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

RX231 Group

RX65N Group

RX66T 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 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 Compiler

C/C++ Compiler Package for RX Family from Renesas Electronics

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 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, and Figure 1.2 shows a system configuration incorporating the firmware update module on a system using FreeRTOS over-the-air (OTA) updates.

On the OS-less system a bootloader module and the firmware update module are used. On the system using FreeRTOS over-the-air (OTA) updates only the firmware update module is used because FreeRTOS over-the-air (OTA) updates includes functionality equivalent to the bootloader module.

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.

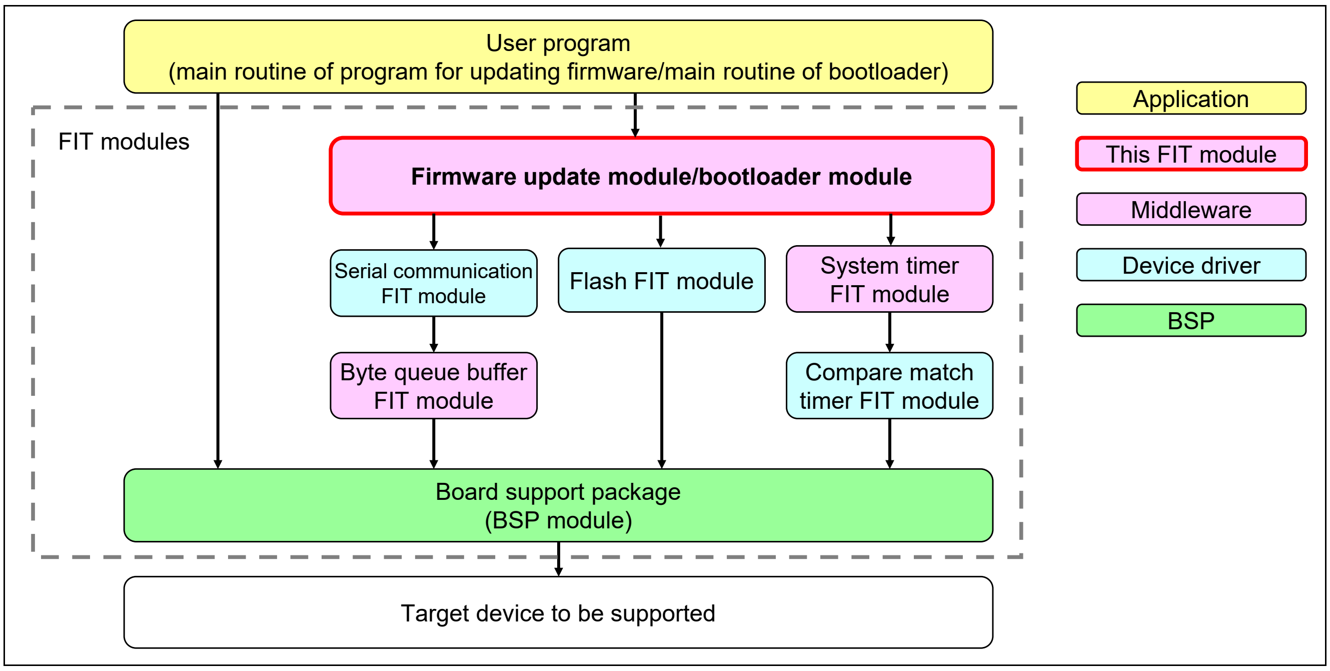

User program
(main routine of program for updating firmware/main routine of bootloader)
FIT modules
Firmware update module/bootloader module
Serial communication FIT module
Flash FIT module
System timer FIT module
Byte queue buffer FIT module
Compare match timer FIT module
Board support package (BSP module)
Target device to be supported
Application
This FIT module
Middleware
Device driver
BSP

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

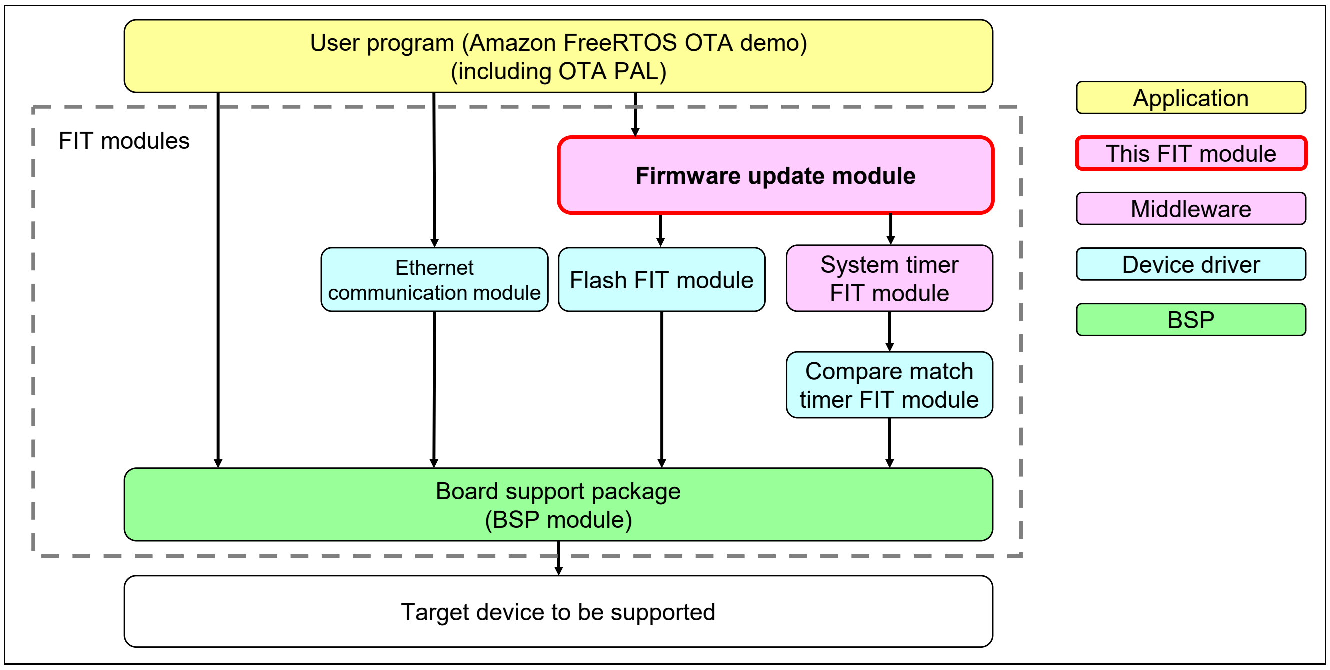

User program (Amazon FreeRTOS OTA demo)
(including OTA PAL)
FIT modules
Firmware update module
Ethernet communication module
Flash FIT module
System timer FIT module
Compare match timer FIT module
Board support package (BSP module)
Target device to be supported
Application
This FIT module
Middleware
Device driver
BSP

Figure 1.2 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	○	—
RX231 Group	○	—
RX65N Group	○	○
RX66T 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 1.3 shows the initial settings for firmware update operation in dual mode.

A tool (Renesas Secure Flash Programmer) 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 user program only or to create initial firmware containing both the bootloader and the user program. By using Flash Programmer or the like to program initial firmware containing both the bootloader and the user program to the on-chip flash memory, the state shown in Figure 1.3 step [4] can be achieved.

Alternatively, the state shown in Figure 1.3 step [1] can be achieved by building the bootloader program and programming the resulting .mot file to the on-chip flash memory. If just the bootloader has been programmed to the on-chip flash memory, it is then possible to use the functions of the bootloader to program initial firmware containing only the user program to the on-chip flash memory.

You can start initial settings from step [1] or step [4], depending on the characteristics of the customer’s system.

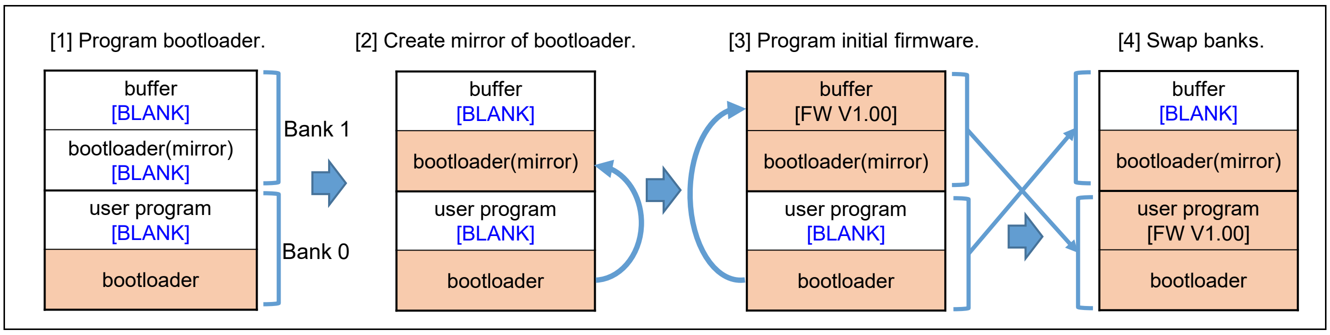


Figure 1.3 Dual Mode Firmware Update Initial Settings

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 containing only the user program (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 1.4 shows dual mode firmware update operation.

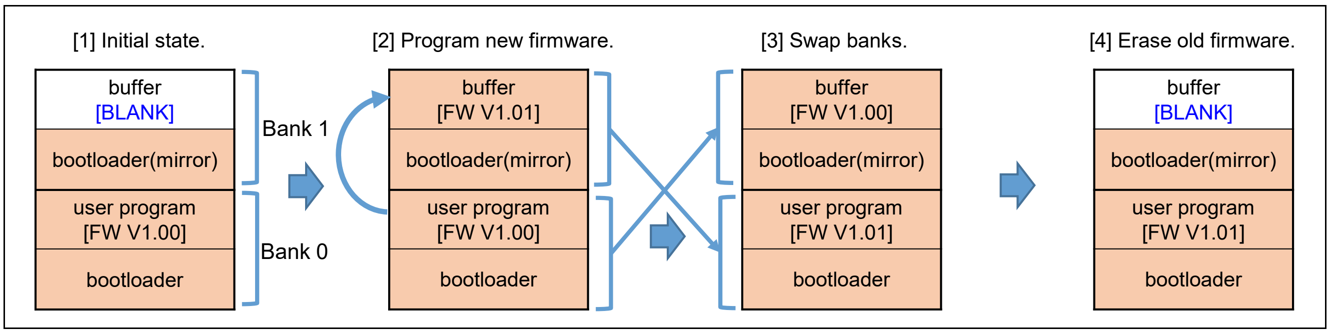


Figure 1.4 Dual Mode Firmware Update Operation

[1] Initial state.
[2] Use the firmware update module incorporated in the user program to program the new firmware version (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 old firmware from bank 1.

1.3.2 Firmware Update Operation Using Linear Mode

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

Figure 1.5 shows the initial settings for firmware update operation in linear mode.

A tool (Renesas Secure Flash Programmer.exe) 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 user program only or to create initial firmware containing both the bootloader and the user program. By using Flash Programmer or the like to program initial firmware containing both the bootloader and the user program to the on-chip flash memory, the state shown in Figure 1.5 step [2] can be achieved.

Alternatively, the state shown in Figure 1.5 step [1] can be achieved by building the bootloader program and programming the resulting .mot file to the on-chip flash memory. If just the bootloader has been programmed to the on-chip flash memory, it is then possible to use the functions of the bootloader to program initial firmware containing only the user program to the on-chip flash memory.

You can start initial settings from step [1] or step [2], depending on the characteristics of the customer's system.

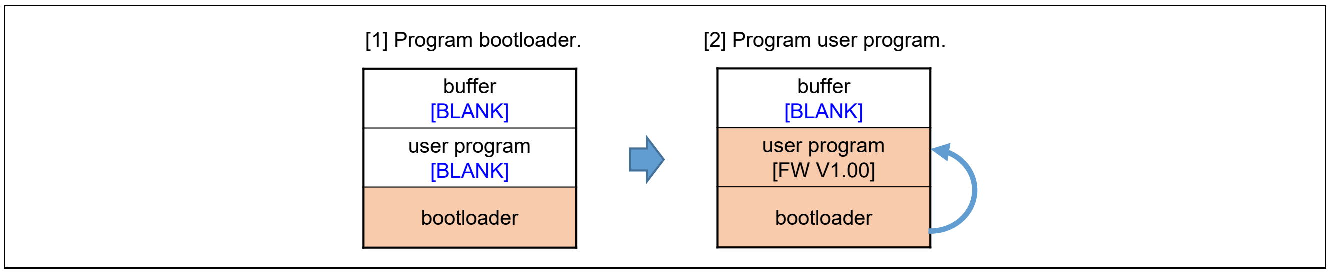


Figure 1.5 Linear Mode Firmware Update Initial Settings

Starting initial settings from step [1]

[1] Use Flash Programmer or the like to program the bootloader to the on-chip flash memory.
[2] Use the bootloader to program the initial firmware containing the user program only (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 1.6 shows firmware update operation in linear mode.

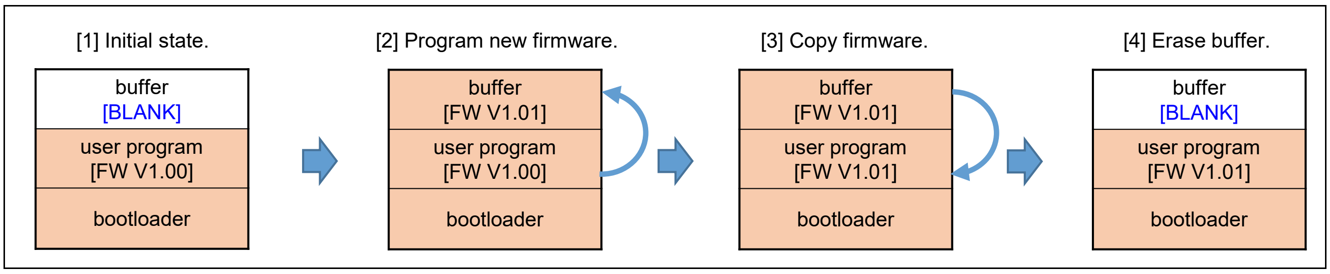


Figure 1.6 Linear Mode Firmware Update Operation

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

1.4 API Overview

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

Table 1.3 API Functions

Function	Function Description	FreeRTOS (OTA)	OS-less	
		Firmware Update Module	Firmware Update Module	Bootloader Module
R_FWUP_Open	Performs processing to open the module.	—	○	—
R_FWUP_Close	Performs processing to close the module.	—	○	—
R_FWUP_Operation	Performs firmware update processing from the user program.	—	○	—
R_FWUP_SoftwareReset	Applies a software reset.	—	○	○
R_FWUP_SetEndOfLife	Performs end of life 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_ActiveNewImage	Activates or launches the new firmware image.	○	—	—
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_CheckFileSigunature	Checks the signature of the specified file.	○	—	—
R_FWUP_ReadAndAssumeCertificate	Reads and returns the specified signer certificate from the file system.	○	—	—
R_FWUP_GetVersion	Returns the version number of the module.	○	○	○

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 FreeRTOS (OTA) program. 3: Implement in firmware update program using OS other than FreeRTOS. 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_USE_SERIAL_FLASH _FOR_BUFFER Note: The default is 0.	Specifies whether or not external serial flash memory is used when obtaining the new version of the firmware. 0: Do not use external serial flash memory (default). 1: Use external serial flash memory.*[1]
FWUP_CFG_SIGNATURE _VERIFICATION Note: The default is 0.	Specifies the signature verification algorithm. 0: Use ECDSA-secp256r1 for signature verification and SHA256 as the hash algorithm (default). More setting values can be added for additional verification algorithms.
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_PRINTF_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. 0: Display character strings in terminal software (default). 1: Do not display character strings in terminal software.
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.

Configuration options in r_fwup _config.h	
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)	1	—	2	—	3
	2: FreeRTOS (OTA) program	4	5	6	7	—
	3: Firmware update program using OS other than FreeRTOS.	8	—	9	10	11

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.

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_SCI_FS	8	Connect an external storage device (an SD card) to the SCI, and obtain the new version of the firmware using the file system.
FWUP_COMM_USB_FS	9	Connect an external storage device (a USB flash drive) to the USB, and obtain the new version of the firmware using the file system.
FWUP_COMM_SDHI_FS	10	Connect an external storage device (an SD card) to the SDHI, and obtain the new version of the firmware using the file system.
FWUP_COMM_QSPI_FS	11	Connect an external storage device (serial flash memory) to the QSPI, and obtain the new version of the firmware using the file system.

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_SCI_FS            8    // External storage (SD card + file system) connected to SCI.
#define FWUP_COMM_USB_FS            9    // External storage (USB flash drive + file system) connected to USB.
#define FWUP_COMM_SDHI_FS           10   // External storage (SD card + file system) connected to SDHI.
#define FWUP_COMM_QSPI_FS           11   // External storage (Serial flash + file system) connected to QSPI.
```

2.6.1 Note on Compiling for RX130 Environment

To use the FIT module on the RSK RX130, change the setting of the board support package (BSP) configuration option for the user stack size (BSP_CFG_USTACK_BYTES) from the default value of 0x400 (1 KB) to 0x1000 (4 KB).

2.7 Code Size

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

Table 2.4 Code Sizes

ROM, RAM and Stack Code Sizes			
Device	**Category**	**Memory Used**	**Remarks**
RX65N	ROM	3,294 bytes	boot_loader project
		3,983 bytes	fwup_main project
		3,050 bytes	eol_main project
		5,396 bytes	aws_demos project
	RAM	36,968 bytes	boot_loader project
		3,217 bytes	fwup_main project
		2,193 bytes	eol_main project
		1,256 bytes	aws_demos project
	Max. stack size used	1,168 bytes	boot_loader project
		2,192 bytes	fwup_main project
		800 bytes	eol_main project
		1,792 bytes	aws_demos project
RX231	ROM	3,665 bytes	boot_loader project
		3,949 bytes	fwup_main project
		2,961 bytes	eol_main project
	RAM	2,961 bytes	boot_loader project
		3,217 bytes	fwup_main project
		2,193 bytes	eol_main project
	Max. stack size used	1,384 bytes	boot_loader project
		2,172 bytes	fwup_main project
		772 bytes	eol_main project

Conditions

• Optimization level:	Level 2
• Link module optimization:	Checked
• Optimization method:	Code size optimization
• Remove unreferenced variables/functions:	Unchecked
• FWUP_CFG_PRINTF_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¥freertos_plus¥aws¥ota¥include¥aws_iot_ota_agent.h

Table 2.5 OTA File Context

```
typedef struct
{
  uint16_t usSize;                              /* Size, in bytes, of the signature. */
  uint8_t ucData[ kOTA_MaxSignatureSize ]; /* The binary signature data. */
} Sig256_t;

typedef struct OTA_FileContext
{
  uint8_t * pucFilePath; /*!< Local file pathname. */
  union
  {
    int32_t lFileHandle;        /*!< Device internal file pointer or handle.
                                 * File type is handle after file is open for write. */
    #if WIN32
        FILE * pxFile;          /*!< File type is stdio FILE structure after file is open for write. */
    #endif
    uint8_t * pucFile;          /*!< File type is RAM/Flash image pointer after file is open for write. */
  };
  uint32_t ulFileSize;          /*!< The size of the file in bytes. */
  uint32_t ulBlocksRemaining;   /*!< How many blocks remain to be received (a code optimization). */
  uint32_t ulFileAttributes;    /*!< Flags specific to the file being received (e.g. secure, bundle, archive). */
  uint32_t ulServerFileID;      /*!< The file is referenced by this numeric ID in the OTA job. */
  uint8_t * pucJobName;         /*!< The job name associated with this file from the job service. */
  uint8_t * pucStreamName;      /*!< The stream associated with this file from the OTA service. */
  Sig256_t * pxSignature;       /*!< Pointer to the file's signature structure. */
  uint8_t * pucRxBlockBitmap;   /*!< Bitmap of blocks received (for de-duping and missing block request). */
  uint8_t * pucCertFilepath;    /*!< Pathname of the certificate file used to validate the receive file. */
  uint8_t * pucUpdateUrlPath;   /*!< Url for the file. */
  uint8_t * pucAuthScheme;      /*!< Authorization scheme. */
  uint32_t ulUpdaterVersion;    /*!< Used by OTA self-test detection, the version of FW that did the update. */
  bool_t xIsInSelfTest;         /*!< True if the job is in self test mode. */
  uint8_t * pucProtocols;       /*!< Authorization scheme. */
} OTA_FileContext_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 2.6 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.

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. Performs processing to open the resources used by the firmware update 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_LESS_MEMORY	: Insufficient memory
	FWUP_ERR_IMAGE_STATE	: Updating not possible in current flash status
	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. Performs processing to close the resources used by the firmware update module, and makes OS end settings (when using an OS).	
Parameters	None	
Return Values	FWUP_SUCCESS	: Normal end
	FWUP_ERR_NOT_OPEN	: Not open
	FWUP_ERR_FLASH	: Flash module error
	FWUP_ERR_COMM	: Communication module error
	FWUP_ERR_STATE_MONITORING	: Status transition monitoring module error
Special Notes	—	

3.3 R_FWUP_Operation Function

Table 3.3 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_STATE_MONITORING	: Firmware update status has not changed for more than specified duration
Special Notes	—	

3.4 R_FWUP_SoftwareReset Function

Table 3.4 R_FWUP_SoftwareReset Function Specifications

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

3.5 R_FWUP_SetEndOfLife Function

Table 3.5 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 accompanied by a bank swap, 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_IMAGE_STATE	: Updating not possible in current flash status
	FWUP_ERR_FLASH	: Flash module error
	FWUP_ERR_COMM	: Communication module error
Special Notes	—	

3.6 R_FWUP_SecureBoot Function

Table 3.6 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_FAIL, the secure boot failed. If necessary, cancel the error and call this function again.	
Parameters	None	
Return Values	FWUP_SUCCESS	: Secure boot normal end
	FWUP_FAIL	: Secure boot error occurred
	FWUP_IN_PROGRESS	: Secure boot in progress
Special Notes	—	

3.7 R_FWUP_ExecuteFirmware Function

Table 3.7 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	—

3.8 R_FWUP_Abort Function

Table 3.8 R_FWUP_Abort Function Specifications

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

3.9 R_FWUP_CreateFileForRx Function

Table 3.9 R_FWUP_CreateFileForRx Function Specifications

Format	OTA_Err_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	kOTA_Err_None	: Normal end
	kOTA_Err_RxFileCreateFailed	: File creation error
Special Notes	—	

3.10 R_FWUP_CloseFile Function

Table 3.10 R_FWUP_CloseFile Function Specifications

Format	OTA_Err_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	kOTA_Err_None	: Normal end
	kOTA_Err_FileClose	: File close error
	kOTA_Err_SignatureCheckFailed	: Signature verification error
Special Notes	—	

3.11 R_FWUP_WriteBlock Function

Table 3.11 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	—	

3.12 R_FWUP_ActiveNewImage Function

Table 3.12 R_FWUP_ActiveNewImage Function Specifications

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

3.13 R_FWUP_ResetDevice Function

Table 3.13 R_FWUP_ResetDevice Function Specifications

Format	OTA_Err_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	kOTA_Err_None : Normal end
Special Notes	Close all open peripheral circuits before calling this function.

3.14 R_FWUP_SetPlatformImageState Function

Table 3.14 R_FWUP_SetPlatformImageState Function Specifications

Format	OTA_Err_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	kOTA_Err_None	: Normal end
	kOTA_Err_CommitFailed	: Commit error
Special Notes	—	

3.15 R_FWUP_GetPlatformImageState Function

Table 3.15 R_FWUP_GetPlatformImageState Function Specifications

Format	OTA_PAL_ImageState_t R_FWUP_GetPlatformImageState (void)
Description	Returns the current life cycle status.
Parameters	None
Return Values	Current life cycle status
Special Notes	—

3.16 R_FWUP_CheckFileSignature Function

Table 3.16 R_FWUP_CheckFileSignature Function Specifications

Format	OTA_Err_t R_FWUP_CheckFileSignature (OTA_FileContext_t * const C)	
Description	Checks the signature of the specified file.	
Parameters	* C	: File context
Return Values	kOTA_Err_None	: Normal end
	kOTA_Err_SignatureCheckFailed	: Signature verification error
Special Notes	—	

3.17 R_FWUP_ReadAndAssumeCertificate Function

Table 3.17 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	—	

3.18 R_FWUP_GetVersion Function

Table 3.18 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 includes a main() function that utilizes the FIT module and its dependent modules. The FIT module includes the following demo project.

4.1 Firmware Update Using Serial Communications Interface (SCI) of RX65N

The firmware update demo utilizes the serial communications interface (SCI) of the RX65N, which is mounted on the RSK RX65N starter kit board. Communication with the terminal software takes place via SCI channels configured as a UART.

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.)
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.

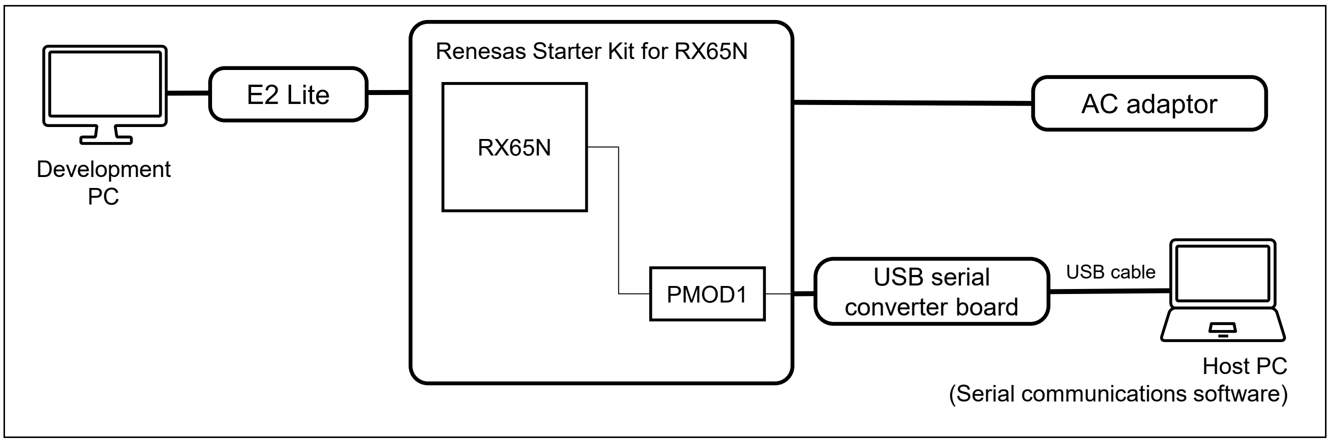

Development PC
E2 Lite
Renesas Starter Kit for RX65N
RX65N
PMOD1
AC adaptor
USB serial converter board
USB cable
Host PC
(Serial communications software)

Figure 4.1 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

4.1.1 Generating the Firmware Update

1. To ensure the integrity of the firmware to be applied as an update, the firmware update is digitally signed (ECDSA + SHA256) and the signature is used to verify its integrity. To perform verification, the following code must be added to the fwup_main_RX65N sample application.
   — Tinycrypt library
   — Base64 decode function
   — Key file used for digital signature

   The procedure for adding these items is as follows.

   (1) Adding the Tinycrypt library
   After obtaining the files from https://github.com/renesas/amazon-freertos/tree/master/libraries/3rdparty/tinycrypt, add the **lib** folder to the **src/src/tinycypt/lib** folder of the **fwup_main_RX65N** project.

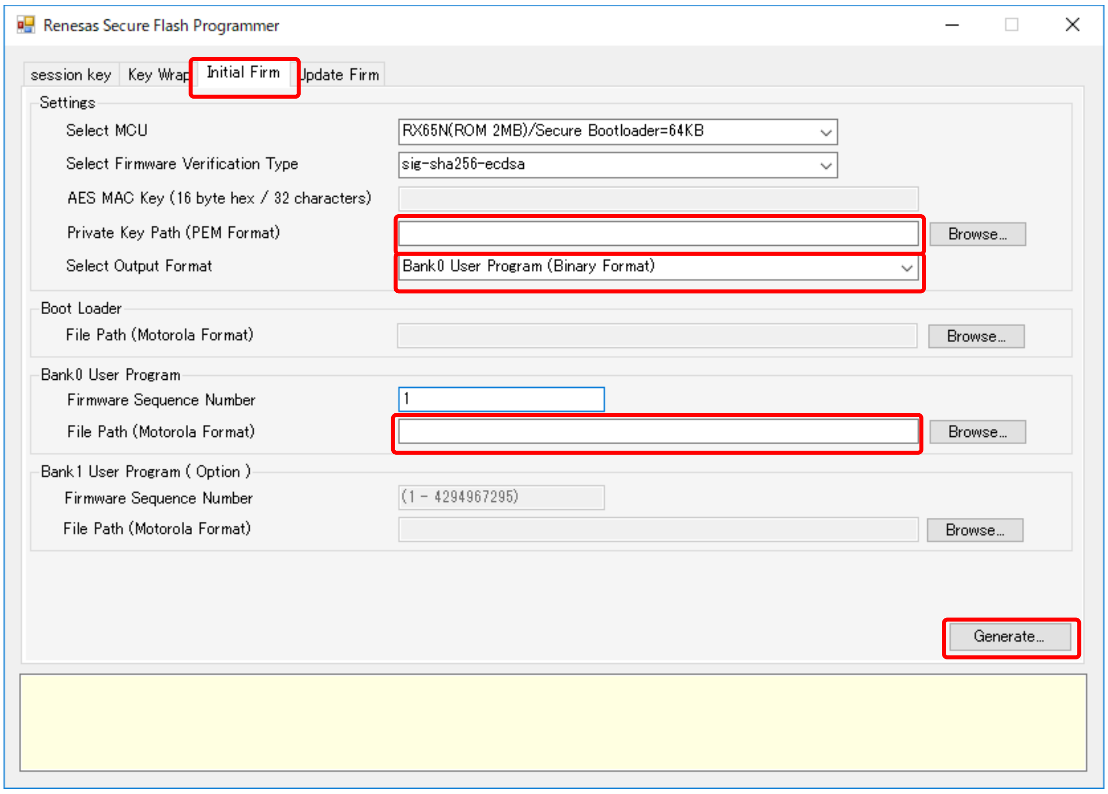
   (2) Adding the Base64 decode function
   After obtaining the files from https://github.com/renesas/amazon-freertos/tree/master/projects/renesas/rx65n-rsk/e2studio/boot_loader/src/src, add **base64_decode.c** and **base64_decode.h** to the **src/src** folder of the **fwup_main_RX65N** project.

   (3) Adding the key file
   After obtaining the files from https://github.com/renesas/amazon-freertos/tree/master/projects/renesas/rx65n-rsk/e2studio/boot_loader/src/, add the **key** folder to the **src/key** folder of the **fwup_main_RX65N** project. After adding the folder, enter the public key information for signature verification in **code_signer_public_key.h**.
   Refer to the following link for instructions on adding the information.

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

   4. Create the keys to be used for firmware verification in OpenSSL.
   5. To enable firmware verification using ECDSA + SHA256, import the public key for signature verification (secp256r1.publickey) by the bootloader.

2. Build the **fwup_main_RX65N** sample application and convert the resulting .mot file into an .RSU file. This is the "initial firmware."
   The procedure for converting the .mot file into an .RSU file is as follows.
   Download Renesas Secure Flash Programmer.exe from
   Release mot file converter tool · renesas/mot-file-converter · GitHub and then run it. (You will also need the other files archived along with it, so download them too.)
   — Select the [Initial Firm] tab and set the parameters as shown in the screenshot below.
   — Set the path of secp256r1.private key in **Private Key Path** of **Settings**.
   — Set Bank0 User Program (Binary Format) to **Select Output format** in **Settings**.
   — For **File Path** under **Bank 0 User Program**, specify the path of the .mot file created as described above.
   — Click the [Generate] button to generate an .RSU file, and store it in the **init_firmware** folder.



Settings
Select MCU: RX65N(ROM 2MB)/Secure Bootloader=64KB
Select Firmware Verification Type: sig-sha256-ecdsa
AES MAC Key (16 byte hex / 32 characters)
Private Key Path (PEM Format)
Select Output Format: Bank0 User Program (Binary Format)

Boot Loader
File Path (Motorola Format)

Bank0 User Program
Firmware Sequence Number: 1
File Path (Motorola Format)

Bank1 User Program (Option)
Firmware Sequence Number: (1 - 4294967295)
File Path (Motorola Format)

Figure 4.2 Renesas Secure Flash Programmer Initial Firm Tab

3. Open **src/main.c** and remove the slashes from the left of the commented-out lines to make them valid.

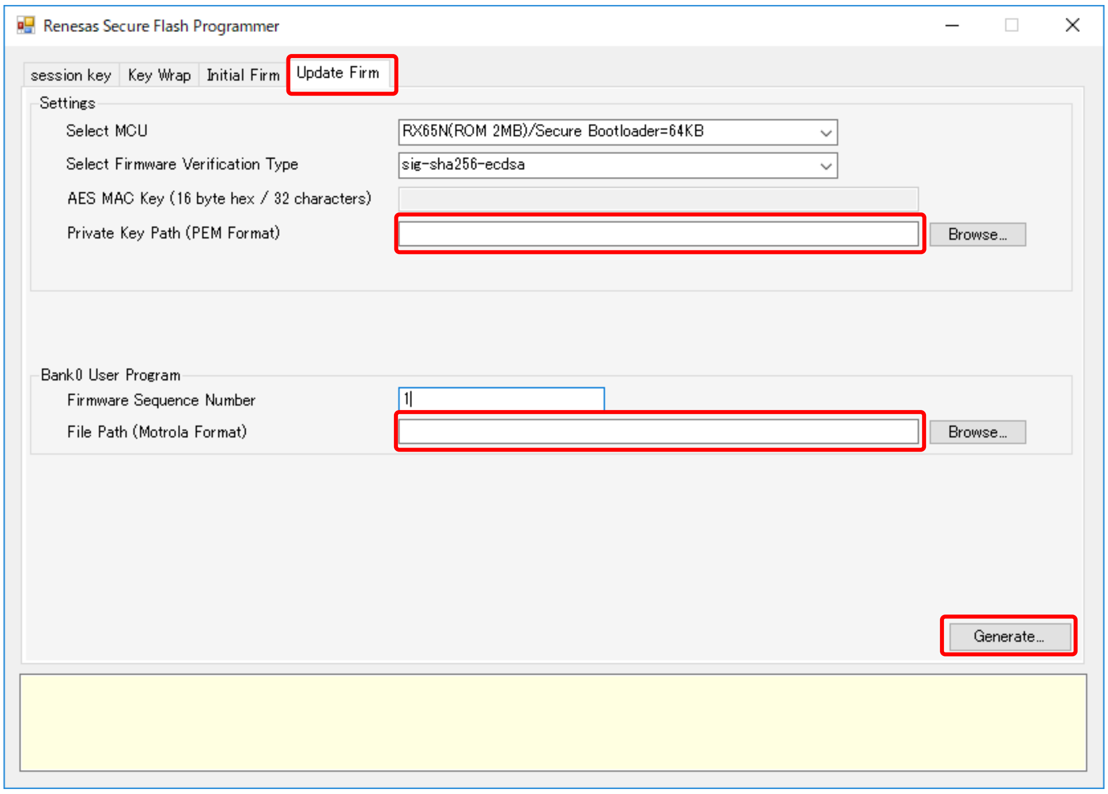
Lines 84 to 88 in main.c

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

Build the project once again, and convert the resulting .mot file into an .RSU file. This is the "next firmware."

The procedure for converting the .mot file into an .RSU file is as follows.

— Select the [Update Firm] tab and set the parameters as shown in the screenshot below.
— Set the path of secp256r1.private key in **Private Key Path** of **Settings**.
— For **File Path** under **Bank 0 User Program**, specify the path of the .mot file created as described above.
— Click the [Generate] button to generate an .RSU file, and store it in the **update_firmware** folder.



Settings
Select MCU: RX65N(ROM 2MB)/Secure Bootloader=64KB
Select Firmware Verification Type: sig-sha256-ecdsa
AES MAC Key (16 byte hex / 32 characters)
Private Key Path (PEM Format)

Bank0 User Program
Firmware Sequence Number: 1
File Path (Motrola Format)

Figure 4.3 Renesas Secure Flash Programmer Update Firm Tab

4.1.2 Updating the Firmware

1. Connect the PC USB port, USB serial converter board, and PMOD1 on the RSK board as shown in Figure 4.1, RSK RX65N Device Connection Diagram.
2. Launch the terminal software 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. Build the bootloader program, download it to the RSK board, and use the debugger to run the application.
5. When you run the software, the following message is output.

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

Select the “send file” function in the terminal software, and send the previously created “initial firmware” .RSU file. (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).
```

6. When installation and signature verification finish, the application for applying the firmware update is launched, and a message prompting you to input the firmware application is output.

```
jump to user program
[R_FWUP_GetPlatformImageState] is called.
Function call: R_FWUP_GetPlatformImageState: [2]
[R_FWUP_CreateFileForRx] is called.
[R_FWUP_CreateFileForRx] Receive file created.
[R_FWUP_GetPlatformImageState] is called.
Function call: R_FWUP_GetPlatformImageState: [2]
-------------------------------------------------
Firmware update user program
-------------------------------------------------
Send Update firmware via UART.
```

Select the “send file” function in the terminal software, and send the previously created “next firmware” .RSU file. (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.

```
[R_FWUP_WriteBlock] is called.
[R_FWUP_Operation] Flash Write: Address = 0xFFE00000, length = 1024 ... OK
[R_FWUP_WriteBlock] is called.
[R_FWUP_Operation] Flash Write: Address = 0xFFE00400, length = 1024 ... OK
```

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

```
jump to user program
[R_FWUP_GetPlatformImageState] is called.
```

8. The firmware application outputs the following message indicating that the demo has completed successfully.

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

4.1.3 Generating EOL Firmware

1. Build the **eol_main_RX65N** sample application and convert the resulting .mot file into an .RSU file. This is the "eol firmware."
   Refer to 4.1.1 above for instructions on converting to .RSU file format.

4.1.4 Firmware EOL

1. Connect the PC USB port, USB serial converter board, and PMOD1 on the RSK board as shown in Figure 4.1, RSK RX65N Device Connection Diagram.
2. Launch the terminal emulation program (terminal software) 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. Build the bootloader program, download it to the RSK board, and use the debugger to run the application.
5. When you run the software, the following message is output.

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

   Select the "send file" function in the terminal software, and send the previously created "eol firmware" .RSU file. (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).
```

6. When installation and signature verification finish, the end of life (EOL) application is launched.

```
-------------------------------------------------
End Of Life (EOL) process of user program
-------------------------------------------------
[R_FWUP_SetEndOfLife] erase install area (code flash):OK
[R_FWUP_SetEndOfLife] update bank1 LIFECYCLE_STATE to [LIFECYCLE_STATE_EOL]
[EOL_main] EOL process completely. Bank swap and software reset.
[R_FWUP_ActivateNewImage] Changing the Startup Bank
[R_FWUP_ResetDevice] Resetting the device.
[R_FWUP_ResetDevice] Swap bank...
```

7. When the end of life (EOL) application finishes, processing returns to the bootloader, and EOL processing is executed within the bootloader.

```
-------------------------------------------------
RX65N secure boot program
-------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xe0 [LIFECYCLE_STATE_EOL]
bank 1 status = 0xf8 [LIFECYCLE_STATE_VALID]
```

8. When the following message is output, EOL processing has completed successfully.

```
End Of Life process finished.
```

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the FIT module.

Table 5.1 Confirmed Operation Environment (Ver. 1.01)

Item	**Contents**
Integrated development environment	Renesas Electronics e^2 studio 2021 04
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.02.00 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.1.01
Board used	Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit+ for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit+ for RX65N (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX66T (product No.: RTK50566T0S00000BE) Renesas Starter Kit+ for RX72N (product No.: RTK5572NNxxxxxxxBE)
USB serial converter board	Pmod USBUART (Digilent, Inc.) https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.16.2021	—	First edition issued
1.01	Jun. 21, 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)

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.