

Introduction

MCUboot is a secure bootloader for 32-bit MCUs. It defines a common infrastructure for the bootloader, defines system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software updates. MCUboot is an operating system and hardware independent and relies on hardware porting layers from the operating system it works with. The Renesas Flexible Software Package (FSP) integrates an MCUboot port starting from FSP v3.0.0. Users can benefit from using the FSP MCUboot Module to create a Root of Trust (RoT) for the system and perform secure booting and fail-safe application updates.

MCUboot is maintained by Linaro on the GitHub mcu-tools page: https://github.com/mcu-tools/mcuboot. There is a \docs folder that holds the documentation for MCUboot in .md file format. This application note refers to the above-mentioned documents wherever possible and is intended to provide additional information that is related to using the Renesas FSP MCUboot Module.

For RA Family RA6M4, RA6M5, and RA6E1 MCU Groups, the internal code flash has a dual bank feature, which can be used to simplify and accelerate firmware updates. This dual bank feature is supported by FSP v3.6.0. This application note demonstrates a secure bootloader design using this dual bank feature for a Non-TrustZone environment based on RA6M4.

Example projects using the EK-RA6M4 evaluation kit are provided in this application project. Users can review the flash layout for RA6E1 and RA6M5 and port the application to RA6E1 and RA6M5. In addition, steps on how to master an application to use with the bootloader and how to update it to a new application are provided. Users can follow these steps to recreate the reference bootloader and link the example application projects included in this application project to use the bootloader.

If you are interested in a secure bootloader design using the MCUboot module with RA6 internal flash in linear mode, please refer to the application project R11AN0497 (<u>Search | Renesas Electronics Corporation</u>).

Required Resources

Development tools and software

- The e² studio IDE v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- SEGGER J-link[®] USB driver

The above three software components, the FSP, J-Link USB drivers, and e² studio, are bundled in a downloadable platform installer available on the FSP webpage at <u>renesas.com/ra/fsp</u>.

- Python v3.9 or later- https://www.python.org/downloads/
- Renesas Flash Programming (RFP) v3.16.00 or later
 <u>https://www.renesas.com/us/en/software-tool/renesas-flash-programmer-programming-gui</u>

Hardware

- EK-RA6M4, Evaluation Kit for RA6M4 MCU Group http://www.renesas.com/ra/ek-ra6m4
- Workstation running Windows[®] 10
- Two USB device cables (type-A male to micro-B male)
- One USB to TTL Serial 3.3-V UART Converter

Prerequisites and Intended Audience

Users of this application project should have some experience with the Renesas e² studio. Users should read the **MCUboot Port** section of the FSP User's Manual as well as the MCU Hardware User's manual



Flash Memory section prior to working with this application project. Users should also have some knowledge of cryptography. Prior knowledge of Python usage is also helpful.

The intended audience includes product developers, product manufacturers, product support, or end users who are involved with designing application systems involving the usage of a secure bootloader.

Using this Application Note

Section 1 is an overview of the code flash dual bank feature of RA6M4 and RA6M5 MCUs. Users who are familiar with the MCU dual bank features can skip this section.

Section 2 covers the general flow of architecting a system using the FSP MCUboot module. For example, memory configuration for a code flash dual bank-based bootloader using MCUboot is introduced in this section.

Section 3 covers the introduction to the example projects included in this application project. Users should review this section to understand how to use the example projects.

Section 4 covers the steps to create a secure bootloader using the code flash dual bank feature and MCUboot module. Users who will customize the bootloader should review this section to understand how the bootloader is structured.

Section 5 provides the steps to configure and sign an application to use the bootloader created in section 4. The examples of projects included are used in this section.

Section 6 provides instructions on how to debug and boot the primary application project and update it to a new image. Users who will use the dual bank feature for the first time should review this section as it includes information about:

- Debugging and booting the primary application
- Downloading a new image using the primary image downloader
- Booting the new image

Section 7 covers the production support of provisioning the new MCU with the bootloaders and the initial application.

Section 8 provides instructions on how to run the included example projects. Users who are familiar with bootloader design using MCUboot can go to this section for a quick evaluation of the included example projects.



Contents

1.	Code Flash Dual Bank Feature	5
1.1	RA6M4 and RA6E1 MCU Group Code Flash Configuration	5
1.2	RA6M5 MCU Group Code Flash Configuration	8
1.3	Option-Setting Memory	11
1.3.1	Code Flash Bank Mode	
1.3.2	Startup Bank Selection	
1.3.3	Bank Swap	
1.3.4	Code Flash Block Protection	
2.	Using the Code Flash Dual Bank Feature with MCUboot Overview	14
2.1	MCUboot Functionalities Overview	
2.2	Using MCUboot for Code Flash Dual Bank Mode	
2.2.1	Use Direct XIP Firmware Update Mode	
2.2.2	Memory Configuration Overview with Dual Bank and MCUboot	
2.3	Designing Bootloader and Initial Primary Application Overview	
2.4	Migrating an Existing Code Flash Linear Mode MCUboot-Based System	
3.	Guidelines for Using the Example Projects Included	16
3.1	Example Projects with Bootloader	
3.2	Example Projects without Bootloader	
4.	Creating the Bootloader Project using Code Flash Dual Bank Mode	17
4. 4.1	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project	17
4. 4.1 4.2	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method	17
4. 4.1 4.2 4.3	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver	
4. 4.1 4.2 4.3 4.4	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code	
4. 4.1 4.2 4.3 4.4 4.5	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project	
4. 4.1 4.2 4.3 4.4 4.5 4.6	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment	
4. 4.1 4.2 4.3 4.4 4.5 4.6 4.7	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support	
 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5. 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project	
 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5. 5.1 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project Configure the Application Project to Use the Bootloader	
 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5.1 5.2 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project	
4. 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5. 5.1 5.2 5.3	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project	
 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5.1 5.2 5.3 6. 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project Configure the Application Project to Use the Bootloader Signing the Application Image Preparation for Production Support Booting the Primary Application and Updating to a New Image	
 4. 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5.1 5.2 5.3 6. 6.1 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project Configure the Application Project to Use the Bootloader Signing the Application Image Preparation for Production Support	
 4. 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5. 5.1 5.2 5.3 6. 6.1 6.2 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project Configure the Application Project to Use the Bootloader Signing the Application Image Preparation for Production Support Booting the Primary Application and Updating to a New Image Prepare a Secondary Image Set Up the Hardware	
 4. 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5.1 5.2 5.3 6. 6.1 6.2 6.3 	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project Configure the Application Project to Use the Bootloader Signing the Application Image Preparation for Production Support Booting the Primary Application and Updating to a New Image Prepare a Secondary Image Set Up the Hardware Erase the MCU	
4. 4.1 4.2 4.3 4.4 4.5 4.6 4.7 5. 5.1 5.2 5.3 6. 6.1 6.2 6.3 6.3.1	Creating the Bootloader Project using Code Flash Dual Bank Mode Include the MCUboot Module in the Bootloader Project Configure the Memory Configuration and Authentication Method Configure the MbedTLS Crypto Only Module and the Flash Driver Add the Boot Code Compile the Bootloader Project Configure the Python Signing Environment Prepare for Production Support Configuring and Signing an Application Project Configure the Application Project to Use the Bootloader Signing the Application Image Preparation for Production Support Booting the Primary Application and Updating to a New Image Prepare a Secondary Image Set Up the Hardware Erase the MCU Use the Renesas Flash Programmer	



6.3.3	Use Renesas Device Partition Manager	39
6.4	Start the Debug Session	41
6.5	Program the New Application Using the Primary Application Downloader	43
6.6	Boot the New Application	45
7.	Production Support Considerations	46
7.1	Protect the Bootloader using Flash Block Protection	46
7.2	Provision the Bootloaders and the Initial Application to MCU	
8.	Compile and Exercise the Included Example Bootloader and Application Projects.	51
8.1	Using USB as the Download Interface	51
8.2	Using the UART as the Download Interface	51
9.	References	52
10.	Website and Support	53
Rev	ision History	54



1. Code Flash Dual Bank Feature

For RA6M4, RA6E1, and RA6M5 MCU groups, the internal flash memory can operate in linear mode or dual bank mode. In linear mode, the code flash memory is used as one area. In dual bank mode, the code flash memory is divided into two areas. In code flash dual bank mode, the bank swap function can be used to boot into a new application for a system that includes a bootloader.

1.1 RA6M4 and RA6E1 MCU Group Code Flash Configuration

Using the 1-Mbyte product as an example, the code flash memory in linear mode for RA6M4 includes the blocks shown in Figure 1.



Figure 1. RA6M4 and RA6E1 Code Flash Memory in Linear Mode

Upper Bank Address in Code Flash Linear Mode

In code linear mode, the upper bank starting address is half of the code flash size. For example, for the 1-Mbyte RA6M4 and RA6E1 MCU used in this example project, the starting address of the upper bank address is 0x80000. The upper bank linear mode address is used when downloading the upper bank bootloader using MCUboot in code flash dual bank mode.



Using the 1-Mbyte product as an example, the code flash memory in dual bank mode includes the blocks shown in Figure 2. The default configuration is highlighted in the red box.



Figure 2. RA6M4 and RA6E1 Code Flash Memory in Dual Bank Mode



Table 1 is a summary of the code flash blocks in linear and dual bank mode. The upper bank address in dual bank mode is 0x200000, regardless of the code flash size. This address should be used with the application image downloader.

Table 1.	RA6M4	and RA6E1	Code Flash
----------	-------	-----------	------------

	Code Flash Range Address	
Product	Linear	Dual
1-Mbyte	0x0000_0000 to 0x000F_FFFF	Lower side bank:
product		0x0000_0000 to 0x0007_FFFF
		Upper side bank:
		0x0020_0000 to 0x0027_FFFF
768-Kbytes	0x0000_0000 to 0x000B_FFFF	Lower side bank:
product		0x0000_0000 to 0x0005_FFFF
		Upper side bank:
		0x0020_0000 to 0x0025 FFFF
512 Kbytes	0x0000_0000 to 0x0007_FFFF	Lower side bank:
product		0x0000_0000 to 0x0003_FFFF
		Upper side bank:
		0x0020_0000 to 0x0023_FFFF

Figure 3 is the code flash block structure for the RA6M4 and RA6E1. The code flash erasing and programming minimum unit is the code flash block size. The block numbering scheme is used in the block protection design.



Figure 3. RA6M4 and RA6E1 Code Flash Block Structure



1.2 RA6M5 MCU Group Code Flash Configuration

Using the 2-Mbyte product as an example, the code flash memory in linear mode for the RA6M5 includes the blocks shown in Figure 4.





Upper Bank Address in Code Flash Linear Mode

In code linear mode, the upper bank starting address is half of the code flash size. For example, for the 2-Mbyte RA6M5 MCUs, the starting address of the upper bank address is 0x100000. The upper bank linear mode address is used when downloading the upper bank bootloader when using MCUboot in code flash dual bank mode.

Using the 2-Mbyte product as an example, the code flash memory for the RA6M5 in dual bank mode includes the blocks shown in Figure 5. The default configuration is highlighted in the red box.





Figure 5. RA6M5 Code Flash Memory in Dual Bank Mode



Table 2 is a summary of the code flash blocks in linear and dual bank mode for the RA6M5. The upper bank address in dual bank mode is 0x200000, regardless of the code flash size. This address should be used with the application image downloader.

Table 2. RA6M5 Cod	le Flash
--------------------	----------

	Code Flash Range Address	
Product	Linear	Dual
2-Mbytes product	0x0000_0000 to 0x001F_FFFF	Lower side bank:
		0x0000_0000 to 0x000F_FFFF
		Upper side bank:
		0x0020_0000 to 0x002F_FFFF
1-MByte product	0x0000_0000 to 0x000F_FFFF	Lower side bank:
		0x0000_0000 to 0x0007_FFFF
		Upper side bank:
		0x0020_0000 to 0x0027_FFFF

Figure 6 is the code flash block structure for RA6M5. The code flash erase and programming minimum unit is the code flash block size. The block numbering scheme is used in the block protection design.



Figure 6. RA6M5 Code Flash Block Structure



1.3 Option-Setting Memory

The description in this section applies to both RA6M4 and RA6M5. The Option-Setting Memory of the RA6M4 and RA6M5 MCUs determines the state of the MCU after a reset. Several property settings that relate to the code flash mode are described in this section.

Address		
0x0100_A2CC to 0x0100_A2FF	Reserved area	
0x0100_A2C0 to 0x0100_A2CB	Block Protect Setting Register Select (BPS_SEL)	
0x0100_A294 to 0x0100_A2BF	Reserved area	
0x0100_A290 to 0x0100_A293	Bank Select Register Select (BANKSEL_SEL)	
0x0100_A284 to 0x0100_A28F	Reserved area	
0x0100_A280 to 0x0100_A283	Option Function Select Register 1 Select (OFS1_SEL)	
0x0100_A26C to 0x0100_A27F	Reserved area	O
0x0100_A260 to 0x0100_A26B	Permanent Block Protect Setting Register Secure (PBPS_SEC)	Secure region
0x0100_A24C to 0x0100_A25F	Reserved area	
0x0100_A240 to 0x0100_A24B	Block Protect Setting Register Secure (BPS_SEC)	
0x0100_A214 to 0x0100_A23F	Reserved area	
0x0100_A210 to 0x0100_A213	Bank Select Register Secure (BANKSEL_SEC)	
0x0100_A204 to 0x0100_A20F	Reserved area)
0x0100_A200 to 0x0100_A203	Option Function Select Register 1 Secure (OFS1_SEC)	
0x0100_A1EC to 0x0100_A1FF	Reserved area	
0x0100_A1E0 to 0x0100_A1EB	Permanent Block Protect Setting Register (PBPS)	
0x0100_A1CC to 0x0100_A1DF	Reserved area	
0x0100_A1C0 to 0x0100_A1CB	Block Protect Setting Register (BPS)	
0x0100_A194 to 0x0100_A1BF	Reserved area	
0x0100_A190 to 0x0100_A193	Bank Select Register (BANKSEL)	
0x0100_A184 to 0x0100_A18F	Reserved area	
0x0100_A180 to 0x0100_A183	Option Function Select Register 1 (OFS1)	
0x0100_A138 to 0x0100_A17F	Reserved area	
0x0100_A134 to 0x0100_A137	Startup Area Setting Register (SAS)	
0x0100_A114 to 0x0100_A133	Reserved area	— Secure region
0x0100_A110 to 0x0100_A113	Dual Mode Select Register (DUALSEL)	
0x0100_A104 to 0x0100_A10F	Reserved area	
0x0100_A100 to 0x0100_A103	Option Function Select Register 0 (OFS0)	

Figure 7. Option-Setting Memory



1.3.1 Code Flash Bank Mode

The register that configures the code flash bank mode is in the Option-Setting Memory of the MCU. As shown in Figure 7, the Dual Mode Select dual bank select register DUALSEL is located at 0x0100A110.

The DUALSEL register defines whether the code flash is in linear or dual bank mode. For a blank MCU, the code flash is in linear mode. The user application can change this configuration. With current FSP support, this register is set up at compile time by configuring the property under the BSP tab (refer to Figure 35).



Figure 8. Register Configuration for Code Flash Dual Bank Mode

1.3.2 Startup Bank Selection

The description in this section applies to RA6E1, RA6M4, and RA6M5 Family MCUs. Bank 0 is the lower bank for a blank RA6M4 or RA6M5 MCU as defined by the Bank Select registers shown in Figure 9.



Figure 9. Bank 0 is Default at Address 0x00000000



Only secure developers can program BANKSEL_SEC and BANKSEL_SEL registers. The BANKSEL_SEC register is for secure developers, and the BANKSEL register is for non-secure developers.

BANKSEL_SEL controls whether the BANKSEL or BANKSEL_SEC setting is applied. When BANKSEL_SEL is 0xFFFFFF8, the setting in BANKSEL is used. When BANKSEL_SEL is 0xFFFFFFFF, the setting in BANKSEL_SEC is used. For Non-Trust Zone-based Flat projects, BANKSEL_SEL selects the corresponding bits in the BANKSEL_SEC register.

1.3.3 Bank Swap

Startup bank selection provides a way to safely update the program by selecting a bank area to be started in dual mode during a reset.



Figure 10. Example of Startup Bank Selection (For Products with 1 Mbyte of Code Flash Memory)

Bank selection can be changed at runtime through the FSP API. The BANKSWP bits in the BANKSEL register can be changed at the application level. The FSP flash driver provides the R_FLASH_HP_BankSwap() API to facilitate this action. This API is automatically called from the FSP MCUboot module. The swap will take effect after the next reset.

1.3.4 Code Flash Block Protection

The RA6M4 and RA6M5 MCUs implement a security function to protect the code flash against illicit tampering with or reading out of data in flash memory. The registers that define this security function reside in the Option-Setting Memory. The code flash memory can be temporarily or permanently protected from programming/erasure operations.

The registers that support the temporary code flash block protection reside in the Option-Setting Memory:

auuress.	BPS: 0x0100_A1C0, 0x0100_A1C4, 0x0100_A1C8 BPS_SEC: 0x0100_A240, 0x0100_A244, 0x0100_A248 BPS_SEL: 0x0100_A2C0, 0x0100_A2C4, 0x0100_A2C8	
Bit position:	31	0
Bit field:		
Value after reset:	User setting ^{*1}	
Note 1. The va	lue in a blank product is 0xFFFFFFFF. It is set to the value written by your application.	

Figure 11. Registers Related to Temporary Code Flash Block Protection

Only secure developers can program the BPS_SEC and BPS_SEL registers. The BPS_SEC register is for secure developers, and the BPS register is for non-secure developers. The applied setting value is determined by the value of the corresponding bit in the BPS_SEL register. BPS_SEL controls whether the BPS or BPS_SEC setting is applied. When BPS_SEL is 0xFFFFFF8, the setting in BPS is used. When



BPS_SEL is 0xFFFFFFF, the setting in BPS_SEC is used. For Non-Trust Zone-based Flat projects, BSP_SEL selects the corresponding bits in the BSP_SEC register. The BPS and BPS_SEC registers invalidate the programming and erasure to the code flash memory. When a BPS/BPS_SEC bit is 0, the programming and erasure to the corresponding block are invalid.

These registers can be set by configuring the BSP stack in the RA configurator, as shown in Figure 84 and Figure 85.

The registers that support the permanent code flash block protection reside in the Option-Setting Memory:

Address:	PBPS: 0x0100_A1E0, 0x0100_A1E4, 0x0100_A1E8 PBPS_SEC: 0x0100_A260, 0x0100_A264, 0x0100_A268	
Bit position:	31 0	
Bit field:		
Value after reset:	User setting*1	
Note 1. The va	lue in a blank product is 0xFFFFFFFF. It is set to the value written by your application.	

Figure 12. Registers Related with Permanent Code Flash Block Protection

Only secure developers can program the PBPS_SEC and PBPS_SEL registers. The PBPS_SEC register is for secure developers, and the PBPS register is for non-secure developers. The applied setting value is determined by the set value of the corresponding bit in the PBPS_SEL register. PBPS_SEL controls whether the BPS or BPS_SEC setting is applied. When PBPS_SEL is 0xFFFFFF8, the setting in PBPS is used. When PBPS_SEL is 0xFFFFFF8, the setting in PBPS is used. When PBPS_SEL is 0xFFFFFFF7, the setting in PBPS_SEC is used. For Non-Trust Zone-based Flat projects, PBSP_SEL selects the corresponding bits in the PBSP_SEC register. The PBPS and PBPS_SEC registers invalidate the programming and erasure to the code flash memory. When a PBPS/PBPS_SEC bit is 0, the programming and erasure to the corresponding block are invalid.

Setting of these registers can be achieved by configuring the BSP Properties in the RA configurator as shown in Figure 86 and Figure 87.

2. Using the Code Flash Dual Bank Feature with MCUboot Overview

MCUboot evolved out of the Apache Mynewt bootloader, which was created by runtime.io. MCUboot was then acquired by JuulLabs in November 2018. The MCUboot Github repo was later migrated from JuulLabs to the <u>mcu-tools github project</u>. In year 2020, MCUboot was moved under the Linaro Community Project umbrella as an open-source project.

2.1 MCUboot Functionalities Overview

MCUboot handles the firmware authenticity check after start-up, and the firmware switch part of the firmware update process. Downloading the new version of the firmware is out-of-scope for MCUboot. Typically, downloading the new version of the firmware is a functionality that is provided by the application project itself. This application project provides an example of downloading a new image using the XModem protocol from the application project.

The functionality of MCUboot during booting and updating follows the process below:

The bootloader starts when the CPU is released from reset. If there are images in the Secondary App memory marked as to be updated, the bootloader performs the following actions:

- 1. The bootloader authenticates the Secondary image.
- 2. Upon successful authentication, the bootloader switches to the new image based on the update method
- selected. Available update methods supported by FSP are overwrite, swap, and direct XIP.
- 3. The bootloader boots the new image.

If there is no new image in the Secondary App memory region, the bootloader authenticates the Primary applications and boots the Primary image.

The authentication of the application is configurable in terms of the authentication methods and whether the authentication is to be performed with MCUboot. If authentication is to be performed, the available methods are RSA or ECDSA. The firmware image is authenticated by hash (SHA-256) and digital signature validation.



The public key used for digital signature validation can be built into the bootloader image or provisioned into the MCU during manufacturing. In the examples included in this application project, the public key is built into the bootloader images.

There is a signing tool included with MCUboot: <u>imgtool.py</u>. This tool provides services for creating root keys, managing key values, and signing and packaging an image with version controls. Read the MCUboot documentation to understand and use these operations.

2.2 Using MCUboot for Code Flash Dual Bank Mode

The FSP supports overwrite, swap, and direct XIP (execute-in-place) update mode. For flash dual bank mode, only direct XIP mode is supported. The benefits of using code flash dual bank mode in a system including a bootloader are concurrent download of new images and faster switching to the new image, in addition to the safety features provided by the MCUboot module as explained in section 2.2.1.

2.2.1 Use Direct XIP Firmware Update Mode

When using direct XIP mode with code flash in linear mode, the active image slot alternates with each firmware update. If this update method is used, then two firmware update images must be generated: one of them is linked to be executed from the primary slot memory region, and the other is linked to be executed from the secondary slot. Direct XIP is supported in FSP versions 3.6.0 and later.

- Advantages:
 - Faster boot time, as there is no overwrite or swap of application images needed.
 - Fail-safe and resistant to power-cut failures.
- Disadvantages:
 - Added application-level complexity to determine which firmware image needs to be downloaded.
 - Encrypted image support is not available.

For an overview and usage of other update modes, refer to R11AN0497 and the MCUboot design page:

https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md

When using direct XIP mode with code flash in dual bank mode, both primary and secondary images are linked to be executed from the primary slot memory region.

Note: For Direct XIP mode, downgrade prevention is supported from the MCUboot side. When using flash dual bank mode, the update image needs to have a version number higher than the current primary image.

2.2.2 Memory Configuration Overview with Dual Bank and MCUboot

The FSP MCUboot module with Flash Dual Bank mode needs a bootloader for both the lower bank and the upper bank, as shown in Figure 13. In addition, the memory allocation for the bootloader and application image must be identical.



Figure 13. Memory Architecture Using Flash Dual Bank Mode and MCUboot



2.3 Designing Bootloader and Initial Primary Application Overview

A bootloader is typically designed with the initial primary application. The following general guidelines apply to designing the bootloader and the initial primary application:

- Develop the bootloader and analyze the MCU memory resource allocation needed for the bootloader and the application. The bootloader memory usage is influenced by the application image update mode, signature type, and whether to validate the Primary Image as well as the cryptographic library used.
- Develop the initial primary application, perform the memory usage analysis, compare it with the bootloader memory allocation for consistency, and adjust it as needed.
- Determine the bootloader configurations in terms of image authentication and new image update mode. This may result in the adjustment of the memory allocated definition in the bootloader project.
- Sign the application image. The signing command is output to the <bootloader project>\Debug\>bootloader project>.bld file. The application image can use a Build Variable to access this .bld file. The IDE tools use the signing command to sign the application and generate a binary file for downloading to the MCU.
- Test the bootloader and the initial primary application.

The above guidelines are demonstrated in the walk-through sections in this application note.

2.4 Migrating an Existing Code Flash Linear Mode MCUboot-Based System

Users can follow the general steps below to migrate an MCUboot-based application system from code flash linear mode to code flash dual bank mode:

- 1. Updates for the bootloader project:
 - A. Update the code flash mode from linear mode to dual mode in the BSP tab, as shown in Figure 35.
 - B. Update the application image code flash memory allocation if needed. See section 4.2 for details.
- 2. Updates for the application projects:
 - A. For image downloader implementation, the image download address needs to be updated. Refer to the \src\header.h in the example application project to understand where the updates need to happen.
 - B. For development purposes, the debug configuration for the primary application needs to be updated. Refer to the debug configuration for the app_primary_usb project under the \example projects with bootloader folder to perform the update.
 - C. For production support, the scripts to generate the .srec file using the signed image need to be updated. Refer to section 5.3 to understand the updates needed.

3. Guidelines for Using the Example Projects Included

Unzip ra6-advanced-mcuboot-flash-dual-bank.zip to unpack the example projects included in this application project.







3.1 Example Projects with Bootloader

Folder \example_projects_with_bootloader includes a bootloader, which supports the flash dual bank feature, as well as example applications using USB or UART as the communication channel to download new application images that are configured to use the bootloader included in this folder. Users with experience working with the MCUboot module can follow section 8 to directly exercise these example projects. The corresponding subfolders are:

- ra_mcuboot_ra6m4_dualbank: Bootloader, which enables dual bank and direct XIP update mode.
- app_primary_usb: Primary application, which is configured to work with the bootloader and implements XModem over USB VCOM to download a new application image. FreeRTOS is used with two threads; one thread blinks the three LEDs on EK-RA6M4, while the other thread downloads the new application image concurrently.
- app_secondary_usb: Secondary application, which implements the same functionality as app_primary_usb except only the blue and green LEDs are blinked.
- app_primary_uart: Primary application, which is configured to work with the bootloader and implements XModem over UART to download a new application image. FreeRTOS is used with two threads; one thread blinks the three LEDs on EK-RA6M4, while the other thread downloads the new application image concurrently.
- app_secondary_uart: Secondary application, which implements the same functionality as app_primary_uart except only the blue and green LEDs are blinked.

3.2 Example Projects without Bootloader

Folder \example_projects_without_bootloader includes standalone example projects that a user can configure to use the bootloader project, following section 5. Note that these application projects do not run correctly if the flash dual bank mode is not enabled because the image downloader routine included assumes the location of the new image is in the upper bank of the RA6M4 code flash. The subfolders are:

- app_primary_usb: Same functionality as \example_projects_with_bootloader\app_primary_usb, except it is not configured to work with the bootloader.
- app_primary_uart: Same functionality as \example_projects_with_bootloader\app_primary_uart, except it is not configured to work with the bootloader.

A user can also use a customized application project that implements image downloading and follow

section 5 to use the bootloader.

4. Creating the Bootloader Project using Code Flash Dual Bank Mode

This section demonstrates the creation process of the bootloader project utilizing MCUboot and the Flash Dual Bank Mode with the RA6M4 running in non-TrustZone mode.

4.1 Include the MCUboot Module in the Bootloader Project

Follow the below steps to start the bootloader project creation and include the MCUboot module in the project:

1. Launch e² studio and start a new C/C++ Project. Click **File** > **New** > **C/C++ Project**.

i workspace - e² studio	
File dit Source Refactor Navigate Search Project Renesas Views Run	Window Help
New Alt+Shift+N > Renesas C/C+-	- Project >
Open File 🖬 Makefile Proje	t with Existing Code
C/C++ Project from File System	Create a new C or C++ project
Recent Files > 📑 Project	

Figure 15. Start a New Project



2. Choose Renesas RA->Renesas RA C/C++ Project. Click Next.

New C/C++ Pr Templates for N	oject – D × ew C/C++ Project
All CMake Make <u>Renesas Debug</u> Renesas RA	Renesas RA C/C++ Project Create an executable or static library C/C++ project for Renesas RA.
?	< Back Next > Finish Cancel

Figure 16. Choose Renesas RA C/C++ Project

- 3. Provide the project name **ra_mcuboot_ra6m4_dualbank** in the next screen. Click **Next**.
- 4. In the next screen, choose EK-RA6M4 for Board and click Next.

Board:	EK-RA6M4	~
Device:	R7FA6M4AF3CFB	



5. When the following screen appears, select Flat (Non-TrustZone) Project.

Renesas RA C/C++ Project			
Renesas RA C/C++ Project		-	
Project Type Selection			
 Project Type Selection Flat (Non-TrustZone) Project Renesas RA device project without TrustZone separation All code, data and peripheral settings will be configured in this project Renesas RA device will remain in secure mode EDMAC RAM buffers will automatically be placed in non-secure RAM TrustZone Secure Project Renesas RA device project for TrustZone secure execution All code, data and peripherals placed in this project will be initialized as secure Secure project settings such as TrustZone partitions, linker maps and a list of secure peripherals will be passed to a selected non-secure project Atter initialization, a call to the non-secure startup handler will be made TrustZone Non-secure Project Renesas RA device project for TrustZone non-secure execution All code, data and peripherals placed in this project will be initialized as non-secure Must be associated with a secure project or smart bundle Non-secure startup handler will be called after secure code initialization 			
0	< Back Next > Finish	Cancel	

Figure 18. Choose Flat Project as Project Type



6. Choose Executable for Build Artifact Selection and No RTOS. Click Next.

	– 🗆 X	
RTOS Selection		
No RTOS	~	
< Back Next > F	inish Cancel	
	RTOS Selection No RTOS	

Figure 19. Choose to Build Executable and No RTOS

7. Choose **Bare Metal – Minimal** for the Project Template in the next screen and click **Finish** to establish the initial project.

Project Template	e Selection	
•	Bare Metal - Blinky Bare metal FSP project that includes BSP and will blink LEDs if available. This project will initialize clocks, pins, stacks, and the C runtime environment.	

Figure 20. Choose the Project Template

8. When the following prompt opens, click **Open Perspective**.

Open Associated Perspective?		×	
Open the FSP Configuration perspective?			
Remember my decision			
	Open Perspective	No	

Figure 21. Choose Open the FSP Configuration Perspective

The project is then created and the bootloader project configuration is displayed. 9. Select the **Pins** tab and uncheck **Generate data** for **RA6M4 EK**.

Select Pin Configuration		📑 Export to CSV file 🔚 Configure Pin Driver Warnings
RA6M4 EK	<u>Manage configurations</u>	Generate data: g_bsp_pin_cfg

Figure 22. Uncheck Generate data for RA6M4 EK Pin Configuration



Use the pull-down menu to switch from **RA6M4 EK** to **R7FA6M4AF3CFB.pincfg** for the **Select Pin Configuration** option, then select the **Generate data** check box and enter **g_bsp_pin_cfg**. Note that here, we choose to use this configuration, which has fewer peripherals/pins configured since the bootloader does not use the extra peripheral or GPIO pins configured in the **RA6M4 EK** configuration. This also reduces some memory usage for the bootloader project.

Select Pin Configuration		🙀 Export to CSV file Configure Pin Driver Warnings
R7FA6M4AF3CFB.pincfg v	Manage configurations	Generate data: g_bsp_pin_cfg

Figure 23. Select g_bsp_pin_cfg and Generate data g_bsp_pin_cfg

10. Once the project is created, click the **Stacks** tab on the RA configurator. Add **New Stack -> Bootloader** -> **MCUboot**.



Figure 24. Add the MCUboot Port

11. Next, configure the **General** properties of **MCUboot**. We will resolve the errors in the configurator in the following steps.

For the MCUboot module, configure the **Update Mode** to **Direct XIP** and **Number of Images Per Application** to **1**.

Image: Stacks of the second	HAL/Common Stacks		
Clocks Pins Interrupts Event Links Stacks Components BSP Clocks Pins Interrupts Event Links Stacks Components ns Console Properties Smart Browser Property Value Common General Custom mcuboot_config.h Upgrade Mode Direct XIP Validate Privention (Overwrite Only) Disabled Number of Images Per Application 	g_ioport I/O Port (r_ioport)	∰ MCUboot	
BSP Clocks Pins Interrupts Event Links Stacks Components ns Console Properties Smart Browser Smart Manual Memory >t Property Value Conmon Custom mcuboot_config.h Direct XIP Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 1	<		
ns Console Properties X Smart Browser Smart Manual Memory t Property Value Common General Custom mcuboot_config.h Upgrade Mode Direct XIP Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 1	BSP Clocks Pins Interrupts Event	Links 🙆 Stacks Components	5
Property Value Common General Custom mcuboot_config.h Upgrade Mode Direct XIP Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 	ns 📃 Console 🔲 Properties 🗙	🌸 Smart Browser 🔑 Smart	Manual 📋 Memory
Property Value Common General Custom mcuboot_config.h Upgrade Mode Direct XIP Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 	ət		
✓ General Custom mcuboot_config.h Upgrade Mode Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application	Property	Value	2
Custom mcuboot_config.h Direct XIP Upgrade Mode Direct XIP Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 1	_ V General		
Upgrade Mode Direct XIP Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 1	Custom mcuboot_config.	.h	
Validate Primary Image Enabled Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 1	Upgrade Mode	Direc	ct XIP
Downgrade Prevention (Overwrite Only) Disabled Number of Images Per Application 1	Validate Primary Image	Enab	led
Number of Images Per Application 1	Downgrade Prevention (Overwrite Only) Disal	bled
	Number of Images Per A	pplication 1	

Figure 25. General Configuration for MCUboot Module



The properties configured are:

- **Custom mcuboot_config.h**: The default mcuboot_config.h file contains the MCUboot Module configuration that the user selected from the RA configurator. The user can create a custom version of this file to achieve additional bootloader functionalities that are available in MCUboot.
- **Upgrade Mode**: This property configures the application image upgrade method. The available options are Overwrite Only, Overwrite Only Fast, Swap, and Direct XIP. Only Direct XIP is supported for flash dual bank operation.
- **Validate Primary Image**: When enabled, the bootloader will perform a hash or signature verification, depending on the verification method chosen, in addition to the MCUboot magic number-based sanity check. When disabled, only a sanity check is performed based on the MCUboot magic number.
- **Number of Images Per Application:** This property allows the user to choose one image for Non-TrustZone-based applications and two images for TrustZone-based applications. Set this property to 1.
- **Downgrade Prevention (Overwrite Only):** This property applies to Overwrite upgrade mode only. When this property is **Enabled**, new firmware with a lower version number will not overwrite the existing application.
- Note: For Direct XIP mode, download grade prevention is supported from the MCUboot side. When using flash dual bank mode, the update image needs to have a version number higher than the current primary image.

4.2 Configure the Memory Configuration and Authentication Method

Configure the Signing Options and Flash Layout of the MCUboot module. For the EK-RA6M4, the default memory for the code flash dual bank mode is shown in Figure 26. This default memory map is used for the example bootloader design.



Figure 26. MCUboot Dual Bank Memory Map

Follow Figure 27 to update the properties for the Flash Layout to match with the MCUboot Dual Bank memory map.



Threads	New Thread HAL/Common Stacks			
meaus	Remove			
 ✓	HAL/Common			
	New Object >			
Objects				
Jummary	BSP Clocks Pins Interrupts Event Links Stacks Com	ponents		
Destate	nu 🗖 Carata 🔲 Provention X 🔍 Carat Proven	Count Manual R Manuary		
(Proble	ems 📮 Console 📋 Properties 🗙 👒 Smart Browser	Smart Manual 🕕 Memory		
ACUbo	ot			
Settings	Property	Value		
API Info	✓ Common			
	Signing and Economics Options			
	Signing and Encryption Options			
	> IrustZone	50004 D 055		
	Signature Type	ECDSA P-256		
	Boot Record	<i></i>		
	Custom	confirm		
	Python	python		
	Python Encryption Scheme	python Encryption Disabled		
	Python Encryption Scheme V Flash Layout	python Encryption Disabled	Signature Type	FCDSA P-25
	Python Encryption Scheme Flash Layout TrustZone	python Encryption Disabled	Signature Type	ECDSA P-25
	Python Encryption Scheme ✓ Flash Layout > TrustZone Bootloader Flash Area Size (Bytes)	python Encryption Disabled	Signature Type Boot Record	ECDSA P-25 None
	Python Encryption Scheme Flash Layout TrustZone Bootloader Flash Area Size (Bytes) Image 1 Header Size (Bytes) Image 1 Header Size (Bytes)	python Encryption Disabled 0x10000 0x200	Signature Type Boot Record Custom	ECDSA P-25 None ECDSA P-25 RSA 2048
	Python Encryption Scheme Flash Layout TrustZone Bootloader Flash Area Size (Bytes) Image 1 Flash Area Size (Bytes) Image 1 Flash Area Size (Bytes)	python Encryption Disabled 0x10000 0x200 0x70000	Signature Type Boot Record Custom Python	ECDSA P-25 None ECDSA P-25 RSA 2048 RSA 3072

Figure 27. Configure the Flash Layout and Signing Options

Explanation of the Above Configurations:

- **Bootloader Flash Area**: Size of the flash area allocated for the bootloader, with a boundary of 0x8000 since 0x8000 is the minimum erase size for RA6M4 code flash.
- **Image 1 Header Size**: Size of the code flash reserved for the application image header. It must meet minimum VTOR alignment requirements based on the number of interrupts implemented on the RA6M4. For the RA6M4, this property should be set to a minimum of 0x200 to support all interrupts.
- **Image 1 Flash Area Size**: Size of application image 1, including the header and trailer. For the RA6M4, this size needs to be on a boundary of 0x8000, which is the smallest flash erase size.
- Scratch Flash Area Size: This property is only needed for Swap mode. This property is not used for the flash dual bank bootloader design.
- **Signature Type:** Signing algorithm selection. The choices are:
 - **NONE:** Select this option for bootloaders that do not support signature verification.
 - ECDSA P-256: Select this option for this example bootloader design.
 - **RSA 2048 and RSA 3072:** Typically, this option is not used as the time used in the authentication is much longer than the ECDSA P-256.
 - Application images using MCUboot must be signed to work with MCUboot. At a minimum, this involves adding a hash and an MCUboot-specific constant value in the image trailer.
- **Custom:** Use the default --confirm for this bootloader design. Switching to a new image is always confirmed, and the new image will be booted after a subsequent system reset. Reverting the image with Direct XIP is not supported with the current FSP version.
- Encryption Scheme: Encryption is disabled in this example implementation.



4.3 Configure the MbedTLS Crypto Only Module and the Flash Driver

Follow the steps below to configure the MbedTLS module and the flash driver:

1. Right-click on Add Crypto Stack and select the MbedTLS (Crypto Only) module.

MCUboot		
MCUboot Port for RA (rm_mcuboot_port)		 MCUboot logging 1
Add Requires a crypto stack New	Add External Memory Implementation (Optional) stom Crypto (Protected Mode) yCrypt (S/W Only) ypto Only)	

Figure 28. Select MbedTLS Crypto Only Module

2. Click on **Add Requires Flash** stack and select Flash (r_flash_hp) stack.



Figure 29. Add the Flash Driver

3. Next, set the **Code Flash Programming** to **Enabled**. As **Data Flash Programming** is not used in the bootloader, select **Disabled** for the **Data Flash Programming** to reduce the bootloader memory footprint.

g_flash0	Flash (r_flash_hp)	
Settings	Property	Value
ADUlata	✓ Common	
API Into	Parameter Checking	Default (BSP)
	Code Flash Programming Enable	Enabled
	Data Flash Programming Enable	Disabled
	 Module g_flash0 Flash (r_flash_hp) 	
	Name	g_flash0
	Data Flash Background Operation	Disabled
	Callback	NULL
	Flash Ready Interrupt Priority	Disabled
	Flash Error Interrupt Priority	Disabled

Figure 30. Configure the Flash Driver



4. Configure the following properties of the MbedTLS (Crypto Only) module:

MbedTLS (Crypto Only)	nponents
is 🔱 Smart Manual 📮 Console 🔲 Properties 🗙 🎕	Smart Browse
s (Crypto Only)	
Property	Value
	Undofino
	Undefine
	Define
	Undefine
	Define
	Undefine
MBEDTLS_MEMORY_BACKTRACE	Undefine
MBEDTLS_MEMORY_B/CRYPTO_CLIENT	Undefine
MBEDTLS_FSA_CRYPTO_SPM	Undefine
MBEDTLS PSA ASSUME EXCLUSIVE BUFFERS	Undefine
MBEDTLS SELF TEST	Undefine
MBEDTLS THREADING ALT	Undefine
MBEDTLS THREADING PTHREAD	Undefine
MBEDTLS_USE_PSA_CRYPTO	Undefine
MBEDTLS_VERSION_FEATURES	Define
MBEDTLS_ERROR_C	Define
MBEDTLS_MEMORY_BUFFER_ALLOC_C	Define
MBEDTLS_PSA_CRYPTO_C	Define
MBEDTLS_PSA_CRYPTO_SE_C	Undefine
MBEDTLS_THREADING_C	Undefine
MBEDTLS_TIMING_C	Undefine
MBEDTLS_VERSION_C	Define
MBEDTLS_MEMORY_ALIGN_MULTIPLE	Undefine

Figure 31. Configure the MbedTLS (Crypto Only) Module

5. Disable RSA to save some memory usage.

MbedTLS (Crypto Only)	-
BSP Clocks Pins Interrupts Event Links 8 Stacks	Components
ns 🔑 Smart Manual 📃 Console 🔲 Properties 🛛	🧠 🖗 Smart Brov
S (Crypto Only)	
Property	Value
✓ Public Key Cryptography (PKC)	
> DHM	
> ECC	
✓ RSA	
MBEDTLS_PK_RSA_ALT_SUPPORT	Undefine
MBEDTLS_RSA_NO_CRT	Define
MBEDTLS_RSA_C	Undefine
MBEDTLS_RSA_GEN_KEY_MIN_BITS	Undefine
MBEDTLS RSA GEN KEY MIN BITS val	ue 1024

Figure 32. Disable RSA



6. Set up the Stack and Heap used by the bootloader based on the authentication mode. Set the following values in the **BSP** tab:



Figure 33. Configure the BSP Stack and Heap Usage

7. Add the Example Production Key module. DO NOT use the example key for production support. Users can reference the R11AN0567 section "Using Custom Signing Key and Encryption Key" for a method to create a customized user signing key.



Figure 34. Add the Example Production Key module

8. Enable the **Dual Bank Mode** under the **BSP** tab.

💦 Probler	ns 🔑 Smart Manual 📮 Console 🔲 Properties 🗙 🧇 Smart Brows	ser 📋 Memory
EK-RA6	М4	
Settings	Property	Value
	package_pins	144
	✓ RA6M4	
	series	6
	✓ RA6M4 Family	
	> Security	
	> OFS0 register settings	
	> OFS1_SEL register settings	
	> OFS1 register settings	
	> Block Protection Settings (BPS)	
	> Permanent Block Protection Settings (PBPS)	
	> Clocks	
	Enable inline BSP IRQ functions	Enabled
	Startup C-Cache Line Size	32 Bytes
	Dual Bank Mode	Enabled
	Main Oscillator Wait Time	8163 cycles

Figure 35. Enable Flash Dual Bank Mode

4.4 Add the Boot Code

Save configuration.xml and click Generate Project Content. Then, expand the Developer Assistance->HAL/Common->MCUboot->Quick Setup and drag Call Quick Setup to the top of the hal entry.c of the bootloader project.

Add the following function call to the top of the hal entry() function:

mcuboot_quick_setup();



4.5 Compile the Bootloader Project

In the RA configurator, click Generate Project Content, then compile the project.

```
Extracting support files...

12:09:00 **** Incremental Build of configuration Debug for project ra_mcuboot_ra6m4_dualbank ****

make -r -j16 all

arm-none-eabi-size --format=berkeley "ra_mcuboot_ra6m4_dualbank.elf"

text data bss dec hex filename

58732 0 6684 65416 ff88 ra_mcuboot_ra6m4_dualbank.elf

12:09:02 Build Finished. 0 errors, 0 warnings. (took 1s.699ms)
```

Figure 36. Compile the Bootloader ra_mcuboot_ra6m4_dualbank

There are warnings from third-party code.

4.6 Configure the Python Signing Environment

Signing the application image can be done using a post-build step in e^2 studio, using the image signing tool imgtool.py, which is included with MCUboot. This tool is integrated as a post-build tool in e^2 studio to sign the application image. If this is **NOT** the first time you have used the Python script signing tool on your computer, you can skip to section 5.

If this is the first time you are using the Python script signing tool on your system, you will need to install the dependencies required for the script to work. Navigate to the **ra_mcuboot_ra6m4_dualbank > ra > mcu-tools > MCUboot** folder in the **Project Explorer**, right click and select **Command Prompt**. This will open a command window with the path set to the \mcu-tools\MCUboot folder.



Figure 37. Open the Command Prompt

We recommend upgrading the pip prior to installing the dependencies. Enter the following command to update pip:

python -m pip install --upgrade pip

Next, in the command window, enter the following command line to install all the MCUboot dependencies:

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

This will verify and install any required dependencies.



Review the Signing Command

The signing command for the application image will be automatically generated when the bootloader is compiled. In the **Project Explorer**, open the

ra_mcuboot_ra6m4_dualbank\Debug\ra_mcuboot_ra6m4_dualbank.bld file. The signing
command is under the section <image>.

The application image uses a **Build Variable** to link with the .bld file. This process is explained in detail in the section 5.1. The application image has access to the .bld file, and the signing command will be automatically executed when the application image is compiled.

<images> <image path="\${BuildArtifactFileBaseName}.bin.signed">python \${workspace_loc:ra_mcuboot_ra6m4_dualbank}/ra/fsp/src/rm_mcuboot_port/rm_mcuboot_port_sign.py sign --header-size 0x200 --align 128 --max-align 128 --slot-size 0x70000 --max-sectors 14 --confirm --pad-header \${BuildArtifactFileBaseName}.bin.signed</image>

Figure 38. Signing Command in the .bld File

4.7 Prepare for Production Support

For production support, generate a .srec file of the bootloader to be loaded to the upper bank. This can be done by configuring a custom **Builder** within e² studio for the bootloader project.

This application project includes a bat file, process_bootloader.bat, which runs a script using srec_cat.exe to generate a .srec file, ra_mcuboot_ra6m4_dualbank_offset.srec, which offsets the bootloader offset to the RA6M4 flash linear mode upper bank address at 0x80000.

Note that for MCUs with different code flash sizes, the upper bank address needs to be updated accordingly. As explained in sections 1.1 and 1.2, this address is at half of the code flash size.

Since the option-setting memory is located outside of the bank range, this process also truncates the bootloader to the bank size, which is 0x80000.

srec_cat Debug\ra_mcuboot_ra6m4_dualbank.srec -crop 0 0x80000 -offset 0x80000 -o
ra_mcuboot_ra6m4_dualbank_offset.srec

Figure 39. Process the Bootloader to Load to the Upper Bank: process_bootloader.bat

Follow the steps below to configure the custom **Builder** in the bootloader project just created:

- Unzip ra6-advanced-mcuboot-flash-dual-bank.zip and copy
 \ra_mcuboot_ra6m4_dualbank\process_bootloader.bat as well as srec_cat.exe, located in
 the same folder, to the project root folder of the bootloader project just created.
- 2. Right-click on the bootloader project, open the **Properties** page, and navigate to the **Builders** page. Click **New** to start creating the customized Builder.

> Resource Builders > C/C++ Build > C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation	• • 8
---	-----------





3. Select **Program** in the next screen, then click **OK**:

Choose configuration type	×	
Choose an external tool type to create:		
Program		
OK Can	cel	

Figure 41. Select the Type of the Builder as Program

4. Next, provide the new Builder name Process Bootloader and click Browse Workspace to select process_bootloader.bat file as the Location of the Builder. Also, click Browse Workspace to set the Working Directory as shown below. Then, click Apply.

te a configuration that will run a program during builds	
e: Process Bootloader	
Main 🔗 Refresh 🚾 Environment 🗁 Build Options 🔤	
workspace_loc:/ra_mcuboot_ra6m4_dualbank/process_bootloader.	.bat}
Browse Workspace Browse File System Va	/ariables
orking Directory:	
workspace_loc:/ra_mcuboot_ra6m4_dualbank	
Browse Workspace Browse File System Va	/ariables
guments:	
V	/ariables
te: Enclose an argument containing spaces using double-guotes (")).
Show Command Line Revert	Apply
	APP.J

Figure 42. Configure the Custom Builder



5. Click **OK**, then **Apply and Close** at the next screen.

 Properties for ra_mcuboot_ra6 type filter text Resource Builders C/C++ Build C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags Validation 	im4_dualbank Builders Configure the builders for the project:	- □ ×
?	Ŀ	Apply and Close Cancel

Figure 43. Custom Builder

6. Recompile the bootloader project and notice that ra_mcuboot_ra6m4_dualbank_offset.srec is created under the bootloader project root directory.

✓ ⅔ ra_mcuboot_ra6m4_dualbank [Debug]
> 🐉 Binaries
> 🔊 Includes
> 🔑 ra
> 🔑 ra_gen
> 😂 src
> 🗁 Debug
> 🗁 ra_cfg
> 🗁 script
🔅 configuration.xml
💿 process_bootloader.bat
📄 R7FA6M4AF3CFB.pincfg
🗎 ra cfa.txt
📄 ra_mcuboot_ra6m4_dualbank_offset.srec
📄 ra mcuboot ra6m4_dualbank Debug_Flat.launch
srec_cat.exe
> ⑦ Developer Assistance

Figure 44. Rebuild the Bootloader with the Custom Builder

5. Configuring and Signing an Application Project

Developing an initial application to use a bootloader starts with developing and testing the application and the bootloader independently. Using the bootloader with an existing application or developing a new application to use the bootloader involves the following common steps:

- Adjust the memory map of the bootloader to allow the application and bootloader to fit the available MCU memory area.
- Configure the application to use the bootloader.
- Sign the application image.
- Developing an application to use a bootloader typically requires the application to have the capability to download a new application. This application project demonstrates how to download a new application using the USB and UART interfaces as examples. Users typically have custom methods to download new application images.



5.1 Configure the Application Project to Use the Bootloader

Users can follow *FSP User's Manual* section Tutorial: Your First RA MCU Project – Blinky to establish a new project. This application note uses the included example project as the initial application project and guides the user through the procedures to configure the example project to use the bootloader established in section 4.

Note that the steps described in this section can be applied to other existing application projects to configure the application project to use the bootloader. Be sure to consider the size the application project. When using the bootloader with a different application project, the **Image 1 Flash Area Size** property should be adjusted accordingly.

Import the desired application projects under folder \example_projects_without_bootloader to the workspace where the bootloader is created. For example, if the intended firmware update channel is USB, import app primary usb into the workspace.

Note: In this section's illustrations, the USB interface is used. The procedure for using the UART interface is similar to using USB.

Right-click on the application project folder app_primary_usb in the **Project Explorer** and select **Properties**. Select C/C++ Build > Build Variables, click Add and set the Variable name to **BootloaderDataFile**, and check the **Apply to all configurations** box. Change the **Type** to **File** and enter the path to the .bld file for the bootloader project ra_mcuboot_ra6m4_dualbank:

• Set \${workspace_loc:ra_mcuboot_ra6m4_dualbank}/Debug/ra_mcuboot_ra6m4_dualbank.bld for the value.

type filter text	Build Variables	↓ ↓ 8
Builders V C/C++ Build Build Variables Environment	Configuration: Debug [Active] V	anage Configurations
Logging Settings Tool Chain Editor > C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation	Define a New Build Variable X Variable name: BootloaderDataFile Apply to all configurations Type: File Value: 6m4_dualbank//Debug/ra_mcuboot_ra6m4_dualbank.bld Browse	Add Edit Delete
	E C U OK Cancel Restore De	ternal builder internal builder may faults Apply

Figure 45. Configure the Build Variable to Use the Bootloader

Click **OK**, then **Apply** and **Apply** and **Close** in the next screen.

5.2 Signing the Application Image

Note: If you rebuild the bootloader project after changing any of the signing and signature **Properties** of the MCUboot module, you will need to **Generate Project Content** again to bring in the updated .bld file.

When using Direct XIP mode, each application can define a version number. This is achieved by defining an Environment Variable: MCUBOOT_IMAGE_VERSION.

For applications that support signature verification, the signing key can be configured using Environment Variable MCUBOOT_IMAGE_SIGNING_KEY. If there is no signature verification, then it is not necessary to set Environment Variable MCUBOOT_IMAGE_SIGNING_KEY.

Open the **Properties** page of the project <code>app_primary_usb</code>, under **Environment**, click **Add** and configure MCUBOOT_IMAGE_VERSION.



> Kesource Builders C/C++ Build Build Variables Environment variables to set Settings Tool Chain Editor > C/C++ General Git Project Natures Project Natures Project References Renesas QE Run/Debug Settings > Validation Configuration: Debug [Active] Manage Configurations. Add Setence Settings O/C C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags Validation OK Cancel OK Cancel OK Cancel OK Cancel OK Cancel OK Cancel Project native environment with specified one	type filter text	Environment					↓ ↓ ↓ §
Logging Settings Tool Chain Editor > C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation Project addressed on the set More variable to set Value Origin AMS_LEEP,FILE \$1 USER: PREFS AMS_LICENSE,PATH \$1 USER: PREFS More variable Value 10.0 Variable Value 10.0 Variables OK Cancel Co	 > Resource Builders C/C++ Build Build Variables Environment 	Configuration: Debug (Active]			 ✓ Manage 	Configurations
	Logging Settings Tool Chain Editor > C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation	Environment variables to Variable AMS_KEP_FILE AMS_LICENSE_PATH New variable Name: MCUBOOT_IMA Value: 1.0.0 Cald to all configuration Add to all configuration O Replace native enviro	Value \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0	Origin USER: PREFS USER: PREFS Variabl	× les		Add Select Edit Delete Undefine

Figure 46. Configure the Application Version

Similarly, add the new variable for MCUBOOT_IMAGE_SIGNING_KEY.

type filter text	Environment 🗢 🔹 🔿 🔹
 ➢ Resource Builders ✓ C/C++ Build Build Variables Environment 	Configuration: Debug [Active] V Manage Configurations
Logging	Environment variables to set Add
Tool Chain Editor > C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation	Variable Value Origin Select AMS_KEEP_FILE \$() USER: PREFS Edit AMS_LICENSE_PATH \$() USER: PREFS Edit Image: McUBOOT_IMAGE_SIGNING_KEY Delete Undefine Value: Ibanki/ra/mcu-tools/MCUboot/root-ec-p256.pem Variables Undefine Image: Add to all configurations OK Cancel Cancel
	Restore Defaults Apply
?	Apply and Close Cancel

Figure 47. Configure the Private Signing Key

Note that the private key used for signing the application image is indicated in the signing command.

\${workspace_loc:ra_mcuboot_ra6m4_dualbank}/ra/mcu-tools/MCUboot/root-ec-p256.pem
is used for the example bootloaders. This key is used for testing purpose only. For real world use case and
production support, users MUST change this to the private key of their choice.



Figure 48 is the result of the above configuration. Click **Apply and Close**.

C/C++ Build	Configuration: Debug [Active]		 Manage Con 	nfiguratio
Build Variables Environment				
Logging	Environment variables to set			Add
Tool Chain Editor	Variable	Value	Origin	Color
C/C++ General	AMS_KEEP_FILE	\$0	USER: PREFS	Selec
Git	AMS_LICENSE_PATH	\$0	USER: PREFS	Edit
Project Natures	CWD	C:\Users\ \fsp_git\ra-solutions-rvc\application_projects\r11an0570\example_proj	BUILD SYSTEM	Dala
Project References	GCC_VERSION	13.2.1	BUILD SYSTEM	Dele
Renesas QE	MCUBOOT_IMAGE_SIGNING_KEY	{workspace_loc:ra_mcuboot_ra6m4_dualbank}/ra/mcu-tools/MCUboot/root-ec-p256.pem	USER: CONFIG	Undef
Run/Debug Settings	MCUBOOT_IMAGE_VERSION	1.0.0	USER: CONFIG	
Task Tags	PATH	C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\13.2 Rel1\bin\;\${renesas.build.utilsPa	BUILD SYSTEM	
Validation	PWD	C:\Users\ \fsp_git\ra-solutions-rvc\application_projects\r11an0570\example_proj	BUILD SYSTEM	
	TCINSTALL	C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\13.2 Rel1\	BUILD SYSTEM	
	TC_VERSION	13.2.1.arm-13-7	BUILD SYSTEM	
	<		>	
	Append variables to pative environ	ment		
	TC_VERSION < Append variables to native environ	13.2.1.arm-13-7	BUILD SYSTEM	

Figure 48. Configure the Application Image version number and Signing Key

To be able to recompile the project whenever the Environment Variables are updated, it is recommended add a Pre-build step to always delete the .elf file, as shown in Figure 49, so the application project is always recompiled.

type filter text	Settings	$\langle \neg \neg \rangle \neq 0$
> Resource Builders		
✓ C/C++ Build	Configuration: Debug [Active]	✓ Manage Configurations
Build Variables		
Environment	No. Tool Sattings 🔊 Toolshain 🌒 Puild Stops 💷 Puild Artifact 📄 Pinany Pars	arro 🙆 Error Darcoro
Settings	👷 Tool Settings 🛞 Toolcham 🎢 Build Steps 🛫 Build Arthact 📷 Binary Pars	
Tool Chain Editor	Pre-build steps	
> C/C++ General	Command(s):	
Git	rm -f \${ProjName}.elf	~
Project Natures	Description:	
Project References		
Renesas OF		~

Figure 49. Configure the Pre-build Command

At this point, a user can click **Generate Project Content** and compile the newly created application project and ensure that \Debug\app primary usb.bin.signed is generated.

5.3 Preparation for Production Support

For production support, a .srec file based on the signed application image needs to be generated. This .srec file offsets the application to the start address of the primary application, 0x10000 based on Figure 26.

```
srec_cat Debug\app_primary_usb.bin.signed -binary -offset 0x10000 -o
app_primary_usb_singed_offset.srec
```

Figure 50. Create app_primary_usb_signed_offset.srec

Follow steps similar to section 4.7 to add the custom **Builder** and compile the primary application:

- 1. Copy \example_projects_with_bootloader\app_primary_usb\srec_cat.exe and process_signed_binary_primary.bat to the root of project app_primary_usb.
- 2. Follow section 4.7 to create the new **Builder**. The finished configuration should look like Figure 51.



🗐 Edit Configuration — 🗆 🗙
Edit launch configuration properties
Create a configuration that will run a program during builds
Name: Process Signed Binary Primary
📄 Main 🤣 Refresh 🖾 Environment 🗁 Build Options
Location:
\${workspace_loc:/app_primary_usb/process_signed_binary_primary.bat}
Browse Workspace Browse File System Variables
Working Directory:
\${workspace_loc:/app_primary_usb}
Browse Workspace Browse File System Variables
Arguments:
~ I
Variables
Note: Enclose an argument containing spaces using double-guotes (").
Show Command Line Revert Apply
OK Cancel

Figure 51. Configure the Custom Builder for the Primary Application

3. Click Generate Project Content and compile the app_primary_usb project. Ensure that app_primary_usb_signed_offset.srec is generated under the root of the app_primary_usb project.

✓ 📂 app_primary_usb
> 🐝 Binaries
> 🔊 Includes
> 🥵 ra
> 😕 ra_gen
> 🥵 src
> 🧽 Debug
> 🧽 ra_cfg
> 🗁 script
app_primary_usb_signed_offset.srec
i app_primary_usb.elf.jlink
📄 app_primary_usb.elf.launch
ຜູ້ configuration.xml
long process_signed_binary_primary.bat
E R/FA6M4AF3CFB.pincfg
i≣ ra_cfg.txt
inij srec_cat.exe
> (?) Developer Assistance

Figure 52. Signed Primary Image Offset to the Primary Slot

6. Booting the Primary Application and Updating to a New Image

To update the application, the primary application needs to provide an image downloader. A new image will also need to be prepared to test the image downloader function.

6.1 Prepare a Secondary Image

In this project, a secondary image is created to test the downloading functionality of the primary application. The new application can be created by either modifying the existing application or creating a new application project. If a new application project is used, the user needs to establish the linkage to the bootloader by



following section 5. The newly created application project must also provide a method to download the new application to the upper bank.

In this application project, we will import the initial application project to the same workspace, rename the new project, and perform minor updates.

Right-click in the white space in the **Project Explorer** area and select **Import** and choose **Rename & Import Existing C/C++ Project into Workspace**.

Figure 53. Import the Initial Application

Once the Import window opens, name the project <code>app_secondary_usb</code>, check Select root directory, and click Browse:

🚱 Import	– 🗆 X
Rename & Import Project	
Select a directory to search for existing Eclipse projects.	
Project name: app_secondary_usb	
Use default location	
Location: C\Users\	Browse
Create Directory for Project	
Choose file system: default 🗸	
Import from:	
Select root directory:	Browse
O Select archive file:	Browse
Projects	
Options	
Keep build configuration output folders	
(?) < Back Next > Finis	th Cancel

Figure 54. Name the New Application

Browse into the Workspace folder and select app_primary_usb.



Import		\times
Rename & Import Project		-
Select a directory to search for existing Eclipse pr	rojects.	-
Project name: app_secondary_usb		
Use default location		
Location: C:\Users\	Browse	6
Create Directory for Proje	ct	
Choose file system: default 🔗		
Import from:		
Select root directory: _ts_without_bootloade	er\app_primary_usb 🗸 Browse	
O Select archive file:	Srowse	
Projects:		
app primary usb		
٢		>
Options		
Keep build configuration output folders		



Click **Finish**. The new application project will be created with the following attributes:

- When importing the primary application, the **Build Variable** and **Environment Variables** are automatically imported.
- The custom Builder "Process Signed Binary Primary" is also imported. For a clean project, a user must manually remove this Builder and the corresponding support files from the secondary project.
- Unlike in normal XIP Mode operation, the linker script symbol XIP_SECONDARY_SLOT_IMAGE must be undefined in Dual Bank mode. By default, XIP_SECONDARY_SLOT_IMAGE is undefined in the linker script symbol, so no action needs to be taken here.

Change the Enviroment variable for the Secondary Image version, shown in Figure 56.

type filter text	Environment				<	□ - ⊂> - 8	
⇒ kesource Builders ✓ C/C++ Build Build Variables Environment Logging	Configuration: Debug [Activ	ve]			 ✓ Manage Co 	nfigurations	
Settings Tool Chain Editor > C/C++ General Git Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation	Variable AMS_KEP_FILE AMS_LICENSE_PATH CWD GCC_VERSION MCUBOOT_IMAGE_SIG MCUBOOT_IMAGE_VER Edit variable Name: MCUBOOT_IMAGE_V Value: 1.1.0	Value \$() \$() C.Clusers\ 13.2.1 \$(workspace_locra_mcu 1.00 /ERSION	Origin USER: PREFS BUILD SYSTEM BUILD SYSTEM USER: CONFIG USER: CONFIG Variable	× rs cel	Restore Defaults	Apply	
0					Apply and Close	Cancel	

Figure 56. Change MCUBOOT_IMAGE_VERSION Variable



Update Existing Application to a New Application

To demonstrate the application update, update the application to blink the blue and green LED only.

Perform the following code updates in blinky thread entry.c:

```
Change below section of code in blinky_thread_entry:
    /* Update all board LEDs */
    for (uint32_t i = 0; i < leds.led_count; i++)
    {
        /* Get pin to toggle */
        uint32_t pin = leds.p_leds[i];
        /* Write to this pin */
        R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);
    }
To:
    /* update the blue led */
    R_BSP_PinWrite(leds.p_leds[0], pin_level);
    /* update the green led */
    R_BSP_PinWrite(leds.p_leds[1], pin_level);</pre>
```

Figure 57. Update the LED Control

Save the updated source file, click Generate Project Content, then compile the new project.

If you create a new application project and would like to debug the new project with the bootloader, follow the instructions in section 5. When debugging an update image with the bootloader, you can treat the update image as the primary application.

6.2 Set Up the Hardware

If using app primary usb as the initial application project:

- Connect J10 (USB Debug) using a USB micro to B cable from the EK-RA6M4 to the development PC to provide power and debug connection using the on-board debugger.
- USB FS device mode jumper setting: connect pin 2, 3 on J12, connect jumper J15.
- Connect J11 (USB FS) using a USB micro to B cable from the EK-RA6M4 to the development PC to provide USB Device connection.

If using app primary uart as the initial application project:

- Connect J10 using a USB micro to B cable from the EK-RA6M4 to the development PC to provide power and debug connection using the on-board debugger.
- Connect the three pins in Table 3 on the UART to USB converter to the EK-RA6M4.

Table 3. Connection through the UART Interface

UART to USB Converter	RA6M4
RX	P101 (TX)
TX	P100 (RX)
GND	GND

6.3 Erase the MCU

When MCUboot is used in flash dual bank mode, the code flash mode needs to start in linear mode. Erasing the MCU Option-Setting Memory settings will configure the code flash mode to linear mode. Erasing the entire MCU memory is recommended. The MCU can be erased through a variety of methods. A user can erase the MCU flash using the Renesas Device Partition Manager, Renesas Flash Programmer, or third-party tools like JFlash Lite.



Note: If the MCU is in code flash dual bank mode, restore to linear mode before proceeding to the rest of the application note sections. The rest of the operations assume the device starts in code flash linear mode. They will not work if the device is already in code flash dual bank mode.

6.3.1 Use the Renesas Flash Programmer

The Renesas Flash Programmer (RFP) can detect the flash mode when a new RFP project is created.

Note: Prior to connecting with the RFP, power cycle the development board.

Connect the EK-RA6M4 to the PC through the J10 USB Debug. Launch RFP and create a new RFP project. Click **File** -> **New Project**.

Ų	File	Target Device Help	
i usi		New Project	E:
¢		Open Project	1 K
a		Save Project	

Figure 58. Create a New RFP Project

Configure the Microcontroller selection and the Tool used for communication. Then, click Connect.

📓 Create New Proje	ct	_		×
Project Information				
Microcontroller:	RA			
Project Name:	new_rfp_project			
Project Folder:	C:\a_dual_bank_boot		Browse	
Communication				
Tool: J-Link	✓ Interface: 2 wire UART ∨			
Tool Details	Num: AutoSelect			
	Connec	t	Cance	el

Figure 59. Configure the New Project

Once the connection is successfully established, the user can open the **Block Settings** page to check the Code Flash configurations.

If RA6M4 flash is in code flash linear mode, Blocks Settings are presented as in Figure 60.

File	File Target Device Help						
Operati	on Operation Settings	Block Settings	Flash Options	Connect Se	ettings	Unique Code	User Keys
Regio	ı	Start	End	Size	Erase	P.V	
=	R7FA6M4AF3CFB						
٠	··· Code Flash 1	0x00000000	0x000FFFFF	1.0 M	\checkmark	\checkmark	
	··· Data Flash 1	0x08000000	0x08001FFF	8 K	\checkmark	\checkmark	
÷.	··· Config Area	0x0100A100	0x0100A2FF	512		\checkmark	
I							

Figure 60. Flash in Linear Mode

If the RA6M4 flash is in flash dual bank mode, **Block Settings** are presented as in Figure 61.





Figure 61. Flash in Dual Bank Mode

Whether the MCU is in flash dual bank mode or flash linear mode, the **Initialize Device** command can erase the entire flash, including the Config Area, and thus return the MCU to code flash linear mode.



Figure 62. Initialize Device Command

If the Initialize Device is successful, the message in Figure 63 will be presented in the status window.



Figure 63. Initialize Device Succeeded

6.3.2 Use the SEGGER J-Flash Lite

J-Flash Lite is a free, simple graphical user interface which allows downloading into flash memory of target systems. J-Flash Lite is part of the J-Link Software and Documentation package that is installed when the <u>J-Link software & documentation pack</u> is installed.

To use J-Flash Lite, connect the USB Debug port J10 to the PC and launch J-Flash Lite. Select the **Device** and debug **Interface** and communication speed.

SEGGER J-Flash Lite V7.98b	_		×
Device R7FA6M4AF	Interface SWD 4000 kHz	C	Ж

Figure 64. Launch the J-Flash Lite

Click **OK**. In the next screen, select **Erase Chip**.



Target
Device Interface Speed R7FA6M4AF SWD 4000 kHz
Data File (bin / hex / mot / srec /) Erase Chip
Program Device
Log
Connecting to J-Link Connecting to target Erasing Done.
Ready

Figure 65. Erase the MCU using J-Flash Lite

Note that when using Segger J-Flash Lite 7.98b or earlier, the Erase operation needs to be performed twice if the device is already in dual bank mode. This may be fixed in later J-Flash Lite versions.

6.3.3 Use Renesas Device Partition Manager

Power cycle the evaluation board EK-RA6M4 after a debug session to use the Renesas Device Partition Manager. Within e² studio, navigate to **Run** -> **Renesas Debug Tools** -> **Renesas Device Partition Manager**. Select J-Link as the connection method and select action **Initialize device back to factory default**.

Click Run. The MCU will be erased.



BRenesas Device Partition M	lanager				×
	_				^
Device Family: Renesas RA 🚿	~				
Action					
Read current device inform	ation	Change deb	ug state		
Set TrustZone secure / non	-secure bounda	nitialize dev	vice back to factor	y default	
Target MCU connection:		J-Link	~		
Connection Type:		SCI	\sim		
Emulator Connection:		Serial No	\sim		
Serial No/IP Address:					
Debugger supply voltage (V):		0	~		
Connection Speed (bps for SC	l, Hz for SWD):	9600	~		
Debug state to change to:		Secure Software De	velopment	~	
Memory partition sizes					
Use Renesas Partition Data	file				
				Brows	se
Code Flash Cosura (KP)	510				
Code Flash NSC (KB)	0				
Data Elach Socura (KB)	0				
SPAM Secure (KB)	256				
SRAM Secure (KB)	250				
SRAM NSC (KB)	0				
Command line tool:					
C:\Users\	pse\com.renesa	s.platform_27591836	59\DebugComp\R	A\Devic Bro	owse
					^
Connecting	. (
Establishing connection	: 5	UCCESSFUL!			
Checking the device's Tr	rustZone type	: SUCCESSFUL!			
CONNECTED.					
Initializing device and rolling b SUCCESSFUL!	oack DLM state	to SSD			
Disconnecting					
DISCONNECTED.					
SUMMARY OF RESUL	_T	7			
Connection : SUCCESSF	UL!				
Device initialization : SUC	CESSFUL!				
		-			~
?	Import	Export	Run	Clo	se
0					

Figure 66. Erase the MCU using Renesas Device Partition Manager



6.4 Start the Debug Session

Follow the steps below to start the debug session:

 Disable flash content caching from the Debugger setting. Right-click on project app_primary_usb -> Debug As -> Debug Configurations, navigate to Debugger -> Debug Tool Settings, and uncheck Allow caching of flash contents. Otherwise, when debugging bootloader applications, the memory window may show wrong information when debugging bootloader applications.

Debug hardware: J-Link ARM 🛛 🗸 Target Device	:: R7FA6M4AF	
GDB Settings Connection Settings Debug Tool Set	ttings	
~ I0		^
Use Default IO Filename	Yes	\sim
IO Filename	\${support_area_loc}	
✓ General Debug		
Reset After Reload	Yes	~
✓ Memory		
Endian	Little Endian	\sim
∨ Break		
Use Flash Breakpoints	Yes	\sim
Allow Simulation	No	\checkmark
∨ Flash		
Flash Bus Type		\sim
Flash Memory Type		\sim
WorkRam Start		
WorkRam End		
Erase on-chip program flash before download	t Yes	\sim
Erase on-chip data flash before download	Yes	\sim
Use CFI-Flash	Yes	\sim
CFI Start	0x0	
CFI End	0x0	
✓ Semihosting		
Semihosting breakpoint address		
✓ RTOS		
RTOS Integration in Debug View	No	\sim
RTOS Debugging - Large Number of Threads.	No	\sim
∨ System		
Allow caching of flash contents	No	~
✓ Time Measurement		
Run Break Time Measurement	Yes	~
Count Every Core Cycle	Yes	\sim

Figure 67. Disable Flash Content Caching



Configure the load image and symbols properties.
 Open the Debug Configurations: app_primary_usb -> Debug As -> Debug Configurations.
 Make sure app_primary_usb Debug_Flat is selected and select the Startup tab.
 Click Add..., then Workspace, navigate to the ra_mcuboot_ra6m4_dualbank project, and select the ra_mcuboot_ra6m4_dualbank.elf file from the debug folder. Click OK.

	-	
Add download module	×	
pecify download module name: ra mcuboot ra6m4 dualbank\Debu	a\ra mcuboot ra6m4 dualbank.elf	
Variables Search Project	Workspace File System	
[OK Cancel	
F	Add download module pecify download module name: a_mcuboot_ra6m4_dualbank\Debu Variables Search Project	Add download module ame: a_mcuboot_ra6m4_dualbank\Debug\ra_mcuboot_ra6m4_dualbank.elf Variables Search Project Workspace File System OK Cancel

Figure 68. Add the Bootloader Project to Debug Configuration

3. Change the **Load type** of the **Program Binaries** for the **app_primary_usb** project to **Symbols only** by clicking on the cell for **Load type** and selecting **Symbols only** from the drop-down menu.

load image and symbols			
Filename	Load type	Offset (hex)	On connect
Program Binary [app_primary_usb.elf]	Symbols only		Yes
✓ ra_mcuboot_ra6m4_dualbank.elf [C:\a_d	Image and Symbols	0	Yes

Figure 69. Select to load Symbols only for the Application Project

4. Follow similar steps to add the signed primary image and the upper bank bootloader. Choose **Image only** as the **Load type** for the upper bank bootloader and choose **Raw Binary** as the **Load type** for the primary application image.

ilename	Load type	Offset (hex)	On connect	Add
Program Binary [app_primary_usb.elf]	Symbols only		Yes	
app_primary_usb.bin.signed [C:\Users\a50	Raw Binary	10000	Yes	Edit
✓ ra_mcuboot_ra6m4_dualbank.elf [C:\Users	Image only	80000	Yes	Remove
✓ ra_mcuboot_ra6m4_dualbank.elf [C:\Users	Image and Symbols	0	Yes	
				Move up

Figure 70. Add the Signed Primary Image and Upper Bank Bootloader

5. Click **Debug**. The debugger should hit the reset handler in the bootloader.



Figure 71. Start the Application Execution



6. Choose **Remember my decision** and click **Switch** if prompted to switch the perspective.

📴 Cor	nfirm Perspective Switch	×
?	This kind of launch is configured to open the Debug perspective when it suspends. This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints. Switch to this perspective?	
Ren	nember my decision Switch No	

Figure 72. Switch the Perspective

Click Resume location to run the project.
 The program should now be paused in the main at the hal_entry() call in the bootloader.



Figure 73. Start the Application Execution

8. Click 🏴 to run again.

The red, blue, and green LEDs on the EK-RA6M4 should now be blinking while the blinky application is running.

6.5 **Program the New Application Using the Primary Application Downloader**

Follow the steps below to program the new application created in section 6.1:

1. Open Tera Term and choose the USB Serial Port (the COM number may be different for your setup). Then click **OK**.

Tera Term: New c	nnection			×
○ тср/ір	Host Service	: myhost.exa ✓ History : Telnet ● SSH ● Other	TCP port#: 22 SSH version: SSH2 IP version: AUTO	
Serial	Port OK	COM7: USB	3 Serial Port (COM7) Help	~

Figure 74. Open the COM Port

Note: When using the UART interface, select the Serial Terminal and set the **Speed** to 115200. Skip this step if using the USB interface.



Tera Term: Serial port setu	up and connection	×
Port:	COM7 ~	New setting
Speed:	115200 ~	
Data:	8 bit \sim	Cancel
Parity:	none ~	
Stop bits:	1 bit \sim	Help
Flow control:	none ~	

Figure 75. Configure the Baud Rate if using the UART Interface

The menu in Figure 76 will be displayed on the Tera Term.



Figure 76. Tera Term Menu

2. Select option **1** to print the image slot information.

>1	
* Primary Image Slot *****	****
Image version: Primary image start a	1.0 (Rev: 0, Build: 0) ddress: 0x00010000
Header size: Protected TLV size:	0x0200 (512 bytes) 0x0000 (0 bytes)
lmage size:	0x0000B2B4 (45748 bytes)
* Secondary Image Slo *****	*****
Image version: Secondary image start	255.255 (Rev: 65535, Build: -1) address: 0x00210000
Header size: Protected TLV size:	ØxFFFF (65535 bytes) ØxFFFF (65535 bytes) ØxFFFF (65535 bytes)

Figure 77. Print the Image Slot Information

3. Select option 2 to download the secondary image using the primary image downloader.







4. Open the **Transfer** interface of the Tera Term.

M	COM5	- Tera Te	erm VT						
File	Edit	Setup	Control	Window	Help				
	New co	onnecti	on	Alt+N	009F6C <4	0812 byt	tes)		
	Duplica	ate sess	ion	Alt+D					
	Cygwir	n conne	ction	Alt+G	255 (Reu:	65535 -	Bui	(1- :h[
	Log				FFFFFF FF (65535	butes)	20112		
	Pause I	Logging)		FF (65535	bytes) 1 hutes	`		
	Comm	ent to l	.og			1			
	View L	og							
	Show L	.og dial	og						
	Stop Lo	ogging	(Q)						
	Send fi	le			w image (XModem)			
Г	Transfe	r		>	Kermit	>			
	SSH SC	:Р			XMODE	vi →		Receive	
	Chang	e direct	ory		YMODE	M >		Send	
	Replay	Log			ZMODE	× N	T		

Figure 79. Start Transfer from Tera Term

5. Choose \app_secondary_usb\Debug\app_secondary_usb.bin.signed, then click **Open**.

Tera Term: XMODEM Send		
Look in: 🔄 Debug	✓ ④ Ø ▷ □•	
Name	Date modified	Туре
na la	9/23/2024 10:29 AM	File folder
ra_gen	9/23/2024 10:29 AM	File folder
src	9/23/2024 10:29 AM	File folder
app_secondary_usb.bin.signed	9/23/2024 10:29 AM	SIGNED File
app_secondary_usb.elf	9/23/2024 10:29 AM	ELF File
app_secondary_usb.elf.in	9/23/2024 10:29 AM	IN File
app_secondary_usb.map	9/23/2024 10:29 AM	Linker Address Map

Figure 80. Choose the Signed Secondary Image

The secondary image is then downloaded and programmed to the upper bank.

Tera Term: XMODE	DEM Send	×
Filename: Protocol: Packet#: Bytes trans Elapsed tin	app_secondary XMODEM (che sferred: me:	y_usb.b ecksum) 837 107136 0:00
	Cancel	81.7%

Figure 81. Download the New Image via XModem

6.6 Boot the New Application

The system will automatically reboot after the new image is downloaded.



Figure 82. The New Image is Booted



Select option **1** to read the swapped memory layout.





Note that even though the secondary image is booted, it cannot be debugged as the symbol downloaded to the debugger is for the primary image.

Also, if you want to perform further updates, the new image must have a version higher than the current image in the primary slot.

7. Production Support Considerations

This section describes one possible production flow. Users may adapt this procedure to their own needs wherever possible.

7.1 Protect the Bootloader using Flash Block Protection

The secure bootloader protects the system's Root of Trust. The application should protect it from alteration. Based on Figure 36, the bootloader is located in the first 64-KB region. Based on Figure 3, the blocks that need to be protected are blocks 0 to 7 for the lower bank and 70 to 77 for the upper bank.

Users can set up these blocks to be temporarily protected in the ra_mcuboot_ra6m4_dualbank project under the BSP tab. If these blocks are protected temporarily, the block protection setting can be reset by performing the MCU erase operations described in section 6.3.

Summary	BSP Clocks Pins Interrup	ts Event Links Stacks Compone	ents			
Rroblen	ns 🚇 Smart Manual 📃	Console 🔲 Properties 🗙 ௸	Smart Browser	🚺 Memory 🐐 Debug		
EK-RA6	/ 14					
Settings	Property		Value			
	package_pins		144			
	✓ RA6M4					
	series		6			
	✓ RA6M4 Family					
	> Security					
	> OFS0 register setti	ngs				
	> OFS1_SEL register	settings				
	> OFS1 register setti	ngs				
	 Block Protection S 	ettings (BPS)				
	V BPSO					
	Flash Block	0	\checkmark			
	Flash Block	1				
	Flash Block	2				
	Flash Block	3				
	Flash Block	4				
	Flash Block	5	\checkmark			
	Flash Block	5				
	Flash Block	7	\checkmark			
	Flash Block	3				
	Flash Block	9				

Figure 84. Temporary Protection of the Lower Bank Bootloader Area



Summary	BSP Clock	Pins Interrupts Event Links Stacks Compo	onents			
Roblen	ns 🚇 Sma	rt Manual 📮 Console 🔲 Properties 🗙 🍕	🔈 Smart Browser 🚺] Memory 🔯 Debug		
EK-RA6	14					
Settings	Property		Value			
	pa	ckage_pins	144			
	✓ RA6M	4				
	ser	ies	6	6		
	✓ RA6M	4 Family				
	> Sec	curity				
	> OF	S0 register settings				
	> OF	S1_SEL register settings				
	> OF	S1 register settings				
	✓ Blo	ock Protection Settings (BPS)				
	>	BPS0				
	>	BPS1				
	~	BPS2				
		Flash Block 70 (Dual Mode Only)				
		Flash Block 71 (Dual Mode Only)				
		Flash Block 72 (Dual Mode Only)				
		Flash Block 73 (Dual Mode Only)				
		Flash Block 74 (Dual Mode Only)				
		Flash Block 75 (Dual Mode Only)				
		Flash Block 76 (Dual Mode Only)				
		Flash Block // (Dual Mode Only)				
		Flash Block 78 (Dual Mode Only)				
		Flash Block 79 (Dual Mode Only)				

Figure 85. Temporary Protection of the Upper Bank Bootloader Area

Users can set up these blocks to be permanently protected in the <code>ra_mcuboot_ra6m4_dualbank</code> project under the BSP tab.

Note: If these blocks are protected permanently, these areas cannot be erased and reprogrammed through the lifetime of the MCU. Users need to be very cautious when setting up permanent protection. The MCU erase operations are described in section 6.3 will not be able to erase these blocks.

Summary	Summary BSP Clocks Pins Interrupts Event Links Stacks Components					
Problem	ns 🔑 Sma	rt Manual 📮 Console 🔲 Properties 🗙 ௸ Sm	art Browsei	r 🧻 Memory 🐐 Debug		
EK-RA6N	14					
Settings	Property	4	Value			
	seri	es	6			
	> Sec	4 Family surity				
	> OFS > OFS	50 register settings 51_SEL register settings				
	> OFS	51 register settings				
	✓ Peri	manent Block Protection Settings (PBPS)				
	~	PBPS0 Flash Block 0				
		Flash Block 1				
		Flash Block 2 Flash Block 3				
		Flash Block 4				
		Flash Block 6				
		Flash Block 7 Flash Block 8				
		Flash Block 9				

Figure 86. Permanent Protection of the Lower Bank Bootloader Area



Summary	SP Clocks Pins Interrupts Eve	ent Links Stacks Components				
Problem	s 🐺 Smart Manual 📃 Conso	le 🔲 Properties 🗙 虆 Sma	rt Browse	r 📋 Memory 🗄	🌾 Debug	
EK-RA6N	14					
Settings	Property		Value			
Jettings	package_pins		144	144		
	✓ RA6M4					
	series			6		
	✓ RA6M4 Family					
	> Security					
	> OFS0 register settings					
	> OFS1_SEL register settings					
	> OFS1 register settings					
	> Block Protection Settings (BPS)					
	 Permanent Block Protection Settings (PBPS) 					
	> PBPS0					
	> PBPS1					
	✓ PBPS2					
	Flash Block 70 (Du	al Mode Only)				
	Flash Block 71 (Du	al Mode Only)				
	Flash Block /2 (Du	ai Mode Only)				
	Flash Block 73 (Du	al Mode Only)				
	Flash Block 74 (Du	al Mode Only)				
	Flash Block 75 (Du	al Mode Only)				
	Flash Block 77 (Du	al Mode Only)				
	Flash Block 78 (Du	al Mode Only)				
	Flash Block 79 (Du	al Mode Only)				
	Thair block 75 (Du	ar mode only)	-			

Figure 87. Permanent Protection of the Upper Bank Bootloader Area

The included example bootloader does not include the block settings to enable block protection. Users can enable them prior to field deployment.

7.2 Provision the Bootloaders and the Initial Application to MCU

Users can combine the .srec files generated from the above sections into one .srec file and program it to the MCU during production.

The three images to be combined are:

- Bootloader for the Lower Bank: ra_mcuboot_ra6m4_dualbank.srec
- Bootloader for the Upper Bank: ra mcuboot ra6m4 dualbank offset.srec
- Application for the Lower Bank: app_primary_usb_signed_offset.srec

The following command assumes the user executes from the location of the srec_cat.exe and has all three input files exist under the same folder as the srec_cat.exe. Use the following command to generate one combined .srec from the above three .srec files:

```
srec_cat ra_mcuboot_ra6m4_dualbank.srec ra_mcuboot_ra6m4_dualbank_offset.srec
app_primary_usb_signed_offset.srec -o combined.srec
```

To download combined.srec, users can use RFP or J-Flash Lite, as shown in Figure 88 and Figure 91.

Note: Prior to download combined.srec, users need to erase the MCU first and follow the instructions in section 6.3.



• Download (*.srec) file using J-Flash Lite.

SEGGER J-Flash Lite V7.98b	- 🗆	×
File Help		
Target Interface Interface SWD	Speed 4000 kHz	
Data File (bin / hex / mot / srec /) 0_mcuboot_dualbank\combine_srec	Eras	e Chip
Program Device		
Log	•	
Selected file: C:\Users\trung.tran-quoc\Documn Data file contains 5 memory ranges: #0:0x00000000 - 0x0000E483 (55500 Bytes) #1:0x00100000 - 0x0100A13 (517252 Bytes) #2:0x0100A140 - 0x0100A13 (20 Bytes) #3:0x0100A140 - 0x0100A137 (4 Bytes) #4:0x0100A200 - 0x0100A2CF (208 Bytes)	ents\RVC_TASK_24_2H	\migrat
<		>
Ready		

Figure 88. Load combined.srec file using J-Flash Lite

• Download (*.srec) file using Renesas Flash Programmer (RFP).

Launch RFP and create a new RFP project. Click **File > New Project**.

File Target Device Help		
New Project	Connect Settings	Unique Code
Open Project Save Project	is connect Settings	Unique Code

Figure 89. Create a New RFP Project

Configure the **Microcontroller** selection and the **Tool** used for communication. Then, click **Connect**.

Create New Project	t	-		×
Project Information Microcontroller: Project Name: Project Folder: Communication	RA [ra_mcuboot_dualbank	~	Browse	
Tool: J-Link Tool Details	V Interface: SWD V	t	Canc	zel

Figure 90. Configure the New Project



Select a program file and execute the command.

File Target Device Help	
Operation Operation Settings Block Settings Con	nnect Settings Unique Code
Project Information	
Current Project: ra_mcuboot_dualbank.r	rpj
Microcontroller: RA	
Program Files	
	\r11an0570_mcuboot_dualbank\combine_srectcombined.sred
CRC-32: B1802DDB	Add/Remove Files
Command	
Erase >> Program >> Verify	
Stort	
Start	

Figure 91. Selecting combined.srec file and execute the command

If the download (*.srec) file is successful, the message in Figure 92 will be presented in the status window.



Figure 92. Download combined.srec file succeeded

Once the device is deployed to the field, the application update can be achieved using the image downloader implemented in the application project.



8. Compile and Exercise the Included Example Bootloader and Application Projects

8.1 Using USB as the Download Interface

For the USB interface, three projects are needed:

- ra mcuboot ra6m4 dualbank
- app_primary_usb
- app_secondary_usb

Users can follow the steps below to run the example projects in the folder \ra6-advanced-mcuboot-flash-dual-bank\example_projects_with_bootloader:

- 1. Follow the instructions in section 6.2 to set up the hardware.
- 2. Import the above-mentioned three projects to a Workspace.
- 3. Open the configuration.xml file from project ra_mcuboot_ra6m4_dualbank.
- 4. Click Generate Project Content.
- 5. Compile the project ra_mcuboot_ra6m4_dualbank.
- 6. Open the configuration.xml file from project app_primary_usb.
- 7. Click Generate Project Content.
- 8. Compile the app_primary_usb.
- 9. Open the configuration.xml file from project app_secondary_usb.
- 10. Click Generate Project Content.
- 11. Compile the app_secondary_usb project.
- 12. Erase the entire chip following the instructions in section 6.3.
- 13. Debug the application from project $app_primary_usb$ in the e² studio environment.
- 14. Resume the program execution twice. All three LEDs should be blinking.
- 15. Stop the debug session and power cycle the EK-RA6M4.
- 16. Open Tera Term with the enumerated COM port (USB Serial Device).
- 17. Use Tera Term to send the \app_secondary_usb\Debug\app_secondary_usb.bin.signed to the MCU following the instructions in section 6.6. This will take about 30 seconds.
- 18. The system will be reset automatically after download.
- 19. Blue and green LEDs should be blinking.
- 20. Enter menu item 1 to confirm that the image with version 1.1.0 is located in the primary slot (lower bank) and the image with version 1.0.0 is located in the secondary slot (upper bank).

8.2 Using the UART as the Download Interface

For the UART interface, three projects are needed:

- ra_mcuboot_ra6m4_dualbank
- app primary uart
- app_secondary_uart

Users can follow the steps below to run the example projects in the folder \ra6-advanced-mcuboot-flash-dual-bank\example_projects_with_bootloader:

- 1. Follow the instructions in section 6.2 to set the hardware.
- 2. Import the above-mentioned three projects to a workspace.
- 3. Open the configuration.xml file from project ra_mcuboot_ra6m4_dualbank.
- 4. Click Generate Project Content.
- 5. Compile the project ra_mcuboot_ra6m4_dualbank.
- 6. Open the configuration.xml file from project app_primary_uart.
- 7. Click Generate Project Content.
- 8. Compile app_primary_uart.
- 9. Open the configuration.xml file from project app_secondary_uart.



10. Click Generate Project Content.

- 11. Compile the app_secondary_uart project.
- 12. Erase the entire chip following the instructions in section 6.3.
- 13. Debug the application from project app_primary_uart in the e² studio environment.
- 14. Resume the program execution twice. All three LEDs should be blinking.
- 15. Stop the debug session and power cycle the EK-RA6M4.
- 16. Open the Tera Term with the enumerated COM port and set up the baud rate as 115200.
- 17. Use Tera Term to send the \app_secondary_uart\Debug\app_secondary_uart.bin.signed to the MCU by following section 6.6. This will take about 50 seconds.
- 18. The system will reset automatically after download.
- 19. Blue and green LEDs should be blinking.
- 20. Enter menu item 1 to confirm that the image with version 1.1.0 is located in the primary slot (lower bank) and the image with version 1.0.0 is located in the secondary slot (upper bank).

9. References

- 1. Renesas RA Family MCU Securing Data at Rest using Security MPU Application Project (R11AN0416)
- 2. Renesas RA Family RA6 Series MCU Basic Secure Bootloader Design using MCUboot with Code Flash Linear Mode Application Project (R11AN0497)
- 3. Renesas RA Family RA2 Series MCU Secure Bootloader Design using MCUboot Application Project (R11AN0516)
- 4. Renesas RA Family RA6 Series MCU Advanced Secure Bootloader Design using MCUboot with Encrypted Image and QSPI (R11AN0567)



10. Website and Support

Visit the following URLs to learn about the RA family of microcontrollers, download tools and documentation, and get support.

EK-RA6M4 Resources RA Product Information Flexible Software Package (FSP) RA Product Support Forum Renesas Support renesas.com/ra/ek-ra6m4 renesas.com/ra renesas.com/ra/fsp renesas.com/ra/forum renesas.com/support



Revision History

		Description			
Rev.	Date	Page	Summary		
1.00	Mar.21.22	-	First release document		
1.10	Nov.11.22	-	Updated Operation Flow based on e ² studio 2022-10 or later. Used FSP v4.0.0. Document title changed from "RA6 Secure Bootloader Update using MCUboot and Flash Dual Bank" to "RA6 Secure Firmware Update using MCUboot and Flash Dual Bank"		
1.11	Nov.23.22	-	Corrected typo, added Figure 56 and included RA6E1.		
1.20	Feb.28.24	-	Minor documentation updates and migration to FSP v5.0.0		
1.30	May.17.24	-	Added bat files.		
1.40	Oct.01.24	-	Update to FSP v5.5.0		



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 is supplied until the 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 systemevaluation test for the given product.

Notice

- Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
- 2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
- 3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
- 4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
- 5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
- 6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

- 7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
- 8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
- 10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
- 12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
- This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
 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/.