

RX Family

MCUboot Firmware Integration Technology

Introduction

This application note describes the MCUboot module that uses Firmware Integration Technology (FIT). This module is hereinafter referred to as the “MCUboot FIT module”.

The MCUboot FIT module is a FIT module created on the basis of MCUboot V2.1.0, which is publicly available at the following “mcu-tools” web page of GitHub: <https://github.com/mcu-tools/mcuboot>.

This application note describes how to use the MCUboot FIT module and how to embed the module into a user application.

The release package of this application note includes a demo project. You can build the environment for executing demonstration by using the procedure described in “4. Demo Project”. With the demonstration, you can check the basic operation of the MCUboot FIT module.

Target Devices

MCUs of the RX261 Group

MCUs of the RX65N and RX651 Groups

MCUs of the RX66N Group

MCUs of the RX671 Group

MCUs of the RX72M Group

MCUs of the RX72N Group

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

Related Application Notes

The following shows a list of application notes that are related to this application note. Refer also to them.

- 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 TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0371)
- RX Family RSIP (Renesas Secure IP) Module in Protected Mode Using Firmware Integration Technology (R20AN0748)
- RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
- RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)

Target Compilers

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

For details of the confirmed operation contents of each compiler, refer to “5.1 Environment Used for Verifying Operation”.

Contents

1.	Overview.....	6
1.1	Overview of MCUboot	6
1.2	Overview of the MCUboot FIT Module.....	6
1.3	System Configuration	7
1.4	Operation of MCUboot	8
1.5	Supported Update Methods of MCUboot.....	9
1.5.1	Overwrite Only/Only Fast Methods	9
1.5.1.1	Overwrite Only Method.....	9
1.5.1.2	Overwrite Only Fast Method	9
1.5.2	Swap Method	10
1.5.3	DirectXIP Method	11
1.5.3.1	DirectXIP Method in Linear Mode	11
1.5.3.2	DirectXIP Method in Dual Mode.....	11
1.6	Package Configuration	12
1.7	Overview of API Functions	14
2.	API Information.....	15
2.1	Hardware Requirements	15
2.2	Software Requirements.....	15
2.3	Supported Toolchain	15
2.4	Header Files	15
2.5	Integer Types.....	15
2.6	Configuration Overview	16
2.7	Code Sizes of the Sample Project.....	19
2.8	Parameters	23
2.9	Return Values.....	23
2.10	How to Add a FIT Module.....	24
2.11	About the “for”, “while”, and “do while” Statements.....	25
2.12	Example of Implementing API Functions	26
3.	API Functions	27
3.1	boot_go.....	27
3.2	RM_MCUBOOT_BootApp.....	27
3.3	RM_MCUBOOT_GetVersion	27
4.	Demo Project.....	28
4.1	Configuration of the Demo Project	28
4.1.1	Details of the Initial Image	29
4.2	Preparing the Operating Environment.....	32
4.2.1	Obtaining Imgtool	32

4.2.2	Installing Terminal Software	32
4.2.3	Installing the Python Runtime Environment	32
4.2.4	Installing the OpenSSL Runtime Environment.....	32
4.2.5	Installing a Flash Memory Writer.....	33
4.2.6	Installing Security Key Management Tool.....	33
4.2.7	USB-to-Serial Conversion Board	33
4.3	Procedure for Executing the Demo Project.....	34
4.3.1	Key Injection.....	34
4.3.1.1	Generating Key Data by Using Security Key Management Tool	34
4.3.1.2	Key Injection.....	36
4.3.2	Embedding the Public Key for Signature Verification	38
4.3.3	Preparing the Images for the Demo Project.....	39
4.3.3.1	Generating a Bootloader Image	39
4.3.3.2	Generating the Initial Image	40
4.3.3.3	Generating an Update Image.....	41
4.3.4	Programming the Demo Project.....	43
4.3.5	Executing the Demo Project.....	47
5.	Appendix.....	48
5.1	Environment Used for Verifying Operation.....	48
5.2	Operating Environment of the Demo Project	51
5.2.1	Environment Used for Verifying Operation of the RX261	51
5.2.1.1	Information on Hardware Component Connections.....	51
5.2.1.2	Memory Allocation and Configuration Option Settings	52
5.2.2	Environment Used for Verifying Operation of the RX65N.....	53
5.2.2.1	Information on Hardware Component Connections for an Update in Linear Mode.....	53
5.2.2.2	Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode	54
5.2.2.3	Information on Hardware Component Connections for an Update in Dual Mode	55
5.2.2.4	Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode	56
5.2.3	Environment Used for Verifying Operation of the RX671	57
5.2.3.1	Information on Hardware Component Connections for an Update in Linear Mode.....	57
5.2.3.2	Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode	58
5.2.3.3	Information on Hardware Component Connections for an Update in Dual Mode	59
5.2.3.4	Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode	60
5.2.4	Environment Used for Verifying Operation of the RX72M	61
5.2.4.1	Information on Hardware Component Connections for an Update in Linear Mode.....	61
5.2.4.2	Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode	62

5.2.4.3 Information on Hardware Component Connections for an Update in Dual Mode 63

5.2.4.4 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode 64

5.2.5 Environment Used for Verifying Operation of the RX72N..... 65

5.2.5.1 Information on Hardware Component Connections for an Update in Linear Mode..... 65

5.2.5.2 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode 66

5.2.5.3 Information on Hardware Component Connections for an Update in Dual Mode 67

5.2.5.4 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode 68

6. Notes 69

6.1 Notes on Transition from Bootloader(MCUboot) to Application..... 69

Revision History 70

1. Overview

1.1 Overview of MCUboot

MCUboot is a secure bootloader for 32-bit microcontrollers.

MCUboot provides a secure bootloader that enables easy software upgrade by defining a common infrastructure for the bootloader and system flash layout on microcontroller systems.

MCUboot does not depend on any specific operating system and hardware. It depends on hardware porting layers from the operating system it works with.

MCUboot is publicly available at the following “mcu-tools” web page of GitHub: <https://github.com/mcu-tools/mcuboot>.

Major functions supported by MCUboot are as follows:

- Function that starts an application
- Function that verifies the signature of an application
- Function that updates or switches an application
- Function that decrypts and updates an encrypted application image

1.2 Overview of the MCUboot FIT Module

The MCUboot FIT module is a FIT module created on the basis of aforementioned MCUboot. When users create a bootloader, they can easily embed MCUboot into it by using this FIT module.

In addition, MCUboot FIT uses Renesas security IP (TSIP and RSIP), which enable secure user key concealment and faster decryption.

The MCUboot FIT module supports the following update methods:

- Overwrite Only (linear mode)
- Overwrite Only Fast (linear mode)
- Swap (linear mode)
- DirectXIP (linear or dual mode)

The MCUboot FIT module supports the following signature verification methods:

- ECDSA NIST P-256
- RSA 2048 (RSASSA-PSS) : RX261 not supported
- No signature verification

1.3 System Configuration

Figure 1-1 shows the system configuration of a bootloader, the MCUboot FIT module used by the bootloader, and the demo application. Table 1-1 lists the modules used in this system.

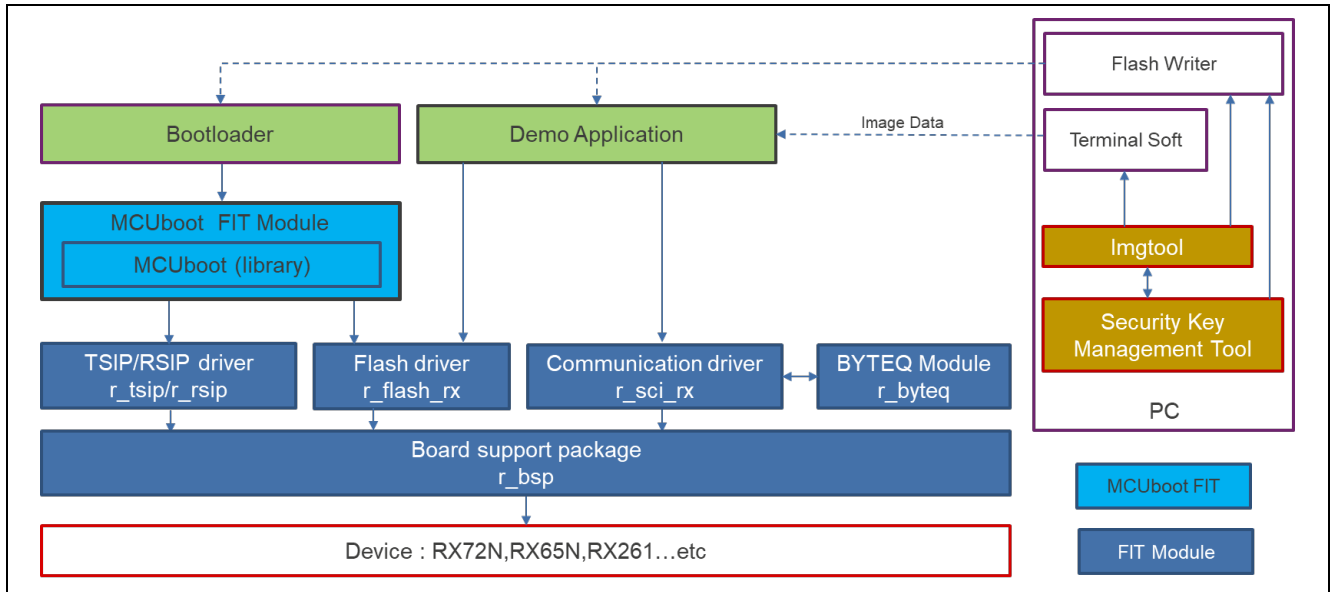


Figure 1-1 System Configuration of the Bootloader and Demo Application

Table 1-1 List of Modules Used by the Bootloader and Demo Application

Type	Application Note Name (Document No.)	FIT Module Name	Remarks
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 TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0371)	r_tsip	Used for the RX65N/RX651/ RX66N/RX671/ RX72M/RX72N
Device driver	RX Family RSIP (Renesas Secure IP) Module in Protected Mode Using Firmware Integration Technology (R20AN0748)	r_rsip_protected_rx	Used for the RX261
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	

1.4 Operation of MCUboot

MCUboot uses the flash memory of the microcontroller by segmenting it as shown in Figure 1-2.

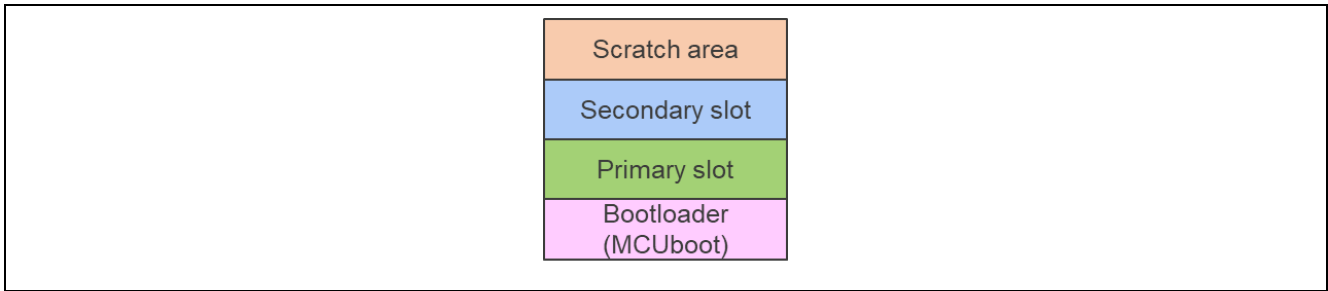


Figure 1-2 Memory Map of the Flash Memory Used by MCUboot

- Bootloader: Area that stores the bootloader that uses MCUboot
- Primary slot: Area that stores the bootable image
(User application started by MCUboot)
- Secondary slot: Area that stores the update image
(To update the image in the primary slot, place the update image in this area.)
- Scratch area: Buffer area used only for an update by the Swap method
(This area is unnecessary for the update methods other than Swap.)

MCUboot operates as follows:

1. The bootloader (MCUboot) starts after the MCU is released from the reset state.
2. MCUboot checks whether an update image is stored in the secondary slot.
3. If there is no update image in the secondary slot, go to step 6.
4. If there is an update image in the secondary slot, MCUboot verifies the signature of the update image.
5. If verification is passed, MCUboot replaces the image in the primary slot by the update image in the secondary slot. Only the Overwrite Only / Only Fast method erases the Secondary slot after the update.
6. MCUboot verifies the signature of the image in the primary slot.
7. If verification is passed, MCUboot activates the image (user application) in the primary slot.

1.5 Supported Update Methods of MCUboot

The MCUboot FIT module supports the following update methods of MCUboot:

- Overwrite Only/Only Fast
- Swap
- DirectXIP

The update methods that can be used depend on the mode of flash memory. In linear mode, the Overwrite Only/Only Fast, Swap, and DirectXIP methods can be used. In dual mode, the DirectXIP method can be used.

For details of these update methods, refer to sections 1.5.1 to 1.5.3.

1.5.1 Overwrite Only/Only Fast Methods

In Overwrite methods, the bootable image is always stored in the primary slot and activated from the slot. The update image is stored in the secondary slot.

When an update image is stored in the secondary slot, signature verification is performed for the content of the secondary slot. If verification is passed, the content of the secondary slot is copied to the primary slot. As a result, the content of the primary slot is updated, the secondary slot is then erased.

1.5.1.1 Overwrite Only Method

In the Overwrite Only method, an update is performed by copying the whole area of the secondary slot to the whole area of the primary slot.

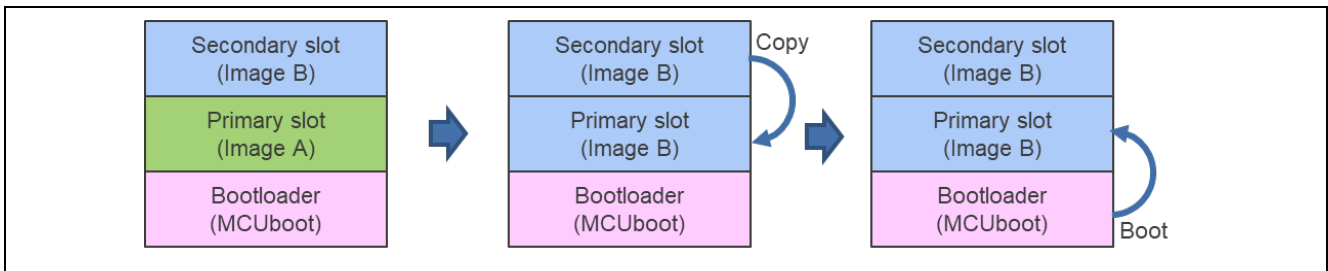


Figure 1-3 Update Operation of the Overwrite Only Method

1.5.1.2 Overwrite Only Fast Method

In the Overwrite Only Fast method, copy from the secondary slot to the primary slot is performed as in the Overwrite Only method. However, the Overwrite Only Fast method copies only the update image, whereas the Overwrite Only method copies the whole area of the secondary slot. The unused area in the secondary slot is not subject to copy. Therefore, if the update size is small, the time required to copy the update can be reduced.

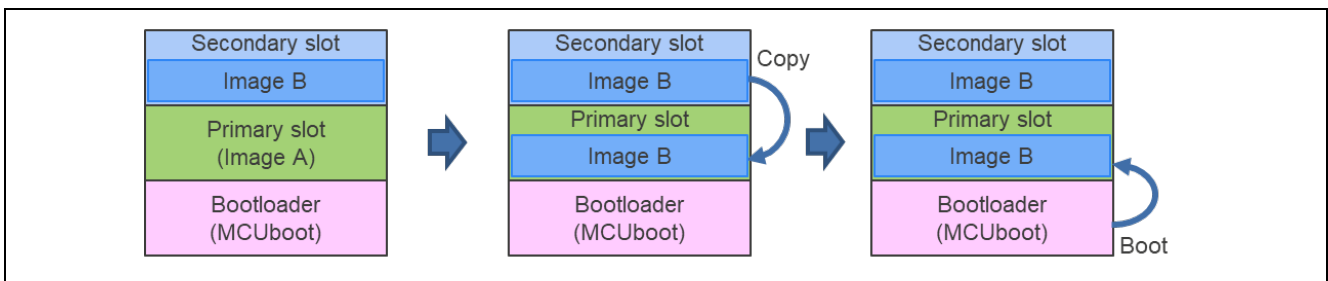


Figure 1-4 Update Operation of the Overwrite Only Fast Method

1.5.2 Swap Method

In the Swap method, the bootable image is always stored in the primary slot and activated from the slot. The update image is stored in the secondary slot.

When an update image is stored in the secondary slot, signature verification is performed for the content of the secondary slot. If verification is passed, the image in the secondary slot is saved in the scratch area, and then the image in the primary slot is copied to the secondary slot. After that, the image saved in the scratch area is copied to the primary slot. As a result, the image in the primary slot is updated.

In this method, the images in both slots are swapped by using the scratch area. Because the image that existed in the primary slot remains in the secondary slot, it is possible to perform a rollback to a pre-update state.

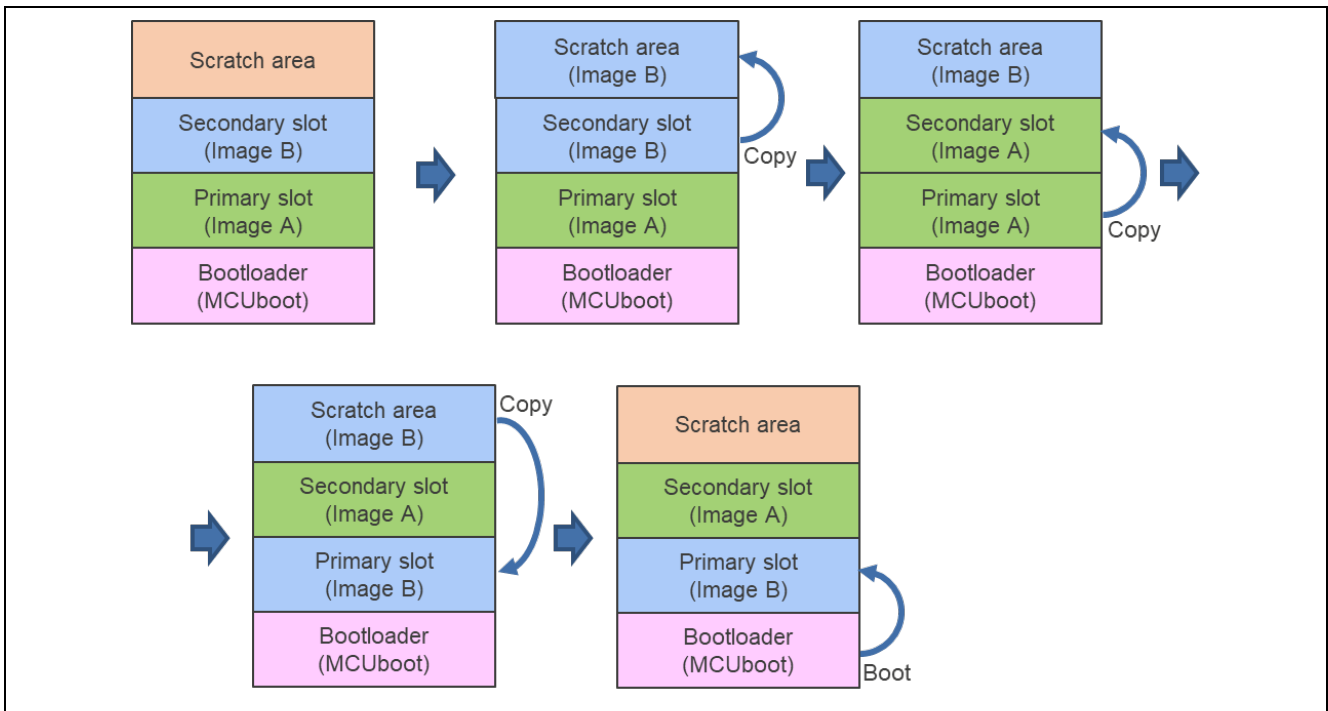


Figure 1-5 Update Operation of the Swap Method

1.5.3 DirectXIP Method

Unlike the Overwrite Only/Only Fast and Swap methods, copy between the primary and secondary slots does not occur in the DirectXIP method. The image in each slot can be directly activated.

The operation of the MCUboot FIT module changes according to the mode of flash memory.

Bootable images exist in both of the primary and secondary slots.

1.5.3.1 DirectXIP Method in Linear Mode

In the DirectXIP method in linear mode, MCUboot switches the valid slot from which to activate an image so that an update is performed.

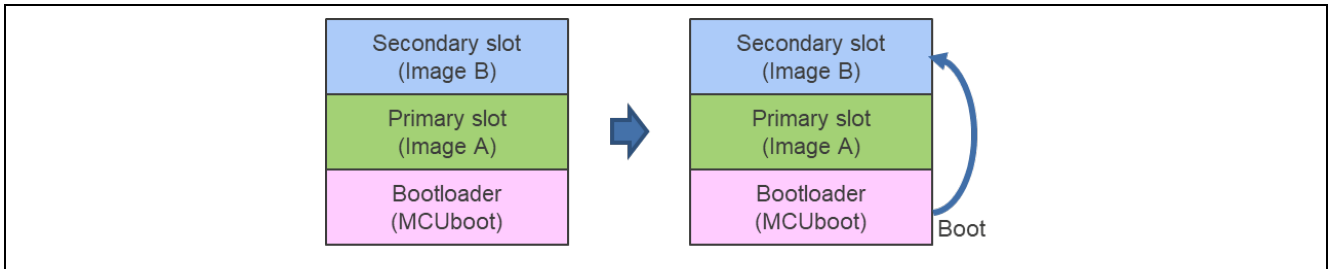


Figure 1-6 Update Operation of the DirectXIP Method in Linear Mode

1.5.3.2 DirectXIP Method in Dual Mode

In the DirectXIP method in dual mode, MCUboot uses the dual bank function of flash memory to swap the two banks of the flash memory so that the images in the primary and secondary slots are swapped.

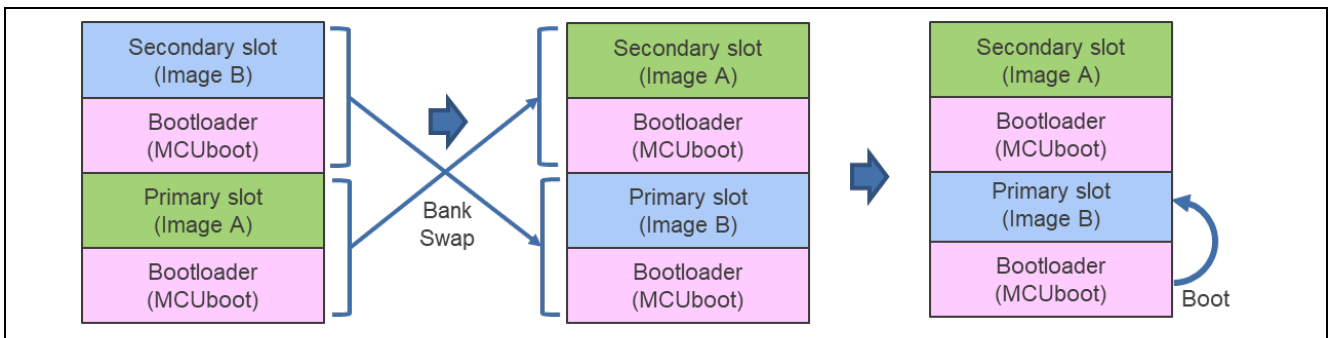


Figure 1-7 Update Operation of the DirectXIP Method in Dual Mode

1.6 Package Configuration

The package of the MCUboot FIT module contains software, tools, and other files. The following table lists these files.

Table 1-2 Folder Configuration of the Package of the MCUboot FIT Module

Folder Name	Description
rm_mcuboot_v1.02	FIT Module
├rm_mcuboot	MCUboot Module
│ └doc	Application Notes
│ └src	
│ │ └rm_mcuboot_port	MCUboot FIT
│ │ └mcu-tools	MCUboot & imgtool
│ └rm_mcuboot_if.h	Interface Header Files
└r_config	
└rm_mcuboot_config.h	Configuration Definition Files
fitdemos	FIT Demo
├common	Sample Common Files
├e2_ccrx	For CC-RX
│ └rx261-ek	For EK-RX261
│ │ └linear	
│ │ │ └application_primary	Application for initial image
│ │ │ └application_primary_another_slot	Application for updated image (for DirectXIP)
│ │ │ └boot_loader	Bootloader
│ │ │ └key_injection	Application of key injection
│ └rx###-rsk	For RSK-RX###
│ │ └dual_bank	For dual mode
│ │ │ └application_primary	Initial Image Application
│ │ │ └boot_loader	Bootloaders
│ │ │ └key_injection	Application of key injection
│ │ └linear	For linear mode
│ │ │ └application_primary	Application for initial image
│ │ │ └application_primary_another_slot	Application for updated image (for DirectXIP)
│ │ │ └boot_loader	Bootloader
│ │ │ └key_injection	Application of key injection
├e2_gcc	For GCC-R
│ └rx261-ek	For EK-RX261
│ │ └linear	
│ │ │ └application_primary	Application for initial image
│ │ │ └application_primary_another_slot	Application for updated image (for DirectXIP)
│ │ │ └boot_loader	Bootloader
│ │ │ └key_injection	Application of key injection
│ └rx###-rsk	For RSK-RX###
│ │ └dual_bank	For dual mode
│ │ │ └application_primary	Initial Image Application
│ │ │ └boot_loader	Bootloaders
│ │ │ └key_injection	Application of key injection

	└linear	For linear mode
	└application_primary	Application for initial image
	└application_primary_another_slot	Application for updated image (for DirectXIP)
	└boot_loader	Bootloader
	└key_injection	Application of key injection
	└iar	For IAR
	└rx261-ek	For EK-RX261
	└linear	
	└application_primary	Application for initial image
	└application_primary_another_slot	Application for updated image (for DirectXIP)
	└boot_loader	Bootloader
	└key_injection	Application of key injection
	└rx###-rsk	For RSK-RX###
	└dual_bank	For dual mode
	└application_primary	Initial Image Application
	└boot_loader	Bootloaders
	└key_injection	Application of key injection
	└linear	For linear mode
	└application_primary	Application for initial image
	└application_primary_another_slot	Application for updated image (for DirectXIP)
	└boot_loader	Bootloader
	└key_injection	Application of key injection

(###: 65N / 671 / 72M / 72N)

1.7 Overview of API Functions

Table 1-3 describes the API functions included in the MCUboot FIT module.

Table 1-3 List of API Functions

Function	Function Description
boot_go	This function obtains the information about the image to be activated. If an update image is identified in the secondary slot during verification of the image information, signature verification of the image is performed. If verification is passed, the update image is stored in the primary slot according to the update method.
RM_MCUBOOT_BootApp	This function closes the drivers related to the MCUboot FIT module, and then activates the image specified in the parameter.
RM_MCUBOOT_GetVersion	This function obtains the version of the module.

2. API Information

Operation of the MCUboot FIT module was verified under the conditions shown in the following sections.

2.1 Hardware Requirements

The MCU being used must support the following components:

- On-chip flash memory
- Hardware cryptography (TSIP/RSIP)

2.2 Software Requirements

The MCUboot FIT module is dependent on the following drivers:

- Board Support Package (r_bsp)
- Flash module (r_flash_rx)
- TSIP/RSIP module (r_tsip/r_rsip_protected_rx)
- Serial Communications Interface (SCI) (asynchronous/clock synchronous mode) (r_sci_rx)
- Byte Queue (BYTEQ) module (r_byteq)

2.3 Supported Toolchain

Operation of the MCUboot FIT module was verified by using the toolchain shown in “5.1 Environment Used for Verifying Operation”.

2.4 Header Files

All API function calls and the interface definitions that support the API function calls are described in `rm_mcuboot_if.h`.

2.5 Integer Types

This project uses ANSI C99. This type is defined in `stdint.h`.

2.6 Configuration Overview

The configuration options of the MCUboot FIT module are set in `rm_mcuboot_config.h`.

Table 2-1 (Configuration Options) shows the names of configuration options and describes the values that can be set.

Table 2-1 Configuration Options

Configuration Options	
RM_MCUBOOT_CFG_UPGRADE_MODE	This option sets the update method. The following update methods of MCUboot can be selected: 0: Overwrite Only [Default] 1: Overwrite Only Fast 2: Swap 3: DirectXIP
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	This option sets whether to perform signature verification of the primary image. Enable this option if you want to perform signature verification of the primary image before activating the image. 0: Disable 1: Enable [Default]
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	This option sets whether to prevent the image from being downgraded during an update. Change the setting of this option when you use the Overwrite Only or Overwrite Only Fast method. 0: Disable [Default] 1: Enable
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	This option must be enabled when a user-defined watchdog feed is used. Enabling this option prevents the system from being reset by watchdog before the processing of MCUboot is completed. 0: Disable [Default] 1: Enable
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	This option registers the user-defined watchdog function in <code>MCUBOOT_WATCHDOG_FEED</code> .
RM_MCUBOOT_CFG_SIGN	This option sets the signature verification method. Use this option to set the method of signature verification for the image. If signature verification for the primary image is enabled, signature verification is performed by the method set by this option. 0: None 1: ECDSA P-256 [Default] 2: RSA 2048
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	This option must be enabled if the update image is to be encrypted. This option can be set when the update method is Overwrite Only/Only Fast or Swap. This option is not available when using the DirectXIP method. 0: Encryption Disabled [Default] 1: Key Wrap

Configuration Options	
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	This option must be enabled if DER-formatted public key data provided by the user is to be used. The setting of this option can be changed only if the method of signature verification is set. 0: Disable [Default] 1: Enable
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	This option sets the address of the public key used for signature verification. [Default: NULL]
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	This option sets the address of the key encryption key used by the image decryption function. [Default: NULL]
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	This option sets the size of the area allocated to MCUboot. (Note: When setting this option, consider the block size of the flash memory mounted on the device you use.) [Default: 0x10000]
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	This option sets the size of the area allocated to the application image. (Note: When setting this option, consider the block size of the flash memory mounted on the device you use.) [Default: RX65N, RX671: 0xF0000 RX66N, RX72M, RX72N: 0x1F0000 RX261: 0x30000]
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	This option sets the size of the scratch area. This option must be set if the specified update method is Swap. (Note: When setting this option, consider the block size of the flash memory mounted on the device you use. The value to be set must be a multiple of the sector size of the flash memory to be used as the scratch area.) [Default: 0x10000]
RM_MCUBOOT_CFG_LOG_LEVEL	This option sets the logging level. MCUboot outputs the log data whose level is equal to or higher than the level set by this option. 0: Off [Default] 1: Error 2: Warning 3: Info 4: Debug

Some of the configuration options listed in Table 2-1 are overridden by other options that are set to specific values. The following tables show the option settings that override other options.

Table 2-2 Combination of the Option That Sets the Update Method and the Option That Sets Whether to Prevent Downgrading

RM_MCUBOOT_CFG_UPGRADE_MODE	RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION
1 (Overwrite Only)	0 (Disable) / 1 (Enable)
2 (Overwrite Only Fast)	0 (Disable) / 1 (Enable)
3 (Swap)	This option is overridden.
4 (DirectXIP)	This option is overridden.

Table 2-3 Combination of the Macro Definitions Related to User-Defined Watchdog Feed

RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION
0 (Disable)	This option is overridden.
1 (Enable)	The user-defined watch dog function is registered.

Table 2-4 Combination of the Options Related to the Signature Verification Method Settings

RM_MCUBOOT_CFG_SIGN	RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT
0 (None)	This option is overridden.	This option is overridden.
1 (ECDSA P-256)	The address of the public key is set.	0 (Disable)
		1 (Enable)
2 (RSA 2048)	The address of the public key is set.	0 (Disable)
		1 (Enable)

Table 2-5 Combination of the Option That Sets Decryption of the Encrypted Image and the Option That Sets the Address of the Key Encryption Key

RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS
0 (Encryption Disable)	This option is overridden.
1 (Key Wrap)	The address of the key encryption key is set.

2.7 Code Sizes of the Sample Project

Table 2-6 shows the sizes of ROM and RAM spaces and the maximum size of stack area used by the sample project included in the package of this application note. The values in this table were verified under the following conditions:

Module revision: MCUboot Module for RX v1.0.2

Compiler versions: Renesas Electronics C/C++ Compiler for RX Family V3.07.00

GCC for Renesas RX 8.3.0.202411

IAR C/C++ Compiler for Renesas RX 5.10.1

CC-RX

- Optimization level: Size & execution speed (-Odefault)
- Option that deletes variables/functions that have never been referenced (-optimize = symbol_delete)
- Subroutine multiple identical instructions (-optimize=same_code)
- Replace instructions with ones that have smaller code size (-optimize=short_format)
- Optimize branch instruction size based on program layout (-optimize=branch)
- Option that generates a functionally cut down version of the set of I/O functions (Yes: Maximally cut-down version)

GCC

- Optimization level: Size (-Os)
- Use newlib-nano (--specs = nano.specs)
- Set 'User defined options, -WI,--no-gc-sections' only for 'RX65N/RX671/RX72M/RX72N,dual-bank, application_primary'

IAR

- Optimization level: High (Balance)

Table 2-6 Sizes of the ROM, RAM, and Stack Areas Required by the Sample Project

compiler	device	bank mode	sample	ROM	RAM	Stack	
CC-RX	RX261	linear	application_primary	17673	9903	192	
			key_injection	17673	9903	192	
			boot_loader	overwrite_only	59876	12236	456
				overwrite_only_fast	60189	12268	456
				swap	62922	13052	457
	directXIP	55123	9696	458			
	RX65N	linear	application_primary	16811	10817	192	
			key_injection	23066	11303	284	
			boot_loader	overwrite_only	43815	19474	320
				overwrite_only_fast	44681	19622	320
				swap	49363	19878	316
		directXIP	33414	18106	248		
		dual	application_primary	17511	9421	188	
			key_injection	21441	11464	188	
			boot_loader directXIP	34342	16694	248	
		RX671	linear	application_primary	17243	10940	192
	key_injection			23457	11442	284	
	boot_loader			overwrite_only	44244	19597	320
				overwrite_only_fast	45113	19745	320
				swap	49791	20001	316
	directXIP		33839	18229	248		
	dual		application_primary	17888	9437	192	
			key_injection	21833	11604	192	
			boot_loader directXIP	33955	16710	248	
	RX72M		linear	application_primary	17806	11088	192
		key_injection		24021	11590	284	
		boot_loader		overwrite_only	45056	20257	320
				overwrite_only_fast	45932	20437	320
				swap	50600	20917	316
		directXIP	34647	18377	248		
		dual	application_primary	18445	9585	192	
			key_injection	22391	11752	192	
			boot_loader directXIP	34725	16853	248	
		RX72N	linear	application_primary	17667	11044	192
	key_injection			23882	11564	284	
	boot_loader			overwrite_only	44920	20213	320
				overwrite_only_fast	45796	20361	320
				swap	50467	20873	316
	directXIP		34514	18333	248		
	dual		application_primary	18312	9541	192	
key_injection			22258	11708	192		
boot_loader directXIP			34598	16814	248		

compiler	device	bank mode	sample	ROM	RAM	Stack	
GCC	RX261	linear	application_primary	14640	12436	68	
			key_injection	21400	11796	260	
			boot_loader	overwrite_only	54408	14752	700
				overwrite_only_fast	52915	14764	700
				swap	57274	15532	700
	directXIP	49023		12200	684		
	RX65N	linear	application_primary	17796	12696	68	
			key_injection	22484	12952	928	
			boot_loader	overwrite_only	42384	21284	928
				overwrite_only_fast	43035	21424	928
				swap	47406	21680	928
		directXIP		33452	20012	928	
		dual	application_primary	21380	10784	68	
			key_injection	20988	12952	928	
			boot_loader directXIP	33267	17964	928	
		RX671	linear	application_primary	18208	12824	68
	key_injection			22904	12952	928	
	boot_loader			overwrite_only	42829	21412	928
				overwrite_only_fast	43480	21552	928
			swap	47851	21808	928	
	directXIP		33896	20140	928		
	dual		application_primary	22044	10784	68	
			key_injection	21392	13080	928	
		boot_loader directXIP	33699	18092	928		
	RX72M	linear	application_primary	18992	12952	68	
			key_injection	23680	13208	928	
			boot_loader	overwrite_only	45148	22052	928
				overwrite_only_fast	45800	22320	928
				swap	50172	22832	928
		directXIP		36212	20268	928	
		dual	application_primary	22564	10912	68	
			key_injection	22160	13208	928	
	boot_loader directXIP		36008	18220	928		
	RX72N	linear	application_primary	18776	12952	68	
			key_injection	23464	13080	928	
			boot_loader	overwrite_only	42384	21284	928
				overwrite_only_fast	45584	22192	928
				swap	49956	22704	928
		directXIP		36004	20140	928	
		dual	application_primary	26188	11040	48	
			key_injection	21944	13080	928	
			boot_loader directXIP	35800	18092	928	

compiler	device	bank mode	sample	ROM	RAM	Stack	
IAR	RX261	linear	application_primary	11516	7259	920	
			key_injection	18406	6618	1604	
			boot_loader	overwrite_only	49243	9578	3044
				overwrite_only_fast	49615	9620	3164
				swap	54867	10404	3148
	directXIP	45378		7044	3012		
	RX65N	linear	application_primary	15384	8507	1332	
			key_injection	20985	8996	1536	
			boot_loader	overwrite_only	41047	17156	2604
				overwrite_only_fast	41716	17307	2604
				swap	46812	17563	2588
		directXIP		31151	15787	2332	
		dual	application_primary	16112	6838	1448	
			key_injection	19400	9006	1472	
			boot_loader directXIP	31236	14097	2332	
		RX671	linear	application_primary	15936	8630	1332
	key_injection			21533	9120	1536	
	boot_loader			overwrite_only	41640	17279	2604
				overwrite_only_fast	42310	17432	2604
				swap	47422	17688	2588
			directXIP	31704	15910	2332	
	dual		application_primary	16669	6857	1448	
			key_injection	19973	9142	1472	
			boot_loader directXIP	31800	14113	2332	
	RX72M		linear	application_primary	16550	8782	1400
		key_injection		22205	9271	1605	
		boot_loader		overwrite_only	43107	17941	2692
				overwrite_only_fast	43785	18094	2692
				swap	48897	18606	2676
			directXIP	33171	16060	2420	
		dual	application_primary	17285	7005	1516	
			key_injection	20589	9293	1540	
			boot_loader directXIP	33251	14261	2420	
		RX72N	linear	application_primary	16375	8737	1400
	key_injection			21996	9228	1604	
	boot_loader			overwrite_only	42924	17900	2692
				overwrite_only_fast	43605	18050	2692
				swap	48717	18562	2676
			directXIP	32991	16145	2420	
	dual		application_primary	17109	6961	1516	
key_injection			20416	9250	1540		
boot_loader directXIP			33087	14217	2420		

2.8 Parameters

This section shows the parameters used in the API functions. These parameters are defined by the following structure, which is specified in bootutil.h together with the prototype declarations of the API functions.

```

struct boot_rsp {
    /** A pointer to the header of the image to be executed. */
    const struct image_header *br_hdr;

    /**
     * The flash offset of the image to execute. Indicates the position of
     * the image header within its flash device.
     */
    uint8_t br_flash_dev_id;
    uint32_t br_image_off;
};

```

Table 2-7 List of Parameters

Structure Name	Member	Description
boot_rsp	image_header br_hdr	Pointer to the header file of the image to be executed
	uint8_t br_flash_dev_id	ID of the flash device
	uint32_t br_image_off	Offset from the image to be executed

2.9 Return Values

This section shows the return values of the API functions. These return values are defined by the following enumeration, which is specified in bootutil.h together with the prototype declarations of the API functions.

```

#define FIH_POSITIVE_VALUE 0
#define FIH_NEGATIVE_VALUE -1

extern fih_ret FIH_SUCCESS;
extern fih_ret FIH_FAILURE;

```

Table 2-8 List of Return Values

Constant Definition	Numeric Value	Description
FIH_SUCCESS	0	A return value of API functions. This value is used to indicate that the processing of the API function was successful.
FIH_FAILURE	-1	A return value of API functions. This value is used to indicate that the processing of the API function failed.

2.10 How to Add a FIT Module

The MCUboot FIT module must be added to each project to be used.

(1) Adding a FIT module by using Smart Configurator on e² studio

A FIT module can be automatically added to a user project by using Smart Configurator of e² studio. For details, refer to "[RX Smart Configurator User's Guide: e² studio](#) (R20AN0451)".

(2) Adding a FIT module by using Smart Configurator in the environment of IAR Embedded Workbench for Renesas RX

If you are using the environment of IAR Embedded Workbench for Renesas RX, use RX Smart Configurator to add a FIT module to a user project. For details, refer to the following application note: "[RX Smart Configurator User's Guide: IAREW](#) (R20AN0535)".

2.11 About the “for”, “while”, and “do while” Statements

The MCUboot FIT module uses the “for”, “while”, and “do while” statements (that is, loop processing) in cases such as when waiting for some settings to be applied to registers. In sections where such loop processing is used, comments including the keyword “WAIT_LOOP” are added. If you want to add fail-safe processing to loop processing, you can locate the relevant sections by using “WAIT_LOOP” as a search string.

The following shows examples of sections where loop processing is coded:

Example of the "while" statement:

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

Example of the "for" statement:

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

Example of the "do while" statement:

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET));
/* WAIT_LOOP */
```

2.12 Example of Implementing API Functions

This section shows an example of implementing API functions in the MCUboot FIT module.

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

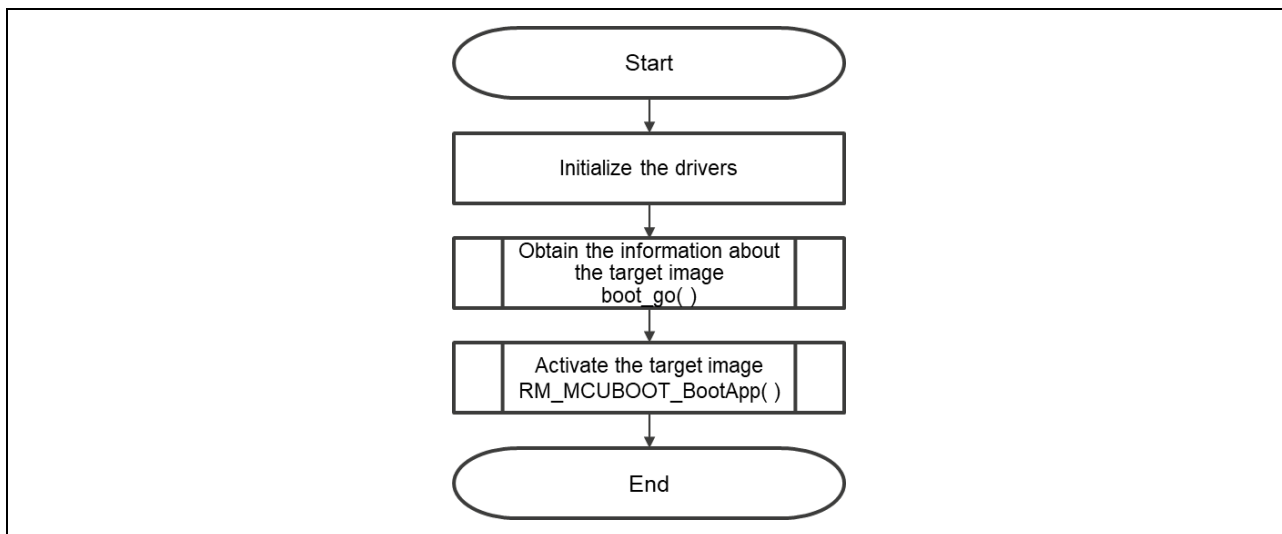


Figure 2-1 Example of Implementing API Functions in the MCUboot FIT Module

3. API Functions

3.1 boot_go

Table 3-1 Specifications of the “boot_go” Function

Format	fih_ret boot_go(struct boot_rsp *rsp)	
Description	<p>This function obtains the information about the image to be activated in the following procedure:</p> <ol style="list-style-type: none"> 1. Checks whether there is an image including a valid image header in each slot. If there is no update image in the secondary slot, this function skips step 2. 2. Verifies the signature of the update image and updates the target image according to the specified update method. The verification and update methods can be changed by using configuration options. <p>Returns the information about the image to be activated.</p>	
Parameters	struct boot_rsp *rsp	
Return Values	FIH_SUCCESS	The information about the image was successfully obtained.
	FIH_FAILURE	The information about the image could not be obtained.
Special Notes	<p>For details, refer to the following web page: https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md</p>	

3.2 RM_MCUBOOT_BootApp

Table 3-2 Specifications of the RM_MCUBOOT_BootApp Function

Format	void RM_MCUBOOT_BootApp (struct boot_rsp * rsp)	
Description	This function closes the drivers related to the MCUboot FIT module, and then activates the image specified in the parameter.	
Parameters	struct boot_rsp * rsp	
Return Values	None	
Special Notes	If the update method is DirectXIP in dual mode and the boot address is at the secondary slot, the RM_MCUBOOT_BootApp function swaps the banks and performs a software reset.	

3.3 RM_MCUBOOT_GetVersion

Table 3-3 Specifications of the RM_MCUBOOT_GetVersion Function

Format	uint32_t RM_MCUBOOT_GetVersion(void)	
Description	This function obtains the version of the MCUboot FIT module.	
Parameters	None	
Return Values	Version number of the MCUboot FIT module	
Special Notes	The major and minor version numbers of the MCUboot FIT module are managed by using the interface header file.	

4. Demo Project

The demo project is a sample program for demonstrating a firmware update by using MCUboot and Serial Communications Interface (SCI).

The following explains the terms used in this chapter.

Table 4-1 Descriptions of Terms

Term	Description
Key injection	Injecting keys into devices at factories, etc. MCUboot FIT injects keys into devices in Wrapped Key format.
User key	An encryption key in plaintext used by the user.
Encrypted Key	Key information generated by adding a MAC value and encrypting a user key using a UFPK . The key data file generated by SKMT.
Wrapped Key	Data containing an encrypted key converted into a format usable by TSIP/RSIP.
UFPK	User Factory Programming Key A keyring set by the user and used to generate an encrypted key from a user key during key injection.
W-UFPK	Wrapped UFPK Key information generated by wrapping a UFPK using an HRK on the DLM server. The UFPK is decrypted using the HRK internally by the TSIP/RSIP.
HRK	A shared encryption key that exists only inside the TSIP/RSIP and in secure rooms within Renesas.
HUK	A device-specific encryption key used to protect key data.
SKMT	Security Key Management Tool This is a support tool for implementing key management systems and secure functions using Renesas security IP.
DLM Server	Device Lifecycle Management Server The DLM server wraps the input UFPK with an HRK to generate a W-UFPK. It is publicly available as the Renesas Key Wrap Service (https://dlm.renesas.com/keywrap/).

4.1 Configuration of the Demo Project

The demo project consists of a bootloader and initial image. The bootloader includes the MCUboot FIT module and other dependent modules. The initial image includes the function for performing a firmware update. The demo project provided by the package of this application note supports the devices and compilers shown in section 1.6.

The demonstration of a firmware update using MCUboot is implemented by using the following projects:

- **Bootloader (MCUboot):** This component is first executed after a reset to verify the target image based on the verification method set by the relevant configuration option of the MCUboot FIT module. If there is an update image, this component updates the target image according to the specified update method.
- **Initial image:** When this component is activated by the bootloader (MCUboot), it downloads the update image through the communications interface and writes the update image to the secondary slot.
- **Update image:** This component is the same as the initial image but has a different version number. The update image can be encrypted but the initial image cannot be encrypted.
- **Key injection program:** This component is used when the public key for signature verification and the key for image decryption are injected.

4.1.1 Details of the Initial Image

The initial image is activated by the bootloader and placed in the primary slot. The initial image receives an update image through the communications interface and writes it to the secondary slot.

Figure 4-1 to Figure 4-5 show a processing flowchart of the initial image.

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

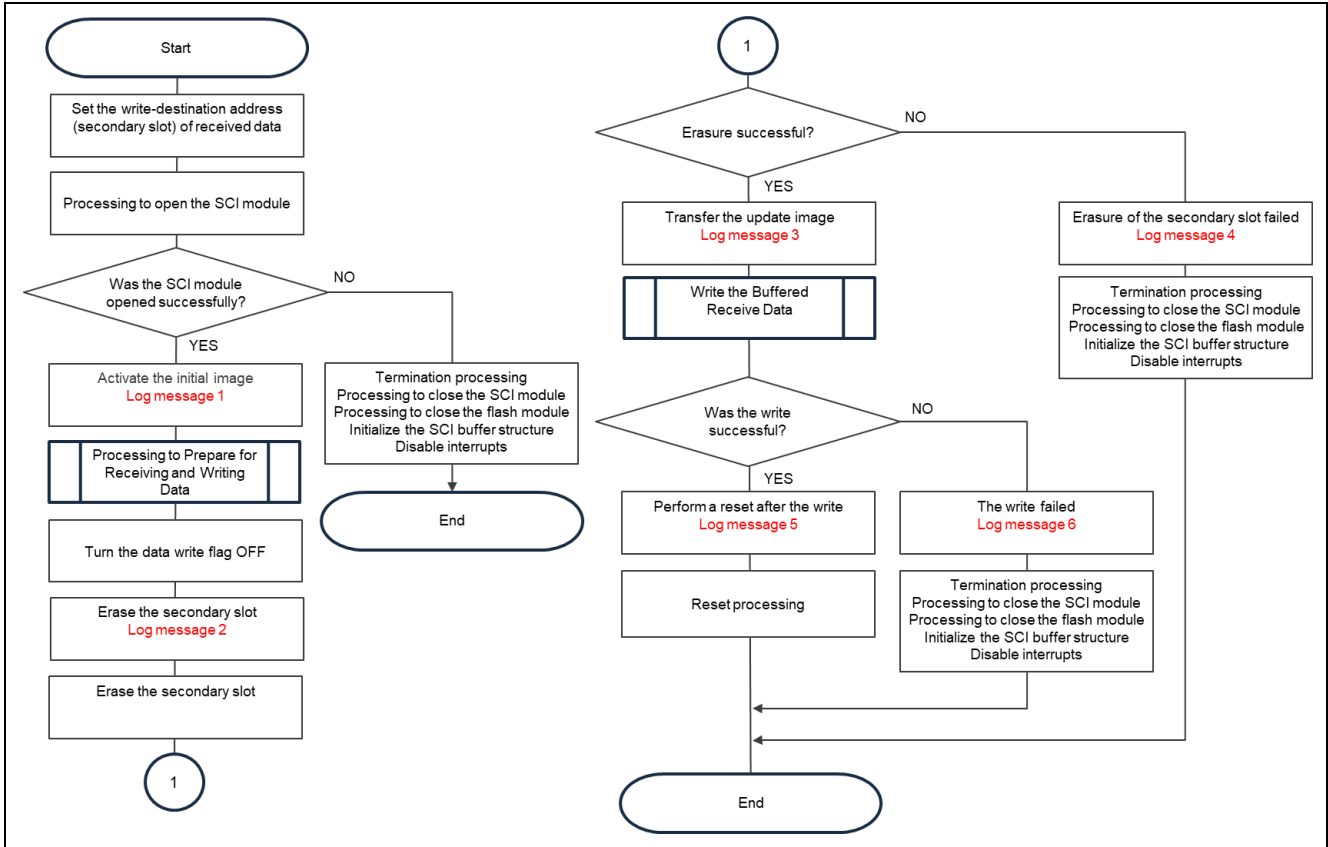


Figure 4-1 Processing Flowchart of the Initial Image (1/5)

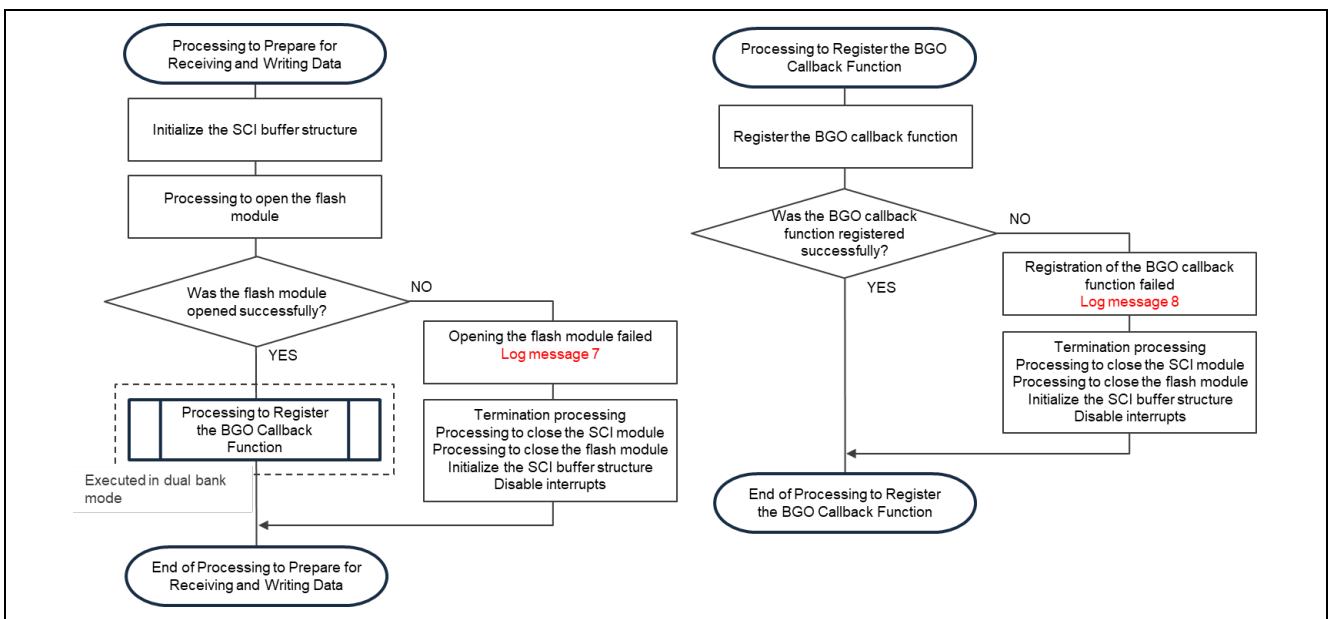


Figure 4-2 Processing Flowchart of the Initial Image (2/5)

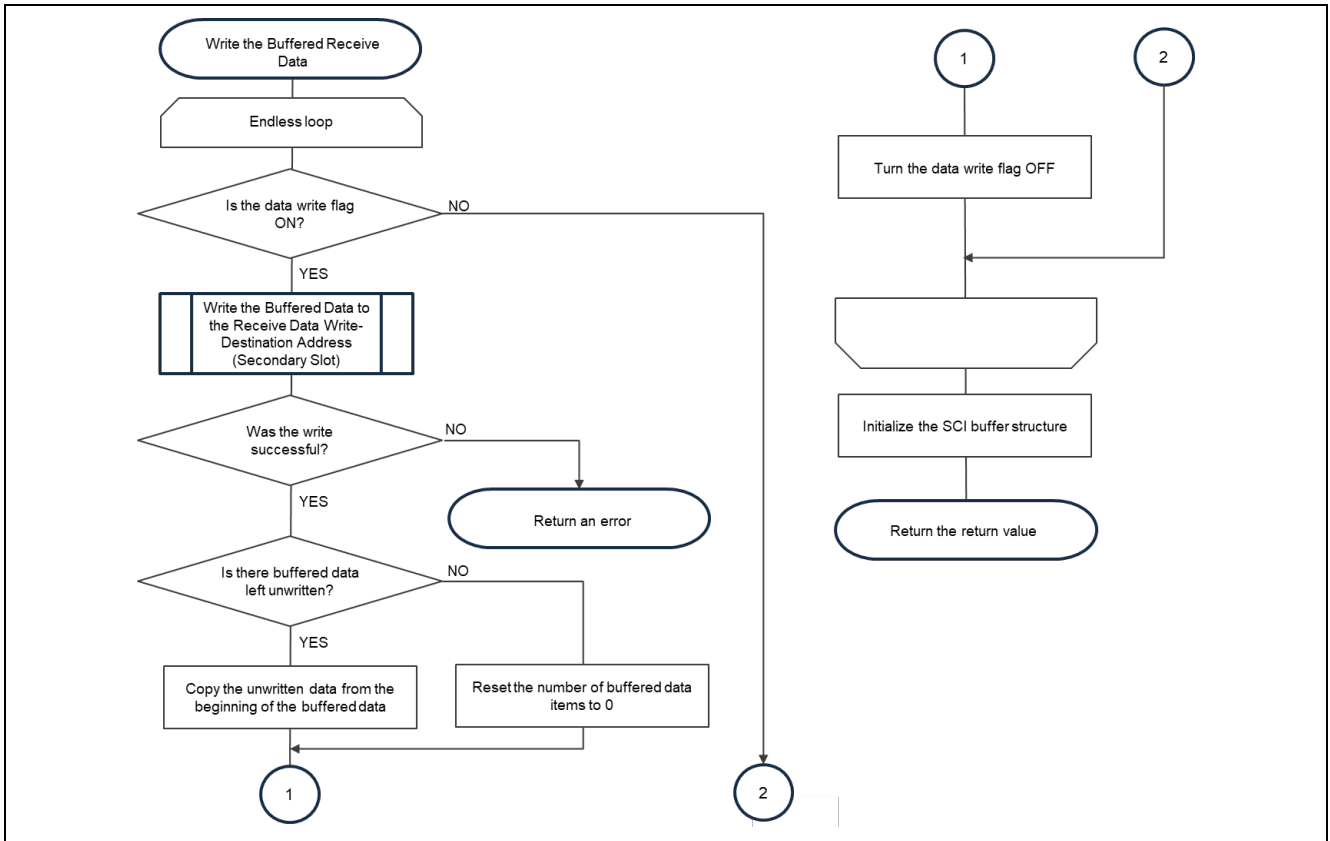


Figure 4-3 Processing Flowchart of the Initial Image (3/5)

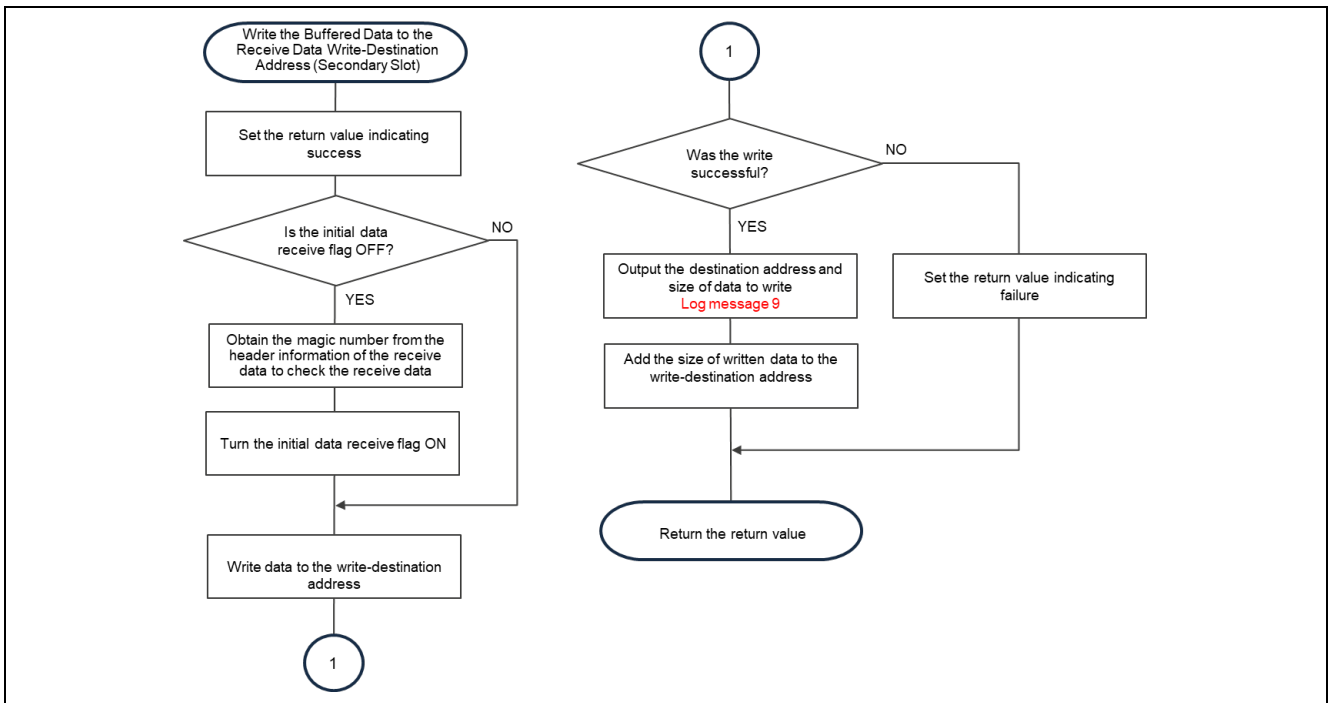


Figure 4-4 Processing Flowchart of the Initial Image (4/5)

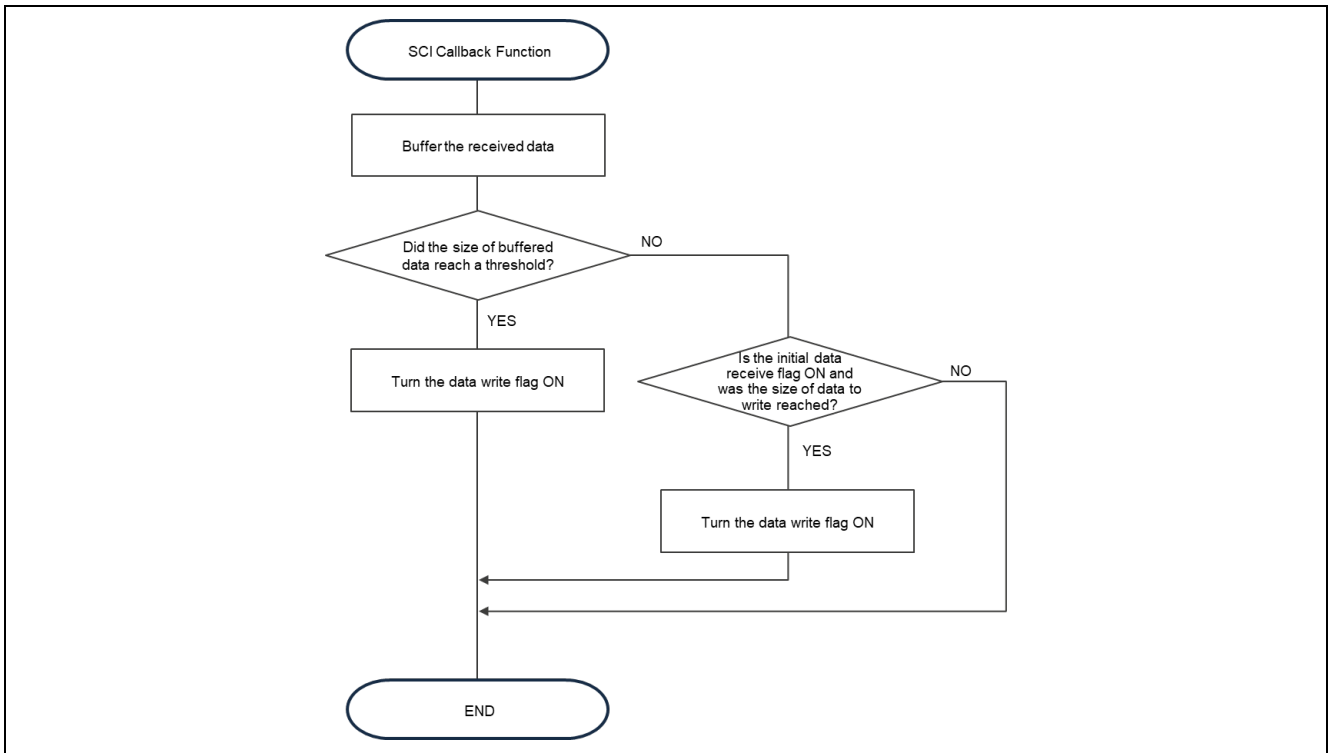


Figure 4-5 Processing Flowchart of the Initial Image (5/5)

Table 4-2 Information Output to the Log for the Initial Image

Output Timing	Output Information
When the initial image is activated (Log message 1)	----- Primary Slot Application Image Start (ver 1.0.0) -----
When the secondary slot is erased (Log message 2)	Erase the code flash of the Secondary Slot.
When the update image is transferred (Log message 3)	send user program (MCUboot image) via UART.
When erasure of the secondary slot fails (Log message 4)	Erase the code flash of the Secondary Slot failed.
When a reset is performed after writing data (Log message 5)	software reset...
When the writing of data fails (Log message 6)	Failed to write code flash.
When opening the flash module fails (Log message 7)	Flash driver open failure.
When registration of the BGO callback function fails (Log message 8)	BGO callback function set failure.
When the destination address and size of data to write are output (Log message 9)	Example: W 0xFFFF0000, 256 ... OK

4.2 Preparing the Operating Environment

Before you can execute the MCUboot demo project, you must install the necessary tools on your Windows PC.

4.2.1 Obtaining Imgtool

Imgtool adds a signature or other information, such as header information and trailer, to an image. It can generate a key pair for signature verification.

Imgtool has been partially modified for MCUboot FIT, so use `imgtool.py` stored in the package.

The version of Imgtool is v2.1.0.

For more information, refer to the information at the following URL about Imgtool of MCUboot:

<https://github.com/mcu-tools/mcuboot/blob/master/docs/imgtool.md>

4.2.2 Installing Terminal Software

Terminal software is required to perform an update by transferring a firmware image from the Windows PC to the target board via serial communication.

We have confirmed that the demo project can operate with Tera Term v5.2.

Specify the communication settings of the serial port as shown in Table 4-3.

Table 4-3 Communication Specifications

Item	Description
Communication mode	Asynchronous mode
Bit rate	115200 bps
Data length	8 bits
Parity	None
Stop bit	1 bit
Flow control	CTS/RTS

4.2.3 Installing the Python Runtime Environment

To use Imgtool included in MCUboot, the Python runtime environment is required.

We have confirmed that the demo project can operate with Python 3.11.4.

Because the encryption library of Python (`pycryptodome`) is used, after installing Python, first execute the following command to update the version of “pip”.

```
python -m pip install --upgrade pip
```

Install the dependencies using the following command.

```
pip3 install --user -r scripts/requirements.txt
```

Note: `scripts/requirements.txt` is included in `mcu-tools` in the MCUboot FIT package.

4.2.4 Installing the OpenSSL Runtime Environment

OpenSSL is used to generate the keys that are required to encrypt images. Download the OpenSSL installer from the following URL, and then install OpenSSL. No problems occur with the Light version.

We have confirmed that the demo project can operate with OpenSSL 3.4.1.

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

4.2.5 Installing a Flash Memory Writer

A flash memory writer is used when the MCUboot FIT module, initial image, key wrapping data, and other items are written to flash memory. This tool and details on how to install it are available at the URL shown below.

In the demo project, Renesas Flash Programmer v3.14.00 (RFP) is used as a flash memory writer.

[Renesas Flash Programmer \(Programming GUI\) | Renesas](#)

4.2.6 Installing Security Key Management Tool

Security Key Management Tool is used to generate key wrapping data. This tool and details on how to install it are available at the URL shown below.

[Security Key Management Tool | Renesas](#)

4.2.7 USB-to-Serial Conversion Board

Note that the on-board USB-to-serial conversion circuit of the RSK-RX65N/RSK-RX72N board is unavailable in linear mode. Therefore, if the target board is RSK-RX65N/RSK-RX72N, use the external USB-to-Serial conversion board introduced at the URL shown below.

If the target board is EK-RX261, the on-board USB-to-serial conversion circuit can be used.

For details on how to connect the conversion board to the target board, refer to “5.2 Operating Environment of the Demo Project”.

Note that the external USB-to-serial conversion board uses a Pmod USBUART (from DIGILENT).

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

4.3 Procedure for Executing the Demo Project

This section describes the procedure for executing the demo project.

Note that the address and other values shown in this section are applicable when the RSK-RX65N is used. For the values to be set for each product, refer to “5.2 Operating Environment of the Demo Project”.

The sample keys are included in the demo project. You can use it in the demo. However, you must generate a new key for the production version.

4.3.1 Key Injection

In the demo project, the TSIP or RSIP module is used to verify signatures and decrypt images. Before the TSIP or RSIP module can be used, you must use Hardware Unique Key (HUK) to wrap a key for wrapping the following keys and then inject the key into the device: the public key for signature verification, the key for image decryption, and the key for wrapping the image encryption key.

For details on key generation and injection, refer to the procedures described in the following subsections.

Note: “TSIP” appearing in the following subsections refers to TSIP or RSIP.

4.3.1.1 Generating Key Data by Using Security Key Management Tool

Use Security Key Management Tool (SKMT) to generate a User Factory Programming Key (UFPK) file. Then, use the file to wrap the key (AES-KeyWrap) for wrapping the public key for signature verification and the key for wrapping the image encryption key.

Also, from Renesas Key Wrap Service, obtain a W-UFPK, which is a UFPK file wrapped by using Hardware Root Key (HRK).

For details on SKMT, refer to the Renesas web page about Security Key Management Tool ([Security Key Management Tool | Renesas](#)) and “Security Key Management Tool User’s Manual (R20UT5349)”.

For details on Renesas Key Wrap Service (<https://dlm.renesas.com/keywrap>), refer to its FAQ and operation manual.

Use the following procedure to generate key data:

Step 1: Use SKMT to generate the UFPK file.

Step 2: Send the UFPK file generated in step 1 to Renesas Key Wrap Service.

Step 3: From Renesas Key Wrap Service, obtain a W-UFPK, which was created by wrapping the UFPK file by HRK.

Step 4: Use “imgtool” to generate a key pair for signature verification. (You will use “imgtool” again when generating images in sections 4.3.3.2 and 4.3.3.3, and when embedding a public key in section 4.3.2.)

Execute “imgtool.py” at mcu-tools\MCUboot\scripts in the Python environment.

ECDSA P-256:

```
python imgtool.py keygen -k ecc_sign_key_pair.pem -t ecdsa-p256
```

RSA 2048:

```
python imgtool.py keygen -k rsa_sign_key_pair.pem -t rsa-2048
```

Step 5: Generate an AES-KeyWrap key that will be used when you use a random number to wrap an image encryption key with OpenSSL.

```
openssl rand 32 -out AES-KeyWrap.key
```

Step 6: Use a random number to generate an image encryption key with OpenSSL. (The generated key will be used to generate an update image in section 4.3.3.3.)

```
openssl rand 32 -out AES-CTR.bin
```

Note: If you do not encrypt the image, generating the key in Steps 5 and 6 is unnecessary. Additionally, since the DirectXIP method cannot encrypt the image, generating the key in Steps 5 and 6 is also unnecessary in this case.

Step 7: SKMT wraps the signature verification key and AES-KeyWrap key using UFPK to generate a key data file. The key data file is output as a binary format file. The signature verification key is generated in Step 4, the AES-KeyWrap key in Step 5, and the UFPK uses the file generated in Step 1.

Step 8: Write the key data file generated in Step 7 to the code flash memory following the procedure in section 4.3.1.2.

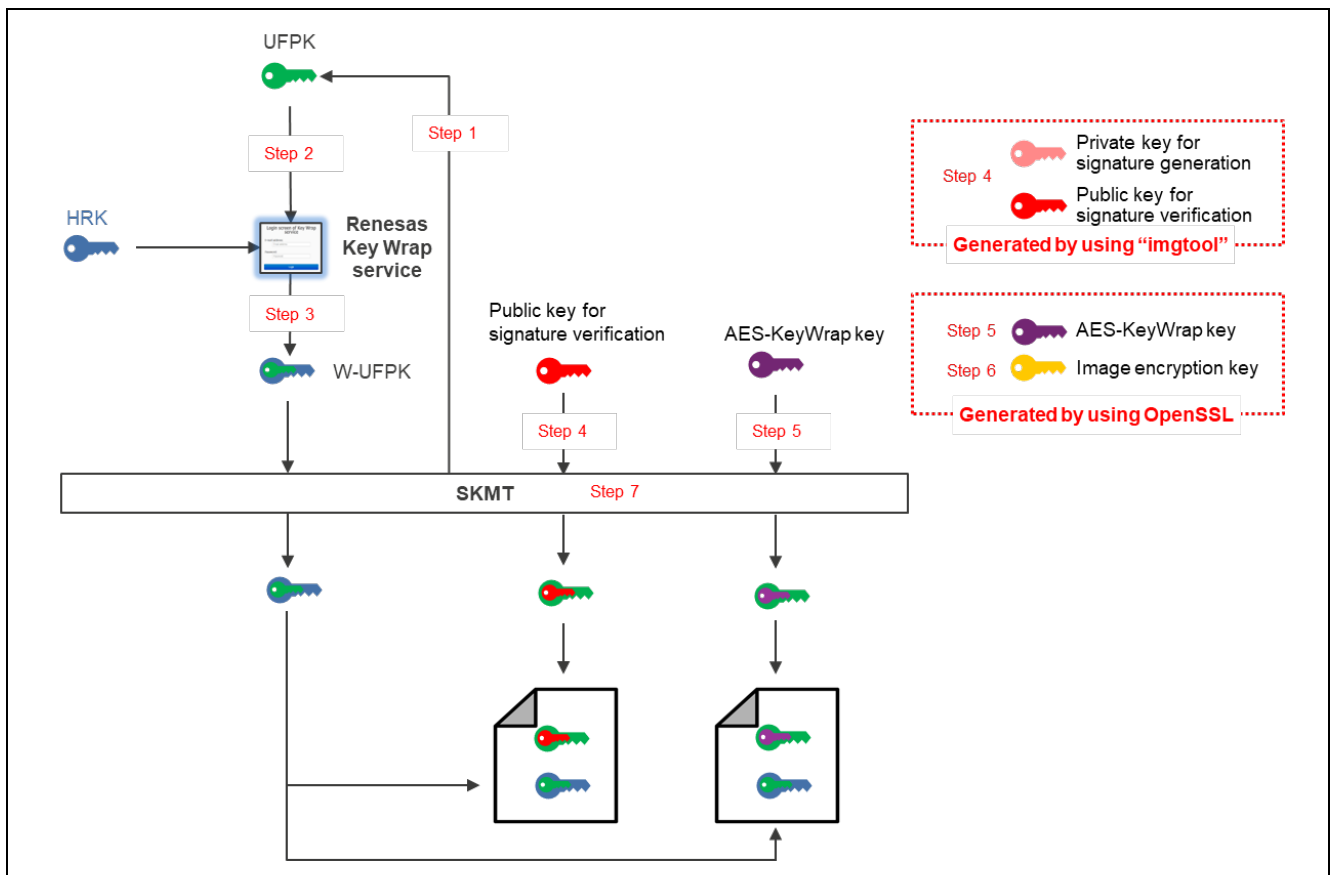


Figure 4-6 Generating Key Data by Using SKMT

4.3.1.2 Key Injection

In section 4.3.1.1, you generated wrapped key data (a W-UFPK, public key for signature verification, and AES-KeyWrap key). Here, you write the key data and a key injection program (provided as a sample program) to flash memory by using Renesas Flash Programmer (RFP).

You execute the key injection program to inject a key.

The key injection program retrieves the UFPK from the W-UFPK stored in the data flash, and then extracts the public key for signature verification and the AES Key-Wrap key wrapped with the UFPK. After that, it wraps these keys with the HUK to generate Wrapped Keys, which are written into the code flash.

Step 1: Prepare the key data file generated in binary format in section 4.3.1.1, and then use RFP to write the file to data flash memory (at 0x00100000).
The write destination is as follows.

Encrypt the image:

Public key for signature verification data : 0x00100000

AES-KeyWrap key data : 0x00100500

Do not encrypt the image, or use the DirectXIP Method:

Public key for signature verification data : 0x00100000

For details on how to use RFP, refer to “Renesas Flash Programmer flash memory programming software User’s Manual (R20UT5517)”.

Step 2: Use RFP to write the key injection program to code flash memory (at 0xFFFF2000).

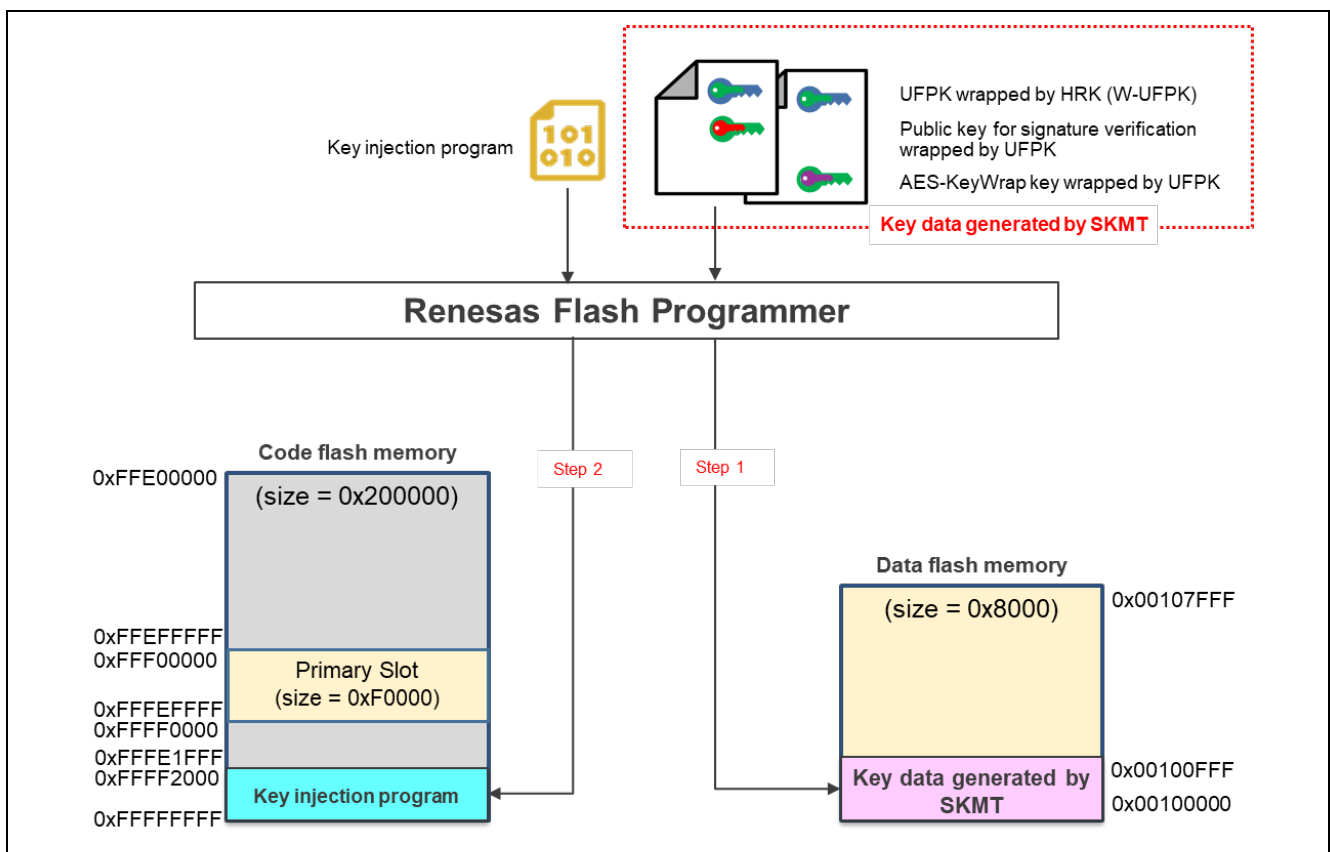


Figure 4-7 Preparation for Key Injection

Step 3: Reset the board, and then execute the key injection program.

For flash memory in linear mode:

Confirm that key data has been stored in data flash memory by the key injection program, and use the key data to wrap by HUK. Then, write the resulting wrapped key to code flash memory (at 0xFFFF0000).

For flash memory in dual mode:

Confirm that key data has been stored in data flash memory by the key injection program, and use the key data to wrap by HUK. Then, write the resulting wrapped key to code flash memory (at 0xFFFF0000 and 0xFFEF0000).

The key data stored in the data flash memory is erased by the injection program.

Note: For the RX261, when connecting the RFP thereafter, please set the following ID code.

ID code : 45FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

4.3.2 Embedding the Public Key for Signature Verification

In this section, you embed the public key for signature verification into “keys.c” in the demo project.

Step 1: Use “imgtool” to extract the public key for signature verification to be embedded into the bootloader.

“imgtool” extracts the public key data from the *.pem file generated in step 4 in section 4.3.1.1 and displays it on the console.

ECDSA P-256:

```
python imgtool.py getpub -k ecc_sign_key_pair.pem
```

RSA 2048:

```
python imgtool.py getpub -k rsa_sign_key_pair.pem
```

Output example:

```
/* Autogenerated by imgtool.py, do not edit. */
const unsigned char ecdsa_pub_key[] = {
    0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2a, 0x86,
    0x48, 0xce, 0x3d, 0x02, 0x01, 0x06, 0x08, 0x2a,
    0x86, 0x48, 0xce, 0x3d, 0x03, 0x03, 0x01, 0x07, 0x03,
    0x42, 0x00, 0x04, 0x53, 0x5a, 0x25, 0x70, 0xe6,
    0xa4, 0xd1, 0x0b, 0xaa, 0x25, 0x52, 0x14, 0xf7,
    0xa2, 0x69, 0x3b, 0xc5, 0x02, 0xe0, 0xe7, 0x96,
    0x0c, 0xa8, 0x59, 0x5f, 0x28, 0x04, 0x95, 0x52,
    0x05, 0x3d, 0xea, 0x46, 0x75, 0xd6, 0xa9, 0xd5,
    0x0b, 0x99, 0x5d, 0x1a, 0x2f, 0x10, 0x31, 0x01,
    0xc9, 0x1e, 0x67, 0x42, 0x6d, 0xea, 0xec, 0x77,
    0x3d, 0x23, 0xd4, 0x23, 0x75, 0x28, 0x67, 0x29,
    0xd1, 0x4f, 0x4a,
};
const unsigned int ecdsa_pub_key_len = 91;
```

Step 2: Embed the data displayed on the console into “keys.c” in the demo project.

```
#include <bootutil/sign_key.h>

const unsigned char root_pub_der[] = {
    /* embed signature verification public key generated in DER format */
};
const unsigned int root_pub_der_len = 0; /* embed "len" */

const struct bootutil_key bootutil_keys[] = {
    {
        .key = root_pub_der,
        .len = &root_pub_der_len,
    },
};
const int bootutil_key_cnt = 1;
```

4.3.3 Preparing the Images for the Demo Project

In this section, you prepare the demo project components (bootloader, initial image, and update image). Note that the addresses of code flash memory at which to store images and parts of the procedure differ depending on the update method.

4.3.3.1 Generating a Bootloader Image

Build the bootloader of the demo project to generate an image in binary format.

4.3.3.2 Generating the Initial Image

Build the initial image of the demo project to generate an image in binary format, and then use “imgtool” to add information with which MCUboot can manage the image.

Step 1: Build the initial image of the demo project (application program that receives the update image via UART communication and writes it to the secondary slot) to generate an image in binary format.

Step 2: Use “imgtool” to add the trailer information that will be managed by MCUboot to the image generated in step 1. The resulting data becomes the new initial image.

Example of the option settings for generating the initial image:

```
imgtool.py sign --version 1.0.0 --header-size 0x200 --align 128
--max-align 128 --slot-size 0xF0000 --max-sectors 16 --confirm
--pad-header --key ecc_sign_key_pair.pem
input_image.bin input_image.bin.ecc_sign
```

--version #.#.#: Specify the version number.

--slot-size 0x#####: Specify the slot size.

--key #####.pem: Set the key pair for signature verification generated in step 4 in section 4.3.1.1.

#####.bin: Set the binary file of the image generated in step 1.

#####.bin.ecc_sign: The initial image is output.

Note: The initial image cannot be encrypted.

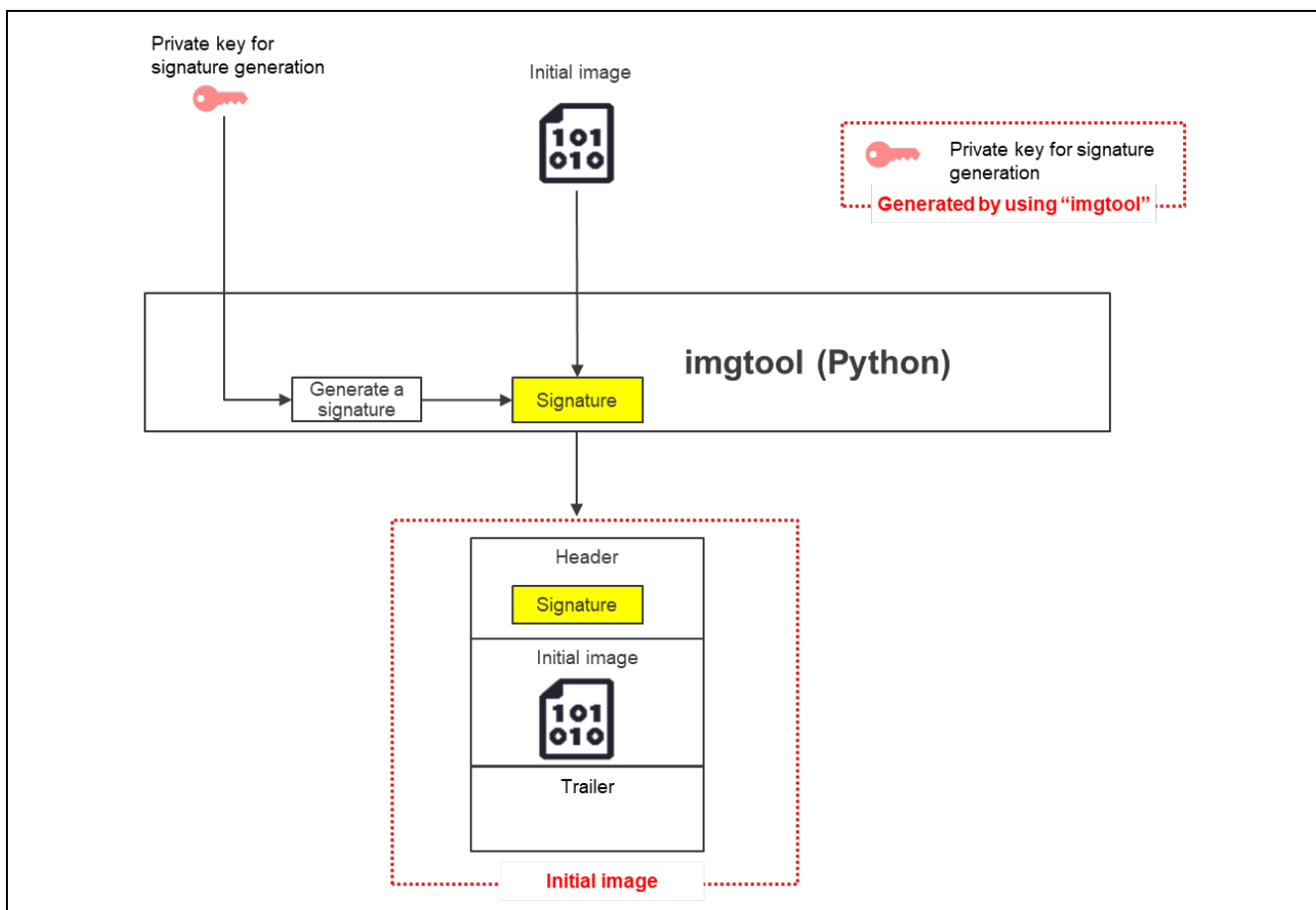


Figure 4-8 Generating the Initial Image

4.3.3.3 Generating an Update Image

In the demo project, you create the update image by reusing the initial image.

When you use “imgtool” to add information, specify the “--version” option with a version number higher than the version number of the initial image.

If you want to encrypt the update image, also specify the encryption key (‘-kw--enckey’ option) and the encryption key for wrapping(‘-kw--kek’ option) when generating the update image.

Note that the DirectXIP method does not support image encryption.

Step 1: Build the initial image of the demo project to generate an image in binary format.

Step 2: Use “imgtool” to add the trailer information that will be managed by MCUboot to the image generated in step 1.

Example of the option settings for generating an encrypted update image:

```
imgtool.py sign --version 1.1.0 --header-size 0x200 --align 128
--max-align 128 --slot-size 0xF0000 --max-sectors 16 -confirm
--pad-header --key ecc_sign_key_pair.pem
-kw--enckey AES-CTR.bin -kw--kek AES-KeyWrap.key
input_image.bin input_image.bin.ecc_sign.enc
```

--version #.#.#: Specify the version number. (Specify a version number higher than the version number of the initial image, so that the generated image can be used as the update image.)

--slot-size 0x#####: Specify the slot size.

--key #####.pem: Set the key pair for signature verification generated in step 4 in section 4.3.1.1.

-kw--enckey #####.bin: Set the image encryption key generated in step 6 in section 4.3.1.1.

-kw--kek #####.key: Set the key to wrap the image encryption key generated in step 5 in section 4.3.1.1.

#####.bin: Set the binary file of the image generated in step 1.

#####.bin.ecc_sign.enc: An encrypted update image is output.

Example of the option settings for generating an unencrypted update image:

```
imgtool.py sign --version 1.1.0 --header-size 0x200 --align 128
--max-align 128 --slot-size 0xF0000 --max-sectors 16 -confirm
--pad-header --key ecc_sign_key_pair.pem
input image.bin input image.bin.ecc sign
```

The option settings for generating an unencrypted update image are the same as those for generating the initial image except that the value of the “--version” option is changed to a higher version number.

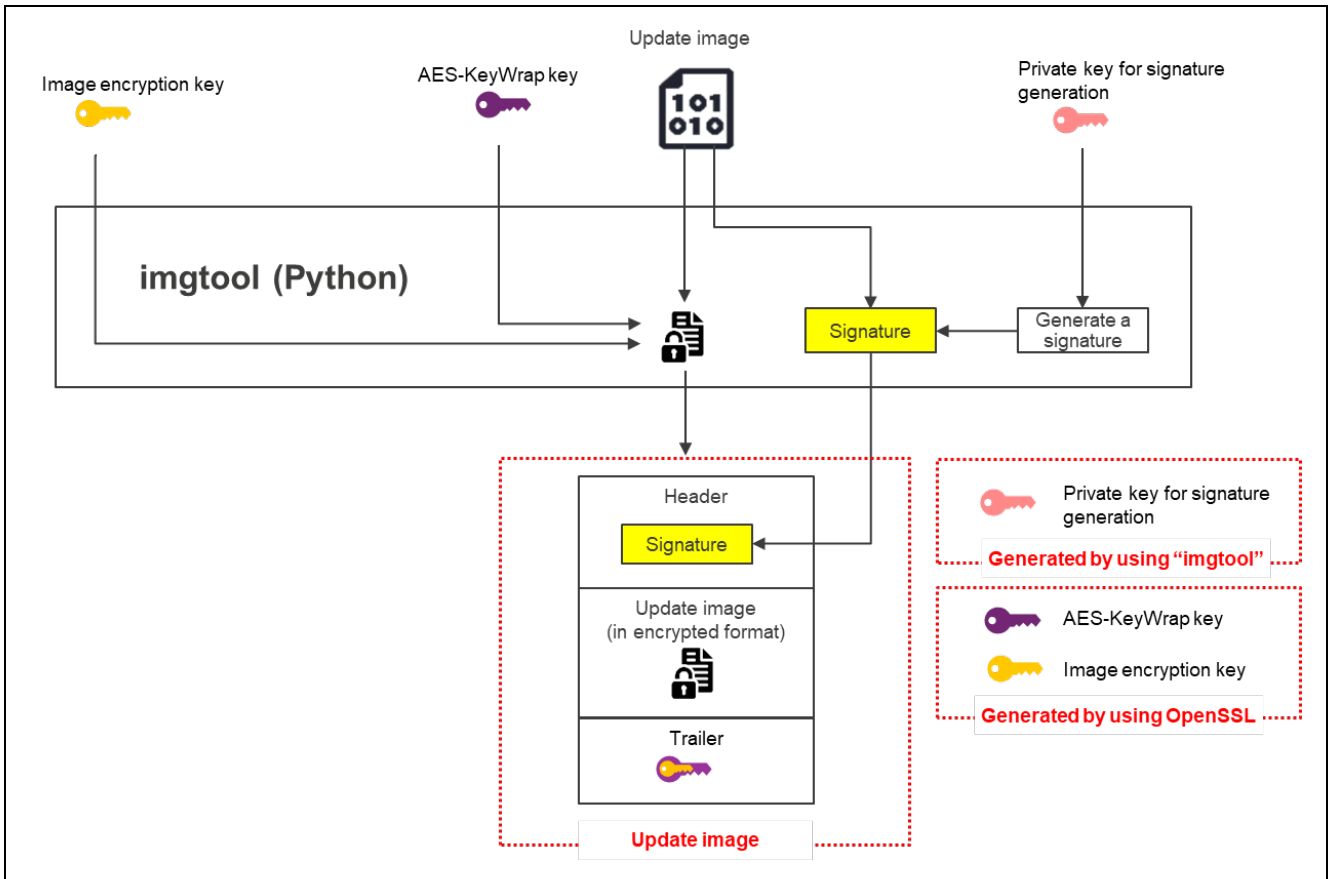


Figure 4-9 Generating an Update Image (in Encrypted Format)

4.3.4 Programming the Demo Project

In this section, you use RFP to write the images to be used for the demo project that was generated in section 4.3.3.

In the procedure described below, you use RFP to erase the flash memory in step 1 and write code in step 2 or later. When you write code after erasure, specify multiple files so that the processing will be completed at once.

Note that the write-destination addresses differ depending on the update method and the configuration of the flash memory. For the write-destination addresses, refer to the memory map for the method you use in “5.2 Operating Environment of the Demo Project”.

In the case of linear mode:

Step 1: Use RFP to erase the code flash memory areas other than the HUK-wrapped key data generated in 4.3.1.3.

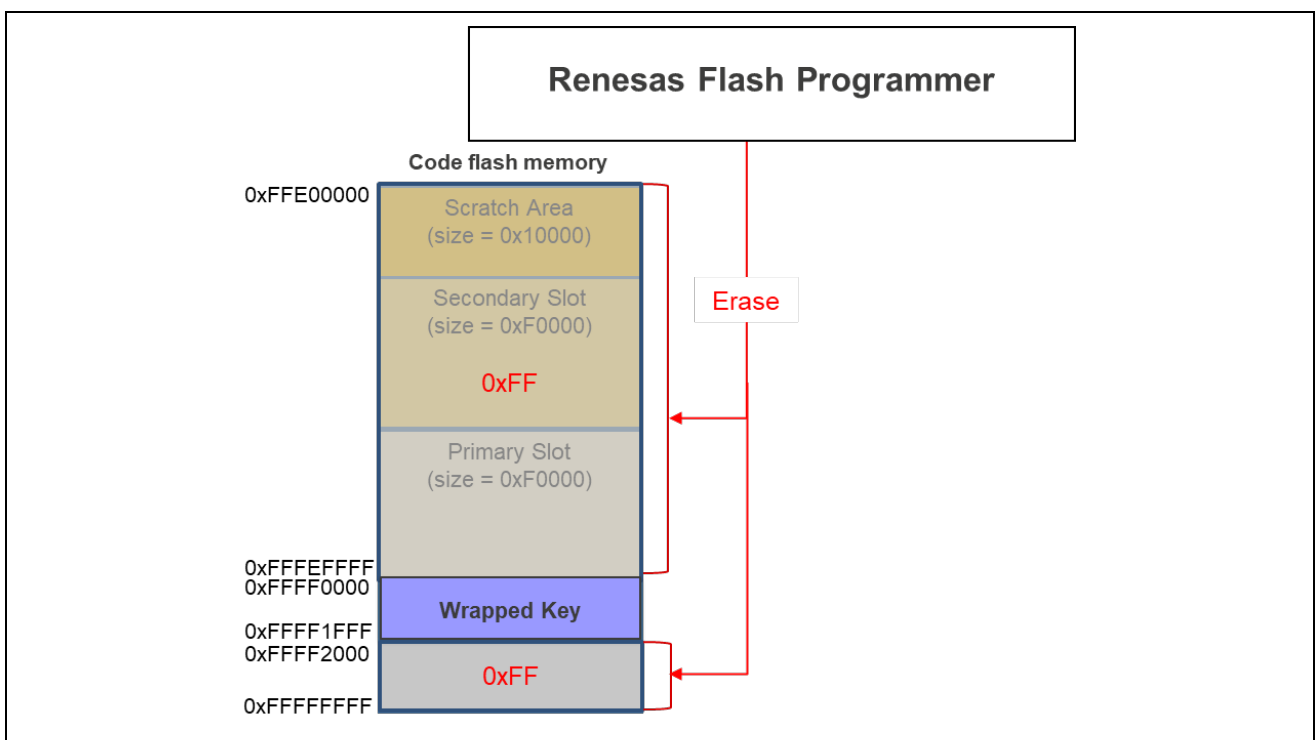


Figure 4-10 Erasing the Write Destinations (in the Case of Linear Mode)

Step 2: Use RFP to write the bootloader (MCUBoot) generated in section 4.3.3.1 to the Bootloader area, and the initial image generated in section 4.3.3.2 to the primary slot.

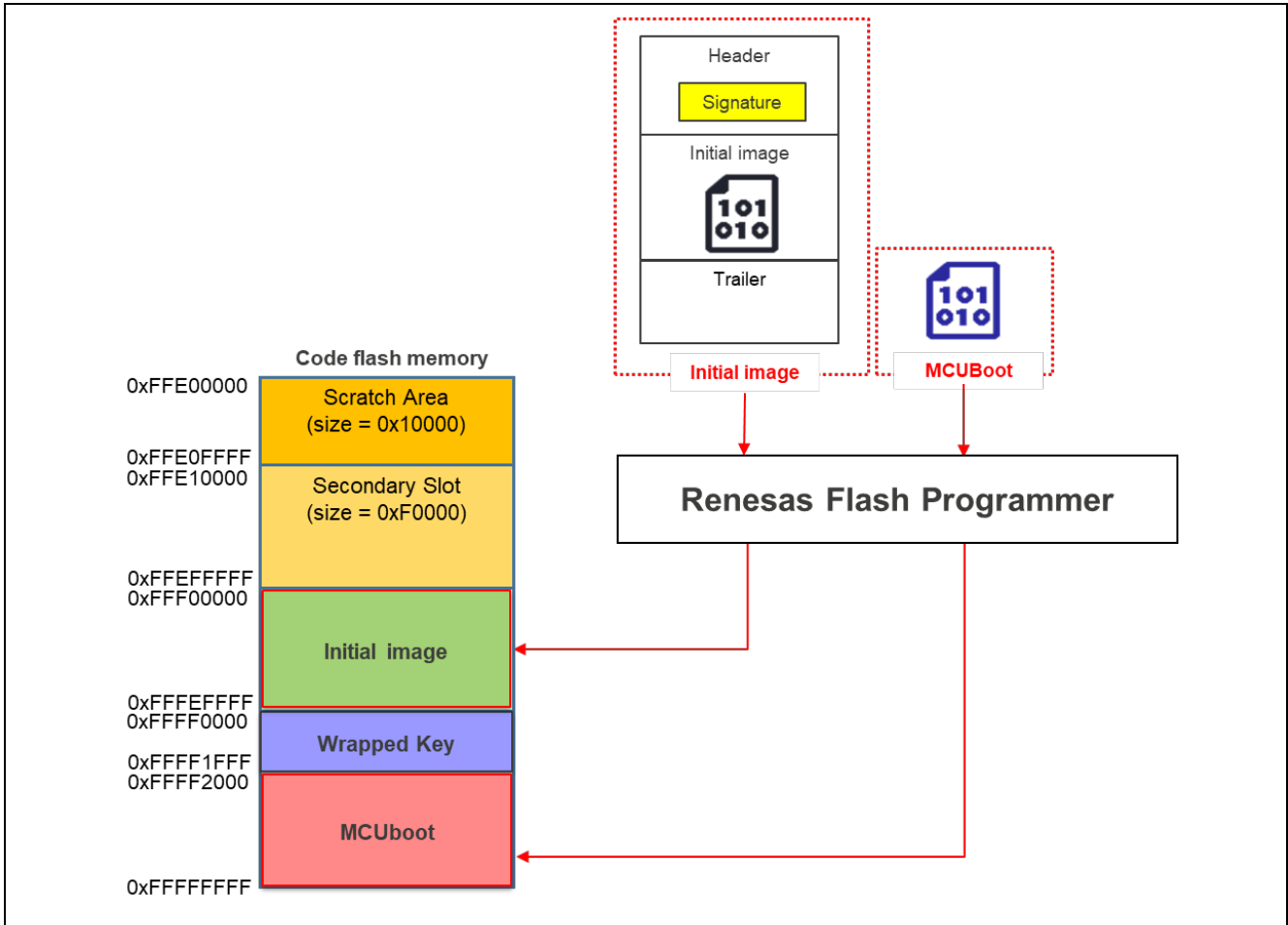


Figure 4-11 Programming the Demo Project (in the Case of Linear Mode)

In the case of dual mode:

Step 1: Use RFP to erase the code flash memory areas other than the HUK-wrapped key data generated in 4.3.1.3.

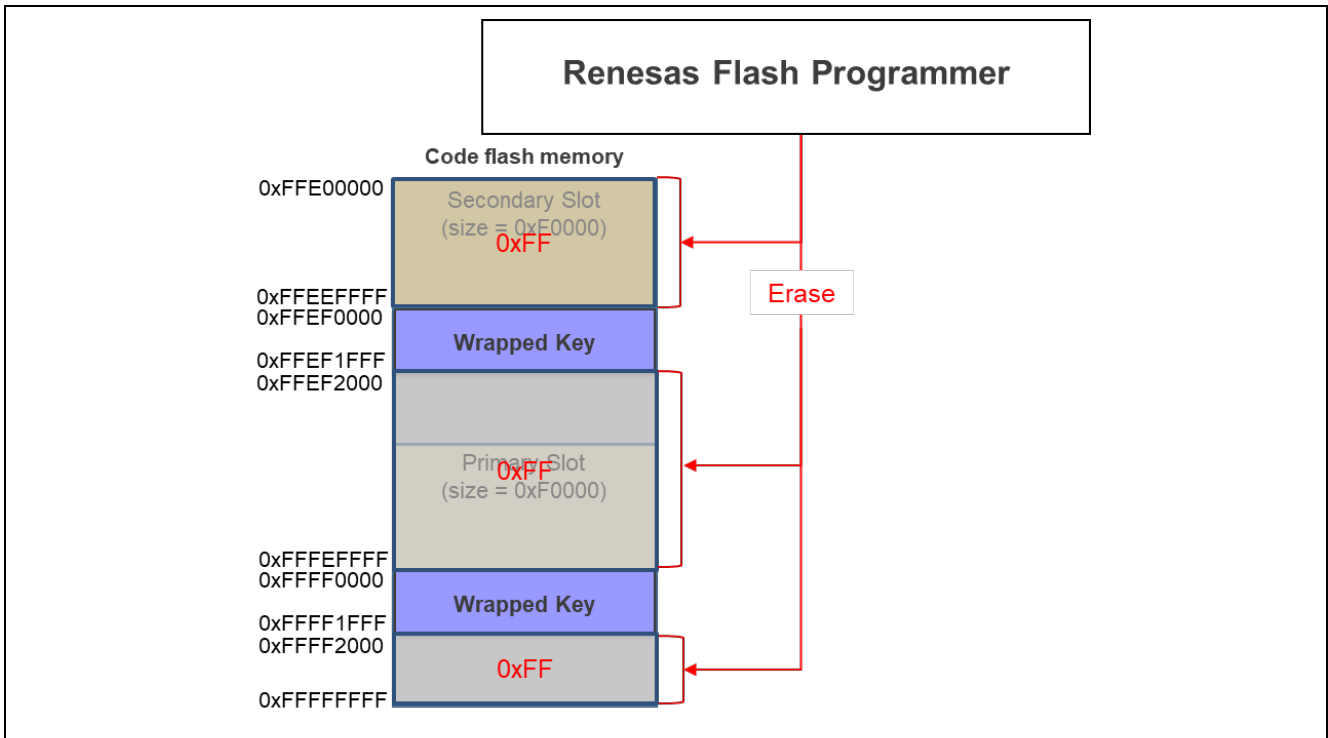


Figure 4-12 Erasing the Write Destinations (in the Case of Dual Mode)

Step 2: Use RFP to write the bootloader (MCUboot) generated in section 4.3.3.1 to the Bootloader areas in Banks 0 and 1, and the initial image generated in section 4.3.3.2 to the primary slot.

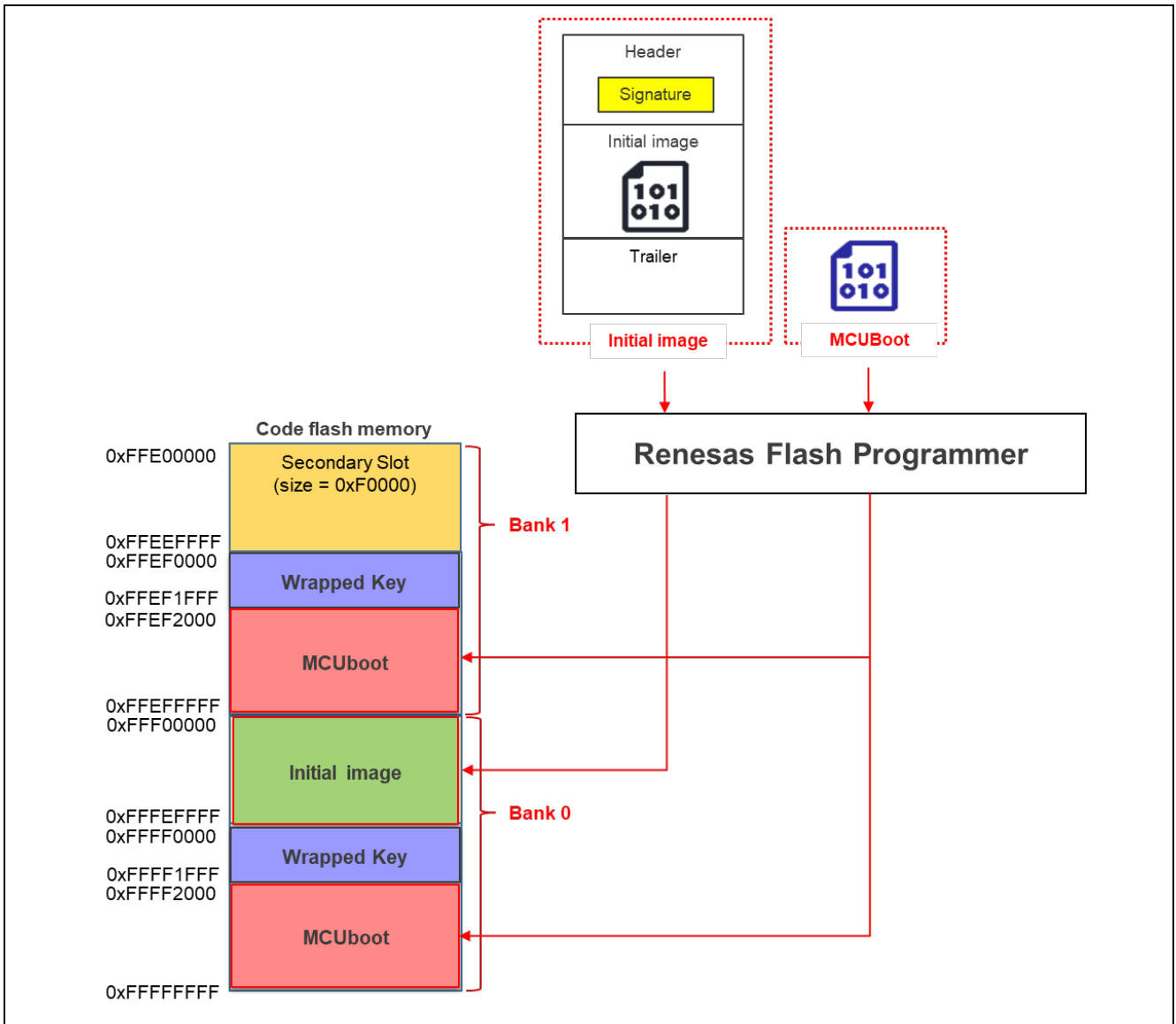


Figure 4-13 Programming the Demo Project (in the Case of Dual Mode)

4.3.5 Executing the Demo Project

In this section, you execute the demo project that was programmed in section 4.3.4.

When you start the demo project, the bootloader (MCUboot) starts and activates the initial image written in the primary slot. In this demo application, the update image received via terminal software is written to the secondary slot.

After the writing is completed, the software is reset, and the bootloader (MCUboot) is started again.

Use the following procedure to execute the demo project:

1. Connect the hardware components by referring to “5.2.2 Environment Used for Verifying Operation of the RX65N”.
2. Start the terminal software on the PC, and then select the serial COM port and specify the connection settings.
3. Turn on the power of the target board. The bootloader (MCUboot) starts, and the initial image written in the primary slot is activated.

```
Update start
[INF] Primary slot: version=1.0.0+0
[INF] Image 0 Secondary slot: Image not found
[INF] Image 0 loaded from the primary slot
-----
Primary Slot Application Image Start (ver 1.0.0)
-----
Erase the code flash of the Secondary slot.
```

4. When the initial image begins to wait for reception of the update image, send the update image from the terminal software.

While the received update image is being written to the secondary slot, the following messages are output:

```
send user program (MCUboot image) via UART.
W 0xffe10000, 512 ... OK
W 0xffe10200, 512 ... OK
...
W 0xffeffc00, 512 ... OK
W 0xffeffe00, 512 ... OK
```

5. When the writing of the update image is completed, the software is reset.

```
software reset..
```

6. The bootloader (MCUboot) starts again and performs an update according to the specified update mode. When the update is completed, the bootloader activates the updated image.

```
Update start
[INF] Swap type: perm
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] Copying the secondary slot to the primary slot: 0xf0000 bytes
-----
Primary Slot Application Image Start (ver 1.1.0)
-----
Erase the code flash of the Secondary slot.
send user program (MCUboot image) via UART.
```

5. Appendix

5.1 Environment Used for Verifying Operation

This appendix shows the environment used for verifying the operation of the MCUboot FIT module.

Table 5-1 Environment Used for Verifying Operation (CC-RX)

Item	Description
Integrated development environment	Renesas Electronics e ² studio 2025-04
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.07.00 Compiler options: The following option was used in addition to the default settings of the integrated development environment: -lang = c99
Endianness	Little endian
Revision number of the module	Rev.1.02
Boards used	Evaluation Kit for RX261 (Product No.: RTK5EK2610SxxxxxBE) Renesas Starter Kit+ for RX65N (Product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX671 (Product No.: RTK55671EHSxxxxxBE) Renesas Starter Kit+ for RX72M (Product No.: RTK5572MNHSxxxxxBE) Renesas Starter Kit+ for RX72N (Product No.: RTK5572NNxxxxxxxBE)

Table 5-2 Environment Used for Verifying Operation (GCC)

Item	Description
Integrated development environment	Renesas Electronics e ² studio 2025-04
C compiler	GCC for Renesas RX 8.3.0.202411 Compiler options: The following option was used in addition to the default settings of the integrated development environment: -std=gnu99
Endianness	Little endian
Revision number of the module	Rev.1.02
Boards used	Evaluation Kit for RX261 (Product No.: RTK5EK2610SxxxxxBE) Renesas Starter Kit+ for RX65N (Product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX671 (Product No.: RTK55671EHSxxxxxBE) Renesas Starter Kit+ for RX72M (Product No.: RTK5572MNHSxxxxxBE) Renesas Starter Kit+ for RX72N (Product No.: RTK5572NNxxxxxxxBE)

Table 5-3 Environment Used for Verifying Operation (IAR)

Item	Description
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RX 5.10.1 RX Smart Configurator V2.25.0
C compiler	IAR Systems IAR C/C++ Compiler for Renesas RX 5.10.1 Compiler options: The default settings of the integrated development environment were used.
Endianness	Little endian
Revision number of the module	Rev.1.02
Boards used	Evaluation Kit for RX261 (Product No.: RTK5EK2610SxxxxxBE) Renesas Starter Kit+ for RX65N (Product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX671 (Product No.: RTK55671EHSxxxxxBE) Renesas Starter Kit+ for RX72M (Product No.: RTK5572MNHSxxxxxBE) Renesas Starter Kit+ for RX72N (Product No.: RTK5572NNxxxxxxxBE)

The following lists the version numbers of the FIT modules used in the demo project when operation of MCUboot was verified.

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

Table 5-4 List of Version Numbers of the FIT Modules (CC-RX)

Device	Project	r_bsp	r_flash_rx	r_tsip	r_rsip_protected_rx	r_sci_rx	r_byteq	rm_mcu boot
RX65N, RX66N, RX671, RX72M, RX72N	application_primary	7.53	5.22	1.22	—	5.41	2.11	1.02
	boot_loader	7.53	5.22	1.22	—	5.41	2.11	1.02
	key_injection	7.53	5.22	1.22	—	5.41	2.11	1.02
RX261	application_primary	7.53	5.22	—	1.00	5.41	2.11	1.02
	boot_loader	7.53	5.22	—	1.00	5.41	2.11	1.02
	key_injection	7.53	5.22	—	1.00	5.41	2.11	1.02

(2) Environment that used GCC for Renesas RX

Table 5-5 List of the Version Numbers of the FIT Modules (GCC)

Device	Project	r_bsp	r_flash_rx	r_tsip	r_rsip_protected_rx	r_sci_rx	r_byteq	rm_mcu boot
RX65N, RX66N, RX671, RX72M, RX72N	application_primary	7.53	5.22	1.22	—	5.41	2.11	1.02
	boot_loader	7.53	5.22	1.22	—	5.41	2.11	1.02
	key_injection	7.53	5.22	1.22	—	5.41	2.11	1.02
RX261	application_primary	7.53	5.22	—	1.00	5.41	2.11	1.02
	boot_loader	7.53	5.22	—	1.00	5.41	2.11	1.02
	key_injection	7.53	5.22	—	1.00	5.41	2.11	1.02

(3) Environment that used IAR C/C++ Compiler for RX

Table 5-6 List of Version Numbers of the FIT Modules (IAR)

Device	Project	r_bsp	r_flash_rx	r_tsip	r_rsip_protected_rx	r_sci_rx	r_byteq	rm_mcu boot
RX65N, RX66N, RX671, RX72M, RX72N	application_primary	7.53	5.22	1.22	—	5.41	2.11	1.02
	boot_loader	7.53	5.22	1.22	—	5.41	2.11	1.02
	key_injection	7.53	5.22	1.22	—	5.41	2.11	1.02
RX261	application_primary	7.53	5.22	—	1.00	5.41	2.11	1.02
	boot_loader	7.53	5.22	—	1.00	5.41	2.11	1.02
	key_injection	7.53	5.22	—	1.00	5.41	2.11	1.02

5.2 Operating Environment of the Demo Project

The demo project supports multiple products. The settings to be specified when using the demo project differ depending on the product. This section shows these differences.

5.2.1 Environment Used for Verifying Operation of the RX261

5.2.1.1 Information on Hardware Component Connections

The following shows the information on hardware component connections for the EK-RX261.

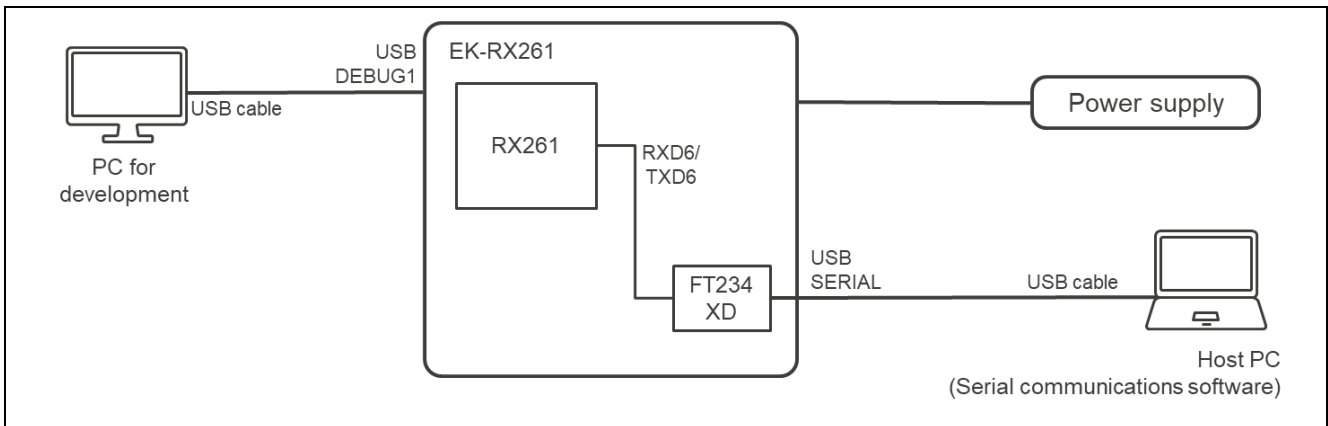


Figure 5-1 Hardware Component Connection Diagram for the EK-RX261

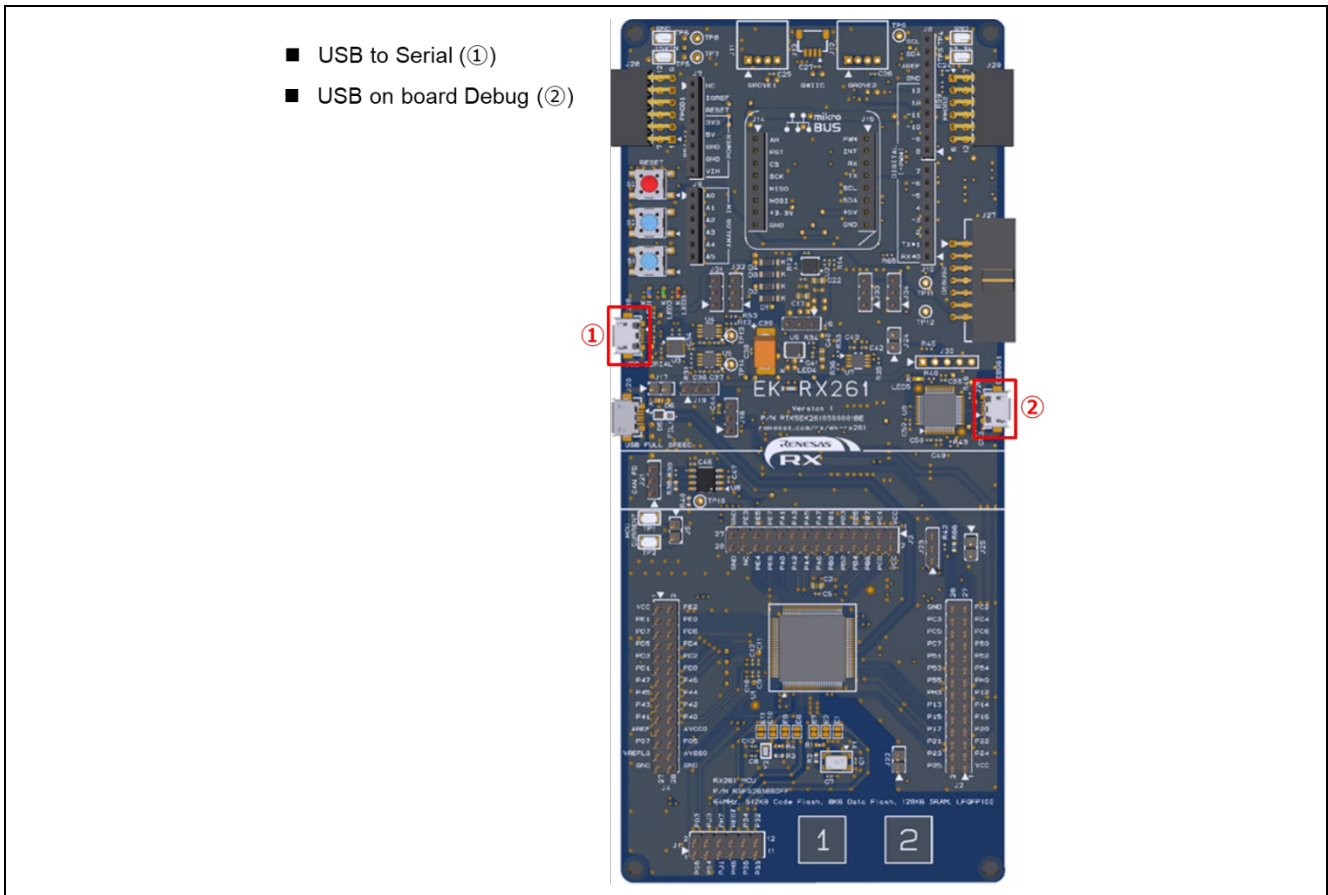


Figure 5-2 Connection Information of the EK-RX261 Board

5.2.1.2 Memory Allocation and Configuration Option Settings

The following figure shows the memory allocation of the demo project for the EK-RX261.

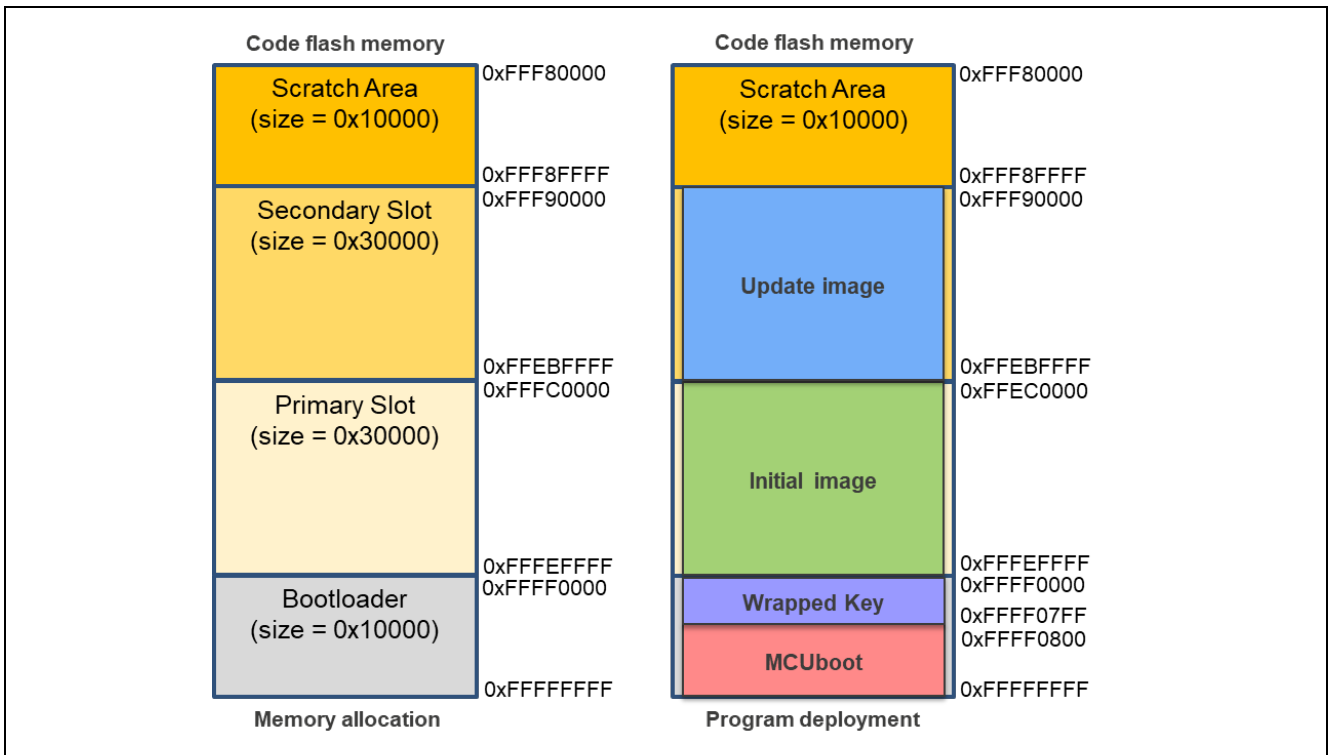


Figure 5-3 Memory Map of the Demo Project for the EK-RX261

Table 5-7 Configuration Option Settings of the Demo Project for the EK-RX261

Configuration options in rm_mcuboot_config.h	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	Select a number in the range from 0 to 3.
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x30000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.2 Environment Used for Verifying Operation of the RX65N

5.2.2.1 Information on Hardware Component Connections for an Update in Linear Mode

The following shows the information on hardware component connections for the RSK-RX65N in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

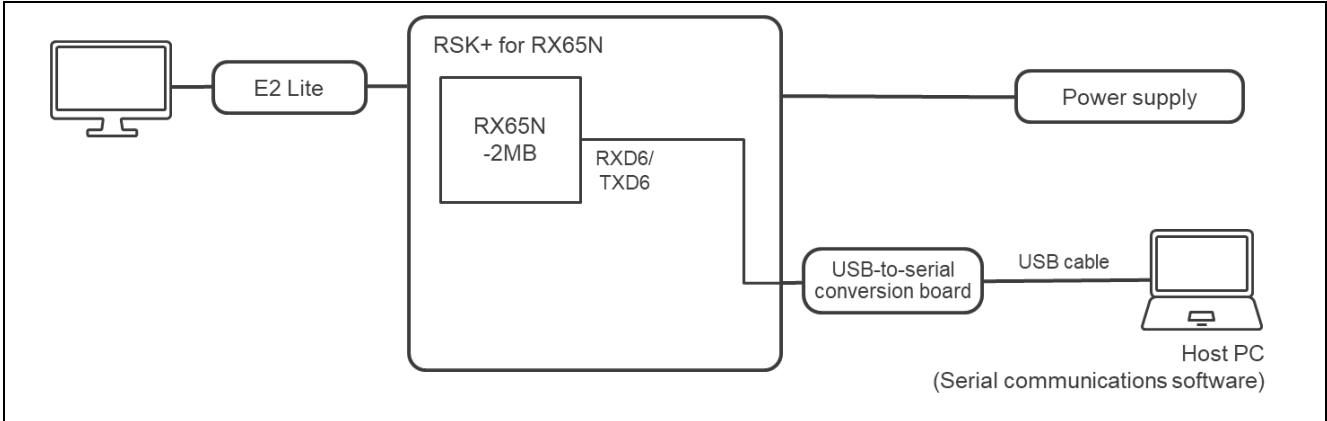


Figure 5-4 Hardware Component Connection Diagram for the RSK-RX65N When the Update Method Uses Linear Mode

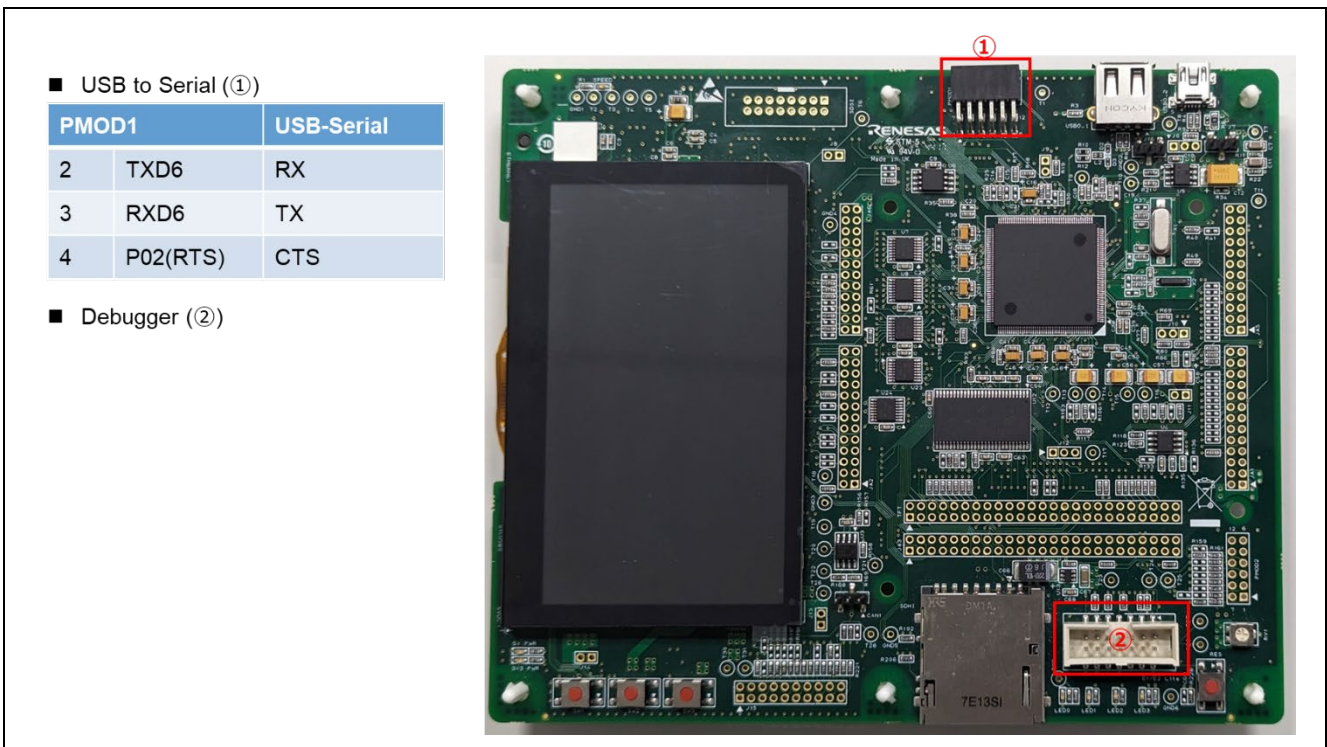


Figure 5-5 Information on Hardware Component Connections for the RSK-RX65N When the Update Method Uses Linear Mode

5.2.2.2 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

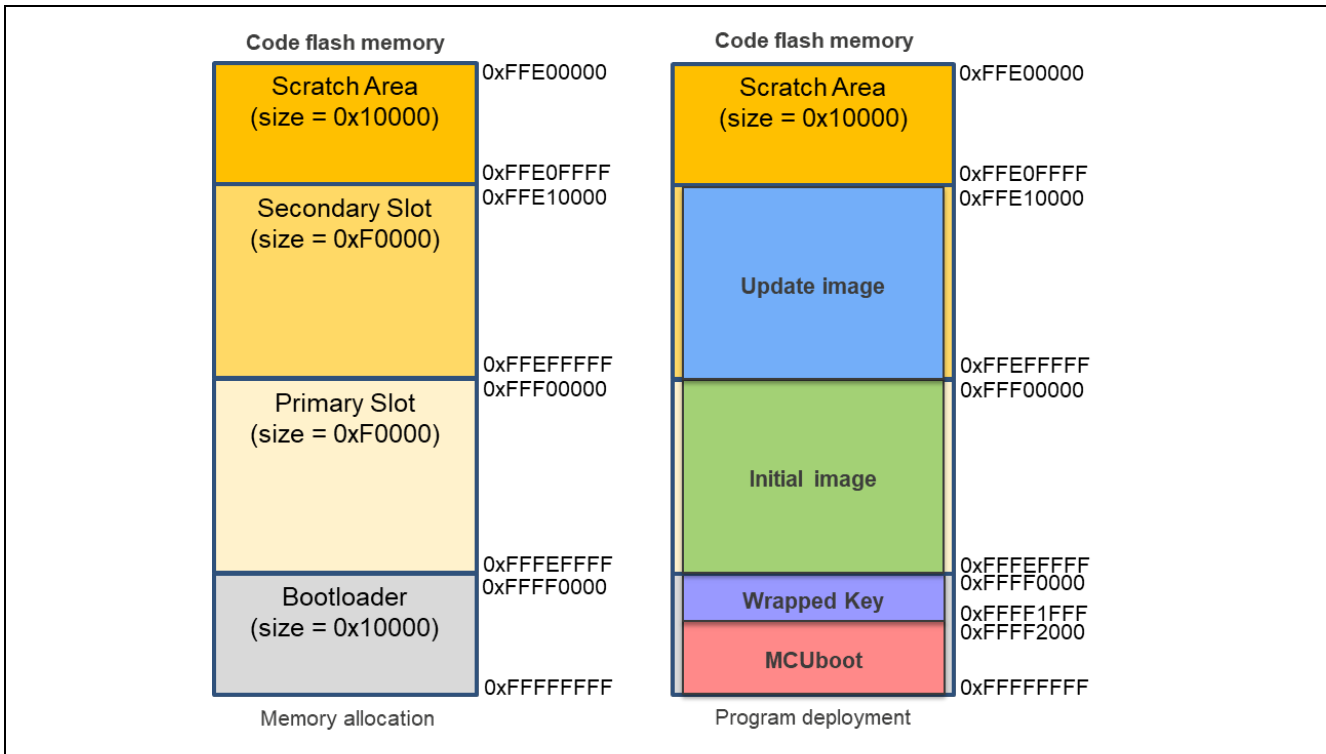


Figure 5-6 Memory Map of the Demo Project for the RSK-RX65N (2 MB) in the Case Where the Update Method Uses Linear Mode

Table 5-8 Configuration Option Settings for the RSK-RX65N (2 MB) in the Case Where the Update Method Uses Linear Mode

Configuration options in rm_mcuboot_config.h	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	Select a number in the range from 0 to 3.
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.2.3 Information on Hardware Component Connections for an Update in Dual Mode

The following shows the information on hardware component connections in the case where the update method is DirectXIP and the flash memory is in dual mode.

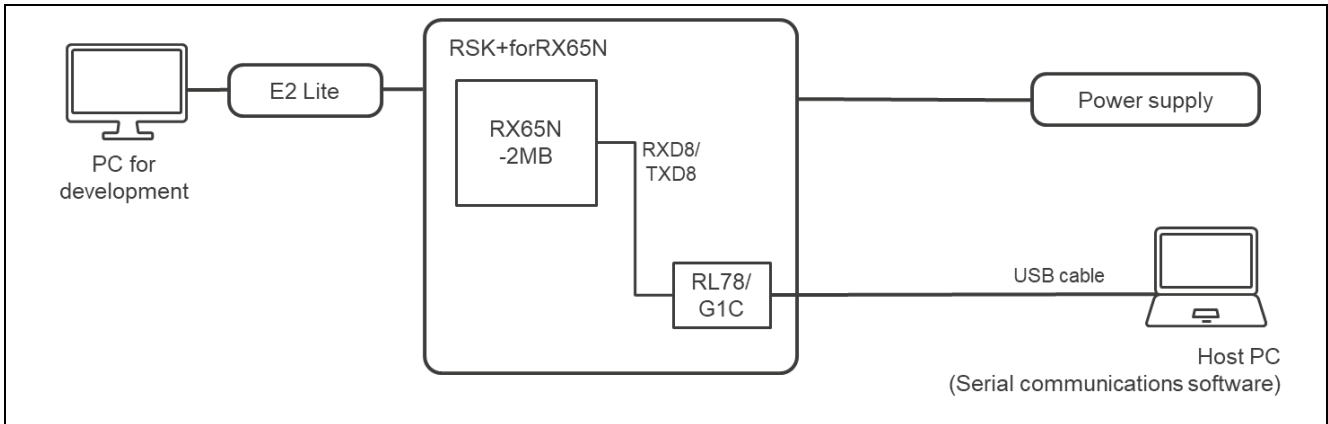


Figure 5-7 Hardware Component Connection Diagram for the RSK-RX65N When the Update Method Uses Dual Mode

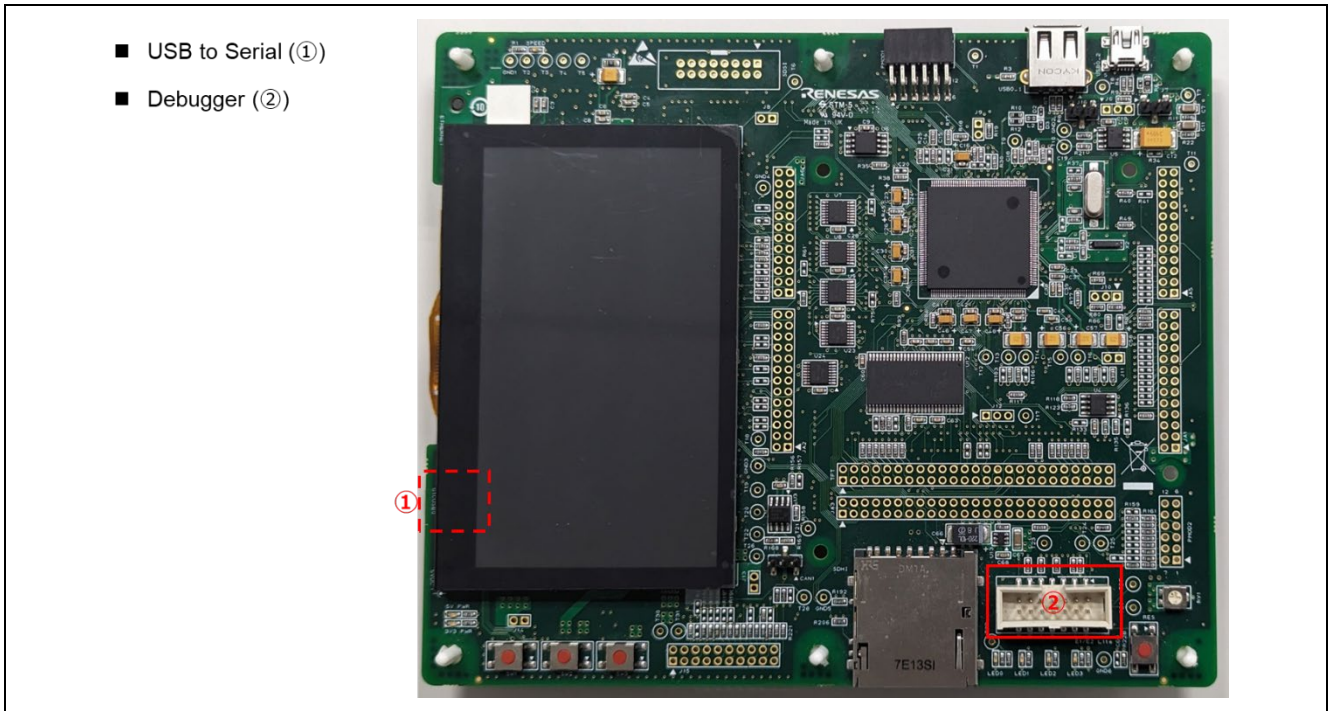


Figure 5-8 Information on Hardware Component Connections for the RSK-RX65N When the Update Method Uses Dual Mode

5.2.2.4 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is DirectXIP and the flash memory is in dual mode.

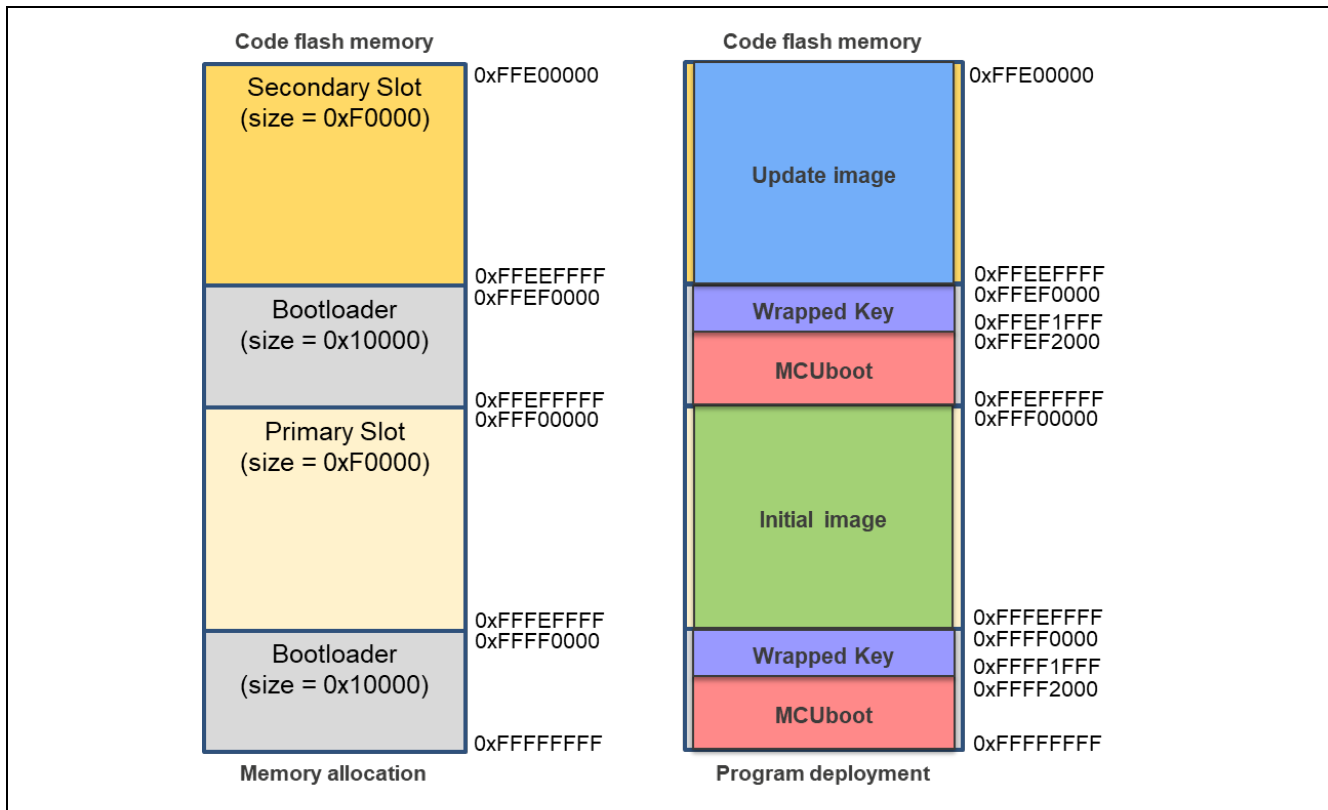


Figure 5-9 Memory Map of the Demo Project for the RSK-RX65N (2 MB) in the Case Where the Update Method Uses Dual Mode

Table 5-9 Configuration Option Settings of the Demo Project for the RSK-RX65N (2 MB) in the Case Where the Update Method Uses Dual Mode

Configuration options in <code>rm_mcuboot_config.h</code>	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3 (DirectXIP)
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.3 Environment Used for Verifying Operation of the RX671

5.2.3.1 Information on Hardware Component Connections for an Update in Linear Mode

The following shows the information on hardware component connections for the RSK-RX671 in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

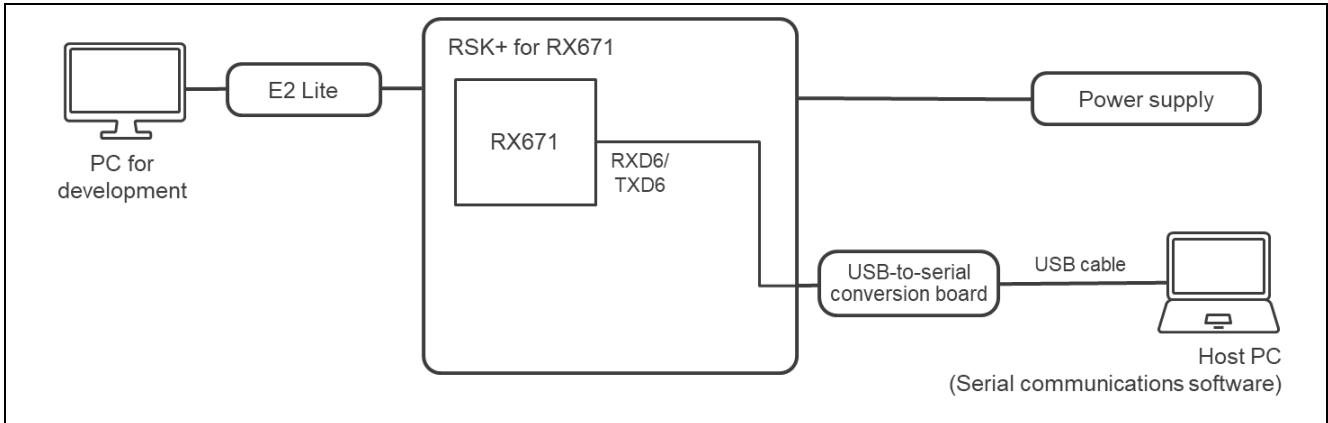


Figure 5-10 Hardware Component Connection Diagram for the RSK-RX671 When the Update Method Uses Linear Mode

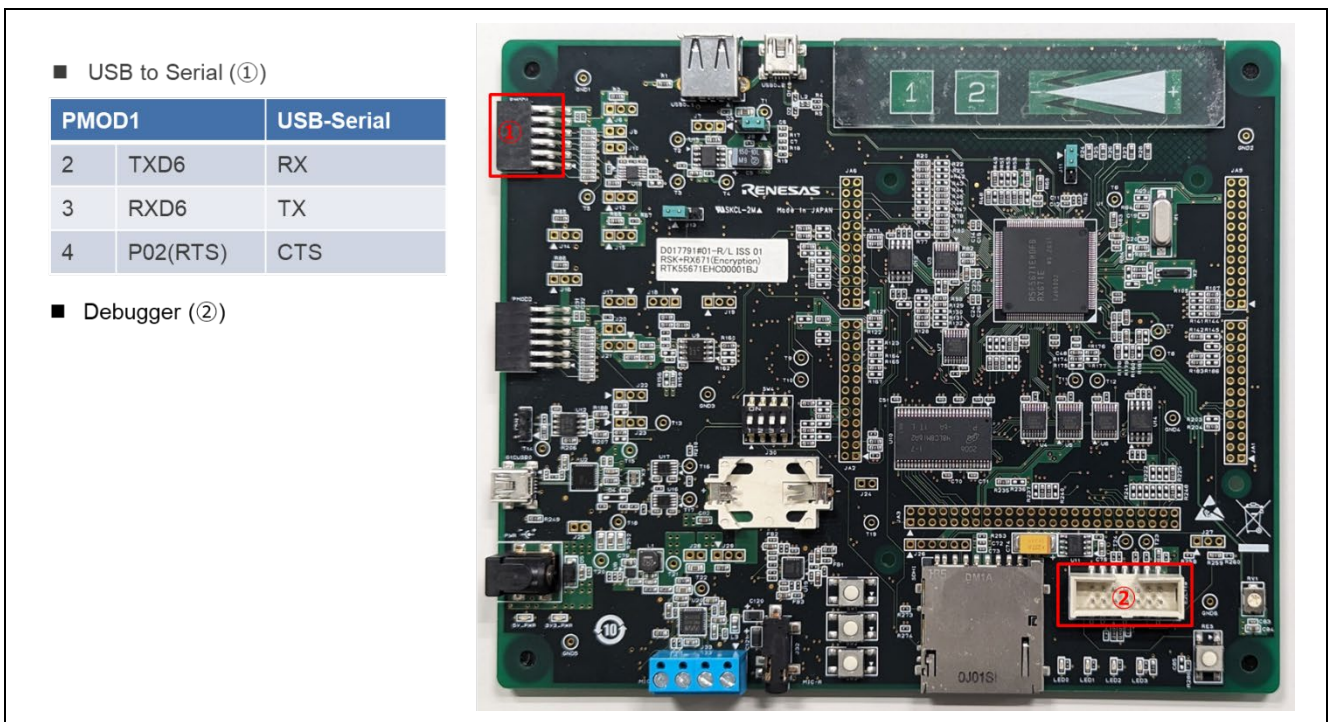


Figure 5-11 Information on Hardware Component Connections for the RSK-RX671 When the Update Method Uses Linear Mode

5.2.3.2 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

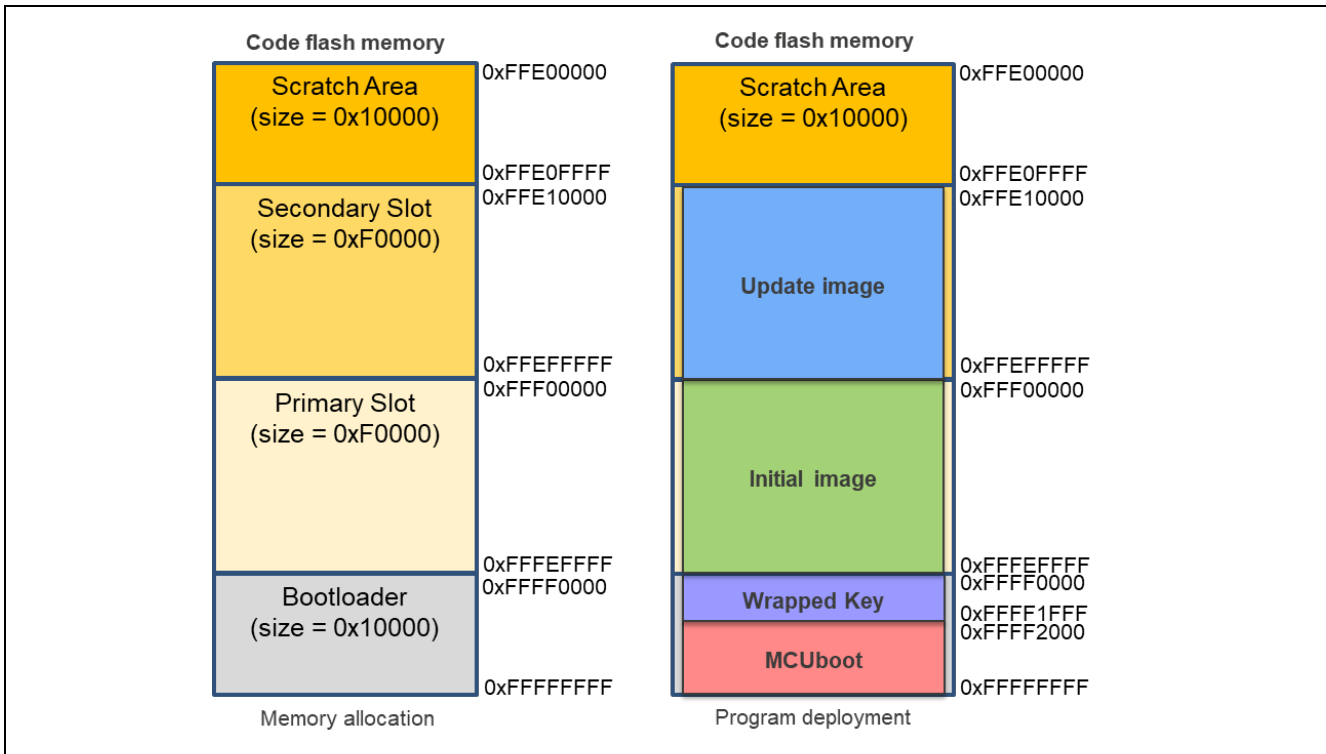


Figure 5-12 Memory Map of the Demo Project for the RSK-RX671 in the Case Where the Update Method Uses Linear Mode

Table 5-10 Configuration Option Settings for the RSK-RX671 in the Case Where the Update Method Uses Linear Mode

Configuration options in rm_mcuboot_config.h	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	Select a number in the range from 0 to 3.
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.3.3 Information on Hardware Component Connections for an Update in Dual Mode

The following shows the information on hardware component connections in the case where the update method is DirectXIP and the flash memory is in dual mode.

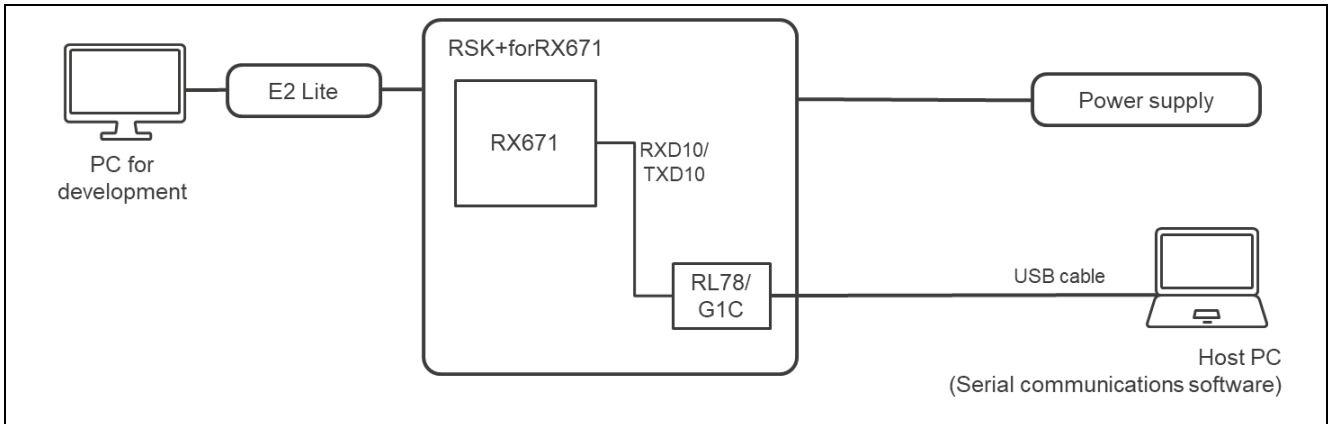


Figure 5-13 Hardware Component Connection Diagram for the RSK-RX671 When the Update Method Uses Dual Mode

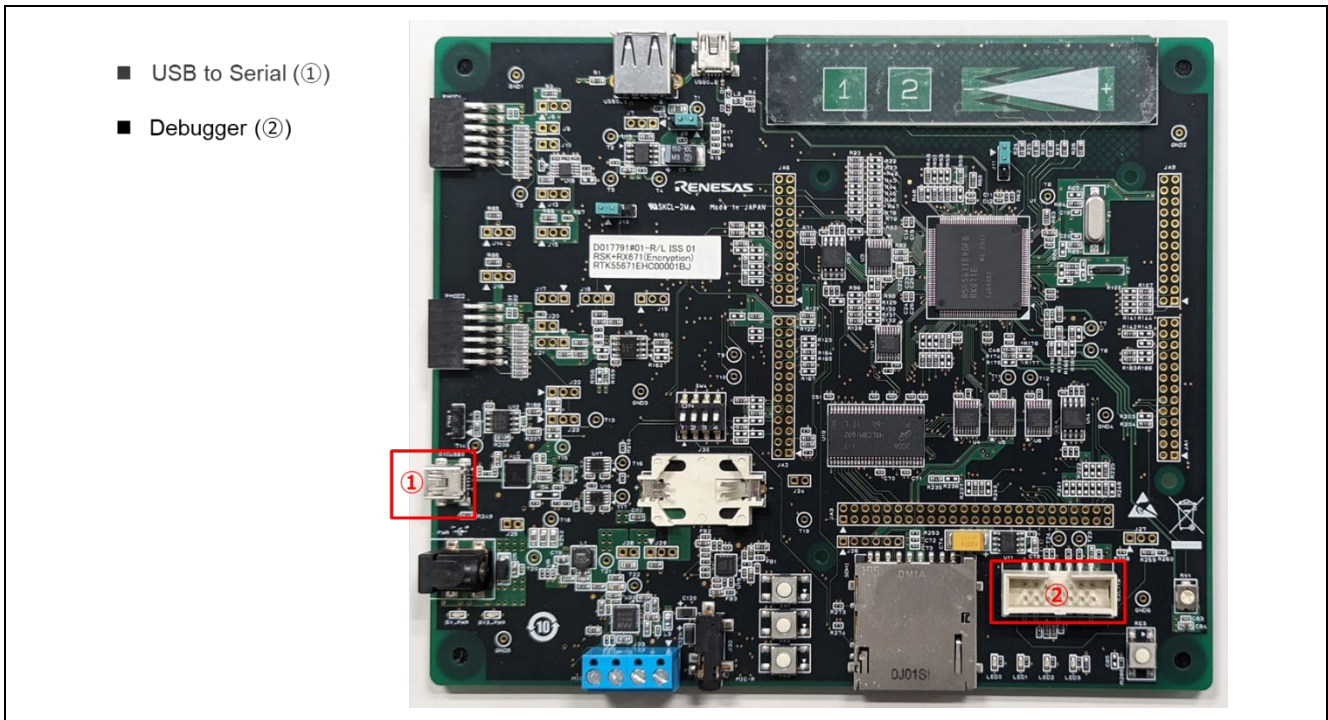


Figure 5-14 Information on Hardware Component Connections for the RSK-RX671 When the Update Method Uses Dual Mode

5.2.3.4 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is DirectXIP and the flash memory is in dual mode.

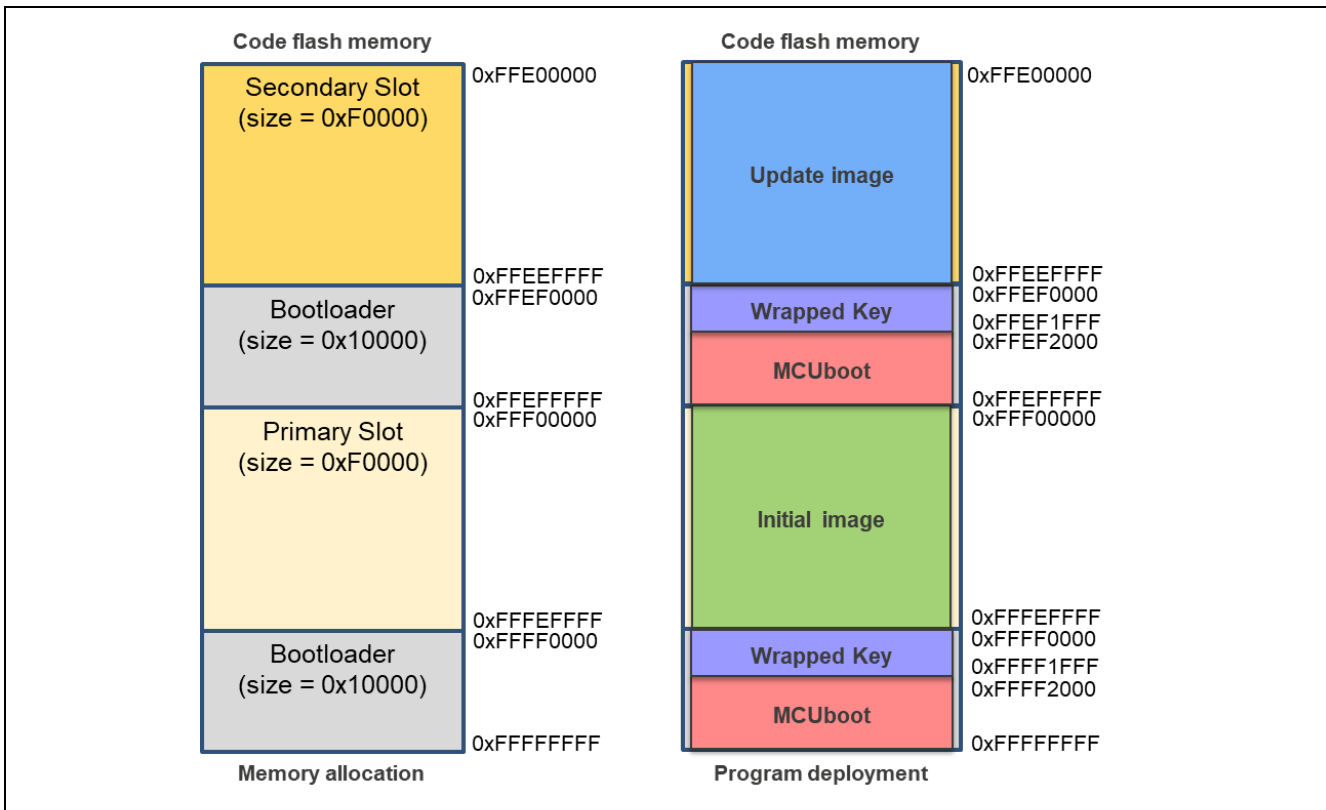


Figure 5-15 Memory Map of the Demo Project for the RSK-RX671 in the Case Where the Update Method Uses Dual Mode

Table 5-11 Configuration Option Settings of the Demo Project for the RSK-RX671 in the Case Where the Update Method Uses Dual Mode

Configuration options in rm_mcuboot_config.h	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3 (DirectXIP)
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.4 Environment Used for Verifying Operation of the RX72M

5.2.4.1 Information on Hardware Component Connections for an Update in Linear Mode

The following shows the information on hardware component connections for the RSK-RX72M in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

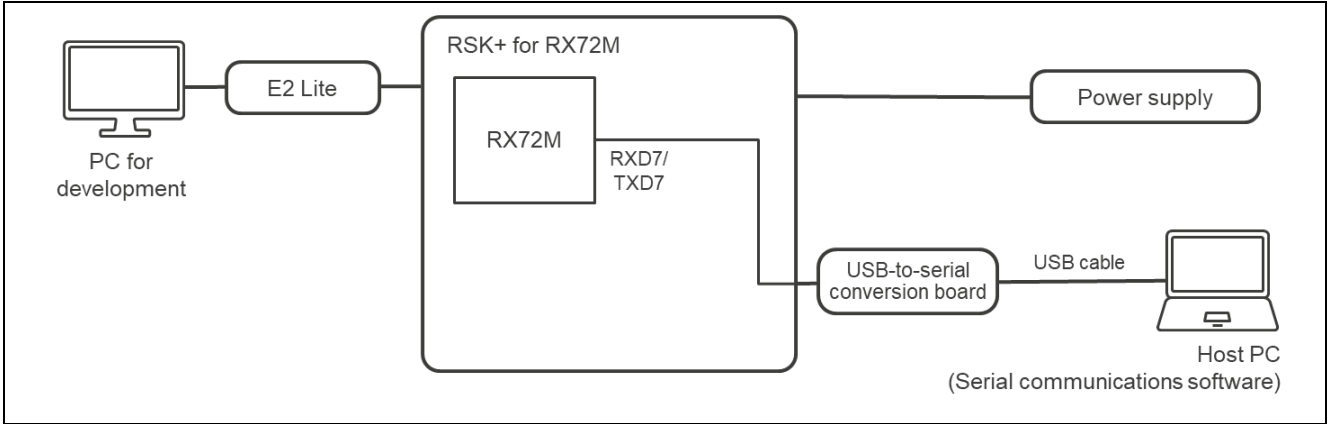


Figure 5-16 Hardware Component Connection Diagram for the RSK-RX72M When the Update Method Uses Linear Mode

- USB Serial (①)

PMOD1		USB-UART
2	TXD7	RX
3	RXD7	TX
4	PH0(RTS)	CTS

- Debugger (②)

Figure 5-17 Information on Hardware Component Connections for the RSK-RX72M When the Update Method Uses Linear Mode

5.2.4.2 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

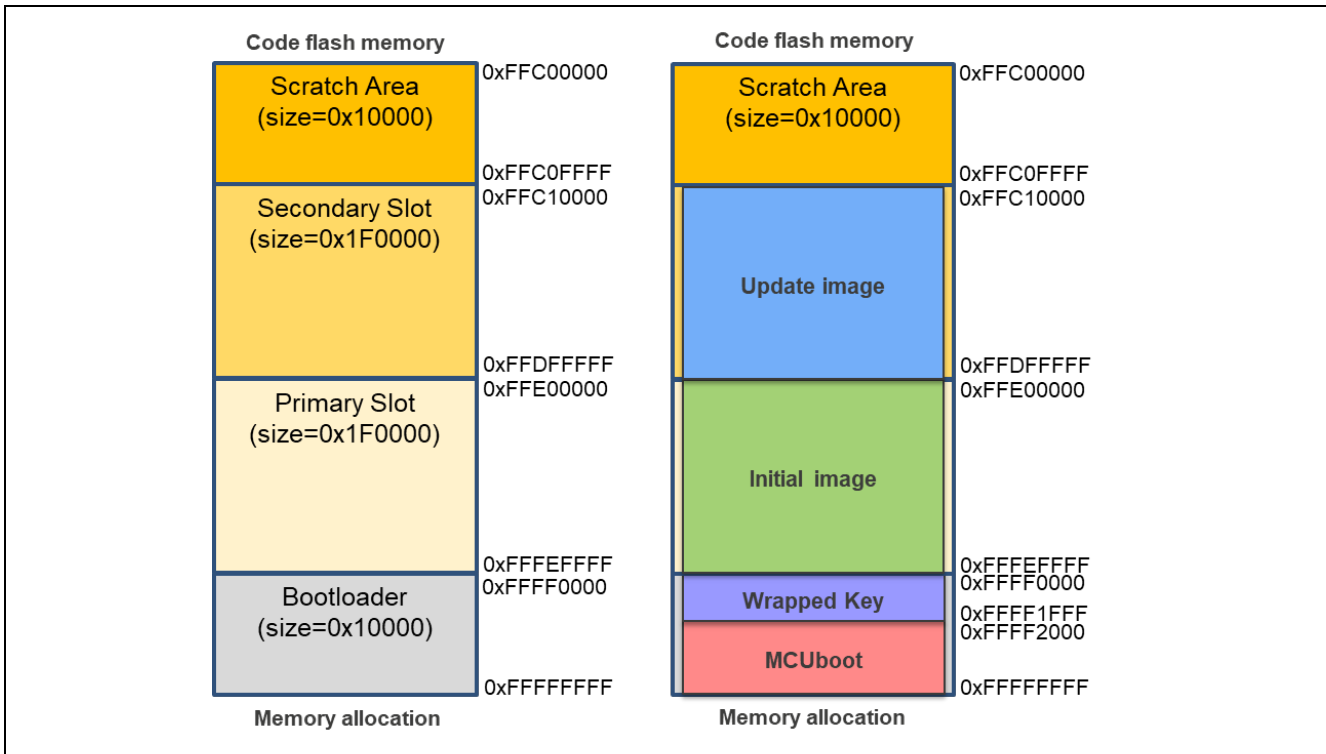


Figure 5-18 Memory Map of the Demo Project for the RSK-RX72M in the Case Where the Update Method Uses Linear Mode

Table 5-12 Configuration Option Settings for the RSK-RX72M in the Case Where the Update Method Uses Linear Mode

Configuration options in rm_mcuboot_config.h	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	Select a number in the range from 0 to 3.
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.4.3 Information on Hardware Component Connections for an Update in Dual Mode

The following shows the information on hardware component connections in the case where the update method is DirectXIP and the flash memory is in dual mode.

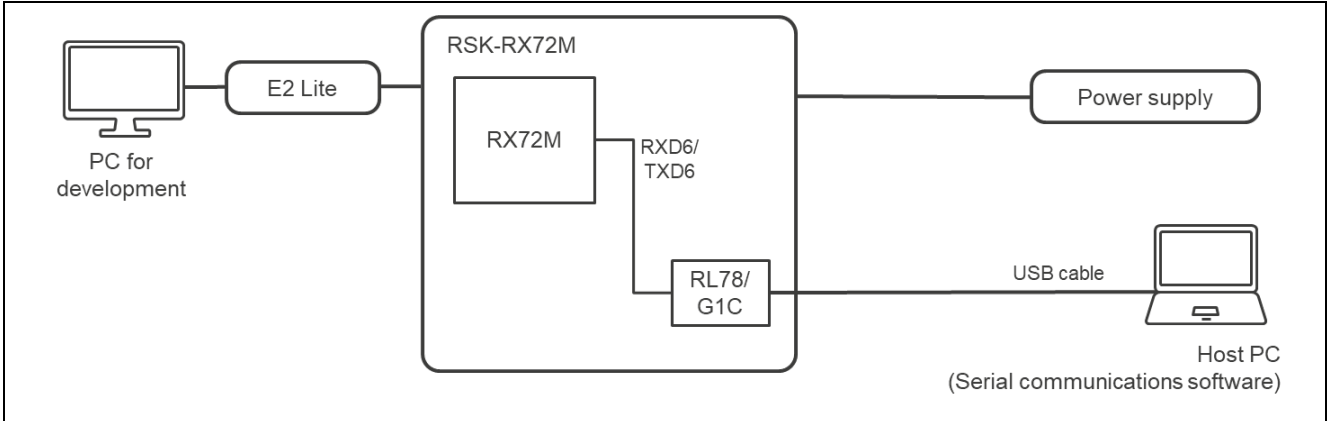


Figure 5-19 Hardware Component Connection Diagram for the RSK-RX72M When the Update Method Uses Dual Mode

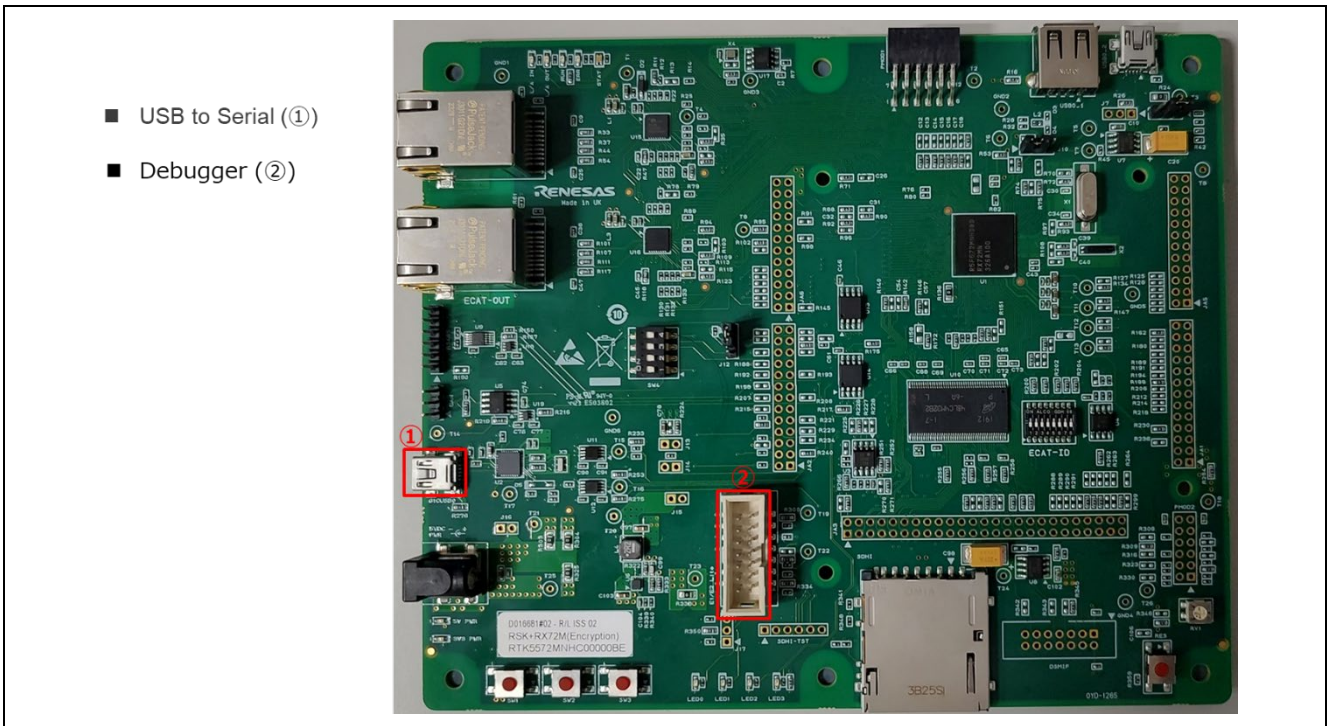


Figure 5-20 Information on Hardware Component Connections for the RSK-RX72M When the Update Method Uses Dual Mode

5.2.4.4 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is DirectXIP and the flash memory is in dual mode.

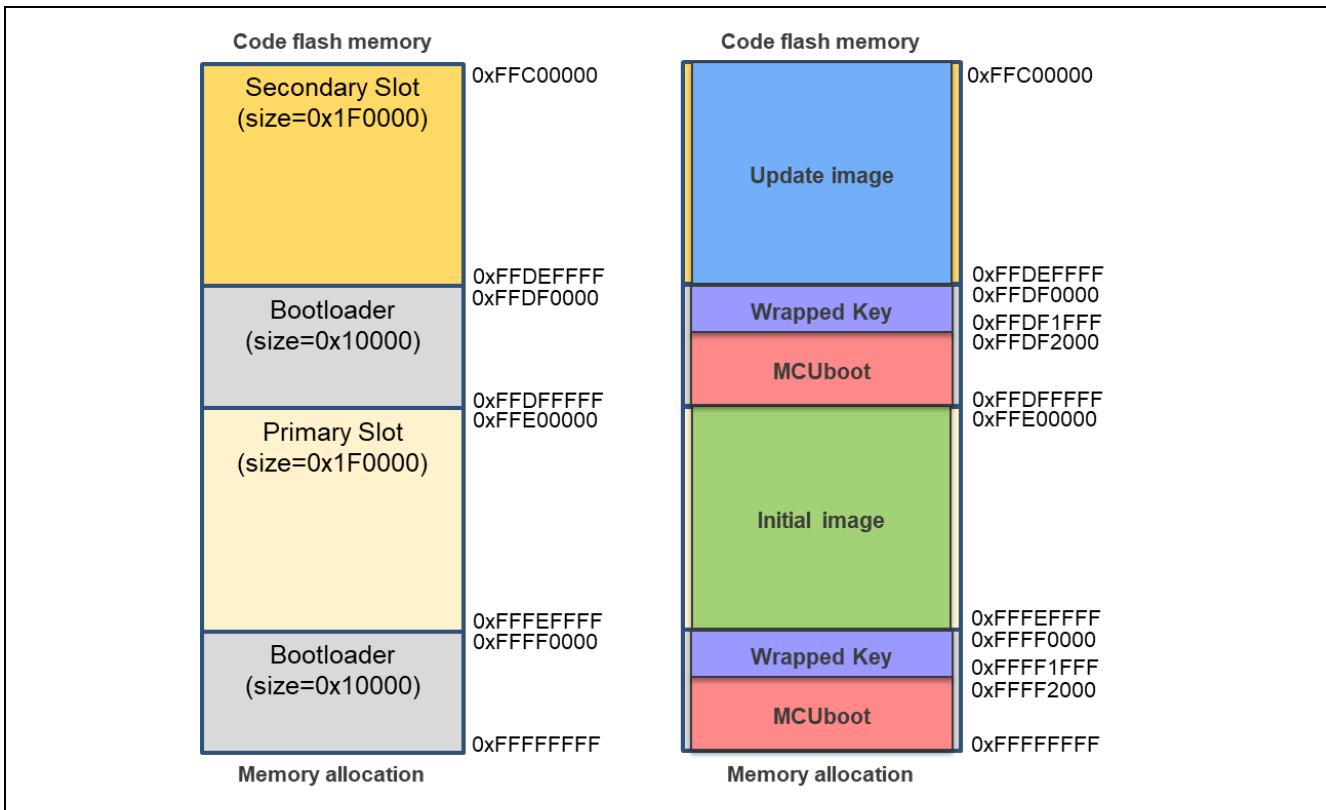


Figure 5-21 Memory Map of the Demo Project for the RSK-RX72M in the Case Where the Update Method Uses Dual Mode

Table 5-13 Configuration Option Settings of the Demo Project for the RSK-RX72M in the Case Where the Update Method Uses Dual Mode

Configuration options in <code>rm_mcuboot_config.h</code>	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3 (DirectXIP)
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.5 Environment Used for Verifying Operation of the RX72N

5.2.5.1 Information on Hardware Component Connections for an Update in Linear Mode

The following shows the information on hardware component connections for the RSK-RX72N in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

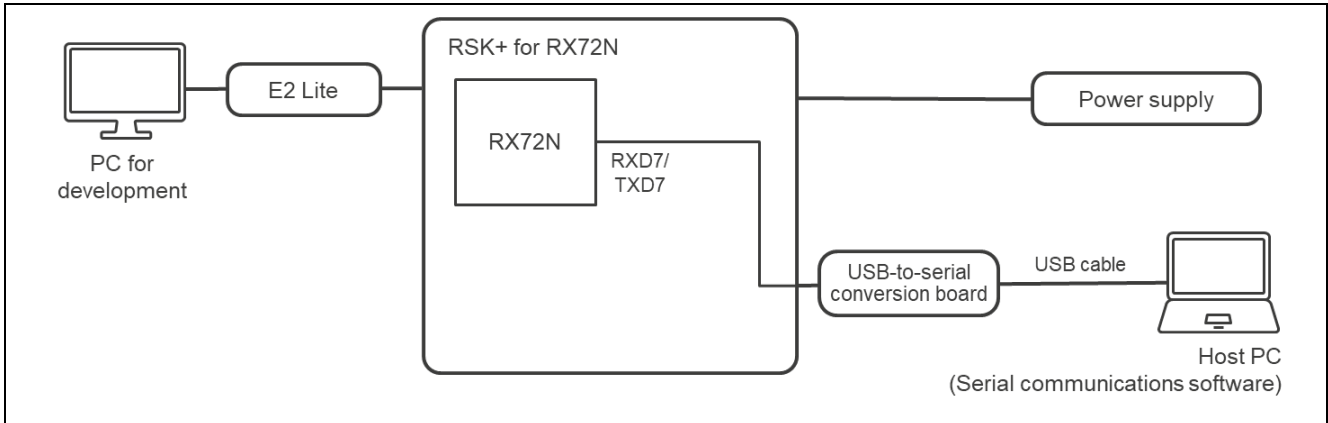


Figure 5-22 Hardware Component Connection Diagram for the RSK-RX72N When the Update Method Uses Linear Mode

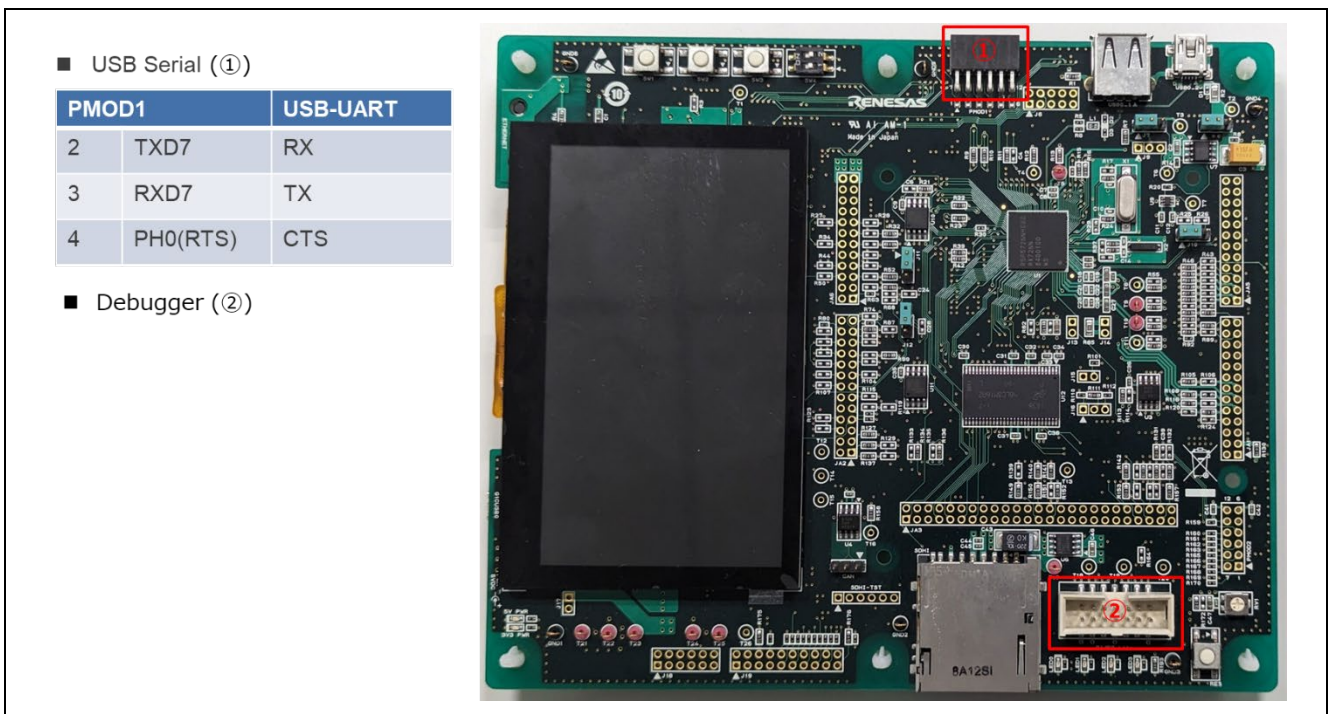


Figure 5-23 Information on Hardware Component Connections for the RSK-RX72N When the Update Method Uses Linear Mode

5.2.5.2 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Linear Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is Overwrite Only, Overwrite Only Fast, Swap, or DirectXIP and the flash memory is in linear mode.

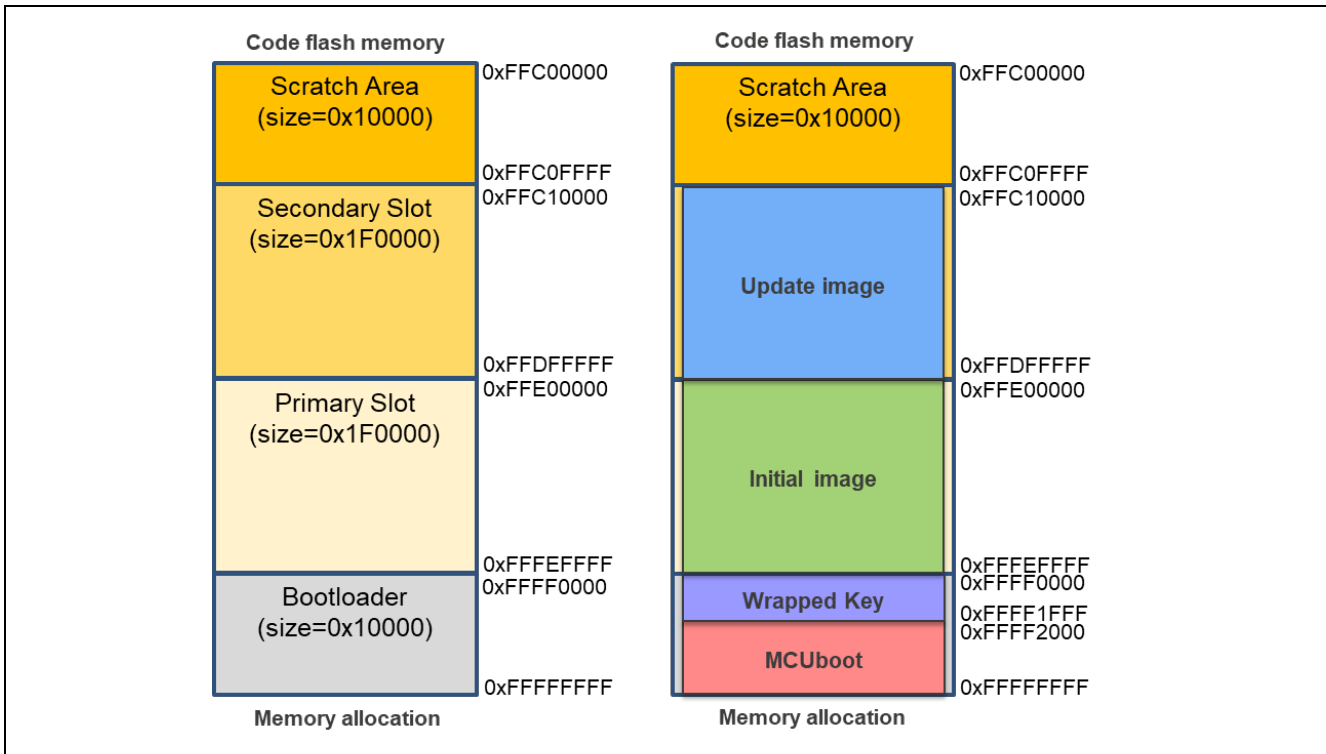


Figure 5-24 Memory Map of the Demo Project for the RSK-RX72N in the Case Where the Update Method Uses Linear Mode

Table 5-14 Configuration Option Settings for the RSK-RX72N in the Case Where the Update Method Uses Linear Mode

Configuration options in <code>rm_mcuboot_config.h</code>	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	Select a number in the range from 0 to 3.
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.5.3 Information on Hardware Component Connections for an Update in Dual Mode

The following shows the information on hardware component connections in the case where the update method is DirectXIP and the flash memory is in dual mode.

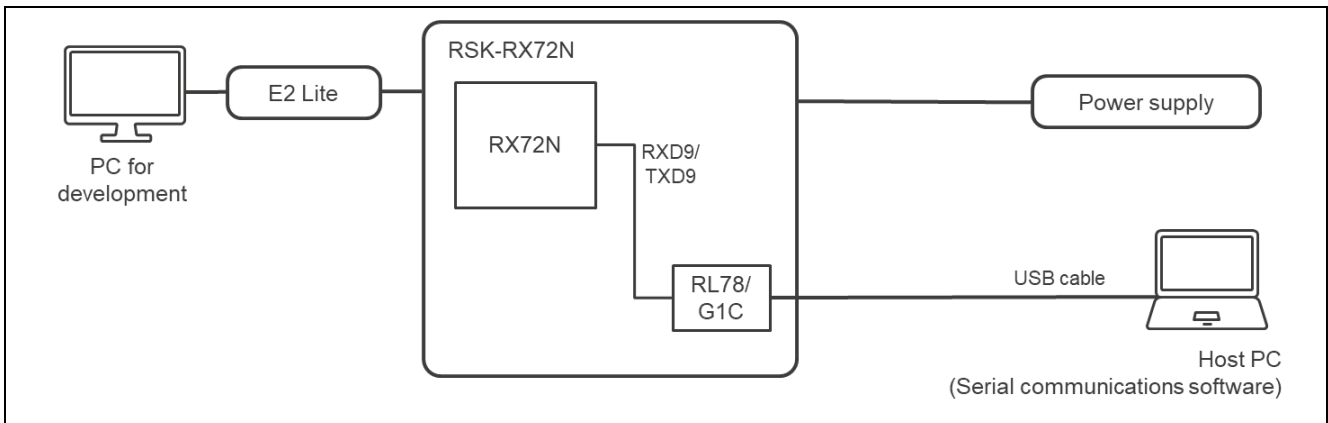


Figure 5-25 Hardware Component Connection Diagram for the RSK-RX72N When the Update Method Uses Dual Mode

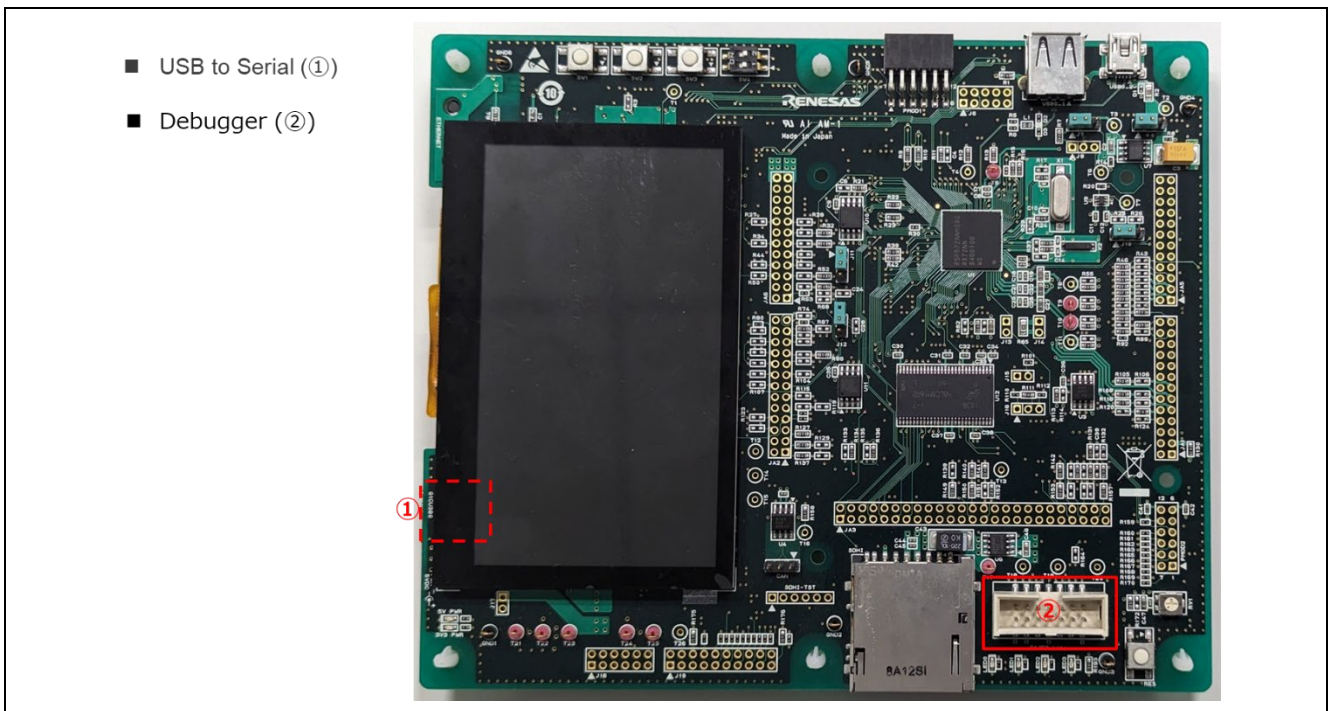


Figure 5-26 Information on Hardware Component Connections for the RSK-RX72N When the Update Method Uses Dual Mode

5.2.5.4 Memory Allocation and Configuration Option Settings in the Case Where the Update Method Uses Dual Mode

The following shows the memory map and configuration option settings of the demo project in the case where the update method is DirectXIP and the flash memory is in dual mode.

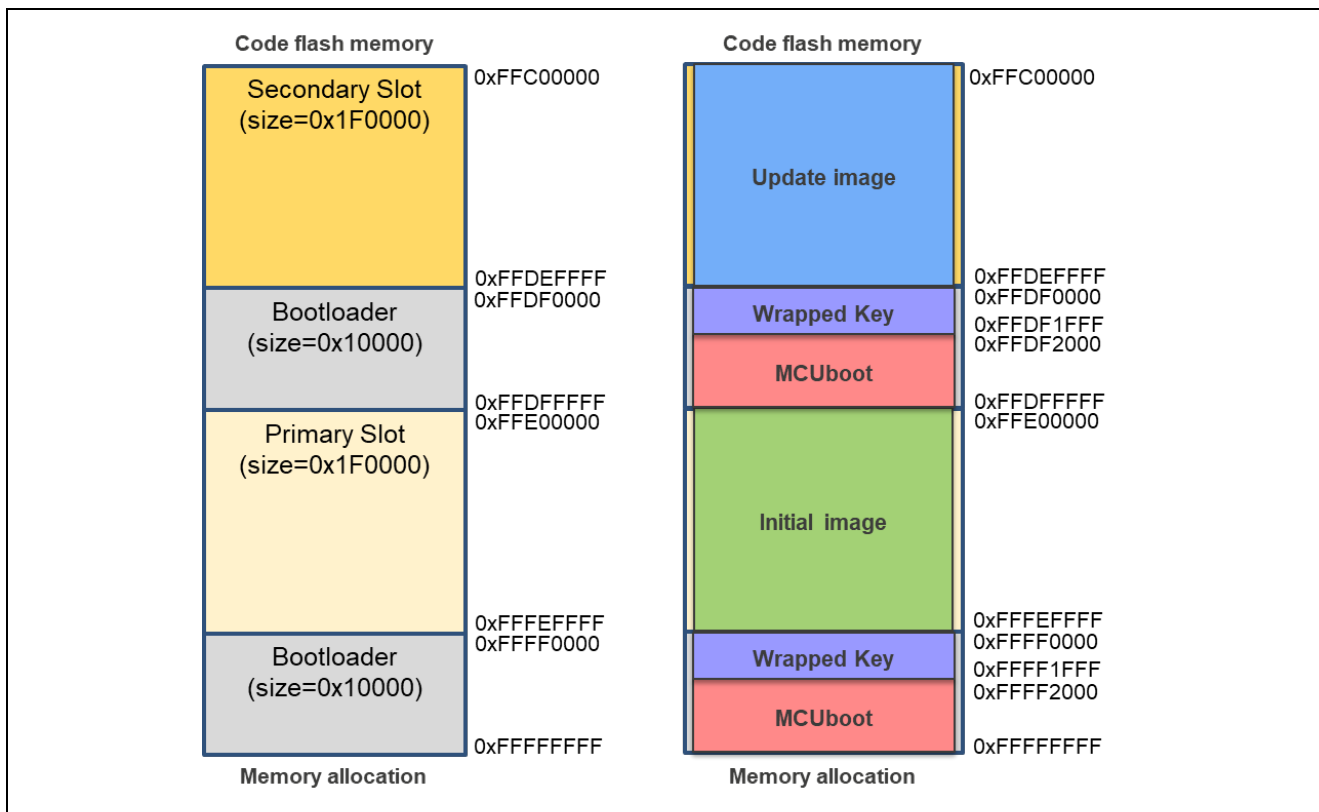


Figure 5-27 Memory Map of the Demo Project for the RSK-RX72N in the Case Where the Update Method Uses Dual Mode

Table 5-15 Configuration Option Settings of the Demo Project for the RSK-RX72N in the Case Where the Update Method Uses Dual Mode

Configuration options in <code>rm_mcuboot_config.h</code>	
Parameter Name	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3 (DirectXIP)
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION_ENCRYPTION_SCHEME	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0 (This setting takes no effect in this update method.)
RM_MCUBOOT_CFG_LOG_LEVEL	3

6. Notes

6.1 Notes on Transition from Bootloader(MCUboot) to Application.

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

For the peripheral functions used in the sample bootloader (Table 7.1), the API functions of each FIT module are closed at the end of the bootloader. Other settings are default values when the smart configurator is used.

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

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

Table 6.1 Notes on peripheral functions used in the bootloader

Peripheral Functions	FIT Modeule	Settings and Notes on the Boot Loader
Board Functions	r_bsp	These are the default values when the BSP FIT module is embedded in the Smart Configurator. The settings are not changed in the bootloader. Please note that the PMR and PFS registers are also set to match the board.
Functions of Flash Memory	r_flash_rx	The Flash FIT API performs Close for peripheral functions related to flash memory and transitions to the application.
Serial Communication Functions	r_sci_rx	For peripheral functions related to serial communication, Close is performed by the SCI FIT API and the transition is made to the application. For the SCI channels used in the bootloader, refer to the device connection diagram for each product in 6.2 Operating Environment for Demo Project.
Option Setting Memory	-	For the option setting memory, set the same value in the bootloader and the application program.
Other Functions	-	As for the settings of other functions, these are the default values when using the Smart Configurator. The PSW's interrupt enable flag is set to interrupt disabled to transition to the application.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr. 21, 2025	-	First edition issued.
1.01	Jun. 30, 2025	1	Added RX66N, RX671, and RX72M to the list of devices for which operation has been confirmed.
		7	Modified the notes in Table 1-1.
		12,13	Modified the contents of Table 1-2.
		17	Modified the default values in Table 2-1.
		19-22	Modified the compiler version and ROM, RAM, and stack in 2.7.
		42,43	Modified the contents of 4.3.3.3.
		49-51	Modified the compiler and module versions in 5.1.
		58-61	Added RX671 operation verification environment to 5.2.3.
		62-65	Added RX72M operation verification environment to 5.2.4.
1.02	Mar. 31, 2026	28	Added Table 4-1.
		34-37	Modified the contents of 4.3.1.
		-	Included spdx file in the package.

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.
7. 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/.