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.

By using the FIT module, users can easily incorporate firmware update functionality and secure boot 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.

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

Operation Confirmation Devices

RX24T Group

RX26T Group

RX65N, RX651 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 document.

- 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 BYTEQ Module Using Firmware Integration Technology (R01AN1683)

Target Compilers

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

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

Contents

1. Overview

1.1 About the Firmware Update Module

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

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

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

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

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

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

- Dual-bank method
- Linear mode partial update method
- Linear mode full update method

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

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

1.2 Configuration of Firmware Update Module

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

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

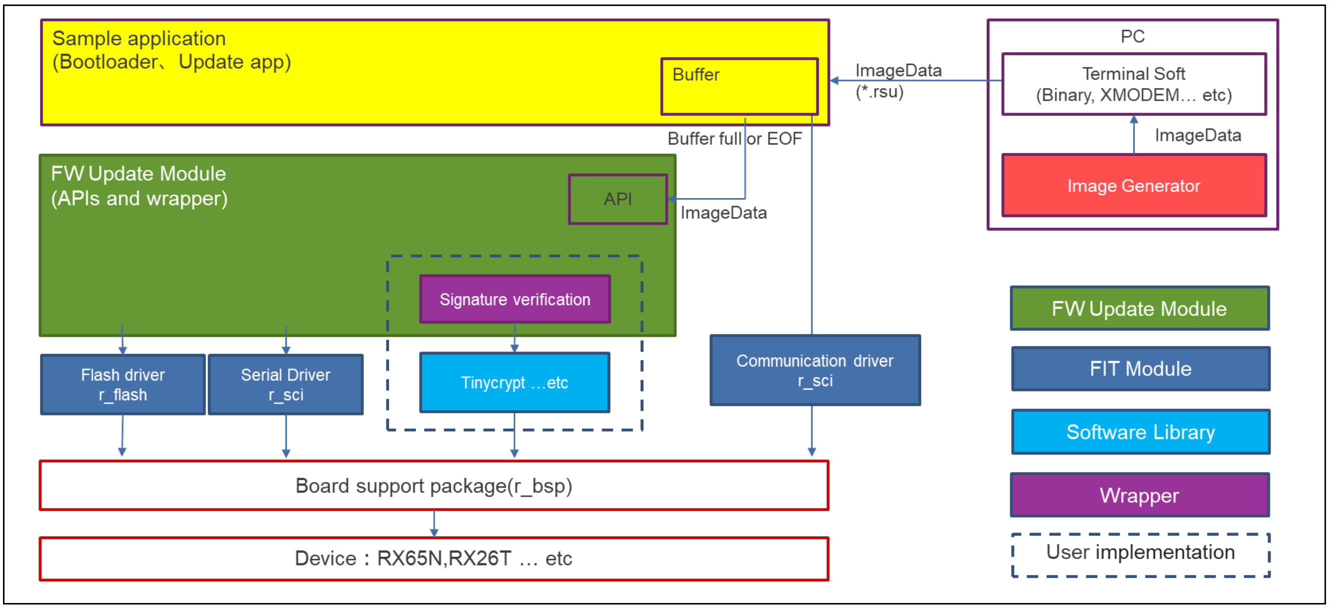


Figure 1.1 Configuration of Modules in Sample Bootloader and Application Program

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

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
Device driver	RX Family SCI Module Using Firmware Integration Technology (R01AN1815)	r_sci_rx
Middleware	RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)	r_byteq

1.3 Firmware Update Operation

The firmware update module for the RX Family supports both the dual mode and the linear mode of the MCU's on-chip flash memory.

Dual mode uses the hardware's dual bank function to store the firmware to be updated (update image) on the buffer plane and then swap banks with the main plane to provide a dual bank method of updating..

For linear mode, two methods are provided: one in which the firmware update (update image) is stored temporarily on the buffer plane and another in which it is programmed directly to the main plane.

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

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

The update method support status varies by device and flash memory capacity, as detailed below.

Table 1.2 Supported Update Methods for Each Product

Type Product	2MB	1,5MB	1MB	756KB	512KB	384KB	256KB	128KB
RX65N/651	a/b	a/b	b	b	b	-	-	-
RX26T	-	-	-	-	a/b	-	b	b
RX24T	-	-	-	-	b	b	b	b
a: Dual-Bank Method b: Linear Mode Partial Update Method / Full Update Method -: Not supported								

1.3.1 Dual-Bank Method

The update image is stored in the buffer plane in the on-chip flash memory, and, after it is validated, the banks are swapped, exchanging the main plane and buffer plane.
his method allows the application program to contain the firmware update functionality.
This means that if the firmware update fails before bank swapping occurs, the pre-update image in the main plane can be launched to retry the firmware update.
Since the on-chip flash memory is divided into two portions by the dual-bank functionality, the size of the on-chip flash memory available to store the application program is equal to the size of one of the two portions into which the on-chip flash memory has been divided minus the size of the bootloader.

1.3.1.1 Operation of Dual-Bank Method

The update image is stored in the buffer plane using the dual-bank functionality of the on-chip flash memory, and the firmware update is accomplished by using the bank-swapping functionality to exchange the banks.

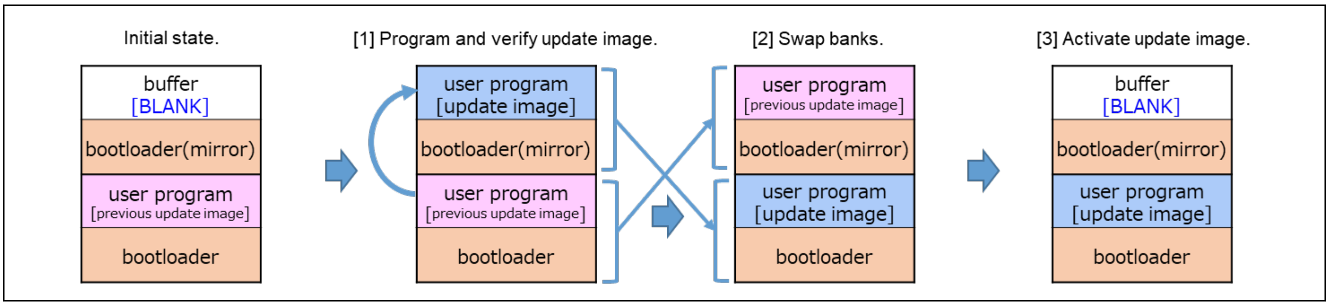


Figure 1.2 Operation of Dual-Bank Method

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

[2] Swap banks.
If verification is successful, the banks are swapped.

[3] Activate update image.
The buffer plane is erased, and the main plane is activated.

1.3.2 Linear Mode Partial Update Method

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

1.3.2.1 Operation of Linear Mode Partial Update Method

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

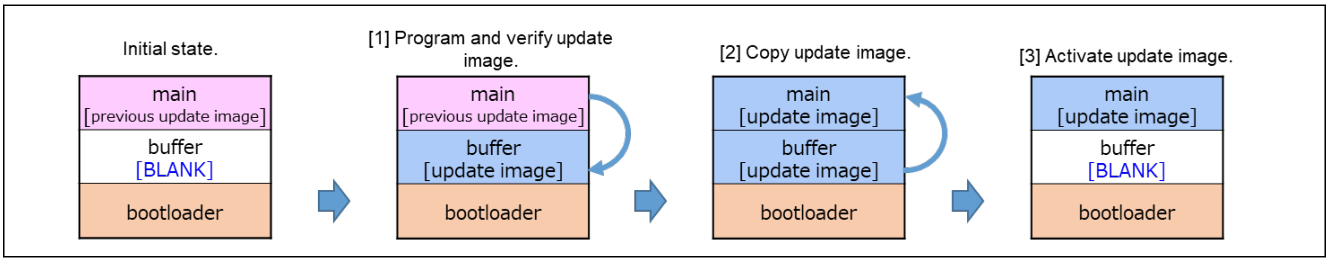


Figure 1.3 Operation of Partial Update Method

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

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

[3] Activate update image.
The buffer plane is erased, and the main plane is activated.

1.3.3 Linear Mode Full Update Method

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

1.3.3.1 Operation of Linear Mode Full Update Method

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

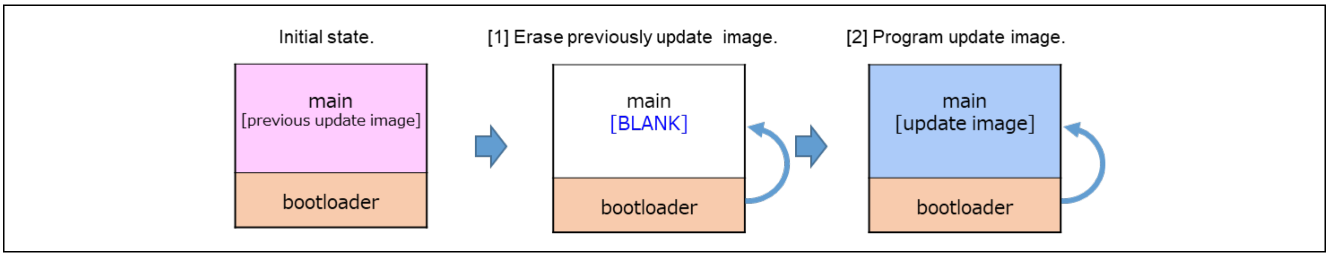


Figure 1.4 Operation of Full Update Method

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

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

1.4 Initial State of Firmware Update

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

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

1.4.1 Initial State of Dual-Bank Method

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

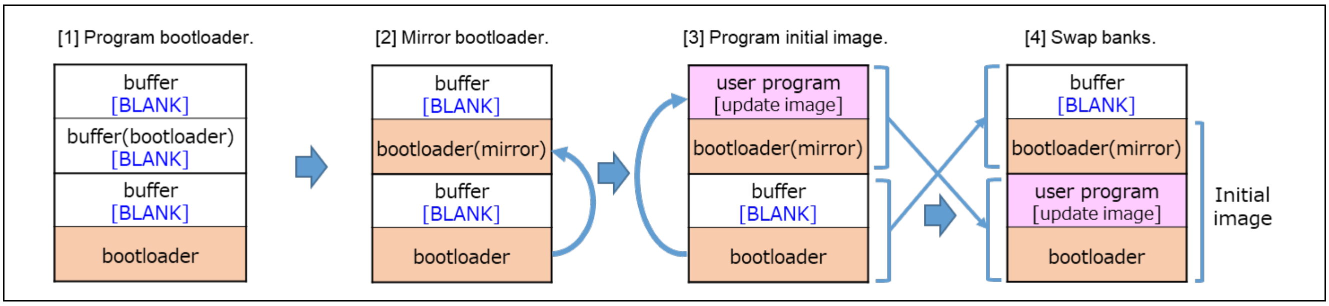


Figure 1.5 Initial Firmware Update Settings Utilizing Bootloader (Example of Dual-Bank Method)

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

[2] Mirror bootloader.
The bootloader is mirrored to bank 1 by the bootloader.

[3] Program initial image.
The initial image is downloaded from an external source and programmed to the buffer plane using the functionality of the bootloader. The programmed firmware is verified.

[4] Swap banks.
If verification is successful, the banks are swapped and processing ends.

1.4.2 Initial State of Linear Mode Partial Update Method

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

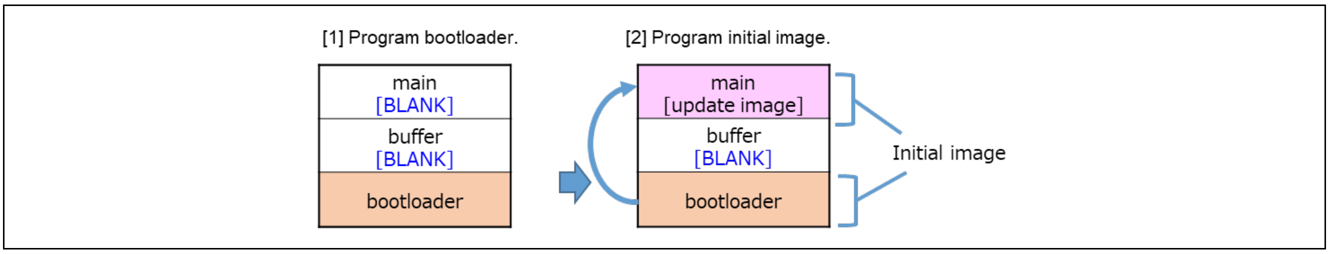


Figure 1.6 Initial Firmware Update Settings Utilizing Bootloader (Example of Partial Update Method)

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

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

1.4.3 Initial State of Linear Mode Full Update Method

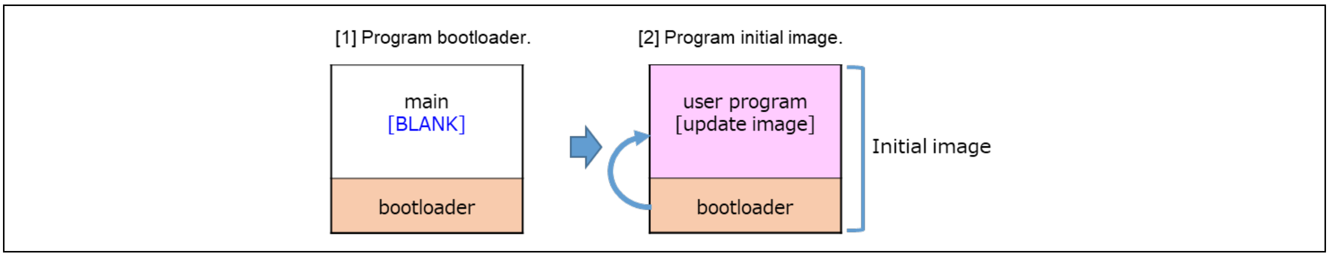


Figure 1.7 Initial Firmware Update Settings Utilizing Bootloader (Example of Full Update Method)

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

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

1.5 Package Contents

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

Table 1.3 Folder Structure of Firmware Update Module Package

Folder Name	Description
r01an6850jj0200-rx-fwupdate.zip\	
├─Demos	Sample projects
│　├─rx	
│　│　├─modules	Drivers and libraries
│　│　│　├─3rd_party	
│　│　│　│　└─tinycrypt	Crypto library
│　│　│　└─etc	
│　│　│　　└─base64	Base64 decode
│　│　├─rx24t-rsk	Sample project for RX24T
│　│　│　├─w_buffer	Linear Mode Partial Update Method
│　│　│　│　├─e2_ccrx	CC-RX version
│　│　│　│　│　├─boot_loader	Bootloader
│　│　│　│　│　├─fwup_leddemo	LED illumination application
│　│　│　│　│　└─fwup_main	User applications including firmware update
│　│　│　│　├─e2_gcc	GCC version
│　│　│　│　│　├─boot_loader	Bootloader
│　│　│　│　│　├─fwup_leddemo	LED illumination application
│　│　│　│　│　└─fwup_main	User applications including firmware update
│　│　│　│　└─iar	IAR version
│　│　│　│　　├─boot_loader	Bootloader
│　│　│　│　　├─fwup_leddemo	LED illumination application
│　│　│　│　　└─fwup_main	User applications including firmware update
│　│　│　└─wo_buffer	Linear Mode Full Update Method
│　│　│　　├─e2_ccrx	CC-RX version
│　│　│　　│　├─boot_loader	Bootloader
│　│　│　　│　└─fwup_leddemo	LED illumination application
│　│　│　　├─e2_gcc	GCC version
│　│　│　　│　├─boot_loader	Bootloader
│　│　│　　│　└─fwup_leddemo	LED illumination application
│　│　│　　└─iar	IAR version
│　│　│　　　├─boot_loader	Bootloader
│　│　│　　　└─fwup_leddemo	LED illumination application
│　│　├─rx26t-mck	Sample project for RX26T
│　│　│　├─dualbank	Dual-Bank Method
│　│　│　│　├─e2_ccrx	CC-RX version
│　│　│　│　│　├─boot_loader	Bootloader
│　│　│　│　│　├─fwup_leddemo	LED illumination application
│　│　│　│　│　└─fwup_main	User applications including firmware update
│　│　│　│　├─e2_gcc	GCC version
│　│　│　│　│　├─boot_loader	Bootloader
│　│　│　│　│　├─fwup_leddemo	LED illumination application
│　│　│　│　│　└─fwup_main	User applications including firmware update

Folder Name	Description
│ │ │ │ └─iar	IAR version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ ├─w_buffer	Linear Mode Partial Update Method
│ │ │ │ ├─e2_ccrx	CC-RX version
│ │ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ │ ├─e2_gcc	GCC version
│ │ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ │ └─iar	IAR version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ └─wo_buffer	Linear Mode Full Update Method
│ │ │ ├─e2_ccrx	CC-RX version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ └─fwup_leddemo	LED illumination application
│ │ │ ├─e2_gcc	GCC version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ └─fwup_leddemo	LED illumination application
│ │ │ └─iar	IAR version
│ │ │ ├─boot_loader	Bootloader
│ │ │ └─fwup_leddemo	LED illumination application
│ │ └─rx65n-rsk	Sample project for RX26T
│ │ ├─dualbank	Dual-Bank Method
│ │ │ ├─e2_ccrx	CC-RX version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ ├─e2_gcc	GCC version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ ├─w_buffer	Linear Mode Partial Update Method
│ │ │ ├─e2_ccrx	CC-RX version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ ├─e2_gcc	GCC version
│ │ │ │ ├─boot_loader	Bootloader
│ │ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ │ └─fwup_main	User applications including firmware update
│ │ │ └─iar	IAR version
│ │ │ ├─boot_loader	Bootloader
│ │ │ ├─fwup_leddemo	LED illumination application
│ │ │ └─fwup_main	User applications including firmware update

Folder Name	Description
│ │ └─wo_buffer	Linear Mode Full Update Method
│ │ ├─e2_ccrx	CC-RX version
│ │ │ ├─boot_loader	Bootloader
│ │ │ └─fwup_leddemo	LED illumination application
│ │ ├─e2_gcc	GCC version
│ │ │ ├─boot_loader	Bootloader
│ │ │ └─fwup_leddemo	LED illumination application
│ │ └─iar	IAR version
│ │ ├─boot_loader	Bootloader
│ │ └─fwup_leddemo	LED illumination application
│ └─r_fwup	Firmware update module (for sample projects)
├─FITModules	Firmware update module
│ ├─r_config	Configuration file
│ └─r_fwup	Source code
└─RenesasImageGenerator	Renesas Image Generator (Python program and parameter files)
└─RenesasImageGenerator.zip	
├─image-gen.py	Python program for Renesas Image Generator
├─RX65N_DualBank_ImageGenerator_PRM.csv	Parameter file for RX65N dual-bank method
├─RX65N_Linear_Half_ImageGenerator_PRM.csv	Parameter file for RX65N linear mode partial update method
├─RX65N_Linear_Full_ImageGenerator_PRM.csv	Parameter file for RX65N linear mode full update method
├─RX26T_DualBank_ImageGenerator_PRM.csv	Parameter file for RX26T dual-bank method
├─RX26T_Linear_Half_ImageGenerator_PRM.csv	Parameter file for RX26T linear mode partial update method
├─RX26T_Linear_Full_ImageGenerator_PRM.csv	Parameter file for RX26T linear mode full update method
├─RX24T_Linear_Half_ImageGenerator_PRM.csv	Parameter file for RX24T linear mode partial update method
└─RX24T_Linear_Full_ImageGenerator_PRM.csv	Parameter file for RX24T linear mode full update method

1.6 API Overview

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

Table 1.4 API Functions

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

Figure 1.8 to Figure 1.12 show flowcharts of example implementations of a bootloader and application program corresponding to each firmware update method described in in 1.3, Firmware Update Operation, using the APIs provided in this module.

For details, refer to the source code of the demo projects included in the package associated with this application note.

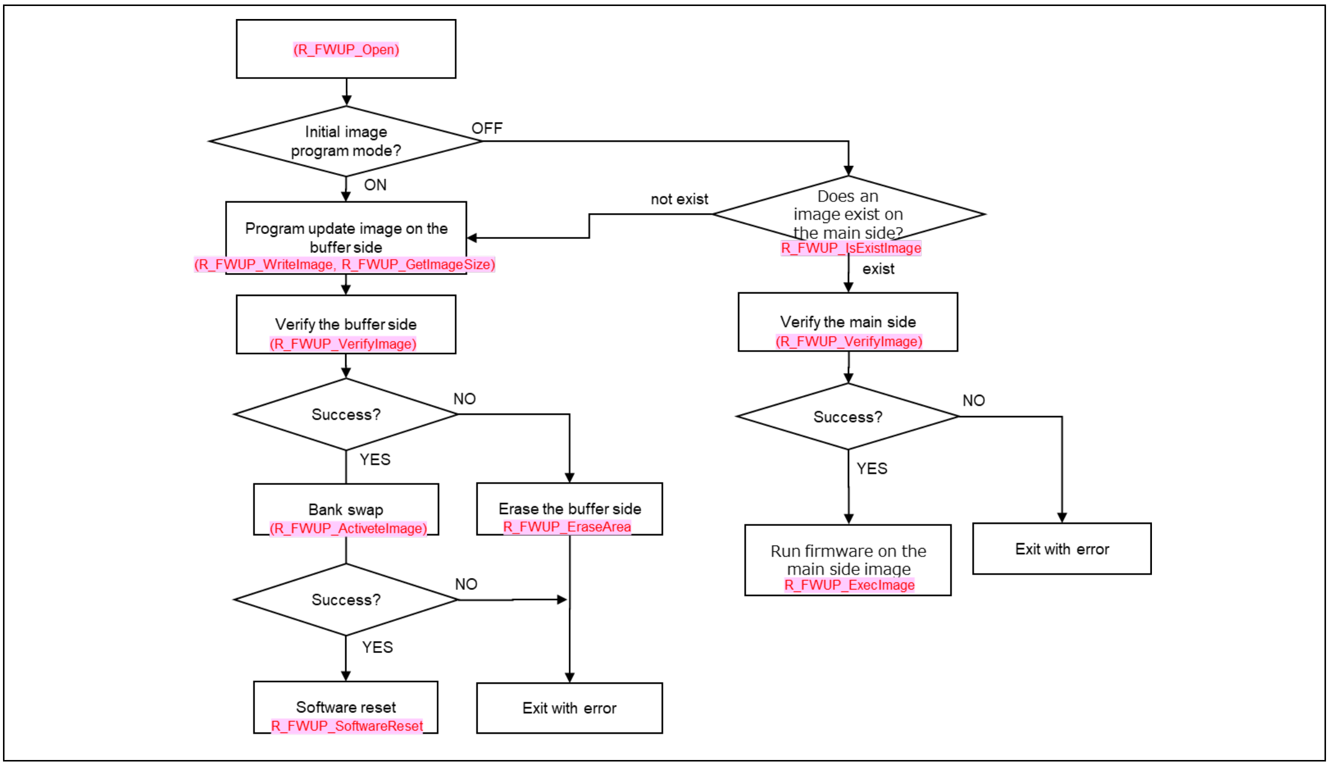


Figure 1.8 Bootloader Implementation Example for Partial/Full Update Method (Using Buffer Plane)

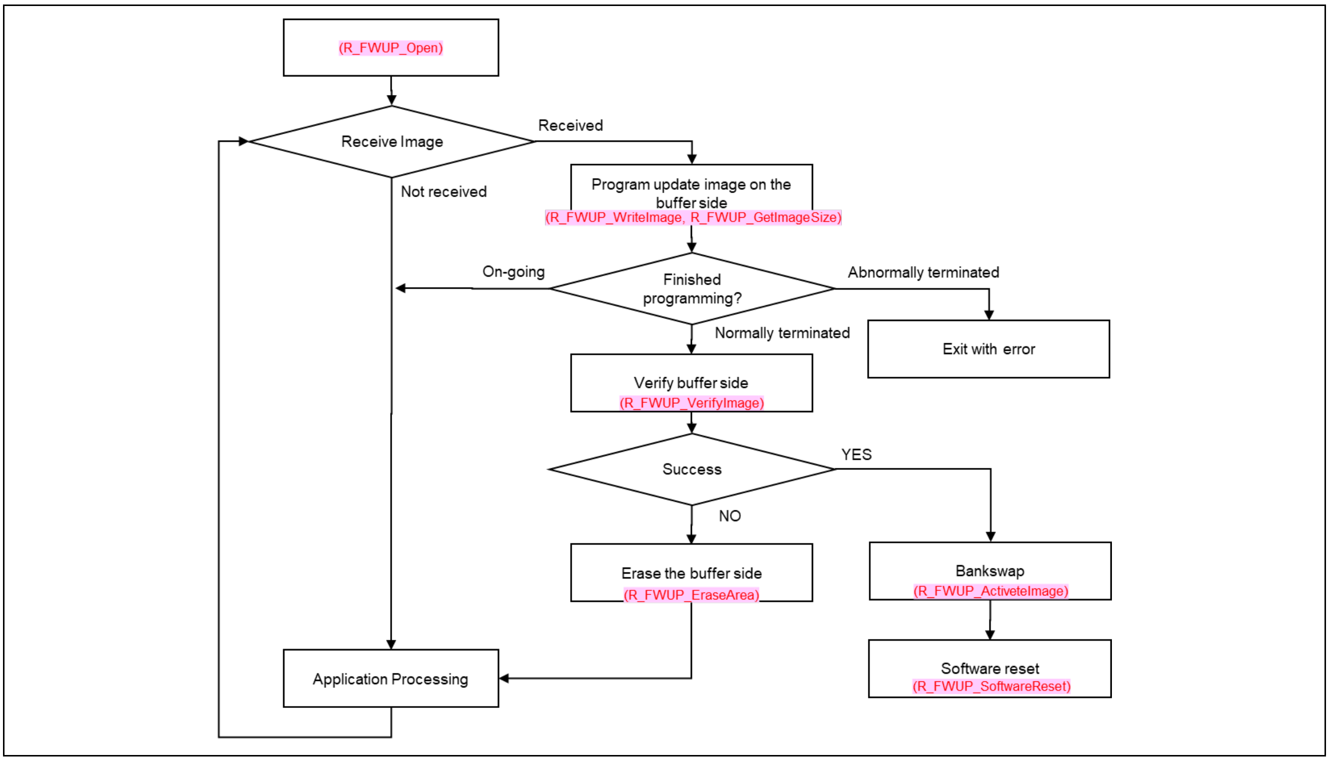


Figure 1.9 Application Program Implementation Example for Partial/Full Update Method (Using Buffer Plane)

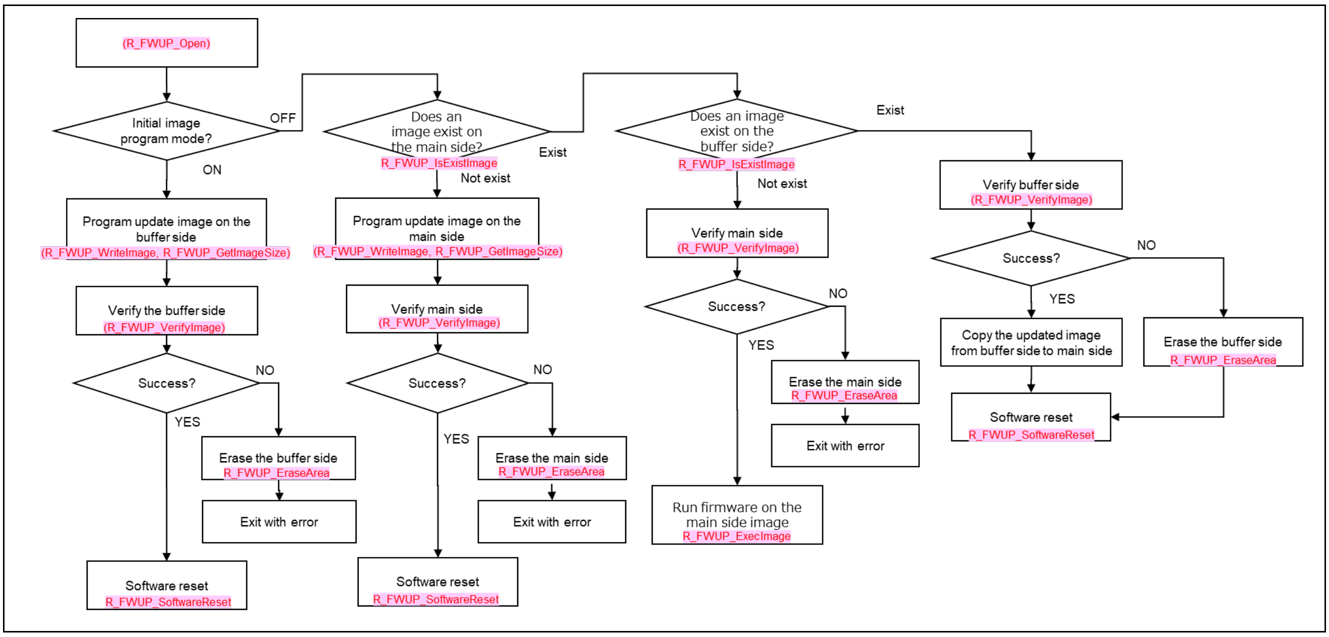


Figure 1.10 Bootloader Implementation Example for Full Update Method (No Buffer)

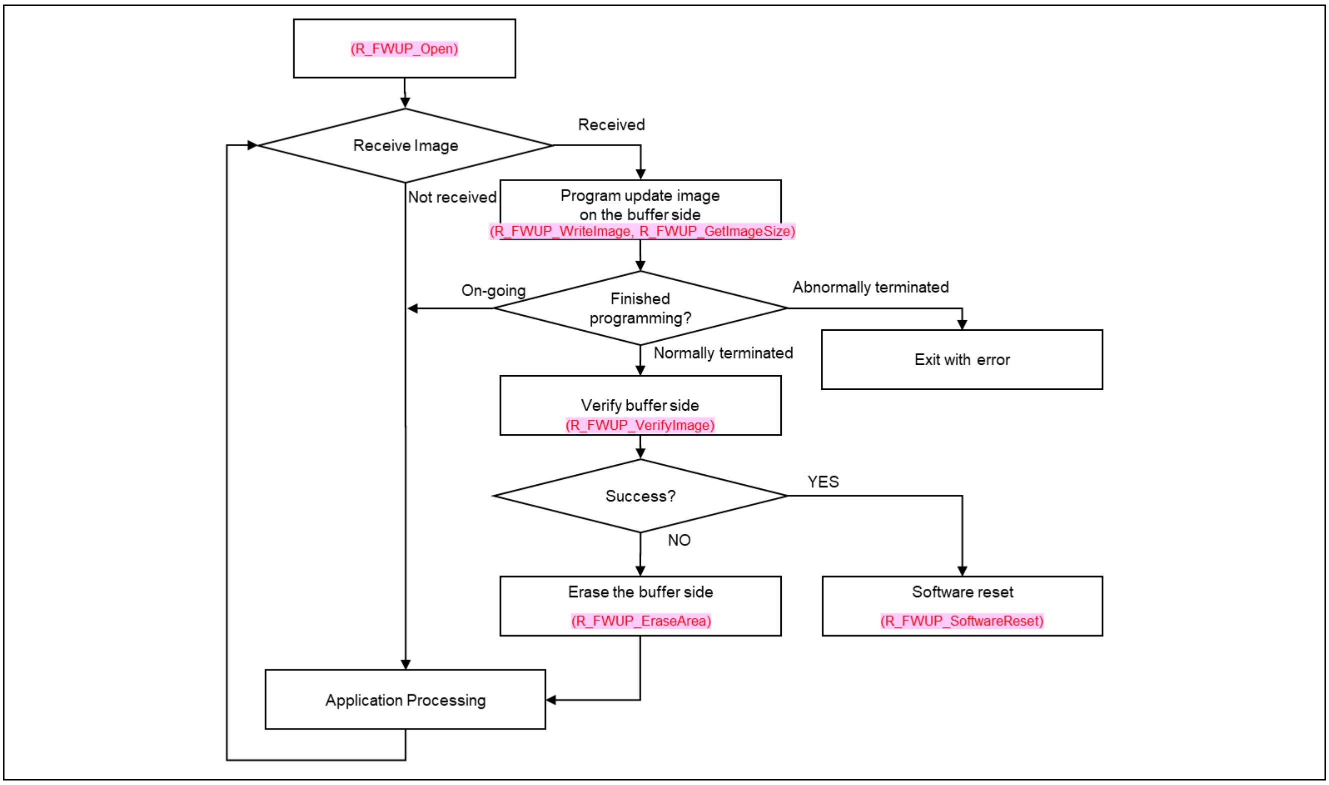


Figure 1.11 Application Program Implementation Example for Linea Mode Partial Update Method

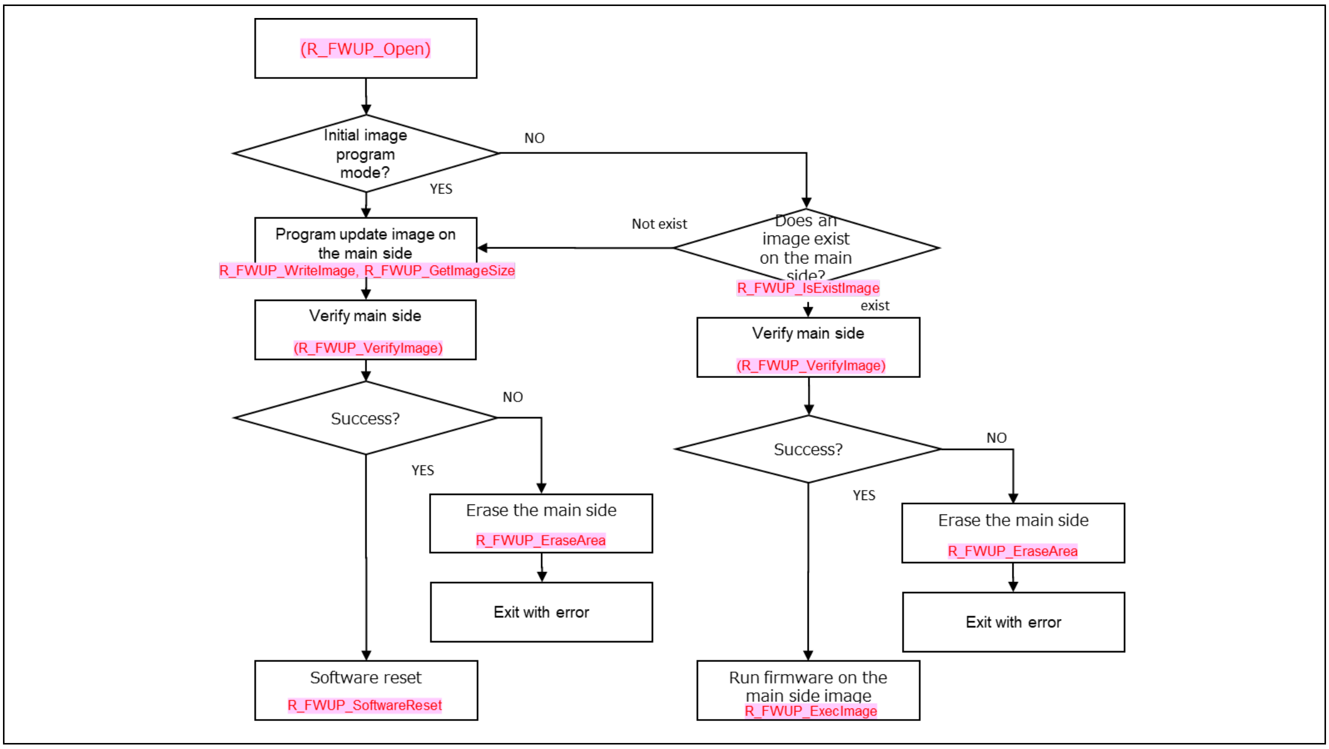


Figure 1.12 Bootloader Implementation Example for Full Update Method

2. API Information

2.1 Hardware Requirements

The MCU used must support the following functions:

- Flash memory

2.2 Software Requirements

The module is dependent upon the following drivers:

- Board support package (r_bsp)
- Flash module (r_flash_rx)
- Serial communications interface (SCI: asynchronous/clock synchronous) (r_sci_rx)
- Byte queue buffer module (r_byteq)

2.3 Supported Toolchains

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

2.4 Header Files

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

2.5 Integer Types

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

2.6 Compile Settings

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

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

Table 2.1 Configuration Settings

Configuration options in r_fwup _config.h	
FWUP_CFG_UPDATE_MODE	Update method 0: Dual-Bank Method 1: Linea Mode Partial Update Method 2: Linea Mode Full Update Method 3: Not available for RX
FWUP_CFG_FUNCTION_MODE	Specifies how the module is used. 0: Bootloader 1: Application program
FWUP_CFG_MAIN_AREA_ADDR_L	Specifies the start address of the main plane.
FWUP_CFG_BUF_AREA_ADDR_L	Specifies the start address of the buffer plane (in on-chip flash memory).
FWUP_CFG_AREA_SIZE	Specifies the size of the main plane and buffer plane.
FWUP_CFG_CF_BLK_SIZE	Specifies the block size of the on-chip code flash.
FWUP_CFG_EXT_BUF_AREA_ADDR_L	Specifies the start address of the buffer plane in external flash memory. (Not subject to change in RX)
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE	Specifies the block size or sector size of the external flash memory. (Not subject to change in RX)
FWUP_CFG_DF_ADDR_L	Start address of data flash.
FWUP_CFG_DF_BLK_SIZE	Block size of data flash.
FWUP_CFG_DF_NUM_BLKS	Block count of data flash. Specify 0 if there is no data flash.
FWUP_CFG_SIGNATURE_VERIFICATION	Verification method 0: ECDSA + SHA256 1: SHA256
FWUP_CFG_PRINTF_DISABLE	Log display setting 0: Enable 1: Disable

2.7 Sample Project Code Sizes

The tables below show the ROM, RAM, and maximum stack sizes for the sample projects included in the package associated with this application note. The values in the table below have been confirmed under the following conditions:

Module revision: Firmware update module for RX, v2.0.0

Compiler version: Renesas Electronics C/C++ Compiler for RX Family V3.04.00
GCC for Renesas RX 8.3.0.202202
IAR C/C++ Compiler for Renesas RX 4.20.1

CC-RX

- Optimization level: Size and execution speed (-O default)
- Delete variables/functions that have never been referenced (optimize=symbol_delete)
- Generate reduced function I/O functions (Yes: maximum reduced version)

GCC

- Optimization level: Size (-Os)
- Use newlib-nano (--specs=nano.specs)

IAR

- Optimization level: High (balanced)

Table 2.2 ROM, RAM, and Maximum Stack Sizes for Sample Projects

ROM, RAM, and Stack Code Sizes					
Device	Category	Memory Used (byte)			Remarks
		CC-RX	GCC	IAR	
RX65N	ROM	24486	25962	22901	boot_loader
		15314	15552	12343	fwup_leddemo
		24245	26166	22777	fwup_main
	RAM	5282	5372	4871	boot_loader
		8045	7908	5533	fwup_leddemo
		5554	5628	6167	fwup_main
	Stack	552	532	1448	boot_loader
		188	68	268	fwup_leddemo
		552	532	1444	fwup_main
RX26T	ROM	23849	24912	22156	boot_loader
		13198	14188	11383	fwup_leddemo
		23674	24078	21925	fwup_main
	RAM	4077	4604	5602	boot_loader
		3289	4196	5409	fwup_leddemo
		4302	5244	5730	fwup_main
	Stack	552	532	1448	boot_loader
		188	68	276	fwup_leddemo
		552	532	1444	fwup_main
RX24T	ROM	21449	22748	21649	boot_loader
		20703	11528	9695	fwup_leddemo
		20903	21848	21095	fwup_main
	RAM	5114	6652	4380	boot_loader
		3411	3892	2464	fwup_leddemo
		5242	6780	4468	fwup_main
	Stack	552	532	1028	boot_loader
		188	68	252	fwup_leddemo
		552	532	1020	fwup_main

2.8 Arguments

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

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

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

2.9 Return Values

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

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

2.10 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 e2 studio

By using the Smart Configurator in e2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e2 studio

By using the FIT Configurator in e2 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:

3. API Functions

3.1 R_FWUP_Open Function

Table 3.1 R_FWUP_Open Function Specifications

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

3.2 R_FWUP_Close Function

Table 3.2 R_FWUP_Close Function Specifications

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

3.3 R_FWUP_IsExistImage Function

Table 3.3 R_FWUP_IsExistImage Function Specifications

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

3.4 R_FWUP_EraseArea Function

Table 3.4 R_FWUP_EraseArea Function Specificationss

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

3.5 R_FWUP_GetImageSize Function

Table 3.5 R_FWUP_GetImageSize Function Specificationss

Format	uint32_t R_FWUP_GetImageSize(void)	
Description	Returns the size of the image in bytes.	
Parameters	None	
Return Values	0	Acquisition in progress
	1 or more	Image size
Special Notes	—	

3.6 R_FWUP_WriteImageHeader Function

Table 3.6 R_FWUP_WriteImageHeader Function Specifications

Format	e_fwup_err_t R_FWUP_WriteImageHeader (e_fwup_area_t area, uint8_t FWUP_FAR *p_sig_type, uint8_t FWUP_FAR *p_sig, uint32_t sig_size)	
Description	Writes a signature that the bootloader uses for verification to the header of the image in the designated area.	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER) p_sig_type: Signature type character string “hash-sha256” or “sig-sha256-ecdsa” p_sig: Signature sig_size: Length of signature (Should be set to 64.)	
Return Values	FWUP_SUCCES	Write completed
	FWUP_ERR_FLASH	Flash module error
Special Notes	—	

3.7 R_FWUP_WriteImageProgram Function

Table 3.7 R_FWUP_WriteImageProgram Function Specifications

Format	e_fwup_err_t R_FWUP_WriteImageProgram (e_fwup_area_t area, uint8_t *p_buf, uint32_t buf_size)	
Description	Writes the program portion of the image to the specified area. Continue calling this function until the total size of the image is reached. The image size is obtained by R_FWUP_GetImageSize().	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER) p_buf: Buffer for program portion of image buf_size: Buffer size*[1]	
Return Values	FWUP_SUCCES	Write completed
	FWUP_PROGRESS	Writing in progress
	FWUP_ERR_FLASH	Flash module error
Special Notes	1. Specify a multiple of the code flash write unit (for example, 64, 128, or 256).	

3.8 R_FWUP_WriteImage Function

Table 3.8 R_FWUP_WriteImage Function Specifications

Format	e_fwup_err_t R_FWUP_WriteImage(e_fwup_area_t area, uint8_t *p_buf, uint32_t buf_size)	
Description	Writes an image (header portion + program portion) to the specified area. Continue calling this function until the total size of the image is reached. The image size is obtained by R_FWUP_GetImageSize().	
Parameters	area: Main plane (FWUP_AREA_MAIN) or buffer plane (FWUP_AREA_BUFFER) p_buf: Image (header + program) buffer buf_size: Buffer size*[1]	
Return Values	FWUP_SUCCES	Write completed
	FWUP_PROGRESS	Writing in progress
	FWUP_ERR_FLASH	Flash module error
Special Notes	1. Specify a multiple of the code flash write unit (for example, 64, 128, or 256).	

3.9 R_FWUP_VerifyImage Function

Table 3.9 R_FWUP_VerifyImage Function Specifications

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

3.10 R_FWUP_ActivateImage Function

Table 3.10 R_FWUP_ActivateImage Function Specifications

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

3.11 R_FWUP_ExecImage Function

Table 3.11 R_FWUP_ExecImage Function Specifications

Format	void R_FWUP_ExecImage(void)
Description	Runs the program in a valid image.
Parameters	None
Return Values	None
Special Notes	—

3.12 R_FWUP_SoftwareReset Function

Table 3.12 R_FWUP_SoftwareReset Function Specifications

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

3.13 R_FWUP_SoftwareDelay Function

Table 3.13 R_FWUP_SoftwareDelay Function Specifications

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

3.14 R_FWUP_GetVersion Function

Table 3.14 R_FWUP_GetVersion Function Specifications

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

3.15 Wrapper Functions (r_fwup_wrap_verify.c, h)

The demo project provided in this package uses Tinycrypt as the crypto library. If you wish to use another crypto library, please implement the crypto library you wish to use and modify the implementation in this wrapper function.

3.15.1 r_fwup_wrap_sha256_init Function

Table 3.15 r_fwup_wrap_sha256_init Function Specifications

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

3.15.2 r_fwup_wrap_sha256_update Function

Table 3.16 r_fwup_wrap_sha256_update Function Specifications

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

3.15.3 r_fwup_wrap_sha256_final Function

Table 3.17 r_fwup_wrap_sha256_final Function Specifications

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

3.15.4 r_fwup_wrap_verify_ecdsa Function

Table 3.18 r_fwup_wrap_verify_ecdsa Function Specifications

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

3.15.5 r_fwup_wrap_get_crypt_context Function

Table 3.19 r_fwup_wrap_get_crypt_context Function Specifications

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

4. Renesas Image Generator

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

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

See 4.1 for how to generate an image, and 4.2 to 4.3 for details on image configuration and parameter files.

Renesas Image Generator is a program that runs on Python.

4.1 Image Generation Methods

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

See 4.1.1 for how to generate an initial image, and 4.1.2 for how to generate an update image.

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

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

Some image-gen.py command options are required and others are optional. Table 4.1 lists the required image-gen.py options, and Table 4.2 lists the optional image-gen.py options.

Table 4.1 Required Options of image-gen.py

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

Table 4.2 Optional Options of image-gen.py

Option	Description
-ibp <file>	Specifies the bootloader. For the character string < file >, specify mot file name (the full path including the file name) of the bootloader program. Specify this option when generating a mot file.
--key <file>	Specify the name of the key file to be used to sign the image using ECDSA. (This option does not need to be set if sha256 is specified for the -vt option.) Store the file **secp256r1.privatekey** in the command execution folder. If the file name has been changed, specify the full path including the file name.
-vt <VerificationType>[sha256 / ecdsa]	Specifies the image verification method in the firmware update module. Appends the hash of the image if sha256 is specified, and the signature of the image if ecdsa is specified. If this option is omitted, SHA-256 is used. If ecdsa is specified, a key file specified with -key is required.
-ff <FileFormat>	Specifies the RSU format type. If BareMetal is specified, it will generate an updated image for this demo project. The updated image for BareMetal adds RSU header signature information. If RTOS is specified, generate update image for FreeRTOS OTA. The update image for FreeRTOS OTA does not add RSU header signature information (0x200 bytes data from the beginning of the update image). If this option is omitted, BareMetal is used.
-h	Output a list of commands. Specify this option to display help information for the tool.

4.1.1 Initial Image Generation Method

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

Command input example

```
> python image-gen.py -iup fwup_main.mot -ip
RX65N_DualBank_ImageGenerator_PRM.csv -o initial_firm -ibp
boot_loader.mot -vt ecdsa
```

fwup_main.mot: The mot file name of the user application program
RX65N_DualBank_ImageGenerator_PRM.csv: The name of the file containing the parameters to be input
(Example of dual bank mode)
initial_firm: The file name of the initial image file to be output
boot_loader.mot: The mot file name of the bootloader program
ecdsa: Specifies that ECDSA is used to sign the image.

4.1.2 Update Image Generation Method

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

Command input example

```
> python image-gen.py -iup fwup_leddemo.mot -ip
RX65N_DualBank_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa
```

fwup_leddemo.mot: The mot file name of the user application program to be applied as an update
RX65N_DualBank_ImageGenerator_PRM.csv: The name of the file containing the parameters to be input
(Example of dual bank mode)
fwup_leddemo: The file name of the update image file to be output
ecdsa: Specifies that ECDSA is used to sign the image.

4.2 Image File

4.2.1 Update Image File

Figure 4.1 shows the configuration diagram of the update image file generated by Renesas Image Generator.

For the format of the RSU header, see Table 43.

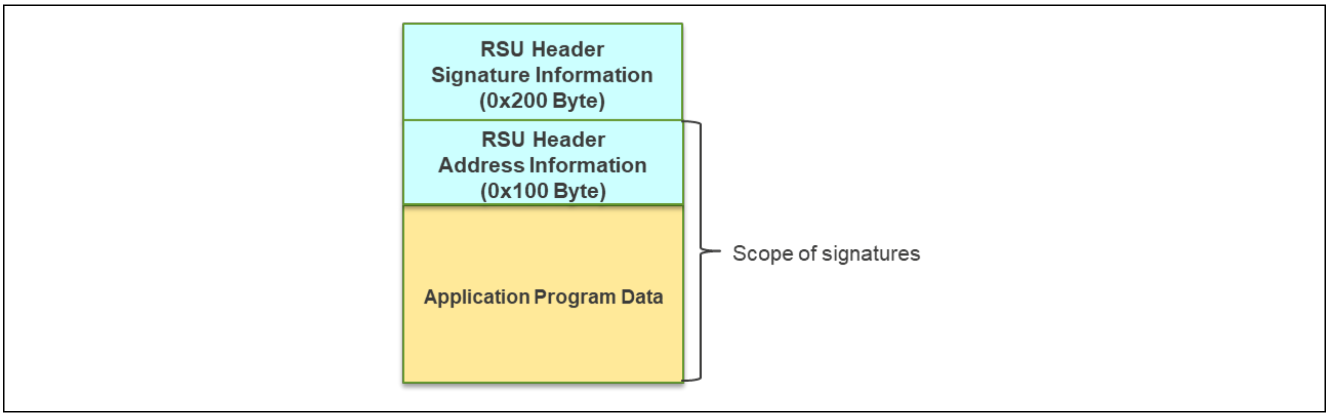


Figure 4.1 Configuring the update image file

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

The update image file has the same configuration for the dual bank method, linear mode half-updating method, and linear mode full-updating method.

Table 4.3 RSU Header Format (1/2)

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

Table 4.3 RSU Header Format (2/2)

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

See Figure 4.2 for the mechanism of generating the update image file.

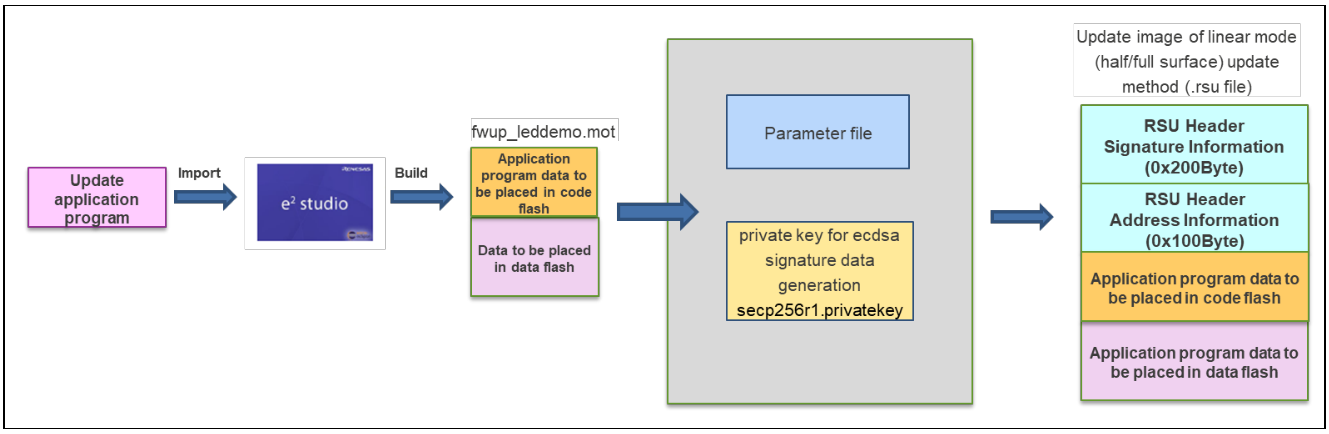

Update application program
Import
e² studio
Build
fwup_leddemo.mot
Application program data to be placed in code flash
Data to be placed in data flash
Parameter file
private key for ecdsa signature data generation
secp256r1.privatekey
Update image of linear mode (half/full surface) update method (.rsu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

Figure 4.2 Updating image of dual bank method/linear mode (half side/whole side) updating method

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

4.2.1 Update Image File

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

Figure 4.3 and Figure 4.4 also show a diagram of the initial image file (dual bank method/linear mode (half/full surface) update method).

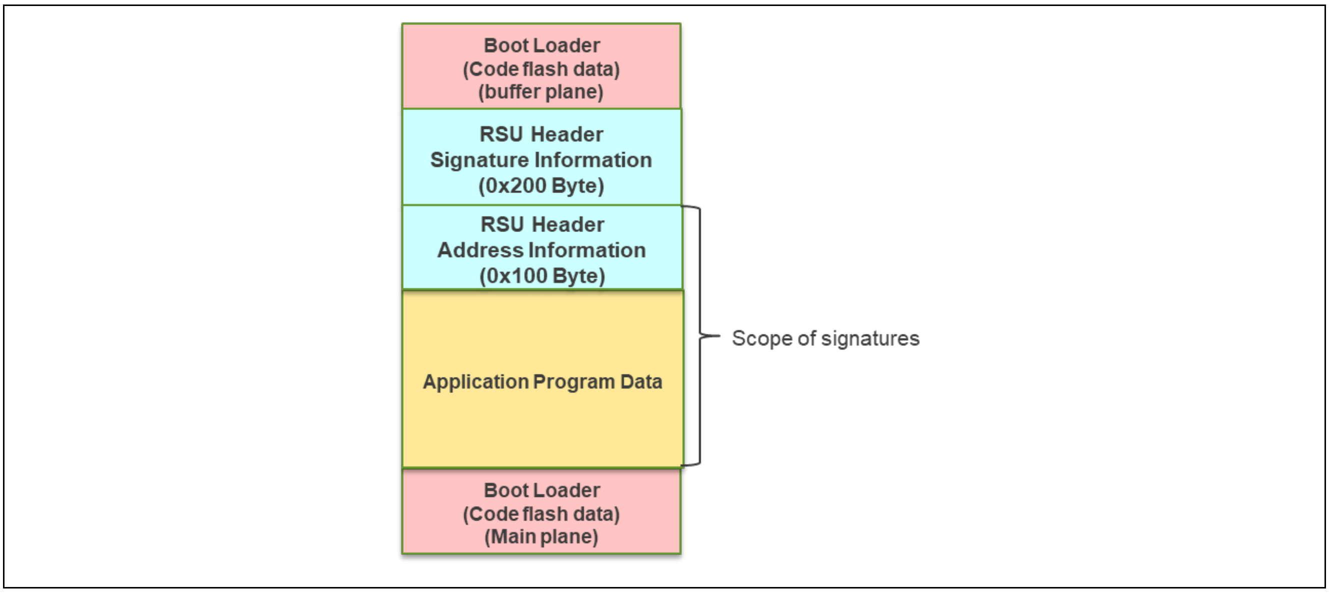


Figure 4.3 Initial image file (dual bank method) configuration

The initial image of the dual-bank method places the same bootloader code flash data on both the main and buffer sides of the code flash to support the bank switching function. The bootloader data to be placed on the main side of Code Flash uses the data in the user-generated bootloader file (boot_loader.mot) as is. The bootloader to be placed on the buffer side of the code flash is the bootloader data on the main side with the address information replaced with the bootloader location on the buffer side.

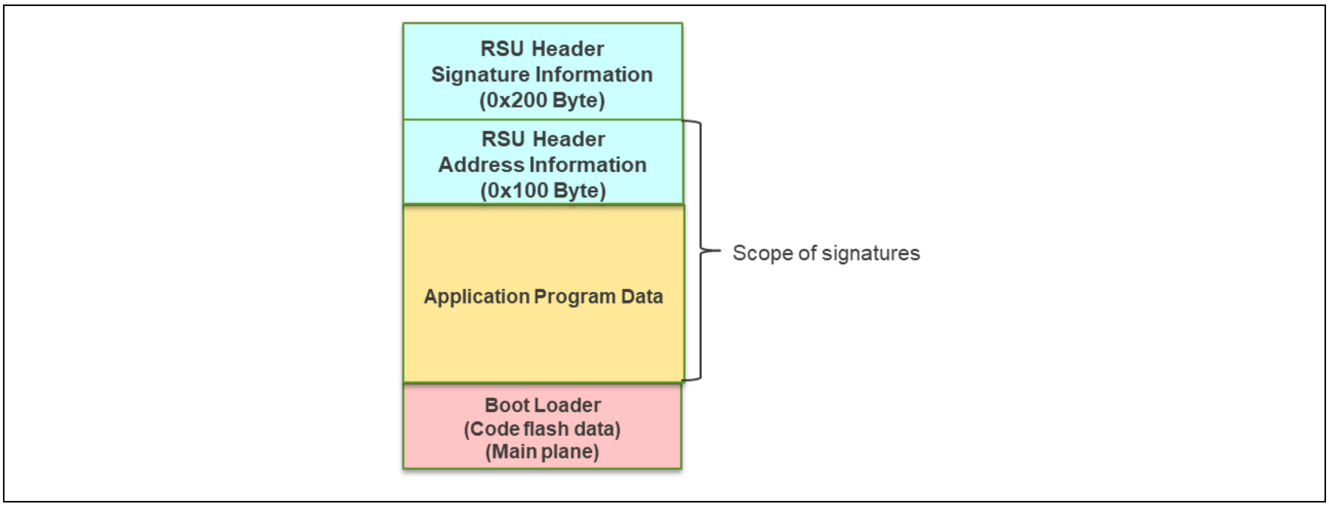


Figure 4.4 Composition of initial image file (linear mode (half/full surface) update method)

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

See Figure 4.5 and Figure 4.6 for the mechanism that generates the initial image file.

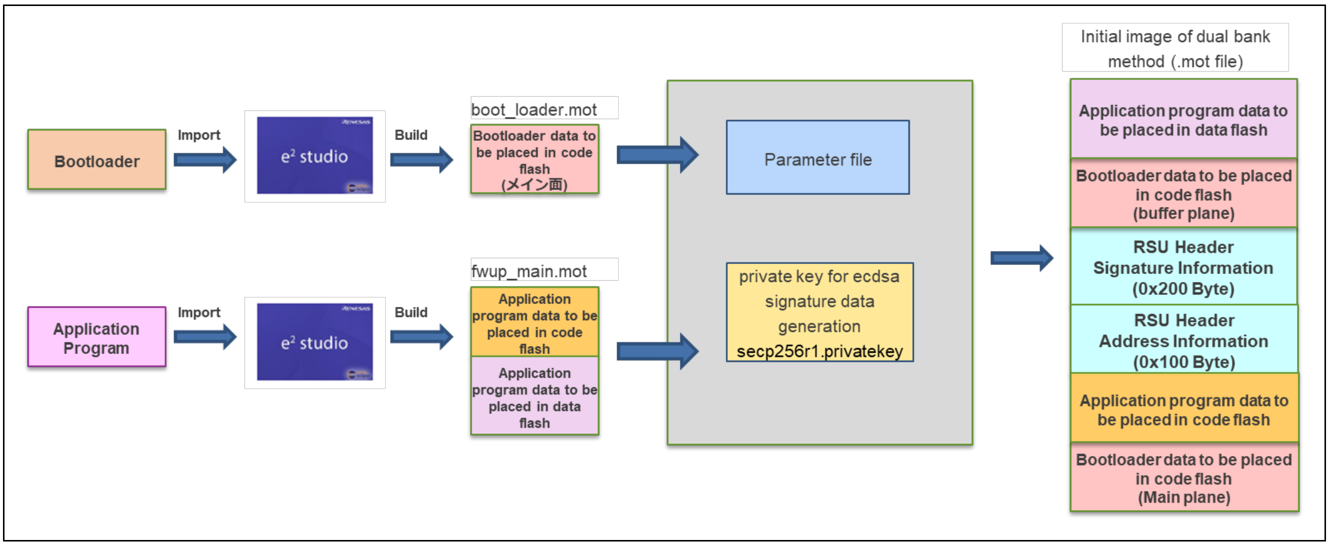

Bootloader
Import
e² studio
Build
boot_loader.mot
Bootloader data to be placed in code flash (メイン面)
Application Program
Import
e² studio
Build
fwup_main.mot
Application program data to be placed in code flash
Application program data to be placed in data flash
Parameter file
private key for ecdsa signature data generation
secp256r1.privatekey
Initial image of dual bank method (.mot file)
Application program data to be placed in data flash
Bootloader data to be placed in code flash (buffer plane)
RSU Header Signature Information (0x200 Byte)
RSU Header Address Information (0x100 Byte)
Application program data to be placed in code flash
Bootloader data to be placed in code flash (Main plane)

Figure 4.5 Initial image of the dual bank method

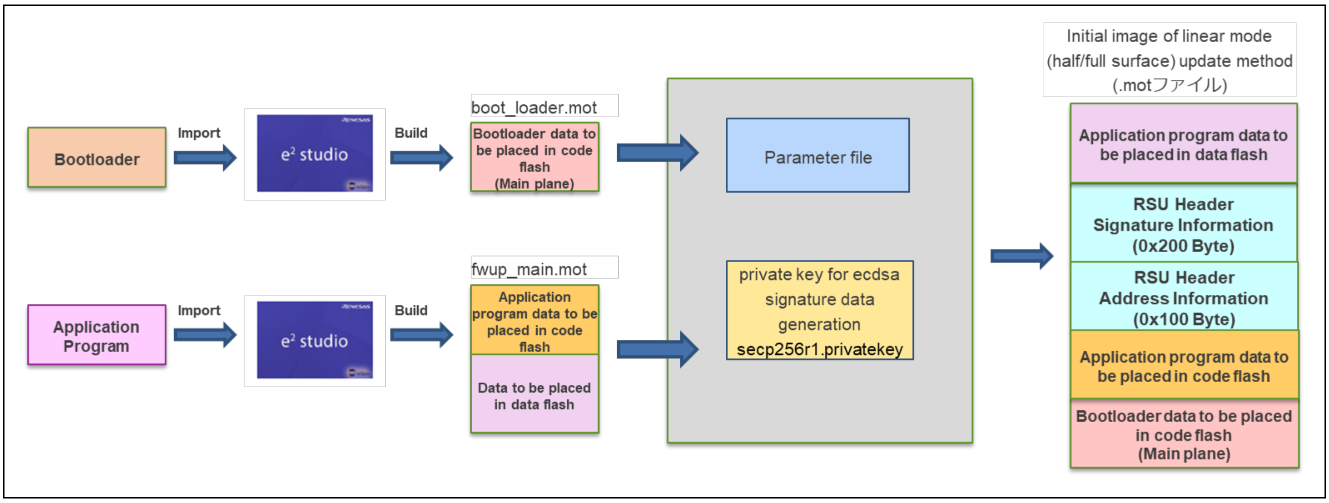

Bootloader
Import
e² studio
Build
boot_loader.mot
Bootloader data to be placed in code flash (Main plane)
Application Program
Import
e² studio
Build
fwup_main.mot
Application program data to be placed in code flash
Data to be placed in data flash
Parameter file
private key for ecdsa signature data generation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

Figure 4.6 Initial image of linear mode (half/full) update method

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

4.3 Parameter File

The parameter file is the information required for Renesas Image Generator to generate the initial and updated image files for the sample program, and is included in the release package as part of the Renesas Image Generator Python It is included in the release package as part of the Renesas Image Generator Python program set (see1.5). When a customer generates an initial or updated image for a demo project, there is no need to change the contents of the parameter file.

If you are using a product with a different flash size than the demo project (see 4.3.2) or if you do not want to include data from the data flash in the image (see 4.3.3), you can do so by editing the parameter file.

As an example, the contents of the parameter file for the RX65N (2MB) dual bank system are shown in 4.3.1.

4.3.1 Contents of Parameter File

The items listed in the parameter file are the same for all devices, but the settings differ for each device. Table 4.4 shows the contents of the parameter file for the RX65N (2MB) dual bank method demonstration project. Figure 4.7 shows the parameters referenced for image generation, and Figure 4.8 shows an example of parameters referenced for initial image generation for the RX65N (2MB) dual bank system.

Table 4.4 Contents of parameter file

Parameter name	Description	Example of setting contents RX65N(2MB)
device Type	Dual Mode：Generation of mot file for dual bank system Linear Mode：Linear mode (half/full surface) update method Mot file generation for	Dual Mode
Code Flash Size(Dual Mode Only)	Code Flash Size (Used to calculate the bootloader address of the buffer surface in the dual bank method)	0x00200000
Bootloader Start Address	Bootloader start address	0xFFFF0000
Bootloader End Address	Bootloader end address	0xFFFFFFFF
User Program Start Address	Starting address of the application program on the main face (In dual mode, application program area on main side)	0xFFF00000
User Program End Address	End address of the application program on the main side (in dual mode, application program area on main side)	0xFFFEFFFF
OFS Data Start Address	OFSM data start address (Set 'No Used.' for non-dual bank products)	0xFE7F5D00
OFS Data End Address	OFSM data end address (Set 'No Used.' for non-dual bank products)	0xFE7F5D7F
Data Flash Start Address	Data flush start address (Set 'No Used.' if data flush data is not to be generated)	0x00100000
Data Flash End Address	Data flash end address (Set 'No Used.' if data flash data is not to be generated)	0x00107FFF
Near Data Start Address(RL78 Only)	Near bootloader start address for RL78 (For RX, set 'No Used.)	No Used.
Near Data End Address(RL78 Only)	Near boot loader start address for RL78 (For RX, set 'No Used.')	No Used.
Flash Write Size	Flash write size (number of bytes required for one write to the flash in decimal)	128

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

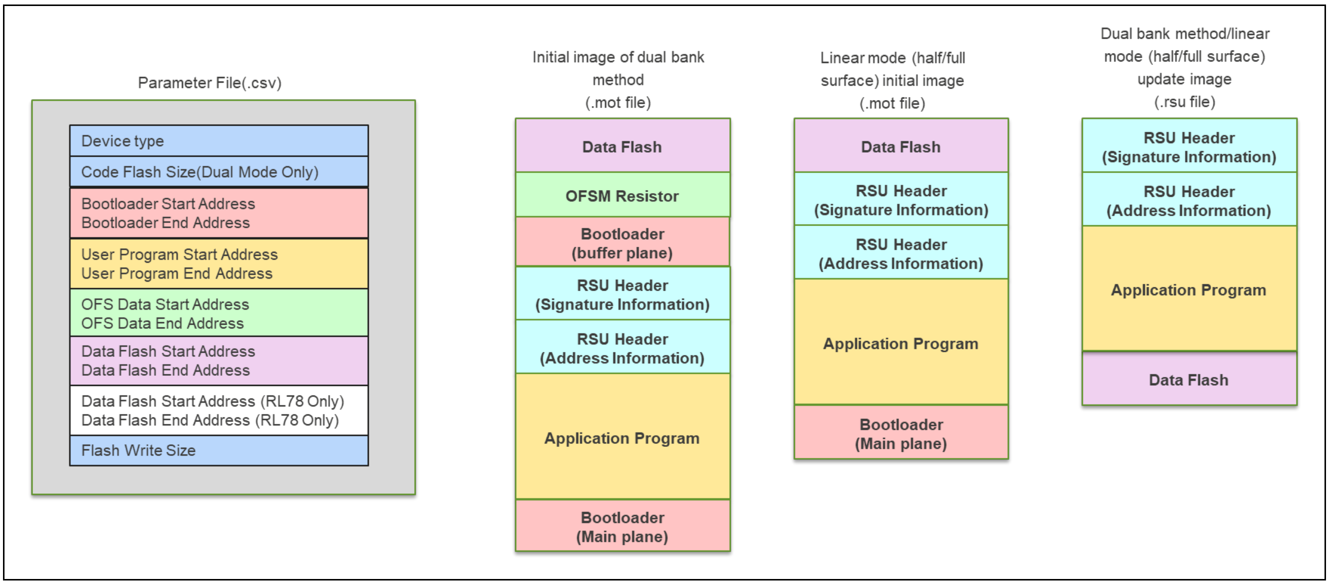
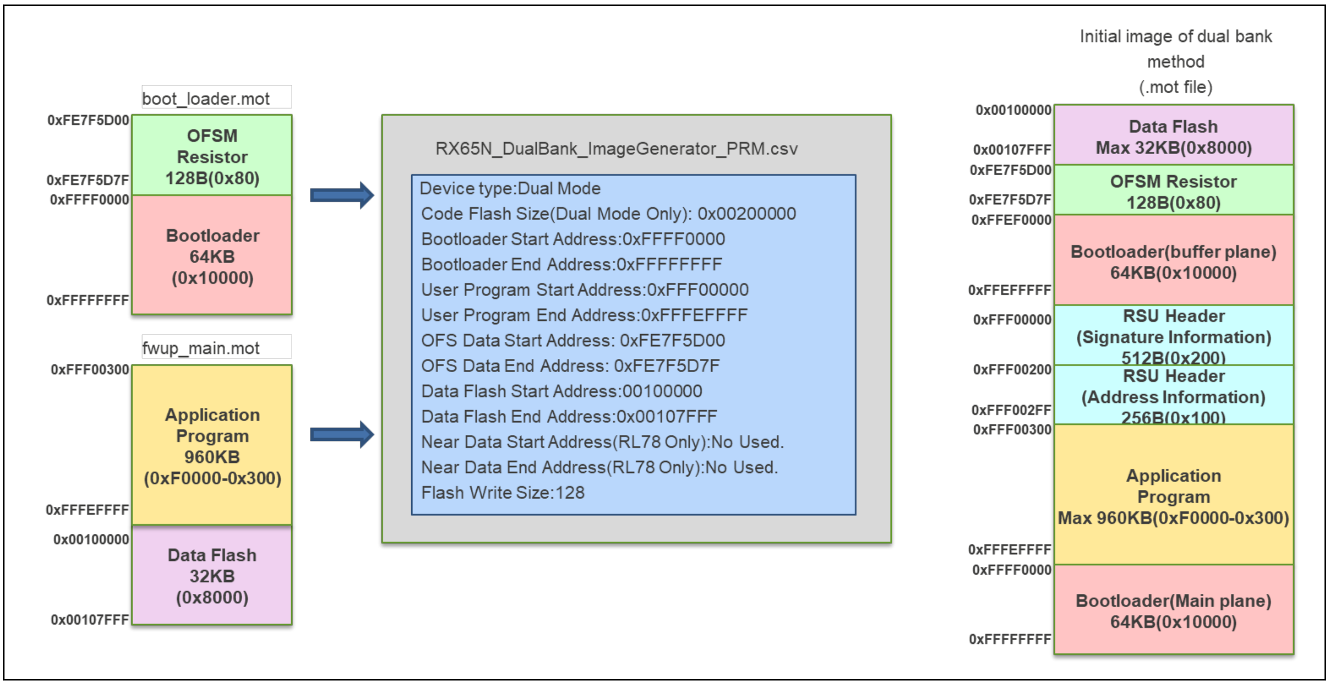

Parameter File(.csv)
Device type
Code Flash Size(Dual Mode Only)
Bootloader Start Address
Bootloader End Address
User Program Start Address
User Program End Address
OFS Data Start Address
OFS Data End Address
Data Flash Start Address
Data Flash End Address
Data Flash Start Address (RL78 Only)
Data Flash End Address (RL78 Only)
Flash Write Size
Initial image of dual bank method (.mot file)
Data Flash
OFSM Resistor
Bootloader (buffer plane)
RSU Header (Signature Information)
RSU Header (Address Information)
Application Program
Bootloader (Main plane)
Linear mode (half/full surface) initial image (.mot file)
Data Flash
RSU Header (Signature Information)
RSU Header (Address Information)
Application Program
Bootloader (Main plane)
Dual bank method/linear mode (half/full surface) update image (.rsu file)
RSU Header (Signature Information)
RSU Header (Address Information)
Application Program
Data Flash

Figure 4.7 Parameters referenced when generating image files

- Device type is used to determine whether to generate a dual-banked initial image; if Device type is Dual Mode, a bootloader (main side) and a bootloader (buffer side) are generated; if Device type is Linear Mode, only the bootloader (Main plane) is generated only in the case of Linear Mode.
- Code Flash Size (Dual Mode Only) is used to determine the address where the bootloader (buffer plane) is placed.
- Using the bootloader file (boot_loader.mot) as input data, the range from Bootloader Start Address to Bootloader End Address is generated as a code flash for the bootloader (main plane).
- With the application program file (.mot) as input data, the range from User Program Start Address to User Program End Address is generated as an application program code flash.
- Using the bootloader file (boot_loader.mot) as input data, the range from OFS Data Start Address to OFS Data End Address is generated as OFS registers.
- Using the application program file (.mot) as input data, the range from Data Flash Start Address to Data Flash End Address is generated as a data flash.
- Flash Write Size is used to set the data size of the RSU header (address information) as the minimum unit when writing to the flash.


boot_loader.mot
0xFE7F5D00
OFSM Resistor 128B(0x80)
0xFE7F5D7F
0xFFFF0000
Bootloader 64KB (0x10000)
0xFFFFFFFF
fwup_main.mot
0xFFF00300
Application Program 960KB (0xF0000-0x300)
0xFFFEFFFF
0x00100000
Data Flash 32KB (0x8000)
0x00107FFF
RX65N_DualBank_ImageGenerator_PRM.csv
Initial image of dual bank method (.mot file)
0x00100000
Data Flash Max 32KB(0x8000)
0x00107FFF
0xFE7F5D00
OFSM Resistor 128B(0x80)
0xFE7F5D7F
0xFFEF0000
Bootloader(buffer plane) 64KB(0x10000)
0xFFEFFFFF
0xFFF00000
RSU Header (Signature Information) 512B(0x200)
0xFFF00200
RSU Header (Address Information) 256B(0x100)
0xFFF002FF
0xFFF00300
Application Program Max 960KB(0xF0000-0x300)
0xFFFEFFFF
0xFFFF0000
Bootloader(Main plane) 64KB(0x10000)
0xFFFFFFFF

```
Device type:Dual Mode
Code Flash Size(Dual Mode Only): 0x00200000
Bootloader Start Address:0xFFFF0000
Bootloader End Address:0xFFFFFFFF
User Program Start Address:0xFFF00000
User Program End Address:0xFFFEFFFF
OFS Data Start Address: 0xFE7F5D00
OFS Data End Address: 0xFE7F5D7F
Data Flash Start Address:00100000
Data Flash End Address:0x00107FFF
Near Data Start Address(RL78 Only):No Used.
Near Data End Address(RL78 Only):No Used.
Flash Write Size:128
```

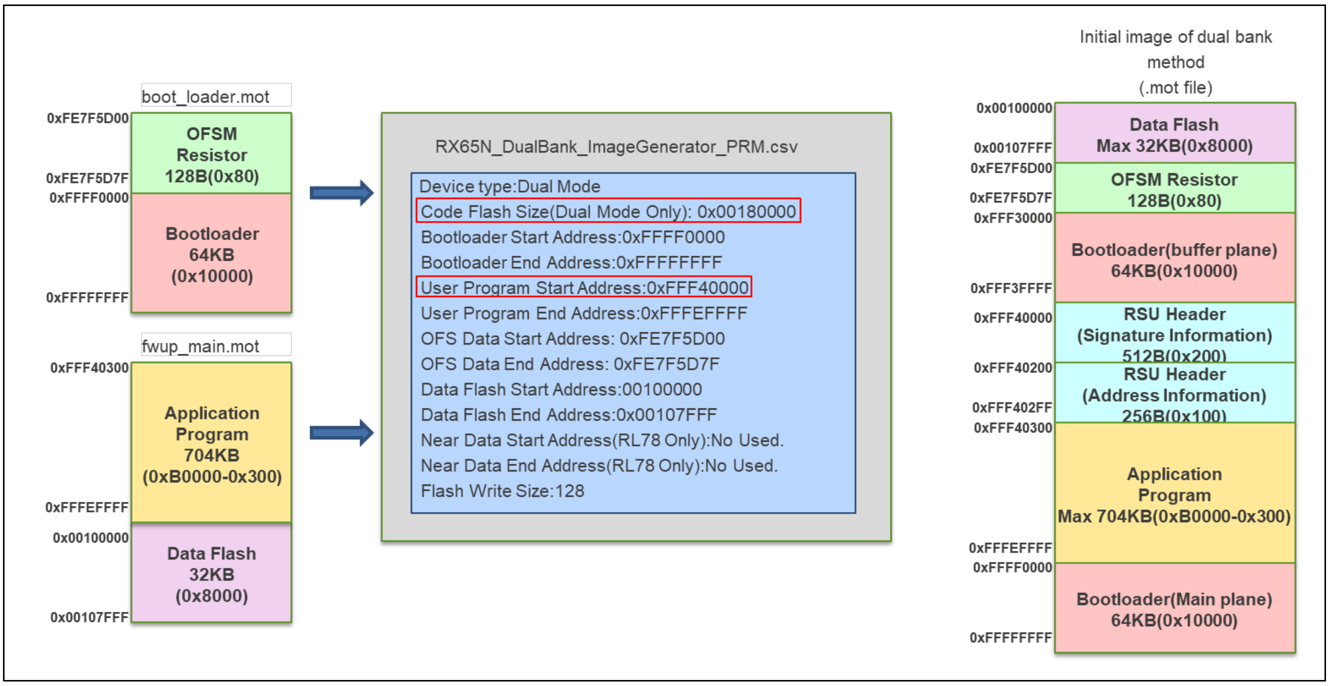
Figure 4.8 RX65N (2MB) Example of parameters referenced for initial image generation for dual bank method

4.3.2 How to generate an image with a flash size different from the demo project

If you want to perform firmware updates on products with different flash sizes that are compatible with the demo project, you can generate initial and updated images by editing the parameter file.

Figure 4.9 shows the contents of the parameter file using the RX65N dual bank method (1.5 MB) as an example. (The parameters in the red box show the differences from RX65N_DualBank_ImageGenerator_PRM.csv)

The parameter file for the RX65N dual bank method (1.5MB) is not included in the package, so customers must edit the parameters themselves.


boot_loader.mot
0xFE7F5D00
OFSM Resistor 128B(0x80)
0xFE7F5D7F
0xFFFF0000
Bootloader 64KB (0x10000)
0xFFFFFFFF
fwup_main.mot
0xFFF40300
Application Program 704KB (0xB0000-0x300)
0xFFFEFFFF
0x00100000
Data Flash 32KB (0x8000)
0x00107FFF
RX65N_DualBank_ImageGenerator_PRM.csv
Initial image of dual bank method (.mot file)
0x00100000
Data Flash Max 32KB(0x8000)
0x00107FFF
0xFE7F5D00
OFSM Resistor 128B(0x80)
0xFE7F5D7F
0xFFF30000
Bootloader(buffer plane) 64KB(0x10000)
0xFFF3FFFF
0xFFF40000
RSU Header (Signature Information) 512B(0x200)
0xFFF40200
RSU Header (Address Information) 256B(0x100)
0xFFF402FF
0xFFF40300
Application Program Max 704KB(0xB0000-0x300)
0xFFFEFFFF
0xFFFF0000
Bootloader(Main plane) 64KB(0x10000)
0xFFFFFFFF

```
Device type:Dual Mode
Code Flash Size(Dual Mode Only): 0x00180000
Bootloader Start Address:0xFFFF0000
Bootloader End Address:0xFFFFFFFF
User Program Start Address:0xFFF40000
User Program End Address:0xFFFEFFFF
OFS Data Start Address: 0xFE7F5D00
OFS Data End Address: 0xFE7F5D7F
Data Flash Start Address:00100000
Data Flash End Address:0x00107FFF
Near Data Start Address(RL78 Only):No Used.
Near Data End Address(RL78 Only):No Used.
Flash Write Size:128
```

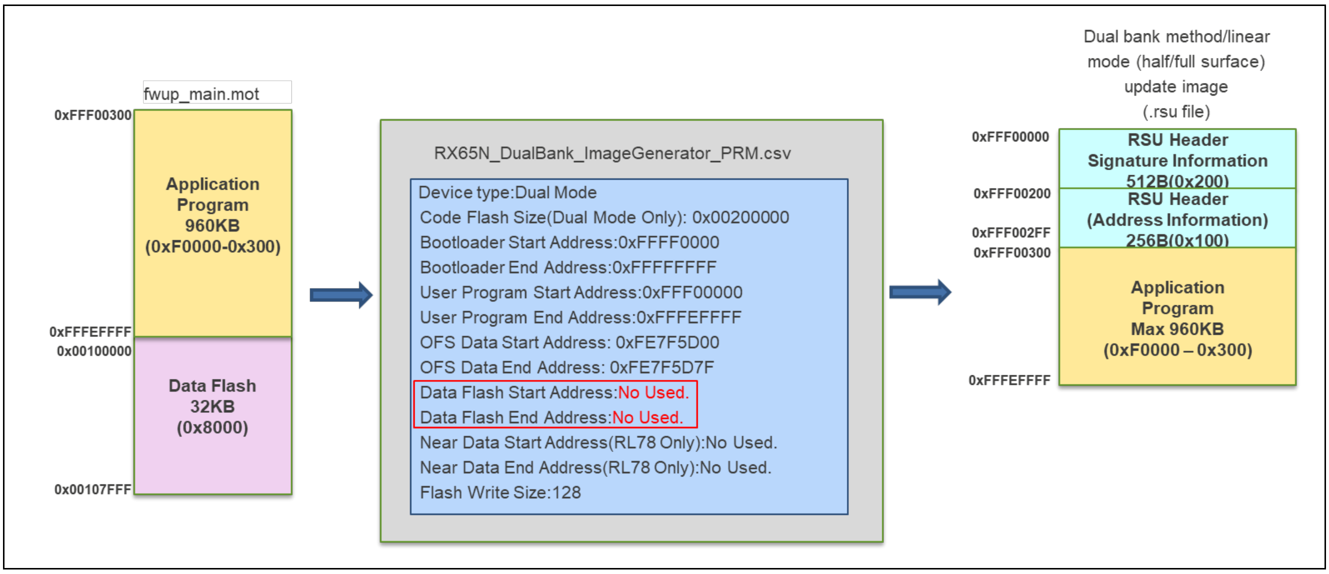
Figure 4.9 Example of parameter setting for RX65N (1.5MB) dual bank method

- Code Flash Size (Dual Mode Only) describes the code flash memory capacity of RX65N (1.5MB).
- The User Program Start Address is the address following the last address of the bootloader (buffer plane).

4.3.3 How to prevent data flash data from being included in the image

By setting the Data Flash Start Address and Data Flash End Address to "No Used." in the parameter file, the data flash data is not included in the initial image or update image. Data flash data is not included in the initial image or update image.

Figure 4.10 shows an example of parameter file settings for the RX65N (2MB) dual-bank system with the data flash not included in the update image.


fwup_main.mot
0xFFF00300
Application Program 960KB (0xF0000-0x300)
0xFFFEFFFF
0x00100000
Data Flash 32KB (0x8000)
0x00107FFF
RX65N_DualBank_ImageGenerator_PRM.csv
Dual bank method/linear mode (half/full surface) update image (.rsu file)
0xFFF00000
RSU Header Signature Information 512B(0x200)
0xFFF00200
RSU Header (Address Information) 256B(0x100)
0xFFF002FF
0xFFF00300
Application Program Max 960KB (0xF0000 – 0x300)
0xFFFEFFFF

```
Device type:Dual Mode
Code Flash Size(Dual Mode Only): 0x00200000
Bootloader Start Address:0xFFFF0000
Bootloader End Address:0xFFFFFFFF
User Program Start Address:0xFFF00000
User Program End Address:0xFFFEFFFF
OFS Data Start Address: 0xFE7F5D00
OFS Data End Address: 0xFE7F5D7F
Data Flash Start Address:No Used.
Data Flash End Address:No Used.
Near Data Start Address(RL78 Only):No Used.
Near Data End Address(RL78 Only):No Used.
Flash Write Size:128
```

Figure 4.10 Parameter settings when data flash is not included in the update image (example for RX65N)

4.4 Image Generation Methods

Renesas Image Generator (image-gen.py) is used to generate firmware update image files. When executing a command, it is necessary to specify the information necessary for generating an image. Refer to section 3.3.1 for the initial image generation method and to section 3.3.2 for the update image generation method.

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

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

Some image-gen.py command options are required and others are optional. Table 3.2 lists the required image-gen.py options, and Table 3.3 lists the optional image-gen.py options.

Table 4.5 Required Options of image-gen.py

Option	Description
-iup <file>	For the character string < file >, specify the mot file name (the full path including the file name) of the user application program.
-ip <file>	For the character string < file >, specify the name of the file (the full path including the file name) containing the parameters to be input.
-o <file>	For the character string < file >, specify the file name (the full path including the file name), excluding the extension, of the firmware update image file to be output. The file extension (.rsu or .mot) is appended based on the file format output by Image Generator.

Table 4.6 Optional Options of image-gen.py

Option	Description
-ibp <file>	For the character string < file >, specify mot file name (the full path including the file name) of the bootloader program. Specify this option when generating a mot file.
--key <file>	Specify the name of the key file to be used to sign the image using ECDSA. (This option is not needed when signing an image using SHA-256.) Store the file **secp256r1.privatekey** in the command execution folder. If the file name has been changed, specify the full path including the file name.
-vt <VerificationType>[sha256 / ecdsa]	Specifies the signature verification method. If this option is omitted, SHA-256 is used. To specify ECDSA, a key file to be used to sign the image using ECDSA must be specified using the -key option.
-ff <FileFormat>	Specifies the RSU format type. If this option is omitted, **BareMetal** is used. To generate an RTOS-compatible RSU file, specify RTOS.
-h	Specify this option to display help information for the tool.

4.4.1 Initial Image Generation Method

For the initial image, specify the options for the bootloader file name (.mot) and user program (initial firmware) file name (.mot) generated by building the projects, the parameter file name (.csv), the output file name (extension omitted), and the signature verification method (ECDSA or SHA-256), then generate an image file (.mot).

A command input example is shown below:

```
python image-gen.py -iup fwup_main.mot -ip RX26T_ImageGenerator_PRM.csv -o initial_firm -ibp
boot_loader.mot -vt ecdsa
```

fwup_main.mot: The mot file name of the user application program
RX26T_ImageGenerator_PRM.csv: The name of the file containing the parameters to be input
initial_firm: The file name of the initial image file to be output
boot_loader.mot: The mot file name of the bootloader program
ecdsa: Specifies that ECDSA is used to sign the image.

4.4.2 Update Image Generation Method

For the update image, specify the options for the user program (initial firmware) file name (.mot), the parameter file name (.csv), the output file name (extension omitted), and the signature verification method (ECDSA or SHA-256), then generate an image file (.rsu).

A command input example is shown below:

```
python image-gen.py -iup fwup_leddemo.mot -ip RX26T_ImageGenerator_PRM.csv -o fwup_leddemo -ibp
boot_loader.mot -vt ecdsa
```

fwup_leddemo.mot: The mot file name of the user application program to be applied as an update
RX26T_ImageGenerator_PRM.csv: The name of the file containing the parameters to be input
fwup_leddemo: The file name of the update image file to be output
boot_loader.mot: The mot file name of the bootloader program
ecdsa: Specifies that ECDSA is used to sign the image.

5. Demo Project

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

5.1 Demo project Structure

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 1.5 are provided.

The firmware update demo consists of the following projects.

Dual-bank method folder structure：Under □□\dualbank\△△\

Linear mode half-surface update method folder structure：Under □□\ w_buffer \△△\

Linear mode full update method folder structure: Under □□\ wo_buffer \△△\

□□: Device name

△△: Compiler (ccrx/gcc/iar)

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

5.2 Operating environment preparation

To run the firmware update demo project, you need to install the tools (see 5.2.1 to 5.2.4) on your Windows PC. Also, use a USB serial conversion board (see 5.2.5) that connects the Windows PC and the target board.

5.2.1 Installing TeraTerm

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

After installation, set the serial port communication settings as shown in Table Table 5.1

Table 5.1 Communication Specifications

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

5.2.2 Installing the Python execution environment

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

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

Install Python 3.9.0 or higher.

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

```
pip install pycryptodome
```

5.2.3 Installing the OpenSSL execution environment

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

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

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

5.2.4 Installing the Flash Writer

A flash writer is required to write the initial image.

The demo project uses Renesas Flash Programmer v3.11.01.

Renesas Flash Programmer (Programming GUI) | Renesas

5.2.5 USB serial conversion board

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

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

Use Pmod USBUART (manufactured by DIGILENT).

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

5.3 Execution environment preparation

5.3.1 Generating Keys for Signature Generation and Verification

Use OpenSSL for key generation. Refer to 5.2.3 in advance and install OpenSSL.

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

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

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

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

5.3.2 Preparing the execution environment for Renesas Image Generator

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

Renesas Image Generator requires a Python execution environment, so refer to 5.2.2 and install Python in advance.

5.4 Demo Project Execution Procedure

The execution procedure of the demo project differs depending on the firmware update method.

This chapter describes the procedure for executing the demo project using the RX65N (2MB) as an example.

The demo project execution procedure is the same for other MCU products, but only the operation check environment differs for each MCU, so check the operation check environment (6.2) for the applicable MCU product.

5.4.1 Dual Bank Method

5.4.1.1 Execution Environment

Prepare the RX65N operation check environment (6.2.1). For MCU products other than RX65N, please refer to the operation confirmation environment of the applicable MCU product.

5.4.1.2 Building The Demo Project

Follow the steps below to build three demo projects for the dual-bank method in linear mode.

1. Import the boot_loader, fwup_leddemo, and fwup_main demo projects into the integrated development environment.
2. Add the public key to be used to verify the image to the demo project.
   Paste the contents of secp256r1.publickey generated as described in 5.3.1 into code_signer_public_key.h in the boot_loader and fwup_main projects.

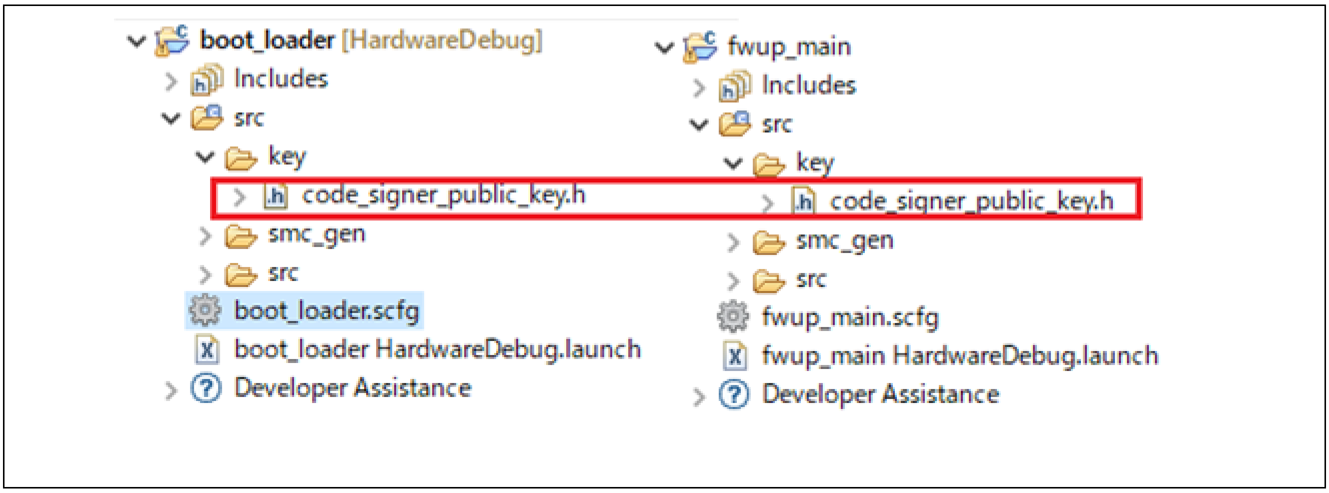


Figure 5.1 Storage Location of code_signer_public_key.h File in Demo Project

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

3. Build the demo projects.
   Build the three demo projects and confirm that the following mot files have been generated: boot_loader.mot, fwup_leddemo.mot, and fwup_main.mot.

5.4.1.3 Creating Initial and Update Images

The procedure for creating the initial and update images, using initial_firm.mot as the name of the initial image and fwup_leddemo.rsu as the name of the update image is, is described below.

1. Store the mot files created by building the demo projects in the same folder as Renesas Image Generator.

```
image-gen.py
RX65N_DualBank_ImageGenerator_PRM.csv
RX65N_Linear_Full_ImageGenerator_PRM.csv
RX65N_Linear_Half_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
```

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

```
> python image-gen.py -iup fwup_main.mot -ip
RX65N_DualBank_ImageGenerator_PRM.csv -o initial_firm -ibp
boot_loader.mot -vt ecdsa

Successfully generated the initial_firm.mot file.
```

3. Execute the following command to create the update image:

```
> python image-gen.py -iup fwup_leddemo.mot -ip
RX65N_DualBank_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa

Successfully generated the fwup_leddemo.rsu file.
```

Confirm that the initial and update image have been created in the same folder as Renesas Image Generator.

```
image-gen.py
RX65N_DualBank_ImageGenerator_PRM.csv
RX65N_Linear_Full_ImageGenerator_PRM.csv
RX65N_ Linear_Half_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
fwup_leddemo.rsu
initial_firm.mot
```

5.4.1.4 Programming the Initial Image

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

5.4.1.5 Executing a Firmware Update

Once the initial image firmware is activated, it waits for the transfer of the update image via the terminal emulator. The received update image is programmed to the flash memory, and after the transfer completes, the signature of the update image is verified and the firmware is activated.

Follow the steps below to execute a firmware update.

1. Refer to 6.2.1, Execution Environment, and connect the devices.

2. Launch the terminal emulator software on the PC, select the serial COM port, and configure the connection settings.

3. Power on the board. The following message is output:

```
==== RX65N : BootLoader [dual bank] ====
verify install area 0 [sig-sha256-ecdsa]...OK
execute new image ...

==== RX65N : Update from User [dual bank] ver 1.0.0 ====
send user program (*.rsu) via UART.
```

4. Send the updated image via the terminal emulator.
   Send file… > check **Binary** > fwup_leddemo.rsu
   The following message is output during the transfer of the update image, and a software reset is applied after installation and signature verification complete.

```
W 0xFFE00000, 256 ... OK
W 0xFFE00100, 256 ... OK
...
W 0xFFE03B00, 128 ... OK
W 0xFFEEFF80, 128 ... OK
```

5. After installing the updated firmware and verifying the signature, it jumps to the updated firmware and executes the program after processing such as bank swap.

```
verify install area 1 [sig-sha256-ecdsa]...OK
bank swap ...
software reset...
```

6. When the bootloader completes signature verification, the update image firmware is activated. When the process completes successfully, the following message is output and the LED flashes.

```
==== RX65N : BootLoader [dual bank] ====
verify install area 0 [sig-sha256-ecdsa]...OK
execute image ...

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

5.4.2 Operation of Linear Mode Partial Update Method

5.4.2.1 Execution Environment

Prepare the RX65N operation check environment (6.2.1). For MCU products other than RX65N, please refer to the operation confirmation environment of the applicable MCU product.

5.4.2.2 Building The Demo Project

Follow the steps below to build three demo projects for the partial update method in linear mode.

1. Import the boot_loader, fwup_leddemo, and fwup_main demo projects into the integrated development environment.
2. Add the public key to be used to verify the image to the demo project.
   Paste the contents of secp256r1.publickey generated as described in 5.3.1 into code_signer_public_key.h in the boot_loader and fwup_main projects.

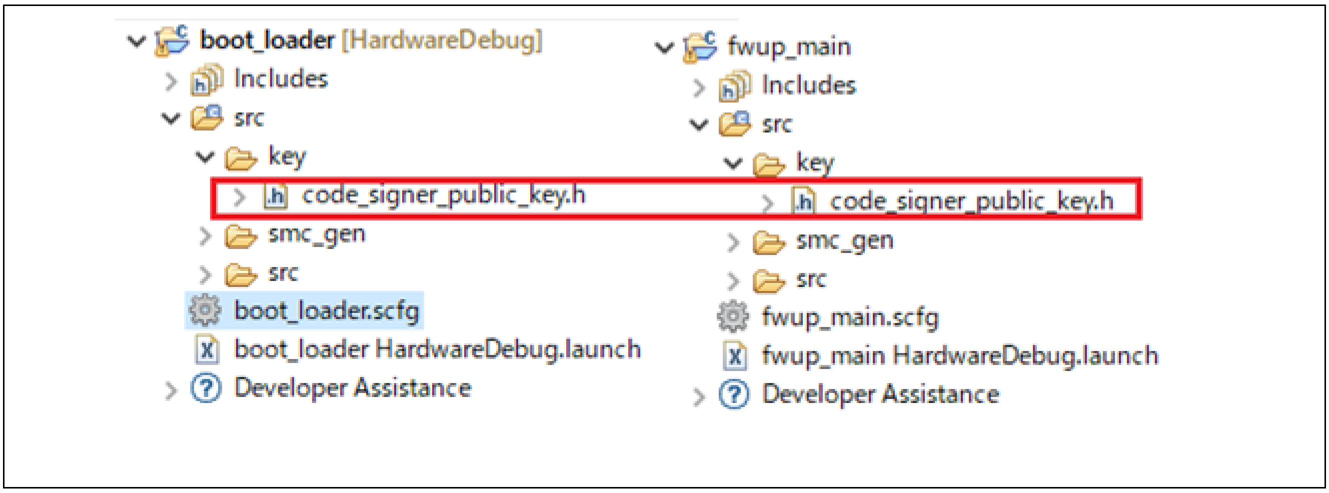


Figure 5.2 Storage Location of code_signer_public_key.h File in Demo Project

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

3. Build the demo projects.
   Build the three demo projects and confirm that the following mot files have been generated: boot_loader.mot, fwup_leddemo.mot, and fwup_main.mot.

5.4.2.3 Creating Initial and Update Images

The procedure for creating the initial and update images, using initial_firm.mot as the name of the initial image and fwup_leddemo.rsu as the name of the update image is, is described below.

1. Store the mot files created by building the demo projects in the same folder as Renesas Image Generator.

```
image-gen.py
RX65N_DualBank_ImageGenerator_PRM.csv
RX65N_Linear_Full_ImageGenerator_PRM.csv
RX65N_Linear_Half_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
```

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

```
> python image-gen.py -iup fwup_main.mot -ip
RX65N_Linear_Half_ImageGenerator_PRM.csv -o initial_firm -ibp
boot_loader.mot -vt ecdsa

Successfully generated the initial_firm.mot file.
```

3. Execute the following command to create the update image:

```
> python image-gen.py -iup fwup_leddemo.mot -ip
RX65N_Linear_Half_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa

Successfully generated the fwup_leddemo.rsu file.
```

Confirm that the initial and update image have been created in the same folder as Renesas Image Generator.

```
image-gen.py
RX65N_DualBank_ImageGenerator_PRM.csv
RX65N_Linear_Full_ImageGenerator_PRM.csv
RX65N_ Linear_Half_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
fwup_leddemo.rsu
initial_firm.mot
```

5.4.2.4 Programming the Initial Image

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

5.4.2.5 Executing a Firmware Update

Once the initial image firmware is activated, it waits for the transfer of the update image via the terminal emulator. The received update image is programmed to the flash memory, and after the transfer completes, the signature of the update image is verified and the firmware is activated.

Follow the steps below to execute a firmware update.

1. Refer to 6.2.1, Execution Environment, and connect the devices.

2. Launch the terminal emulator software on the PC, select the serial COM port, and configure the connection settings.

3. Power on the board. The following message is output:

```
==== RX65N : BootLoader [with buffer] ====
verify install area 0 [sig-sha256-ecdsa]...OK
execute image ...

==== RX65N : Update from User [with buffer]  ver 1.0.0 ====
send image(*.rsu) via UART.
```

4. Send the updated image via the terminal emulator.
   Send file… > check **Binary** > fwup_leddemo.rsu
   The following message is output during the transfer of the update image, and a software reset is applied after installation and signature verification complete.

```
W 0xFFF00000, 256 ... OK
W 0xFFF00100, 256 ... OK
...
W 0xFFF03B00, 128 ... OK
W 0xFFFEFF80, 128 ... OK
verify install area 1 [sig-sha256-ecdsa]...OK
software reset...
```

5. Activation processing is performed by the bootloader and a second software reset is applied.

```
==== RX65N : BootLoader [with buffer] ====
verify install area 1 [sig-sha256-ecdsa]...OK
activating image ... OK
software reset...
```

6. When the bootloader completes signature verification, the update image firmware is activated. When the process completes successfully, the following message is output and the LED flashes.

```
==== RX65N : BootLoader [with buffer] ====
verify install area 0 [sig-sha256-ecdsa]...OK
execute image ...

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

5.4.3 Operation of Linear Mode Full Update Method

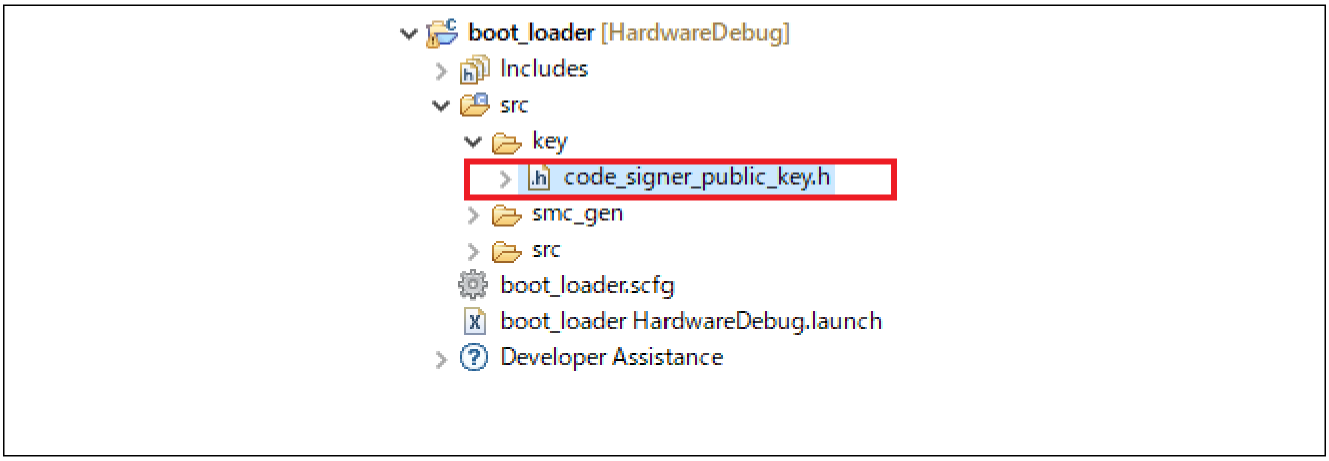
5.4.3.1 Execution Environment

Prepare the RX65N operation check environment (6.2.1). For MCU products other than RX65N, please refer to the operation confirmation environment of the applicable MCU product.

5.4.3.2 Building The Demo Project

Follow the steps below to build two demo projects for the full update method in linear mode.

1. Import the boot_loader, fwup_leddemo, and fwup_main demo projects into the integrated development environment.
2. Add the public key to be used to verify the image to the demo project.
   Paste the contents of secp256r1.publickey generated as described in 5.3.1 into code_signer_public_key.h in the boot_loader and fwup_main projects.



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

Figure 5.3 Storage Location of code_signer_public_key.h File in Demo Project

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

3. Build the demo projects.
   Build the first project (boot_loader) to generate boot_loader.mot.
   Build the second project (fwup_leddemo) to generate fwup_leddemo.mot.
   Rename fwup_leddemo.mot to fwup_leddemo_011.mot.
   Change the version of the second project (fwup_leddemo) as follows, then build it to generate fwup_leddemo.mot.

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

   Rename fwup_leddemo.mot to fwup_leddemo_012.mot.

5.4.3.3 Creating Initial and Update Images

The procedure for creating the initial and update images, using initial_firm.mot as the name of the initial image and fwup_leddemo.rsu as the name of the update image is, is described below.

1. Store the mot files created by building the demo projects in the same folder as Renesas Image Generator.

```
image-gen.py
RX65N_DualBank_ImageGenerator_PRM.csv
RX65N_Linear_Full_ImageGenerator_PRM.csv
RX65N_Linear_Half_ImageGenerator_PRM.csv
boot_loader.mot
fwup_leddemo_011.mot
fwup_leddemo_012.mot
secp256r1.privatekey
```

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

```
> python image-gen.py -iup fwup_leddemo_011.mot -ip
RX65N_Linear_Full_ImageGenerator_PRM.csv -o initial_firm -ibp
boot_loader.mot -vt ecdsa

Successfully generated the initial_firm.mot file.
```

3. Execute the following command to create the update image:

```
> python image-gen.py -iup fwup_leddemo_012.mot -ip
RX65N_Linear_Full_ImageGenerator_PRM.csv -o fwup_leddemo_012 -vt
ecdsa

Successfully generated the fwup_leddemo.rsu file.
```

Confirm that the initial and update image have been created in the same folder as Renesas Image Generator.

```
image-gen.py
RX65N_DualBank_ImageGenerator_PRM.csv
RX65N_Linear_Full_ImageGenerator_PRM.csv
RX65N_Linear_Half_ImageGenerator_PRM.csv
boot_loader.mot
fwup_main.mot
fwup_leddemo.mot
secp256r1.privatekey
fwup_leddemo_012.rsu
initial_firm.mot
```

5.4.3.4 Programming the Initial Image

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

5.4.3.5 Executing a Firmware Update

Once the initial image firmware is activated, it waits for the transfer of the update image via the terminal emulator. The received update image is programmed to the flash memory, and after the transfer completes, the signature of the update image is verified and the firmware is activated.

Follow the steps below to execute a firmware update.

1. Refer to 6.2.1, Execution Environment, and connect the devices.

2. Launch the terminal emulator software on the PC, select the serial COM port, and configure the connection settings.

3. Power on the board. The following message is output:

```
==== RX65N : BootLoader [without buffer] ====
verify install area 0 [sig-sha256-ecdsa]...OK
execute image ...

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

4. Press RESET_SW while holding down USER_SW.

```
==== RX65N : Image updater [without buffer] ====
send image(*.rsu) via UART.
```

5. Send the updated image via the terminal emulator.
   Send file… > check **Binary** > fwup_leddemo_012.rsu

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

```
W 0xFFE00000, 128 ... OK
W 0xFFE00080, 128 ... OK
...
W 0xFFE03B00, 128 ... OK
W 0xFFFEFF80, 128 ... OK
verify install area 0 [sig-sha256-ecdsa]...OK
software reset...
```

6. When the bootloader completes signature verification, the update image firmware is activated. When the process completes successfully, the following message is output and the LED flashes.

```
==== RX65N : BootLoader [without buffer] ====
verify install area 0 [sig-sha256-ecdsa]...OK
execute image ...

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

6. Appendices

6.1 Confirmed Operation Environments

This section describes environments on which the operation of the FIT module has been confirmed.

Table 6.1 Confirmed Operation Environment (CC-RX)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio 2023-01
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.2.00
Board used	Renesas Starter Kit+ for RX65N（product No.：RTK50565N2SxxxxxBE） Renesas Flexible Motor Control Kit for RX26T (product No.: RTK0EMXE70S00020BJ) Renesas Starter Kit+ for RX24T（product No.：RTK500524TS00000BE）

Table 6.2 Confirmed Operation Environment (GCC)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio 2023-01
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.2.00
Board used	Renesas Starter Kit+ for RX65N（product No.：RTK50565N2SxxxxxBE） Renesas Flexible Motor Control Kit for RX26T (product No.: RTK0EMXE70S00020BJ) Renesas Starter Kit+ for RX24T（product No.：RTK500524TS00000BE）

Table 6.3 Confirmed Operation Environment (IAR)

Item	Description
Integrated development environment	IAR Embedded Workbench for Renesas RX 4.20.1
C compiler	IAR C/C++ Compiler for Renesas RX 4.20.1 Compiler option: The default settings of the integrated development environment
Endian order	Little endian
Revision of the module	Rev.2.00
Board used	Renesas Starter Kit+ for RX65N（product No.：RTK50565N2SxxxxxBE） Renesas Flexible Motor Control Kit for RX26T (product No.: RTK0EMXE70S00020BJ) Renesas Starter Kit+ for RX24T（product No.：RTK500524TS00000BE）

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 6.4 FIT Module Versions (CC-RX)

Device	Project	r_bsp	r_byteq	r_flash_rx	r_sci_rx	r_fwup_rx
RX65N	boot_loader	7.21	2.10	4.91	4.60	2.00
	fwup_main	7.21	2.10	4.91	4.60	2.00
	fwup_leddemo	7.21	2.10	-	4.60	-
RX26T	boot_loader	7.30	2.10	5.00	4.80	2.00
	fwup_main	7.30	2.10	5.00	4.80	2.00
	fwup_leddemo	7.30	2.10	-	4.80	-
RX24T	boot_loader	7.21	2.10	4.91	4.60	2.00
	fwup_main	7.21	2.10	4.91	4.60	2.00
	fwup_leddemo	7.21	2.10	-	4.60	-

(2) GCC for Renesas RX

Table 6.5 FIT Module Versions (GCC)

Device	Project	r_bsp	r_byteq	r_flash_rx	r_sci_rx	r_fwup_rx
RX65N	boot_loader	7.21	2.10	4.91	4.60	2.00
	fwup_main	7.21	2.10	4.91	4.60	2.00
	fwup_leddemo	7.21	2.10	-	4.60	-
RX26T	boot_loader	7.30	2.10	5.00	4.80	2.00
	fwup_main	7.30	2.10	5.00	4.80	2.00
	fwup_leddemo	7.30	2.10	-	4.80	-
RX24T	boot_loader	7.21	2.10	4.91	4.60	2.00
	fwup_main	7.21	2.10	4.91	4.60	2.00
	fwup_leddemo	7.21	2.10	-	4.60	-

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

Table 6.6 FIT Module Versions (IAR)

Device	Project	r_bsp	r_byteq	r_flash_rx	r_sci_rx	r_fwup_rx
RX65N	boot_loader	7.21	2.10	4.91	4.60	2.00
	fwup_main	7.21	2.10	4.91	4.60	2.00
	fwup_leddemo	7.21	2.10	-	4.60	-
RX26T	boot_loader	7.40	2.10	5.00	4.80	2.00
	fwup_main	7.40	2.10	5.00	4.80	2.00
	fwup_leddemo	7.40	2.10	-	4.80	-
RX24T	boot_loader	7.21	2.10	4.91	4.60	2.00
	fwup_main	7.21	2.10	4.91	4.60	2.00
	fwup_leddemo	7.21	2.10	-	4.60	-

6.2 Operating Environment for Demo Project

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

6.2.1 Operation Confirmation Environment for RX65N

The execution environment and connection diagram are shown below.

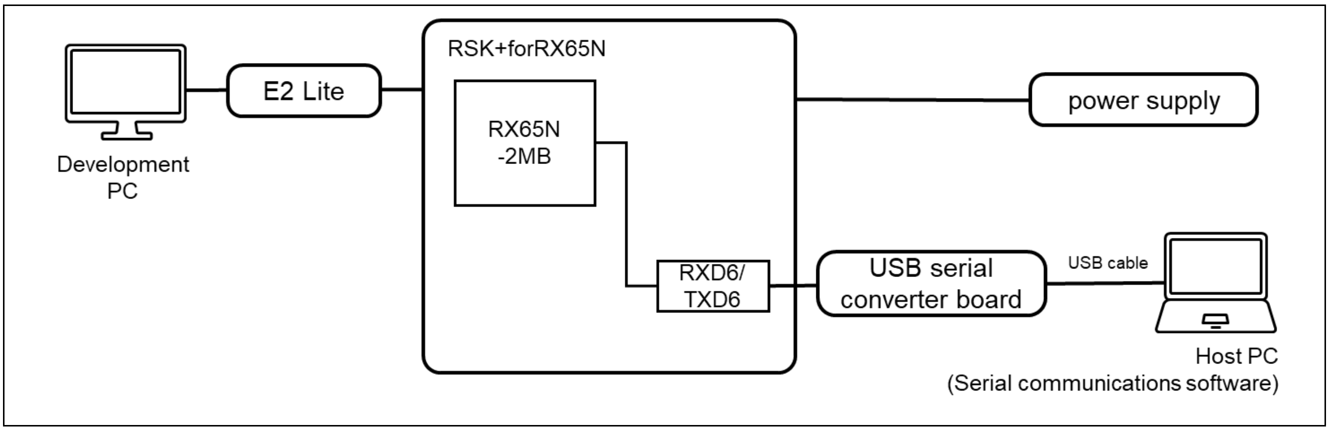


Figure 6.1 RSK-RX65N Device Connection Diagram

The pin assignment is shown in the figure below.

■ UART(Red)

PMOD1		USB-UART
2	TXD6	RX
3	RXD6	TX
4	P02(RTS)	CTS

■ USER_SW (Green)

SW1	Note
P03	LOW : USER_SW is ON

■ Reset Switch (Blue)

RES1	Note
RES#	LOW : USER_SW is ON

■ USER_LED (Yellow)

LED0	Note
P73	LED0

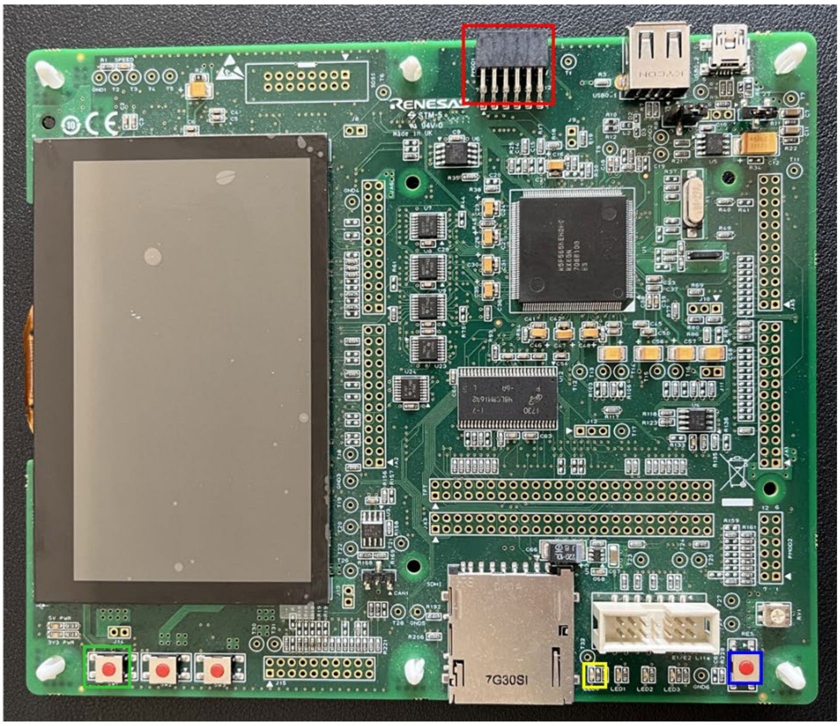
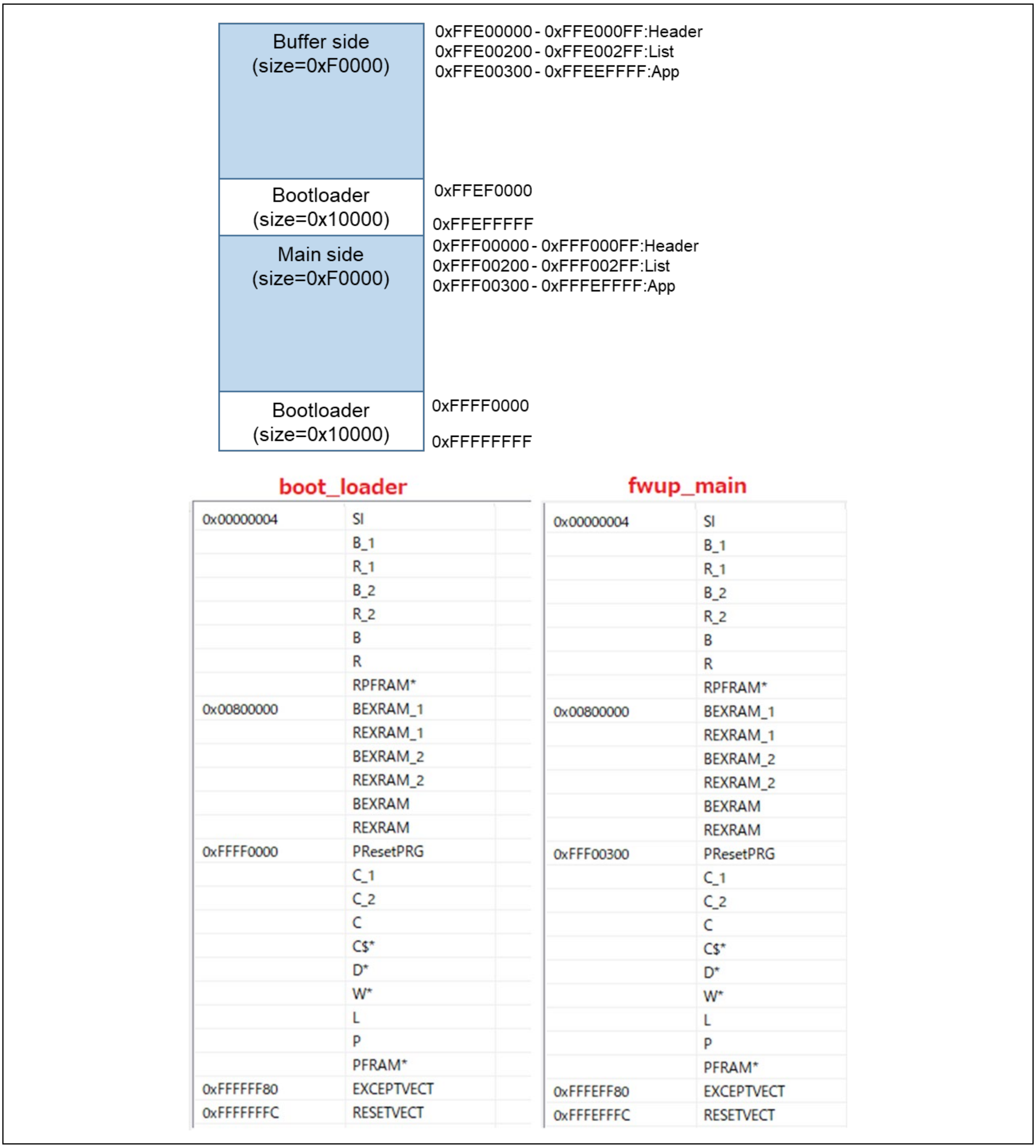


Figure 6.2 RSK-RX65N Pin Information

6.2.1.1 Memory map of dual bank method demo project

The memory map and configuration settings for the RX65N dual-bank method demo project are shown below.


Buffer side
(size=0xF0000)
0xFFE00000 - 0xFFE000FF:Header
0xFFE00200 - 0xFFE002FF:List
0xFFE00300 - 0xFFEEFFFF:App
Bootloader
(size=0x10000)
0xFFEF0000
0xFFEFFFFF
Main side
(size=0xF0000)
0xFFF00000 - 0xFFF000FF:Header
0xFFF00200 - 0xFFF002FF:List
0xFFF00300 - 0xFFFEFFFF:App
Bootloader
(size=0x10000)
0xFFFF0000
0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFFF0000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_main

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFF00300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFEFF80	EXCEPTVECT
0xFFFEFFFC	RESETVECT

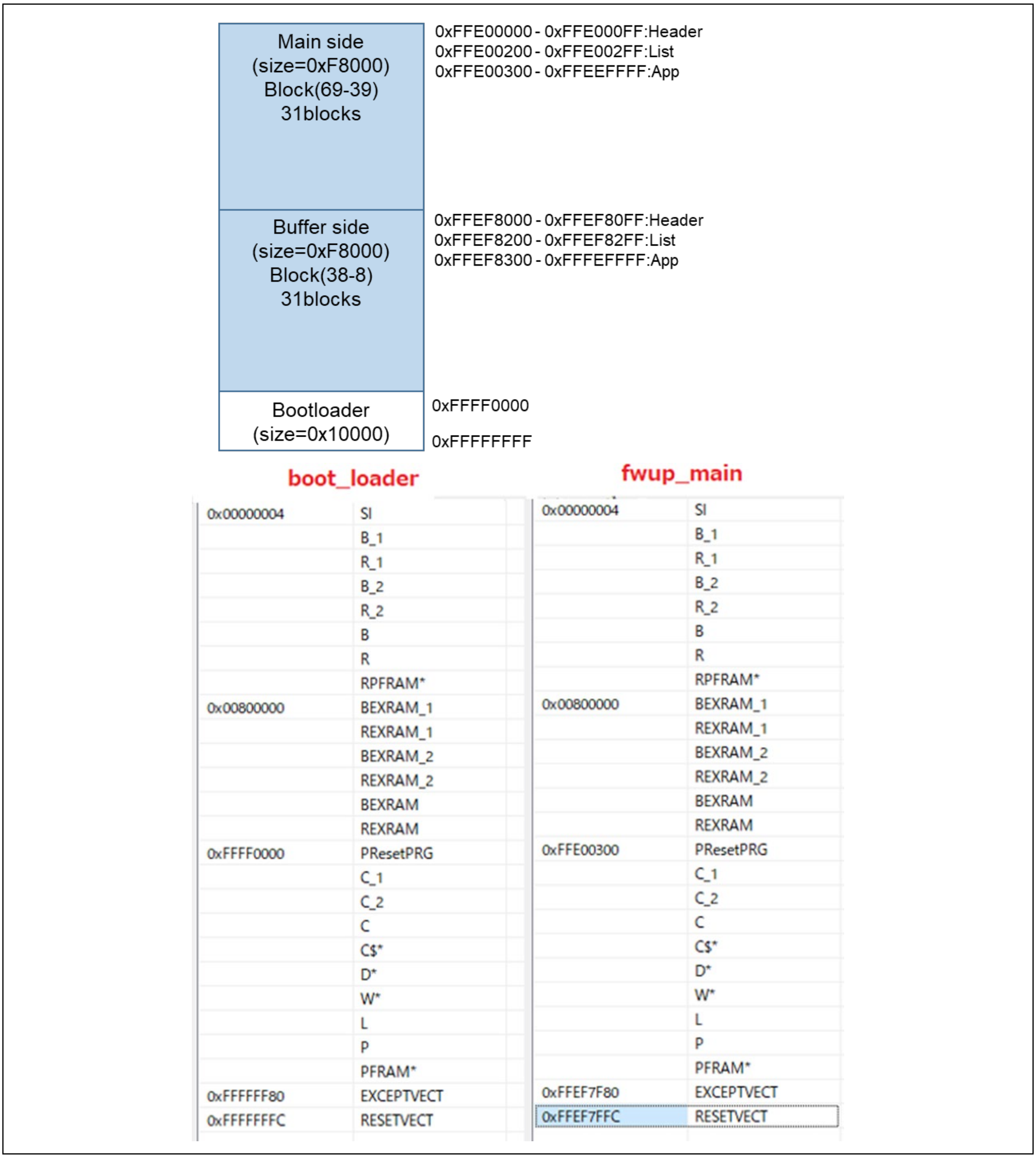
Figure 6.3 RX65N dual bank method demo project memory map

Table 6.7 RX65N dual bank method configuration settings

Configuration options in r_fwup _config.h		
parameter name	**boot_loader**	**fwup_main**
FWUP_CFG_UPDATE_MODE	0	0
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFF00000	0xFFF00000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFE00000	0xFFE00000
FWUP_CFG_AREA_SIZE	0xF0000	0xF0000
FWUP_CFG_CF_BLK_SIZE	0x8000	0x8000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096	4096
FWUP_CFG_DF_ADDR_L	0x00100000	0x00100000
FWUP_CFG_DF_BLK_SIZE	64	64
FWUP_CFG_DF_NUM_BLKS	512	512
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	0	0

6.2.1.2 Memory map of demo project for half-surface update method in linear mode

Shown below are the memory map of the RX65N linear mode half-surface update method demo project and the memory map of the configuration settings.



Main side (size=0xF8000) Block(69-39) 31blocks	0xFFE00000 - 0xFFE000FF:Header 0xFFE00200 - 0xFFE002FF:List 0xFFE00300 - 0xFFEEFFFF:App
Buffer side (size=0xF8000) Block(38-8) 31blocks	0xFFEF8000 - 0xFFEF80FF:Header 0xFFEF8200 - 0xFFEF82FF:List 0xFFEF8300 - 0xFFFEFFFF:App
Bootloader (size=0x10000)	0xFFFF0000 0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFFF0000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_main

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFE00300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFEF7F80	EXCEPTVECT
0xFFEF7FFC	RESETVECT

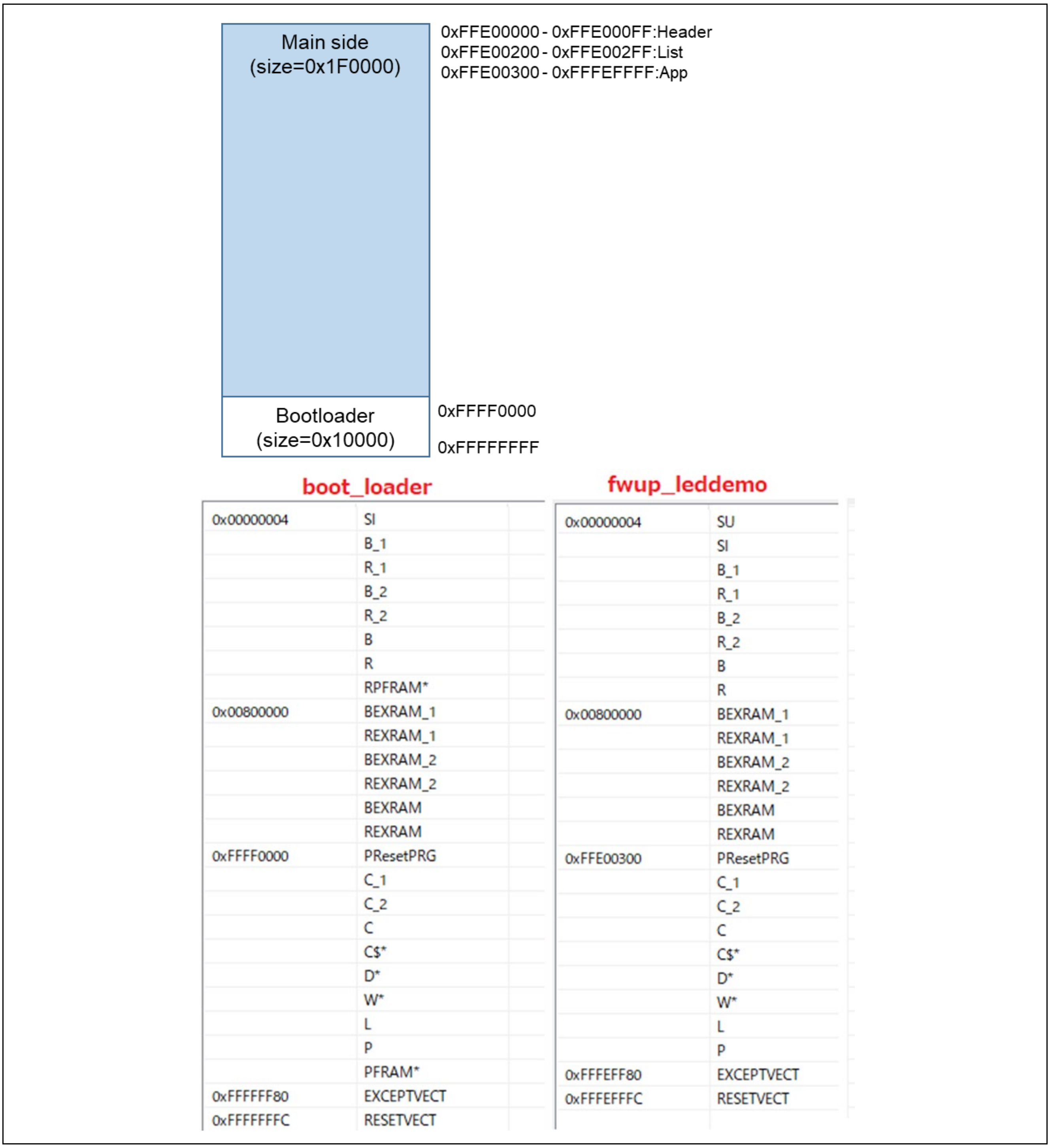
Figure 6.4 RX65N linear mode half-surface update method demo project memory map

Table 6.8 RX65N linear mode half-surface update method configuration setting

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFE00000	0xFFE00000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFEF8000	0xFFEF8000
FWUP_CFG_AREA_SIZE	0xF8000	0xF8000
FWUP_CFG_CF_BLK_SIZE	0x8000	0x8000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096	4096
FWUP_CFG_DF_ADDR_L	0x00100000	0x00100000
FWUP_CFG_DF_BLK_SIZE	64	64
FWUP_CFG_DF_NUM_BLKS	512	512
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	0	0

6.2.1.3 Memory map of demo project for full update method in linear mode

The memory map of the RX65N linear mode full update method demo project and the memory map of the configuration settings are shown below.


Main side
(size=0x1F0000)
0xFFE00000 - 0xFFE000FF:Header
0xFFE00200 - 0xFFE002FF:List
0xFFE00300 - 0xFFFEFFFF:App
Bootloader
(size=0x10000)
0xFFFF0000
0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFFF0000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_leddemo

0x00000004	SU
	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFE00300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
0xFFFEFF80	EXCEPTVECT
0xFFFEFFFC	RESETVECT

Figure 6.5 RX65N linear mode full update method demo project memory map

Table 6.9 RX65N linear mode full update method configuration setting

Configuration options in r_fwup _config.h	
parameter name	boot_loader
FWUP_CFG_UPDATE_MODE	2
FWUP_CFG_FUNCTION_MODE	0
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFE00000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFE00000
FWUP_CFG_AREA_SIZE	0x1F0000
FWUP_CFG_CF_BLK_SIZE	0x8000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096
FWUP_CFG_DF_ADDR_L	0x00100000
FWUP_CFG_DF_BLK_SIZE	64
FWUP_CFG_DF_NUM_BLKS	512
FWUP_CFG_SIGNATURE_VERIFICATION	0
FWUP_CFG_PRINTF_DISABLE	0

6.2.2 Operation Confirmation Environment for RX26T

The execution environment and connection diagram are shown below.

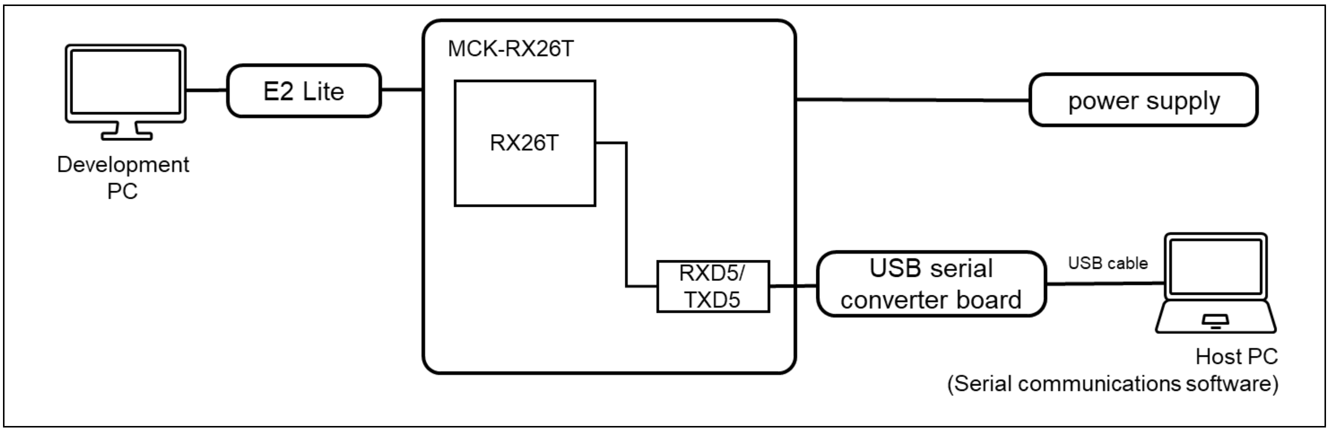
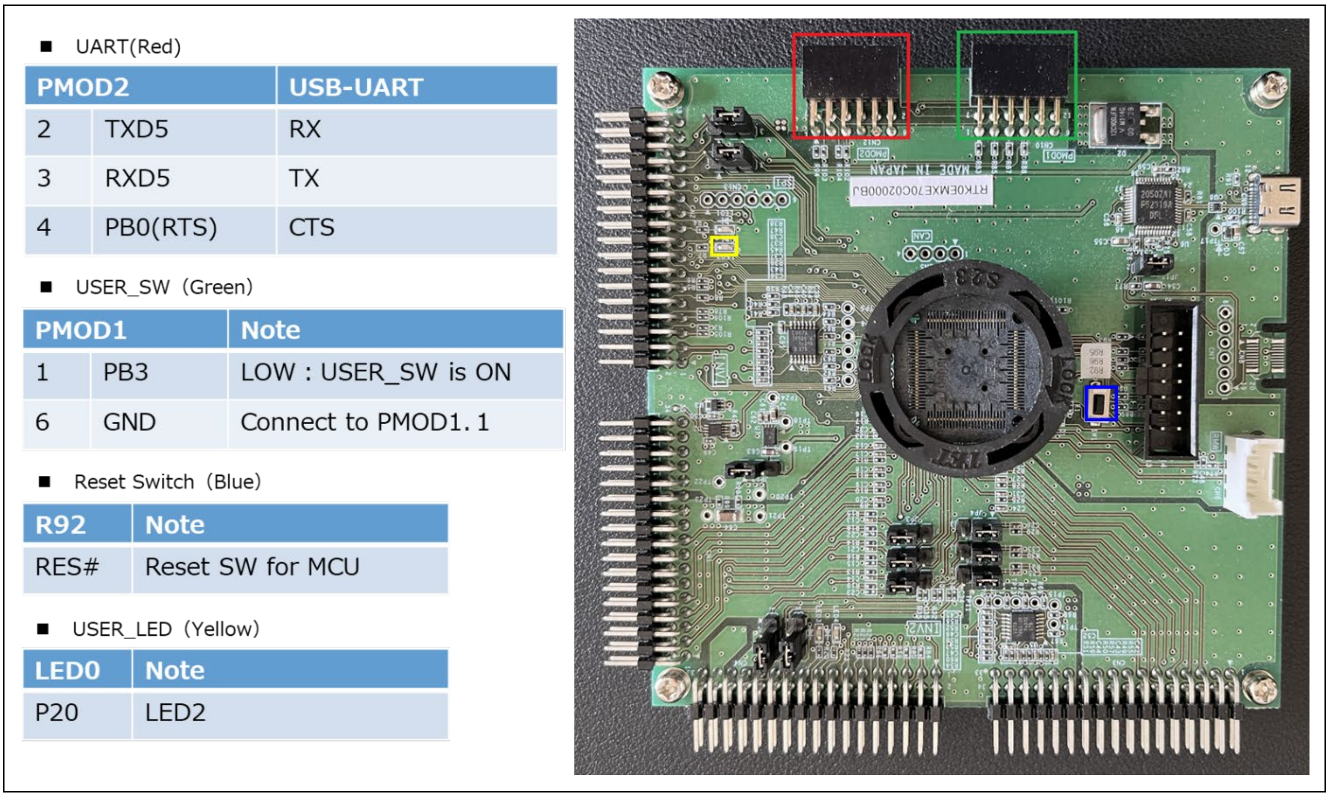

Development PC
E2 Lite
MCK-RX26T
RX26T
RXD5/ TXD5
power supply
USB serial converter board
USB cable
Host PC (Serial communications software)

Figure 6.6 MCK-RX26T Device Connection Diagram

The pin assignment is shown in the figure below.



■ UART(Red)

PMOD2		USB-UART
2	TXD5	RX
3	RXD5	TX
4	PB0(RTS)	CTS

■ USER_SW（Green)

PMOD1		Note
1	PB3	LOW : USER_SW is ON
6	GND	Connect to PMOD1. 1

■ Reset Switch（Blue)

R92	Note
RES#	Reset SW for MCU

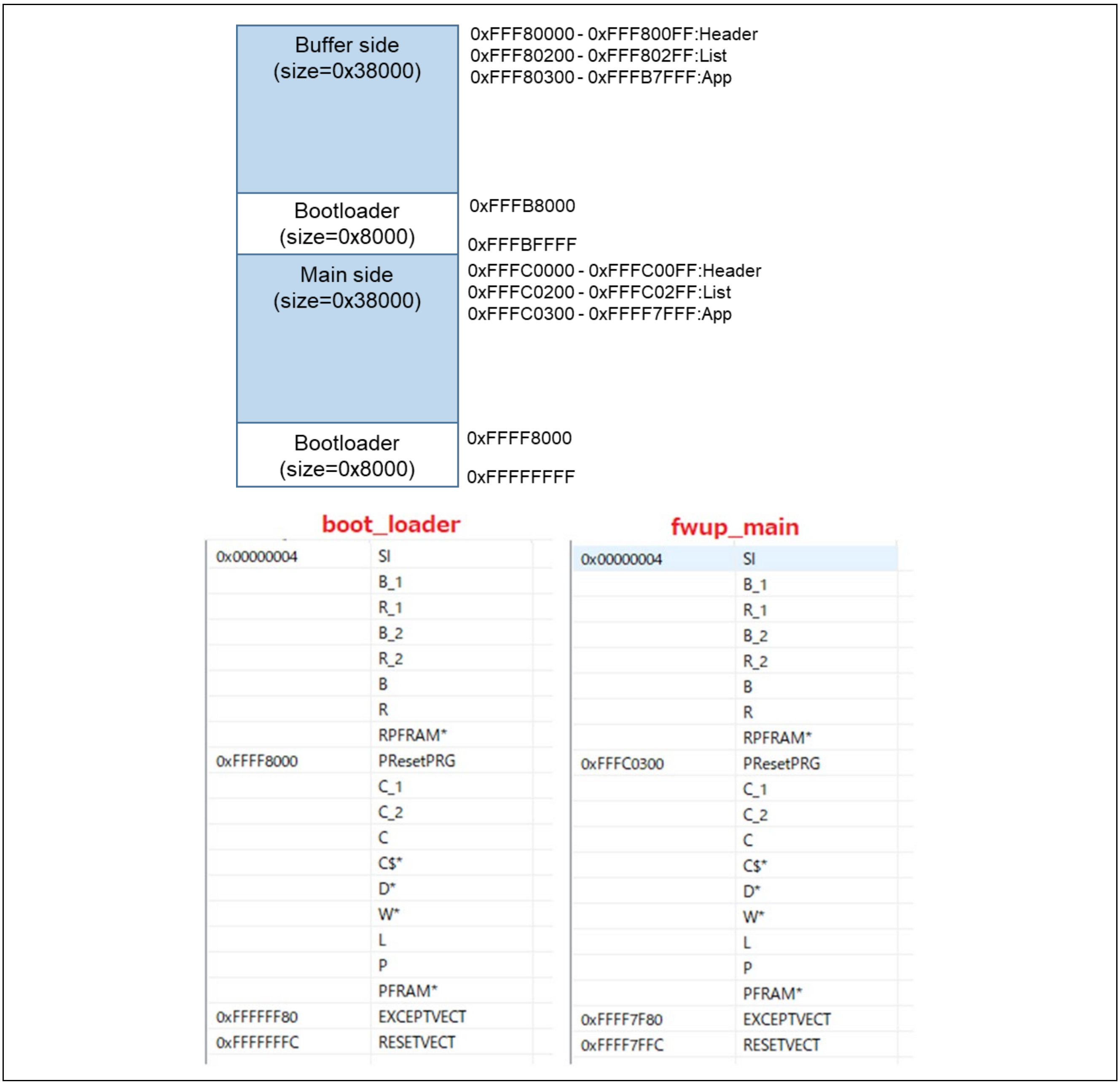
■ USER_LED（Yellow)

LED0	Note
P20	LED2

Figure 6.7 MCK-RX26T Pin Information

6.2.2.1 Memory map of dual bank method demo project

The memory map and configuration settings for the RX26T dual-bank method demo project are shown below.


Buffer side
(size=0x38000)
0xFFF80000 - 0xFFF800FF:Header
0xFFF80200 - 0xFFF802FF:List
0xFFF80300 - 0xFFFB7FFF:App
Bootloader
(size=0x8000)
0xFFFB8000
0xFFFBFFFF
Main side
(size=0x38000)
0xFFFC0000 - 0xFFFC00FF:Header
0xFFFC0200 - 0xFFFC02FF:List
0xFFFC0300 - 0xFFFF7FFF:App
Bootloader
(size=0x8000)
0xFFFF8000
0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFF8000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_main

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFC0300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFF7F80	EXCEPTVECT
0xFFFF7FFC	RESETVECT

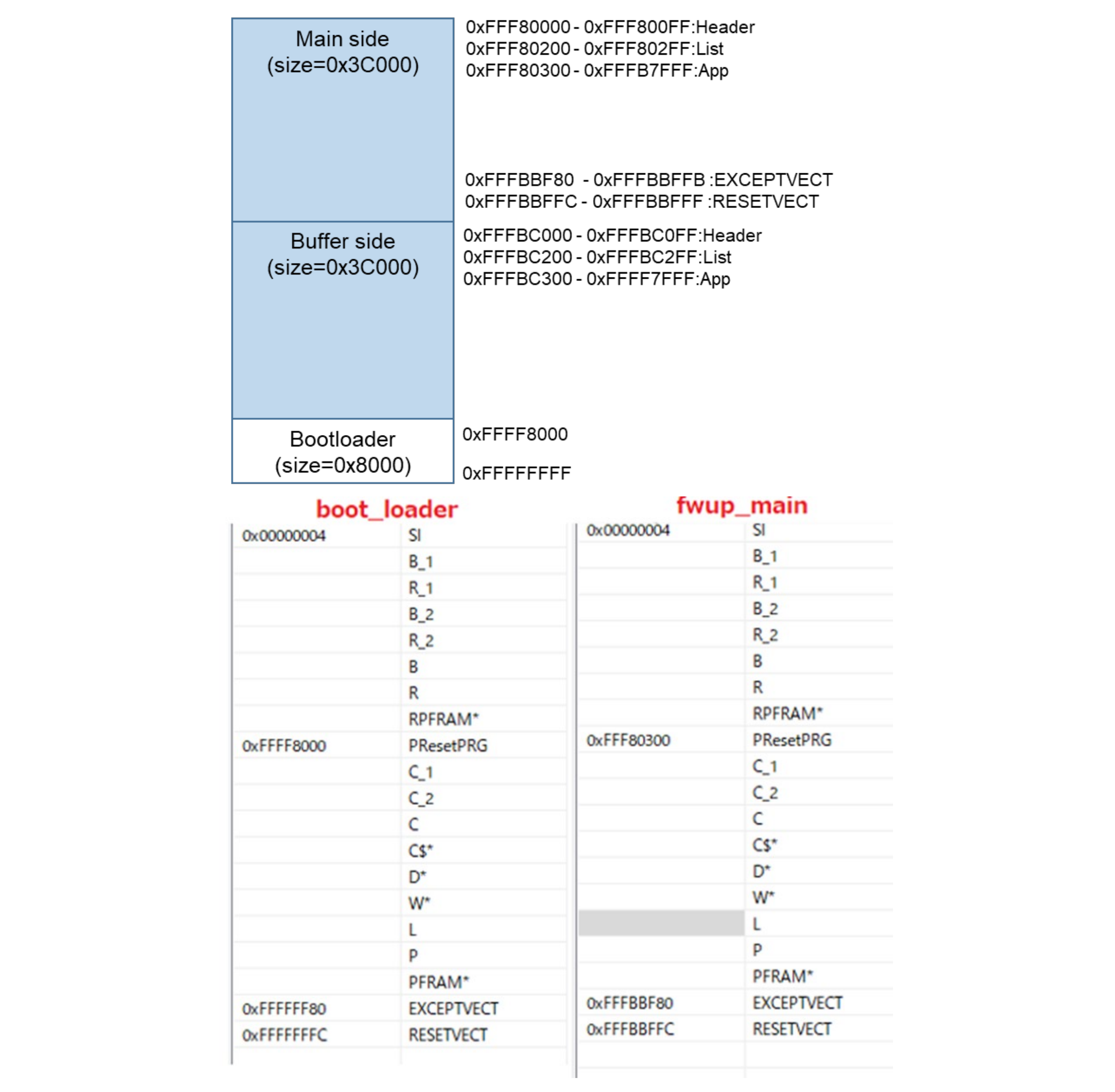
Figure 6.8 RX26T dual bank method demo project memory map

Table 6.10 RX26T dual bank method configuration settings

Configuration options in r_fwup _config.h		
parameter name	boot_loader	fwup_main
FWUP_CFG_UPDATE_MODE	0	0
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFFC0000	0xFFFC0000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFF80000	0xFFF80000
FWUP_CFG_AREA_SIZE	0x38000	0x38000
FWUP_CFG_CF_BLK_SIZE	0x4000	0x4000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096	4096
FWUP_CFG_DF_ADDR_L	0x00100000	0x00100000
FWUP_CFG_DF_BLK_SIZE	64	64
FWUP_CFG_DF_NUM_BLKS	256	256
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	0	0

6.2.2.2 Memory map of demo project for half-surface update method in linear mode

Shown below are the memory map of the RX26T linear mode half-surface update method demo project and the memory map of the configuration settings.


Main side
(size=0x3C000)
0xFFF80000 - 0xFFF800FF:Header
0xFFF80200 - 0xFFF802FF:List
0xFFF80300 - 0xFFFB7FFF:App
0xFFFBBF80 - 0xFFFBBFFB :EXCEPTVECT
0xFFFBBFFC - 0xFFFBBFFF :RESETVECT
Buffer side
(size=0x3C000)
0xFFFBC000 - 0xFFFBC0FF:Header
0xFFFBC200 - 0xFFFBC2FF:List
0xFFFBC300 - 0xFFFF7FFF:App
Bootloader
(size=0x8000)
0xFFFF8000
0xFFFFFFFF
boot_loader
fwup_main

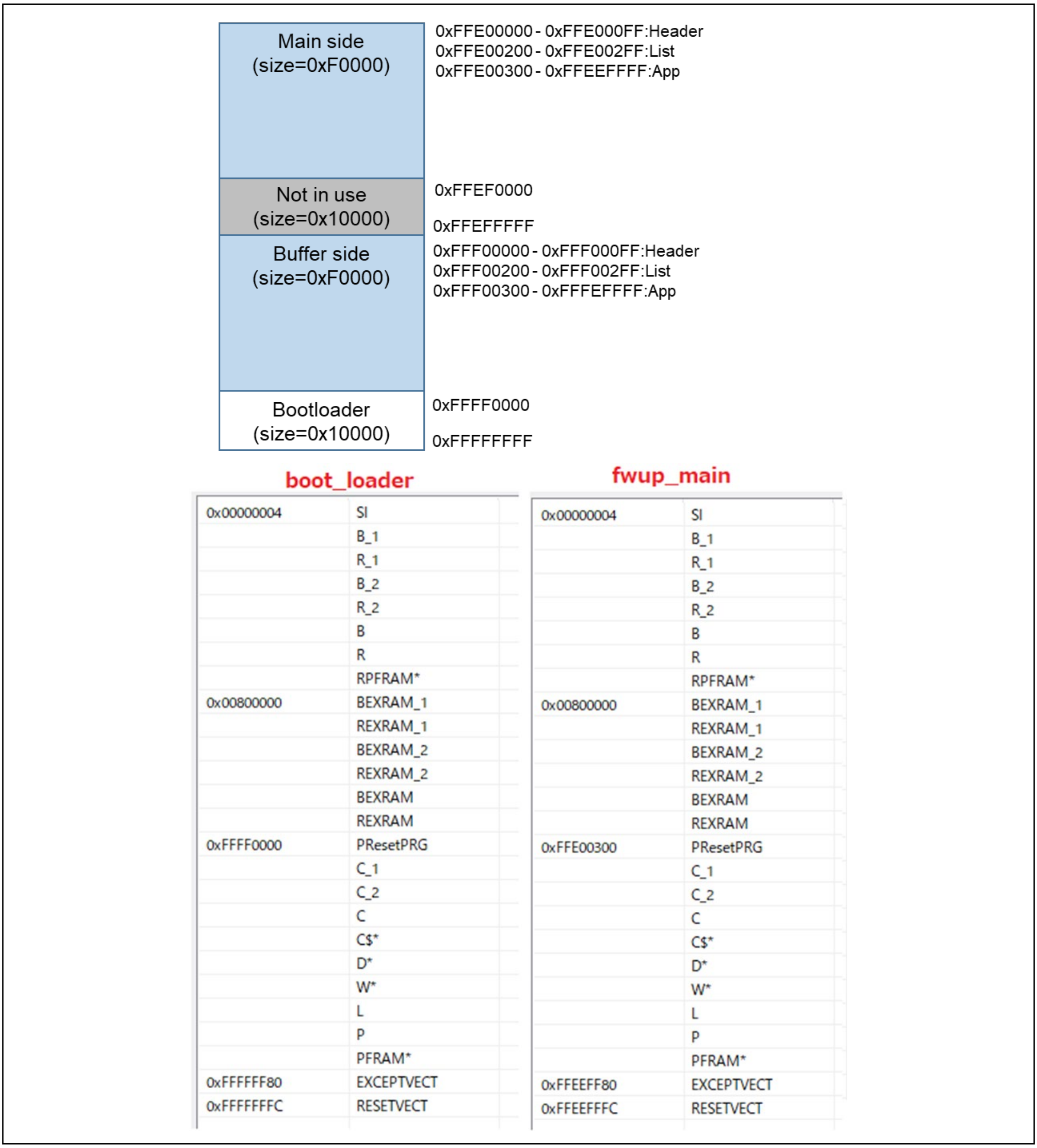
0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFF8000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFF80300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFBBF80	EXCEPTVECT
0xFFFBBFFC	RESETVECT

Figure 6.9 RX26T linear mode half-surface update method demo project memory map

Table 6.11 RX26T linear mode half-surface update method configuration setting

Configuration options in r_fwup _config.h		
parameter name	**boot_loader**	**fwup_main**
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFF80000	0xFFF80000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFFBC000	0xFFFBC000
FWUP_CFG_AREA_SIZE	0x3C000	0x3C000
FWUP_CFG_CF_BLK_SIZE	0x4000	0x4000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096	4096
FWUP_CFG_DF_ADDR_L	0x00100000	0x00100000
FWUP_CFG_DF_BLK_SIZE	64	64
FWUP_CFG_DF_NUM_BLKS	256	256
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	0	0


Main side
(size=0xF0000)
0xFFE00000 - 0xFFE000FF:Header
0xFFE00200 - 0xFFE002FF:List
0xFFE00300 - 0xFFEEFFFF:App
Not in use
(size=0x10000)
0xFFEF0000
0xFFEFFFFF
Buffer side
(size=0xF0000)
0xFFF00000 - 0xFFF000FF:Header
0xFFF00200 - 0xFFF002FF:List
0xFFF00300 - 0xFFFEFFFF:App
Bootloader
(size=0x10000)
0xFFFF0000
0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFFF0000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_main

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0x00800000	BEXRAM_1
	REXRAM_1
	BEXRAM_2
	REXRAM_2
	BEXRAM
	REXRAM
0xFFE00300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFEEFF80	EXCEPTVECT
0xFFEEFFFC	RESETVECT

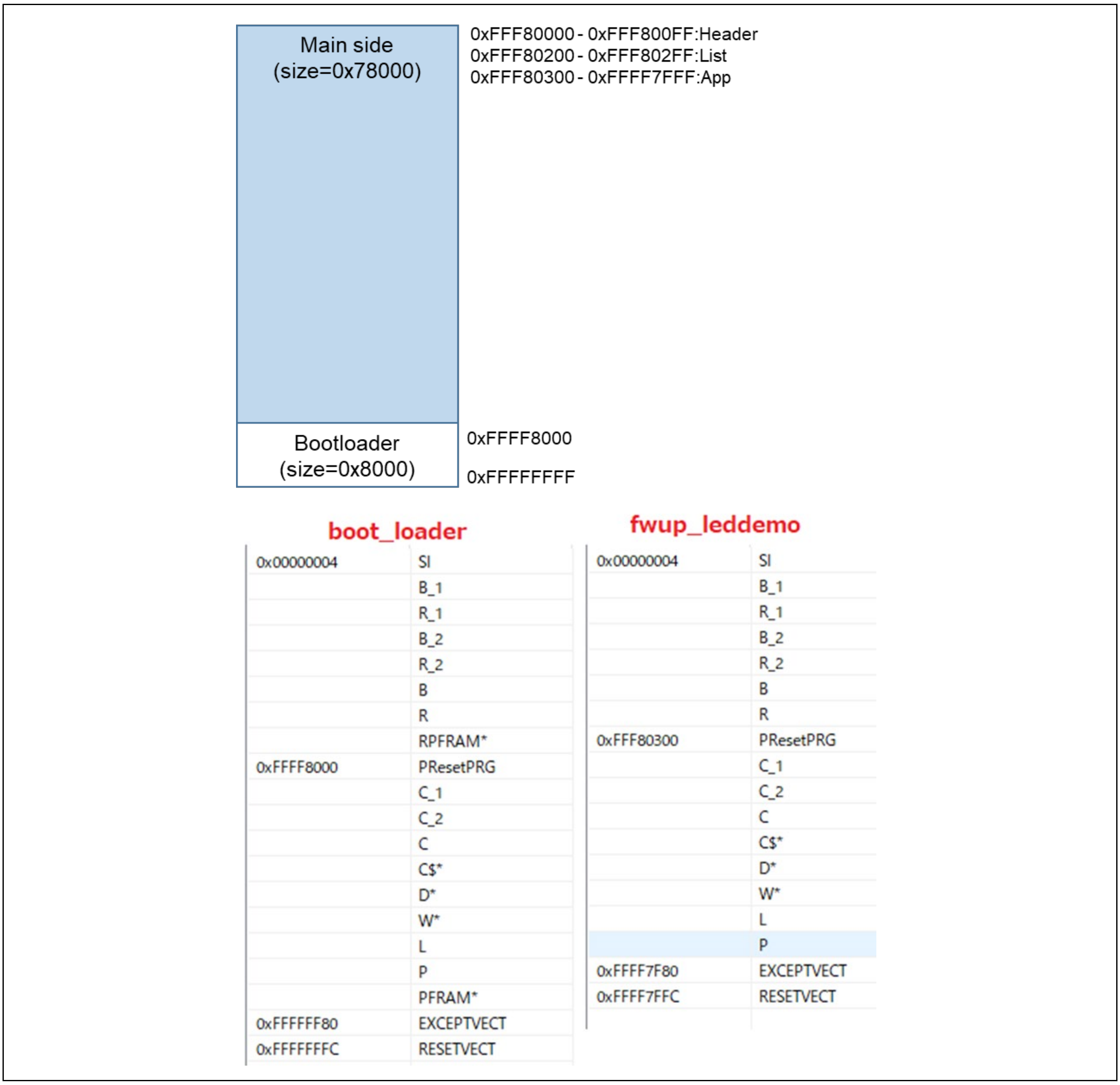
Figure 6.10 RX65N dual bank method demo project memory map

Table 6.12 RX65N linear mode half-surface update method configuration setting

Configuration options in r_fwup _config.h		
parameter name	**boot_loader**	**fwup_main**
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFE00000	0xFFE00000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFF00000	0xFFF00000
FWUP_CFG_AREA_SIZE	0xF0000	0xF0000
FWUP_CFG_CF_BLK_SIZE	0x8000	0x8000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096	4096
FWUP_CFG_DF_ADDR_L	0x00100000	0x00100000
FWUP_CFG_DF_BLK_SIZE	64	64
FWUP_CFG_DF_NUM_BLKS	512	512
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	0	0

6.2.2.3 Memory map of demo project for full update method in linear mode

The memory map of the RX26T linear mode full update method demo project and the memory map of the configuration settings are shown below.


Main side
(size=0x78000)
0xFFF80000 - 0xFFF800FF:Header
0xFFF80200 - 0xFFF802FF:List
0xFFF80300 - 0xFFFF7FFF:App
Bootloader
(size=0x8000)
0xFFFF8000
0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFF8000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_leddemo

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
0xFFF80300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
0xFFFF7F80	EXCEPTVECT
0xFFFF7FFC	RESETVECT

Figure 6.11 RX26T linear mode full update method demo project memory map

Table 6.13 RX26T linear mode full update method configuration setting

Configuration options in r_fwup _config.h	
parameter name	**boot_loader**
FWUP_CFG_UPDATE_MODE	2
FWUP_CFG_FUNCTION_MODE	0
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFF80000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFF80000
FWUP_CFG_AREA_SIZE	0x78000
FWUP_CFG_CF_BLK_SIZE	0x4000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096
FWUP_CFG_DF_ADDR_L	0x00100000
FWUP_CFG_DF_BLK_SIZE	64
FWUP_CFG_DF_NUM_BLKS	256
FWUP_CFG_SIGNATURE_VERIFICATION	0
FWUP_CFG_PRINTF_DISABLE	0

6.2.3 Operation Confirmation Environment for RX24T

The execution environment and connection diagram are shown below.

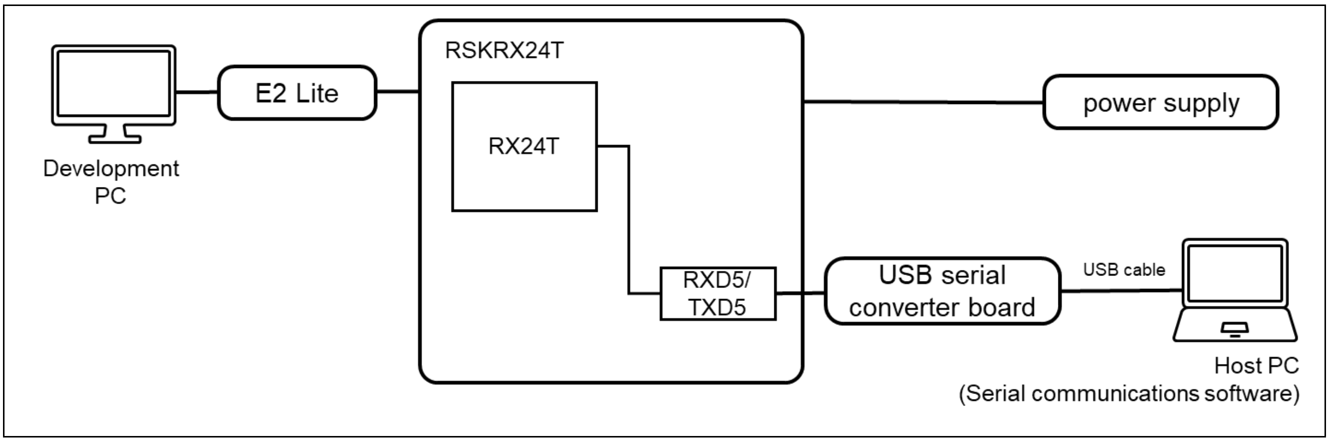


Figure 6.12 RSK-RX24T Device Connection Diagram

The pin assignment is shown in the figure below.

■ UART(Red)

PMOD1		USB-UART
2	TXD5	RX
3	RXD5	TX
4	PB7(RTS)	CTS

■ USER_SW (Green)

SW1	Note
P10	LOW : USER_SW is ON

■ Reset Switch (Blue)

RES	Note
RESn	Reset SW for MCU

■ USER_LED (Yellow)

LED0	Note
PB3	LED2

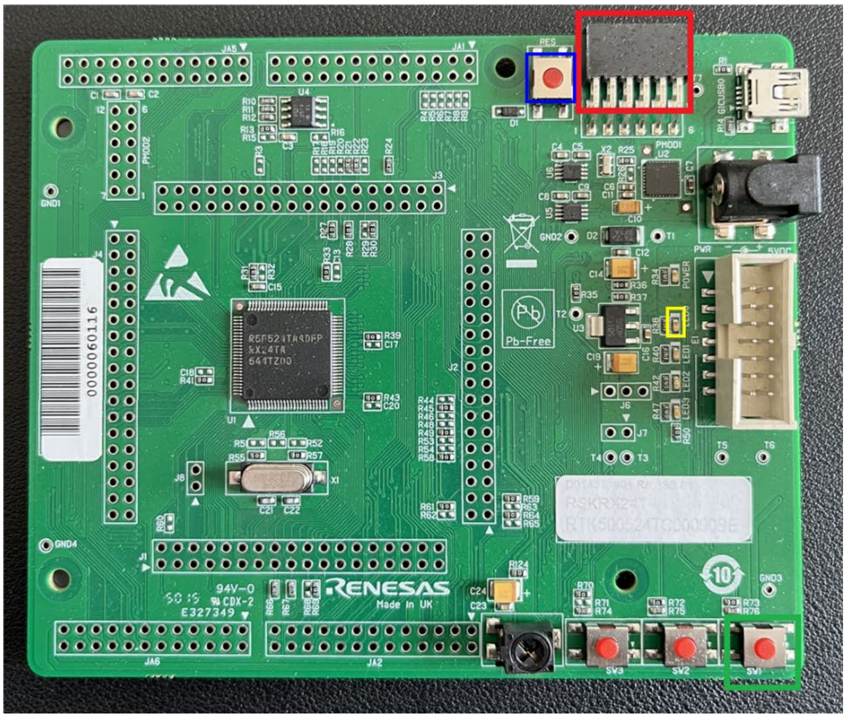
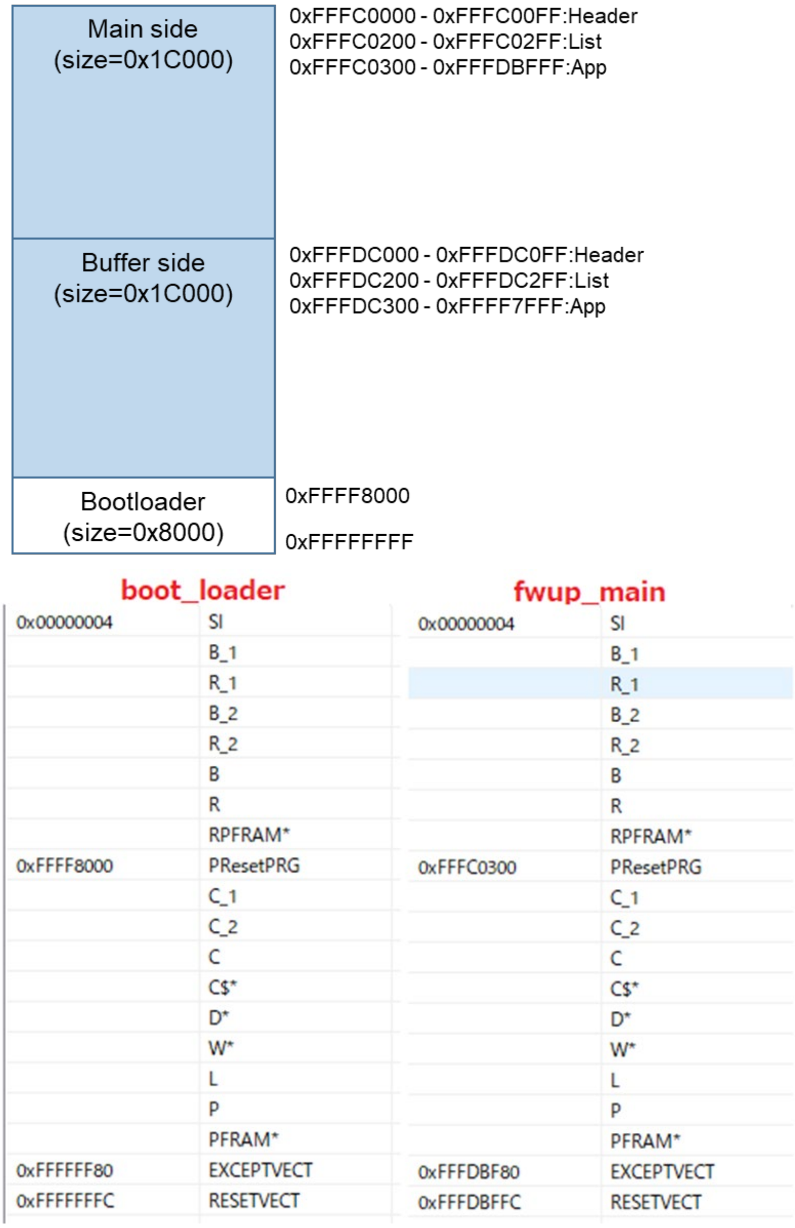


Figure 6.13 RSK-RX24T Pin Information

6.2.3.1 Memory map of demo project for half-surface update method in linear mode

Shown below are the memory map of the RX24T linear mode half-surface update method demo project and the memory map of the configuration settings.


Main side
(size=0x1C000)
0xFFFC0000 - 0xFFFC00FF:Header
0xFFFC0200 - 0xFFFC02FF:List
0xFFFC0300 - 0xFFFDBFFF:App
Buffer side
(size=0x1C000)
0xFFFDC000 - 0xFFFDC0FF:Header
0xFFFDC200 - 0xFFFDC2FF:List
0xFFFDC300 - 0xFFFF7FFF:App
Bootloader
(size=0x8000)
0xFFFF8000
0xFFFFFFFF

boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFF8000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_main

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFC0300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFDBF80	EXCEPTVECT
0xFFFDBFFC	RESETVECT

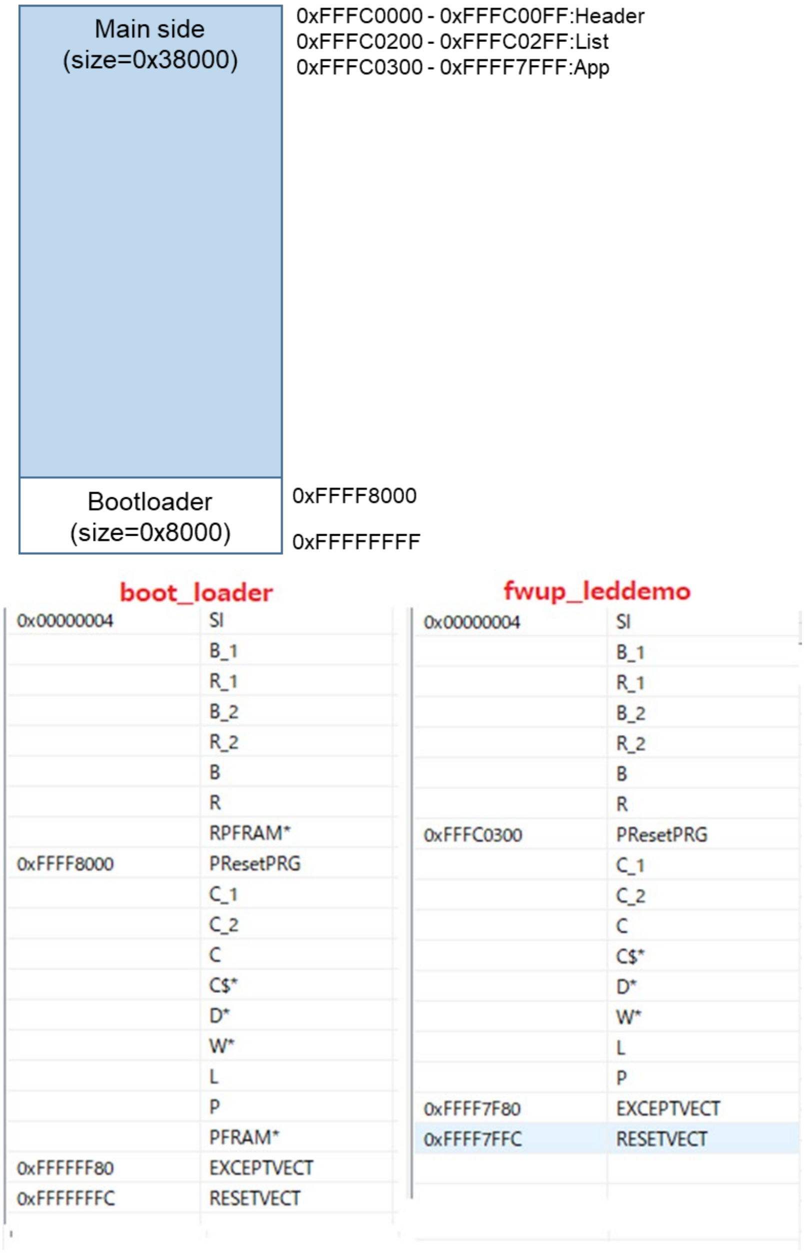
Figure 6.14 RX24T linear mode half-surface update method demo project memory map

Table 6.14 RX24T linear mode half-surface update method configuration setting

Configuration options in r_fwup _config.h		
parameter name	**boot_loader**	**fwup_main**
FWUP_CFG_UPDATE_MODE	1	1
FWUP_CFG_FUNCTION_MODE	0	1
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFE00000	0xFFE00000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFF00000	0xFFF00000
FWUP_CFG_AREA_SIZE	0xF0000	0xF0000
FWUP_CFG_CF_BLK_SIZE	0x8000	0x8000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096	4096
FWUP_CFG_DF_ADDR_L	0x00100000	0x00100000
FWUP_CFG_DF_BLK_SIZE	64	64
FWUP_CFG_DF_NUM_BLKS	512	512
FWUP_CFG_SIGNATURE_VERIFICATION	0	0
FWUP_CFG_PRINTF_DISABLE	0	0

6.2.3.2 Memory map of demo project for full update method in linear mode

The memory map of the RX24T linear mode full update method demo project and the memory map of the configuration settings are shown below.



boot_loader

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
	RPFRAM*
0xFFFF8000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
	PFRAM*
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

fwup_leddemo

0x00000004	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
0xFFFC0300	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P
0xFFFF7F80	EXCEPTVECT
0xFFFF7FFC	RESETVECT

Figure 6.15 RX24T linear mode full update method demo project memory map

Table 6.15 RX24T linear mode full update method configuration setting

Configuration options in r_fwup _config.h	
parameter name	**boot_loader**
FWUP_CFG_UPDATE_MODE	2
FWUP_CFG_FUNCTION_MODE	0
FWUP_CFG_MAIN_AREA_ADDR_L	0xFFE00000
FWUP_CFG_BUF_AREA_ADDR_L	0xFFE00000
FWUP_CFG_AREA_SIZE	0x1F0000
FWUP_CFG_CF_BLK_SIZE	0x8000
FWUP_CFG_EXT_BUF_AREA_ADDR_L (unused)	0x0000
FWUP_CFG_EXT_BUF_AREA_BLK_SIZE (unused)	4096
FWUP_CFG_DF_ADDR_L	0x00100000
FWUP_CFG_DF_BLK_SIZE	64
FWUP_CFG_DF_NUM_BLKS	512
FWUP_CFG_SIGNATURE_VERIFICATION	0
FWUP_CFG_PRINTF_DISABLE	0

6.3 Open source license information used in the demo project

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

Check out the TinyCrypt license terms below.

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

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

6.4 Notes on setting peripheral functions during the transition from bootloader to application.

When transitioning from the bootloader sample program to the application, the settings of the bootloader's peripheral functions are taken over by the application. Therefore, the bootloader sample program is implemented as follows.

For FIT modules (SCI, Flash) used in the bootloader, the API functions are closed at the end of the bootloader. Other settings are default values when Smart Configurator is used.

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

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

Please note that the Clock setting in the sample program for RSK-RX65N has some differences from the default value set by the Smart Configurator.
The differences are as follows.

- Each sample program of the dual bank method for CC-RX

  External bus clock selection (BCLK): 60MHz (initial value) -> 120MHz

- Sample programs of the Dual-Bank Method and Linear Mode Partial Update Method, and sample programs of the bootloader in Linear Mode Full Update Method for GCC

  External bus clock selection (BCLK): 60MHz (initial value) -> 120MHz

  Sub-clock: Used (initial value) -> Unused

- Sample program for LED demo of Linear Mode Full Update Method for GCC

  RTCSCK: Used (initial value) -> Unused

Revision History

Rev.	Date	Description	
		Page	Summary
2.00	Jul. 20, 2023	–	First edition issued

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/.