Renesas RA Family

RA4 Secure Bootloader Using MCUboot and Internal Code Flash

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 update. MCUboot is independent of 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 in 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 MCUboot Module with Renesas RA FSP v3.0.0 or later.

This application note guides you through application project creation using the MCUboot Module on Renesas EK-RA4M3 kits for the internal flash usage using FSP v5.2.0. Example projects for the use case of designing with TrustZone® for multi-image support are provided for EK-RA4M3 internal flash. Example projects for the use case of designing with single-image support are provided for EK-RA4M3 internal flash. The MCUboot Module is supported across the entire RA MCU Family. The example projects included can be easily ported to other RA4 Cortex-M33 MCU series.

Required Resources

Development tools and software

- e² studio IDE v2024-01 or later.
- Renesas Flexible Software Package (FSP) v5.2.0.
- SEGGER J-link® USB driver.

Note: The above three software components 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/)

Hardware

- EK-RA4M3 Evaluation Kit for RA4M3 MCU Group (http://www.renesas.com/ra/ek-ra4m3).
- Workstation running Windows® 10 and Tera Term console or similar application.
- One USB device cable (type-A male to micro-B male).

Prerequisites and Intended Audience

This application note assumes that you have some experience with the Renesas e² studio IDE and Arm® TrustZone-based development models with e² studio. You also need to understand the device lifecycle management of Renesas RA TrustZone-based MCU groups. This knowledge can be acquired by reading the HW User's Manual section "Security Features" and Renesas Application Project R11AN0469. In addition, you should read the entire MCUboot Port section of the FSP User's Manual prior to moving forward with this application project. This application project also assumes that you have some knowledge of cryptography.

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

Contents

1. Overview of MCUboot

1.1 History of MCUboot

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.

1.2 MCUboot Functionalities Overview

MCUboot handles the firmware authenticity check after startup and the firmware switch stage 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.

1.2.1 Validate Application before Booting and Updating

For applications using MCUboot, the MCU memory is separated into MCUboot, Primary App, Secondary App, and the Scratch Area. Figure 1 is an example of the single-image MCUboot memory map. For more information on the MCUboot memory layout, refer to the Flash Map section of the MCUboot website.

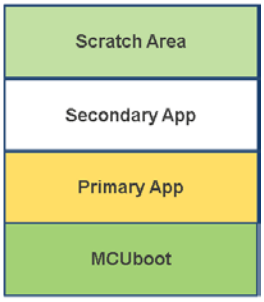

Figure 1. Single Image MCUboot Memory Flash Map

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

The bootloader is started when the CPU is released from reset. For TrustZone-based MCUs, MCUboot is designed to run in secure mode with all access privileges available to it. 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 App image.
2. Upon successful authentication, the bootloader switches to the new image based on the update method selected. Available update methods are introduced in section 1.2.2.
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 use and understand these operations.

1.2.2 Applications Update Strategies

The following are the update strategies supported by MCUboot. The analysis of pros and cons is based on the MCUboot functionality, but not the FSP MCUboot Module functionality. In addition, this application note is not intended to provide all details on the MCUboot application update strategies. We recommend acquiring more details on these update strategies by referring to the MCUboot design page:

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

- **Overwrite**
  In the Overwrite update mode, the active firmware image is always executed from the Primary slot, and the Secondary slot is a staging area for new images. Before the new firmware image is executed, the entire contents of the Primary slot are overwritten with the contents of the Secondary slot (the new firmware image).
  - Pros
    - Fail-safe and resistant to power-cut failures.
    - Less memory overhead, with a smaller MCUboot trailer and no Scratch Area.
    - Encrypted image support is available when using external flash.
  - Cons
    - Does not support pre-testing of the new image prior to overwrite.
    - Does not support automatic application fallback mechanism.

  Overwrite upgrade mode is supported by Renesas RA FSP v3.0.0 or later. External flash memory support is supported by FSP v3.5.0 or later. The overwrite update mode is demonstrated in sections 3.3 and 3.6.

- **Swap**
  In the Swap image upgrade mode, the active image is also stored in the Primary slot and is always started by the bootloader. If the bootloader finds a valid image in the Secondary slot that is marked for upgrade, then contents of the Primary slot and the Secondary slot are swapped. The new image then starts from the Primary slot. Upgrading an old image with a new one by swapping can be a two-step process. In this process, MCUboot performs a "test" swap of image data in flash and boots the new image. The new image can then update the contents of flash at runtime to mark itself "OK", and MCUboot will then still choose to run it during the next boot.
  - Pros
    - The bootloader can revert the swapping as a fallback mechanism to recover the previous working firmware version after a faulty update.
    - The application can perform a self-test to mark itself permanent.
    - This image upgrade mode is fail-safe and resistant to power-cut failures.
    - Encrypted image support is available when using external flash.
  - Cons
    - Need to allocate a Scratch Area.
    - Larger memory overhead, due to a larger image trailer and additional Scratch Area.
    - Larger number of write cycles in the Scratch Area, faster wearing out of Scratch sectors.

  Swap upgrade mode is supported by Renesas RA FSP v3.0.0 or later. Runtime image testing is supported by FSP v3.4.0 or later, excluding v3.5.0. External flash memory support is supported by FSP v3.5.0 or later. The swap update mode without test mode is demonstrated in section 3.4 and the swap update mode with test mode is demonstrated in section 3.7.

- **Direct execute-in-place (DXIP)**
  In the direct execute-in-place 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.
  - Pros
    - Faster boot time, as there is no overwrite or swap of application images needed.
    - Fail-safe and resistant to power-cut failures.
  - Cons
    - Added application-level complexity to determine which firmware image needs to be downloaded.
    - Encrypted image support is not available.

  Direct execute-in-place mode is enabled in FSP for the code flash linear mode as well as code flash dual bank mode. The DXIP update mode is demonstrated in section 3.5.

- **RAM loading firmware update**
  Like the direct-XIP mode, RAM loading firmware update mode selects the newest image by reading the image version numbers in the image headers. However, instead of executing it in place, the newest image is copied to RAM for execution. The load address (the location in RAM where the image is copied to) is stored in the image header. This upgrade method is not typically used in an MCU environment. Refer to the RAM Loading section in the MCUboot page for more information on this update strategy. This image update mode does not support encrypted images (see MCUboot documentation on encrypted image operation).
  RAM loading update mode is not supported by the Renesas RA FSP.

2. Architecting an Application with MCUboot Module using FSP

This section provides an overview of the FSP MCUboot Module, which integrates MCUboot as a module into the FSP. The available upgrade modes and memory architecture design are discussed. In addition, signing and mastering new images are discussed.

2.1 MCU Memory Configuration using MCUboot Module with FSP

For single-image projects, refer to Figure 1 from section 1.2.1 to see the default memory map layout. For applications with two separately updateable images, such as TrustZone applications where the Secure and Non-Secure images can be updated separately, the default memory map layout is shown in Figure 2.

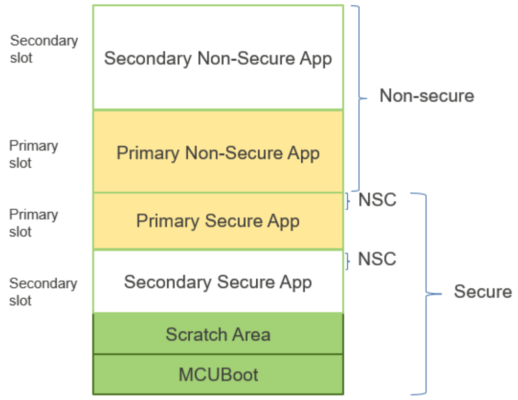

Figure 2. Two-Image MCUboot Module Memory Map (TrustZone)

2.2 Overview of FSP MCUboot Module

This section provides a high-level overview of the MCUboot Module in the FSP. Currently, the FSP supports four firmware update methods:

- **Overwrite Only**: The entire Primary slot is overwritten with the Secondary slot.
- **Overwrite Only Fast**: Only sizeof(secondary_image) is copied into Primary slot. Unused sectors are not copied.
- **Swap**: The entire Primary and Secondary slots are swapped. A Scratch region is required.
- **Direct XIP**: The new image is run directly from its flash partition.

We recommended reviewing MCUboot Port section of the FSP User's Manual to understand the Build Time Configurations for MCUboot. This section is not meant to cover all the configurable properties. Only some of the most frequently used configuration options are introduced.

2.2.1 General Configuration

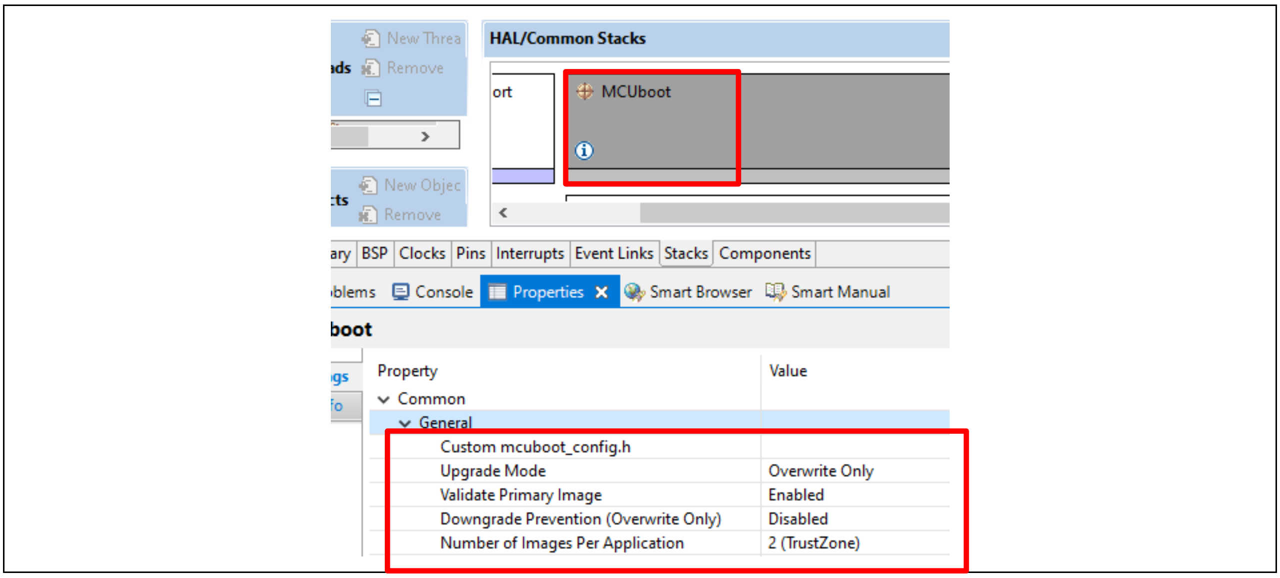

Figure 3. FSP MCUboot Module General Configuration Properties

General configuration properties include:

- **Custom mcuboot_config.h**: The default `mcuboot_config.h` file contains the MCUboot Module configuration that you selected from the RA configurator. You can create a custom version of this file to achieve additional bootloader functionalities available in MCUboot.
- **Upgrade Mode**: This property configures the application image update method selection explained at the beginning of section 2.2. The options are Overwrite Only, Overwrite Only Fast, Swap, and Direct XIP, as shown in Figure 4. Overwrite Only is the default setting.

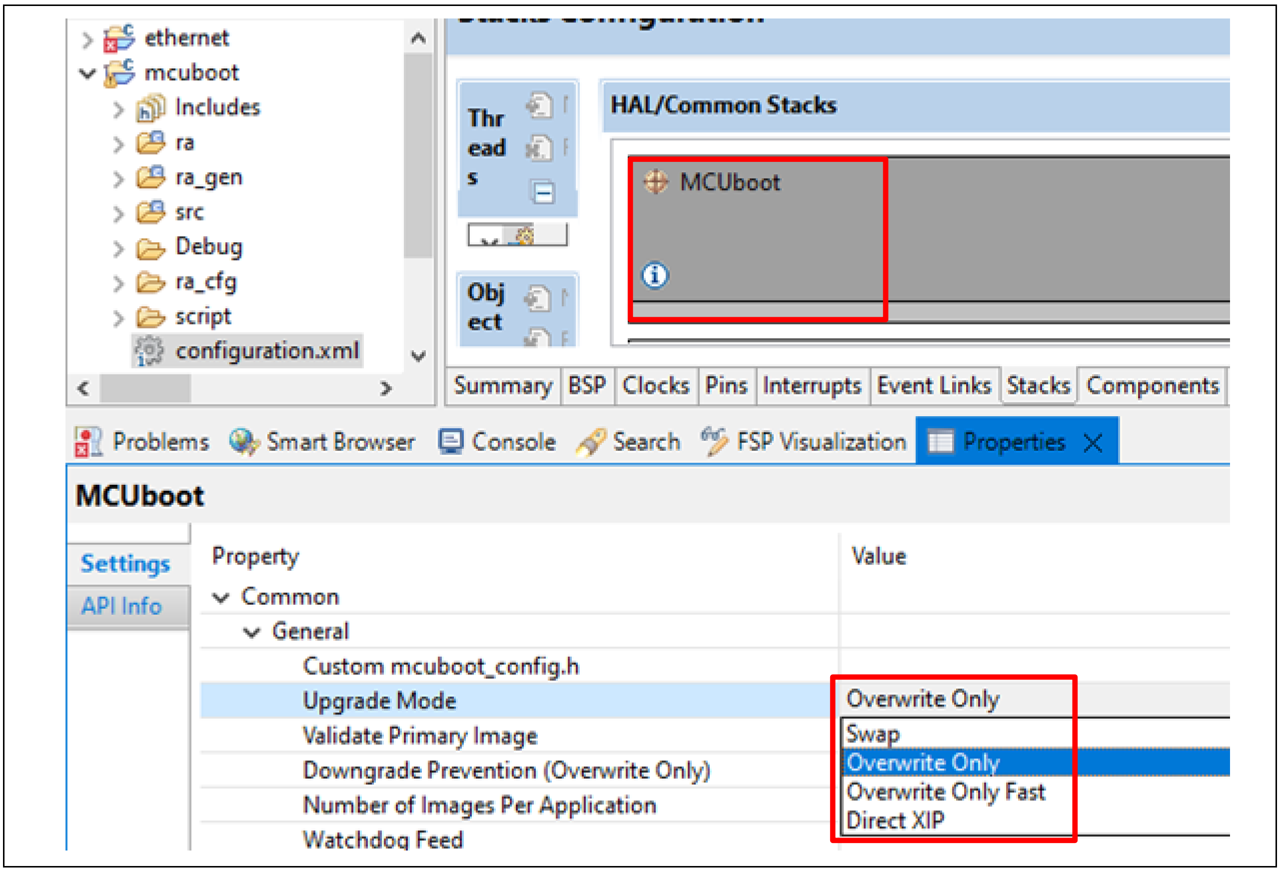

Figure 4. Application Image Update Mode

Figure 5 is a more detailed application image format that can be referenced to understand the various MCUboot property definitions.

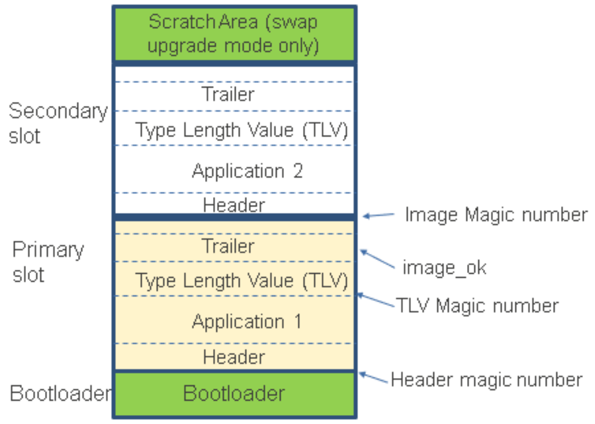

Figure 5. General Configuration for MCUboot Module

- **Validate Primary Image**:
  When Validate Primary Image is enabled, the bootloader performs a hash or signature verification, depending on the verification method chosen, in addition to the MCUboot sanity check based on the image header and TLV area magic numbers. The Header and TLV area magic numbers are always checked as part of the sanity checking prior to the integrity checking and the signature verification.
  When Validate Primary Image is disabled, only the integrity check based on hash is performed as well as the sanity check is performed. It is highly recommended to always enable this property if boot time is not a concern. Note that the image magic number is not part of the image validation; it is a reference value that can be used for sanity check during application upgrade debugging process. This image magic number is written to the flash after a successful image upgrade.
- **Downgrade Prevention (Overwrite Only):** This property applies to Overwrite upgrade mode only. When this property is enabled, new firmware with a lower version number will not overwrite the existing application.
- **Number of Images Per Application:** This property allows you to choose one image for Non-TrustZone-based applications and two images for TrustZone-based applications.

2.2.2 Application Image Signature Type Options

Application images using MCUboot must also be signed to work with MCUboot. At a minimum, this involves adding a hash and an MCUboot-specific constant value in the image trailer.

Figure 6 shows the signature types available for the application image signing methods supported by the MCUboot module. For memory restricted devices, you can choose **None** for **Signature Type**, which will reduce the bootloader size. For example, the example bootloader for the Overwrite update mode uses a flash area of 64 KB when using ECDSA P-256 signature type, but when signature support is not used, the bootloader reduces to about 19 KB.

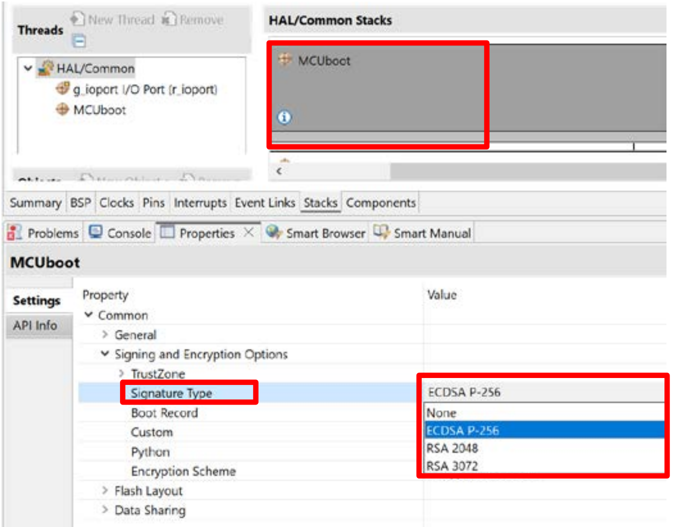

Figure 6. Application Image Signature Type for FSP MCUboot Module

2.2.3 Signing Options

Figure 7 shows the default **Custom** signing option configuration provided by FSP.

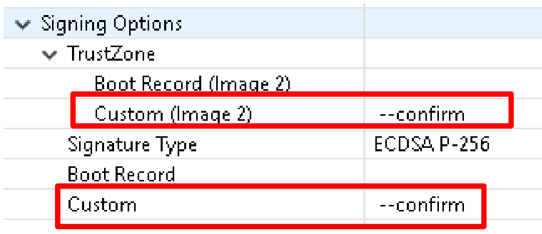

Figure 7. FSP Default Signing Option

By default, FSP sets **--confirm** for the **Custom** property for both Image 1 and Image 2 when TrustZone is used. For TrustZone-based applications, the Secure Image (Image 1) and Non-Secure Image (Image 2) can have different configurations such that there is different update policy for the Secure and Non-Secure Images. Some commonly used signing options are:

- Option **--pad**:
  This option places a trailer on the image that indicates that the image should be considered for an upgrade. Writing this image in the Secondary slot causes the bootloader to upgrade to it. When Swap mode is selected, this option generates a signing command such that the Secondary image will first be swapped with the Primary application image. On the next reset, the Primary application previously used will be swapped back and rebooted.
- Option **--confirm**:
  When Swap mode is selected, this option generates a signing command such that the Secondary image will first be swapped with the Primary application. At the next reset, there will be no swap between the Primary and Secondary application and the Secondary application will be booted. Confirm is the default Force Upgrade configuration.
- No input:
  If no option is put in this property, application images signed with the signing command generated from this setting will not be updated.

When Overwrite mode is selected, the **--pad** or **--confirm** option generates signing commands such that the overwrite will occur and the Secondary application will overwrite the Primary application.

The image signing tool `Imgtool.py` is included with MCUboot. It is integrated as a post-build tool in e[2] studio to sign the application image. For detailed information about using this tool with e[2] studio, refer to the application image signing information in section 5.2. For more information on the possible options available for this property setting, refer to the description in the imagetool.py md file and visit the MCUboot documentation page https://docs.mcuboot.com/imgtool.html.

2.2.4 MCU Memory Configuration

Figure 8 shows the default memory configuration options provided by the FSP configurator for RA4 MCU groups.

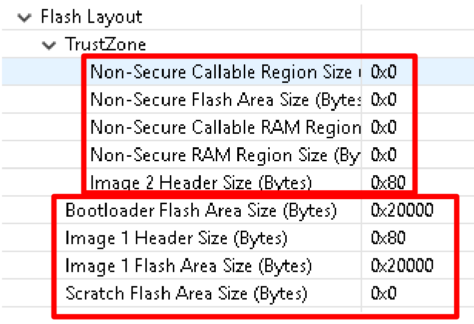

Figure 8. MCU Memory Configuration Default Settings

For both single-image and two-image configurations, the following four properties need to be defined:

- **Bootloader Flash Area**: Size of the flash area allocated for the bootloader.
- **Image 1 Header Size**: Size of the flash area allocated for the application header for single image configuration or the secure application image header size in the case of a TrustZone-based application. This property should be set to 0x200 for RA4 and RA6 MCUs and 0x100 for RA2 MCUs.
- **Image 1 Flash Area Size**: Size of the flash area allocated for the application image for single image configuration or the secure application image in the case of a TrustZone-based application.
- **Scratch Flash Area Size**: This property is only needed for Swap mode. The Scratch area must be large enough to store the largest sector that is going to be swapped. For RA4M3, the Scratch area is set up to be 32KB (0x8000).

The properties under **TrustZone** are for TrustZone-based applications:

- **Non-Secure Callback Region Size (Bytes)**: This area is used for the TrustZone Non-Secure Callable area plus the MCUboot trailer. This property needs to be set to a multiple of 1024 bytes. Each Non-Secure Callable function takes 8 bytes of flash area. The non-secure callback function usage can be identified by referring to the section .sgstub in the secure application map file. For Swap mode, the MCUboot trailer size is calculated as 128*(5+(3*BOOT_MAX_IMG_SECTORS). BOOT_MAX_IMG_SECTORS is the number of flash sectors in either the secure or the non-secure image, whichever is larger.

  For Overwrite mode, the image trailer is less than 256 bytes, for a typical application with limited number of Non-Secure Callable APIs, it is recommended to set the Non-Secure Callable Region Size to 0x400.
- **Non-Secure Flash Area Size:** Size of the Non-Secure flash region. You can compile the non-secure application to get the size of the image and set this value accordingly. This value must be a multiple of the flash block size.
- **Non-Secure Callable RAM Region:** This property is the size of the Non-Secure Callable RAM region of the Secure image. This property needs to be set to a multiple of 1024 bytes.
- **Non-Secure RAM Region Size**: Size of the Non-Secure RAM region. This property must be an integer multiple of 8192 bytes.
- **Image 2 Header Size**: The non-secure application header size. This property should be set up by following the same rule as explained for the **Image 1 Header Size**.

2.3 Designing Bootloader and the Initial Primary Application Overview

A bootloader is typically designed with the initial Primary application. The following are the general guidelines for 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. The bootloader maintains a memory map of all the different images shown in Figure 1 and Figure 2.
- 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 adjustment of the memory allocated definition in the bootloader project.
- Test the bootloader and the initial Primary application.

Most of these design aspects are addressed in the walk-through in section 4.

2.4 General Guidelines using the MCUboot Module Across RA Family MCUs

The MCUboot Module is supported on all RA Family MCUs.

The cryptographic support is provided via MbedTLS Crypto only module and Tiny Crypt module. Both crypto modules are supported on all RA MCUs either through software or MCU hardware. The MbedTLS Crypto Only module is supported by the MCU hardware if the corresponding algorithms are supported by the hardware crypto engine, otherwise MbedTLS software stack will be used. The MbedTLS offers more crypto algorithms, is generally faster and has a larger memory footprint. On the other hand, the TinyCrypt module offers a smaller number of algorithms, is slower but has a much smaller memory footprint. Users can consider disable the image validation of the primary image prior to execution at MCU reset to reduce the boot time. See explanations on the validation property in section 2.2.1.

Table 1 is the typical cryptographic selection recommendations when using MCUboot with RA MCUs. If memory footprint is a priority, users can choose the TinyCrypt module over the MbedTLS Crypto Only module for some of these use cases. To improve the verification speed and reduce boot time when using Tiny Crypt, user can consider disable image validation to improve verification and boot time performance.

Table 1. Typical Cryptographic Selection Recommendations for RA MCUs

Crypto stack	RA2 No Encryption	RA2 with Encryption	RA4E1, RA6E1, RA4W1, RA4M1, RA6T2 No Encryption	RA4E1, RA6E1, RA4W1, RA4M1, RA6T2 with Encryption	RA6M1/M2/M3, RA6T1, RA4M2/M3, RA6M4/M5 with or without Encryption	RA8M1, RA8D1, RA8T1 with or without Encryption
MbedTLS (Crypto Only) HW					x	x
TinyCrypt (HW AES)		x		x		
TinyCrypt (SW Only)	x		x			

For the Renesas RA Cortex-M33 MCU series internal flash usage, refer to the RA4M3 example projects demonstrated in this application project.

For MCUboot with encryption support, refer to the RA4M3 example projects demonstrated in the Booting Encrypted Image using MCUboot and QSPI application project R11AN0868.

For the Renesas RA Cortex-M4 MCUs RA6 MCU series internal flash usage, refer to the RA6M3 example projects demonstrated in the application project R11AN0497.

For the Renesas RA Cortex-M23 MCU series, refer to the RA2E1 example projects demonstrated in the application project R11AN0516.

For the Renesas RA6 MCUboot with code flash dualbank support, refer to the RA6M4 example projects demonstrated in the application project R11AN0570.

2.5 Customize the Bootloader

The following aspects need to be considered when customizing the bootloader in a product design:

- Customized method to download the application.
- Use various optimization methods to reduce bootloader and application image size. For example, compile the bootloader by Optimize size.

2.6 Production Support

2.6.1 Key Provisioning

By default, the public key is embedded in the bootloader code and its hash is added to the image manifest as a `KEYHASH TLV` entry. See section 4.1.3 for more details about the public key and private key that are used for testing purpose. For production support, follow the example shown in `key.c` to add the public key. In addition, you must update the private key for application image signing. Refer to and Figure 62 for the private key selection in the signing command.

As an alternative, the bootloader can be made independent of the included test keys by setting the `MCUBOOT_HW_KEY` option. In this case, the hash of the public key must be provisioned to the target device and MCUboot must be able to retrieve the key-hash from there. For this reason, the target must provide a definition for the `boot_retrieve_public_key_hash()` function that is declared in `boot/bootutil/include/bootutil/sign_key.h`. The full option for the `-public-key-format` imgtool argument is also required to add the whole public key (`PUBKEY TLV`) to the image manifest instead of its hash (`KEYHASH TLV`).

During boot, the public key is validated before it is used for signature verification. MCUboot calculates the hash of the public key from the TLV area and compares it with the key-hash that was retrieved from the device. This way, MCUboot is independent from the public key(s). The key(s) can be provisioned any time and by different parties.

2.6.2 Make the Bootloader Immutable for Enhanced Security

For a Cortex-M33 MCU, refer to section 6.1 to make the bootloader immutable.

2.6.3 Advance the Device Lifecycle States Prior to the Deploying the Product to the Field

For a Cortex-M33 MCU, refer to section 6.2 for the device lifecycle management of the MCU.

3. Running the Example Projects

This section provides a walk-through of running the included example projects. To recreate the bootloader example projects demonstrated in this section, refer to section 4.1 for the Cortex-M33 implementation.

The bootloader projects introduced have similar functionality except that the memory map definition and application image update mode are different.

Unzip `example_projects_with_bootloader.zip` and you will see that there are three folders. Each folder contains example projects for the specific MCU which include bootloader project and example application projects.

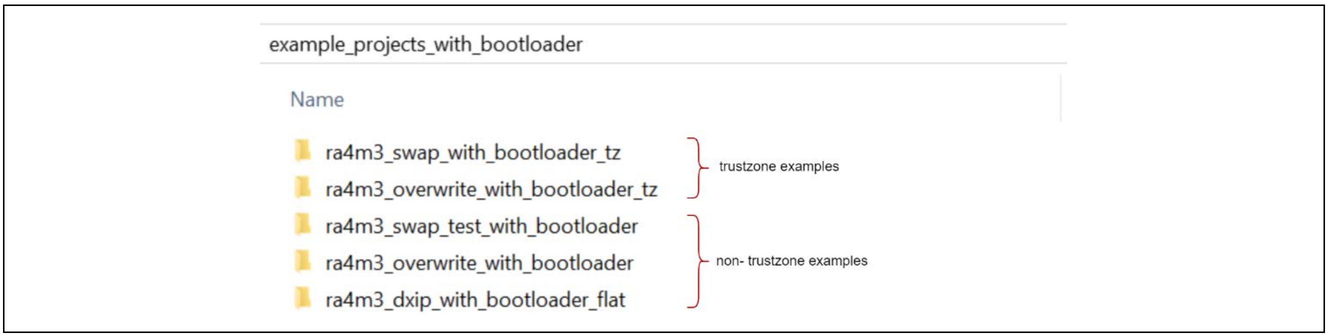

Figure 9. Example Projects with Bootloader Support

Set up the Python development environment by following section 3.3 step 3.2. Note that this step only needs to be performed once.

3.1 Set Up the Hardware

3.1.1 Set up EK-RA4M3

- Jumper setting: J12 is set to pins 2-3 and J15 is closed.
- Connect J10 using a USB micro to B cable from EK-RA4M3 to the development PC to provide power and debug connection using the on-board debugger.

Once the EK-RA4M3 is powered up, initialize the MCU prior to exercising the bootloader project.

Erase the entire MCU flash and ensure the MCU is in Secure Software Development Device Lifecycle State. This can be achieved using the Renesas Device Partition Manager.

1. Power cycle the board, launch e[2] studio, and open the Renesas Device Partition Manager.

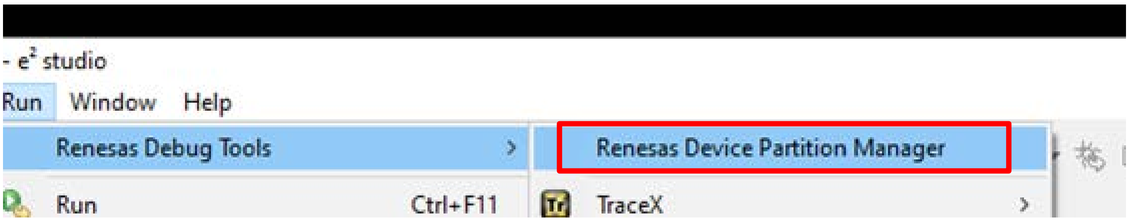

Figure 10. Open Renesas Device Partition Manager

2. Select **Read current device information**.
   If the DLM state is SSD, NSECSD, or DPL, proceed to step 3. Otherwise, you must switch to a different kit to continue the rest of the operation. Below is an example of the readout from an RA4M3 MCU that is in the SSD state.

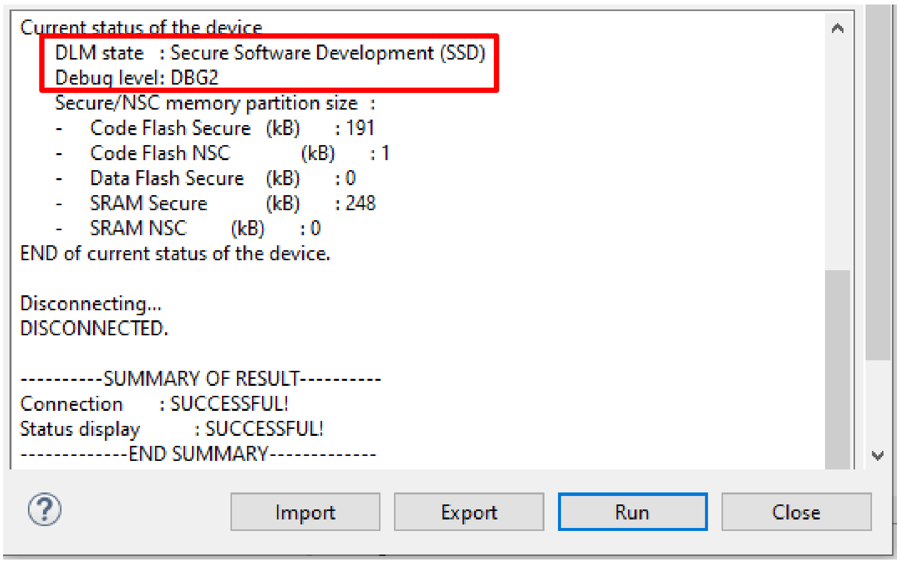

Figure 11. Read the Device Lifecycle States

3. Select **Initialize device back to factory default**, choose **J-Link** as the connection method, and click **Run**.

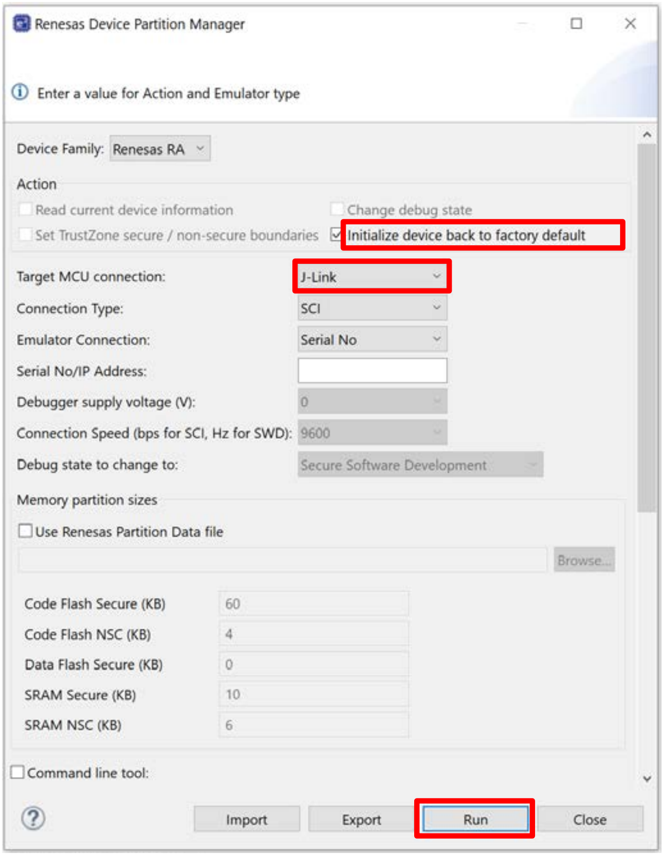

Figure 12. Initialize RA4M3 using Renesas Device Partition Manager

The entire flash will be erased if there are not permanently locked down sections. In addition, if the device is in the NSECSD or DPL state, the RA4M3 will be initialized to the SSD state.

4. Power cycle the EK-RA4M3 after successfully initializing the device to the SSD state by disconnecting the USB cable and reconnecting it to the development PC.

3.2 Configure the Python Signing Environment

If this is **NOT** the first time you have used the Python script signing tool on your computer, you can skip this section. Note that section 3.3 to section 3.7 can be evaluated independently; it is not necessary to follow a particular sequence.

Download and Install Python v3.9 or later from https://www.python.org/downloads/.

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:

- From the included example project sets (refer to Figure 9), choose the set of projects you would like to exercise first.
- Import that set of projects into a workspace. In this example, we assume you have chosen to import the projects under folder:

  `\example_projects_with_bootloader\ra4m3_overwrite_with_bootloader_tz.`

- Navigate to folder `\MCUboot` in the bootloader project included, eg. `ra_mcuboot_ra4m3>ra>mcu-tools>MCUboot`, right click, and select **Command Prompt**. This opens a command window with the path set to the `\mcu-tools\MCUboot` folder.

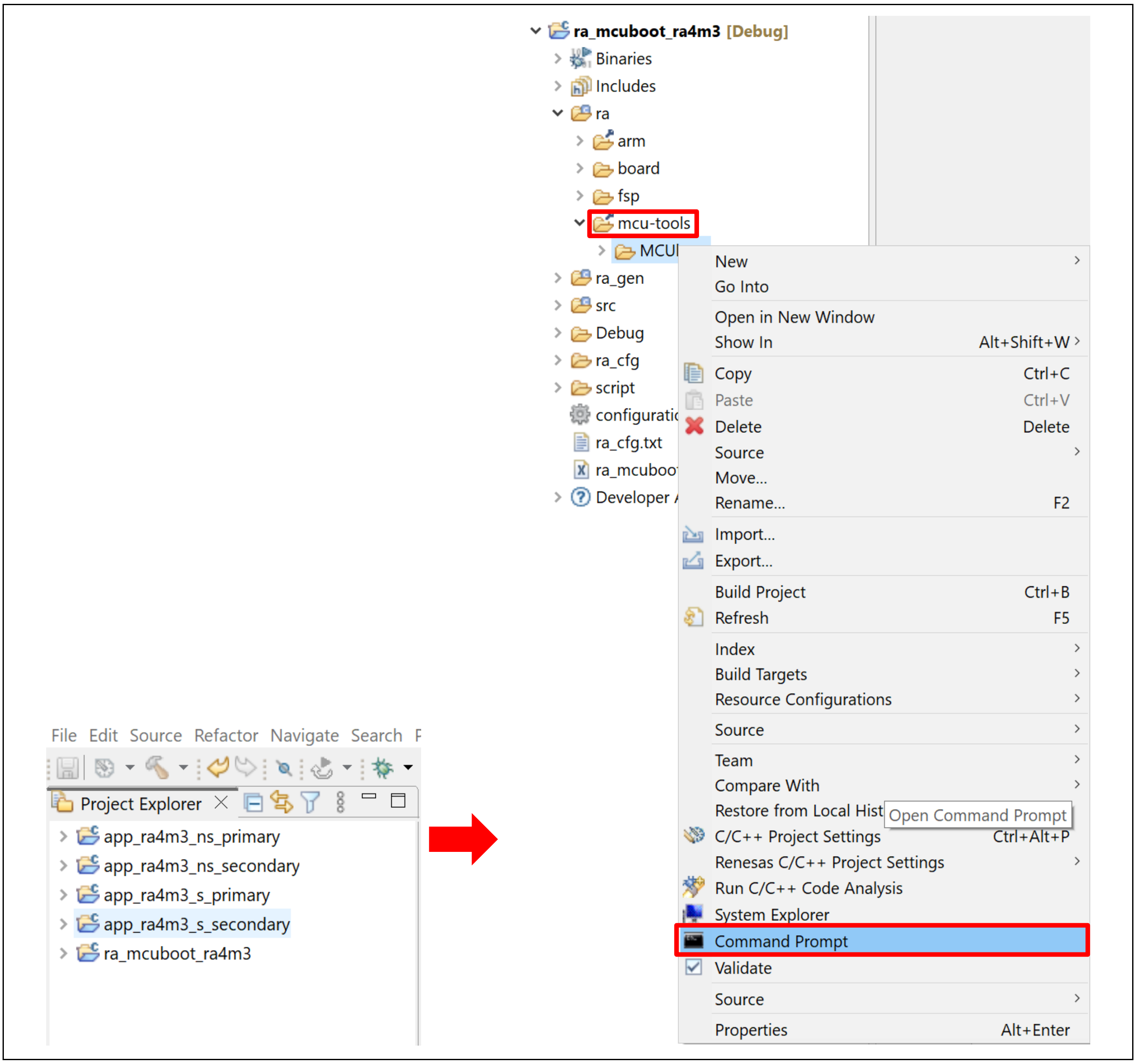

Figure 13. Open the Command Prompt

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

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

- Note that if you have multiple Python versions installed, make sure to check that the Python version is version 3.9.0 or later.
- 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. Make sure this step runs successfully prior to moving to the following sections.

3.3 Running the EK-RA4M3 Overwrite Update Mode Example with TrustZone

Follow the steps below to run the example projects for EK-RA4M3 using the MCUboot Module Overwrite Only Update mode with TrustZone.

3.3.1 Initialize the RA4M3 MCU

Follow section 3.1.1 to initialize the RA4M3 MCU.

3.3.2 Import the Projects under `\ra4m3_overwrite_with_bootloader_tz`

New users should refer to the FSP User's Manual section on Importing Projects into the IDE for guidelines. Ensure the Python signing environment is set up referencing section 3.2.

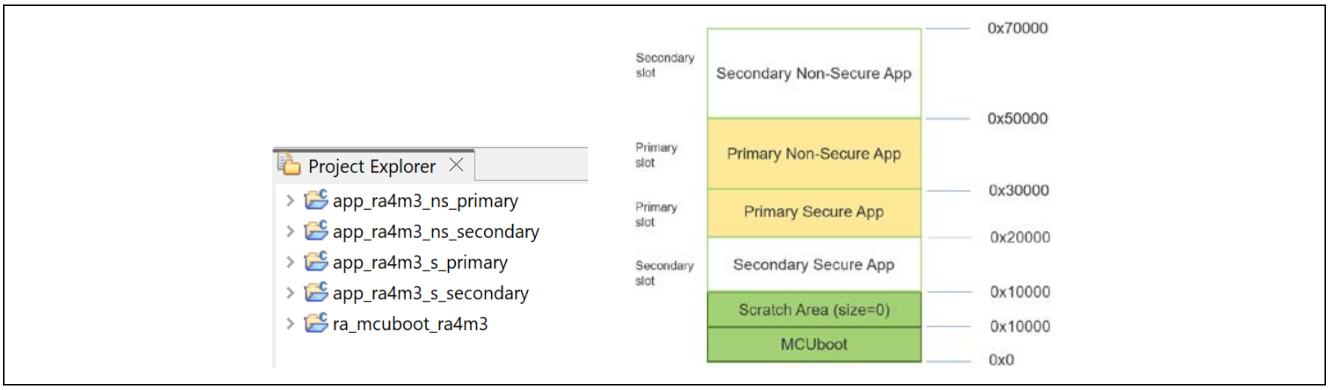

Figure 14. Example Projects for RA4M3 Overwrite Update Mode with TrustZone

- **`ra_mcuboot_ra4m3`**: The bootloader project configured with Overwrite update mode.
- **`app_ra4m3_s_primary`**: The Primary Secure application project with FSP flash driver support with the flash driver configured as Non-Secure Callable.
- **`app_ra4m3_ns_primary`**: The Primary Non-Secure application project which calls the Non-Secure Callable flash driver to erase and write to a code flash region at the top of the code flash area. Upon successful flash operation, all three LEDs blink.
- **`app_ra4m3_s_secondary`**: The Secondary Secure application project with FSP flash driver support with the flash driver configured as Non-Secure Callable. This application image has the same functionality as the Primary Secure application, you can use this project as a template to update the different functionality and exercise the operation of updating the Secure image independent of the Non-Secure Image update.
- **`app_ra4m3_ns_secondary`**: The Secondary Non-Secure application project which calls the Non-Secure Callable flash driver to erase and write to a code flash region at the top of the code flash area. Upon successful flash operation, only the blue and green LEDs blink.

3.3.3 Compile All the Projects

The bootloader project must be compiled first prior to compiling the application projects. In addition, the secure project must be compiled first prior to the compiling of the corresponding non-secure project. For each project, open the `configuration.xlm` file, click **Generate Project Contents** and then click [build icon] to build the project. Compile the projects following the order listed below:

1. `ra_mcuboot_ra4m3`
2. `app_ra4m3_s_primary`
3. `app_ra4m3_ns_primary`
4. `app_ra4m3_s_secondary`
5. `app_ra4m3_ns_secondary`

For the application projects, the post-build command will also sign the corresponding images. The signed image for the application project is located under the `/Debug` folder and is named `<application_project_name>.bin.signed` (For example, `/app_ra4m3_s_primary/Debug/app_ra4m3_s_primary.bin.signed`).

3.3.4 Debug the Applications and Boot the Primary Applications

Right-click on project `app_ra4m3_s_primary` and select **Debug As > Debug Configurations** and confirm the following configuration information:

- The bootloader is downloaded using the .elf format (which includes image and symbol).
- The Primary secure and non-secure images (`app_ra4m3_s_primary.bin.signed`, `app_ra4m3_ns_primary.bin.signed`) are downloaded using the signed binary as Raw Binary/.
- The Primary secure and non-secure image symbols are included using the .elf files.

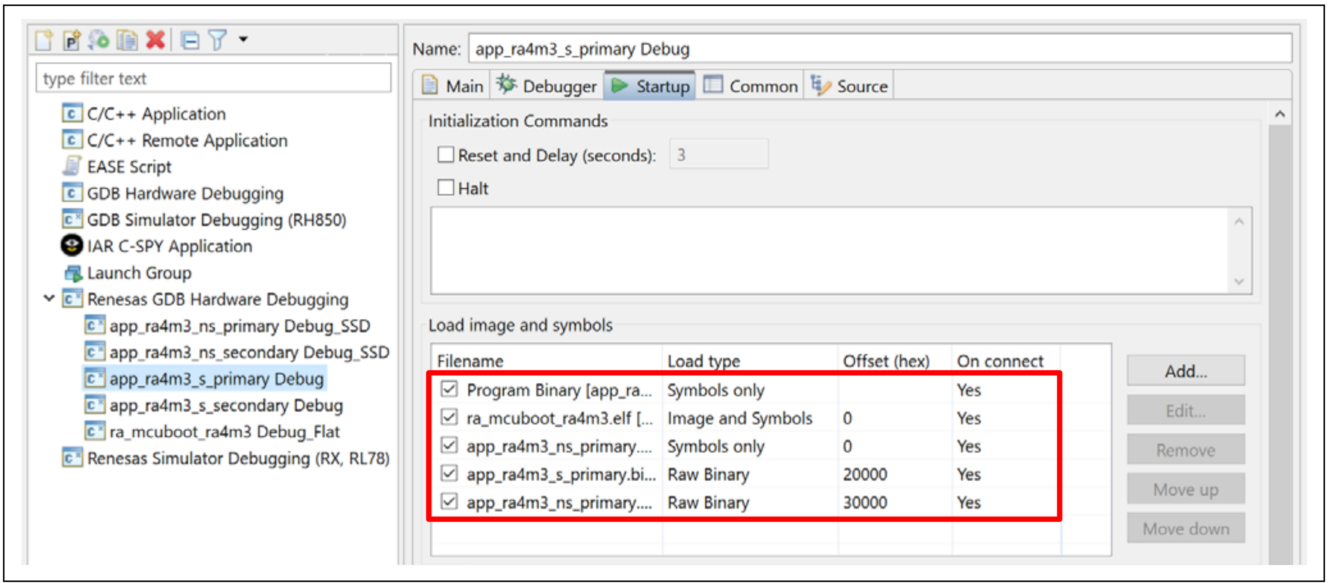

Figure 15. Debug Configuration RA4M3 Overwrite

Click **Debug**.

The debugger should hit the reset handler in the bootloader. Note the address is in the bootloader image.

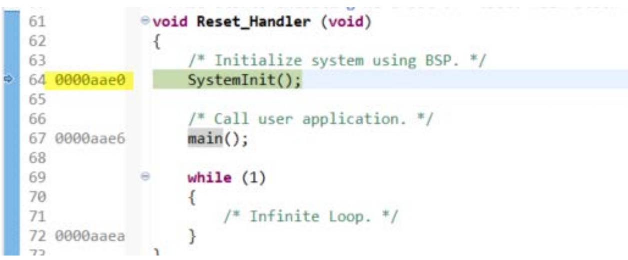

Figure 16. Start the Application Execution

Click Resume twice and boot the Primary image. All three LEDs should be blinking. Pause the execution and confirm the execution is in the Non-secure Primary slot.

Click to run again.

3.3.5 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown below. Set up the search range as: 0x2003e000 0x8000.

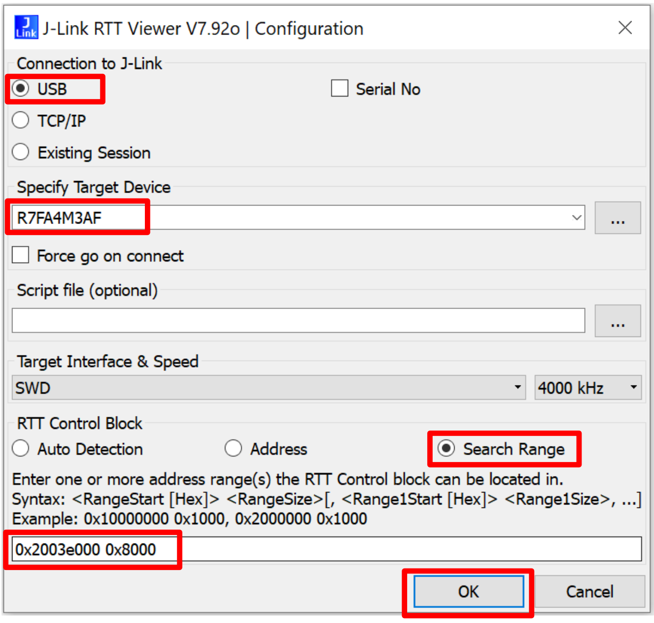

Figure 17. Configure the RTT Viewer

Click **OK** and observe the output on the RTT Viewer. This output shows the Primary application is being executed and all three LEDs are blinking.

Figure 18. Execution of Primary Non-Secure Application for Overwrite Mode

3.3.6 Downloading and Running the Secondary Applications

During development, you can use the ancillary loading capability to load the new secure image to the intended location. You can use the example new secure application provided in this project and follow the steps below to perform an application upgrade:

1. Press the button to pause the program.
2. On the top of the e² studio toolbar, click the Load Ancillary File button to load the new application images as shown in Figure 19 and Figure 20 to the Secondary slot region. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

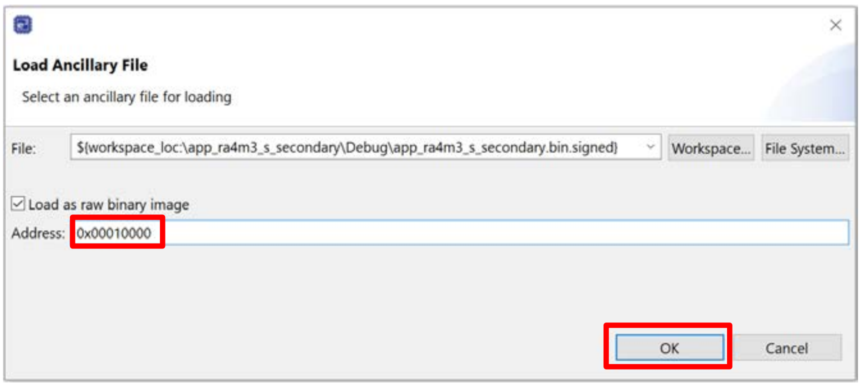

Figure 19. Load the Secondary Secure Application Image for Overwrite Update Mode

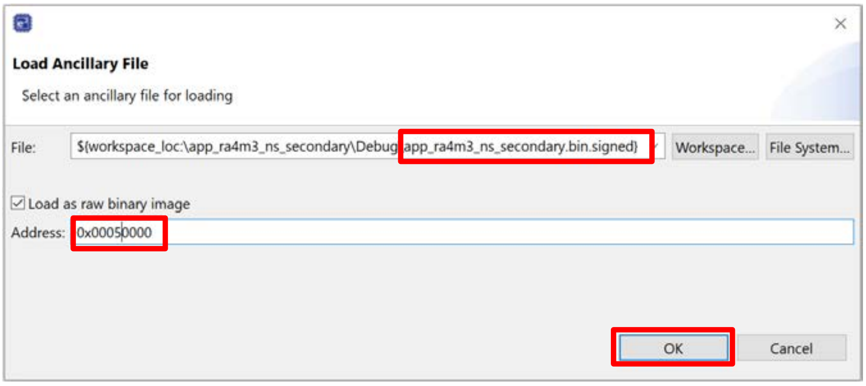

Figure 20. Load the Secondary Non-Secure Application Image for Overwrite Update Mode

3. Click **Resume** . The overwrite occurs and the new image is executed. The blue and green led will be blinking instead of all three LEDs. If the connection between J-Link OB debugger and the target board is terminated, please click **Reset** before clicking Resume button.
4. On the RTT Viewer output, confirm that the following messages are printed and only the blue and green LEDs are blinking.

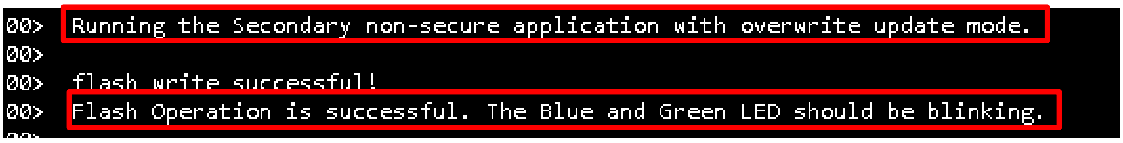

Figure 21. Executing the Secondary Non-Secure image for Overwrite Update Mode

3.3.7 Update the Non-Secure Secondary Image

This step is provided as a reference for implementation of individual image update when designing in a TrustZone environment.

Click **Pause** again and download the Primary Non-Secure application to the Secondary Non-Secure slot using the **Load Ancillary File** tool. Click **OK**. Click **Resume** again. The three LEDs start to blink again and the RTT Viewer shows the same message as Figure 36.

- For Overwrite update mode, if the Secondary image is marked for update, overwrite always occurs.
- It is possible to update the Secure and Non-Secure applications individually with proper application design.

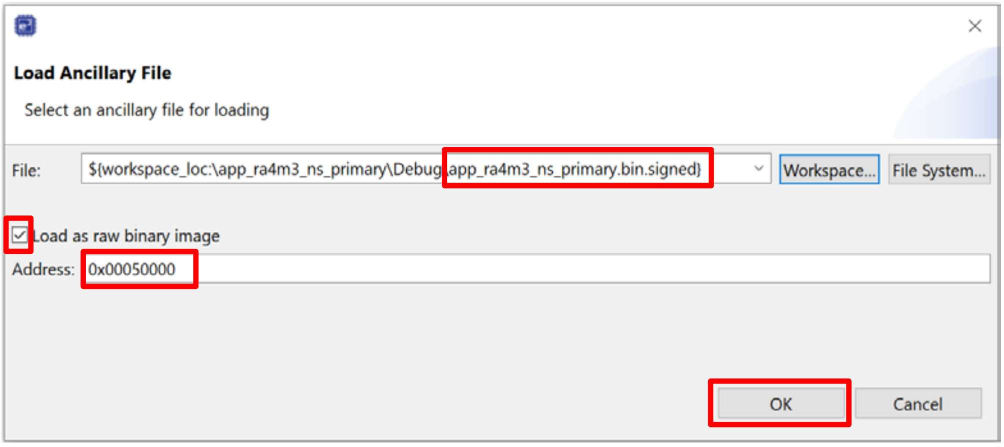

Figure 22. Load the Primary Non-Secure Image to the Second Slot

3.4 Running the EK-RA4M3 Swap Update Mode Example with TrustZone

The process of running the EK-RA4M3 Swap Update mode is similar to the Overwrite Update mode with TrustZone. This section focuses on the difference in the operation:

1. Follow section 3.1.1 to initialize the RA4M3 MCU.
2. Import the project under folder `\ra4m3_swap_with_bootloader_tz` to a workspace.

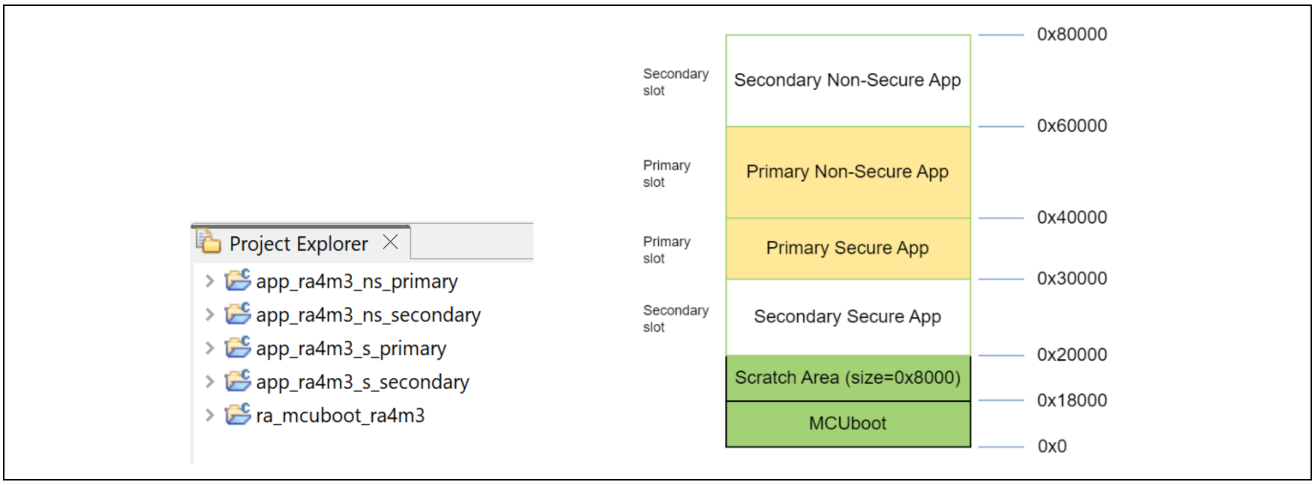

Figure 23. Example Projects for RA4M3 Swap Update Mode

- The bootloader project `ra_mcuboot_ra4m3` has similar functionality as the bootloader with Overwrite Update mode introduced in section 3.3 step 3.3.3 except that the memory map definition and application image update mode are different.
- The functionalities of the application projects are the same as the Overwrite Update mode.

3. Configure the Python Signing Environment by following section 3.2 if this is the first time you are signing the application image.
4. Compile the example projects in the same order as the Overwrite update mode by referencing section 3.3 step 3.3.3. Ensure the signed image for the application project is located under the /Debug folder and is named <application_project_name>.bin.signed.
5. Review the Debug Configuration and boot the Primary applications by referencing section 3.3.4.

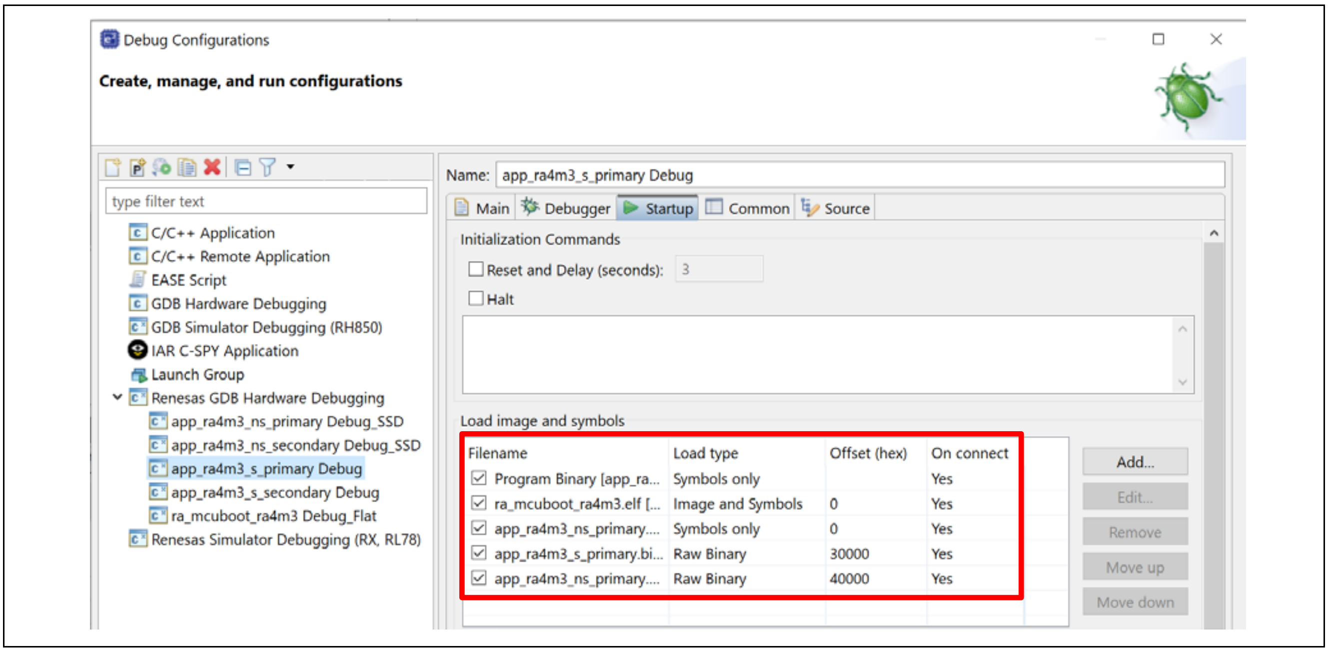

Figure 24. Debug Configuration RA4M3 Swap Update Mode

6. Open the J-Link RTT Viewer and set up the same configuration as Figure 17.
7. Click **OK** and observe the following output on the RTT Viewer. This output shows the Primary application is being executed and all three LEDs are blinking.

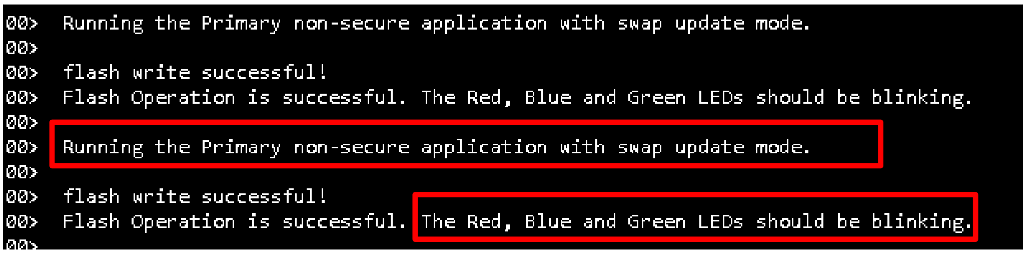

Figure 25. Execution of Primary Non-Secure Application for Swap Update Mode

3.4.1 Downloading and Running the Secondary Applications

During development, you can use the Ancillary loading capability to load the new Secure image to the intended location. You can use the example new Secure application provided in this application and follow the steps below to perform an application upgrade. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

1. Press the button to pause the program.
2. Load the secure new application image to the Secondary slot region using the Ancillary loading capability from the top of the e² studio toolbar in a similar way as Figure 19 except use address 0x20000.
3. Load the non-secure new application image to the Secondary slot region using the Ancillary loading capability from the top of the e² studio toolbar in a similar way as Figure 20 except use address 0x60000.
4. Click **Resume** . The swap occurs, and the new image is executed. Only the blue and green LEDs should be blinking.
5. Confirm the execution result.

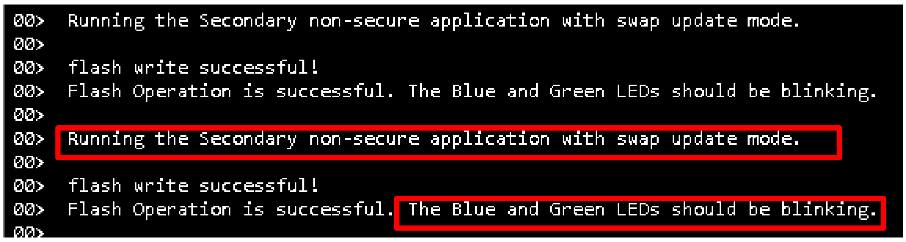

Figure 26. Executing the Secondary Non-Secure Image for Swap Update Mode

3.5 Running the EK-RA4M3 DXIP Update Mode Example

The process of running the EK-RA4M3 DXIP Update Modes is similar to the Overwrite Update mode. This section will focus on the difference in the operation:

1. Follow section 3.3 to initialize the RA4M3 MCU.
2. Import the project under folder `\ra4m3_dxip_with_bootloader_flat` to a workspace and see the following set of example projects in Figure 27.

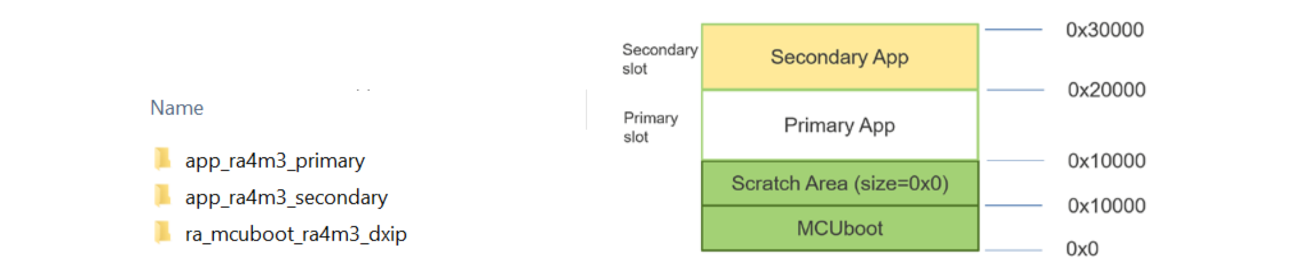

Figure 27. Example Projects for RA4M3 Direct XIP Update Mode

The functionalities of the application projects are blinking the LEDs and providing RTT viewer outputs.

3. Configure the Python signing environment by following section 3.2 if this is the first time you are signing the application image.
4. The bootloader needs to be compiled first. For each project, open the `configuration.xlm` file, click **Generate Project Contents** and then click [build icon] to build the project. Compile the example projects following the orders below. Ensure the signed image for the application project is located under the `/Debug` folder and is named <application_project_name>.bin.signed
    1. `ra_mcuboot_ra4m3_dxip`
    2. `app_ra4m3_primary`
    3. `app_ra4m3_secondary`
5. Verify the debug configuration and follow section 3.3 step 3.3.4 to start debugging the application.

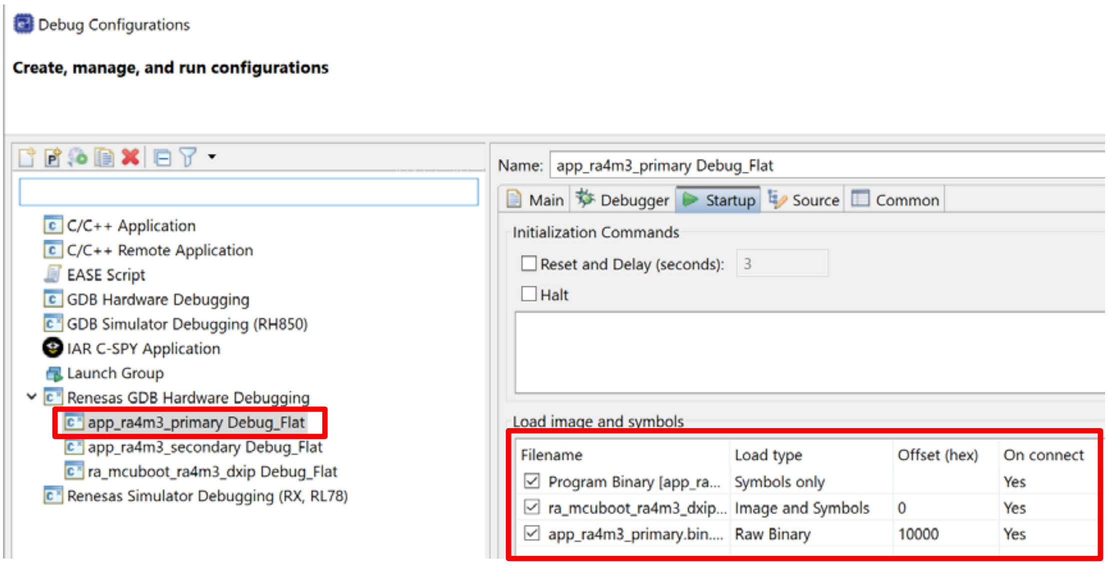

Figure 28. Debug Configuration DXIP Update Mode

6. Open the J-Link RTT Viewer and set up configuration similar as Figure 17 except change the search range to 0x20000000 0x8000.
7. Click **OK** and observe the following output on the RTT Viewer. This output shows the Primary application is being executed and all three LEDs are blinking.

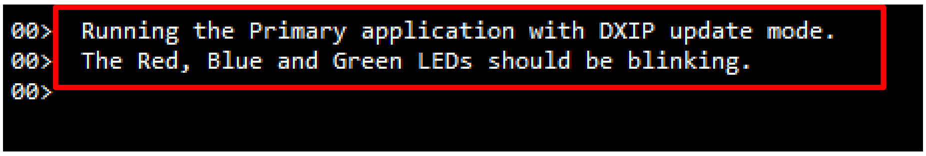

Figure 29. Execution of Primary Application for DXIP Update Mode

3.5.1 Downloading and Running the Secondary Applications

Refer to section 3.8 for trouble shooting when using the Load Ancillary File function.

During development, you can use the Ancillary loading capability from the top of the e² studio toolbar to load the new image to the intended location. You can use the example new application provided in this application and follow the steps below to perform an application upgrade:

1. Press the button to pause the program.
2. Load the new application image to the Secondary slot region using the Ancillary loading capability from the top of the e² studio toolbar.

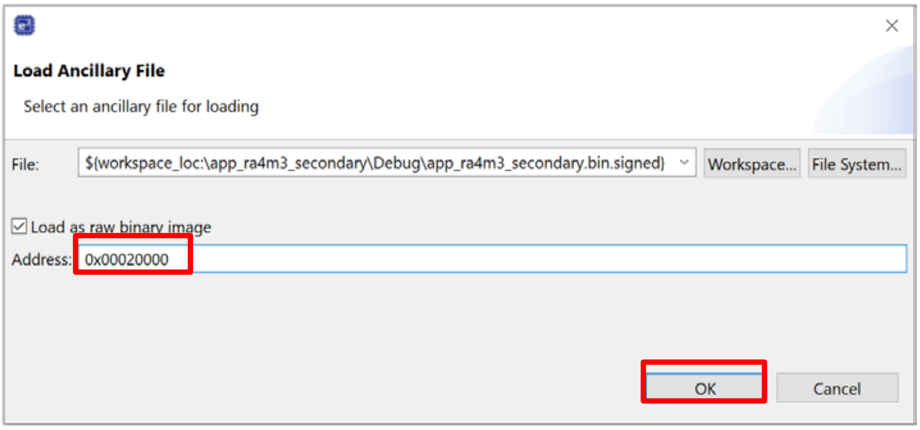

Figure 30. Load the Secondary Secure Application Image for DXIP Update Mode

3. Click **Resume** . The swap occurs, and the new image is executed. Only the blue LED should be blinking.
4. Confirm the same configuration as shown in Figure 17, then click **OK.**

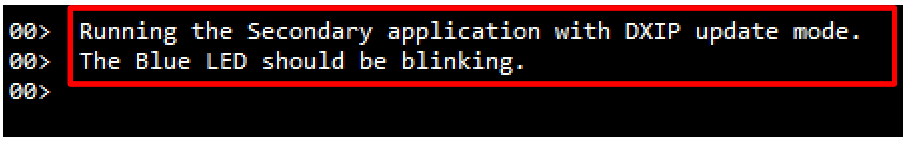

Figure 31. Executing the Secondary Image for DXIP Update Mode

3.6 Running the EK-RA4M3 Overwrite Update Mode Example without TrustZone

Follow the steps below to run the example projects for EK-RA4M3 using the MCUboot Overwrite Only Update mode without TrustZone.

3.6.1 Import the Projects under Folder `\ra4m3_overwrite_with_bootloader` to a Workspace

The following example projects are included in this folder:

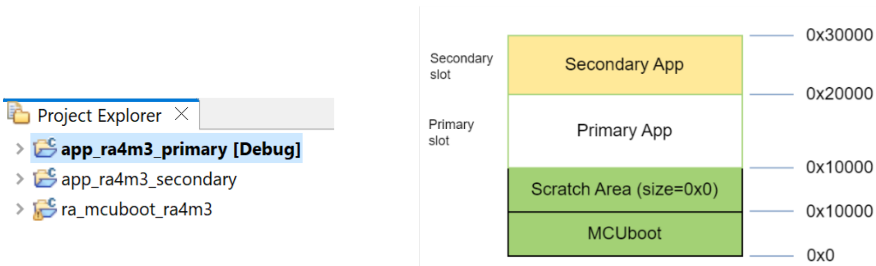

Figure 32. Example Projects for RA4M3 Overwrite Update Mode

- Project `ra_mcuboot_ra4m3` is the bootloader project.
- Project `app_ra4m3_primary` is the initial Primary application project. This project blinks the three LEDs on the EK-RA4M3 kit.
- Project `app_ra4m3_secondary` is the Secondary application project. This project blinks the blue LED on the EK-RA4M3 kit.

Follow section 3.2 to set up the Python signing environment if this is the first time you are signing the application image.

3.6.2 Compile the Projects

The bootloader needs to be compiled first. For each project, open the `configuration.xml` file, click **Generate Project Contents,** and then click to build the project. For the application projects, the post-build command will also sign the corresponding images. The signed image is located under the `\Debug` folder and is named `<project_name>.bin.signed` (for example, `/app_ra4m3_primary/Debug/app_ra4m3_primary.bin.signed`)

1. `ra_mcuboot_ra4m3`
2. `app_ra4m3_primary`
3. `app_ra4m3_secondary`

3.6.3 Debug the Applications and Boot the Primary Application

Right-click on project **app_ra4m3_primary** and select **Debug As > Debug Configuration.**

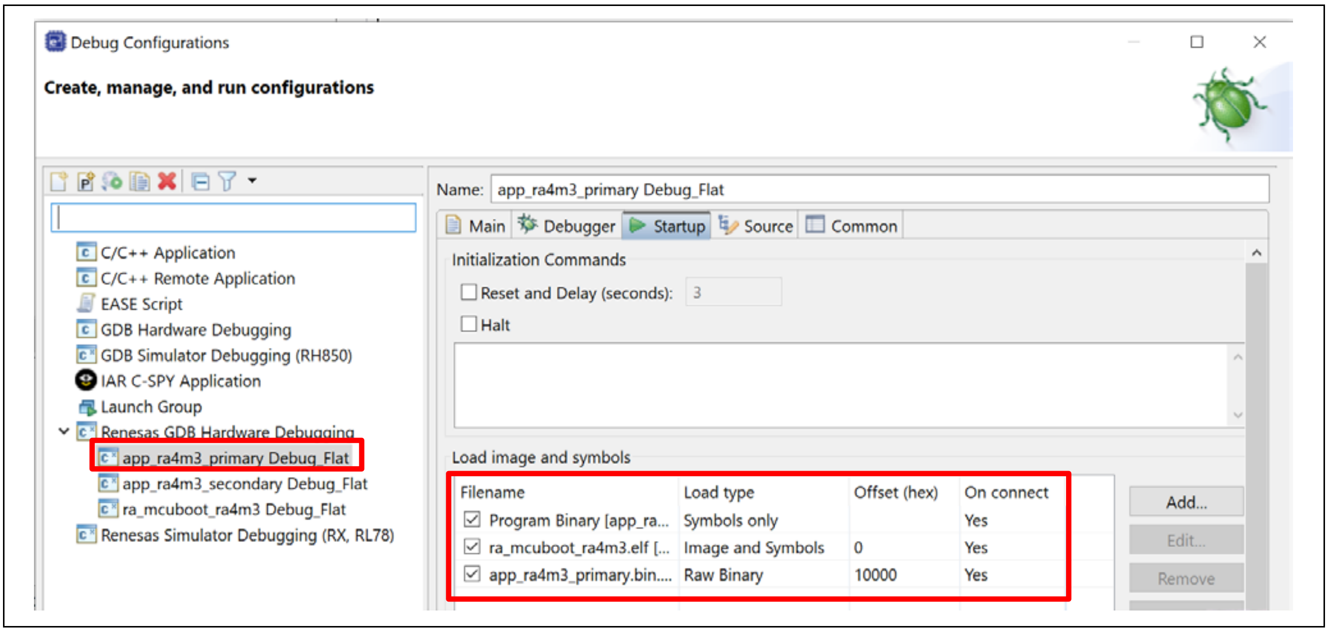

Figure 33. Debug Configuration RA4M3 Overwrite Update

Click **Debug**.

The debugger should be at the reset handler in the bootloader. Note the address is in the bootloader image.

Figure 34. Start the RA4M3 Application Execution

Click Resume twice and boot the Primary image. All three LEDs should be blinking.

3.6.4 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown below. Configure the address search range as 0x20000000 0x8000.

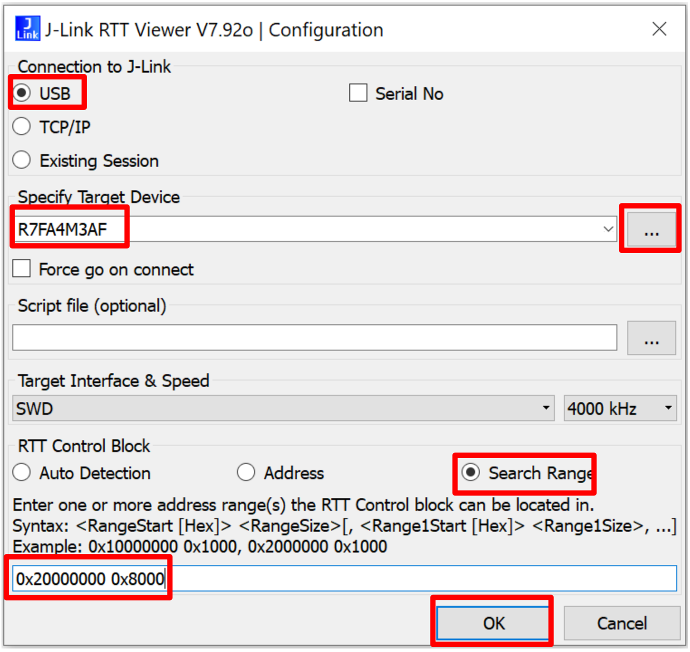

Figure 35. Configure the RTT Viewer for RA4M3 Project

Click **OK** and observe the following output on the RTT Viewer. This output shows the Primary application is being executed and all three LEDs are blinking.

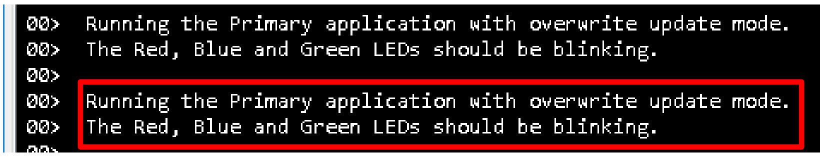

Figure 36. Execution of Primary Application for Overwrite Mode

3.6.5 Downloading and Running the Secondary Applications

During development, you can use the Ancillary loading capability  to load the new Secure image to the intended location. Follow the steps below to perform an application upgrade. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

1. Press to pause the program.
2. Load the new application image to the Secondary slot region using the Ancillary loading capability from the top of the e[2] studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x20000.

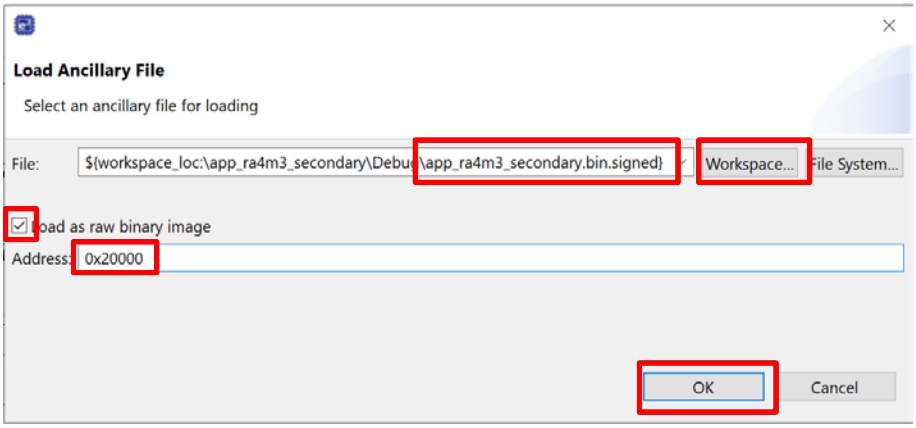

Figure 37. Load the Secondary Application Image for Overwrite Mode

3. Click Resume . The overwrite occurs and the new image is executed. Now only the Blue LED should be blinking.
4. Confirm the same configuration as shown in Figure 35, then click **OK**. The following output is printed and only the blue LED blinks.

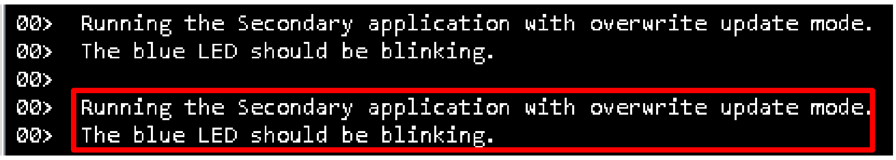

Figure 38. Executing the Secondary Application Image for Overwrite Update Mode

3.7 Running the EK-RA4M3 Swap Test Update Mode Example without TrustZone

Follow the steps below to run the example projects for EK-RA4M3 using the MCUboot Swap Test Update mode without TrustZone.

3.7.1 Import the Projects

Import the projects under Folder `\ra4m3_swap_test_with_bootloader` to a Workspace.

The following example projects are included in this folder:

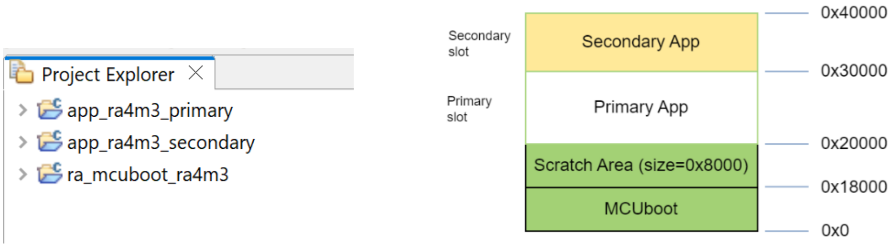

Figure 39. Example Projects for RA4M3 Swap Test Update Mode

- Project `ra_mcuboot_ra4m3` is the bootloader project.
- Project `app_ra4m3_primary` is the initial Primary application project. This project blinks the three LEDs on the EK-RA4M3 kit.
- Project `app_ra4m3_secondary` is the Secondary application project. This project blinks the blue LED on the EK-RA4M3 kit.

Follow section 3.2 to set up the Python signing environment if this is the first time you are signing the application image.

3.7.2 Compile the Projects

The bootloader project needs to be compiled first. For each project, open the `configuration.xml` file, click **Generate Project Contents,** and then click to build the project. Compile the projects in following the order:

1. `ra_mcuboot_ra4m3`
2. `app_ra4m3_primary`
3. `app_ra4m3_secondary`

For the application projects, the post-build command will also sign the corresponding images. The signed image is located under the `\Debug` folder and is named `<project_name>.bin.signed` (for example, `/app_ra4m3_primary/Debug/app_ra4m3_primary.bin.signed`)

3.7.3 Debug the Applications and Boot the Primary Application

Right-click on project **app_ra4m3_primary** and select **Debug As > Debug Configuration.**

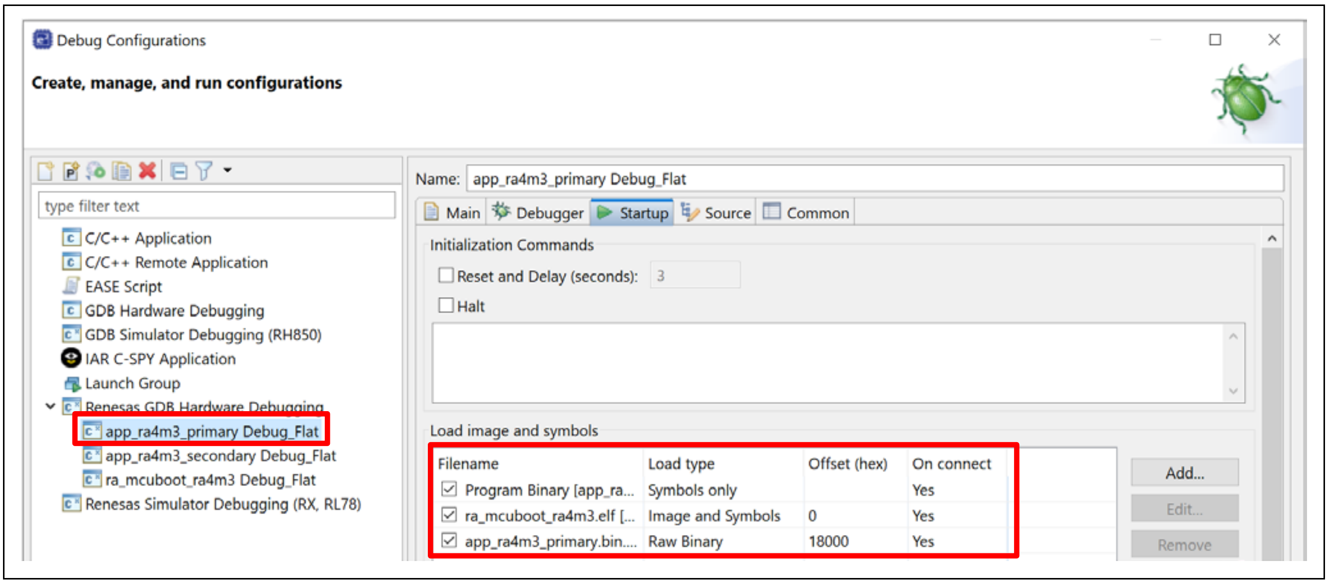

Figure 40. Debug Configuration RA4M3 Swap Update

Click **Debug**.

Click Resume twice and boot the Primary image. All three LEDs should be blinking.

3.7.4 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown in Figure 35. Observe the following output on the RTT Viewer. This output shows the Primary application is being executed and all three LEDs are blinking.

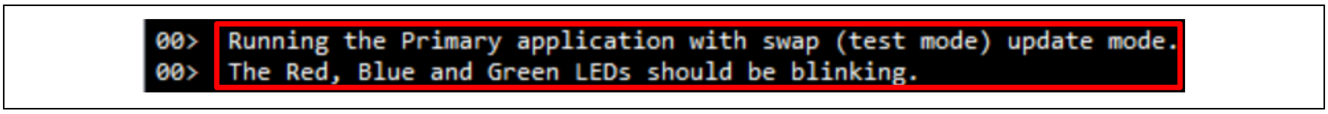

Figure 41. Execution of Primary Application for Swap Test Mode

3.7.5 Downloading and Running the Secondary Applications

During development, you can use the Ancillary loading capability to load the new Secure image to the intended location. Follow the steps below to perform an application upgrade. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

1. Press to pause the program.
2. Load the new application image to the Secondary slot region using the Ancillary loading capability from the top of the e^2 studio toolbar in a similar way as Figure 37. Select **Load as raw binary image** and configure the **Address** to 0x28000.
3. Click Resume . The swap occurs and the new image is executed. Now only the blue LED should be blinking.
4. Confirm the same configuration as shown in Figure 35, then click **OK**. The following output is printed and only the blue LED should blink.

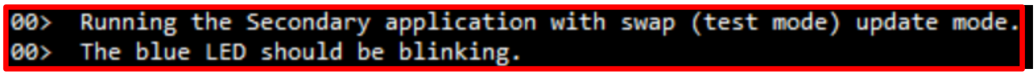

Figure 42. Executing the Secondary Application Image for Swap Test Update Mode

5. Pause and reset the application from the debugger.

3.8 Troubleshooting

When running the example projects, you may experience USB Debug connection or the RTT Viewer connection issue when using the "Load Ancillary File" button to download the Secondary image. To recover from these failures:

- If the USB Debug connection disconnects, the recommendation is to try out another available USB port for the USB Debug connection. If failure persists, contact Renesas support.
- If the RTT Viewer disconnects, the recommendation is to power cycle the board and restart the debug session.

4. Creating the Bootloader

This section provides a walk-through of the bootloader creation of the example projects as well as how to link the standalone application with the bootloader. For most of the steps, the considerations and configurations in creating bootloader with the different upgrade mode are common. Whenever there is a difference in the implementation of the different update mode, the difference will be addressed.

The walk-through of the bootloader creation in this section targets the bootloader used in section 3.3 for the TrustZone enabled system. Wherever there is a need to address the Non-TrustZone enabled implementation, it will be addressed.

4.1 Creating a Bootloader Project for RA Family

The screen captures used in these sections are based on the RA4M3 based bootloader projects used in section 3.3, 3.4, and 3.5. Follow this section to establish the bootloader projects used in section 3.3, which uses Overwrite Only as the application update mode. Updates needed for the bootloader projects used in the section 3.4 and 3.5.are addressed.

The creation of the RA4M3 based bootloaders used in section 3.6 and 3.7 are very similar. Wherever there is a difference in the operation, it will be addressed inline.

4.1.1 Start Bootloader Project Creation with e² studio

Follow the steps below to create the initial bootloader project based on EK-RA4M3:

1. From the e² studio Workspace, navigate to the **File > New > Renesas C/C++ Project > Renesas RA** and then select **Renesas RA C/C++ Project** and press **Next.**
   Provide the project name `ra_mcuboot_ra4m3` and click **Next**. The exact name needs to be provided to follow the default instructions in this section. If a different name is provided, all instructions related to the name of the bootloader project need to be updated accordingly.
   In the next screen, select **FSP version 5.2.0** and the **EK-RA4M3** board. Use the default **Debugger** setting **J-Link Arm** and click **Next.**
   Note that if the creation process is using other newer FSP versions, some details on the error messages shown when the MCUboot module is initially added may be different. Adapt the actions accordingly to satisfy the dependencies.
2. When the following screen appears, select **Flat (Non-TrustZone) Project**.

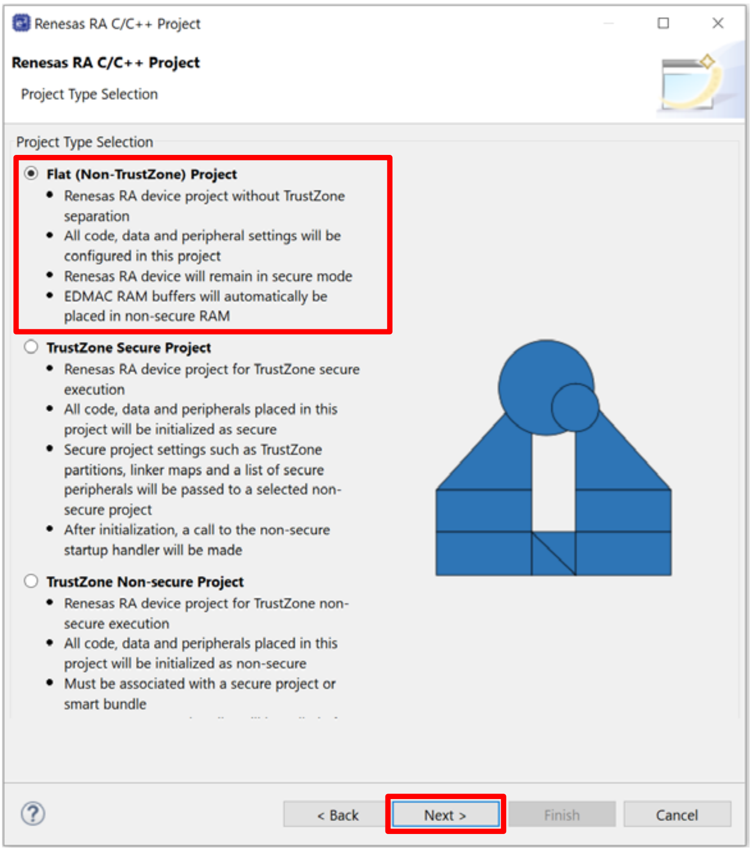

Figure 43. Choose Flat Project as Project Type

3. Choose **Executable** as the **Build Artifact Selection** and **No RTOS**. Click **Next**.

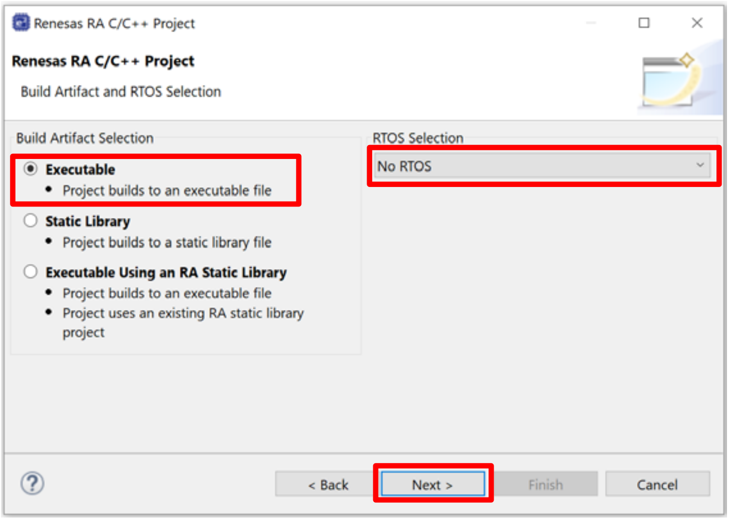

Figure 44. Choose Executable and No RTOS

4. In the next screen, select the project template.
   Choose **Bare-Metal – Minimal** as the **Project Template Selection** and click **Next**.
5. Update the Pin configuration file.
   The project will now be created, and the bootloader project configuration will be displayed. Select the **Pins** tab and deselect the **Generate data** check box. Use the pull-down menu to switch from **RA4M3 EK** to **R7FA4M3AF3CFB.pincfg** for the **Select Pin Configuration** option, select the **Generate data** check box and enter **g_bsp_pin_cfg**.

Figure 45. Uncheck Generate Data for RA4M3 EK Pin Configuration

Figure 46. Select R7FA4M3AF3CFB.pincfg and Generate data g_bsp_pin_cfg

   Note that when we select the Flat Project model, the I/Os are configured as Secure by default. Updating the pin configuration as shown above selects the pin configuration with the minimal number of pins defined because any I/O that is defined in the Flat project will not be available for use in the Non-Secure application and can only be accessed by the Secure application.
6. Add the MCUboot module.
   Change to the **Stacks** tab and select **New Stack > Bootloader > MCUboot**.

Figure 47. Add the MCUboot Module

4.1.2 Resolve the Configurator Dependencies

After the MCUboot module is brought into the configurator, follow the steps in this section to resolve the dependencies:

1. Resolve the following dependency of the MCUboot by adding the **MbedTLS (Crypto Only)** stack.

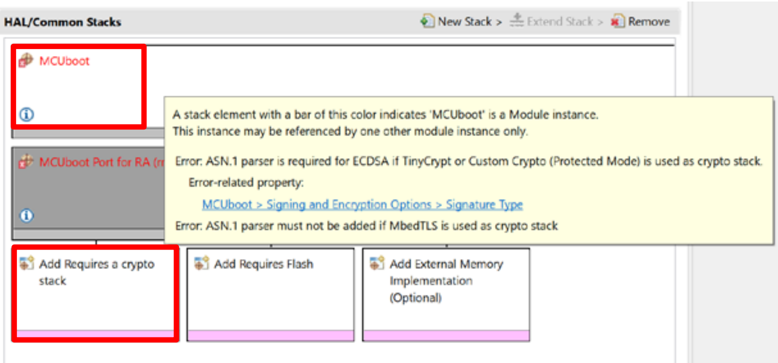

Figure 48. MCUboot Module Dependency

Left click on Add Crypto Stack, choose **New** and add the **MbedTLS (Crypto Only)** stack.

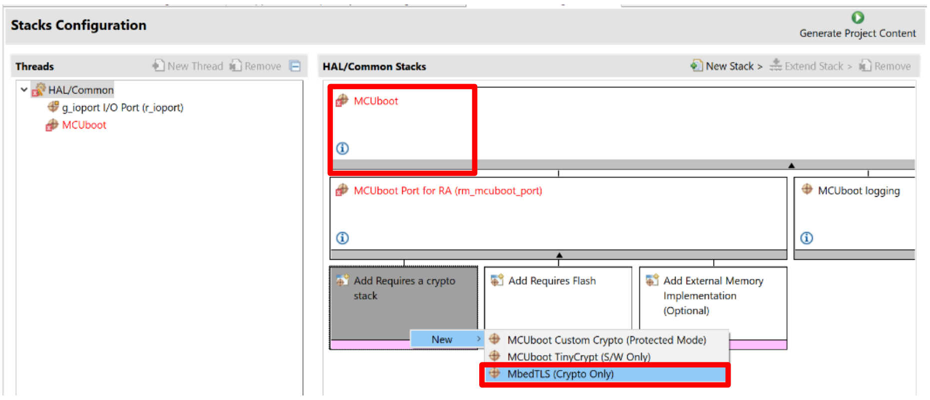

Figure 49. Add MbedTLS (Crypto Only) Module

2. Configure the Mbed Crypto dependencies.
   Follow the prompt in Figure 50 to update the corresponding properties for the MCUboot Port for RA Module.

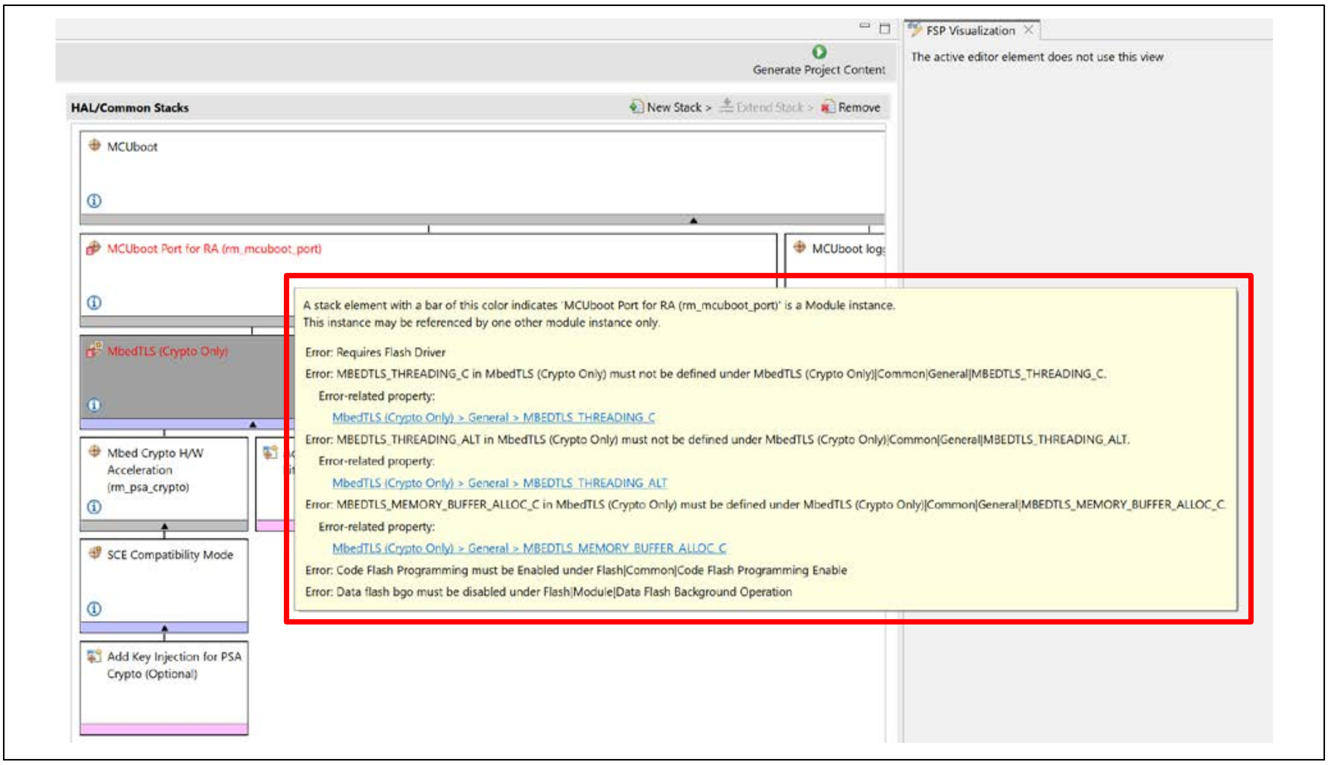

Figure 50. Dependencies of MCUboot Module for RA Stack

Configure the following properties:

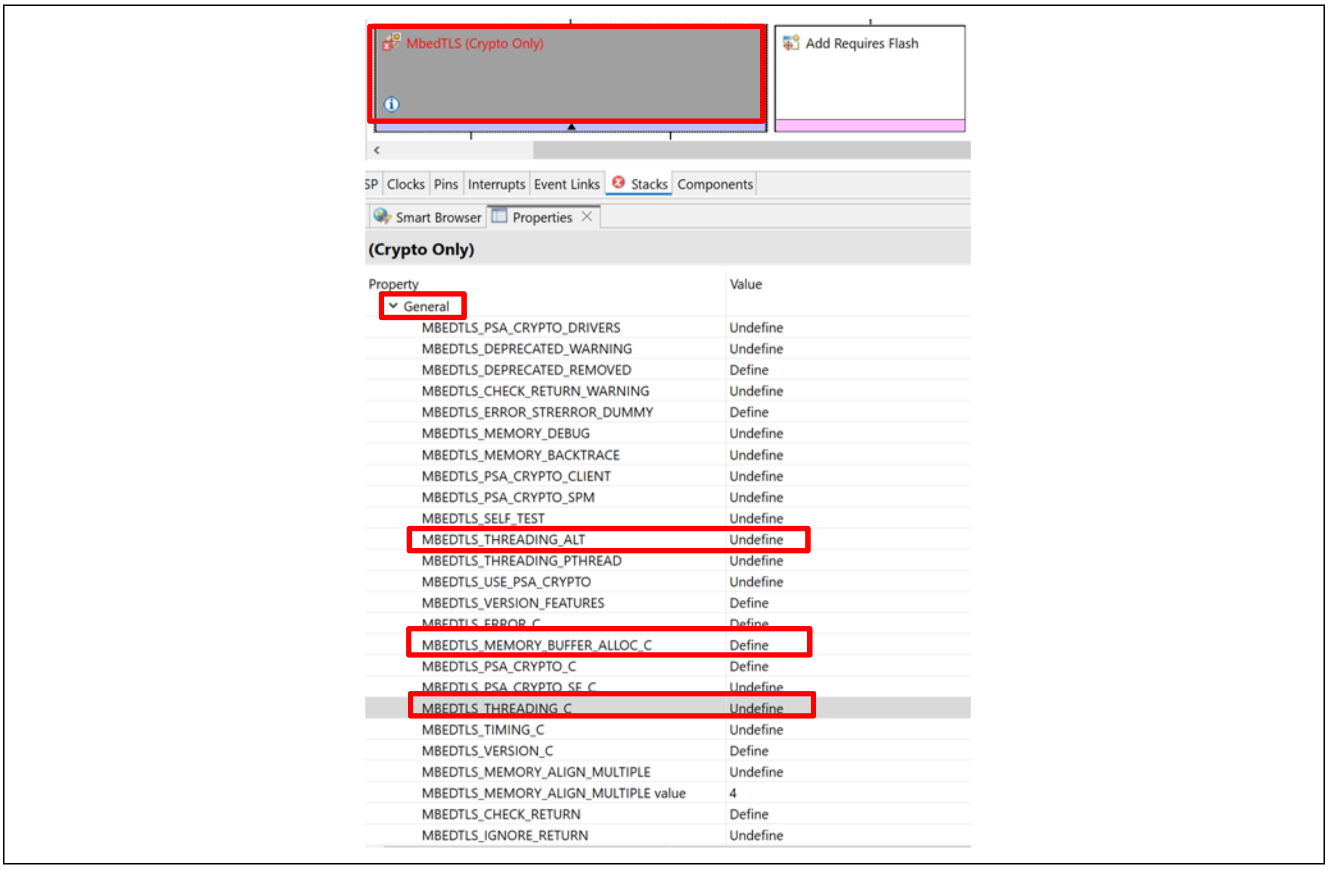

Figure 51. Configure Highlighted Properties for the MbedTLS (Crypto Only) Stack

Add the r_flash_hp module:

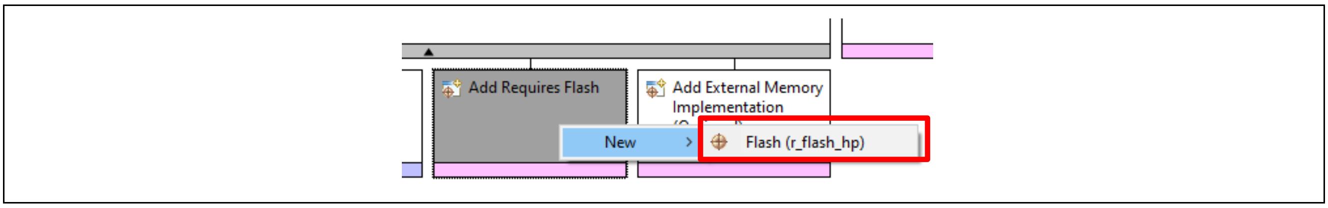

Figure 52. Add the r_flash_hp Stack

Configure the r_flash_hp stack:

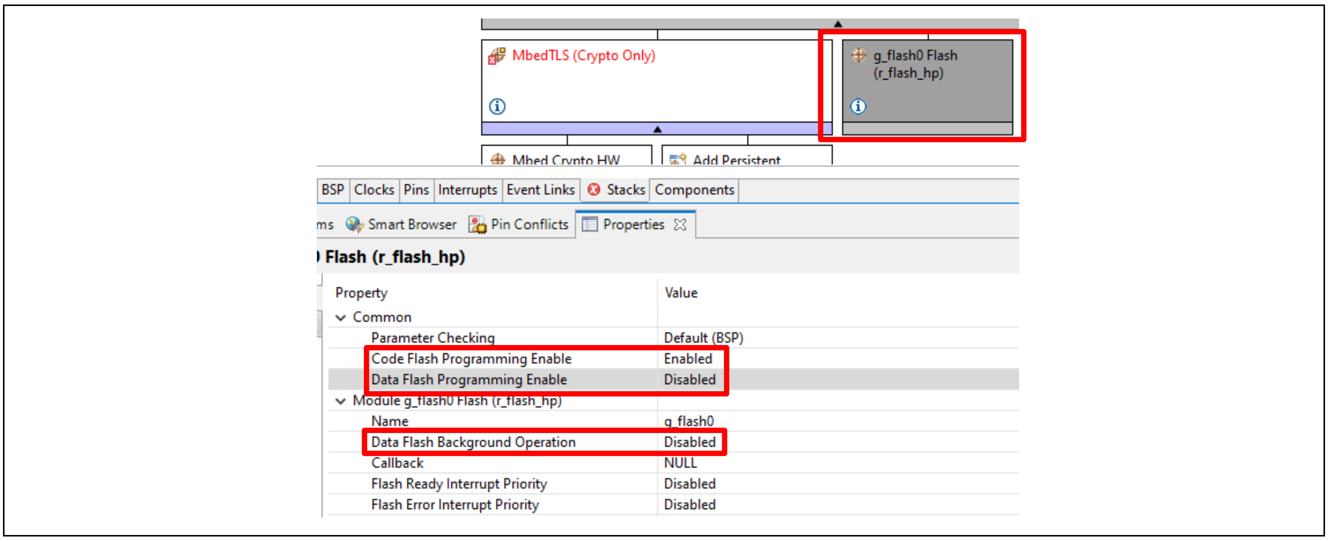

Figure 53. Configure the r_flash_hp Stack

3. Hover the cursor over MbedTLS (Crypto Only) stack. You will see warnings as shown in Figure 54.
4. Under the BSP tab, set up the stack and heap size to support ECC:
   - **RA Common** > (set **Main stack size** to 0x1000 and **Heap size** to 0x400)

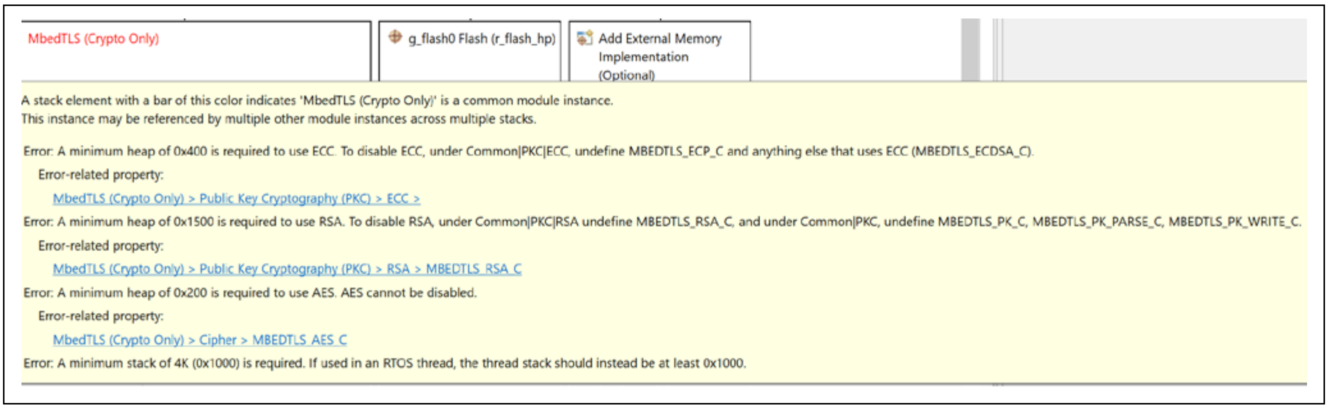

Figure 54. Dependencies of Mbed TLS (Crypto Only) Stack

5. Disable RSA following the prompt in Figure 55. This bootloader design uses ECC for signature generation. Disable the RSA algorithm to save the BSP Heap size.

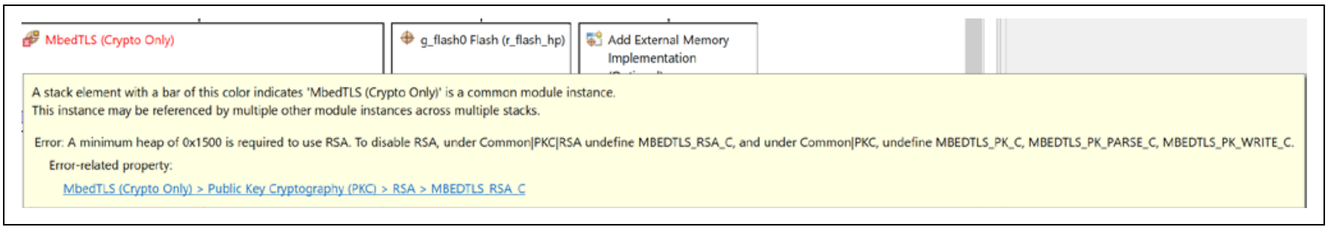

Figure 55. Dependencies of RSA

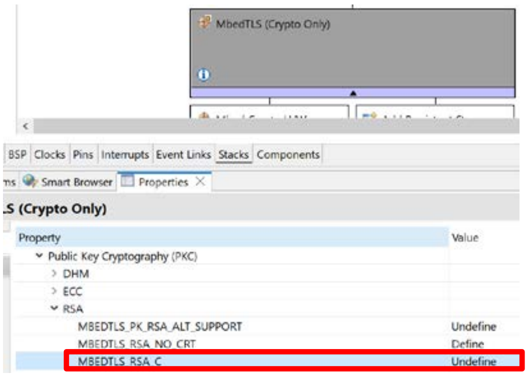

Figure 56. Disable RSA

At this point the error message in the stack window should have been resolved.

6. Decide the number of application images.

   For MCUs with TrustZone support:

   - If the application uses TrustZone, there will be two application images in each slot: secure and non-secure application. In this case, set the **Number of images per Application** to 2.
   - If the application does not use TrustZone, there will be one application image in each slot. In this case, set the **Number of images per Application** to 1.
   - The bootloader used in section 3.3 uses TrustZone, so for this example bootloader, set the Number of images per Application to 2. **MCUboot > Common > General > Number of images per Application** (change from **1** to **2**).

   For MCUs without TrustZone support, set this property to 1.

7. Configure the **Flash Layout** for RA4M3 Overwrite Update with TrustZone as shown below based on the standalone application projects described in section 5. For your application projects, you can follow the guidelines in section 2.3 to design the bootloader memory allocation. This configuration matches the bootloader used in section 3.3.

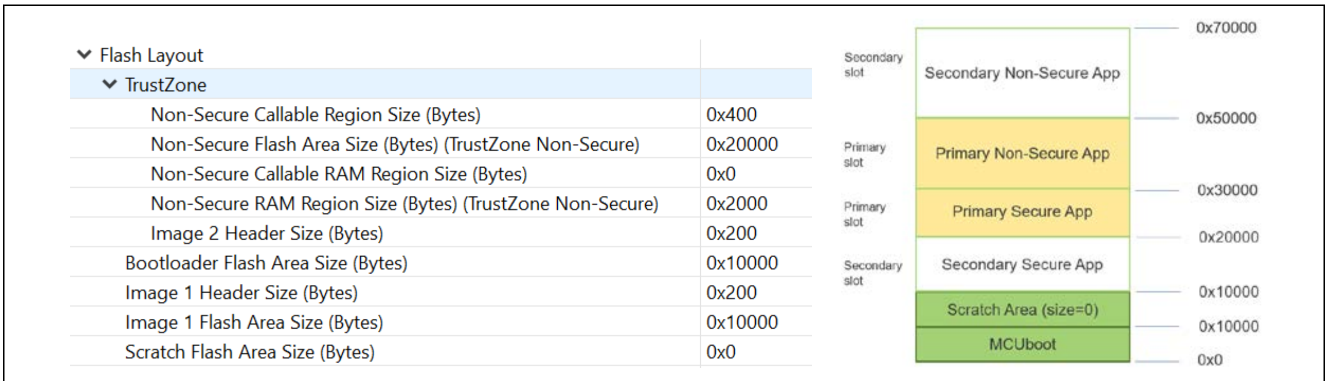

Figure 57. Memory Configuration of Overwrite Update Mode RA4M3 with TrustZone

Configure the MCUboot module and application memory allocation based on RA4M3 Swap Update mode with TrustZone as shown below based on the standalone application projects described in section 5. This configuration matches the bootloader used in section 3.4.

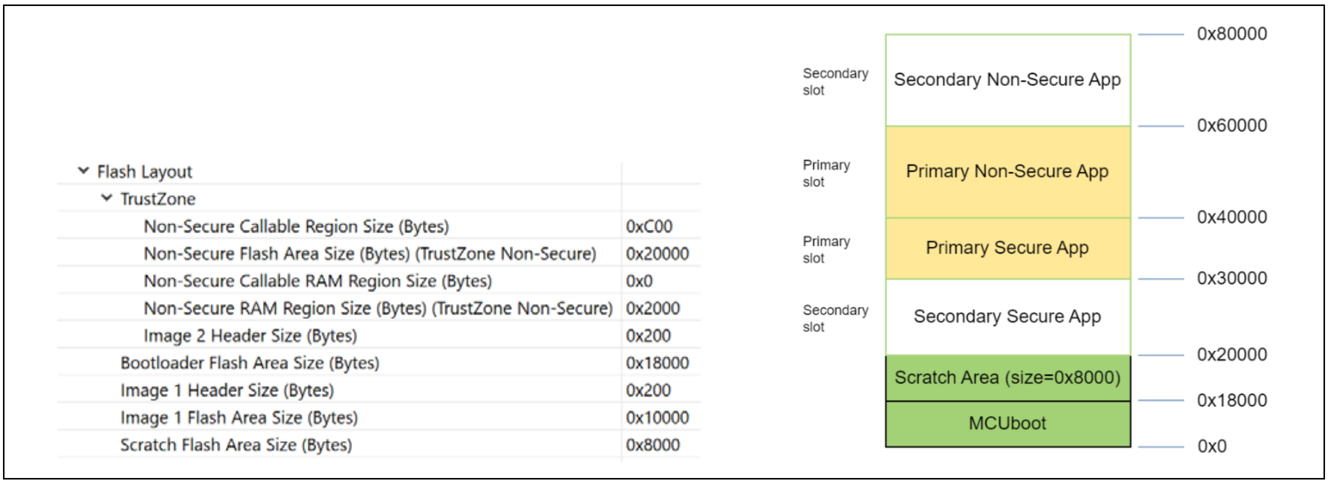

Figure 58. Memory Configuration of Swap Update Mode RA4M3 with TrustZone

Configure the MCUboot module and application memory allocation based on RA4M3 Direct XIP mode based on the example projects presented in section 3.5. This configuration matches the bootloader used in section 3.5.

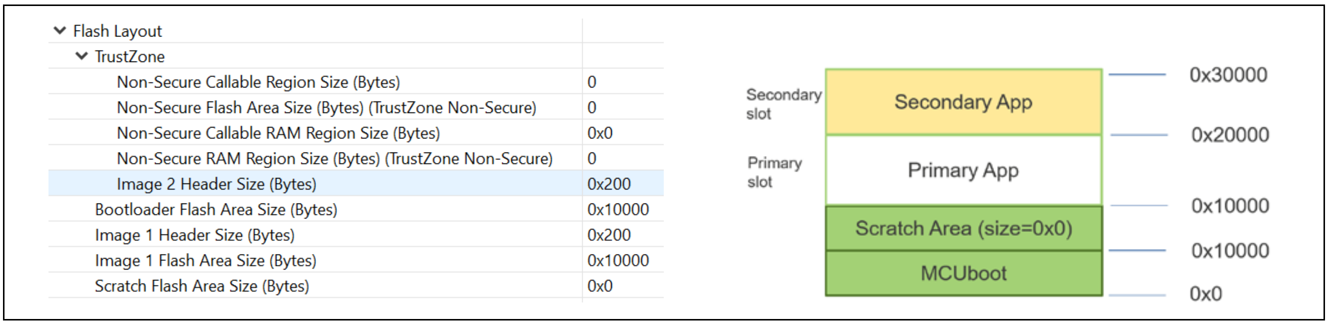

Figure 59. Memory Configuration of Direct XIP Update Mode RA4M3

Configure the MCUboot module and application memory allocation based on the RA4M3 Overwrite Update mode without TrustZone based on the example projects presented in section 3.6. This configuration matches the bootloader used in section 3.6.

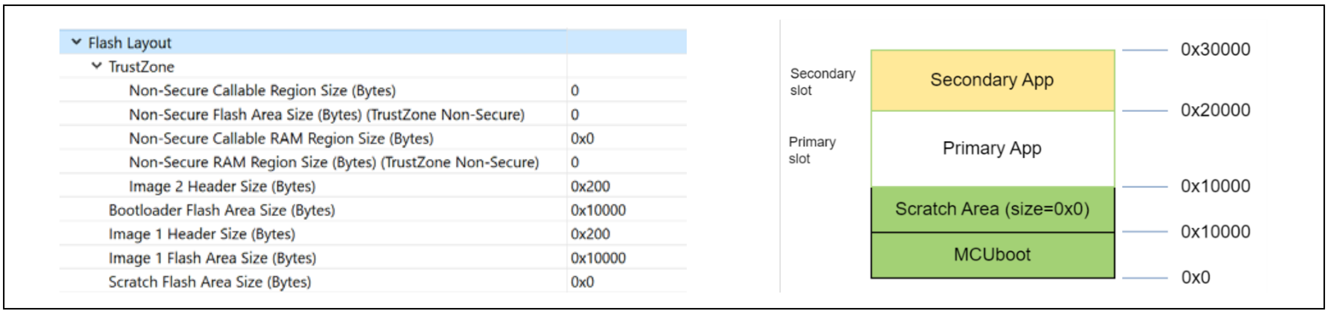

Figure 60. Memory Configuration of Overwrite Update Mode RA4M3 without TrustZone

Configure the MCUboot module and application memory allocation based on RA4M3 Swap Test Update mode without TrustZone based on the example projects presented in section 3.7. This configuration matches the bootloader used in section 3.7.

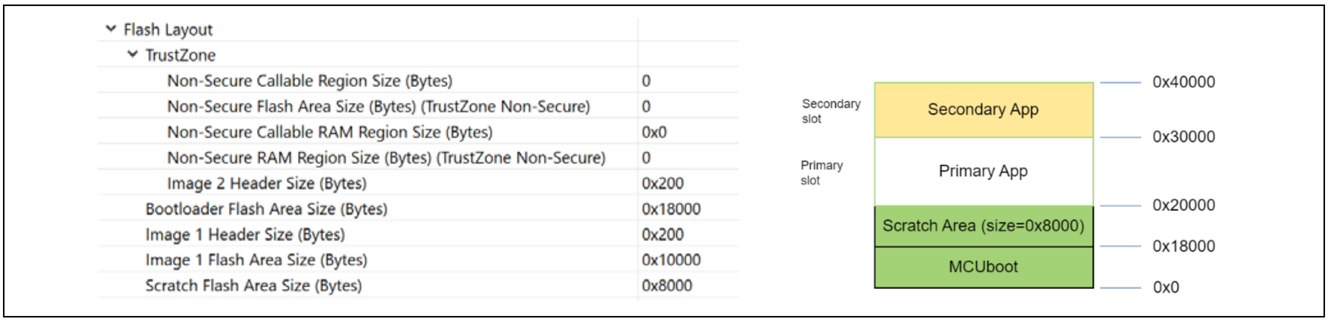

Figure 61. Memory Configuration of Swap Test Update Mode RA4M3 without TrustZone

For the configuration of the swap test mode run time support, refer to application note R11AN0516 to understand the operation.

4.1.3 Setting up the Booting Authentication Support

To use the example keys, select **Add Example Keys** > **New** > **MCUboot Example Keys (NOT FOR PRODUCTION)**.

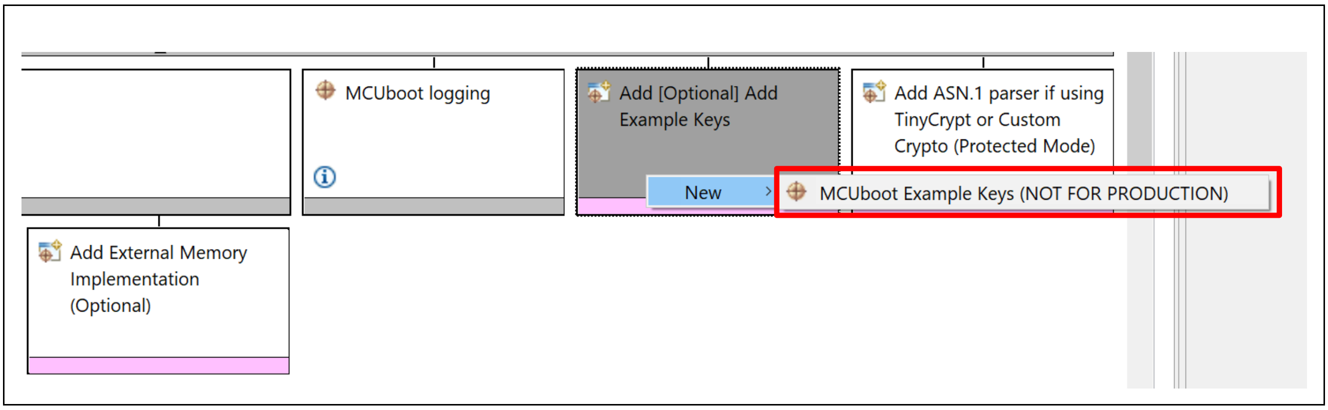

Figure 62. Add the MCUboot Example Key

Note: The example public key and private key used in the MCUboot is for testing purposes only. Refer to section 2.6 for guidelines on selecting the public key and private key for production support. Application Project R11AN0868 includes procedures to create customized key pair preparation. Refer to R11AN0868 to create customized key pairs.

You can choose to use the default pair of public/private keys included in MCUboot for testing purposes:

- The default public keys are defined in `/ra_mcuboot_ra4m3/ra/mcu-tools/MCUboot/sim/mcuboot-sys/csupport/keys.c`.
- The default private keys are included in folder `/ra_mcuboot_ra4m3/ra/mcu-tools/MCUboot/sim/`.

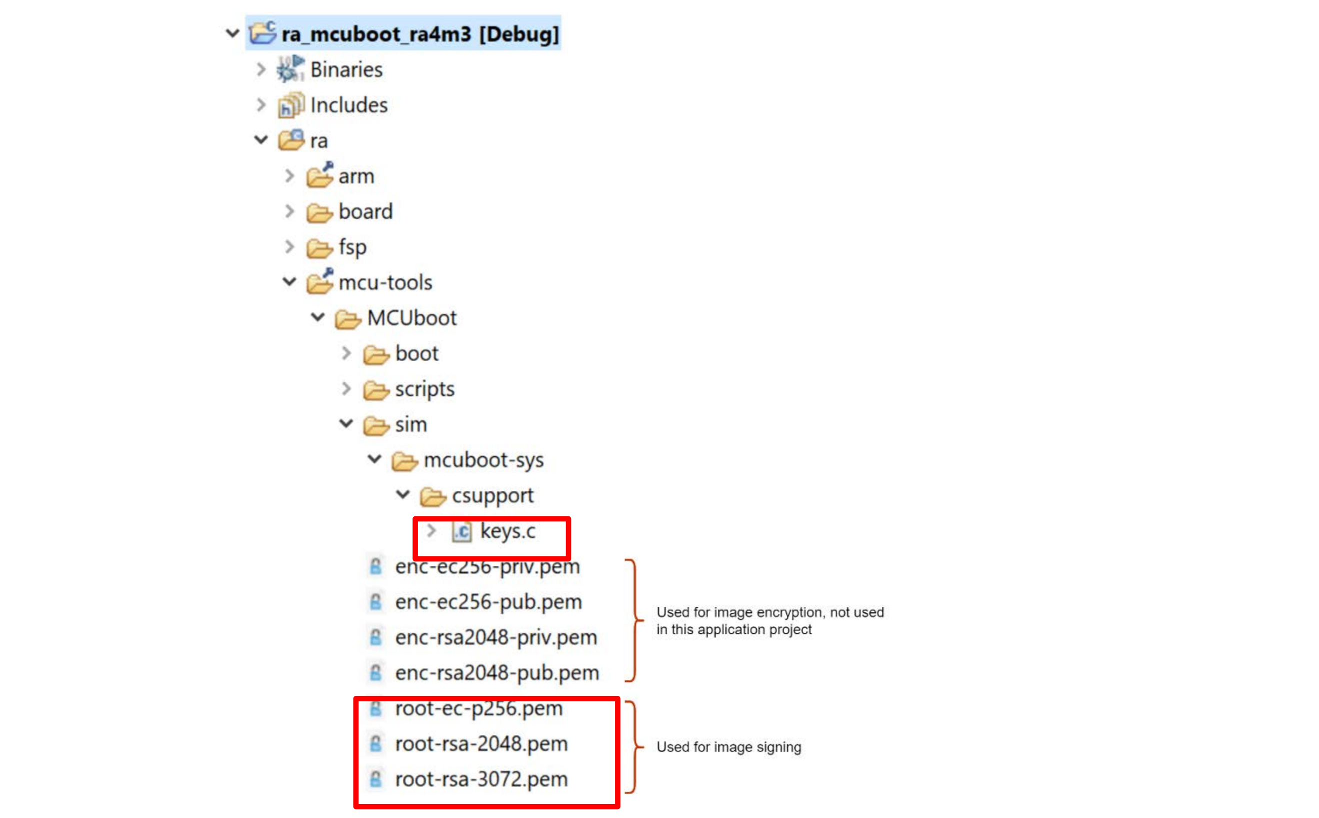

Figure 63. Example Public Keys and Private Keys Included in MCUboot Port Stack

4.1.4 Setting up the Application Authentication Signature Type

There are three signature types supported in FSP as shown below. Open the **Property** page of stack **MCUboot** > **Common** > **Signing Options** to look at the signing options. In this example implementation, ECDSA P-256 is used for all the example bootloaders demonstrated in section 3.

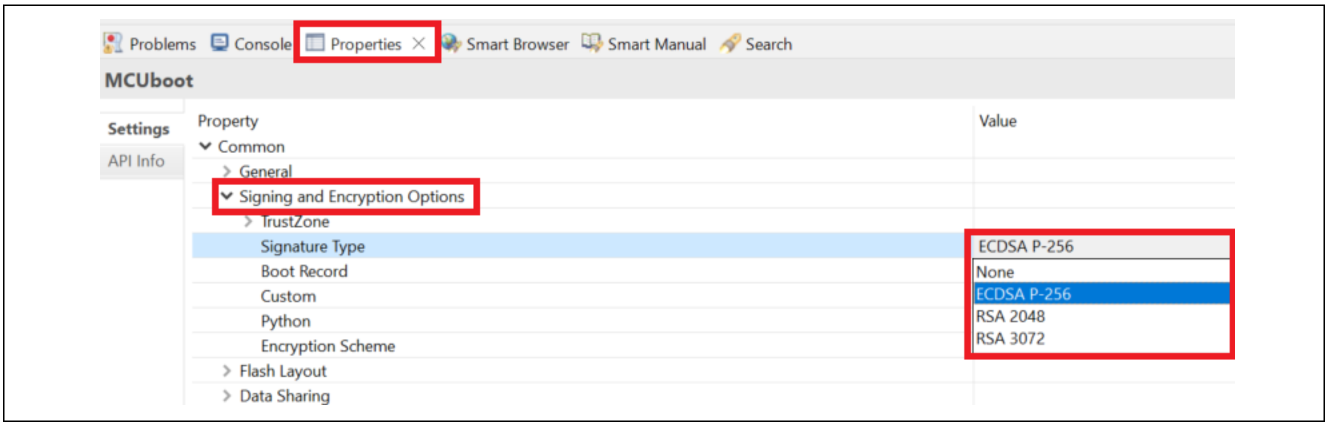

Figure 64. Application Authentication Signing Options

4.1.5 Add MCUboot Activation Code

Follow the steps below to add the MCUboot activation code and compile the bootloader:

1. Add the source code and compile the bootloader.
   Follow the steps below to add the source code to the bootloader project and compile the project.
   - Open `hal_entry.c`.
   - Open **Developer Assistance**.
   - Go to **HAL/Common** > **MCUboot** > **Quick Setup**. Drag **Call Quick Setup** to the top of the `hal_entry.c` file before the `hal_entry()` function call.
   - Call this function at the top of the `hal_entry()` function
     - `mcuboot_quick_setup();`

   Notes on the `mcuboot_quick_setup` function
   - The main functionality established in the bootloader project is established by function `mcuboot_quick_setup`, which performs the following functions:
     - The `boot_go` function does most of the functions of a bootloader except for the final step of jumping to the main image. This function returns a structure pointer (`rsp` for return structure pointer) for the image to boot from.
     - The `RM_MCUBOOT_PORT_BootApp` function cleans up resources used by the bootloader and jumps to the application image.
2. Compile the bootloader project.
   - Save the project (save the source code and the `configuration.xml` file) and click **Generate Project Content** and then compile the project.

5. Using the Bootloader with Applications

A set of existing non-bootloader-based projects are used to demonstrate how to configure existing application projects to use the bootloader. General guidelines are also provided for adapting to other existing applications. Unzip `example_projects_no_bootloader.zip`.

These projects have the same functionality as the projects demonstrated in section 3.3 except these projects are not configured to use the bootloader. Follow the steps below to configure the standalone application projects to use the bootloader and sign the application.

5.1 Import the Standalone Application Projects

Import the RA4M3 standalone example project to the same workspace as the bootloader project you created in the previous section. In this section, we will update these existing projects to use the bootloader created in the previous section.

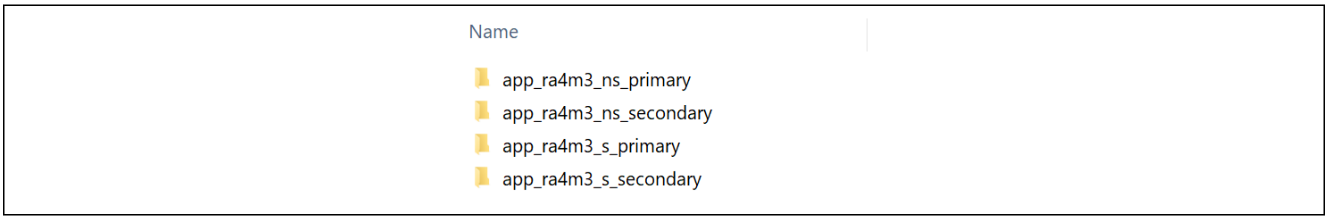

Figure 65. Standalone Example Projects for RA4M3 with No Bootloader support

5.2 Configure the Application Projects to Use the Bootloader

We will now alter the project **Properties** configuration to have it use the bootloader. Right-click on the `app_ra4m3_s_primary` folder 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 **${workspace_loc:ra_mcuboot_ra4m3}/Debug/ra_mcuboot_ra4m3.bld** for the value. Click **OK** to save the changes.

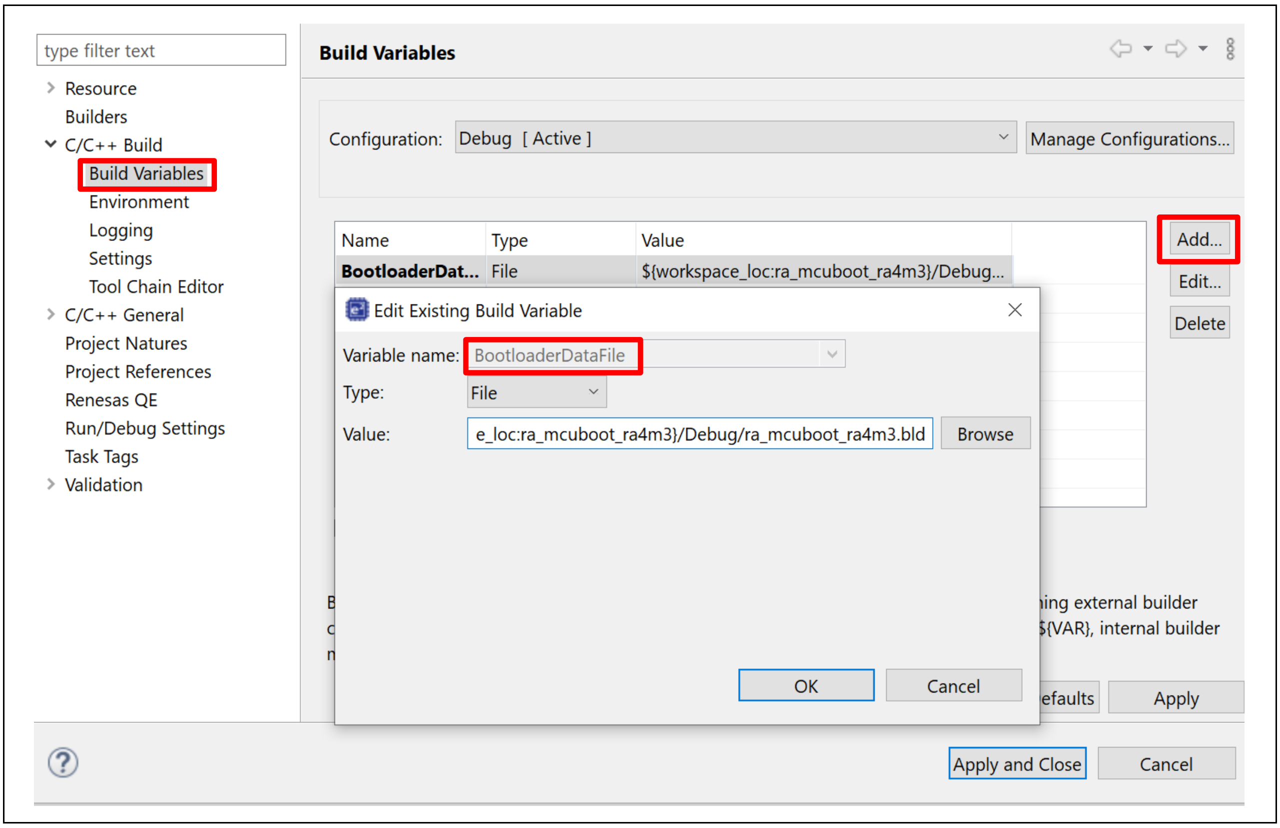

Figure 66. Configure the Build Variable to Use the Bootloader

Follow the same procedure and settings as shown in Figure 66 to configure the other three projects:

- app_ra4m3_ns_primary
- app_ra4m3_s_secondary
- app_ra4m3_ns_secondary

5.3 Signing the Existing Application Projects to Use the Bootloader

The signing command for the application image will be automatically generated when the bootloader is compiled. In the **Project Explorer**, navigate to the <boot_project > debug > <boot_project > .bld file. The signing command is under the section <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 select **Generate Project Content** again to bring in the updated .bld file.

Each application can have a defined version number. This version number can be used in the Overwrite Upgrade mode when **Downgrade Prevention** is **Enabled**. This is achieved by defining an Environment Variable: MCUBOOT_IMAGE_VERSION. If there is signature verification, then it is necessary to set the Environment Variable: MCUBOOT_IMAGE_SIGNING_KEY.

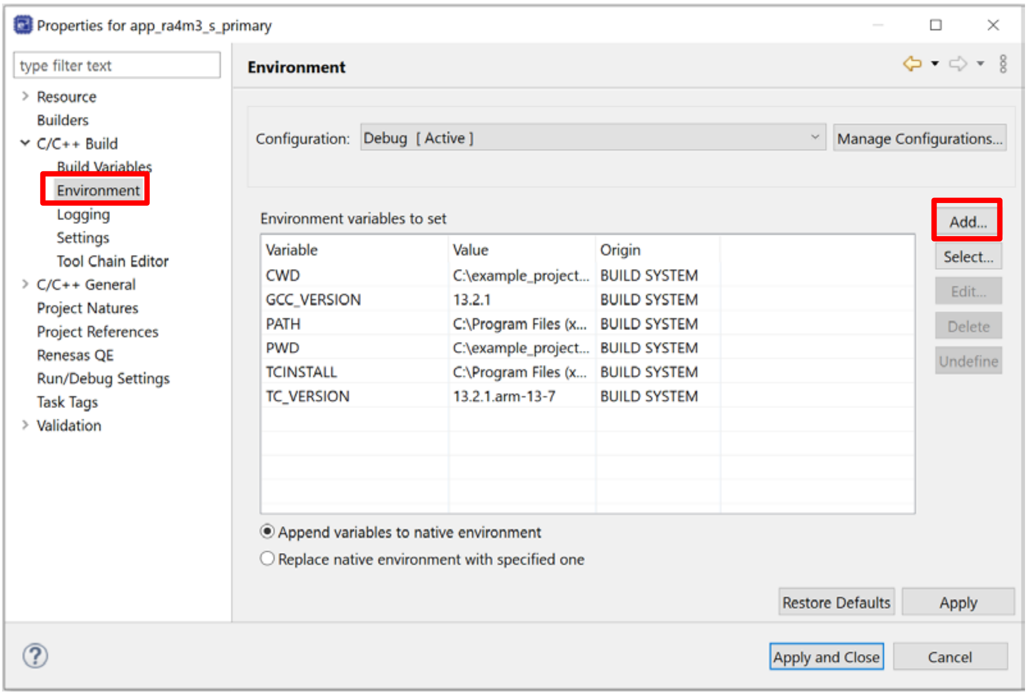

Figure 67. Add New Environment Variable

Add Environment variable for the application image version.

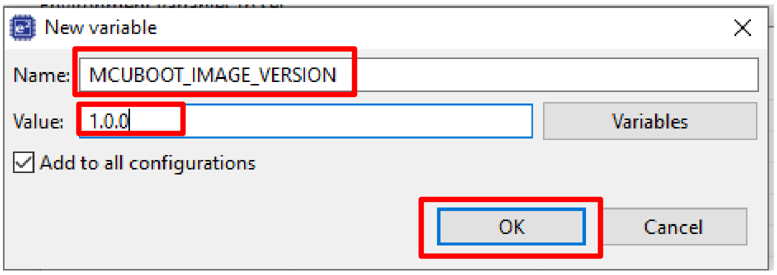

Figure 68. Add MCUBOOT_IMAGE_VERSION Variable

Add an Environment variable to configure the application image signing key with the value `${workspace_loc:ra_mcuboot_ra4m3}/ra/mcu-tools/MCUboot/root-ec-p256.pem`.

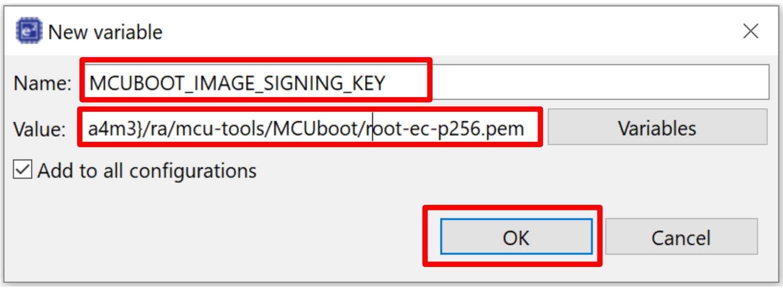

Figure 69. Add MCUBOOT_IMAGE_SIGNING_KEY Variable

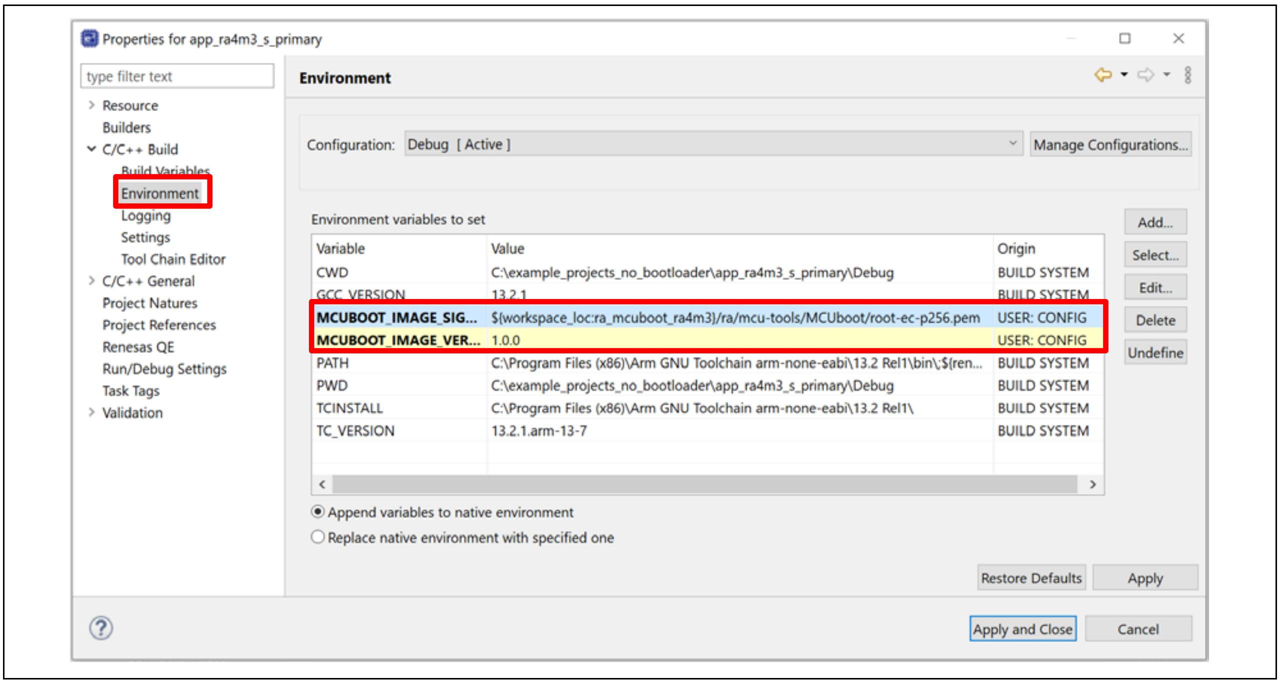

Figure 70. Configure the Signing Key and Application Version

Note: The private key used for signing the application image is indicated in the signing command. `/ra/mcu-tools/MCUboot/root-ec-p256.pem` is used for the example bootloader. This key is used for testing purposes only. For real world use case and production support, you **MUST** change this to the private key of their choice.

To be able to always recompile the project when the environment variables or the linker script are updated, we recommend adding a **Pre-build step** to always delete the `.elf` file with the command

`rm -f ${ProjName}.elf` as shown in Figure 71.

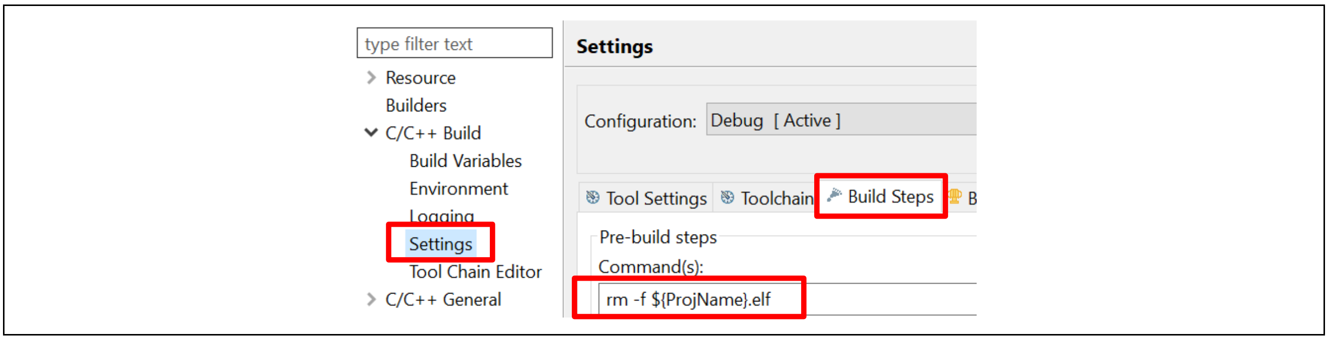

Figure 71. Configure the Pre-build Command

Follow the same procedure to configure the other three projects:

- `app_ra4m3_ns_primary`
- `app_ra4m3_s_secondary`
- `app_ra4m3_ns_secondary`

5.3.1 Click Generate Project Content and Compile All Four Application Projects

For both Primary and Secondary applications, compile the Secure application first and then the Non-Secure application.

5.3.2 Configure the debug configuration

1. Open the **Debug Configurations**: **app_ra4m3_s_primary > Debug As > Debug Configurations**
   Make sure that **app_ra4m3_s_primary Debug** is selected and select the **Startup** tab.

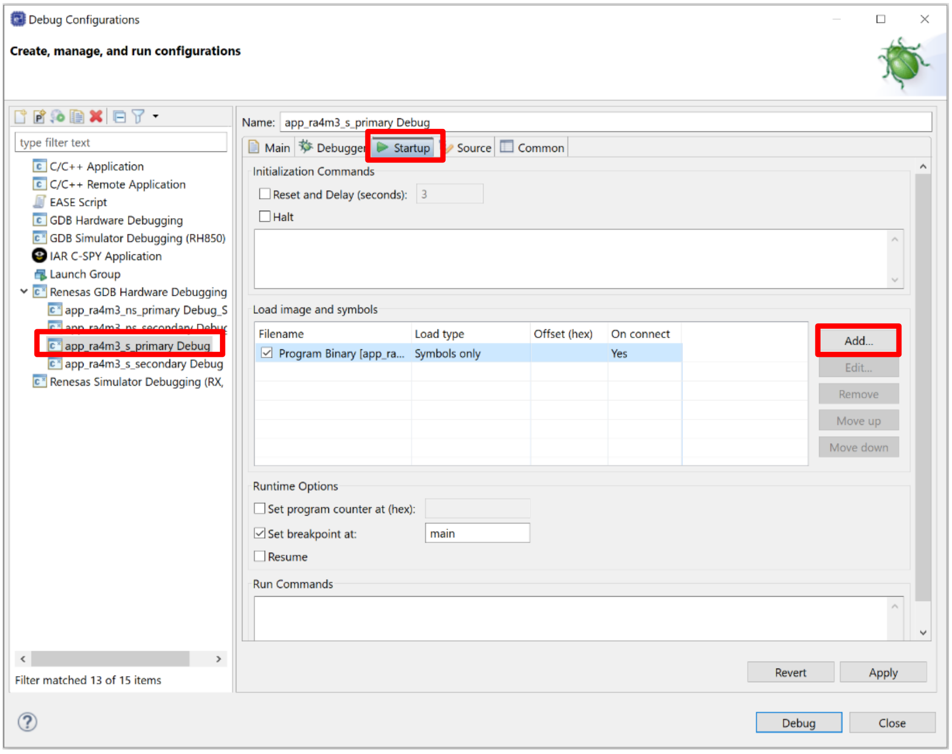

Figure 72. Configure the Primary Secure Project Debug Startup

2. Set up the **Debug Configurations**.
   Click **Add…** and then **Workspace**. Navigate to the **ra_mcuboot_ra4m3** project and select the **ra_mcuboot_ra4m3.elf** file from the debug folder. Click **OK**.

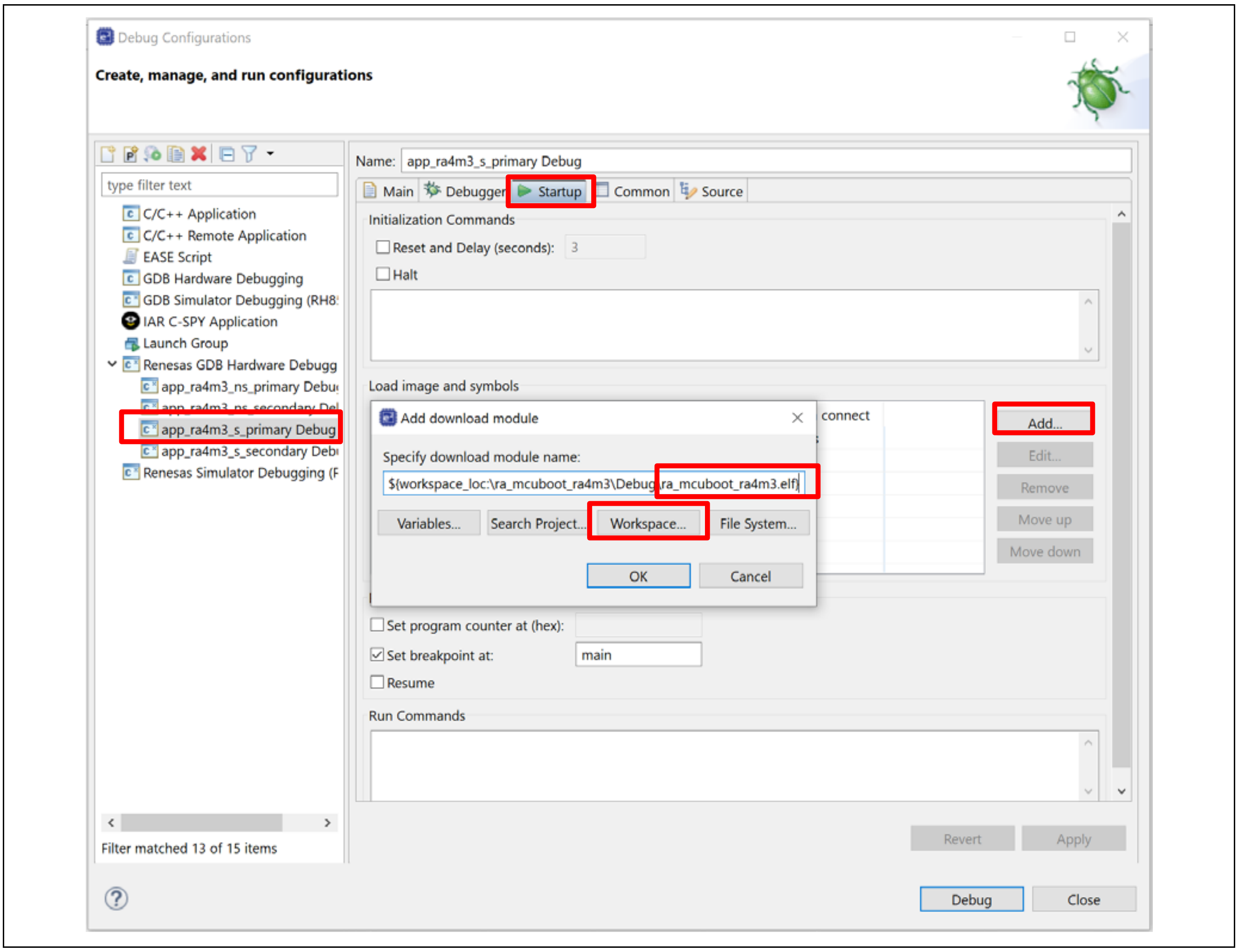

Figure 73. Add the Bootloader Project

Click **Add** again and add the **app_ra4m3_ns_primary** project binary **app_ra4m3_ns_primary.elf** as in the prior step. Click **OK**.

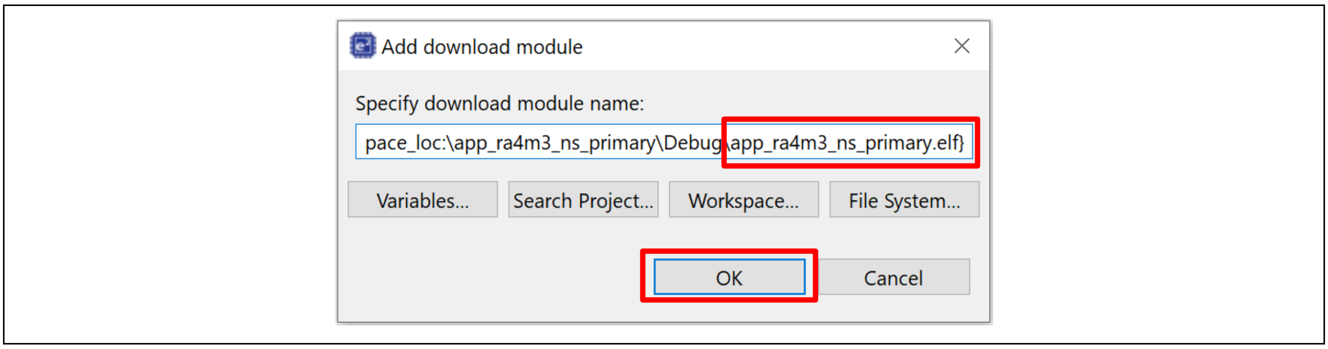

Figure 74. Add the Non-Secure Project

Change the load type of the Program Binaries for the **app_ra4m3_ns_primary** and **app_ra4m3_s_primary** to **Symbols only** by clicking on the cell for load type and selecting **Symbols only** from the drop-down menu.

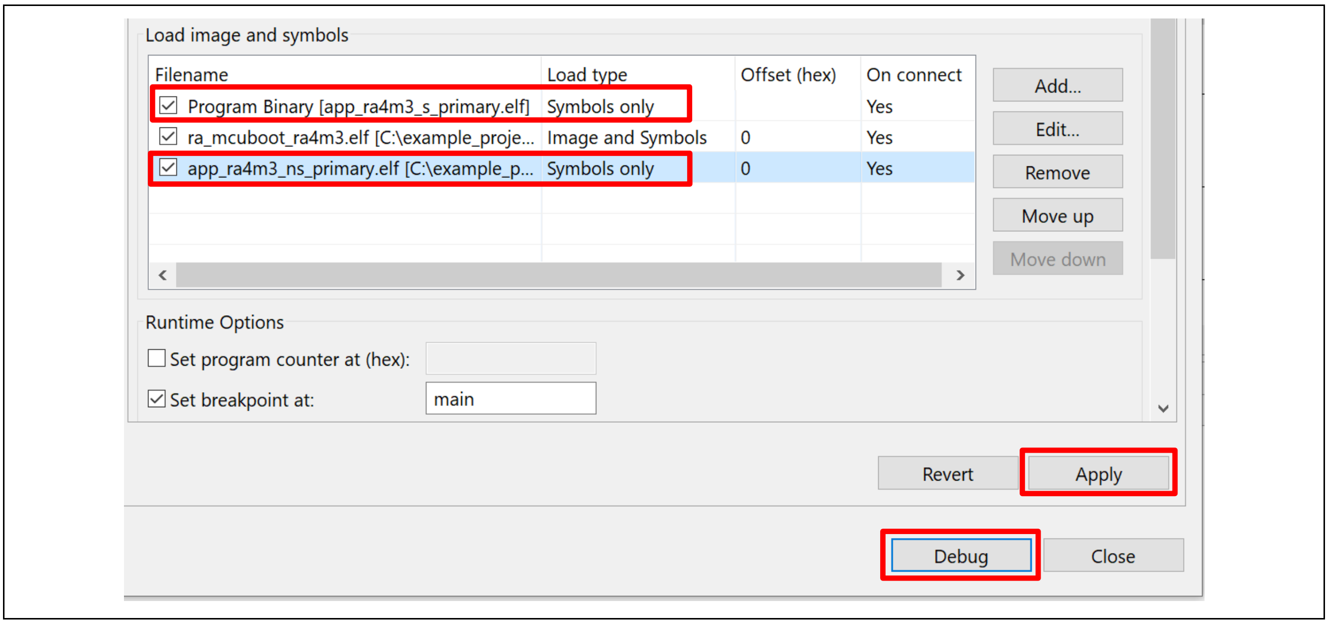

Figure 75. Select to load Symbols only for the Secure and Non-Secure Project

3. Add the signed binary image to the download options using Raw Binary Load type.

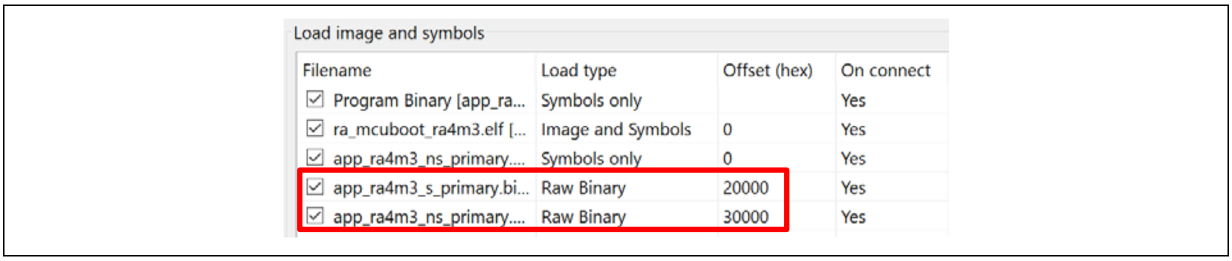

Figure 76. Load the Signed Images

Note that for different update modes and different application images, the load address needs to be update. For the example projects included in this application project, you can reference the memory configuration images included in Figure 57 to Figure 61 to set up the load address.

4. After the above is set up, follow section 3.3 to run the projects if Overwrite Update mode is used or follow section 3.4 to run the projects if Swap Update mode is used.

5.4 Mastering and Delivering a New Application

Mastering and delivering a new application involves similar steps described above in section 4.2 and section 5.2. Typically, the following aspects must be considered in the designing of delivering new applications:

1. Create the new application and sign the new application by following the steps below:
   a) Refer to the *RA Family MCU Security Design with TrustZone with IP Protection Application Project* for new project creation with TrustZone support.
   b) Refer to section 4 to configure the new application to use the bootloader and sign the new application.
2. Download the new application to the Secondary slots.

   This step varies based on the downloading method selected by each user. In this application project, the Ancillary file download capability from e^2 studio is used for demonstration purposes. You can use this method as a testing tool when developing a customized new image downloader. Application Projects R11AN0570 and R11AN0576 include image downloader examples using XModem over COM port and can be used for reference.

6. Appendix

6.1 Making the Bootloader for Cortex-M33 Immutable

To make the bootloader immutable, you must lock the flash blocks containing the bootloader from being programmed and erased.

The RA4M3 features two sets of registers which facilitate flash block locking. Block Protect Setting (BPS) Registers feature bits that map to individual flash blocks. When a bit is set to zero, the corresponding flash block cannot be erased or programmed. The Permanent Block Protect Setting (PBPS) Registers have a similar bit mapping to flash blocks. When a bit is set in one of these registers, the corresponding flash block is permanently locked from being erased and programmed if the same bit in the Block Protect Setting Register is also cleared to zero. This process is irreversible. Once a flash block is permanently locked, it cannot be unlocked again.

Based on the example bootloaders provided in this application project, the flash blocks used by the bootloader are:

- RA4M3 Overwrite Mode: block 0-7
- RA4M3 Swap Mode: block 0-8

Refer to the *RA Family MCU Securing Data at Rest using TrustZone Application Project* to understand the operational flow of setting up the Flash Block Protection.

Note that ticking the BPS0 and PBPS0 Flash Block settings will permanently lock the flash blocks. This CANNOT be reversed. Further details can be found in sections 6.2.5 and 6.2.6 of the RA4M3 Hardware User's Manual.

6.2 Device Lifecycle Management for Renesas RA Cortex-M33 MCUs

Once the bootloader development is finished, you may want to transition the Device Lifecycle State of the RA Cortex-M33 MCU to lock down the debugger and the serial programming interface.

We recommend referring to the Device Lifecycle State Transitions in the Production Flow section in the *Renesas RA Family MCU Device Lifecycle Management Key Installation Application Note* to understand the device lifecycle management options during production.

The operational overview of how to use Renesas Flash Programmer to perform these transitions is explained in the Overview of Device Lifecycle State Transitions using Renesas Flash Programmer section.

7. References

1. Renesas RA Family MCU Securing Data at Rest using Security MPU and Flash Access Window Application Project (R11AN0416)
2. Renesas RA Family MCU Securing Data at Rest using Arm TrustZone Application Project (R11AN0468)
3. Renesas RA Family MCU Device Lifecycle Management Key Installation Application Note (R11AN0469)
4. Renesas RA Family MCU Security Design with TrustZone – IP Protection (R11AN0467)
5. Renesas RA Family RA2 MCU Secure Bootloader Design using MCUboot (R11AN0516)
6. Renesas RA Family MCU Secure Bootloader Design using Dualbank and MCUboot (R11AN0570)
7. Renesas RA Family MCU Booting Encrypted Image using MCUboot and External Flash Memory (R11AN0868)

8. Website and Support

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

EK-RA4M3 Resources	renesas.com/ra/ek-ra4m3
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	Apr.24.24	-	First release document.

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.

Corporate Headquarters

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

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/.