Renesas RA Family

Using MCUboot with Dual core RA8 MCUs

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 TrustedFirmware 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 multicore application project creation using the MCUboot Module on Renesas EK-RA8P1 kit using FSP v6.4.0. Users can follow the steps provided to recreate the bootloader and link the application projects to work with the bootloader using FSP Solution. It also covers procedures for configuring an application to work with bootloader, and for downloading and updating a new application image.

Required Resources

Development tools and software

- The e^2 studio IDE v2025-12
- Renesas Flexible Software Package (FSP) v6.4.0

Note: The above software components are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

- Python v3.12 or later (https://www.python.org/downloads/)

Target Devices

Below are the Renesas MCU products to which the information within this document is applicable:

- RA8P1 MCU Group (https://www.renesas.com/en/products/ra8p1)
- RA8M2 MCU Group (https://www.renesas.com/en/products/ra8m2)
- RA8D2 MCU Group (https://www.renesas.com/en/products/ra8d2)
- RA8T2 MCU Group (https://www.renesas.com/en/products/ra8t2)

Hardware

- EK-RA8P1 Evaluation Kit for RA8P1 MCU Group (EK-RA8P1 - Evaluation Kit for RA8P1 MCU Group | Renesas)
- PC running Windows® 11 OS
- One USB device cable (type-A male to type-C male)

Prerequisites and Intended Audience

This application note assumes you have some experience with the Renesas e^2 studio IDE and RA Family Flexible Software Package (FSP). Before you perform the procedures in this application note, follow the procedure in the *FSP User Manual* to build and run the Multicore Blinky project. Doing so enables you to become familiar with the e^2 studio and the FSP and validates that the debug connection to your board functions properly. 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 multi-image 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, and it now resides under TrustedFirmware (www.trustedfirmware.org/projects/mcuboot).

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. Downloading of the new firmware version is typically handled 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 multi-image MCUboot memory map. For more information on the MCUboot memory layout, refer to the Flash Map section of the MCUboot website.

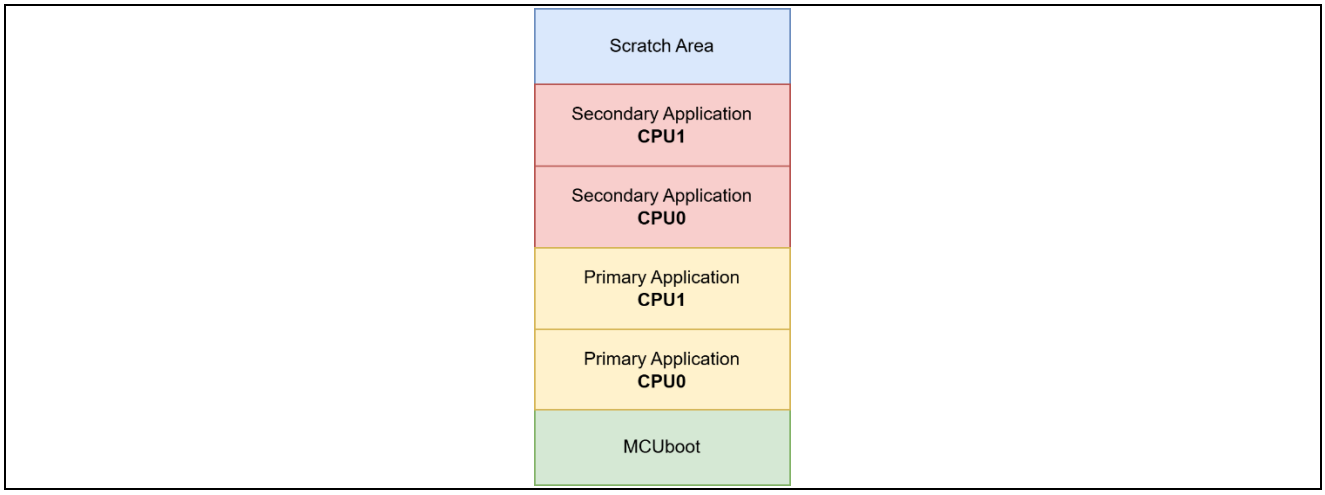

Figure 1. Multi-image MCUboot Memory Map

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

The bootloader starts 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 Application memory marked as to be updated, the bootloader performs the following actions:

1. The bootloader authenticates the Secondary Application 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 Application 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, ECDSA or ML-DSA. The firmware image is authenticated by hash (SHA) 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 both internal and 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.

- **Swap**
  In the Swap image upgrade mode, the active image is 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 permanently.
    - This image upgrade mode is fail-safe and resistant to power-cut failures.
    - Encrypted image support is available when using both internal and 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, thus 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.

- **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. For RA8x2 MCUs, the dual bank mode is not available.

- **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 typically not 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).
  Please note that 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. It describes the available upgrade modes, signature options, and covers dual core use cases with MCUboot.

2.1 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. From FSP v6.4.0, the Direct XIP method is not supported for Trustzone based setup.

We recommend 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.

2.1.1 General Configuration

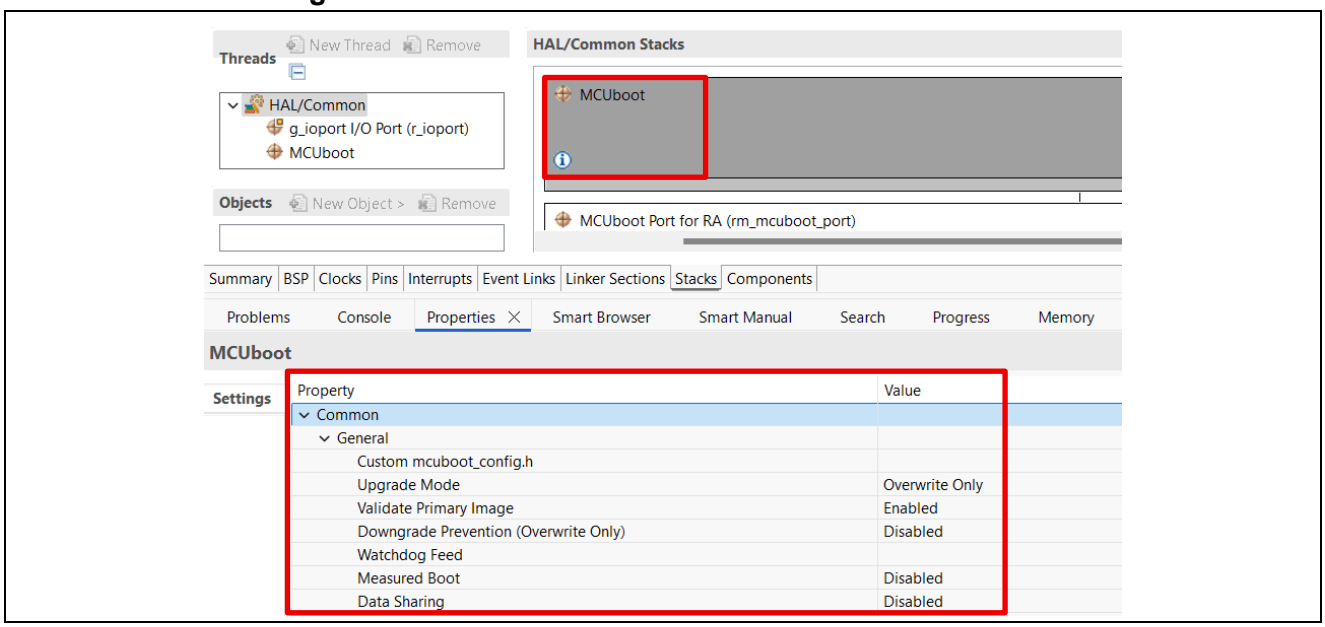

Figure 2. FSP MCUboot Module General Configuration Properties

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 2.1. The options are Overwrite Only, Overwrite Only Fast, Swap, and Direct XIP, as shown in Figure 3. Overwrite Only is the default setting.

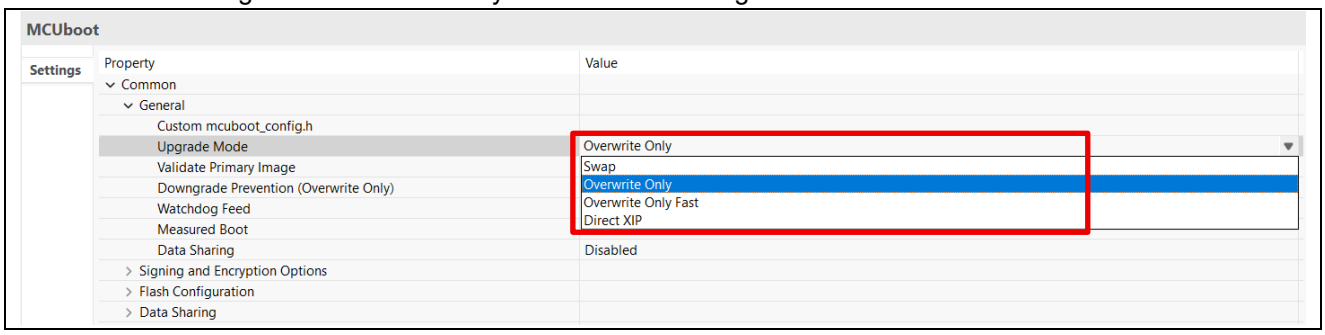

Figure 3. Application Image Update Mode

- **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, 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.

2.1.2 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 4 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 MCUboot with Overwrite update mode project uses a MRAM area of 50 KB when using ECDSA P-256 signature type, but when signature support is not used, the bootloader size reduces to about 20 KB.

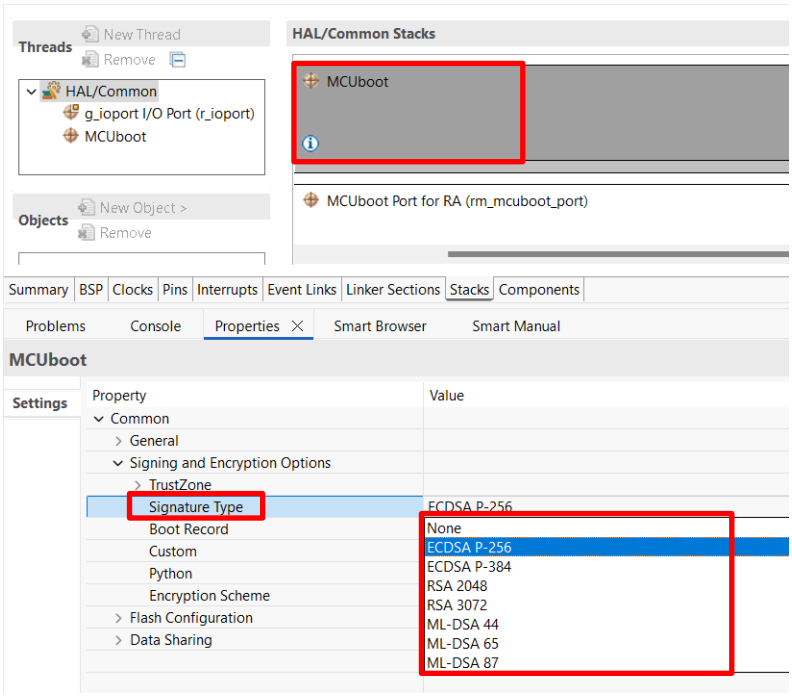

Figure 4. Application Image Signature Type for FSP MCUboot Module

MCUboot in FSP also supports image encryption alongside signature verification to provide confidentiality of image data while in transport to the device or while residing in external flash. User can refer to **R11AN0567** application project for more information on using MCUboot with encrypted images. Image encryption is not supported when using the ML-DSA signature type.

2.1.3 Signing Options

Figure 5 shows the default **Custom** signing option configuration provided by FSP.

Figure 5. 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 entered 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 MCUboot Dual core Use Cases

2.2.1 Cortex-M33 is used for application-specific offloading

The RA8P1 features two cores: a Cortex-M85 and a Cortex-M33. In this case, the Cortex-M85 acts as the primary boot and security core, while the Cortex-M33 is used for application-specific offloading.

At power-on reset, the system boots on the Cortex-M85, where MCUboot is executed as the secure bootloader. All cryptographic and security-related operations are handled by the Cortex-M85, including firmware authentication and image integrity verification. The Cortex-M85 is responsible for performing the secure boot process and validating the application images before application execution is allowed. After successfully completing the secure boot and firmware validation sequence, the firmware running on the Cortex-M85 initializes system and then starts the Cortex-M33. The Cortex-M33 subsequently runs the user application or executes offloaded tasks as required by the system.

The Renesas FSP MCUboot integration supports a multi-image secure boot flow, allowing MCUboot on the Cortex-M85 to authenticate multiple firmware images stored in MRAM. This includes both the Cortex-M85 application image and the Cortex-M33 application image.

With this architecture, all secure boot, image authentication, and trust enforcement are centralized on the Cortex-M85, while the Cortex-M33 is used solely for application-specific offloading after the system has entered a verified and trusted state.

The example applications described in Section 3 are demonstrated based on this use case, where the Cortex-M85 manages secure boot and the Cortex-M33 performs application-specific offloading.

3. Running the Example Projects

The example projects demonstrate MCUboot operation with multi-image support in Overwrite Only mode on RA8 dual core series using EK-RA8P1. The figure below illustrates the major events involved in the embedded system design using MCUboot as the secure bootloader in these example projects.

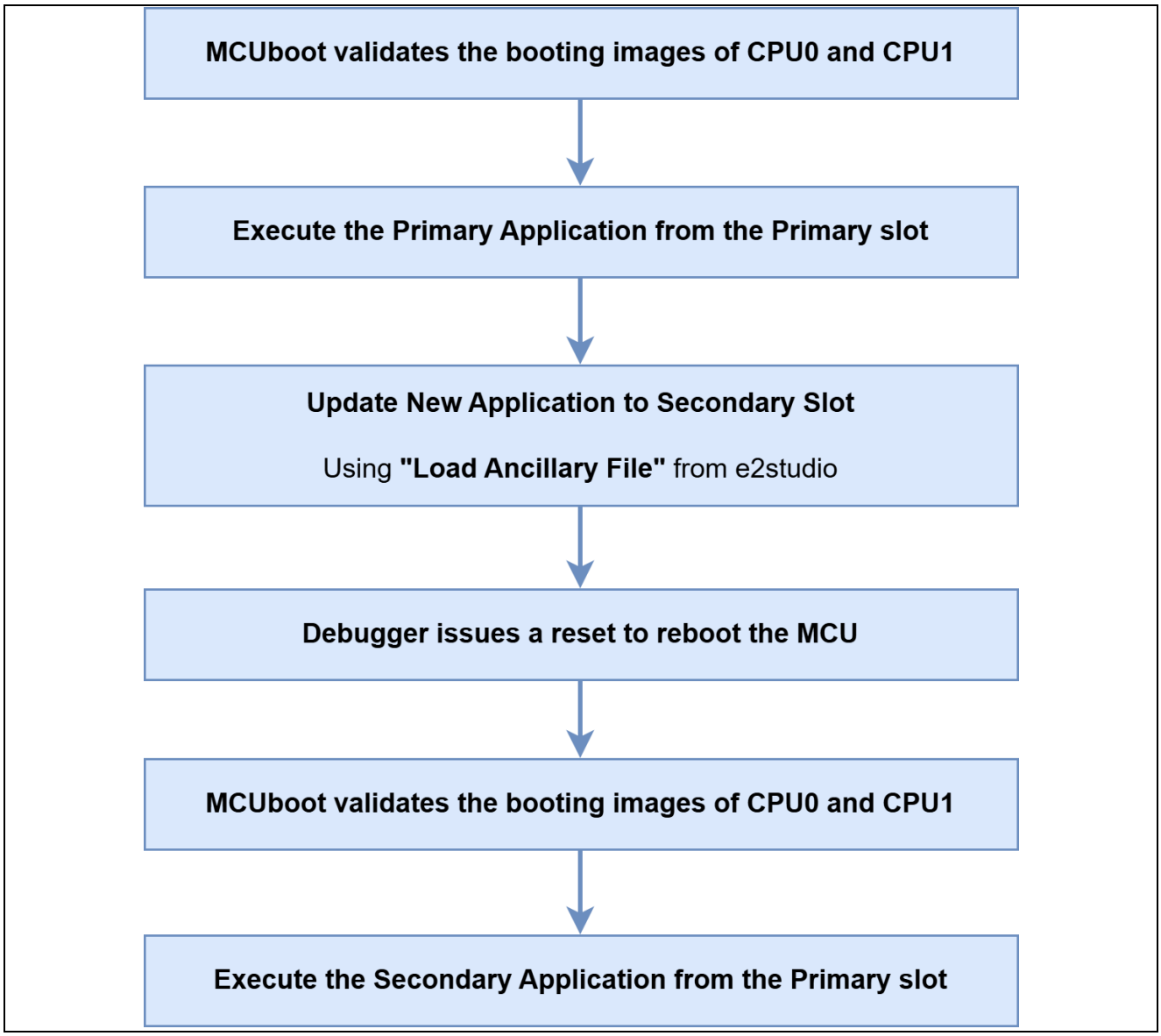

Figure 6. System Overview

Sections 3.1 to 3.2 provide guidance on how to set up the environment, prepare the required hardware, and run the example project.

3.1 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.

Download and Install Python v3.12 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:

- Launch e2 studio and import the set of projects under folder `ra8_dualcore_with_bootloader` into a workspace.
- Choose `ra8p1_bootloader` folder, open the `configuration.xml` file then click **Generate Project Contents**. Navigate `ra8p1_bootloader\ra\mcu-tools\MCUboot` right click and select **Command Prompt**. Depending on your PC policy, administrator privileges may be required when running the Command Prompt. This opens a command window with the path set to the `\mcu-tools\MCUboot` folder.

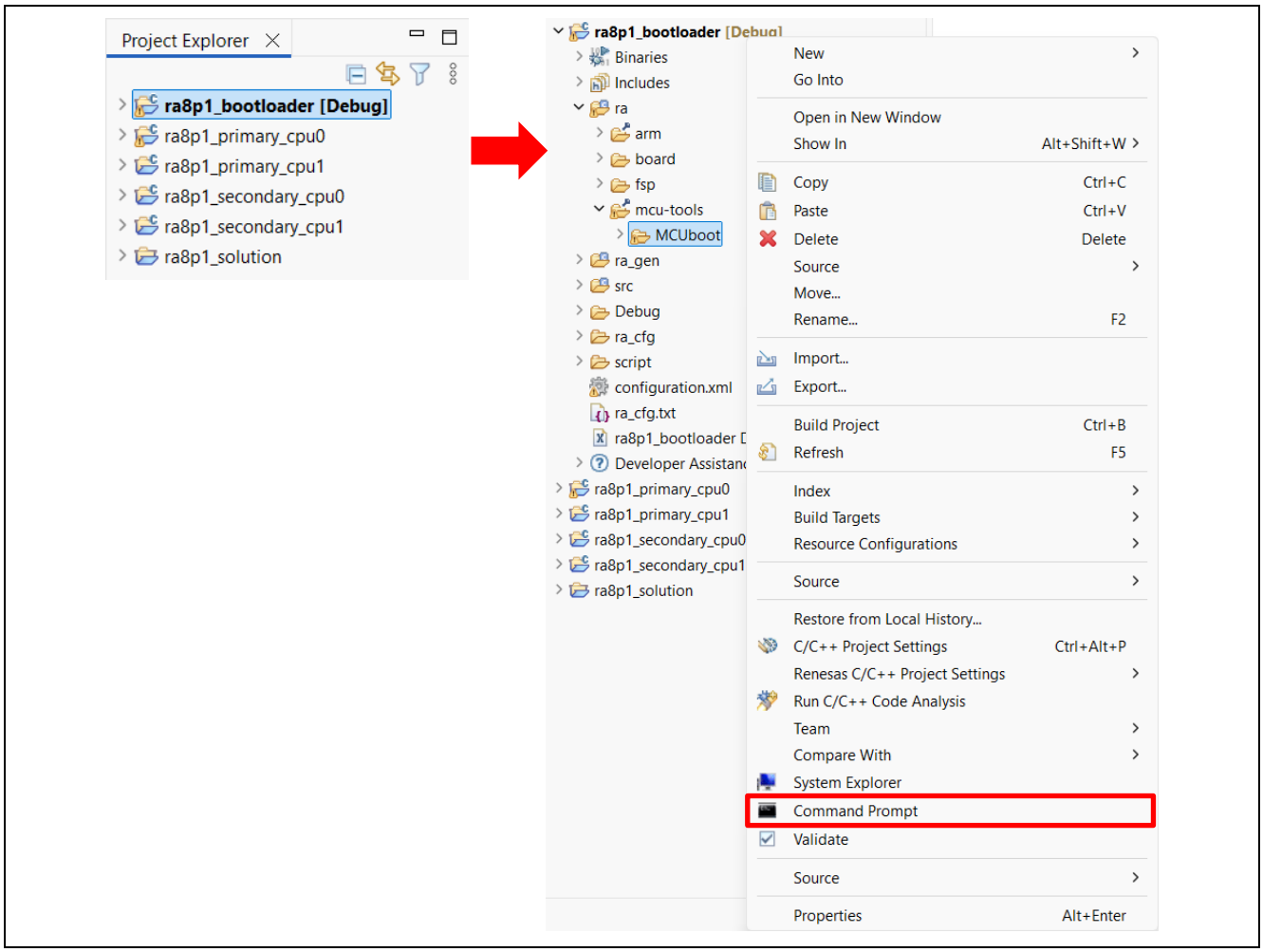

Figure 7. Open the Command Prompt

- 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 3.12.0 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. If your project path contains special characters or spaces, an error may occur when executing the python script.

3.2 Running example project

Follow the steps below to run the example projects on EK-RA8P1 dual-core using the MCUboot Module Overwrite Only Update mode.

3.2.1 Set Up the Hardware

Connect J10 using a USB type-A to type-C cable from EK-RA8P1 to the development PC to provide power and debug connection using the on-board debugger.

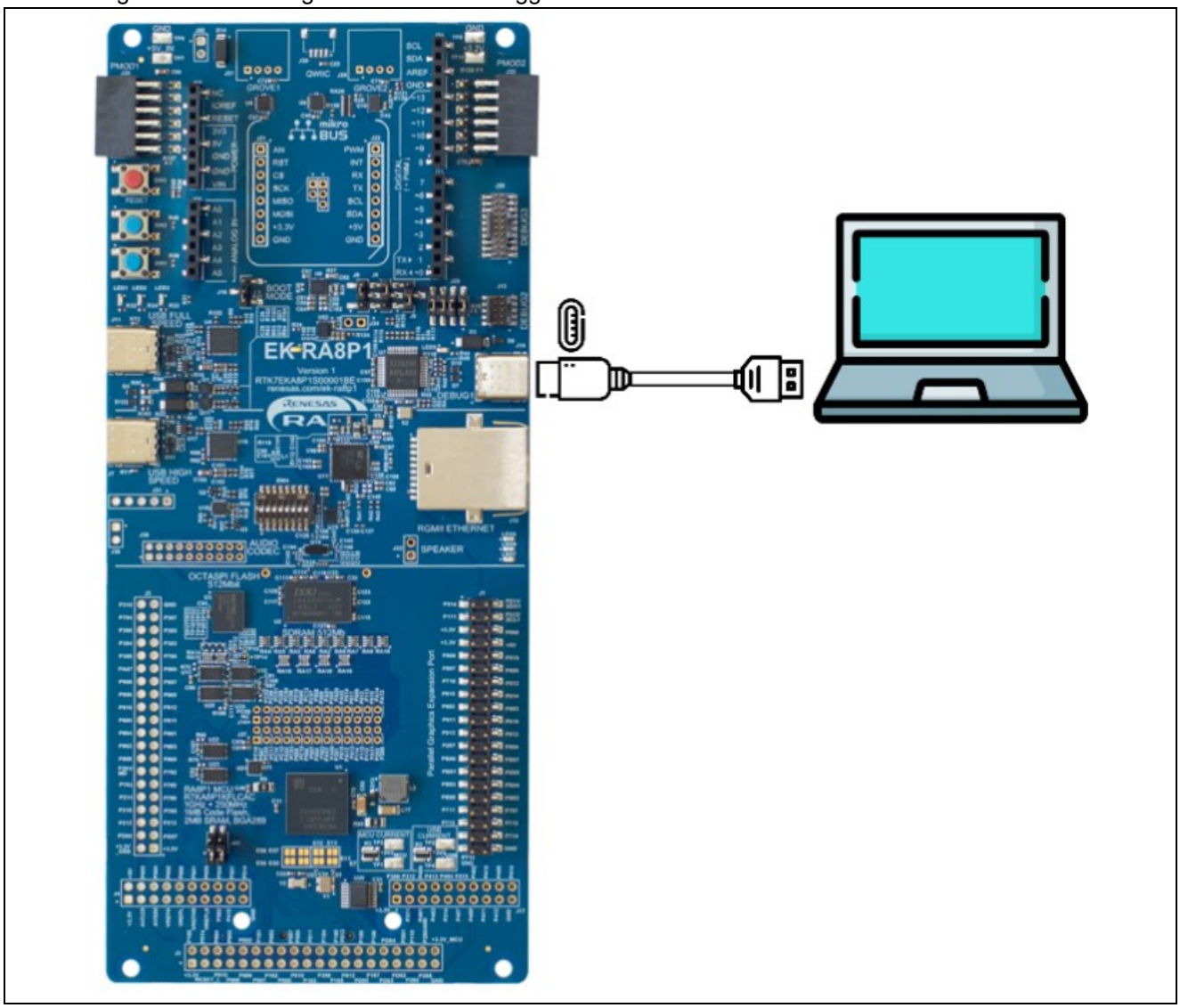

Figure 8. Hardware Setup

Once the EK-RA8P1 is powered up, initialize the MCU prior to exercising the bootloader project. Erase the entire MCU MRAM and ensure the MCU is in Original Equipment Manufacturer (OEM) Device Lifecycle State with PL2 and AL2. 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.

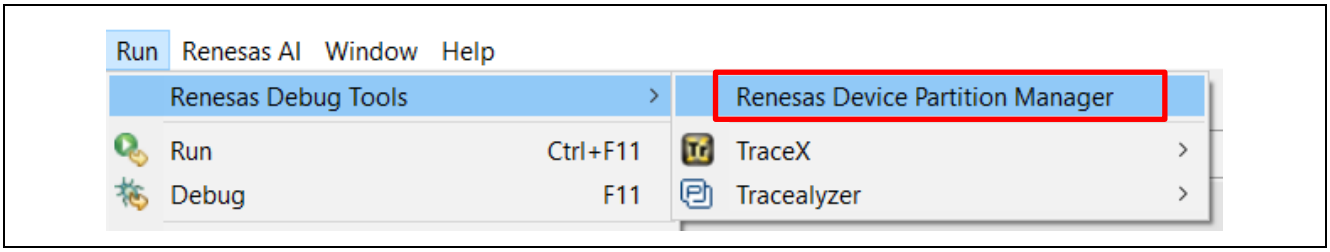

Figure 9. Open Renesas Device Partition Manager

2. Select **Read current device information** and click **Run**.
   If the DLM state is OEM (PL2, PL1 or PL0), 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 RA8P1 MCU that is in the OEM state.

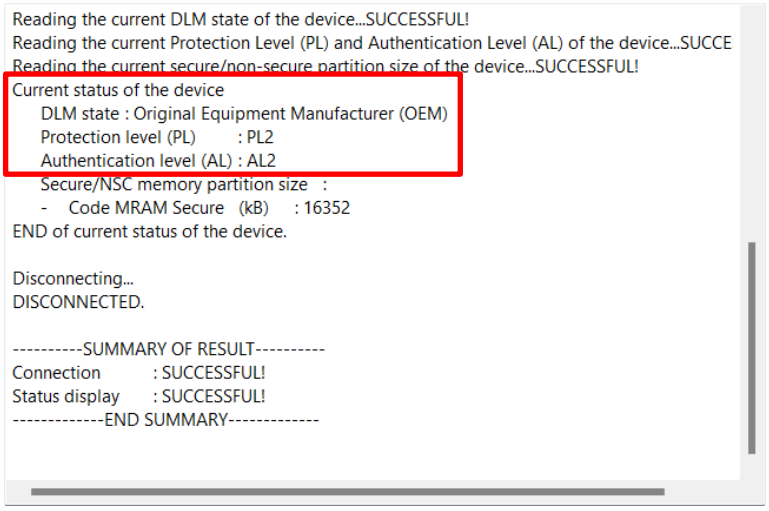

Figure 10. Read the Device Lifecycle States

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

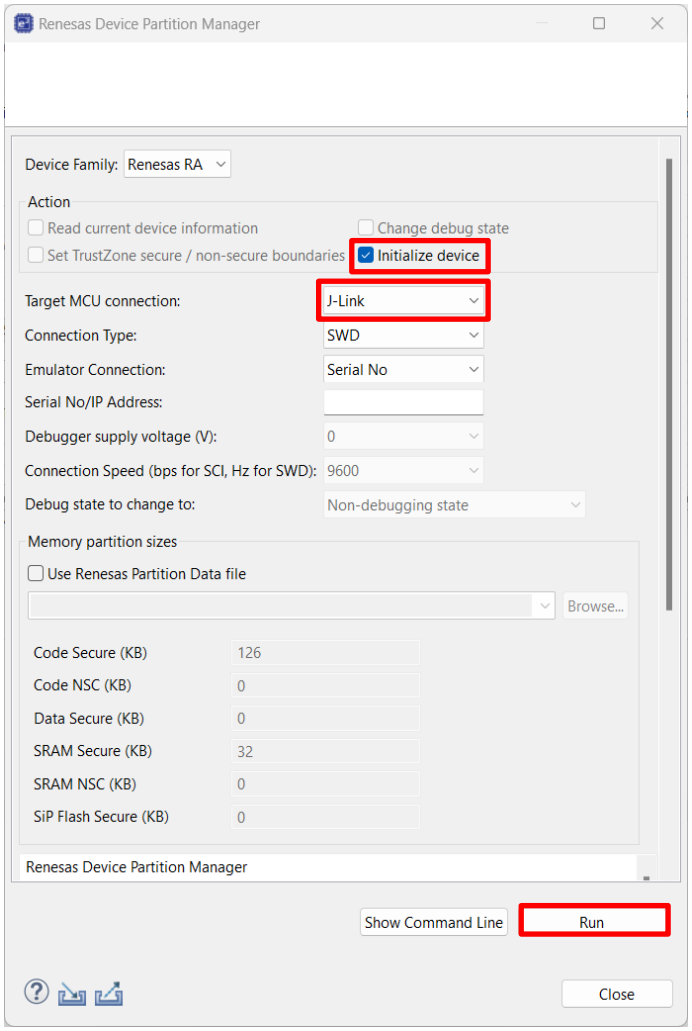

Figure 11. Initialize RA8P1 using Renesas Device Partition Manager

The entire MRAM will be erased if there are not permanently locked down sections. In addition, if the device is in the PL1 or PL0 state, the RA8P1 will be initialized to the PL2 state.

3.2.2 Import the Projects

Launch e2 studio and import `ra8_dualcore_with_bootloader.zip` file to a workspace.

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 3.1.

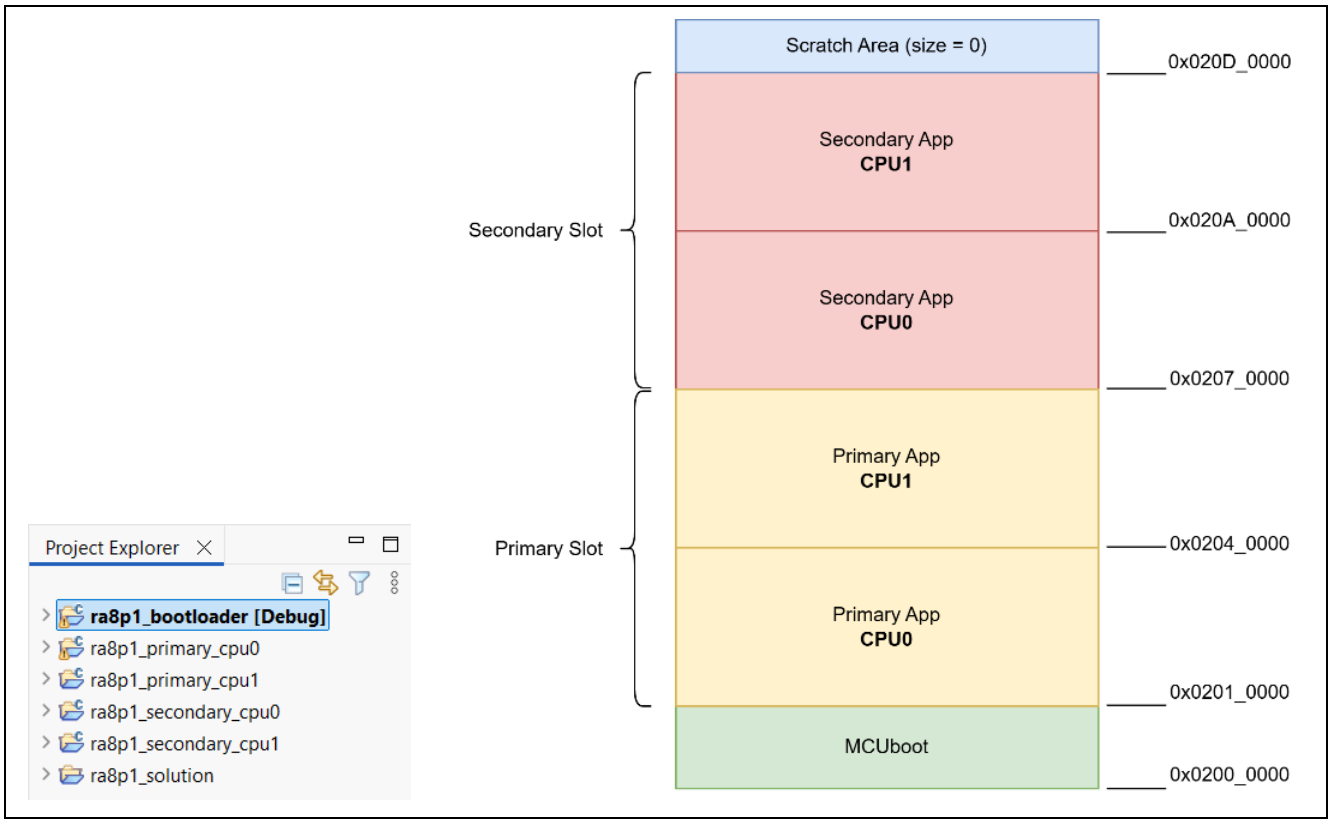

Figure 12. Example Projects for RA8P1 Overwrite Update Mode

- **`ra8p1_bootloader`**: The bootloader project configured with Overwrite update mode.
- **`ra8p1_primary_cpu0`**: The Primary application running on CPU0, programmed into the MRAM code region, is responsible for enabling CPU1 and turning on the blue LED.
- **`ra8p1_primary_cpu1`**: The Primary application running on CPU1, programmed into the MRAM code region, turns on the green LED after CPU1 is enabled by CPU0.
- **`ra8p1_secondary_cpu0`**: The Secondary application image for CPU0 provides the same functionality as the primary application. However, it modifies the LED control behavior by blinking the blue LED, demonstrating an updated application image for CPU0.
- **`ra8p1_secondary_cpu1`**: The Secondary application image for CPU1 provides the same functionality as the primary application but changes the LED behavior to blink the green LED, demonstrating an updated application image for CPU1.
- **`ra8p1_solution`**: The Solution project used to link bootloader with Primary application to define memory map for bootloader, primary slot and secondary slot.

3.2.3 Compile All the Projects

Right click to `ra8p1_solution`, choose **Build Project**.

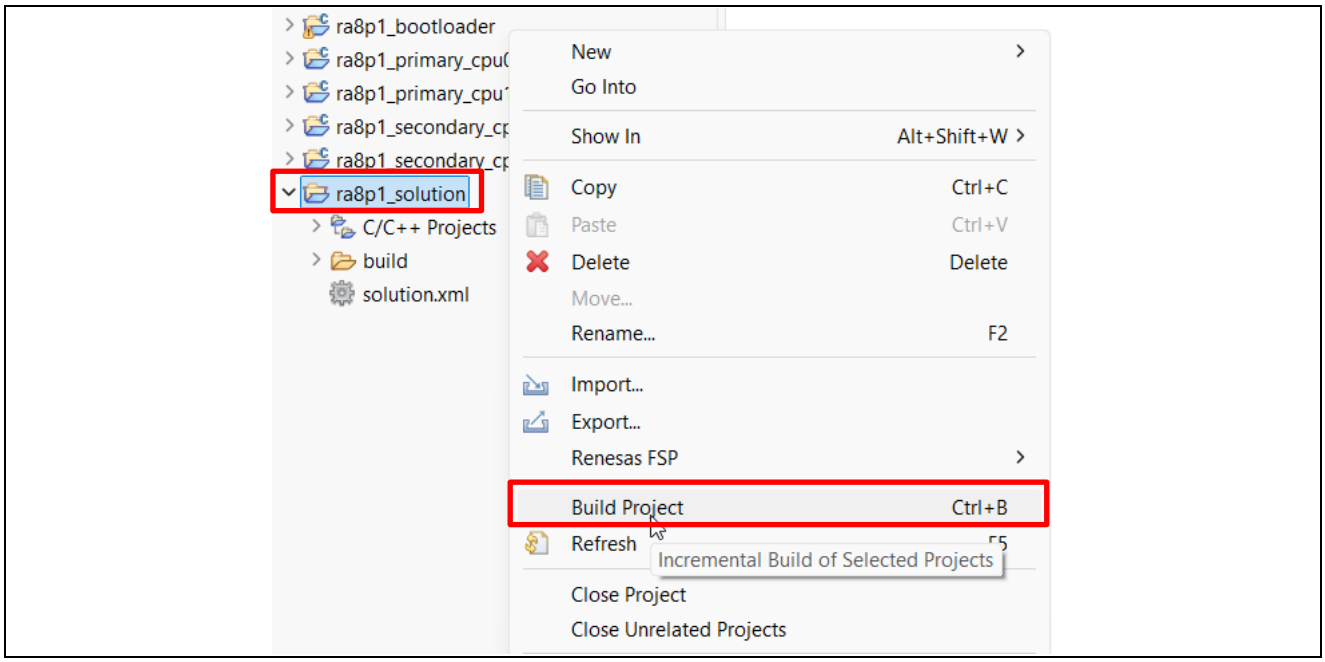

Figure 13. Build Solution project

After selecting **Build Project** for the Solution project, the bootloader project will be compiled first, followed by the primary application project for CPU0 then CPU1.

For secondary projects, users must compile it separately after the solution has been completely built. To build the secondary projects, choose `ra8p1_secondary_cpu0` project, open the `configuration.xml` file, click **Generate Project Contents** and then build the secondary project for CPU0. Repeat the same steps for the secondary project for **CPU1**.

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, `/ra8p1_secondary_cpu0/Debug/ra8p1_secondary_cpu0.bin.signed`).

3.2.4 Debug the Applications and Boot the Primary Applications

Although it is possible to start a debug session for CPU0 and then manually start a second debug session for CPU1, it is more convenient to let e² studio handle both operations together. This is achieved by using a launch group, which is then used to start a multicore debug session, instead of launching the individual debug configurations separately.

Set up the “Multicore” launch configuration to connect to CPU0 by right-clicking on `ra8p1_primary_cpu1` project and selecting **Debug As > Debug Configurations > Renesas GDB Hardware Debugging > ra8p1_primary_cpu1 Debug_Multicore > Start up** then confirming the following configuration information:

- The bootloader is downloaded using the `.elf` format, Load type as Image and Symbols.
- The Primary CPU0 and CPU1 images (`ra8p1_primary_cpu0.bin.signed`, `ra8p1_primary_cpu1.bin.signed`) are downloaded using the signed binary as Raw Binary.
- The Primary image symbol for CPU0 is included using the .elf files.

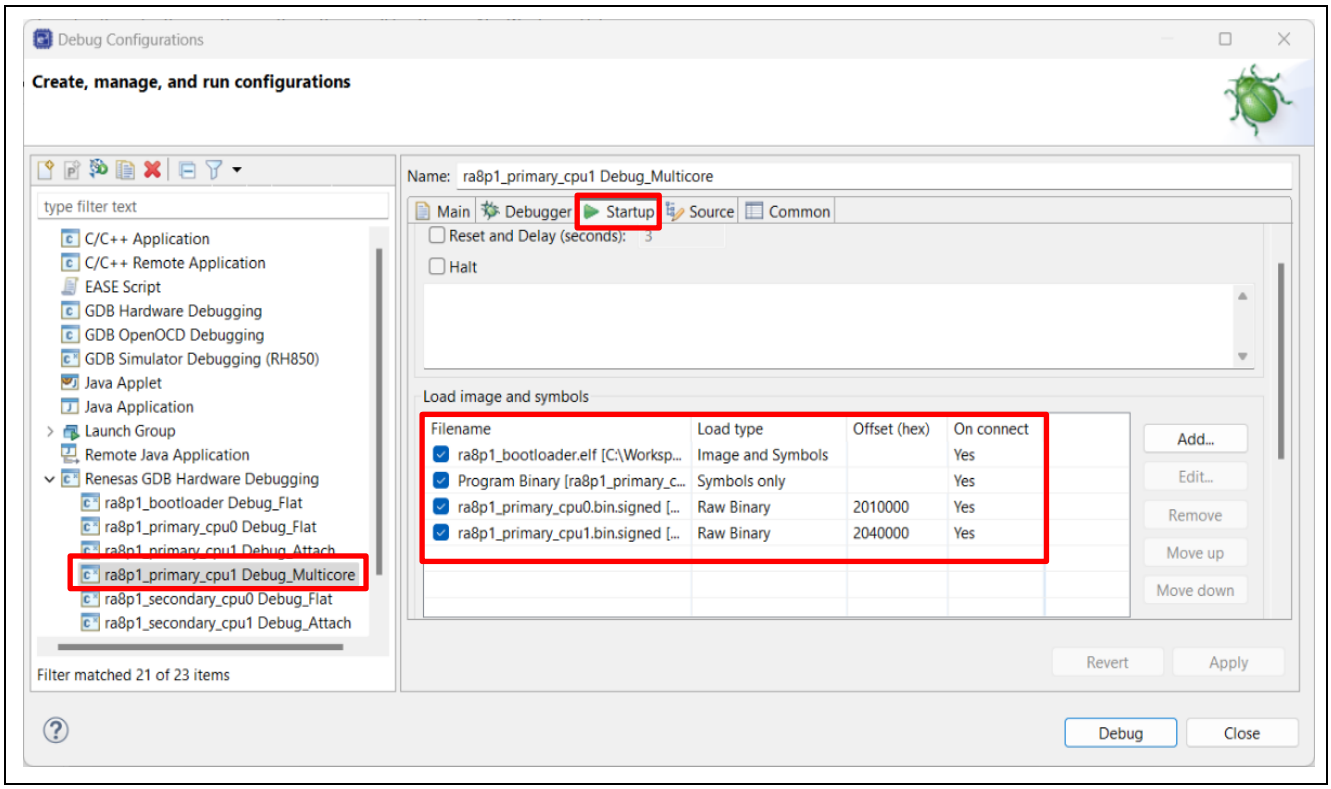

Figure 14. Debug_Multicore Configurations on EK-RA8P1

Navigate to **ra8p1_primary_cpu1 Debug_Attach** and confirm that the Primary image symbol for CPU1 is included by using the .elf file, as shown in figure below to configure the connection to CPU1.

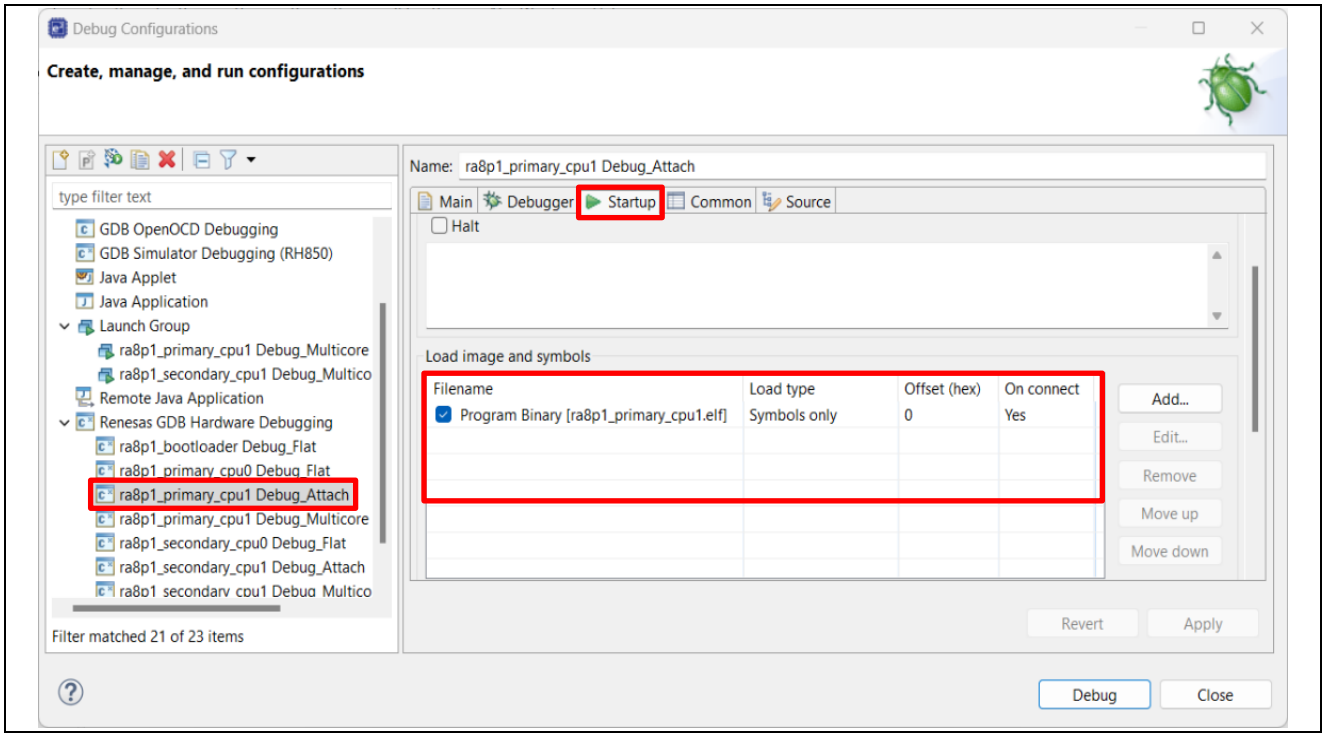

Figure 15. Debug_Attach Configurations on EK-RA8P1

Choose **Launch Group > ra8p1_primary_cpu1 Debug_Multicore Launch Group** and confirm 2 launch configurations set up as in Figure 14 and Figure 15 are selected then click **Debug**.

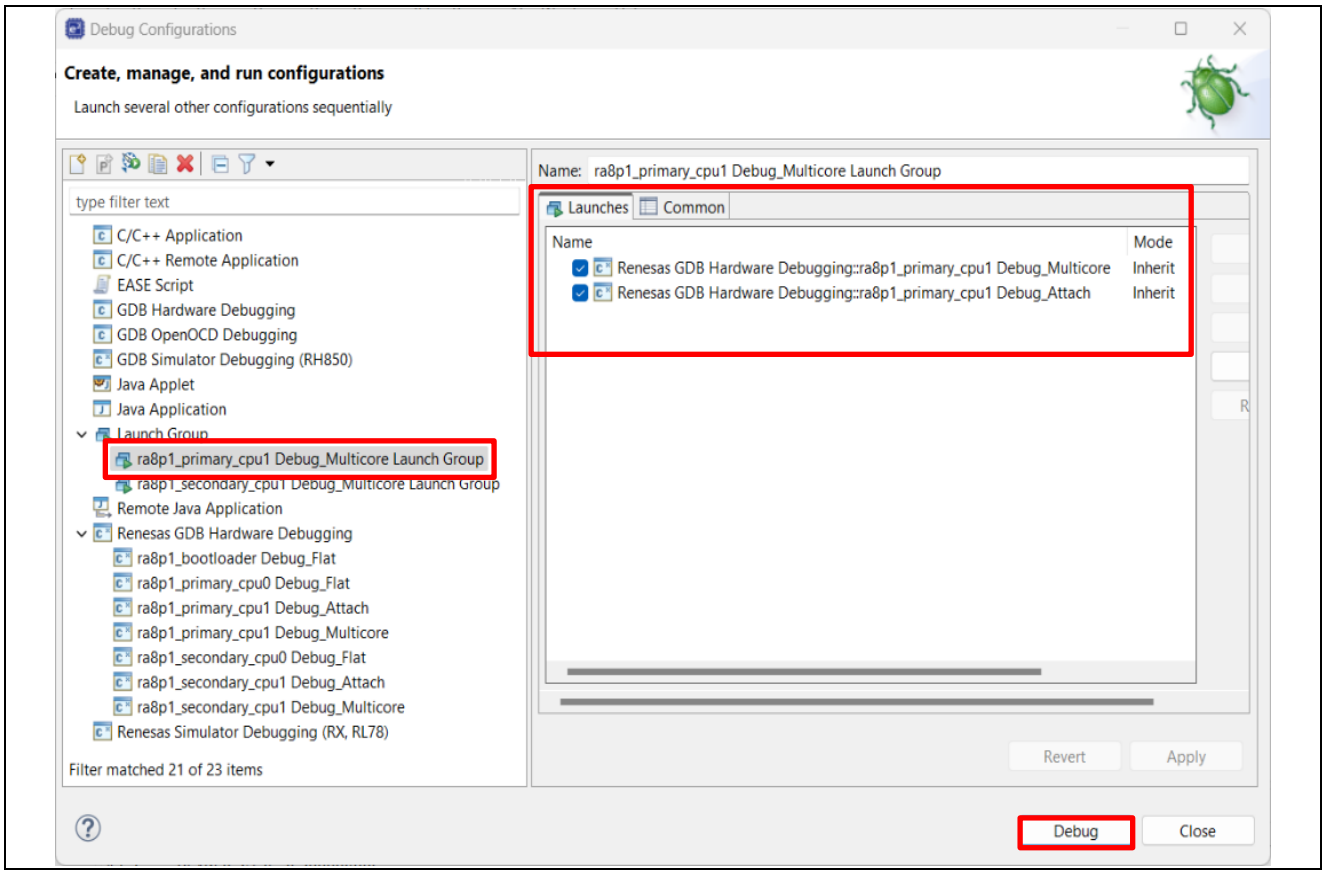

Figure 16. Debug_Multicore Launch Group Configurations on EK-RA8P1

The debugger should hit the reset handler in the bootloader.

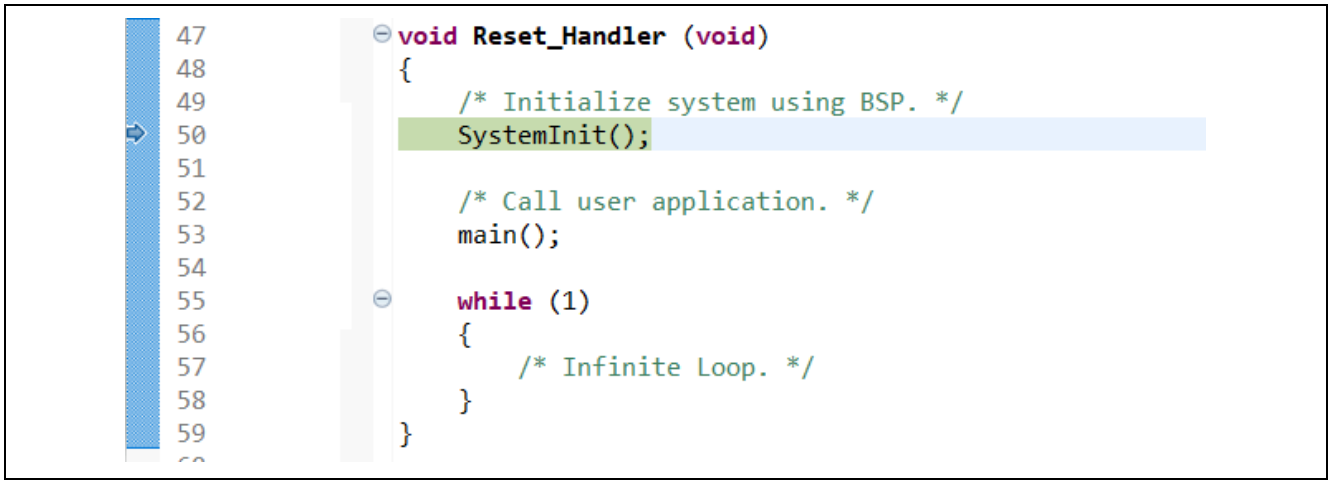

Figure 17. Start the Application Execution

Click **Resume** twice to boot the Primary image. The primary application runs first, and the blue LED should turn on.

The debugger then switches to CPU1 debug session and stops at the reset handler of the primary image on CPU1. Click again to run the application on CPU1, and the green LED should turn on.

3.2.5 Downloading and Running the Secondary Applications

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

1. Press the button to pause programs on both CPU0 and CPU1 debug sessions.
2. On the top of the e² studio toolbar, click the Load Ancillary File button to load the new application images to the Secondary slot region.

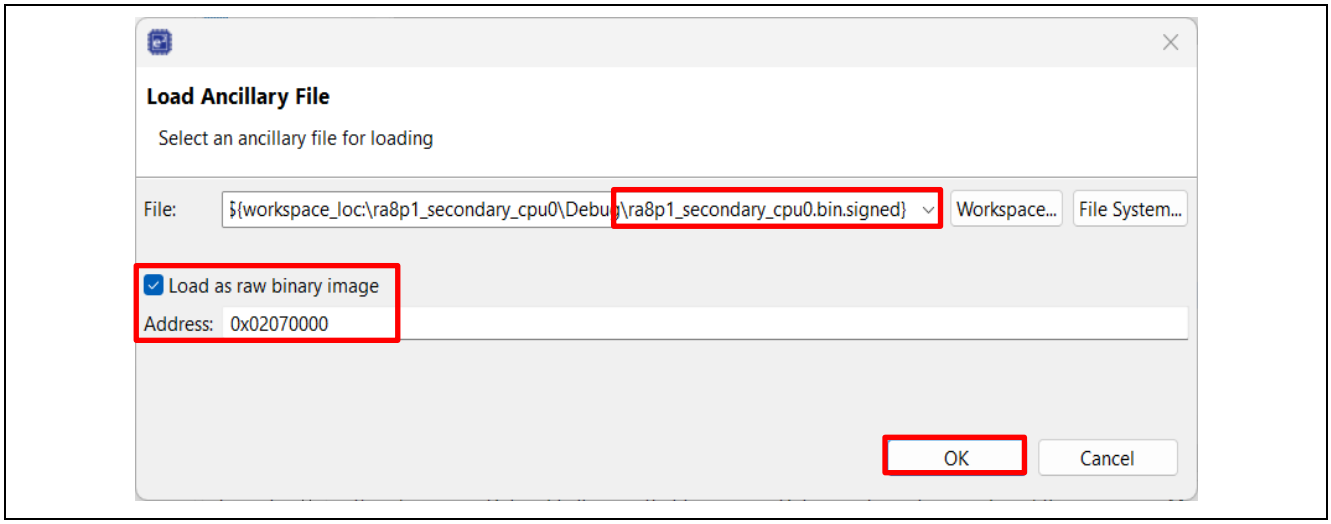

Figure 18. Load the CPU0 Secondary Application Image

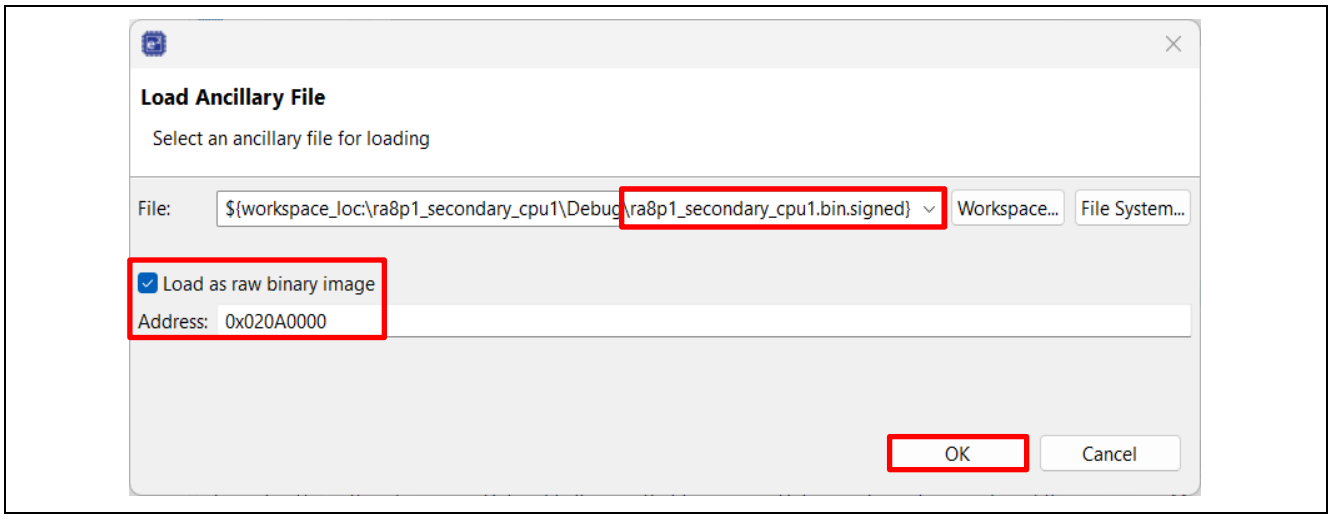

Figure 19. Load the CPU1 Secondary Application Image

3. Click **Resume**. The overwrite occurs and the new images for CPU0 and CPU1 are executed. The blue and green LEDs will blink instead of remaining on.

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. The screen captures used in these sections are based on the RA8P1 based bootloader projects used in section 3.2.

4.1 Start Bootloader Project Creation with e² studio

Follow the steps below to create the initial bootloader project based on EK-RA8P1:

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 `ra8p1_bootloader` 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 6.4.0,** the **EK-RA8P1** board and **Core** is **CPU0**. Use the default **Debugger** setting **J-Link Arm**, **Toolchain** is **LLVM** version 21.1.1 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.

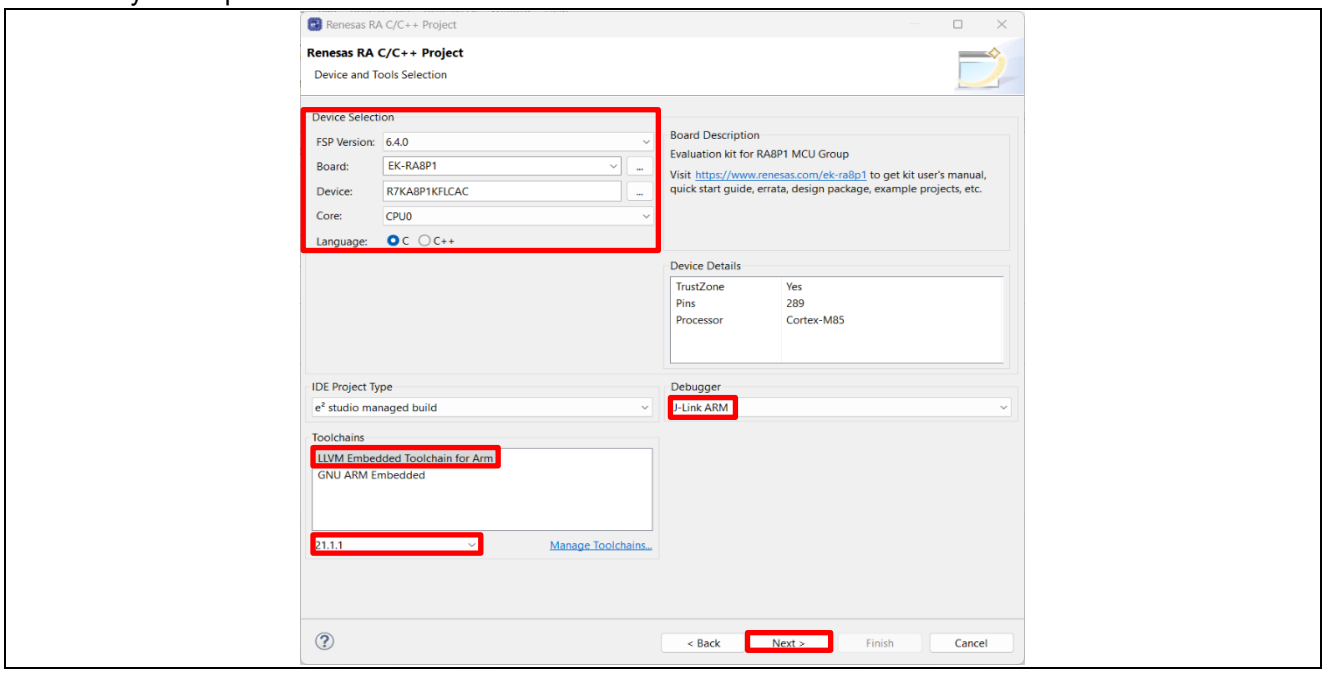

Figure 20. Device and Tools Selection

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

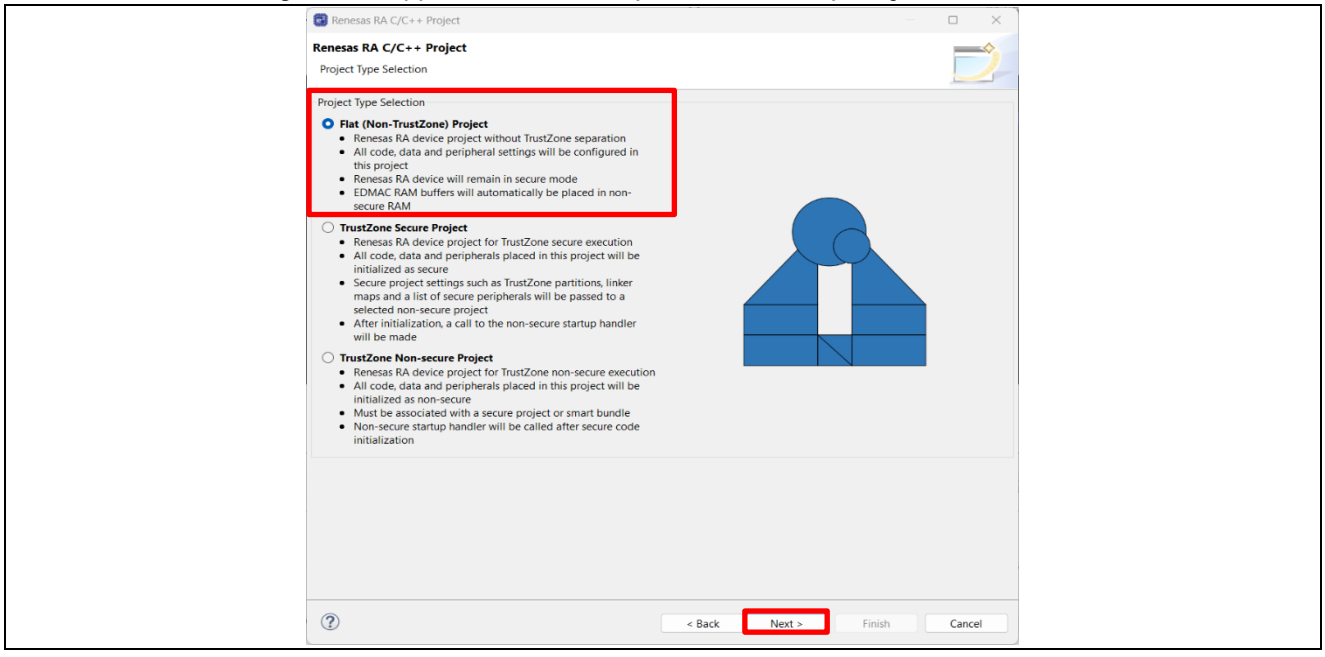

Figure 21. Choose Flat Project as Project Type

3. Choose **None** for Preceding Project or Smart Bundle Selection and click **Next**.

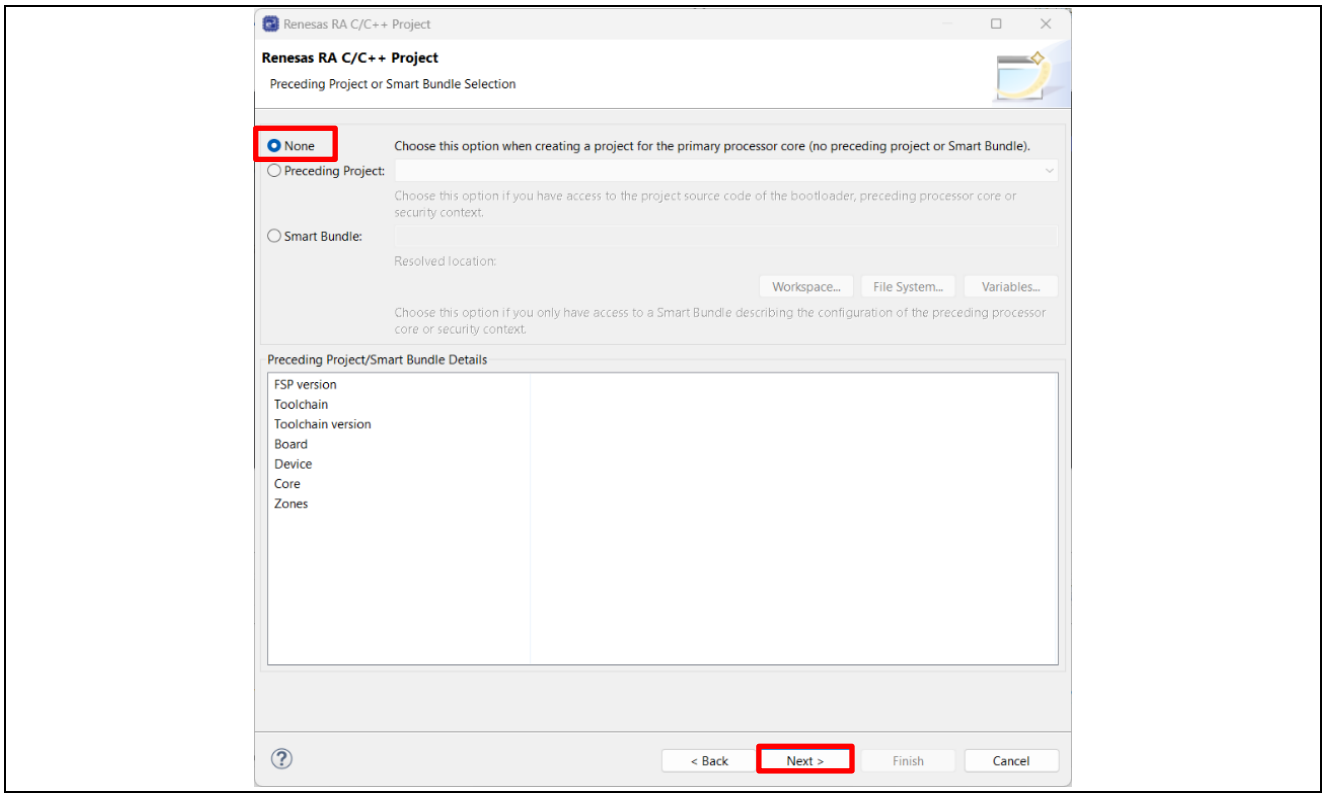

Figure 22. Configure Preceding project

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

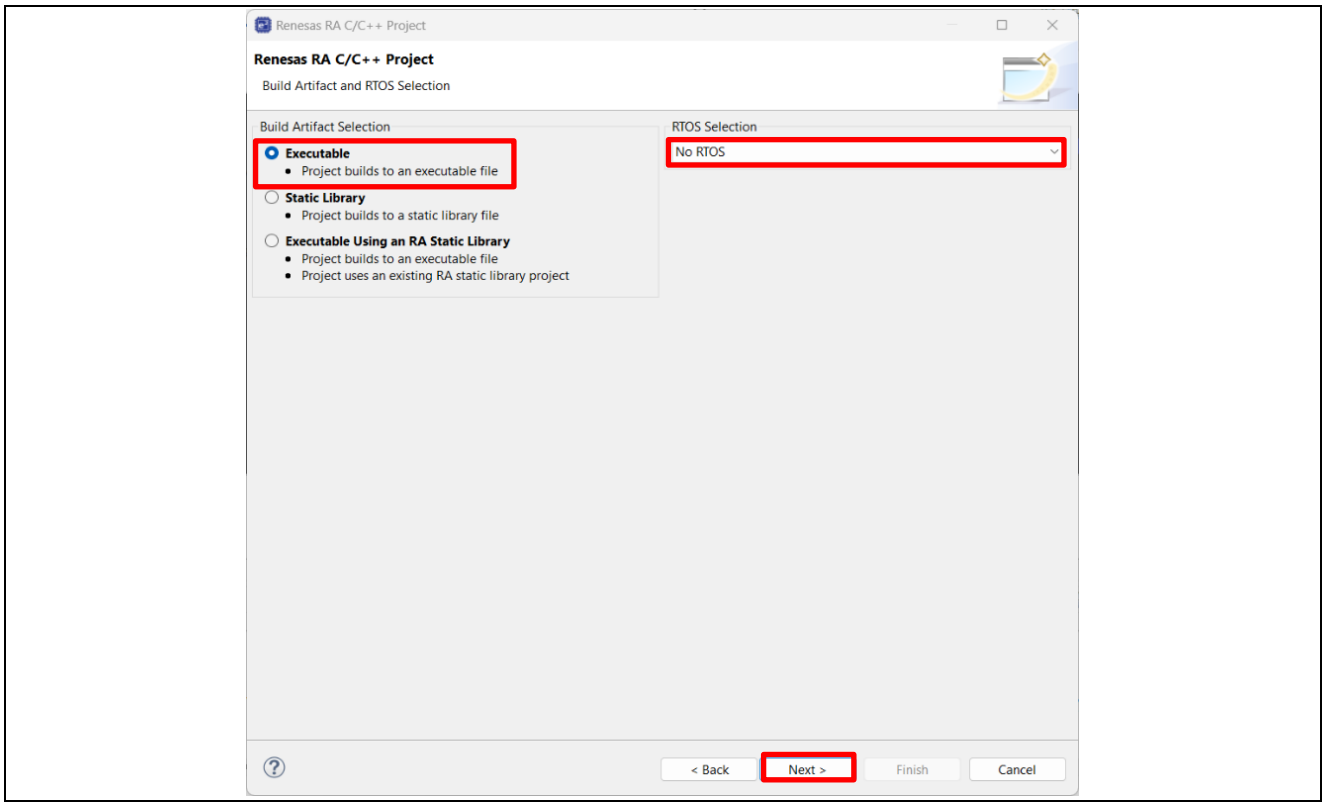

Figure 23. Build Artifact and RTOS Selection

5. In the next screen, select the project template. Choose **Bare-Metal – Minimal** as the **Project Template Selection** and click **Finish.**

6. Add the MCUboot module.
   The project will now be created, and the bootloader project configuration will be displayed. Change to the **Stacks** tab and select **New Stack > Bootloader > MCUboot**.

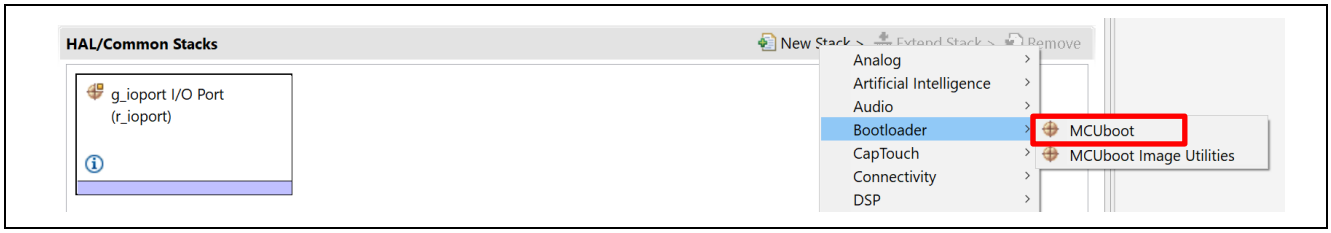

Figure 24. Add the MCUboot Module

4.2 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.

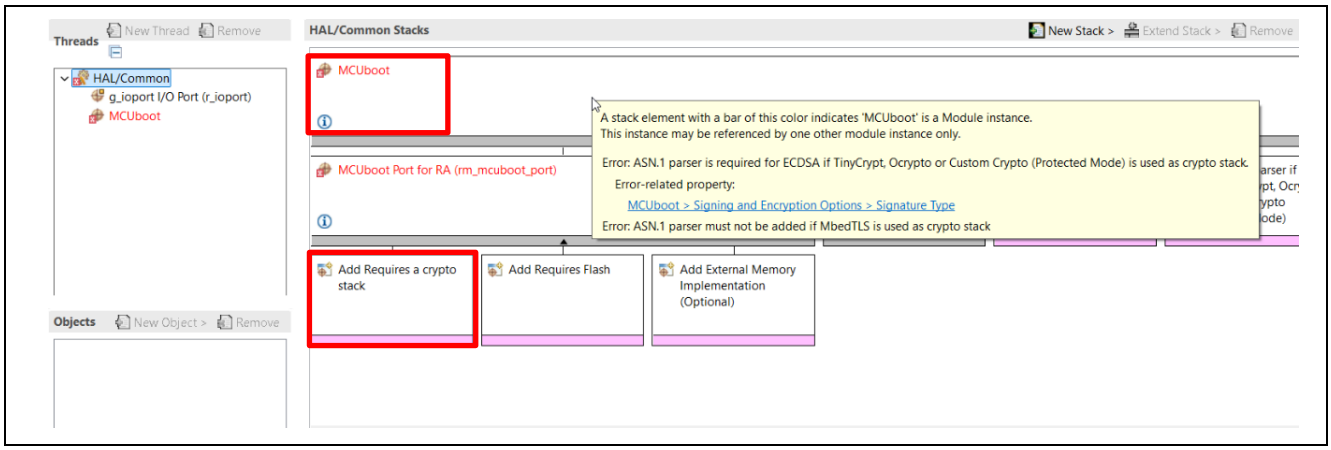

Figure 25. MCUboot Module Dependencies

Click on **Add requires a crypto stack**, choose **New** and add the **MbedTLS (Crypto Only)** stack.

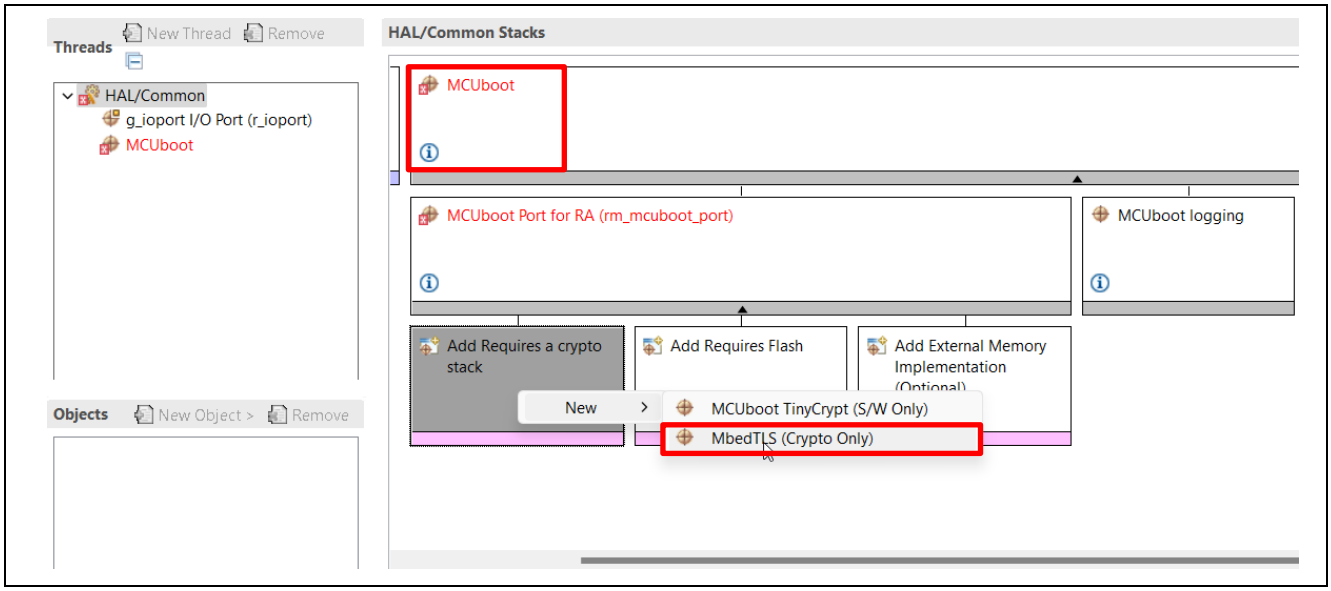

Figure 26. Add MbedTLS (Crypto Only) Module

2. Resolve dependencies of the **MCUboot Port for RA** as shown in the figure below.

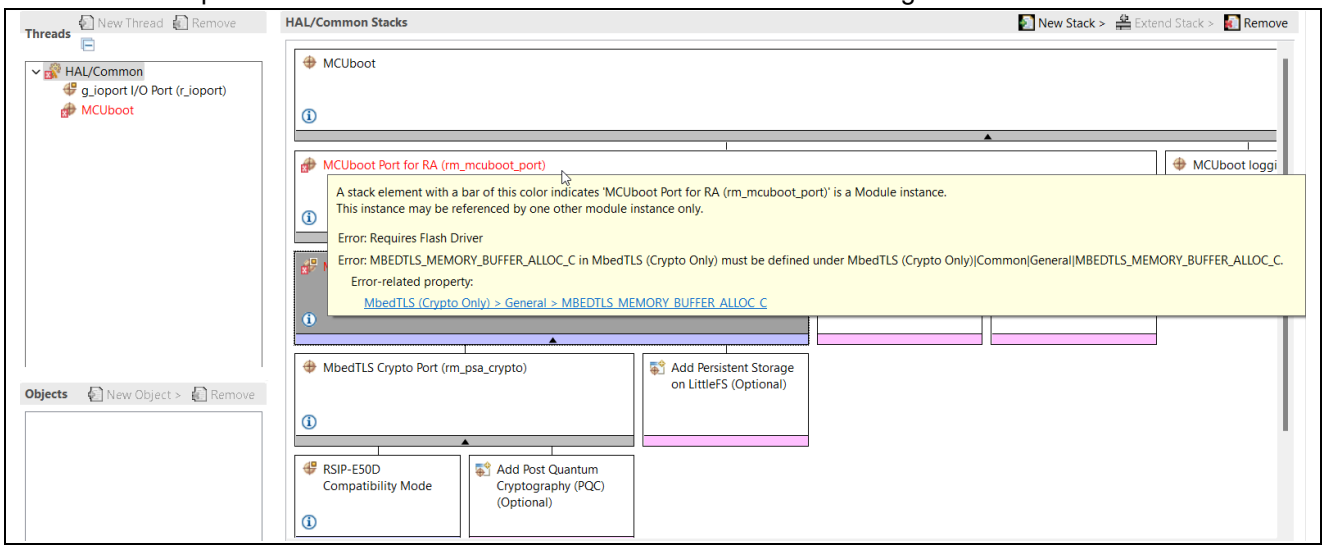

Figure 27. MCUboot Port for RA Module Dependencies

Add the `r_mram` module and define `MBEDTLS_MEMORY_BUFFER_ALLOC_C`. Users can quickly access the property by selecting the link that appears when hovering on the “MCUboot Port for RA” stack.

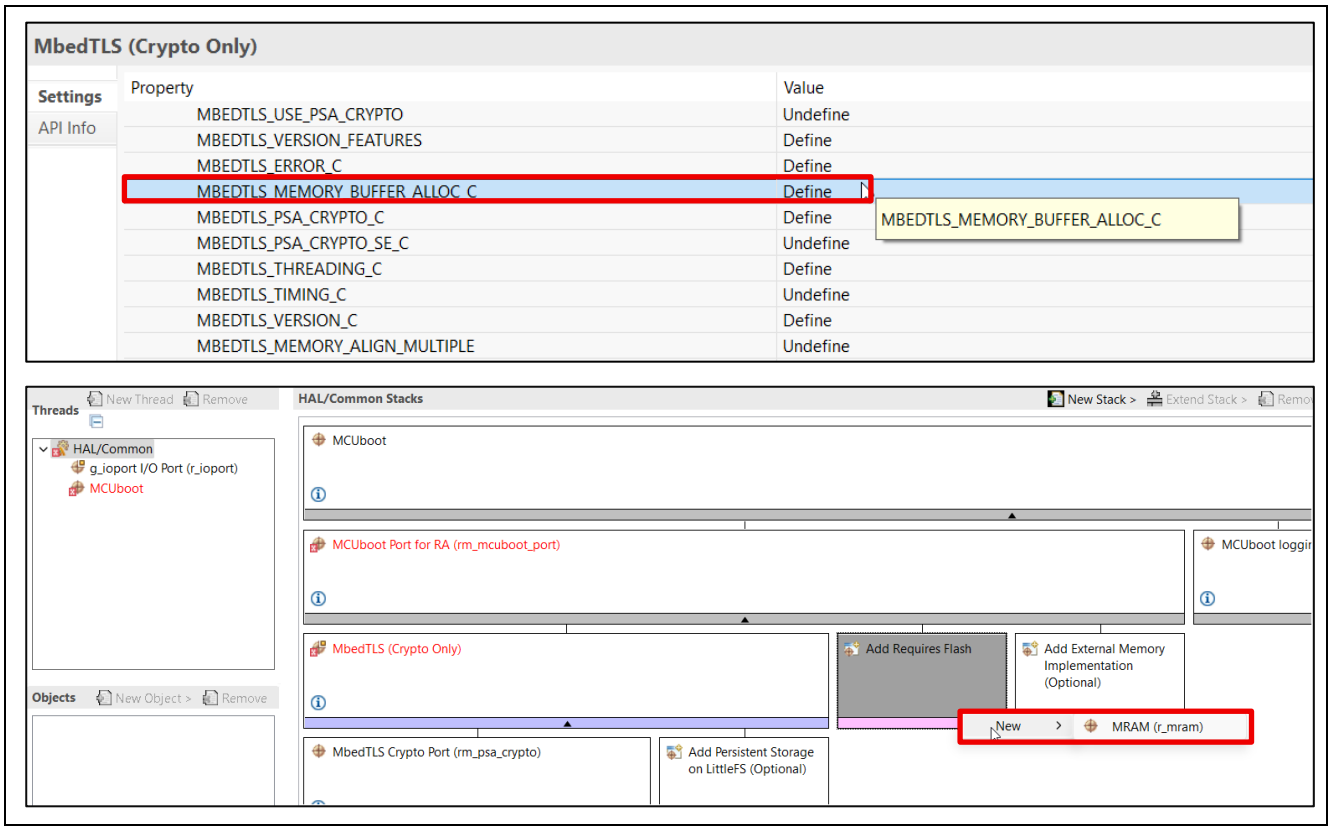

Figure 28. Resolve configurations for MCUboot **Port for RA stack**

3. Configure the MbedTLS Crypto dependencies.
   Follow the prompt in Figure 29 to update the corresponding properties for the MbedTLS (Crypto Only) Module.

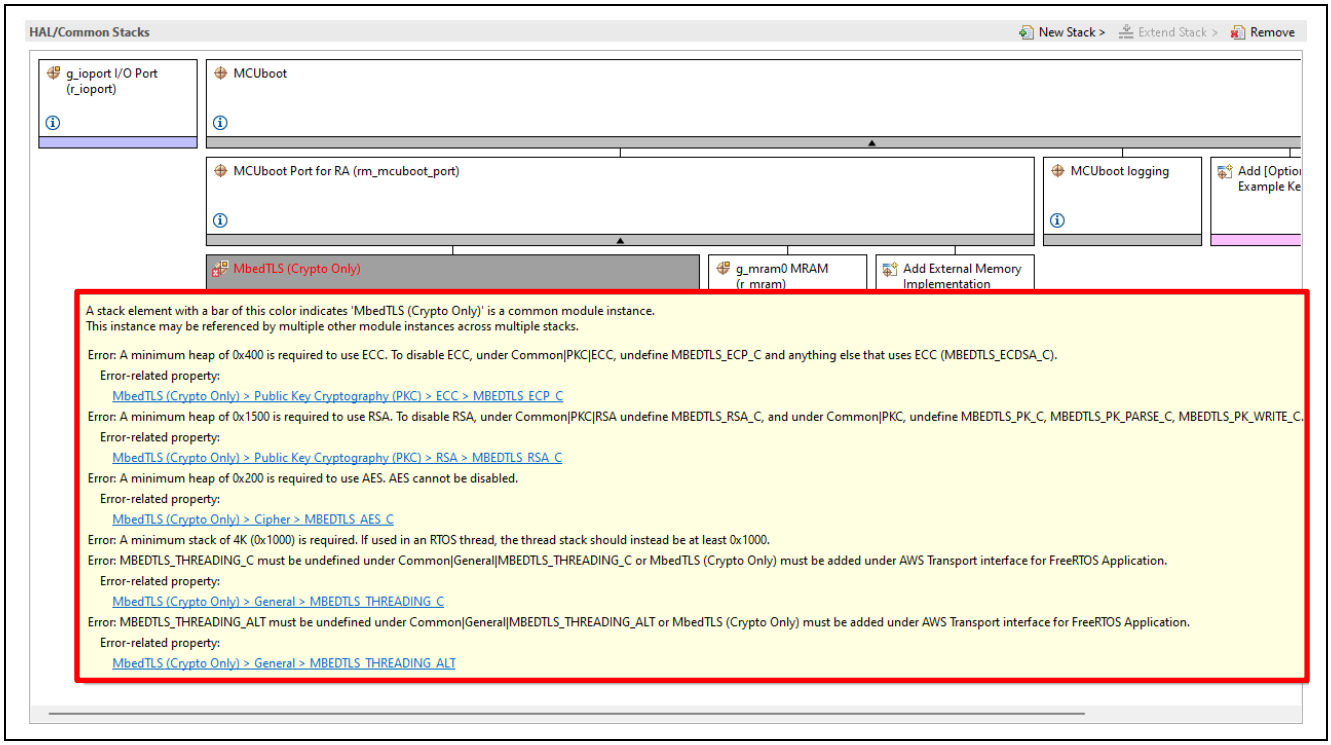

Figure 29. Dependencies of MbedTLS (Crypto Only) Stack

Configure the following properties:

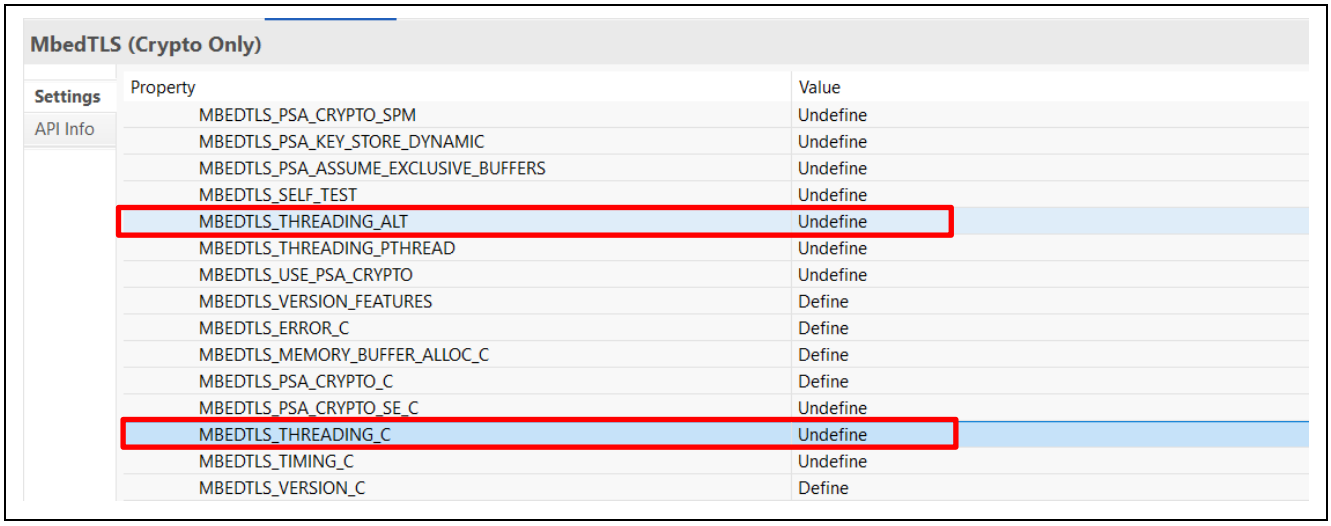

Figure 30. Configure Highlighted Properties for the MbedTLS (Crypto Only) Stack

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
- This bootloader design uses ECC for signature generation. Therefore, disabling RSA following the prompt in Figure 31 saves BSP Heap size for other possible user requirements.

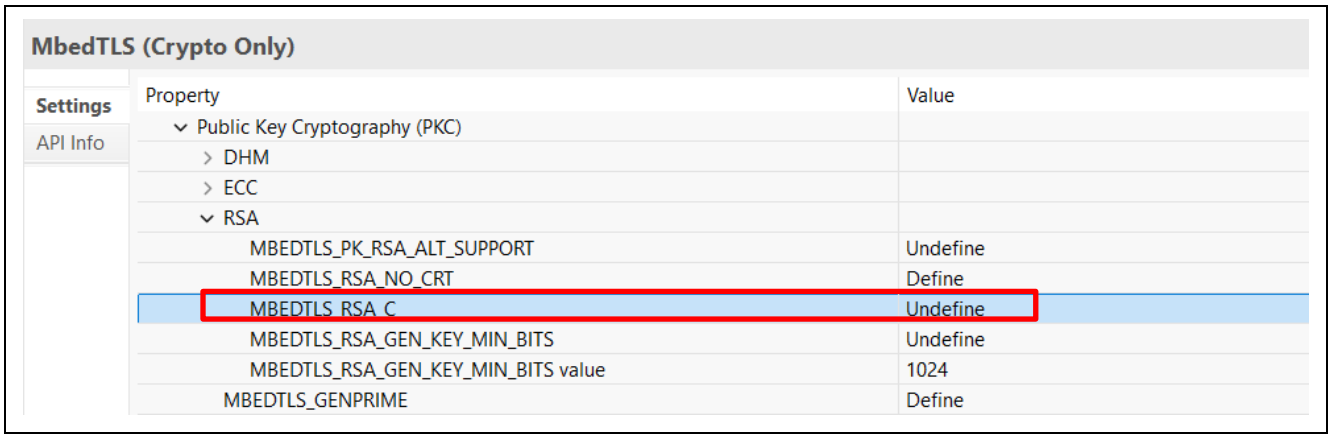

Figure 31. Disable RSA

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

4.3 Setting up the Booting Authentication Support

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 `/ra8p1_bootloader/ra/mcu-tools/MCUboot/sim/mcuboot-sys/csupport/keys.c`. (Available after completing “Add MCUboot Example Keys” (Figure 33) followed by Generate project content)
- The default private keys are included in folder `/ra8p1_bootloader/ra/mcu-tools/MCUboot/sim`.

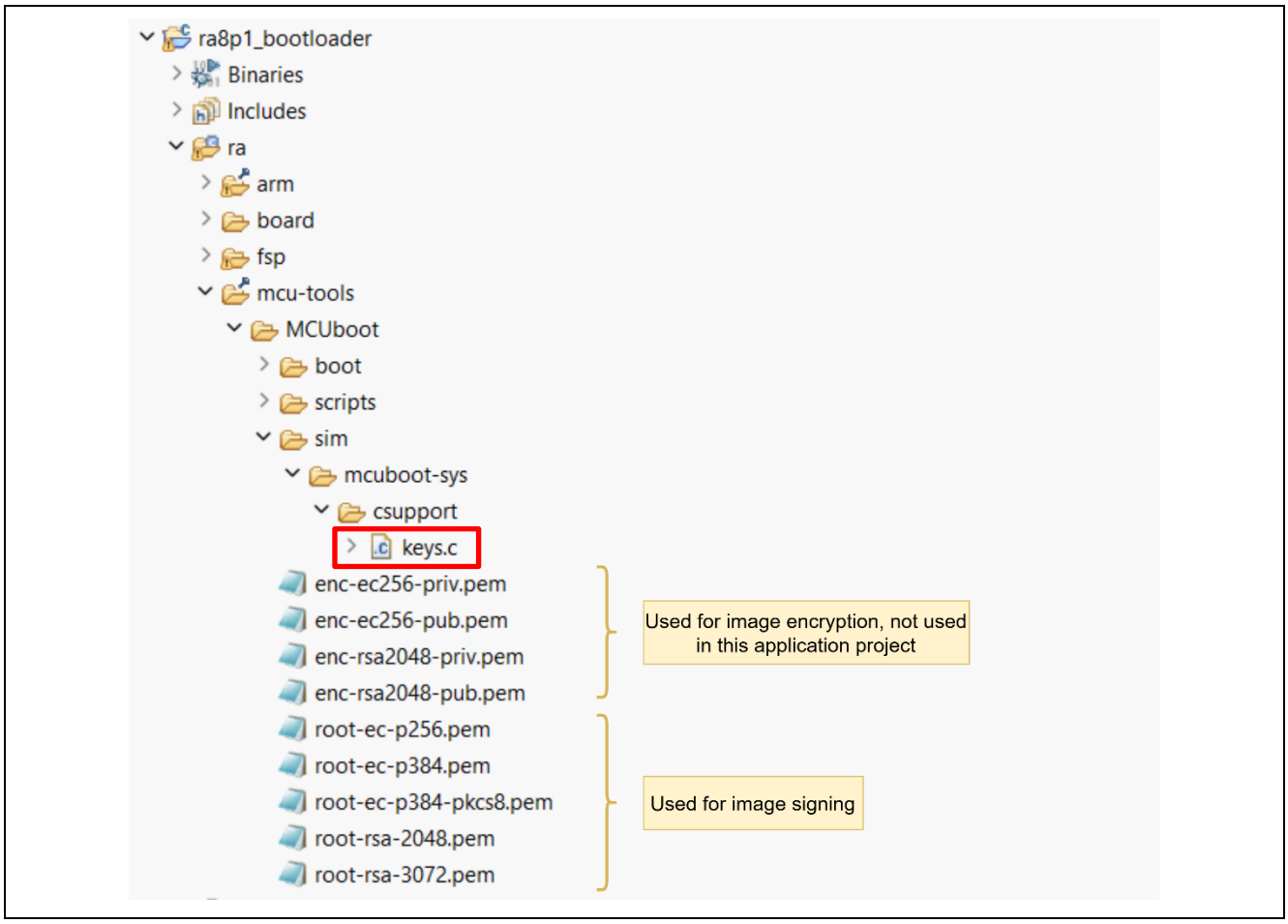

Figure 32. Example Public Keys and Private Keys Included in MCUboot Port Stack

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

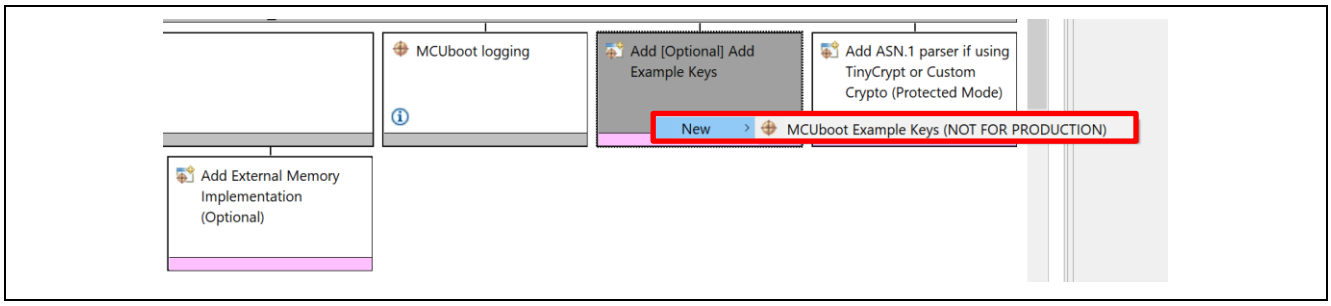

Figure 33. Add the MCUboot Example Key

Note: The example public key and private key used in the MCUboot is for testing purposes only. Refer to section 6.1 for guidelines on selecting the public key and private key for production support. Application Project R11AN0567 includes procedures to create customized key pair preparation. Refer to R11AN0567 to create customized key pairs.

4.4 Setting up the Application Authentication Signature Type

There are 4 signature types supported in FSP as shown below.

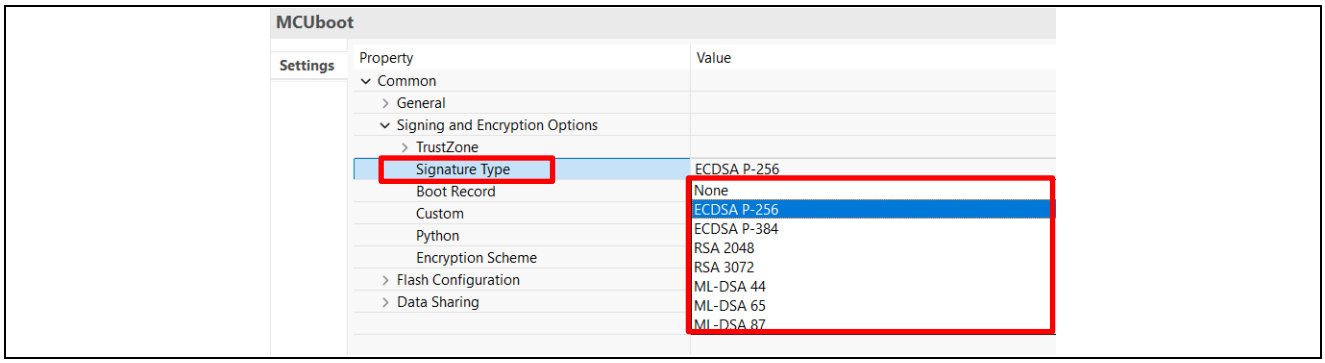

Figure 34. Signature Type

Open the **Property** page of stack **MCUboot** > **Common** > **Signing and Encryption Options** to look at the signing options. In this example implementation, **ECDSA P-256** is used for the example bootloaders.

4.5 Add MCUboot Initialization Code

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

1. Add the source code and compile the bootloader.

   After creating and setting up the MCUboot module, save the `configuration.xml` file and click **Generate Project Content**, then follow the steps below to add the source code to the bootloader project.

   - Open `src/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 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 source code and then compile the project.

5. Using the Bootloader with Applications

Follow the steps below to configure the dual-core application projects to use the bootloader and sign the application.

5.1 Configure the Application Projects to Use the Bootloader

With dual core applications using with bootloader, when creating the application project for CPU0, at the **Preceding Project or Smart Bundle Selection** step (as shown in Figure 35), make sure to select the previously created bootloader project as the Preceding Project. This example is using `ra8p1_bootloader` project.

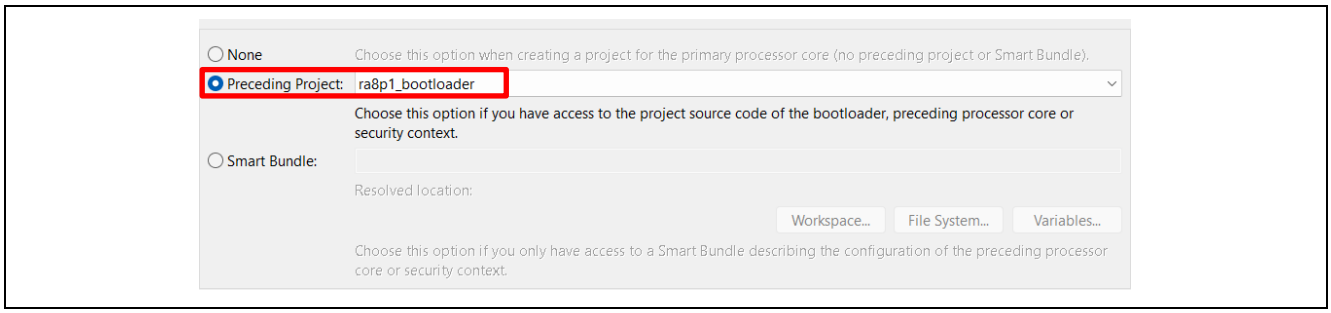

Figure 35. Choose bootloader project as Preceding Project

Continue to create CPU1 application as usual, choose CPU0 application as the **Preceding Project or Smart Bundle Selection** step.

If a dual-core application was previously created, the project property configuration must now be modified to work with the bootloader.

Right-click on CPU0 application folder and select **Properties**.

Select **C/C++ Build > Build Variables** > click **Add…**, set the **Variable name** to **SmartBundle**.

For **Type**, select **File** then **Browse** to the `*.sbd` file created for the associated MCUboot project. If the MCUboot project was specified as the Preceding Project during project creation as guideline at beginning of this section, this step is done automatically.

Click **OK**, then **Apply**.

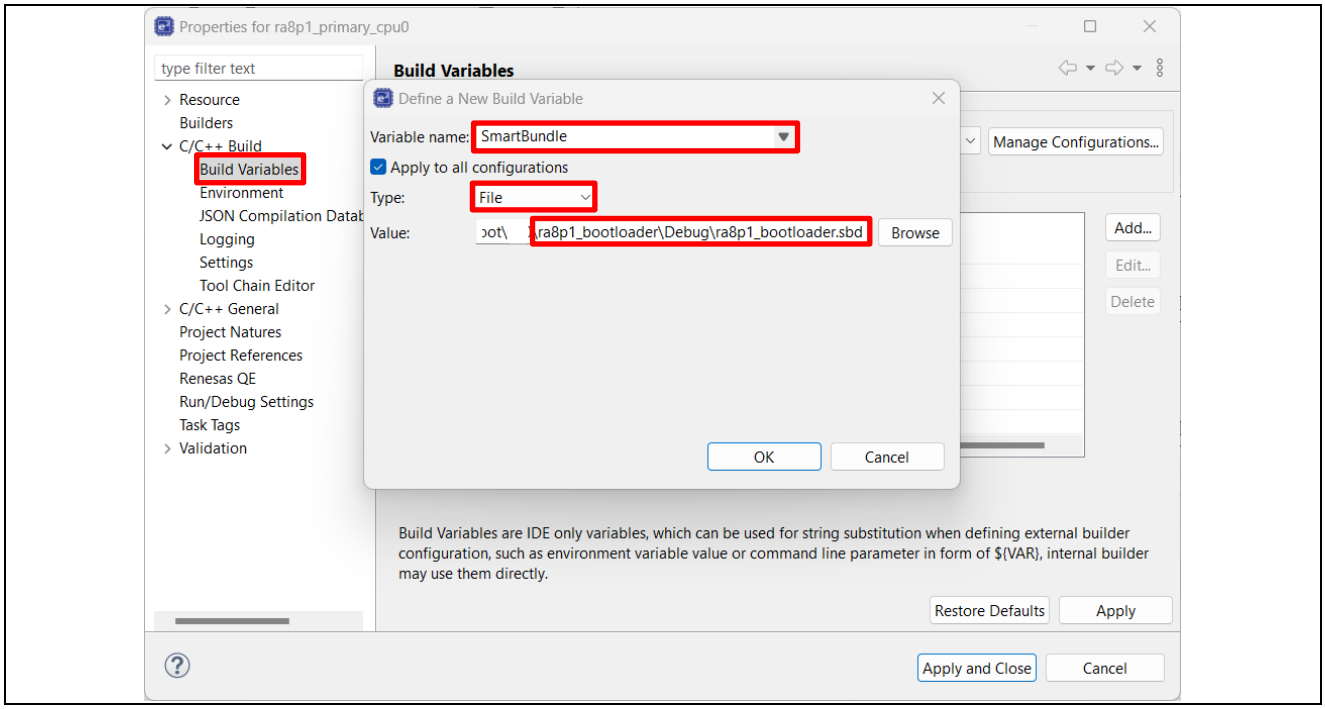

Figure 36. Add Smart Bundle Variable

5.2 Signing the Existing Application Projects to Use the Bootloader

Each application can have a defined version number. This version 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` with value in this case is `${workspace_loc:ra8p1_bootloader}/ra/mcu-tools/MCUboot/root-ec-p256.pem`.

Select **Environment**, then click **Add...** to add the required variables for each CPU application.

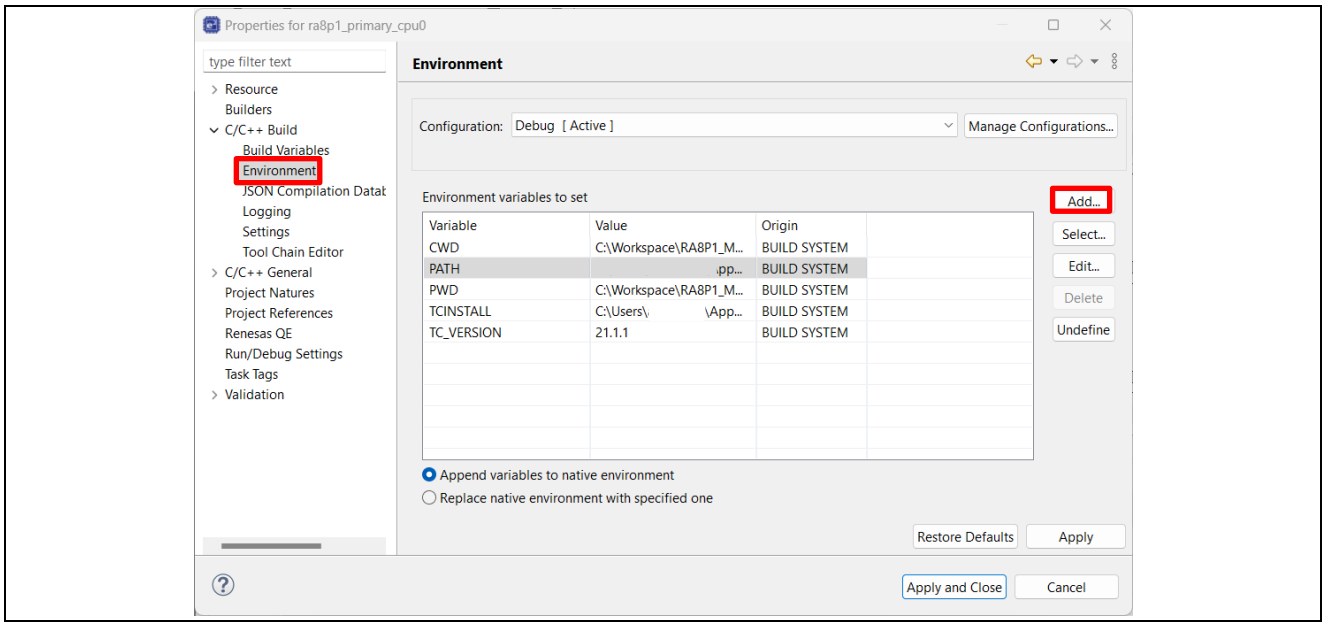

Figure 37. Configure the Build Variable to Use the Bootloader

Add environment variable for the application image version.

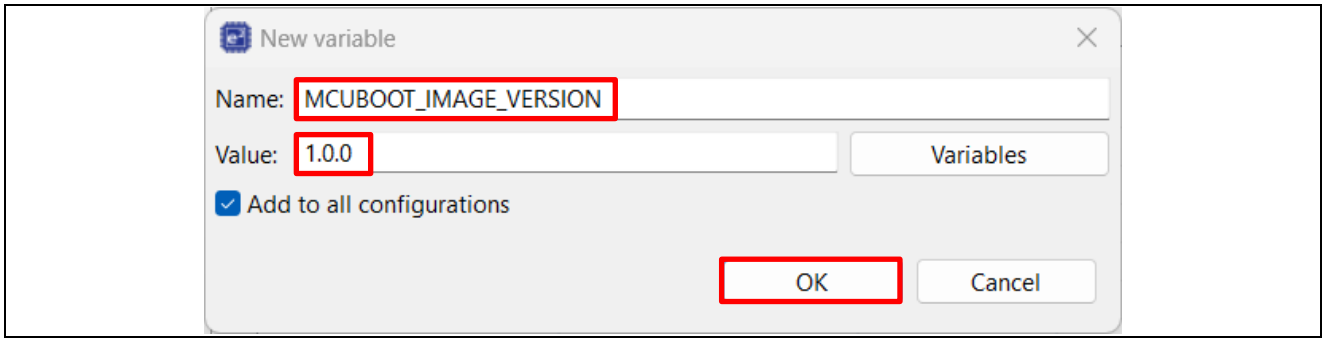

Figure 38. Add MCUBOOT_IMAGE_VERSION Variable

Add an environment variable to configure the application image signing key.

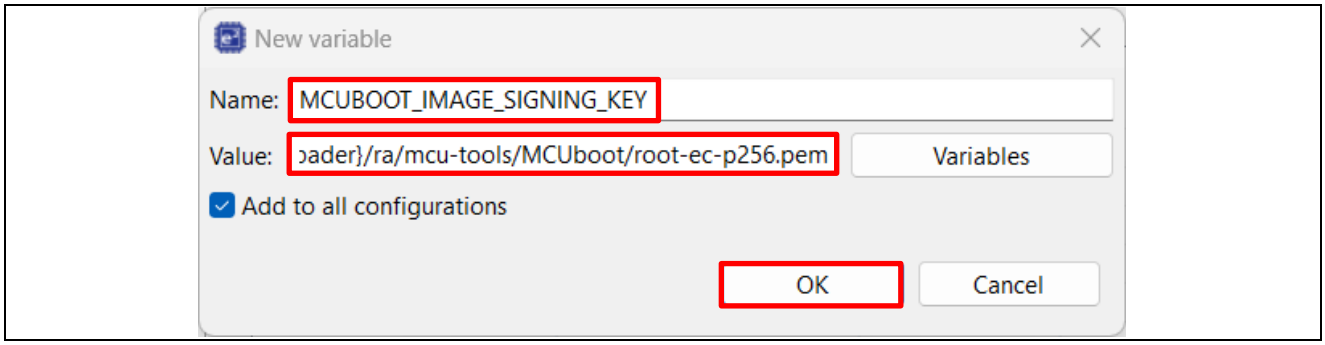

Figure 39. Add MCUBOOT_IMAGE_SIGNING_KEY Variable

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, user **MUST** change this to the private key of their choice.

Add an environment variable to convert the .elf file to a raw binary format used in the image signing process. Set variable `MCUBOOT_APP_BIN_CONVERTER` to the path of the appropriate tool—objcopy, arm-none eabi-objcopy, fromelf, or ielftool—depending on the toolchain used for the project. This example project is using LLVM toolchain, so this variable's value is set to `${TCINSTALL}bin\llvm-objcopy.exe`.

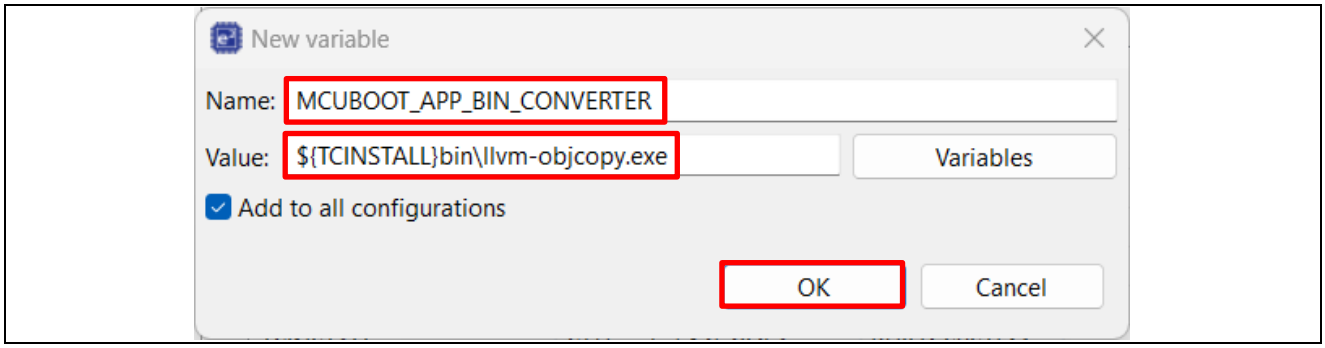

Figure 40. Add MCUBOOT_APP_BIN_CONVERTER Variable

After adding the above variables, click **Apply and Close**.

If the application has already been created using solution project, the user must delete the existing solution project and remove the `/Debug` folder from each CPU application, then rebuild them before proceeding to the next step.

5.3 Linking Application with Bootloader

From FSP 6.0.0, the MCUboot project and application project must be linked by using RA FSP solution project to set up the bootloader slots, the slot size and header information was previously in the MCUboot module. Please ensure that both the bootloader project and the application project have been successfully built before proceeding with the linking process. The following section outlines the steps to create a FSP solution project.

1. From the e2 studio Workspace, navigate to the **File > New > C/C++ Project > All** and then select **Renesas FSP Solution Project (Advanced)** and press **Next**. Provide the solution project name `ra8p1_solution` and click **Next**.
2. When the following screen appears, select the created CPU1 primary project as the final project of a chain of projects that constitute a complete FSP application and click **Finish**.

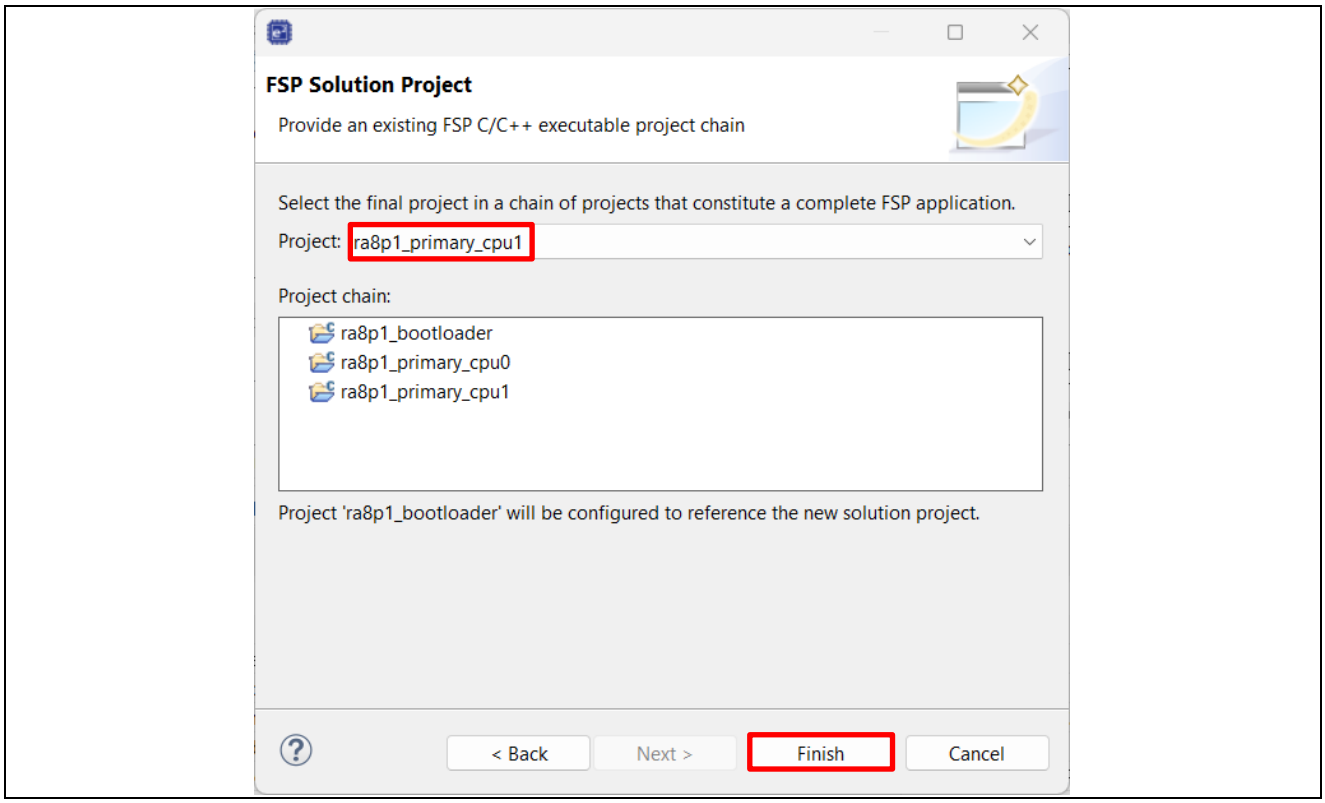

Figure 41. Choose CPU1 primary application as final project

3. The solution project will now be created, and the project configuration will be displayed. Select the Memories tab then click on **Flash > Add Partition**

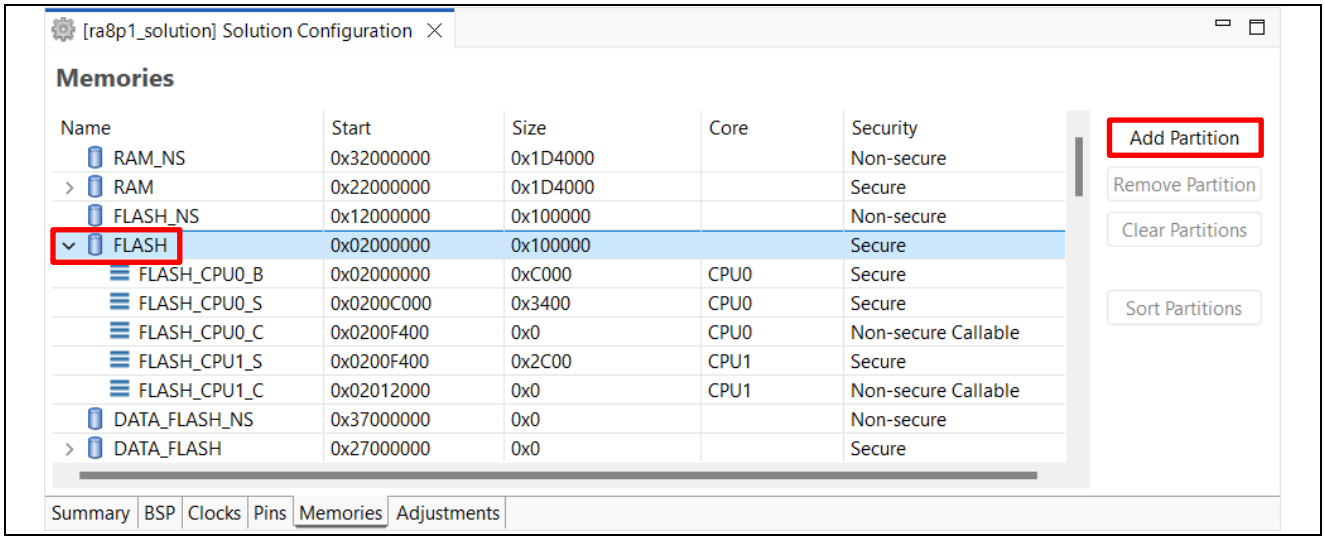

Figure 42. Add Partitions for Flash Memory

4. Add the memory layout for bootloader and application. In this example, the memory partitions are defined according to the memory map shown in Figure 43. About the symbol conventions for memory partitioning, user can refer to the guide at FSP User's Manual under RA Flexible Software Package Documentation > API Reference > Modules > Bootloader > MCUboot Port (rm_mcuboot_port).

   Once all partitions have been defined, choose **Flash > Sort Partitions** to automatically arrange them in order based on their addresses.

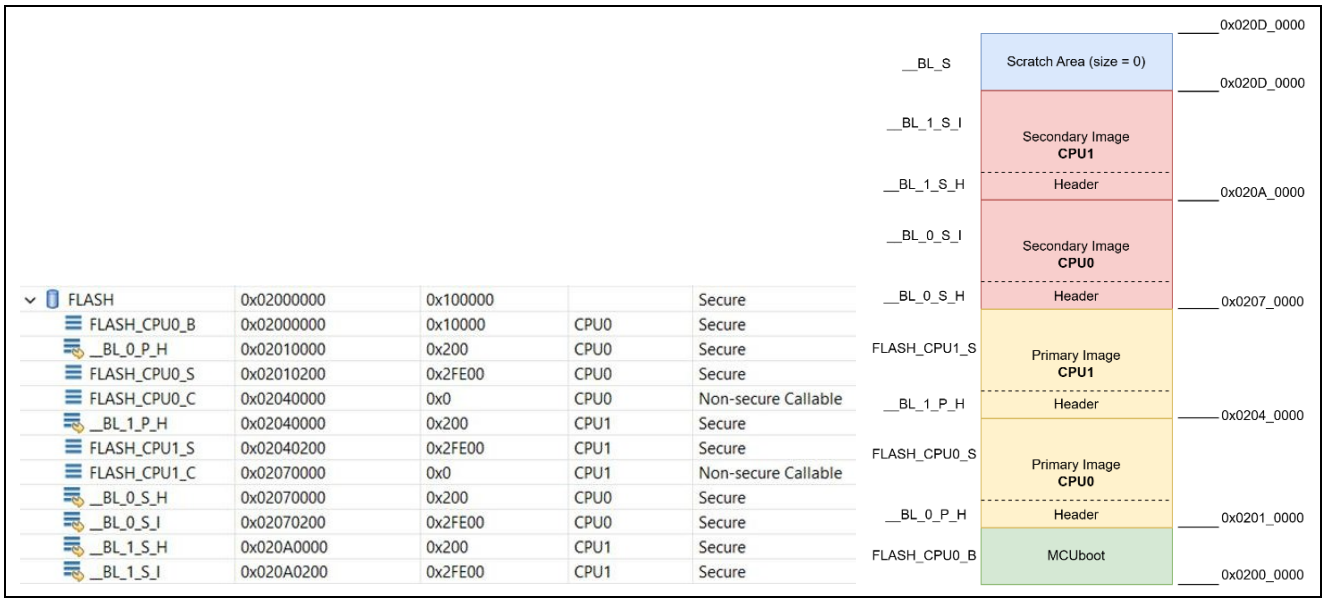

Figure 43. Define Flash Partitions for Bootloader and Application Images

5. After configuring the memory layout in the Solution Project, right-click on the FSP Solution Project → Build Project. Both the bootloader and the primary application project will be rebuilt using the updated memory partition settings.

Note: The secondary application projects must be compiled separately after the Solution Project has been fully built to apply the updated memory map settings.

5.4 Optimizing SRAM Allocation

The **Renesas FSP Solution Project (Advanced)** project template is also used when developing applications for Renesas Dual Core MCU. These applications have a unified SRAM that is used by both cores, but these are typically separate. As such, any RAM areas are defined as separate areas. In an MCUboot application, the RAM areas can overlap as the bootloader and application are never running at the same time and both applications can access the full RAM area.

Maximize the MCU SRAM by the following steps:

1. Navigate to the **Solution Project**: `ra8p1_solution`
2. In the **RAM Memory Partition**, configure the **Start address** and **maximum RAM size** for the application project:

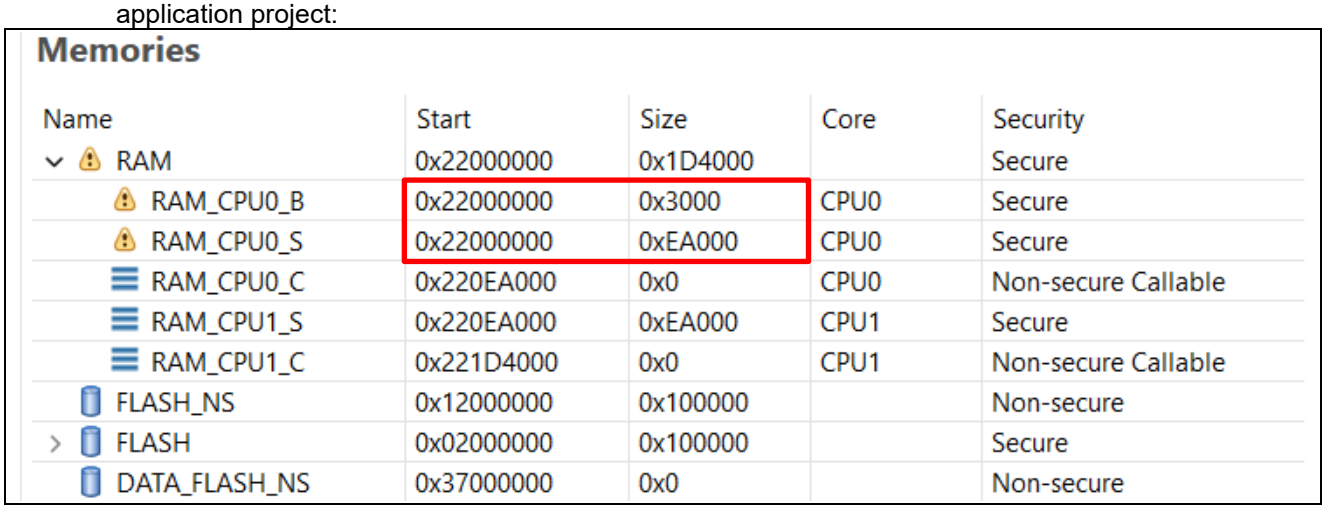

Figure 44. Adjust the maximum SRAM for the applications in EK-RA8P1

The warning that appears is a **Partition Overlapping** warning, but this can be ignored.

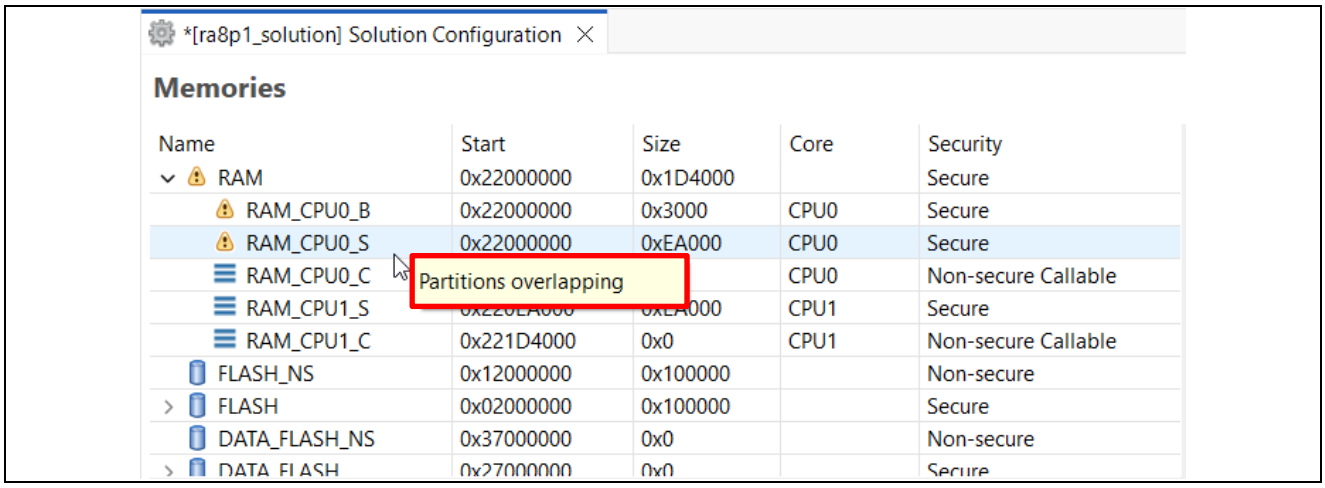

Figure 45. Partition Overlapping Warning

Press Ctrl + S to save the configuration

5.5 Configure the Debug Configuration

Open the Debug Configurations: **ra8p1_primary_cpu1 > Debug As > Debug Configurations**. Make sure that `ra8p1_primary_cpu1 Debug_Multicore` is selected and select the **Startup** tab.

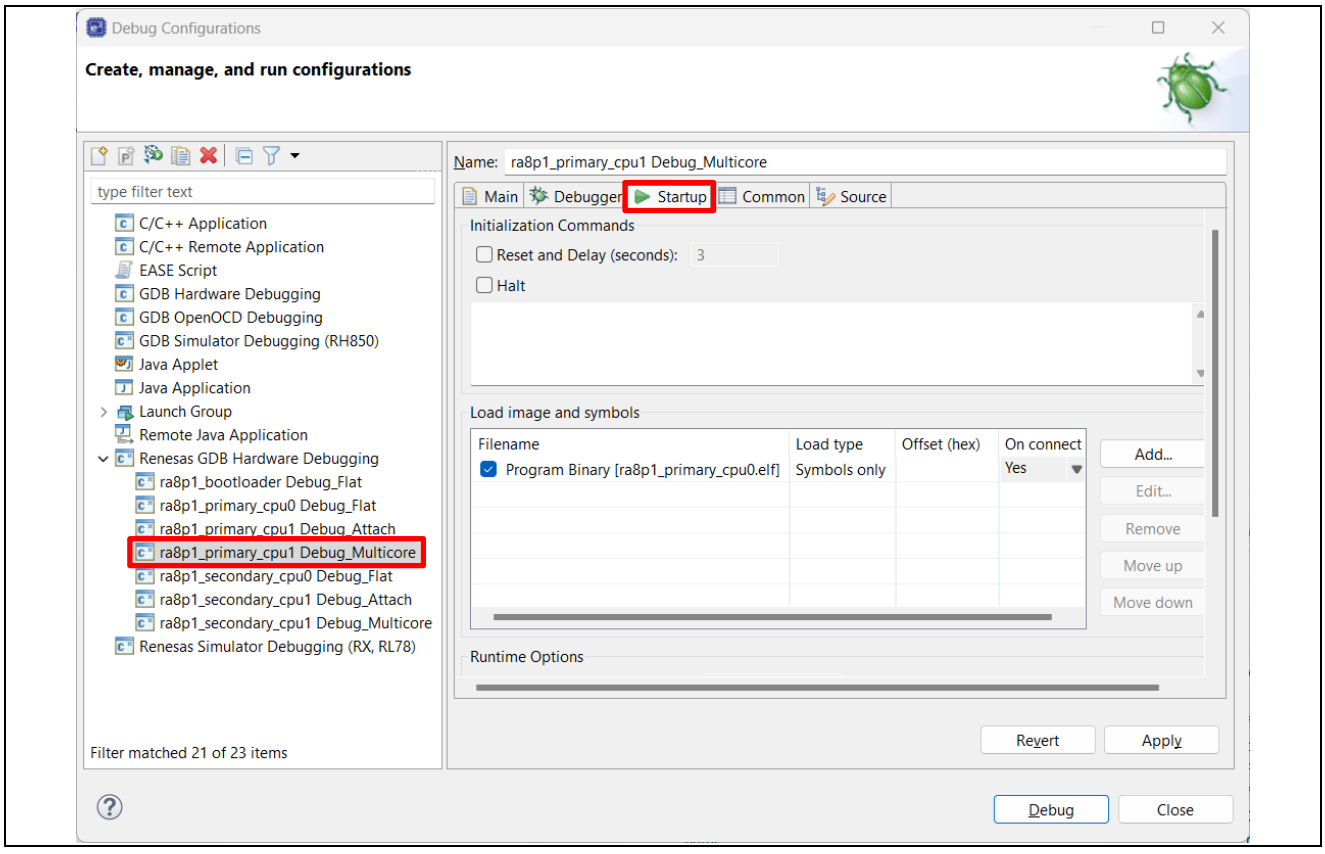

Figure 46. Select Debug_Multicore Configuration

Set up the Debug Configurations. Click **Add…** and then **Workspace**. Navigate to the ra8p1_bootloader project and select the `ra8p1_bootloader.elf` file from the debug folder. Click **OK.**

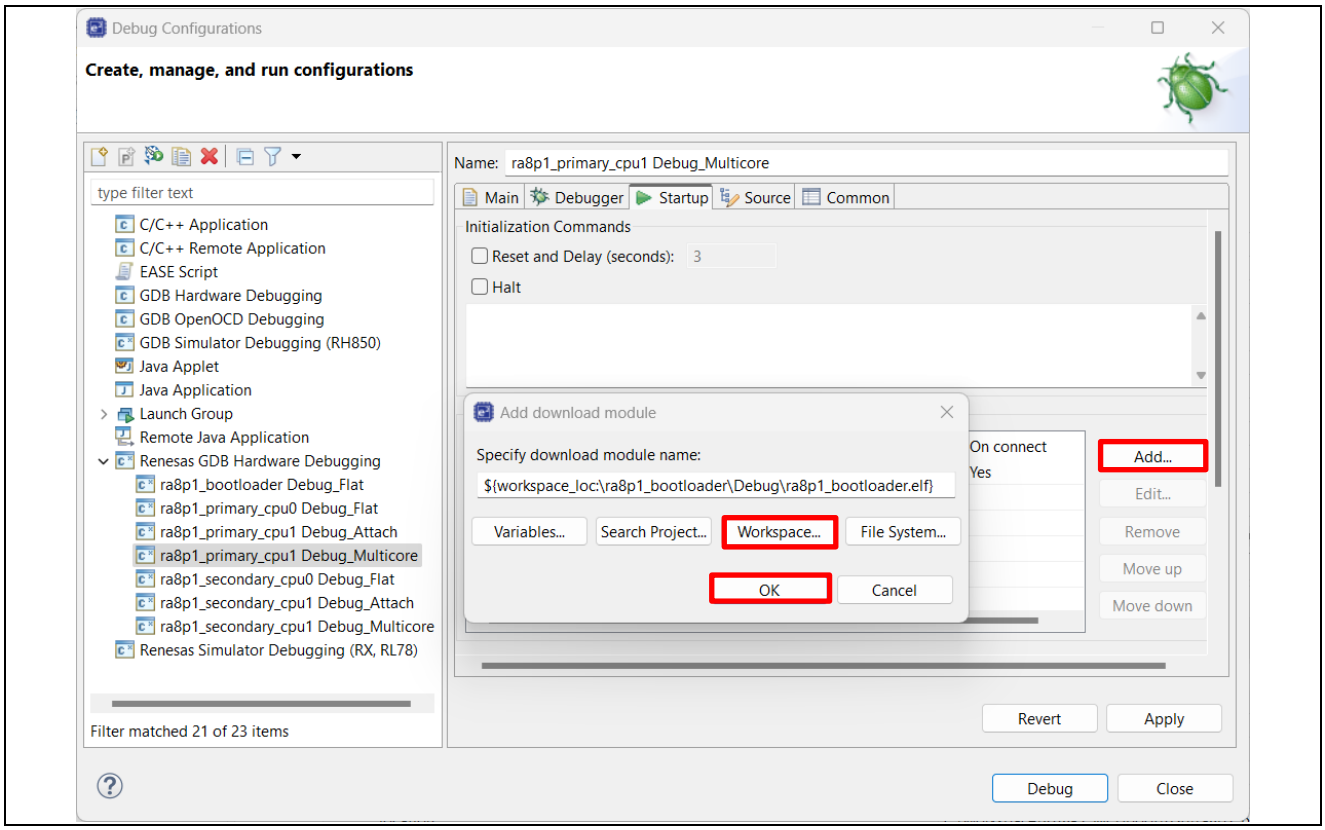

Figure 47. Add bootloader module

Click Add again and add the CPU0 and CPU1 signed binary images to the download options as in the prior step. Click OK

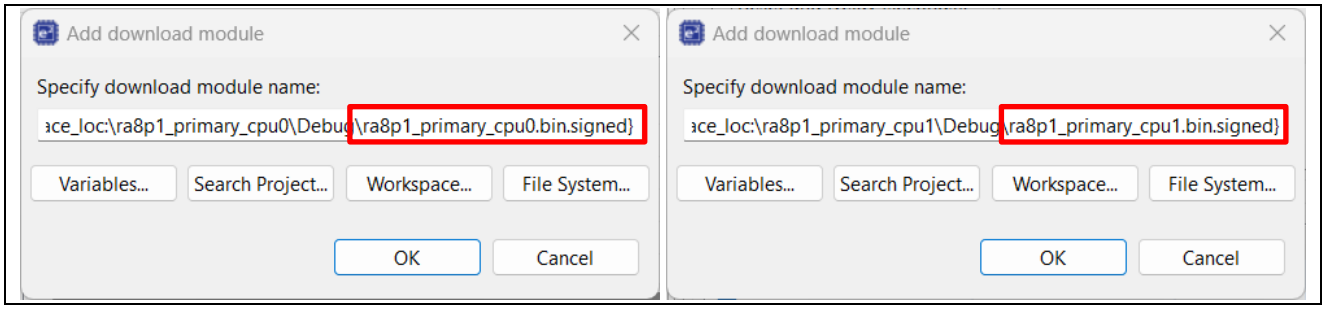

Figure 48. Add primary images to the download options

Change download options to **Raw Binary** as Load type for both CPU0 and CPU1 primary images.

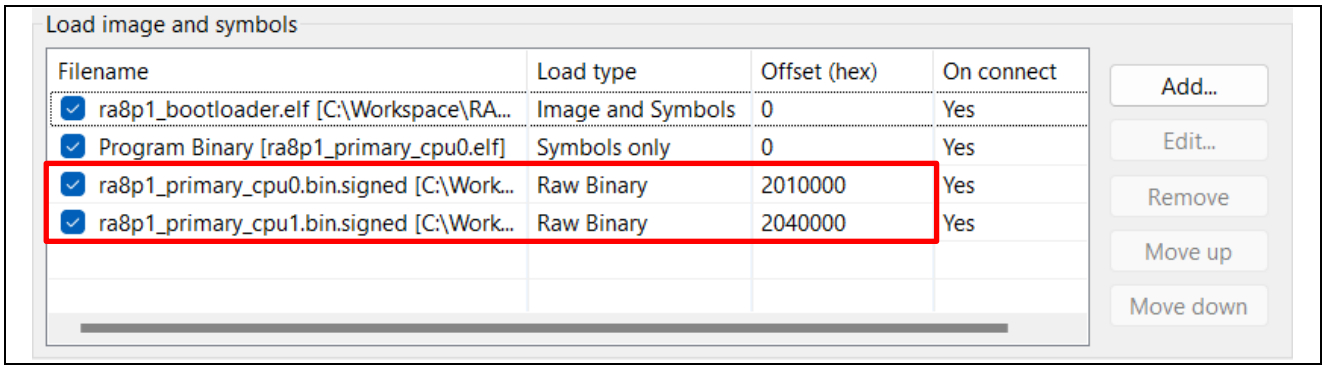

Figure 49. Load the signed primary images

Note that for different update modes and different application images, the load address needs to be updated. For the example projects included in this application project, you can reference the memory configuration images included in Figure 43 to set up the load address.

Navigate `ra8p1_primary_cpu1 Debug_Attach`, select the **Startup** tab and make sure that the ra8p1_primary_cpu1.elf is configured as figure below.

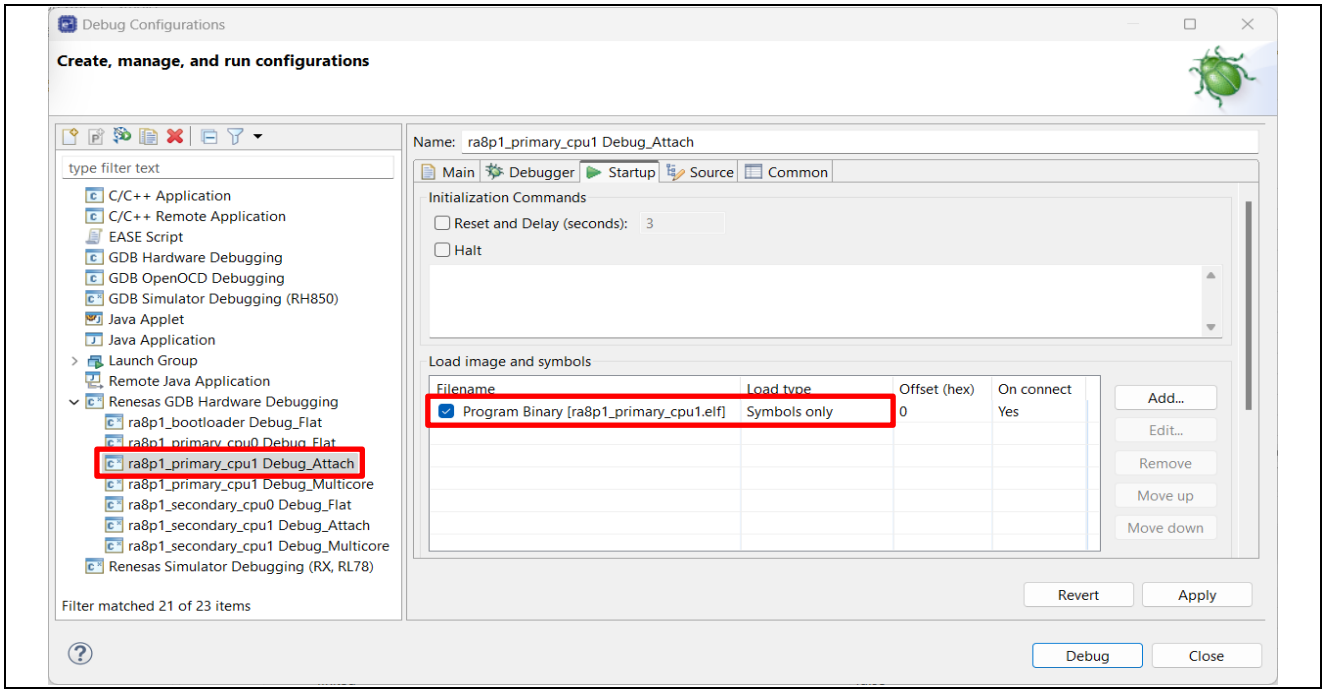

Figure 50. Debug_Attach Setup

After the above is set up, follow section 3.2.4 to run the projects using Launch Group Debugging.

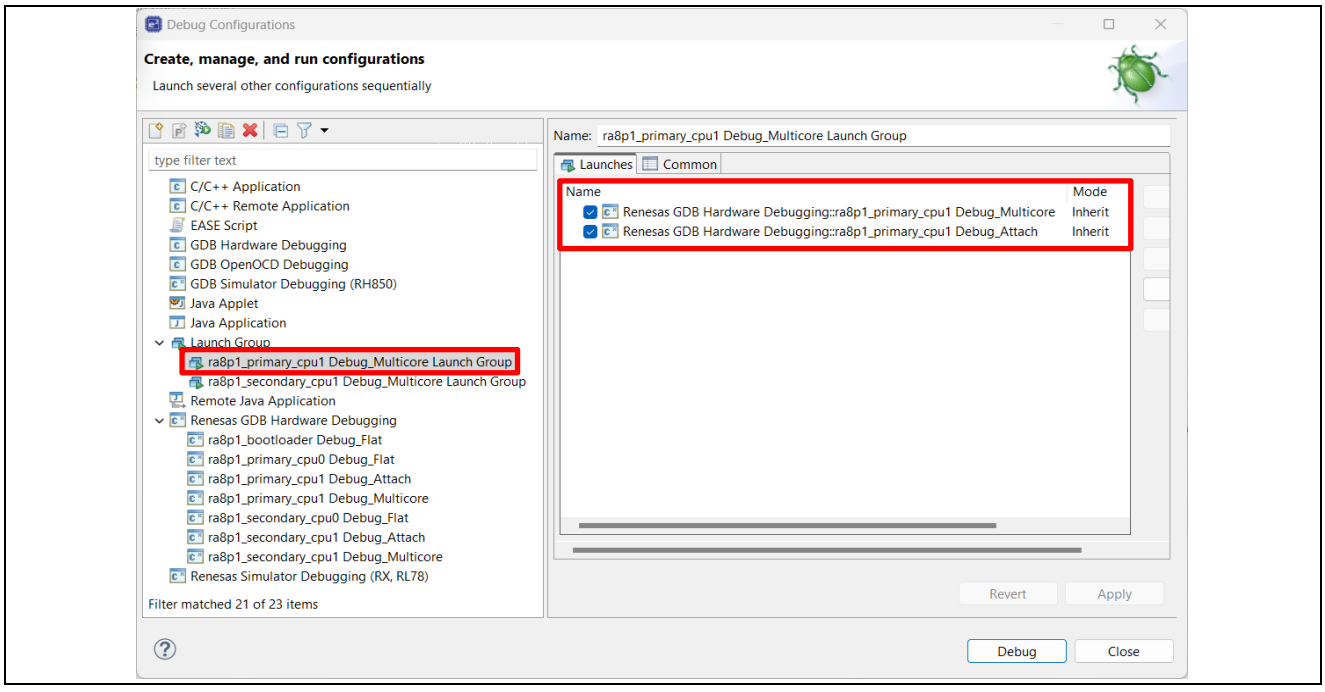

Figure 51. Using Launch Group for Multicore Debugging

6. Production Support Considerations

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 4.3 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 Figure 32 and Figure 33 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 provided 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 in order 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.

6.2 Mastering and Delivering a New Application

Mastering and delivering a new application involves similar steps described above in section 4 and section 5. Typically, the following aspects must be considered in the design of delivering new applications:

1. Create and sign the new applications.
   Refer to Section 5 for instructions on how to configure the new application to work with the bootloader and how to sign it
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. These projects can be used for references.

7. References

- RA8 Basic Secure Bootloader Using MCUboot and Internal Code Flash (R11AN0909)
- RA6 Secure Bootloader Using MCUboot with Protected mode and Internal Code Flash (R11AN1048)
- RA0/RA2 MCUboot and USB-Based Firmware Updates (R11AN1088)

8. Website and Support

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

EK-RA8P1 Resources	renesas.com/ra/ek-ra8p1
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
1.0.0	Mar.31.26	-	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.

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/.