offset.	—	○	○	○	—
R_FWUP_SoftwareReset	Applies a software reset.	—	○	○	○	—
R_FWUP_DirectUpdate*2	Performs a firmware update without using a buffer area.	—	—	—	○	—
R_FWUP_SetEndOfLife	Performs end of life (EOL) processing for the user program.	○	○	○	○	—
R_FWUP_SecureBoot	Performs secure boot processing using the bootloader.	—	—	—	—	○
R_FWUP_ExecuteFirmware	Transfers processing to the installed or updated firmware.	—	—	—	—	○
R_FWUP_Abort	Stops OTA update processing.	○	—	—	—	—
R_FWUP_CreateFileForRx	Applies initial settings for OTA.	○	—	—	—	—
R_FWUP_CloseFile	Closes the specified file.	○	—	—	—	—
R_FWUP_WriteBlock	Writes a data block to the specified file at the specified offset.	○	—	—	—	—
R_FWUP_ActivateNewImage	Activates or launches the new firmware image.	○	—	—	—	—
R_FWUP_ResetDevice	The software resets. Then the new firmware boots via the boot loader.	○	—	—	—	—
R_FWUP_SetPlatformImageState	Sets the life cycle status to the status specified by an argument.	○	—	—	—	—
R_FWUP_GetPlatformImageState	Returns the current life cycle status.	○	—	—	—	—
R_FWUP_CheckFileSignature [OS-less usage]	Checks the signature of the firmware programmed to the flash memory.	—	—	○	—	—
R_FWUP_CheckFileSignature [OTA usage]	Checks the signature of the firmware programmed to the flash memory.	○	—	○	—	—

Function	Function Description	Firmware Update Module				Bootloader Module
		OS present		OS-less		
		Free RTOS (OTA)	Azure (ADU)	Module-External Communication Control*1	Module-Internal Communication Control	
R_FWUP_ReadAndAssumeCertificate	Reads and returns the specified signer certificate from the file system.	○	—	—	—	—
R_FWUP_GetVersion	Returns the version number of the module.	○	○	○	○	○

Notes: 1. Module-external communication control operation has been verified only for the partial overwrite method (CC-RX) on the RX66T.
2. Operation of the R_FWUP_DirectUpdate function has been verified only for the full overwrite method (CC-RX) on the RX66T.

2. API Information

The FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

- Flash memory
- Serial communications interface: optional
- Ethernet: optional
- System timer module

2.2 Software Requirements

The driver is dependent upon the following FIT module:

- Board support package (r_bsp)
- Byte queue buffer module (r_byteq)
- Compare match timer (r_cmt_rx)
- Flash module (r_flash_rx)
- Serial communications interface (SCI: asynchronous/clock synchronous) (r_sci_rx): optional
- Ethernet module (r_ether_rx): optional
- System timer module (r_sys_time_rx)

2.3 Supported Toolchain

The driver has been confirmed to work with the toolchain listed in 5.1, Confirmed Operation Environment.

2.4 Header Files

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

2.5 Integer Types

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

2.6 Compile Settings

The configuration option settings of the FIT 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_IMPLEMENTATION _ENVIRONMENT Note: The default is 0.	Specifies the user program environment where the FIT module will be implemented. The API functions that can be used differ depending on the implementation target. Enter one of the following setting values. 0: Implement in bootloader program (default). 1: Implement in user program firmware update program (OS-less system). 2: Implement in user program firmware update program (OS-less system with module-external communication control). 3: Implement in FreeRTOS (OTA) program. 4: Implement in Azure (ADU) program. More setting values can be added for additional implementation environments.
FWUP_CFG_COMMUNICATION _FUNCTION Note: The default is 0.	This configuration setting specifies the communication channel used to obtain the new version of the firmware used by the user program for the firmware update. Enter one of the following setting values. 0: Connection via SCI communication (default) 1: Connection via Ethernet communication 2: Connection via USB*1 3: Connection via SDHI*1 4: Connection via QSPI*1 More setting values can be added for additional communication channels.
FWUP_CFG_BOOT_PROTECT _ENABLE Note: The default is 0.	Turns boot protection on or off. 0: Boot protection disabled (default). 1: Boot protection enabled.*2
FWUP_CFG_OTA_DATA _STORAGE Note: The default is 0.	Specifies the storage destination for FreeRTOS (OTA) data. This setting is valid when OTA updating of FreeRTOS is performed. Also, ensure that the settings in the boot program and FreeRTOS (OTA) program match. 0: Data flash (default) 1: Code flash
FWUP_CFG_NO_USE_BUFFER Note: The default is 0.	Specifies the method of programming the firmware update. 0: Partial overwrite method (default) 1: Full overwrite method
FWUP_CFG_BOOTLOADER _LOG_DISABLE Note: The default is 0.	Suppresses display of character strings by sending printf statements to the terminal software in order to minimize ROM usage. This setting is enabled when FWUP_CFG_IMPLEMENTATION_ENVIRONMENT is set to “0”. 0: Display character strings in terminal software (default). 1: Do not display character strings in terminal software.

Configuration options in r_fwup _config.h	
FWUP_CFG_LOG_LEVEL Note: The default is 3.	Specifies the log output level. This setting is valid when FWUP_CFG_IMPLEMENTATION_ENVIRONMENT is set to 1. 0: No log output 1: Output of error messages only 2: Output of warnings and error messages 3: Information, warnings, and error messages (default) 4: All log output
FWUP_CFG_SERIAL_TERM_SCI Note: The default is 8.	Specifies the SCI channel used to download the firmware.
FWUP_CFG_SERIAL_TERM_SCI _BITRATE Note: The default is 115,200.	Specifies the UART baud rate setting used to download the firmware.
FWUP_CFG_SERIAL_TERM_SCI _INTERRUPT_PRIORITY Note: The default is 15.	Specifies the SCI interrupt priority level used when downloading the firmware.
FWUP_CFG_SCI_RECEIVE_WAIT Note: The default is 300.	Specifies the UART receive wait time after transmit ends (RTS set to HIGH). The setting unit is microseconds.
FWUP_CFG_PORT_SYMBOL Note: The default is PORTC on the RSK-RX231.	Specifies the port symbol of the I/O port used for RTS, the UART receive request pin.
FWUP_CFG_BIT_SYMBOL Note: The default is B4 on the RSK-RX231.	Specifies the bit symbol of the I/O port used for RTS, the UART receive request pin.

Notes: 1. This item is unsupported, so entering this setting value has no effect.
2. This function prevents the area where the bootloader is stored from being overwritten. Once boot protection is enabled it may not be possible to change the setting back to “boot protection disabled,” or to change the accessible area or startup area protection function settings, depending on the environment. Exercise due caution regarding the handling of the boot protection setting.

Some combinations of the configuration option settings FWUP_CFG_IMPLEMENTATION_ENVIRONMENT and FWUP_CFG_COMMUNICATION_FUNCTION are allowed and others are not. The allowed combinations are shown below.

Table 2.2 Allowable Compile Setting Combinations

		FWUP_CFG_COMMUNICATION_FUNCTION				
		0: SCI	1: Ethernet	2: USB	3: SDHI	4: QSPI
FWUP_CFG_IMPLEMENTATION_ENVIRONMENT	0: Bootloader program	0	—	—	—	—
	1: User program firmware update program (OS-less system) 2: User program firmware update program (OS-less system with module-external communication control)	1	—	2*1	—	3*1
	3: FreeRTOS (OTA) program	4*1	5	6*1	7*1	—
	4: Azure (ADU) program	—	8	—	—	—

Note: In the table above, a numeral represents the setting value of FWUP_ENV_COMMUNICATION_FUNCTION, and a dash (—) represents an invalid combination of settings.
This item is unsupported, so entering this setting value has no effect.

The conditions constituting a valid combination of the implementation environment setting and communication channel setting are retained as macros in r_fwup_private.h.

Table 2.3 Valid Combination Macro Values

Macro	Value	Description
FWUP_COMM_SCI_BOOTLOADER	0	Connect a PC (COM port) to the SCI, and perform bootloader processing.
FWUP_COMM_SCI_PRIMITIVE	1	Connect a PC (COM port) to the SCI, and obtain the new version of the firmware via terminal software.
FWUP_COMM_USB_PRIMITIVE	2	Connect a PC (COM port) to the USB, and obtain the new version of the firmware via terminal software.
FWUP_COMM_QSPI_PRIMITIVE	3	Connect an external storage device (an SD card) to the QSPI, and obtain the new version of the firmware.
FWUP_COMM_SCI_AFRTOS	4	Connect a wireless module (SX-ULPGN, BG96, etc.) to the SCI, and obtain the new version of the firmware using FreeRTOS over-the-air (OTA) updates.
FWUP_COMM_ETHER_AFRTOS	5	Connect via Ethernet, and obtain the new version of the firmware using FreeRTOS over-the-air (OTA) updates.
FWUP_COMM_USB_AFRTOS	6	Connect an LTE modem to the USB, and obtain the new version of the firmware using FreeRTOS over-the-air (OTA) updates.
FWUP_COMM_SDHI_AFRTOS	7	Connect a wireless module (Type 1DX, etc.) to the SDHI, and obtain the new version of the firmware using FreeRTOS over-the-air (OTA) updates.
FWUP_COMM_ETHER_AZURE	8	Connect via Ethernet (or Wi-Fi) and obtain the new version of the firmware from Azure (ADU).

When additional combinations of the implementation environment setting and communication channel setting are added, additional macro settings can be added.

ex.)

```
##define FWUP_COMM_SCI_BOOTLOADER    0    // Used for Bootloader with SCI connection from COM port.
#define FWUP_COMM_SCI_PRIMITIVE      1    // SCI connection from COM port using primitive R/W.
#define FWUP_COMM_USB_PRIMITIVE      2    // USB connection from COM port using primitive R/W.
#define FWUP_COMM_QSP_PRIMITIVE      3    // Connect external storage (SD card) to QSPI using primitive R/W.
#define FWUP_COMM_SCI_AFRTOS         4    // Connect wireless module to SCI with Amazon FreeRTOS.
#define FWUP_COMM_ETHER_AFRTOS       5    // Connect Eathernet with Amazon FreeRTOS.
#define FWUP_COMM_USB_AFRTOS         6    // Connect LTE modem to USB with Amazon FreeRTOS.
#define FWUP_COMM_SDHI_AFRTOS        7    // Connect wireless module to SDHI with Amazon FreeRTOS.
#define FWUP_COMM_ETHER_AZURE        8    // Connect Eathernet with Azure ADU.
```

2.6.1 Note on Compiling for RX130 and RX140 Environment

To use the FIT module on the RSK RX130 or RX140, change the setting of the board support package (BSP) configuration option for the user stack size (BSP_CFG_USTACK_BYTES) from the default value to 0x1000 (4 KB).

2.7 Code Size

The code sizes associated with the FIT module are listed in the table below.

One representative device is listed for each flash type.*1

Table 2.4 Code Sizes

ROM, RAM, and Stack Code Sizes					
		Memory Used			
Device	Category	C/C++ Compiler Package for RX Family	GCC for Renesas RX	IAR CC++ Compiler for RX	Remarks
RX65N (flash type 4)	ROM	3,213 bytes	3,220 bytes	3,052 bytes	boot_loader project
		4,560 bytes	3,891 bytes	3,243 bytes	fwup_main project
		4,560 bytes	3,891 bytes	1,579 bytes	eol_main project
		5,342 bytes	4,251 bytes	—	aws_demos project
	RAM	36,968 bytes	36,957 bytes	36,946 bytes	boot_loader project
		3,261 bytes	3,257 bytes	3,246 bytes	fwup_main project
		3,261 bytes	3,257 bytes	2,222 bytes	eol_main project
		1,504 bytes	1,500 bytes	—	aws_demos project
	Max. stack size used	1,184 bytes	836 bytes	1,527 bytes	boot_loader project
		2,712 bytes	2,356 bytes	2,480 bytes	fwup_main project
		2,712 bytes	2,356 bytes	1,284 bytes	eol_main project
		1,880 bytes	1,736 bytes	—	aws_demos project
RX66T (flash type 3)	ROM	4,448 bytes	3,476 bytes	3,274 bytes	boot_loader project
		4,194 bytes	3,573 bytes	3,132 bytes	fwup_main project
		4,192 bytes	3,573 bytes	1,528 bytes	eol_main project
		2,999 bytes	—	—	fwup_main woSciDrv project
	RAM	36,969 bytes	36,957 bytes	36,946 bytes	boot_loader project
		3,257 bytes	3,253 bytes	3,242 bytes	fwup_main project
		3,257 bytes	3,253 bytes	2,218 bytes	eol_main project
		3,257 bytes	—	—	fwup_main woSciDrv project
	Max. stack size used	1,412 bytes	864 bytes	1,560 bytes	boot_loader project
		2,672 bytes	2,332 bytes	2,472 bytes	fwup_main project
		2,668 bytes	2,332 bytes	1,276 bytes	eol_main project
RX231 (flash type 1)	ROM	1,088 bytes	—	—	boot_loader project
		4,329 bytes	3,393 bytes	3,266 bytes	fwup_main project
		4,335 bytes	3,686 bytes	3,134 bytes	eol_main project
	RAM	4,335 bytes	3,686 bytes	1,529 bytes	boot_loader project
		6,249 bytes	6,237 bytes	6,226 bytes	fwup_main project
		3,257 bytes	3,253 bytes	3,242 bytes	eol_main project
	Max. stack size used	3,257 bytes	3,253 bytes	2,218 bytes	boot_loader project
		1,412 bytes	856 bytes	1,556 bytes	fwup_main project
		2,668 bytes	2,344 bytes	2,468 bytes	eol_main project

Note: 1. Refer to the application note RX Family Flash Module Using Firmware Integration Technology (R01AN2184) for a detailed description of flash types.

[Conditions]

C/C++ Compiler Package for RX Family

- Optimization level: Level 2
- Link module optimization: Checked
- Optimization method: Code size optimization
- Remove unreferenced variables/functions: Unchecked
- FWUP_CFG_BOOTLOADER_LOG_DISABLE (Config): 1

GCC for Renesas RX

- Optimization level: Optimize size (-Os)
- Debug level: None
- Link options: -Wl,--no-gc-sections
- FWUP_CFG_BOOTLOADER_LOG_DISABLE (Config): 1

IAR C/C++ Compiler for RX

- Optimization level: High (Size)
- FWUP_CFG_BOOTLOADER_LOG_DISABLE (Config): 1

Reference: ROM and RAM usage of bootloader

The ROM and RAM usage of the bootloader project on various products is listed below for reference.

Table 2.5 ROM and RAM Usage of Bootloader

ROM and RAM Usage of Bootloader				
		Memory Used		
Device	Category	C/C++ Compiler Package for RX Family	GCC for Renesas RX	IAR C/C++ Compiler
RX130	ROM	36,240 bytes	52,092 bytes	25,298 bytes
	RAM	11,304 bytes	11,140 bytes	12,457 bytes
RX140	ROM	31,167 bytes	48,288 bytes	23,319 bytes
	RAM	11,803 bytes	11,684 bytes	15,183 bytes
RX231	ROM	35,516 bytes	50,324 bytes	24,747 bytes
	RAM	12,388 bytes	12,164 bytes	15,516 bytes
RX671	ROM	36,838 bytes	55,541 bytes	30,466 bytes
	RAM	41,230 bytes	40,868 bytes	45,724 bytes
RX65N	ROM	30,370 bytes	54,762 bytes	27,097 bytes
	RAM	41,186 bytes	43,388 bytes	43,660 bytes
RX66T	ROM	37,310 bytes	53,036 bytes	25,377 bytes
	RAM	43,628 bytes	43,268 bytes	45,403 bytes
RX660	ROM	38,840 bytes	51,932 bytes	24,722 bytes
	RAM	42,935 bytes	42,660 bytes	44,944 bytes
RX72N	ROM	39,296 bytes	55,818 bytes	30,730 bytes
	RAM	41,350 bytes	40,996 bytes	45,828 bytes

[Conditions]

C/C++ Compiler Package for RX Family

- Optimization level: Level 2
- Link module optimization: Checked
- Optimization method: Code size optimization
- Remove unreferenced variables/functions: Unchecked
- I/O function: Basic version
- FWUP_CFG_BOOTLOADER_LOG_DISABLE (Config): 1

GCC for Renesas RX

- Optimization level: Optimize size (-Os)
- Debug level: None
- Link options: -Wl,--no-gc-sections
- FWUP_CFG_BOOTLOADER_LOG_DISABLE (Config): 1

IAR C/C++ Compiler for RX

- Optimization level: High (Size)
- FWUP_CFG_BOOTLOADER_LOG_DISABLE (Config): 1

2.8 Arguments

Regarding structures used as API function arguments, the file context settings for the Amazon FreeRTOS (OTA) 202002.00 environment are used for other environments as well.

The reused structure is shown below.

Note: Settings that apply to Amazon FreeRTOS when using over-the-air (OTA) updates may change due to version upgrades or the like. You will therefore need to check for any setting changes when applying version upgrades.
Location of declaration in FreeRTOS environment using over-the-air (OTA) updates:
aws_demos¥libraries¥ota_for_aws¥source¥include¥ota_private.h

Table 2.6 OTA File Context

```
typedef struct
{
    uint16_t size;                              /*!< @brief Size, in bytes, of the signature. */
    uint8_t data[ kOTA_MaxSignatureSize ];  /*!< @brief The binary signature data. */
} Sig256_t;

typedef struct OtaFileContext
{
    uint8_t * pFilePath;            /*!< @brief Update file pathname. */
    uint16_t filePathMaxSize;       /*!< @brief Maximum size of the update file path */
    #if defined( WIN32 ) || defined( __linux__ )
        FILE * pFile;               /*!< @brief File type is stdio FILE structure after file is open for
write. */
    #else
        uint8_t * pFile;            /*!< @brief File type is RAM/Flash image pointer after file is open
for write. */
    #endif
    uint32_t fileSize;              /*!< @brief The size of the file in bytes. */
    uint32_t blocksRemaining;       /*!< @brief How many blocks remain to be received (a code
optimization). */
    uint32_t fileAttributes;        /*!< @brief Flags specific to the file being received (e.g. secure,
bundle, archive). */
    uint32_t serverFileID;          /*!< @brief The file is referenced by this numeric ID in the OTA
job. */
    uint8_t * pJobName;             /*!< @brief The job name associated with this file from the job
service. */
    uint16_t jobNameMaxSize;        /*!< @brief Maximum size of the job name. */
    uint8_t * pStreamName;          /*!< @brief The stream associated with this file from the OTA
service. */
    uint16_t streamNameMaxSize;     /*!< @brief Maximum size of the stream name. */
    uint8_t * pRxBlockBitmap;       /*!< @brief Bitmap of blocks received (for deduplicating and
missing block request). */
    uint16_t blockBitmapMaxSize;    /*!< @brief Maximum size of the block bitmap. */
    uint8_t * pCertFilepath;        /*!< @brief Pathname of the certificate file used to validate the
receive file. */
    uint16_t certFilePathMaxSize;   /*!< @brief Maximum certificate path size. */
    uint8_t * pUpdateUrlPath;       /*!< @brief Url for the file. */
    uint16_t updateUrlMaxSize;      /*!< @brief Maximum size of the url. */
    uint8_t * pAuthScheme;          /*!< @brief Authorization scheme. */
    uint16_t authSchemeMaxSize;     /*!< @brief Maximum size of the auth scheme. */
    uint32_t updaterVersion;        /*!< @brief Used by OTA self-test detection, the version of
Firmware that did the update. */
    bool isInSelfTest;              /*!< @brief True if the job is in self test mode. */
    uint8_t * pProtocols;           /*!< @brief Authorization scheme. */
```

```
    uint16_t protocolMaxSize;       /*!< @brief Maximum size of the supported protocols string.  */
    uint8_t * pDecodeMem;           /*!< @brief Decode memory.  */
    uint32_t decodeMemMaxSize;      /*!< @brief Maximum size of the decode memory.  */
    uint32_t fileType;              /*!< @brief The file type id set when creating the OTA job.  */
    Sig256_t * pSignature;          /*!< @brief Pointer to the file's signature structure.  */
} OtaFileContext_t;
```

2.9 Return Values

This section describes return values of API functions. This enumeration is located in r_fwup_if.h as are the prototype declarations of API functions.

Table 2.7 API Return Value Settings

```
typedef enum e_fwup_err
{
    FWUP_SUCCESS = 0,               //  Normally terminated.
    FWUP_FAIL,                      //  Illegal terminated.
    FWUP_IN_PROGRESS,               //  Firmware update is in progress.
    FWUP_END_OF_LIFE,               //  End Of Life process finished.
    FWUP_ERR_ALREADY_OPEN,          //  Firmware Update module is in use by another process.
    FWUP_ERR_NOT_OPEN,              //  R_FWUP_Open function is not executed yet.
    FWUP_ERR_IMAGE_STATE,           //  Platform image status not suitable for firmware update.
    FWUP_ERR_LESS_MEMORY,           //  Out of memory.
    FWUP_ERR_FLASH,                 //  Detect error of r_flash module.
    FWUP_ERR_COMM,                  //  Detect error of communication module.
    FWUP_ERR_STATE_MONITORING,      //  Detect error of state monitoring module.
} fwup_err_t;
```

2.10 Adding the FIT Module to Your Project

The module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the FIT Configurator in the IAR Embedded Workbench for Renesas RX environment
If you want to add a FIT module in the IAR Embedded Workbench for Renesas RX environment, use the RX Smart Configurator to add the FIT module to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

(4) Deleting unnecessary modules
When you add the firmware update FIT module to your project using Smart Configurator, additional dependent modules are added as well.
When adding the firmware update module to an OS-less system using module-external communication control, modules such as r_sys_time_rx and r_sci_rx are added because it is assumed that they may be used by the user application, even if they are not used by the firmware update module. You can delete any unnecessary modules if you wish.

2.11 Note on Status Transition Monitoring Using System Timer

The module uses the system timer to perform status transition monitoring, and the specification stipulates that an error end occurs when more than the specified duration elapses without a status transition. The default value is one minute. Take appropriate measures to ensure that the status does not remain fixed for longer than the specified duration.

3. API Functions

3.1 R_FWUP_Open Function

Table 3.1 R_FWUP_Open Function Specifications

Format	fwup_err_t R_FWUP_Open (void)	
Description	Performs processing to open the firmware update module and bootloader module. Performs processing to open the resources used by the firmware update module and bootloader module, makes OS initial settings (when using an OS), and initializes variables.	
Parameters	None	
Return Values	FWUP_SUCCESS	: Normal end
	FWUP_ERR_ALREADY_OPEN	: Already open
	FWUP_ERR_FLASH	: Flash module error
	FWUP_ERR_COMM	: Communication module error
	FWUP_ERR_STATE_MONITORING	: Status transition monitoring module error
Special Notes	—	

3.2 R_FWUP_Close Function

Table 3.2 R_FWUP_Close Function Specifications

Format	fwup_err_t R_FWUP_Close (void)	
Description	Performs processing to close the firmware update module and bootloader module. Performs processing to close the resources used by the firmware update module and bootloader module, and makes OS end settings (when using an OS).	
Parameters	None	
Return Values	FWUP_SUCCESS	: Normal end
	FWUP_ERR_NOT_OPEN	: Not open
Special Notes	—	

3.3 R_FWUP_Initialize Function

Table 3.3 R_FWUP_Initialize Function Specifications

Format	fwup_err_t R_FWUP_Initialize (void)	
Description	Checks the status of the flash memory and erases the flash memory before the firmware update is performed by the user program. • If the status of the flash memory to be updated is other than VALID or INITIAL_FIRM_INSTALLING, the firmware cannot be updated, so a value of FWUP_ERR_IMAGE_STATE is returned. • If the return value is FWUP_IN_PROGRESS, the function ends normally. Use FirmwareChunk()function to continue firmware update processing.	
Parameters	None	
Return Values	FWUP_IN_PROGRESS	: Normal end Continue firmware update.
	FWUP_ERR_IMAGE_STATE	: Updating not possible in current flash memory status
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.4 R_FWUP_Operation Function

Table 3.4 R_FWUP_Operation Function Specifications

Format	fwup_err_t R_FWUP_Operation (void)	
Description	Performs firmware update processing from the user program. Obtains the firmware data to be applied as an update from the communication channel specified in the configuration settings, programs the flash memory, and performs signature verification. • If the status of the flash memory to be updated is other than VALID or INITIAL_FIRM_INSTALLING, the firmware cannot be updated, so a value of FWUP_ERR_IMAGE_STATE is returned. • If the return value is FWUP_IN_PROGRESS, a firmware update is currently in progress, so call this function again later. • If the return value is FWUP_SUCCESS, the firmware update is complete. Call the R_FWUP_SoftwareReset function. Processing transitions to the new firmware after a software reset is applied. • If the return value is FWUP_FAIL, the firmware update failed. Cancel the error and call this function again.	
Parameters	None	
Return Values	FWUP_SUCCESS	: Firmware update normal end
	FWUP_FAIL	: Firmware update error occurred
	FWUP_IN_PROGRESS	: Firmware update in progress
	FWUP_ERR_NOT_OPEN	: Not open
	FWUP_ERR_IMAGE_STATE	: Updating not possible in current flash status
	FWUP_ERR_FLASH	: Flash module error
	FWUP_ERR_STATE_MONITORING	: Firmware update status has not changed for more than specified duration
Special Notes	—	

3.5 R_FWUP_PutFirmwareChunk Function

Table 3.5 R_FWUP_PutFirmwareChunk Function Specifications

Format	fwup_err_t R_FWUP_PutFirmwareChunk (uint32_t ulOffset, uint8_t * const pData, uint32_t ulBlockSize)	
Description	Writes a data block at the specified offset. When the operation is successful, returns FWUP_IN_PROGRESS.	
Parameters	ulOffset	: Code flash write destination offset
	* pData	: Write data
	ulBlockSize	: Write data size
Return Values	FWUP_IN_PROGRESS	: Normal end
	FWUP_FAIL	: Error writing to code flash
Special Notes	When using this function in an OS-less environment it is necessary to set the ulOffset and ulBlockSize to a multiple of the minimum program size of the target area in the flash memory. In addition, ulBlockSize has a maximum value of 1024.	

3.6 R_FWUP_SoftwareReset Function

Table 3.6 R_FWUP_SoftwareReset Function Specifications

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

3.7 R_FWUP_DirectUpdate Function

Table 3.7 R_FWUP_DirectUpdate Function Specifications

Format	fwup_err_t R_FWUP_DirectUpdate (void)	
Description	Starts firmware update processing. Sets the status of the currently running firmware to BLANK and causes a reset to be generated. After the reset is cleared, the bootloader erases the current firmware and performs processing to apply the firmware update.	
Parameters	None	
Return Values	FWUP_FAIL	A firmware update error occurred.
Special Notes	This function is only used by the full overwrite method. When processing finishes successfully, the function generates a reset internally and does not return any values. When an error occurs, FWUP_FAIL is returned.	

3.8 R_FWUP_SetEndOfLife Function

Table 3.8 R_FWUP_SetEndOfLife Function Specifications

Format	fwup_err_t R_FWUP_SetEndOfLife (void)	
Description	Performs end of life processing for the user program. [Note] When the status is normal end (FWUP_SUCCESS) after this function is called, end of life (EOL) processing is not yet complete. To finish end of life (EOL) processing after this function runs, it is necessary to call the R_FWUP_SoftwareReset function to apply a software reset (software reset with bank swap in dual bank mode), and to execute the remaining end of life processing using the bootloader.	
Parameters	None	
Return Values	FWUP_SUCCESS	: Normal end
	FWUP_ERR_NOT_OPEN	: Not open
	FWUP_ERR_FLASH	: Flash module error
Special Notes	—	

3.9 R_FWUP_SecureBoot Function

Table 3.9 R_FWUP_SecureBoot Function Specifications

Format	int32_t R_FWUP_SecureBoot (void)	
Description	Performs secure boot processing using the bootloader. • Performs signature verification to check for tampering before allowing the newly installed firmware to run. • If no firmware is installed, the function obtains the firmware data to be applied as an update from the communication channel specified in the configuration settings, programs the flash memory, and performs signature verification. • If the firmware to be applied as an update is specified by the user program, it is substituted as the startup firmware. • If end of life (EOL) processing is specified by the user program, this function erases the firmware. • If the return value is FWUP_IN_PROGRESS, a secure boot is currently in progress, so call this function again later. • If the return value is FWUP_SUCCESS, the secure boot is complete. Call the R_FWUP_ExecuteFirmware function to transition processing to the newly installed or updated firmware. • If the return value is “FWUP_END_OF_LIFE”, the processing at the end of life (EOL) of the user program is complete. • If the return value is FWUP_FAIL, the secure boot failed. If necessary, cancel the error and call this function again. For full overwrite method • When the full overwrite method is used, the acquired firmware is written directly to the execution area in the code flash. If programming of the code flash and signature verification fail, the code flash is erased and FWUP_FAIL is returned. (At the next startup, the bootloader requests the initial firmware.)	
Parameters	None	
Return Values	FWUP_SUCCESS	: Secure boot normal end
	FWUP_FAIL	: Secure boot error occurred
	FWUP_IN_PROGRESS	: Secure boot in progress
	FWUP_END_OF_LIFE	: End of life (EOL) processing completed
	FWUP_ERR_NOT_OPEN	: Not open
	FWUP_ERR_STATE_MONITORING	: Firmware update status has not changed for more than specified duration
Special Notes	—	

3.10 R_FWUP_ExecuteFirmware Function

Table 3.10 R_FWUP_ExecuteFirmware Function Specifications

Format	void R_FWUP_ExecuteFirmware (void)
Description	Transfers processing to the installed or updated firmware. [Note] The start address of the firmware to which processing is transferred may differ depending on the MCU family or series. It may be necessary to implement processing to obtain the firmware start address to match the implementation environment. [Example: RX65N] Transfer processing to the address set in macro USER_RESET_VECTOR_ADDRESS.
Parameters	None
Return Values	None
Special Notes	—

3.11 R_FWUP_Abort Function

Table 3.11 R_FWUP_Abort Function Specifications

Format	OtaPalStatus_t R_FWUP_Abort (OTA_FileContext_t * const C)	
Description	Stops OTA update processing.	
Parameters	* C	: File context
Return Values	OtaPalSuccess	: Normal end
	OtaPalFileClose	: File context close error
Special Notes	—	

3.12 R_FWUP_CreateFileForRx Function

Table 3.12 R_FWUP_CreateFileForRx Function Specifications

Format	OtaPalStatus_t R_FWUP_CreateFileForRx (OTA_FileContext_t * const C)	
Description	Applies initial settings for OTA. Creates a file to store the received data.	
Parameters	* C	: File context
Return Values	OtaPalSuccess	: Normal end
	OtaPalRxFileCreateFailed	: File creation error
Special Notes	—	

3.13 R_FWUP_CloseFile Function

Table 3.13 R_FWUP_CloseFile Function Specifications

Format	OtaPalStatus_t R_FWUP_CloseFile (OTA_FileContext_t * const C)	
Description	Closes the specified file. Performs signature verification on the firmware image downloaded to a buffer area in a temporary area. Writes header information for the buffer area in the temporary area.	
Parameters	* C	: File context
Return Values	OtaPalSuccess	: Normal end
	OtaPalFileClose	: File close error
	OtaPalSignatureCheckFailed	: Signature verification error
Special Notes	—	

3.14 R_FWUP_WriteBlock Function

Table 3.14 R_FWUP_WriteBlock Function Specifications

Format	int16_t R_FWUP_WriteBlock (OTA_FileContext_t * const C, uint32_t ulOffset, uint8_t * const pacData, uint32_t ulBlockSize)	
Description	Writes a data block to the specified file at the specified offset. When the operation is successful, returns the number of bytes written.	
Parameters	* C	: File context
	ulOffset	: Code flash write destination offset
	* pacData	: Write data
	ulBlockSize	: Write data size
Return Values	R_OTA_ERR_QUEUE_SEND_FAIL (-2)	: Error writing to code flash
	Other than above:	: Number of bytes written to code flash
Special Notes	—	

3.15 R_FWUP_ActivateNewImage Function

Table 3.15 R_FWUP_ActivateNewImage Function Specifications

Format	OtaPalStatus_t R_FWUP_ActivateNewImage (void)	
Description	Activates or launches the new firmware image. Calls the R_FWUP_ResetDevice() function to apply a software reset.	
Parameters	None	
Return Values	OtaPalSuccess	: Normal end
Special Notes	—	

3.16 R_FWUP_ResetDevice Function

Table 3.16 R_FWUP_ResetDevice Function Specifications

Format	OtaPalStatus_t R_FWUP_ResetDevice (void)	
Description	Calling this function generates a software reset, after which the new firmware is launched through processing by the bootloader.	
Parameters	None	
Return Values	OtaPalSuccess	: Normal end
Special Notes	Close all open peripheral circuits before calling this function. This function does not return because a software reset occurs. If it returns, the system has not been reset or has an error.	

3.17 R_FWUP_SetPlatformImageState Function

Table 3.17 R_FWUP_SetPlatformImageState Function Specifications

Format	OtaPalStatus_t R_FWUP_SetPlatformImageState (OTA_ImageState_t eState)	
Description	Sets the life cycle status to the status specified by a parameter. When updating to the new firmware finishes, the function erases the buffer area in the temporary area.	
Parameters	eState	: Specified status
Return Values	OtaPalSuccess	: Normal end
	OtaPalCommitFailed	: Commit error
	OtaPalBadImageState	: The state of the specified OTA image is out of range
Special Notes	—	

3.18 R_FWUP_GetPlatformImageState Function

Table 3.18 R_FWUP_GetPlatformImageState Function Specifications

Format	OtaPalImageState_t R_FWUP_GetPlatformImageState (void)	
Description	Returns the current life cycle status.	
Parameters	None	
Return Values	OtaPalImageStatePendingCommit	: Waiting for update
	OtaPalImageStateValid	: Valid
	OtaPalImageStateInvalid	: Invalid
Special Notes	—	

3.19 R_FWUP_CheckFileSignature Function [OS-Less Usage]

Table 3.19 R_FWUP_CheckFileSignature Function Specifications

Format	fwup_err_t R_FWUP_CheckFileSignature(void)	
Description	Verifies the signature of the firmware programmed to the flash memory.	
Parameters	None	
Return Values	FWUP_SUCCESS	: Normal end
	FWUP_FAIL	: Signature verification error
Special Notes	—	

3.20 R_FWUP_CheckFileSignature Function [OTA Usage]

Table 3.20 R_FWUP_CheckFileSignature Function Specifications

Format	OtaPalStatus_t R_FWUP_CheckFileSignature (OTA_FileContext_t * const C)	
Description	Verifies the signature of the firmware programmed to the flash memory.	
Parameters	* C	: File context
Return Values	OtaPalSuccess	: Normal end
	OtaPalSignatureCheckFailed	: Signature verification error
	OtaPalBadSignerCert	: The signer certificate was unreadable or was zero in length
Special Notes	—	

3.21 R_FWUP_ReadAndAssumeCertificate Function

Table 3.21 R_FWUP_ReadAndAssumeCertificate Function Specifications

Format	uint8_t * R_FWUP_ReadAndAssumeCertificate (const uint8_t * const pucCertName uint32_t * const ulSignerCertSize)	
Description	Reads and returns the specified signer certificate from the file system.	
Parameters	* pucCertName	: Certificate file name
	* ulSignerCertSize	: Certificate size
Return Values	Pointer to certificate data	
Special Notes	—	

3.22 R_FWUP_GetVersion Function

Table 3.22 R_FWUP_GetVersion Function Specifications

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

The demo project comprises the FIT 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 4.1 are provided.

The firmware update demo consists of the following projects.

Dual mode folder structure: Under ☐☐\dualbank\△△\

Linear mode (partial overwrite) folder structure: Under ☐☐\non-dualbank2\△△\

Linear mode (full overwrite) folder structure: Under ☐☐\non-dualbank3\△△\

☐☐: Device name

△△: Compiler (ccrx/gcc/iar)

- boot_loader: Bootloader program
  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: User program (initial firmware/firmware update)
  This is a user program (initial firmware) that downloads the firmware update. This program can also be edited so that it can be used as the user program (firmware update).
- eol_main
  This program performs EOL processing. To ensure that the system is discarded safely, the EOL processing erases both the user program in the code flash and the contents of the data flash.

4.1 Demo Project List

The demo projects included in this package are shown below.

```
+-rx65n-rsk                          : Demo project set folder using RSK-RX65N starter kit
|  |  amazon-freertos-gcc.zip         :   Amazon FreeRTOS (OTA), CC-RX version demo project
|  |  amazon-freertos.zip             :   Amazon FreeRTOS (OTA), GCC version demo project
|  |
|  +-dualbank
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|      |      boot_loader.zip        :   Project of boot loader
|      |      eol_main.zip           :   Project of EOL processing
|      |      fwup_main.zip          :   Project of Firmware update
|      |
|      +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|      |      boot_loader_gcc.zip    :   Project of boot loader
|      |      eol_main_gcc.zip       :   Project of EOL processing
|      |      fwup_main_gcc.zip      :   Project of Firmware update
|      |
|      +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|             boot_loader_iar.zip    :   Project of boot loader
|             eol_main_iar.zip       :   Project of EOL processing
|             fwup_main_iar.zip      :   Project of Firmware update
|
+-rx66t-rsk                          : Demo project set folder using RSK-RX66T starter kit
|  +-non-dualbank2
|  |   +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|  |   |      boot_loader.zip        :   Project of boot loader
|  |   |      eol_main.zip           :   Project of EOL processing
|  |   |      fwup_main.zip          :   Project of Firmware update
|  |   |      fwup_main_woSciDrv.zip :   Project of Firmware update (w/o SCI Driver)
|  |   |
|  |   +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|  |   |      boot_loader_gcc.zip    :   Project of boot loader
|  |   |      eol_main_gcc.zip       :   Project of EOL processing
|  |   |      fwup_main_gcc.zip      :   Project of Firmware update
|  |   |
|  |   +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|  |          boot_loader_iar.zip    :   Project of boot loader
|  |          eol_main_iar.zip       :   Project of EOL processing
|  |          fwup_main_iar.zip      :   Project of Firmware update
|  |
|  +-non-dualbank3
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|             boot_loader.zip        :   Project of boot loader
|             eol_main.zip           :   Project of EOL processing
|             fwup_main.zip          :   Project of Firmware update
|
+-rx72n-rsk                          : Demo project set folder using RSK-RX72N starter kit
|   +-dualbank
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|      |      boot_loader.zip        :   Project of boot loader
|      |      eol_main.zip           :   Project of EOL processing
|      |      fwup_main.zip          :   Project of Firmware update
|      |
|      +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|      |      boot_loader_gcc.zip    :   Project of boot loader
|      |      eol_main_gcc.zip       :   Project of EOL processing
|      |      fwup_main_gcc.zip      :   Project of Firmware update
|      |
|      +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|             boot_loader_iar.zip    :   Project of boot loader
|             eol_main_iar.zip       :   Project of EOL processing
|             fwup_main_iar.zip      :   Project of Firmware update
|
+-rx130-rsk                          : Demo project set folder using RSK-RX130 starter kit
|  +-non-dualbank2
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|      |      boot_loader.zip        :   Project of boot loader
|      |      eol_main.zip           :   Project of EOL processing
|      |      fwup_main.zip          :   Project of Firmware update
|      |
|      +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|      |      boot_loader_gcc.zip    :   Project of boot loader
|      |      eol_main_gcc.zip       :   Project of EOL processing
|      |      fwup_main_gcc.zip      :   Project of Firmware update
|      |
|      +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|             boot_loader_iar.zip    :   Project of boot loader
|             eol_main_iar.zip       :   Project of EOL processing
|             fwup_main_iar.zip      :   Project of Firmware update
```

Figure 4.1 Demo Project List (1)

```
+-rx140-rsk                          : Demo project set folder using RSK-RX140 starter kit
|  +-non-dualbank2
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|      |      boot_loader.zip        :   Project of boot loader
|      |      eol_main.zip           :   Project of EOL processing
|      |      fwup_main.zip          :   Project of Firmware update
|      |
|      +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|      |      boot_loader_gcc.zip    :   Project of boot loader
|      |      eol_main_gcc.zip       :   Project of EOL processing
|      |      fwup_main_gcc.zip      :   Project of Firmware update
|      |
|      +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|             boot_loader_iar.zip    :   Project of boot loader
|             eol_main_iar.zip       :   Project of EOL processing
|             fwup_main_iar.zip      :   Project of Firmware update
|
+-rx231-rsk                          : Demo project set folder using RSK-RX231 starter kit
|  +-non-dualbank2
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|      |      boot_loader.zip        :   Project of boot loader
|      |      eol_main.zip           :   Project of EOL processing
|      |      fwup_main.zip          :   Project of Firmware update
|      |
|      +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|      |      boot_loader_gcc.zip    :   Project of boot loader
|      |      eol_main_gcc.zip       :   Project of EOL processing
|      |      fwup_main_gcc.zip      :   Project of Firmware update
|      |
|      +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|             boot_loader_iar.zip    :   Project of boot loader
|             eol_main_iar.zip       :   Project of EOL processing
|             fwup_main_iar.zip      :   Project of Firmware update
|
+-rx660-rsk                          : Demo project set folder using RSK-RX660 starter kit
|  +-dualbank
|      +-ccrx                        : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
|      |      boot_loader.zip        :   Project of boot loader
|      |      eol_main.zip           :   Project of EOL processing
|      |      fwup_main.zip          :   Project of Firmware update
|      |
|      +-gcc                         : Update firmware demo set folder : for GCC for Renesas RX
|      |      boot_loader_gcc.zip    :   Project of boot loader
|      |      eol_main_gcc.zip       :   Project of EOL processing
|      |      fwup_main_gcc.zip      :   Project of Firmware update
|      |
|      +-iar                         : Update firmware demo set folder : for IAR C/C++ Compiler for RX
|             boot_loader_iar.zip    :   Project of boot loader
|             eol_main_iar.zip       :   Project of EOL processing
|             fwup_main_iar.zip      :   Project of Firmware update
|
+-rx671-rsk                          : Demo project set folder using RSK-RX671 starter kit
   +-dualbank
      +-ccrx                         : Update firmware demo set folder : for C/C++ Compiler Package for RX Family
      |      boot_loader.zip         :   Project of boot loader
      |      eol_main.zip            :   Project of EOL processing
      |      fwup_main.zip           :   Project of Firmware update
      |
      +-gcc                          : Update firmware demo set folder : for GCC for Renesas RX
      |      boot_loader_gcc.zip     :   Project of boot loader
      |      eol_main_gcc.zip        :   Project of EOL processing
      |      fwup_main_gcc.zip       :   Project of Firmware update
      |
      +-iar                          : Update firmware demo set folder : for IAR C/C++ Compiler for RX
             boot_loader_iar.zip     :   Project of boot loader
             eol_main_iar.zip        :   Project of EOL processing
             fwup_main_iar.zip       :   Project of Firmware update
```

Figure 4.2 Demo Project List (2)

4.2 Building the Demo Project

Before performing a firmware update, the bootloader program, user program (initial firmware), and user program (firmware update) must be prepared.

4.2.1 Preparation Beforehand

Perform the following steps beforehand as preparation.

4.2.1.1 Preparing the Integrated Development Environment

The operation of the demo project has been confirmed in the environments described in 5.1. Prepare an integrated development environment that matches the device, compiler, etc., to be used to implement the demo.

4.2.1.2 Generating Public Key and Secret Key Information for Signature Verification

To ensure the integrity of the firmware updated by the project, the firmware update is digitally signed (ECDSA + SHA256), and the signature is used to verify its integrity. Public key information for signature verification must therefore be incorporated into the various programs.

In addition, secret key information for signature verification (secp256r1.privatekey) is used as a file that is input to Image Generator, as described in 4.3.

Refer to the webpage linked to below and generate the ECDSA + SHA256 key pair.

https://github.com/renesas/amazon-freertos/wiki/OTA の活用#手順まとめ

Refer to “4. ファームウェア検証に使用する鍵を OpenSSL で作成する” [4. Use OpenSSL to create a key for firmware verification.] and “5. ファームウェア検証に ECDSA+SHA256 を使用するため、ブートローダに署名検証用の公開鍵=secp256r1.publickey を仕込む” [5. To use ECDSA + SHA256 for firmware verification, embed the public key for signature verification (secp256r1.publickey) into the bootloader.] on the above webpage and generate a public key for signature verification (code_signer_public_key.h).

4.2.2 Bootloader Program

Extract the contents of the boot_loader.zip file of the project of your choice, create a workspace for the bootloader program, and import boot_loader.

Copy the public key for signature verification (code_signer_public_key.h) generated as described in 4.2.1.2 to the boot_loader\src\key folder, then build the project. A file called boot_loader.mot is created in the HardwareDebug folder. The boot_loader.mot file will be used as an input file by Image Generator (see 4.3).

4.2.3 User Program (Initial Firmware)

Extract the contents of the fwup_main.zip file of the project of your choice, create a workspace for the user program (initial firmware), and import fwup_main.

Copy the public key for signature verification (code_signer_public_key.h) generated as described in 4.2.1.2 to the boot_loader\src\key folder, then build the project. A file called fwup_main.mot is created in the HardwareDebug folder. The fwup_main.mot file will be used as an input file by Image Generator (see 4.3).

4.2.4 User Program (Firmware Update)

Extract the contents of the fwup_main.zip file of the project of your choice, create a workspace for the user program (firmware update), and import fwup_main.

Copy the public key for signature verification (code_signer_public_key.h) generated as described in 4.2.1.2 to the boot_loader\src\key folder.

Open the file fwup_main\src\main.c and remove the slashes from the left of the commented-out lines to make them valid.

```
main.c
//    printf("[FWUP_main DEMO] Firmware update demonstration completed.¥r¥n");
//    while(1)
//    {
//        /* infinity loop */
//    }
```

Build the project. A file called fwup_main.mot is created in the HardwareDebug folder. The fwup_main.mot file will be used as an input file by Image Generator (see 4.3).

4.3 Using Image Generator to Convert the Firmware Update Image File

Use Image Generator to convert the mot file generated as described in 4.2 into a file configured as a firmware update image (see 5.4).

To obtain Image Generator, download the entire contents of the mot-file-converter/Renesas Image Generator/bin/Debug/ folder from the URL below. (The files listed alongside Renesas Image Generator.exe are also necessary.)

Release mot file converter tool · renesas/mot-file-converter · GitHub

Double-click Renesas Image Generator.exe to launch Image Generator.

4.3.1 Generating a Bootloader and User Program (Initial Firmware) Image File

When the initial settings for the firmware update start from a state where the bootloader and user program (initial firmware) have already been programmed (Figure 1.4 step [4] for dual mode, Figure 1.6 step [2] for linear mode (partial overwrite), Figure 1.8 step [2] for linear mode (full overwrite)), the mot file generated by building the bootloader project (see 4.2.2) and the mot file generated by building the user program (initial firmware) (see 4.2.3), as well as the ECDSA + SHA256 secret key for signature verification (see 4.2.1.2), are input to Image Generator and converted into a mot file. Also, refer to 5.5.1 for a description of the image generation mechanism.

Launch Image Generator, set the parameters as shown below, and generate a mot file. The mot file created will be used in 4.4.1.

[1] Select the [Initial Firm] tab. ([Initial Firm] is selected by default.)
[2] For Select MCU under Settings, select the target MCU.
[3] For Select Firmware Verification Type under Settings, select sig-sha256-ecdsa.
[4] For Private Key Path under Settings, enter the path of the file generated as described in 4.2.1.2 (secp256r1.privatekey).
[5] For Select Output Format under Settings, select Bank0 User Program + Boot Loader (Motorola S Format).
[6] For File Path (Motorola Format) under Boot Loader, enter the path of the boot_loader.mot file generated as described in 4.2.2.
[7] For Firmware Sequence Number under Bank0 User Program, enter 1.
[8] For File Path (Motorola Format) under Bank0 User Program, enter the path of the fwup_main.mot file generated as described in 4.2.3.
[9] Click the [Generate] button and specify the file path of the userprog.mot (Motrola S format) file to be generated.

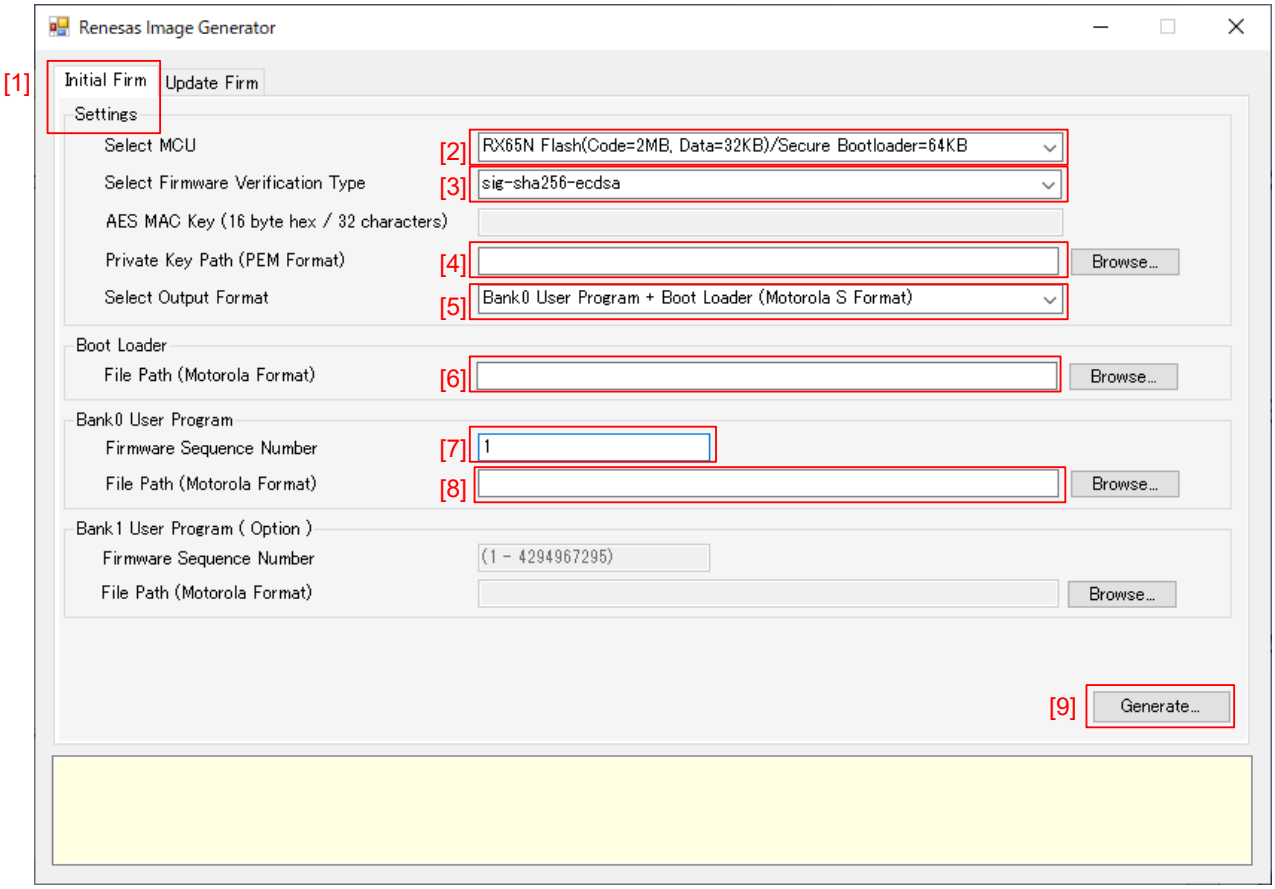


Figure 4.3 Generating a Bootloader and User Program (Initial Firmware) Image File

4.3.2 Generating a User Program (Firmware Update) RSU Image File

When generating a firmware update, the mot file generated by building the user program (firmware update) (see 4.2.4) and the ECDSA + SHA256 secret key for signature verification (see 4.2.1.2) are input to Image Generator and converted into an RSU file. Also, refer to 5.5.2 for a description of the image generation mechanism.

Launch Image Generator, set the parameters as shown below, and generate a RSU file. The RSU file created will be used in 4.4.

[1] Select the [Update Firm] tab. ([Initial Firm] is selected by default.)
[2] For Select MCU under Settings, select the target MCU.
[3] For Select Firmware Verification Type under Settings, select sig-sha256-ecdsa.
[4] For Private Key Path under Settings, enter the path of the file generated as described in 4.2.1.2 (secp256r1.privatekey).
[5] For Firmware Sequence Number under Bank0 User Program, enter 1.
[6] For File Path (Motorola Format) under Bank0 User Program, enter the path of the fwup_main.mot file generated as described in 4.2.4.
[7] Click the [Generate] button and specify the file path of the userprog.rsu (Renesas Secure Update) file to be generated.

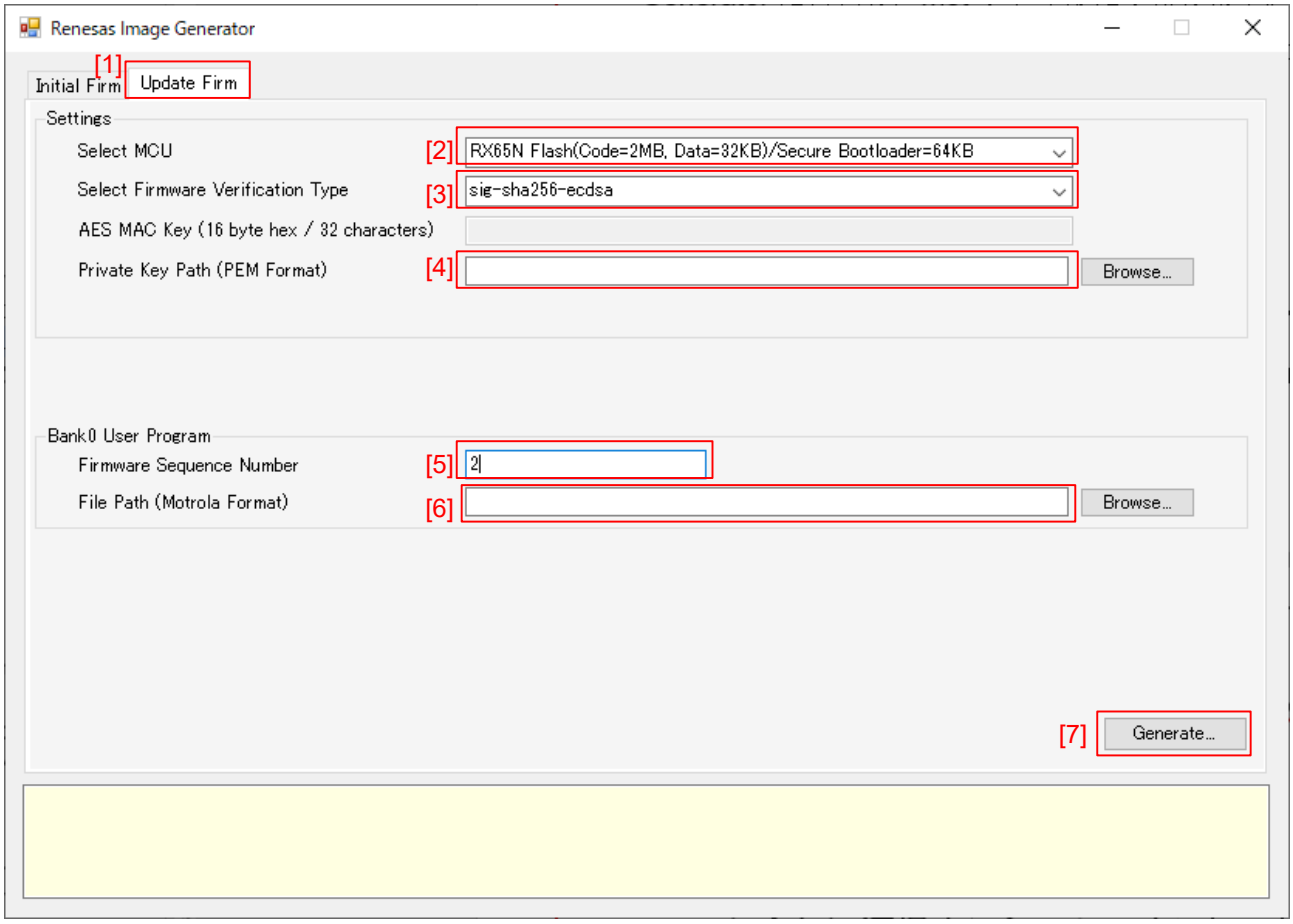


Figure 4.4 Generating a User Program (Firmware Update) RSU Image File

4.3.3 Generating a User Program (Initial Firmware) RSU Image File

When the initial settings for the firmware update start from a state where the bootloader has already been programmed (Figure 1.4 step [1] for dual mode, Figure 1.6 step [1] for linear mode (partial overwrite), Figure 1.8 step [1] for linear mode (full overwrite)), first the mot file generated by building the bootloader project (see 4.2.2) is written to the MCU board using Flash Programmer. Next, the mot file generated by building the user program (initial firmware) (see 4.2.3) and the ECDSA + SHA256 secret key for signature verification (see 4.2.1.2) are input to Image Generator and converted into an RSU file. Also, refer to 5.5.3 for a description of the image generation mechanism.

Launch Image Generator, set the parameters as shown below, and generate a RSU file. The RSU file created will be used in 4.4.

[1] Select the [Initial Firm] tab. ([Initial Firm] is selected by default.)
[2] For Select MCU under Settings, select the target MCU.
[3] For Select Firmware Verification Type under Settings, select sig-sha256-ecdsa.
[4] For Private Key Path under Settings, enter the path of the file generated as described in 4.2.1.2 (secp256r1.privatekey).
[5] For Select Output Format under Settings, select Bank0 User Program (Binary Format).
[6] For Firmware Sequence Number under Bank0 User Program, enter 1.
[7] For File Path (Motorola Format) under Bank0 User Program, enter the path of the fwup_main.mot file generated as described in 4.2.3.
[8] Click the [Generate] button and specify the file path of the userprog.rsu (Renesas Secure Update) file to be generated.

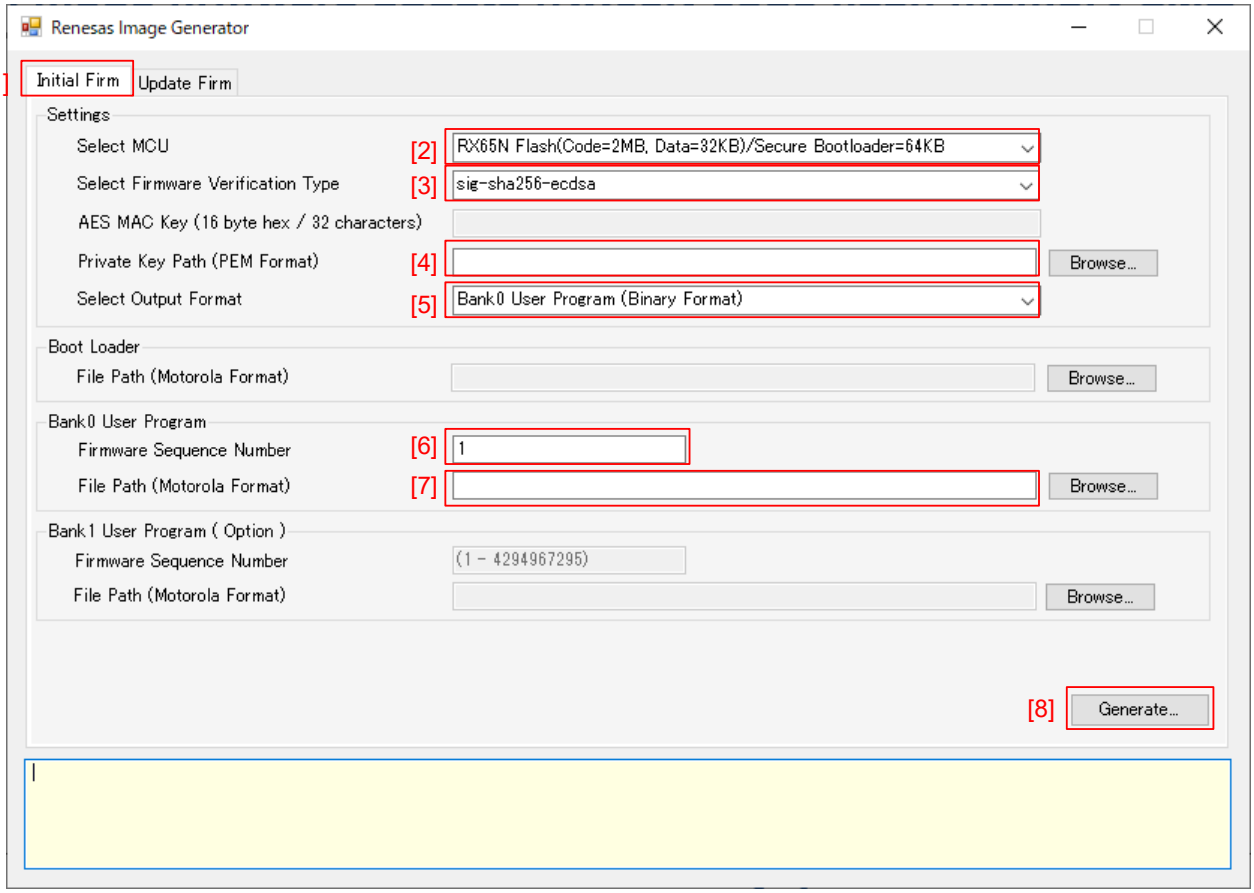


Figure 4.5 Generating a User Program (Initial Firmware) RSU Image File

4.4 Firmware Update Using Serial Communications Interface (SCI)

This section describes implementation of a firmware update demo using the serial communications interface (SCI) and dual mode, linear mode (partial overwrite), or linear mode (full overwrite). The firmware updating process involves communication with terminal software via SCI channels configured as a UART.

4.4.1 Dual Mode Firmware Update

In the example described below, the firmware update demo uses the serial communications interface (SCI) of the RX65N, which is mounted on the RSK RX65N starter kit board.

4.4.1.1 Preparing the Execution Environment

The firmware update demo uses serial port SCI6, which interfaces with the PMOD1. The PMOD1 connector is connected to a serial converter board.

A PC running terminal software is required for data input and output.

Table 4.1 Device Configuration

No.	Device	Description
1	Development PC	The PC used for development.
2	Evaluation board (Renesas Starter Kit for RX65N)	—
3	Host PC (running terminal software such as TeraTerm)	PC running serial communication software that supports XMODEM/SUM transfer protocol (The development PC may also be used for this purpose.) The operation of the demo has been confirmed using TeraTerm version 4.105.
4	USB serial converter board	Converts the serial I/O signals of the Renesas Starter Kit for RX65N to and from USB serial format and connects to the host PC via a USB cable.
5	USB cable	Implements a USB connection between the USB serial converter board and the host PC.

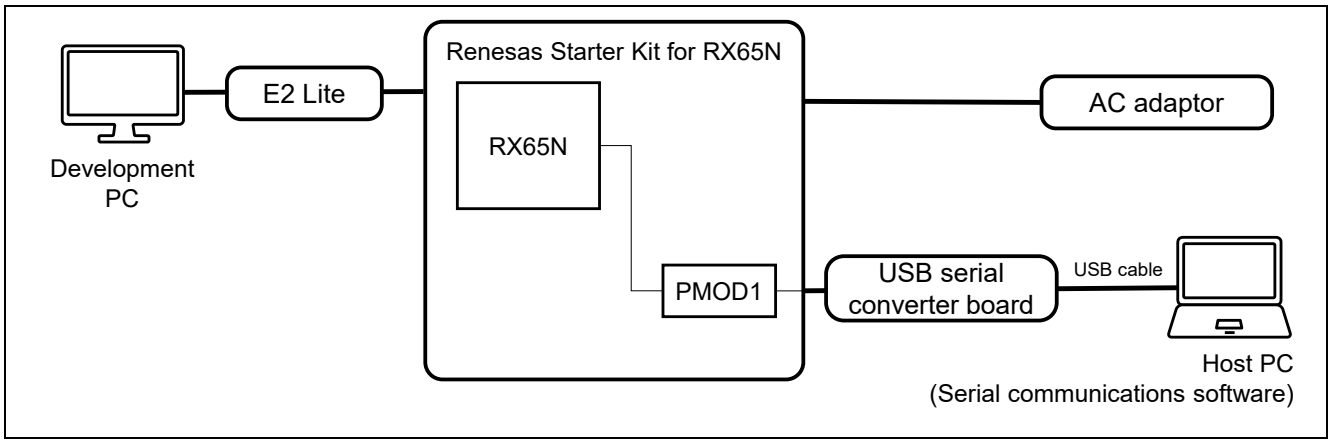


Figure 4.6 RSK RX65N Device Connection Diagram

Table 4.2 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	None

4.4.1.2 Programming the Bootloader and User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader and initial firmware have already been programmed (Figure 1.4 step [4]), the mot file of the bootloader and user program (initial firmware) generated as described in 4.3.1 is written to the MCU board using Flash Programmer. After programming is completed, the board should be powered off and disconnected from the emulator, etc.

4.4.1.3 Executing the Firmware Update

The initial firmware waits for the firmware update to be transferred via serial communication. The transferred program is then written to the code flash. After the transfer completes and the signature of the transferred firmware update has been verified, the firmware is updated (see 1.3.1).

Perform the steps below to apply the firmware update.

1. Connect the USB port of the PC to the USB serial converter board and the USB serial converter board to the PMOD1 connector of the RSK board as shown in Figure 4.6, RSK RX65N Device Connection Diagram.
2. Launch the terminal emulation program (TeraTerm 4.105) on the PC. Then select the serial COM port assigned to the USB serial converter board.
3. Enter serial communication settings in the terminal software to match the settings of the sample application: 115,200 bps, 8 data bits, no parity, 1 stop bit, no flow control.
4. Power on the board. The following message is output.

```
jump to user program
[INFO] Receive file created.
----------------------------------------
FIRMWARE UPDATE demo version 0.1.1
FWUP FIT module version 1.06
----------------------------------------
The firmware update will start.
```

   Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The following messages are output while the .RSU file data is being received and written to the code flash.

```
[INFO] Flash Write: Address = 0xFFE00000, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00400, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00800, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00C00, length = 1024byte ... OK
```

5. When installation and signature verification of the firmware update finish, execution jumps to the firmware update following a bank swap and other processing.

```
jump to user program
[INFO] Receive file created.
```

6. The firmware update outputs the following message indicating that the demo has completed successfully.

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

4.4.1.4 Programming the User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader has already been programmed (Figure 1.4 step [1]), the bootloader (mot file) built as described in 4.2.2 must be written to the MCU board beforehand using Flash Programmer.

Powering on the board launches the bootloader, which then waits for the initial firmware to be transferred via serial communication.

The transferred program is then written to the code flash, and after the transfer completes and the signature has been verified, the initial firmware is launched. After the initial firmware is launched, it waits for the firmware update. The transferred program is programmed to the code flash. After the transfer completes and the signature of the transferred firmware update has been verified, the firmware is updated (see 1.3.1). Perform the steps below to apply the firmware update.

1. Connect the USB port of the PC to the USB serial converter board and the USB serial converter board to the PMOD1 connector of the RSK board as shown in Figure 4.6, RSK RX65N Device Connection Diagram.
2. Launch the terminal emulation program on the PC. Then select the serial COM port assigned to the USB serial converter board.
3. Enter serial communication settings in the terminal software to match the settings of the sample application: 115,200 bps, 8 data bits, no parity, 1 stop bit, no flow control.
4. When the software is run, the following message is displayed.

```
send "userprog.rsu" via UART.
```

   Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.3. (Make sure to select the binary transfer option.) The following messages are output while the .RSU file data is being received and written to the code flash.

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

5. When installation and signature verification finish, the initial firmware is launched, and a message prompting you to input the firmware application is output.

```
jump to user program
[INFO] Receive file created.
---------------------------------------
FIRMWARE UPDATE demo version 0.1.1
FWUP FIT module version 1.06
---------------------------------------
The firmware update will start.
```

   Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The following messages are output while the .RSU file data is being received and written to the code flash.

```
[INFO] Flash Write: Address = 0xFFE00000, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00400, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00800, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00C00, length = 1024byte ... OK
```

6. When installation and signature verification of the firmware update finish, execution jumps to the firmware update following a bank swap and other processing.

```
jump to user program
[INFO] Receive file created.
```

7. The firmware update outputs the following message indicating that the demo has completed successfully.

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

4.4.2 Firmware Update Using Linear Mode (Partial Overwrite)

In the example described below, the firmware update demo uses the serial communications interface (SCI) of the RX66T, which is mounted on the RSK RX66T starter kit board.

4.4.2.1 Preparing the Execution Environment

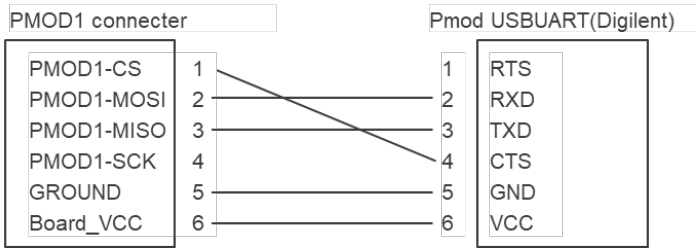
The firmware update demo uses serial port SCI6, which interfaces with the PMOD1. The PMOD1 connector is connected to a serial converter board.

A PC running terminal software is required for data input and output.

Table 4.3 Device Configuration

No.	Device	Description
1	Development PC	The PC used for development.
2	Evaluation board (Renesas Starter Kit for RX66T)	Short-circuit the J7 setting as a 5V power supply is required.
3	Host PC (running terminal software such as TeraTerm)	PC running serial communication software that supports XMODEM/SUM transfer protocol (The development PC may also be used for this purpose.) The operation of the demo has been confirmed using TeraTerm version 4.105.
4	USB serial converter board	Converts the serial I/O signals of the Renesas Starter Kit for RX66T to and from USB serial format and connects to the host PC via a USB cable. *1
5	USB cable	Implements a USB connection between the USB serial converter board and the host PC.

Note: 1. This demonstration project uses Digilent's Pmod USBUART for operation. Pmod USBUART and PMOD1 should be connected as follows.



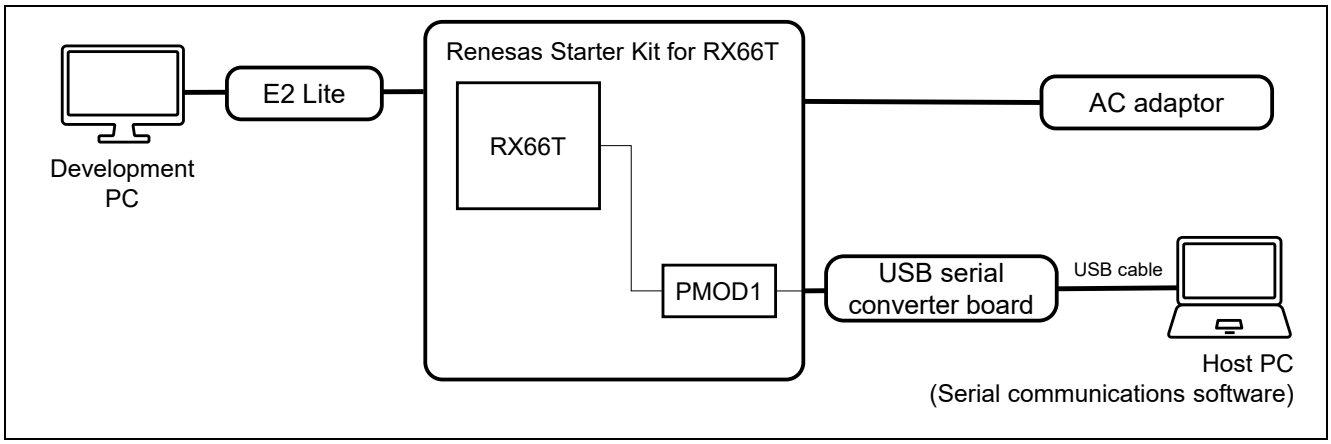


Figure 4.7 RSK RX66T Device Connection Diagram

Table 4.4 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	RTS/CTS

4.4.2.2 Programming the Bootloader and User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader and initial firmware have already been programmed (Figure 1.6 step [2]), the mot file of the bootloader and user program (initial firmware) generated as described in 4.3.1 is written to the MCU board using Flash Programmer. After programming is completed, the board should be powered off and disconnected from the emulator, etc.

4.4.2.3 Executing the Firmware Update

The initial firmware waits for the firmware update to be transferred via serial communication. The transferred program is then written to the code flash. After the transfer completes and the signature of the transferred firmware update has been verified, the firmware is updated (see 1.3.2.1).

Perform the steps below to apply the firmware update.

1. Connect the USB port of the PC to the USB serial converter board and the USB serial converter board to the PMOD1 connector of the RSK board as shown in Figure 4.7, RSK RX66T Device Connection Diagram.
2. Launch the terminal emulation program (TeraTerm 4.105) on the PC. Then select the serial COM port assigned to the USB serial converter board.
3. Enter serial communication settings in the terminal software to match the settings of the sample application: 115,200 bps, 8 data bits, no parity, 1 stop bit, flow control (RTS/CTS).
4. Power on the board. The following message is output.

```
jump to user program
[INFO] Receive file created.
---------------------------------------
FIRMWARE UPDATE demo version 0.1.1
FWUP FIT module version 1.06
---------------------------------------
The firmware update will start.
```

Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The transferred file data is received and written to user program area 1 by the initial firmware. The following messages are output while data is being received and written.

```
[INFO] Flash Write: Address = 0xFFF80000, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFF80400, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFF80800, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFF80C00, length = 1024byte ... OK
```

5. When installation and signature verification of the firmware update finish, the firmware update written to user program area 1 is copied to user program area 0, after which user program area 1 is erased, user program area 0 (the firmware update) is launched, and the program runs.

```
jump to user program
[INFO] Receive file created.
```

6. The firmware update outputs the following message indicating that the demo has completed successfully.

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

4.4.2.4 Programming the User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader has already been programmed (Figure 1.6 step [1]), the bootloader (mot file) built as described in 4.2.2 must be written to the MCU board beforehand using Flash Programmer.

Powering on the board launches the bootloader, which then waits for the initial firmware to be transferred via serial communication.

After the transferred program has been programmed to the code flash and data flash, the signature is verified and the initial firmware is launched. After the initial firmware is launched, it waits for the firmware update to be transferred. The transferred program is programmed to the code flash. After the transfer completes and the signature of the transferred firmware update has been verified, the firmware is updated (see 1.3.2.1). Perform the steps below to apply the firmware update.

1. Connect the USB port of the PC to the USB serial converter board and the USB serial converter board to the PMOD1 connector of the RSK board as shown in Figure 4.7, RSK RX66T Device Connection Diagram.
2. Launch the terminal emulation program on the PC. Then select the serial COM port assigned to the USB serial converter board.
3. Enter serial communication settings in the terminal software to match the settings of the sample application: 115,200 bps, 8 data bits, no parity, 1 stop bit, flow control (RTS/CTS).
4. When the software is run, the following message is displayed.

```
send "userprog.rsu" via UART.
```

   Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.3. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The following messages are output while the .RSU file data is being received and written to the code flash.

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

5. When installation and signature verification finish, the initial firmware is launched, and a message prompting you to input the firmware application is output.

```
jump to user program
[INFO] Receive file created.
----------------------------------------
FIRMWARE UPDATE demo version 0.1.1
FWUP FIT module version 1.06
----------------------------------------
The firmware update will start.
```

   Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The following messages are output while the .RSU file data is being received and written to the code flash.

```
[INFO] Flash Write: Address = 0xFFF80000, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFF80400, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFF80800, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFF80C00, length = 1024byte ... OK
```

6. When installation and signature verification of the firmware update finish, the firmware update written to user program area 1 is copied to user program area 0, after which user program area 1 is erased, user program area 0 (the firmware update) is relaunched, and the program runs.

```
jump to user program
[INFO] Receive file created.
```

7. The firmware update outputs the following message indicating that the demo has completed successfully.

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

4.4.3 Firmware Update Using Linear Mode (Full Overwrite)

In the example described below, the firmware update demo uses the serial communications interface (SCI) of the RX66T, which is mounted on the RSK RX66T starter kit board.

4.4.3.1 Preparing the Execution Environment

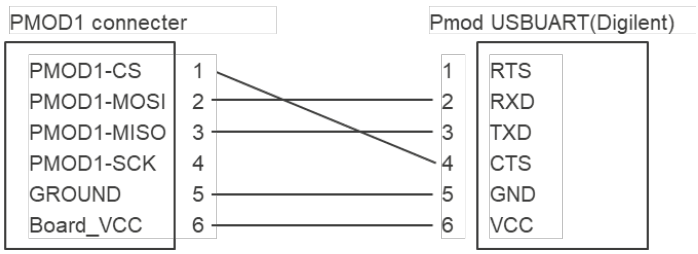
The firmware update demo uses serial port SCI6, which interfaces with the PMOD1. The PMOD1 connector is connected to a serial converter board.

A PC running terminal software is required for data input and output.

Table 4.5 Device Configuration

No.	Device	Description
1	Development PC	The PC used for development.
2	Evaluation board (Renesas Starter Kit for RX66T)	Short-circuit the J7 setting as a 5V power supply is required.
3	Host PC (running terminal software such as TeraTerm)	PC running serial communication software that supports XMODEM/SUM transfer protocol (The development PC may also be used for this purpose.) The operation of the demo has been confirmed using TeraTerm version 4.105.
4	USB serial converter board	Converts the serial I/O signals of the Renesas Starter Kit for RX66T to and from USB serial format and connects to the host PC via a USB cable. *1
5	USB cable	Implements a USB connection between the USB serial converter board and the host PC.

Note: 1. This demonstration project uses Digilent's Pmod USBUART for operation. Pmod USBUART and PMOD1 should be connected as follows.



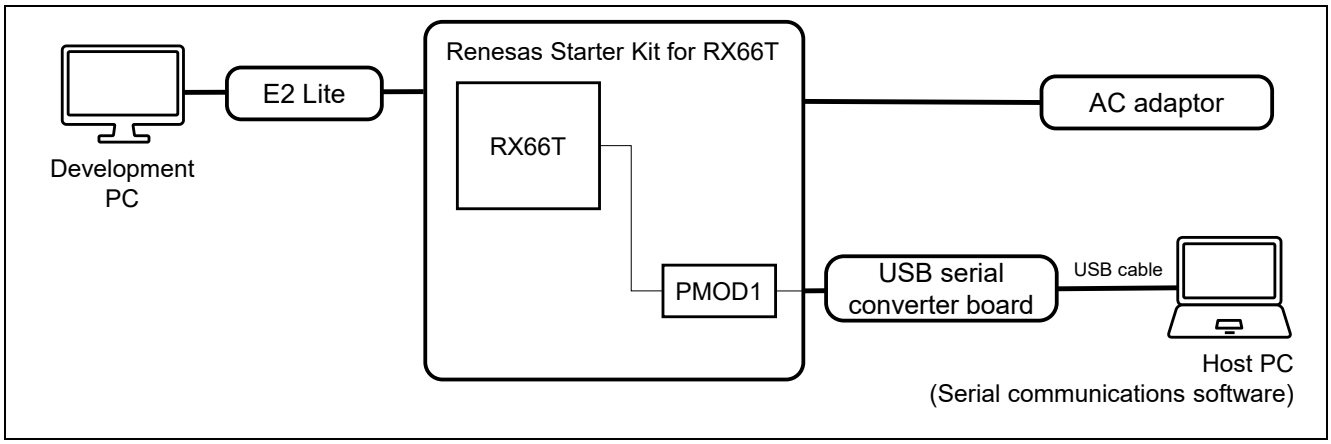


Figure 4.8 RSK RX66T Device Connection Diagram

Table 4.6 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	RTS/CTS

4.4.3.2 Programming the Bootloader and User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader and initial firmware have already been programmed (Figure 1.6 step [2]), the mot file of the bootloader and user program (initial firmware) generated as described in 4.3.1 is written to the MCU board using Flash Programmer. After programming is completed, the board should be powered off and disconnected from the emulator, etc. (When the initial settings for the firmware update start from a state in which only the bootloader has been programmed, the bootloader (mot file) built as described in 4.2.2 must be written to the MCU board beforehand using Flash Programmer.)

4.4.3.3 Executing the Firmware Update

Powering on the board launches the initial firmware, after which the image flag for executing the firmware update to the update file is cleared immediately, and a software reset occurs. The software reset causes the bootloader to be launched, which erases the user program area and waits for the firmware update to be transferred. The transferred program is then written to the code flash. After the transfer completes and the signature of the transferred firmware update has been verified, the firmware is updated (see 1.3.2.2).

Perform the steps below to apply the firmware update.

1. Connect the USB port of the PC to the USB serial converter board and the USB serial converter board to the PMOD1 connector of the RSK board as shown in Figure 4.8, RSK RX66T Device Connection Diagram.
2. Launch the terminal emulation program (TeraTerm 4.105) on the PC. Then select the serial COM port assigned to the USB serial converter board.
3. Enter serial communication settings in the terminal software to match the settings of the sample application: 115,200 bps, 8 data bits, no parity, 1 stop bit, flow control (RTS/CTS).
4. Power on the board. The following message is output.

```
------------------------------------------------
BOOTLOADER demo version 0.1.1
FWUP FIT module version 1.06
------------------------------------------------
RX66T secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
start installing user program.
========== install user program phase ==========
erase install area (data flash): OK
erase install area (code flash): OK
send "userprog.rsu" via UART.
```

Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The transferred file data is received and written to the user program area by the bootloader. The following messages are output while data is being received and written.

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

5. When installation and signature verification of the firmware update finish, execution jumps to the user program area (the firmware update), and the program runs.

```
jump to user program
[INFO] Receive file created.
```

6. The firmware update outputs the following message indicating that the demo has completed successfully.

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

4.4.3.4 Programming the User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader has already been programmed (Figure 1.8 step [1]), the bootloader (mot file) built as described in 4.2.2 must be written to the MCU board beforehand using Flash Programmer.

Powering on the board launches the bootloader, which then waits for the initial firmware to be transferred via serial communication.

Powering on the board launches the initial firmware, after which the image flag for executing the firmware update to the update file is cleared immediately, and a software reset occurs. The software reset causes the bootloader to be launched, which erases the user program area and waits for the firmware update to be transferred. The transferred program is then written to the code flash. After the transfer completes and the signature of the transferred firmware update has been verified, the firmware is updated (see 1.3.2.2).

Perform the steps below to apply the firmware update.

1. Connect the USB port of the PC to the USB serial converter board and the USB serial converter board to the PMOD1 connector of the RSK board as shown in Figure 4.8, RSK RX65N Device Connection Diagram.
2. Launch the terminal emulation program on the PC. Then select the serial COM port assigned to the USB serial converter board.
3. Enter serial communication settings in the terminal software to match the settings of the sample application: 115,200 bps, 8 data bits, no parity, 1 stop bit, flow control (RTS/CTS).
4. When the software is run, the following message is displayed.

```
send "userprog.rsu" via UART.
```

Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The following messages are output while the .RSU file data is being received and written to the code flash.

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

5. When installation and signature verification finish, the initial firmware is launched, and a message prompting you to input the firmware application is output.

```
jump to user program
[INFO] Receive file created.
----------------------------------------
FIRMWARE UPDATE demo version 0.1.1
FWUP FIT module version 1.06
----------------------------------------
The firmware update will start.
[INFO] Update ExeHeader ImageFlag : OK
[INFO] Resetting the device.
------------------------------------------------
BOOTLOADER demo version 0.1.1
FWUP FIT module version 1.06
------------------------------------------------
RX66T secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
start update user program.
========== install user program phase ==========
erase install area (data flash): SKIP
erase install area (code flash): OK
send update firmware via UART.
```

Select the “send file” function in the terminal software, and send the firmware update (.RSU file) generated as described in 4.3.2. (Make sure to select the binary transfer option.) Please start sending files within 1 minute. The following messages are output while the .RSU file data is being received and written to the code flash.

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

6. When installation and signature verification finish, execution jumps to the user program area (the firmware update), and the program runs.

```
jump to user program
[INFO] Receive file created.
```

7. The firmware update outputs the following message indicating that the demo has completed successfully.

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the FIT module.

Table 5.1 Confirmed Operation Environment (CC-RX)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2022 10
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.04.00 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.1.06
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX72N (product No.: RTK5572NNxxxxxxxBE) Renesas Starter Kit+ for RX671 (product No.: RTK55671EHS10000BE) Renesas Starter Kit for RX66T (product No.: RTK50566T0S00000BE) Renesas Starter Kit for RX660 (product No.: RTK556609HCxxxxxBJ) Renesas Starter Kit+ for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit for RX140-256KB (product No.: RTK551406BxxxxxBJ)
USB serial converter board	Pmod USBUART (Digilent, Inc.) https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

Table 5.2 Confirmed Operation Environment (GCC)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio 2022 10
C compiler	GCC for Renesas RX 8.3.0.202202 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
Endian order	Little endian
Revision of the module	Rev.1.06
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX72N (product No.: RTK5572NNxxxxxxxBE) Renesas Starter Kit+ for RX671 (product No.: RTK55671EHS10000BE) Renesas Starter Kit for RX66T (product No.: RTK50566T0S00000BE) Renesas Starter Kit for RX660 (product No.: RTK556609HCxxxxxBJ) Renesas Starter Kit+ for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit for RX140-256KB (product No.: RTK551406BxxxxxBJ)
USB serial converter board	Pmod USBUART (Digilent, Inc.) https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

Table 5.3 Confirmed Operation Environment (IAR)

Item	Contents
Integrated development environment	IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	IAR C/C++ Compiler for Renesas RX 4.20.3 Compiler option: The default settings of the integrated development environment
Endian order	RX smart configurator V2.14.0
Revision of the module	Little endian
Board used	Rev.1.06
USB serial converter board	Renesas Starter Kit+ for RX65N (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX72N (product No.: RTK5572NNxxxxxxxBE) Renesas Starter Kit+ for RX671 (product No.: RTK55671EHS10000BE) Renesas Starter Kit for RX66T (product No.: RTK50566T0S00000BE) Renesas Starter Kit for RX660 (product No.: RTK556609HCxxxxxBJ) Renesas Starter Kit+ for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit for RX140-256KB (product No.: RTK551406BxxxxxBJ)
Integrated development environment	Pmod USBUART (Digilent, Inc.) https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

The versions of the FIT modules used by the demo project to confirm firmware update operation are listed below.

(1) Renesas Electronics C/C++ Compiler Package for RX Family

Table 5.4 FIT Module Versions (CC-RX)

Device	Project	r_bsp	r_byteq	r_flash_rx	r_fwup	r_sys_time_rx	r_sci_rx	r_cmt_rx
RX130	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX140	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX231	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX65N	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
	aws_demos	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX66T (non-dualbank2)	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
	fwup_main_ woSciDrv	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX66T (non-dualbank3)	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX660	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX671	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX72N	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20

(2) GCC for Renesas RX

Table 5.5 FIT Module Versions (GCC)

Device	Project	r_bsp	r_byteq	r_flash_rx	r_fwup	r_sys_time_rx	r_sci_rx	r_cmt_rx
RX130	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX140	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX231	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX65N	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
	aws_demos	7.00	2.00	4.90	1.06	1.01	4.40	5.20
RX66T	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX660	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX671	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX72N	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20

(3) IAR C/C++ Compiler for RX

Table 5.6 FIT Module Versions (IAR)

Device	Project	r_bsp	r_byteq	r_flash_rx	r_fwup	r_sys_time_rx	r_sci_rx	r_cmt_rx
RX130	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX140	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX231	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX65N	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX66T	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX660	boot_loader fwup_main eol_main	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX671	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20
RX72N	boot_loader_gcc fwup_main_gcc eol_main_gcc	7.20	2.00	4.90	1.06	1.01	4.40	5.20

5.2 Compiler-Dependent Settings

This module supports multiple compilers. To use this module, different settings are required for each compiler as shown below.

5.2.1 Using Renesas Electronics C/C++ Compiler Package for RX Family

This section describes how to use Renesas Electronics C/C++ Compiler Package for RX Family as the compiler. The process of setting up the linker sections must be performed in e^2 studio.

5.2.1.1 Compiler Options

Add the following option to the default settings of the integrated development environment.

-lang = c99

5.2.1.2 Changing Address Assignments in Flash Memory

The linker section settings need to be changed in order to assign the bootloader and user program to execution areas in the flash memory.

1. In the **Project Explorer** view, click the project to be debugged.
2. Select **File** → **Properties** to open the **Properties** window.
3. In the **Properties** window, select **C/C++ Build** → **Settings**.
4. Select the **Tool Settings** tab, select **Linker** → **Action**, and click the **[...]** button to open the **Section Viewer** window.

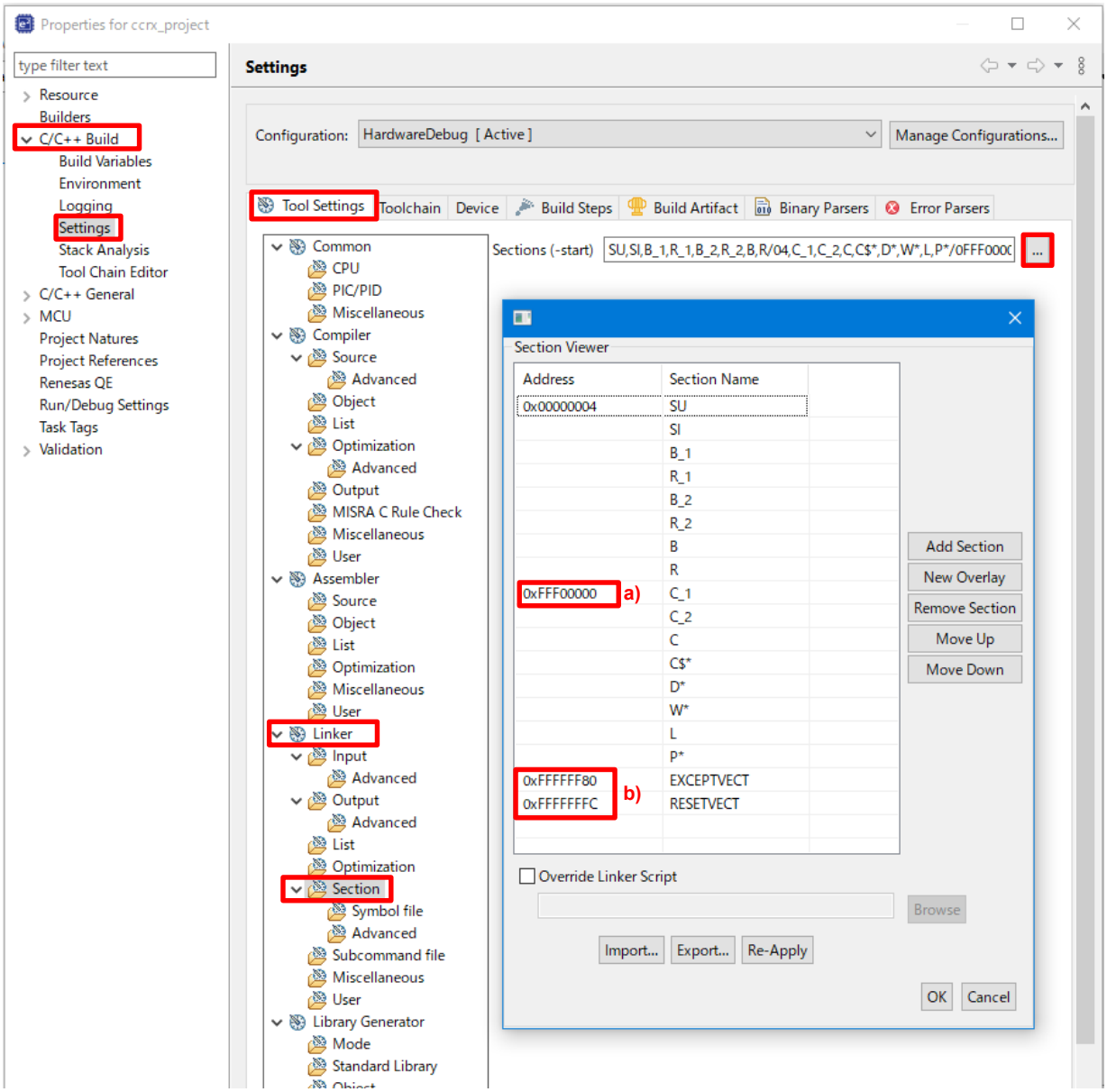


Figure 5.1 Section Settings in Renesas Electronics C/C++ Compiler Package for RX Family

5. Change the values of a) and b) in the **Section Viewer** window to match your environment.
   Example: The settings are as follows when using the RX65N in dual mode and the bootloader size is 64 KB.

Code	Description	Bootloader Settings	User Program Settings
a)	Start address in flash memory	0xFFFF0000	0xFFF00300
b)	Exception vector and reset vector addresses	0xFFFFFF80 0xFFFFFFFC	0xFFFEFF80 0xFFFEFFFC

5.2.1.3 Settings for Programming Flash Memory

Settings must be configured in order to write the user program and boot program to flash memory. Refer to the following application note for details of the settings.

Section 5.3.1, Using Renesas Electronics C/C++ Compiler Package for RX Family, in RX Family Flash Module Using Firmware Integration Technology (R01AN2184).

5.2.2 Using GCC for Renesas RX

This section describes how to use GCC for Renesas RX as the compiler. For the linker settings it is necessary to edit the linker settings file generated by e^2 studio.

5.2.2.1 Compiler Options

1. Compiler options: Add the following option to the default settings of the integrated development environment.
   -std=gnu99
2. Link options: When using the **Optimize size (-Os)** option, add the following options to the default settings of the integrated development environment.
   -Wl,--no-gc-sections
   This is a workaround to prevent the linker from mistakenly discarding interrupt handlers declared in FIT peripheral modules.
3. Compiler options: When debugging the bootloader, add the following option to the default settings of the integrated development environment.
   Optimization level: Optimize for debug (-Og)

5.2.2.2 Changing Address Assignments in Flash Memory

The linker settings need to be changed in order to assign the bootloader and user program to execution areas in the on-chip flash memory.

1. In the Project Explorer view, right-click the linker settings file (linker_script.ld) and select **Open**.
2. In the **linker_script.ld** window, click the **linker_script.ld** tab.

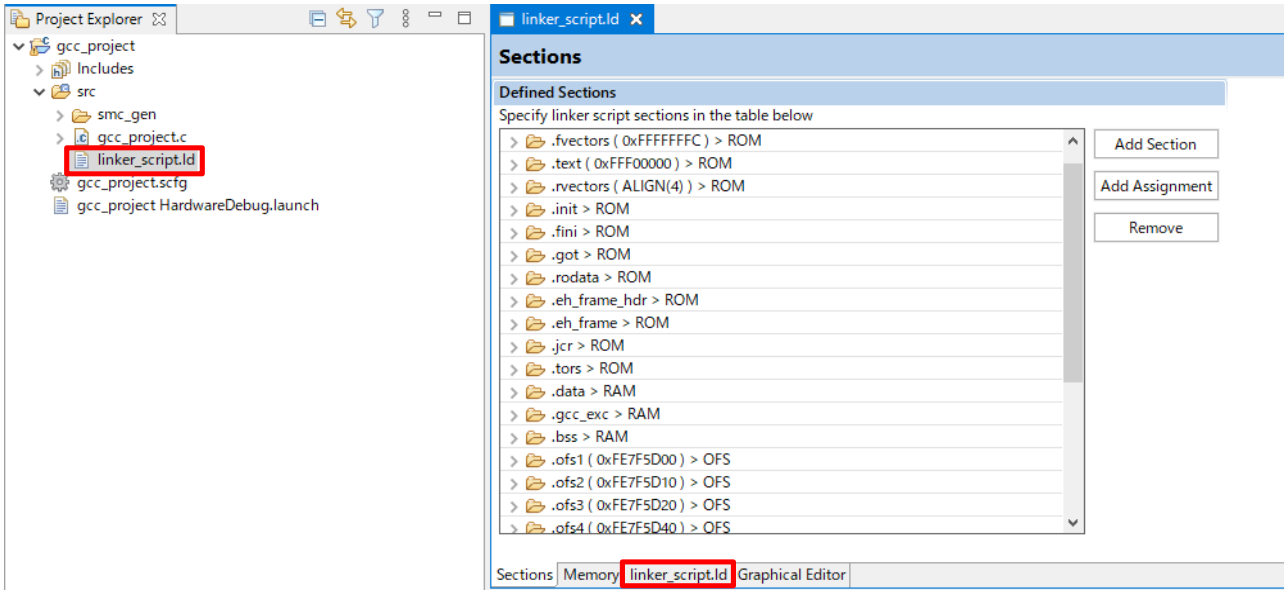


Figure 5.2 Section Settings in GCC for Renesas RX (1/2)

3. Change the values of a) to d) below to match your environment.

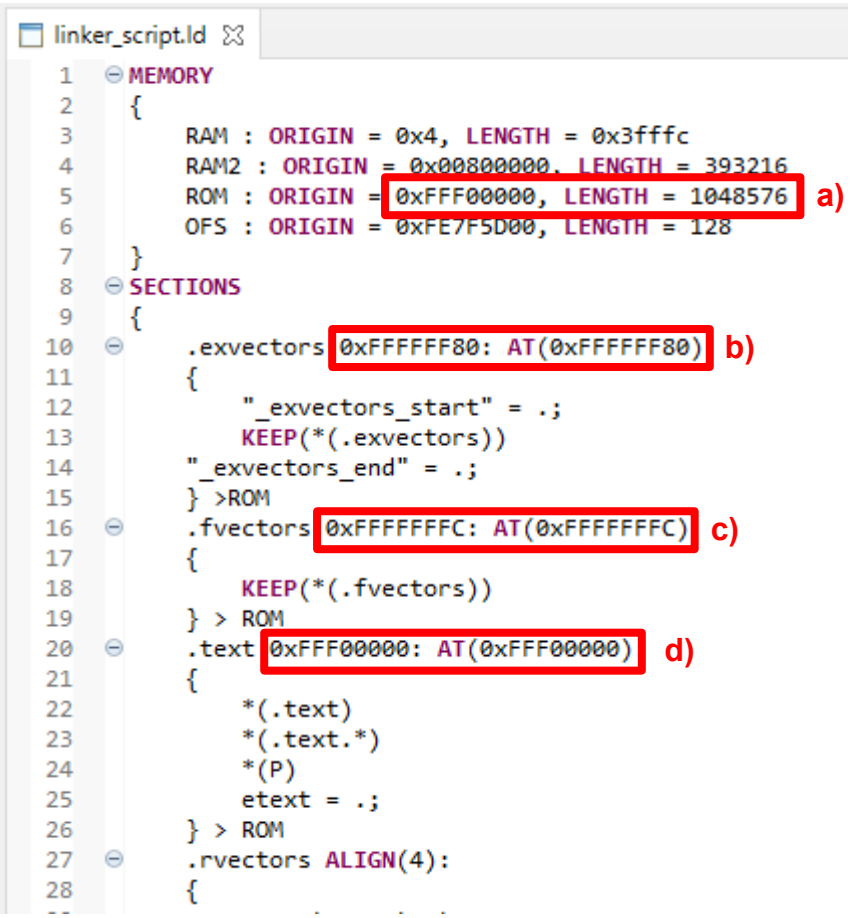


Figure 5.3 Section Settings in GCC for Renesas RX (2/2)

Example: The settings are as follows when using the RX65N in dual mode and the bootloader size is 64 KB.

Code	Description	Bootloader Settings	User Program Settings
a)	Code flash start address and code flash size	ORIGIN = 0xFFFF0000 LENGTH = 65536	ORIGIN = 0xFFF00300 LENGTH = 982272
b)	Exception vector address	0xFFFFFF80	0xFFFEFF80
c)	Reset vector address	0xFFFFFFFC	0xFFFEFFFC
d)	Code flash start address = same address as a)	0xFFFF0000	0xFFF00300

5.2.2.3 Settings for Programming Flash Memory

Settings must be configured in order to write the user program and boot program to flash memory. Refer to the following application note for details of the settings.

Section 5.3.2, Using GCC for Renesas RX, in RX Family Flash Module Using Firmware Integration Technology (R01AN2184).

5.2.2.4 Warning Message During Build

When building the FIT module, a warning message may appear indicating that the stack area used by the function exceeds the byte size specified by the -Wstack-usage option (“warning: stack usage is XXX bytes [-Wstack-usage=]”). (The default is 100 bytes.) If there is a problem, make appropriate changes to the build option settings.

5.2.3 Using IAR C/C++ Compiler for RX

This section describes how to use IAR C/C++ Compiler for RX as the compiler.

5.2.3.1 Compiler Options

In the project option settings of IAR Embedded Workbench for Renesas RX, set the output converter → output settings to output Motorola S-records.
Change the extension of the output file from the default “* .srec” to “* .mot”.

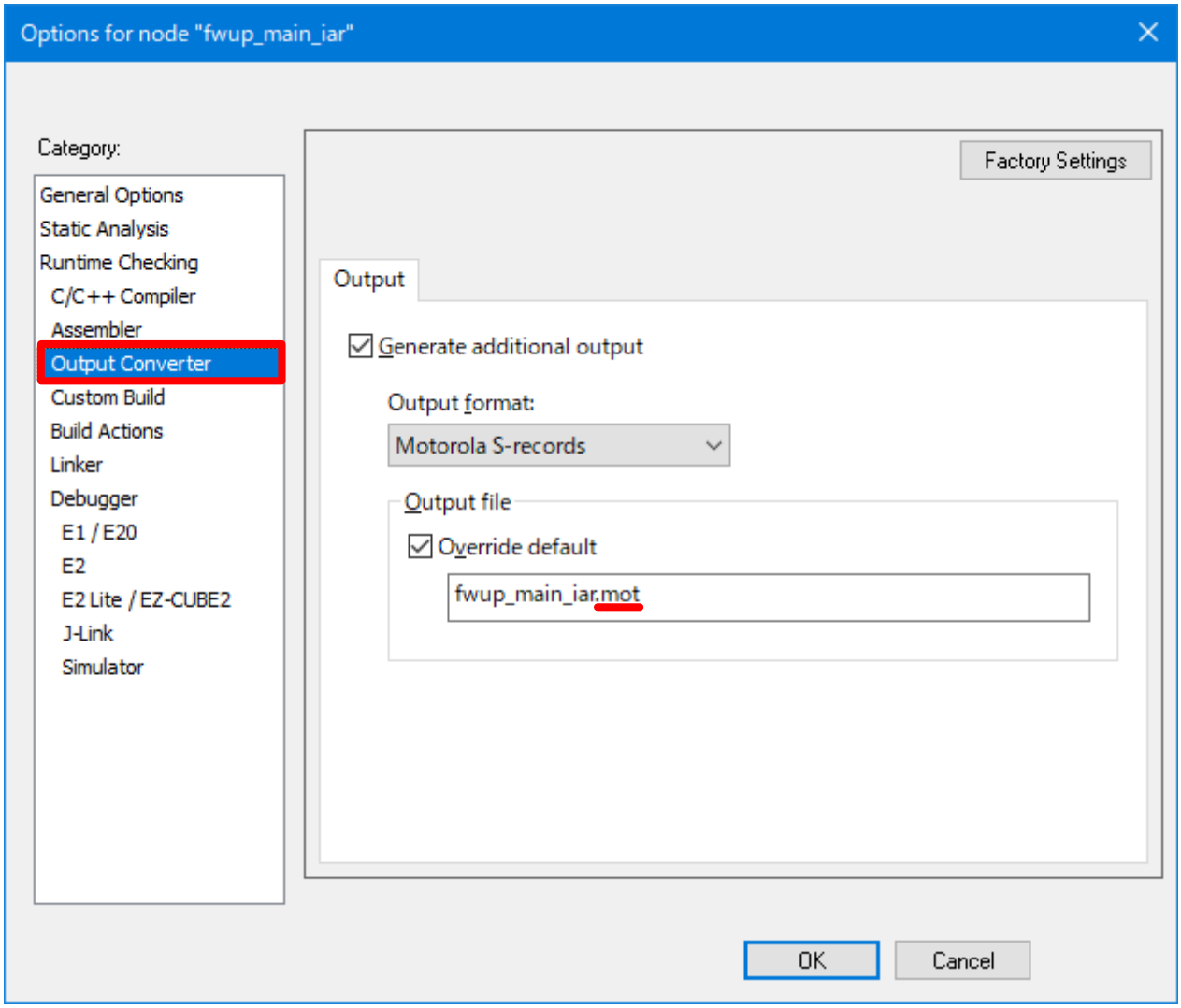


Figure 5.4 Changing the Extension of the Output File

5.2.3.2 Settings for Programming Flash Memory

Settings must be configured in order to write the user program and boot program to flash memory. Refer to the following application note for details of the settings.

Section 5.3.3, Using IAR C/C++ Compiler for Renesas RX, in RX Family Flash Module Using Firmware Integration Technology (R01AN2184).

5.2.3.3 Changing Address Assignments in Flash Memory

The linker settings file created as described in 5.2.3.2 needs to be changed in order to assign the bootloader and user program to execution areas in the on-chip flash memory.

1. Open the linker settings file (*.icf) created as described in 5.2.3.2 in an editor.
2. Change the following addresses (a) to (c) according to the user's environment.
   (Example: RX65N boot loader linker configuration file).

```
define region RAM_region1  = mem:[from 0x00000004 to 0x0003FFFF];
define region RAM_region2  = mem:[from 0x00800000 to 0x0085FFFF];

define region RAM_region16 = mem:[from 0x00000004 to 0x00007FFF];
define region RAM_region24 = RAM_region1 | RAM_region2;
define region RAM_region32 = RAM_region1 | RAM_region2;

define region STANDBY_RAM  = mem:[from 0x000A4000 to 0x000A5FFF];

define region ROM_region16 = mem:[from 0xFFFF0000 to 0xFFFFFFFF];
define region ROM_region24 = mem:[from 0xFFFF0000 to 0xFFFFFFFF];  a)
define region ROM_region32 = mem:[from 0xFFFF0000 to 0xFFFFFFFF];

define region DATA_FLASH   = mem:[from 0x00100000 to 0x00107FFF];
```

```
place at address mem:0xFE7F5D00      { ro section .option_mem };
place at address mem:0xFFFFFFFC  b)  { ro section .resetvect };
place at address mem:0xFFFFFF80  c)  { ro section .exceptvect };
```

The settings are as follows when using the RX65N in dual mode and the bootloader size is 64 KB.

Code	Description	Bootloader Settings	User Program Settings
a)	Code flash start address and code flash size	from 0xFFFF0000 to 0xFFFFFFFF	from 0xFFF00300 to 0xFFFEFFFF
b)	Reset vector address	0xFFFFFFFC	0xFFFEFFFC
c)	Exception vector address	0xFFFFFF80	0xFFFEFF80

5.3 Storage Destination for FreeRTOS Data (RX65N-2MB Only)

You can use a configuration option to select between the code flash and data flash as the storage destination for PKCS11 data (code signing certificate, etc.) used for OTA updating of FreeRTOS. This selection applies to RX65N-2MB products only.

5.3.1 Storage Destination Selection

The following configuration option is used to select the storage destination for PKCS11 data.

Note that this setting is valid when OTA updating of FreeRTOS is performed. Also, ensure that the settings in the boot program and FreeRTOS (OTA) program match.

FWUP_CFG_OTA_DATA_STORAGE

0: Data flash (default)

1: Code flash

The storage area in the data flash is 0x00100000 to 0x00107FFF (32 KB).

The storage area in the code flash is 0xFFE00000 to 0xFFE07FFF (32 KB).

5.3.2 Section Settings

When placing the PKCS11 data in the code flash, configure the section settings of the FreeRTOS (OTA) program as shown in the figure below.

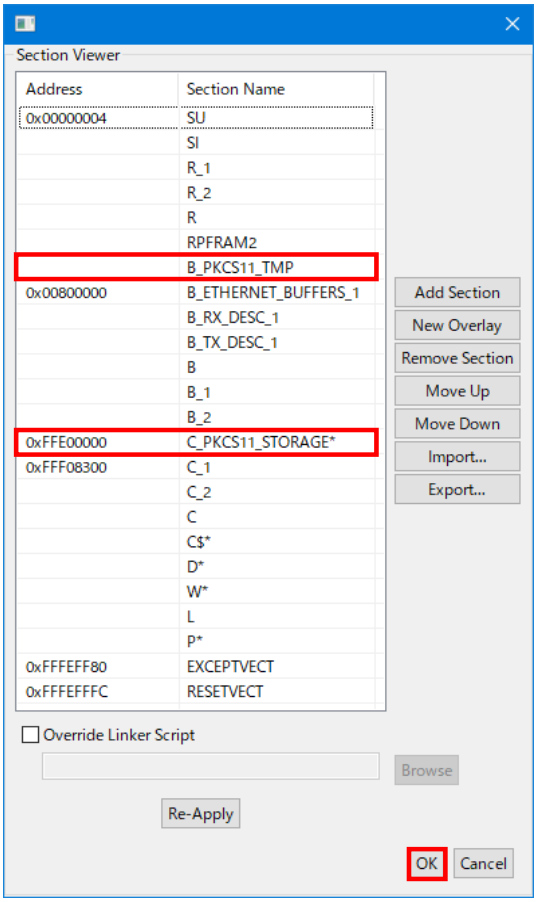


Figure 5.5 Section Settings when Code Flash Selected

When placing the PKCS11 data in the data flash, refer to the section settings in the sample program.

5.3.3 Conversion to .RSU File when Code Flash Selected

The method of converting to an .RSU file when the code flash is selected is described below.

Build the FreeRTOS (OTA) program, then use Renesas Image Generator to convert the resulting .mot file into an .RSU file.

In Renesas Image Generator, select the [Initial Firm] tab and set **Select MCU** to **RX65N Flash(Code=2MB, Data=0KB)**, then convert the file.

Refer to 4.3.3, Generating a User Program (Initial Firmware) RSU Image File, for the file conversion procedure.

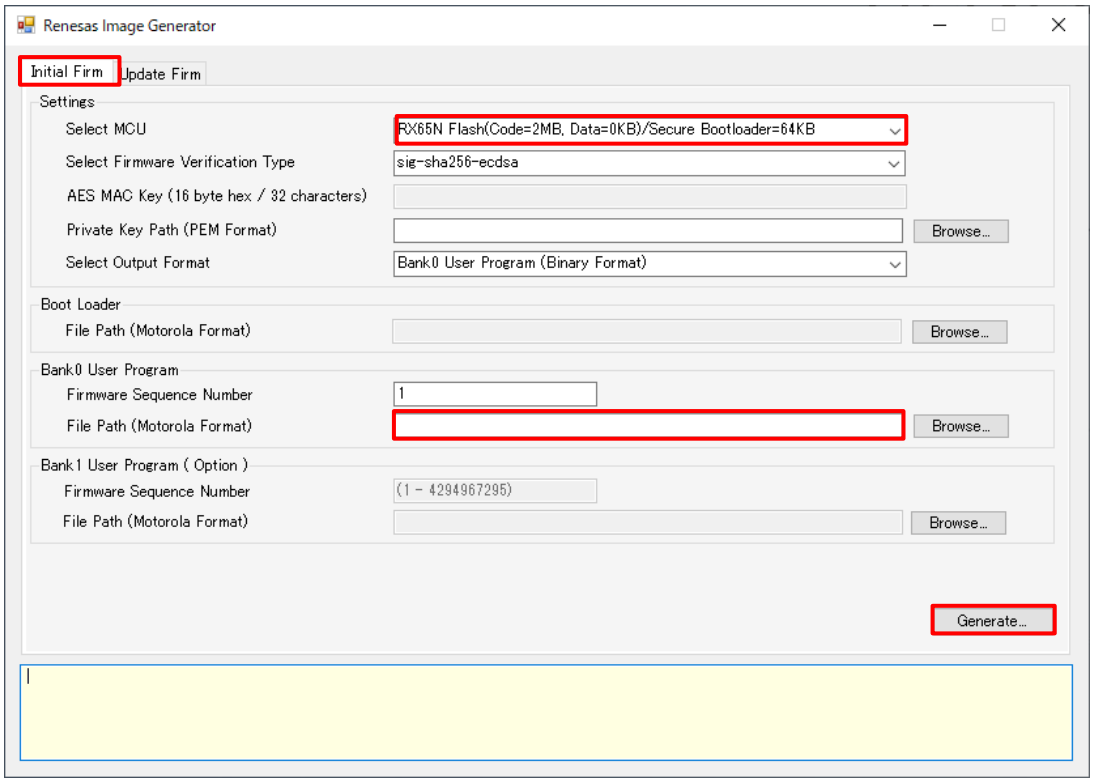


Figure 5.6 Conversion to .RSU File when Code Flash Selected (Initial Firm Tab)

In Renesas Image Generator, select the [Update Firm] tab and set **Select MCU** to **RX65N Flash(Code=2MB, Data=0KB)**, then convert the file.

Refer to 4.3.2, Generating a User Program (Firmware Update) RSU Image File, for the file conversion procedure.

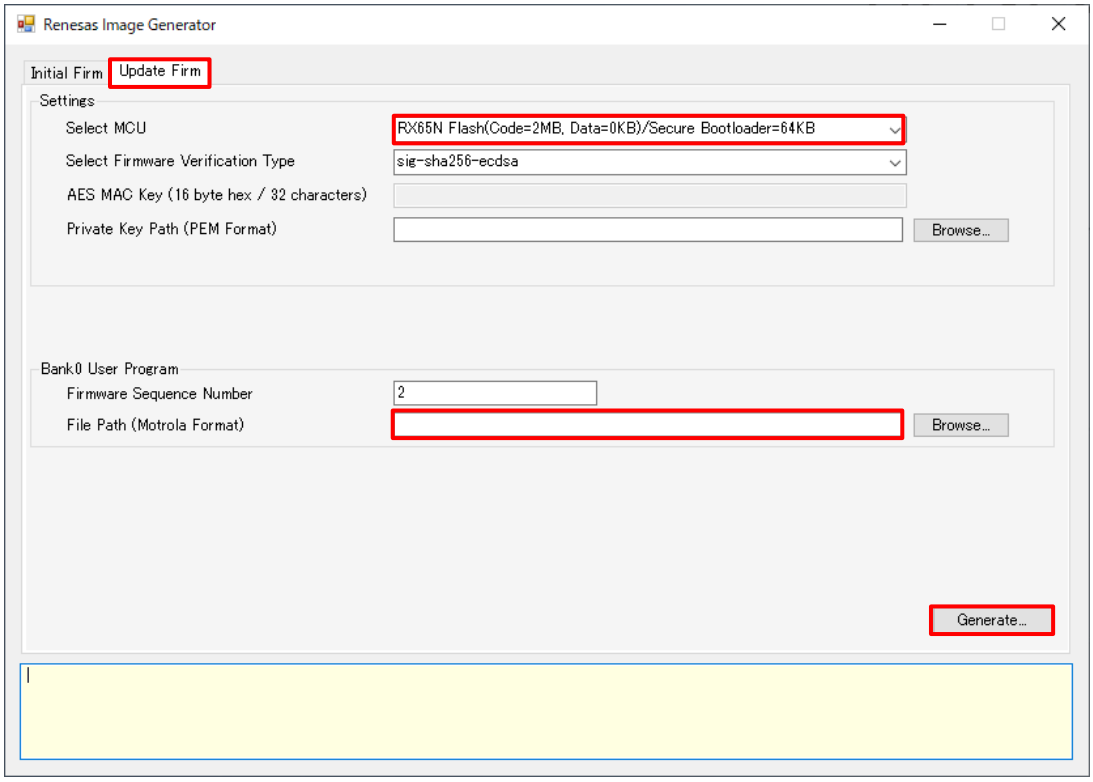


Figure 5.7 Conversion to .RSU File when Code Flash Selected (Update Firm Tab)

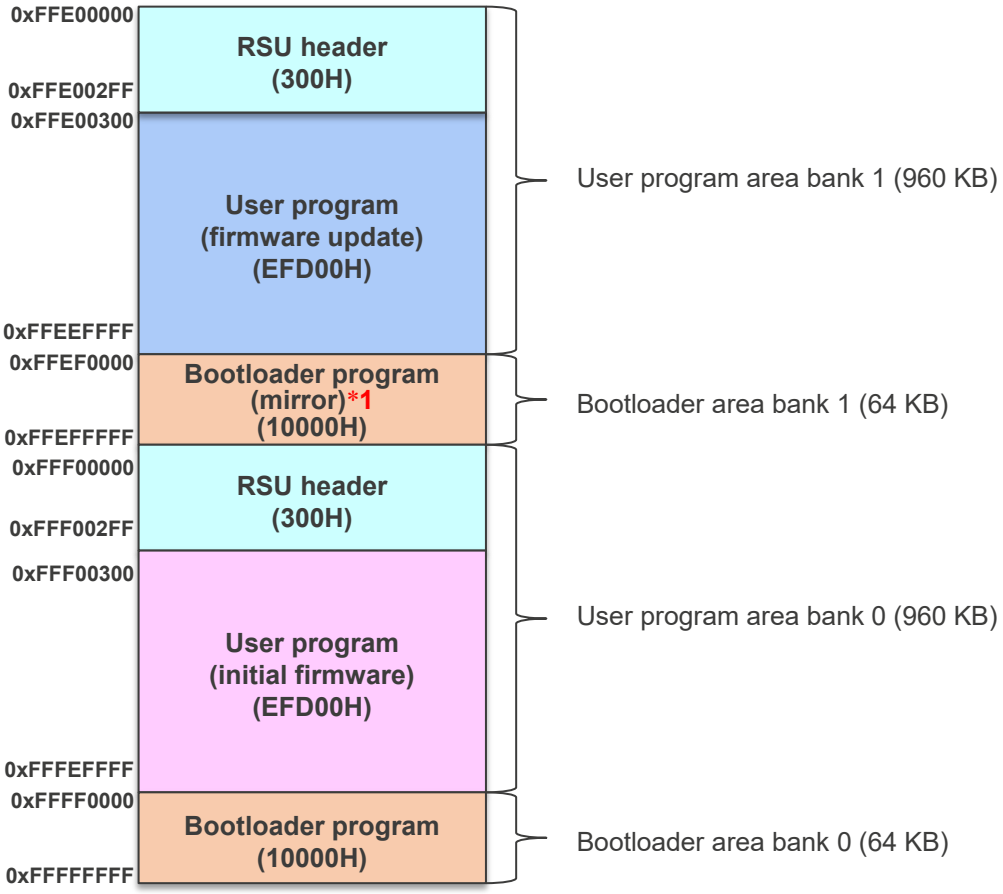
5.4 Configuration of Firmware Update Images Created by Image Generator

The firmware update memory configuration differs between dual mode and linear mode.

5.4.1 Memory Configuration in Dual Mode

In dual mode the initial firmware (bootloader and user program) is assigned to the bank 0 area and the user program update is assigned to the bank 1 area. Note that an RSU header is added at the beginning of the user program by the Image Generator (see Table 5.7).

Refer to 1.3.1 for the operation specifications for a firmware update in dual mode. The figure below shows the configuration of the firmware update in the code flash memory (2 MB) of the RX65N as an example.



Note: 1. Copied to boot program area in bank 0 area by bootloader.

Figure 5.8 Firmware Update Memory Configuration in Code Flash Memory (2 MB) of RX65N

5.4.2 Memory Configuration in Linear Mode (Partial Overwrite)

In linear mode (partial overwrite) the bootloader is assigned to the lower 64 KB of the code flash memory, and of the remaining area the upper half (area 1) is assigned to the user program update and the lower half (area 0) is assigned to the initial user program. Note that an RSU header is added at the beginning of the user program by the Image Generator (see Table 5.7).

Refer to 1.3.2.1 for the operation specifications for a firmware update in linear mode (partial overwrite).

The figure below shows the configuration of the firmware update in the code flash memory (512 MB) of the RX66T as an example.

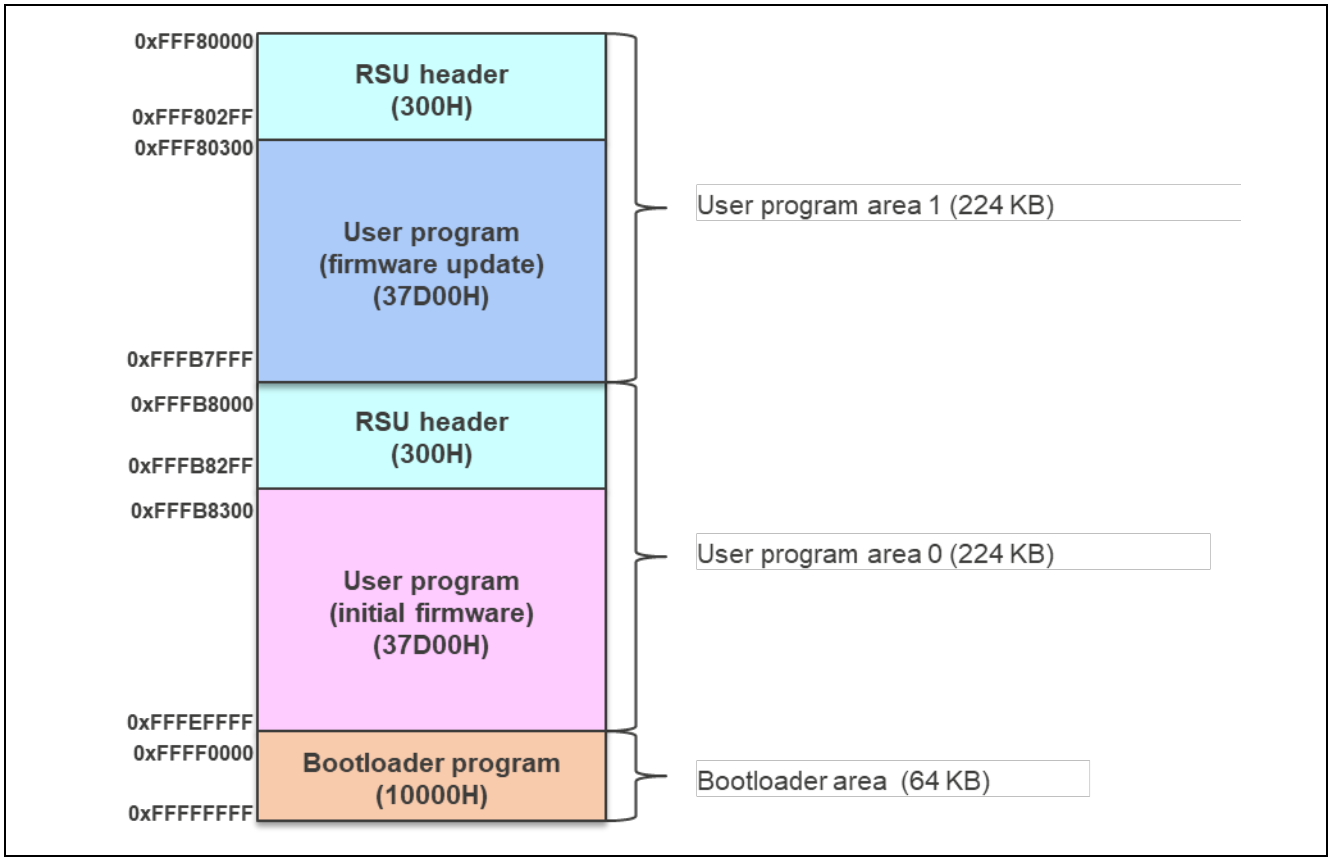


Figure 5.9 Firmware Update Memory Configuration in Code Flash Memory (512 KB) of RX66T

5.4.3 Memory Configuration in Linear Mode (Full Overwrite)

In linear mode (full overwrite) the bootloader is assigned to the lower 64 KB of the code flash memory, and the remaining area is assigned to the user program (initial firmware and firmware update). Note that an RSU header is added at the beginning of the user program by the Image Generator (see Table 5.7).

Refer to 1.3.2.2 for the operation specifications for a firmware update in linear mode (full overwrite).

The figure below shows the configuration of the firmware update in the code flash memory (512 MB) of the RX66T as an example.

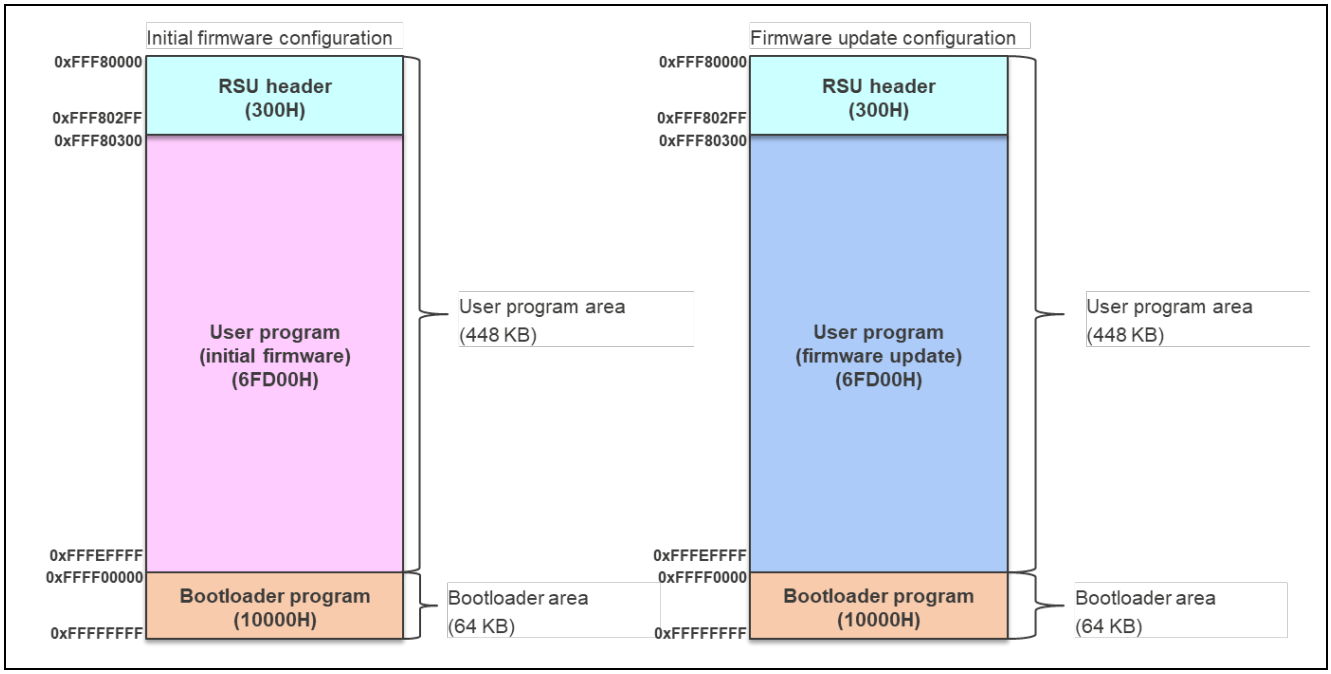


Figure 5.10 Firmware Update Memory Configuration in Code Flash Memory (512 KB) of RX66T

5.5 Details of Firmware Update Images Created by Image Generator

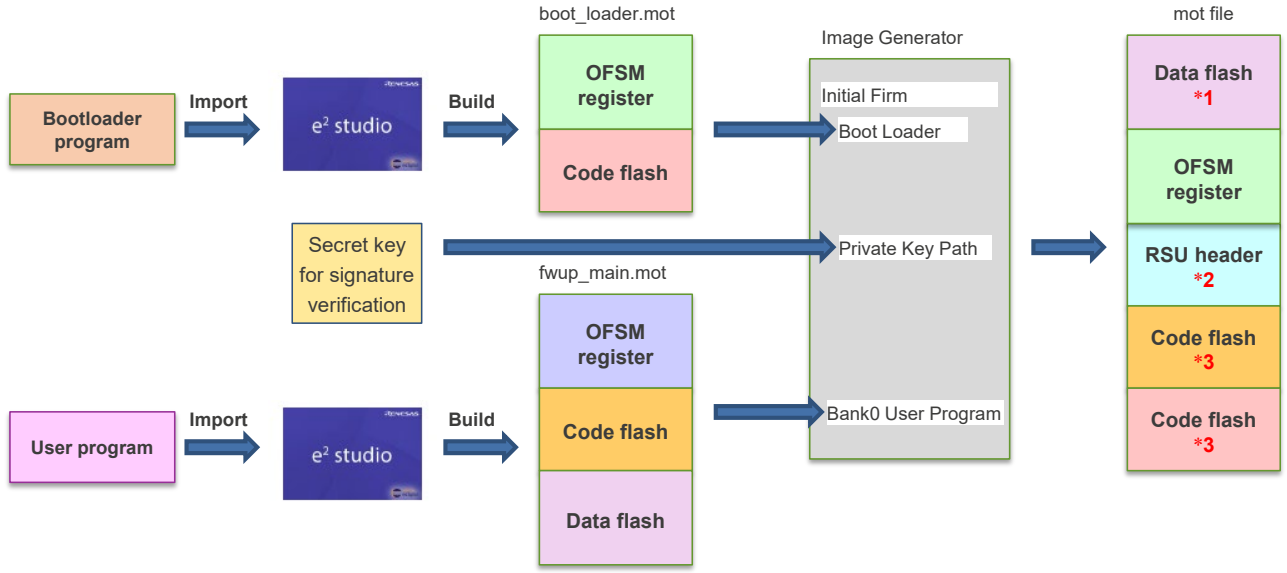
Using programs generated by building projects such as the bootloader or user program as a basis, Image Generator creates image files that provide the firmware update memory configuration shown in 5.4. Since the user program (.mot) file does not contain the information the bootloader needs to verify the signature of the user program, Image Generator creates the RSU header (0x300h) shown below and adds it at the beginning of the user program.

Table 5.7 RSU Header Details

Offset	Item	Length (byte)	Description
0x00000000	Magic Code	7	Magic code (“Renesas”)
0x00000007	Image Flags	1	Status flag used by the bootloader to verify the user program
0x00000008	Firmware Verification Type	32	Identifier specifying the firmware verification method (specified by Image Generator)
0x00000028	Signature size	4	Data size of signature value, MAC value, hash value, etc., used for firmware verification
0x0000002C	Signature	256	Signature value, MAC value, hash value, etc., used for firmware verification
0x0000012C	Data Flash Flag	4	Flag indicating whether or not data for the data flash memory is included
0x00000130	Data Flash Start Address	4	Data flash start address
0x00000134	Data Flash End Address	4	Data flash end address
0x00000138	Reserved(0x00)	200	Reserved area
0x00000200	Sequence Number	4	Sequence number (set to a value specified to Image Generator)
0x00000204	Start Address	4	Start address in serial flash memory to which the user program is programmed (set automatically by Image Generator)
0x00000208	End Address	4	End address in serial flash memory to which the user program is programmed (set automatically by Image Generator)
0x0000020C	Execution Address	4	Reserved area
0x00000210	Hardware ID	4	Hardware ID decided when a selection is made for “Select MCU” in Image Generator; used by the bootloader.
0x00000214	Reserved(0x00)	236	Reserved area

5.5.1 Details of Image Containing Bootloader and User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader and user program (initial firmware) have already been programmed (Figure 1.4 step [4] for dual mode, Figure 1.6 step [2] for linear mode (partial overwrite), Figure 1.8 step [2] for linear mode (full overwrite)), the mot file generated by building the bootloader and user program (initial firmware), as well as the secret key for signature verification generated as described in 4.2.1.2, are input to Image Generator, which generates a mot file. The resulting mot file is written to the MCU board using Flash Programmer.

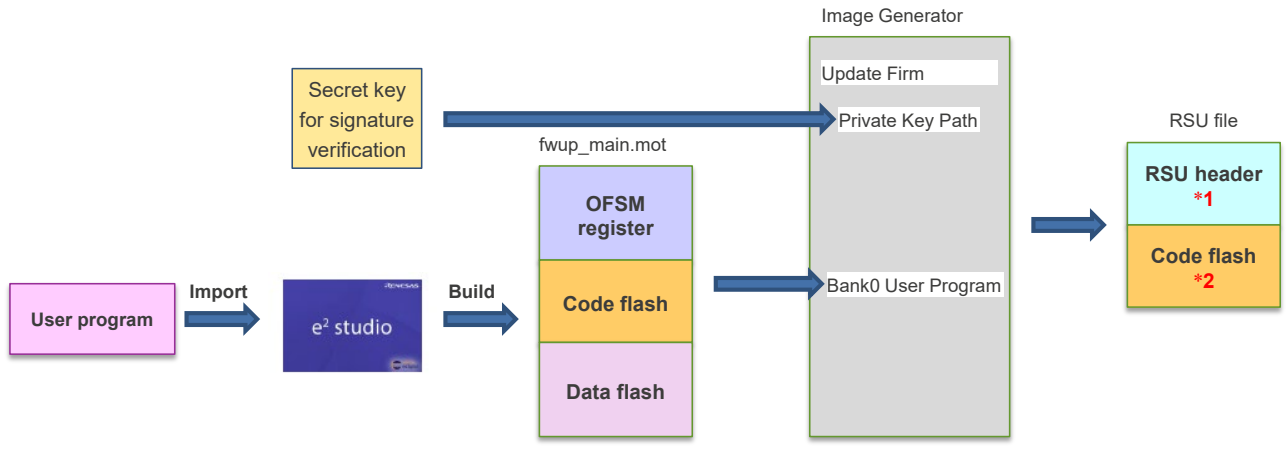


Notes: 1. Data consisting of the value 0xFF is generated for the unused area of the data flash memory.
2. Image Generator generates the RSU header.
3. Data consisting of the value 0xFF is generated for the unused area of the code flash memory.

Figure 5.11 Details of Image Containing Bootloader and User Program (Initial Firmware)

5.5.2 Details of RSU Image Containing User Program (Firmware Update)

When generating a firmware update, the mot file generated by building the user program (firmware update) and the secret key for signature verification generated as described in 4.2.1.2 are input to Image Generator, which generates an RSU file. The resulting RSU file contains the user program (firmware update) that is programmed by the bootloader.

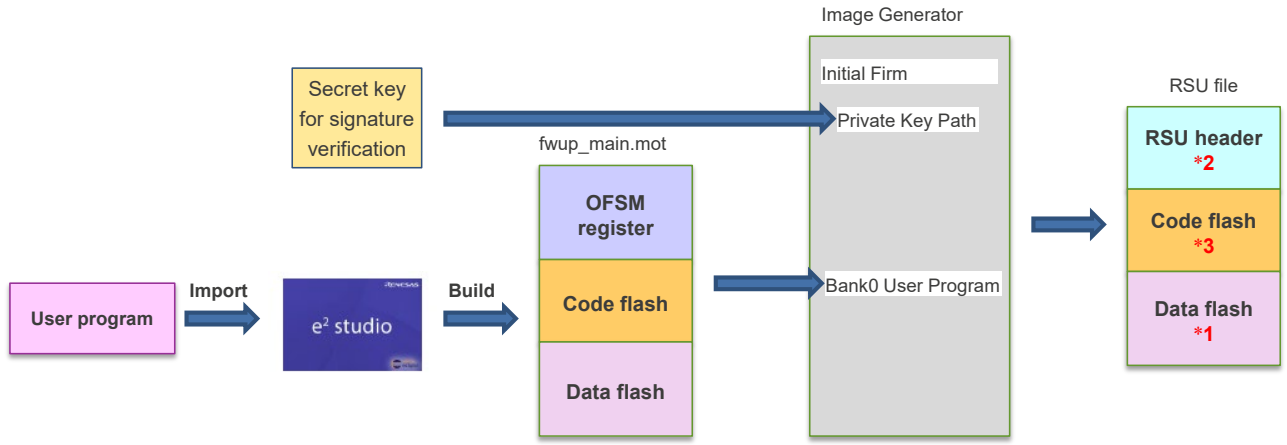


Notes: 1. Image Generator generates the RSU header.
2. Data consisting of the value 0xFF is generated for the unused area of the code flash memory.

Figure 5.12 Details of Image Containing User Program (Firmware Update)

5.5.3 Details of RSU Image Containing User Program (Initial Firmware)

When the initial settings for the firmware update start from a state where the bootloader has already been programmed (Figure 1.4 step [1] for dual mode, Figure 1.6 step [1] for linear mode (partial overwrite), Figure 1.8 step [1] for linear mode (full overwrite)), the mot file generated by building the user program (initial firmware) and the secret key for signature verification generated as described in 4.2.1.2 are input to Image Generator, which generates an RSU file. The resulting RSU file contains the user program (initial firmware) that is programmed by the bootloader.



Notes: 1. Data consisting of the value 0xFF is generated for the unused area of the data flash memory.
2. Image Generator generates the RSU header.
3. Data consisting of the value 0xFF is generated for the unused area of the code flash memory.

Figure 5.13 Details of Image Containing User Program (Initial Firmware)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr. 16, 2021	—	First edition issued
1.01	May 11, 2021	Cover	RX72N Group, RX66T Group, and RX130 Group added to Target Devices
		4	Content of 1. Overview revised
		12	Setting options added to 2.6 Compile Settings • FWUP_CFG_SERIAL_TERM_SCI • FWUP_CFG_SERIAL_TERM_SCI_BITRATE • FWUP_CFG_SERIAL_TERM_SCI_INTERRUPT_PRIORITY Descriptions revised
		15	2.6.1 Note on Compiling for RX130 Environment added
		17	Description of OTA file context added to 2.8 Arguments
			2.12 “for”, “while” and “do while” Statements deleted as it no longer applies
		24	Special Notes added to 3.13 R_FWUP_ResetDevice Function
		32	Additions made to Table 5.1 Confirmed Operation Environment (Rev. 1.01)
1.02	Oct. 29, 2021	Cover	RX671 Group added to Target Devices
		6	Description of bootloader module and firmware update module on OS-less system and system using FreeRTOS over-the-air (OTA) updates deleted from 1.2 Configuration of Firmware Update Module
			Connections between serial communication FIT module and byte queue buffer FIT module revised in Figure 1.1
		7	Bootloader module added to Figure 1.2
		8	RX671 Group added to Table 1.2
		9	Explanation added to Figure 1.4
		11	Information in Table 1.3 changed • Information on bootloader module changed Changed to indicate use of R_FWUP_Open and R_FWUP_Close by bootloader module
		17	Code sizes for other than RX65N “boot_loader project” and “aws_demos project” added to GCC for Renesas RX column in 2.7 Code Size
			Code sizes for GCC for Renesas RX added to 2.7 Code Size
		18	Indications of bootloader ROM and RAM usage added
		21	Bootloader module added to description in Table 3.1
			Bootloader module added to description in Table 3.2
		34	Table 5.1 Confirmed Operation Environment revised to Rev. 1.02 in 5.1 Confirmed Operation Environment
		35	Table 5.2 and Table 5.3 listing versions of FIT modules used by the demo project added
			Added versions of FIT modules used by the demo project under GCC on other than the RX65N to Table 5.3
		36	5.2 Compiler-Dependent Settings added
		38	Added optimization level setting when debugging the bootloader to 5.2.2.1 Compiler Options

Rev.	Date	Description	
		Page	Summary
1.03	Dec. 28, 2021	Cover	Added RX Smart Configurator User's Guide: IAREW to related Application Notes
			Added IAR C/C++ Compiler for RX to target compiler
		11	• Modify R_FWUP_SoftwareReset in Table 1.3 • Corrected an error in the function name in Table 1.3 Fixed from R_FWUP_ActiveNewImage to R_FWUP_ActivateNewImage
		13	Added FWUP_CFG_IMPLEMENTATION_ENVIRONMENT setting values in Table 2.1
		14	Added a note about unsupported combinations that do not work even if set in Table 2.2
		17	Added code size in IAR C/C++ Compiler for RX environment to Table 2.4
		18	Added the conditions of IAR C/C++ Compiler for RX when measuring the size in Table 2.4.
		19	Added the ROM and RAM sizes used by the boot loader in the IAR C/C++ Compiler for RX environment and the conditions for the IAR C/C++ Compiler for RX when measuring the size in Table 2.5.
		21	Added how to add FIT module in IAR Embedded Workbench for Renesas RX environment to "2.10 Adding the FIT Module to Your Project"
		22	Fixed the return value of the R_FWUP_Close function
		23	Fixed the return value of the R_FWUP_Operation function
		24	• Added description of R_FWUP_SetEndOfLife function • Fixed the return value of the R_FWUP_SetEndOfLife function
		25	Added description of R_FWUP_SecureBoot function and return value
		27	• Fixed the return value of the R_FWUP_CloseFile function • Corrected an error in the function name Fixed from R_FWUP_ActiveNewImage to R_FWUP_ActivateNewImage
		28	Fixed the return value of the R_FWUP_SetPlatformImageState function
		29	Fixed the return value of the R_FWUP_CheckFileSignature function
		33	Change source image in src / main.c
		36	Added IAR operation check environment to Table 5.2
		38	Added a list of FIT module versions to Table 5.5 when checking the operation in the IAR C/C++ Compiler for RX environment
		43	Added chapter "5.2.3 IAR C/C++ Compiler for RX"
1.04	May 24, 2022	Cover	Added RX140 Group to Target Devices
		8	Added RX140 Group to Table 1.2
		9	Amended description in 1.3.1 Firmware Update Operation Using Dual Mode
		14, 15	Amended Table 2.1. Deleted FWUP_CFG_USE_SERIAL_FLASH_FOR_BUFFER and FWUP_CFG_SIGNATURE_VERIFICATION. Renamed FWUP_CFG_PRINTF_DISABLE to FWUP_CFG_BOOTLOADER_LOG_DISABLE. Added FWUP_CFG_OTA_DATA_STORAGE and FWUP_CFG_LOG_LEVEL.

Rev.	Date	Description	
		Page	Summary
1.04	May 24, 2022	17	Added information on RX140 to description in section 2.6.1
		20	Added RX140 ROM and RAM sizes to Table 2.5
		21	Modified Table 2.6 OTA File Context
		27	Changed return values of R_FWUP_Abort
			Changed return values of R_FWUP_CreateFileForRx
		28	Changed return values of R_FWUP_CloseFile
			Changed return values of R_FWUP_ActivateNewImage
		29	Changed return values of R_FWUP_ResetDevice
			Changed return values of R_FWUP_SetPlatformImageState
		30	Changed return values of R_FWUP_CheckFileSignature
		33	Modified Figure 4.2
		34	Modified Figure 4.3
		35, 36	Changed log output
		37, 38	Changed Table 5.1, Table 5.2 and Table 5.3 Confirmed Operation Environment
		39 to 41	Changed Table 5.4, Table 5.5, and Table 5.6 FIT Module Versions
		49	Added 5.3 Storage Destination for FreeRTOS Data (RX65N-2MB Only)
1.05	Aug 10, 2022	Cover	Added RX660 Group to Target Devices
		8	Added RX660 Group to Table 1.2
		12	Added description of R_FWUP_ResetDevice function to Table 1.3
		14	Added description of FWUP_CFG_BOOTLOADER_LOG_DISABLE function to Table 1.3
		19	Changed build conditions when measuring code size
		20	Added RX660 ROM/RAM size to Table 1.2
		20	Changed build conditions when measuring ROM/RAM size used by bootloader
		23	Deleted FWUP_ERR_LESS_MEMORY and FWUP_ERR_IMAGE_STATE from Return Values of R_FWUP_Open
		26	Deleted FWUP_ERR_ALREADY_OPEN from Return Value of R_FWUP_SecureBoot and added description of FWUP_ERR_NOT_OPEN
		29	Added description to Special Notes for R_FWUP_ResetDevice
		31,32	Added demo project list to Figure 4-1
		34	Added description of demo program
		34	4.2.1 (2) Deleted the description of Base64 (3) Deleted the description of the Key file
		39-41	Added operation check environment for RX660
		42-44	Added description of RX660 to Table 5-7/5-8/5-9 and updated FIT module version of RX65N, RX671, RX72N
1.06	Dec. 28, 2022	7	Added description related to modules for OS-less systems without a communication driver
		8	Added Figure 1.2 System Configuration of Firmware Update Module on OS-less (Module-External Communication Control) System
		12, 13	Added description of partial overwrite firmware update
		13, 14	Added description of full overwrite firmware update

Rev.	Date	Description	
		Page	Summary
		15-17	Added 1.4 Firmware Update Communication Control on OS-Less System
		18, 19	Added OS-less without communication driver and Azure ADU items to Table 1.3 API Functions
		21	Added member and changed description of 4 under FWUP_CFG_IMPLEMENTATION_ENVIRONMENT on Table 2.1 Configuration Settings
		23	Added member and changed description of 4 under FWUP_CFG_IMPLEMENTATION_ENVIRONMENT on Table 2.2 Allowable Compile Setting Combinations
		24	Changed description of 8 and deleted 9 to 11 on Table 2.3 Valid Combination Macro Values
		25	Added fwup_main_woSciDrv under RX66T on Table 2.4 Code Sizes
		29	Added “(4) Deleting unnecessary modules” to 2.10 Adding the FIT Module to Your Project
		32	Added 3.3 R_FWUP_Initialize Function
		33	Added 3.5 R_FWUP_PutFirmwareChunk Function
			Added 3.7 R_FWUP_DirectUpdate Function
		35	Added “For full overwrite method” description to 3.9 R_FWUP_SecureBoot Function
		39	Added [OS-Less Usage] to 3.20 R_FWUP_CheckFileSignature Function
		42, 43	Changed Figure 4.1 Demo Project List (1) and Figure 4.2 Demo Project List (2)
		44, 45	Added 4.2 Building the Demo Project
		45-48	Added 4.3 Using Image Generator to Convert the Firmware Update Image File
		49-57	Revised 4.4
		58	Added Rev. 1.06 as confirmed operation environment to Table 5.1 Confirmed Operation Environment (CC-RX) Deleted old confirmed operation environments
			Added Rev. 1.06 as confirmed operation environment to Table 5.2 Confirmed Operation Environment (GCC) Deleted old confirmed operation environments
		59	Added Rev. 1.06 as confirmed operation environment to Table 5.3 Confirmed Operation Environment (IAR) Deleted old confirmed operation environments
		60	Updated FIT module versions and added fwup_main_woSciDrv under RX66T on Table 5.4 FIT Module Versions (CC-RX)
		61	Updated FIT module versions on Table 5.5 FIT Module Versions (GCC)
		62	Updated FIT module versions on Table 5.6 FIT Module Versions (IAR)
		73-75	Added 5.4 Configuration of Firmware Update Images Created by Image Generator
		76-78	Added 5.5 Details of Firmware Update Images Created by Image Generator

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/.