

Renesas RA Family

Secure Bootloader using MCUboot with Post-Quantum Cryptography

Introduction

MCUboot is an open-source secure bootloader for 32-bit MCUs that provides a standardized framework for secure boot, firmware authentication, and fail-safe firmware update. It defines a common flash layout, image format, and verification flow, enabling portable and scalable secure boot implementations across different platforms. MCUboot is hardware-agnostic and can operate either using software-based cryptographic libraries or by leveraging hardware-accelerated cryptographic engines through platform-specific integrations.

On Renesas RA MCUs, the Flexible Software Package (FSP) integrates MCUboot as a module starting from FSP v3.0.0. This integration allows MCUboot to utilize the on-chip Security Engine (the Secure Crypto Engine, SCE, or Renesas Secure IP, RSIP) for hardware-accelerated cryptographic operations, such as signature verification, improving both performance and security. Depending on the system design, developers can choose between software-based cryptography (e.g., MbedTLS) or hardware-accelerated cryptography using the Security Engine.

As quantum computing technology continues to advance, traditional public-key cryptographic algorithms such as RSA and Elliptic Curve Cryptography (ECC) are expected to become vulnerable to future quantum-computing attacks. To address this emerging threat, the cybersecurity industry is transitioning toward Post-Quantum Cryptography (PQC), a new generation of cryptographic algorithms designed to provide security against both classical and quantum computers. The National Institute of Standards and Technology (NIST) has standardized several PQC algorithms, including ML-KEM for key establishment and ML-DSA for digital signatures, enabling organizations to begin preparing for long-term cryptographic resilience.

The Renesas Flexible Software Package (FSP) provides integrated support for Post-Quantum Cryptography on RA Family MCUs through its cryptographic framework and PSA Crypto integration. Current FSP releases support the NIST-standardized ML-DSA digital signature algorithm and provide support for ML-DSA-44, ML-DSA-65, and ML-DSA-87 parameter sets. This enables developers to evaluate and deploy quantum-resistant firmware authentication solutions on supported RA devices without requiring external cryptographic libraries or custom toolchains.

This application note demonstrates how to configure and use MCUboot with Post-Quantum Cryptography on Renesas RA Family MCUs. Example projects are provided for RA8, RA6, RA4, and RA2 devices, illustrating the use of ML-DSA-based firmware authentication together with various MCUboot update strategies, including Swap and Direct XIP modes. The objective is to help developers evaluate quantum-resistant secure boot implementations and prepare embedded products for future security requirements.

This application note does not cover ML-KEM, secure communications, TLS migration, or production key-management infrastructure.

Target devices

- RA8 Series
- RA6 Series
- RA4 Series
- RA2 Series

Note: The example projects require at least 32 KB of RAM. Devices with less than 32 KB of RAM, such as RA2L2, RA2E1/E2/E3, and RA2T1, are not supported.

Required Resources

Development tools and software

- e² studio IDE v2026-07.0 or later.
- RA Family Flexible Software Package (FSP) v6.6.0 or later.
- SEGGER J-link® V9.42 or later.
- Renesas Flash Programmer (RFP) V3.23.00 or later.
- LLVM for Arm v22.1.0.
- Python v3.13.7.

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

Hardware

- EK-RA8M2
- EK-RA8M1
- EK-RA6M5
- EK-RA6M3
- EK-RA4C1
- EK-RA2A2
- USB Micro-B cables (USB Type-A male to USB Micro-B male) for EK-RA8M1, EK-RA6M5, EK-RA6M3, and EK-RA2A2
- USB Type-C cables for EK-RA8M2 and EK-RA4C1
- Host PC running Windows® 11 OS

Prerequisites and Intended Audience

This application note assumes you have some experience with the Renesas e² 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 Blinky project. Doing so enables you to become familiar with the e² studio and the FSP and validates that the debug connection to your board functions properly. In addition, the application note assumes that you have some knowledge of MCUboot and cryptography. Prior knowledge of Python usage is also helpful. Prerequisite reading [RA Flexible Software Package Documentation: MCUboot Port \(rm_mcuboot_port\)](#). The goal of reading is to understand MCUboot in FSP.

The intended audience is developers who want to build a secure bootloader that uses Post-Quantum Cryptography in bare-metal environments using Renesas RA Family MCUs.

For information on other MCUboot topics, refer to the following application notes:

Topic	RA2 series	RA4 series	RA6 series	RA8 series	RA8 dual core
Secure boot (Internal code flash)	R11AN0516 - Secure Bootloader for RA2 MCU Series	R11AN0869 - RA4 Secure Bootloader Using MCUboot and Internal Code Flash	R11AN0497 - RA6 Basic Secure Bootloader Using MCUboot and Internal Code Flash	R11AN0909 - RA8 Basic Secure Bootloader Using MCUboot and Internal Code Flash	R11AN1099 - Using MCUboot with Dual core RA8 MCUs
	R11AN1065 - Secure Bootloader using MCUboot with the Ocrypto Cryptographic Library	R11AN1048 - Secure Bootloader Using MCUboot with SCE9 Protected mode and Internal Code Flash	R11AN0570 - RA6 MCU Advanced Secure Bootloader Design using MCUboot and Code Flash Dualbank Mode	R01AN7556 - RA8 MCU Advanced Secure Bootloader Design using MCUboot and Code Flash Dualbank Mode	
	R01AN7766 - RA2 MCU Advanced Secure Bootloader Design using MCUboot Internal Code Flash and Memory Mirror Function		R11AN1048 - Secure Bootloader Using MCUboot with SCE9 Protected mode and Internal Code Flash		
Image encryption (External flash)		R11AN0868 - Booting Encrypted Image on RA4 using MCUboot and QSPI	R11AN0567 - RA6 Booting Encrypted Image using MCUboot and QSPI		
Firmware update	R11AN1088 - RA0/RA2 MCUboot and USB-Based Firmware Updates				

Contents

1. Overview of MCUboot.....	6
1.1 History of MCUboot	6
1.2 MCUboot Functionality Overview	6
1.3 Overview of Application Booting Process	6
1.4 Application Update Strategies.....	6
2. Overview of Post-Quantum Cryptography	8
2.1 Quantum Computing and the Cryptographic Threat	8
2.1.1 What is Quantum Computing?	8
2.1.2 The Cryptographic Threat	9
2.2 What Is Post-Quantum Cryptography (PQC) and Why PQC Is Important Today?.....	9
2.2.1 What Is Post-Quantum Cryptography (PQC)?	9
2.2.2 Why PQC Is Important Today	10
2.2.3 Key and Signature Differences Among ECDSA, RSA, and ML-DSA	10
2.3 Post-Quantum Cryptography Support in the Renesas Flexible Software Package (FSP)	10
3. Running the Example project.....	12
3.1 Set Up the Hardware	13
3.1.1 Set up EK-RA8M2, EK-RA8M1, EK-RA6M5 and EK-RA4C1	13
3.1.2 Set up EK-RA6M3 and EK-RA2A2	15
3.2 Configure the Python Signing Environment.....	17
3.3 Running the EK-RA8M2 Swap Update Mode Example	19
3.3.1 Import the Projects	19
3.3.2 Compile the Projects	19
3.3.3 Debug the Applications and Boot the Primary Application	20
3.3.4 Downloading and Running the Secondary Applications	22
3.4 Running the EK-RA8M1 Swap Update Mode Example	23
3.4.1 Import the Projects	24
3.4.2 Compile the Projects	24
3.4.3 Debug the Applications and Boot the Primary Application	25
3.4.4 Downloading and Running the Secondary Applications	26
3.5 Running the EK-RA6M5 Swap Update Mode Example	27
3.5.1 Import the Projects	27
3.5.2 Compile the Projects	27
3.5.3 Debug the Applications and Boot the Primary Application	28
3.5.4 Downloading and Running the Secondary Applications	29
3.6 Running the EK-RA6M3 Swap Update Mode Example	30
3.6.1 Import the Projects	30
3.6.2 Compile the Projects	30
3.6.3 Debug the Applications and Boot the Primary Application	31

3.6.4	Downloading and Running the Secondary Applications	32
3.7	Running the EK-RA4C1 Swap Update Mode Example	33
3.7.1	Import the Projects	33
3.7.2	Compile the Projects	33
3.7.3	Debug the Applications and Boot the Primary Application	34
3.7.4	Downloading and Running the Secondary Applications	35
3.8	Running the EK-RA2A2 Direct XIP Update Mode and Dual Bank Mode Example	36
3.8.1	Import the Projects	36
3.8.2	Compile the Projects	36
3.8.3	Debug the Applications and Boot the Primary Application	37
3.8.4	Downloading and Running the Secondary Applications	38
4.	Creating the Bootloader Project	39
4.1	Start Bootloader Project Creation with e2 studio	39
4.2	Resolve the Configurator Dependencies	48
4.3	Using MCUboot Quick Setup	53
4.4	Compiling the Bootloader Project	55
4.5	Configuring the Python Signing Environment	55
5.	Using the Bootloader with Application	57
5.1	Generate the Primary User Application Project	57
5.2	Signing the User Application Image	60
5.3	MCUboot Memory Configuration with Renesas FSP Solution Project	63
5.3.1	How to Set Up the Renesas FSP Solution Project	63
5.3.2	Memory Configurations	65
5.3.3	Build the Solution Project	74
5.3.4	Optimizing SRAM Allocation	75
5.4	Erase the MCU	76
5.4.1	Initialize Device using Renesas Device Partition Manager (RDPM)	76
5.4.2	Initialize Device or Erase the MCU using the Renesas Flash Programmer (RFP)	76
5.4.3	Use the J-Flash Lite	78
5.5	Configure the Debugger	80
5.5.1	Single core projects	80
5.5.2	Dual core RA8 MCU projects	82
5.6	Mastering and Delivering a New Application	84
5.6.1	Prepare a Secondary Image	84
5.6.2	Downloading and Booting the Secondary Application	87
6.	Production Support Considerations	88
7.	Website and Support	89
	Revision History	90

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 repository was later migrated from JuulLabs to the [mcu-tools github project](https://github.com/mcu-tools/mcu-boot). In 2020, MCUboot was moved under the Linaro Community Project umbrella as an open-source project, and it now resides under Trusted Firmware (<https://www.trustedfirmware.org/projects/mcu-boot/>).

1.2 MCUboot Functionality Overview

MCUboot handles the firmware integrity and authenticity check after startup and the firmware switch part of the firmware update process. The operation of switching the firmware from the original image to a new image depends on the image upgrade method. The image upgrade methods are described in section 1.4. Downloading a new firmware version is out of scope for MCUboot. This functionality is provided by the application project or tools described in this document.

1.3 Overview of Application Booting Process

For applications using MCUboot, the MCU memory is partitioned into the following regions: MCUboot, Primary Application, Secondary Application, and Scratch Area. Below is an example of the single-image MCUboot memory map. For additional details on the MCUboot memory layout, refer to the Flash Map section on the MCUboot website.

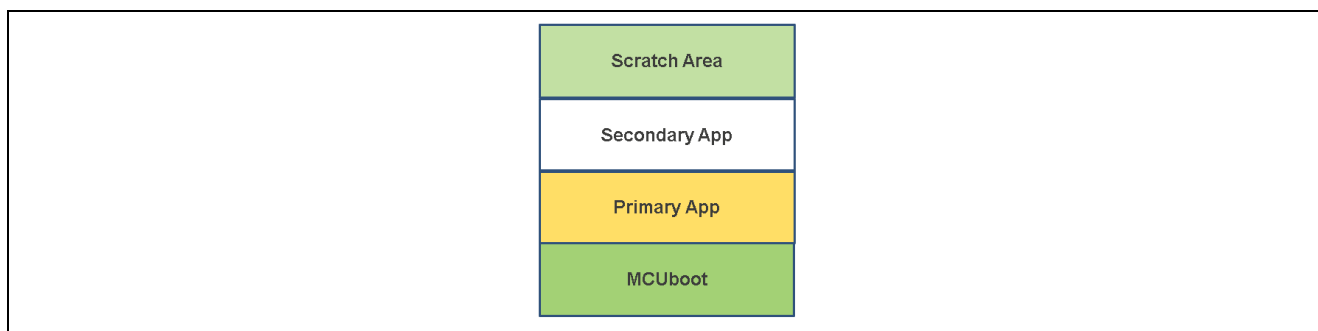


Figure 1. Single Image MCUboot Memory Flash Map

MCUboot functionality during the boot and update phases adheres to the following process:

1. The bootloader starts when the MCU exits from reset state.
2. If images in the Secondary App memory are marked for update, the bootloader performs the following actions:
 - A. The bootloader verifies the integrity and authenticity of the Secondary image.
 - B. Upon successful authentication, the bootloader switches to the new image based on the selected update method.
 - C. The bootloader boots the new image.
3. If there is no new image in the Secondary App memory region, the bootloader authenticates the Primary applications and boots the Primary image.

Application authentication is configurable in terms of the authentication methods and whether the authentication is performed by MCUboot. The firmware image can be authenticated by hash (SHA-256) and digital signature validation.

There is a signing tool included with MCUboot: `imgtool.py`. This tool provides services for creating Root keys, key management, and signing and packaging an image with version controls. Read the MCUboot documentation to understand and use these operations.

1.4 Application Update Strategies

The following update strategies are supported by MCUboot. The Renesas FSP MCUboot Module supports one or more of the following strategies depending on the FSP version. The analysis of any advantages or disadvantages is based on the MCUboot functionality, not the FSP MCUboot Module functionality. In addition, this application note is not intended to provide all details on the MCUboot application update strategies. For more details on these update strategies, refer to the MCUboot design page:

<https://github.com/mcu-tools/mcu-boot/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).

- Advantages:
 - 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.
- Disadvantages:
 - 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.

- **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.

- Advantages:
 - 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.
 - Fail-safe and resistant to power-cut failures.
 - Encrypted image support is available when using external flash.
- Disadvantages:
 - 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, wearing the Scratch sectors out faster.

Swap upgrade mode is supported by Renesas RA FSP v3.0.0 or later.

- **Direct execute-in-place (XIP)**

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.

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

Direct execute-in-place is supported by Renesas FSP v3.4.0 or later.

2. Overview of Post-Quantum Cryptography

Post-Quantum Cryptography (PQC) refers to a new generation of cryptographic algorithms designed to remain secure against attacks from both classical computers and future large-scale quantum computers. PQC aims to protect digital communications, software authenticity, data integrity, and secure transactions against future quantum-computing attacks that may compromise current public-key cryptographic systems.

2.1 Quantum Computing and the Cryptographic Threat

2.1.1 What is Quantum Computing?

Quantum computing is an emerging computing paradigm that **uses quantum-mechanical** principles **such as superposition** and entanglement to perform certain calculations much more efficiently than classical computers.

- **Classical computers** store and process information using bits. A bit is either 0 or 1, and a computer is built from billions of these simple 0/1 bits.
- **Quantum computers** use **qubits** (quantum bits). Instead of being strictly 0 or 1, a qubit can be **0, 1, or both at the same time** (a property called **superposition**).

Superposition allows multiple qubits to represent many possible states at the same time, enabling quantum computers to solve certain problems more efficiently than classical computers.

- Two classical bits can represent only **one** of four possible values at a time:
 - 00 (0) or
 - 01 (1) or
 - 10 (2) or
 - 11 (3)
- Two qubits can represent a superposition of all **four combinations at the same time**:
 - $\psi = a \cdot 00 + b \cdot 01 + c \cdot 10 + d \cdot 11$

The coefficients a, b, c, and d represent the contribution of each state. The probability of each state is the square of its coefficient, and the total probability is always 1.

 - $a^2 + b^2 + c^2 + d^2 = 1$

For example, when calculating 7^2 , a classical computer evaluates the expression and returns 49. In contrast, before measurement, a quantum computer can represent multiple possible states, such as 7^0 , 7^1 , 7^2 , and 7^3 , at the same time.

Table 1 Comparison between Classical and Quantum computing

Feature	Classical Computing	Quantum Computing
Basic unit	Bit (always definitely 0 or 1)	Qubit (a superposition of 0 and 1, yielding one definite value only when measured)
Processing style	Deterministic operations; many cores and GPUs run in parallel.	Superposition and interference across many qubits.
Key quantum feature	None (states are independent)	Entanglement (correlations between qubits with no classical analog)
Physical basis	Transistors (electrical states)	Photons, trapped ions, superconducting circuits, etc.
Reliability	Bits are extremely stable; errors are rare	Qubits are fragile, decohere quickly, and need heavy error correction
Best suited for	General-purpose, everyday computing	Specific tasks: factoring (Shor), unstructured search (Grover), simulating quantum systems
Maturity	Mature, ubiquitous	Early-stage, experimental (NISQ era)

While quantum computers have the potential to accelerate scientific research, optimization, and simulation, they also pose a significant long-term threat to modern cryptography.

2.1.2 The Cryptographic Threat

Most current public-key cryptographic systems, including Rivest–Shamir–Adleman (**RSA**) and Elliptic Curve Cryptography (**ECC**), **rely on mathematical problems** that are computationally infeasible for classical computers to solve. However, **quantum algorithms** such as Shor's algorithm **could efficiently solve these problems, allowing a sufficiently powerful, fault-tolerant quantum computer to:**

- Derive private keys from their corresponding public keys
- Decrypt protected communications
- Forge digital signatures
- Bypass authentication mechanisms

As a result, many security mechanisms used today, including **TLS, secure boot, firmware signing, certificates, and authentication systems**, would **become vulnerable once such a quantum computer exists**.

Importantly, this threat does not require waiting for quantum computers to arrive to start causing harm: adversaries can employ a "**harvest now, decrypt later**" strategy, **recording encrypted data today for decryption once capable quantum hardware becomes available**. This makes migration urgent for any data requiring long-term confidentiality.

Symmetric-key cryptography (e.g., AES) and cryptographic hash functions are comparatively less impacted, since Grover's algorithm provides only a quadratic speedup against them, a risk that can be mitigated by using larger key sizes.

2.2 What Is Post-Quantum Cryptography (PQC) and Why PQC Is Important Today?

Embedded systems frequently remain deployed for 10-20 years. A device deployed today may still be operational when practical quantum computers become available. Therefore, firmware authentication schemes must begin transitioning to quantum-resistant algorithms now.

2.2.1 What Is Post-Quantum Cryptography (PQC)?

Post-quantum cryptography is a class of cryptographic algorithms designed to remain secure against attacks from both classical and quantum computers. It replaces today's quantum-vulnerable public-key algorithms (such as RSA and ECC) with new algorithms based on mathematical problems believed to resist both classical and quantum attacks. Unlike quantum cryptography, PQC does not require specialized quantum hardware and can run on conventional processors and embedded systems.

The transition focuses on public-key cryptography because that is where the quantum threat is acute. Symmetric algorithms (e.g., AES) and hash functions are affected only modestly (a quantum attack effectively halves their security, which can be countered by using larger key or output sizes) so they do not require replacement in the way public-key algorithms do.

Common PQC algorithm families under NIST consideration include (among others):

- Lattice-based cryptography
- Hash-based cryptography
- Code-based cryptography

The first NIST-standardized algorithms:

- **ML-KEM** (formerly CRYSTALS-Kyber): lattice-based, **for key establishment**
- **ML-DSA** (formerly CRYSTALS-Dilithium): lattice-based, **for digital signatures**
- **SLH-DSA** (formerly SPHINCS+): **a hash-based digital signature alternative**

Code-based cryptography (e.g., Classic McEliece) remains under NIST evaluation for future standardization.

2.2.2 Why PQC Is Important Today

Although large-scale quantum computers capable of breaking RSA and ECC are not yet available, organizations are already preparing for the transition to PQC. Several factors drive this urgency:

- **Long product lifetimes:** Embedded, industrial, automotive, and IoT systems often remain deployed for 10-20 years or longer.
- **Harvest Now, Decrypt Later (HNDL):** Adversaries may collect encrypted data today and decrypt it in the future once quantum computers become available.
- **Migration complexity:** Replacing cryptographic infrastructure requires significant updates to software, hardware, protocols, and operational processes.

To prepare for the post-quantum era, the National Institute of Standards and Technology (NIST) published its first three PQC standards in August 2024:

- **FIPS 203:** ML-KEM (**M**odule-**L**attice-Based **K**ey **E**ncapsulation **M**echanism)
- **FIPS 204:** ML-DSA (**M**odule-**L**attice-Based **D**igital **S**ignature **A**lgorithm)
- **FIPS 205:** SLH-DSA (Stateless Hash-Based Digital Signature Algorithm)

Organizations are encouraged to begin migrating to these finalized standards now rather than waiting for additional algorithms to be finalized.

For embedded systems and secure firmware applications, **PQC is particularly important because secure boot, firmware authentication, and secure communications often rely on digital signatures and key exchange mechanisms. Early adoption and evaluation of PQC algorithms can help ensure long-term security for future products and deployed devices.**

2.2.3 Key and Signature Differences Among ECDSA, RSA, and ML-DSA

- RSA (Rivest–Shamir–Adleman) is based on Integer factorization.
- ECDSA (Elliptic Curve Digital Signature Algorithm) is based on Geometric structure of elliptic curves over finite fields.
- ML-DSA uses Module Learning With Errors (MLWE) and Short Integer Solution (SIS) which is based on Algebraic lattice problems.

Table 2 Size comparison between RSA-2048, ECDSA P-256 and ML-DSA-44

Features	RSA-2048	ECDSA P-256	ML-DSA-44
Private key	~512 bytes	32 bytes	2,528 bytes
Public key	~256 bytes	64 bytes	1,312 bytes
Signature	256 bytes	64 bytes	2,420 bytes

RSA was introduced in 1977. It is based on large-integer arithmetic and uses relatively large keys, but it is conceptually simple. ECDSA was introduced in 1992. It is more efficient than RSA and uses points (x,y) on an elliptic curve, resulting in smaller keys. It is mostly used in MCUboot today. However, these algorithms are vulnerable to Shor's algorithm on a sufficiently powerful quantum computer.

ML-DSA was standardized in 2024. It uses polynomial vectors and matrices, including 256-coefficient polynomials and NTT-based polynomial multiplication. It is considered quantum resistant.

2.3 Post-Quantum Cryptography Support in the Renesas Flexible Software Package (FSP)

As part of its ongoing security roadmap, Renesas has added native support for Post-Quantum Cryptography (PQC) within FSP, enabling embedded developers to evaluate and adopt quantum-resistant algorithms directly on supported RA MCU hardware.

This integration allows PQC to be used for the same use cases where classical public-key cryptography is applied today in embedded systems (most notably secure boot, firmware signing/verification, and secure communications) without requiring a separate cryptographic toolchain or external library integration.

FSP includes support for **ML-DSA (Module-Lattice-Based Digital Signature Algorithm)**, the NIST-standardized digital signature scheme (FIPS 204), based on the CRYSTALS-Dilithium submission. Current FSP releases support all three NIST parameter sets:

- **ML-DSA-44** - key generation, signing, and verification
- **ML-DSA-65** - key generation, signing, and verification
- **ML-DSA-87** - key generation, signing, and verification

A subsequent FSP release updated the underlying PQC-Lib to v2.00, which added **ML-DSA-87** support.

Table 3 Comparison of ML-DSA Algorithms

Algorithm	Security Level	Typical Use
ML-DSA-44	Lowest	Resource-constrained devices
ML-DSA-65	Medium	General-purpose
ML-DSA-87	Highest	High-security systems

Secure boot integration: Post-quantum signature verification is not currently supported in upstream MCUboot, and existing implementations are provided by vendor-specific ports. From FSP v6.4.0 onward, MCUboot support in FSP has been extended to accept ML-DSA-44, ML-DSA-65, and ML-DSA-87 as signature verification options for firmware images. MCUboot requires digital signatures to authenticate firmware images. Therefore, this application note focuses on ML-DSA. ML-KEM is a key-establishment algorithm and is not used for firmware signature verification.

A few implementation notes worth carrying into the document:

- The current ML-DSA implementation is **too large for devices with less than 32 KB of RAM**, which is a real constraint for smaller RA MCUs.

PQC support is exposed through both a standalone PQC module and via **PSA Crypto** (rm_psa_crypto), though the PSA implementation is based on a **release candidate** of PQC-Lib v2.00 and may see backward-incompatible changes before MbedTLS officially supports ML-DSA/ML-KEM upstream. PQC is currently supported in Compatibility Mode only and is not supported in Protected Mode.

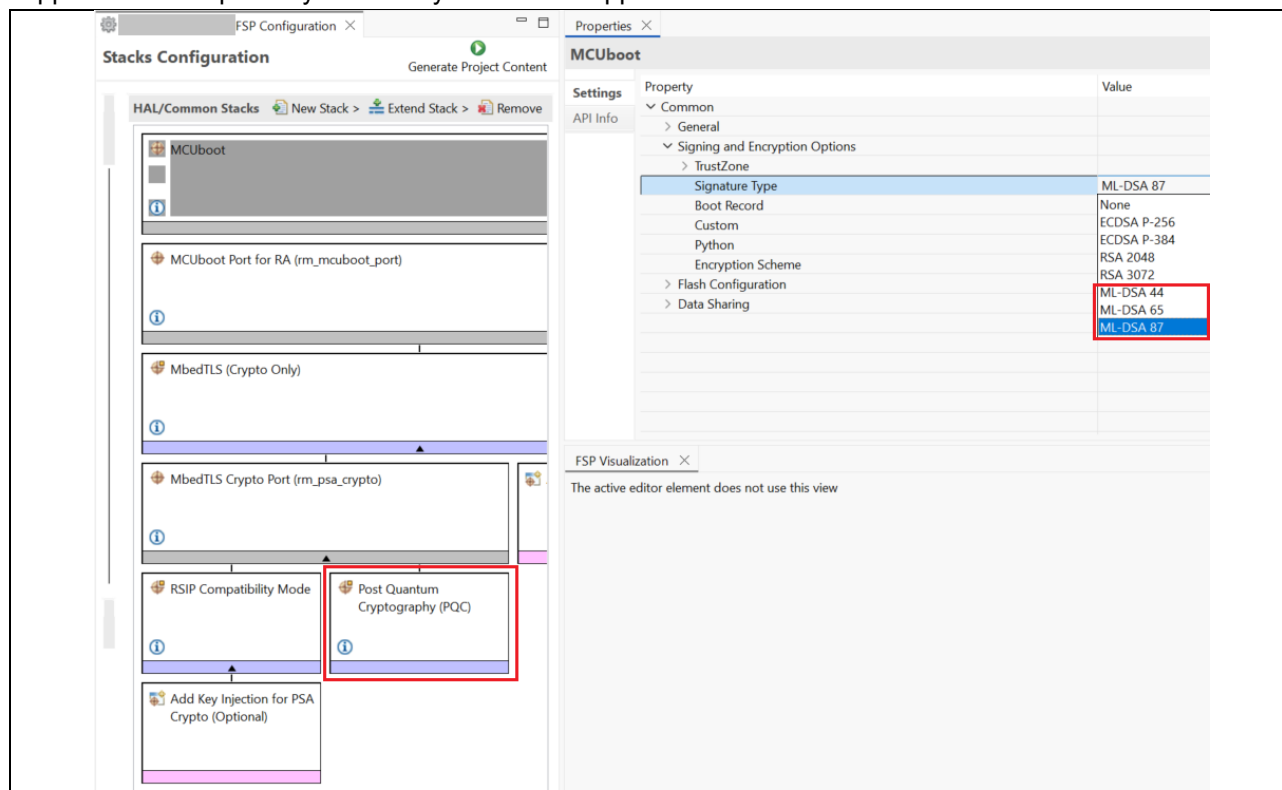


Figure 2. PQC support in the FSP

3. Running the Example project

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](#).

Primary and secondary applications used in these example projects are Blinky Bare-Metal samples. To recreate the primary applications included memory map definitions used in this section, refer to [Section 5](#).

Extract MCUboot_with_PQC.zip. The package contains six example project archives. Each package contains example projects for the specific MCU which include bootloader project and example application projects.

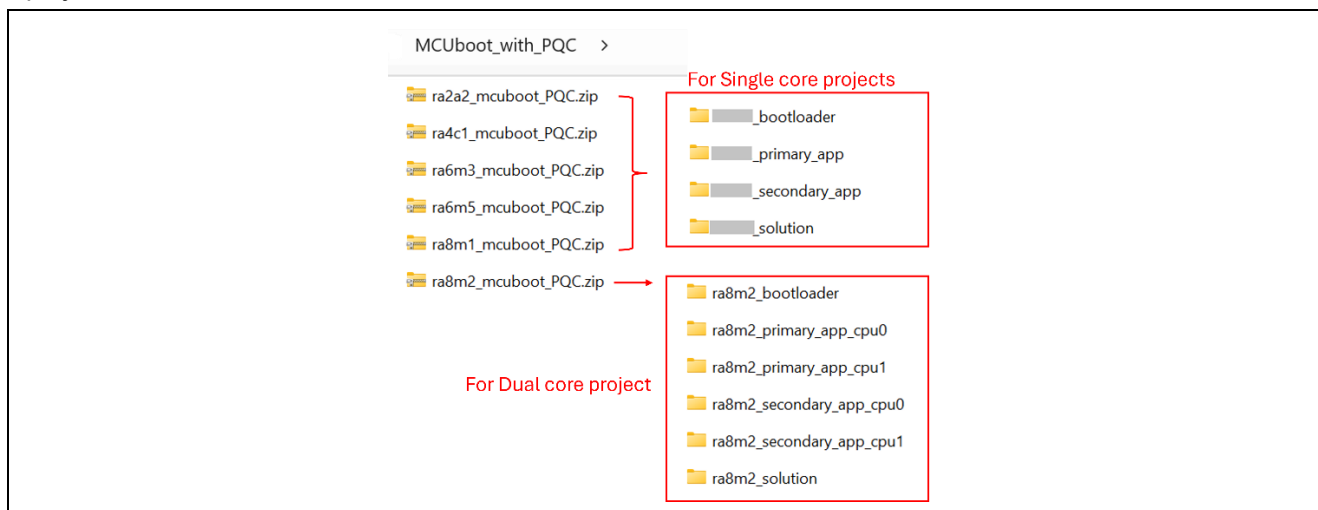


Figure 3. Example Projects with Bootloader Support

Set up the Python development environment by following section [3.2](#). Note that this step only needs to be performed once. Set up the hardware for RA8M2, RA8M1, RA6M5 and RA4C1 by following section [3.1.1](#). Set up the hardware for RA6M3 and RA2A2 by following section [3.1.2](#).

Table 4 Overview MCUboot information on Example package

Project name	Core	MCU	Board	Upgrade method	Signature type
ra2a2_mcuboot_PQC	CM23	RA2A2	EK-RA2A2	Direct XIP (Dual bank mode - Instant Bank Swap)	ML-DSA 44
ra4c1_mcuboot_PQC	CM33	RA4C1	EK-RA4C1	Swap	ML-DSA 44
ra6m3_mcuboot_PQC	CM4	RA6M3	EK-RA6M3	Swap	ML-DSA 87
ra6m5_mcuboot_PQC	CM33	RA6M5	EK-RA6M5	Swap	ML-DSA 87
ra8m1_mcuboot_PQC	CM85	RA8M1	EK-RA8M1	Swap	ML-DSA 87
ra8m2_mcuboot_PQC	CM85 & CM33	RA8M2	EK-RA8M2	Swap (Multi-image update)	ML-DSA 87

3.1 Set Up the Hardware

3.1.1 Set up EK-RA8M2, EK-RA8M1, EK-RA6M5 and EK-RA4C1

- Jumper settings: Follow settings in Default Jumper Configurations of User Manuals:
 EK-RA8M2: [EK-RA8M2 v1 User's Manual](#)
 EK-RA8M1: [EK-RA8M1 v1 User's Manual](#)
 EK-RA6M5: [EK-RA6M5 v1 - User's Manual](#)
 EK-RA4C1: [EK-RA4C1 v1 User's Manual](#)
- Connect J10 using a USB micro to B cable from EK board (RA8M1/RA6M5) to the development PC to provide power and debug connection using the on-board debugger.
- Connect J10 using a USB Type-C cable from EK board (RA8M2/RA4C1) to the development PC to provide power and debug connection using the on-board debugger.

Once the device is powered up, initialize the MCU prior to exercising the bootloader project.

This can be achieved using the Renesas Device Partition Manager.

1. Power cycle the board, launch e² studio, and open the Renesas Device Partition Manager.

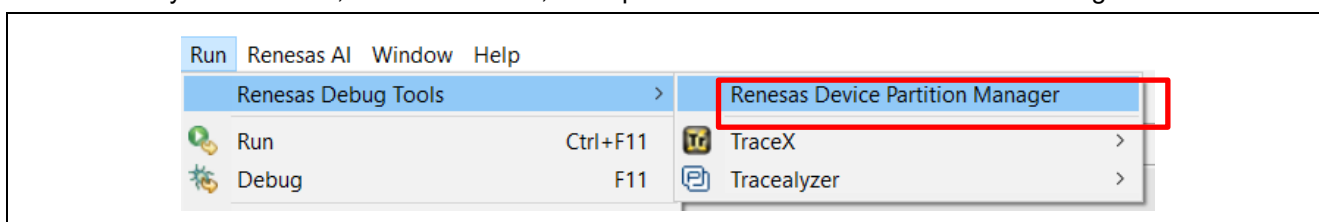


Figure 4. Open Renesas Device Partition Manager

2. Select **Read current device information**.

For RA8M2 and RA8M1, 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 the MCU that is in the OEM state.

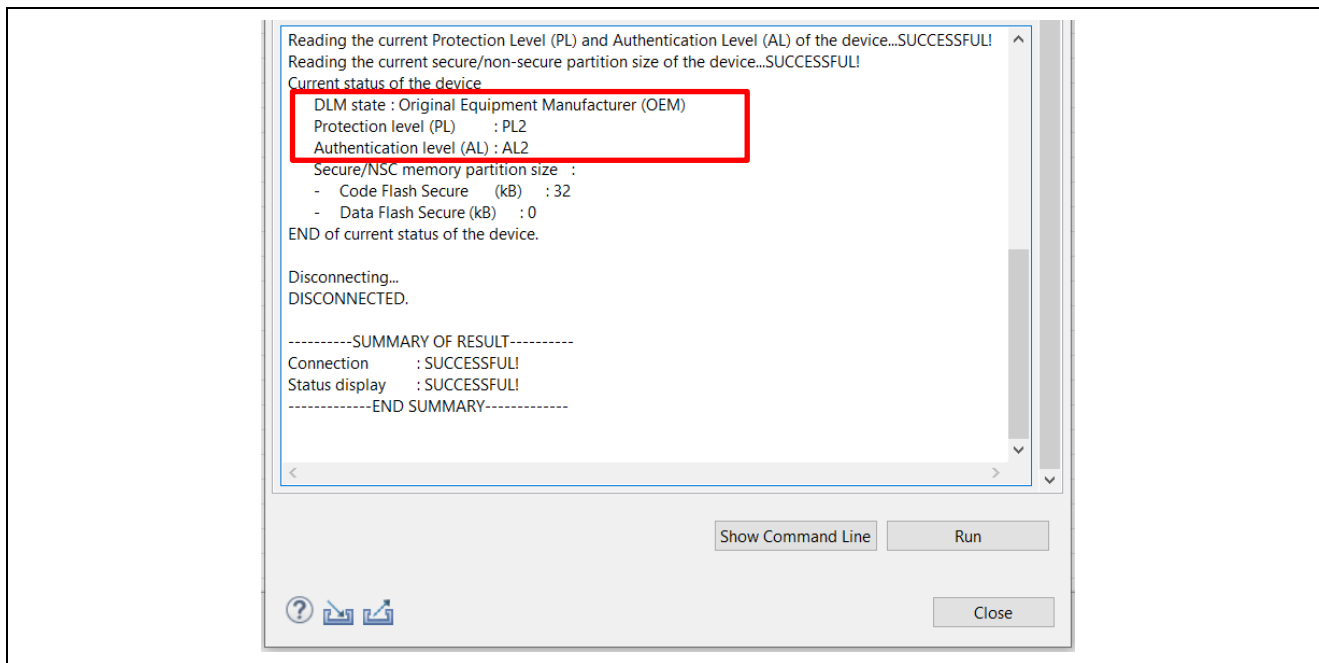


Figure 5. Read the Device Lifecycle States on RA8M2/RA8M1

For RA6M5 and RA4C1, 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 the MCU that is in the SSD state.

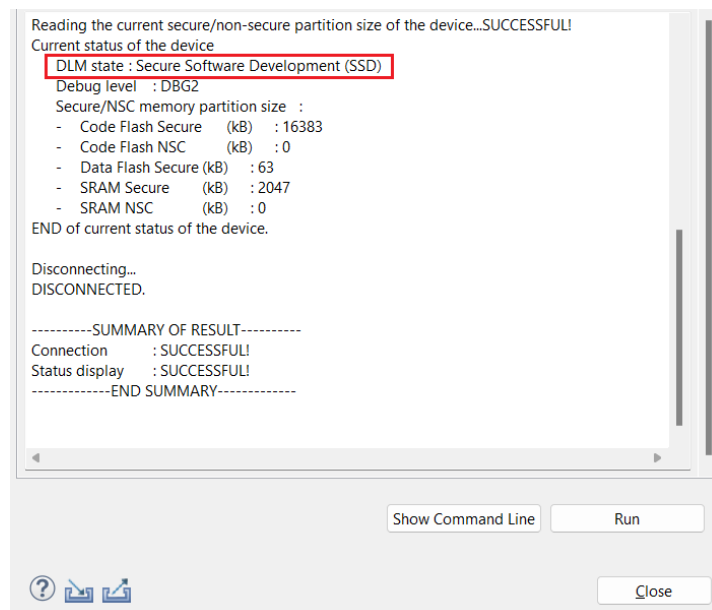


Figure 6. Read the Device Lifecycle States on RA6M5/RA4C1

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

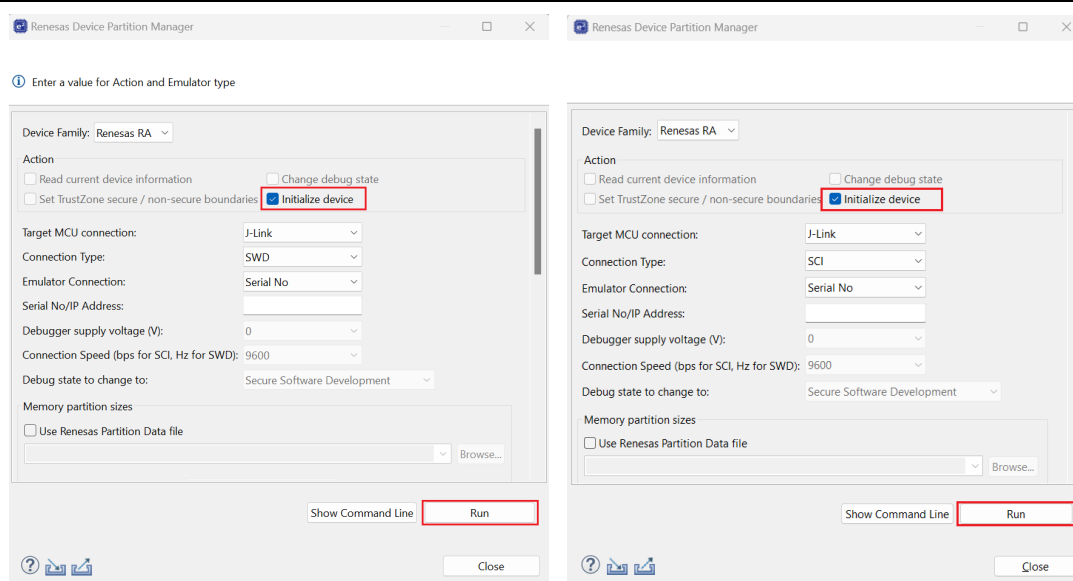


Figure 7. Initialize device using Renesas Device Partition Manager

The entire flash will be erased if there are not permanently locked down sections. In addition, if the RA8 device is in the PL1 or PL0 state, the device will be initialized to the PL2 state. If the RA6 and RA4 devices are in the NSECSD or DPL state, the devices will be initialized to the SSD state. In most cases, the connection type is SWD. However, for the EK-RA8M1, SCI should be selected as the connection type when initializing the device using RDPM.

3.1.2 Set up EK-RA6M3 and EK-RA2A2

- Jumper settings: Follow settings in Default Jumper Configurations of User Manuals:
EK-RA6M3: [EK-RA6M3 v1 – User's Manual](#)
EK-RA2A2: [EK-RA2A2 v2 – User's Manual](#)
- Connect J10 using a USB micro to B cable from EK board to the development PC to provide power and debug connection using the on-board debugger.

Once the device is powered up, erase the entire MCU flash prior to exercising the bootloader project. This can be achieved using the Renesas Flash Programmer (RFP) or J-Flash Lite.

3.1.2.1 Use the Renesas Flash Programmer (RFP)

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

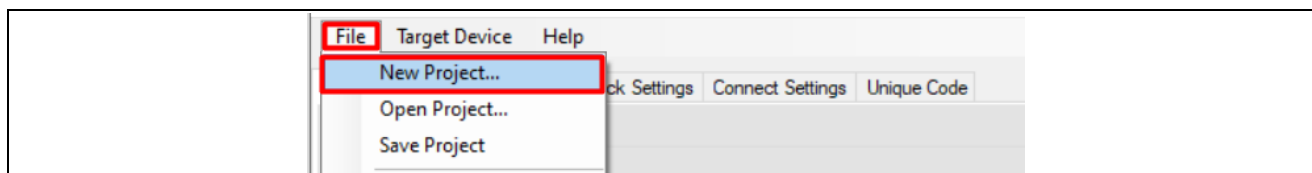


Figure 8. Create a New RFP Project

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

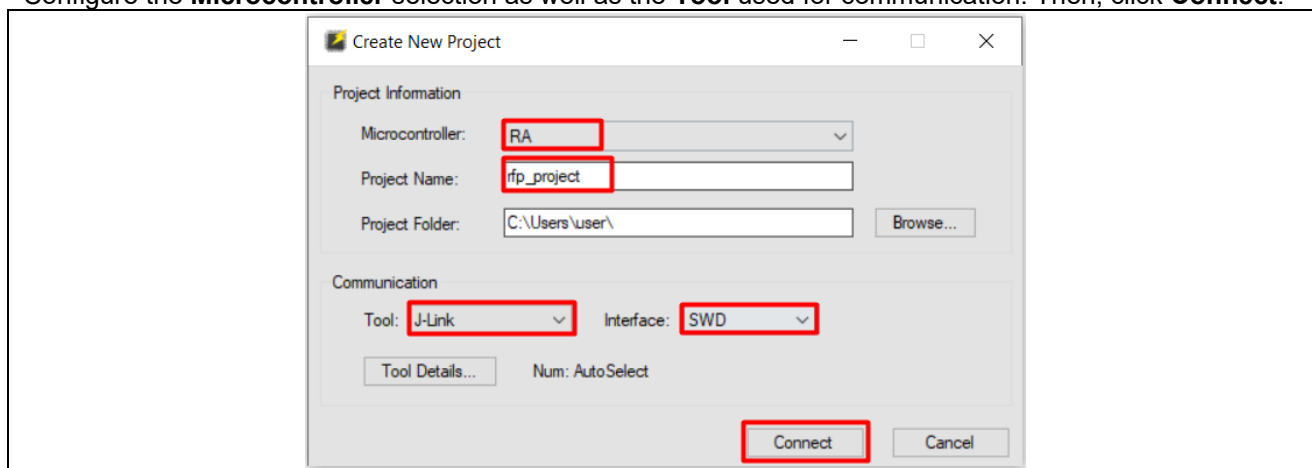


Figure 9. Configure a New Project

On successful connection, open the **Block Settings** tab to check the Code Flash configurations.

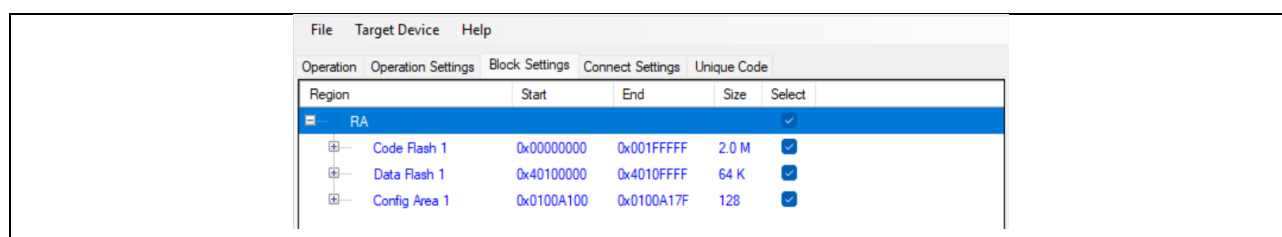


Figure 10. Code Flash configurations on RA6M3

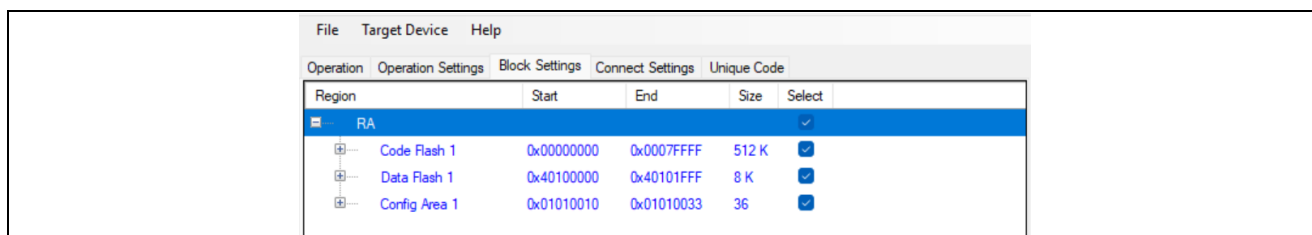


Figure 11. Code Flash configurations on RA2A2

Navigate to the **Operation Settings** tab and choose the **Erase** command.

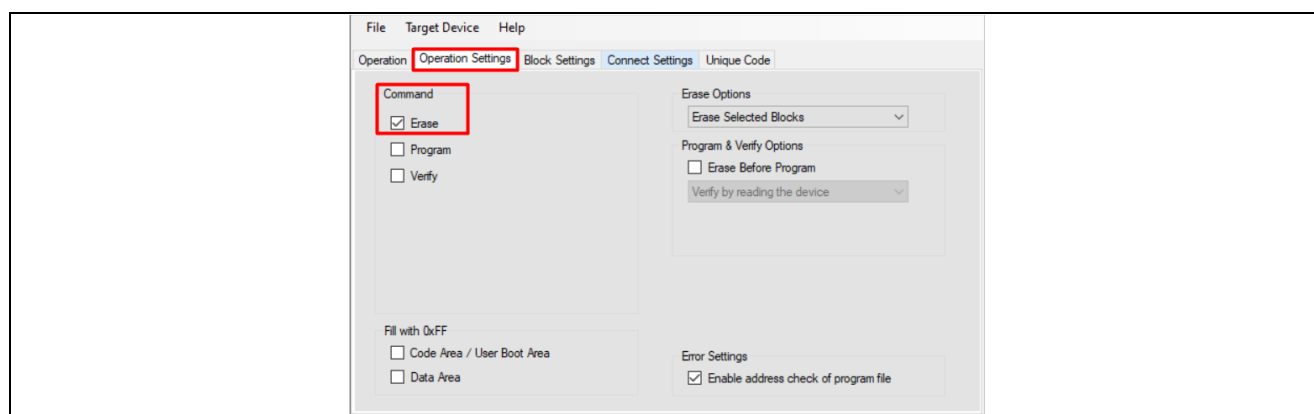


Figure 12. Choose the Erase Command

Choose the **Operation** tab and select **Start**.

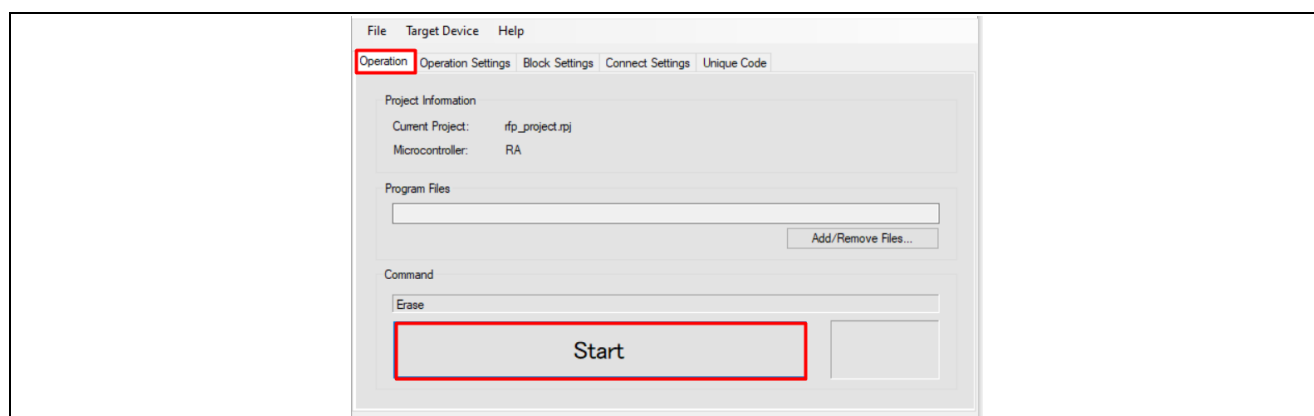


Figure 13. Start Erase Chip

When the chip is erased, **Operation completed** will be displayed.



Figure 14. Erase Chip Succeeded

3.1.2.2 Use the J-Flash Lite

Launch J-Flash Lite. Select the **Device**, debug **Interface**, and communication speed.

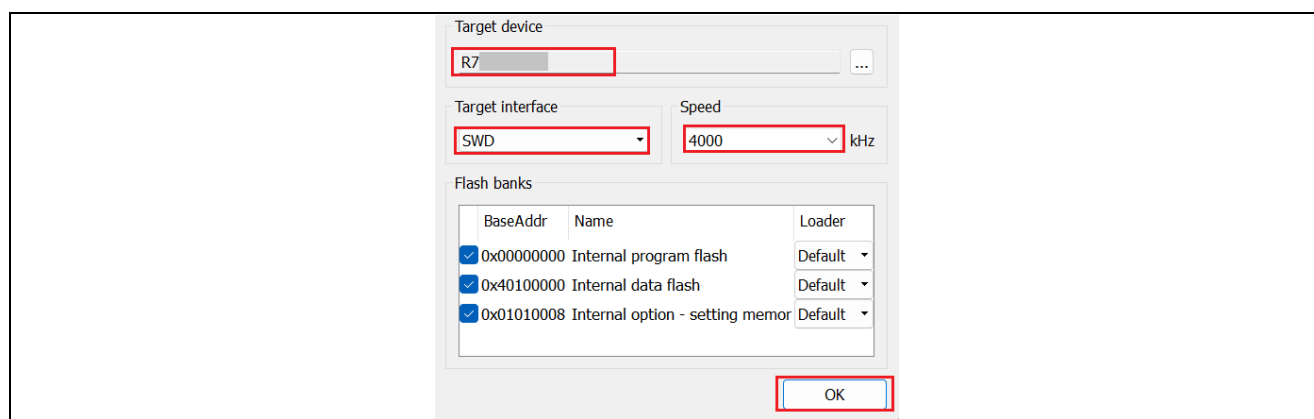


Figure 15. Launch J-Flash Lite

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

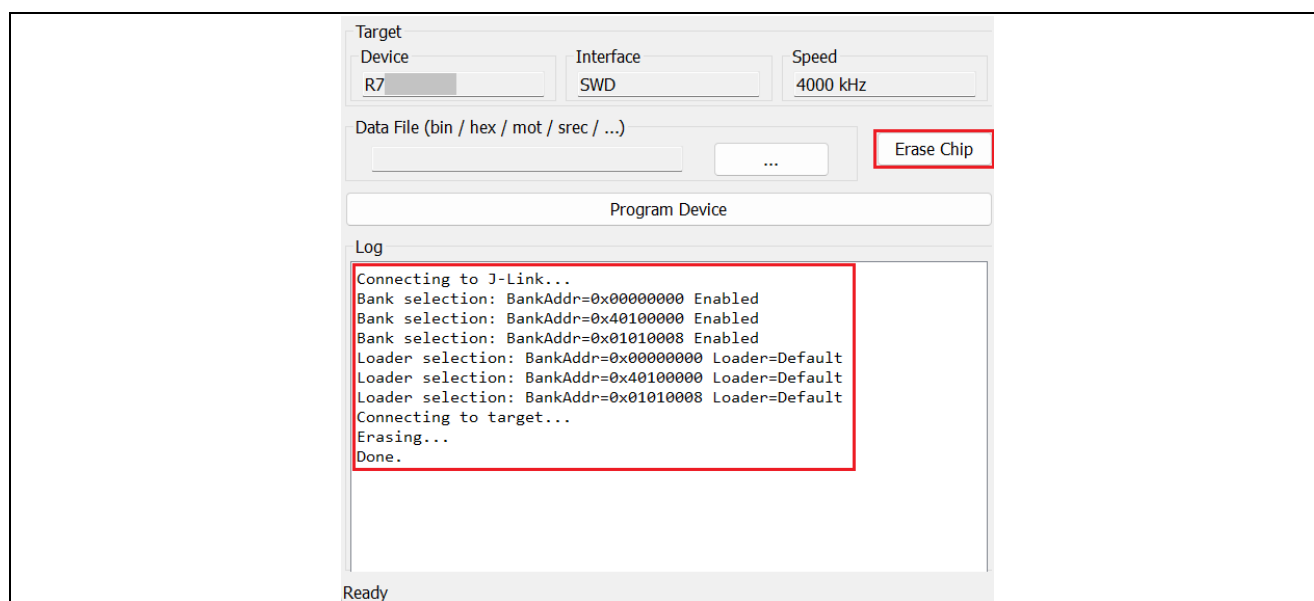


Figure 16. Erase MCU Succeeded

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.

Download and Install Python v3.13.7 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 example project sets (refer to **Figure 3**) choose the set of projects you would like to do first.
- Import that set of projects into a workspace. In this example, we assume you have chosen to import the projects **ra8m2_mcuboot_PQC** and build the **ra8m2_solution** to ensure all project contents were generated.
- Navigate to the folder **\MCUboot** in the bootloader project included, for example, **ra8m2_mcuboot_PQC\ra8m2_bootloader\ra\mcu-tools\MCUboot**, right click, and select **Command Prompt**. Depending on your PC policy, administrator privileges may be required when running Command Prompt. This opens a command window with the path set to the **\mcu-tools\MCUboot** folder.

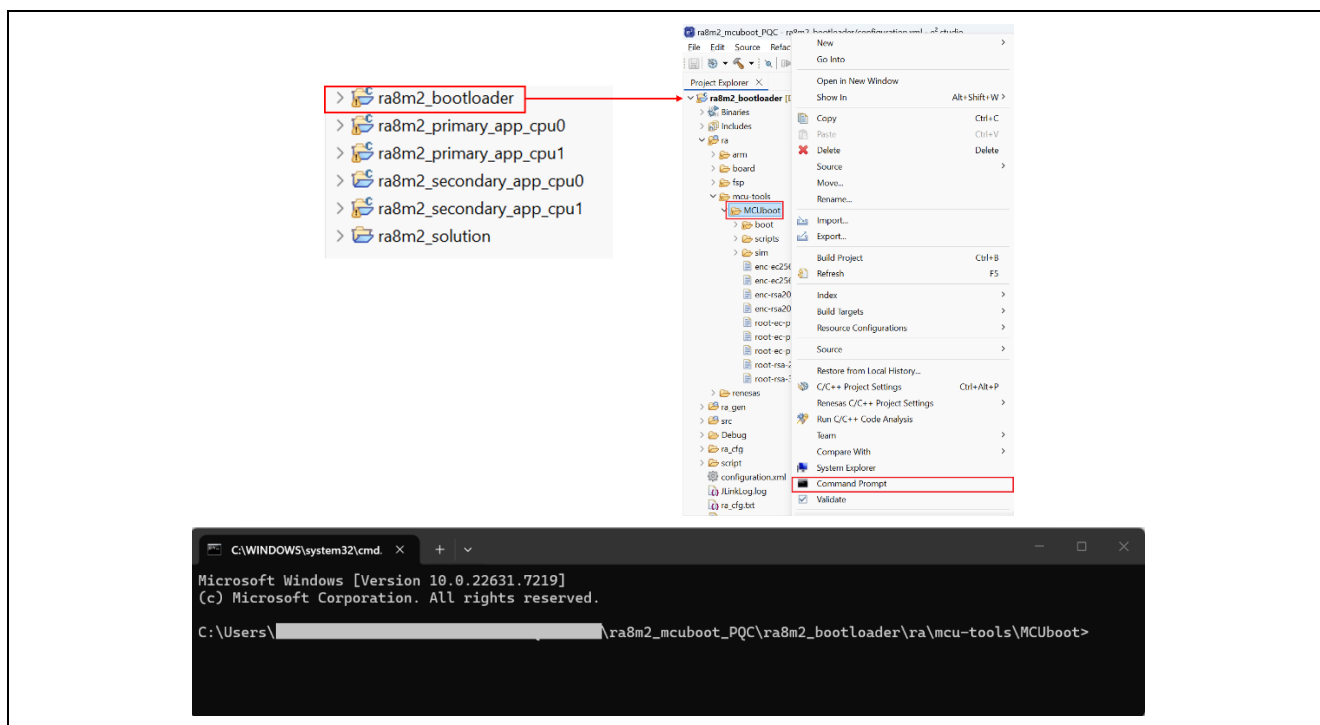


Figure 17. Open the Command Prompt

- 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 3.13.7 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.

3.3 Running the EK-RA8M2 Swap Update Mode Example

3.3.1 Import the Projects

Import the Projects ra8m2_mcuboot_PQC.zip to a Workspace

The following example projects are included in this folder and the memory Flash map overview used in the project:

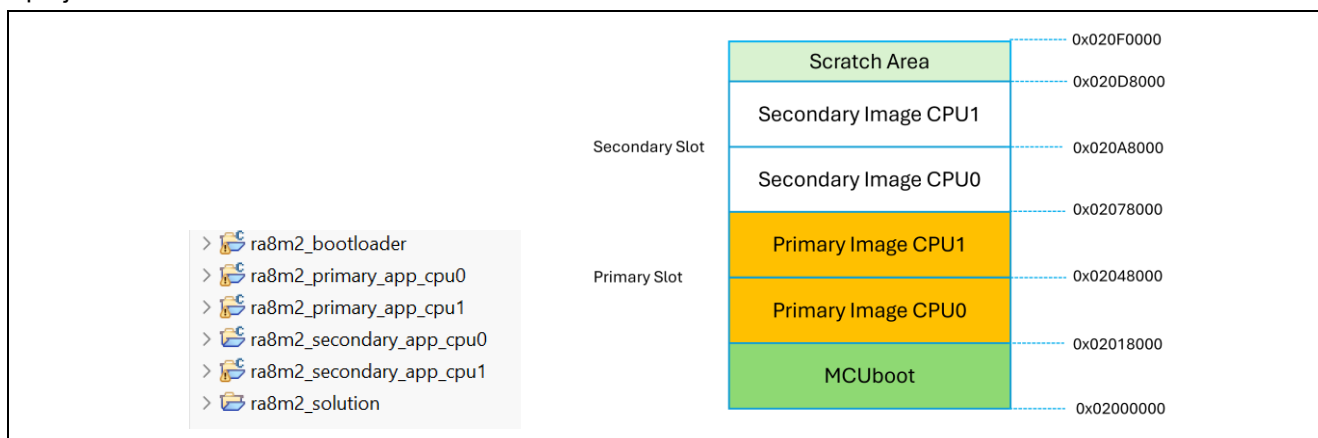


Figure 18. Example Projects for RA8M2 Swap Update Mode

- Project ra8m2_bootloader is the bootloader project.
- Project ra8m2_primary_app_cpu0 is the initial Primary application project on CPU0. This project blinks the LED1 on the EK-RA8M2 kit.
- Project ra8m2_primary_app_cpu1 is the initial Primary application project on CPU1. This project blinks the LED2 on the EK-RA8M2 kit.
- Project ra8m2_secondary_app_cpu0 is the Secondary application project on CPU0. This project blinks the blue LED3 on the EK-RA8M2 kit.
- Project ra8m2_secondary_app_cpu1 is the Secondary application project on CPU1. This project blinks the blue LED1 on the EK-RA8M2 kit with slower frequency.
- Project ra8m2_solution is the solution project. This project configures memory areas.

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

3.3.2 Compile the Projects

The solution project includes the bootloader and primary application projects. Right click on solution project and select **Build Project (Ctrl+B)** to build the bootloader and primary application projects. The bootloader image is located under the \Debug folder and is named <project_name>.elf. 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, /ra8m2_primary_app_cpu0/Debug/ra8m2_primary_app_cpu0.bin.signed)

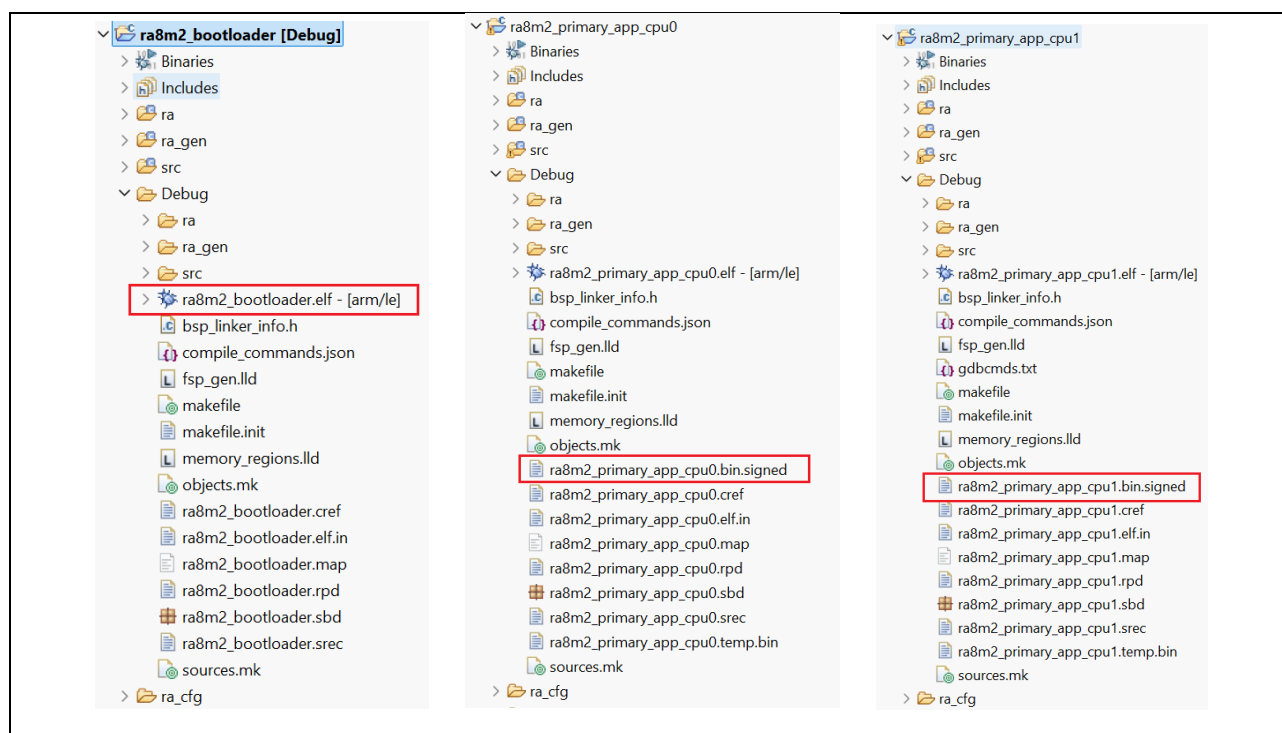


Figure 19. Successfully built images on RA8M2

3.3.3 Debug the Applications and Boot the Primary Application

Right-click on project `ra8m2_primary_app_cpu1` and select **Debug As > Debug Configurations**.

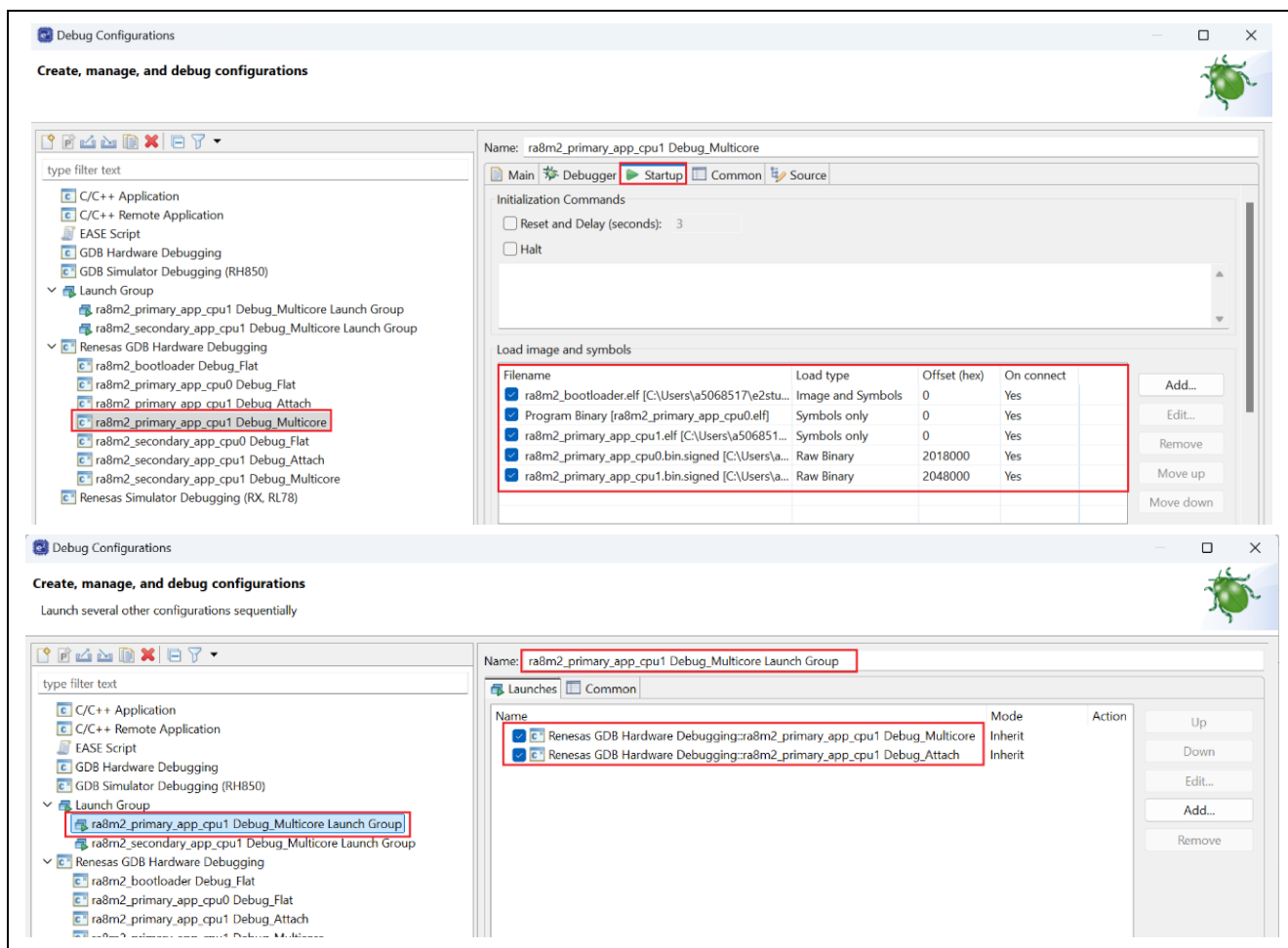


Figure 20. Debug Configuration RA8M2

Drop down **Launch Group** and select **ra8m2_primary_app_cpu1 Debug_Multicore Launch Group**. Click **Debug**.

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

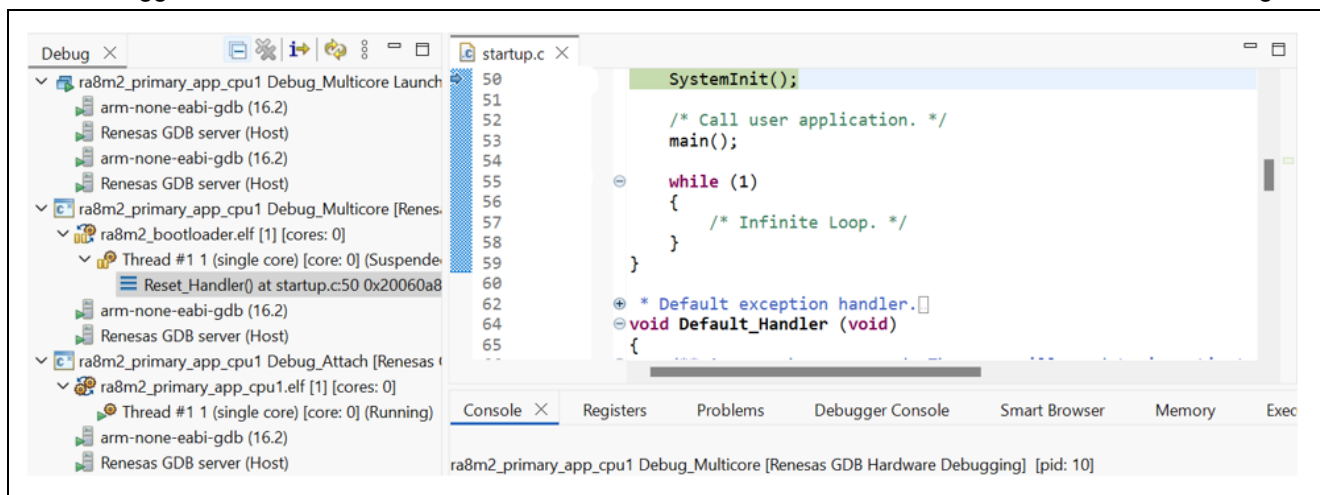


Figure 21. Start the RA8M2 Application Execution on CPU0

Click Resume twice  and boot the Primary image CPU0. LED1 should be blinking.

The debugger now should be at the reset handler in the primary image CPU1. The address is in the primary image CPU1.



Figure 22. Start the RA8M2 Application Execution on CPU1

Click Resume  and boot the Primary image CPU1. LED2 should be blinking.

3.3.4 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.

Note:

Ensure compiling ra8m2_secondary_app_cpu0 and ra8m2_secondary_app_cpu1 before starting. The signed image is located under the \Debug folder and is named <project_name>.bin.signed (for example, /ra8m2_secondary_app_cpu0/Debug/ra8m2_secondary_app_cpu0.bin.signed).

1. Press  to pause the program which now is primary application on CPU1.

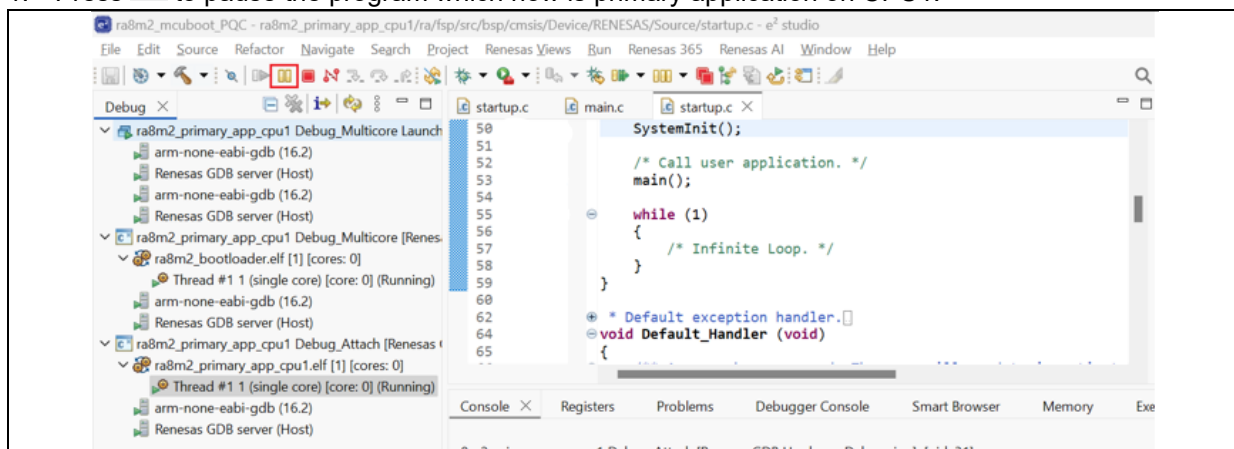


Figure 23. Pause primary application on CPU1

- Load the new CPU1 application images to the Secondary slot region using the Ancillary loading capability  from the top of the e² studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x020A8000.

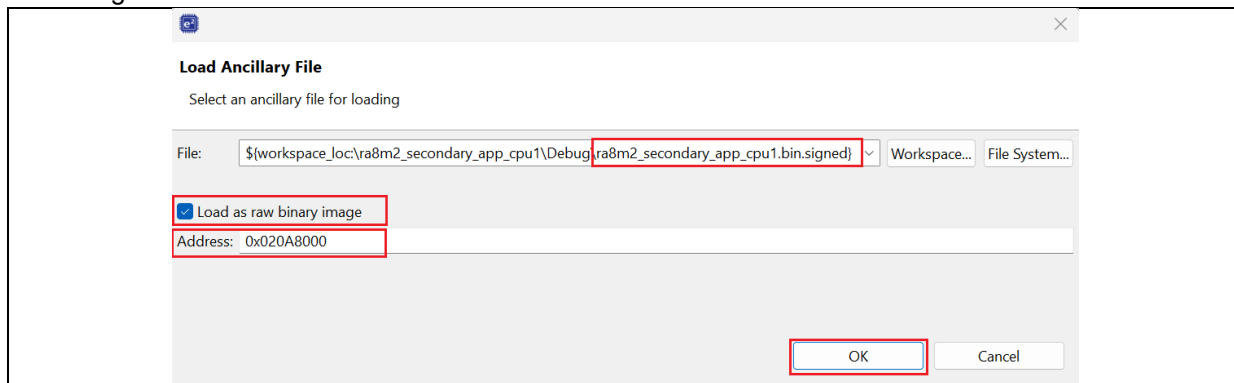


Figure 24. Load the Secondary Application Image CPU1

- Select CPU0 Debugging section. Press  to pause the program on CPU0.

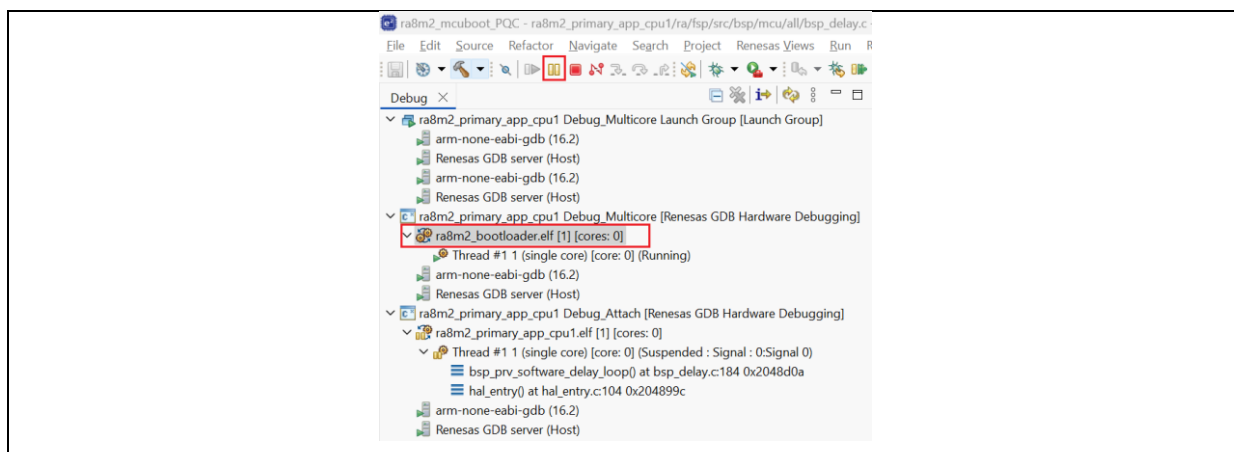


Figure 25. Pause primary application on CPU0

- Load the new CPU0 application images to the Secondary slot region using Ancillary loading. Select **Load as raw binary image** and configure the **Address** to 0x02078000.

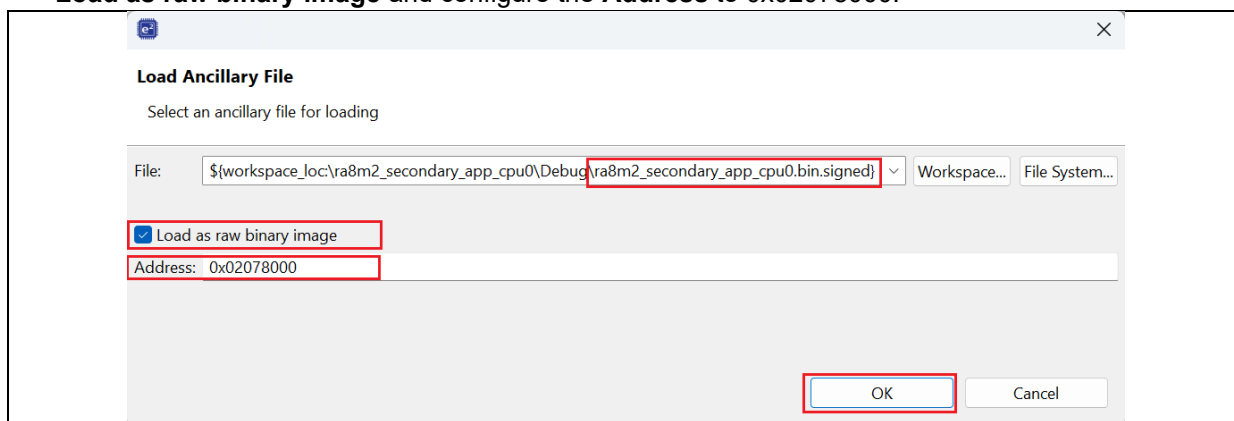


Figure 26. Load the Secondary Application Image CPU0

- Click Resume all . The swap occurs and the new images are executed. Now the LED3 should be blinking and the LED1 should be blinking with slower frequency.

3.4 Running the EK-RA8M1 Swap Update Mode Example

Follow the steps below to run the example projects for EK-RA8M1 using the MCUboot Swap Update mode.

3.4.1 Import the Projects

Import the project ra8m1_mcuboot_PQC.zip to a Workspace.

The following example projects are included in this folder and the memory Flash map overview used in the project:

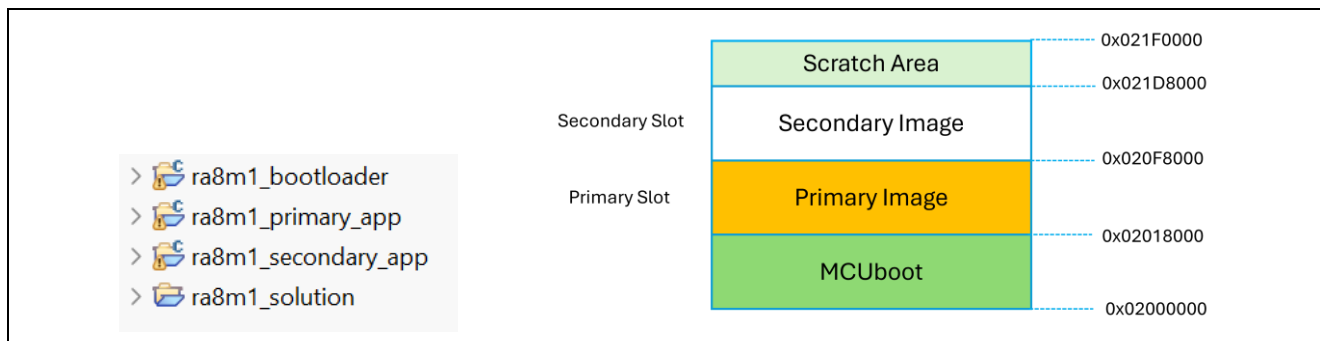


Figure 27. Example Projects for RA8M1 Swap Update Mode

- Project ra8m1_bootloader is the bootloader project.
- Project ra8m1_primary_app is the initial Primary application project. This project blinks the three LEDs on the EK-RA8M1 kit.
- Project ra8m1_secondary_app is the Secondary application project. This project blinks the LED1 on the EK-RA8M1 kit.
- Project ra8m1_solution is the solution project. This project configures memory areas.

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

3.4.2 Compile the Projects

The solution project includes the bootloader and primary application projects. Right click on solution project and select **Build Project (Ctrl+B)** to build the bootloader and primary application projects. The bootloader image is located under the \Debug folder and is named <project_name>.elf. For the application project, 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, /ra8m1_primary_app/Debug/ra8m1_primary_app.bin.signed)

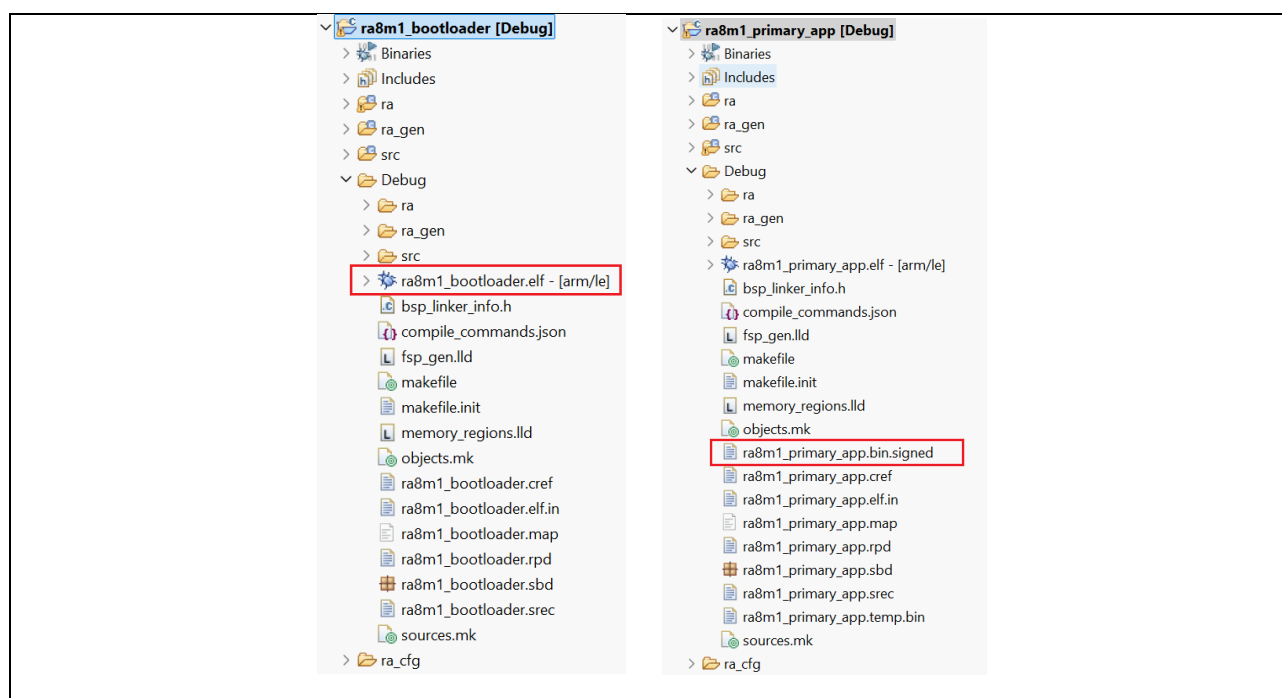


Figure 28. Successfully built images on RA8M1

3.4.3 Debug the Applications and Boot the Primary Application

Right-click on project **ra8m1_primary_app** and select **Debug As > Debug Configurations**.

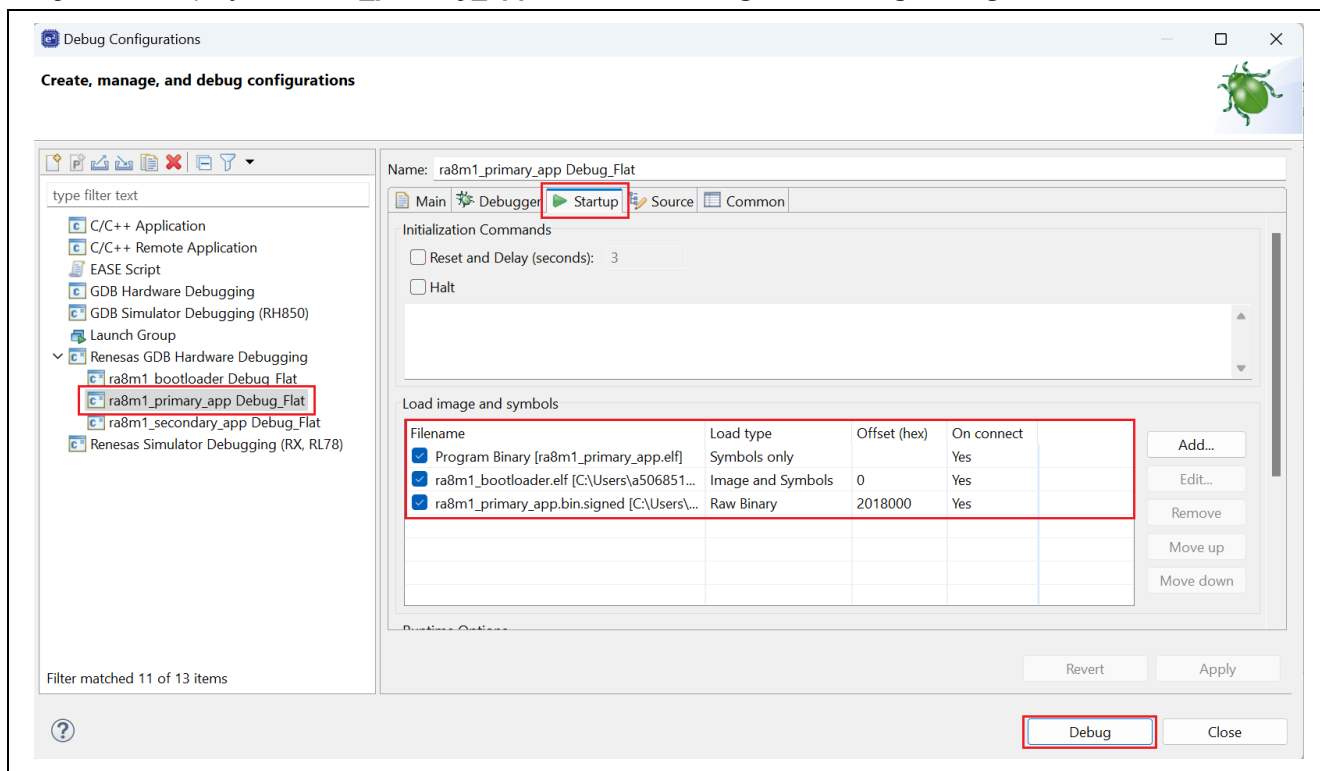


Figure 29. Debug Configuration RA8M1 Overwrite Update

Select **ra8m1_primary_app Debug_Flat** then click **Debug**.

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

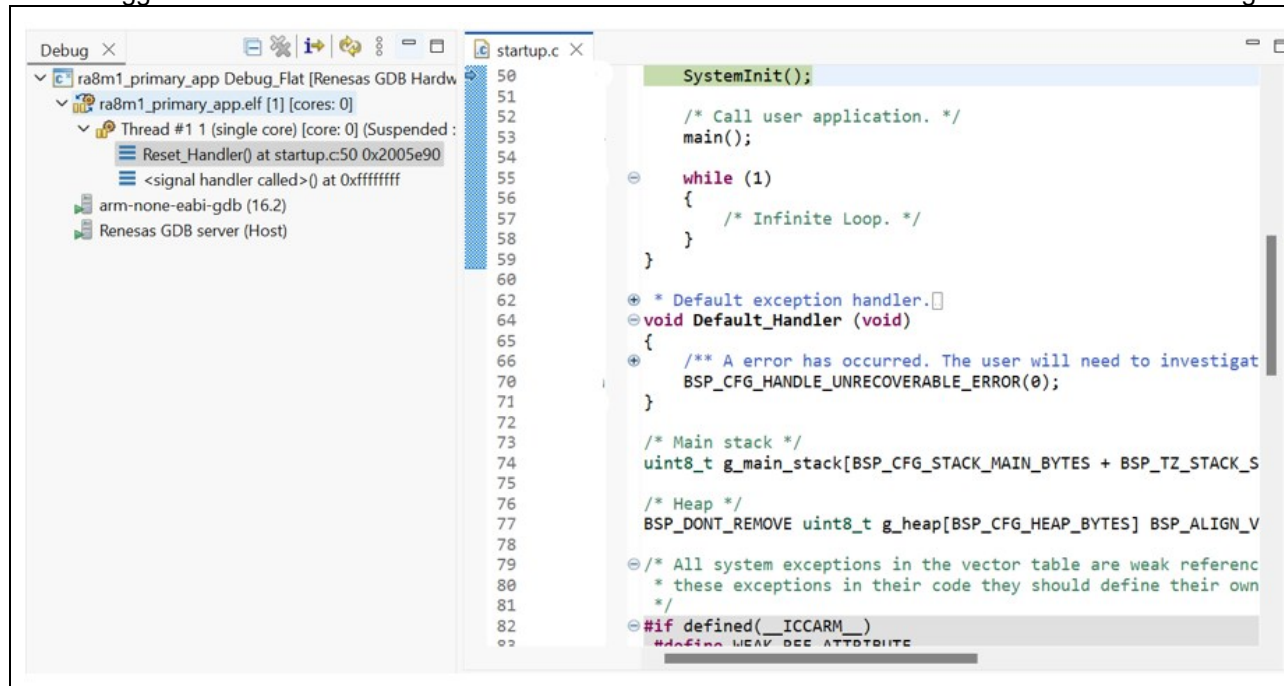


Figure 30. Start the RA8M1 Application Execution

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

3.4.4 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.

Note:

Ensure compiling ra8m1_secondary_app before starting. The signed image is located under the \Debug folder and is named <project_name>.bin.signed (for example, /ra8m1_secondary_app /Debug/ra8m1_secondary_app.bin.signed).

1. Press  to pause the program.
2. Load the new application images to the Secondary slot region using the Ancillary loading capability  from the top of the e² studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x020F8000.

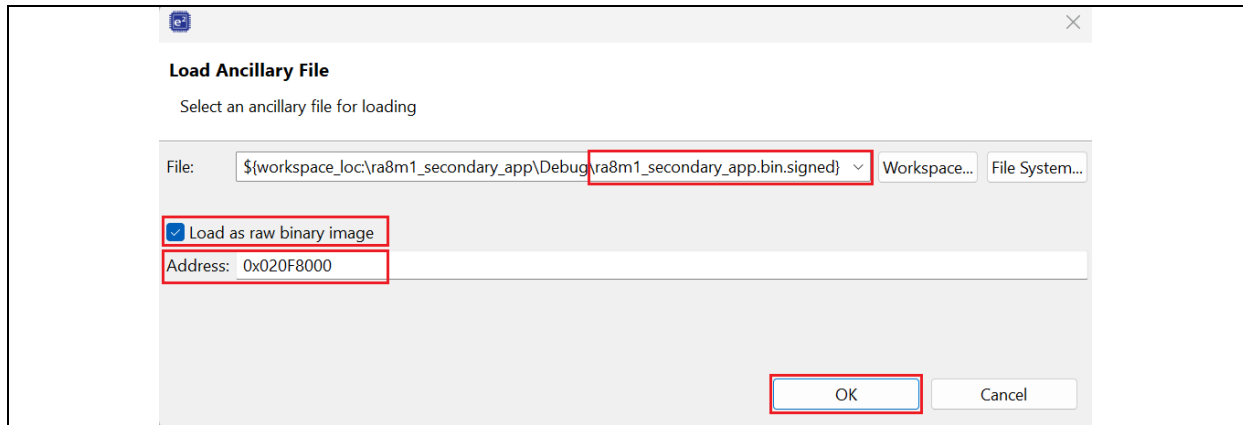


Figure 31. Load the Secondary Application Image on RA8M1

3. Click Resume . A swap occurs and the new image is executed. Now only the LED1 should be blinking.

3.5 Running the EK-RA6M5 Swap Update Mode Example

Follow the steps below to run the example projects for EK-RA6M5 using the MCUboot Swap Update mode.

3.5.1 Import the Projects

Import the project ra6m5_mcuboot_PQC.zip to a Workspace.

The following example projects are included in this folder and the memory Flash map overview used in the project:

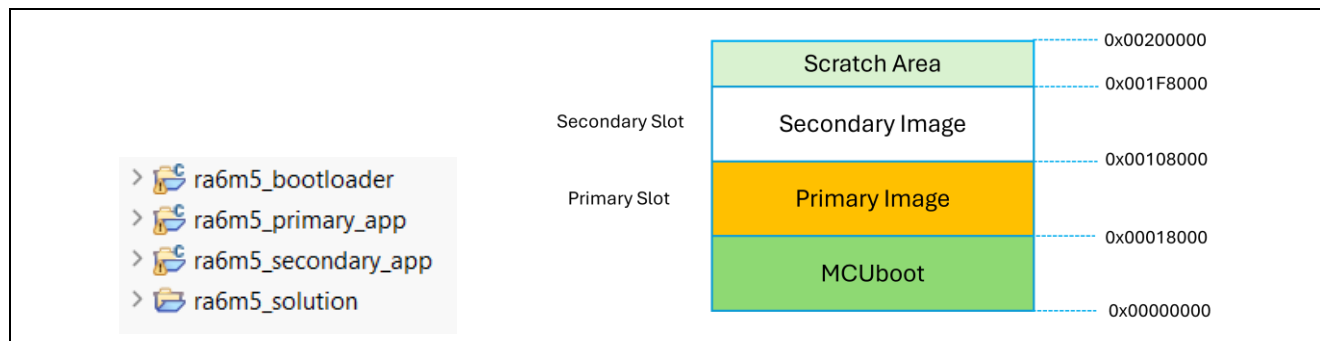


Figure 32. Example Projects for RA6M5 Swap Update Mode

- Project ra6m5_bootloader is the bootloader project.
- Project ra6m5_primary_app is the initial Primary application project. This project blinks the three LEDs on the EK-RA6M5 kit.
- Project ra6m5_secondary_app is the Secondary application project. This project blinks the LED1 on the EK-RA6M5 kit.
- Project ra6m5_solution is the solution project. This project configures memory areas.

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

3.5.2 Compile the Projects

The solution project includes the bootloader and primary application projects. Right click on solution project and select **Build Project (Ctrl+B)** to build the bootloader and primary application projects. The bootloader image is located under the \Debug folder and is named <project_name>.elf. For the application project, 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, /ra6m5_primary_app/Debug/ra6m5_primary_app.bin.signed)

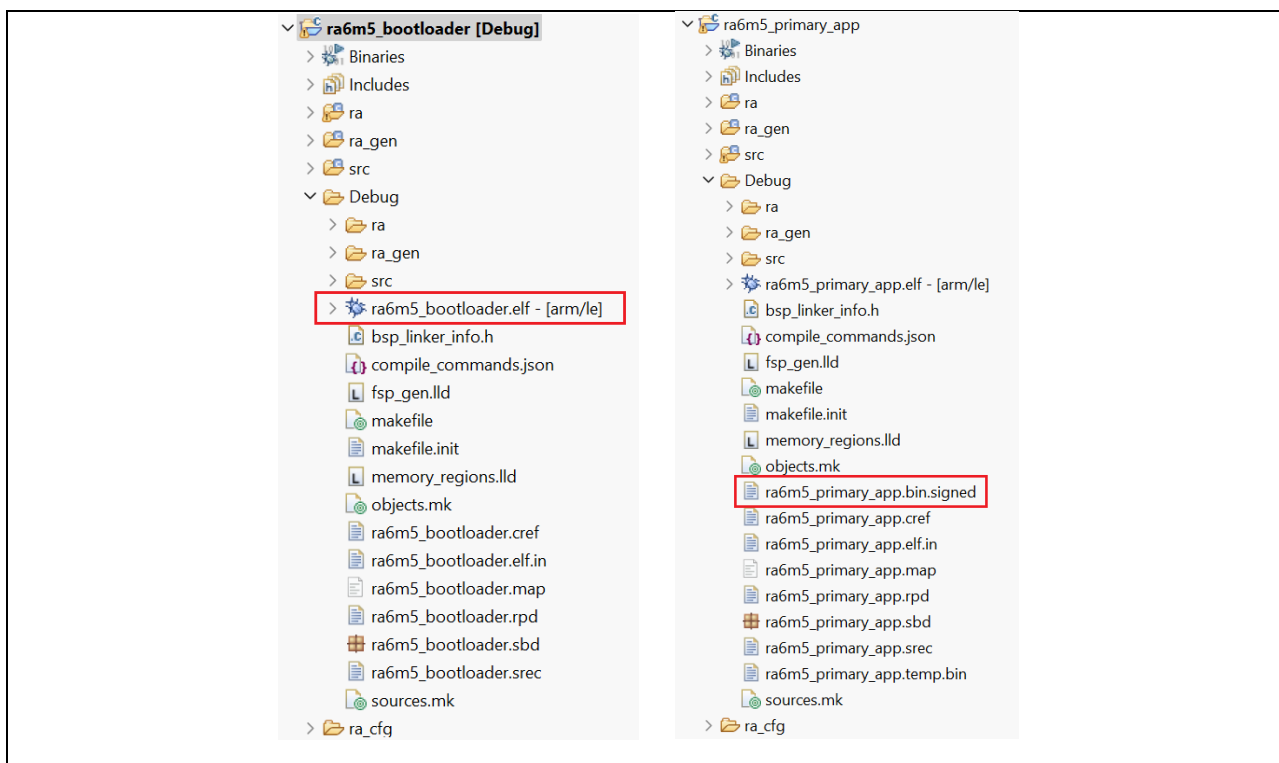


Figure 33. Successfully built images on RA6M5

3.5.3 Debug the Applications and Boot the Primary Application

Right-click on project **ra6m5_primary_app** and select **Debug As > Debug Configurations**.

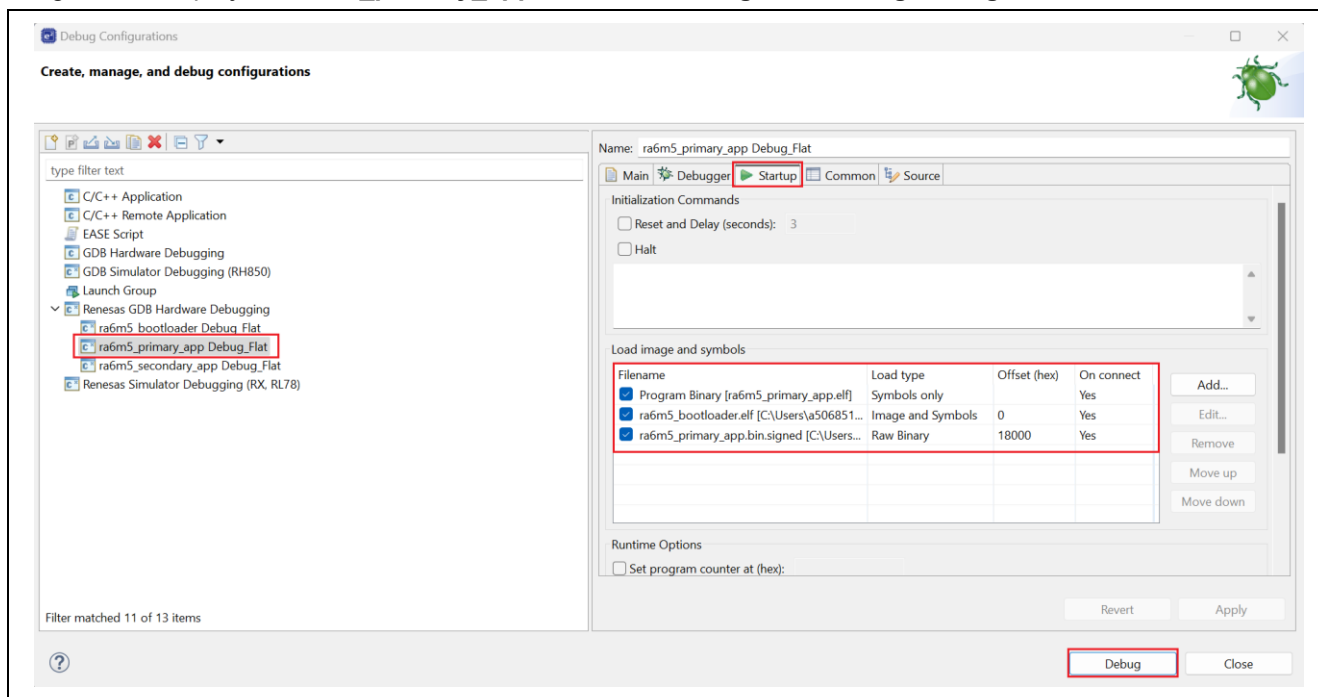


Figure 34. Debug Configuration RA6M5 Swap Update

Select **ra6m5_primary_app Debug_Flat** then click **Debug**.

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

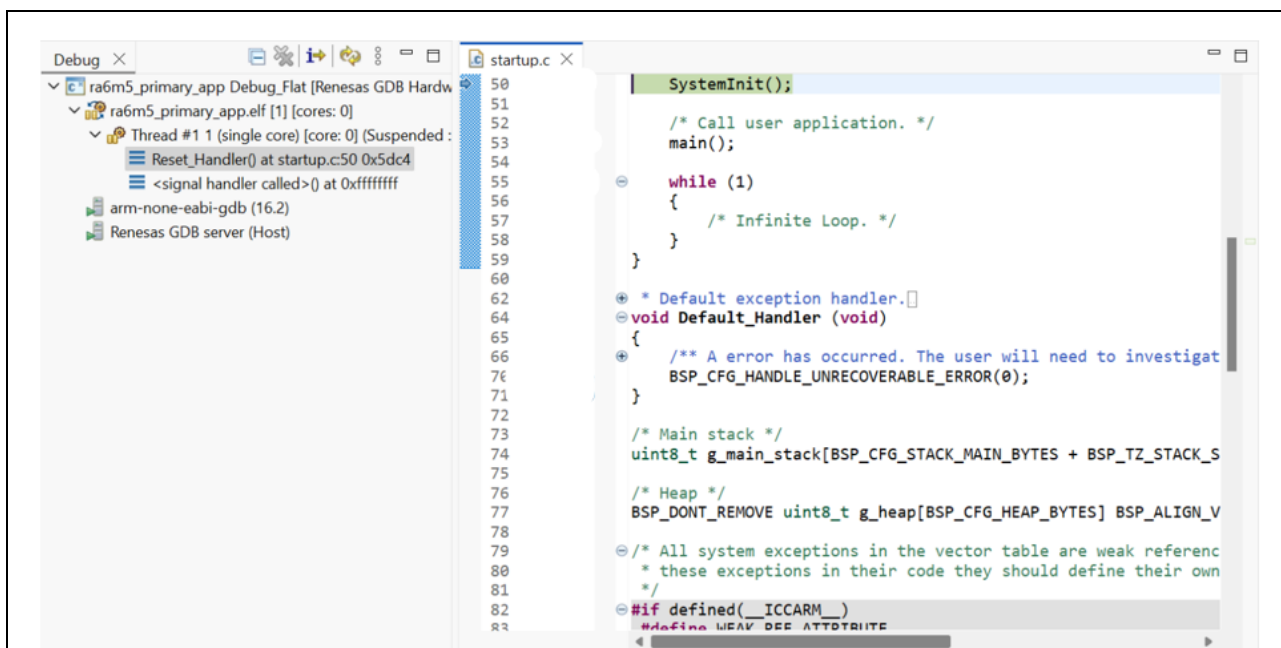


Figure 35. Start the RA6M5 Application Execution

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

3.5.4 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.

Note:

Ensure compiling ra6m5_secondary_app before starting. The signed image is located under the \Debug folder and is named <project_name>.bin.signed (for example, /ra6m5_secondary_app /Debug/ra6m5_secondary_app.bin.signed).

1. Press  to pause the program.
2. Load the new application images to the Secondary slot region using the Ancillary loading capability  from the top of the e2 studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x00108000.

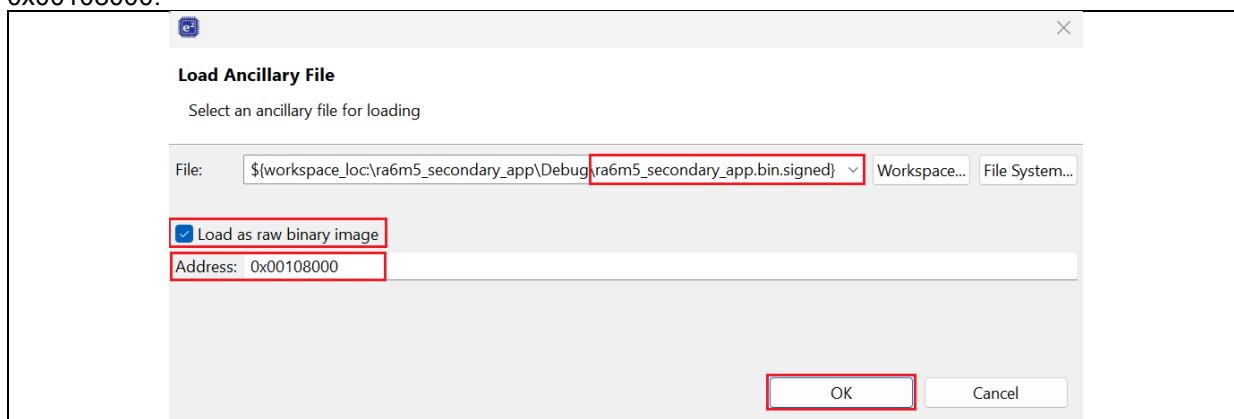


Figure 36. Load the Secondary Application Image on RA6M5

3. Click Resume . A swap occurs and the new image is executed. Now only the LED1 should be blinking.

3.6 Running the EK-RA6M3 Swap Update Mode Example

Follow the steps below to run the example projects for EK-RA6M3 using the MCUboot Swap Update mode.

3.6.1 Import the Projects

Import the project ra6m3_mcuboot_PQC.zip to a Workspace.

The following example projects are included in this folder and the memory Flash map overview used in the project:

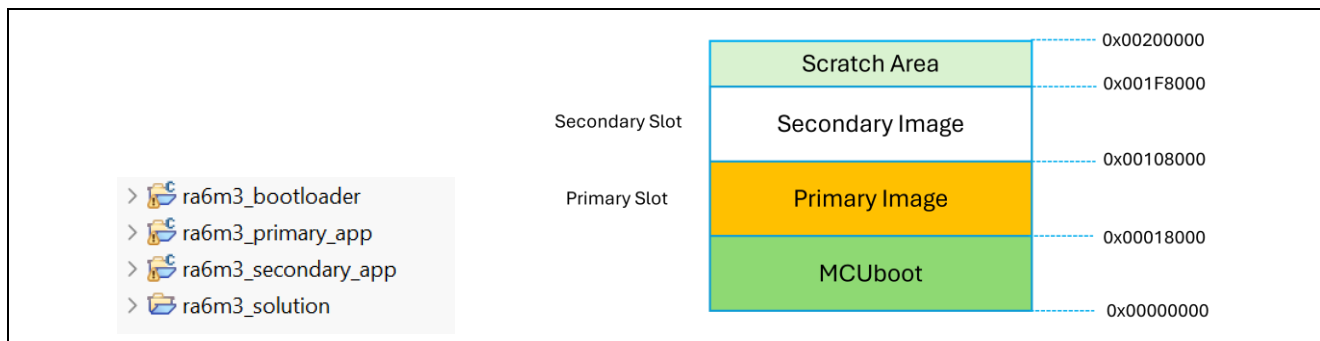


Figure 37. Example Projects for RA6M3 Swap Update Mode

- Project ra6m3_bootloader is the bootloader project.
- Project ra6m3_primary_app is the initial Primary application project. This project blinks the three LEDs on the EK-RA6M3 kit.
- Project ra6m3_secondary_app is the Secondary application project. This project blinks the LED1 on the EK-RA6M3 kit.
- Project ra6m3_solution is the solution project. This project configures memory areas.

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 solution project includes the bootloader and primary application projects. Right click on solution project and select **Build Project (Ctrl+B)** to build the bootloader and primary application projects. The bootloader image is located under the \Debug folder and is named <project_name>.elf. For the application project, 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, /ra6m3_primary_app/Debug/ra6m3_primary_app.bin.signed)

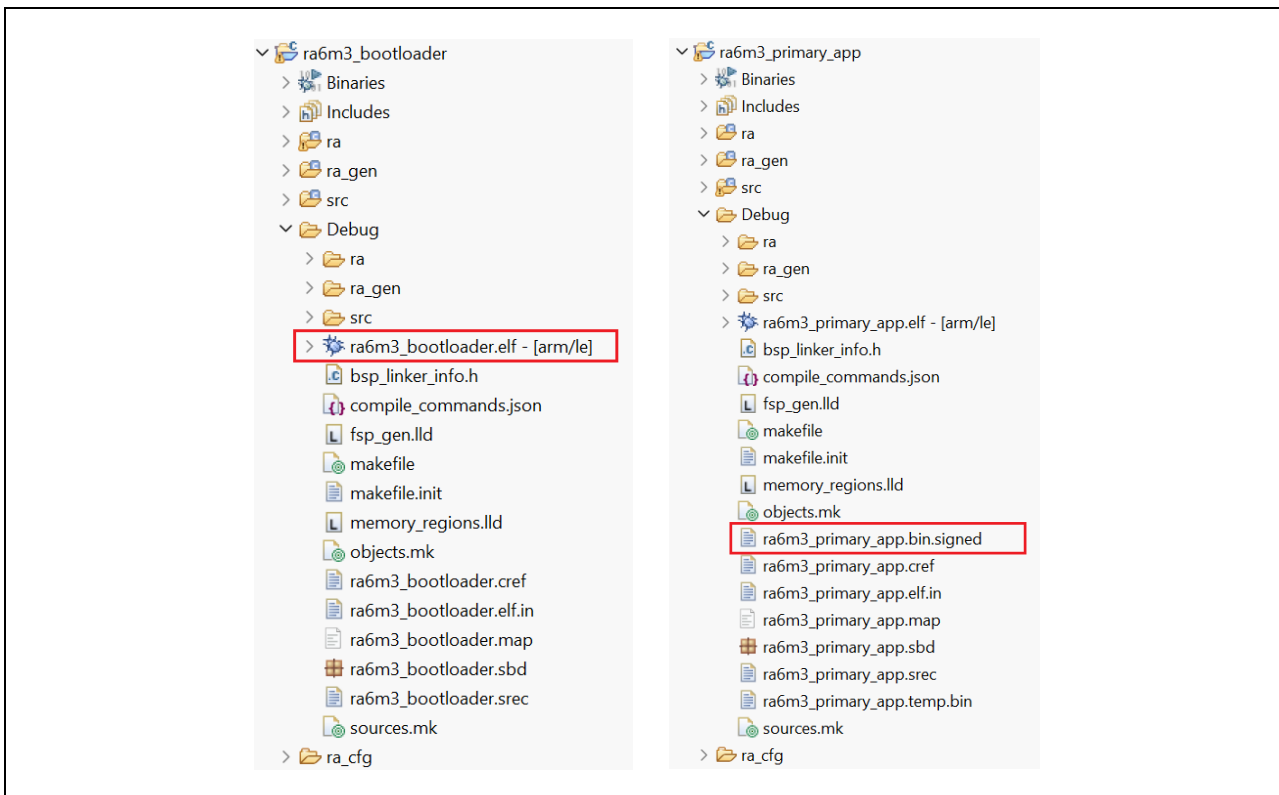


Figure 38. Successfully built images on RA6M3

3.6.3 Debug the Applications and Boot the Primary Application

Right-click on project **ra6m3_primary_app** and select **Debug As > Debug Configurations**.

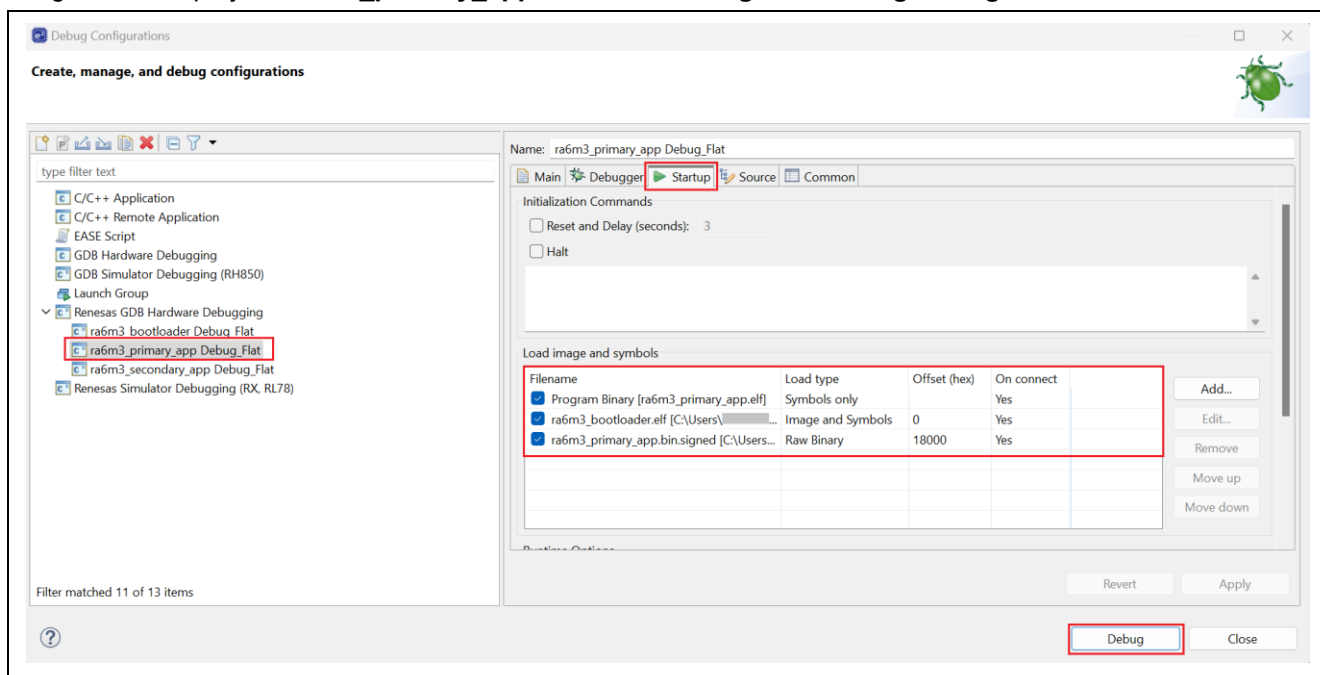


Figure 39. Debug Configuration RA6M3 Swap Update

Select **ra6m3_primary_app Debug_Flat** then click **Debug**.

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

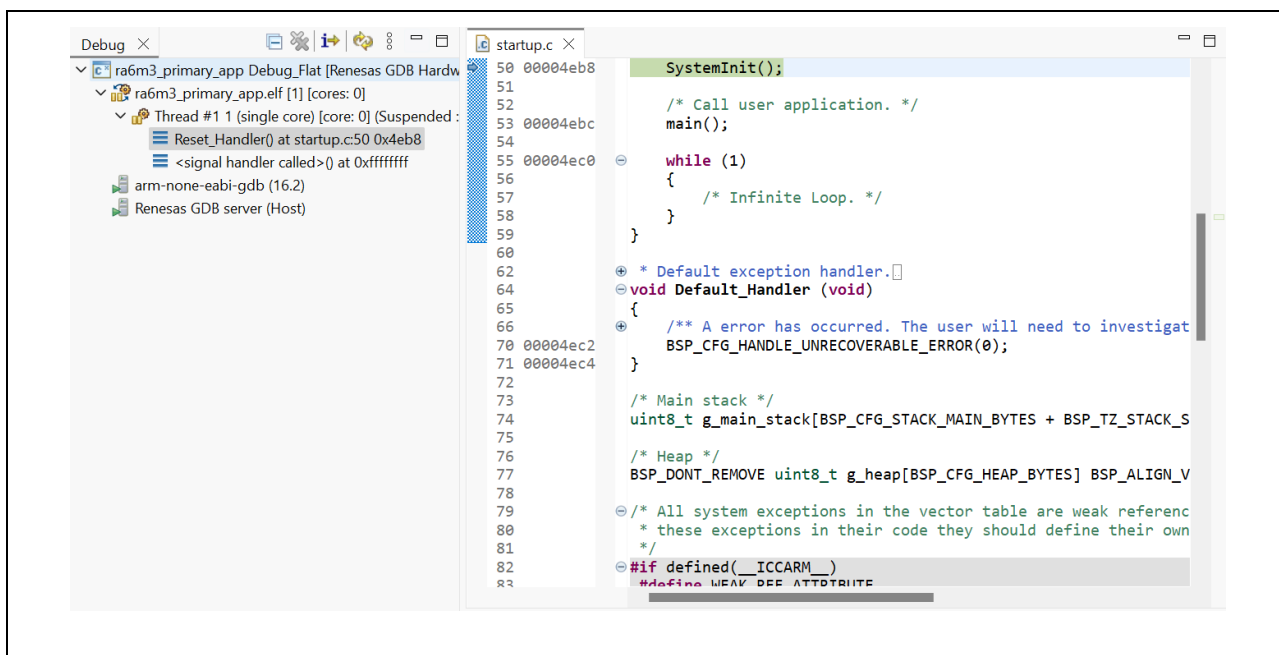


Figure 40. Start the RA6M3 Application Execution

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

3.6.4 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.

Note:

Ensure compiling ra6m3_secondary_app before starting. The signed image is located under the \Debug folder and is named <project_name>.bin.signed (for example, /ra6m3_secondary_app /Debug/ra6m3_secondary_app.bin.signed).

1. Press  to pause the program.
2. Load the new application images to the Secondary slot region using the Ancillary loading capability  from the top of the e2 studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x00108000.

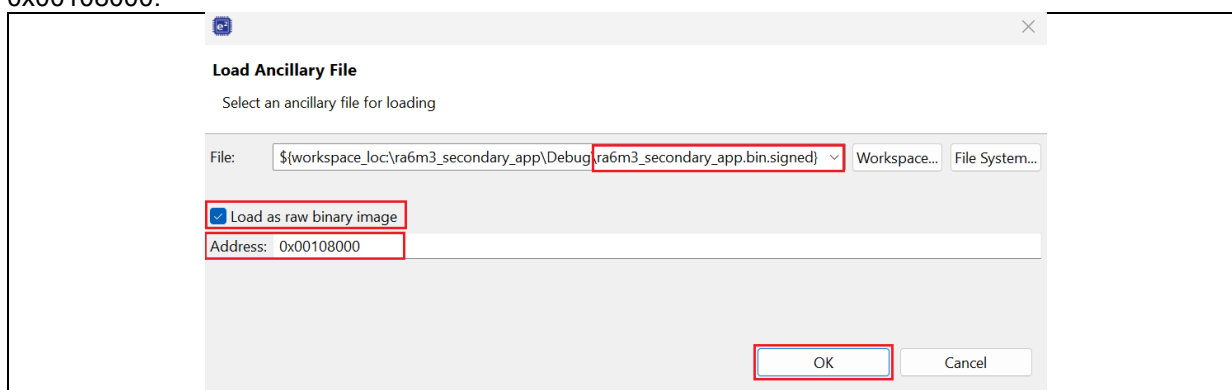


Figure 41. Load the Secondary Application Image on RA6M3

3. Click Resume . A swap occurs and the new image is executed. Now only the LED1 should be blinking.

3.7 Running the EK-RA4C1 Swap Update Mode Example

Follow the steps below to run the example projects for EK-RA4C1 using the MCUboot Swap Update mode.

3.7.1 Import the Projects

Import the project ra4c1_mcuboot_PQC.zip to a Workspace.

The following example projects are included in this folder and the memory Flash map overview used in the project:

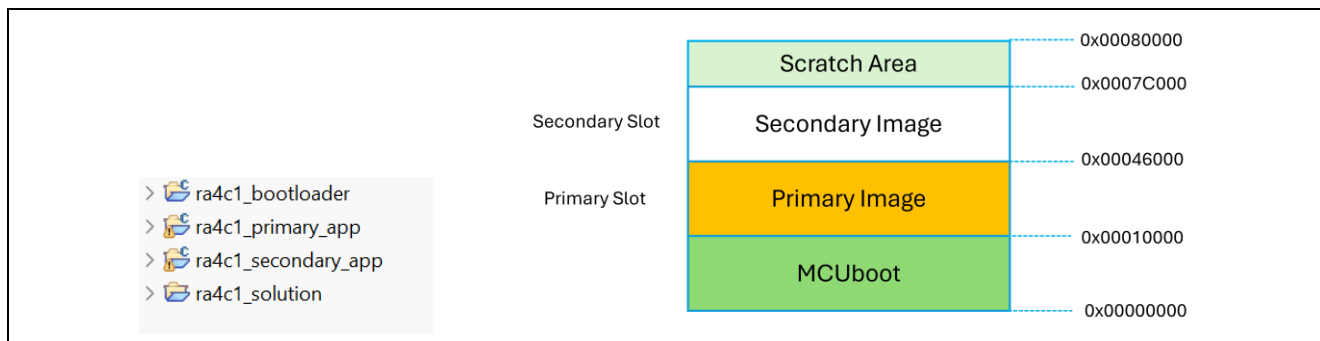


Figure 42. Example Projects for RA4C1 Swap Update Mode

- Project ra4c1_bootloader is the bootloader project.
- Project ra4c1_primary_app is the initial Primary application project. This project blinks the three LEDs on the EK-RA4C1 kit.
- Project ra4c1_secondary_app is the Secondary application project. This project blinks the LED1 on the EK-RA4C1 kit.
- Project ra4c1_solution is the solution project. This project configures memory areas.

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 solution project includes the bootloader and primary application projects. Right click on solution project and select **Build Project (Ctrl+B)** to build the bootloader and primary application projects. The bootloader image is located under the \Debug folder and is named <project_name>.elf. For the application project, 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, /ra4c1_primary_app/Debug/ra4c1_primary_app.bin.signed)

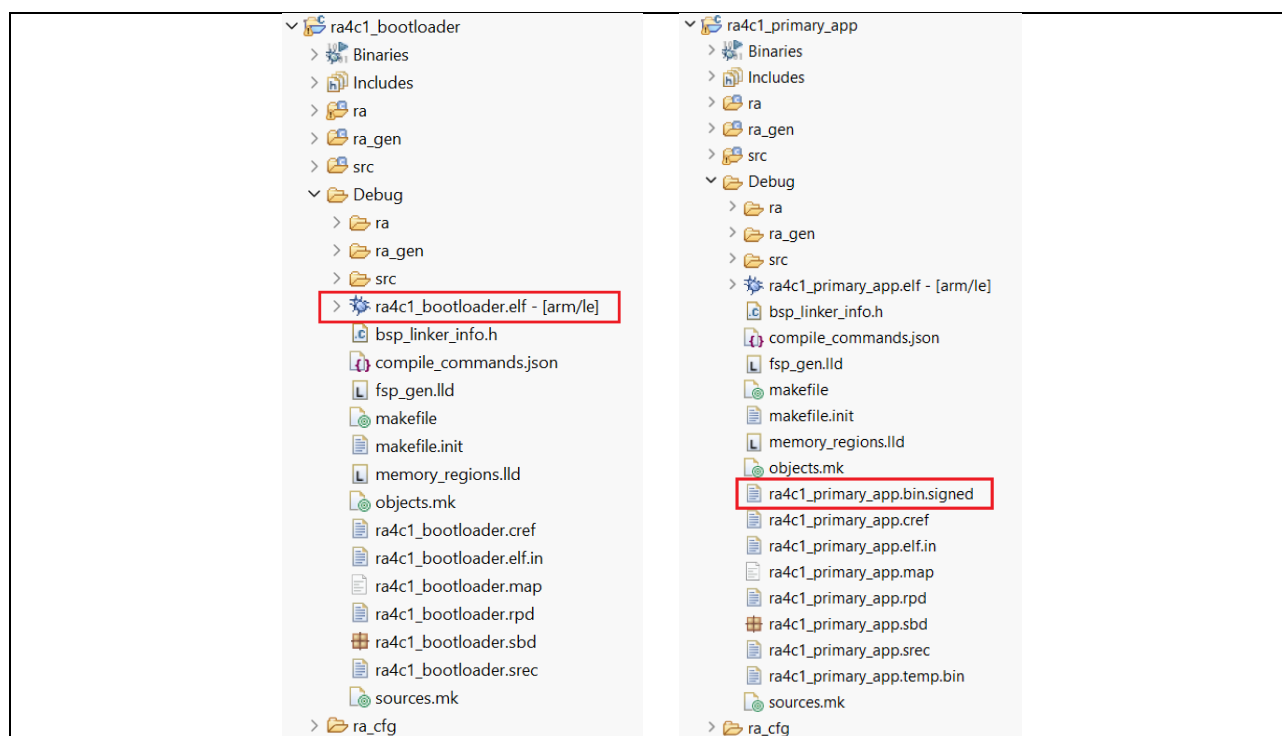


Figure 43. Successfully built images on RA4C1

3.7.3 Debug the Applications and Boot the Primary Application

Right-click on project **ra4c1_primary_app** and select **Debug As > Debug Configurations**.

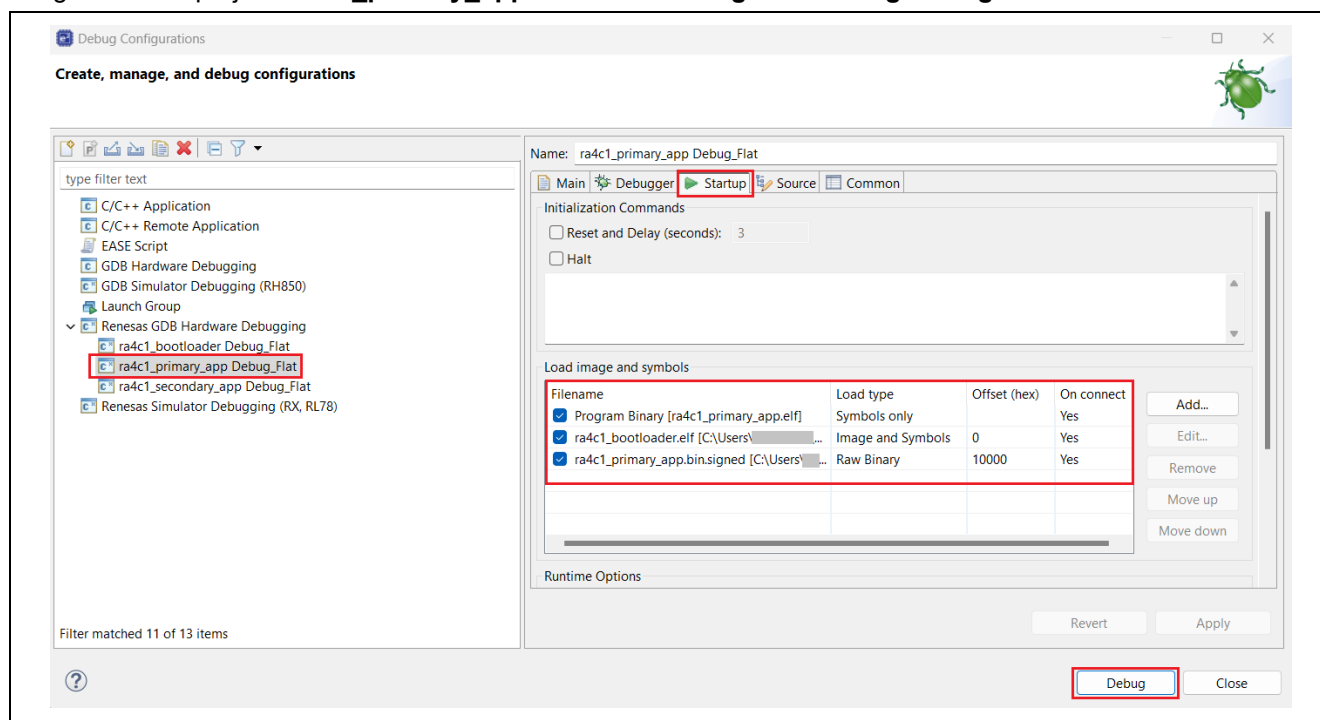


Figure 44. Debug Configuration RA4C1 Swap Update

Select **ra4c1_primary_app Debug_Flat** then click **Debug**.

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

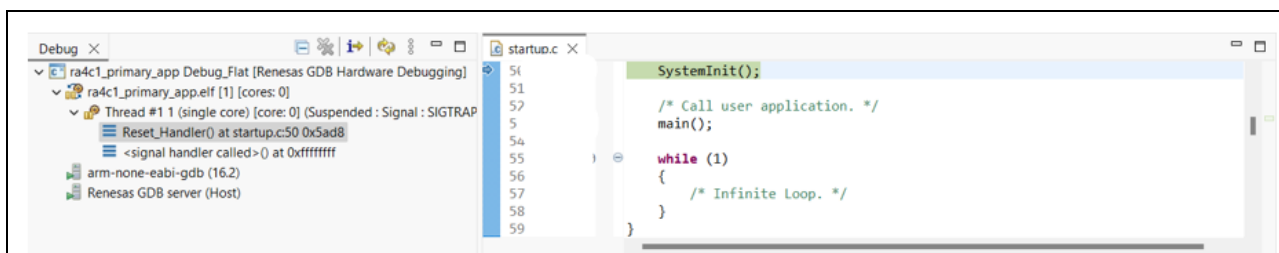


Figure 45. Start the RA4C1 Application Execution

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

3.7.4 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.

Note:

Ensure compiling ra4c1_secondary_app before starting. The signed image is located under the \Debug folder and is named <project_name>.bin.signed (for example, /ra4c1_secondary_app /Debug/ra4c1_secondary_app.bin.signed).

1. Press  to pause the program.
2. Load the new application images to the Secondary slot region using the Ancillary loading capability  from the top of the e² studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x00046000.

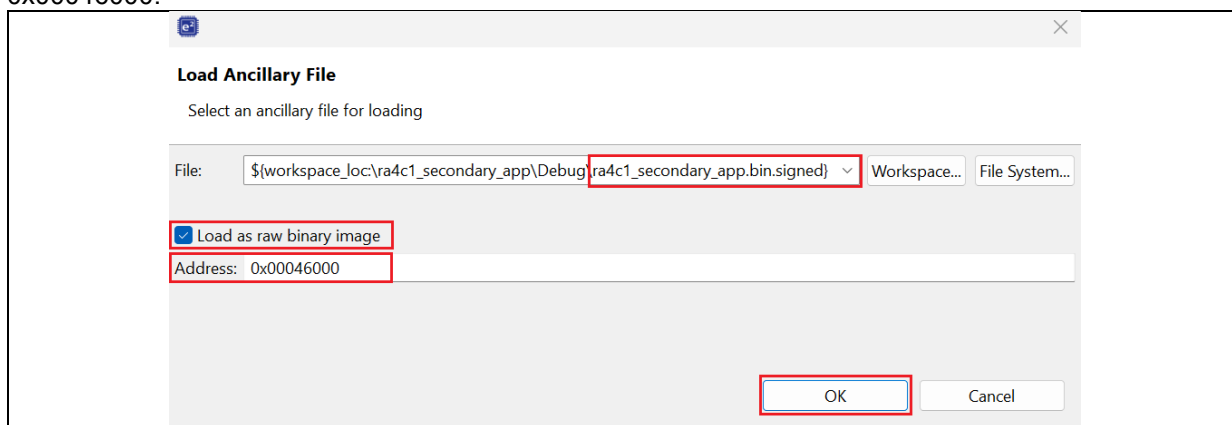


Figure 46. Load the Secondary Application Image on RA6M3

3. Click Resume . A swap occurs and the new image is executed. Now only the LED1 should be blinking.

3.8 Running the EK-RA2A2 Direct XIP Update Mode and Dual Bank Mode Example

Follow the steps below to run the example projects for EK-RA2A2 using the MCUboot Direct XIP Update mode and Dual Bank mode.

3.8.1 Import the Projects

Import the project ra2a2_mcuboot_PQC.zip to a Workspace.

The following example projects are included in this folder and the memory Flash map overview used in the project:

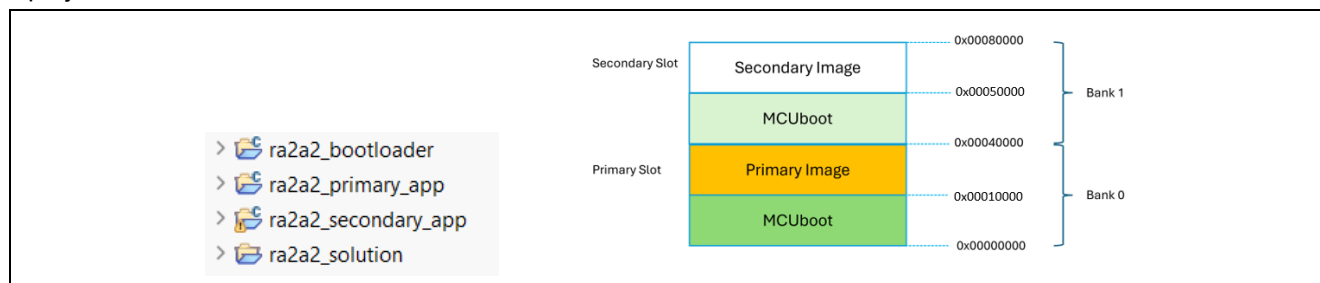


Figure 47. Example Projects for RA2A2 Direct XIP Update Mode

- Project ra2a2_bootloader is the bootloader project.
- Project ra2a2_primary_app is the initial Primary application project. This project blinks the three LEDs on the EK-RA2A2 kit.
- Project ra2a2_secondary_app is the Secondary application project. This project blinks the LED1 on the EK-RA2A2 kit.
- Project ra2a2_solution is the solution project. This project configures memory areas.

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

3.8.2 Compile the Projects

The solution project includes the bootloader and primary application projects. Right click on solution project and select **Build Project (Ctrl+B)** to build the bootloader and primary application projects. The bootloader image is located under the \Debug folder and is named <project_name>.elf. For the application project, 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, /ra2a2_primary_app/Debug/ra2a2_primary_app.bin.signed)

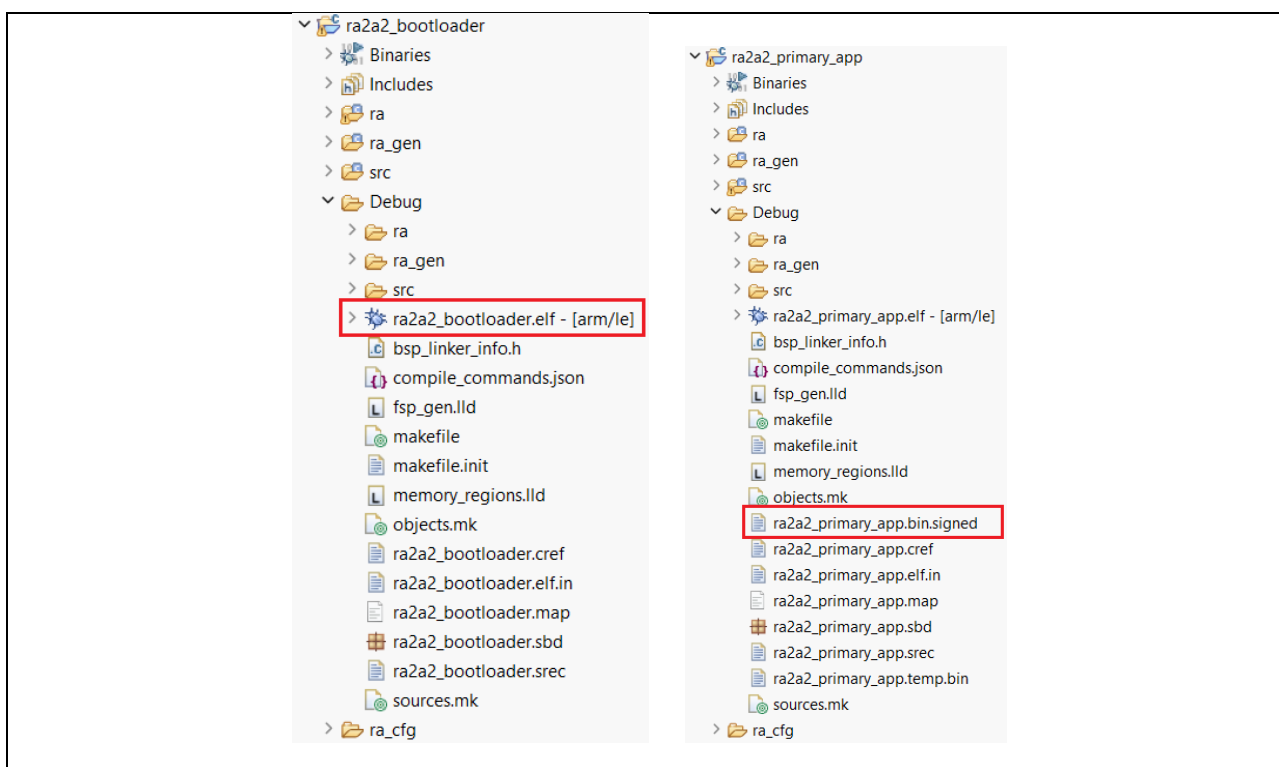
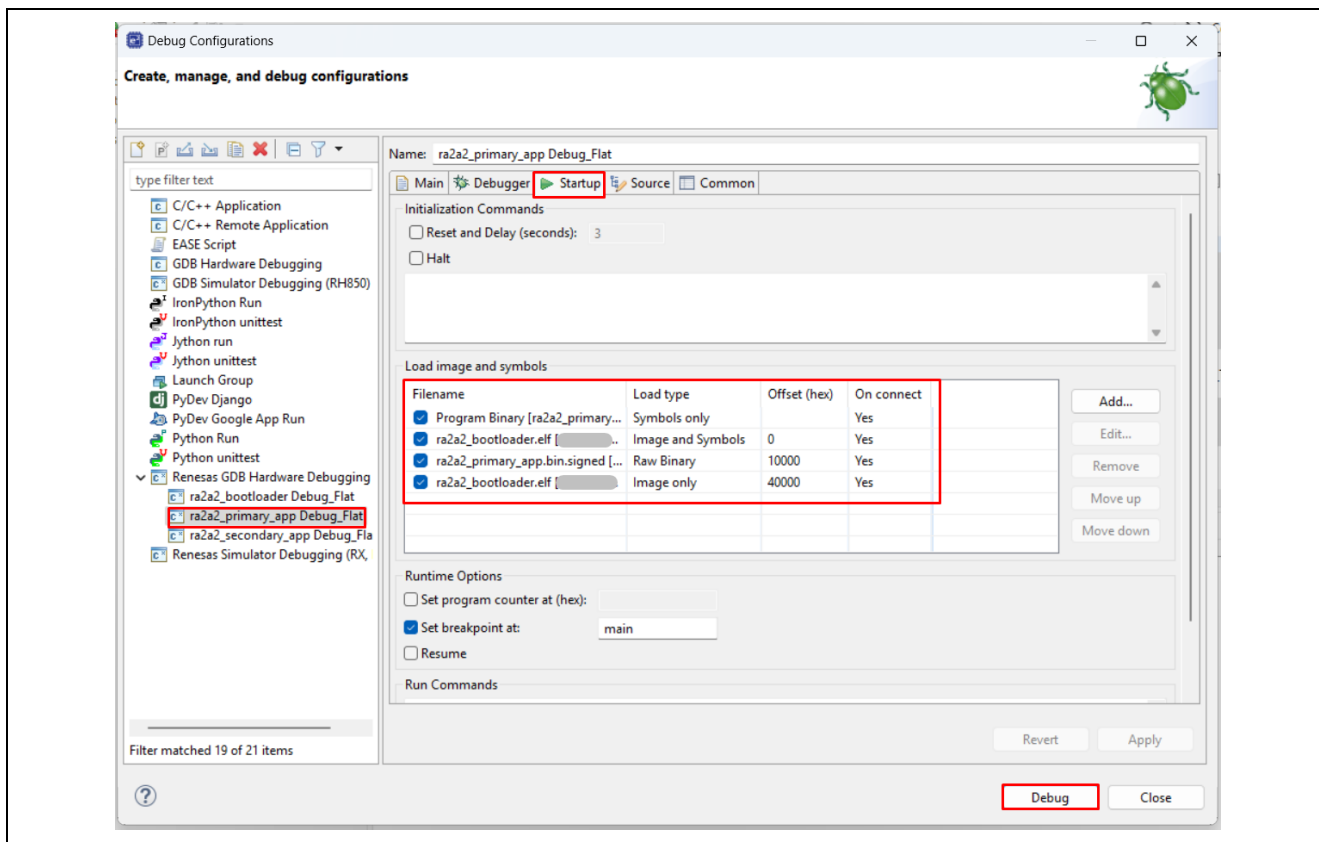


Figure 48. Successfully built images on RA2A2

3.8.3 Debug the Applications and Boot the Primary Application

Right-click on project **ra2a2_primary_app** and select **Debug As > Debug Configurations**.



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

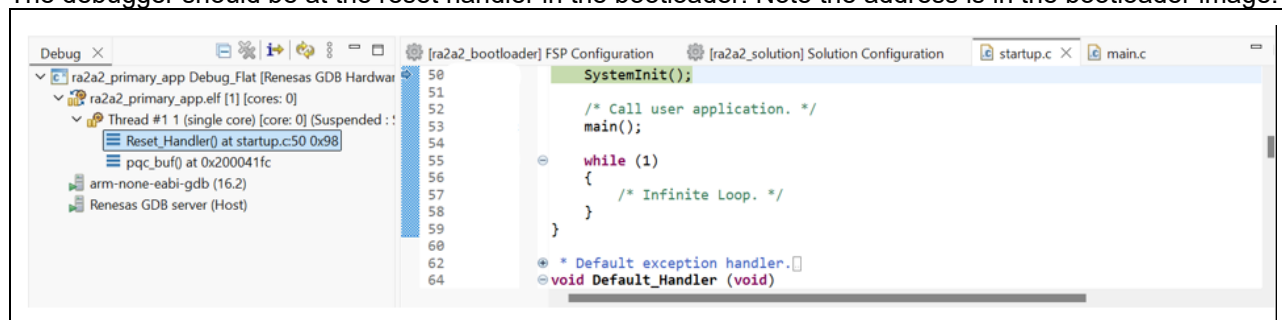


Figure 50. Start the RA2A2 Application Execution

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

3.8.4 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.

Note:

Ensure compiling ra2a2_secondary_app before starting. The signed image is located under the \Debug folder and is named <project_name>.bin.signed (for example, /ra2a2_secondary_app /Debug/ra2a2_secondary_app.bin.signed).

1. Press  to pause the program.
2. Load the new application images to the Secondary slot region using the Ancillary loading capability  from the top of the e² studio toolbar. Select **Load as raw binary image** and configure the **Address** to 0x00050000.

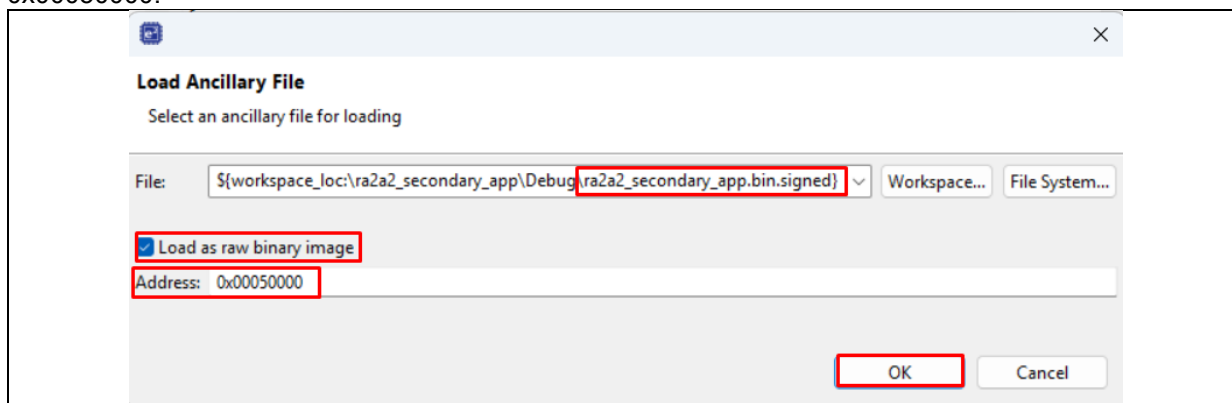


Figure 51. Load the Secondary Application Image on RA2A2

3. Click Resume . The bank swap occurs and the highest version slot is selected. The new image is executed. Now only the LED1 should be blinking.

4. Creating the Bootloader Project

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.

4.1 Start Bootloader Project Creation with e2 studio

Follow the steps below to create the initial bootloader project. The procedure is the same for the RA8, RA6, RA4, and RA2 families when using **Swap mode** and **Direct XIP mode** as the upgrade mode. Note the following family-specific settings in this application note:

- **RA8 (RA8M2, RA8M1) and RA6 (RA6M5, RA6M3):** Set Signature Type to ML-DSA 87, upgrade method: Swap mode.
- **RA4 (RA4C1):** Set Signature Type to ML-DSA 44, upgrade method: Swap mode
- **RA2 (RA2A2):** Set Signature Type to ML-DSa 44, upgrade method: Direct XIP and enable Dual Bank mode

1. Launch e2 studio and start to create a new C/C++ Project.

Click **File > New > Renesas C/C++ Project > Renesas RA**.

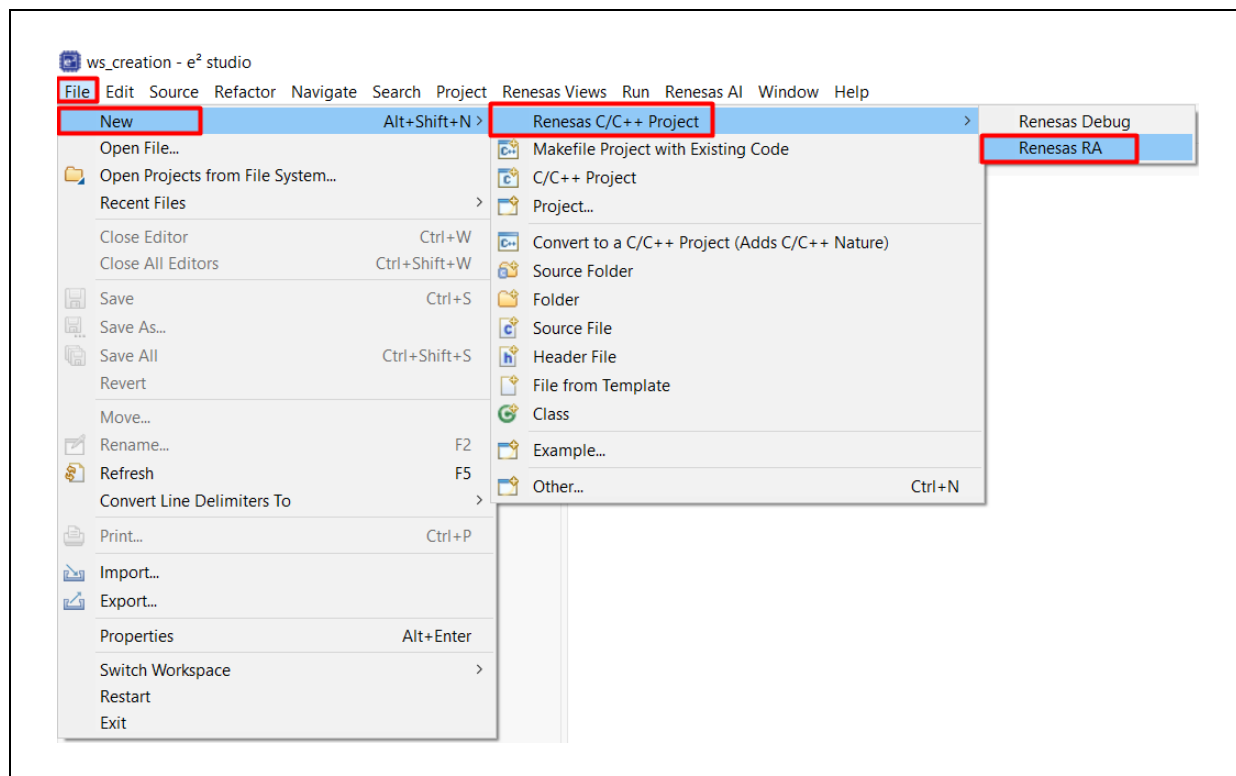


Figure 52. Start a New Project

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

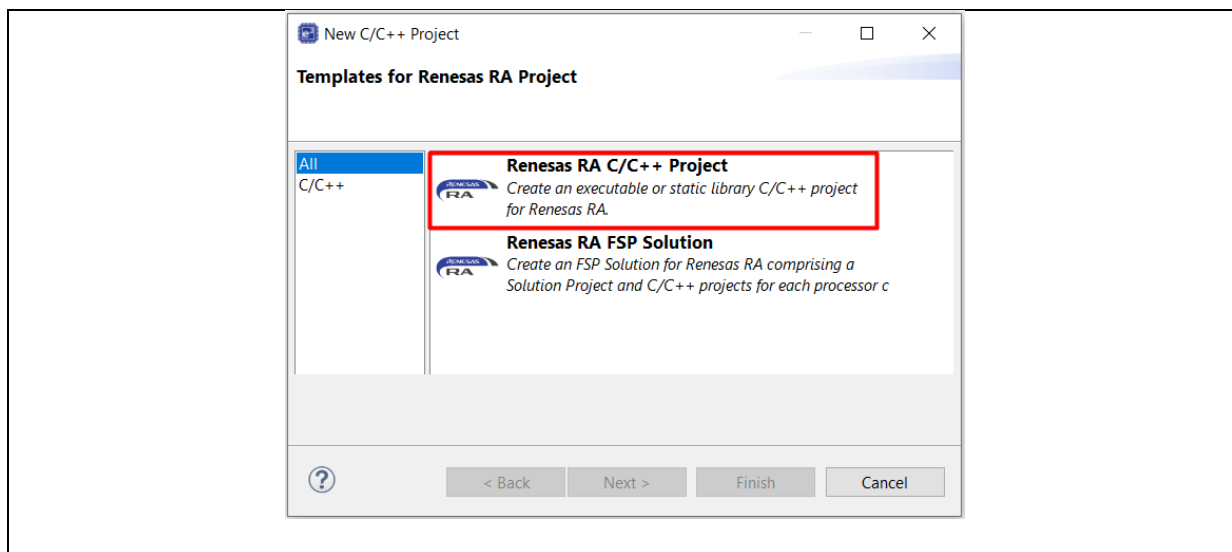


Figure 53. Choose Renesas RA C/C++ Project

3. Provide the **Project name** and click **Next**.

Project name based on MCU series and the names will persist in the instructions used in this application note.

MCU Series	Bootloader Project name
RA8M2	ra8m2_bootloader
RA8M1	ra8m1_bootloader
RA6M5	ra6m5_bootloader
RA6M3	ra6m3_bootloader
RA4C1	ra4c1_bootloader
RA2A2	ra2a2_bootloader

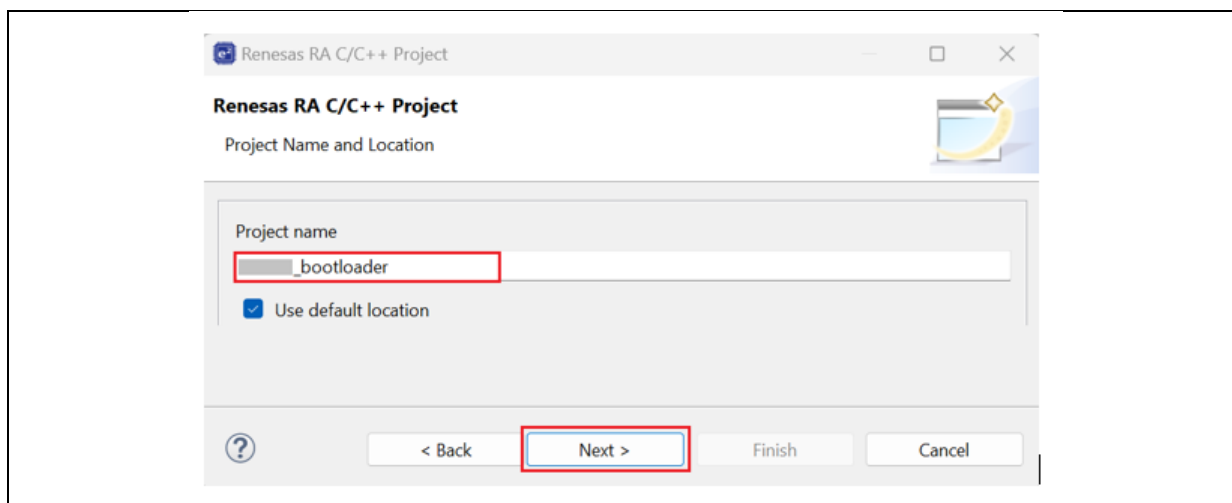


Figure 54. Name the Bootloader Project

If a different name for the bootloader is chosen, then adapt the corresponding instructions in this application note accordingly.

4. Select **FSP Version**, **Board** and **Toolchains LLVM**. Click **Next**.

MCU Series	RA8M2	RA8M1	RA6M5	RA6M3	RA4C1	RA2A2
FSP Version:	6.6.0					
Board:	EK-RA8M2	EK-RA8M1	EK-RA6M5	EK-RA6M3	EK-RA4C1	EK-RA2A2
Device:	R7KA8M2JFLCAC	R7FA8M1AHECBD	R7FA6M5BH3CFC	R7FA6M3AH3CFC	R7FA4C1BD3CFP	R7FA2A2AD3CFP
Core:	CPU0	CM85	CM33	CM4	CM33	CM23
Toolchains	LLVM Embedded Toolchain for Arm					

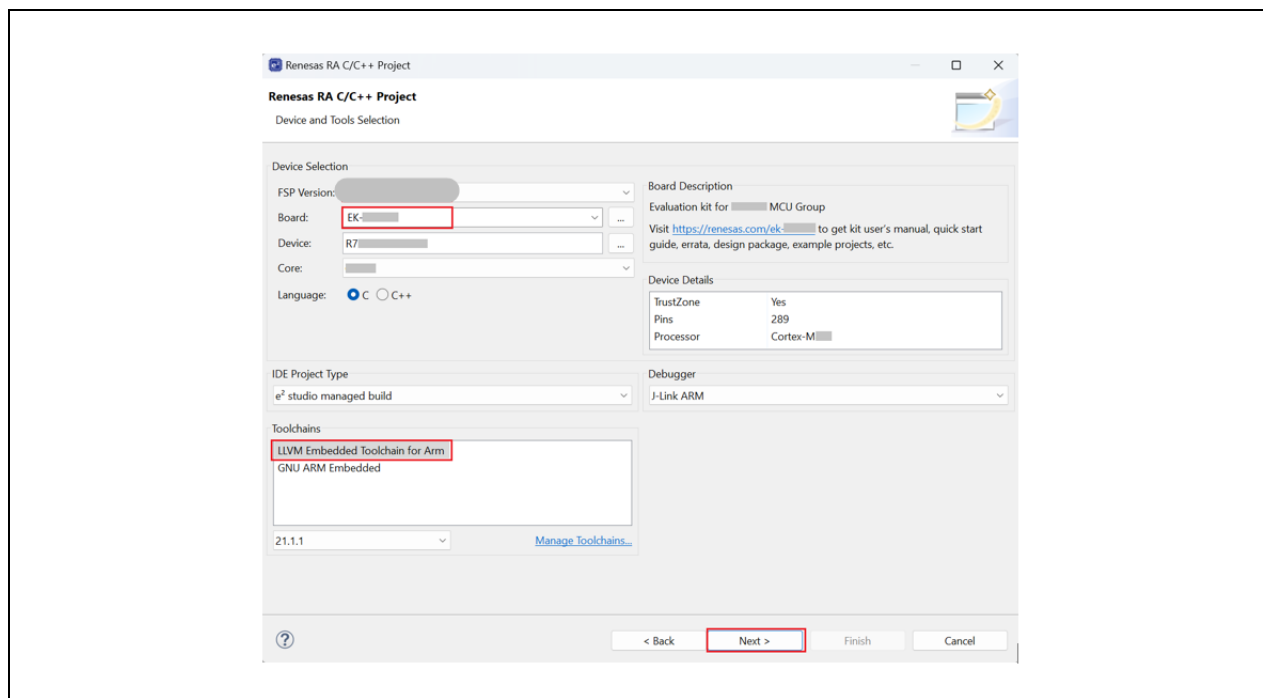


Figure 55. Select the FSP, Board & Toolchain

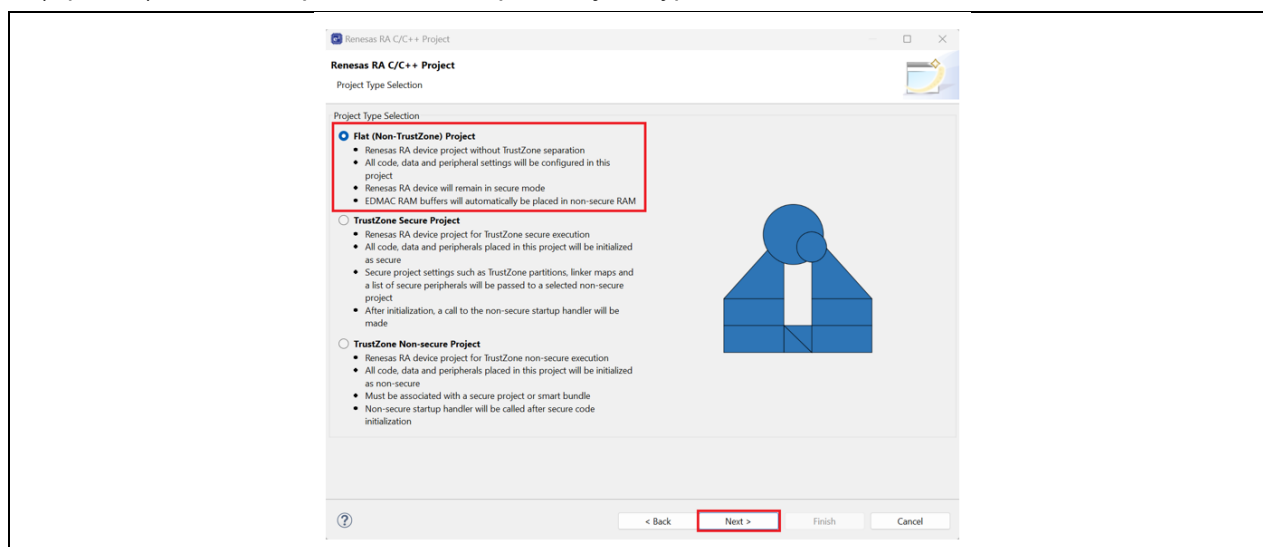
5. (Optional) Select **Flat (Non-TrustZone)** for Project Type Selection. Click **Next**.

Figure 56. Select Project Type (available only for devices that support TrustZone)

6. Select **None** for Preceding Project or Smart Bundle Selection screen. Click **Next**.

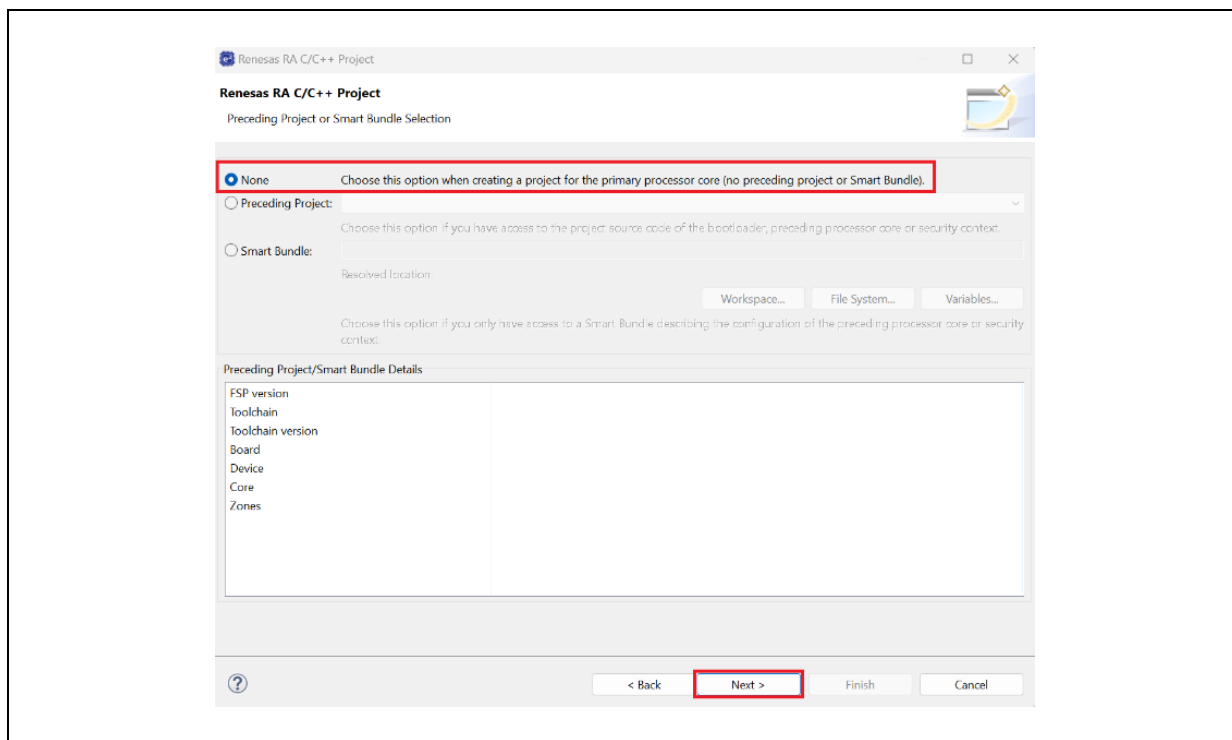


Figure 57. Select the None option for Preceding Project or Smart Bundle Selection screen

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

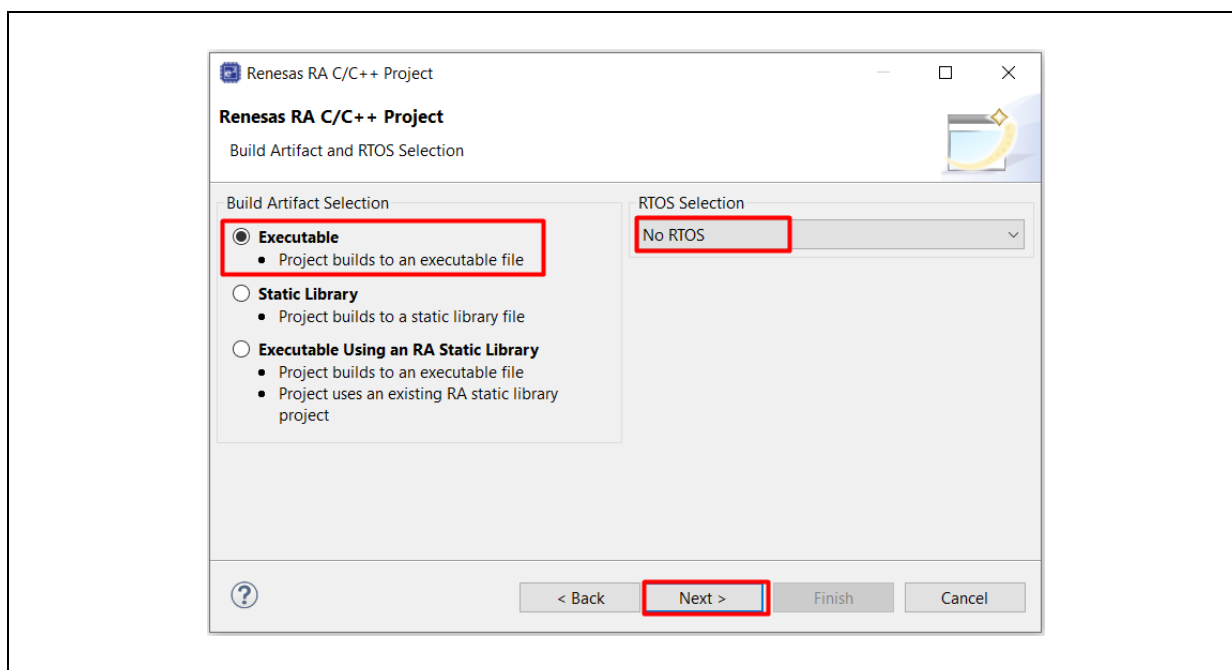


Figure 58. Choose to Build Executable and No RTOS

8. Choose **Bare Metal – Minimal** for the Project Template in the next screen. Click **Finish** to establish the initial project.

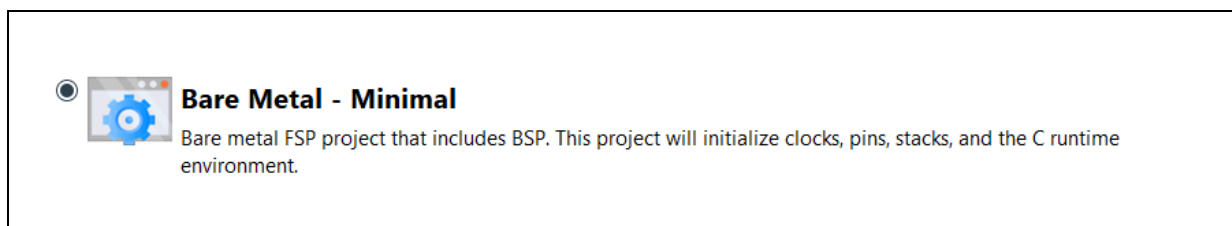


Figure 59. Choose the Project Template

9. If prompted, click **Open Perspective**.

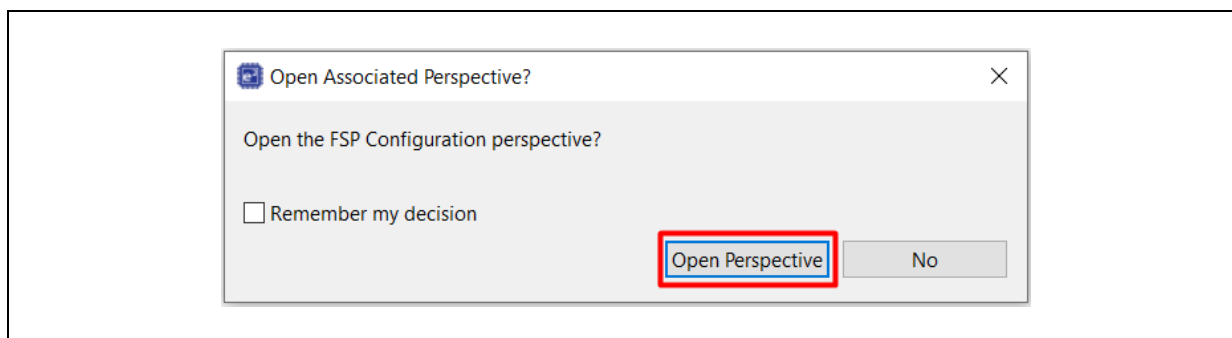


Figure 60. Choose Open the FSP Configuration Perspective

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

10. Click the **Stacks** tab on the RA configurator.
Add **New Stack > Bootloader > MCUboot**.

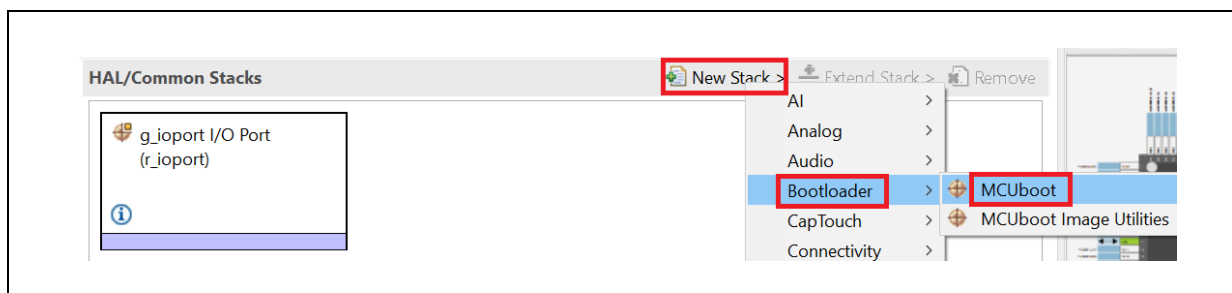


Figure 61. Add the MCUboot Stack

11. Configure the **General** properties of **MCUboot**.
Set **Upgrade Mode** to **Swap**, as shown in Figure 62

MCUboot		
Settings	Property	Value
	▼ Common	
API Info	▼ General	
	Custom mcuboot_config.h	
	Upgrade Mode	Swap
	Validate Primary Image	Enabled
	Downgrade Prevention (Overwrite Only)	Disabled
	Watchdog Feed	
	Measured Boot	Disabled
	Data Sharing	Disabled
	> Signing and Encryption Options	
	> Flash Configuration	
	> Data Sharing	

Figure 62. General Properties for MCUboot with Swap Upgrade Mode

Set **Upgrade Mode** to **Direct XIP** for RA2A2, as shown below figure

	Property	Value
	▼ Common	
	▼ General	
	Custom mcuboot_config.h	
	Upgrade Mode	Direct XIP
	Validate Primary Image	Enabled
	Downgrade Prevention (Overwrite Only)	Disabled
	Watchdog Feed	
	Measured Boot	Disabled
	Data Sharing	Disabled
	> Signing and Encryption Options	
	> Flash Configuration	
	> Data Sharing	

Figure 63. General Properties for MCUboot with Direct XIP Mode (RA2A2 only)

Figure 64. General Configuration for MCUboot Module shows a more detailed application image format that can be referenced to understand the various MCUboot property definitions.

- The header magic number is used for image validation sanity check (refer to the description of **Validate Primary Image**).
- The **image_ok** byte is a flag used by the bootloader for swap test mode confirmation.
- The trailer magic number is written after the image upgrade is finished.

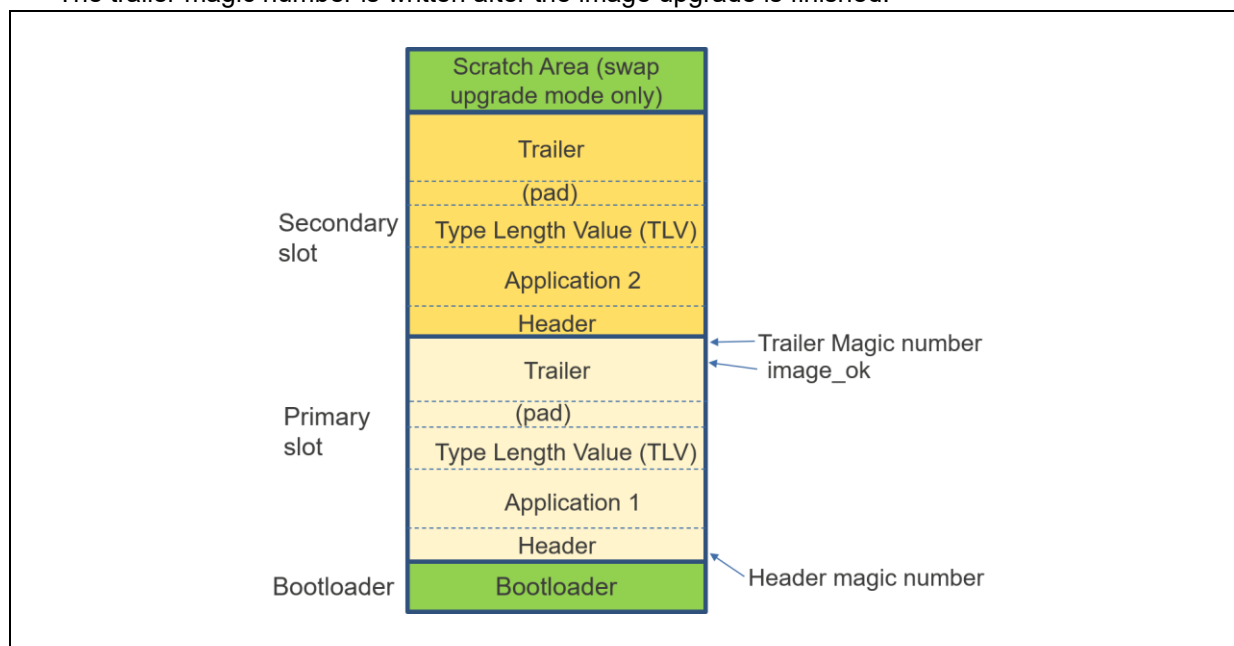
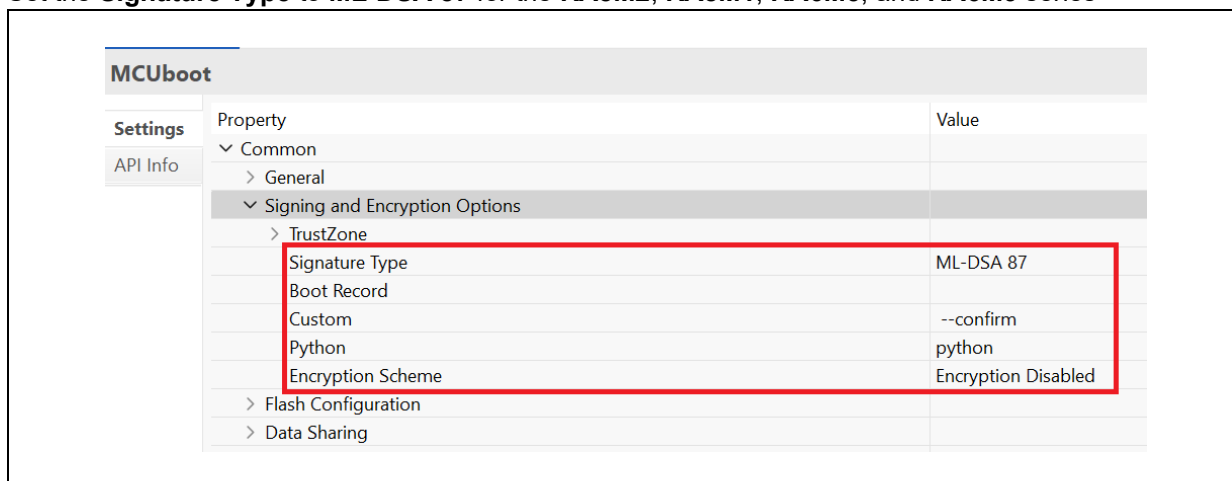
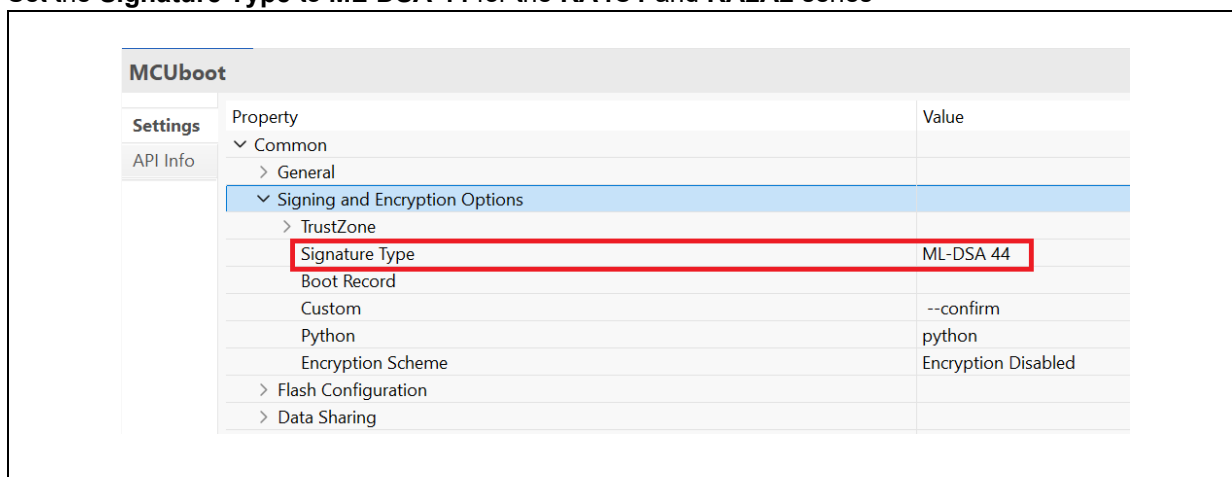


Figure 64. General Configuration for MCUboot Module

The MCUboot General properties are:

- **Custom mcuboot_config.h:** The default mcuboot_config.h file contains the MCUboot Module configuration that you select 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 upgrade method. The available options are Overwrite Only, Overwrite Only Fast, Swap, and Direct XIP.
- **Validate Primary Image:**
 - When Enabled**
 - The bootloader performs:
 - Hash or signature verification (depending on the selected verification method).
 - MCUboot sanity check (based on the image header magic number).
 - The header magic number is always checked as part of the sanity checking prior to the integrity checking and the signature verification.
 - When Disabled**
 - Only sanity check is performed based on the MCUboot header magic numbers.
- **Note**
 - It is **strongly recommended** to enable this property, as it adds critical security handling to the bootloader while requiring less than 32 bytes of additional code.
 - The image magic number is not part of the image validation.
 - It is a **reference value**, useful for sanity checks during the application upgrade debugging process.
 - The image magic number is written to flash only after a successful image upgrade.
- **Downgrade Prevention (Overwrite Only):** This property applies to Overwrite upgrade mode only. When this property is **Enabled**, a new firmware with a lower version number will not overwrite the existing application.

12. Configure the **Signing and Encryption Options**.Set the **Signature Type** to **ML-DSA 87** for the **RA8M2**, **RA8M1**, **RA6M5**, and **RA6M3** series**Figure 65. Signature Type is ML-DSA 87 (RA8M2, RA8M1, RA6M5 and RA6M3)**Set the **Signature Type** to **ML-DSA 44** for the **RA4C1** and **RA2A2** series**Figure 66. Signature Type is ML-DSA 44 (RA4C1 and RA2A2)****The MCUboot Signing and Encryption properties are:**

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

- **Signature Type** is the signing algorithm selection. Application images using MCUboot must be signed to work with MCUboot. At a minimum, this involves adding a hash and an MCUboot-specific constant value in the image trailer.
- When using **PQC** as the cryptographic support for MCUboot, only ML-DSA signature verification is supported. The choices are:
 - **NONE:** This option is selected for the bootloaders that do not support signature verification. With this choice, the bootloader size will be decreased.
 - **ML-DSA 44/65/87:** This option is selected for the example bootloaders that support signature verification included in this application project. ML-DSA-87 provides the highest security level but requires larger keys and longer verification times. ML-DSA-44 offers the best performance with a lower security level.
- **Custom:** This property allows you to input any specific arguments for the signing command. By default --confirm is set for this property, which has the following influence on the Secondary image:
 - For Overwrite upgrade mode, the new image will always overwrite the original application image upon successful verification.
 - For Swap upgrade mode, the Primary image slot will be marked as Confirmed after the swap update. No swap happens upon the next reset after the swap update.

If the **Custom** property is set to --pad, the system behavior is:

- For Overwrite upgrade mode, the system behavior is same as when --confirm is set.

- For Swap upgrade mode, the system behavior depends on whether the application has routines to mark the Primary image slot as Confirmed.

Primary image boot behavior is not influenced by the choice between --confirm or --pad.

Note that the Encryption haven't supported when using PQC, therefore, the **Encryption Scheme** is set to **Disabled**.

- If creating a bootloader with signature verification support, then the **MCUboot Example Keys** stack is required.

Click on the **Add [Optional] Add Example Keys** stack and choose **New** to add **MCUboot Example Keys (NOT FOR PRODUCTION)**.

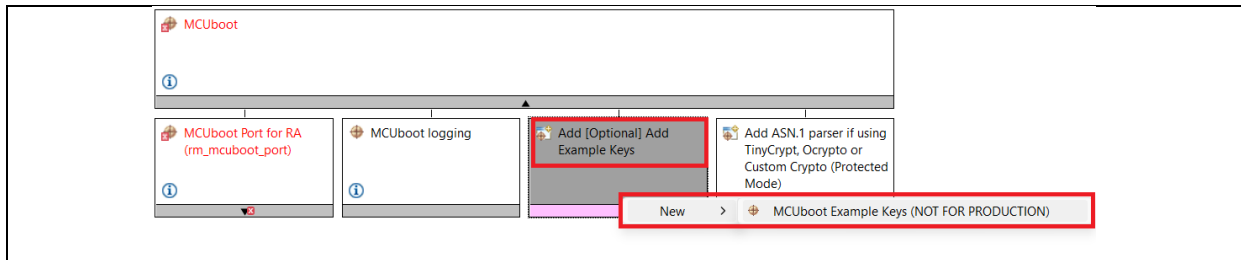


Figure 67. Add the MCUboot Example Keys

- Add the **crypto** module under **MCUboot Port for RA**.

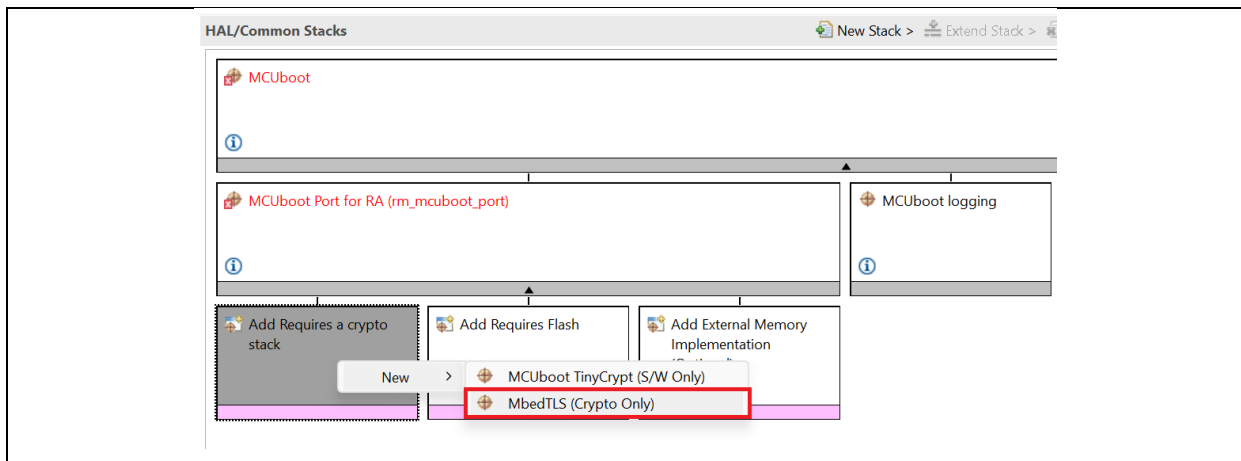


Figure 68. Select MbedTLS (Crypto Only)

- Under **MbedTLS Crypto Port (rm_psa_crypto)**, add **Post Quantum Cryptography (PQC)** stack

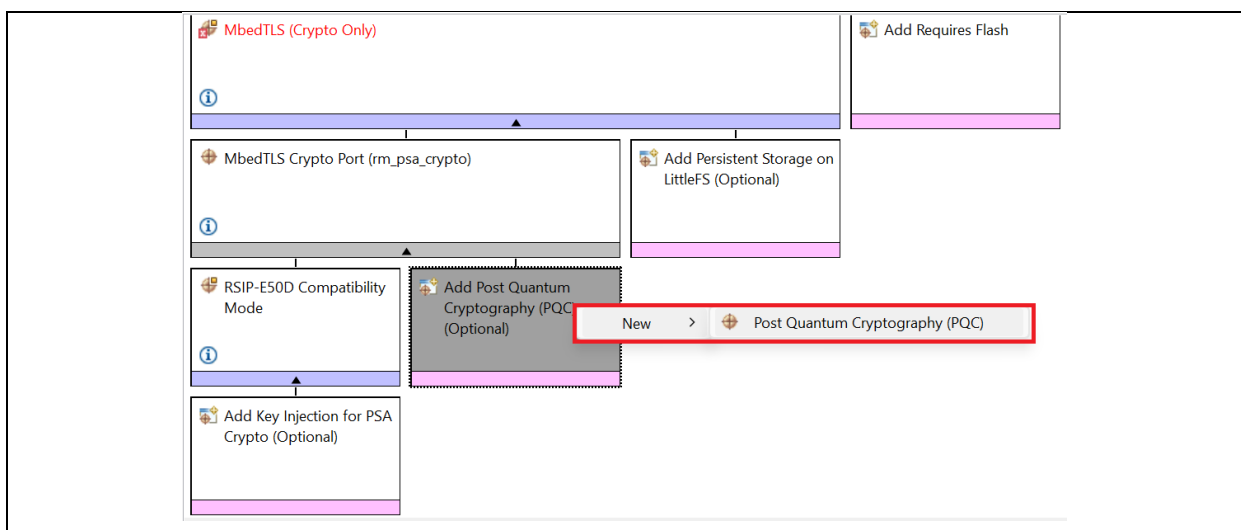


Figure 69. Add Post Quantum Cryptography (PQC) stack

16. **(For RA2A2 only)** Configure the **Flash Configuration** properties of **MCUboot** to enable Dual Bank Mode.

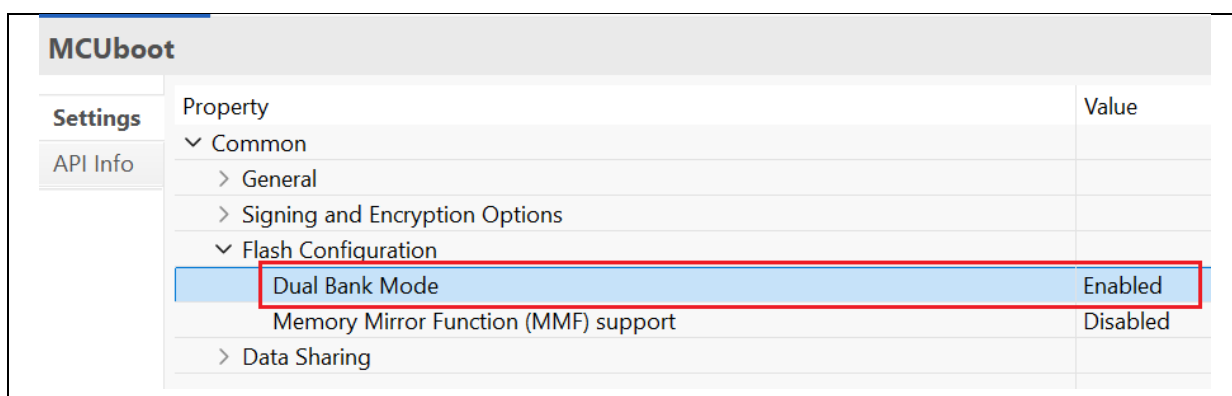


Figure 70. Enable Dual Bank Mode (RA2A2)

4.2 Resolve the Configurator Dependencies

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

1. Resolve dependencies of **MCUboot** as shown in the figures below

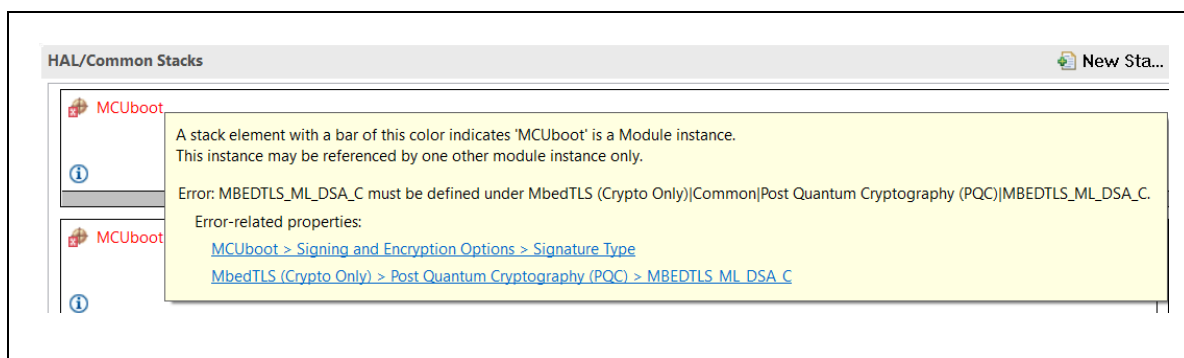


Figure 71. MCUboot dependencies

The Signature Type is ML-DSA, MBEDTLS_ML_DSA_C must be defined. In **MbedTLS (Crypto Only)**, **Post Quantum Cryptography (PQC)** property, set **MBEDTLS_ML_DSA_C** to **Define**

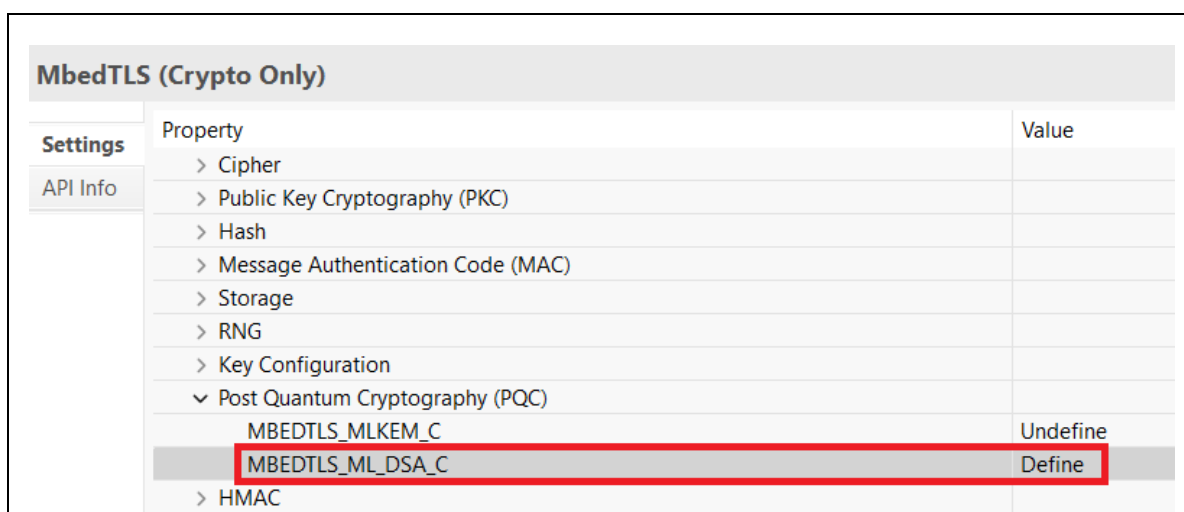


Figure 72. Set MBEDTLS_ML_DSA_C to Define to resolve MCUboot dependencies

2. Resolve dependency of **MCUboot Port for RA** as shown in the figures below

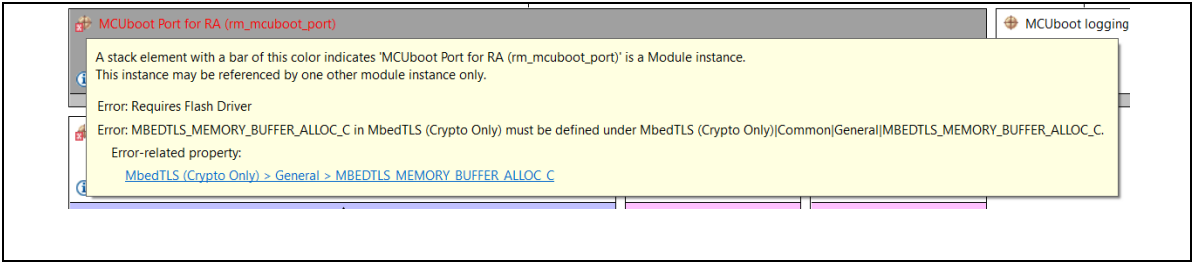


Figure 73. MCUboot Port for RA dependency

In **MbedTLS (Crypto Only)**, under the **General** properties, set **MBEDTLS_MEMORY_BUFFER_ALLOC_C** to **Define** to resolve the MCUboot Port for RA dependencies. This setting is required for the bare-metal MCUboot project because MbedTLS needs to use its own memory allocator in this environment.

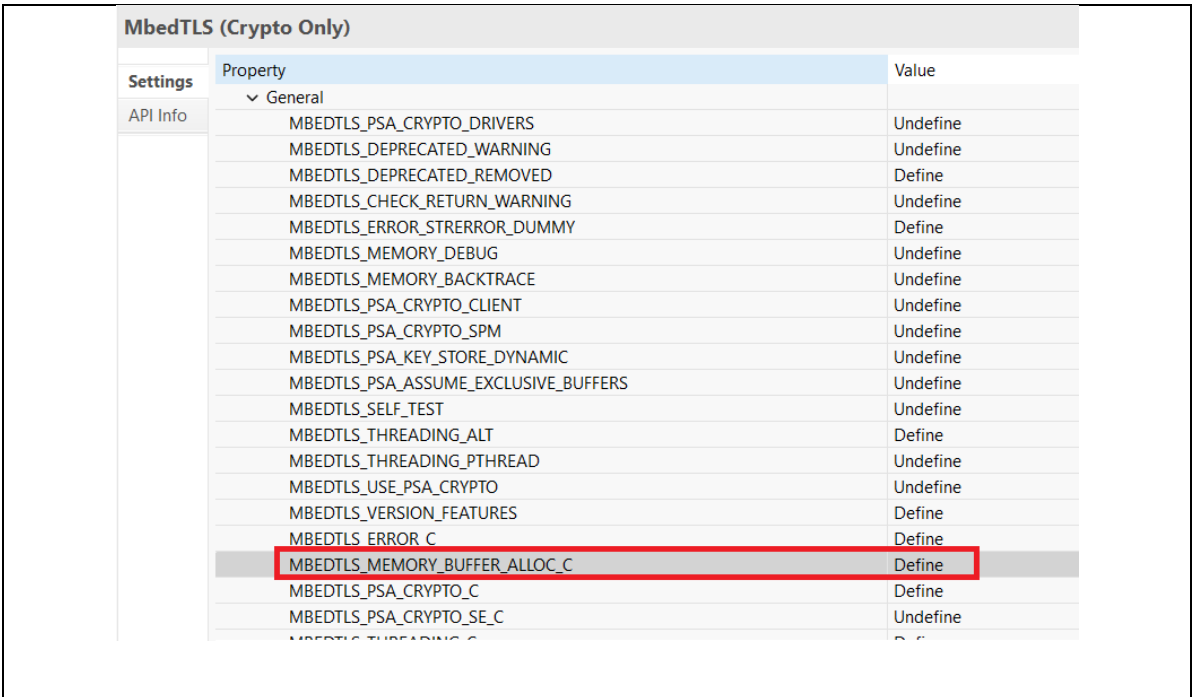


Figure 74. Resolve dependency for MCUboot Port for RA

Add the Flash driver to resolve the remaining dependency for “MCUboot Port for RA”. For the **RA8M2** series, use **MRAM (r_mram)** as the Flash driver. For the **RA8M1**, **RA6M5**, **RA6M3**, and **RA4C1** series, use **r_flash_hp**. For the **RA2A2** series, use **r_flash_lp**.

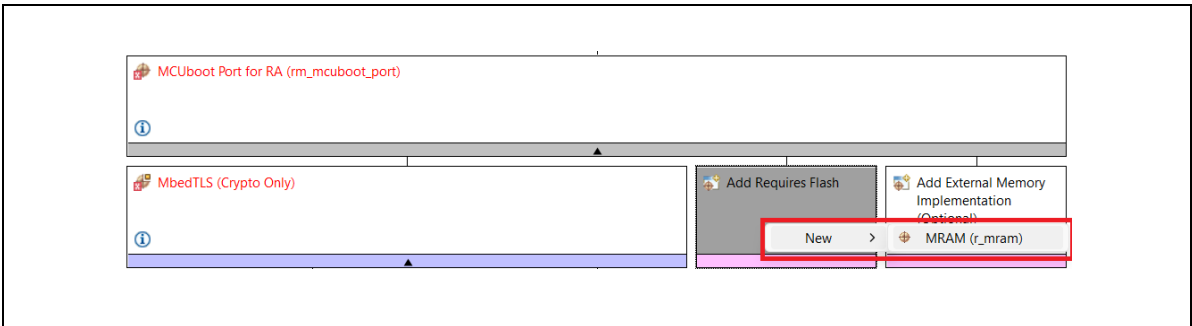


Figure 75. Add MRAM (r_mram) as the Flash driver for RA8M2.

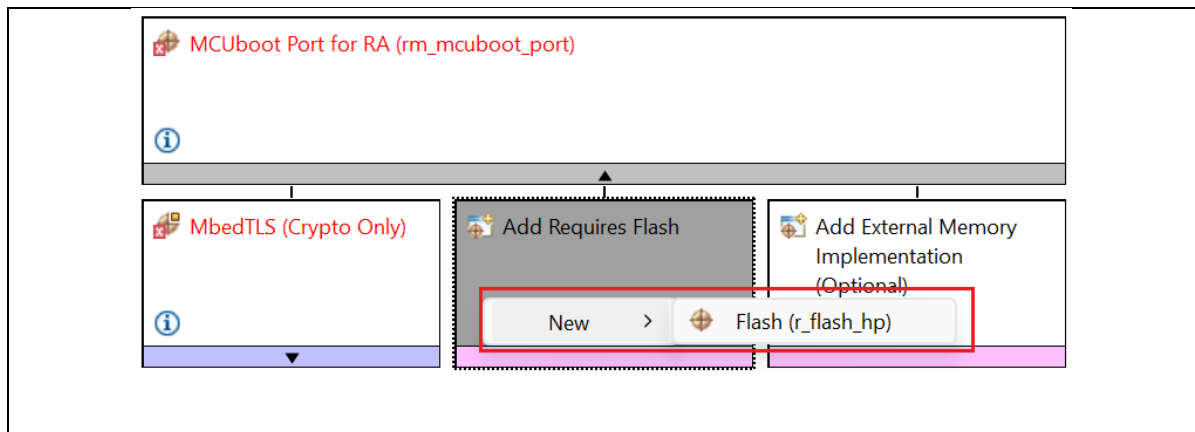


Figure 76. Add Flash HP (r_flash_hp) as the Flash driver for RA8M1, RA6M5, RA6M3 and RA4C1

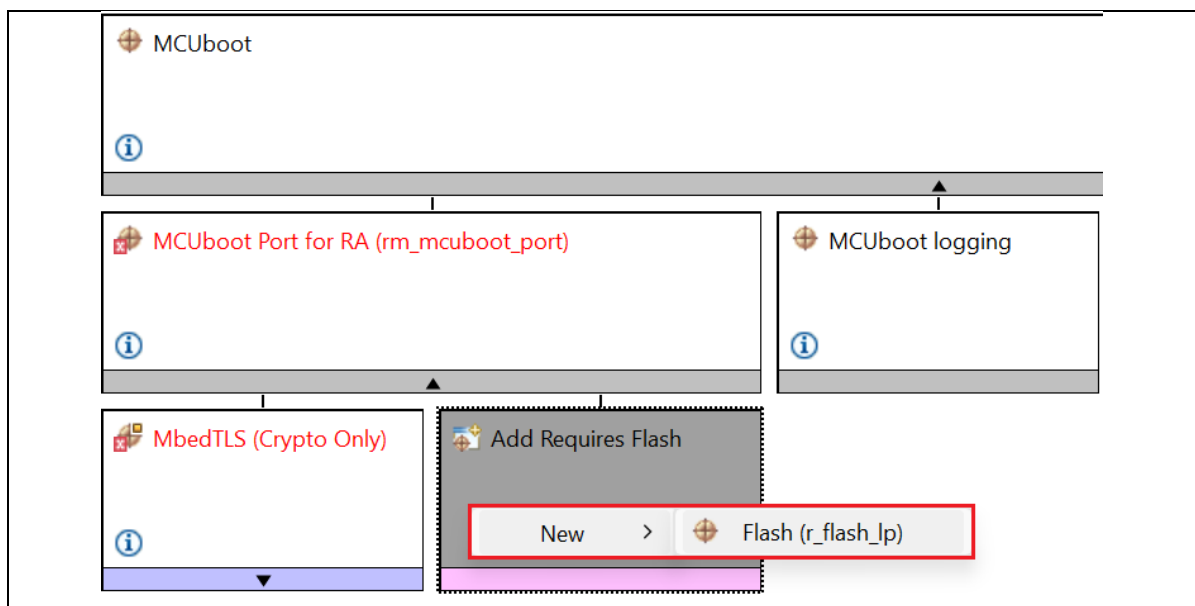


Figure 77. Add Flash LP (r_flash_lp) as the Flash driver for RA2A2

3. Resolve dependency of **MBedTLS (Crypto Only)** as shown in the figures below

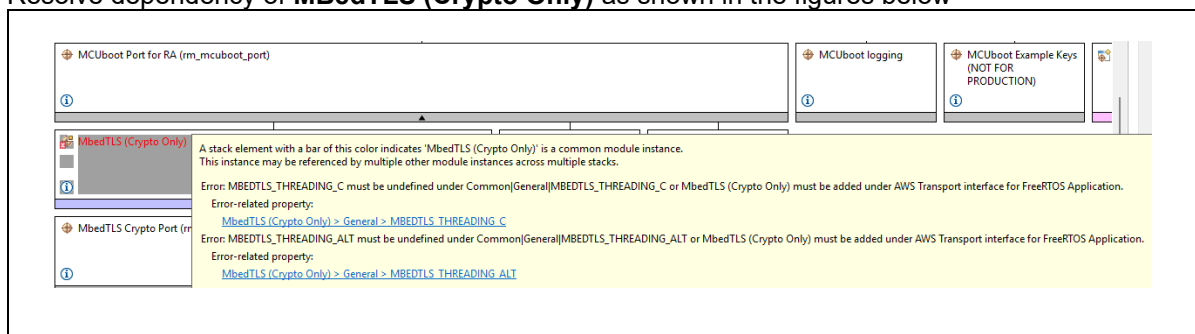


Figure 78. MbedTLS (Crypto Only) dependency

In **MbedTLS (Crypto Only)**, under **General** properties, set **MBEDTLS_THREADING_C** and **MBEDTLS_THREADING_ALT** to **Undefine** to resolve the dependency requirements for a BareMetal MCUboot project.

MbedTLS (Crypto Only)		
Settings	Property	Value
API Info	General	
	Property	Value
	MBEDTLS_PSA_CRYPTO_DRIVERS	Undefine
	MBEDTLS_DEPRECATED_WARNING	Undefine
	MBEDTLS_DEPRECATED_REMOVED	Define
	MBEDTLS_CHECK_RETURN_WARNING	Undefine
	MBEDTLS_ERROR_STRERROR_DUMMY	Define
	MBEDTLS_MEMORY_DEBUG	Undefine
	MBEDTLS_MEMORY_BACKTRACE	Undefine
	MBEDTLS_PSA_CRYPTO_CLIENT	Undefine
	MBEDTLS_PSA_CRYPTO_SPM	Undefine
	MBEDTLS_PSA_KEY_STORE_DYNAMIC	Undefine
	MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS	Undefine
	MBEDTLS_SELF_TEST	Undefine
	MBEDTLS_THREADING_ALT	Undefine
	MBEDTLS_THREADING_PTHREAD	Undefine
	MBEDTLS_USE_PSA_CRYPTO	Undefine
	MBEDTLS_VERSION_FEATURES	Define
	MBEDTLS_ERROR_C	Define
	MBEDTLS_MEMORY_BUFFER_ALLOC_C	Define
	MBEDTLS_PSA_CRYPTO_C	Define
	MBEDTLS_PSA_CRYPTO_SE_C	Undefine
	MBEDTLS_THREADING_C	Undefine
	MBEDTLS_TIMING_C	Undefine
	MBEDTLS_VERSION_C	Define

Figure 79. Resolve dependencies for MbedTLS (Crypto Only)

4. In **BSP** tab, under **RA Common** Property, set the **BSP Main Stack size (bytes)** to **0x2000** and set **Heap size (bytes)** to **0x1800**.

Summary	BSP	Clocks	Pins	Interrupts	Event Links	Linker Sections	Stacks	Components
Properties	×	Problems	Smart Browser					
EK-								
Settings	Property							Value
	▼ RA Common							
	Main stack size (bytes)							0x2000
	Heap size (bytes)							0x1800
	Bootloader Secondary XIP							Disabled
	MCU Vcc (mV)							3300

Figure 80. Update the BSP Main Stack Size

Note: It is recommended to set the Main Stack size to 0x2000 and the Heap size to 0x1800. If the allocated sizes are insufficient, for example, a Main Stack size of 0x1500 and a Heap size of 0x1800—the following issue may be encountered during MCU debugging:

primary_app Debug_Flat [Renesas GDB Hardware Debugging] primary_app.elf [1] [cores: 0] Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:Trace/b Default_Handler() at startup.c:70 0x4c16 <signal handler called>() at 0xffffffff9 mbedtls_aes_setkey_enc() at ctr_drbg_alt.c:1,447 0x1654 ctr_drbg_update_internal() at ctr_drbg_alt.c:409 0x1590 mbedtls_ctr_drbg_random_with_add() at ctr_drbg_alt.c:705 0x18a6 mbedtls_ctr_drbg_random() at ctr_drbg_alt.c:728 0x1cf8 mbedtls_mdsha_get_random() at mldsa.h:91 0xa12	arm-none-eabi-gdb (16.2) Renesas GDB server (Host)
--	---

Figure 81. Debug issue due to insufficient stack and heap allocation

5. Disabling RSA to save BSP Heap size for other possible user requirements. Under **Public Key Cryptography (PKC)**, navigate to the **RSA**, set **MBEDTLS_RSA_C** to **Undefine**

Mbedtls (Crypto Only)		
Settings	Property	Value
API Info	Common	
	Hardware Acceleration	
	Platform	
	General	
	Cipher	
	Public Key Cryptography (PKC)	
	DHM	
	ECC	
	RSA	
	MBEDTLS_PK_RSA_ALT_SUPPORT	Undefine
	MBEDTLS_RSA_NO_CRT	Define
	MBEDTLS_RSA_C	Undefine
	MBEDTLS_RSA_GEN_KEY_MIN_BITS	Undefine
	MBEDTLS_RSA_GEN_KEY_MIN_BITS value	1024
	MBEDTLS_GENPRIME	Define

Figure 82. Set MBEDTLS_RSA_C to undefine

6. Resolve dependency of **Flash (r_flash_hp)** and **Flash (r_flash_lp)**
Set the Code Flash Programming to Enabled. As Data Flash Programming, Data Flash Background Operation Support, and Data Flash Background Operation are not used in the bootloader, select Disabled for these two properties to reduce the bootloader memory footprint.

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

Figure 83. Resolve dependency for Flash HP

g_flash0 Flash (r_flash_lp)		
Settings	Property	Value
API Info	Common	
	Parameter Checking	Default (BSP)
	Code Flash Programming	Enabled
	Code Flash Background Operation Support With Bank Programming	Disabled
	Data Flash Programming	Disabled
	Data Flash Background Operation Support	Disabled
	Instant Bank Swap (No Reset Required)	After Reset
	Module g_flash0 Flash (r_flash_lp)	
	Name	g_flash0
	Data Flash Background Operation	Disabled
	Callback	NULL
	Flash Ready Interrupt Priority	Disabled

Figure 84. Resolved dependency for Flash LP

For more details, refer to RA2 Secure Firmware update using MCUboot and Flash Dual Bank (R11AN1069) application note.

4.3 Using MCUboot Quick Setup

After resolving the configurator dependencies, follow the steps below to implement the bootloader project using MCUboot Quick Setup in Developer Assistance:

1. Save configuration.xml and click **Generate Project Content**.

Expand the **Developer Assistance > HAL/Common > MCUboot > Quick Setup** and drag **Call Quick Setup** to the top of the `hal_entry.c` of the bootloader project.

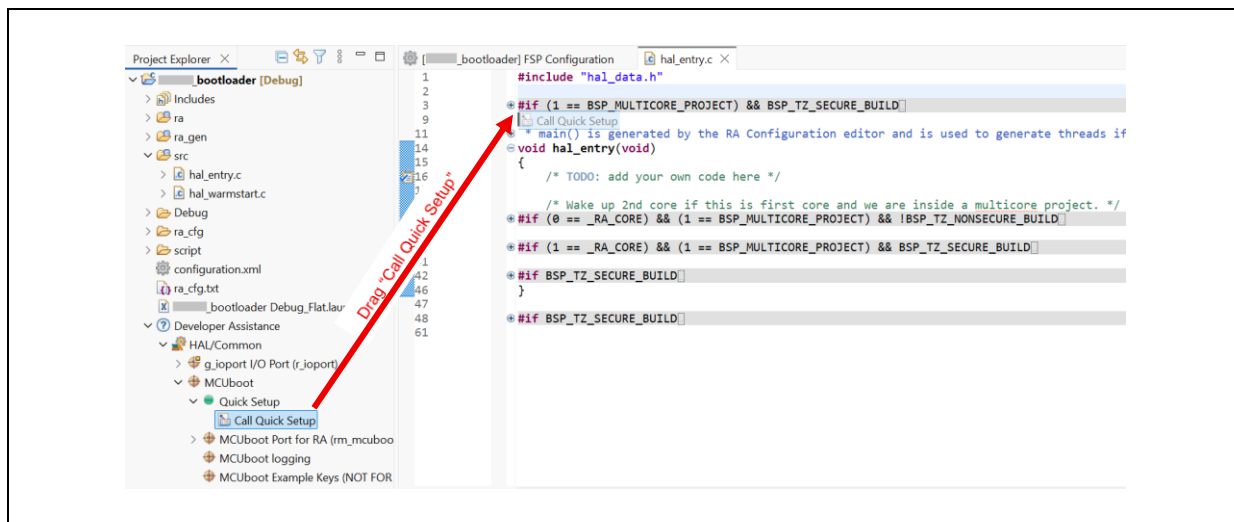


Figure 85. Add mcuboot_quick_setup() using Developer Assistance

Call `mcuboot_quick_setup()` function on the top of the `hal_entry()` function:

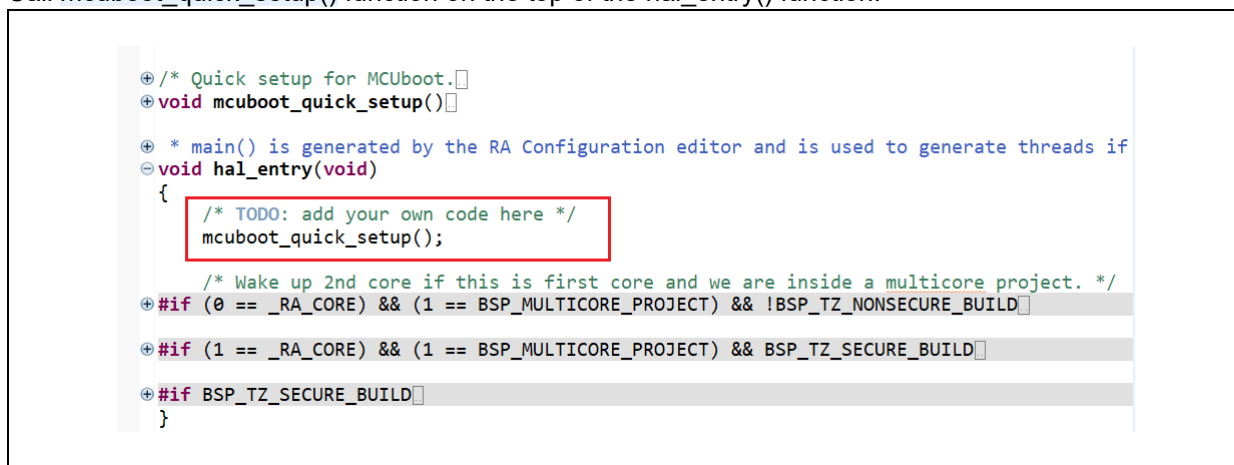


Figure 86. Add function call to hal_entry()

- By default, the **I/O Port Driver** is brought into the project when the project is created. As the **I/O Port Driver** is not used in the bootloader project, this stack can be removed to reduce the bootloader project size. Right click on the **I/O Port** stack and choose **Delete**.

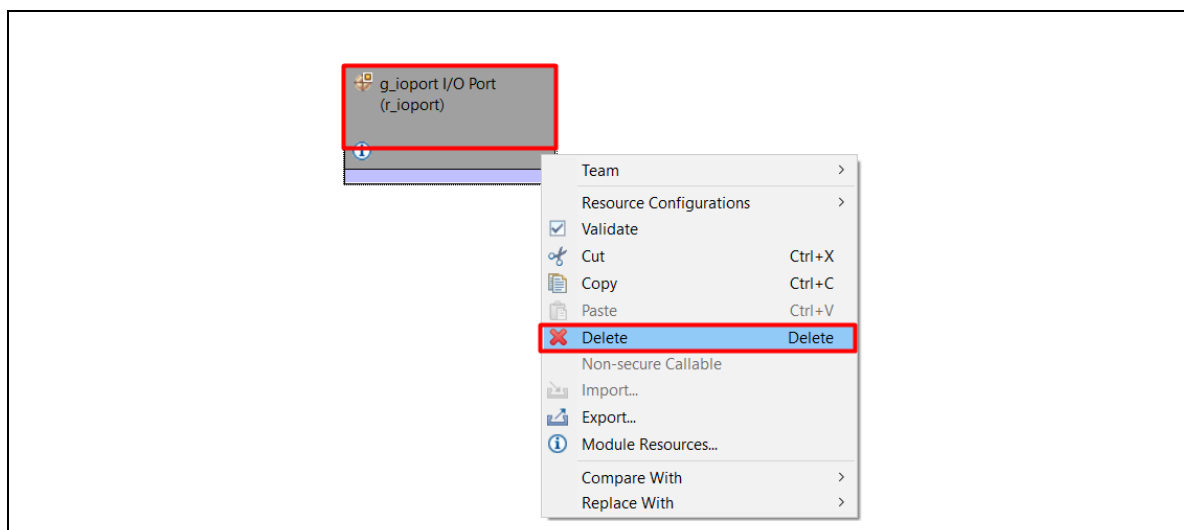


Figure 87. Remove the I/O Port Stack

After the I/O Port is deleted, remove all sections of code referencing the I/O Port API. For example, in `hal_warmstart.c` file, remove configure pins source code.

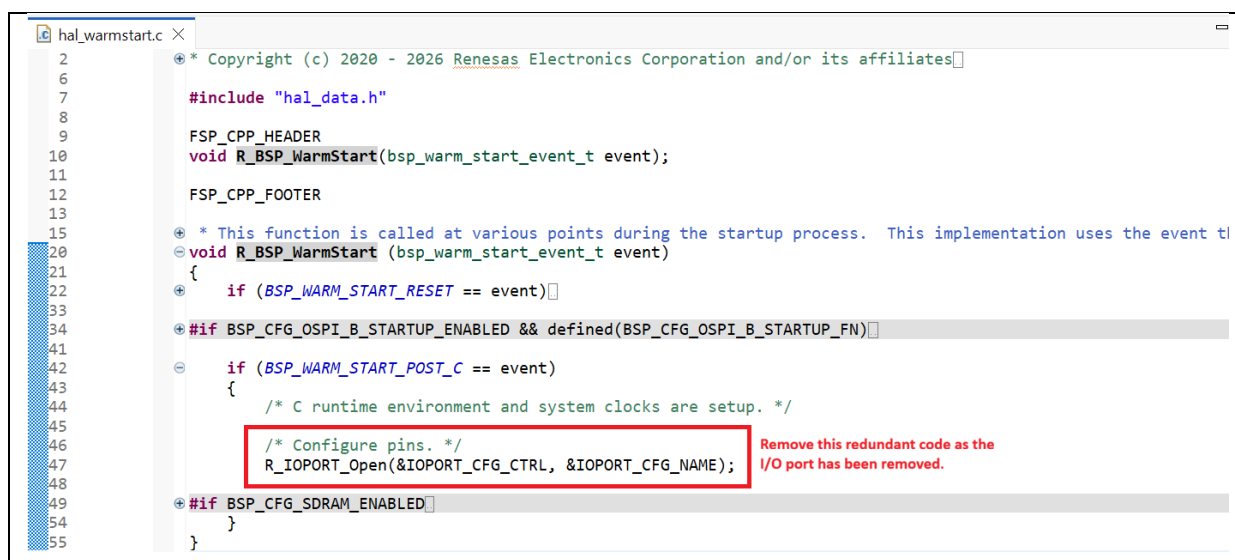


Figure 88. Remove I/O Port redundant code in hal_warmstart.c

4.4 Compiling the Bootloader Project

Depending on the upgrade mode selected, the size may differ slightly. In addition, migrating projects to a later FSP version may result in minor size variations.

```
llvm-size --format=berkeley "ra8m2_bootloader.elf"
text      data      bss      dec      hex filename
43742     20      19896   63658   f8aa ra8m2_bootloader.elf
llvm-objcopy "ra8m2_bootloader.elf" -O srec "ra8m2_bootloader.srec"

18:38:21 Build Finished. 0 errors, 0 warnings. (took 1s.666ms)
```

Figure 89. Compilation Result in RA8M2

```
llvm-size --format=berkeley "ra8m1_bootloader.elf"
text      data      bss      dec      hex filename
41658      0      20348   62006   f236 ra8m1_bootloader.elf

16:59:32 Build Finished. 0 errors, 0 warnings. (took 2s.11ms)
```

Figure 90. Compilation Result in RA8M1

```
llvm-size --format=berkeley "ra6m5_bootloader.elf"
text      data      bss      dec      hex filename
42430      0      20380   62810   f55a ra6m5_bootloader.elf
llvm-objcopy "ra6m5_bootloader.elf" -O srec "ra6m5_bootloader.srec"

20:08:50 Build Finished. 0 errors, 30 warnings. (took 16s.251ms)
```

Figure 91. Compilation Result in RA6M5

```
llvm-size --format=berkeley "ra6m3_bootloader.elf"
text      data      bss      dec      hex filename
37434      0      20364   57798   e1c6 ra6m3_bootloader.elf

10:23:40 Build Finished. 0 errors, 0 warnings. (took 830ms)
```

Figure 92. Compilation Result in RA6M3

```
llvm-size --format=berkeley "ra4c1_bootloader.elf"
text      data      bss      dec      hex filename
40294      0      20560   60854   edb6 ra4c1_bootloader.elf
llvm-objcopy "ra4c1_bootloader.elf" -O srec "ra4c1_bootloader.srec"

19:32:17 Build Finished. 0 errors, 0 warnings. (took 5s.758ms)
```

Figure 93. Compilation Result in RA4C1

```
llvm-size --format=berkeley "ra2a2_bootloader.elf"
text      data      bss      dec      hex filename
27366      0      18344   45710   b28e ra2a2_bootloader.elf

16:34:52 Build Finished. 0 errors, 0 warnings. (took 463ms)
```

Figure 94. Compilation Result in RA2A2

4.5 Configuring the Python Signing Environment

Signing the application image can be done using a post-build step in e² studio using the image signing tool `imgtool.py`, which is included with MCUboot. This tool is integrated as a post-build tool in e² studio to sign the application image.

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

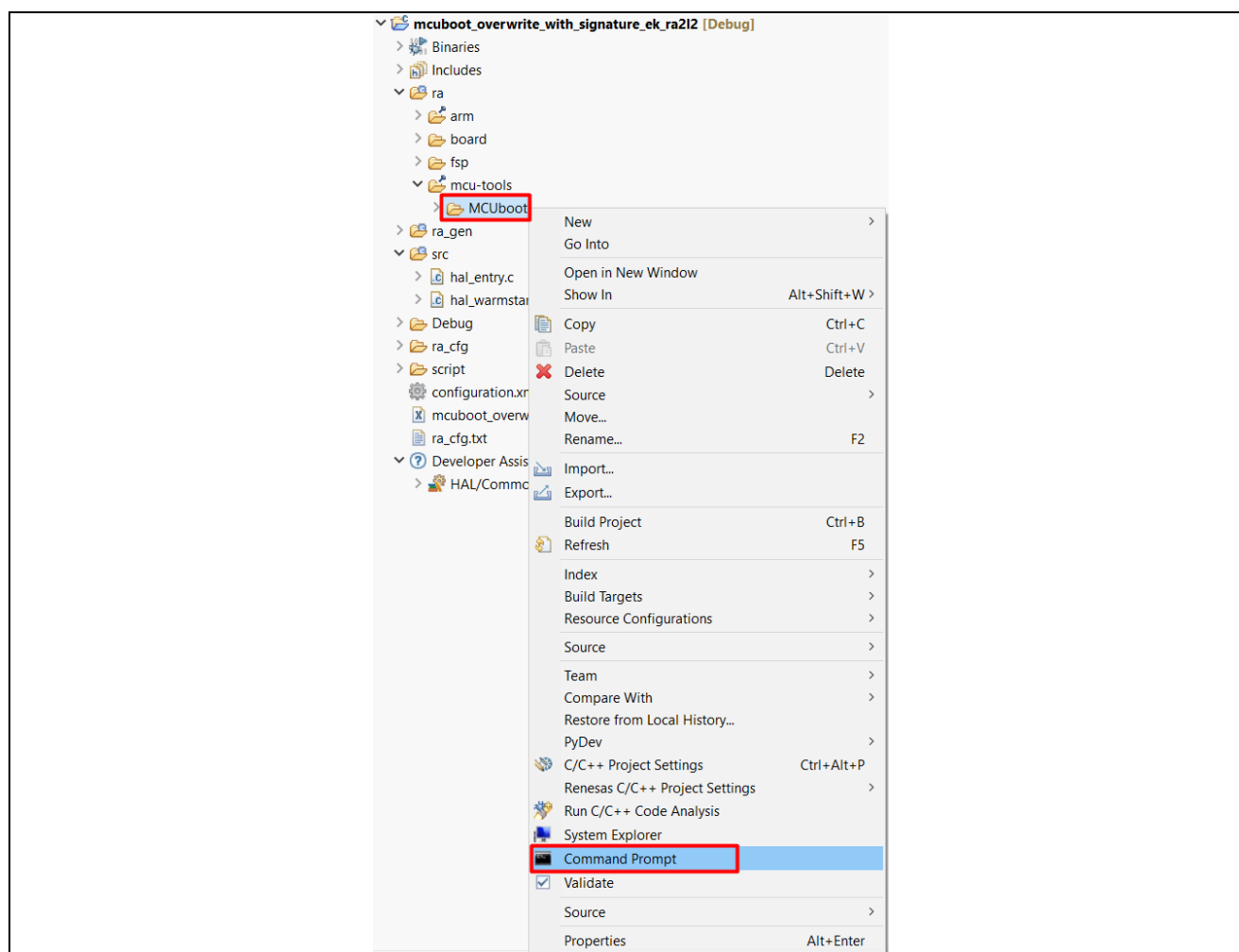


Figure 95. Open the Command Prompt

It is recommended to upgrade pip prior to installing the dependencies. Enter the following command to update pip:

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

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.

5. Using the Bootloader with Application

Developing a user application to use MCUboot typically starts with developing and testing the application and the bootloader independently. For use with the bootloader, the user application must consider the following common steps:

- Adjust the memory map of the bootloader to allow the user application and bootloader to fit the available MCU memory area.
- Configure the application to use the bootloader.
- Signing the user application image.

This section shows how to use the MCUboot bootloader with the LED blinky. After the initial blinky project is created, the project is configured to use the MCUboot bootloader project, generated in the previous section, and is also signed using the signing command from the bootloader project.

The sample code uses SEGGER RTT for log output. Because this feature is used only for testing, it is not covered in this chapter.

5.1 Generate the Primary User Application Project

Follow the steps below to create a blinky application project as the Primary User Application Project. The steps in section 0 are identical when generating a blinky project. Launch e² studio and open a Workspace then:

1. Click **File > New > Renesas C/C++ Project > Renesas RA** and choose **Renesas RA C/C++ Project**. Click **Next**.
2. Assign the project name based on Table 5.

Table 5 Name the Initial Application Project

Bootloader project name	User application project name
ra8m2_bootloader	ra8m2_primary_app_cpu0 ra8m2_primary_app_cpu1 (*)
ra8m1_bootloader	ra8m1_primary_app
ra6m5_bootloader	ra6m5_primary_app
ra6m3_bootloader	ra6m3_primary_app
ra4c1_bootloader	ra4c1_primary_app
ra2a2_bootloader	ra2a2_primary_app

(*)1) Create and build the ra8m2_primary_app_cpu0 project before creating the ra8m2_primary_app_cpu1 project. For details, refer to step (8).

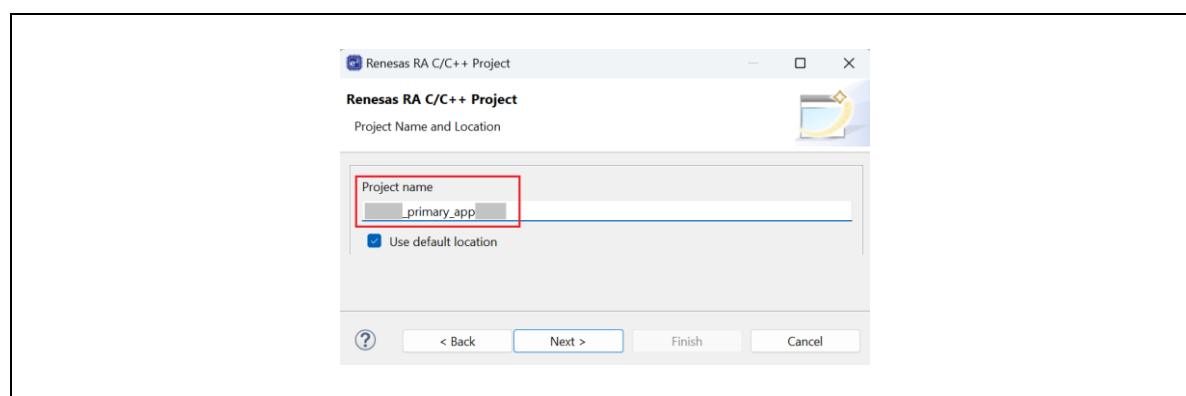


Figure 96. Set Primary application Project name

3. Click **Next**, choose **RA board** as the **Board** from the drop-down menu, and choose the **LLVM Embedded Toolchain**. Click **Next**.
4. Choose Project Type is **Flat (Non-Trustzone) Project** and click **Next**.
5. Choose **Preceding Project** with Bootloader Project (xxxx_bootloader) from the drop-down menu:

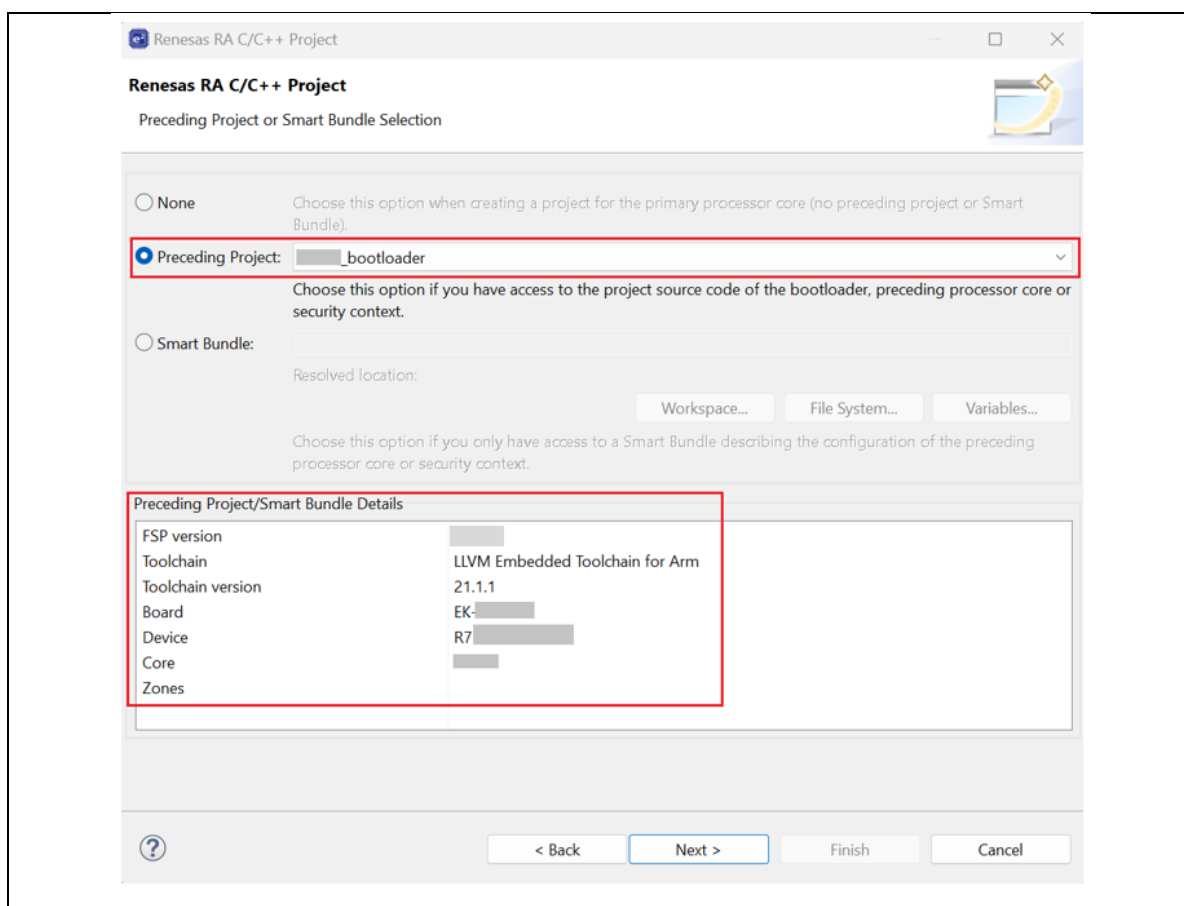


Figure 97. Selecting the Preceding Project

6. In the next screen, select **Executable** as the **Build Artifact** and **No RTOS** for the **RTOS Selection**. Click **Next**.
7. Select the **Bare Metal - Blinky** as the **Project Template** for the board and click **Finish**. The initial application project is now created. Before proceeding to Step (8) or Section 5.3, ensure that the project has been built successfully with zero errors. Otherwise, the subsequent project cannot be linked to it as the Preceding Project.

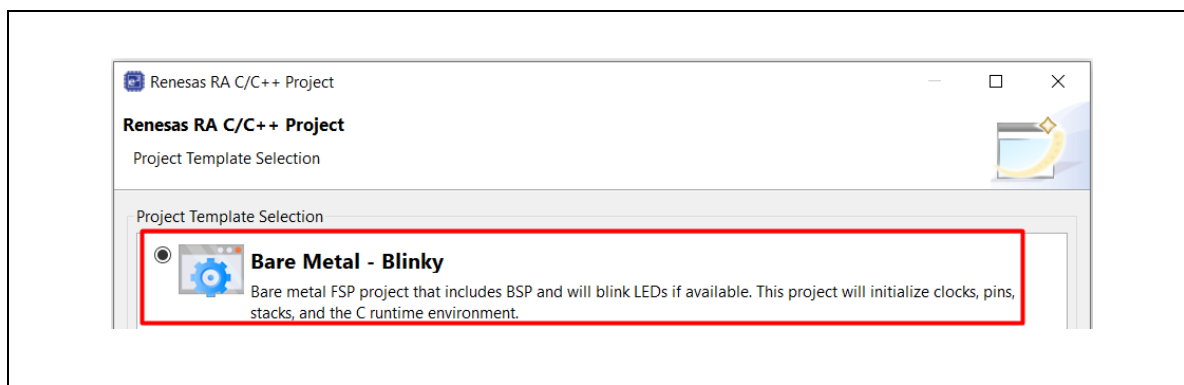


Figure 98. Choose Bare Metal – Blinky as Project Template

8. (Only applies for MCUs that support dual core, e.g. RA8M2) Create the primary project for CPU1:
 - a. Build primary project of CPU0, mention in this application note: ra8m2_primary_app_cpu0.
 - b. Repeat creating project from step 1 & 2 as above to create and assign project name as in Table 5.
 - c. In **Device & Tools selection** step, choose **Core: CPU1** as below picture.

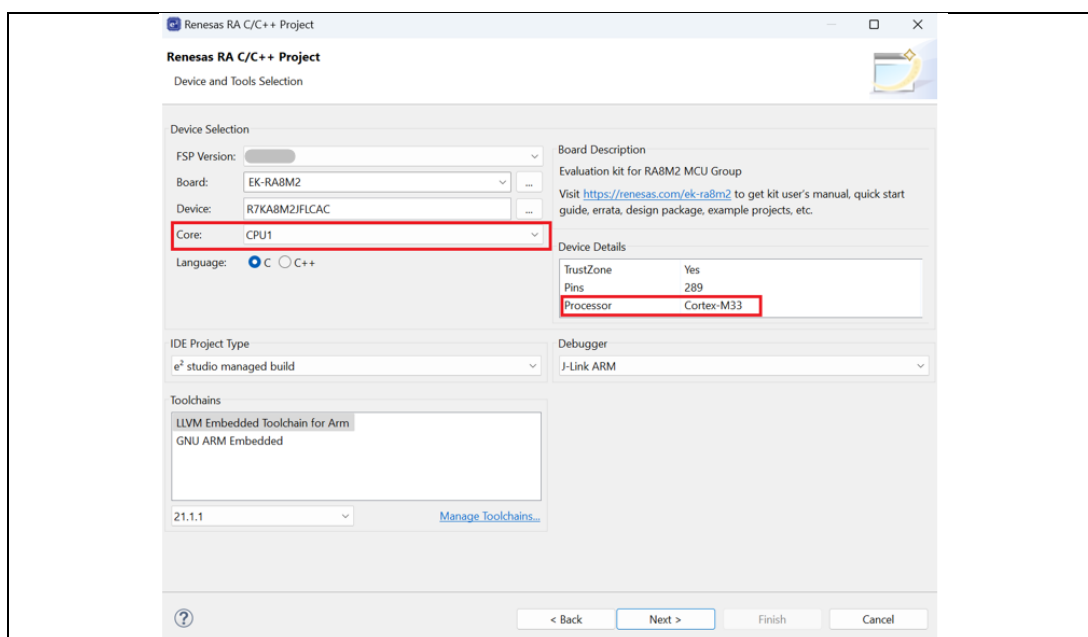


Figure 99. Device and tools selection for CPU1

- d. Choose **Project Type** is **Flat (Non-TrustZone) Project** and click Next.
- e. Choose **Preceding Project** with Primary application project for CPU0 (**ra8m2_primary_app_cpu0**) from the drop-down menu:

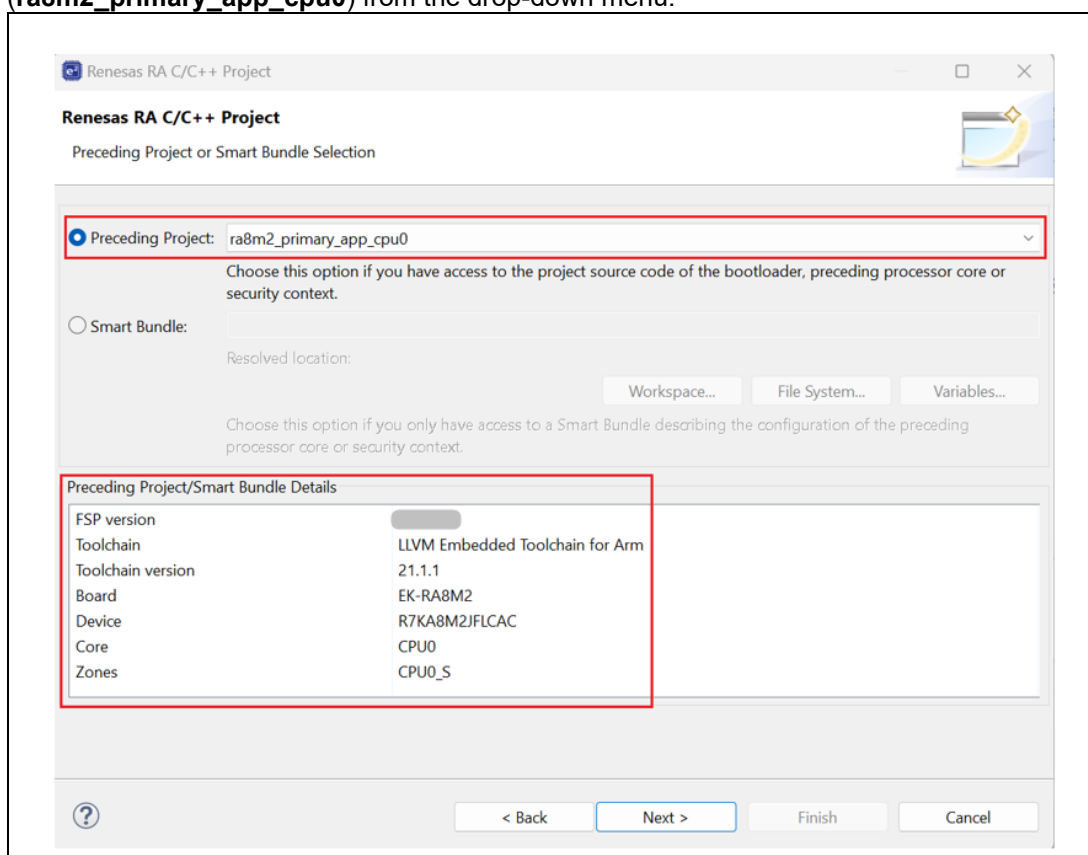


Figure 100. Choose preceding project for RA8M2 primary application CPU1

- f. Repeat step 6 & 7 as above to complete creating **Executable, No RTOS** and **Bare Metal – Blinky** project.

5.2 Signing the User Application Image

Note: If the bootloader project is re-built after changing any of the signing and signature **Properties** of the MCUboot module, the developer must repeat **Generate Project Content** to update the .sbd file, because the application project is linked to the bootloader project through this file.

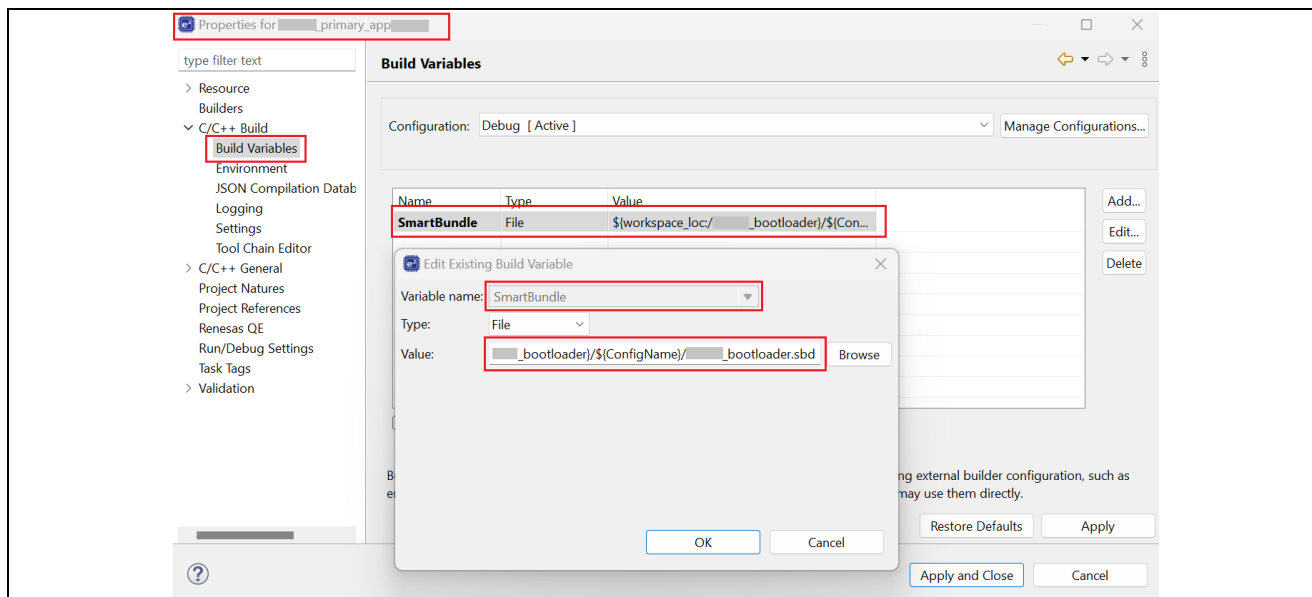


Figure 101. Access the .sbd file in the Application Project

Each application can have a defined version number. This version number can be used in upgrade mode. This is achieved by defining an Environment Variable: MCUBOOT_IMAGE_VERSION.

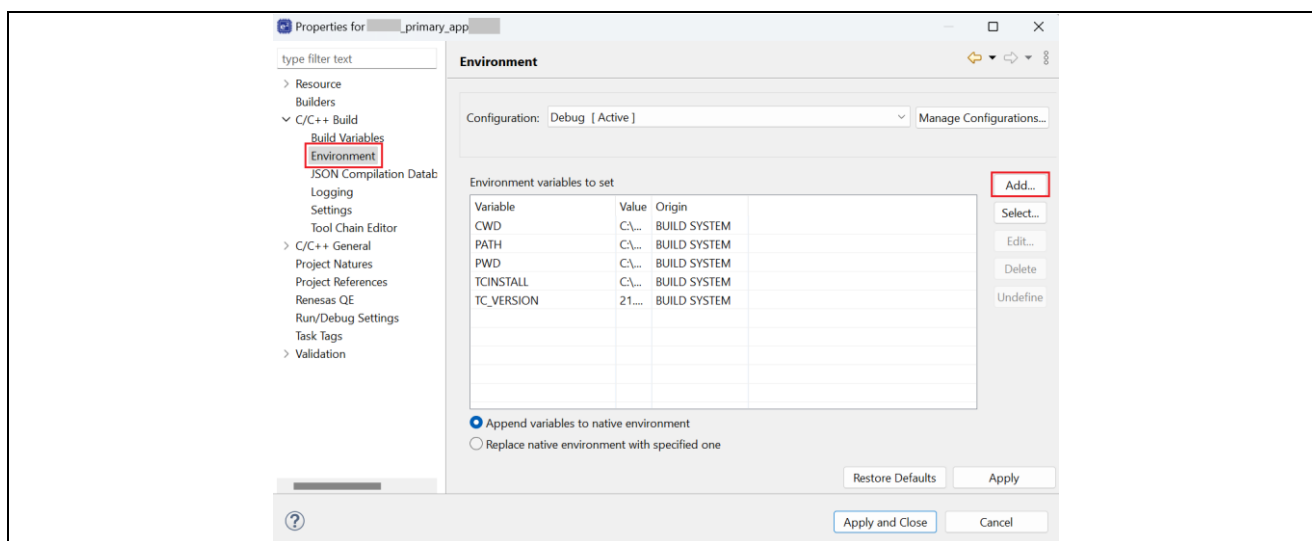


Figure 102. Adding New Environment Variable

Add the Environment Variable: **MCUBOOT_IMAGE_VERSION** for the application image version.

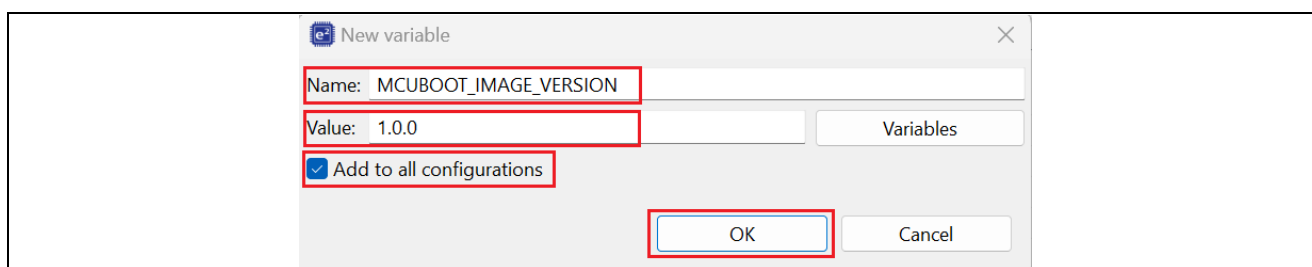


Figure 103. Add MCUBOOT_IMAGE_VERSION Variable

Set the Environment Variable: **MCUBOOT_IMAGE_SIGNING_KEY**

Value: `${workspace_loc:<...>_bootloader}/ra/mcu-tools/MCUboot/root-mlds-87.pem`

ML-DSA MCUboot Example Keys can be generated and be available at bootloader project
 “<ra2a2/ra4c1/ra6m3/ra6m5/ra8m1/ra8m2>_bootloader}/ra/mcu-tools/MCUboot/root-<key>.pem”

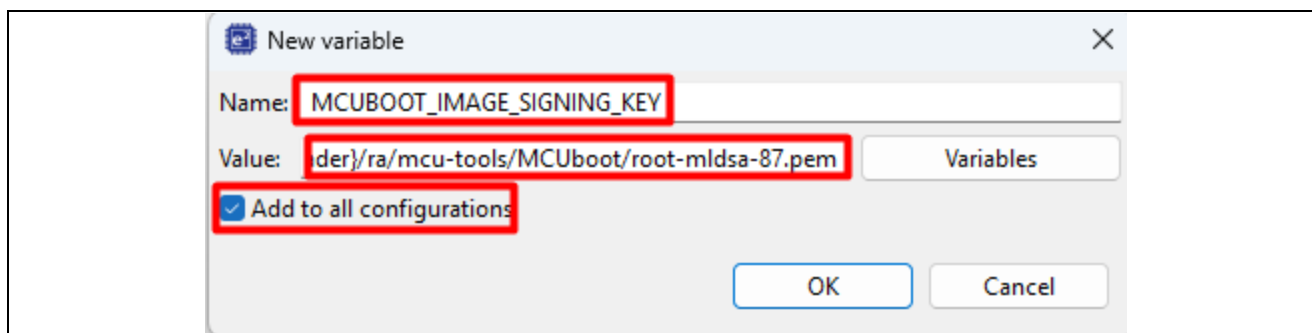


Figure 104. Add MCUBOOT_IMAGE_SIGNING_KEY Variable (MLDSA-87)

When using the **LLVM Embedded Toolchain**, add the Environment Variable:
MCUBOOT_APP_BIN_CONVERTER to sign the application image.

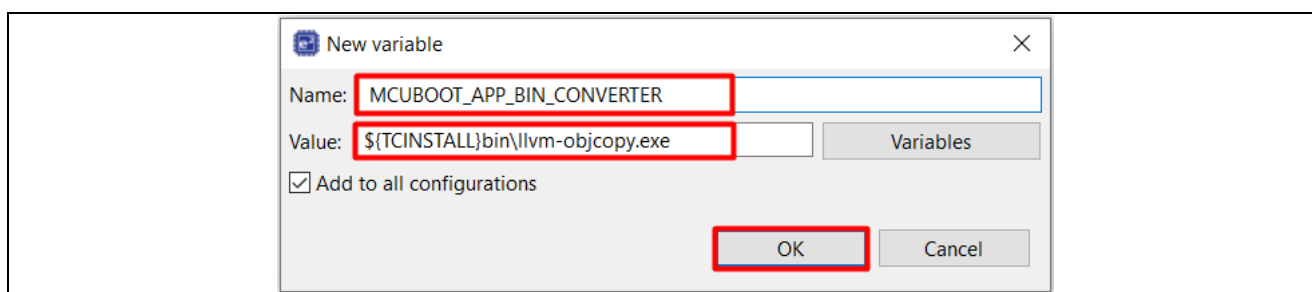


Figure 105. Add MCUBOOT_APP_BIN_CONVERTER Variable

Note:

- The value of the **MCUBOOT_APP_BIN_CONVERTER** variable corresponds to the installation path of the LLVM toolchain - specifically, `${TCINSTALL}bin\llvm-objcopy.exe`.

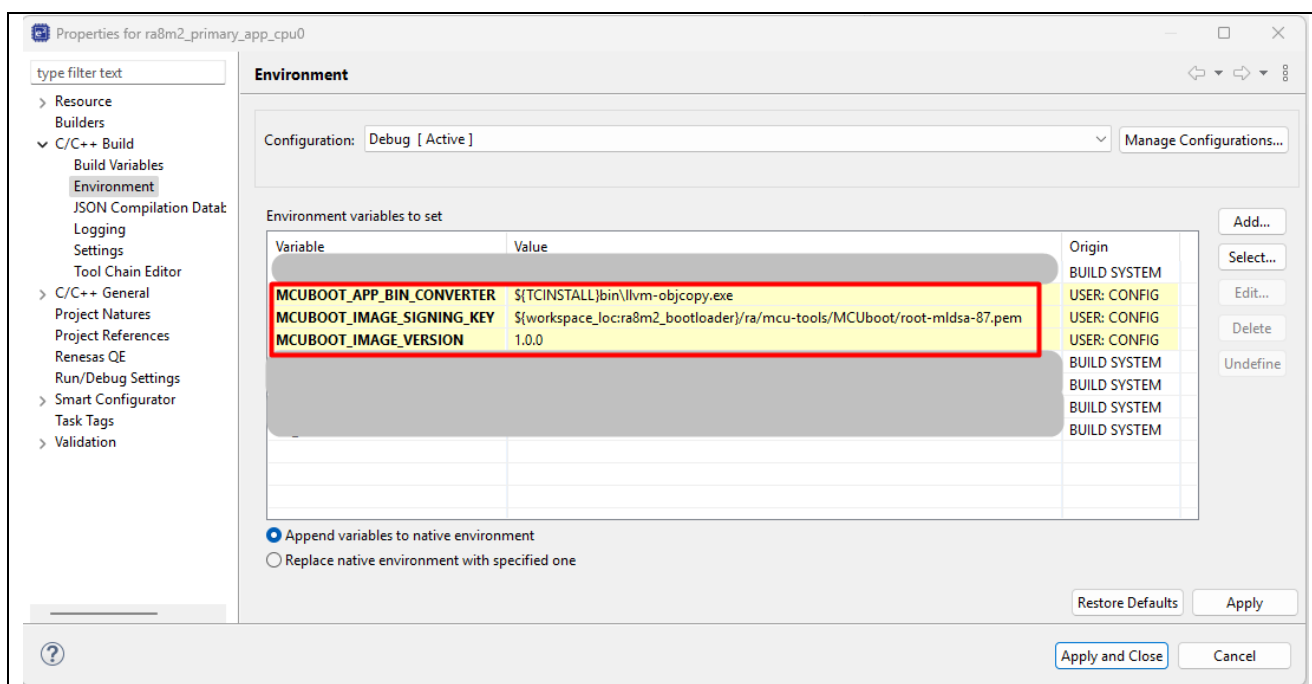


Figure 106. Configuration result

To ensure that the project is re-built when the Environment Variables or the linker script is modified, it is recommended to add a **Pre-build** step that will delete the .elf file, as shown in **Figure 107**.

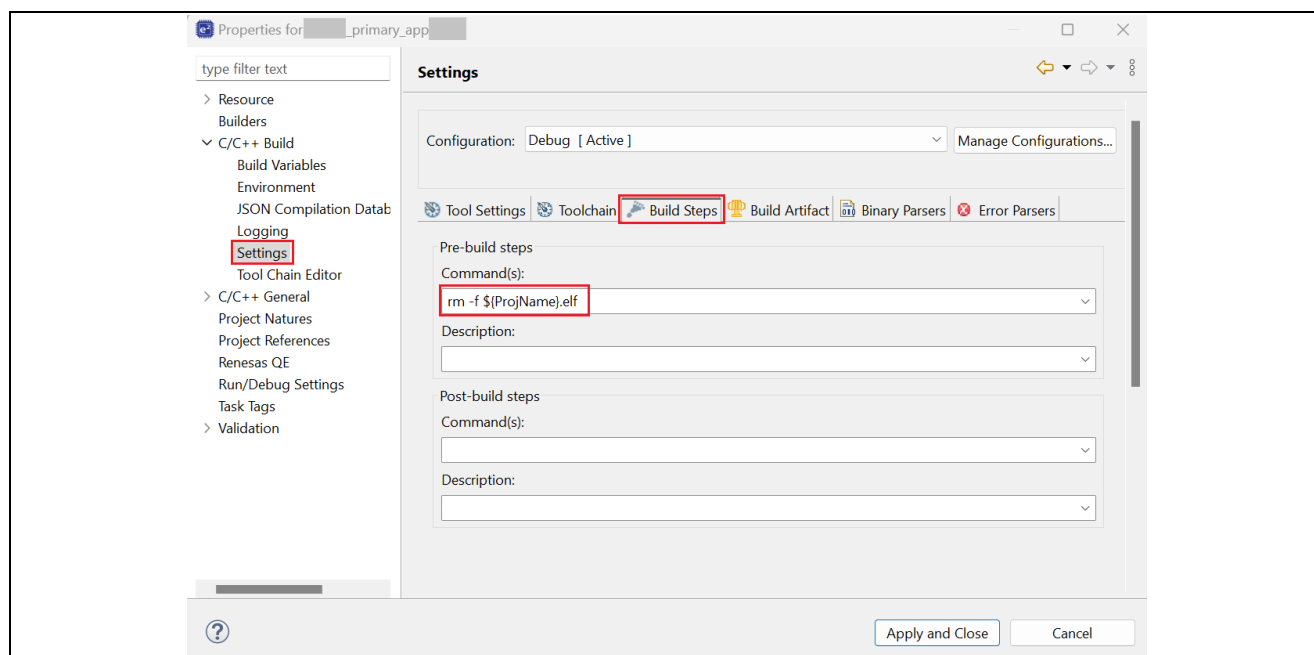


Figure 107. Configuration of the Pre-build Command

Note: For MCU support dual core, e.g. RA8M2, repeat these signing steps for application on CPU1.

Build primary application project. When building the application project, an error will appear in the **Console** tab, as shown in **Figure 108**



Figure 108. Error message when building the application project

To resolve this issue, users should refer to next section for the detailed procedure.

5.3 MCUboot Memory Configuration with Renesas FSP Solution Project

Starting from FSP v6.0.0 with e²studio v2025-04.1, when creating a Bootloader Project, developers must use the **Renesas FSP Solution Project (Advanced)** project template to merge the MCUboot and Application projects together. This provides a more effective and intuitive bootloader memory management process by allowing users to edit the “solution.xml” file within the **Solution Project**.

The **FSP Solution Project (Advanced)** is a chain of projects, specifically consisting of the MCUboot bootloader and the application project that have been created in the previous steps.

5.3.1 How to Set Up the Renesas FSP Solution Project

1. Click **File > New > C/C++ Project** and select **Renesas FSP Solution Project (Advanced)**, as shown in **Figure 109**.

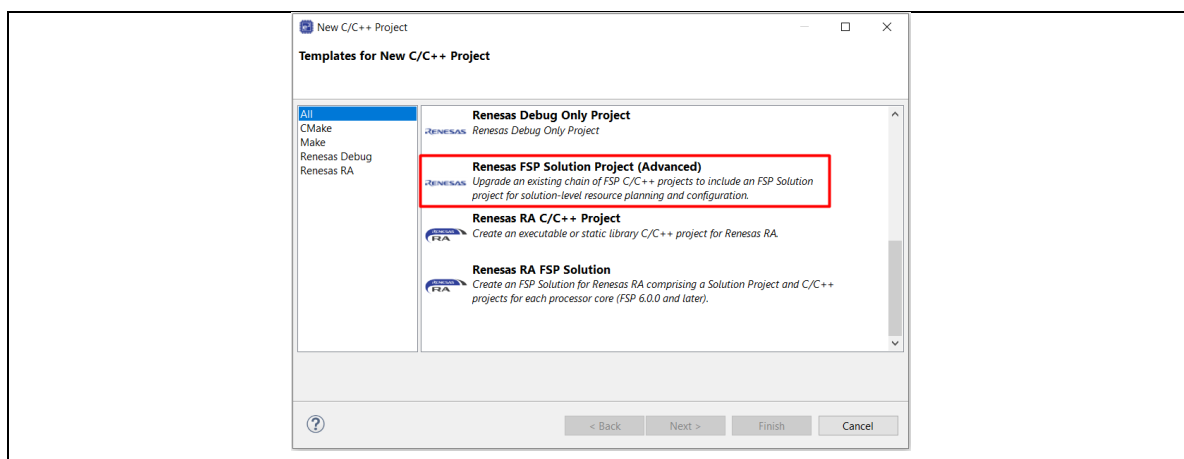


Figure 109. Renesas FSP Solution Project (Advanced)

2. Assign the project name based on Table 6 and click Next.

Table 6 Renesas Solution Project names

Bootloader project name	Primary application project name	Renesas Solution Project name
ra8m2_bootloader	ra8m2_primary_app_cpu0 ra8m2_primary_app_cpu1	ra8m2_solution
ra8m1_bootloader	ra8m1_primary_app	ra8m1_solution
ra6m5_bootloader	ra6m5_primary_app	ra6m5_solution
ra6m3_bootloader	ra6m3_primary_app	ra6m3_solution
ra4c1_bootloader	ra4c1_primary_app	ra4c1_solution
ra2a2_bootloader	ra2a2_primary_app	ra2a2_solution

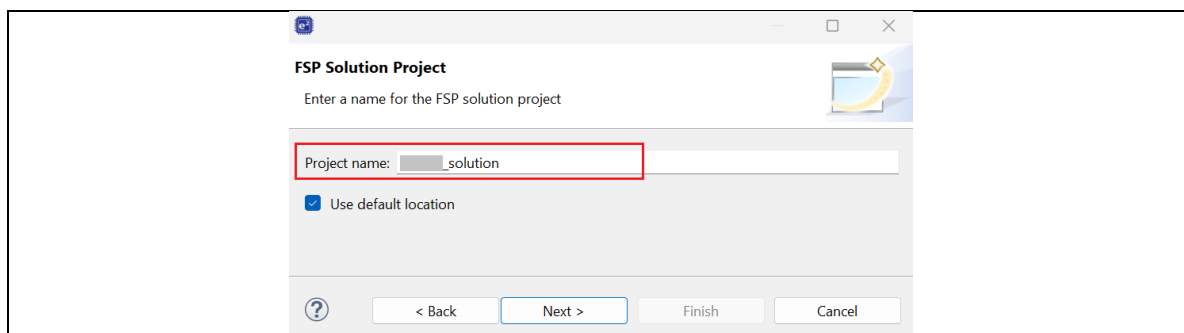


Figure 110. Set Solution Project name

3. Select the final project in a chain of projects from drop-down menu.
For example, with RA8M2, in the **Project** option, select the **ra8m2_primary_app_cpu1**, which is created in section [5.1](#). Click **Finish**.

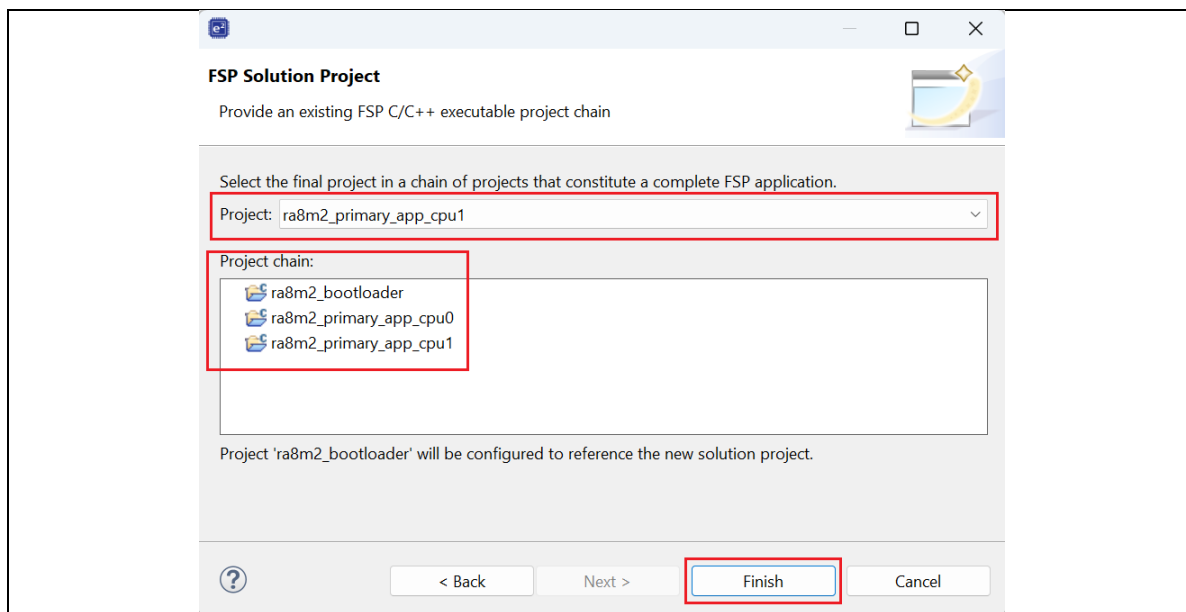


Figure 111. Project chain in RA8M2 FSP solution project

With others, in the **Project** option, select the **xxxx_primary_app**, which is created in section [5.1](#). Click **Finish**.

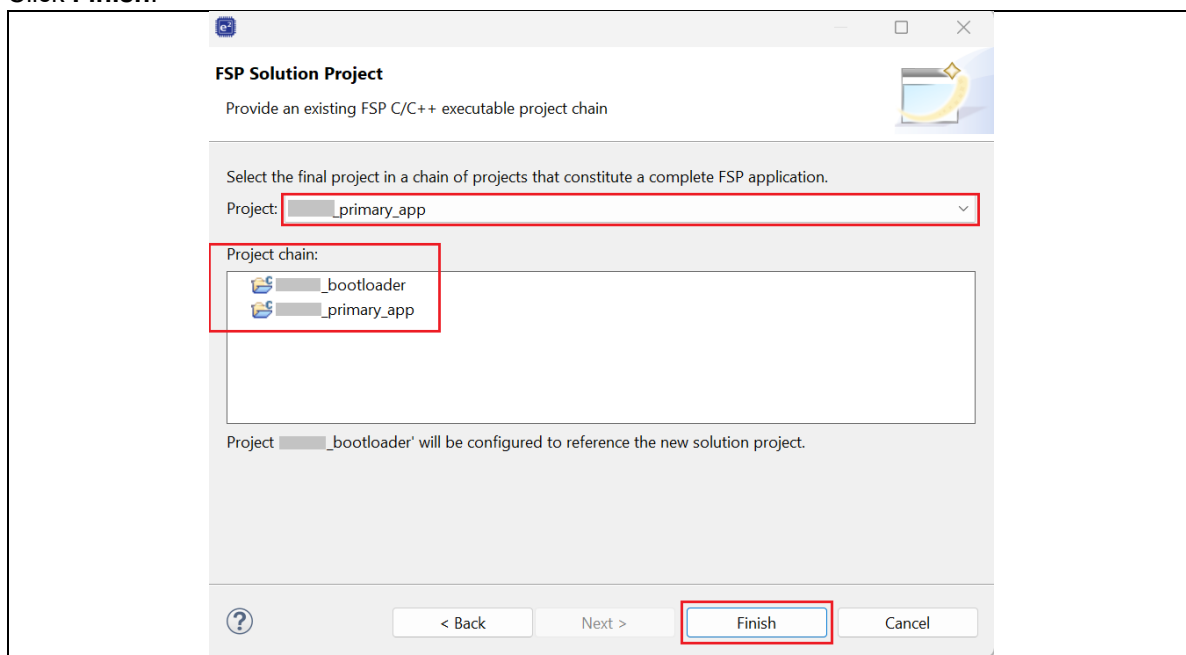


Figure 112. Select the application project in chain of projects (single-core)

5.3.2 Memory Configurations

In the Solution Project, double-click **solution.xml** to open the configuration. The **Memories** tab provides the capability to manage Flash partitions, including adding, removing, and sorting memory regions. Following figure is Memories tab in RA8M2.

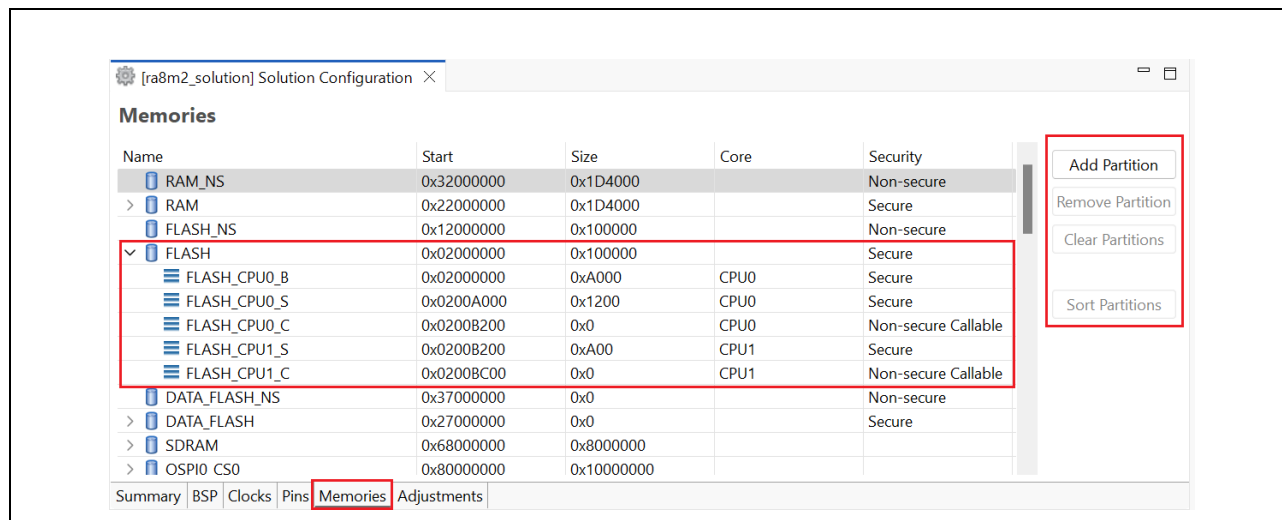


Figure 113. Default Memory Partitions in Solution Project on RA8M2 (Dual core)

To add a Flash partition, select **FLASH** and click **Add Partition** button. In the **New Partition** window, specify the required fields, including **Name**, **Start** address, **Size**, **Core**, and **Security** level. For naming conventions, refer to the [RA Flexible Software Package Documentation: MCUboot Port \(rm_mcuboot_port\)](#) or use the reference definitions provided in this application note. Other configurations should be adjusted based on the target MCU and specific use case.

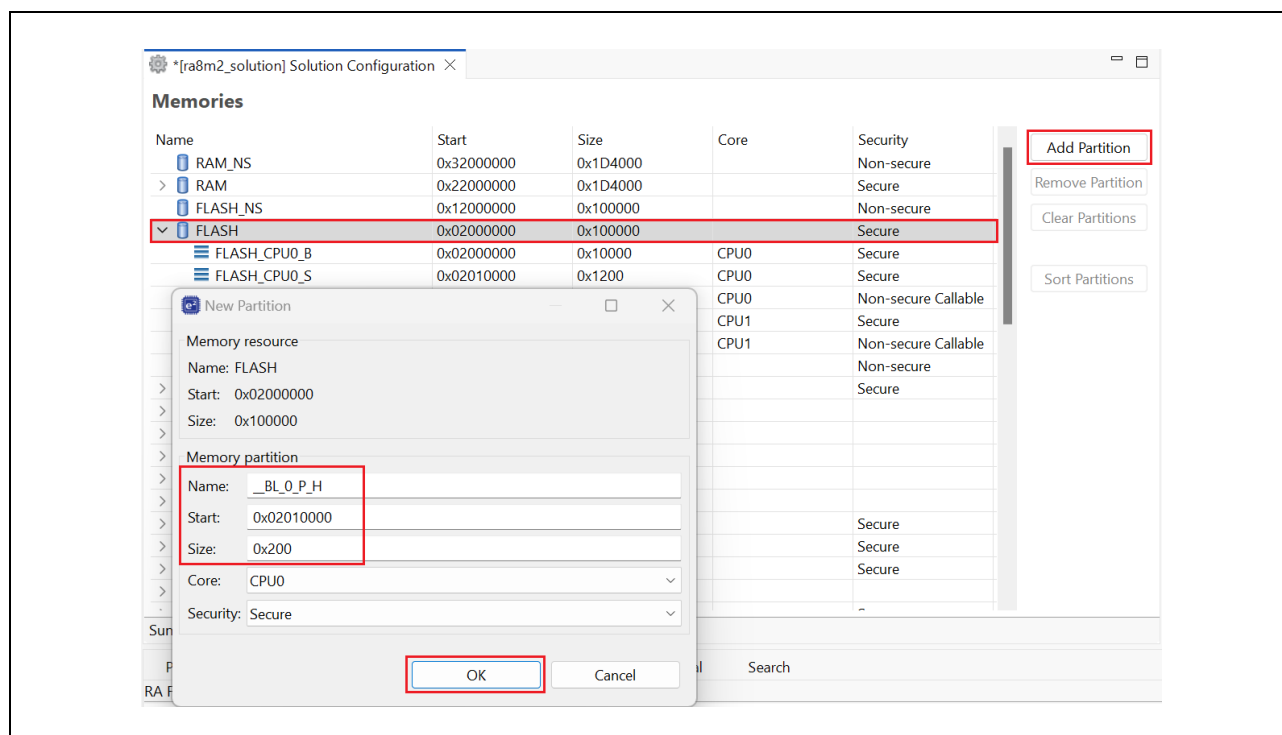


Figure 114. Add additional Flash partition

To remove a Flash partition, select the desired partition (for example, **__BL_0_P_H**) and click the **Remove Partition** button. A **Confirmation** dialog appears, click **OK** to confirm the removal.

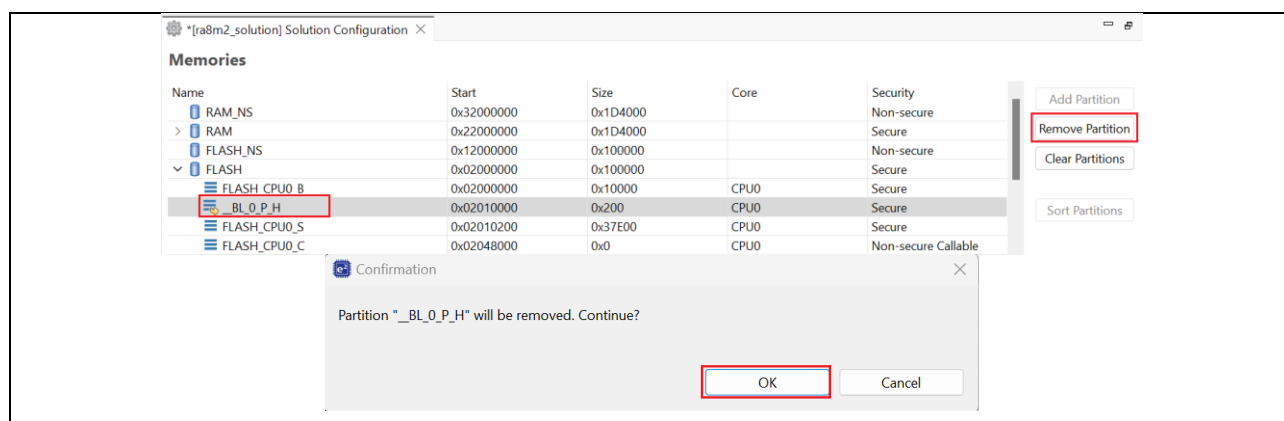


Figure 115. Remove Flash partition

During configuration, if a conflict occurs, select **Flash** and click **Sort Partitions** to resolve the issue by automatically reordering the partitions.

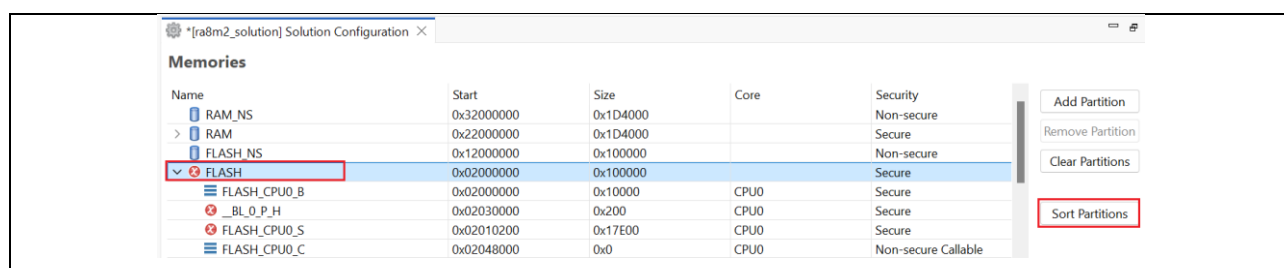


Figure 116. Using Sort Partitions to Resolve Configuration Errors

Table 7 and Table 8 define the required labels for creating memory partitions in dual-core and single-core projects.

Table 7 Memory Partition Labels for the MCUboot Flash Map (Single-Core)

Memory Partition Name	Flash Map	Definition of Memory Partition
__BL_S	Scratch Area	Scratch area (for swap mode only)
__BL_0_S_T	Trailer	Image 0 Secondary Trailer
__BL_0_S_I	TLV	Image 0 Secondary Image
	secondary_app.bin	
__BL_0_S_H	Header	Image 0 Secondary Header
__BL_0_P_T	Trailer	Image 0 Primary Trailer
FLASH_CM85 or FLASH_CM33 or FLASH_CM4 or FLASH_CM23	TLV	Image 0 Primary Image
	primary_app.bin	
__BL_0_P_H	Header	Image 0 Primary Header
FLASH_CM85_B or FLASH_CM33_B or FLASH_CM4_B or FLASH_CM23_B	MCUboot	Bootloader area

Table 8 Memory Partition Labels for the MCUboot Flash Map (Dual-Core)

Memory Partition Name	Flash Map	Definition of Memory Partition
__BL_S	Scratch Area	Scratch area
__BL_1_S_T	Trailer	Image 1 Secondary Trailer
__BL_1_S_I	TLV	Image 1 Secondary Image
	secondary_app_cpu1.bin	
__BL_1_S_H	Header	Image 1 Secondary Header
__BL_0_S_T	Trailer	Image 0 Secondary Trailer
__BL_0_S_I	TLV	Image 0 Secondary Image
	secondary_app_cpu0.bin	
__BL_0_S_H	Header	Image 0 Secondary Header
__BL_1_P_T	Trailer	Image 1 Primary Trailer
FLASH_CPU1_S	TLV	Image 1 Primary Image
	primary_app_cpu1.bin	
__BL_1_P_H	Header	Image 1 Primary Header
__BL_0_P_T	Trailer	Image 0 Primary Trailer
FLASH_CPU0_S	TLV	Image 0 Primary Image
	primary_app_cpu0.bin	
__BL_0_P_H	Header	Image 0 Primary Header
FLASH_CPU0_B	MCUboot	Bootloader area

Sequentially copy the name from Table 8 Memory Partition Labels for the MCUboot Flash Map (Dual-Core) and Table 7 Memory Partition Labels for the MCUboot Flash Map (Single-Core) into the Memory Partition of the **Solution Project** and then use **Ctrl + S** to save the settings.

Note:

- It is recommended to adjust the **Start** address and **Size** for **FLASH_xxxx** (application image) before adding other memory partitions to avoid conflicts.
- If a configuration conflict occurs, use **Sort Partitions** to resolve the issue.
- For dual-core devices, ensure the correct CPU is selected. For example, partitions with the prefix **__BL_1_xxx** should be assigned to CPU1.

5.3.2.1 Memory configurations on RA8M2

Name	Start	Size	Core	Security
FLASH	0x02000000	0x100000		Secure
FLASH_CPU0_B	0x02000000	0x18000	CPU0	Secure
__BL_0_P_H	0x02018000	0x200	CPU0	Secure
FLASH_CPU0_S	0x02018200	0x2FD00	CPU0	Secure
__BL_0_P_T	0x02047F00	0x100	CPU0	Secure
__BL_1_P_H	0x02048000	0x200	CPU1	Secure
FLASH_CPU1_S	0x02048200	0x2FD00	CPU1	Secure
__BL_1_P_T	0x02077F00	0x100	CPU0	Secure
FLASH_CPU0_C	0x02078000	0x0	CPU0	Non-secure Callable
__BL_0_S_H	0x02078000	0x200	CPU0	Secure
__BL_0_S_I	0x02078200	0x2FD00	CPU0	Secure
__BL_0_S_T	0x020A7F00	0x100	CPU0	Secure
__BL_1_S_H	0x020A8000	0x200	CPU1	Secure
__BL_1_S_I	0x020A8200	0x2FD00	CPU1	Secure
__BL_1_S_T	0x020D7F00	0x100	CPU0	Secure
__BL_S	0x020D8000	0x18000	CPU0	Secure
FLASH_CPU1_C	0x02100000	0x0	CPU1	Non-secure Callable
DATA_FLASH_NS	0x37000000	0x0		Non-secure

Figure 117. The MCUboot Memory Partition in RA8M2 project

Table 9. Reference memory configuration on RA8M2

RA8M2 – Memory partition name	Reference flash layout on MRAM 1MB product	
	Start	Size
FLASH_CPU0_B	0x02000000	0x18000
__BL_0_P_H	0x02018000	0x200
FLASH_CPU0_S	0x02018200	0x2FE00
__BL_0_P_T	0x02047F00	0x100
__BL_1_P_H	0x02048000	0x200
FLASH_CPU1_S	0x02048200	0x2FE00
__BL_1_P_T	0x02077F00	0x100
FLASH_CPU0_C	0x02078000	0x0
__BL_0_S_H	0x02078000	0x200
__BL_0_S_I	0x02078200	0x2FD00
__BL_0_S_T	0x020A7F00	0x100
__BL_1_S_H	0x020A8000	0x200
__BL_1_S_I	0x020A8200	0x2FD00
__BL_1_S_T	0x020D7F00	0x100
__BL_S	0x020D8000	0x18000

Bootloader size: 0x18000 (3 x 32 KB), as RA MCUboot port treat the internal block size is 32 KB (0x8000).

Slot area (primary & secondary): (flash size – bootloader – scratch) / 2 and round down to a whole number of 32KB.

Scratch (__BL_S): 0x18000 (3 x 32 KB).

5.3.2.2 Memory configurations on RA8M1

Name	Start	Size	Core	Security
RAM_NS	0x32000000	0xE0000		Non-secure
> RAM	0x22000000	0xE0000		Secure
FLASH_NS	0x12000000	0x1F8000		Non-secure
FLASH	0x02000000	0x1F8000		Secure
FLASH_CM85_B	0x02000000	0x18000	CM85	Secure
__BL_0_P_H	0x02018000	0x200	CM85	Secure
FLASH_CM85_S	0x02018200	0xDFD00	CM85	Secure
__BL_0_P_T	0x020F7F00	0x100	CM85	Secure
__BL_0_S_H	0x020F8000	0x200	CM85	Secure
__BL_0_S_I	0x020F8200	0xDFD00	CM85	Secure
__BL_0_S_T	0x021D7F00	0x100	CM85	Secure
__BL_S	0x021D8000	0x18000	CM85	Secure
FLASH_CM85_C	0x021F8000	0x0	CM85	Non-secure Callable

Figure 118. The MCUboot Memory Partition in RA8M1 project for 2MB product

Table 10. Reference memory configuration on RA8M1 in Linear

RA8M1 – Memory partition name	Reference flash layout on Flash_HP	
	2MB product	
	Start	Size
FLASH_CM85_B	0x02000000	0x18000
__BL_0_P_H	0x02018000	0x200
FLASH_CM85_S	0x02018200	0xDFD00
__BL_0_P_T	0x020F7F00	0x100
__BL_0_S_H	0x020F8000	0x200
__BL_0_S_I	0x020F8200	0xDFD00
__BL_0_S_T	0x021D7F00	0x100
__BL_S	0x021D8000	0x18000

Bootloader size: $0x18000 = \text{region0} (8 \times 8 \text{ KB block}) + \text{region1} (1 \times 32 \text{ KB block})$.

Slot area (primary & secondary): $(\text{flash size} - \text{bootloader} - \text{scratch}) / 2$ and round down to a whole number of 32KB sector.

Scratch (__BL_S): $0x18000 (3 \times 32 \text{ KB sector})$. Choose 3 sectors to improve copy cycles

5.3.2.3 Memory configurations on RA6M5

Name	Start	Size	Core	Security
RAM	0x20000000	0x80000		
FLASH	0x00000000	0x200000		
FLASH_CM33_B	0x00000000	0x18000	CM33	Secure
__BL_0_P_H	0x00018000	0x200	CM33	Secure
FLASH_CM33_S	0x00018200	0xEFD00	CM33	Secure
__BL_0_P_T	0x00107F00	0x100	CM33	Secure
__BL_0_S_H	0x00108000	0x200	CM33	Secure
__BL_0_S_I	0x00108200	0xEFD00	CM33	Secure
__BL_0_S_T	0x001F7F00	0x100	CM33	Secure
__BL_S	0x001F8000	0x8000	CM33	Secure
FLASH_CM33_C	0x00200000	0x0	CM33	Non-secure Callable

Figure 119. The MCUboot Memory Partition in RA6M5 project 2MB Flash product

Table 11. Reference memory configuration on RA6M5

RA6M5 – Memory partition name	Reference flash layout on Flash_HP	
	2MB product	
	Start	Size
FLASH_CM33_B	0x00000000	0x18000
__BL_0_P_H	0x00018000	0x200
FLASH_CM33_S	0x00018200	0xEFD00
__BL_0_P_T	0x00107F00	0x100
__BL_0_S_H	0x00108000	0x200
__BL_0_S_I	0x00108200	0xEFD00
__BL_0_S_T	0x001F7F00	0x100
__BL_S	0x001F8000	0x8000

Bootloader size: 0x18000 = region0 (8 x 8 KB block) + region1(1×32 KB block)

Slot area (primary & secondary): (flash size – bootloader – scratch) / 2 and round down to a whole number of 32 KB sector.

Scratch (__BL_S): 1 x 32 KB block (one sector-aligned).

5.3.2.4 Memory configurations on RA6M3

Name	Start	Size	Core
RAM	0x1FFE0000	0xA0000	
RAM_CM4_B	0x1FFE0000	0x10000	CM4
RAM_CM4	0x1FFE0000	0xA0000	CM4
FLASH	0x00000000	0x200000	
FLASH_CM4_B	0x00000000	0x18000	CM4
__BL_0_P_H	0x00018000	0x200	CM4
FLASH_CM4	0x00018200	0xEFD00	CM4
__BL_0_P_T	0x00107F00	0x100	CM4
__BL_0_S_H	0x00108000	0x200	CM4
__BL_0_S_I	0x00108200	0xEFD00	CM4
__BL_0_S_T	0x001F7F00	0x100	CM4
__BL_S	0x001F8000	0x8000	CM4
DATA_FLASH	0x40100000	0x10000	
QSPI_FLASH	0x60000000	0x4000000	
SDRAM	0x90000000	0x8000000	

Figure 120. The MCUboot Memory Partition in RA6M3 project 2MB Flash product

Table 12. Reference memory configuration on RA6M3

RA6M3 – Memory partition name	Reference flash layout on Flash_HP	
	2MB product	
	Start	Size
FLASH_CM4_B	0x00000000	0x18000
__BL_0_P_H	0x00018000	0x200
FLASH_CM4	0x00018200	0xEFD00
__BL_0_P_T	0x00107F00	0x100
__BL_0_S_H	0x00108000	0x200
__BL_0_S_I	0x00108200	0xEFD00
__BL_0_S_T	0x001F7F00	0x100
__BL_S	0x001F8000	0x8000

Bootloader size: $0x18000 = \text{region0} (8 \times 8 \text{ KB block}) + \text{region1} (1 \times 32 \text{ KB block})$

Slot area (primary & secondary): $(\text{flash size} - \text{bootloader} - \text{scratch}) / 2$ and round down to a whole number of 32 KB sector.

Scratch (__BL_S): 1 x 32 KB block (one sector-aligned).

5.3.2.5 Memory configurations on RA4C1

Name	Start	Size	Core	Security
RAM	0x20000000	0x18000		
FLASH	0x00000000	0x80000		
FLASH_CM33_B	0x00000000	0x10000	CM33	Secure
__BL_0_P_H	0x00010000	0x200	CM33	Secure
FLASH_CM33_S	0x00010200	0x35D00	CM33	Secure
__BL_0_P_T	0x00045F00	0x100	CM33	Secure
__BL_0_S_H	0x00046000	0x200	CM33	Secure
__BL_0_S_I	0x00046200	0x35D00	CM33	Secure
__BL_0_S_T	0x0007BF00	0x100	CM33	Secure
__BL_S	0x0007C000	0x4000	CM33	Secure
FLASH_CM33_C	0x00080000	0x0	CM33	Non-secure Callable
DATA_FLASH	0x08000000	0x2000		
QSPI_FLASH	0x60000000	0x8000000		

Figure 121. The MCUboot Memory Partition in RA4C1 project 512KB flash product

Table 13. Reference memory configuration on RA4C1

RA4C1 – Memory partition name	Reference flash layout on Flash_HP	
	512KB product	
	Start	Size
FLASH_CM33_B	0x00000000	0x10000
__BL_0_P_H	0x00010000	0x200
FLASH_CM33_S	0x00010200	0x35D00
__BL_0_P_T	0x00045F00	0x100
__BL_0_S_H	0x00046000	0x200
__BL_0_S_I	0x00046200	0x35D00
__BL_0_S_T	0x0007BF00	0x100
__BL_S	0x0007C000	0x4000

Bootloader size: 0x1000 (64KB)

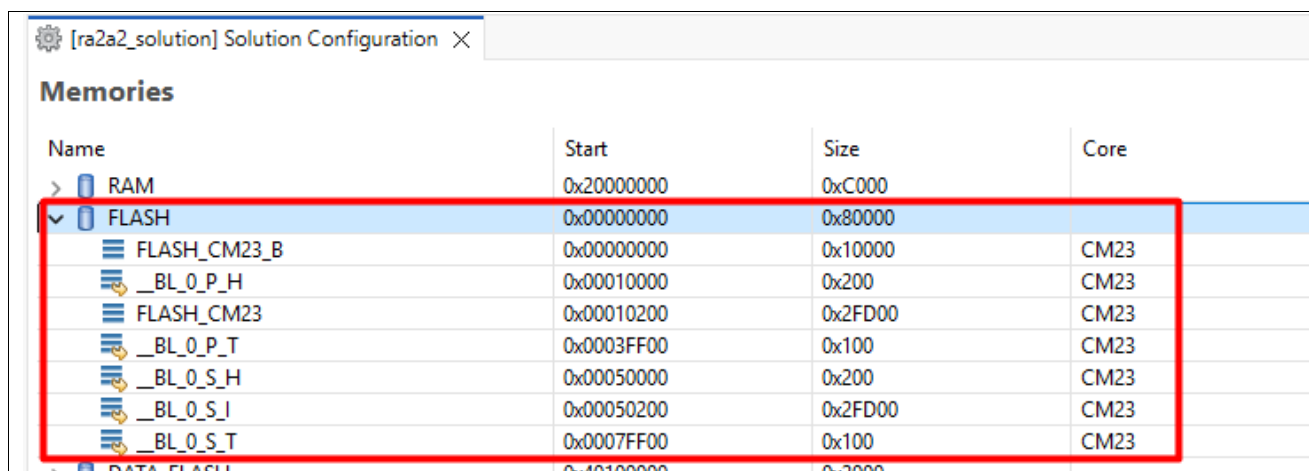
Slot area (primary & secondary): (flash size – bootloader – scratch) / 2 and round down to a whole number of 2KB sector.

Scratch (__BL_S): 0x4000 (8 x 2KB sector). Choose 8 sectors to improve copy .

5.3.2.6 Memory configurations on RA2A2

RA2A2 will enable Dual bank mode and using upgrade mode is Direct XIP.

(*) Dual bank mode only supports Direct XIP.



Name	Start	Size	Core
> RAM	0x20000000	0xC000	
✓ FLASH	0x00000000	0x80000	
FLASH_CM23_B	0x00000000	0x10000	CM23
__BL_0_P_H	0x00010000	0x200	CM23
FLASH_CM23	0x00010200	0x2FD00	CM23
__BL_0_P_T	0x0003FF00	0x100	CM23
__BL_0_S_H	0x00050000	0x200	CM23
__BL_0_S_I	0x00050200	0x2FD00	CM23
__BL_0_S_T	0x0007FF00	0x100	CM23

Figure 122. The MCUboot Memory Partition in RA2A2 project

Table 14. Reference memory configuration on RA2A2

RA2A2 – Memory partition name	Reference flash layout on Flash_LP	
	512 KB product	
	Start	Size
FLASH_CM23_B	0x00000000	0x10000
__BL_0_P_H	0x00010000	0x200
FLASH_CM23	0x00010200	0x2FD00
__BL_0_P_T	0x0003FF00	0x100
__BL_0_S_H	0x00050000	0x200
__BL_0_S_I	0x00050200	0x2FD00
__BL_0_S_T	0x0007FF00	0x100

Note: With Flash LP in Dual Bank mode, bank 0 will start from 0x0 to 0x3FFFF, bank 1 will start from 0x40000 - 0x7FFFF, the flash layout is shown as following figure:

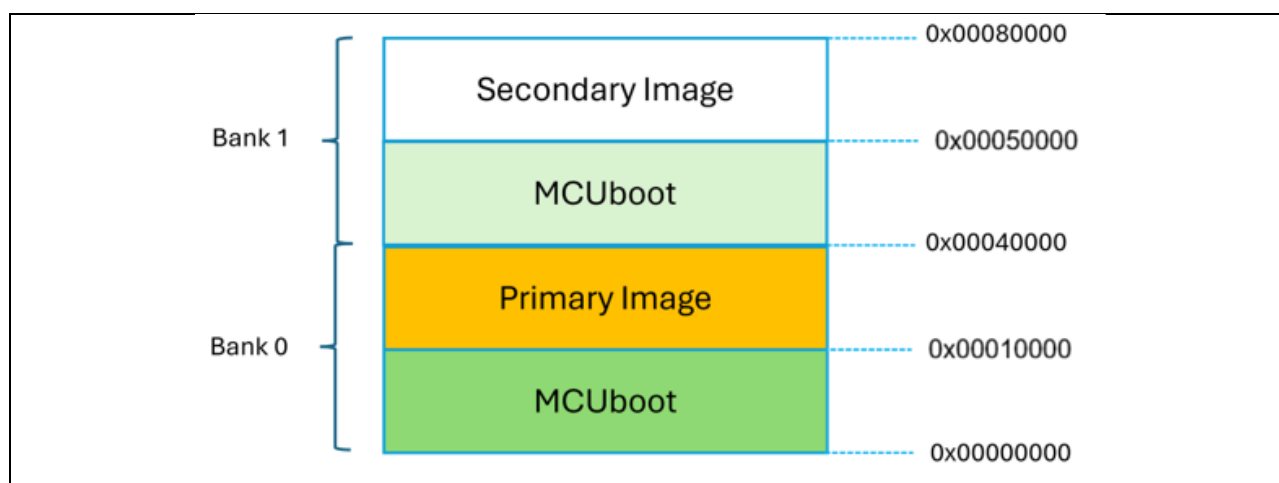


Figure 123. Flash LP layout in Dual bank mode

The start address of Secondary image will start from start address of bank 1 plus size of bootloader size. In this case, $__BL_0_S_H = 0x40000 + 10000 = 0x50000$.

5.3.3 Build the Solution Project

Right-click on the **Solution Project** and select **Build Project**.

This command will build all projects within the **Solution Project**.

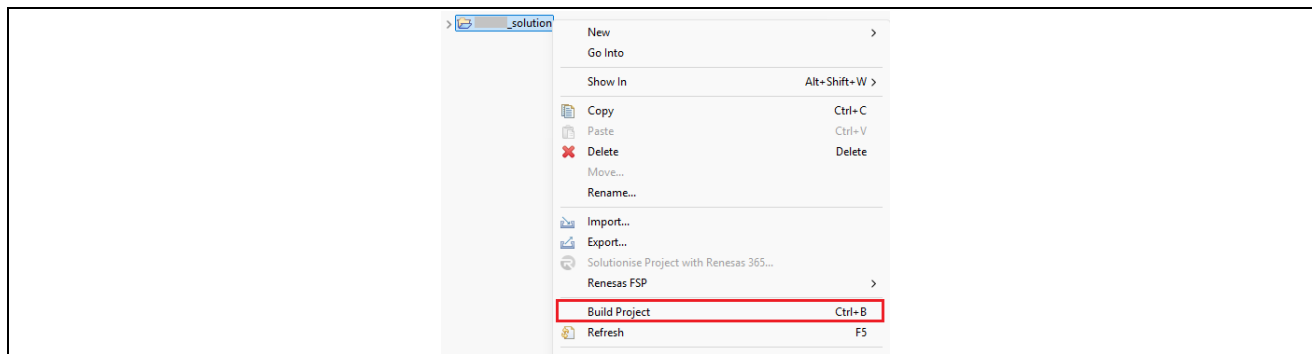


Figure 124. Build all projects within the Solution Project

This will generate the files as following table:

Table 15. Signed primary application images:

Primary application project name	Location and name of signed primary application image
ra8m2_primary_app_cpu0	\\Debug\\ra8m2_primary_app_cpu0.bin.signed
ra8m2_primary_app_cpu1	\\Debug\\ra8m2_primary_app_cpu1.bin.signed
ra8m1_primary_app	\\Debug\\ra8m1_primary_app.bin.signed
ra6m5_primary_app	\\Debug\\ra6m5_primary_app.bin.signed
ra6m3_primary_app	\\Debug\\ra6m3_primary_app.bin.signed
ra4c1_primary_app	\\Debug\\ra4c1_primary_app.bin.signed
ra2a2_primary_app	\\Debug\\ra2a2_primary_app.bin.signed

Note: If an error related to RAM region overflow appears in the **Console** tab, the RAM region in the **Memory Partition** settings of the **Solution Project** will have to be adjusted.

- For the Bootloader Project, adjust the **RAM_<CPU>_B** size (corresponding with MCU, the CPU is CPU0/CPU1, CM33, CM4 or CM23).
 - Recommended RAM size for **Overwrite and Direct XIP modes** are 0x5600 (~22KB)
 - Recommended RAM size for **Swap mode** is 0x6000 (24KB)
- For the Application Project, adjust the **RAM_<CPU>** size (corresponding with MCU, the CPU is CPU0/CPU1, CM33, CM4 or CM23).

5.3.4 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 a 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.

Example of maximize the MCU SRAM by the following steps:

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

Name	Start	Size	Core
RAM	0x22000000	0x1D4000	
RAM_CPU0_B	0x22000000	0x10000	CPU0
RAM_CPU0_S	0x22000000	0xEA000	CPU0
RAM_CPU0_C	0x220EA000	0x0	CPU0
RAM_CPU1_S	0x220EA000	0xEA000	CPU1
RAM_CPU1_C	0x221D4000	0x0	CPU1
FLASH_NS	0x12000000	0x100000	

Figure 125. Adjust the maximum SRAM for the application in RA8M2

Name	Start	Size	Core	Security
RAM	0x22000000	0xE0000		Secure
RAM_CM85_B	0x22000000	0x10000	CM85	Secure
RAM_CM85_S	0x22000000	0xD0000	CM85	Secure
RAM_CM85_C	0x220E0000	0x0	CM85	Non-secure Callable
FLASH_NS	0x12000000	0x1F8000		Non-secure
FLASH	0x02000000	0x1F8000		Secure

Figure 126. Adjust the maximum SRAM for the application in RA8M1

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

Name	Start	Size
RAM	0x22000000	0xE0000
RAM_CM85_B	0x22000000	0x10000
RAM_CM85_S	0x22000000	0xD0000
RAM_CM85_C	0x220E0000	0x0

Figure 127. Partition Overlapping Warning

3. Press Ctrl + S to save the configuration

5.4 Erase the MCU

To create a clean environment for starting the initial application project, it is recommended that the MCU be returned to a clean state. The following methods can be used:

1. **Initialize Device** using Renesas Device Partition Manager (RDPM) integrated into e2studio or the Renesas Flash Programmer (RFP).
2. **Erase the MCU** using Renesas Flash Programmer (RFP) or a third-party tool such as SEGGER J-Flash Lite.

Note: Depending on the MCU family, memory architecture, and device lifecycle state, the **Initialize Device** operation may be required instead of the **Erase** operation to completely clear the device.

5.4.1 Initialize Device using Renesas Device Partition Manager (RDPM)

The **Initialize Device** operation is commonly used with RA4, RA6, and RA8 devices that implement Device Lifecycle Management (DLM) or TrustZone security features. It erases device memory and restores the device to its default development state.

1. In e2studio, go to **Run -> Renesas Debug Tools -> Renesas Device Partition Manager**.
2. **Click Initialize device and start Run**

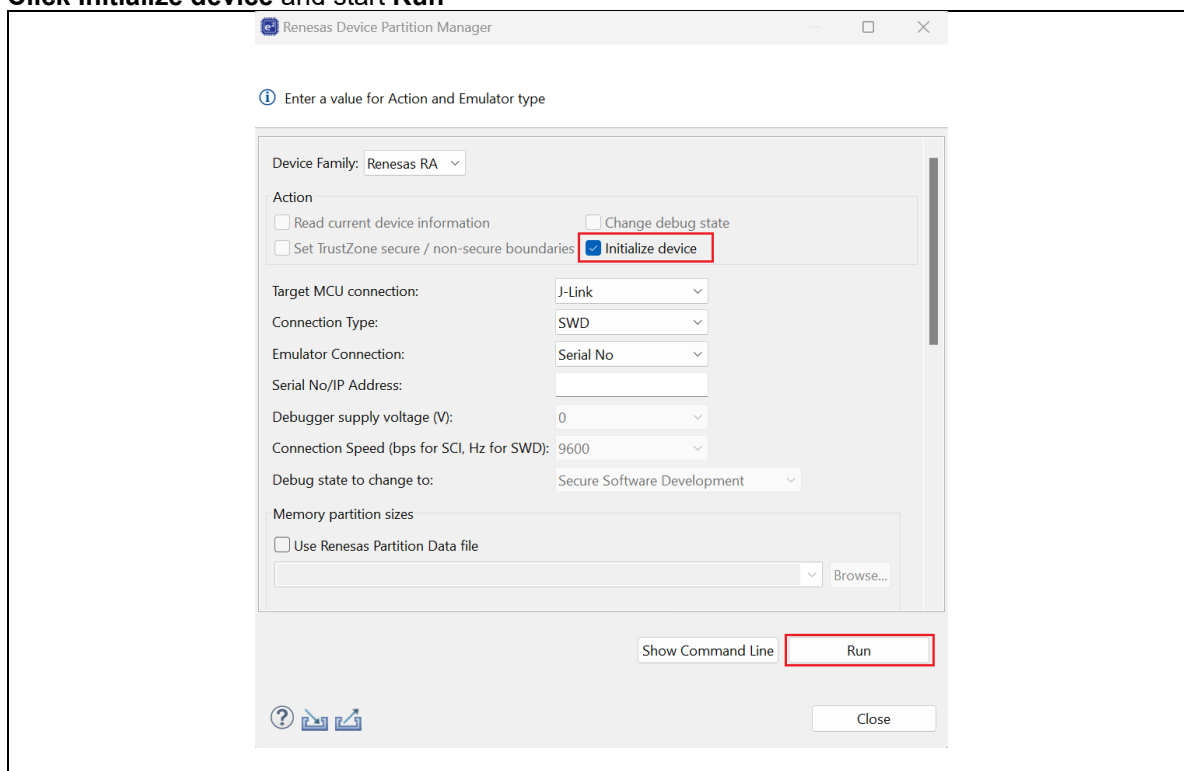


Figure 128. Initialize Device using connection type is SWD

Note: for the EK-RA8M1, SCI should be selected as the connection type.

5.4.2 Initialize Device or Erase the MCU using the Renesas Flash Programmer (RFP)

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

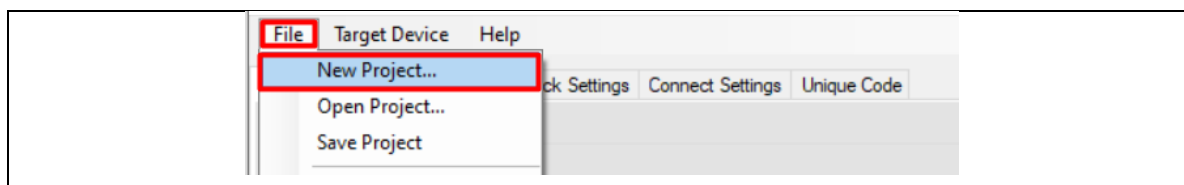


Figure 129. Create a New RFP Project

3. Select the target **Microcontroller** and Communication **Tool**. Then, click **Connect**.

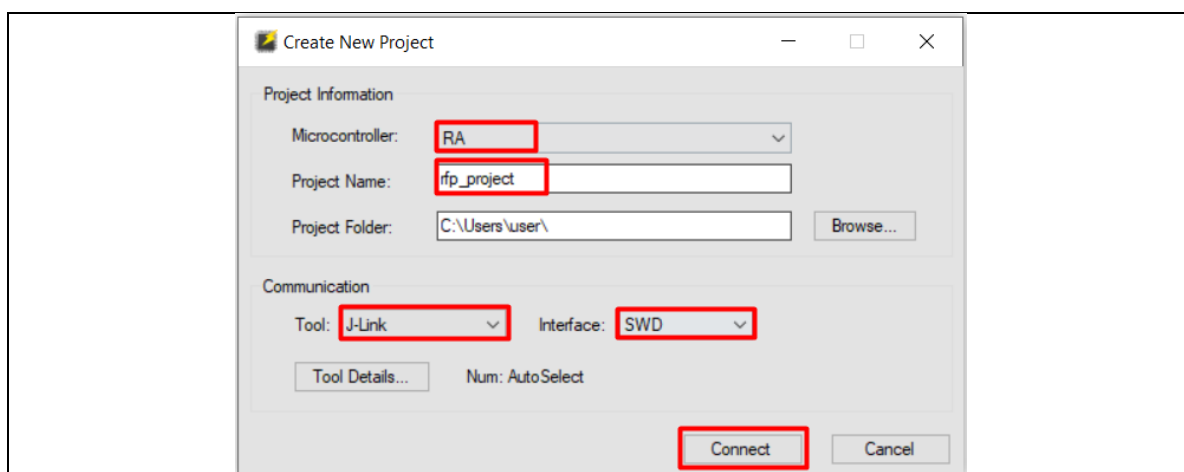


Figure 130. Configure a New Project

- On successful connection, open the **Block Settings** tab to check the Code Flash configurations.

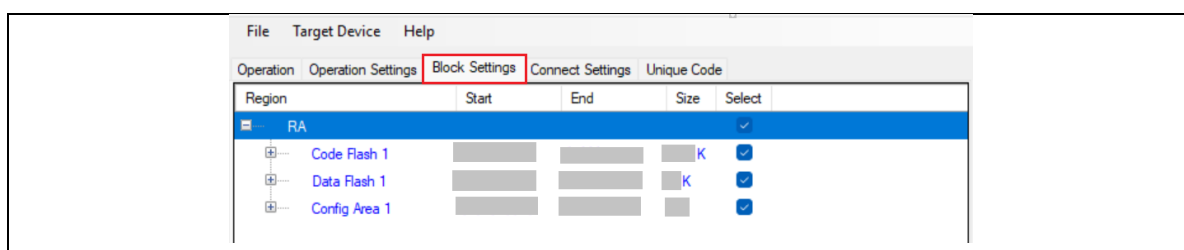


Figure 131. Example of Block Settings on successful connection

- To **Initialize device** (for supported MCUs), select **Target Device > Initialize Device**

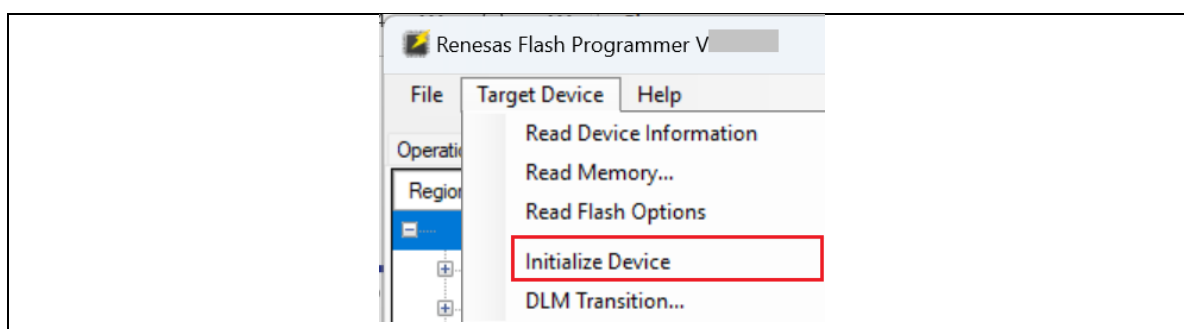


Figure 132. Initialize Device using RFP

If the **Initialize Device** operation completes successfully, the device is ready for use. **Proceed to the next section.**

- To **Erase the MCU**, open the **Operation Settings** tab and select the **Erase** command.

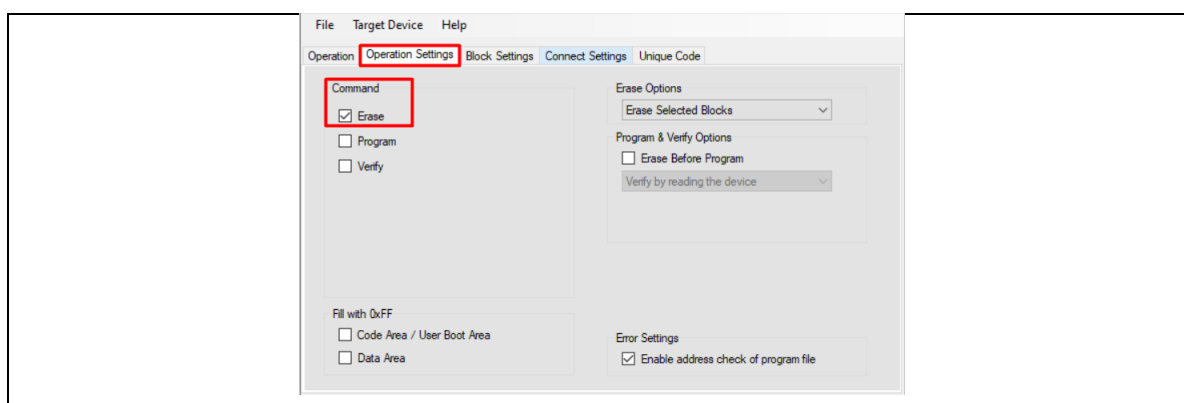


Figure 133. Choose the Erase Command

7. Choose the **Operation** tab and select **Start**.

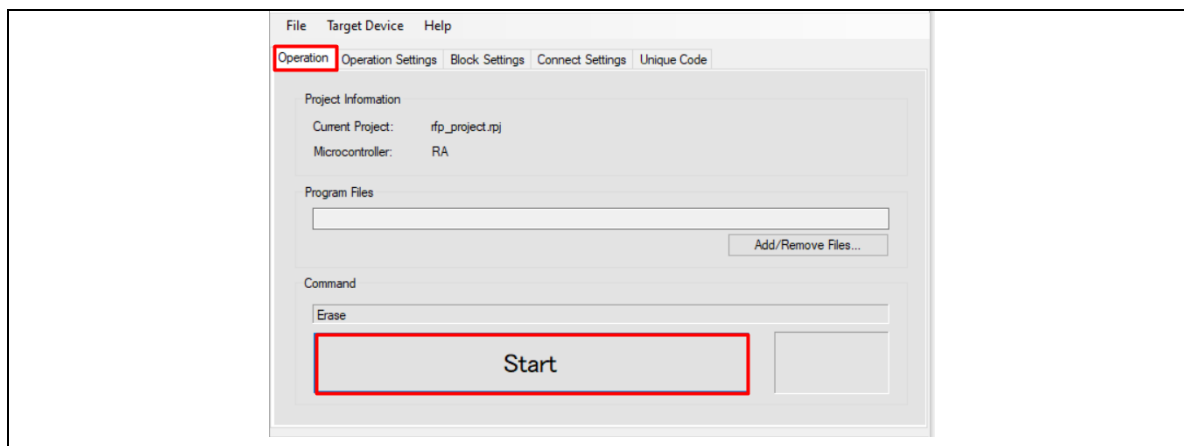


Figure 134. Start Erase Chip

8. When the chip is erased, **Operation completed** will be displayed.

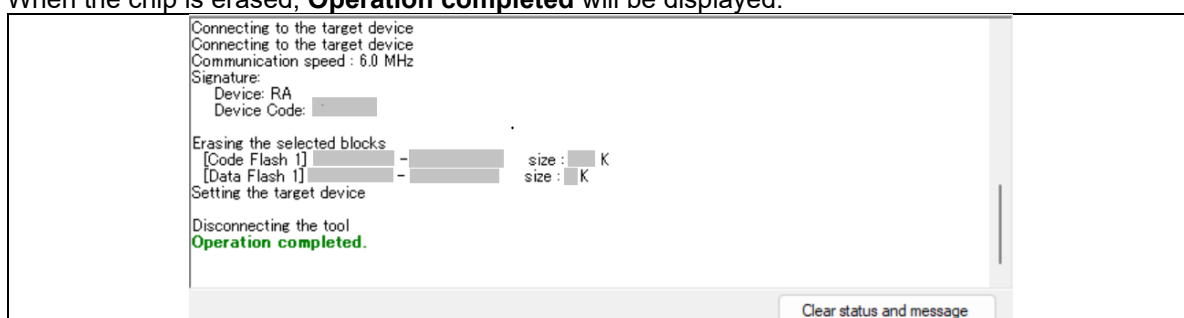


Figure 135. Succeeded Operation in RFP

5.4.3 Use the J-Flash Lite

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

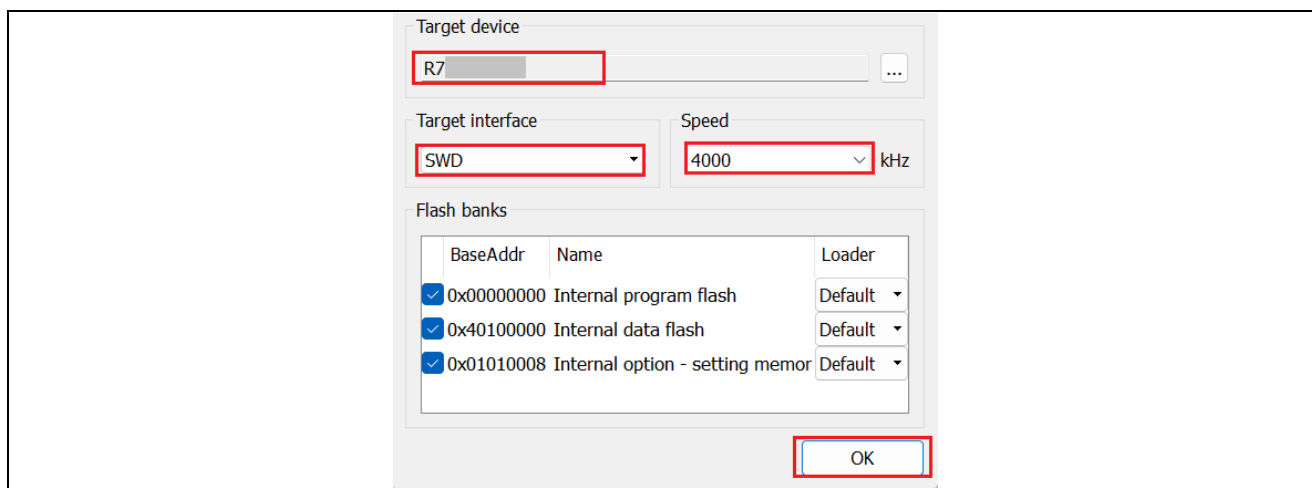
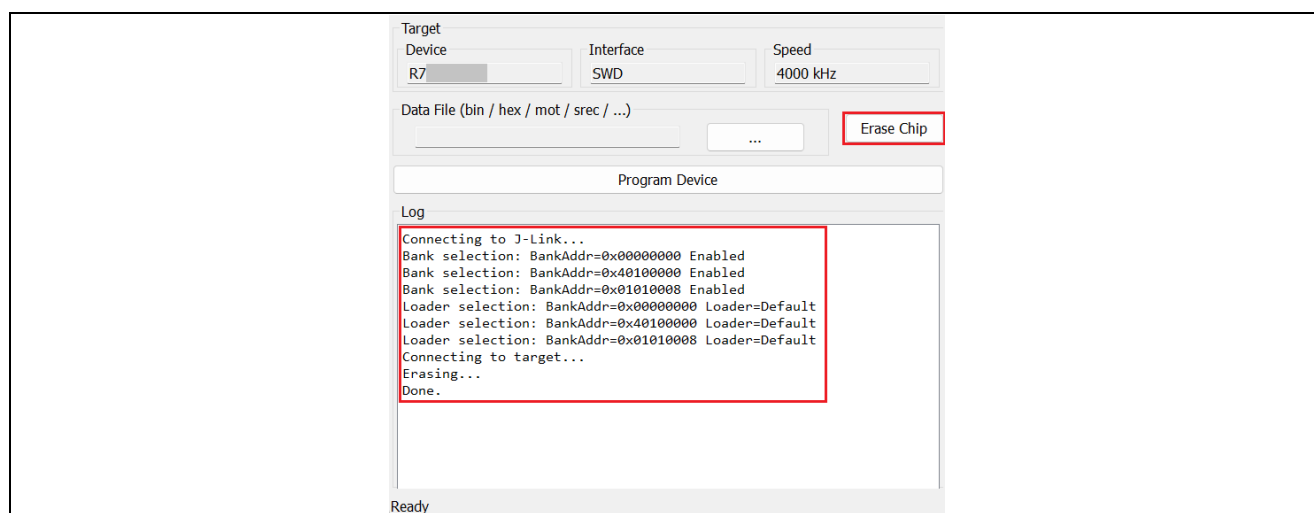


Figure 136. Launch J-Flash Lite

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

**Figure 137. Erase MCU Succeeded**

5.5 Configure the Debugger

5.5.1 Single core projects

Navigate to the **Debugger > Debug Tool Settings** and **Set Allow caching of flash contents to No**, as shown in below figure.

If set to yes, when debugging the bootloader applications, the memory window information may show the wrong information.

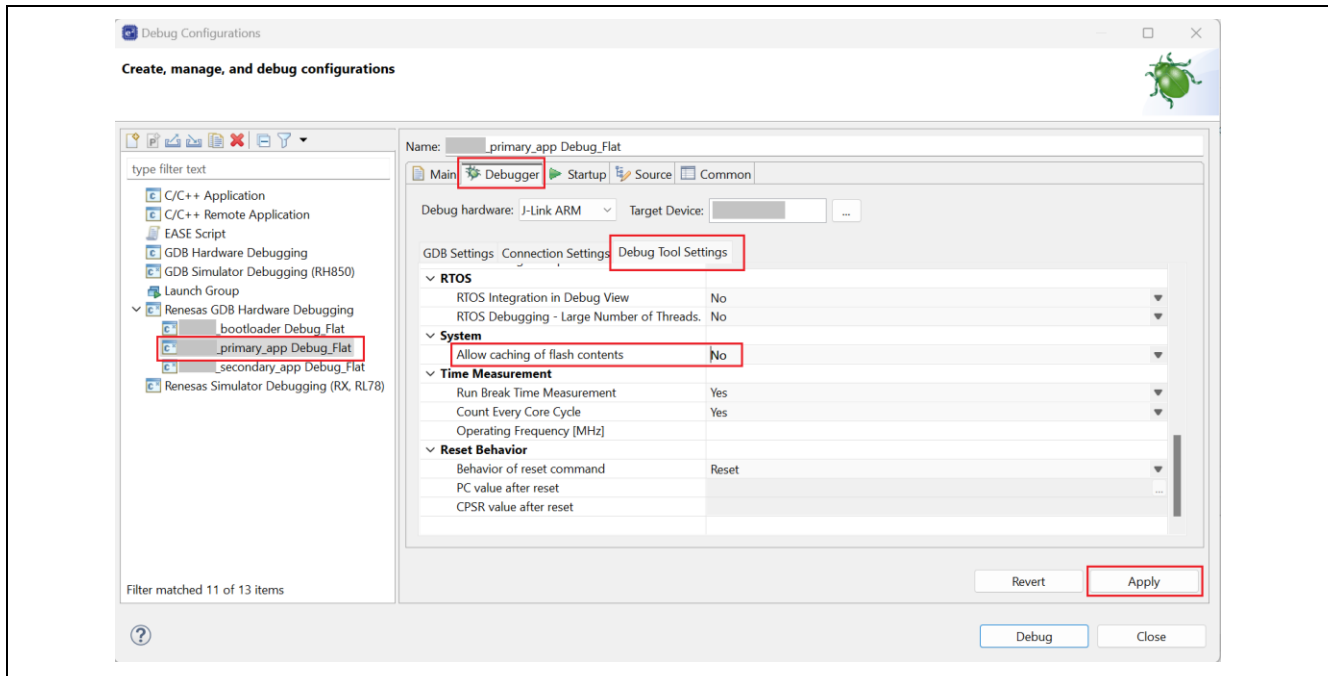


Figure 138. Disable Flash Content Caching

Move to **Startup** tab,

- Change **Load type** of `xxxx_primary_app.elf` to **Symbols Only**.
- Click **Add...** -> choose **Workspace...** then navigate to the `xxxx_bootloader/Debug/xxxx_bootloader.elf`. Ensure **Load type** is **Image and Symbols**, set **Offset (hex)** is **0**
- Continue add for `xxxx_primary_app/Debug/xxxx_primary_app.bin.signed`. Change **Load type** to **Raw Binary**, set **Offset (hex)** is value of `__BL_0_P_H` (for example, 2010000).

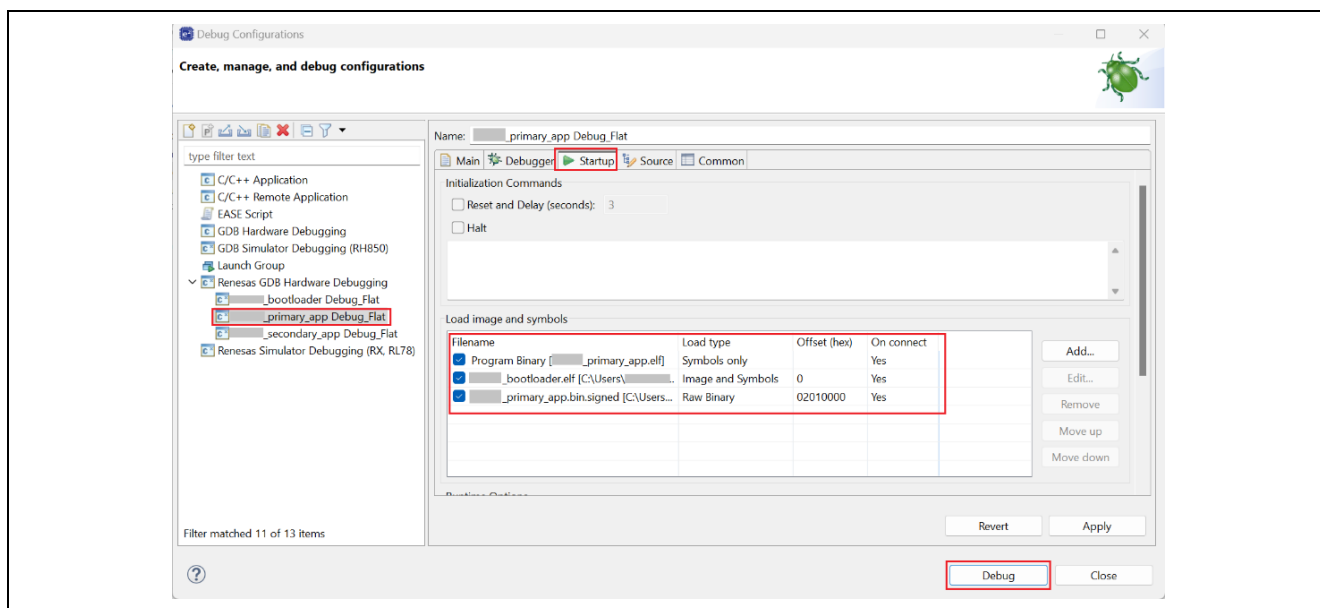


Figure 139. Configure the Primary Project Debug Startup

Table 16: Reference offsets for primary images

MCU	Reference offsets for primary images (__BL_0_P_H)
RA8M1	0x02010000
RA6M5	0x00018000
RA6M3	0x00018000
RA4C1	0x00010000
RA2A2	0x00010000

For **RA2A2** which uses **Dual Bank mode**, the bootloader image must be set 2 times. The first one has offset is 0 and Load type is Image and Symbols. The second one has offset is start address of bank 1 (0x40000) and Load type is Image only. For details about dual bank, please refer to application note R11AN1069 - RA2 Secure Firmware update using MCUboot and Flash Dual Bank.

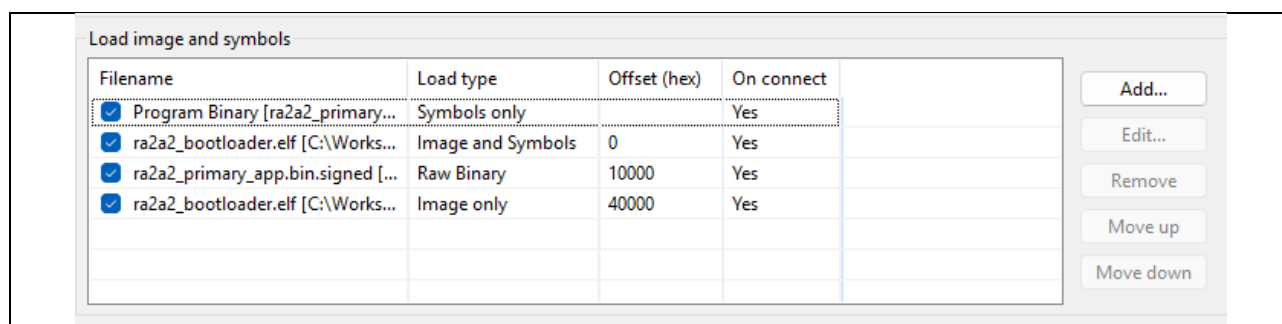


Figure 140. Configure the Primary Project Debug Startup for RA2A2 use Dual bank mode

For **RA8** series, it recommends going to **Debugger -> Connection Settings -> TrustZone**, changing **Set Trustzone secure/non-secure boundaries** to **No** to avoid impacts of TrustZone settings on MCUboot Flash configurations.

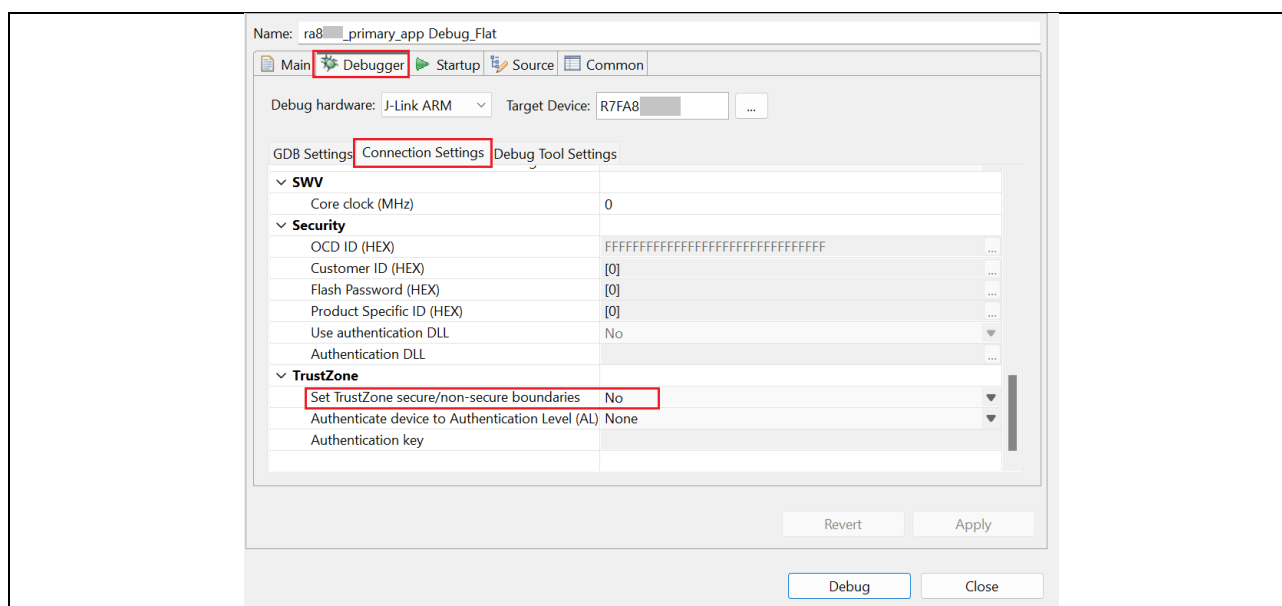


Figure 141. Set TrustZone boundaries for RA8 series

Click **Debug**. The debugger should hit the `Reset_Handler()` in the bootloader project.

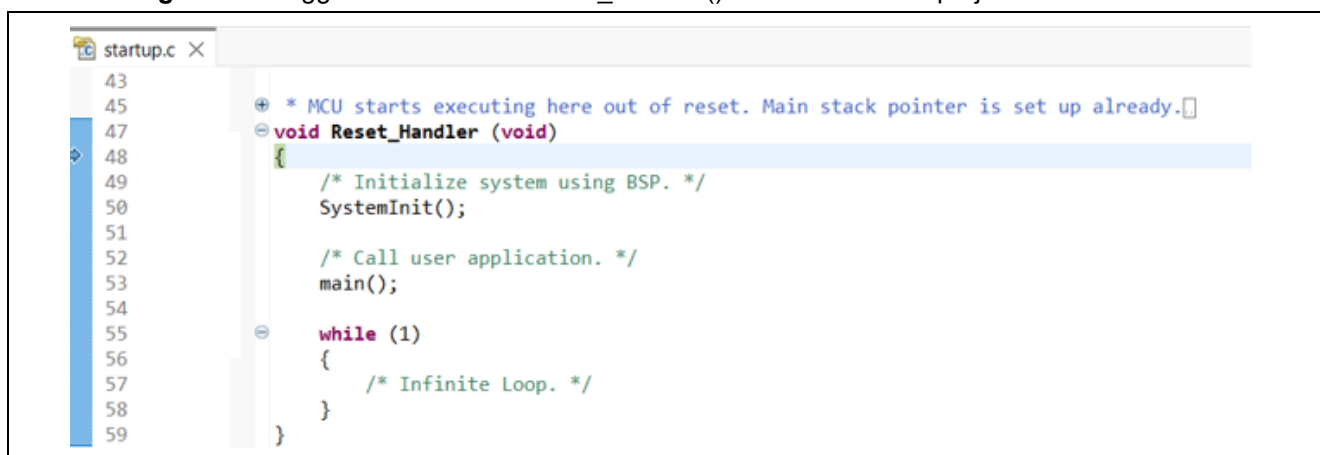


Figure 142. Start the Application Execution

Click **Resume**  twice to run the project. The bootloader and the primary application project will be programmed and then the primary application project will be booted.

Check LEDs status on the board.

5.5.2 Dual core RA8 MCU projects

Following are brief steps to configure the debugger in case dual core with MCUboot project. For detail, refer to R11AN1099 application note which introduce for using MCUboot with dual core RA8 MCU. This is example for RA8M2:

1. Right-click on the `ra8m2_primary_app_cpu1` project, and open **Debug As -> Debug Configurations...**
2. In Debug Configurations window, under **Renesas GDB Hardware Debugging**, select **ra8m2_primary_app_cpu1 Debug_Multicore**.

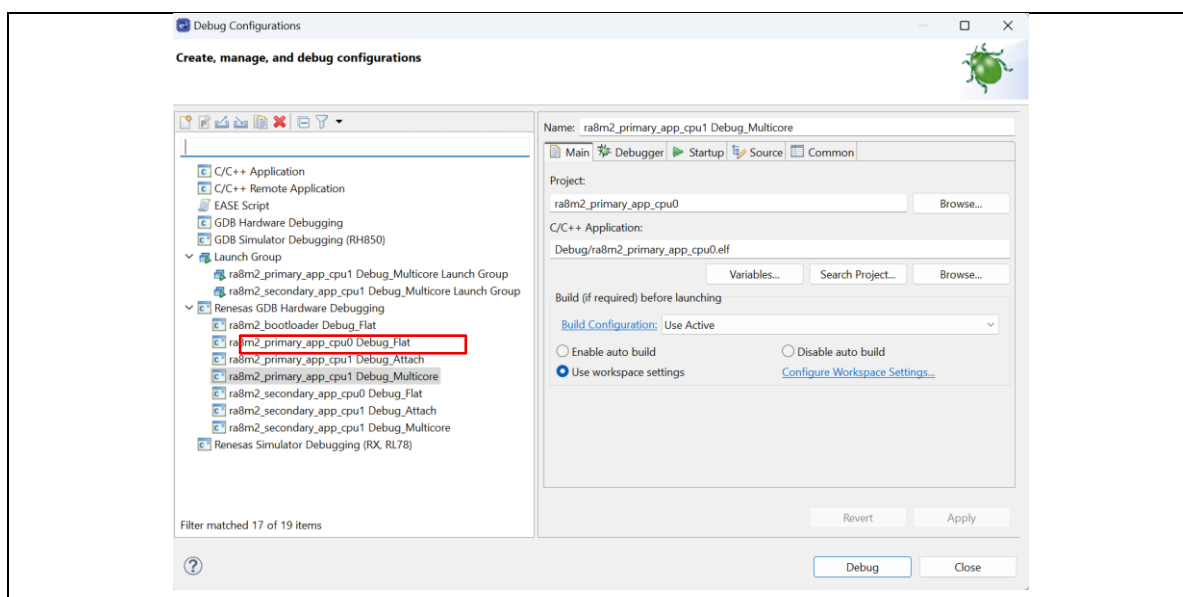


Figure 143. Select ra8m2_primary_app_cpu1 Debug_multicore

- On **Startup** tab (Main Debugger Startup Common Source), under “Load image and symbols” section, use **Add...->Workspace...** to add for signed images and change load types as following figure:

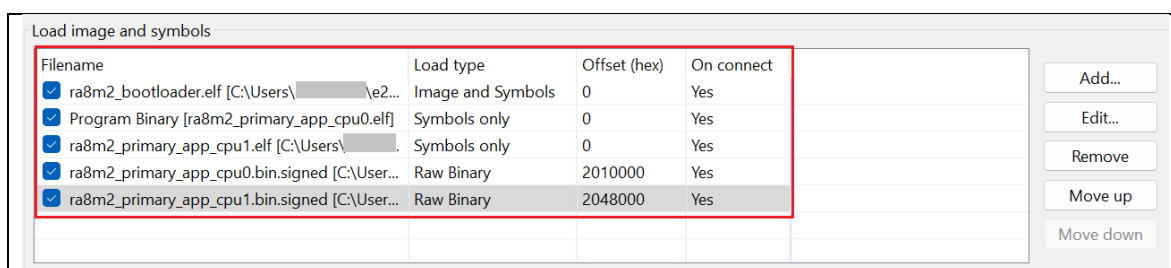


Figure 144. Load image and symbols for RA8M2

- Switch to Debugger tab, under Debug Tool Settings -> System, set Allow caching of flash contents to No and click Apply. If set to yes, when debugging the bootloader applications, the memory window information may show the wrong information.
- Under **Connection Settings -> TrustZone**, change **Set Trustzone secure/non-secure boundaries** to **No** to avoid impacts of TrustZone settings on MCUboot Flash configurations.
- Then, under “**Launch Group**”, select **ra8m2_primary_app_cpu1 Debug_Multicore Launch**, ensure inheritance as below picture then click **Debug**.

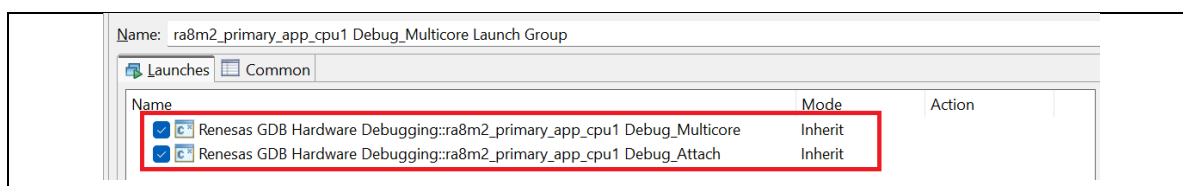


Figure 145. Information on Debug_Multicore Launch

- Under **Confirmation Perspective Switch** window, click **Switch** button.
- Click Resume (F8) two times to start primary application project on CPU0, continue with Resume (F8) to start primary application project on CPU1.

5.6 Mastering and Delivering a New Application

This section explains how to prepare the Secondary Application from the Primary Application and how to download the Secondary Application to the target board.

5.6.1 Prepare a Secondary Image

The secondary application can be created by repeat steps in [Section 5.1](#) or use import the initial project to the same workspace and rename the new project.

Following steps are guide for using import the import the initial project and rename the new project. It is applied for Single core project. As the complicated of Dual core project, recommend repeating steps in [Section 5.1](#).

Initial project to the same workspace and rename the new project:

1. Click **File > Import...** and select **General > Rename & Import Existing C/C++ Project into Workspace**.

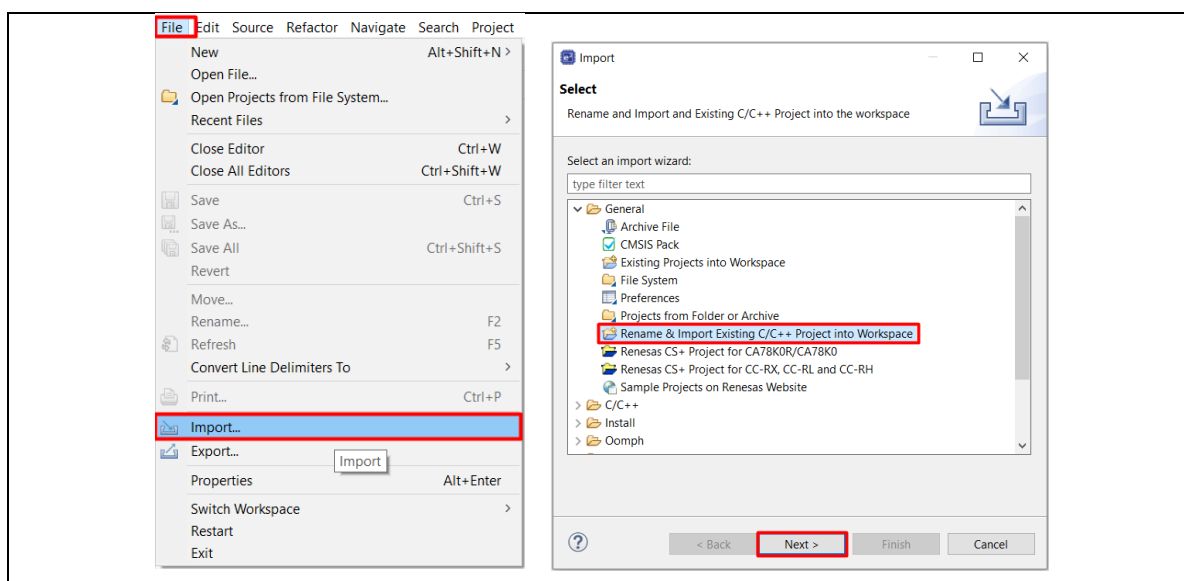


Figure 146. Select Rename and Import the Primary Application

2. Click **Next**. When the **Import** window opens, name the project and click **Browse** for **Select root directory** to import. Then select imported project in Project: box and click **Finish**.

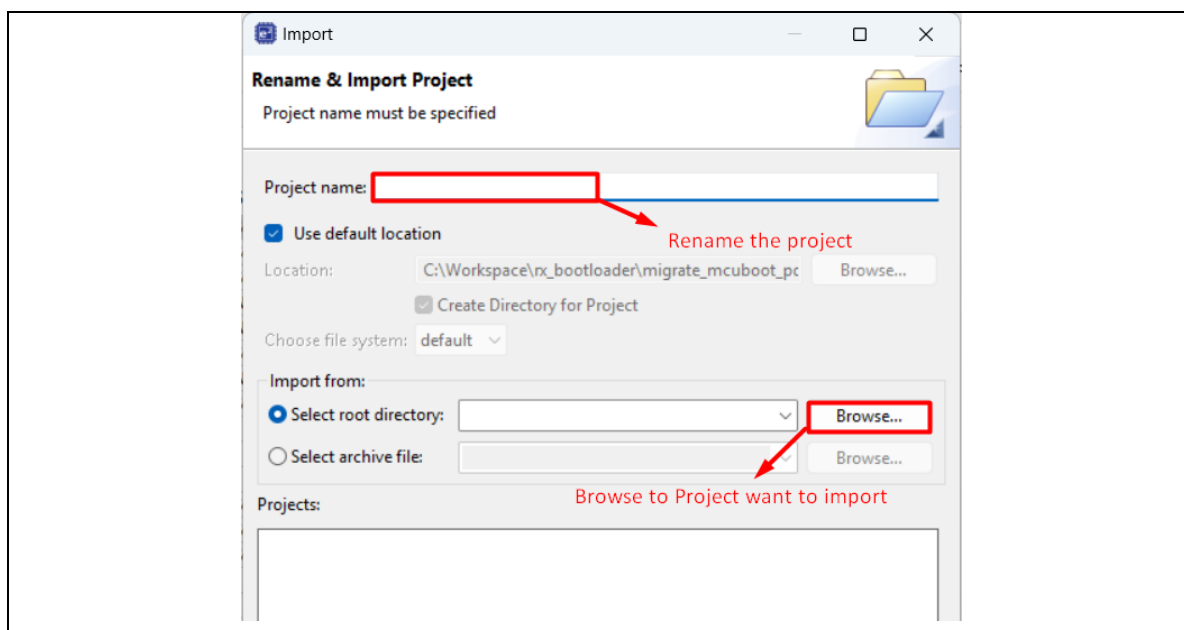


Figure 147. Rename the Project

3. Update the **MCUBOOT_IMAGE_VERSION** to **1.1.0**, click **OK** and **Apply and Close**

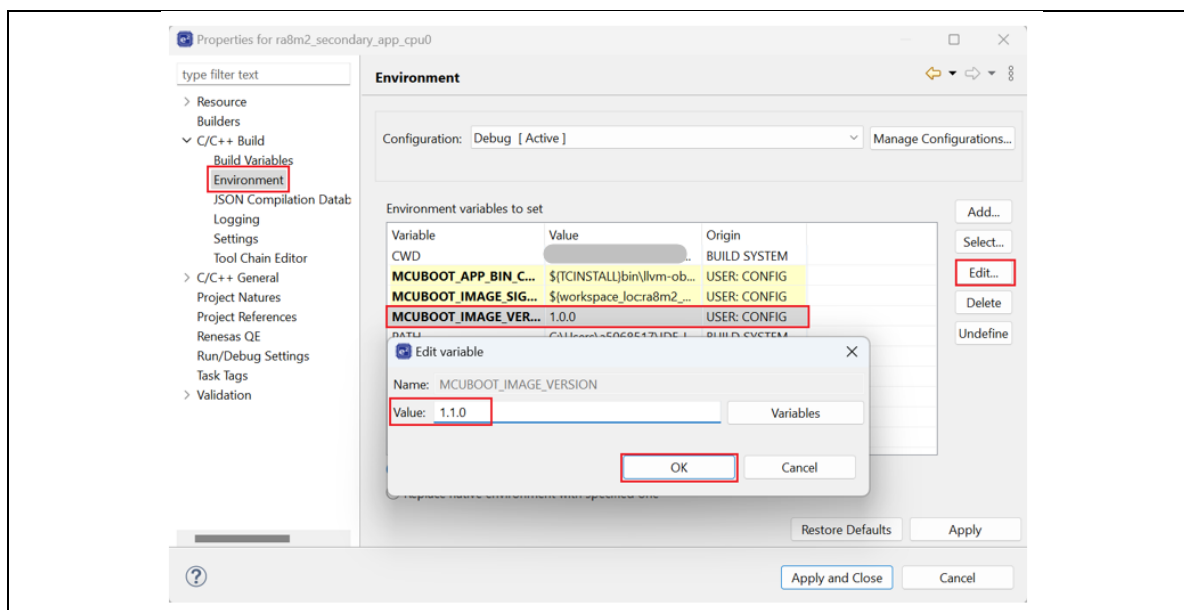


Figure 148. Update MCUBOOT_IMAGE_VERSION

4. Modify the behavior of LEDs to verify for secondary application execution. **This step** can apply to create secondary application on **single-core** and dual-core **CPU0** projects.

In case of **dual-core CPU1** project, it is required to **modify the SmartBundle** file to correct preceding project.

Follow following steps to modify for the **SmartBundle**:

- Build secondary application on CPU0, ensure **Debug/xxxx_secondary_app_cpu0.sbd** file is created.
- Navigate to the **Properties** of secondary project on CPU1 (for example, **ra8m2_secondary_app_cpu1**) project, under **C/C++ Build**, select **Build Variables**.
- Select **SmartBundle** and click **Edit...** button. Modify to secondary application CPU0 project.

For example,

Old:

`${workspace_loc:/ra8m2_primary_app_cpu0}/${ConfigName}/ra8m2_primary_app_cpu0.sbd`

New:

`${workspace_loc:/ra8m2_secondary_app_cpu0}/${ConfigName}/ra8m2_secondary_app_cpu0.sbd`

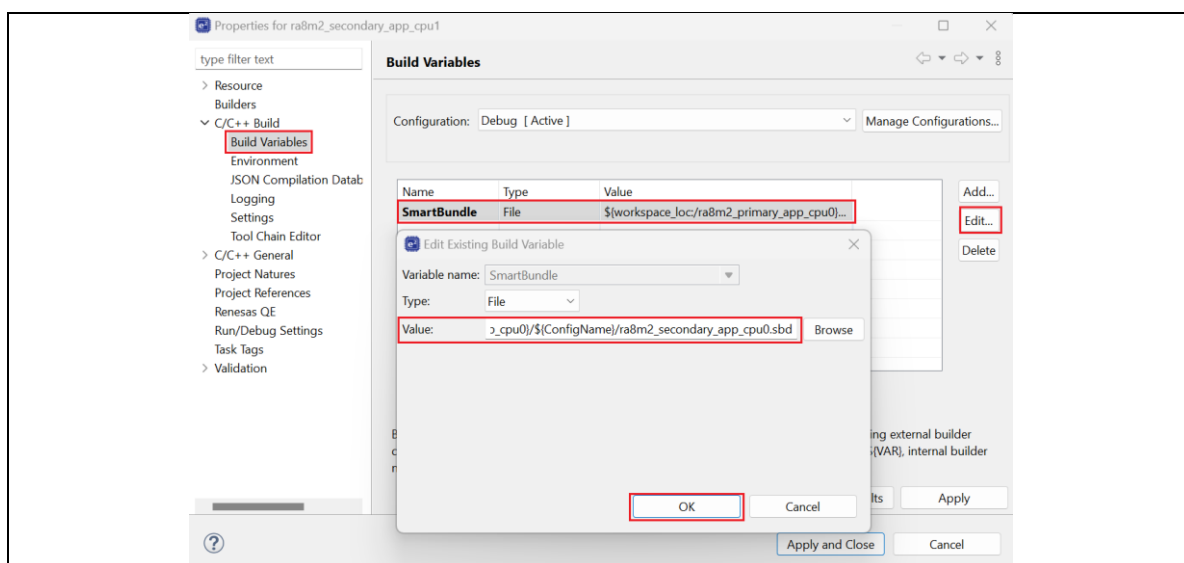


Figure 149. Modify SmartBundle for Secondary project on CPU1

5. Click **OK** and navigate to the **Environment** to update **MCUBOOT_IMAGE_VERSION** to **1.1.0**
6. Click **OK** and **Apply and Close** to apply the setting.
7. Modify the behavior of LEDs to verify for secondary application execution.
8. Build the secondary application project.

Table 17 shown the name of the secondary projects:

Table 17 Secondary Project Naming

Bootloader project name	Primary application project name	Secondary Application Project Name
ra8m2_bootloader	ra8m2_primary_app_cpu0	ra8m2_secondary_app_cpu0 ^(*)
	ra8m2_primary_app_cpu1	ra8m2_secondary_app_cpu1 ^(*)
ra8m1_bootloader	ra8m1_primary_app	ra8m1_secondary_app
ra6m5_bootloader	ra6m5_primary_app	ra6m5_secondary_app
ra6m3_bootloader	ra6m3_primary_app	ra6m3_secondary_app
ra4c1_bootloader	ra4c1_primary_app	ra4c1_secondary_app
ra2a2_bootloader	ra2a2_primary_app	ra2a2_secondary_app

(*) Recommended to repeat steps in section [5.1](#) to create secondary applications for Dual Core projects.

5.6.2 Downloading and Booting the Secondary Application

This section will detail for downloading the new application using the **Load Ancillary File** tool , available in e2studio.

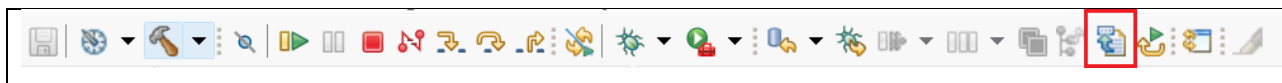


Figure 150. Load Ancillary File tool

Using the Load Ancillary File tool with Single Core RA MCU project

1. Click **Pause**  and use the **Ancillary Download**  button (which is available under the e2studio Debug view) to load the compiled Secondary Application **xxxx_secondary_app.bin.signed**.
2. Select the new application image and set the download address. The download address depends on the bootloader flash memory allocation.

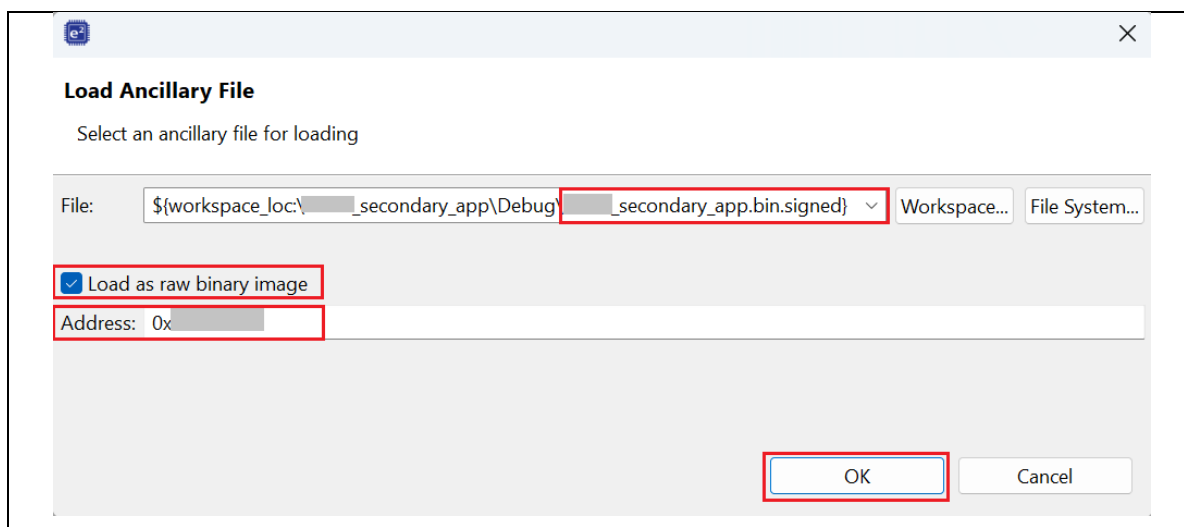


Figure 151. Download the Secondary Application Image

Table 18 Reference address for secondary images

MCU	Address: (<u>_BL_0_S_H</u> / <u>_BL_1_S_H</u>)
RA8M2_CPU0	0x02078000
RA8M2_CPU1	0x020A8000
RA8M1	0x020F8000
RA6M5	0x00108000
RA6M3	0x00108000
RA4C1	0x00046000
RA2A2	0x00047000

3. Click OK, and then click **Resume**  to allow the system to perform image swap / bank swap and the new image will be booted. The LED1 on board is blinking.

Note that for user-created customized applications, the download address needs to be adjusted by referencing the specific flash layout.

6. Production Support Considerations

Mastering and delivering a new application involves similar steps to those described in Sections [5.6](#). Typically, the following aspects must be considered when designing the process for delivering new applications:

1. Create and sign the new applications.

Refer to Section [5.1](#), [5.2](#) and [5.3](#) 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 may vary depending on the download method chosen by the user. In this example project, the *Ancillary File Download* feature in e² studio is used for demonstration purposes. This method can serve as a useful testing tool during the development of a custom image downloader. Alternatively, you can use the **Renesas Flash Programmer** tool to program the bootloader and application in a single operation. For other references, application projects **R01AN7766** and **R11AN0567** provide examples of image downloaders using XMODEM protocol and the USB PCDC interface.

7. Website and Support

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

RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
Renesas Flash Programmer (RFP)	renesas.com/en/software-tool/renesas-flash-programmer
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.10.26	-	Initial version

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/.