Renesas

Renesas MCU Firmware Update Design Policy

Summary

This application note describes the firmware update design policy for Renesas MCUs.

Target Devices

RX Family

Related Documents

- Implementing Firmware Updates on the RX Family (R01AN5549EJ0100), which applies to the RX Family
- Amazon FreeRTOS repository wiki for Renesas GitHub account
  https://github.com/renesas/amazon-freertos/wiki/OTA%E3%81%AE%E6%B4%BB%E7%94%A8
  This document is the original. Refer to it for the latest information.

Contents

1. Definitions of Terms

Firmware update	To update the firmware
Self-programming	The operation that takes place when the firmware update target is the firmware itself
Bootloader	Software meeting the requirements for a bootloader as defined by Amazon Web Services
User program area (exe) for self-programming Labeled “execute area” in figures.	Area for storing software that performs user system operations and firmware update operations
Temporary area (tmp) for rewriting Labeled “temporary area” in figures.	An area of the same size as the user program area (exe) used for self-programming. Used to temporarily store new firmware during a firmware update.
Bootloader area (including reset vector) for validating the user program area Labeled “bootloader” in figures.	Area where the bootloader is stored
A copy of the bootloader area (including reset vector) for validating the user program area Labeled “bootloader(mirror)” in figures.	A copy of the bootloader. Required when using the bank swapping mechanism on a device such as the RX65N.

2. Overview

2.1 Why Firmware Update Functionality is Needed

There is growing demand for firmware update functionality in embedded systems. Some reasons for this are the following:

- To correct system malfunctions due to software bugs that have become apparent once the product is already on the market
- To apply updates to open source software
- To respond to changes in market requirements

Recently, as embedded systems gain internet connectivity, there is a growing risk of system malfunctions due to software defects being exploited by malicious third parties. Governments in various countries are taking measures to deal with this issue, such as establishing guidelines for the implementation of firmware update functionality in electronic devices connected to the internet and defining firmware update functionality to be applied to specific standard certifications.

Examples in Japan:

IoT Security Guidelines

https://www.soumu.go.jp/main_content/000428393.pdf

Guidelines on Standard Certifications of Terminal Devices Based on the Telecommunications Business Act (Version 1)

https://www.soumu.go.jp/main_content/000615696.pdf

2.2 Firmware Update Functionality and Internet Connectivity

Smartphones are a familiar example of a product that utilizes firmware updates. Nowadays, smartphones are used by people all over the world, and most of them are connected to the internet. An operating system (OS) is incorporated into the smartphone, and the OS provides internet connection functionality. The above-mentioned reasons why firmware update functionality is needed also apply to smartphones, and OS vendors update their OSes and distribute the updates frequently.

If a smartphone did not have firmware update functionality, security holes discovered by malicious third parties could not be corrected, and the user would be exposed to a constant risk that the personal information stored on the device (name, address, age, email address, credit card number, passwords, etc.) might be disclosed to unauthorized parties.

If a smartphone did not have internet connectivity, the danger of the above risks being exposed by malicious third parties would be reduced.

If a smartphone had internet connectivity but no firmware update functionality, malicious third parties could expose the above risks and this situation would continue indefinitely.

With firmware update functionality, the above risks could exposed by malicious third parties, but the situation would not continue indefinitely.

These correlations are listed in the table below. **Advantageous items are indicated by light blue text**, and **disadvantageous ones by red text**.

	Internet connectivity	**No internet connectivity**
Firmware update functionality	**High possibility of risks surfacing** **Surfaced risks not permanent**	**Low possibility of risks surfacing** **Surfaced risks not permanent**
No firmware update functionality	**High possibility of risks surfacing** **Surfaced risks are permanent**	**Low possibility of risks surfacing** **Surfaced risks are permanent**

From the viewpoint of security risk, the combination of **firmware update functionality** and **no internet connectivity** is the **most secure**.

However, the trend nowadays is for all embedded devices, including automobiles, surveillance cameras, transportation systems, home appliances, industrial robots, building equipment, and smart meters, to have internet connectivity. The ability to connect to the internet brings significant functional improvements to electronic devices, but there are cases where security measures such as firmware update functionality have not been adequately integrated into new products, leading to problems when they reach the market.

This situation is represented by **internet connectivity** and **no firmware update functionality** in the above table, which is the **most dangerous combination** in terms of security.

Therefore, when developing products with **internet connectivity**, security measures such as **firmware update functionality** are essential. In such cases, there is a **high possibility of risks surfacing**. Therefore, it is necessary to take additional security measures throughout the product life cycle, based on the following two points:

1. Product development (hardware and software) that applies a security development process that can minimize risk
2. Building an operating structure to quickly deliver firmware updates when risks become apparent

Note that even in cases of **no internet connectivity**, **firmware update functionality** is still recommended as it provides the advantage of **surfaced risks not being permanent**.

This document omits discussion of the product life cycle and focuses on the firmware update design policy.

2.3 Definition of Firmware Update Functionality

Electronic devices such as smartphones and embedded devices are generally classified as “computers.” Most computers have a configuration like that shown below, with CPU, memory, and I/O connected by a bus.

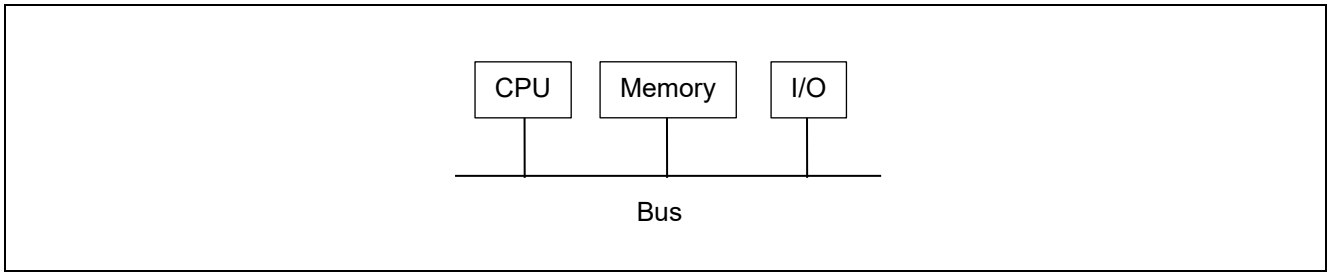


Figure 2.1 System Block Diagram of a Typical Computer

Next, let’s review the configuration of an actual Renesas MCU. A detail view of the system block diagram of the RX65N, an MCU in the RX family, is shown below. In Figure 2.1 and Figure 2.2, RX CPU is the CPU, ROM and RAM are the memory, and the various ports comprise the I/O.

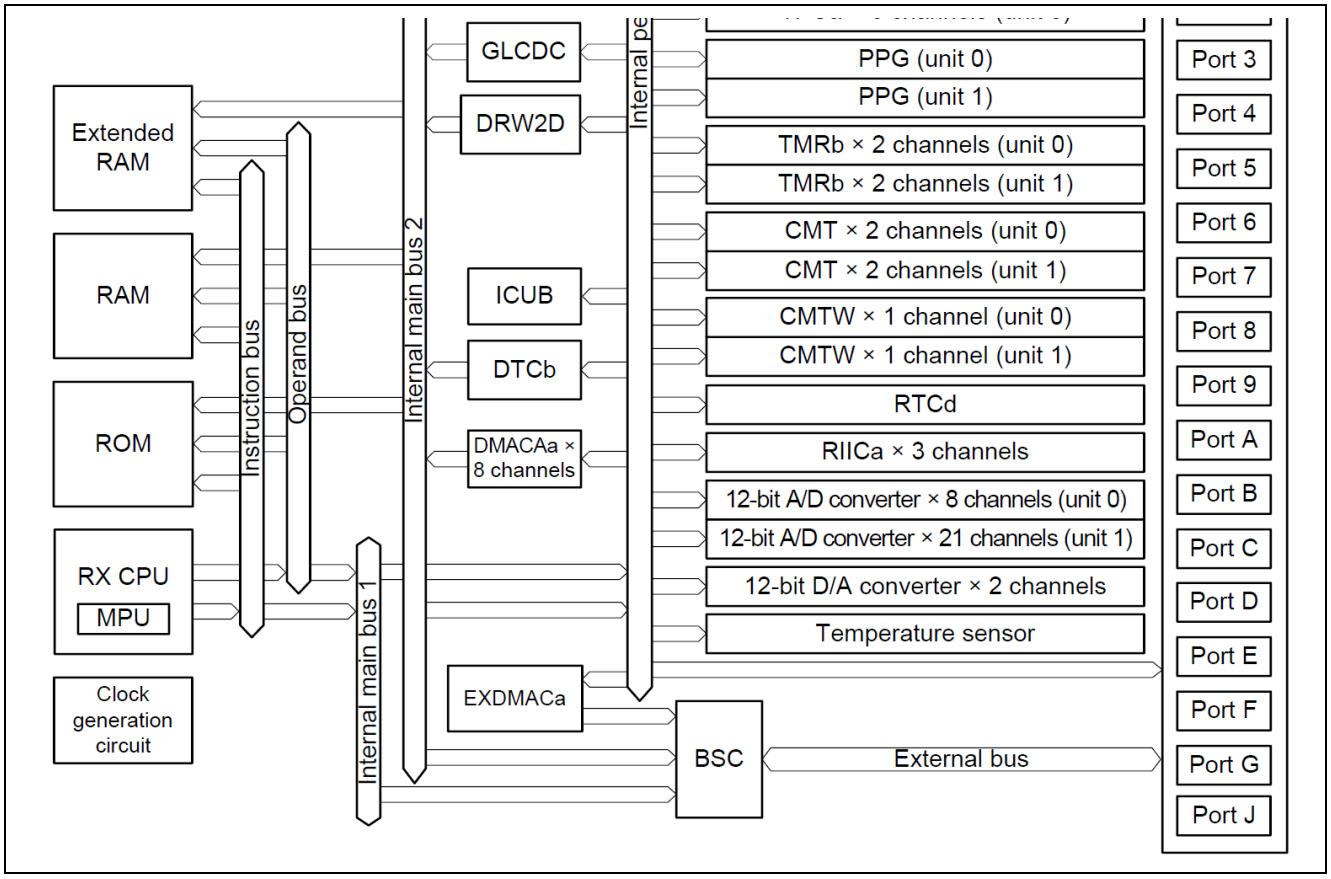


Figure 2.2 RX65N System Block Diagram (Detail View)

On the RX65N software code is generally stored in the ROM. The RX65N is a microcontroller (MCU) with on-chip flash memory, and ROM refers to flash memory in the above figure. Flash memory is non-volatile memory, so the data written to the flash memory is not lost even when no power is being supplied. In addition, the RX65N (like many other MCUs with on-chip flash memory) has a mechanism that allows the CPU to send commands such as programming and erasing commands to the flash memory. This functionality can be used to overwrite software previously written to the flash memory. Using this functionality to update software after it is put on the market is called “applying a firmware update.” The operation itself is called “self-programming.”

3. Flash Memory Rewrite Specifications

3.1 RX Family

Below we present a specific example of the method for rewriting the flash memory, using the RX65N for illustration.

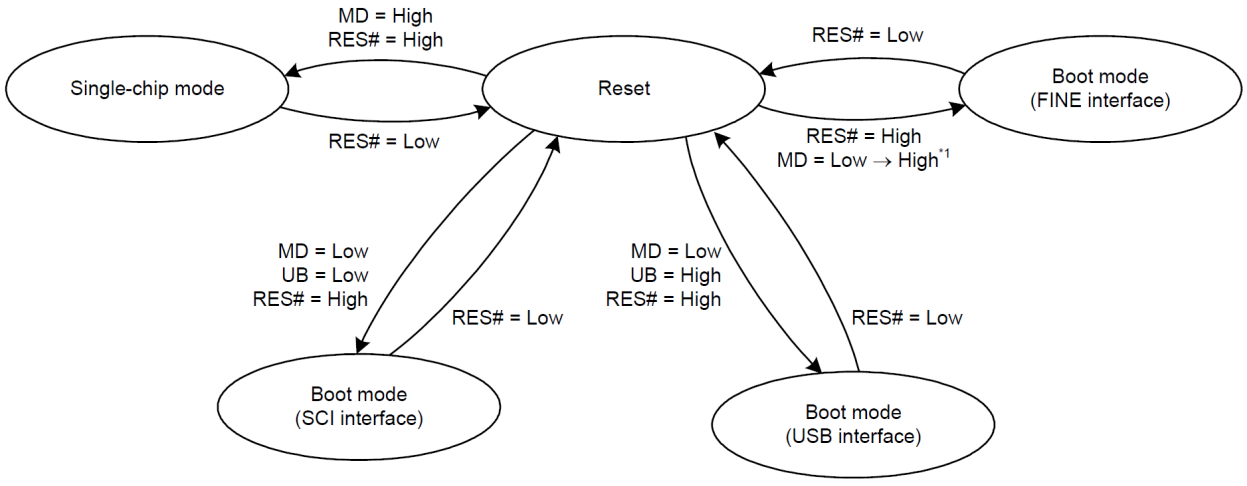
On-board programming (Serial programming/Self-programming)	Programming/erasure in boot mode (for the SCI interface) • The asynchronous serial interface (SCI1) is used. • The transfer rate is adjusted automatically. Programming/erasure in boot mode (for the USB interface) • USBb is used. • Dedicated hardware is not required, so direct connection to a PC is possible. Programming/erasure in boot mode (for the FINE interface) • FINE is used Programming and erasure by self-programming • This allows code flash memory programming/erasure without resetting the system.	
Off-board programming*4	Programming and erasure of the code flash memory and option-setting memory by using a parallel programmer is possible.	Programming or erasure of the data flash memory by a parallel programmer is not possible.

Figure 3.1 Excerpt of RX65N Flash Memory Specifications

The methods of rewriting the flash memory on the RX65N can be roughly divided into “on-board programming” and “off-board programming.” Off-board programming is mainly used in the electronic device assembly stage. Typically, the process is as follows. The RX65N is in a blank state (with nothing written to the flash memory) at the time of shipment from Renesas. A provider of device programming services writes (programs) the software to the blank RX65N MCUs and sends them to the assembly plant. There, the programmed RX65N MCUs are mounted on circuit boards in electronic devices, after which they undergo final inspection and shipment.

To perform off-board programming after an electronic device is already on the market would require removing the RX65N MCU from the circuit board inside the device, and it would be unrealistic to ask the customer to undertake off-board programming simply to apply a firmware update. For this reason, on-board programming is used to apply firmware updates.

On-board programming can be further subdivided into “serial programming” and “self-programming.” Serial programming utilizes the MCU’s “boot mode” (various types), and self-programming utilizes “single-chip mode.” The transitions between the various operating modes of the RX65N are shown in the diagram below.



Note 1. After release from the reset state, switch the MD pin to high level within 20 to 100 msec.

Figure 3.2 Excerpt of RX65N Flash Memory Specifications

MD, UB, and RES# in the above figure are the names of pins on the RX65N. System operation takes place in single-chip mode. But applying a firmware update in boot mode requires the end user to manipulate the MD, UB, and RES# pins. If the end user is a person who can operate the MD, UB, and RES# pins by means of buttons on the product or the like whenever necessary, applying a firmware update in boot mode is a possibility.

Nevertheless, applying a firmware update to a smartphones does not force users to manipulate the MD, UB, and RES# pins, and most consumers are already accustomed to such a system. Therefore, in the absence of special circumstances requiring otherwise, we recommend that firmware update functionality be implemented in single-chip mode as part of the operation of the user system.

In addition, in contrast to embedded devices that nearly everyone owns like smartphones, firmware update functionality utilizing single-chip mode is a mandatory requirement on devices such as bridge condition monitoring systems or photovoltaic panels that need to operate continuously when no one is nearby.

4. Firmware Update Mechanism

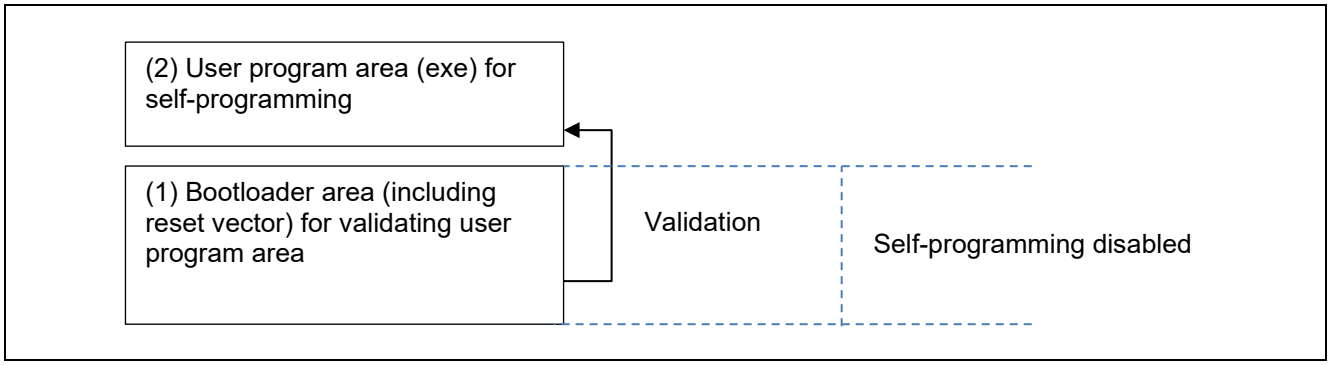
4.1 Design Policy

To implement firmware update functionality, the updated version of the firmware is received from outside the MCU via a specific communication path (UART etc.), and self-programming is performed. There are two major implementation issues here.

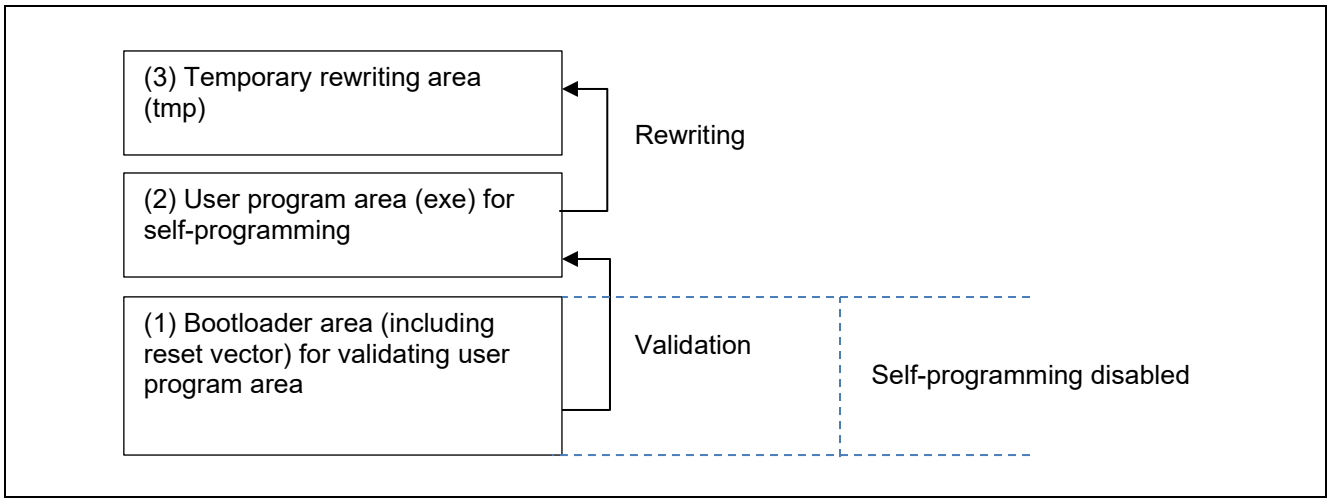
Issue 1: Is it possible to recover if power is cut off during self-programming?
(How do you detect the interruption after power is restored?)

Issue 2: Is it possible to recover if power is cut off while overwriting the software that executes self-programming?

To resolve issue 1, it is necessary to separate (2) the user program area for self-programming and (1) the bootloader area for validating the user program area, and to ensure that (1) is always run after a reset. To ensure that area (1) is not corrupted by a self-programming error, it must be possible to disable self-programming of area (1) only.

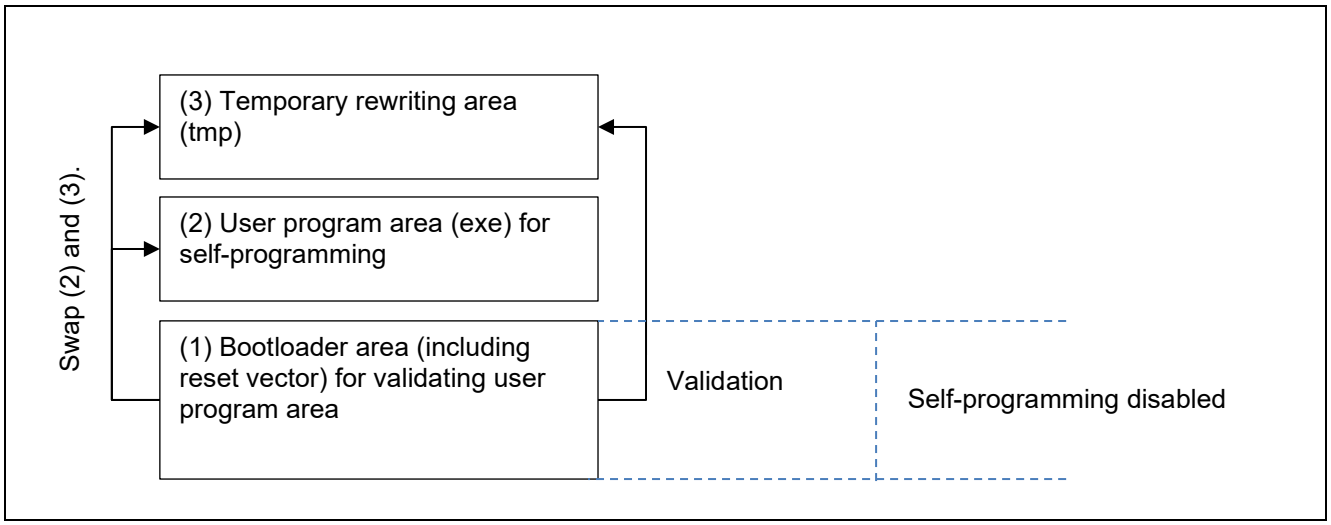


To resolve issue 2, two areas are set aside: (2) the user program area (exe) for self-programming and (3) a temporary rewriting area (tmp) of the same size. Thus, even if power is interrupted while (3) is being overwritten and validation fails (rewrite failure), (2) can be maintained in a validated state (rewrite success).



Resolving issue 2 gives rise to the following new issues:

Issue 3: In many cases the new firmware transferred by the firmware update functionality is created with the assumption that it will be run in (2) the user program area (exe) for self-programming, and it cannot be run if left in (3) the temporary rewriting area (tmp).



To resolve issue 3, the bootloader area validates (3) the temporary rewriting area (tmp), and if validated (rewrite success), replaces (2) with (3) ((2) is deleted and (3) is copied to area (2)). If the MCU has dual bank or bank swap functionality, as is the case on the RX65N, we recommend using it to eliminate the need for the above copy operation.

4.2 General Bootloader Requirements 1

Generally, it is necessary to implement a bootloader as the basic component in order to provide firmware update functionality. With regard to firmware updates on Renesas MCUs, please refer to the eight requirements below, extracted from documentation provided by Amazon Web Services.

https://github.com/aws/amazon-freertos/blob/4d52fdb0e85c14d88c895ae6c144668ced8409af/tests/Amazon%20FreeRTOS%20Qualification%20Developer%20Guide.pdf

→ 5.10 Appendix J: Bootloader

Note: The above URL links to documentation for Amazon FreeRTOS version 1.4.7, and newer versions may contain updates to the ideas outlined below. Revisions may be necessary to reflect the ideas contained in the latest version.

1. The bootloader shall be stored in non-volatile memory so it cannot be overwritten.
2. The bootloader shall verify the cryptographic signature of the downloaded application image. Signature verification must be consistent with the OTA image signer. See Appendix I: OTA Updates for supported signatures.
3. The bootloader shall not allow rolling back to a previously installed application image.
4. The bootloader shall maintain at least one image that can be booted.
5. If the MCU contains more than one image then the image that is executed shall be the latest (newest). The newest version can be determined based on implementation, for example a user defined sequence number, application version etc. As per other requirements, this can only be the case until a newer version has been verified and proven functional.
6. If the MCU cannot verify any images then it shall place itself into a controlled benign state. In this state it prevents itself from being taken over by ensuring no actions are performed.
7. These requirements shall not be breached even in the presence of an accidental or malicious write to any MCU memory location (key store, program memory, RAM, etc.)
8. The bootloader shall support self-test of a new OTA image. If test execution fails, the bootloader shall roll back to the previous valid image. If test execution succeeds, the image shall be marked valid and the previous version erased.

4.3 General Bootloader Requirements 2

The Internet Engineering Task Force (IETF) summarizes its requirements for firmware updates as follows:

https://datatracker.ietf.org/doc/draft-ietf-suit-architecture/

Renesas will continue to update appropriate bootloader implementations while gathering information on common bootloader requirements of the sort referenced above.

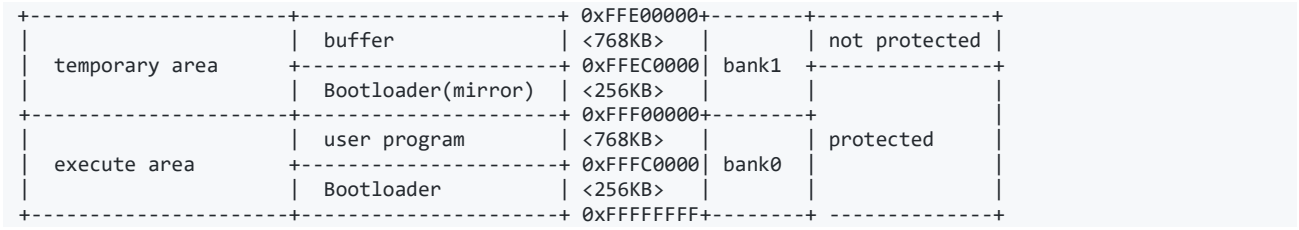
4.4 Bringing RX65N Bootloader Implementation into Conformance with General Bootloader Requirements

Below we review the consistency of the RX65N implementation with items 1 to 8 as defined in general bootloader requirements 1.

1. The bootloader shall be stored in non-volatile memory so it cannot be overwritten.

   [Implementation on RX65N]

   The bootloader is stored in the flash memory (non-volatile memory).
   Overwriting of the bootloader is prohibited by using the flash access window (FAW) function to disable self-programming of the area from 0xFFEC0000 to 0xFFFFFFFF.
   In addition, the contents of the FAW register can be completely secured by clearing the FSPR (access window protection) bit in the FAW register to 0, entirely prohibiting subsequent overwriting of the FAW register itself. At the same time, overwriting of the areas marked as “protected” in the figure below is also prohibited, so caution is necessary.



   Additional information:
   The RX65N has a bank swap function. The addresses of bank0 and bank1 in the figure above can be swapped, but since the swapped addresses also include the bootloader and since the bank swap target includes the reset vector at address 0xfffffffc, it is necessary that Bootloader(mirror) containing the same contents as the bootloader be allocated to bank1 as well. At initial startup the bootloader copies data identical to itself to Bootloader(mirror) using self-programming.
   Note that the bank0 and bank1 addresses are swapped during bank swapping, but the FAW protection target does not change. In other words, the self-programmable area is always only a buffer area.

2. The bootloader shall verify the cryptographic signature of the downloaded application image. Signature verification must be consistent with the OTA image signer. See Appendix I: OTA Updates for supported signatures.

   [Implementation on RX65N]

   The ECDSA-SHA256 algorithm is implemented in the bootloader and user program for verifying cryptographic signatures. This algorithm is compatible with the OTA image signer of Amazon Web Services.

3. The bootloader shall not allow rolling back to a previously installed application image.

   [Implementation on RX65N]

   The bootloader prevents rolling back to the previously installed application because it deletes the user program area (which contains the previously installed application) after verifying the buffer area (which contains the new application image).

4. The bootloader shall maintain at least one image that can be booted.

   [Implementation on RX65N]

   The contents of the flash memory are overwritten at the following two points in time:
   (1) When the user program area erases/overwrites the buffer area
   (2) When the bootloader area erases/overwrites the user program area

   Since the user program area in (1) and the buffer area in (2) are maintained in an intact state, even if the system is rebooted due to a power interruption at any time, the bootloader will detect that either the buffer area or the user program area is intact and attempt recovery as indicated below.

Table 4.1 Bootloader Behavior

	User program area intact	User program area corrupted
Buffer area intact	Delete user program area. → Copy buffer area to user program area. → Launch user program.	Delete user program area. → Copy buffer area to user program area. → Launch user program.
Buffer area corrupted	Launch user program.	Unable to launch.

5. If the MCU contains more than one image then the image that is executed shall be the latest (newest). The newest version can be determined based on implementation, for example a user defined sequence number, application version etc. As per other requirements, this can only be the case until a newer version has been verified and proven functional.

   [Implementation on RX65N]

   In Table 4.1, Bootloader Behavior, the new image is always stored in the buffer area. If both the buffer area and the user program area are in an intact state, the image that will be launched subsequently will always be the one in the buffer area (which is copied to the user program area).
   In future we plan to make use of the sequence number contained in the image and implement improvements so that the image will not be accepted unless it has a sequence number larger than the sequence number of the previous image. At that point, it will probably be more advantageous to link with the security IP module to provide stronger protection of the sequence number. At present there seems to be little market demand for such a feature, so only the framework has been prepared but not implemented.

6. If the MCU cannot verify any images then it shall place itself into a controlled benign state. In this state it prevents itself from being taken over by ensuring no actions are performed.

   [Implementation on RX65N]

   This requirement applies to the situation in Table 4.1, Bootloader Behavior, where “buffer area corrupted” and “user program area corrupted” coincide. In this state it is not possible to launch a program or to transition to a different state.

7. These requirements shall not be breached even in the presence of an accidental or malicious write to any MCU memory location (key store, program memory, RAM, etc.)

   [Implementation on RX65N]

   Overwriting of the bootloader is completely prohibited by the flash access window (FAW) register and the FSPR bit. The above requirement is also implemented in the bootloader itself. The bootloader cannot be corrupted even if a malicious third party were to exploit a fault in the user program and issue unauthorized ROM programming commands.

8. The bootloader shall support self-test of a new OTA image. If test execution fails, the bootloader shall roll back to the previous valid image. If test execution succeeds, the image shall be marked valid and the previous version erased.

   [Implementation on RX65N]

   The bootloader can validate an OTA image, but performing a self-test requires launching the new image and actually attempting network communication. But if the self-test fails it would be necessary to roll back to the previous image, which is inconsistent with an earlier requirement. We therefore ignore this requirement because it cannot be satisfied. Moving forward, it will probably be necessary to discuss this requirement with the definers and attempt to resolve the inconsistency, or to confirm the definition in the standard.

5. Bootloader Methods

Bootloader methods or approaches can be broadly divided into the following three types, depending on where the data is buffered when downloading new firmware:

Method 1: buffering destination = one of the banks (dual bank/bank swap mechanism used)
Method 2: buffering destination = one of the banks (dual bank/bank swap mechanism not used)
Method 3: buffering destination = external memory (EEPROM or serial flash memory)

In addition, the following options are available as the communication channel through which the firmware is downloaded:

A. Via UART
B. Via the file system on a device such as an SD card or USB flash drive
C. Via a network connection using Ethernet or Wi-Fi (over-the-air (OTA))

Firmware updates for Renesas MCUs are designed to allow the above to be combined flexibly.

The following combinations were available as of June 30, 2020:

Bootloader: 1. A

User program: 1. C

Sample code that utilizes these combinations is available on the following webpage:

https://github.com/renesas/amazon-freertos

Refer to the application note below for details such as operating instructions:

Implementing Firmware Updates on the RX Family (R01ANxxxx), which applies to the RX Family

5.1 Method 1: Buffering Destination = One of the Banks (Dual Bank/Bank Swap Mechanism Used)

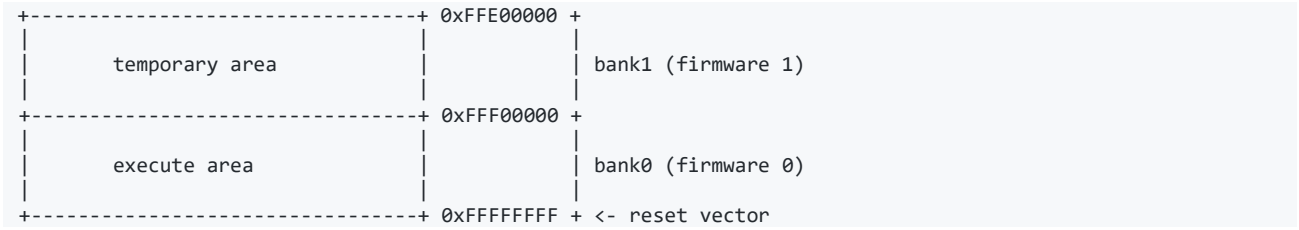
“Bootloader method 1: buffering destination = one of the banks (dual bank/bank swap mechanism used)” is described below, using the RX65N as a representative example.

5.1.1 Flash Memory Specifications

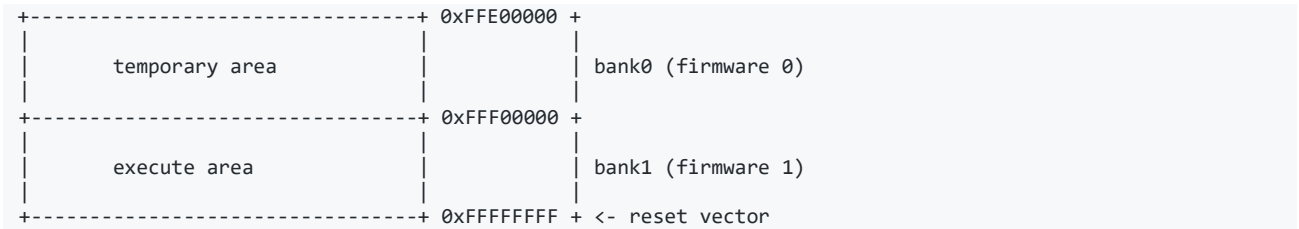
Since the RX65N generation, the code flash area of the flash memory has had a dual bank mechanism, allowing the physical memory capacity of 2 MB (max.) to be divided into two sections.

Note: In the following example, the physical memory size is assumed to be 2 MB. However, the RX65N is also available in a version with a ROM capacity of 1.5 MB. Therefore, you may need to replace the details of the description below with those that apply to the MCU version you are using.

The mode in which the available memory is divided into two sections on the RX65N is called “dual mode.” In dual mode a function called “bank swapping” is available. The banks are defined as follows in this document: bank0 is the execute area and bank1 is the buffer area (temporary area). The reset vector is always at an address in the range from 0xfffffffc to 0xffffffff. Therefore, after a bank swap occurs, the firmware that is launched after a reset is that stored in bank1 (firmware 1).



When a bank swap occurs (specifically, when a reset occurs after the bank swap command in R_FLASH_Control() of the flash API is issued), the code in bank0 and the code in bank1 are swapped (exchanged).



5.1.2 Review of Flash Memory Specifications: Additional Information 1

Without bank swapping it would be necessary to keep track of the firmware for bank0 and the firmware for bank1 at the time of compilation by identifying and managing which bank’s firmware was being compiled. By limiting execution to the execute area, this management becomes unnecessary. The dual bank mechanism is a commonly used method of implementing firmware updates on a single chip. To perform a firmware update with one chip requires a buffer area for storing the new image on the chip, and in most cases, this works out to “using half of the physical memory as a buffer,” as described above.

The RX65N has a maximum ROM capacity of 2 MB, but it should be noted that the amount available to the user is 1 MB. This limit does not apply unless there is a restriction on the cost of components that can be allocated as storage, such as external serial flash memory or EEPROM. If there is no such limit, it is a sound practice to use external serial flash memory or EEPROM as the buffer area and the entirety of the MCU’s internal flash memory as the user area, even if the MCU is one with a dual bank mechanism like the RX65N.

5.1.3 Review of Flash Memory Specifications: Additional Information 2

In order to rewrite the flash memory on a conventional MCU with on-chip flash memory, it was necessary to place in RAM a program for overwriting the firmware, cause the program counter to jump to RAM, and execute the flash memory rewrite command from RAM. This meant it was very difficult to rewrite the flash memory while maintaining system operation. Among MCUs with on-chip flash memory having a bank structure, such as the RX65N, there are products that can use background operation (BGO) to avoid this problem. By using BGO it is possible to allocate to and execute from bank0 a program that overwrites the firmware in bank1, for example. This makes it very easy to rewrite the flash memory while maintaining system operation if the interrupt vector and the interrupt handler at the jump destination of the interrupt vector are all placed in bank0.

5.2 Method 2: Buffering Destination = One of the Banks (Dual Bank/Bank Swap Mechanism Not Used)

We plan to add a description of this item in a future version of this document.

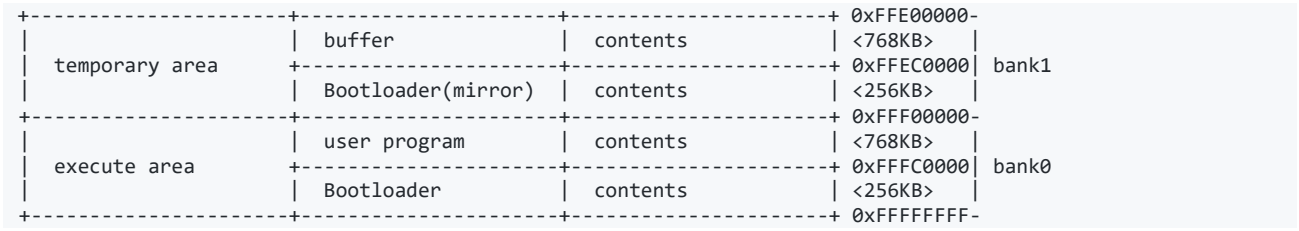
5.3 Method 3: Buffering Destination = External Memory (EEPROM or Serial Flash Memory)

We plan to add a description of this item in a future version of this document.

6. Bootloader Implementation

6.1 Memory Map Definition

The RX65N memory map for implementing the firmware update mechanism is defined as shown below. The firmware initially contains no data (blank chip), and the items indicated as “contents” below are replaced with new data names when data is written, as described below.



The 256 KB for the bootloader is user-dependent because it is based on the processing to needs to be executed at boot time. The allocated 256 KB may seem fairly large, but when initial loading of the firmware over a network is being considered, it could prove insufficient if a network stack such as Amazon FreeRTOS needs to be included in the bootloader. Conversely, if network loading is not necessary and a UART is used to load the initial firmware, a much smaller capacity (such as 16 KB) may be sufficient. This document assumes that initial loading of the firmware will be via a network and that 256 KB will suffice. In this case, the ROM capacity available for use by the user application is 768 KB. In practice, you will need to adjust the capacity actually allocated to the bootloader as appropriate to match your system.

“Bootloader(mirror)” is necessary to ensure that exactly the same bootloader operates even after the contents of bank0 and bank1 are exchanged following bank swapping. Since the reset vector is 0xFFFFFFFC as far as the MCU is concerned, it is necessary to place the same bootloader in the lower area of bank0 and bank1 in preparation for bank swap operation.

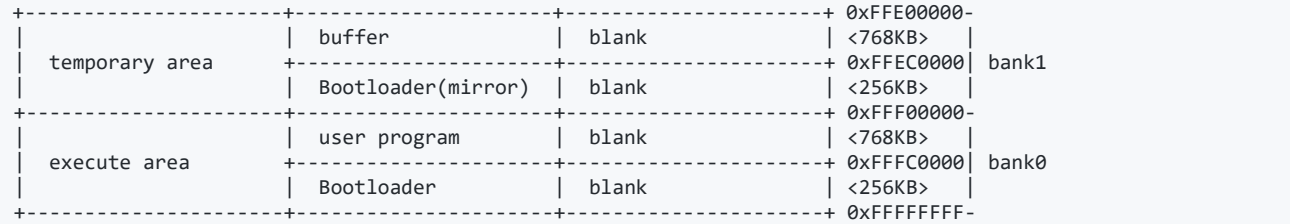
Since the first 0x300 bytes of the user program is the area where the bootloader stores the signature data, etc., for verifying the user program, it is necessary to shift the start address of the user program backward by 0x300 bytes when specifying sections. The data format is described in detail in another section of this document.

6.2 Firmware Write Sequence

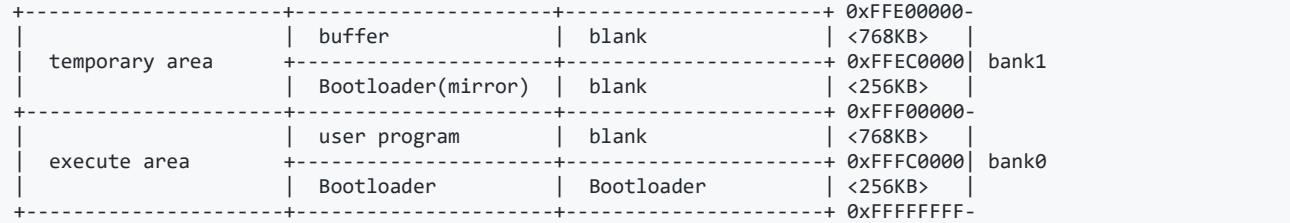
1. Prepare a blank chip.
2. Write Bootloader to the flash memory with a ROM writer.
3. Execute Bootloader and copy Bootloader(mirror) by self-programming.
4. Use Bootloader’s load function to write the initial firmware to the temporary area and execute a software reset.
5. Bootloader verifies the initial firmware in the temporary area.
6. Issue a software reset and execute a bank swap.
7. Bootloader(mirror), relocated to bank0, verifies the initial firmware in the temporary area relocated to bank0.
8. Lock Bootloader(mirror), the user program, and Bootloader using FAW.
9. Bootloader(mirror) jumps to the initial firmware (the initial firmware launches).
10. The initial firmware writes the next firmware to the temporary area.
11. The initial firmware verifies the next firmware in the temporary area.
12. Issue a software reset and execute a bank swap.
13. Bootloader, relocated to bank0, verifies the next firmware relocated to bank0 and erases the initial firmware remaining in bank1.
14. Bootloader jumps to the next firmware (the next firmware launches).
15. Repeat.

6.3 Firmware Write Sequence (Illustration)

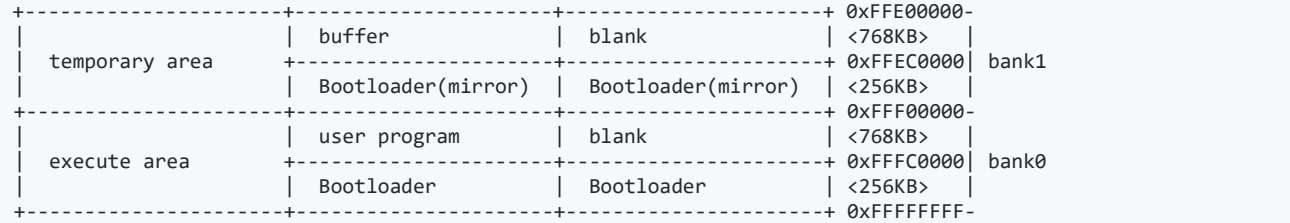
1. Prepare a blank chip.



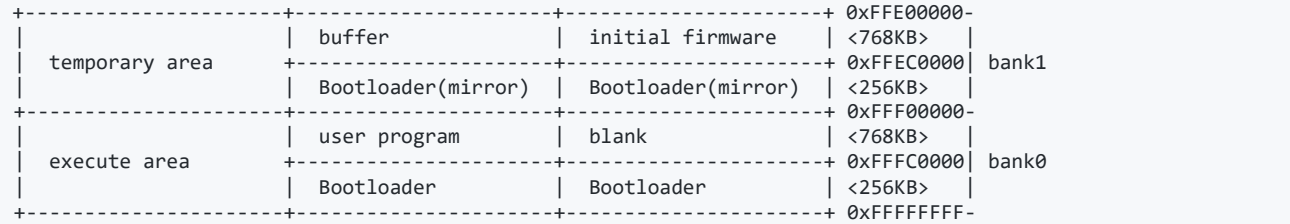
2. Write Bootloader to the flash memory with a ROM writer.



3. Execute Bootloader and copy Bootloader(mirror) by self-programming.

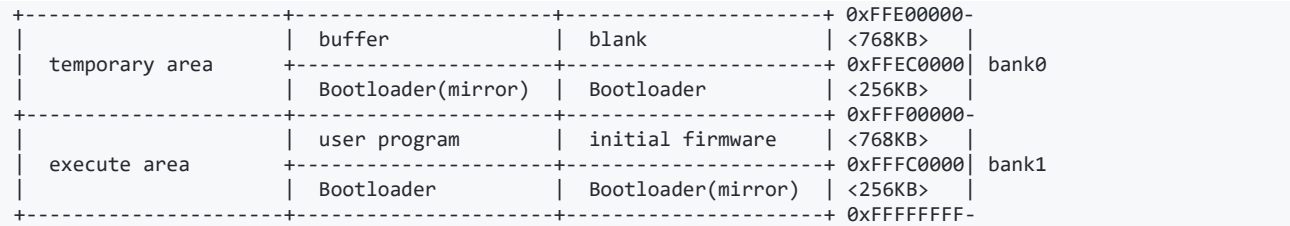


4. Use Bootloader's load function to write the initial firmware to the temporary area and execute a software reset.



5. Bootloader verifies the initial firmware in the temporary area.

6. Issue a software reset and execute a bank swap.



7. Bootloader(mirror), relocated to bank0, verifies the initial firmware in the temporary area relocated to bank0.

8. Lock Bootloader(mirror), the user program, and Bootloader using FAW.
   This operation can be accomplished using the flash module's R_FLASH_Control() function. Writing 0 to FSPR applies a permanent lock. Since this is potentially dangerous, the relevant line in the source code must be overwritten by the user in order to perform this operation. To apply the lock, change "faw.BIT.FSPR = 1;" to "faw.BIT.FSPR = 0;" in r_flash_type4.c.

   → Code for changing the setting of FSPR

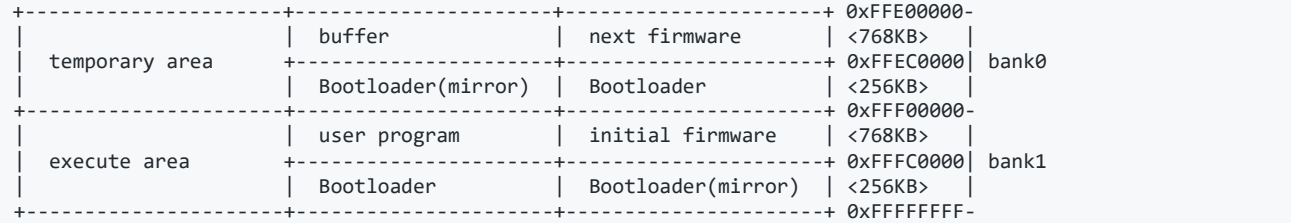
   In the bootloader, the flash API's R_FLASH_Control() function is used to change the setting of the FAW register after copying of Bootloader(mirror) finishes. Since it will be necessary to restore the chip to the blank state while experimenting with the sample code, the code for applying the lock is not implemented.

   → Location where FAW lock is applied

   When a bank swap is executed, it is unclear whether or not the FAW register's setting range would also be swapped, but no swapping takes place. Referring to the figure below, if the address range from 0xFFE00000 to 0xFFEC0000 is set in FAW and a bank swap is executed, the setting of FAW remains 0xFFE00000 to 0xFFEC0000.

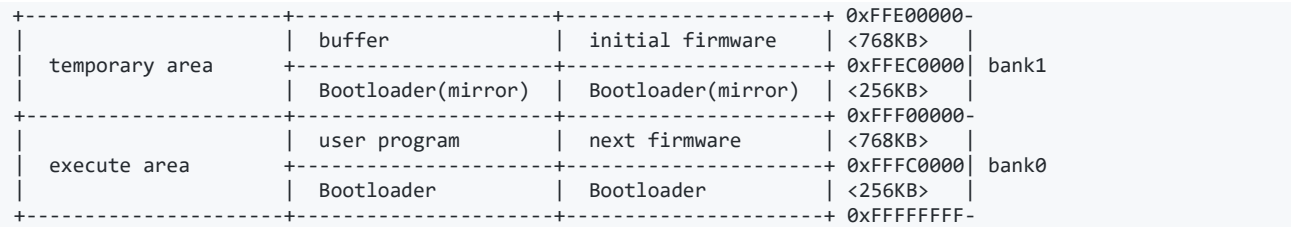
9. Bootloader(mirror) jumps to the initial firmware (the initial firmware launches).

10. The initial firmware writes the next firmware to the temporary area.

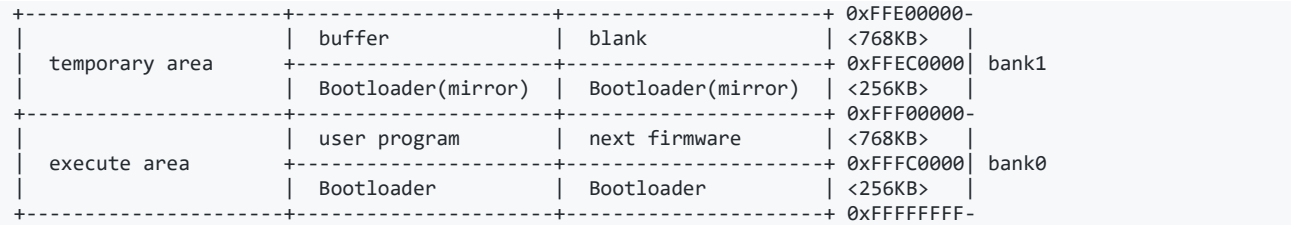


11. The initial firmware verifies the next firmware in the temporary area.

12. Issue a software reset and execute a bank swap.



13. Bootloader, relocated to bank0, verifies the next firmware relocated to bank0 and erases the initial firmware remaining in bank1.



14. Bootloader jumps to the next firmware (the next firmware launches).

15. Repeat.

7. Creating Firmware Update Data

A MOT (Motorola S record format) file, a commonly used firmware data format, does not have a mechanism for holding data (signature values, etc.) other than the actual data to be written to the device. Also, since MOT files consist of text data, the data size tends to be about twice that of an equivalent file in binary format. Firmware is distributed from a server when a firmware update is performed. Since the fee charged to the server increases as the amount of data transferred grows, MOT files are not suitable as a data format for distributing firmware updates.

On the other hand, there is as yet no suitable data format that solves the above problems and is as widely used as the MOT file format. Until a de facto standard is established in future, we will stipulate the use of Renesas Secure Update (RSU), a proprietary system established by Renesas.

7.1 Download Data Format

```
-------------------------------------------------------------------------------------------------------
output *.rsu
reference: https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/microchip-bootloader.html
-------------------------------------------------------------------------------------------------------
offset              component           contents name               length(byte)    OTA Image(Signed area)
0x00000000          Header              Magic Code                  7
0x00000007                              Image Flags                 1
0x00000008          Signature           Firmware Verification Type  32
0x00000028                              Signature size              4
0x0000002c                              Signature                   256
0x0000012c          Option              Dataflash Flag              4
0x00000130                              Dataflash Start Address     4
0x00000134                              Dataflash End Address       4
0x00000138                              Image Size                  4
0x0000013c                              Resereved(0x00)             196
0x00000200          Descriptor          Sequence Number             4               ---
0x00000204                              Start Address               4                |
0x00000208                              End Address                 4                |
0x0000020c                              Execution Address           4                |
0x00000210                              Hardware ID                 4                |
0x00000214                              Resereved(0x00)             236              |
0x00000300          Application Binary                              N               --- <- provided as mot
0x00000300 + N      Dataflash Binary                                M                   <- provided as mot
-------------------------------------------------------------------------------------------------------
Magic Code              : Renesas
Image Flags             : 0xff The application image contains no data, and cannot be executed under any
                          circumstances.
                          0xfe Indicates that the application image is undergoing updating or testing.
                          0xfc Indicates that the application image is undergoing initial installation.
                          0xf8 The application image is marked as valid.
                          0xf0 The application image is marked as invalid.
                          0xe0 The application image is marked as end of life.
Firmware Verification Type
                        : Identifier for specifying the firmware validation method.
                          Example: sig-sha256-ecdsa
Signature/MAC/Hash size : The data size of the signature value, MAC value, or hash value used to validate
                          the firmware.
Signature/MAC/Hash      : The signature value, MAC value, or hash value used to validate the firmware.
Dataflash Flag          : [Specific to RX MCUs] Flag indicating whether or not data for data flash is
                          included.
Dataflash Start Address : [Specific to RX MCUs] Data flash start address.
Dataflash End Address   : [Specific to RX MCUs] Data flash end address.
Image Size              : Size of the OTA image in bytes.
Sequence Number         : The sequence number must be incremented before a new OTA image is built.
                          The sequence number can be specified by the user in Renesas Secure Flash
                          Programmer. The bootloader uses this number to determine which image to boot from.
                          The range of valid values is 1 to 4294967295.
Start Address           : The start address of the OTA image on the device. This is set automatically by
                          Renesas Secure Flash Programmer, and does not need to be specified by the user.
End Address             : The end address of the OTA image, excluding the image trailer, on the device. This
                          is set automatically by Renesas Secure Flash Programmer, and does not need to be
                          specified by the user.
Hardware ID             : Unique hardware ID used by the bootloader to verify whether or not the OTA image
                          was built for the correct platform.
                          Example: 0x00000001    MCUROM_RX65N_2M_SB_64KB
```

7.2 MOT File Conversion Tool

A tool for converting standard MOT files to RSU files in Renesas' proprietary format is available. Please refer to the following document for specific instructions:

Implementing Firmware Updates on the RX Family (R01ANxxxx), which applies to the RX Family

7.3 Generating ECDSA-SHA256 Key Pairs with OpenSSL

The MOT file conversion tool requires that an ECDSA-SHA256 public key or private key be specified. These can be generated using OpenSSL. The method of generating ECDSA-SHA256 public and private keys with OpenSSL is shown below. Replace the various input values with your own values as necessary.

7.3.1 Create a CA Certificate

```
$ openssl ecparam -genkey -name secp256r1 -out ca.key
using curve name prime256v1 instead of secp256r1
$ openssl req -x509 -sha256 -new  -nodes -key ca.key -days 3650 -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:JP
State or Province Name (full name) [Some-State]:Tokyo
Locality Name (eg, city) []:Kodaira
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Renesas Electronics
Organizational Unit Name (eg, section) []:Software Development Division
Common Name (e.g. server FQDN or YOUR name) []:Ishiguro
Email Address []:hiroki.ishiguro.fv@renesas.com
```

7.3.2 Generate a Key Pair for Elliptic Curve Cryptography (Parameter: secp256r1)

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

7.3.3 Create a Key Pair Certificate

```
$ openssl req -new -sha256 -key secp256r1.keypair > secp256r1.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:JP
State or Province Name (full name) [Some-State]:Tokyo
Locality Name (eg, city) []:Kodaira
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Renesas Electronics
Organizational Unit Name (eg, section) []:Software Development Division
Common Name (e.g. server FQDN or YOUR name) []:Ishiguro
Email Address []:hiroki.ishiguro.fv@renesas.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

7.3.4 Use a CA Certificate to Create a Key Pair Certificate

```
$ openssl x509 -req -sha256 -days 3650 -in secp256r1.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out secp256r1.crt
Signature ok
subject=/C=JP/ST=Tokyo/L=Kodaira/O=Renesas Electronics/OU=Software Development
Division/CN=Ishiguro/emailAddress=hiroki.ishiguro.fv@renesas.com
Getting CA Private Key
```

7.3.5 Extract a Private Key Using Elliptic Curve Cryptography (Parameter: secp256r1)

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

7.3.6 Extract a Public Key Using Elliptic Curve Cryptography (Parameter: secp256r1)

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

7.4 Generating/Verifying Signatures for Testing with OpenSSL

The method of generating and verifying signature data using ECDSA-SHA256 public and private keys with OpenSSL is shown below.

7.4.1 First, Sign test.rsu with a Private Key Created with OpenSSL

```
$ openssl dgst -sha256 -sign secp256r1.privatekey test.rsu > signature.dat
```

7.4.2 Verify with a Public Key Created with OpenSSL

```
$ openssl dgst -sha256 -verify secp256r1.publickey -signature signature.dat test.rsu
Verified OK
```

8. Considerations Regarding Memory Protection During Mass Production

The above discussion covered protection during system operation. Generally speaking, there is also an approach in which the connection of debuggers or ROM writers is prohibited during mass production in order to boost security, and to accommodate it RX MCUs have the memory protection functions described in the document below. When the two approaches are used together, the security of the entire system can be enhanced.

https://www.renesas.com/us/en/doc/products/mpumcu/apn/rx/013/r01an0896ej0500-rx.pdf

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 31, 2020	—	First edition issued

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall 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.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

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/.