Renesas RA Family

RA6 MCU Advanced Secure Bootloader Design using MCUboot and Code Flash Dualbank Mode

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 independent operating system and hardware 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 from FSP v3.6.0. This application note demonstrates 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 RA5M5 and port the application to RA6E1 and RA6M5.In addition, steps for how to master an application to use with the bootloader and how to update 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, reference application project R11AN0497 (Search | Renesas Electronics Corporation).

Required Resources

Development tools and software

- The e^2 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 e^2 studio are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

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

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 e^2 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. Code Flash Dual Bank Feature 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. Using the Code Flash Dual Bank Feature with MCUboot Overview covers the general flow of architecting a system using the FSP MCUboot module. For example, the memory configuration for a code flash dual bank-based bootloader using MCUboot is introduced in this section.

Section 3. Guidelines for Using the Example Projects Included 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. Creating the Bootloader Project using Code Flash Dual Bank Mode 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.Configuring and Signing an Application Project provides the steps to configure and sign an application to use the bootloader created in section 4. Creating the Bootloader Project using Code Flash Dual Bank Mode. The included example projects are used in this section.

Section 6. Booting the Primary Application and Updating to a New Image provides instructions on how to debug and boot the primary application project and update 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. Production Support Considerations covers the production support of provisioning the new MCU with the bootloaders and the initial application.

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

For the 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. RA6M4 and RA6E1 Code Flash Memory in Linear Mode.

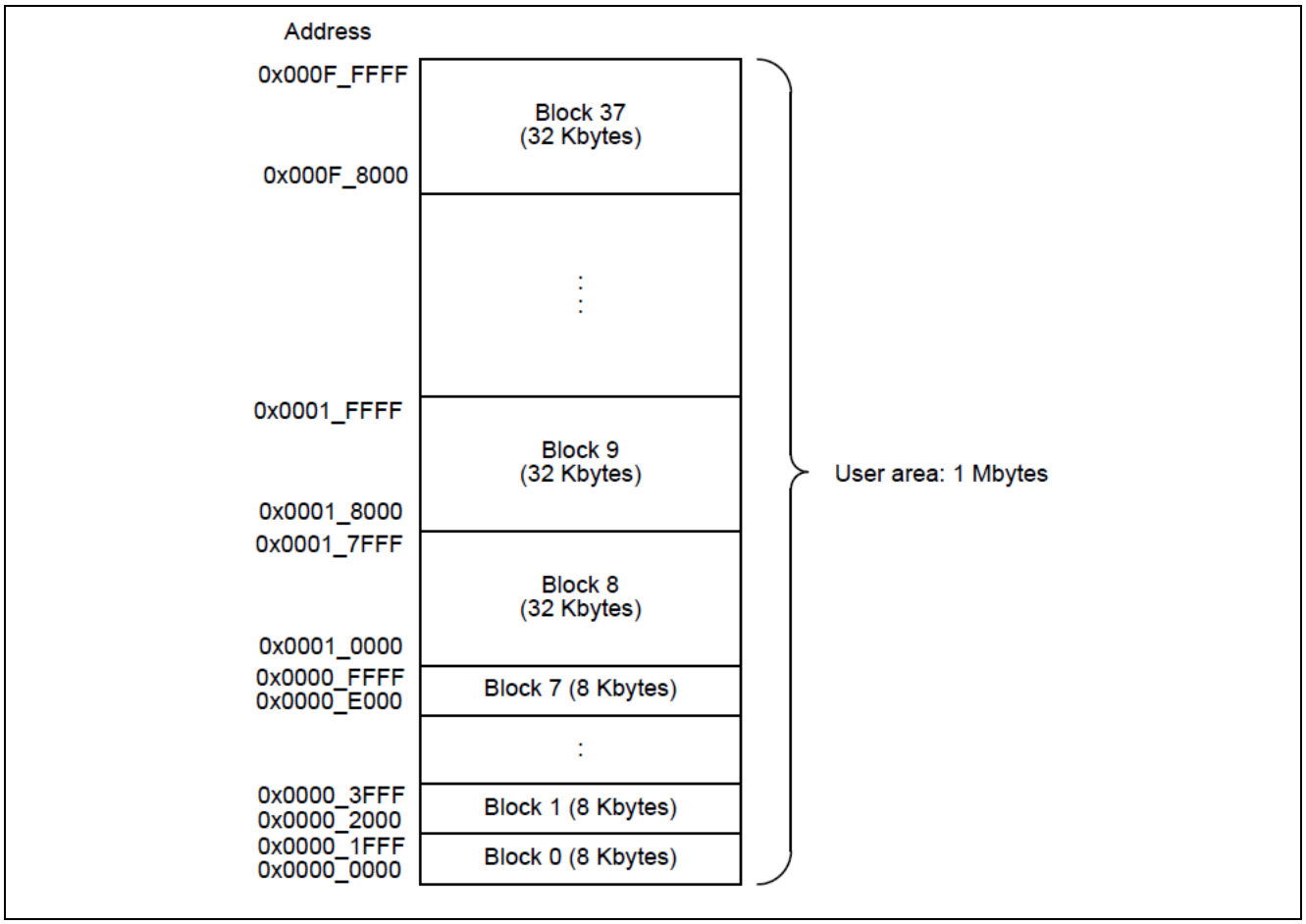

Figure 1. RA6M4 and RA6E1 Code Flash Memory in Linear Mode

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. RA6M4 and RA6E1 Code Flash Memory in Dual Bank Mode. The default configuration is highlighted in the red box.

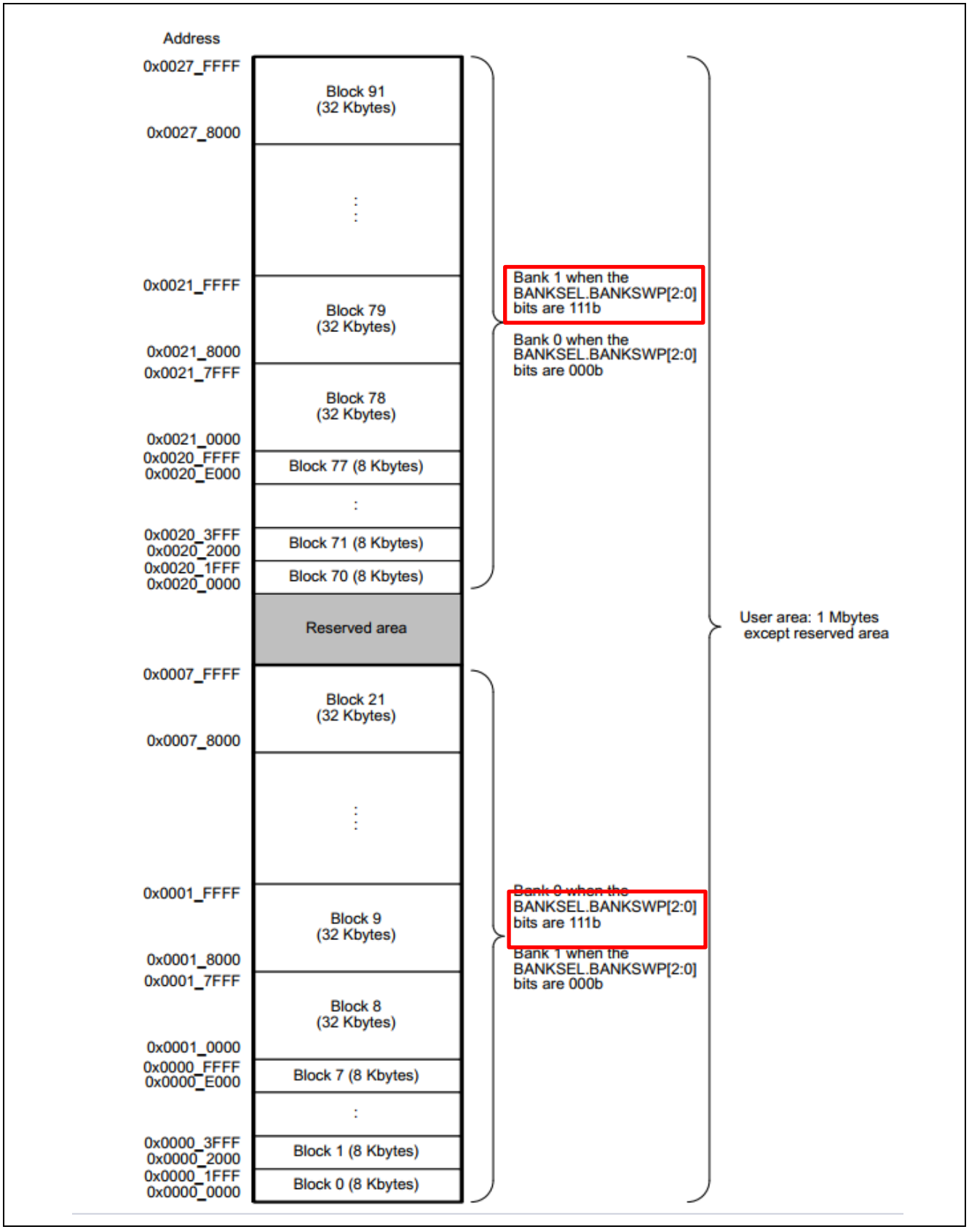

Figure 2. RA6M4 and RA6E1 Code Flash Memory in Dual Bank Mode

Table 1. RA6M4 and RA6E1 Code Flash 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

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

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

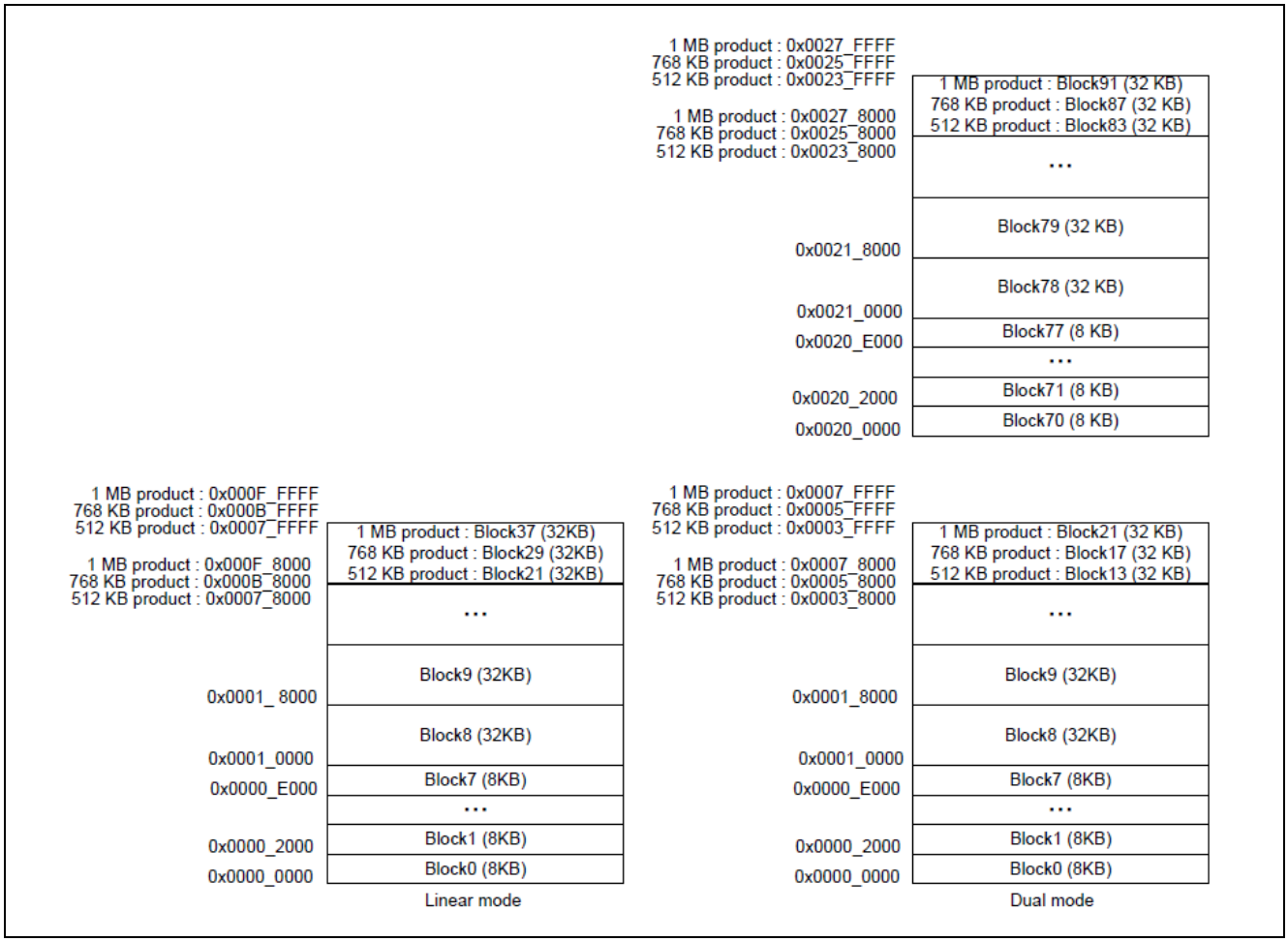

Figure 3. RA6M4 and RA6E1 Code Flash Block Structure

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. RA6M5 Code Flash Memory in Linear Mode.

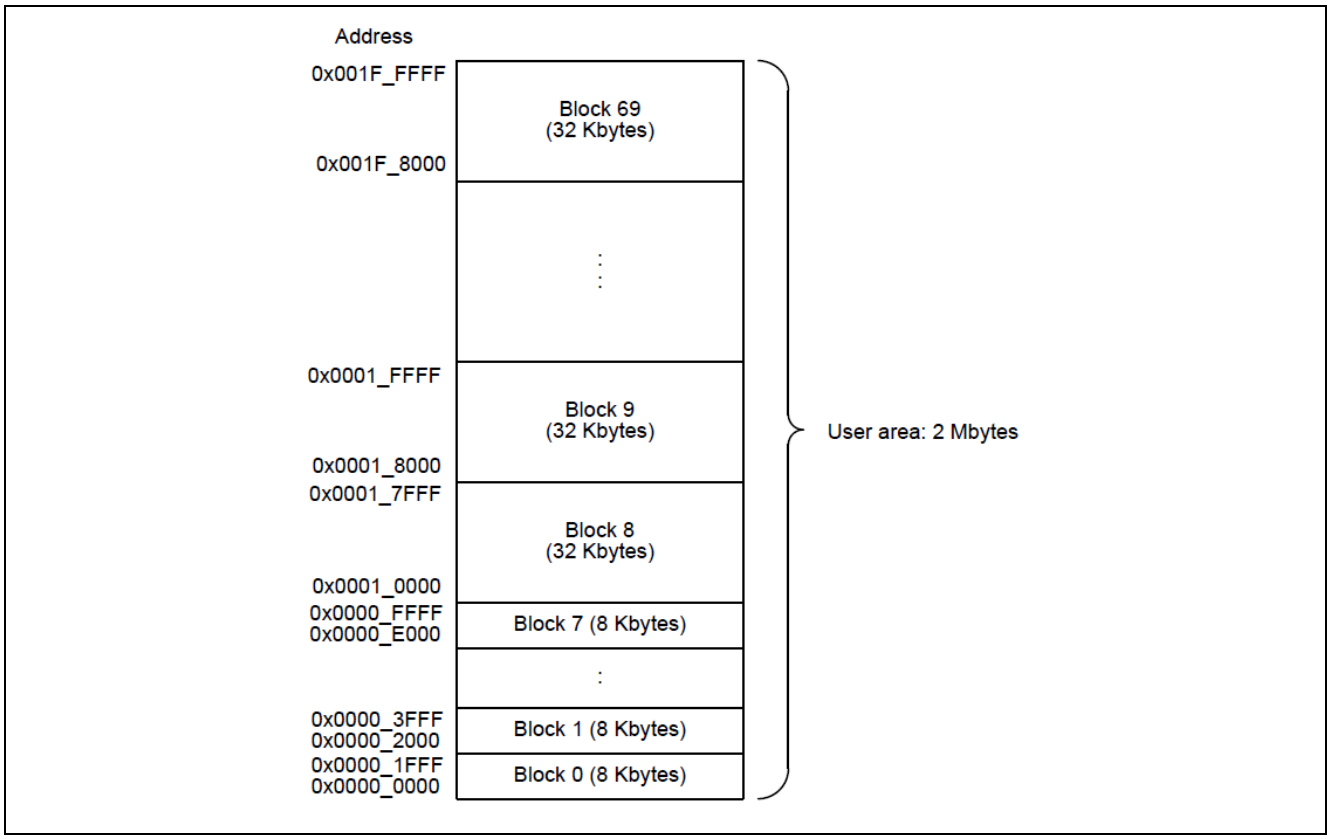

Figure 4. RA6M5 Code Flash Memory in Linear Mode

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 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. RA6M5 Code Flash Memory in Dual Bank Mode. The default configuration is highlighted in the red box.

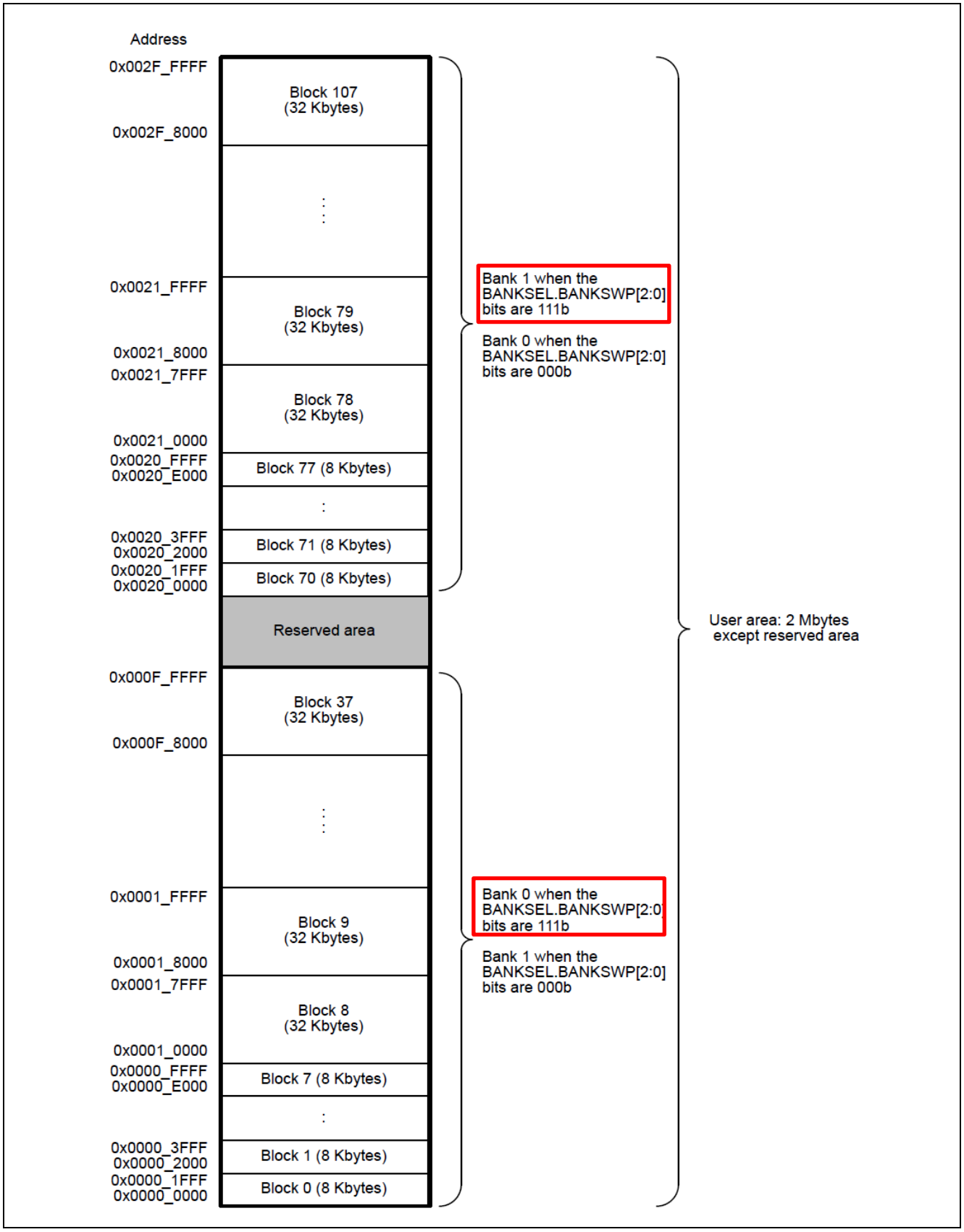

Figure 5. RA6M5 Code Flash Memory in Dual Bank Mode

Table 2. RA6M5 Code Flash 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 Code Flash

Product	Code Flash Range Address	
	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. RA6M5 Code Flash Block Structure is the code flash block structure for RA6M5. The code flash erase minimum unit is the code flash block size. The block numbering scheme is used in the block protection design.

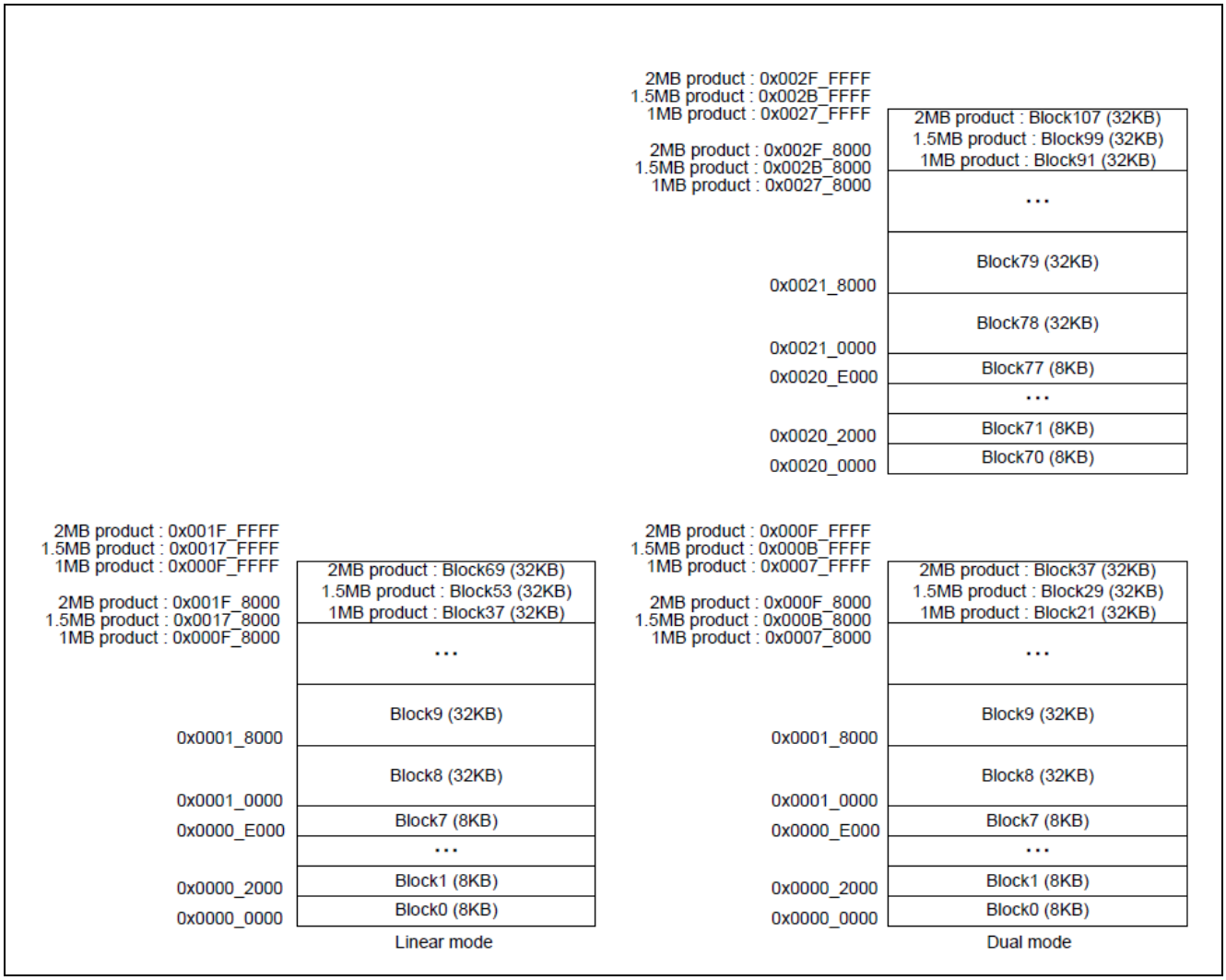

Figure 6. RA6M5 Code Flash Block Structure

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.

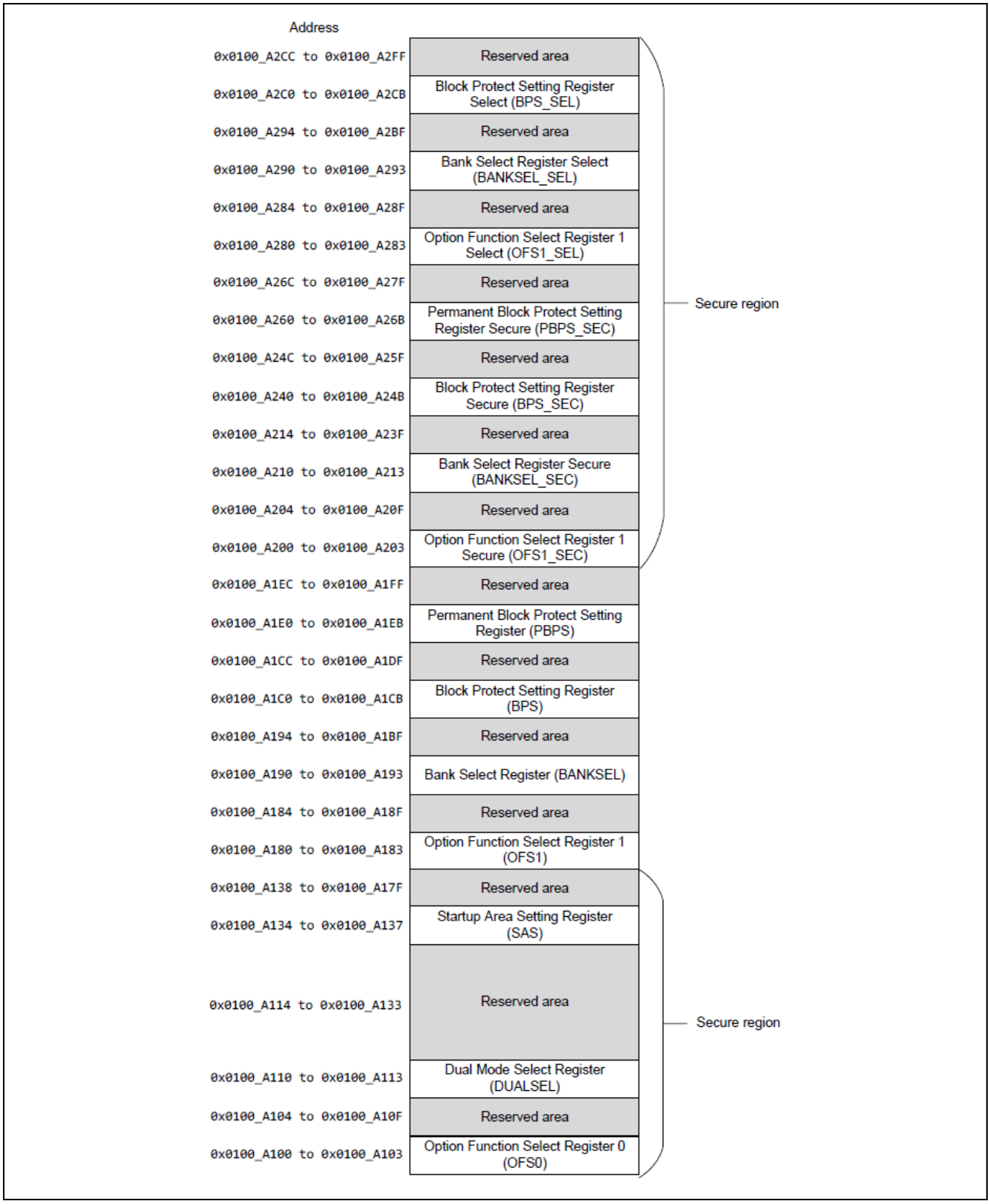

Figure 7. Option-Setting Memory

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. Option-Setting Memory, 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. Enable Flash Dual Bank Mode).

6.2.2 DUALSEL : Dual Mode Select Register

address: 0x0100_A110

Bit position:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset:	User setting*1															

Bit position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	BANKMD[2:0]		
Value after reset:	User setting*1															

Bit	Symbol	Function	R/W
2:0	BANKMD[2:0]	Bank Mode Select 0 0 0: Dual mode 1 1 1: Linear mode Others: Setting prohibited	R
31:3	—	When read, these bits return the written value. The write value should be 1.	R

Note 1. The value in a blank product is 0xFFFF_FFFF. It is set to the value written by your application

BANKMD[2:0] bit (Bank Mode Select)

The BANKMD[2:0] bits select bank mode of the dual bank function of the code flash memory

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. Bank 0 is Default at Address 0x00000000.

6.2.5 BANKSEL, BANKSEL_SEC, BANKSEL_SEL : Bank Select Register

Address: BANKSEL: 0x0100_A190
BANKSEL_SEC: 0x0100_A210
BANKSEL_SEL: 0x0100_A290

Bit position:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	BLCKSWP[3:0]			
Value after reset:	User setting*1															

Bit position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	BANKSWP[2:0]		
Value after reset:	User setting*1															

Bit	Symbol	Function	R/W
2:0	BANKSWP[2:0]	Startup Bank Switch This setting is valid in dual mode. 0 0 0: Start address of Bank0 is 0x0020_0000 and Bnak1 is 0x0000_0000 in dual mode 1 1 1: Start address of Bank0 is 0x0000_0000 and Bnak1 is 0x0020_0000 in dual mode Others: Setting prohibited	R/W
15:3	—	When read, these bits return the written value. The write value should be 1.	R/W
19:16	BLCKSWP[3:0]	Block Swap Select When all bits are set to 1, the block swap is disabled. When at least one bit is set to 0, block swap is enabled and the corresponding blocks of code flash memory are swapped. This setting is valid in linear mode.	R/W
31:20	—	When read, these bits return the written value. The write value should be 1.	R/W

Note 1. The value in a blank product is 0xFFFF_FFFF. It is set to the value written by your application.

Figure 9. Bank 0 is Default at Address 0x00000000

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

BANKSEL_SEL controls whether the BANKSEL or BANKSEL_SEC setting is applied. When BANKSEL_SEL is 0xFFFFFFF8, the setting in BANKSEL is used. When BANKSEL_SEL is 0xFFFFFFFF, the setting in BANKSEL_SEC is used. For Non-TrustZone 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.

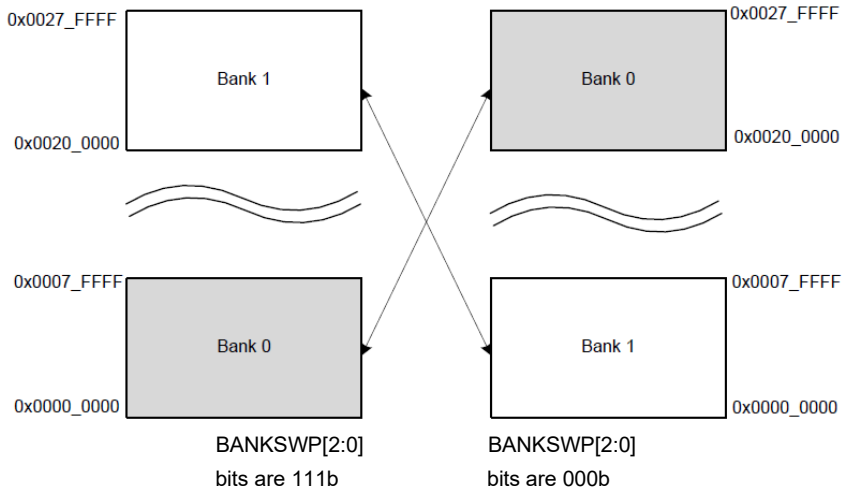

Figure 10. Example of Startup Bank Selection (For Products with 1 Mbyte of Code Flash Memory)

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 takes 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 operation.

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

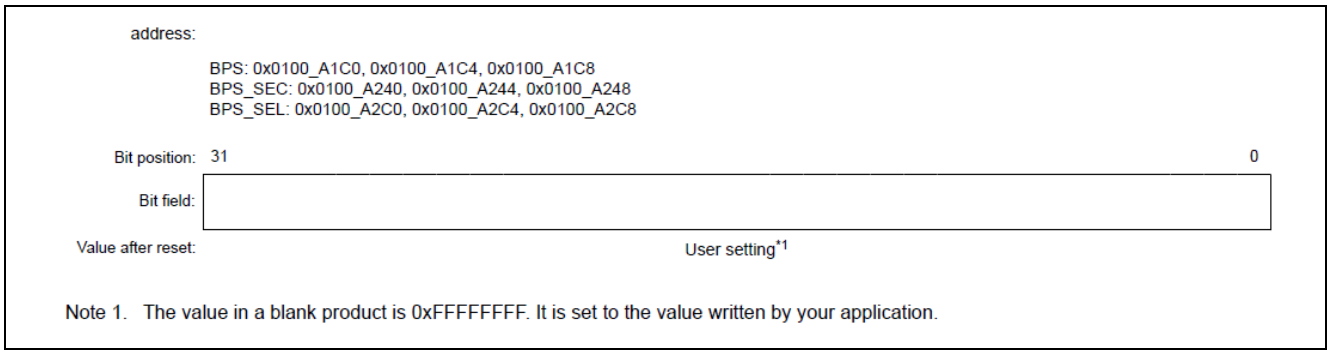

Figure 11. Registers Related to Temporary Code Flash Block Protection

The BPS_SEL register is the security attribution register; it controls whether the BPS or BPS_SEC setting is applied. When the bit of PBS_SEL is set to 0, the corresponding bit of the Secure register BPS_SEC is applied.

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. For Non-TrustZone based Flat projects, BPS_SEL selects the corresponding bits in the BPS_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. Temporary Protection of the Lower Bank Bootloader Area and Figure 85. Temporary Protection of the Upper Bank Bootloader Area.

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

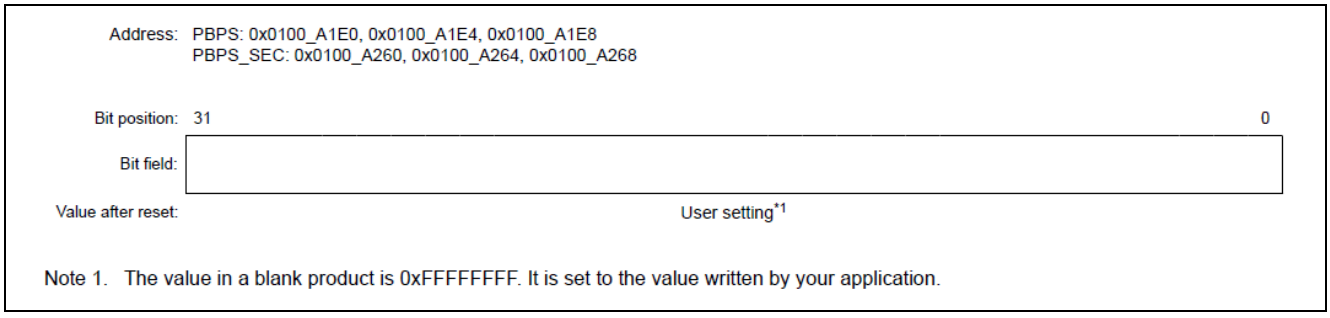

Figure 12. Registers Related with Permanent Code Flash Block Protection

The BPS_SEL also controls whether the PBPS or PBPS_SEC setting is applied. When the bit of PBS_SEL is set to 0, the corresponding bit of the Secure register PBPS_SEC is applied.

Only secure developers can program the PBPS_SEC register. The PBPS_SEC register is for secure developers, and the PBPS register is for non-secure developers. For Non-TrustZone based Flat projects, BPS_SEL selects the corresponding bits in the PBPS_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.

The PBPS and PBPS_SEC registers invalidate writes to bits of BPS and BPS_SEC registers. The bit of this register can be set to 0 when the corresponding bit of BPS and BPS_SEC is set to 0. When the bit of the PBPS or PBPS_SEC register is set to 0, writing the corresponding bit of the BPS and BPS_SEC register is invalid. Once the bit of this register is set to 0, it is impossible to change the bit to 1.Setting of these registers can be achieved by configuring the BSP Properties in the RA configurator, as shown in Figure 86. Permanent Protection of the Lower Bank Bootloader Area and Figure 87. Permanent Protection of the Upper Bank Bootloader Area.

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 mcu-tools github project. In 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 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: `imgtool.py`. This tool provides services for creating Root keys, key management, 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 image 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 Use Direct XIP Firmware Update Mode.

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. Memory Architecture Using Flash Dual Bank Mode and MCUboot. In addition, the memory allocation for the bootloader and application image must be identical.

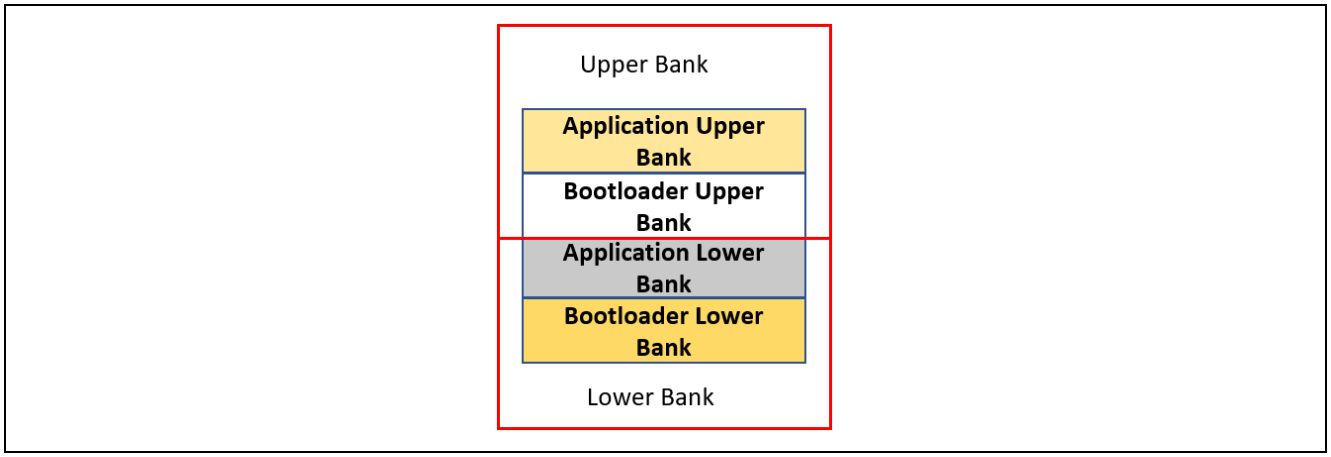

Figure 13. Memory Architecture Using Flash Dual Bank Mode and MCUboot

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, and compare with the bootloader memory allocation for consistency and adjust as needed.
- Determine the bootloader configurations in terms of image authentication and new image update mode. This may result in an 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. Enable Flash Dual Bank Mode.
   B. Update the application image code flash memory allocation if needed. See section 4.2 Configure the Memory Configuration and Authentication Method 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 Preparation for Production Support to understand the updates needed.

3. Guidelines for Using the Example Projects Included

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

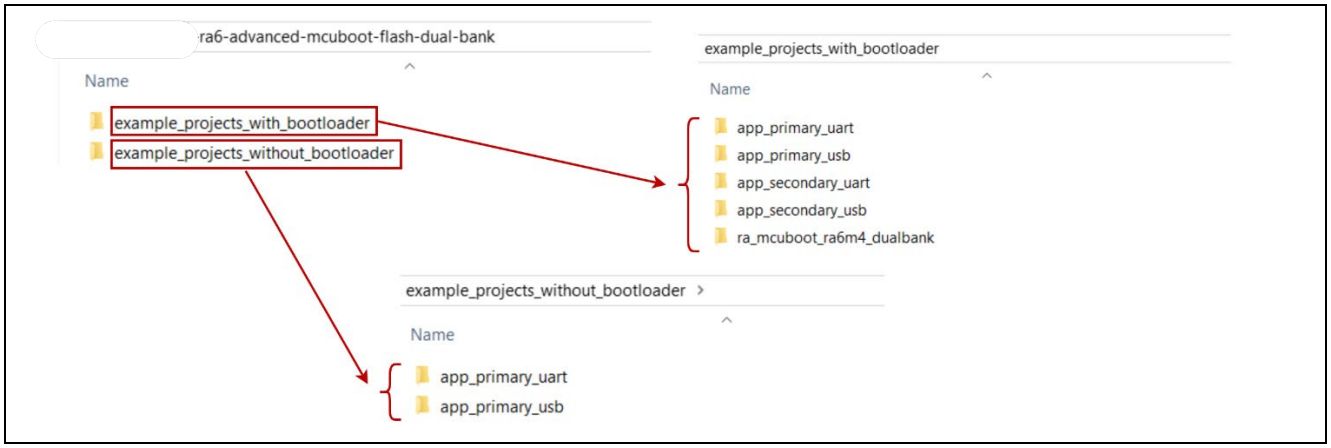

Figure 14. Example Projects Included

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 which are configured to use the bootloader included in this folder. Users with experience working with MCUboot module can follow section 8. Compile and Exercise the Included Example Bootloader and Application Projects 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. Configuring and Signing an Application Project. 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. Configuring and Signing an Application Project 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

The following steps will guide you on starting 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.**

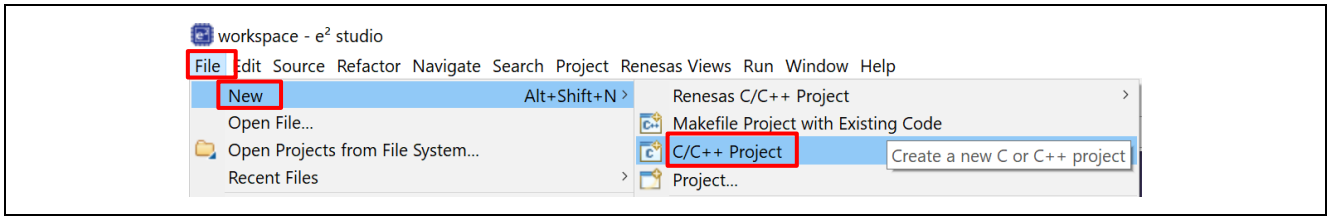

Figure 15. Start a New Project

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

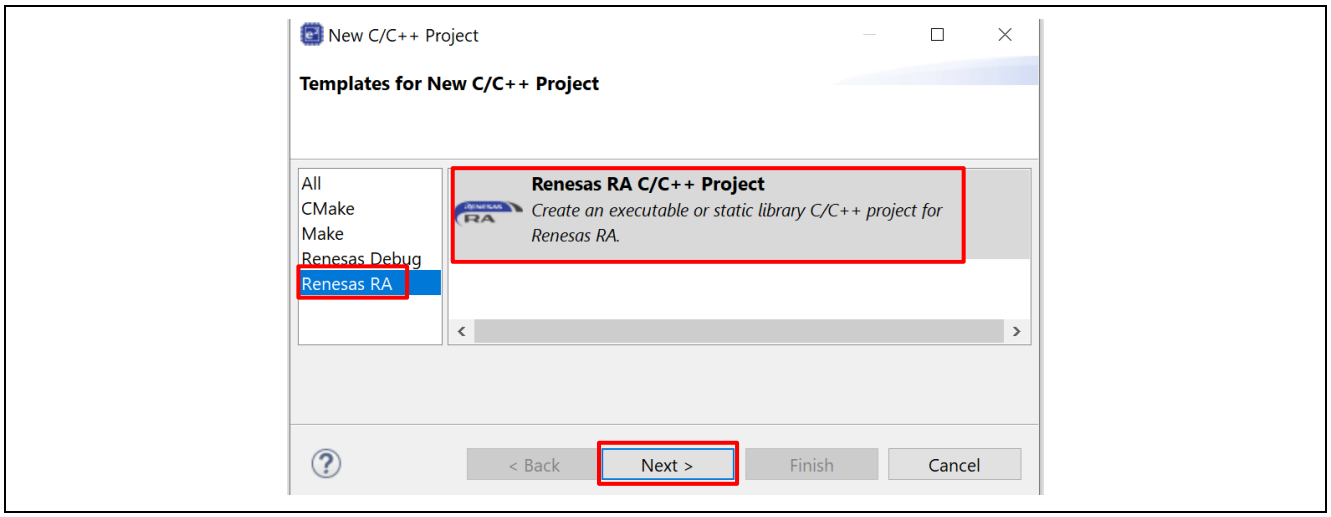

Figure 16. Choose Renesas RA C/C++ Project

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**.

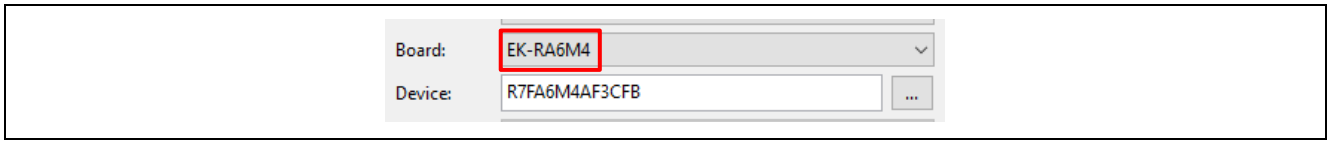

Figure 17. Select the Board

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

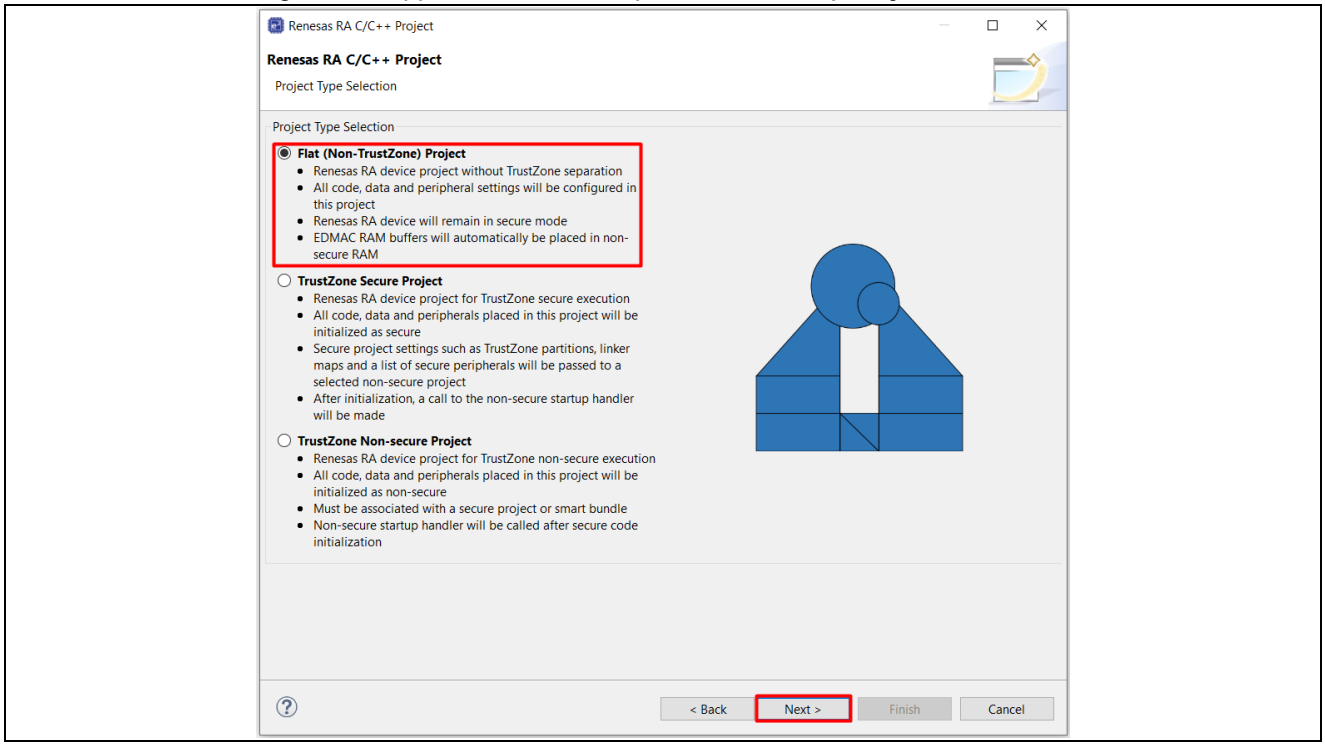

Figure 18. Choose Flat Project as Project Type

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

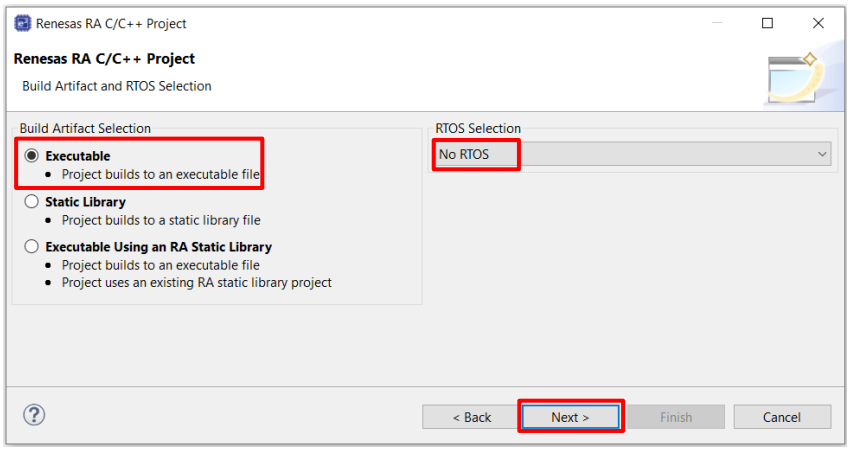

Figure 19. Choose to Build Executable and No RTOS

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

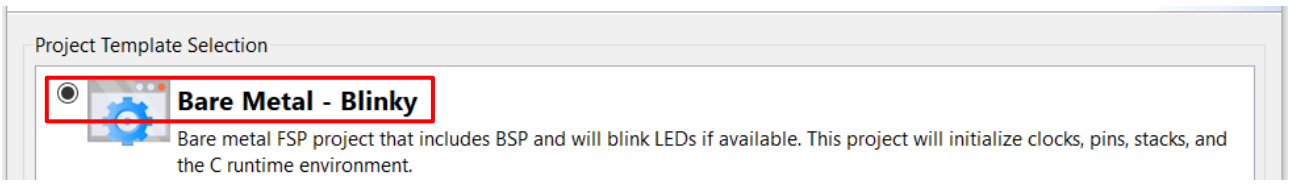

Figure 20. Choose the Project Template

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

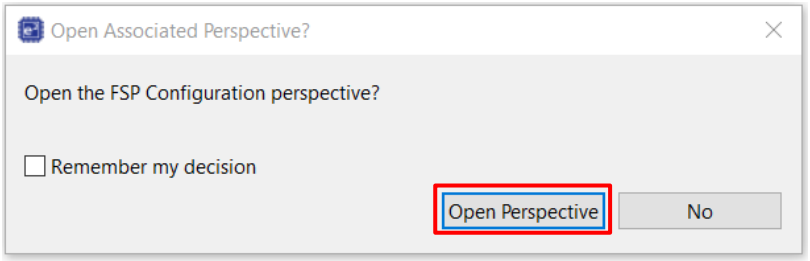

Figure 21. Choose Open the FSP Configuration Perspective

The project is then created, and the bootloader project configuration is displayed.

9. Select the **Pins** tab and uncheck **Generate data** for **RA6M4 EK.**

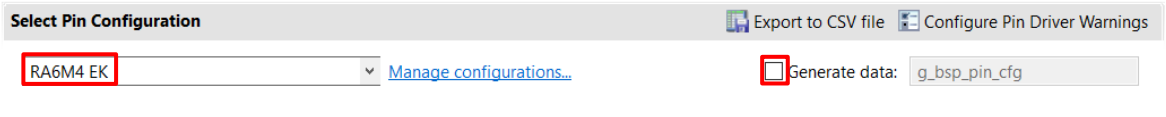

Figure 22. Uncheck Generate data for RA6M4 EK Pin Configuration

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.

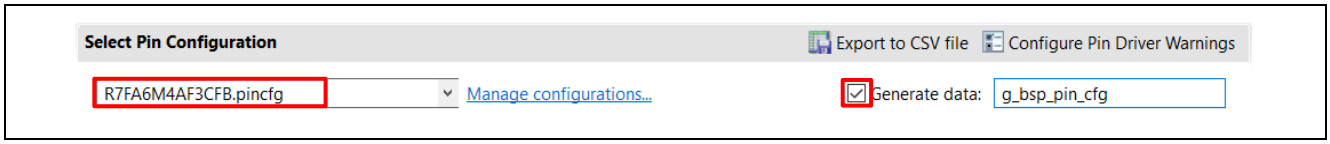

Figure 23. Select g_bsp_pin_cfg and Generate data g_bsp_pin_cfg

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

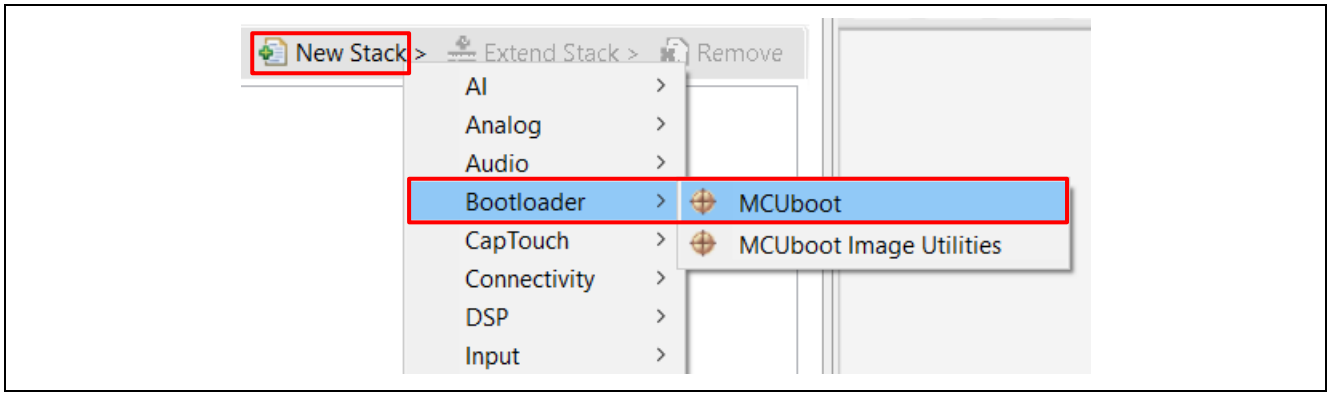

Figure 24. Add the MCUboot Port

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**.

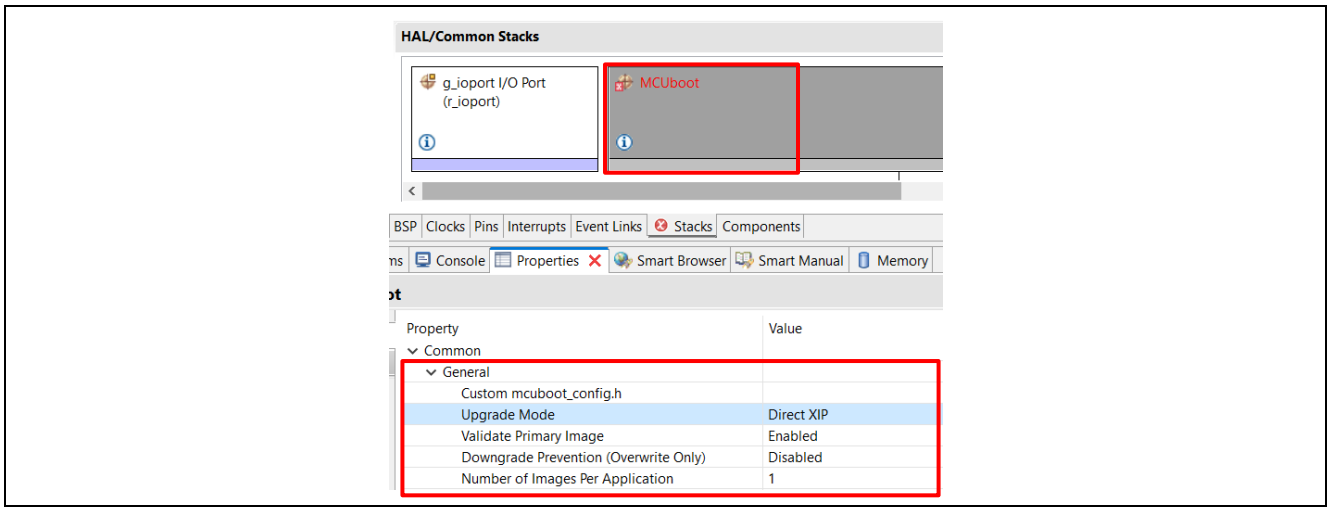

Figure 25. General Configuration for MCUboot Module

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 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**, a 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. MCUboot Dual Bank Memory Map. This default memory map is used for the example bootloader design.

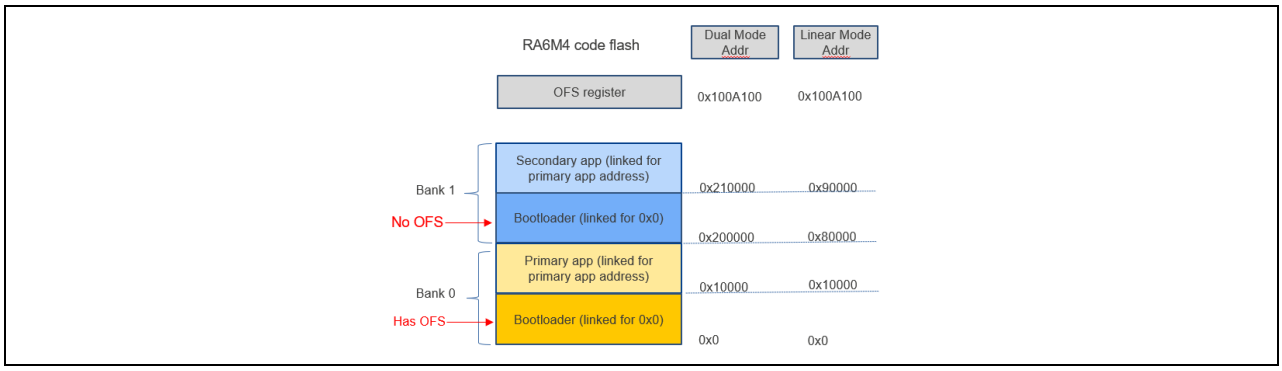

Figure 26. MCUboot Dual Bank Memory Map

From the configurator point of view, there is no need to update any of the properties for the Flash Layout as it already matches with the memory map shown in Figure 26. MCUboot Dual Bank Memory Map.

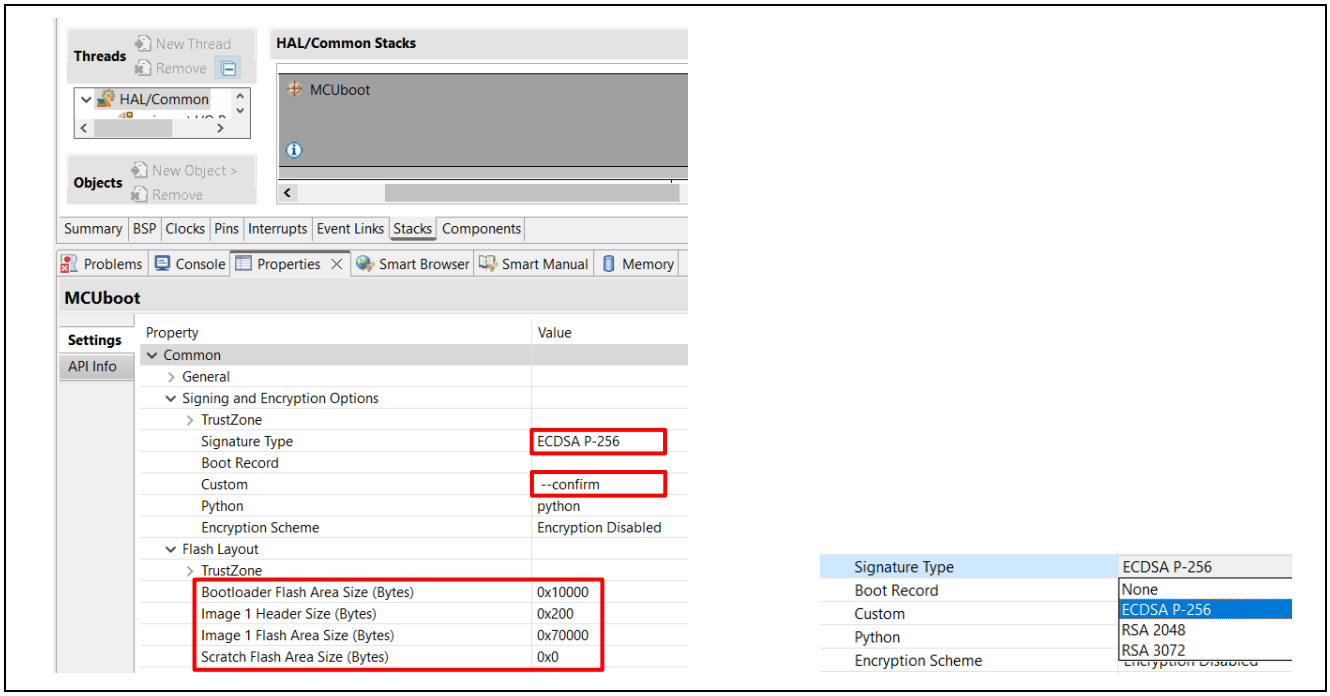

Figure 27. Configure the Flash Layout and Signing Options

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

The following steps will guide you to configure the MbedTLS module and the flash driver:

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

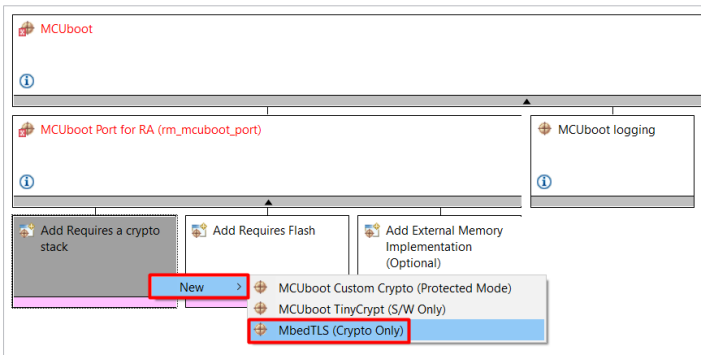

Figure 28. Select MbedTLS Crypto Only Module

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

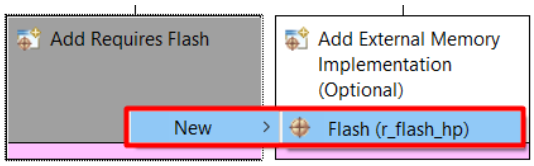

Figure 29. Add the Flash Driver

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.

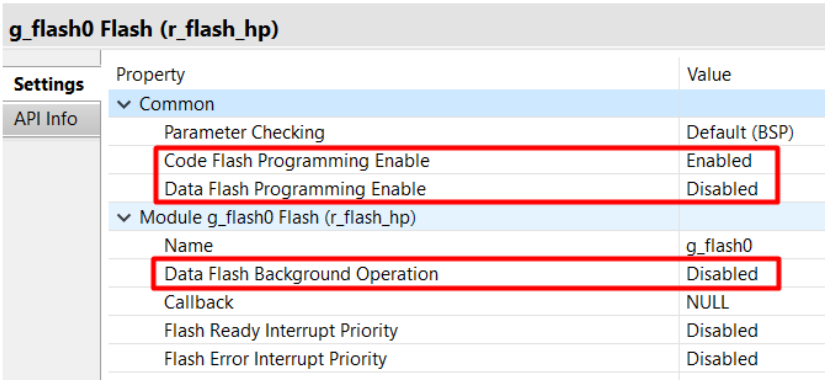

Figure 30. Configure the Flash Driver

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

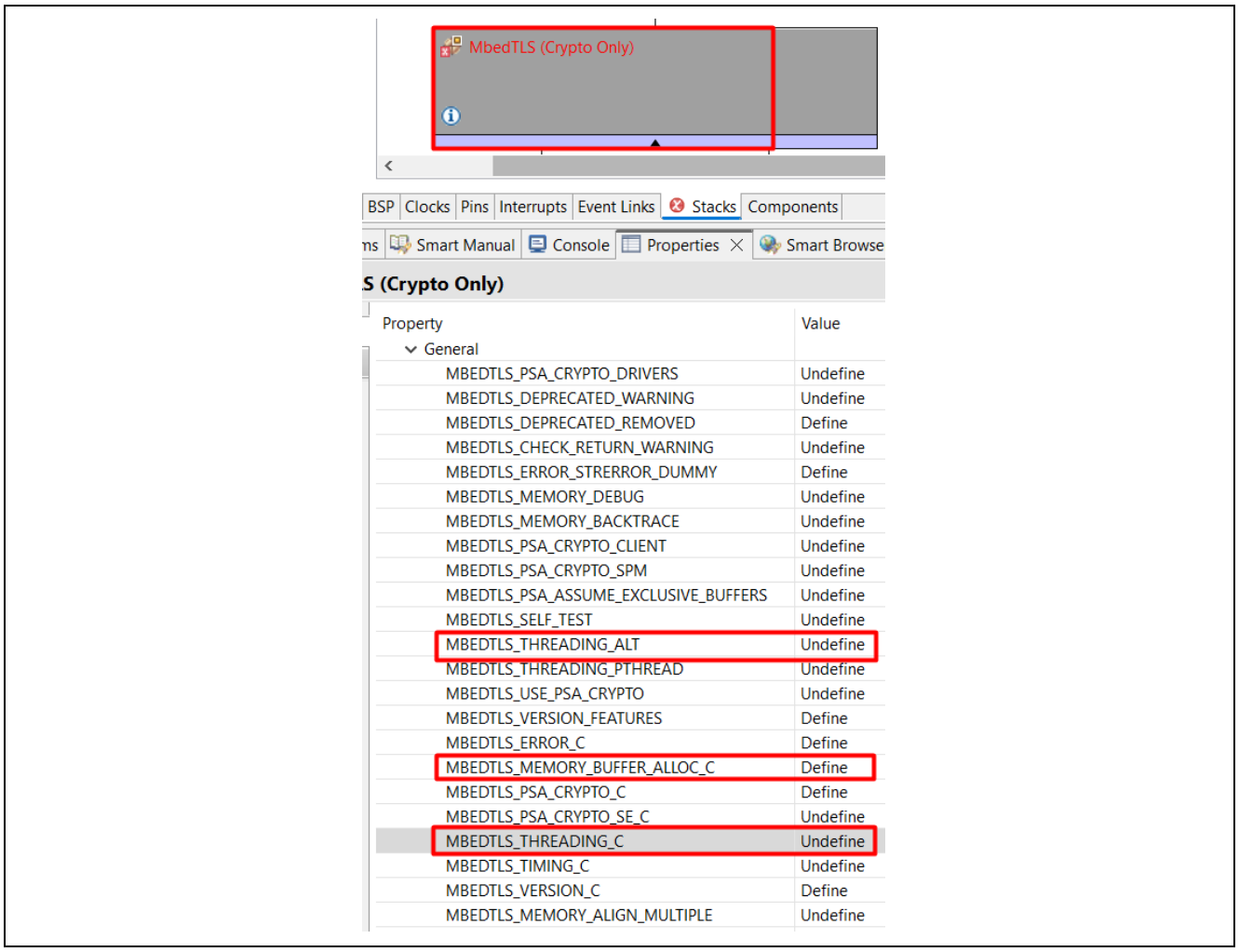

Figure 31. Configure the MbedTLS (Crypto Only) Module

5. Disable RSA to save some memory usage.

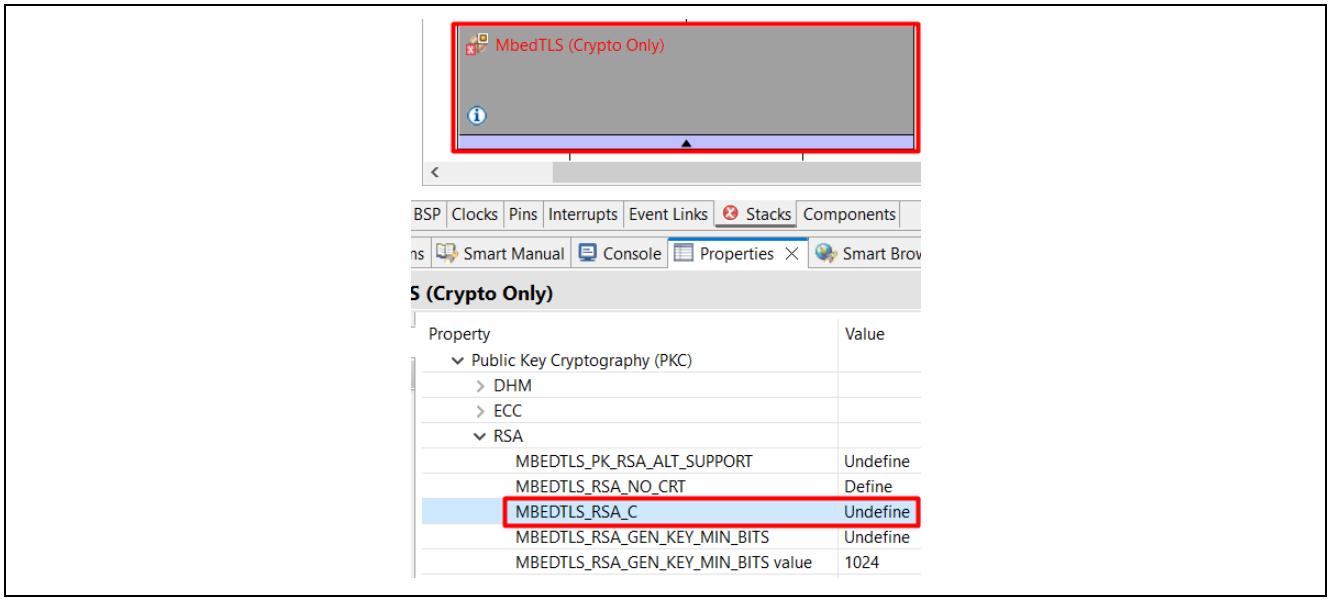

Figure 32. Disable RSA

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

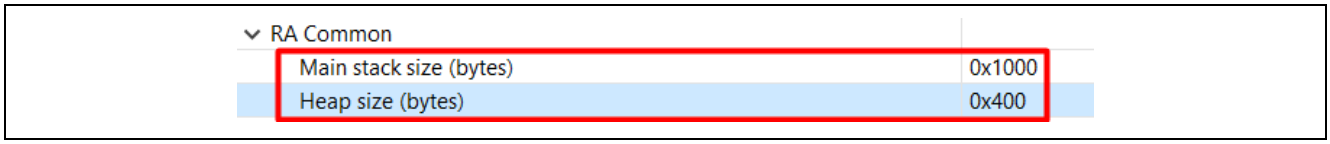

Figure 33. Configure the BSP Stack and Heap Usage

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

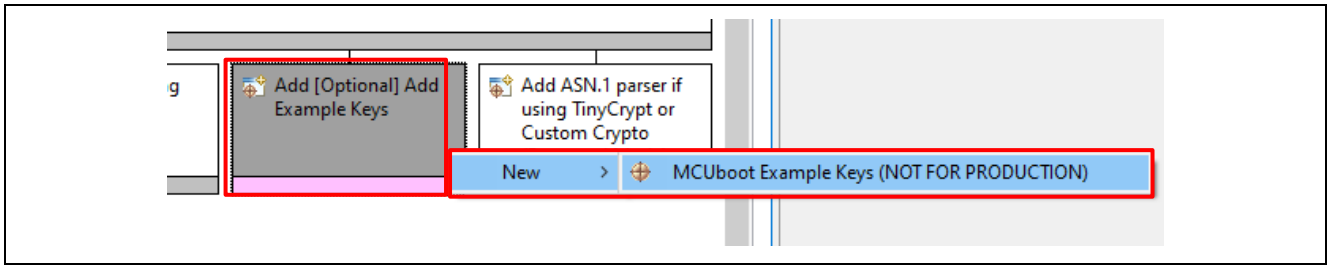

Figure 34. Add the Example Production Key module

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

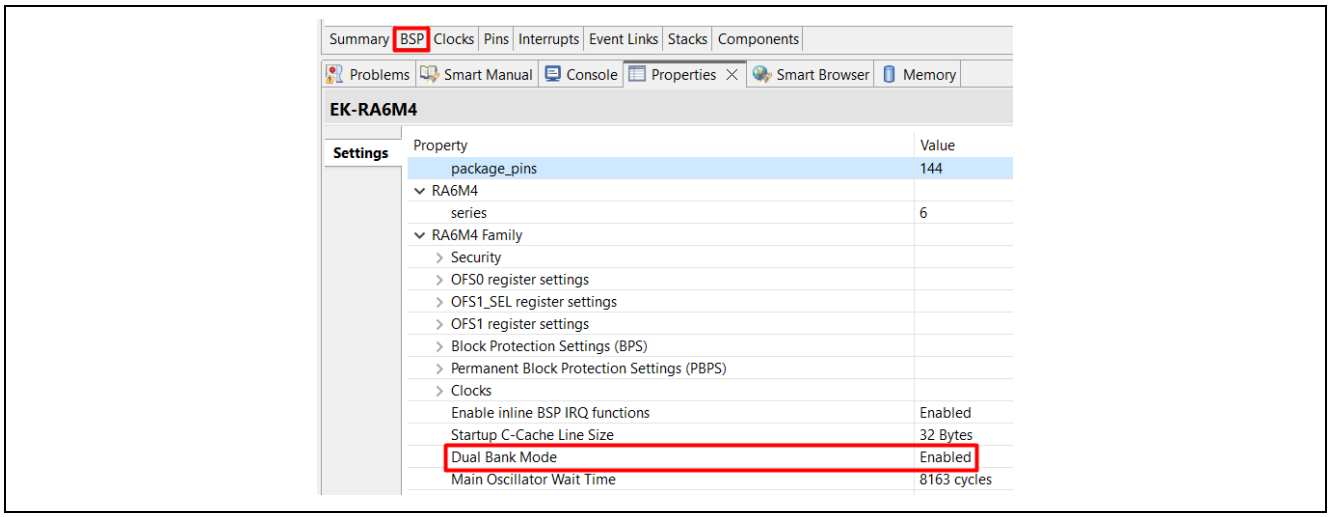

Figure 35. Enable Flash Dual Bank Mode

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 e² studio, using the image signing tool `imgtool.py`, which is included with MCUboot. This tool is integrated as a post-build tool in e² 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. Configuring and Signing an Application Project.

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.

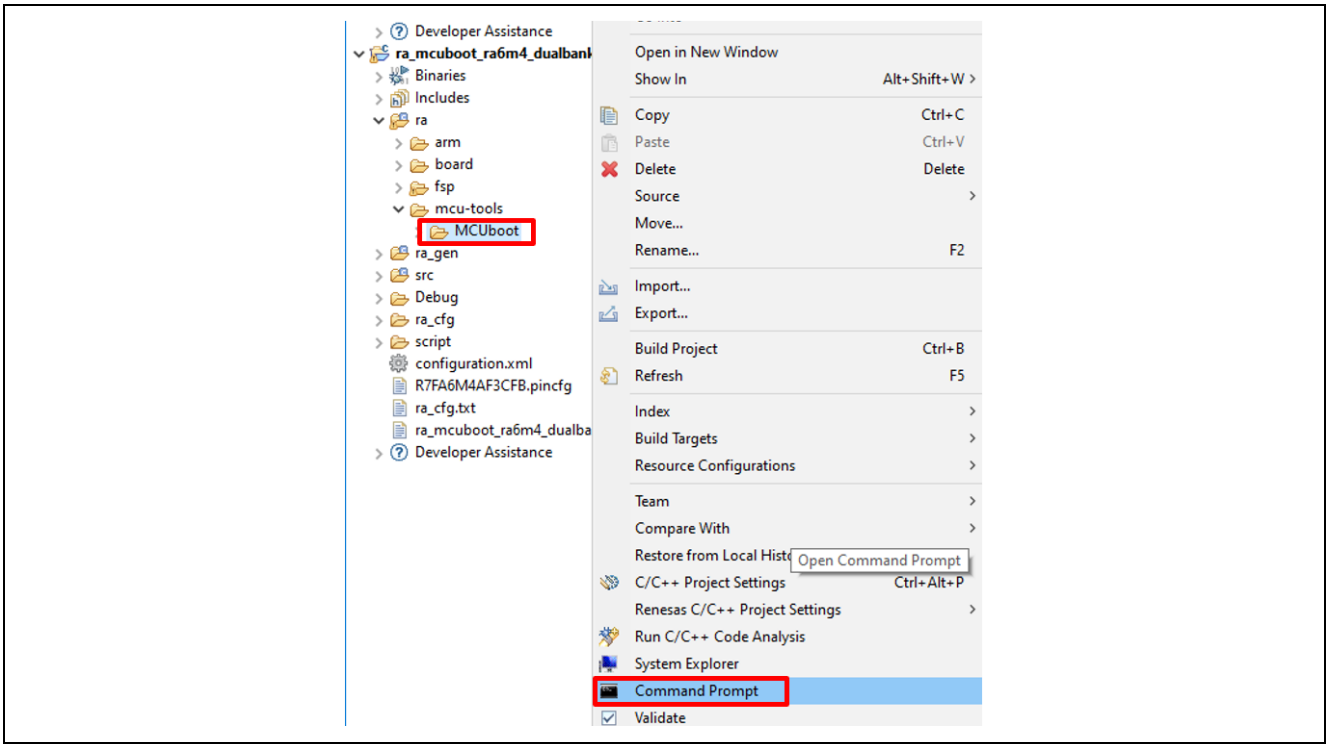

Figure 37. Open the Command Prompt

We recommend upgrading 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 dependencies that are required.

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 section 5.1 Configure the Application Project to Use the Bootloader. 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
    ${BuildArtifactFileName} ${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 RA6M4 and RA6E1 MCU Group Code Flash Configuration and section 1.2 RA6M5 MCU Group Code Flash Configuration, 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:

1. Unzip `r11an0570eu0140-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 **Builders** page**.** Click **New** to start creating the customized Builder.

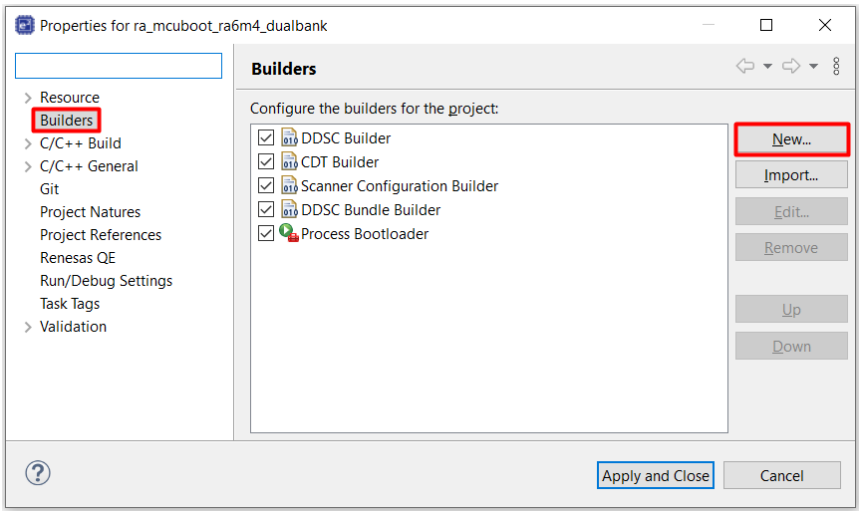

Figure 40. Create a New Custom Builder Entry

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

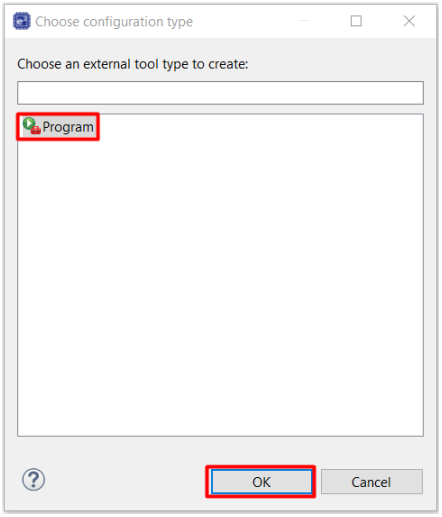

Figure 41. Select the Type of the Builder as Program

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**.

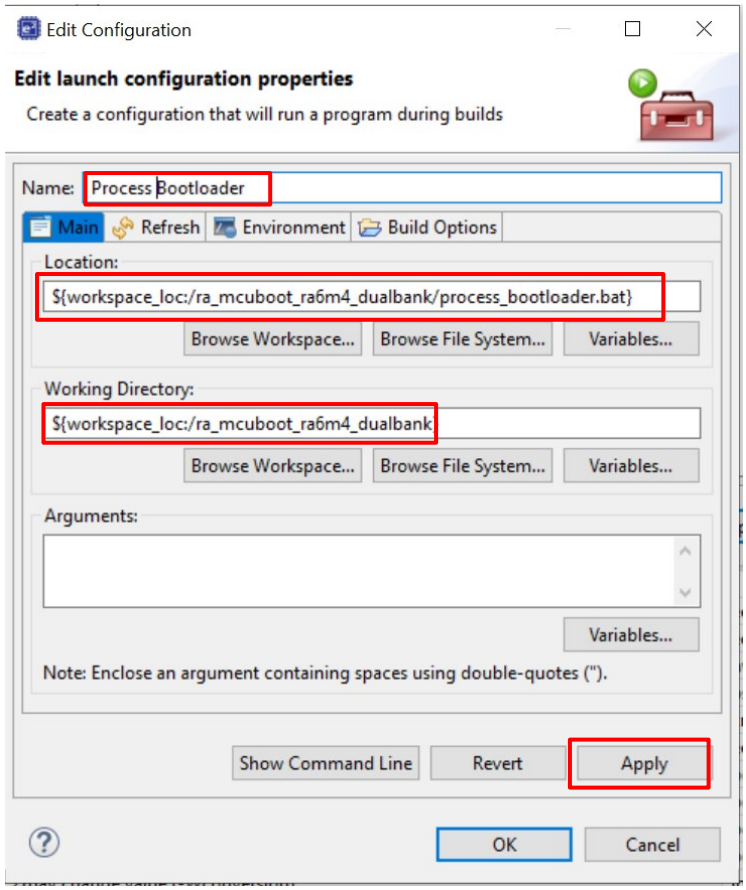

Figure 42. Configure the Custom Builder

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

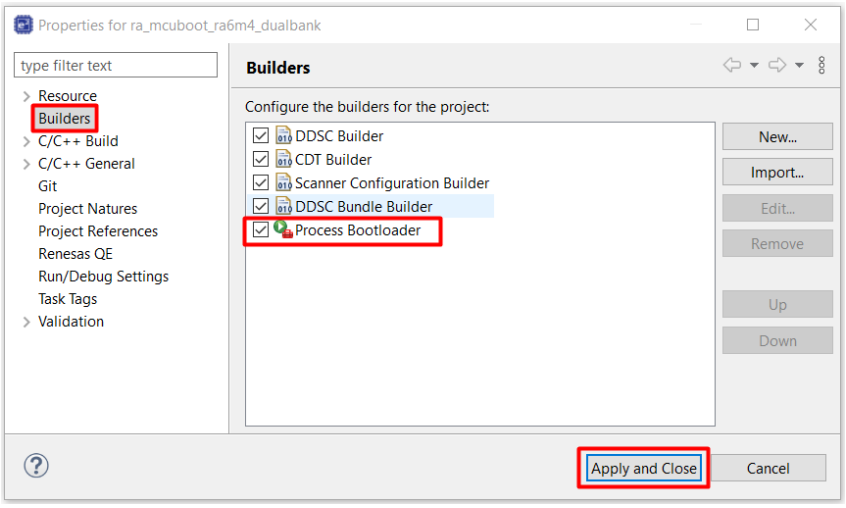

Figure 43. Custom Builder

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

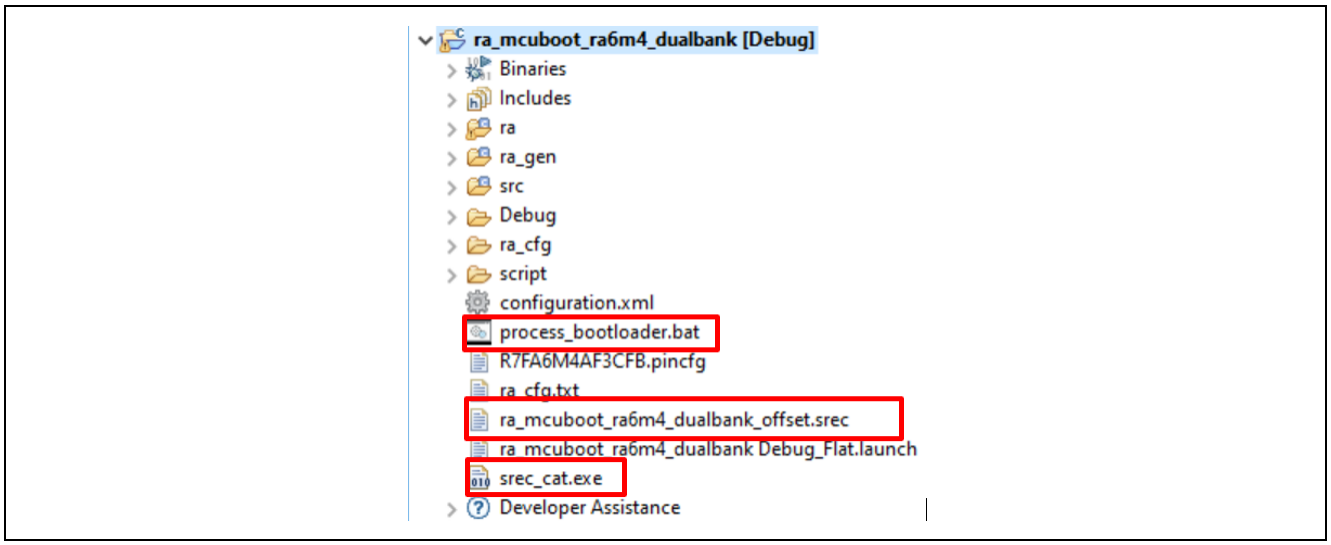

Figure 44. Rebuild the Bootloader with the Custom Builder

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

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 of 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.

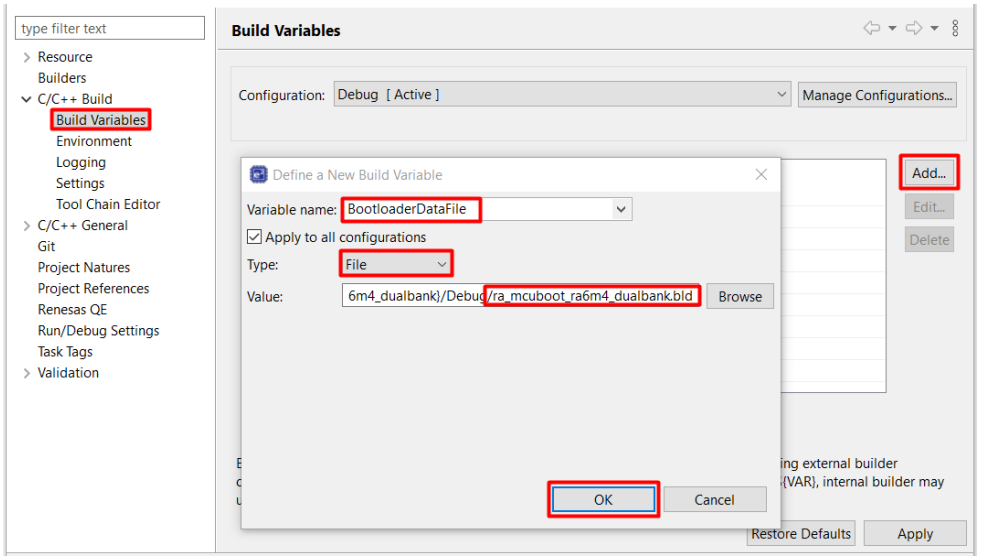

Figure 45. Configure the Build Variable to Use the Bootloader

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 the 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 `app_primary_usb`, under **Environment**, click **Add** and configure MCUBOOT_IMAGE_VERSION.

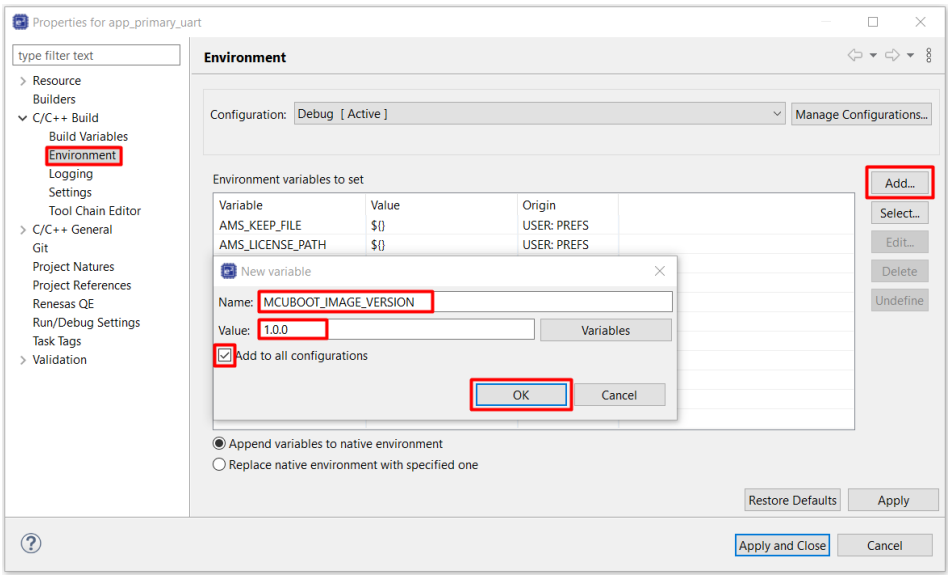

Figure 46. Configure the Application Version

Similarly, add the new variable for MCUBOOT_IMAGE_SIGNING_KEY.

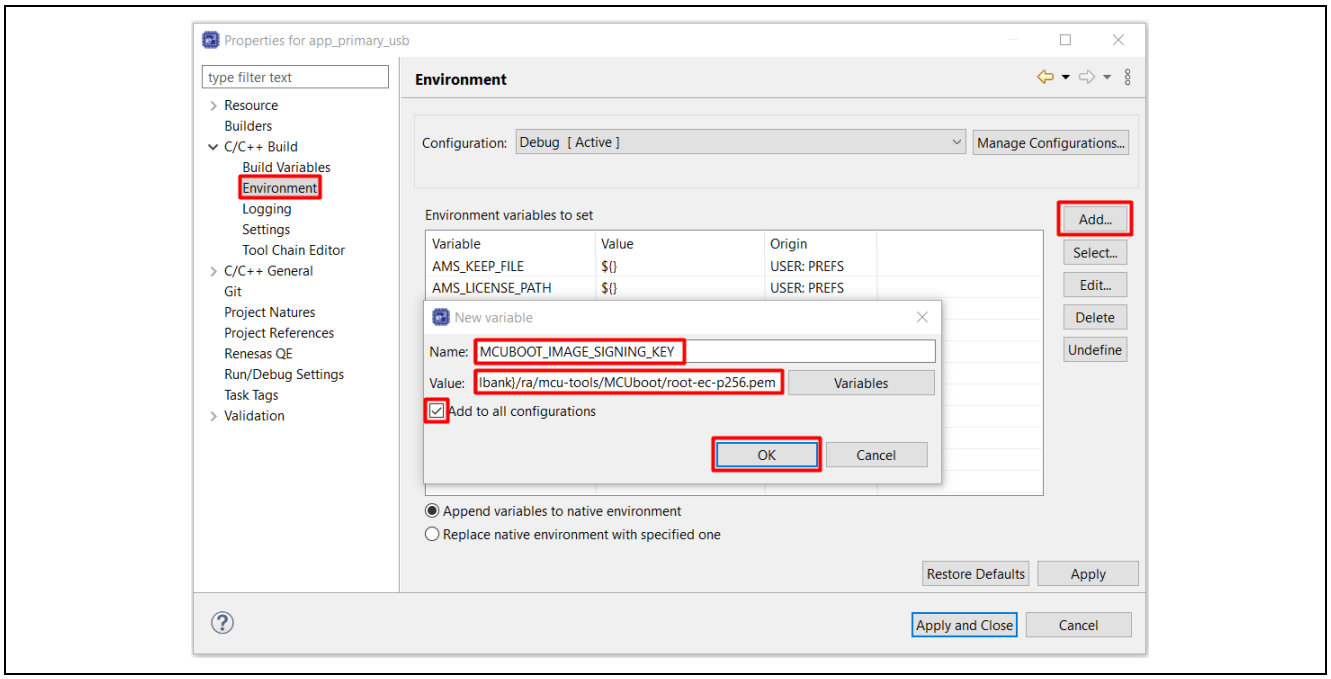

Figure 47. Configure the Private Signing Key

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 purposes only. For real-world use case and production support, users MUST change this to the private key of their choice.

Figure 48. Configure the Application Image version number and Signing Key is the result of the above configuration. Click **Apply and Close**.

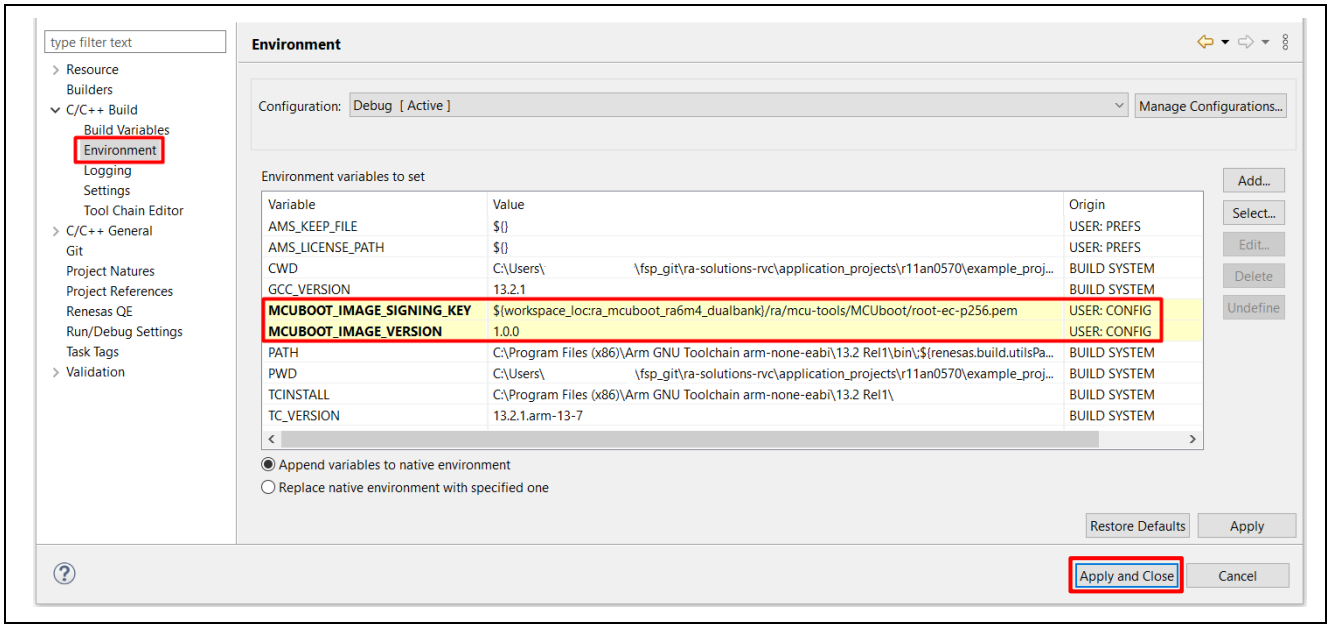

Figure 48. Configure the Application Image version number and Signing Key

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

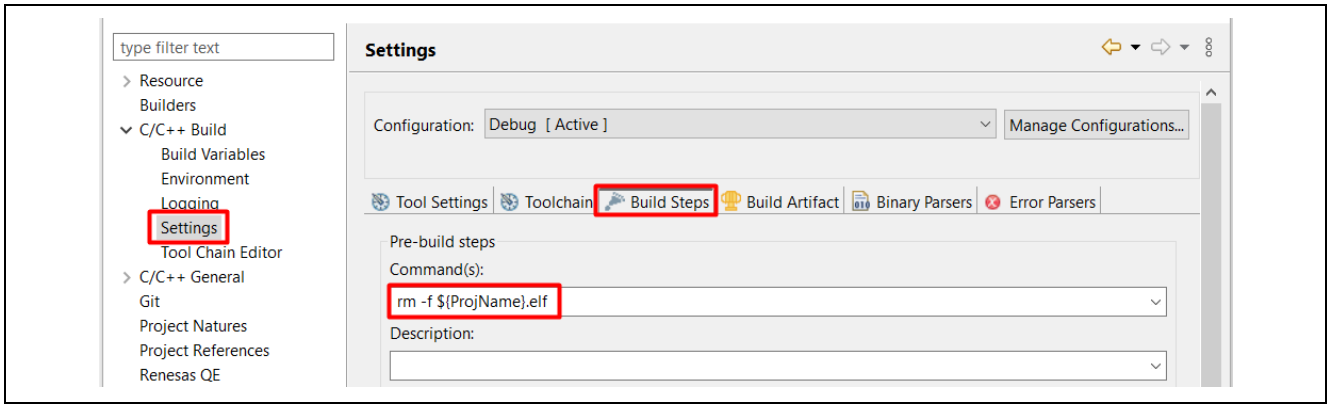

Figure 49. Configure the Pre-build Command

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. MCUboot Dual Bank Memory Map.

```
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 Prepare for Production Support 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 Prepare for Production Support to create the new **Builder**. The finished configuration should look like Figure 51. Configure the Custom Builder for the Primary Application.

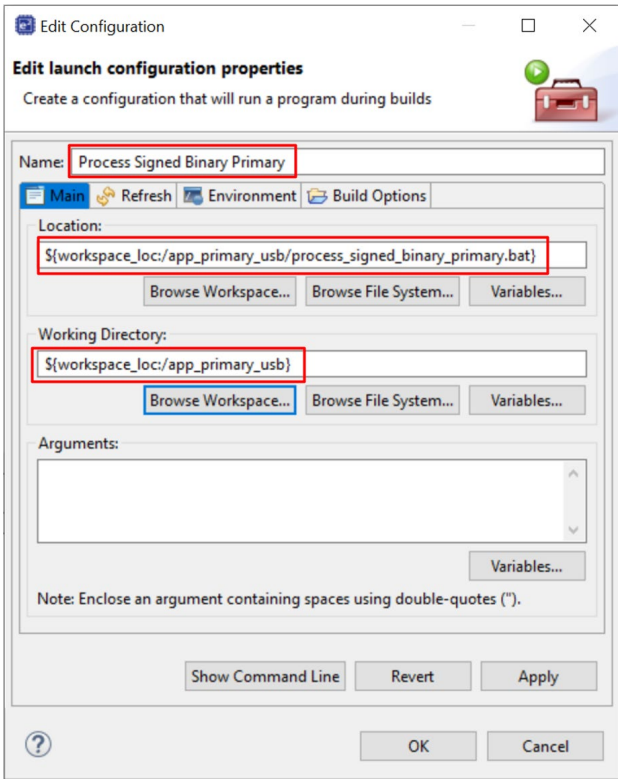

Figure 51. Configure the Custom Builder for the Primary Application

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.

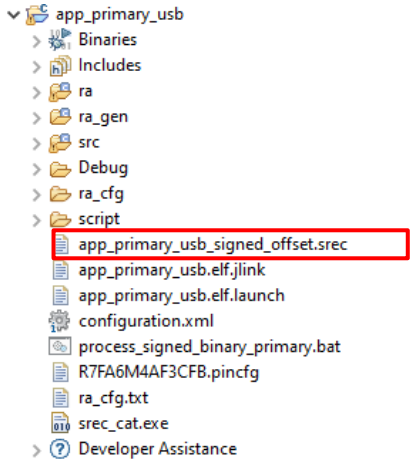

Figure 52. Signed Primary Image Offset to the Primary Slot

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. Configuring and Signing an Application Project. 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**.

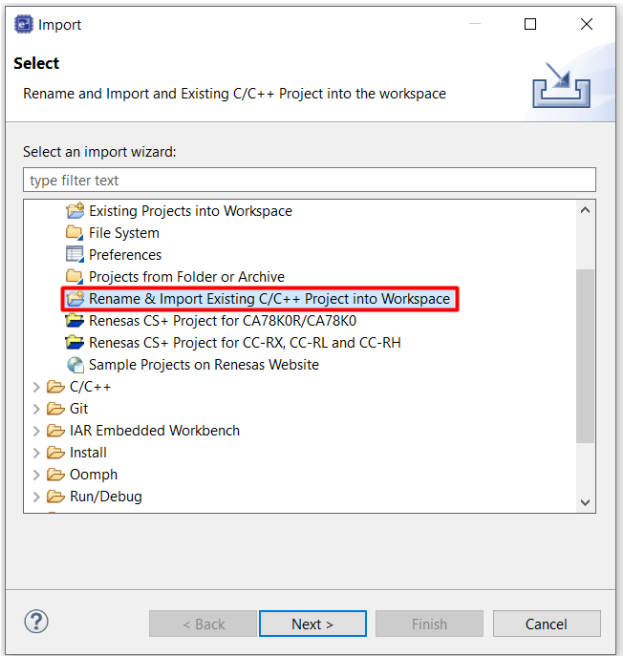

Figure 53. Import the Initial Application

Once the **Import** window opens, name the project `app_secondary_usb`, check **Select root directory**, and click **Browse**:

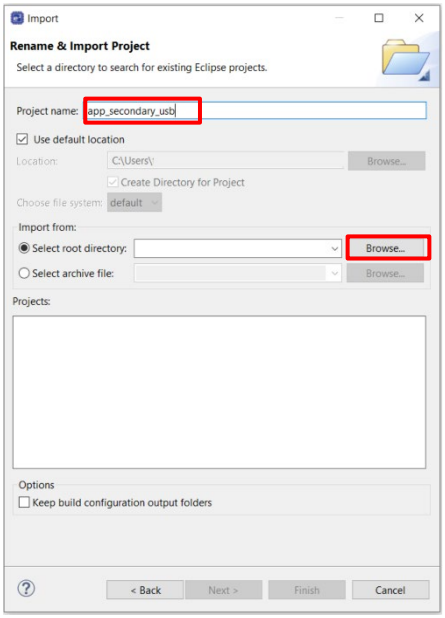

Figure 54. Name the New Application

Browse into the Workspace folder and select `app_primary_usb`.

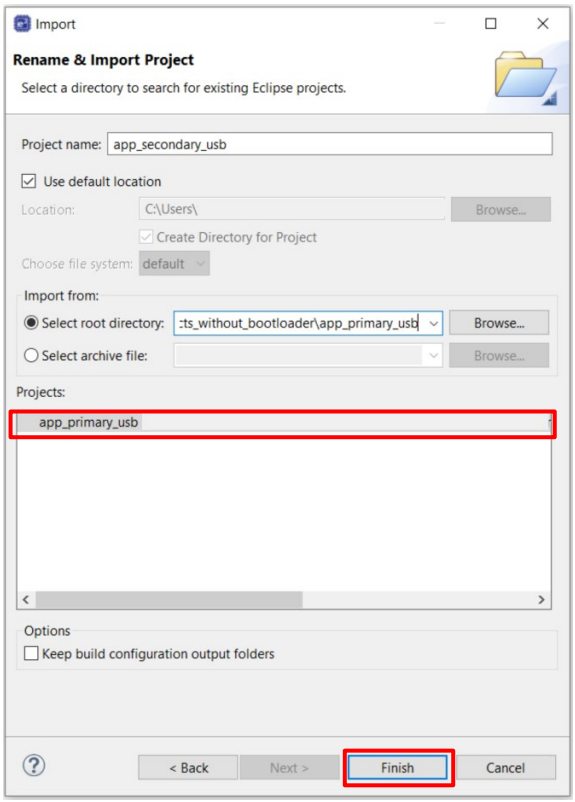

Figure 55. Select Initial Primary Application

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 Environment variable for the Secondary Image version, shown in Figure 56. Change MCUBOOT_IMAGE_VERSION Variable.

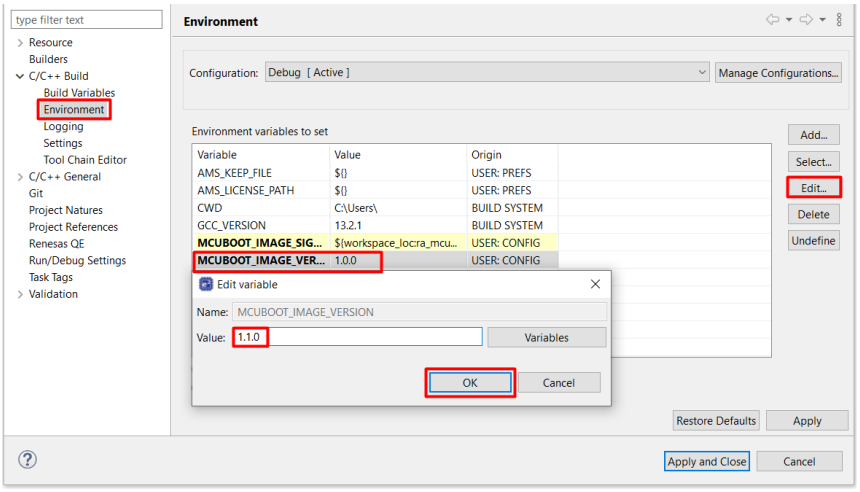

Figure 56. Change MCUBOOT_IMAGE_VERSION Variable

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);
```

Figure 57. Update the LED Control

Save the updated source file, click **Generate Project Content**, and 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. Configuring and Signing an Application Project. 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 onboard debugger.
- USB FS device mode jumper setting: Connect pins 2 and 3 on J12, and 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 onboard debugger.
- Connect the three pins in Table 3. Connection through the UART Interface 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, make sure to restore to linear mode prior to 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 J10 USB Debug. Launch RFP and create a new RFP project. Click **File** -> **New Project**.

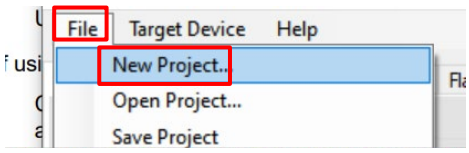

Figure 58. Create a New RFP Project

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

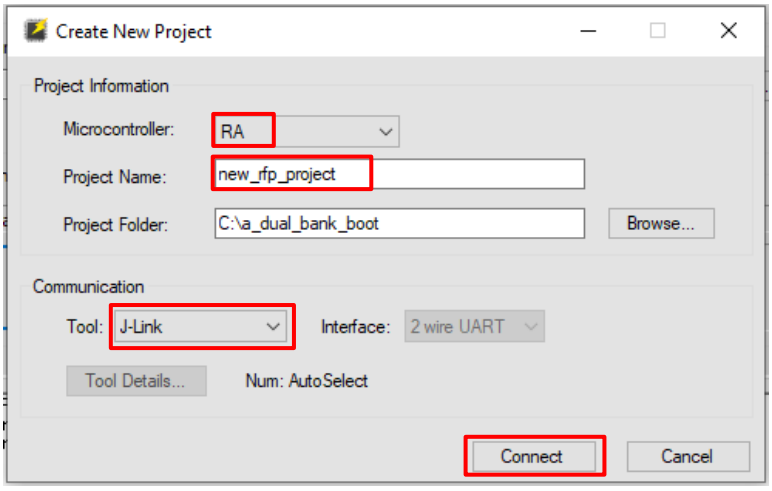

Figure 59. Configure the New Project

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

If the RA6M4 flash is in code flash linear mode, **Blocks Settings** are presented as in Figure 60. Flash in Linear Mode.

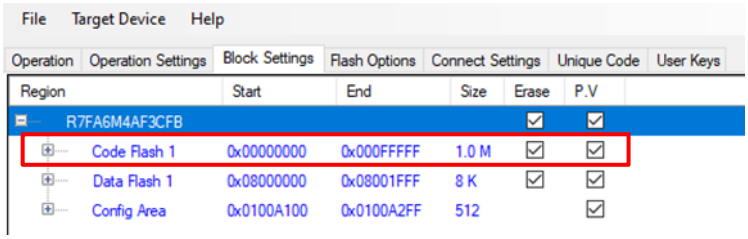

Figure 60. Flash in Linear Mode

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

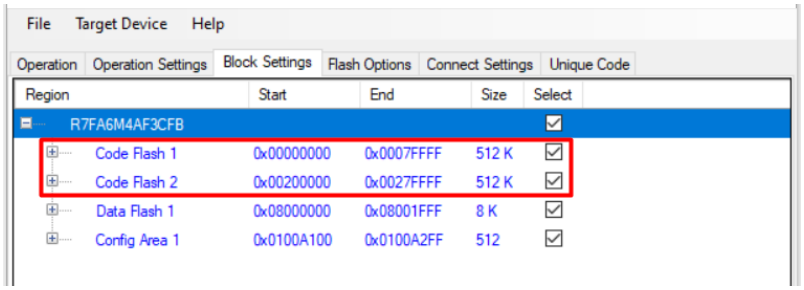

Figure 61. Flash in Dual Bank Mode

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.

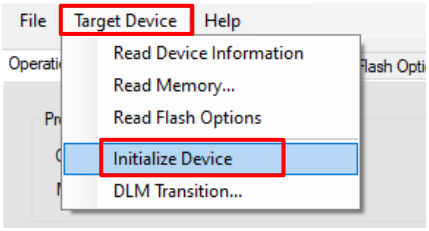

Figure 62. Initialize Device Command

If the **Initialize Device** is successful, the message in Figure 63. Initialize Device Succeeded 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 that 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 J-Link software & documentation pack 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.

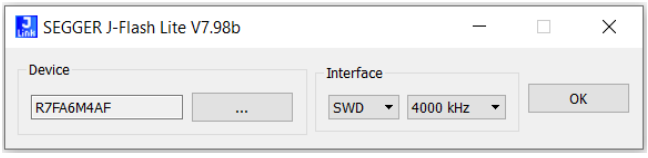

Figure 64. Launch the J-Flash Lite

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

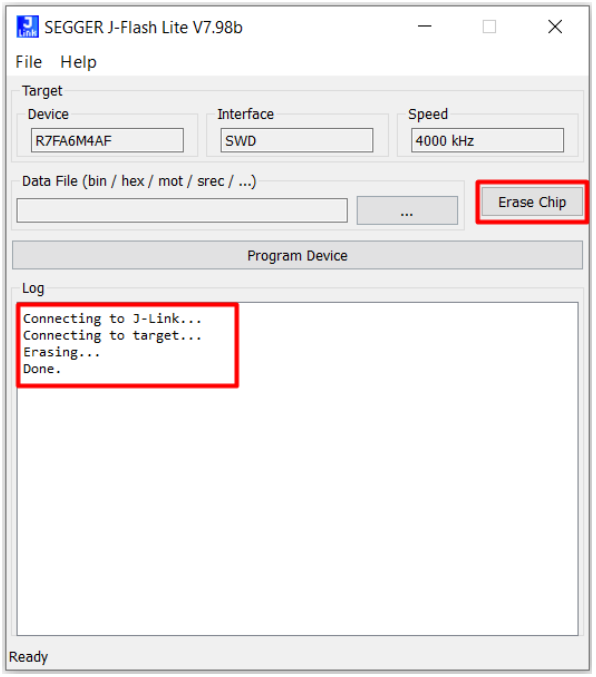

Figure 65. Erase the MCU using J-Flash Lite

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 e^2 studio, navigate to **Run** -> **Renesas Debug Tools** -> **Renesas Device Partition Manager**. Select J-Link as the connection method and select the action **Initialize device back to factory default**.

Click **Run**. The MCU will be erased.

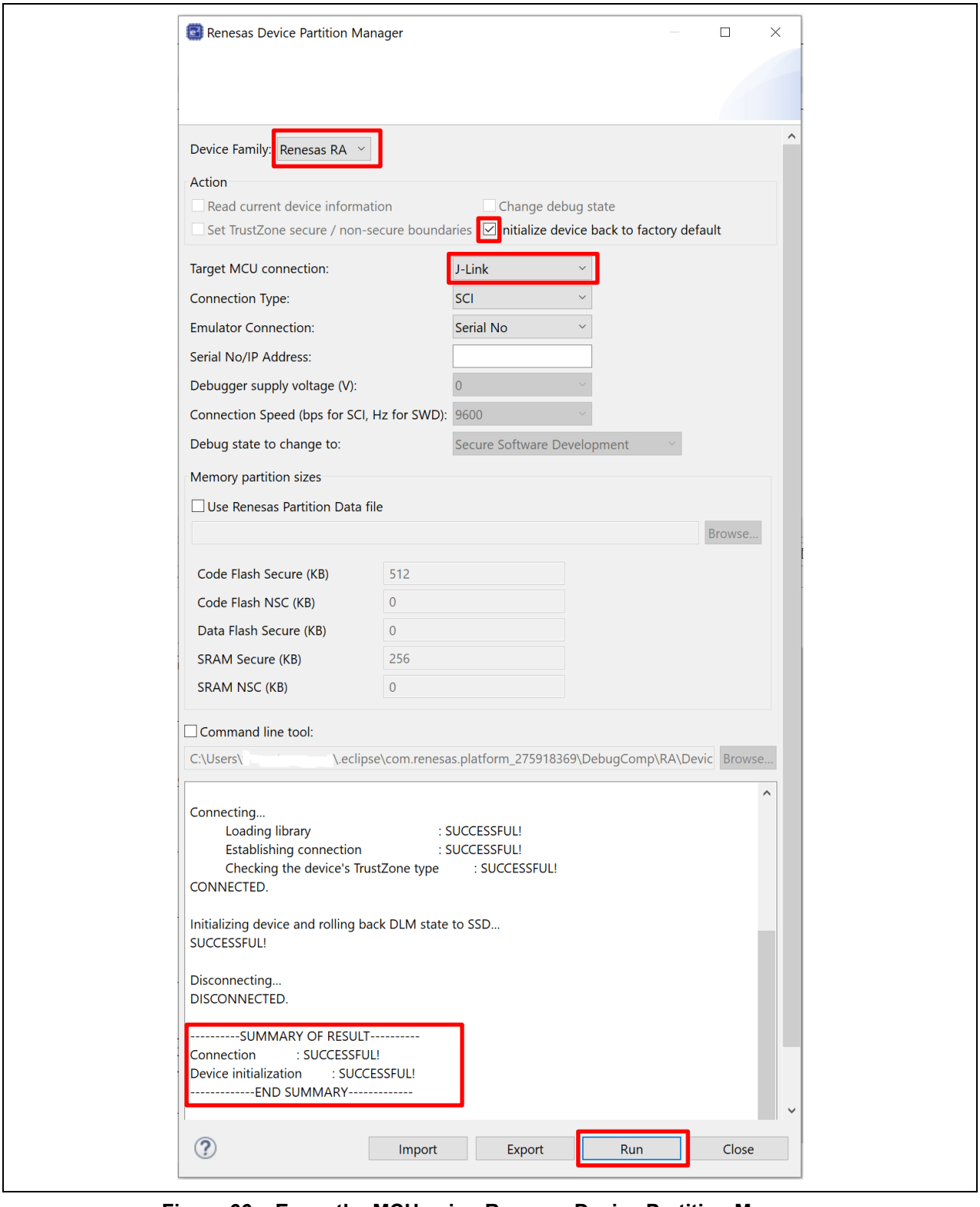

Figure 66. Erase the MCU using Renesas Device Partition Manager

6.4 Start the Debug Session

Follow the steps below to start the debug session:

1. 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.

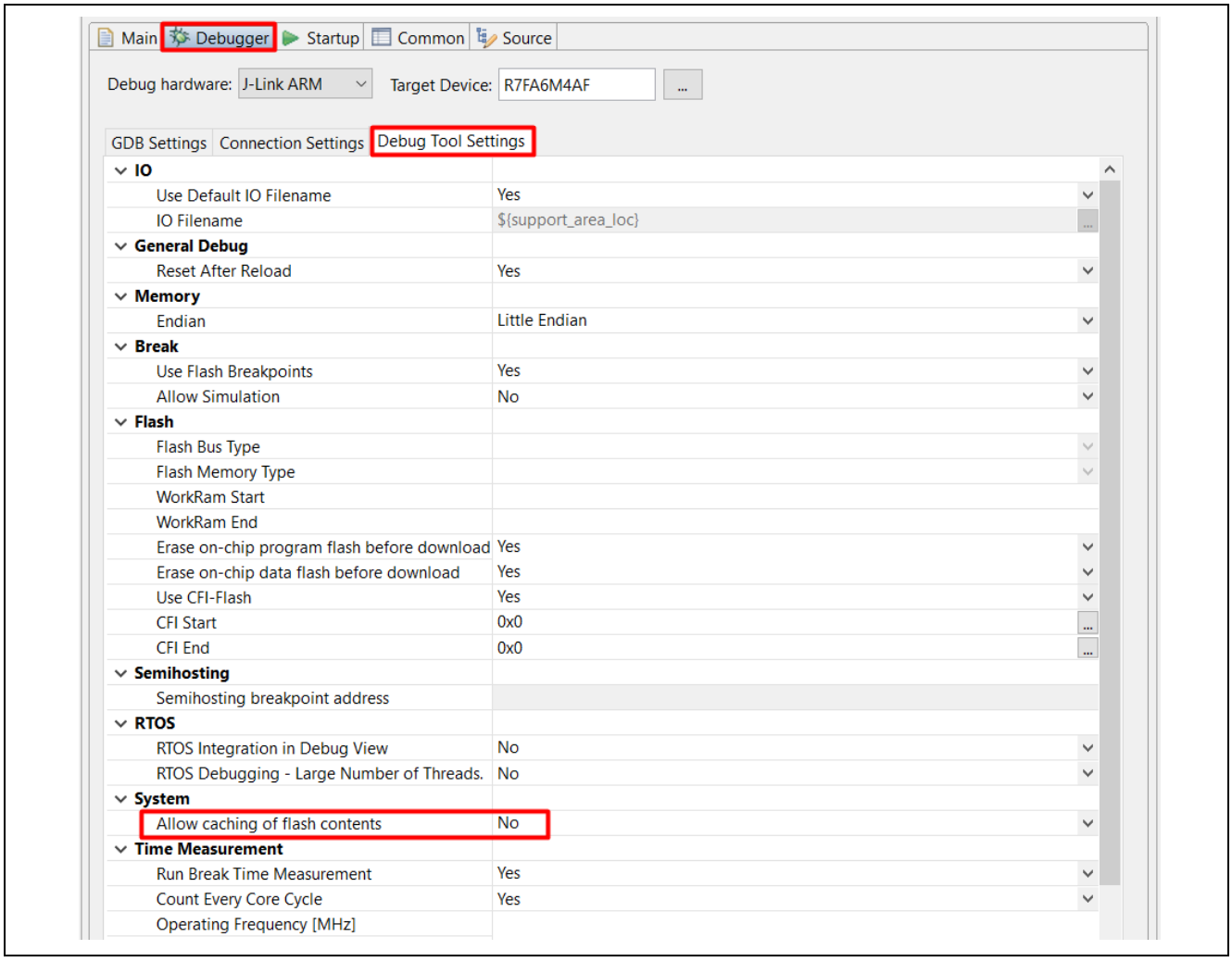

Figure 67. Disable Flash Content Caching

2. 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**.

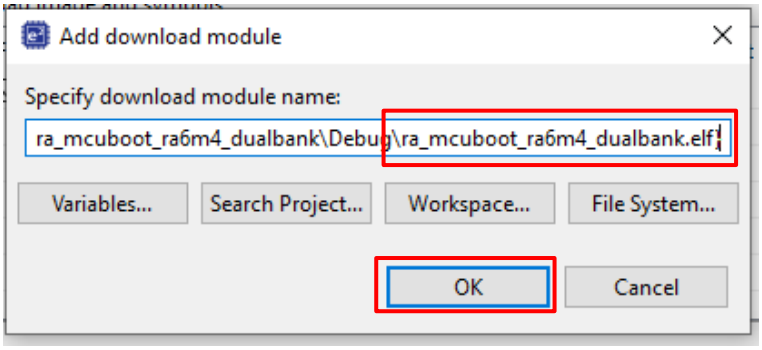

Figure 68. Add the Bootloader Project to Debug Configuration

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.

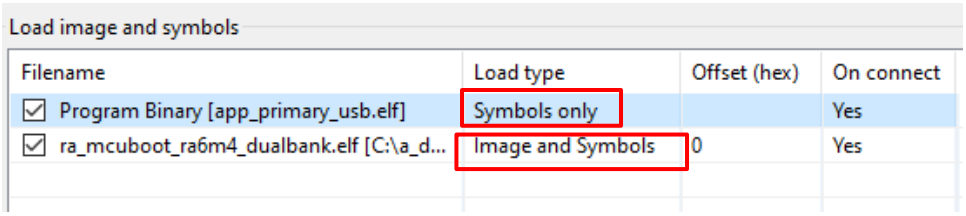

Figure 69. Select to load Symbols only for the Application Project

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.

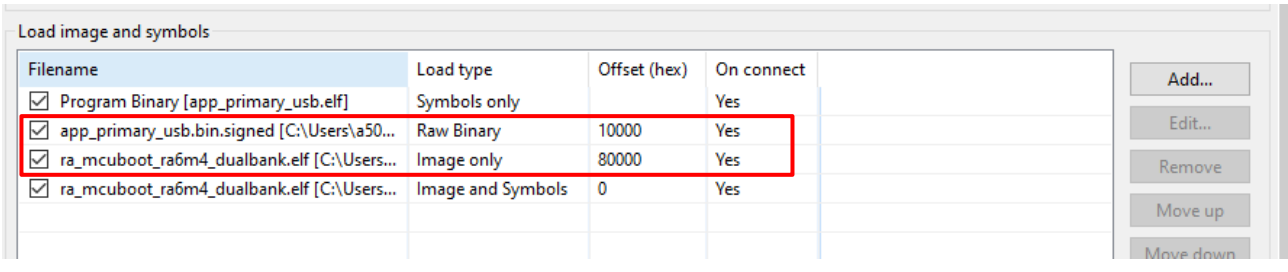

Figure 70. Add the Signed Primary Image and Upper Bank Bootloader

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

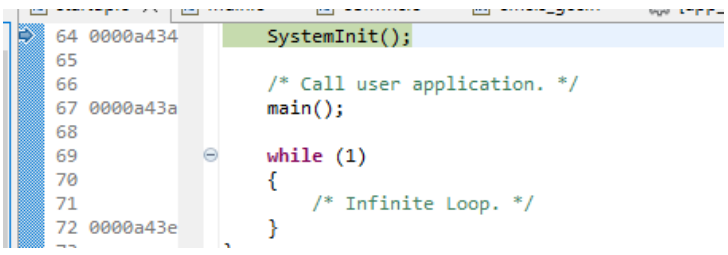

Figure 71. Start the Application Execution

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

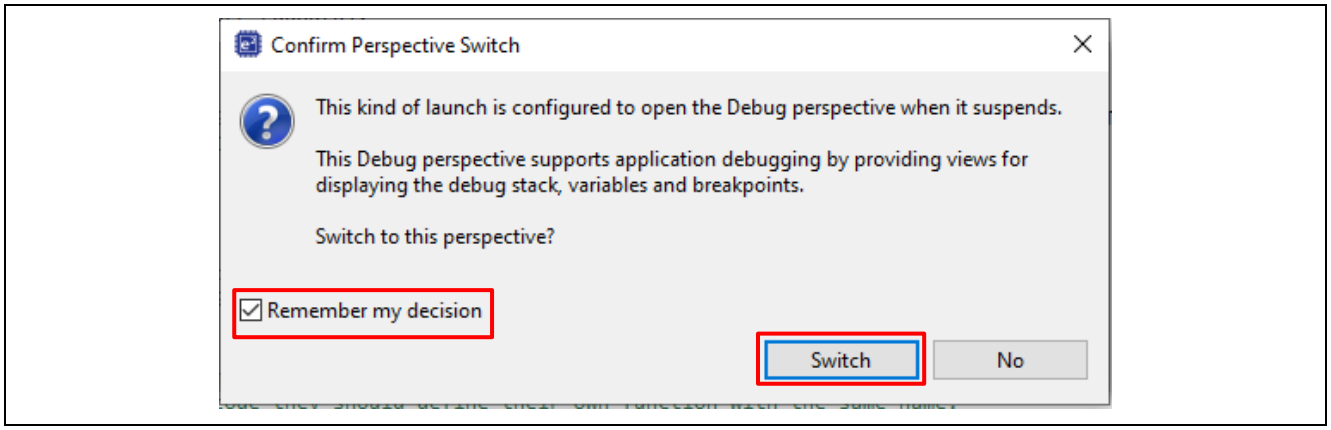

Figure 72. Switch the Perspective

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

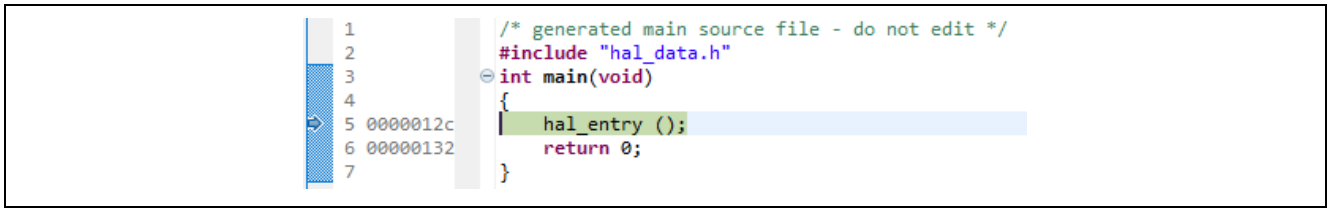

Figure 73. Start the Application Execution

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 Prepare a Secondary Image:

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

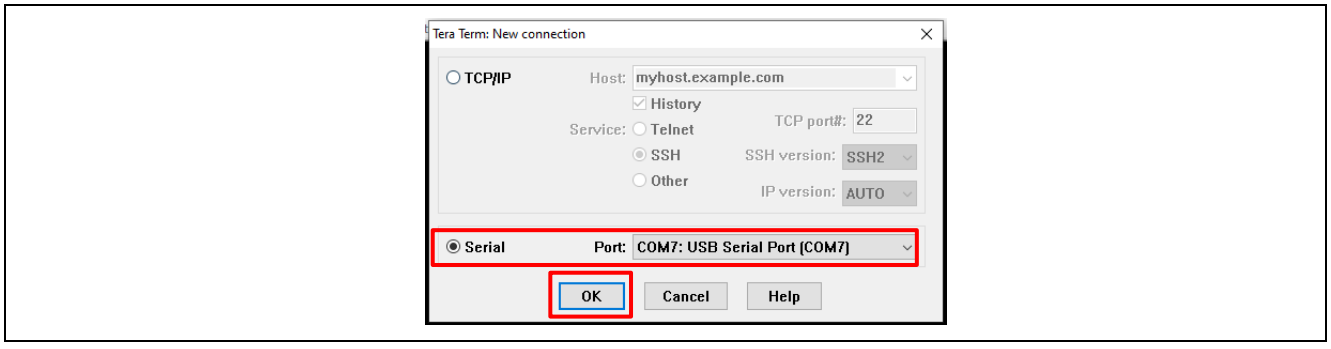

Figure 74. Open the COM Port

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

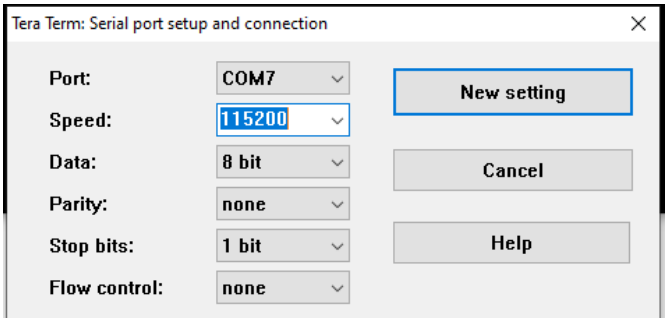

Figure 75. Configure the Baud Rate if using UART Interface

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

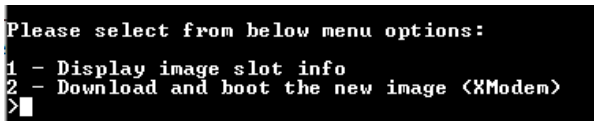

Figure 76. Tera Term Menu

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

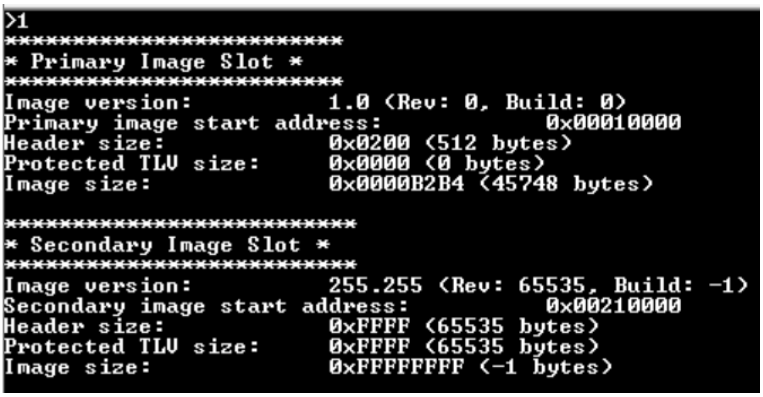

Figure 77. Print the Image Slot Information

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

Figure 78. Choose Option 2 to Download the New Image using XModem

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

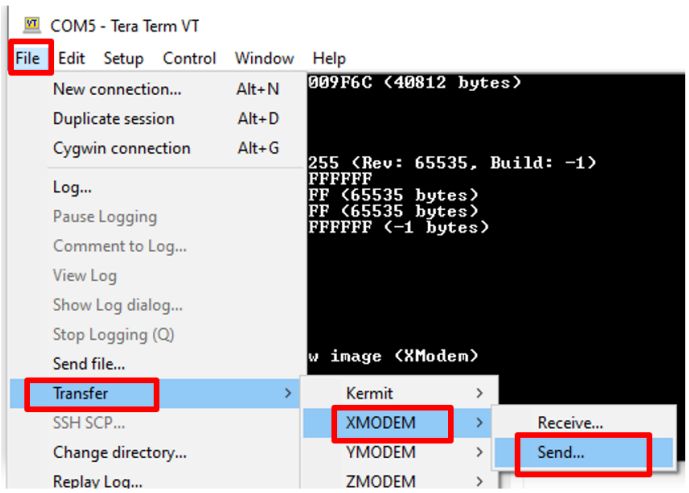

Figure 79. Start Transfer from Tera Term

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

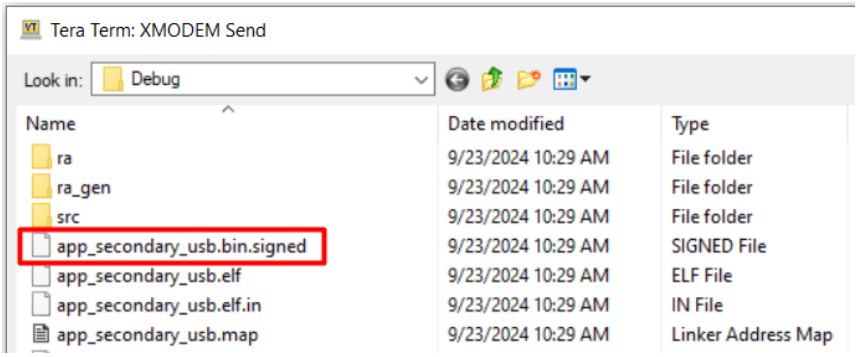

Figure 80. Choose the Signed Secondary Image

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

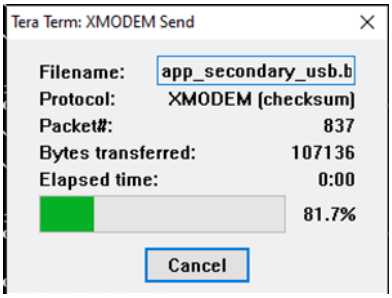

Figure 81. Download the New Image via XModem

6.6 Boot the New Application

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

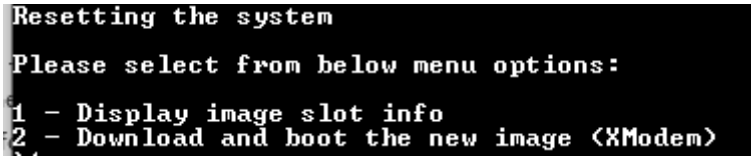

Figure 82. The New Image is Booted

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

```
>1
*************************
* Primary Image Slot *
*************************
Image version:          1.1 (Rev: 0, Build: 0)
Primary image start address:              0x00010000
Header size:            0x0200 (512 bytes)
Protected TLV size:     0x0000 (0 bytes)
Image size:             0x00009F74 (40820 bytes)

*************************
* Secondary Image Slot *
*************************
Image version:          1.0 (Rev: 0, Build: 0)
Secondary image start address:            0x00210000
Header size:            0x0200 (512 bytes)
Protected TLV size:     0x0000 (0 bytes)
Image size:             0x0000B2B4 (45748 bytes)
```

Figure 83. The Slot Layout After New Image is Booted

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 Root of Trust of the system. It should be protected from alteration by the application. Based on Figure 36. Compile the Bootloader ra_mcuboot_ra6m4_dualbank; the bootloader is located in the first 64-KB region. Based on Figure 3. RA6M4 and RA6E1 Code Flash Block Structure, 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 Erase the MCU.

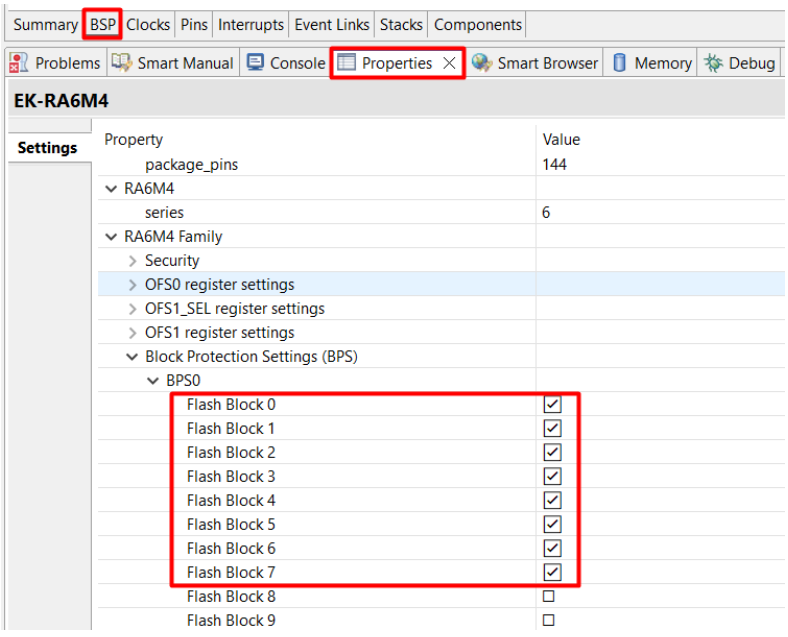

Figure 84. Temporary Protection of the Lower Bank Bootloader Area

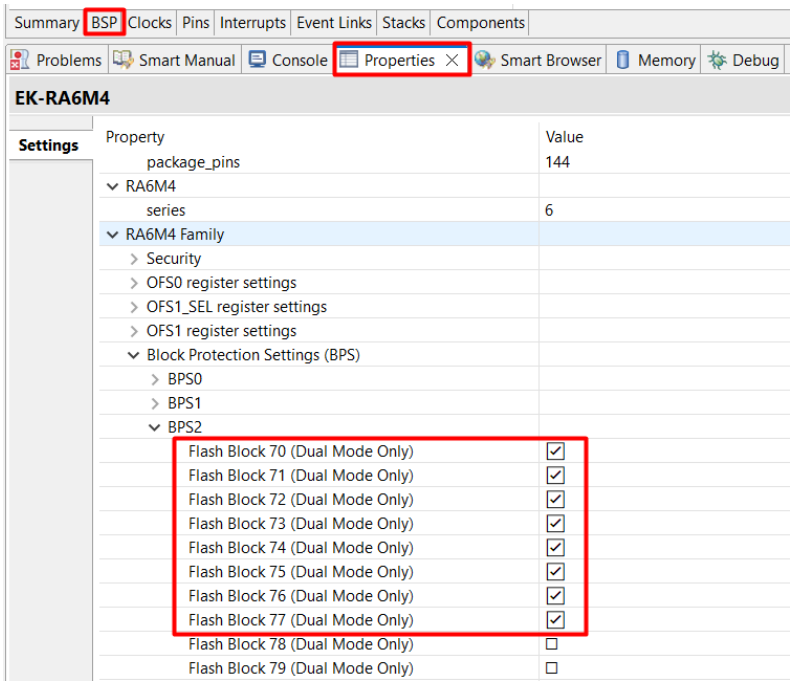

Figure 85. Temporary Protection of the Upper Bank Bootloader Area

Users can set up these blocks to be permanently protected in the `ra_mcuboot_ra6m4_dualbank` 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 described in section 6.3 Erase the MCU will not be able to erase these blocks.

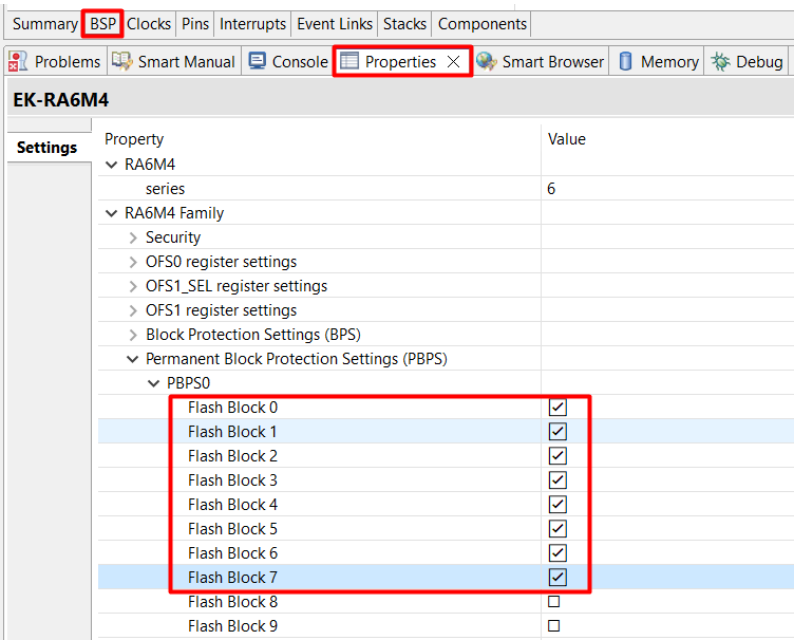

Figure 86. Permanent Protection of the Lower Bank Bootloader Area

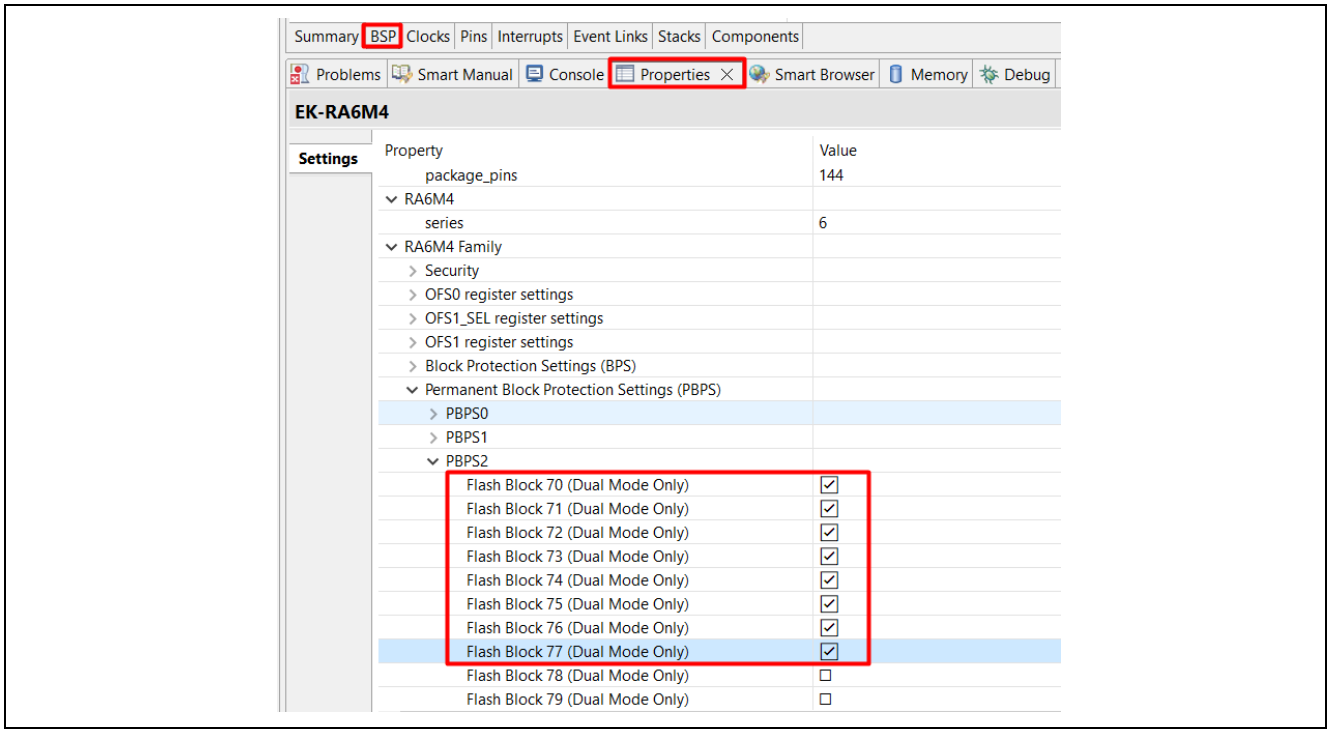

Figure 87. Permanent Protection of the Upper Bank Bootloader Area

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 have 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. Load combined.srec file using J-Flash Lite and Figure 91. Selecting combined.srec file and execute the command.

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

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

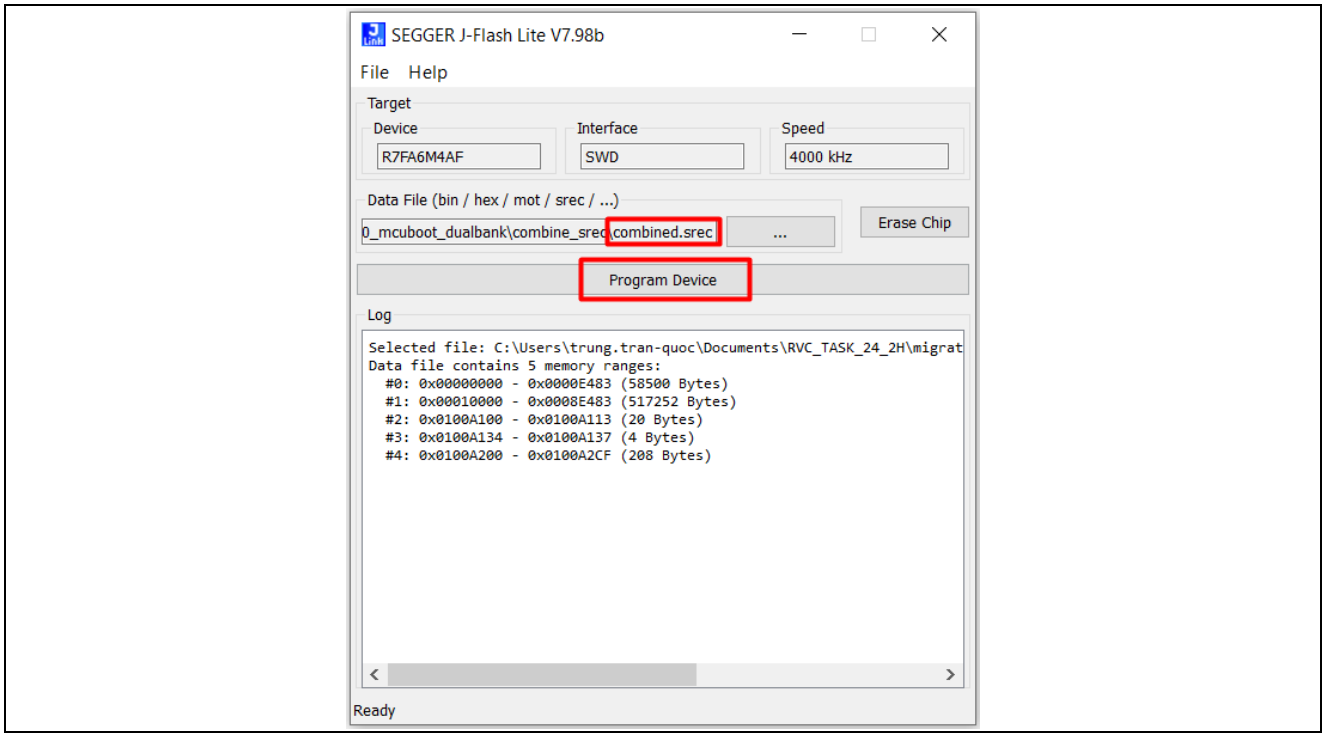

Figure 88. Load combined.srec file using J-Flash Lite

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

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

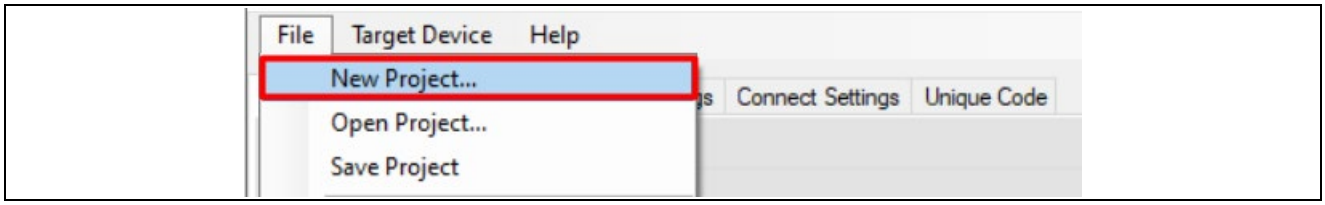

Figure 89. Create a New RFP Project

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

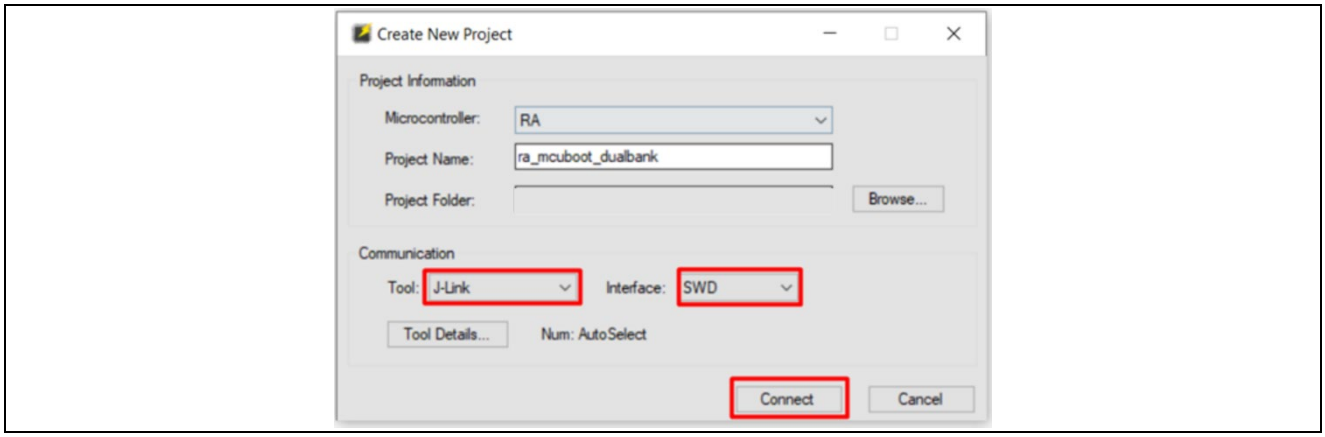

Figure 90. Configure the New Project

Select a program file and execute the command.

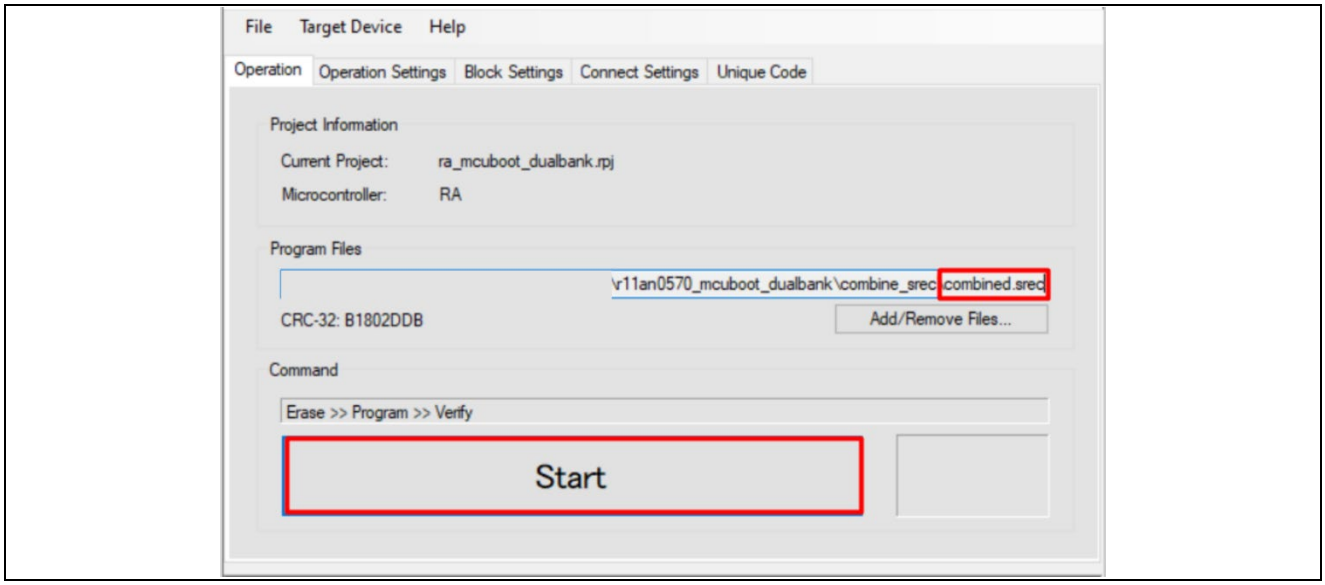

Figure 91. Selecting combined.srec file and execute the command

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

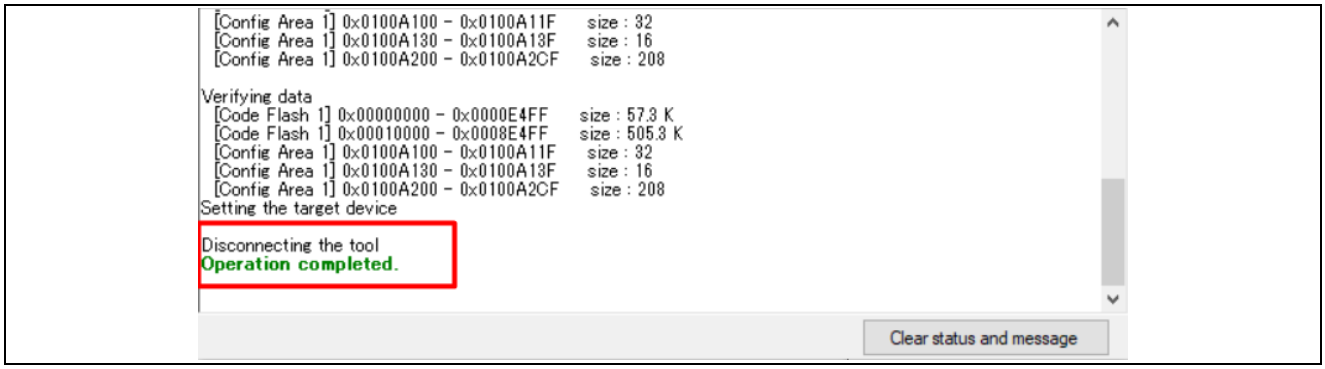

Figure 92. Download combined.srec file succeeded

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-dual-bank-flash-mcuboot\example_projects_with_bootloader`:

1. Follow the instructions in section 6.2 Set Up the Hardware 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 instructions in section 6.3 Erase the MCU.
13. Debug the application from project `app_primary_usb` in the e^2 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 Boot the New Application. This will take about 30 seconds.
18. The system will be reset automatically after the download.
19. Blue and green LEDs should be blinking.
20. Enter menu item 1 to confirm 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-dual-bank-flash-mcuboot\example_projects_with_bootloader`:

1. Follow the instructions in section 6.2 Set Up the Hardware 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 Erase the MCU.
13. Debug the application from project `app_primary_uart` in the e^2 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 Boot the New Application. This will take about 50 seconds.
18. The system will reset automatically after the download.
19. Blue and green LEDs should be blinking.
20. Enter menu item 1 to confirm 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	renesas.com/ra/ek-ra6m4
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.21.22	-	First release document
1.10	Nov.11.22	-	Updated Operation Flow based on e²studio 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 migrate to FSP v5.0.0
1.30	May.17.24	-	Added bat files.
1.40	Oct.01.24	-	Update to FSP v5.5.0
1.41	Mar.13.25	-	Correct description about code flash programming unit and block protect setting

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.