Renesas RA8 Series

Multicore Setup and Running Hello World on Dual-Core

Introduction

This application note serves as a practical "How-To" guide to help users set up, build, and run multi-core projects on the RA8P1 MCU using BareMetal and FreeRTOS. Specifically, it demonstrates how to execute a basic "Blinky Hello World" application on both cores of the RA8P1 device, showcasing the capabilities of its dual-core architecture.

The document provides an overview of the RA8P1 MCU's dual-core design, which integrates Cortex-M85 and Cortex-M33 cores, and outlines the steps required to configure, build, and debug dual-core applications. It also covers key topics such as CPU core selection (primary and secondary), core initialization, and the allocation of MCU resources for concurrent core operation.

For theoretical background, this app note should be used in conjunction with the "Developing with RA8 Dual-core MCU" and "Getting Started with IPC on Dual-Core RA8P1" app notes. These references provide deeper insight into dual-core development and the mechanisms behind inter-core communication.

This application note covers:

- Overview of the RA8P1 dual-core MCU, setting up dual-core configurations, and sharing resources.
- Explanation of how to set up and build multicore projects and run hello world on both cores with sample code examples.
- Techniques for managing core communication using semaphores, NMIs, message FIFOs, and FreeRTOS-based IPC.
- Step-by-step setup instructions, creating an application on both the cores, running the project on both the cores, and debugging the program on both the cores.
- Leveraging tool support (e^2 studio, Segger, GDB) for dual-core development.

Required Resources

- Flexible Software Package (FSP) v6.0.0

Target devices

- RA8P1

Contents

1. Dual-Core System Architecture

RA dual-core MCUs integrate two processing cores within a single chip to enhance performance, parallel processing, and power efficiency. These MCUs are commonly used in real-time applications, IoT devices, industrial automation, and automotive control systems.

Renesas RA dual-core MCU RA8P1 architecture falls into heterogeneous Dual-core MCU, with Cortex-M85 (CM85) is designated as Core 0, serving as the high-performance processing unit, while Cortex-M33 (CM33) is designated as Core 1, handling additional real-time control and system management tasks.

1.1 Overview of RA8P1 Dual-Core MCU

The Renesas RA8P1 is a next-generation MCU from the RA family, built on powerful dual-core Arm® architecture. It targets high-performance and secure embedded applications, offering advanced processing capabilities alongside robust security and low-power consumption.

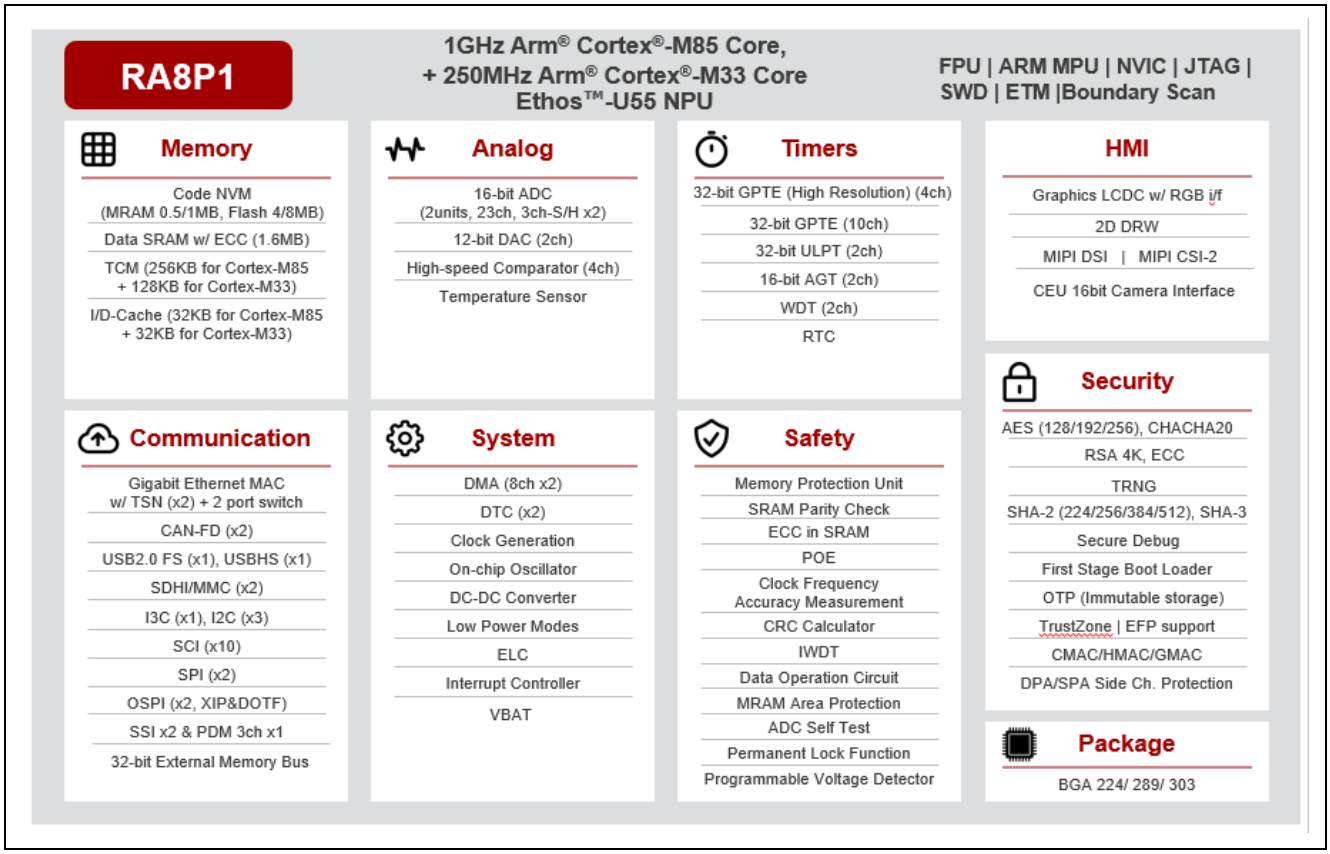

Figure 1. RA8P1 Dual-Core MCU Overview

Key Features:

- Dual-core architecture:
  - Arm Cortex-M85 core for real-time and high-performance processing.
  - Arm Cortex-M33 core for additional real-time tasks and secure operations.
- Up to 1GHz (CM85) and 250 MHz (CM33) clock speeds.
- TrustZone® support for hardware-enforced security domains.
- FPU, DSP, and Helium (MVE) support Cortex-M85.
- Rich set of peripherals, connectivity, and low-power modes.
- Integrated hardware security engine, cryptographic accelerators, and secure boot.

RA8P1 introduces a heterogeneous dual-core architecture that combines the Arm Cortex-M85 and Cortex-M33 cores on a single die. This setup is designed to provide optimal performance and task partitioning:

- Cortex-M85 Core (Primary CPU):
    - Serves as the high-performance core with Arm Helium (MVE) vector extensions for machine learning (ML) and digital signal processing (DSP).
    - Ideal for data-intensive tasks such as motor control, audio processing, and advanced ML inference.
    - Includes TrustZone for Secure/Non-Secure domain separation.
- Cortex-M33 Core (Secondary CPU):
    - Optimized for low-latency real-time tasks and secure workload management.
    - Often used for managing communication stacks, safety functions, or operating in an isolated secure domain.
- This asymmetric architecture allows developers to:
- Offload real-time or secure functions to the CM33, freeing the CM85 for compute-heavy tasks.
- Run independent software environments, including separate RTOS instances or bare-metal code.
- Leverage IPC to coordinate tasks between cores efficiently.

The architecture in the Renesas RA dual-core RA8P1 MCU is engineered to facilitate efficient communication and data exchange between cores and connected peripherals. The system primarily employs a shared bus architecture, where both cores interface with a common system bus linked to memory and peripherals. To maintain orderly access and prevent conflicts, arbitration mechanisms are implemented to manage resource sharing and prioritize requests fairly. In addition, dedicated buses are utilized for accessing Instruction Tightly Coupled Memory (ITCM) and Data Tightly Coupled Memory (DTCM), enabling low-latency and high-speed memory operations.

To manage shared resource access and coordinate operations between the two cores, the RA8P1 architecture utilizes IPC mechanisms. These include FIFO buffers, interrupt signaling, shared memory regions, and semaphores, all of which support synchronization and efficient data exchange. IPC ensures that both CPUs can work together effectively while preserving system stability and performance.

The underlying dual-core bus architecture is built on standard interconnects such as AXI, AHB, and APB buses, which facilitate communication between the Cortex-M85 and Cortex-M33 cores, as well as with memory and peripherals.

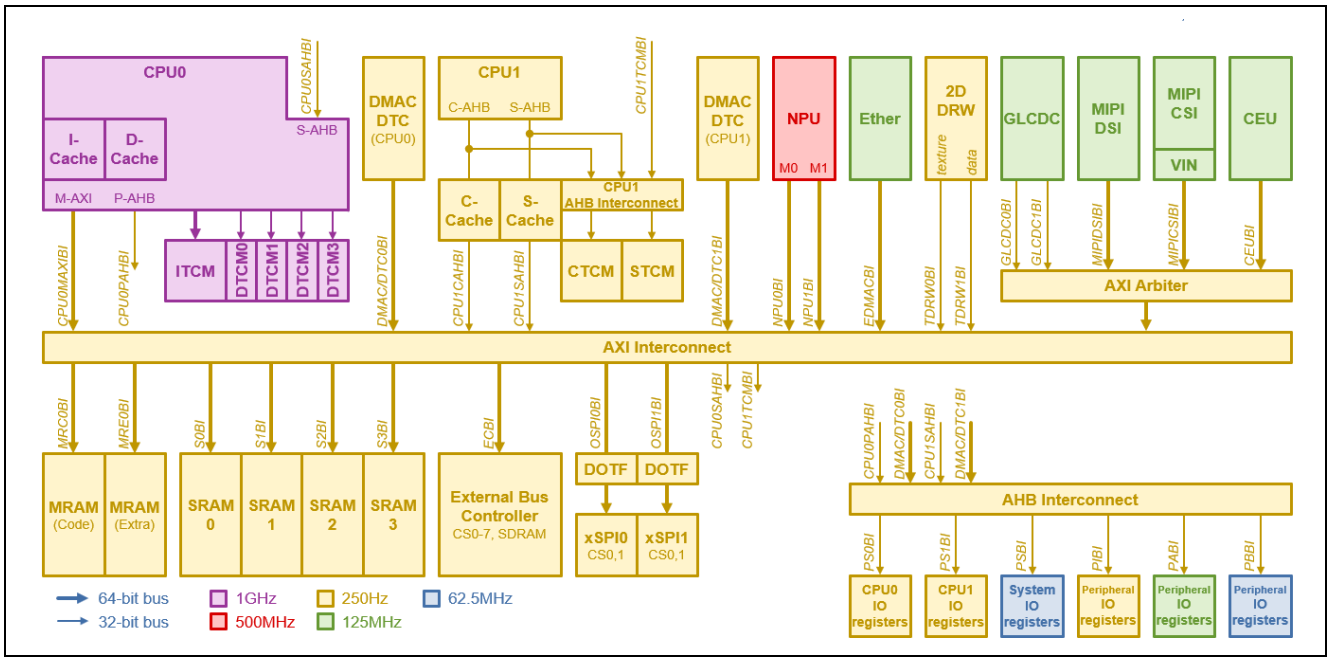

Figure 2. RA8P1 Dual-Core System Architecture Overview

The RA8P1 MCU features a high-performance Arm® Cortex®-M85 core alongside an Arm® Cortex®-M33 core offers the following key features:

- Up to 1 MB of MRAM for non-volatile memory storage
- 2 MB of SRAM, including:
- 256 KB TCM RAM for the Cortex-M85 core
- 128 KB TCM RAM for the Cortex-M33 core
- 1.664 MB of user-accessible SRAM
- Integrated Arm® Ethos™-U55 Neural Processing Unit (NPU)
- Octal Serial Peripheral Interface (OSPI) for high-speed flash memory access
- Layer 3 Ethernet Switch Module (ESWM), USB Full-Speed (USBFS), USB High-Speed (USBHS), and SD/MMC Host Interface
- Graphics LCD Controller (GLCDC) for advanced display support
- 2D Drawing Engine (DRW) for hardware-accelerated graphics rendering
- Support for MIPI DSI/CSI interfaces for display and camera connectivity
- Rich set of analog peripherals
- Comprehensive security and safety features for reliable operation

1.1.1 Arm® Cortex®-M85 Core Processor

The CM85 core in the Renesas RA8P1 MCU family is built on Arm's high-performance Cortex®-M85 processor, delivering a significant boost in processing capabilities for next-generation embedded applications. This core is based on the Arm®v8.1-M architecture and brings a combination of high-speed execution, enhanced math processing, and improved security, making it well-suited for compute-intensive tasks such as motor control, AI/ML, signal processing, and industrial automation.

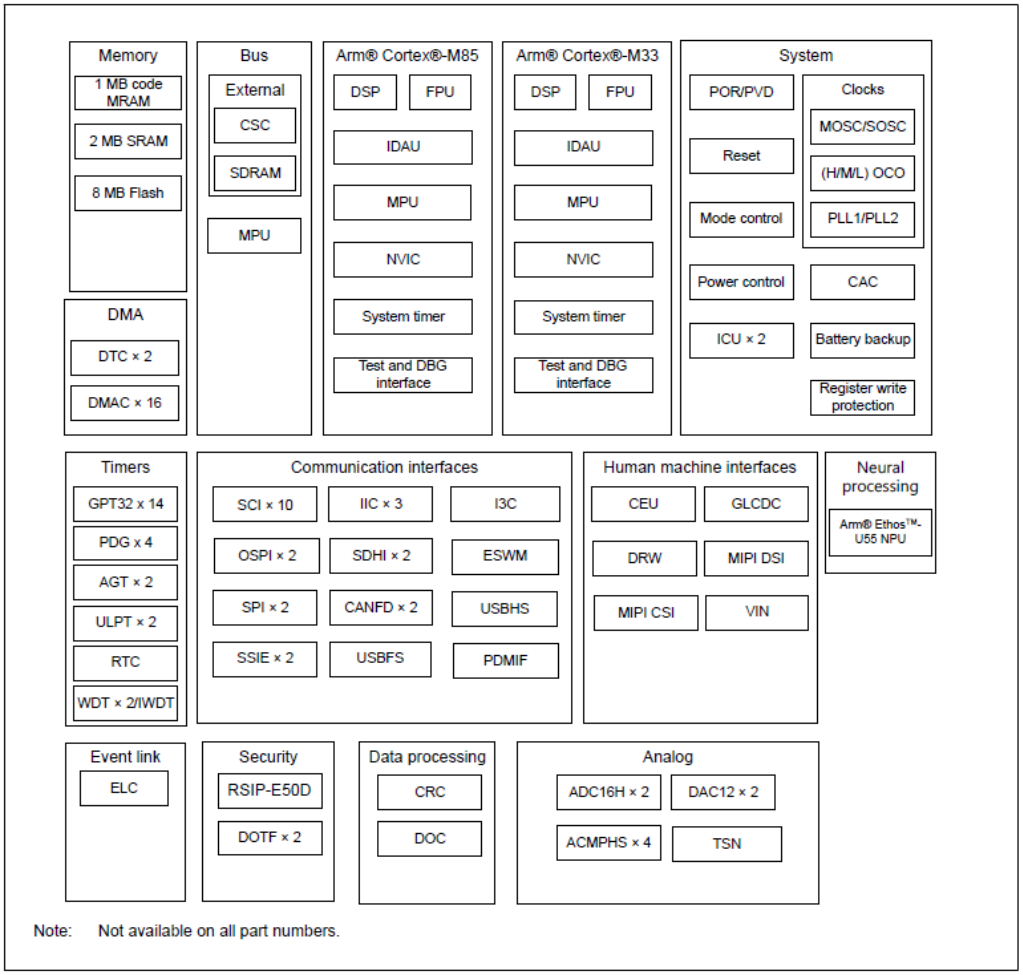

Figure 3. RA8P1 Dual-Core System Block Overview

Developers can refer to the “High Performance with RA8 Helium” application note and Arm's official documentation for detailed technical references.

Arm Cortex-M85 Processor

- A high-performance embedded processor designed for demanding applications.
- Based on ARMv8.1-M architecture, which brings significant performance and security enhancements over earlier Cortex-M cores like M4 and M7.

Floating-Point Unit (FPU)

- IEEE 754-2008 compliant: Ensures accuracy and portability for floating-point math.
- Supports scalar half (16-bit), single (32-bit), and double (64-bit) precision floating-point operations.
- FPU exceptions can raise interrupt notifications, allowing precise error detection (e.g., divide-by-zero, overflow).

M-Profile Vector Extension (MVE)

- Also called Helium; introduces SIMD-like vector processing to Cortex-M.
- MVE-F: Supports integer, half-, and single-precision float vector operations.
- Useful for DSP, AI/ML, and signal/image processing in real-time embedded systems.

Security Features

Armv8-M Security Extension

- Allows TrustZone, enabling secure and non-secure execution environments for isolating critical code/data.

SAU (Security Attribution Unit)

- 8 programmable regions define which parts of memory are Secure vs. Non-Secure.

IDAU (Implementation Defined Attribution Unit)

- Works alongside the SAU, implemented by the SoC vendor to fine-tune secure attribution.

Memory Protection

MPU (Memory Protection Unit)

- Supports ARM's PMSAv8 (Protected Memory System Architecture).
- Split into:
  - Secure MPU (MPU_S): 8 regions for code/data running in secure state.
  - Non-Secure MPU (MPU_NS): 8 regions for normal application-level code.
- Helps detect invalid memory access and isolate components.

System Timers

- SysTick
  - A timer for OS tick or periodic tasks.
- Two instances:
- SysTick_S: Secure world
- SysTick_NS: Non-Secure world

Reference Clock

- From CPUCLK0 (core clock), or a 1 MHz clock derived from MOCO/8 (Middle-speed On-Chip Oscillator divided by 8).

Low-Power Modes

- Sleep Mode: CPU halts, but clocks and peripherals may stay active.
- Deep Sleep Mode: Further power reduction; clocks may be stopped or slowed.

CPU Reset

- Each CPU can be reset individually, enabling fine control in multicore systems.

Cache

Instruction Cache

- 16 KB with ECC: Speeds up instruction fetch; ECC (Error Correcting Code) improves reliability.

Data Cache

- 16 KB with ECC: Speeds up access to frequently used data.

Tightly Coupled Memory (TCM)

Low-latency memory mapped close to the CPU for real-time performance.

ITCM (Instruction TCM)

- 128 KB (16 × 8 KB blocks), ECC protected

DTCM (Data TCM)

- 128 KB (16 × 8 KB blocks), ECC protected

Interrupt Controller

NVIC (Nested Vectored Interrupt Controller)

- Handles 96 interrupt sources with nested priorities.
- Key to real-time response.

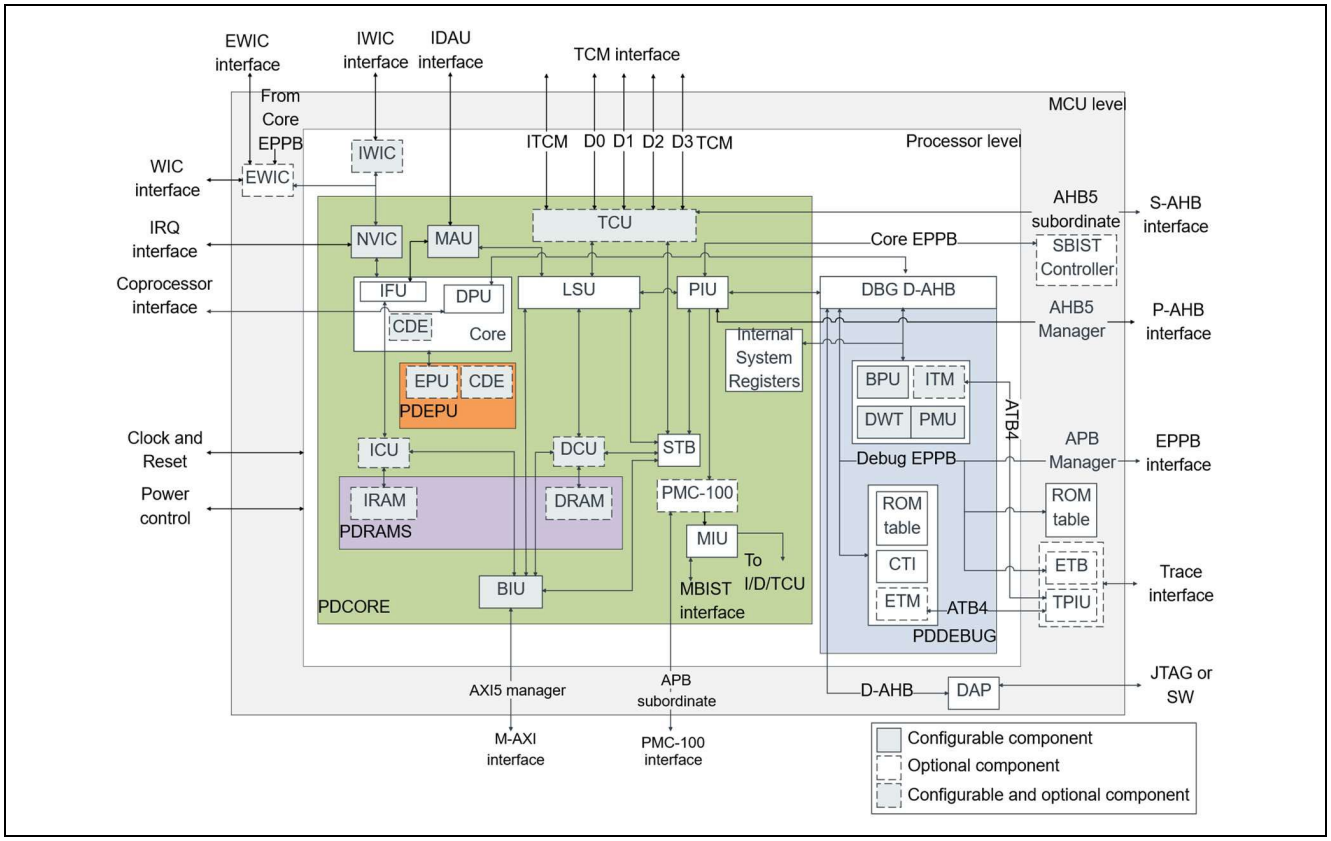

Figure 4. Arm Cortex-M85 Core System Overview

1.1.2 Arm® Cortex®-M33 Core Processor

- A mid-to-high performance embedded processor core based on the ARMv8-M architecture, targeting secure IoT and real-time applications.
- Supports TrustZone, DSP, and floating-point operations.

Architecture

- **ARMv8-M Profile**
  - Includes enhancements for:
    - Security (via TrustZone)
    - Digital signal processing (DSP)
    - Energy efficiency
  - Focuses on low-latency interrupt handling and deterministic behavior.

Floating-Point Unit (FPU)

- Compliant with ANSI/IEEE Std 754-2008
    - Enables accurate and standardized floating-point math.
- Supports only single-precision (32-bit) floating point.
- FPU exception handling: Can trigger interrupts on faults (e.g., divide by zero, invalid operation).

Extensions

- **Armv8-M Security Extension**
    - Enables TrustZone:
        - Separates execution into Secure and Non-Secure worlds.
        - Used for protecting sensitive operations (e.g., crypto, boot code).
    - Can be disabled permanently via OTP (One-Time Programmable) memory:
        - If disabled, CPU1 loses TrustZone support.
        - Also, CPU1 cannot act as the primary CPU unless this is enabled.
- **Armv8-DSP Extension**
    - Adds DSP instructions like MAC (Multiply-Accumulate), saturation arithmetic, and SIMD-like parallelism for 16-bit and 8-bit data.
    - Enhances performance in signal processing, audio, control systems, and AI/ML inference.

Security and Memory Protection

- **SAU (Security Attribution Unit)**
    - Defines Secure/Non-Secure memory regions.
    - 8 programmable regions.
- **IDAU (Implementation Defined Attribution Unit)**
    - Configured by the silicon vendor (not part of the core IP).
    - Works with SAU to define fixed secure areas (e.g., bootloader, hardware registers).
- **MPU (Memory Protection Unit)**
    - Part of the PMSAv8 (Protected Memory System Architecture).
    - Enforces access control, avoiding accidental corruption/misuse of memory.
    - **Split into:**
        - Secure MPU (MPU_S): 8 regions
        - Non-Secure MPU (MPU_NS): 8 regions

System Timer

- **SysTick**
    - A configurable timer typically used for OS tick (RTOS).
    - Two separate instances:
        - SysTick_S for secure execution
        - SysTick_NS for non-secure tasks
    - Clock source options:
        - CPUCLK1: Main clock driving CPU1
        - 1 MHz clock from MOCO/8: A fixed internal oscillator divided by 8

Power Modes

- Sleep Mode
  - CPU halts, but peripherals and clocks remain active.
  - Fast wake up.
- Deep Sleep Mode
  - Greater power savings than Sleep mode.
  - Some clocks and power domains may be shut down.

Caches

- Code-bus Cache (C-Cache): 16 KB
  - Optimizes code fetch performance from flash or external memory.
  - ECC protected: detects and corrects bit errors.
- System-bus Cache (S-Cache): 16 KB
  - Speeds up data transactions to/from system bus (e.g., peripherals or RAM).
  - Also protected with ECC for data integrity.

Tightly Coupled Memory (TCM)

Ultra-fast SRAM is located close to the core for deterministic real-time performance.

- CTCM (Code TCM): 64 KB
  - Used for storing time-critical code.
  - With ECC.
- STCM (System/Data TCM): 64 KB
  - Used for fast data access.
  - Also ECC protected.

Interrupt Controller

- Nested Vectored Interrupt Controller (NVIC)
  - Supports up to 96 IRQs (interrupt sources).
  - Allows prioritization and preemption.
  - Essential for real-time response and efficient task scheduling.

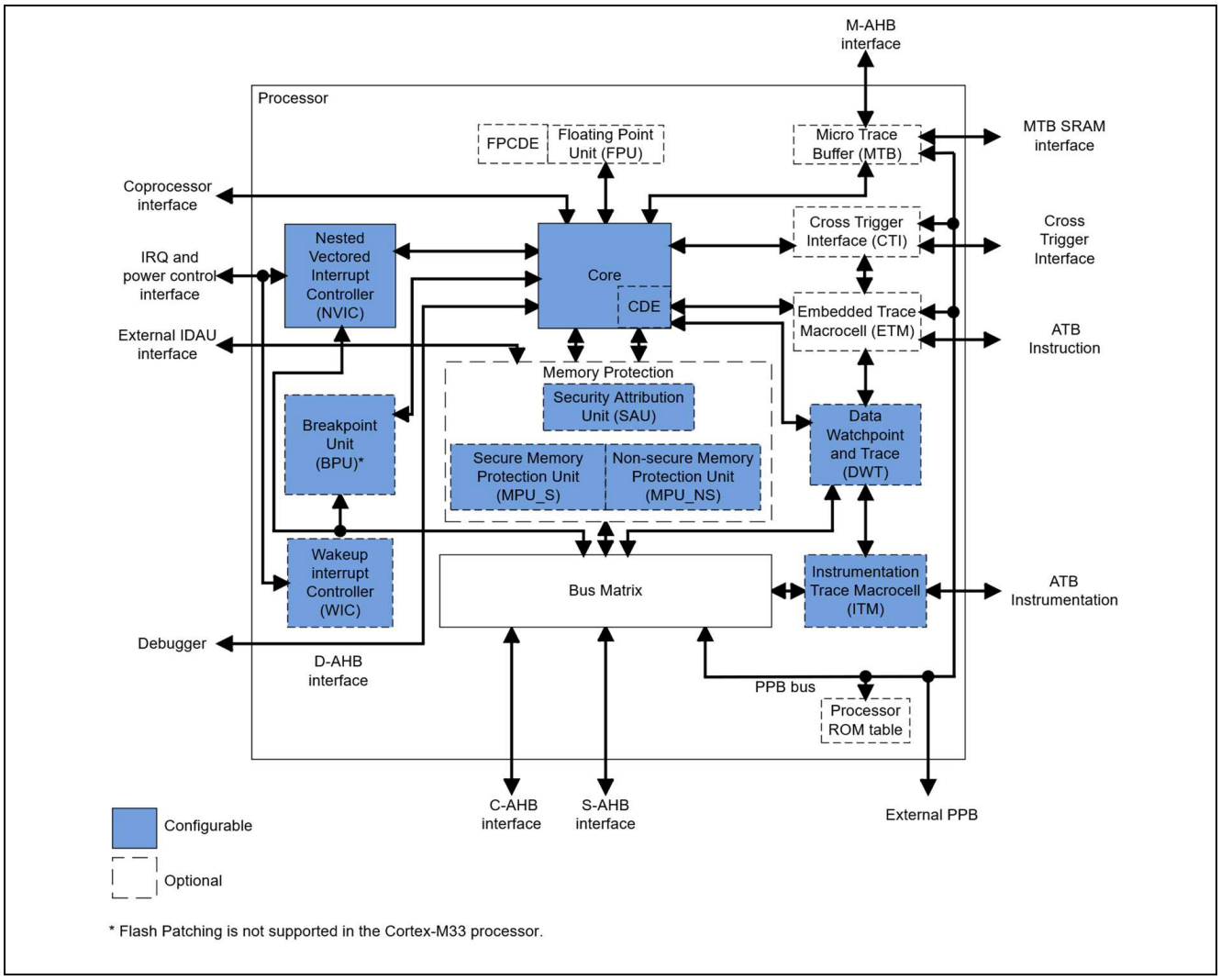

Figure 5. Arm Cortex-M33 Core System Overview

2. Procedure for Creating Dual-Core Projects

Developing dual-core projects for Renesas RA MCUs involves a slightly different approach compared to creating standard single-core RA projects, even though many of the fundamental steps remain the same.

While both project types use the Renesas Flexible Software Package (FSP), board support packages (BSPs), and e² studio IDE workflows, dual-core development introduces additional considerations. These include managing inter-core communication, defining memory boundaries, coordinating startup sequences, and configuring multicore debug environments.

In this section, a detailed, step-by-step guide is provided to walk you through the dual-core project creation process. It covers the following key aspects:

- Selecting the appropriate RA dual-core device.
- Using the Multicore Solutions Project Wizard to generate CPU0 and CPU1 projects.
- Initializing the device properly to set up security levels and TrustZone boundaries.
- Setting up Inter-Processor Communication (IPC) mechanisms.
- Creating and configuring a Launch Group for multicore debugging.
- Building and linking the projects to ensure synchronized execution.

By following this procedure, you can effectively create and manage dual-core applications tailored to your system requirements, leveraging the full power of the RA MCU’s multicore architecture.

2.1 Primary and Secondary CPU Selection

In a dual-core setup on the RA8P1 MCU, by default CPU0 (Cortex-M85) is designated as the primary core, while CPU1 (Cortex-M33) is the secondary core. The FSP and BSP, along with the e² studio IDE, support project creation and configuration for both cores.

There are two main approaches to creating dual-core projects:

1) Manual Project Creation (Individual Projects and Linking Them)

This method involves creating separate projects for each core (CPU0 and CPU1) and linking them using an SBD (Solution Bundle) file. Although this approach requires more manual effort, it provides greater flexibility.

- Supports Bare Metal, RTOS-based, and TrustZone (Secure/Non-Secure) configurations
- Allows full control over memory maps, project structure, and build settings.

2) Solution Project Creation (Automated Dual-Core Setup)

The solution project method simplifies dual-core development by automatically generating the individual projects for both cores under a unified solution.

- Generates a top-level solution project that manages both CPU0 and CPU1 projects.
- Provides an integrated graphical memory configuration interface for peripheral and resource setup across both cores.
- Currently it supports only Bare Metal configurations (no RTOS or TrustZone setup in initial versions).

2.2 Procedure for Creating Manual Dual-Core Projects and Their Configuration

This section outlines the detailed steps for creating a dual-core project on the RA8P1 MCU series. If needed, refer to the FSP documentation for additional guidance.

During project creation, you will:

- Choose the project type
- Specify the project name and location
- Configure essential settings, including:
    - FSP version
    - Target board
    - Selected core (Cortex-M85 or Cortex-M33)
    - RTOS inclusion (if applicable)
    - Toolchain version

These instructions will walk you through the complete process using a simple Blinky application as an example, ensuring you understand how to set up and configure a functional dual-core project.

Open e² studio, then navigate to File → New → Renesas C/C++ Project → Renesas RA

Select Renesas RA C/C++ Project and click Next to proceed.

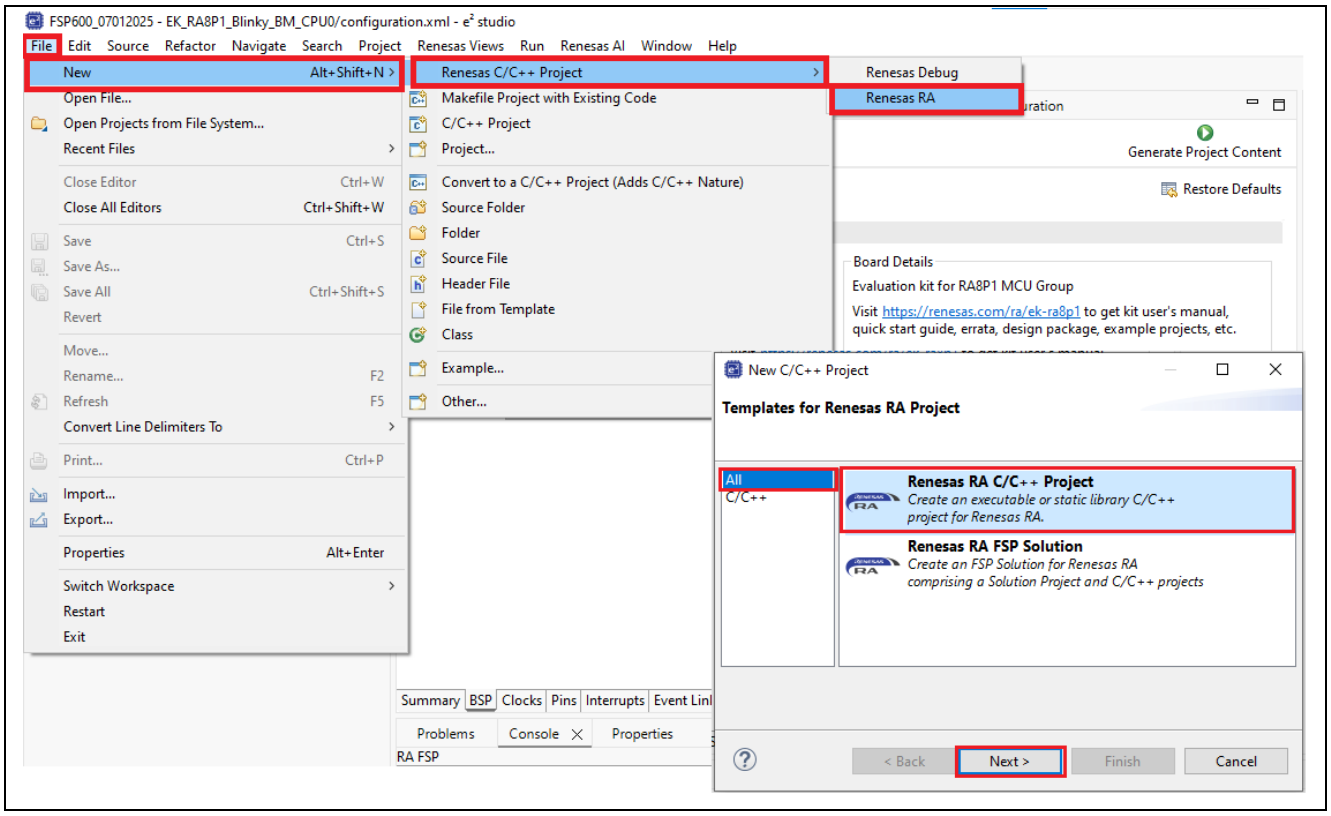

Figure 6. Create an EK-RA8P1 CPU0 Project using e² studio

Enter a name for your new project “EK_RA8P1_CPU0” and click Next.

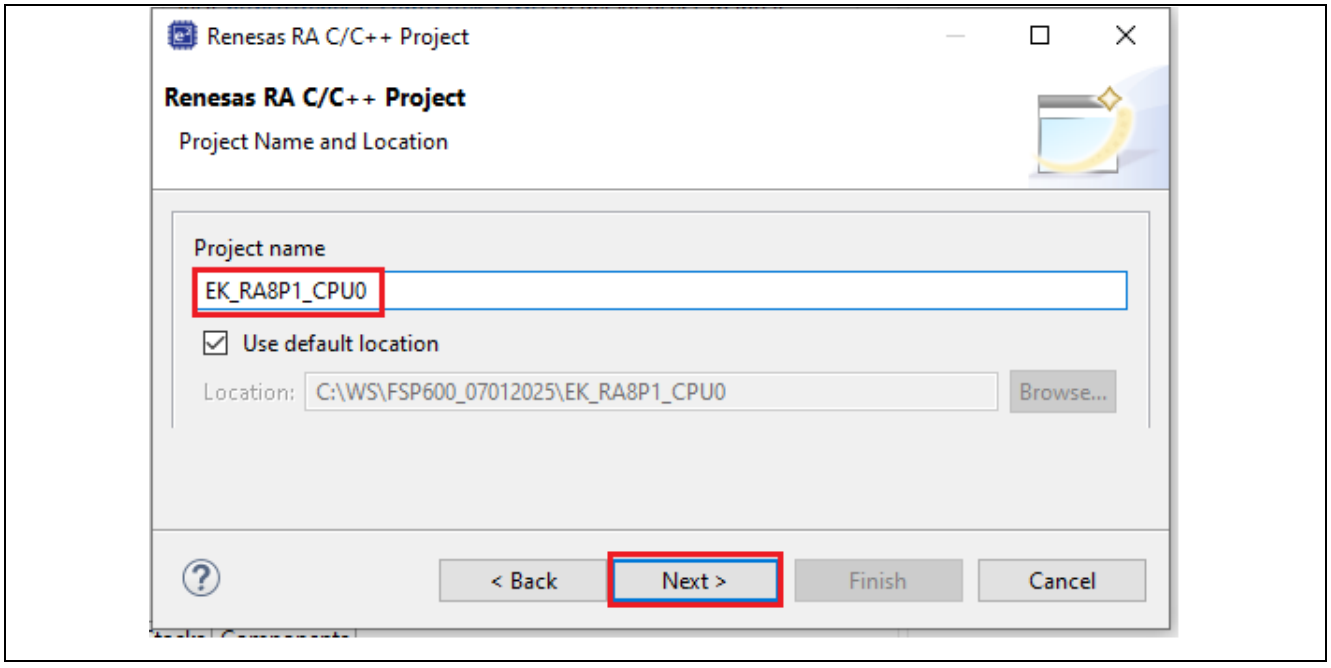

Figure 7. Project Name for EK-RA8P1 CPU0 Project using e² studio

In the Device and Tools Selection window:

- Set the board to EK-RA8P1.
- Choose CPU0 (Cortex-M85) as the target core.
- Select the LLVM Embedded Toolchain for Arm as the toolchain.

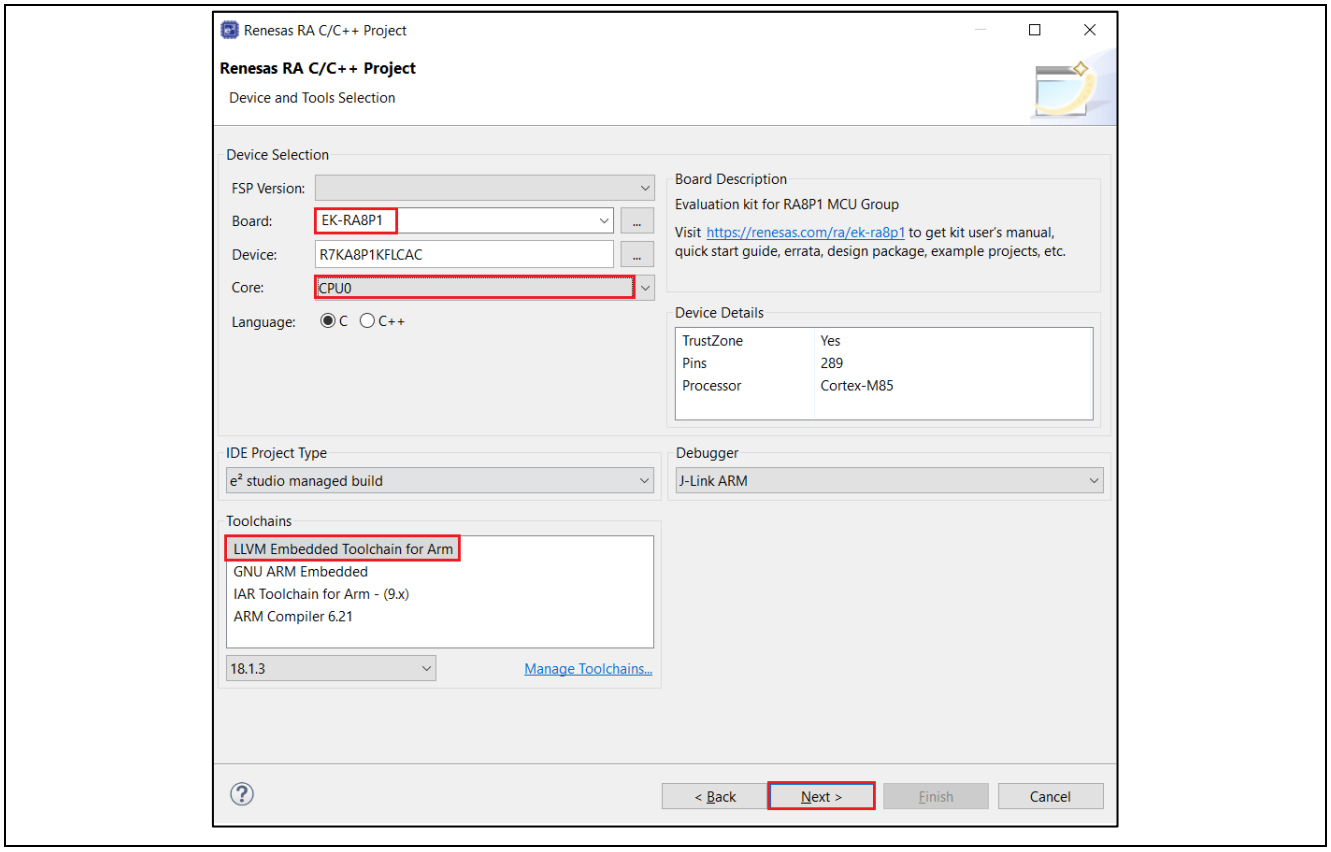

Figure 8. Selecting from the Device and Tools Selection CPU0

Select Flat (Non-TrustZone) Project in Project Type Selection and click Next.

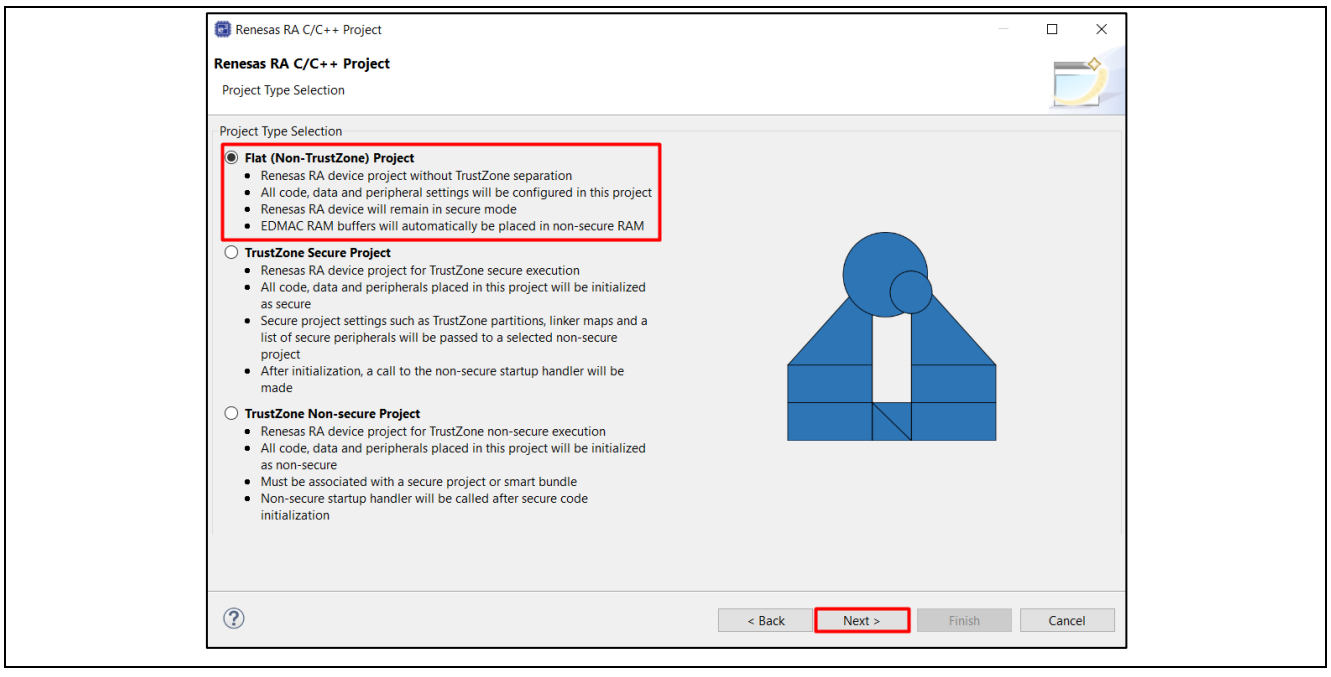

Figure 9. Flat (Non-TrustZone) Project Type Selection

Select None for Preceding Project or Smart Bundle Selection and click Next.

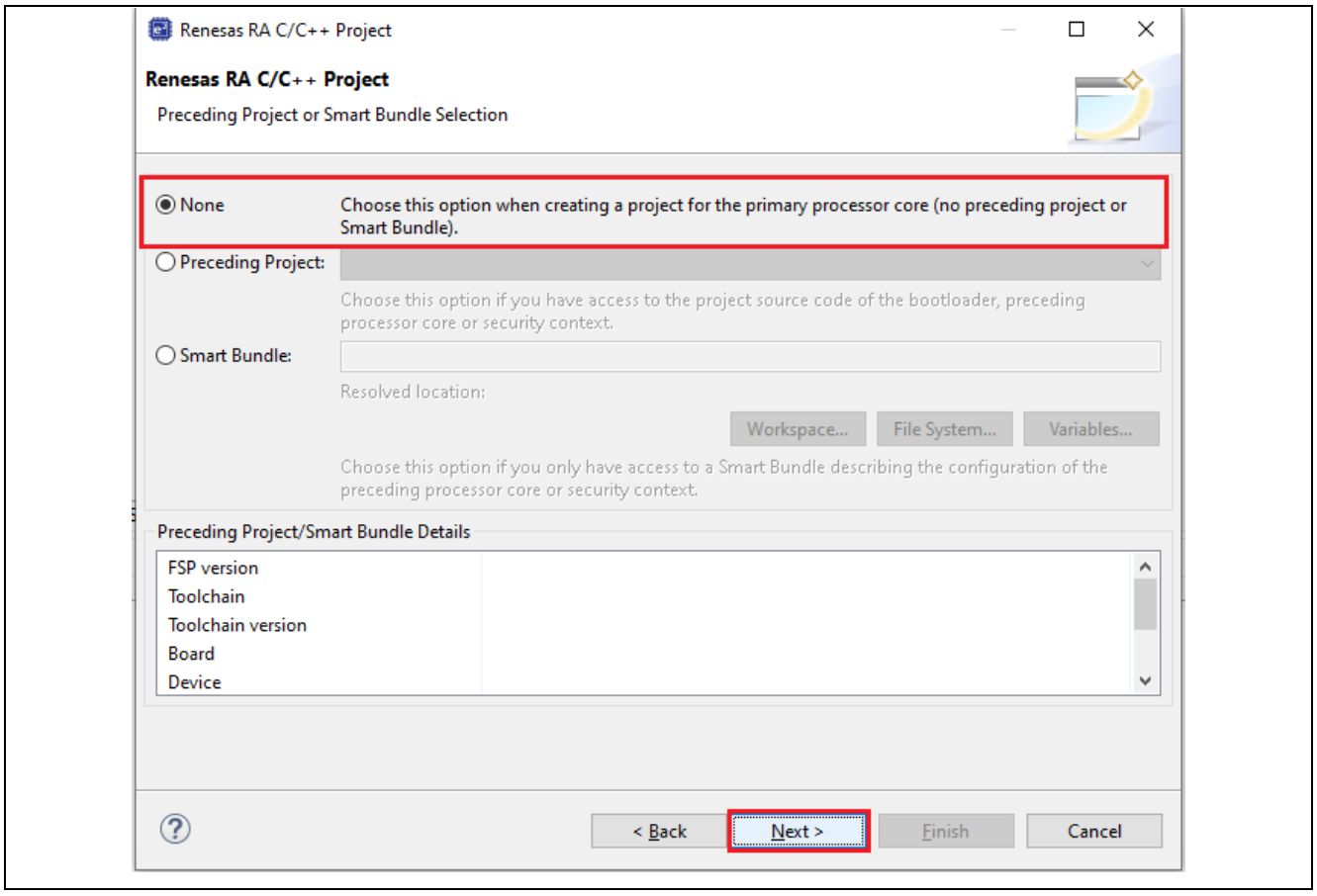

Figure 10. No preceding project or Smart Bundle Selection

Select Executable for Build Artifact Selection and No RTOS for RTOS Selection, and click Next.

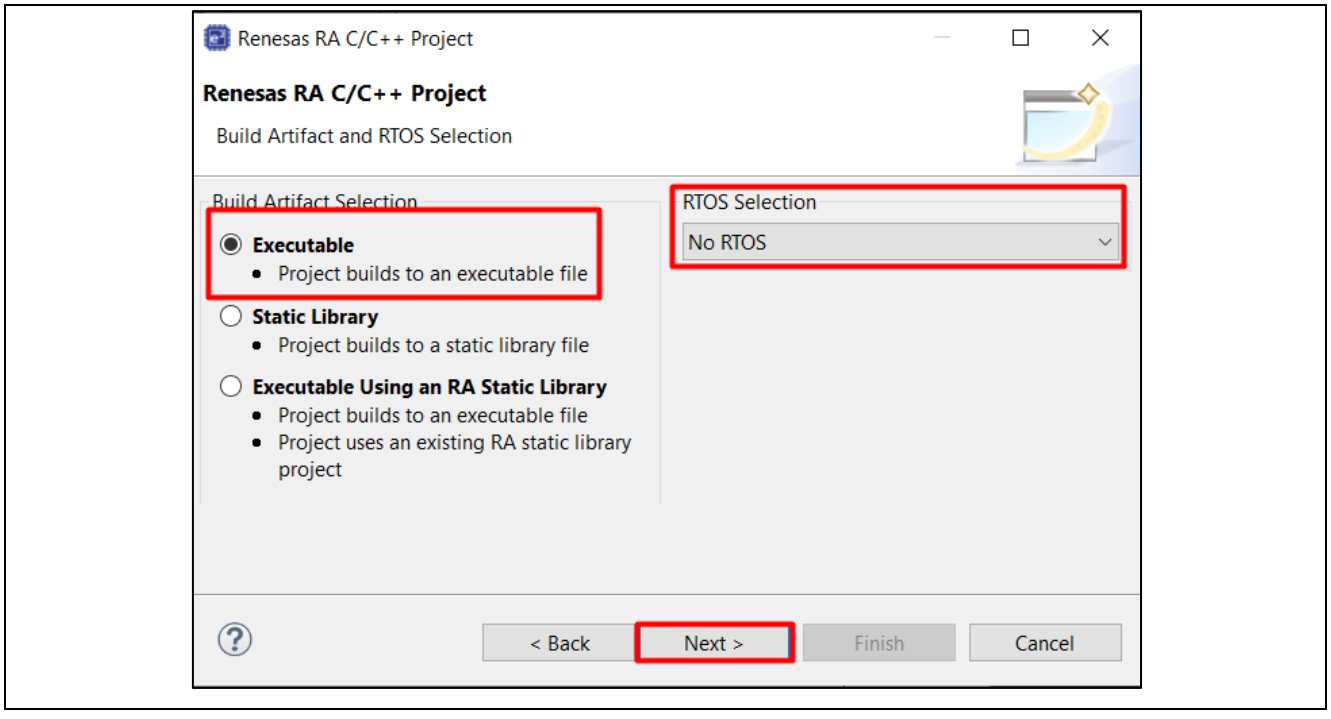

Figure 11. Executable for Build Artifact and No RTOS Selection

Select Bare Metal - Blinky for this example and click Finish.

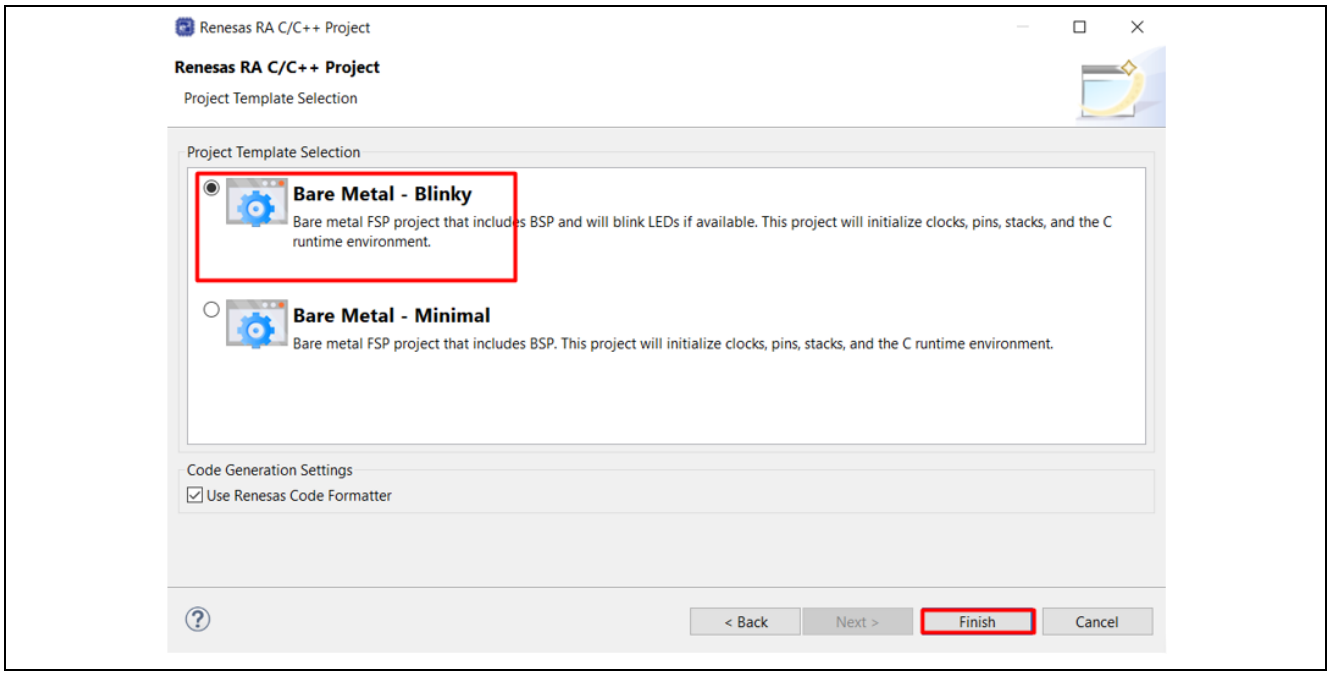

Figure 12. Blinky Project Template Selection

Generate Project Content and compile the project template.

Double-click Configuration.xml to open the configurator. Click Generate Project Content as shown in Figure 13.

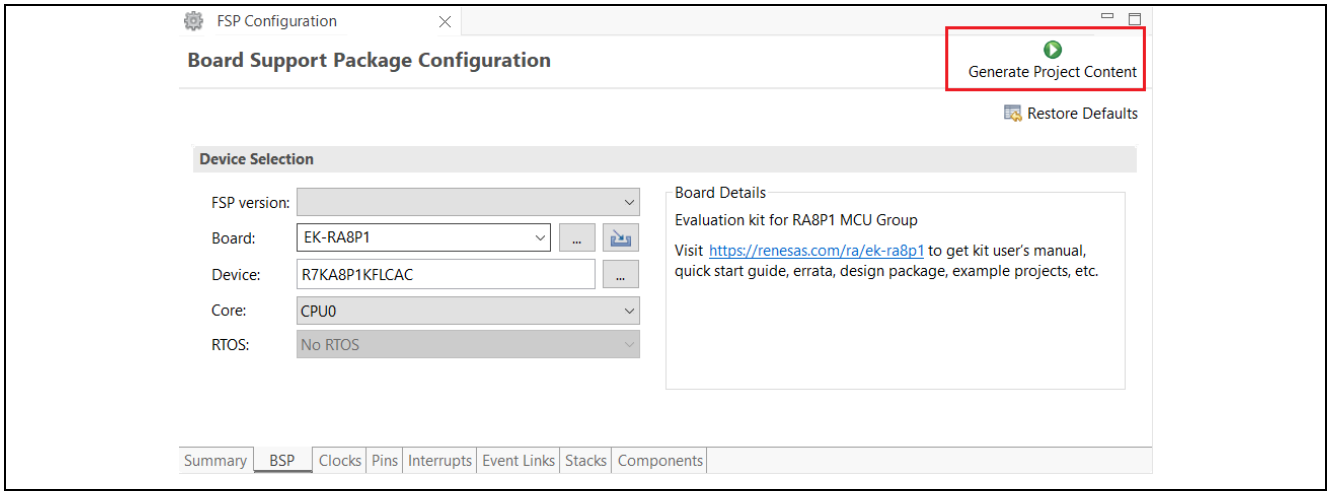

Figure 13. Generate Project Content

Note: Add the MACRO setting to define under the project settings → C/C++ Build → Settings → Compiler → Includes → Macro Defines (-D) → "BSP_PARTITION_FLASH_CPU1_S_START" as shown in the snapshot below to avoid the compiler/linker error.

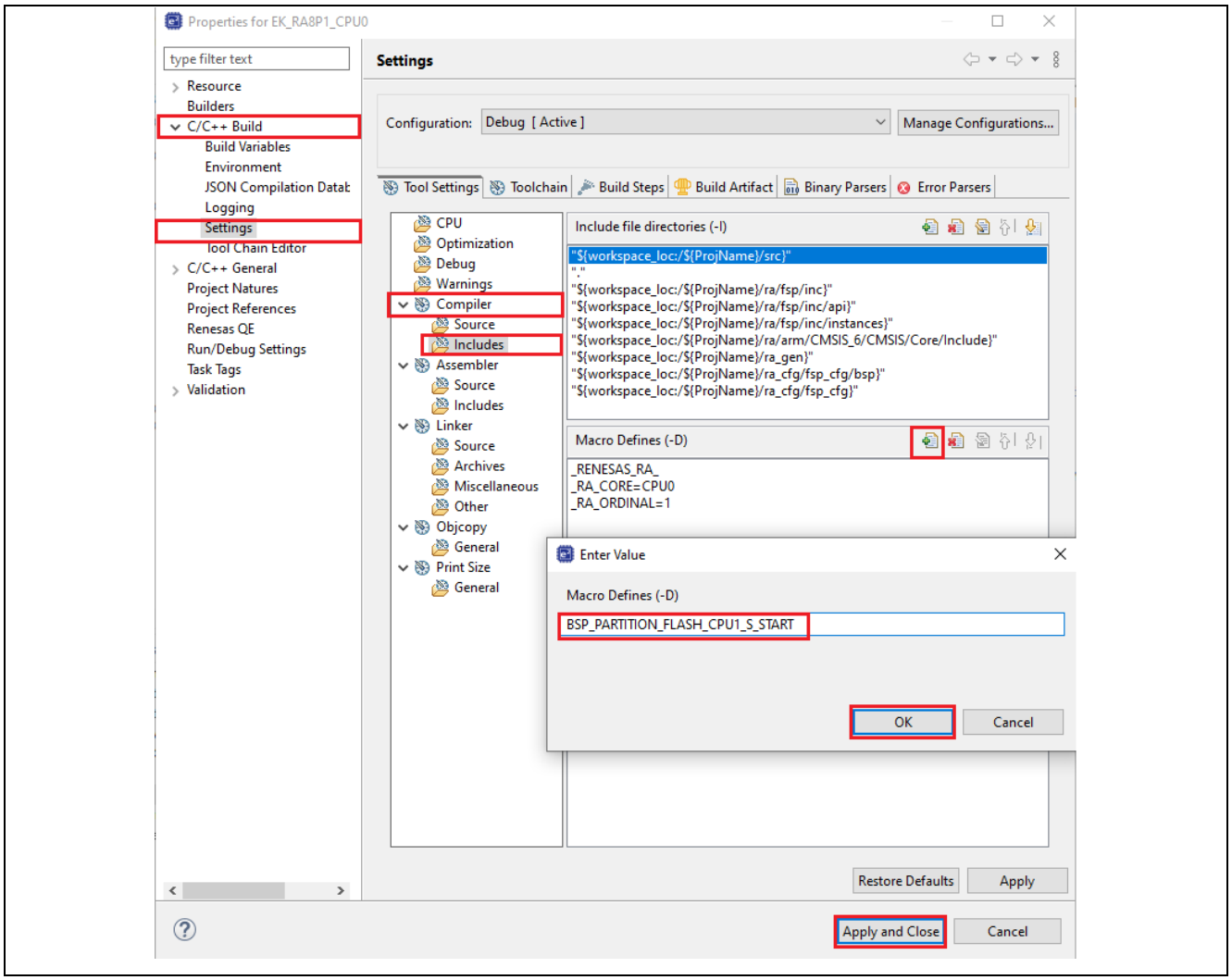

Figure 14. Add the MACRO under the project settings - CPU0 Blink Project

Right-click on the project and select the Build Project.

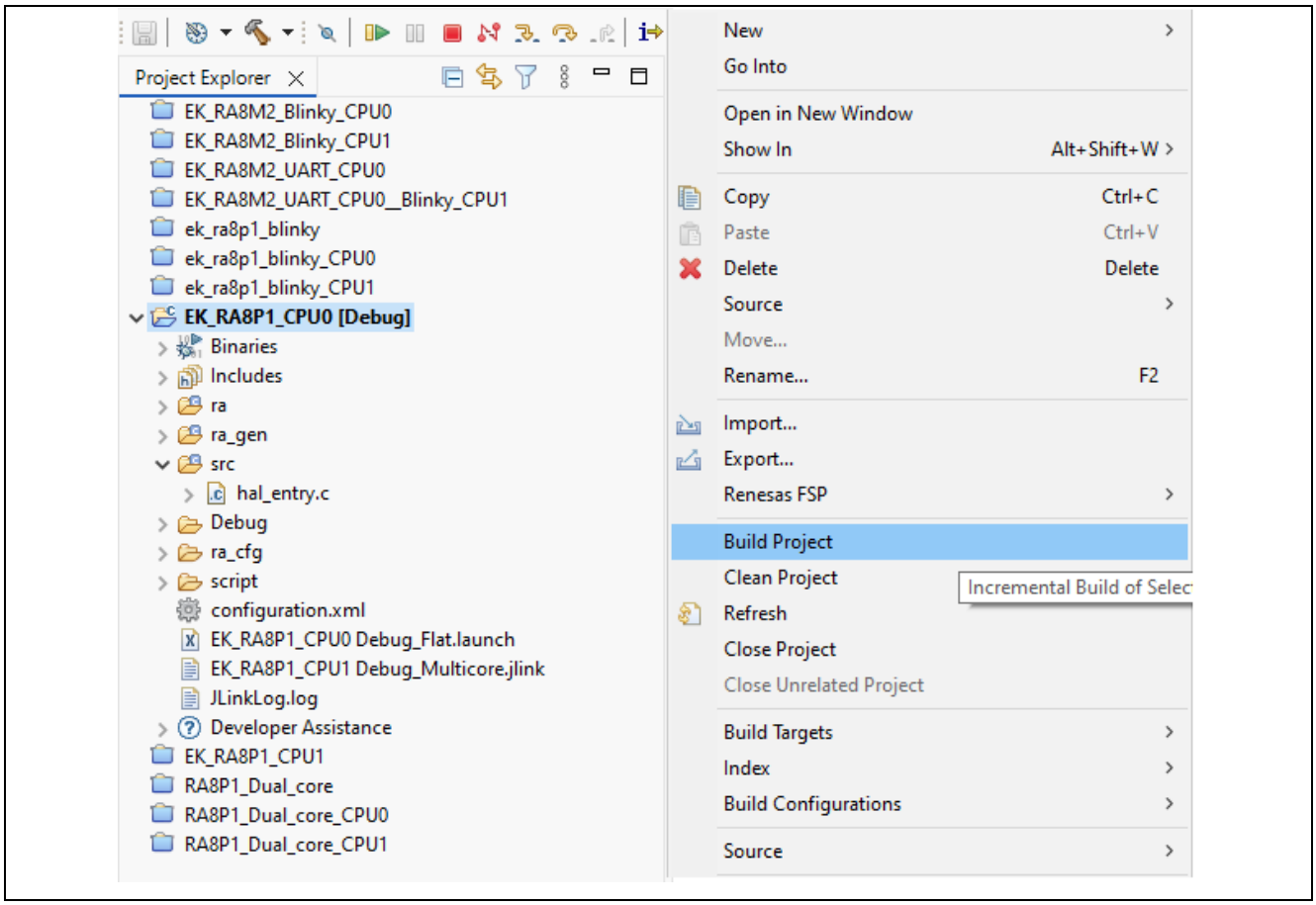

Figure 15. Compile the EK_RA8P1_CPU0 Blinky Project

After successful compilation, the Smart bundle.sbd is generated as shown in Figure 16.

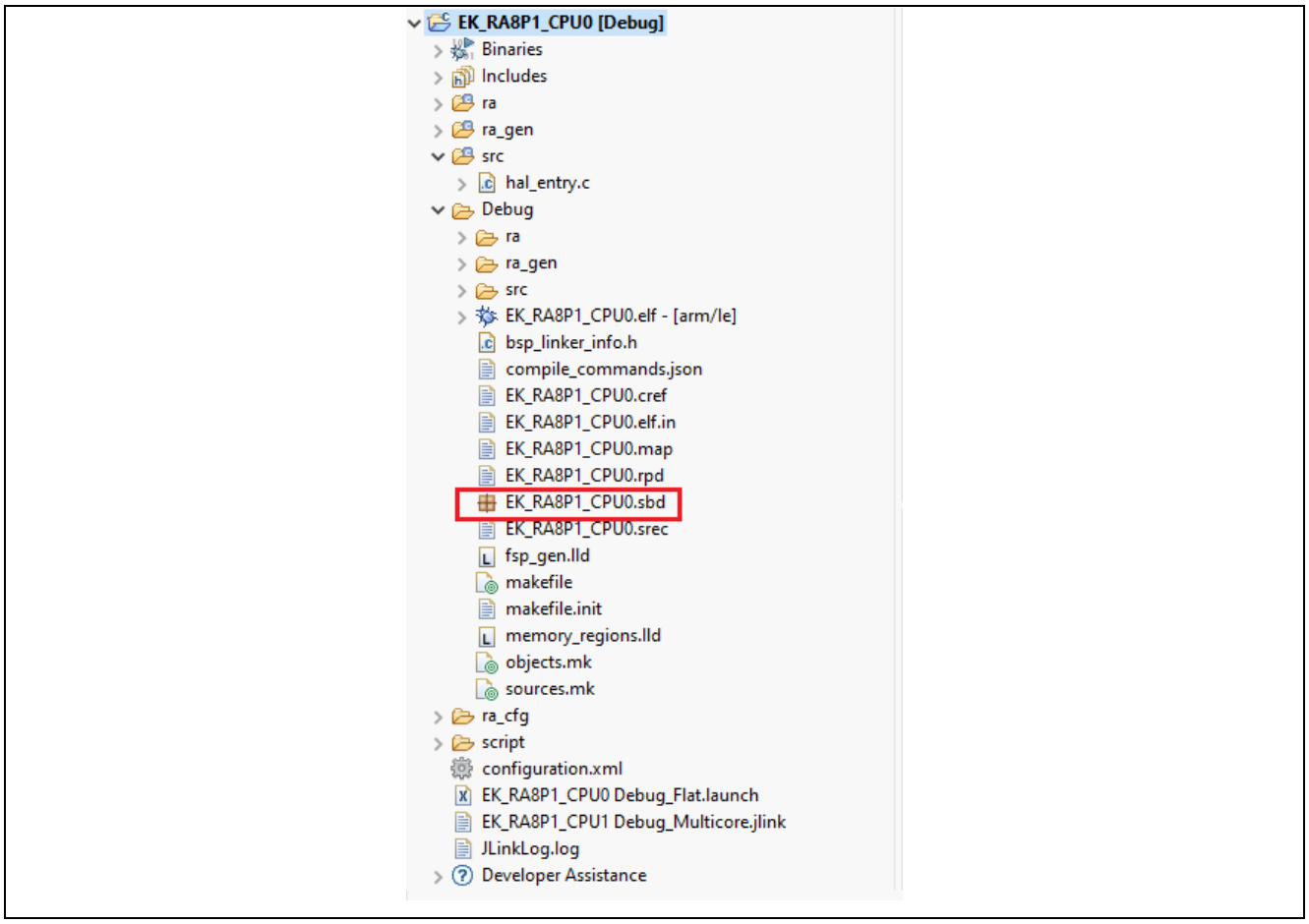

Figure 16. Smart Bundle Generated

Create a Flat Blinky Project with CM33 core

Click File → New → Renesas C/C++ Project → Renesas RA and select Renesas RA C/C++ Project → Click Next.

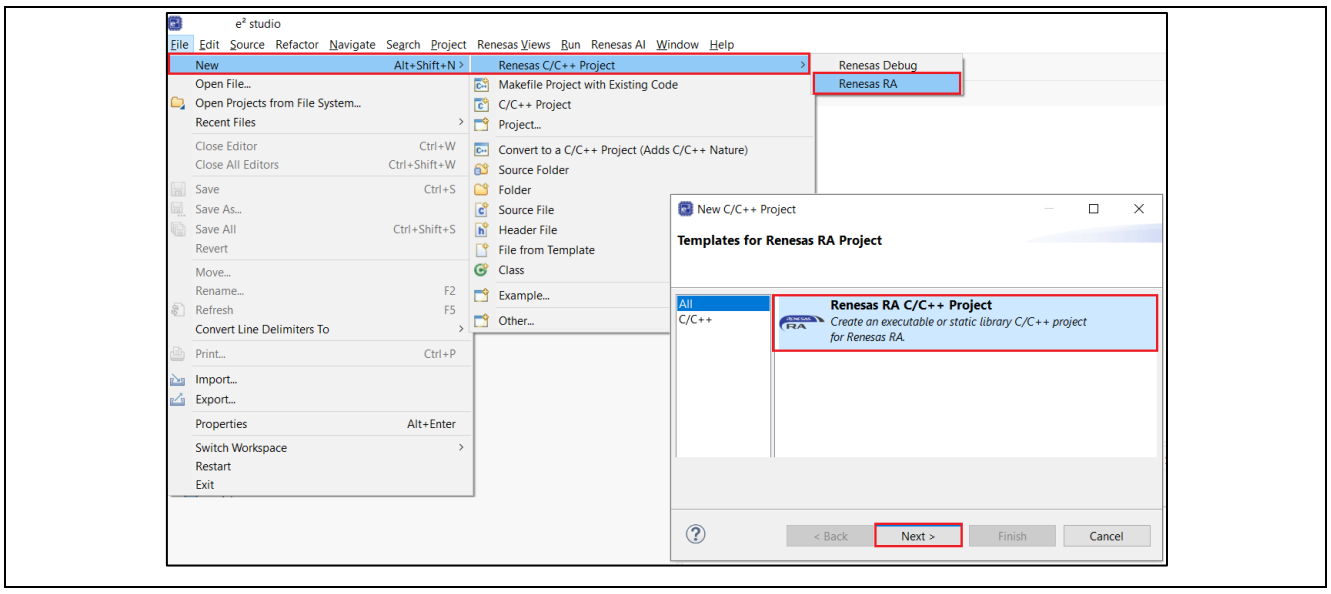

Figure 17. Create an EK-RA8P1 CPU1 Project using e² studio

Assign a name for this new project, “EK_RA8P1_CPU1”. Selecting from the Device and Tools Selection. Select Board type as EK-RA8P1, the core for this project is CPU1, and select the LLVM Embedded Toolchain for Arm.

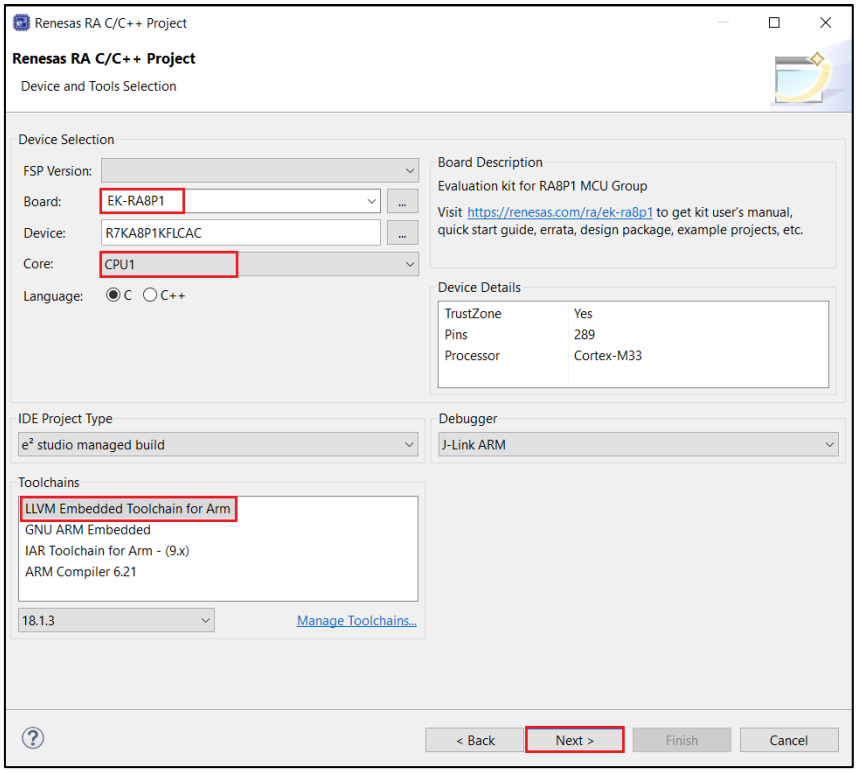

Figure 18. Selecting from the Device and Tools Selection CPU1

Select Flat (Non-TrustZone) Project in Project Type Selection and click Next.

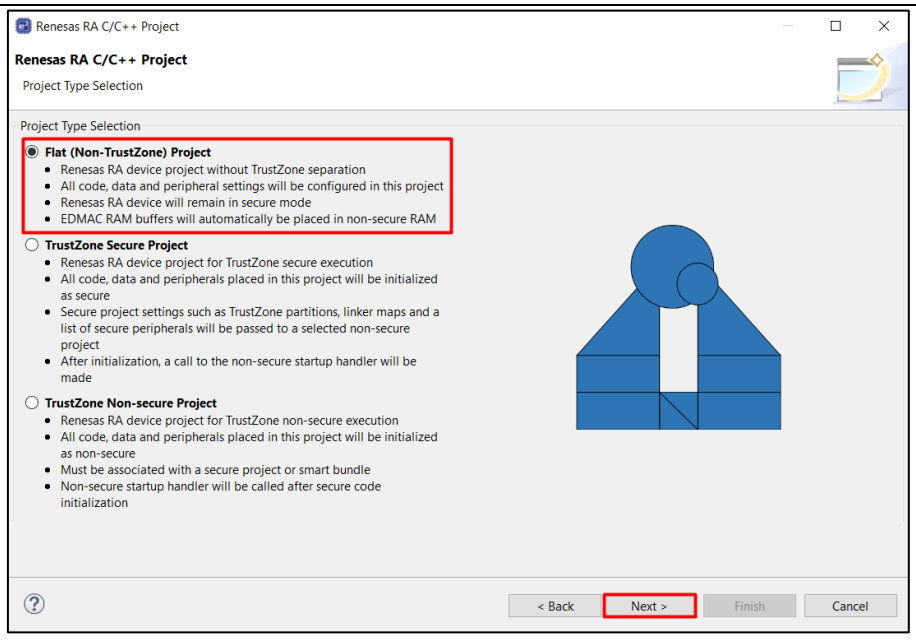

Figure 19. Flat Project Type Selection CPU1

Select the previous project or the Smart Bundle file in the Debug folder of the previously created and built CPU0 project.

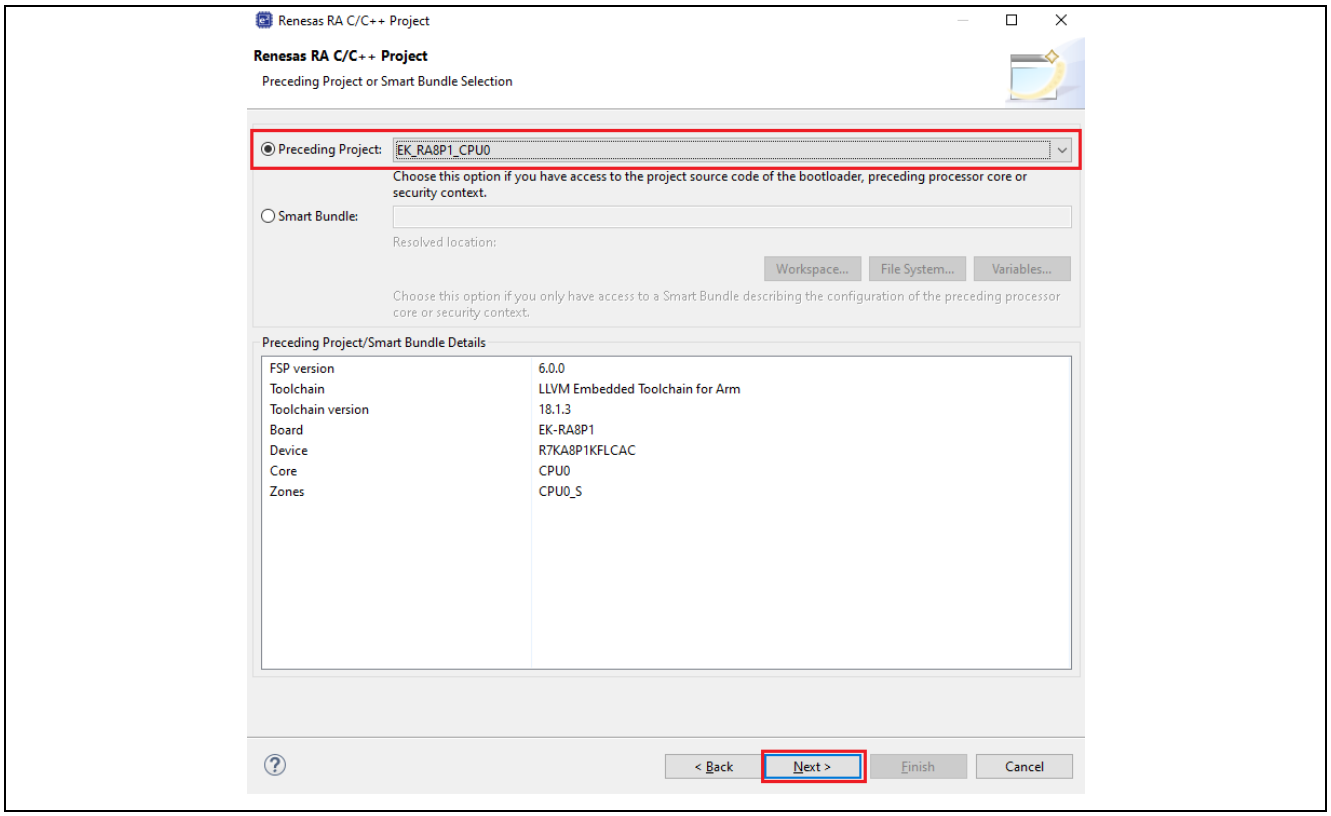

Figure 20. Preceding project or Smart Bundle Selection CPU1

Select Executable for Build Artifact Selection and No RTOS for RTOS Selection and click Next.

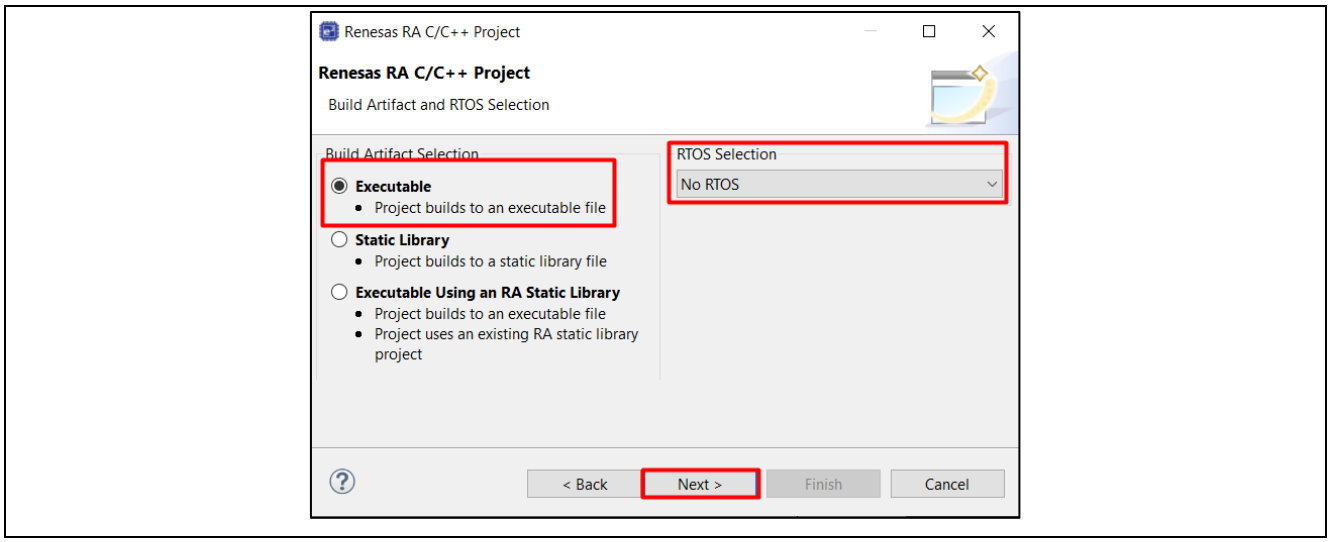

Figure 21. Executable for Build Artifact and No RTOS Selection

Select Bare Metal – Blinky for this example and click Finish.

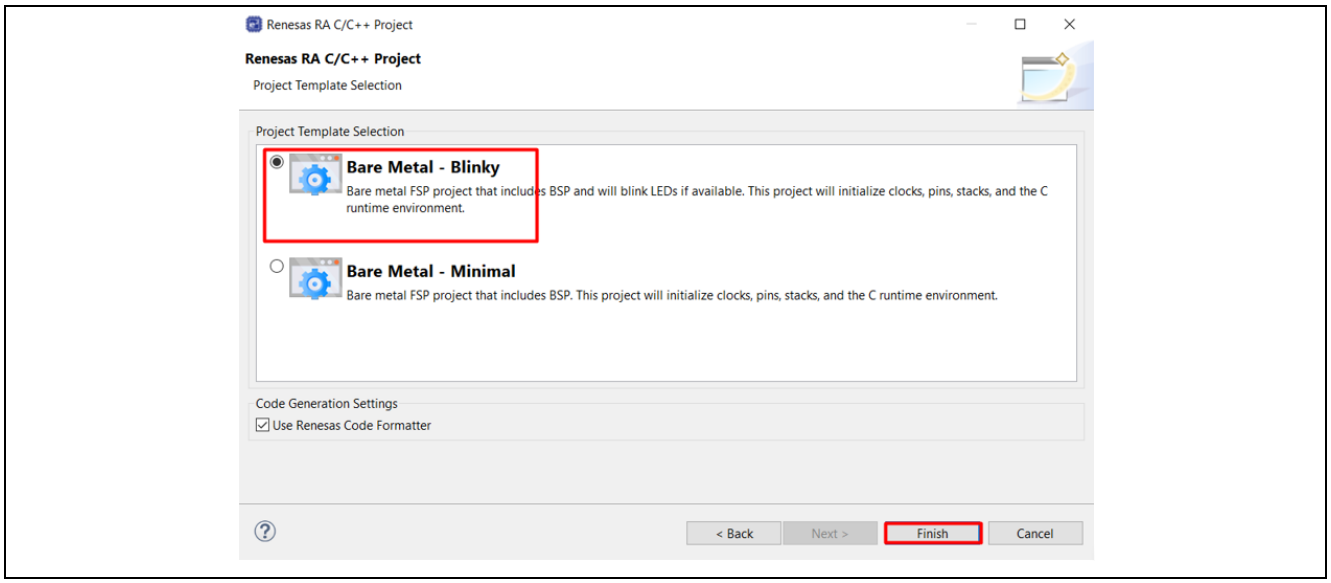

Figure 22. Blinky Project Template Selection CPU1

Generate Project Content and compile the project.

Double-click Configuration.xml to open the configurator. Click Generate Project Content as shown in Figure 23.

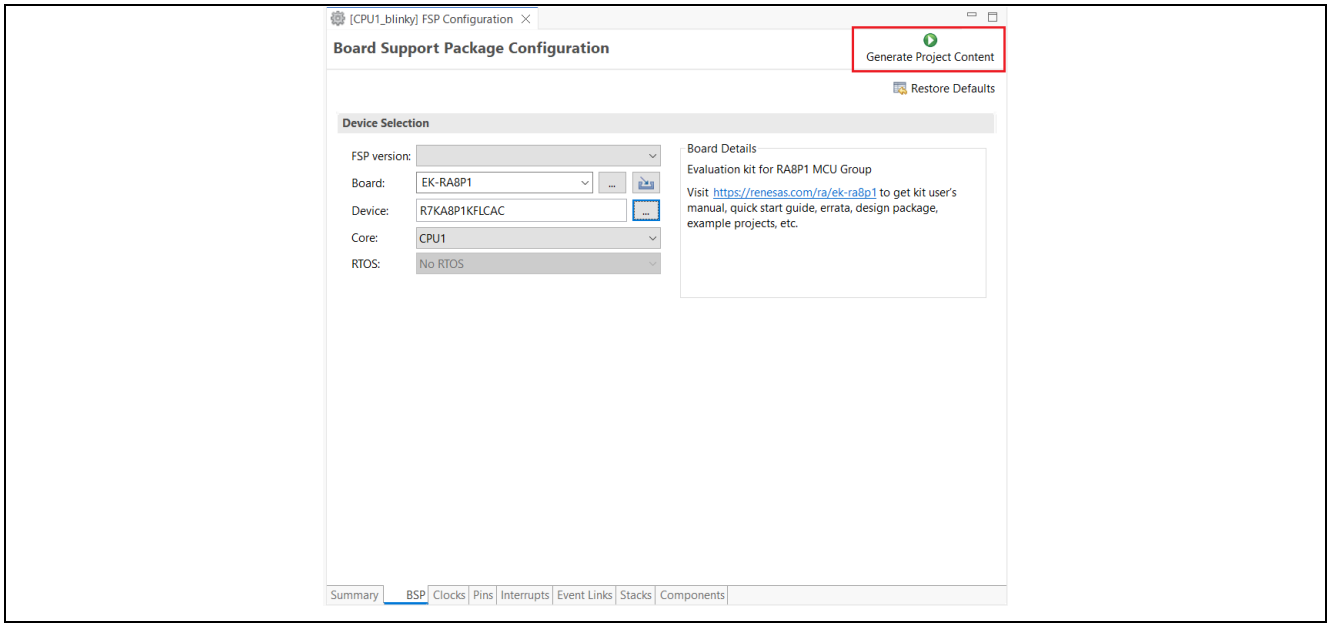

Figure 23. Generate Project Content CPU1

Right-click on the project and select the Build Project.

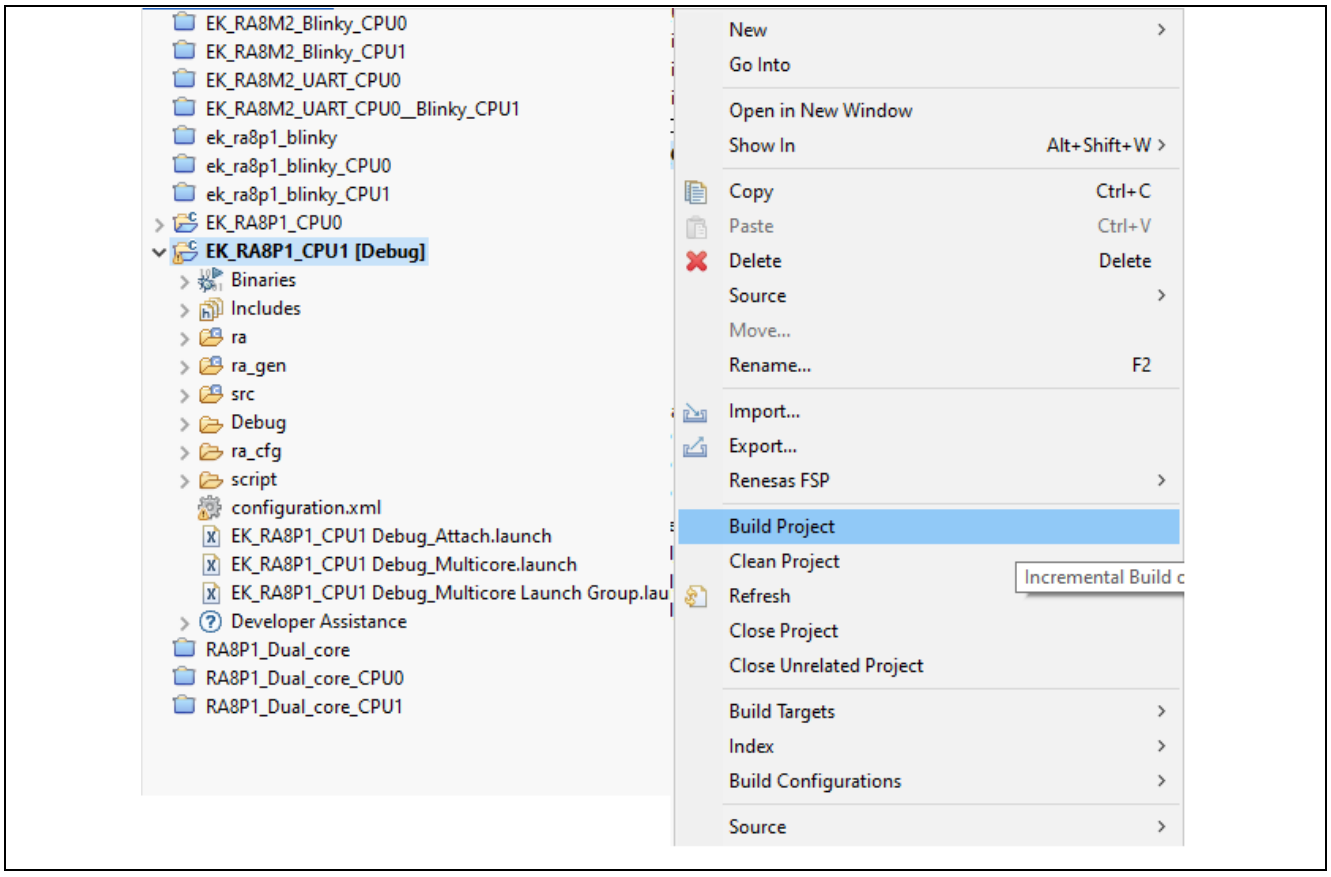

Figure 24. Compile the Template Project CPU1

After the building of projects for both CPU0 and CPU1 successfully, make sure the image was generated in Debug folder.

2.3 Procedure for Creating a Solution Project for Dual-Core and its Configuration

This section outlines the detailed steps for creating a solution-based approach for a dual-core project on the RA8P1 MCU series. If needed, refer to the FSP documentation for additional guidance.

During project creation, you will:

- Choose the project type
- Specify the project name and location.
- Configure essential settings, including:
  - FSP version
  - Target board
  - Toolchain version

These instructions will walk you through the complete process using a simple Blinky application as an example, ensuring you understand how to set up and configure a functional dual-core project.

Open e² studio, then navigate to File > New > Renesas C/C++ Project > Renesas RA.

Select Renesas RA FSP Solution and click Next to proceed.

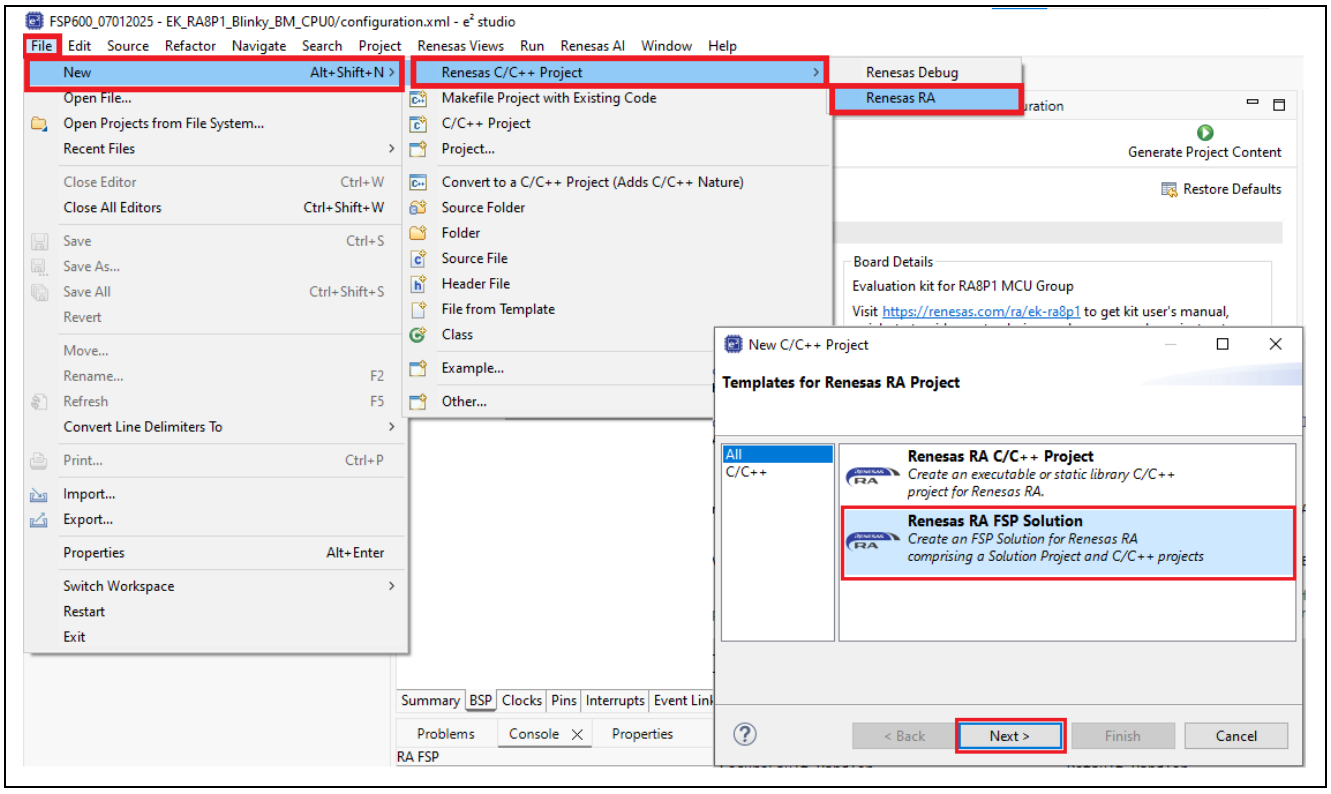

Figure 25. Create an EK-RA8P1 CPU0 Solution Project using e² studio

Enter a name for your new project “EK_RA8P1_SP” and click Next.

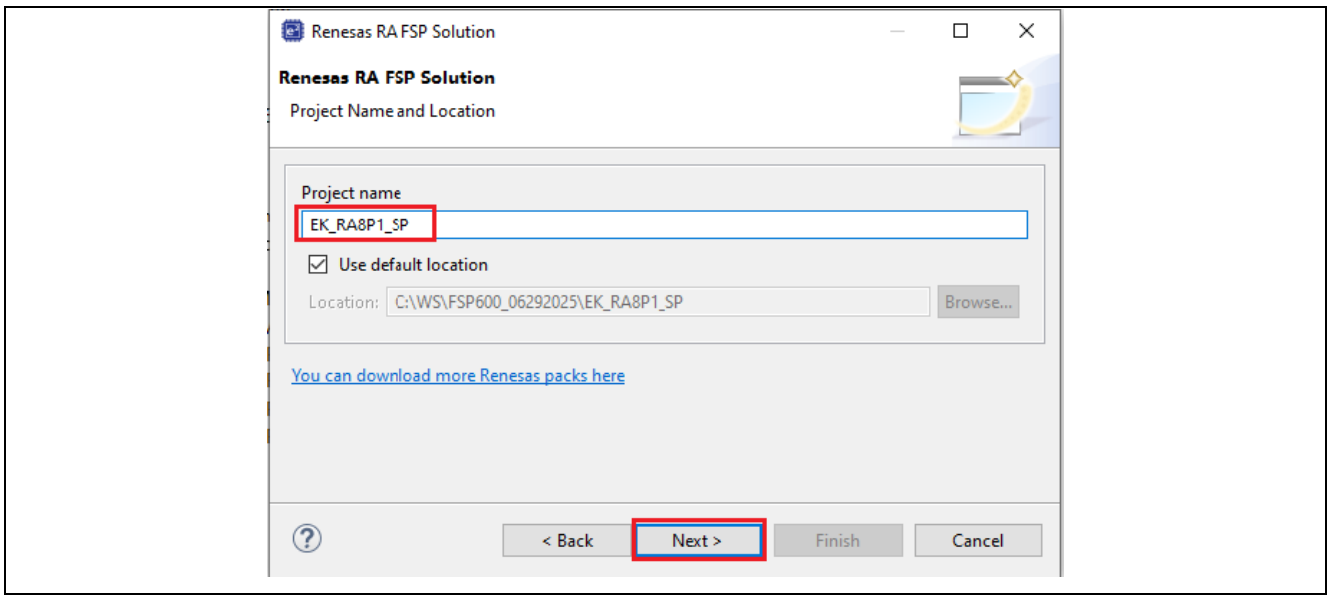

Figure 26. Solution Project Name for EK-RA8P1 using e² studio

In the Device and Tools Selection window:

- Set the board to EK-RA8P1.
- Language : C
- Select the LLVM Embedded Toolchain for Arm as the toolchain.
- Debugger : J-Link ARM

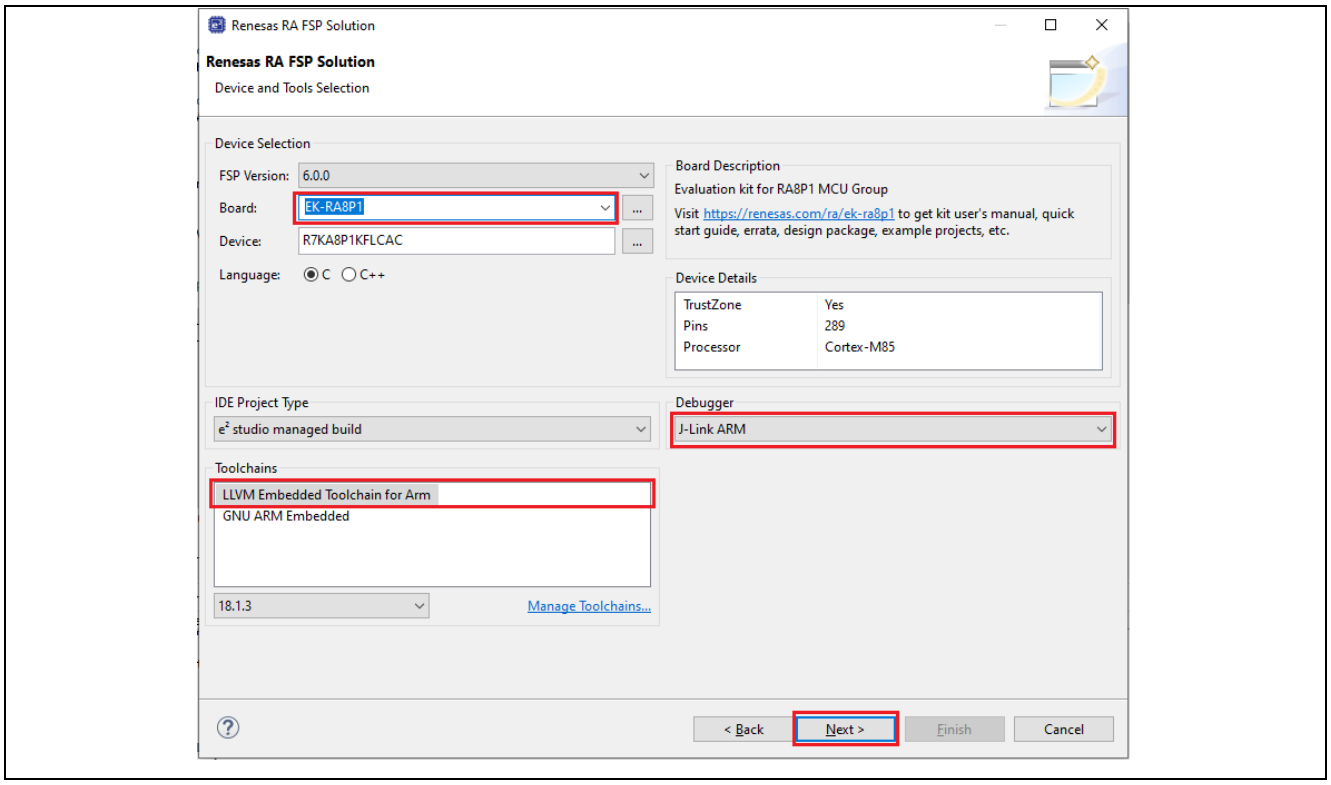

Figure 27. Selecting from the Device and Tools Selection

Select Bare Metal – Blinky for this example and click Finish.

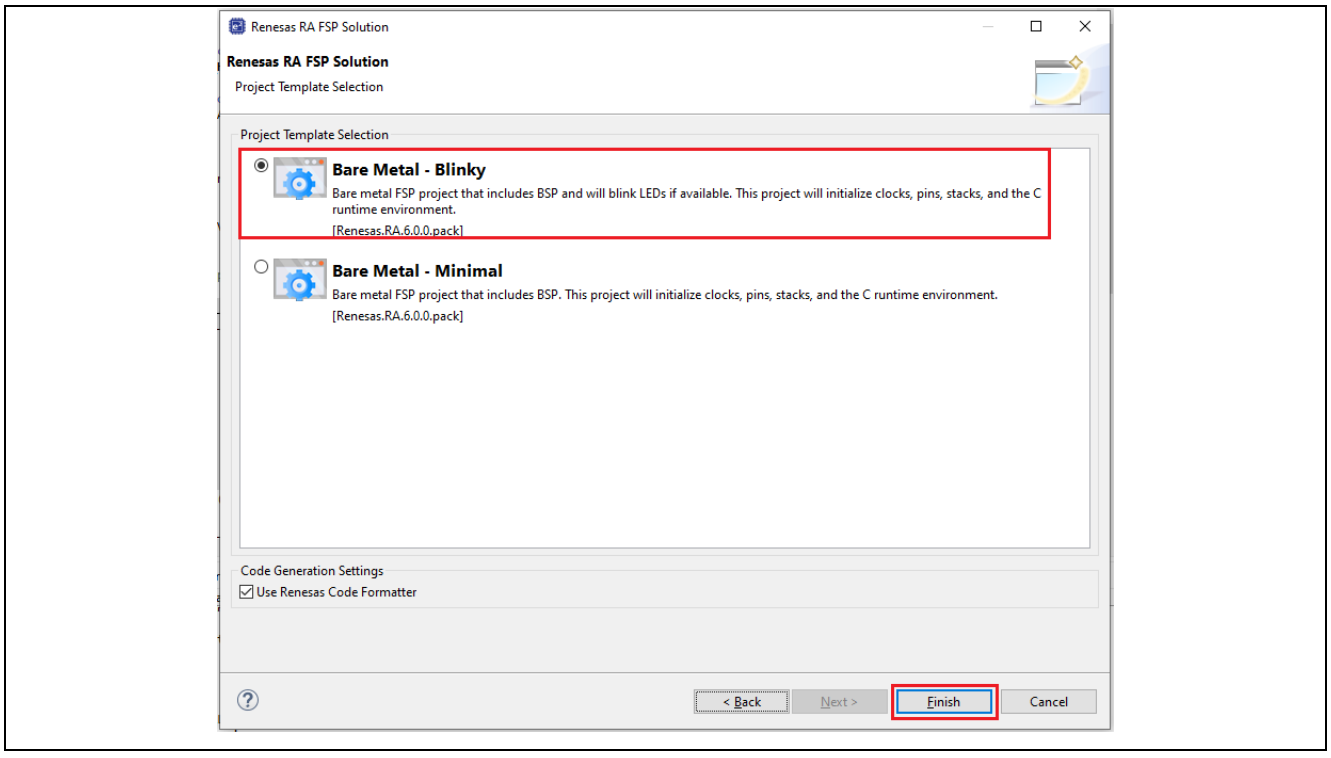

Figure 28. Blinky Project Template for Solution Project

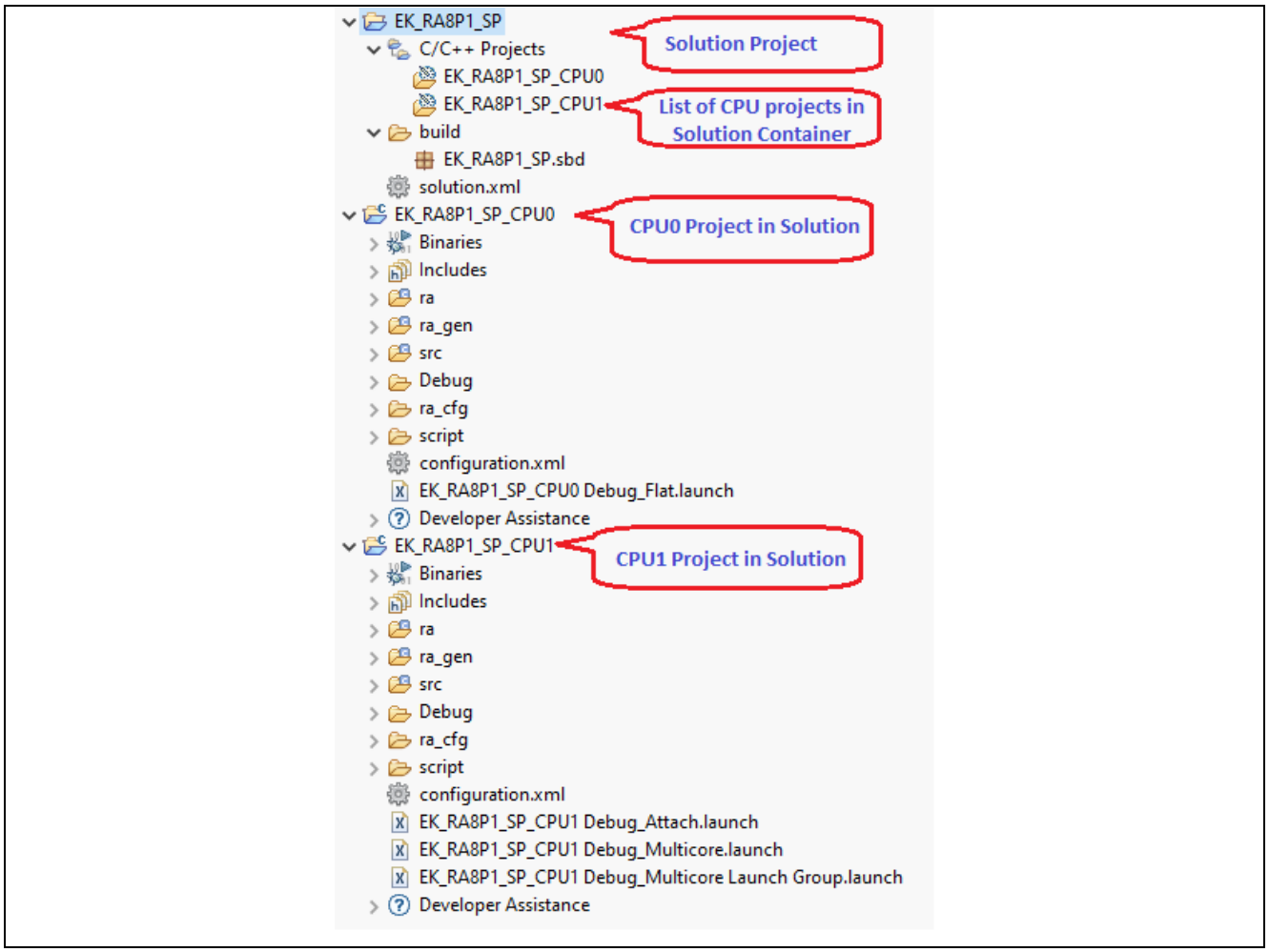

Figure 29. Blinky Solution Project creation

After clicking on the Finish button in the wizard, this will cause a set of projects to be created, with appropriate linkages between them. Two of the projects, “EK_RA8P1_SP_CPU0” and “EK_RA8P1_SP_CPU1,” are “normal” projects that contain source code and combined will generate the overall multicore application that will be downloaded to the target MCU, with the appropriate code running on each CPU. The third project is the “container” Solution project “EK_RA8P1_SP”, which contains and controls configuration of the overall MCU and the CPU projects.

Note: After the project is created, an initial automated build of both CPU projects will be performed to populate certain configuration data stored in the central solution project. The snapshot of the automated compilation as part of the solution project creation is shown in Figure 30.

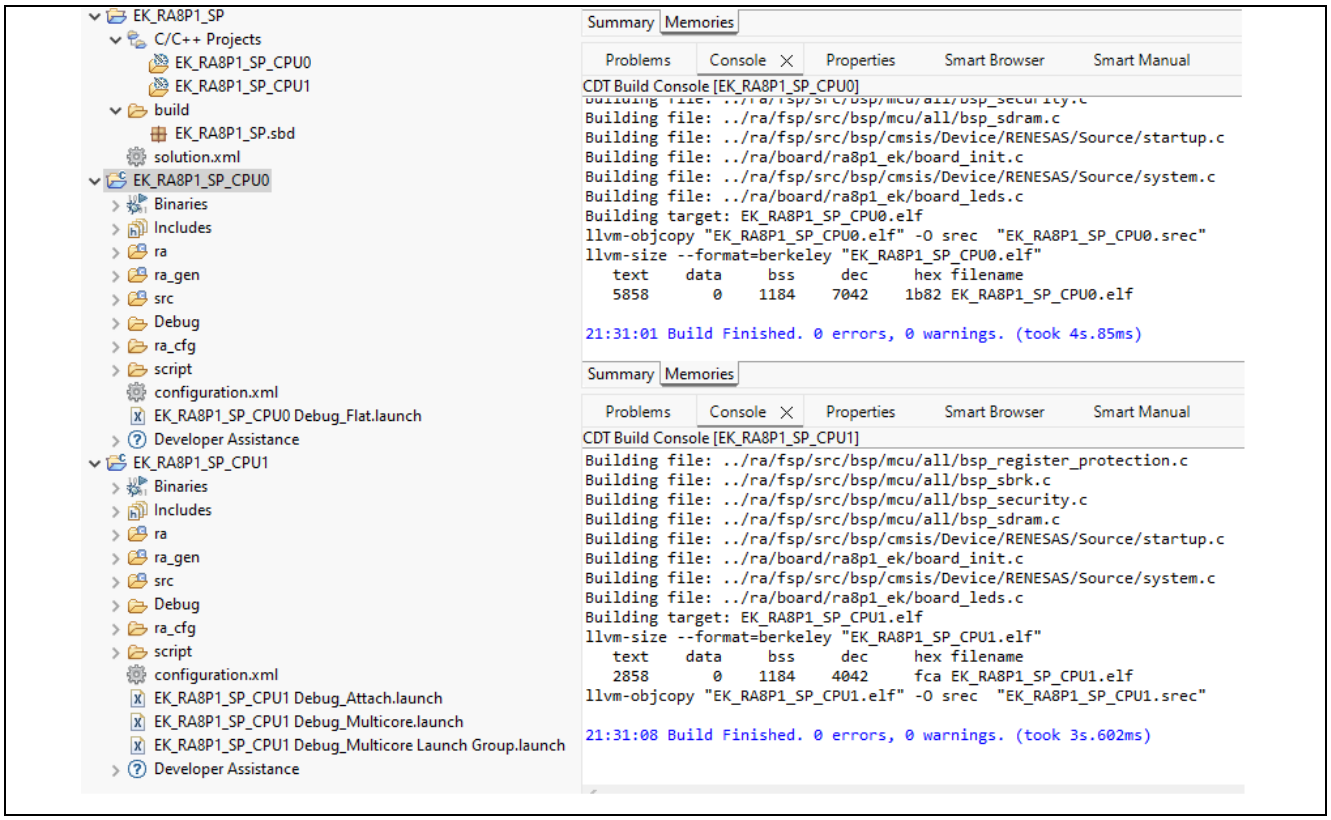

Figure 30. Snapshot of Compiled project for CPU0 and CPU1 after solution project creation

Note: With this approach, users can adopt an RA solution-based method to create dual-core projects for CPU0 and CPU1. This approach is easier and more user-friendly. The solution project generated above serves as a template for the Renesas RA Dual-Core; however, in actual application development, the "Bare Metal - Minimal" option can be selected instead of "Bare Metal - Blinky." FSP modules required for the application can then be added on top of the template project. Further details on adding FSP modules are provided in the FSP User Manual, which users are encouraged to refer to for comprehensive guidance.

2.4 Procedure to Create Dual-Core Projects Using FreeRTOS for Both Cores

Creating RTOS-based projects on dual-core has the advantage that separate instances of RTOS will be running on two cores. In this app note we will be covering how the FreeRTOS-based projects are created. This section outlines the detailed steps for creating a dual-core project using FreeRTOS on the RA8P1 MCU series.

During project creation, you will:

- Choose the project type
- Specify the project name and location
- Configure essential settings, including:
    - FSP version
    - Target board
    - Selected core (Cortex-M85 or Cortex-M33)
    - RTOS inclusion - FreeRTOS
    - Toolchain version

These instructions will walk you through the complete process using a simple Blinky application as an example, ensuring you understand how to set up and configure a functional dual-core project.

Open e² studio, then navigate to File → New → Renesas C/C++ Project → Renesas RA.

Select Renesas RA C/C++ Project and click Next to proceed.

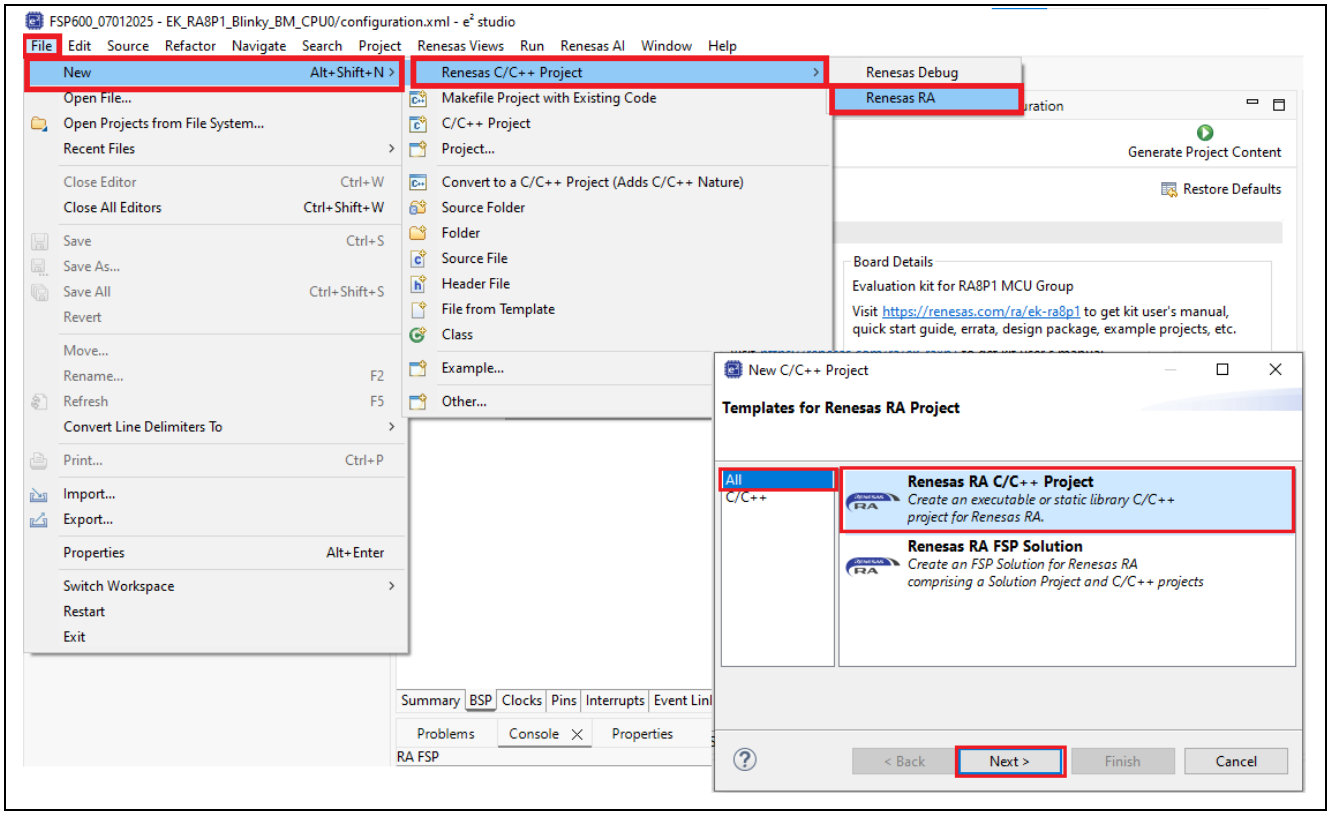

Figure 31. Create an EK-RA8P1 CPU0 Project using e² studio

Enter a name for your new project “EK_RA8P1_FREERTOS_CPU0” and click Next.

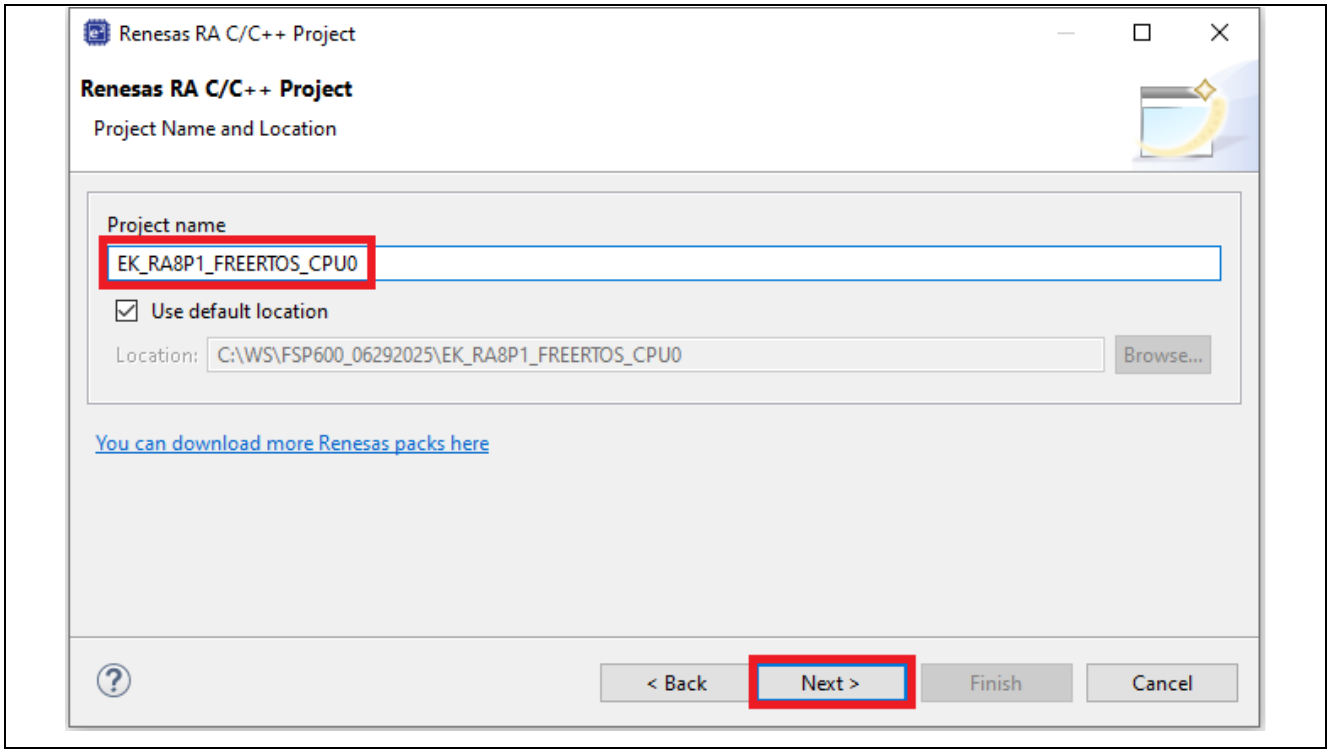

Figure 32. Project Name for FreeRTOS based EK-RA8P1 CPU0 Project using e² studio

In the Device and Tools Selection window:

- Set the board to EK-RA8P1.
- Choose CPU0 (Cortex-M85) as the target core.
- Select the LLVM Embedded Toolchain for Arm as the toolchain.

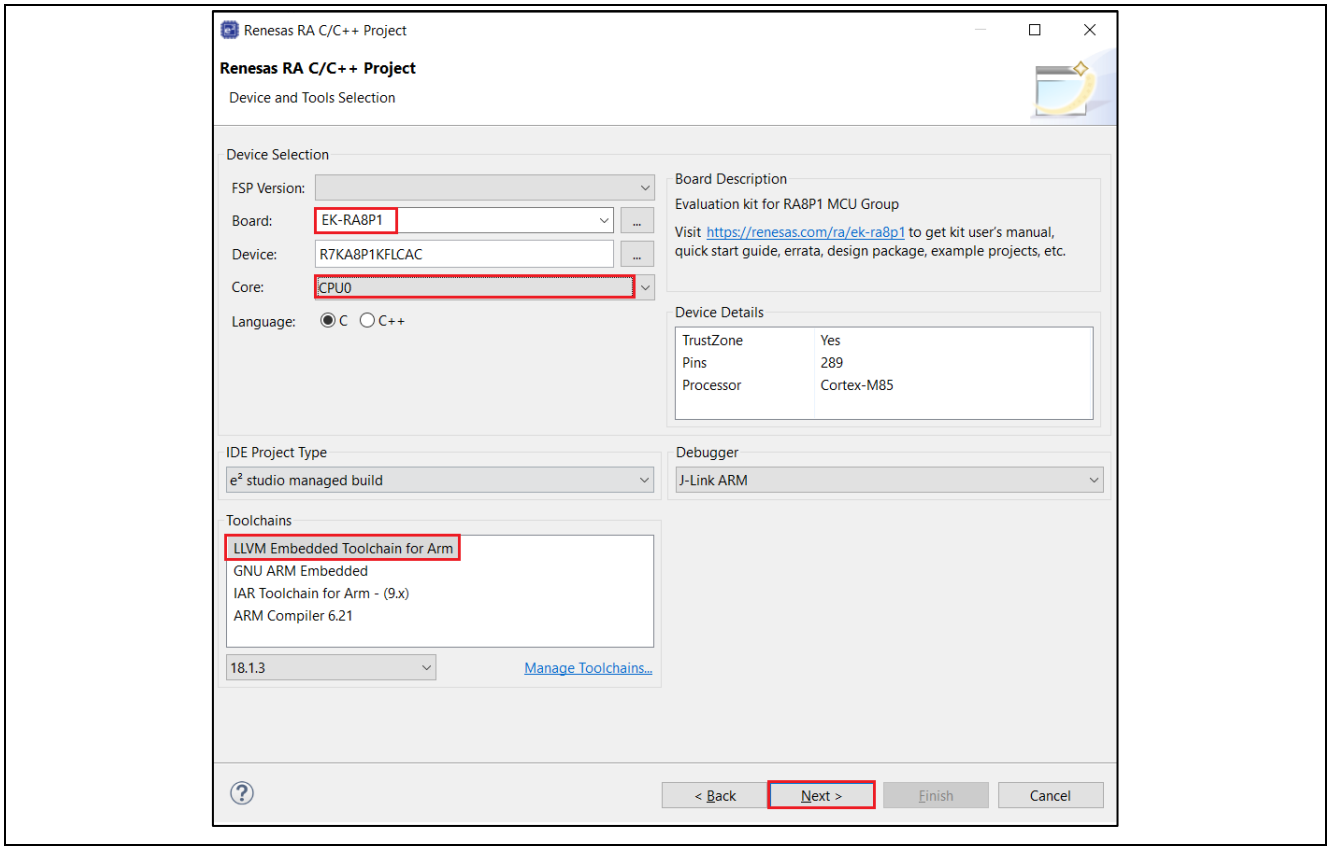

Figure 33. Selecting from the Device and Tools Selection CPU0

Select Flat (Non-TrustZone) Project in Project Type Selection and click Next.

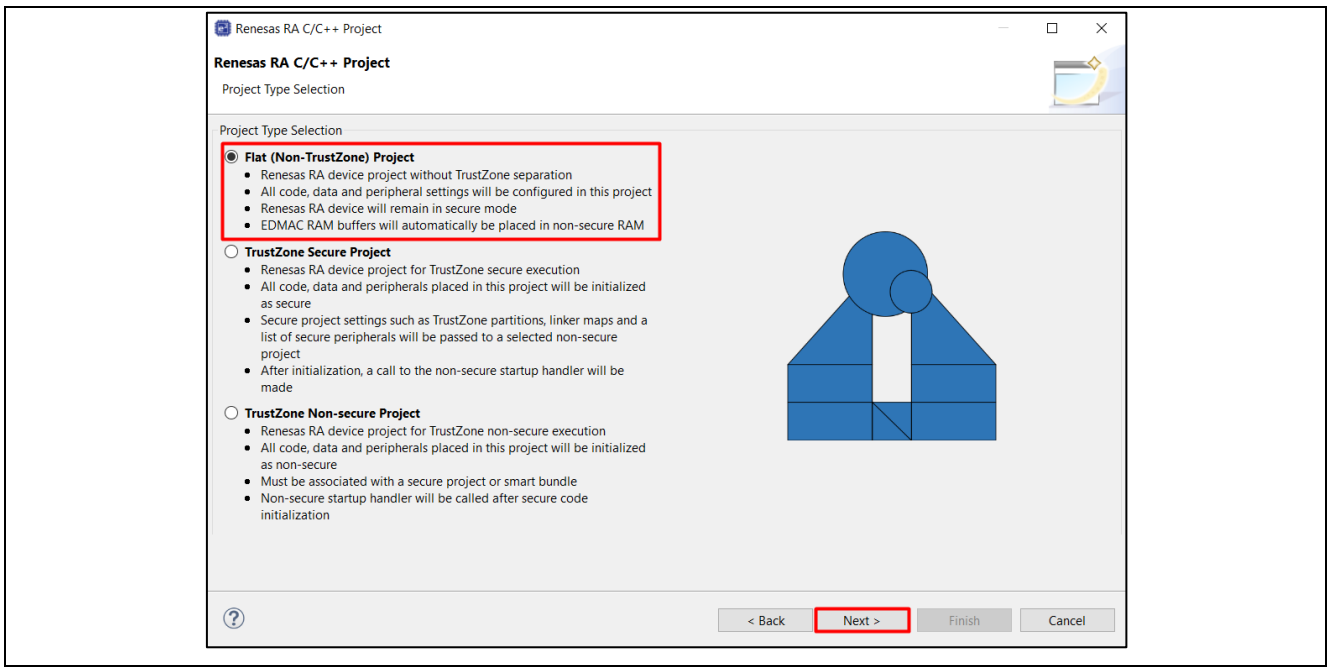

Figure 34. Flat (Non-TrustZone) Project Type Selection

Select None for Preceding Project or Smart Bundle Selection and click Next.

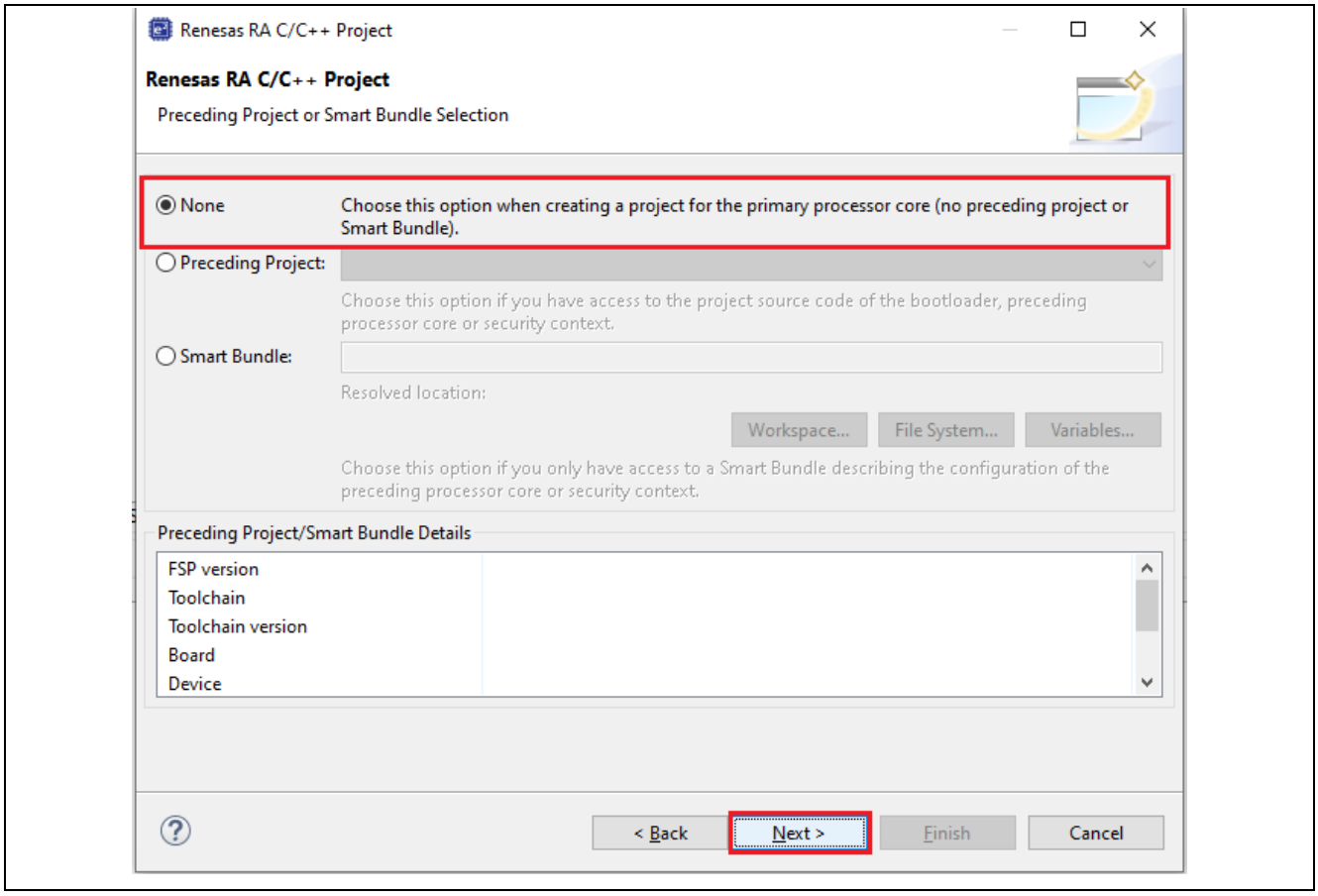

Figure 35. No preceding project or Smart Bundle Selection

Select Executable for Build Artifact Selection and No RTOS for RTOS Selection and click Next.

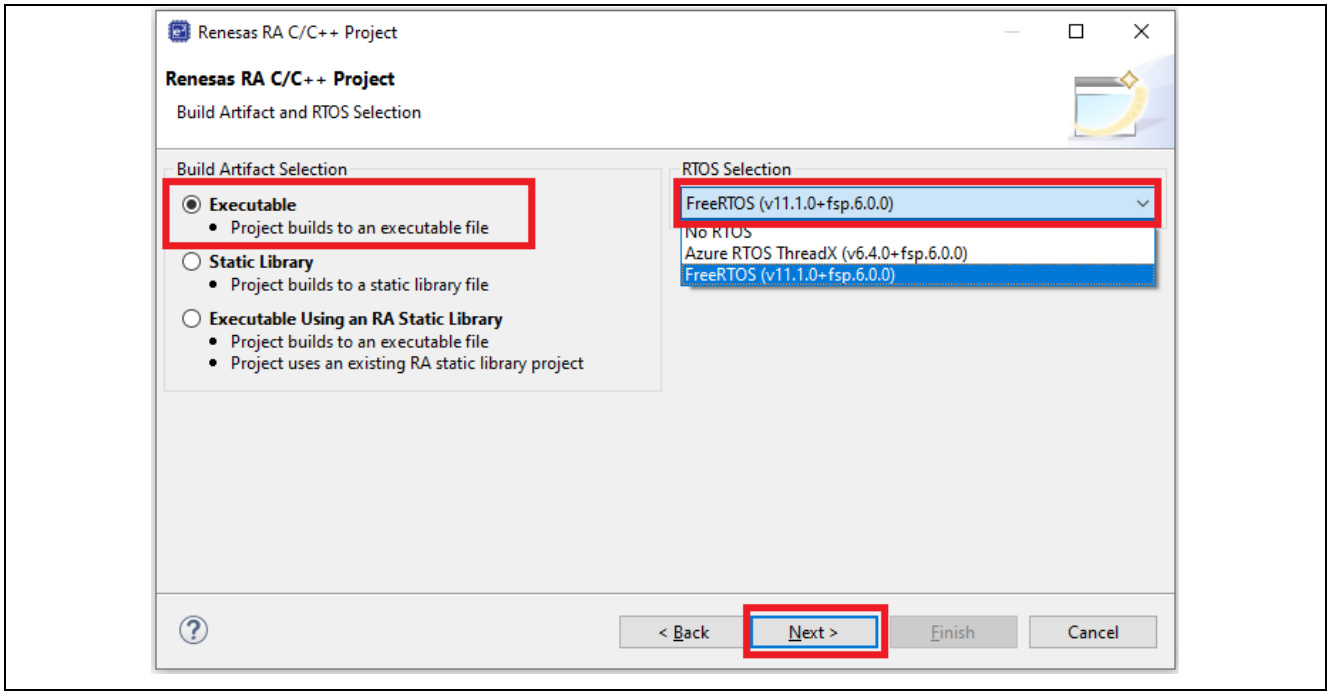

Figure 36. Snapshot of RTOS Selection

Select Bare Metal – Blinky for this example and click Finish.

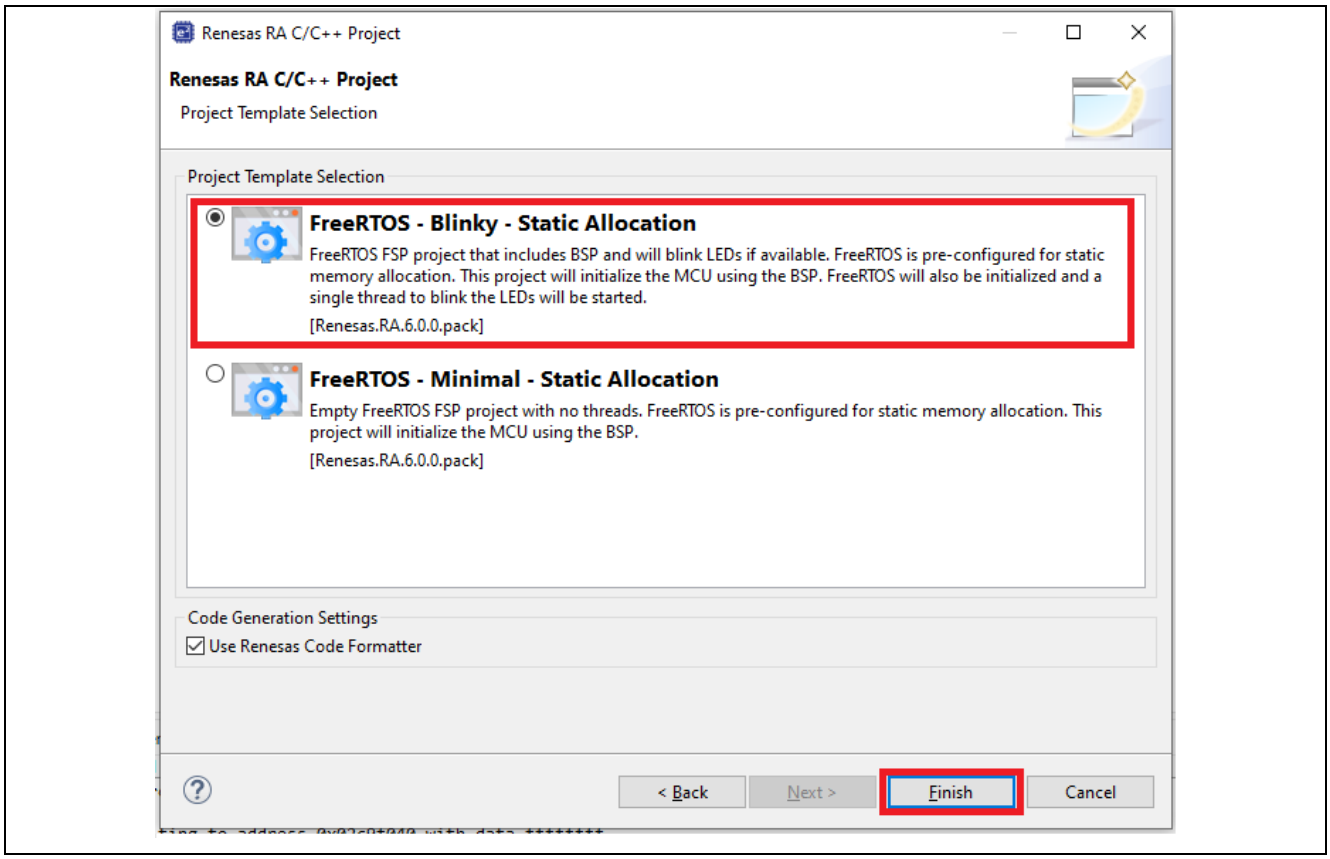

Figure 37. FreeRTOS Blinky Project Template Selection

Generate Project Content and compile the project template.

Double-click Configuration.xml to open the configurator. Click Generate Project Content as shown in Figure 38.

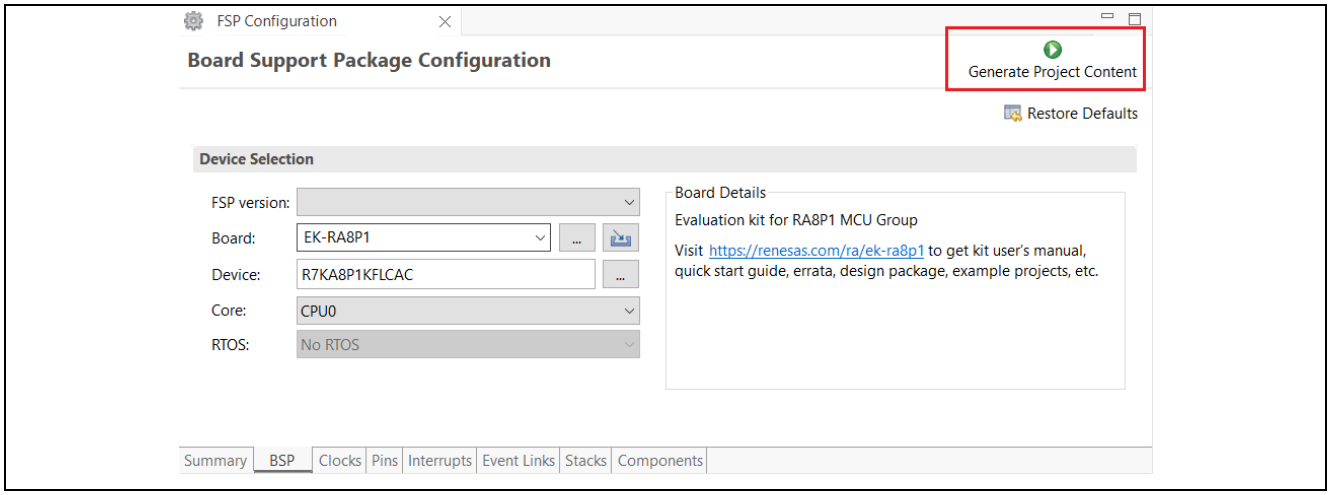

Figure 38. Generate Project Content

Note: As a workaround, add the MACRO setting to define under the project settings → C/C++ Build → Settings → Compiler → Includes → Macro Defines (-D) → "BSP_PARTITION_FLASH_CPU1_S_START" as shown in the snapshot below to avoid the compiler/linker error.

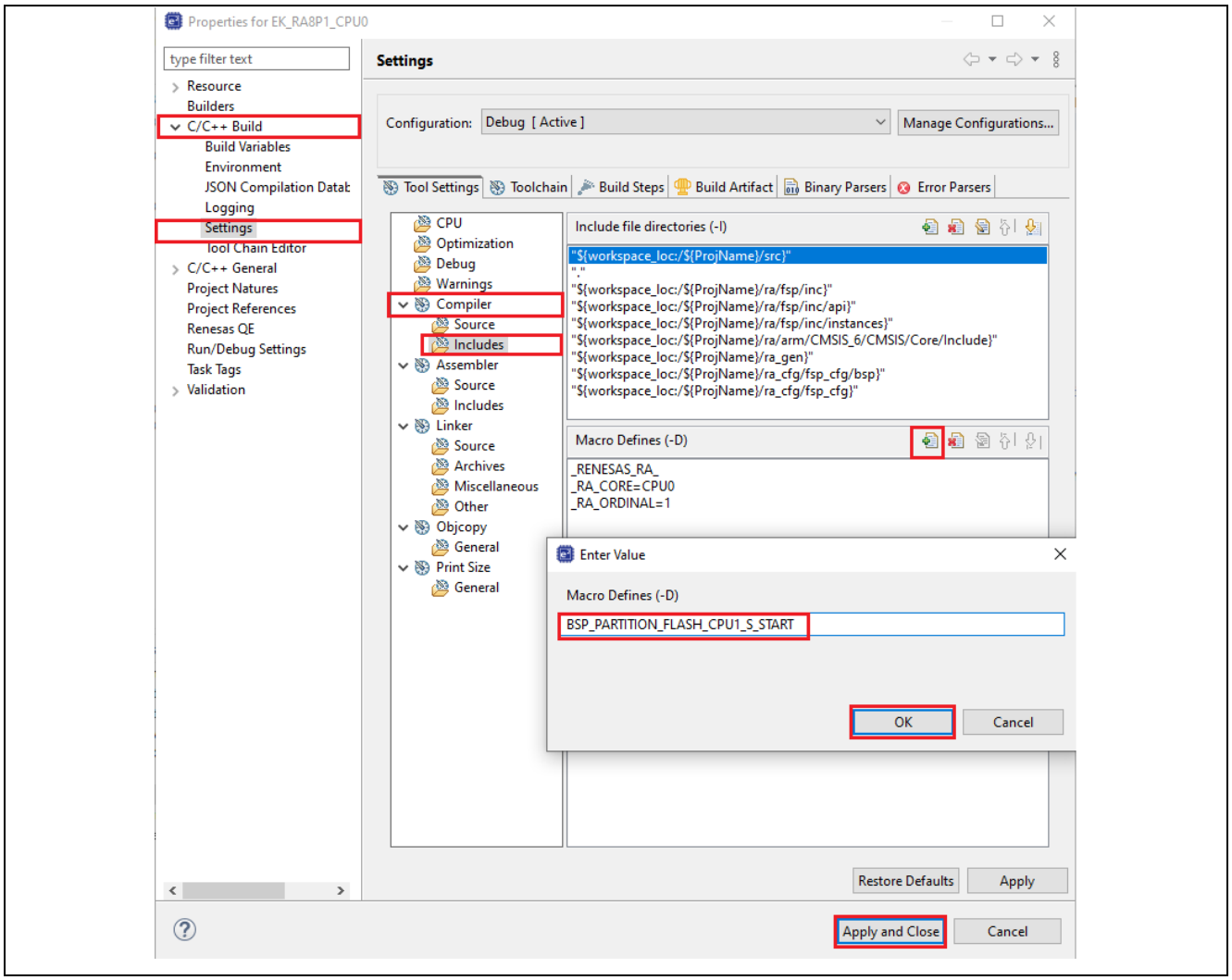

Figure 39. Add the MACRO under the project settings - CPU0 Blink Project

Right-click on the project and select the Build Project.

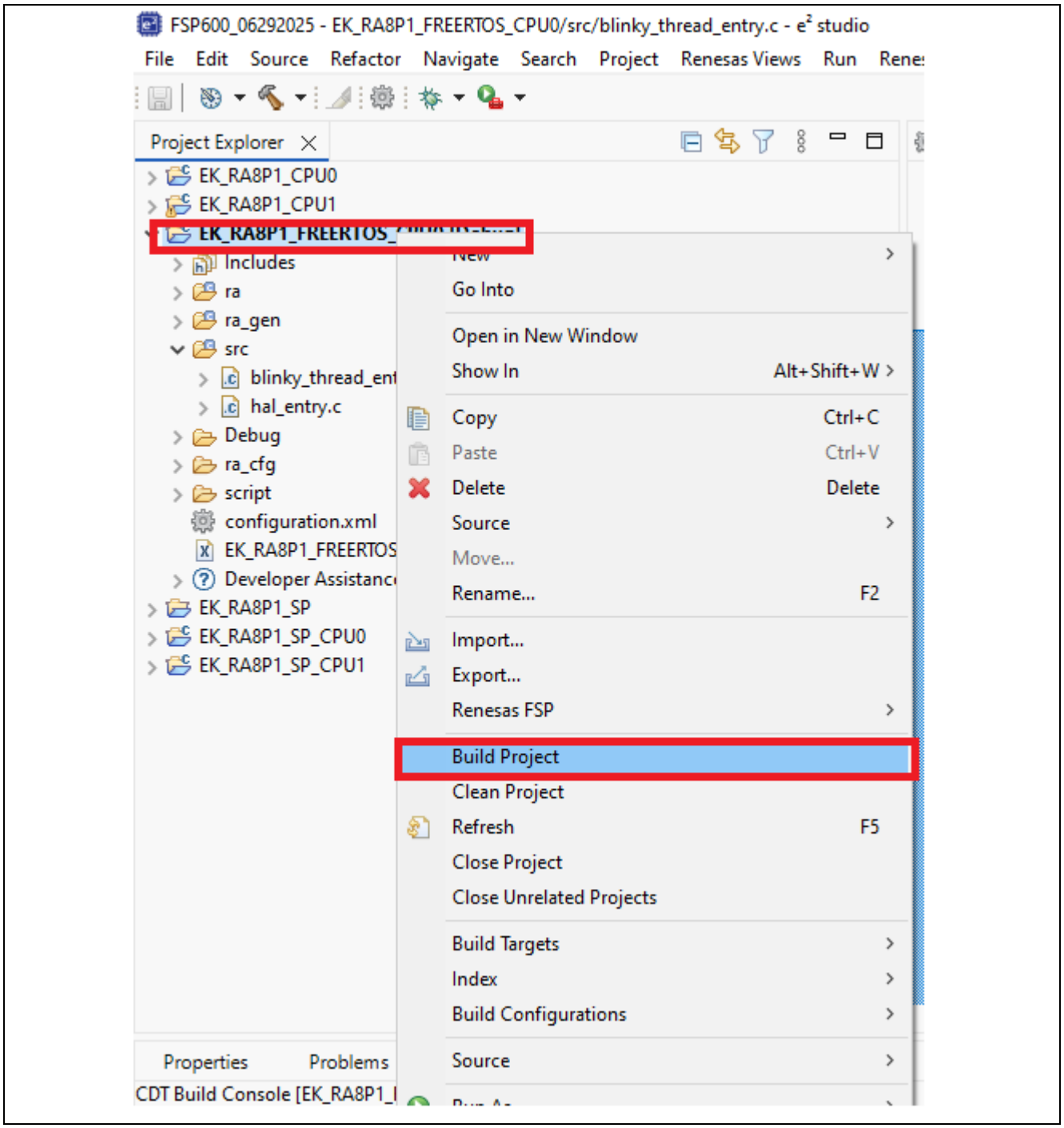

Figure 40. Compile the EK_RA8P1_CPU0 Blinky Project

After successful compilation, the Smart bundle.sbd is generated as shown in Figure 41.

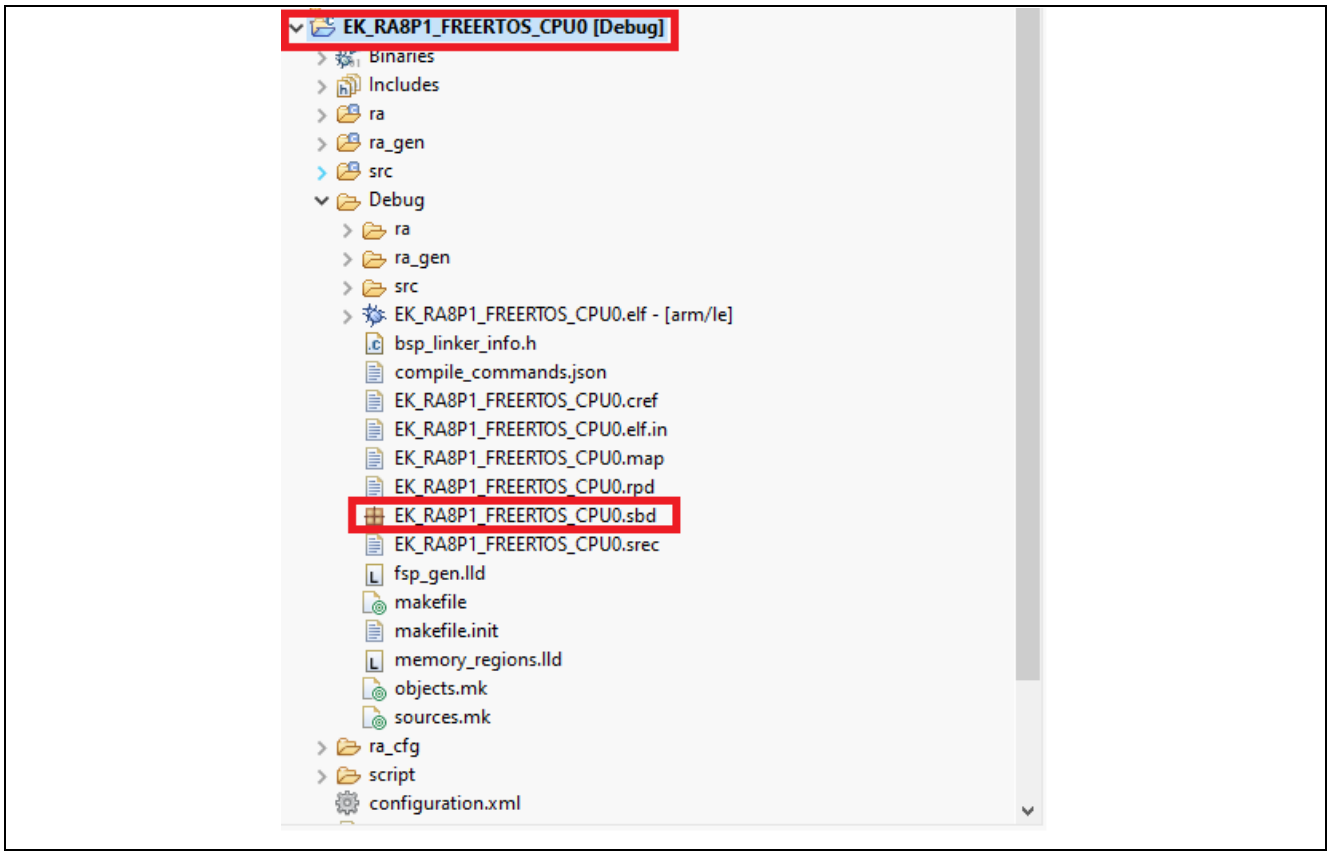

Figure 41. Smart Bundle Generated

Create a FreeRTOS Blinky Project with the CM33 core.

Click File → New → Renesas C/C++ Project → Renesas RA and select Renesas RA C/C++ Project → Click Next.

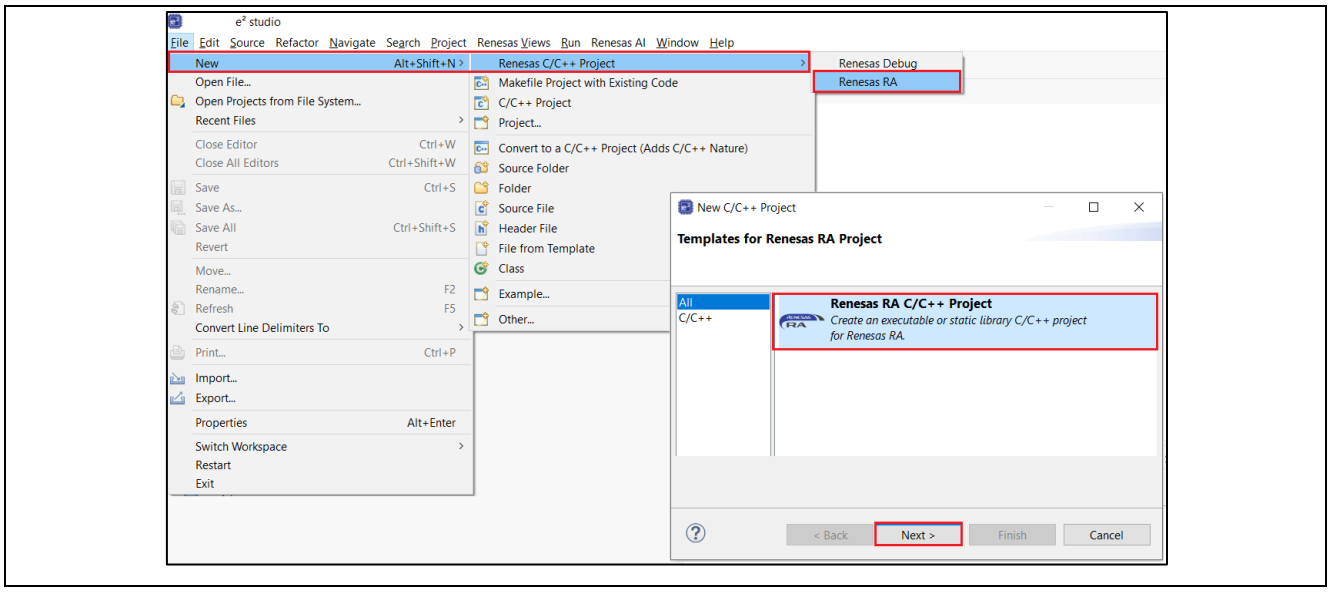

Figure 42. Create an EK-RA8P1 CPU1 Project using e² studio

Assign a name for this new project: “EK_RA8P1_FREERTOS_CPU1”. Selecting from the Device and Tools Selection. Select Board type as EK-RA8P1, the core for this project is CPU1, and select the LLVM Embedded Toolchain for Arm.

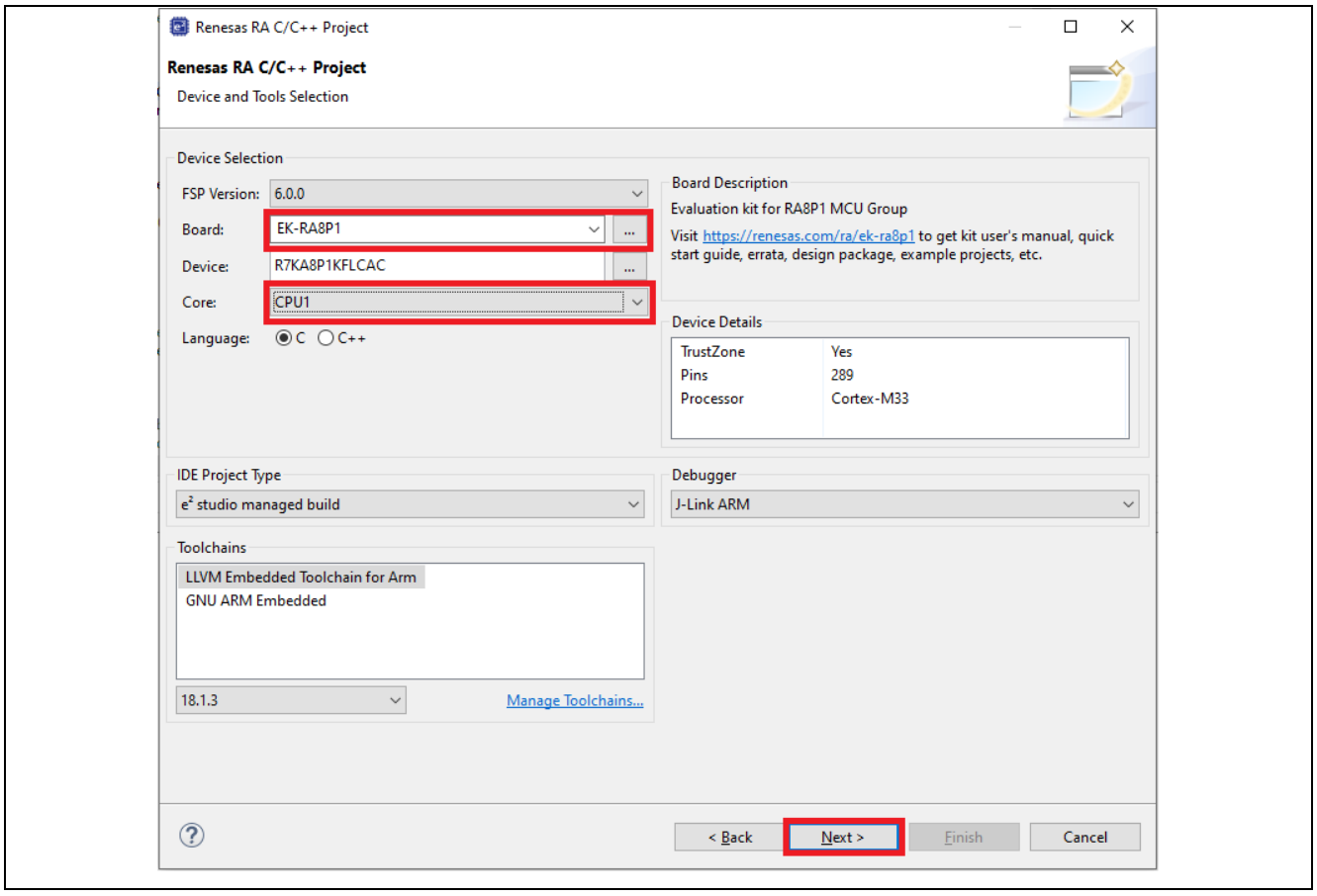

Figure 43. Selecting from the Device and Tools Selection CPU1

Select Flat (Non-TrustZone) Project in Project Type Selection and click Next.

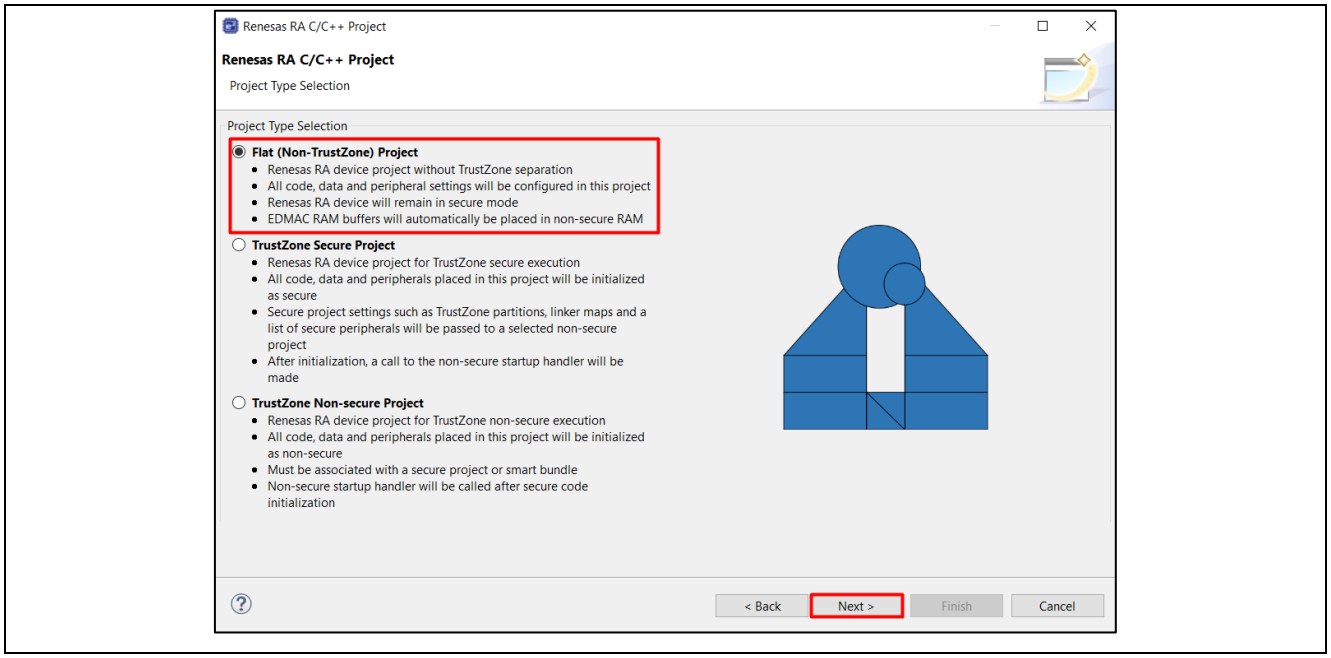

Figure 44. Flat Project Type Selection CPU1

Select the previous project or the Smart Bundle file in the Debug folder of the previously created and built CPU0 project.

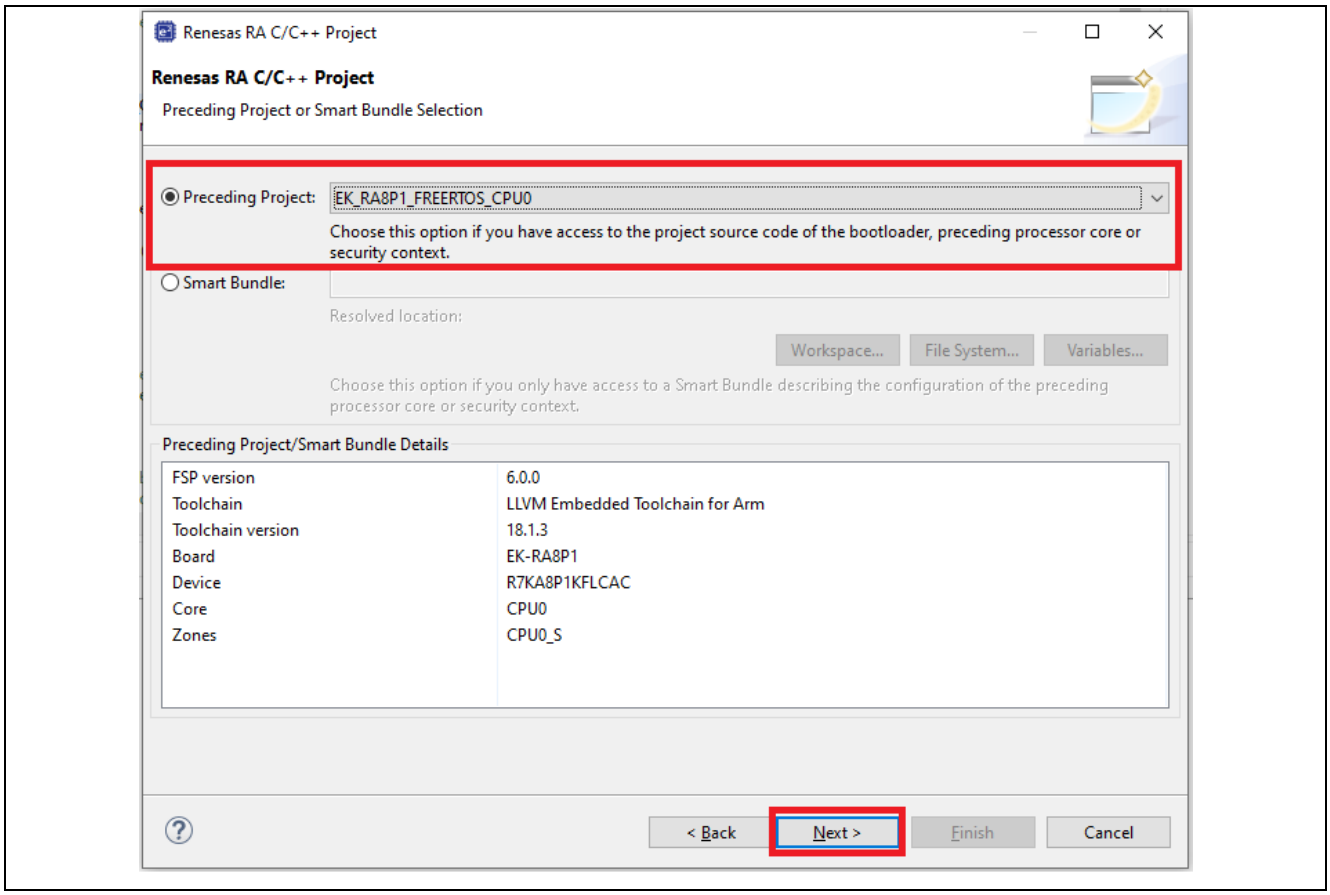

Figure 45. Preceding project or Smart Bundle Selection CPU1

Select Executable for Build Artifact Selection and FreeRTOS for RTOS Selection and click Next.

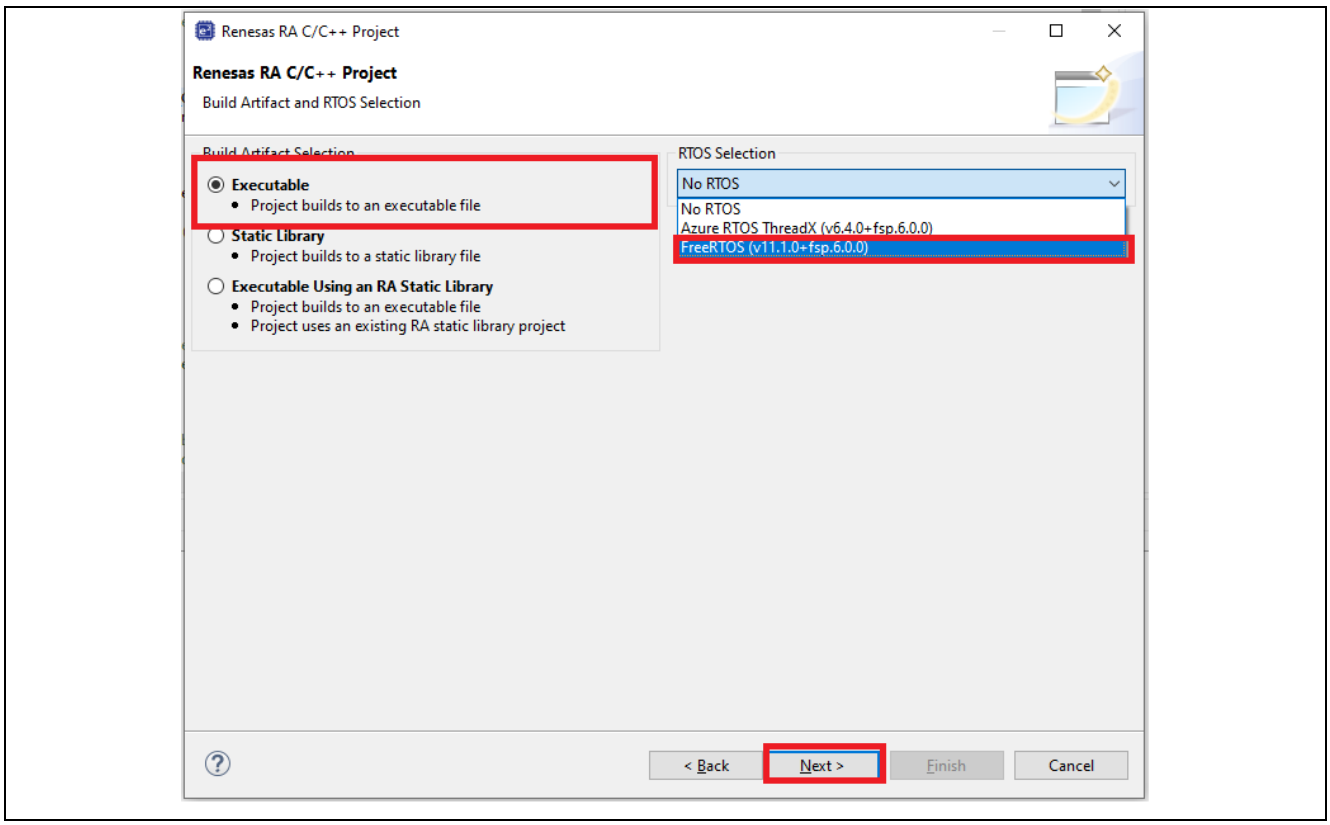

Figure 46. Executable for Build Artifact and No RTOS Selection

Select FreeRTOS - Blinky - Static Allocation for this example and click Finish.

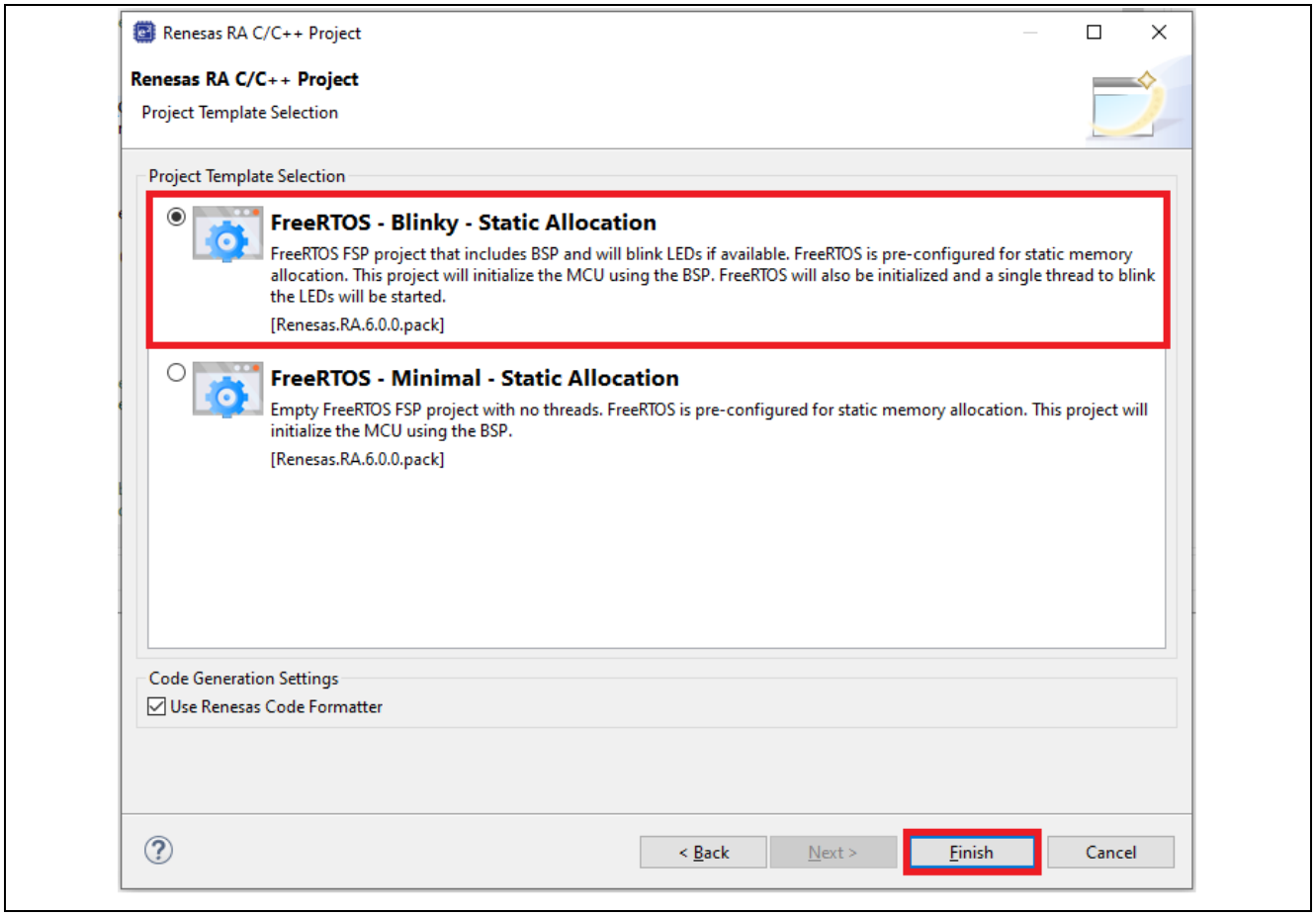

Figure 47. Blinky Project Template Selection CPU1

Generate Project Content and compile the project template.

Double-click Configuration.xml to open the configurator. Click Generate Project Content as shown in Figure 48.

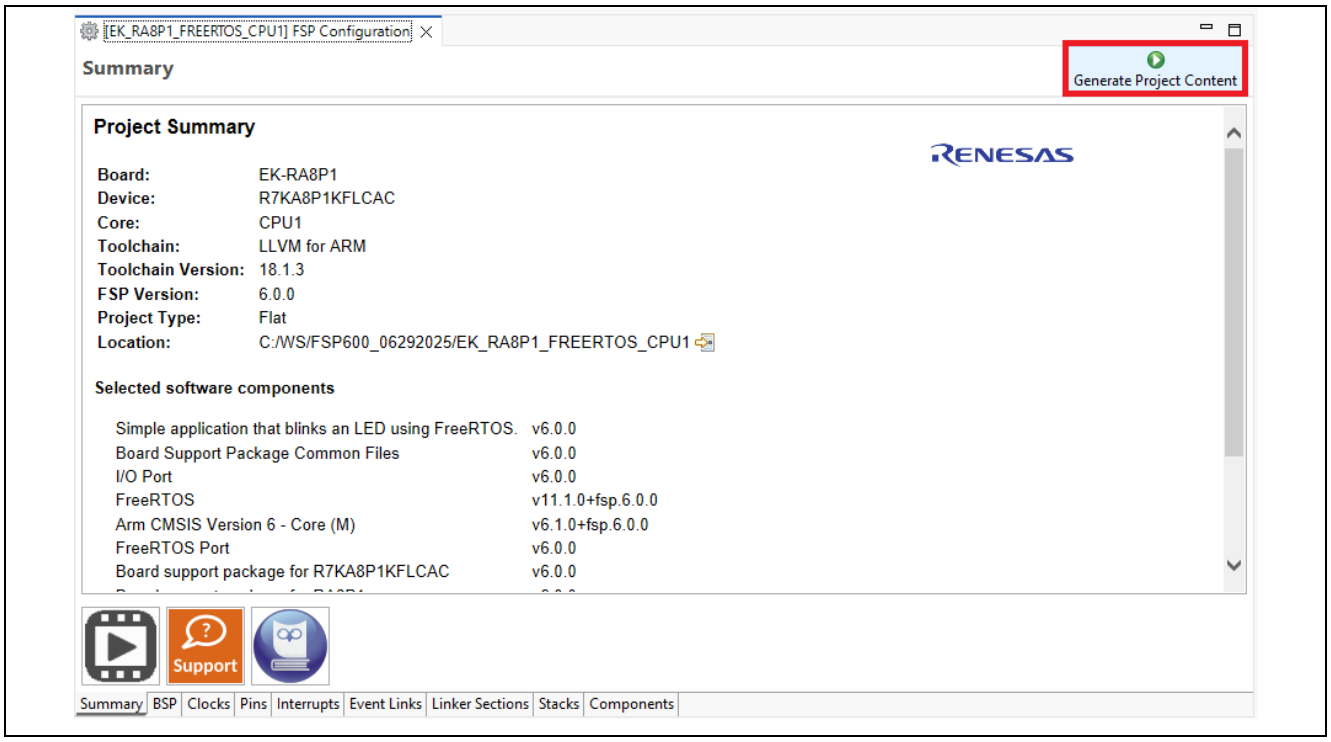

Figure 48. Generate Project Content CPU1

Right-click on the project and select the Build Project.

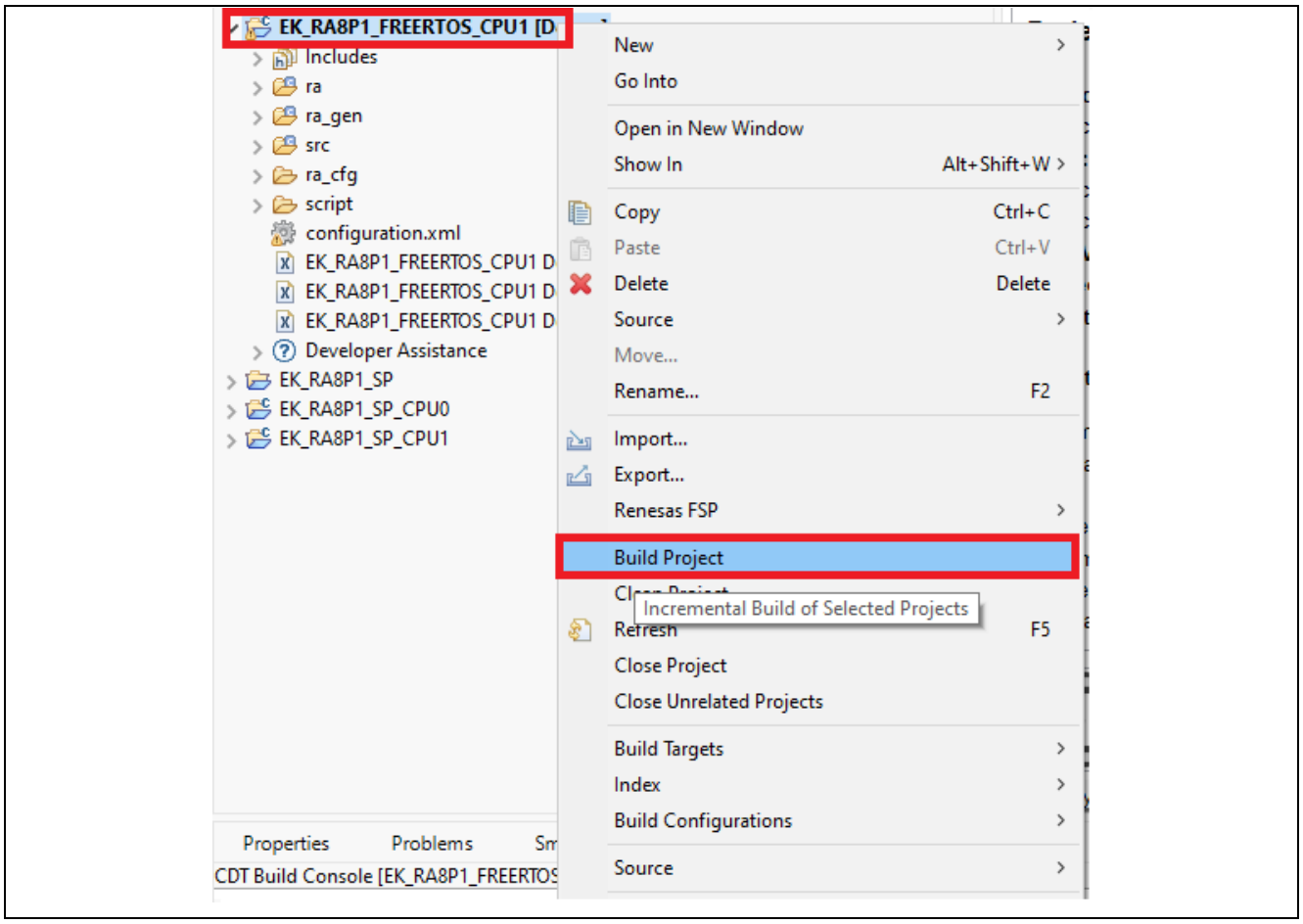

Figure 49. Compile the FreeRTOS Blinky Template Project for CPU1

After the building of projects for both CPU0 and CPU1 successfully, make sure the image was generated in Debug folder.

Here, in the case of FreeRTOS-based dual-core projects, 2 instances of FreeRTOS will be created for CPU0 and CPU1, as the dual-core is heterogeneous in nature.

The CPU0 starts by default, and user code can be viewed at blinky_thread_entry(). It also starts the CPU1 by calling the R_BSP_SecondaryCoreStart();

3. Inter-Processor Communication (IPC) Mechanisms in RA8P1

In the RA8P1 microcontroller, efficient inter-core communication is critical to maximizing system performance and making optimal use of available resources in dual-core applications. The IPC mechanism serves as a key in facilitating this coordination between CPU cores.

To streamline development, the FSP provides a dedicated IPC Hardware Abstraction Layer, which abstracts low-level hardware complexities such as register configurations, FIFO handling, and interrupt control. Instead, developers can utilize a set of well-defined, high-level APIs to implement reliable and efficient inter-core communication.

These APIs enable:

- Seamless message transmission between cores using hardware-backed FIFO queues, ensuring low-latency, unidirectional data flow.
- The generation and handling of maskable interrupts, allowing one core to asynchronously signal events or status changes to the other.
- Registration of user-defined callback functions for processing incoming messages or events in a non-blocking, event-driven fashion.

This structured communication framework not only encourages modular and maintainable software architecture but also minimizes risks related to race conditions and timing issues. As a result, it significantly speeds up development cycles and improves system reliability, especially in complex applications such as real-time control, industrial automation, and secure processing.

For further insights into IPC implementation and dual-core communication strategies on the RA8P1, refer to the application note “Getting Started with IPC on Dual-Core RA8P1”.

3.1 Reference to IPC App Note and Application Example

Reference applications demonstrating the use of IPC and the dual-core architecture of the RA8P1 microcontroller are available in the application notes “Getting Started with IPC on Dual-Core RA8P1” and “R01AN7881EU0100-Developing with RA8 Dual-Core MCU”. These resources provide practical examples of how to implement efficient inter-core communication using the IPC HAL driver and APIs. Developers can use these as a foundation to better understand core synchronization, message passing, and interrupt-driven signaling, enabling them to build robust, scalable dual-core applications that fully leverage the performance and parallel processing capabilities of the RA8P1 MCU.

4. Running a Dual-Core Application on Both Cores

4.1 Establish a Debugging Environment to Run the RA8 Dual-Core Project

To debug both cores simultaneously, it is required to begin by initializing your MCU using either the Renesas Flash Programmer or the Renesas Device Partition Manager. This step ensures that the device is set to Protection Level 2 and that the TrustZone boundary has not already been configured. If the boundary has been previously set, you must reset it to establish a suitable environment for debugging.

Completing this initialization is essential; without it, you may encounter issues when downloading project images or starting the debug session.

Initialize the device using the Renesas Device Partition Manager.

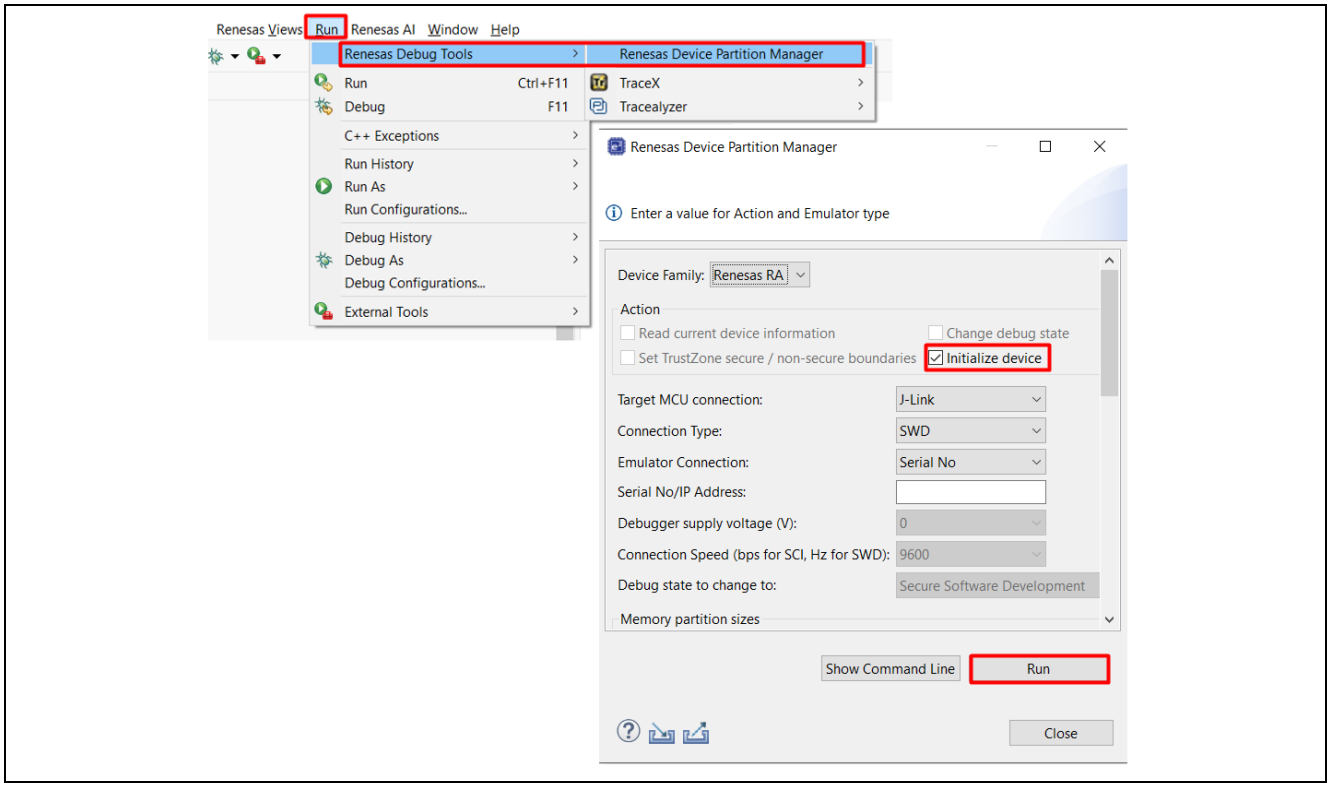

Figure 50. Initialize MCU with RDPM

Figure 51 shows the snapshot of the message displayed after successful device initialization.

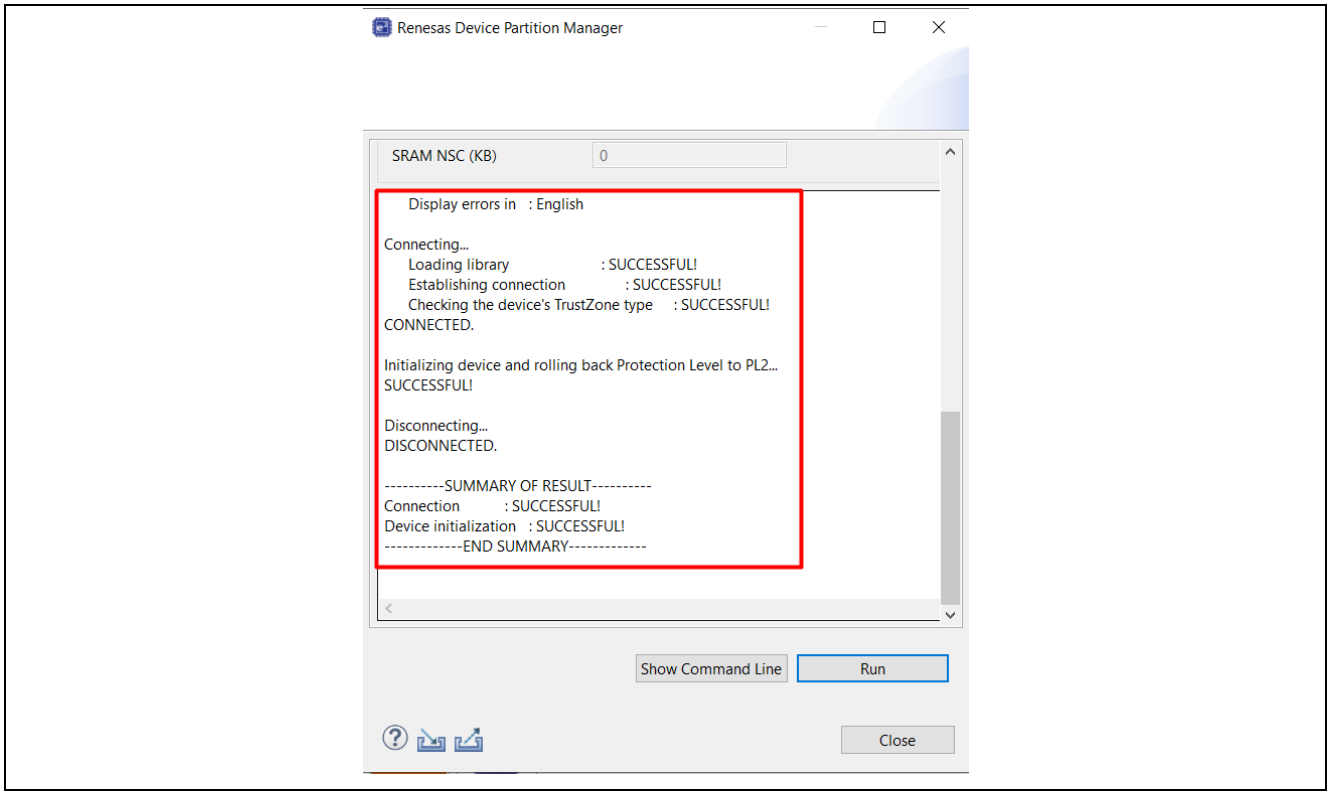

Figure 51. Successful Device Initialization Message on RDPM

To initialize the device using Renesas Flash Programmer:

1. Open the Renesas Flash Programmer software.

2. Create a new project and establish a connection to the target MCU.
3. Navigate to the Target Device tab.
4. Click Initialize Device to perform the initialization operation.

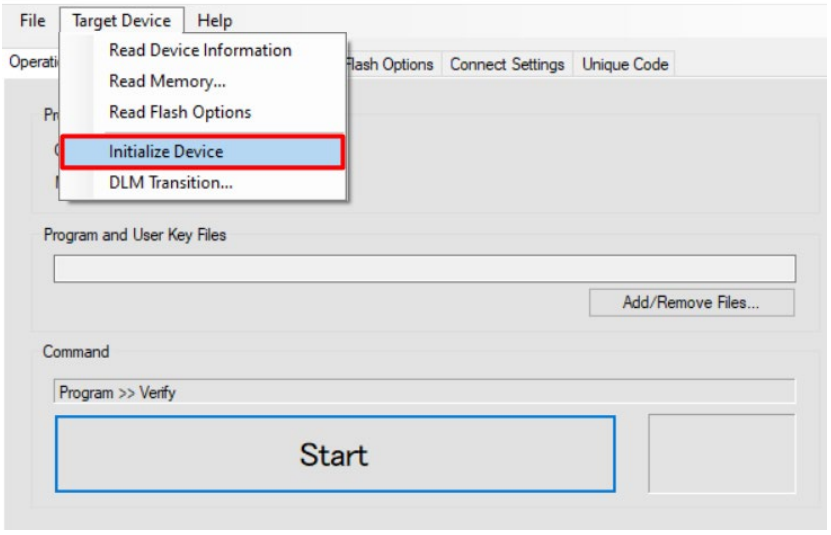

Figure 52. Initialize MCU with RFP

The snapshot of the status message that will be displayed on the console is shown in Figure 53.

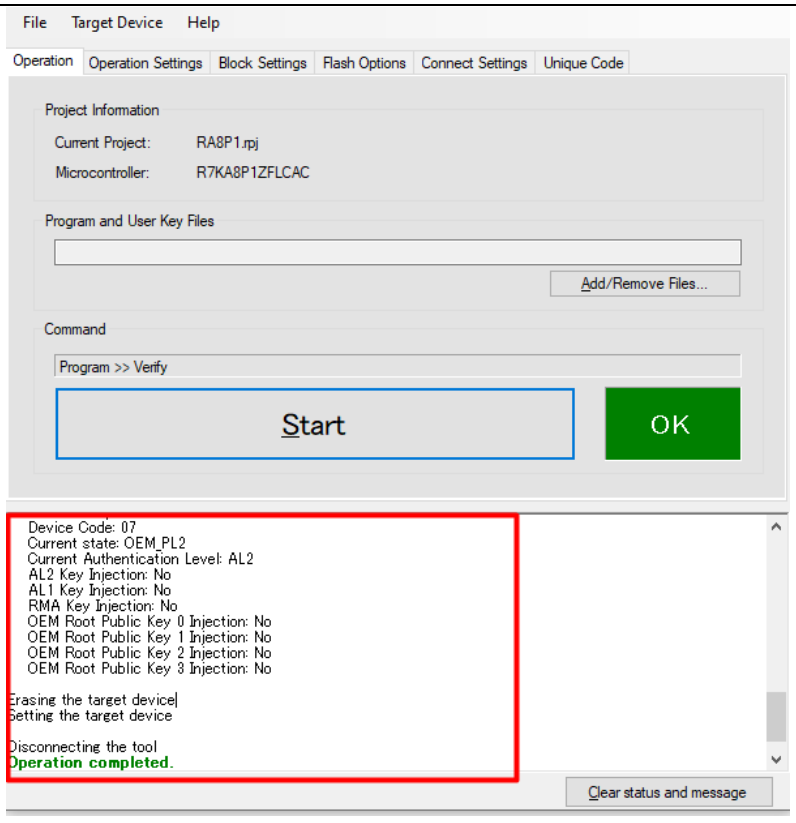

Figure 53. Successful Device Initialization Message on RFP.

After initializing the device with RDPM or RFP, access the Debug Configuration. Select EK_RA8P1_blinky_CPU1 Debug_Multicore Launch Group, then navigate to the Debugger tab and click on Connection Settings. Ensure that the TrustZone boundary settings are disabled.

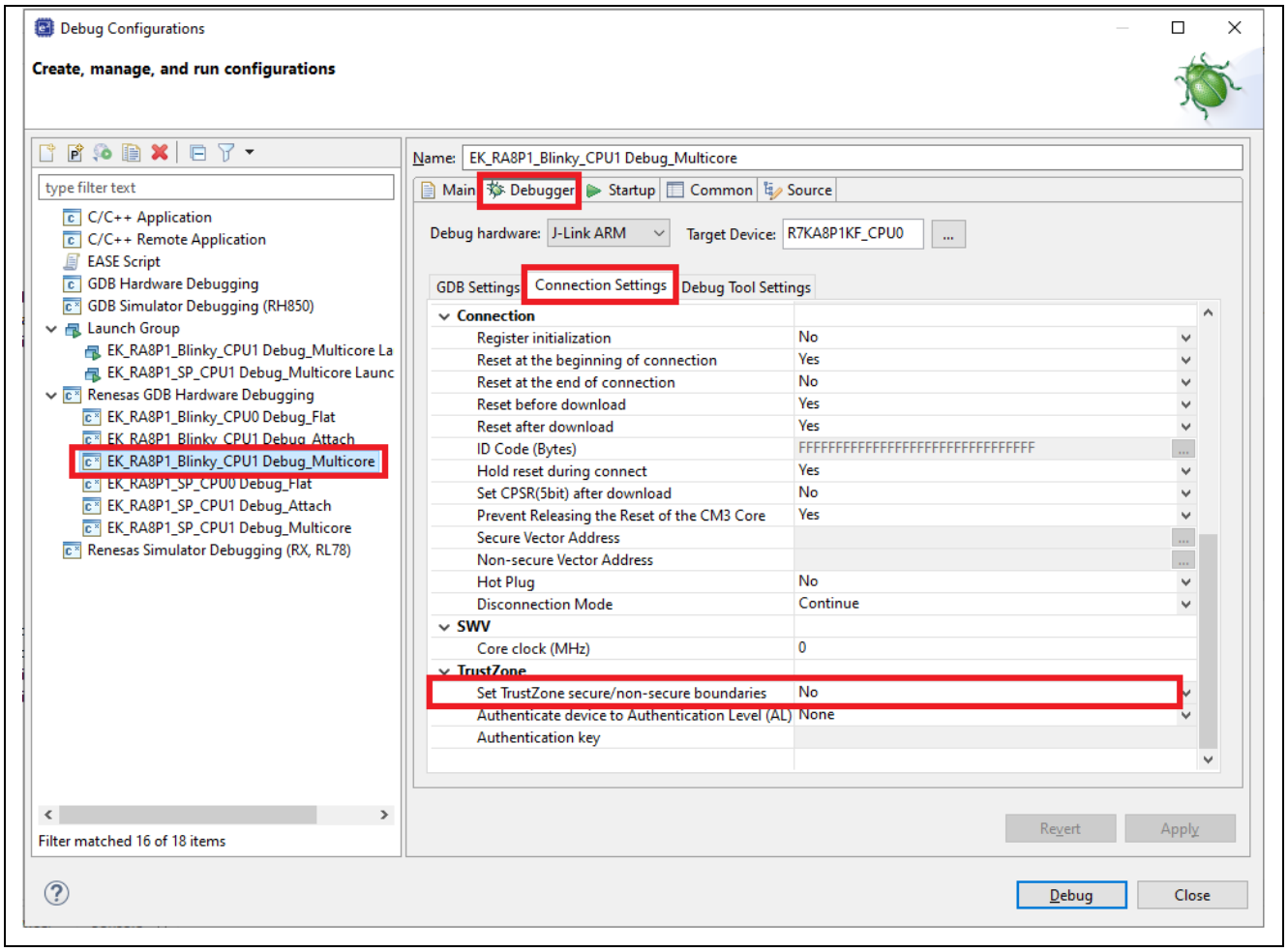

Figure 54. Snapshot of EK_RA8P1_CPU1 Debug Multicore

To run a dual-core application on the RA8P1 using e² studio, you can start by launching a debug session for CPU0 and then manually initiate a second session for CPU1. Also, a more convenient approach is to let e² studio handle both operations automatically.

The Multicore Solutions Project Wizard facilitates this by creating a Launch Group within the CPU1 project. This group includes individual launch configurations for both CPU0 and CPU1. Instead of launching each configuration separately, this Launch Group is used to initiate a combined multicore debug session.

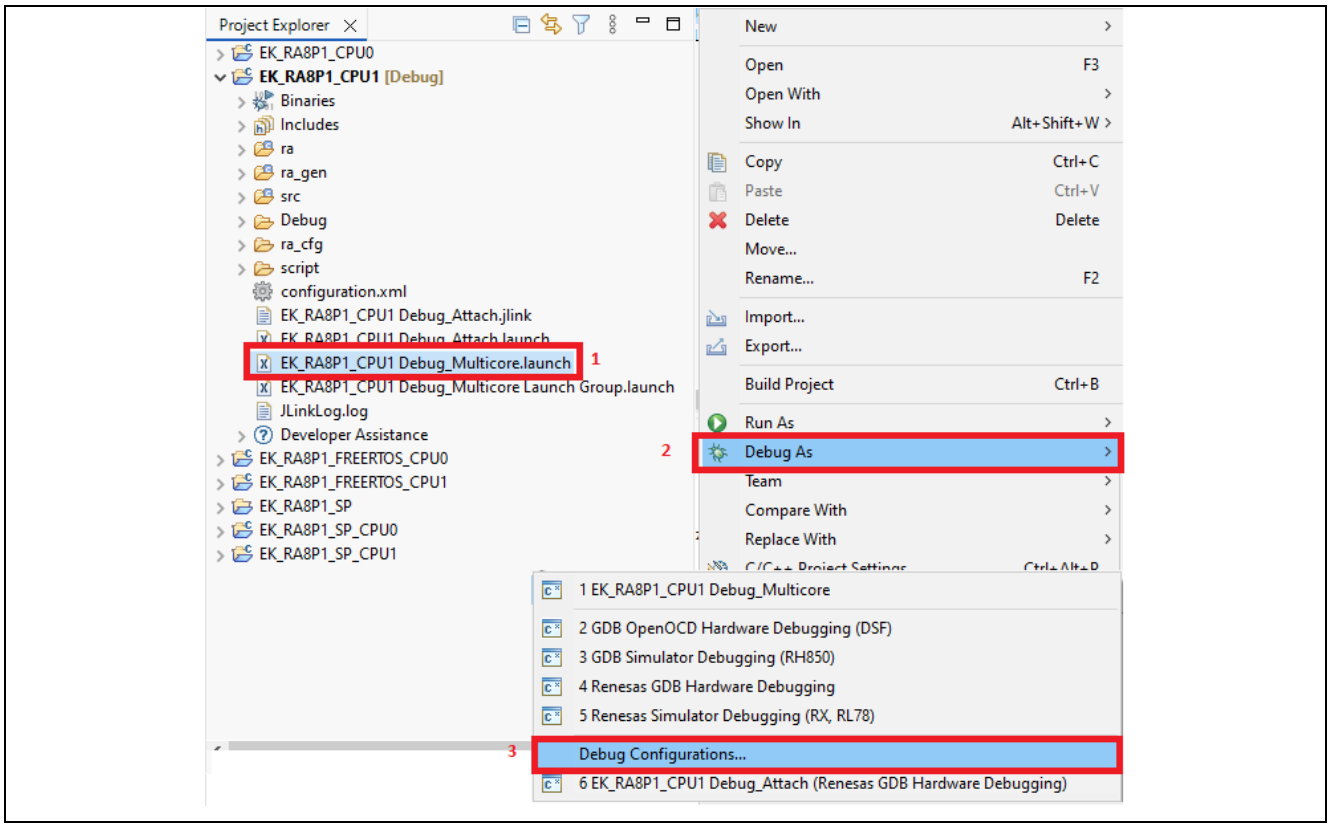

Figure 55. Debugging via Launch Group for the Multicore Project

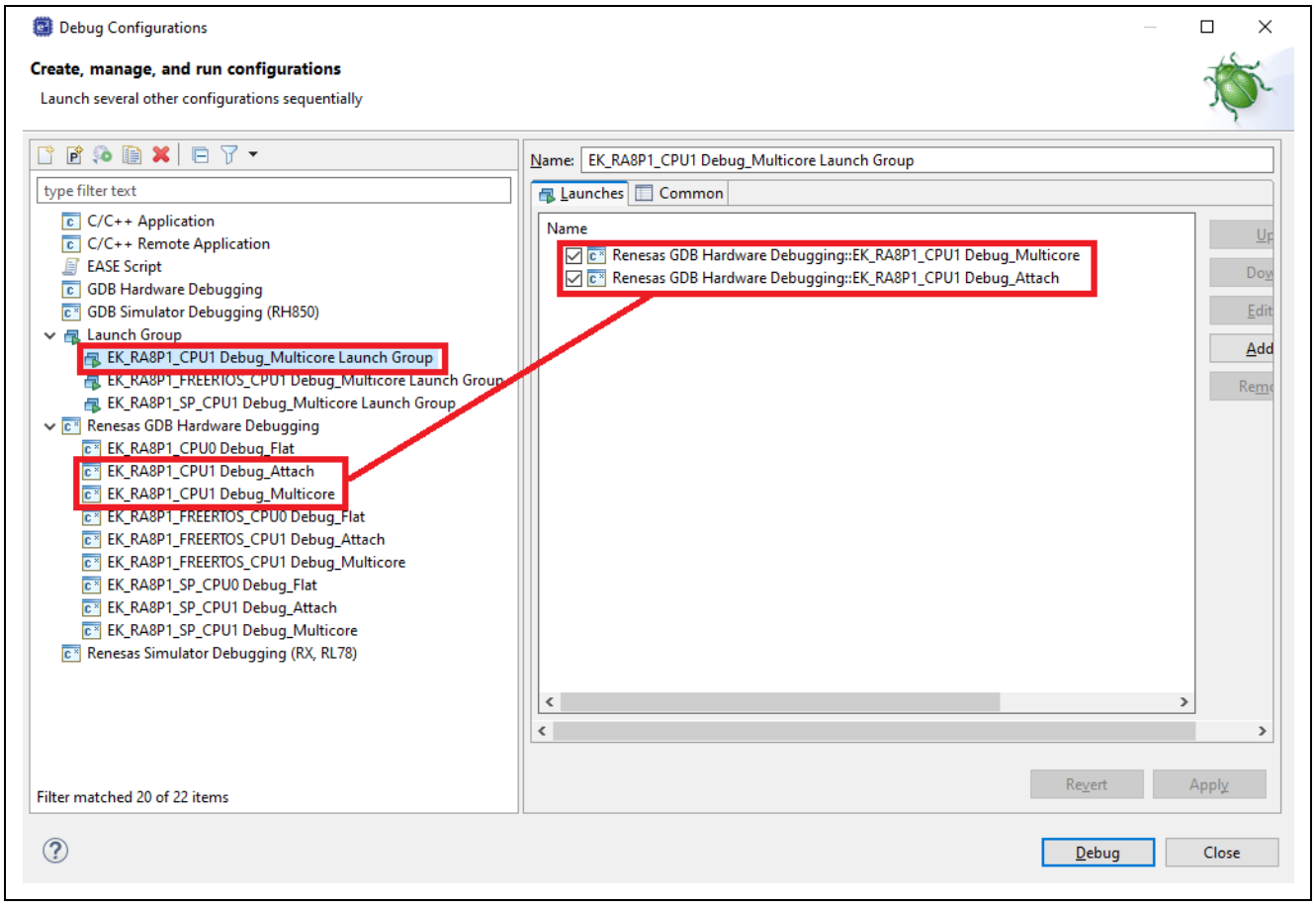

Figure 56. Sample Multicore Debug Launch Group

4.2 Importing the Project

1. Launch the e² studio IDE.
2. Select any workspace in Workspace Launcher.
3. Close the **Welcome** window.
4. Select **File → Import**.
5. Select **Existing Projects into Workspace** from the **Import** dialog box.
6. Select archive file “EK_RA8P1_Dual_core_Projects.zip”
7. Select Dual-core project manually created project samples on each core as shown below; click **Finish.**

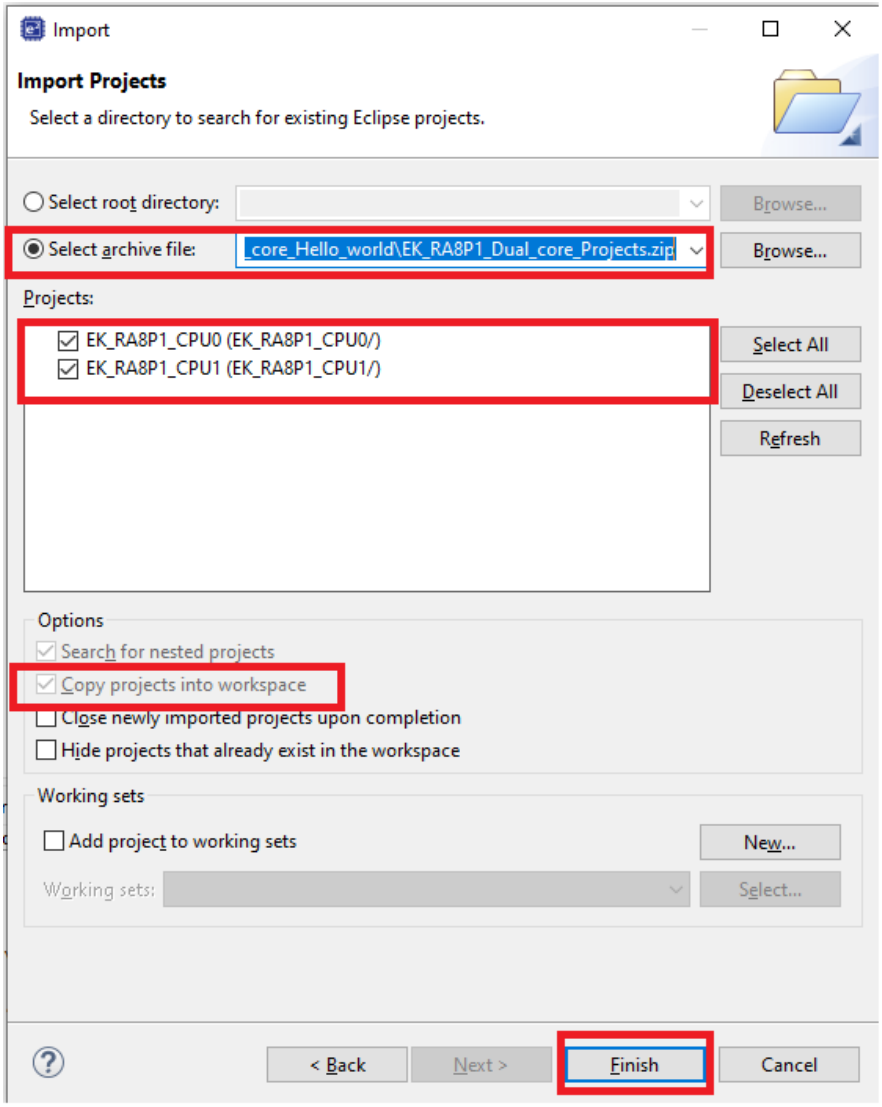

Figure 57. Example of Importing Projects into Workspace.

For the Solution Project, follow the similar steps from 1 to 5 as part of section 4.2 and continue the steps listed below.

1. Select archive file “EK_RA8P1_Dual_core_Solution_Projects.zip”.
2. Select the dual core solution, create project samples on each core as shown below, and click **Finish**.

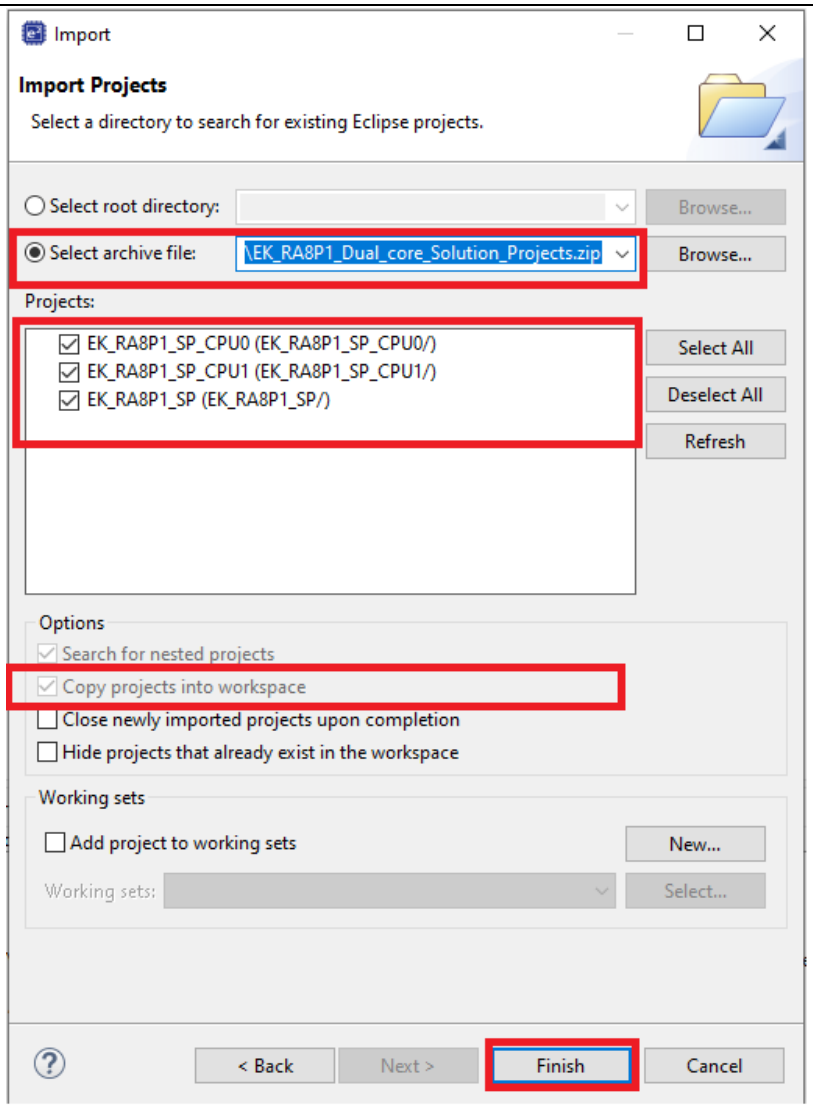

Figure 58. Example of Importing Projects into Workspace.

4.3 Build Projects

When developing a dual-core application on the RA8P1 microcontroller, it is important to follow the correct build sequence to ensure proper initialization and operation of both cores. The application associated with the primary core must be created and built before the one for the secondary core.

By default, the CM85 core is designated as the primary core, and the CM33 core acts as the secondary core. Therefore, the development process should begin with the CPU0 project. Once the CPU0 project is created and successfully built, you can proceed with creating and building the CPU1 project.

This build order is crucial because the primary core typically handles the system initialization and brings up the secondary core using the dedicated startup R_BSP_SecondaryCoreStart() API.

4.3.1 Compile Project Developed on CM85 Core.

Double-click on configuration.xml located in the EK_RA8P1_CPU0 → Click “Generate Project Content.

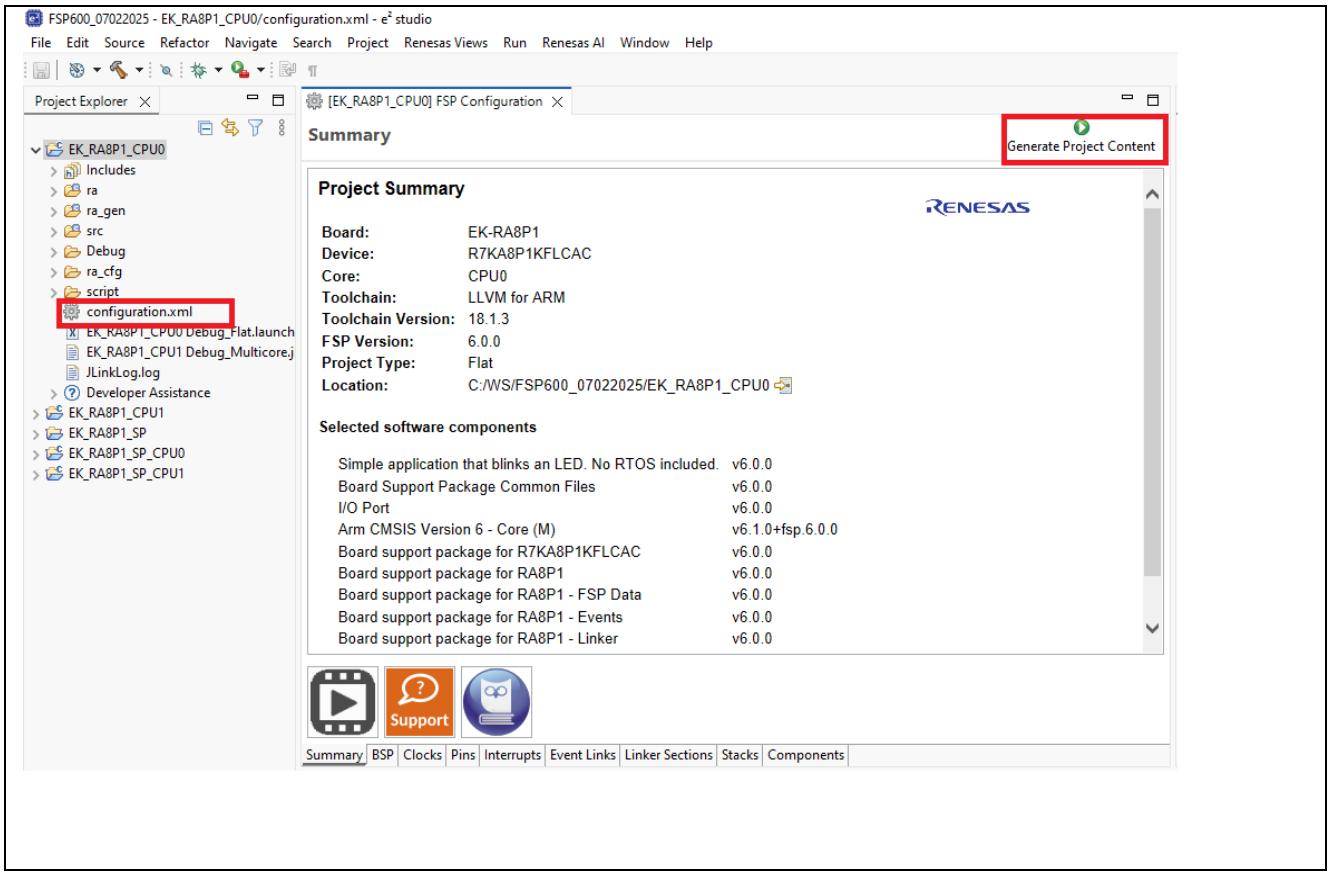

Figure 59. Example of Generating Project Content on CPU0 Project

After generating the project content, right-click on EK_RA8P1_CPU0, then select Build Project to compile the CPU0 core application.

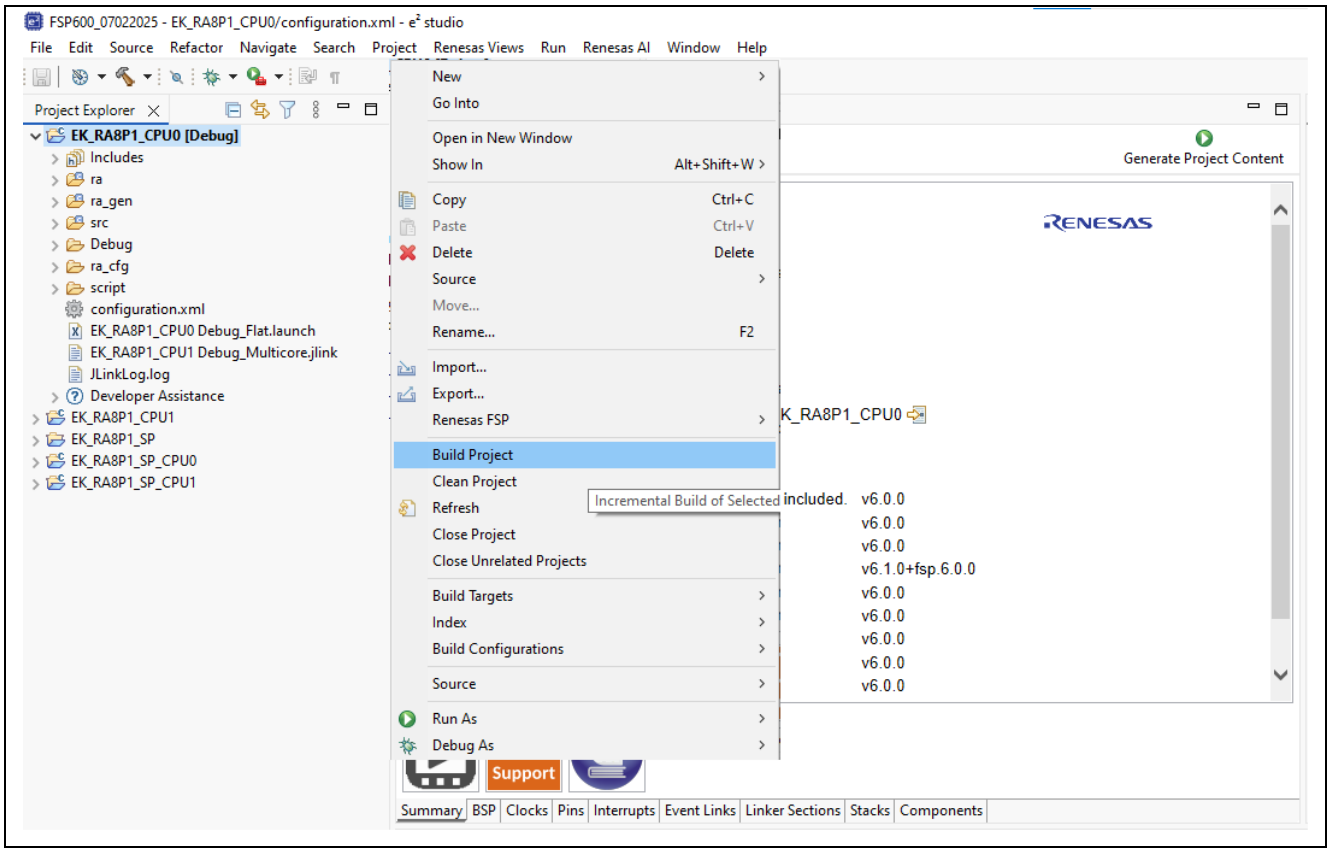

Figure 60. Example of Building CPU0 Dual-Core Project

Verify that the build completes successfully by checking the output in the Build Log console.

4.3.2 Compile Project Developed on CM33 Core.

In the EK_RA8P1_CPU1 project, double-click on configuration.xml, then click Generate Project Content to apply the configuration settings for the CPU1 core.

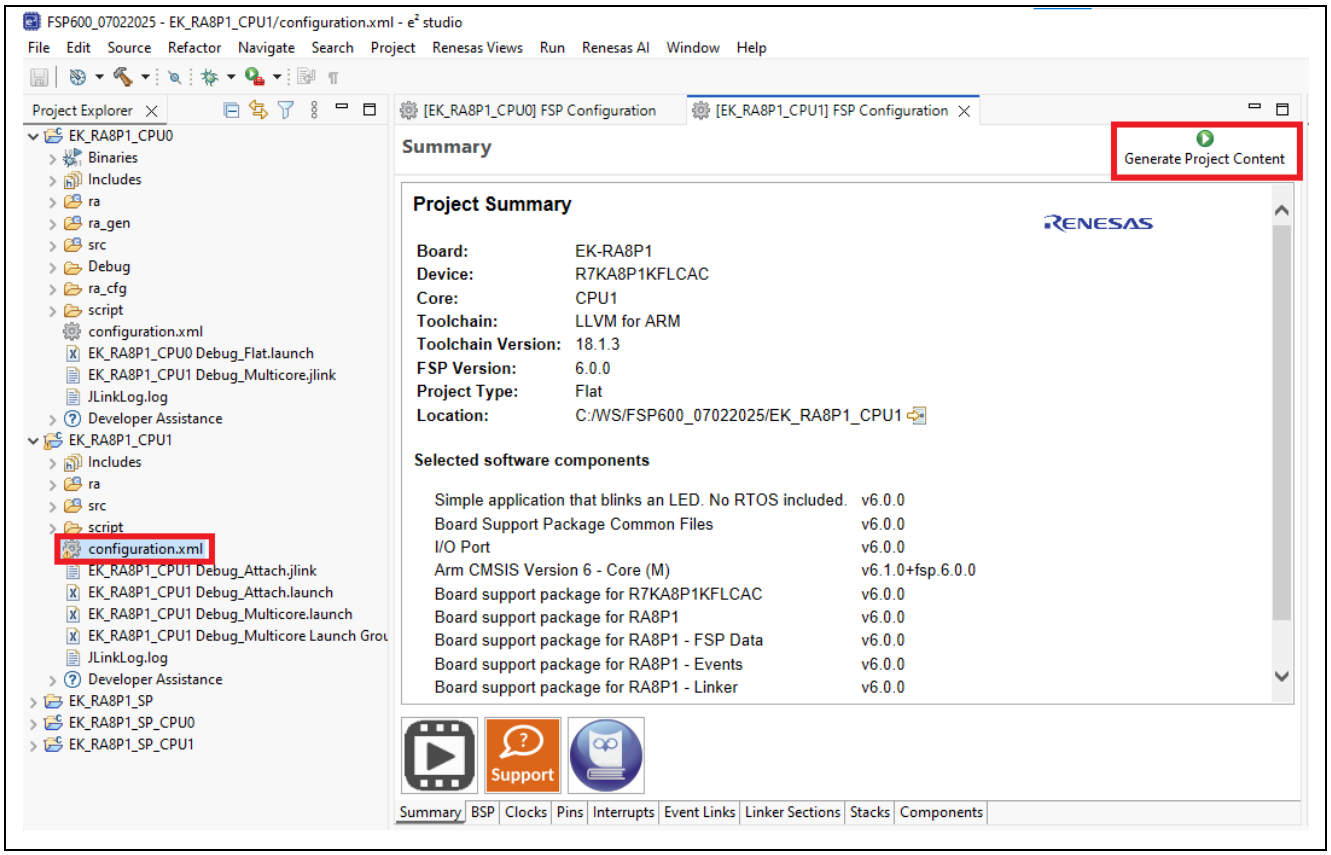

Figure 61. Example of Generate Project Content on CPU1 Project

After the content is generated, right-click on EK_RA8P1_CPU1, then select Build Project to compile the CPU1 core application.

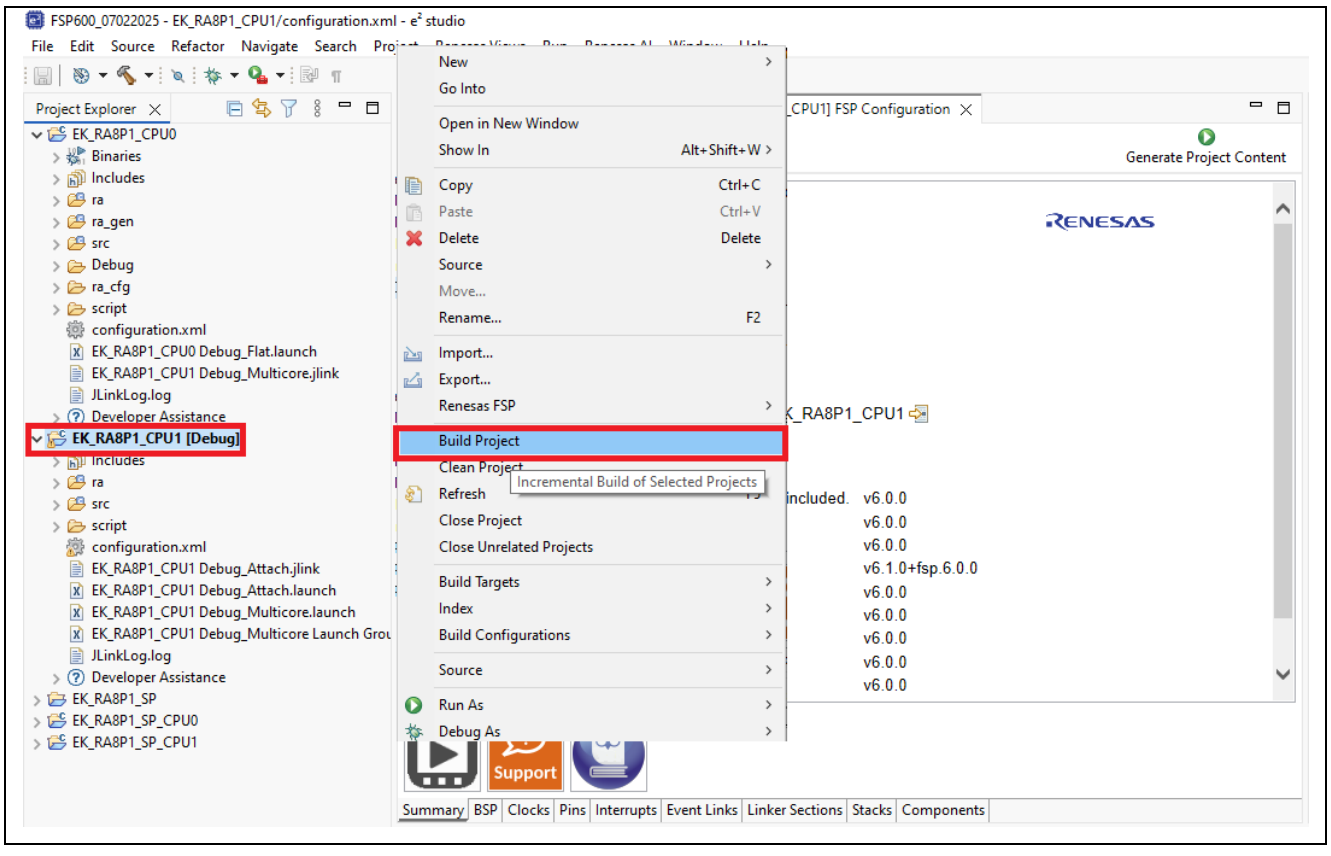

Figure 62. Example of Build CPU1 Dual-Core Project

Ensure that the build completes successfully by checking the output in the Build Log console.

4.3.3 Build Process for Both Cores Using the Solution Project Approach.

If the project was created using the Renesas solution-based approach, alternatively, the build process for both cores can be executed through the solution project without manually building the CPU0 and CPU1 projects. Right-click on the EK_RA8P1_SP solution project and select Build Project, as shown in Figure 63. This command will sequentially build all projects within the solution, following the order: CPU0 → CPU1.

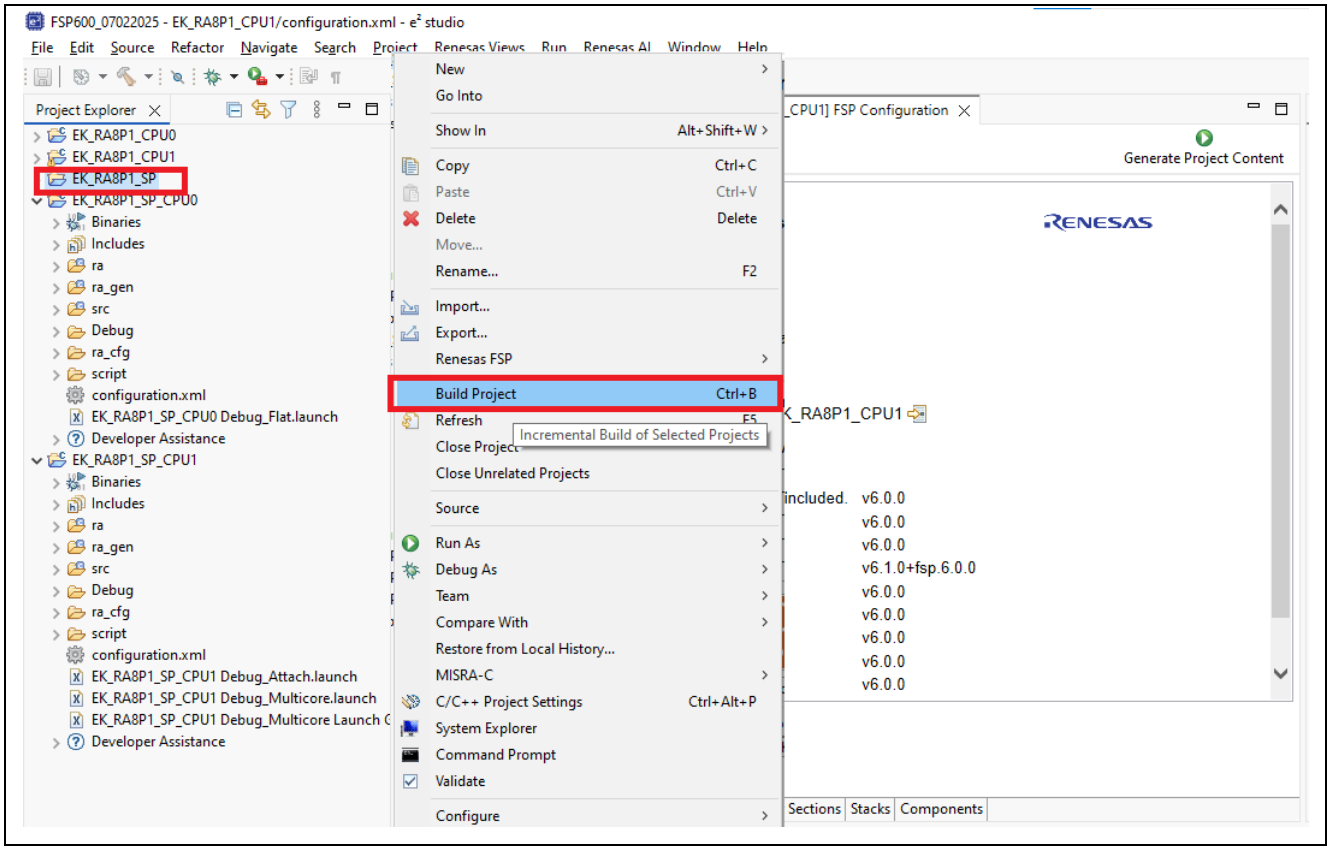

Figure 63. Example of Build RA Solution Project Dual-Core

4.4 Download and Run Projects

As described in the debug settings in section 4.1, the device must be initialized in the OEM_PL2 state, and TrustZone boundary settings are not required for this configuration.

To start a dual-core debug session, open the Debug Configurations dialog as shown in Figure 64.

Select "EK_RA8P1_CPU1 Debug_Multicore Launch Group," then click Debug as illustrated in Figure 65 to simultaneously launch debug sessions for both cores.

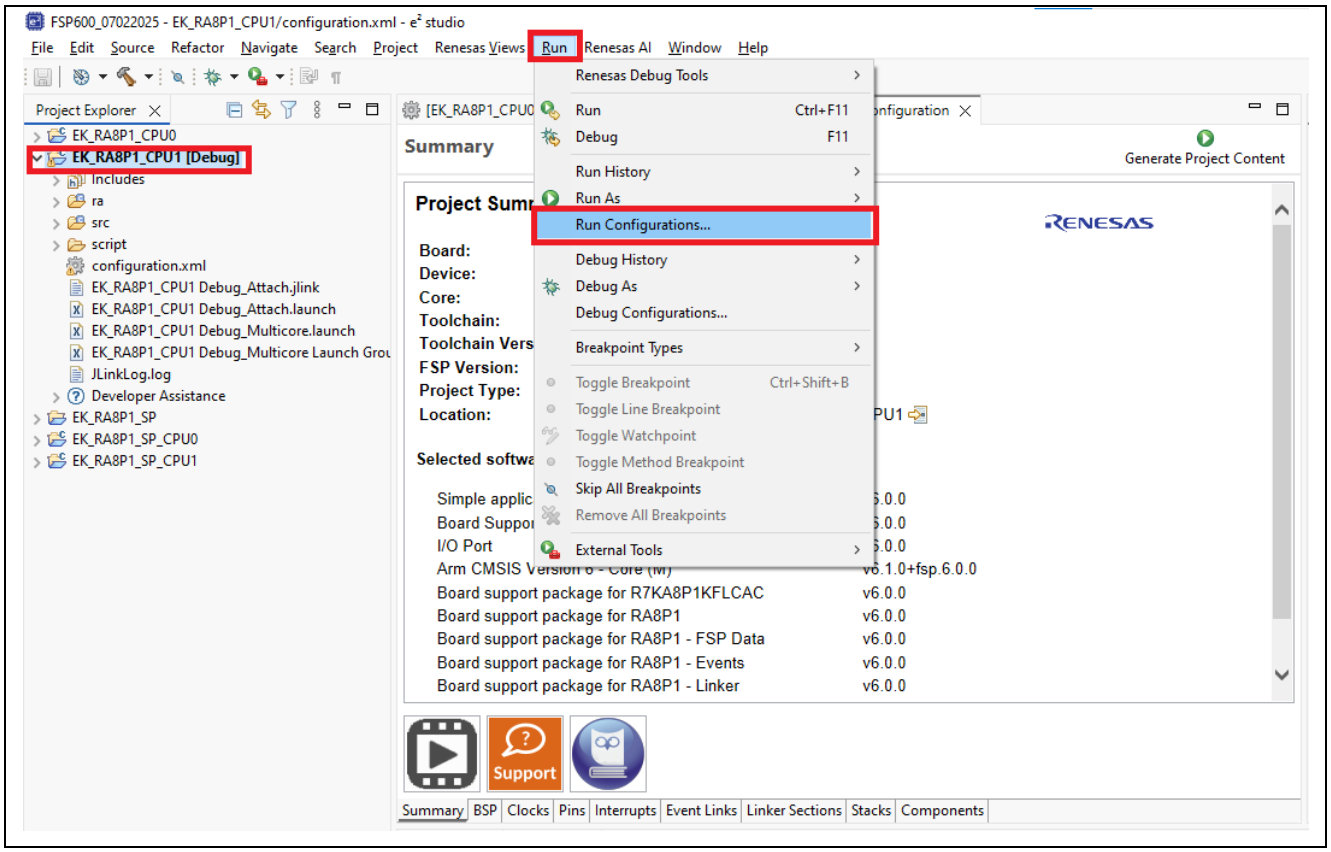

Figure 64. Example of Select Debug configuration on Toolbars

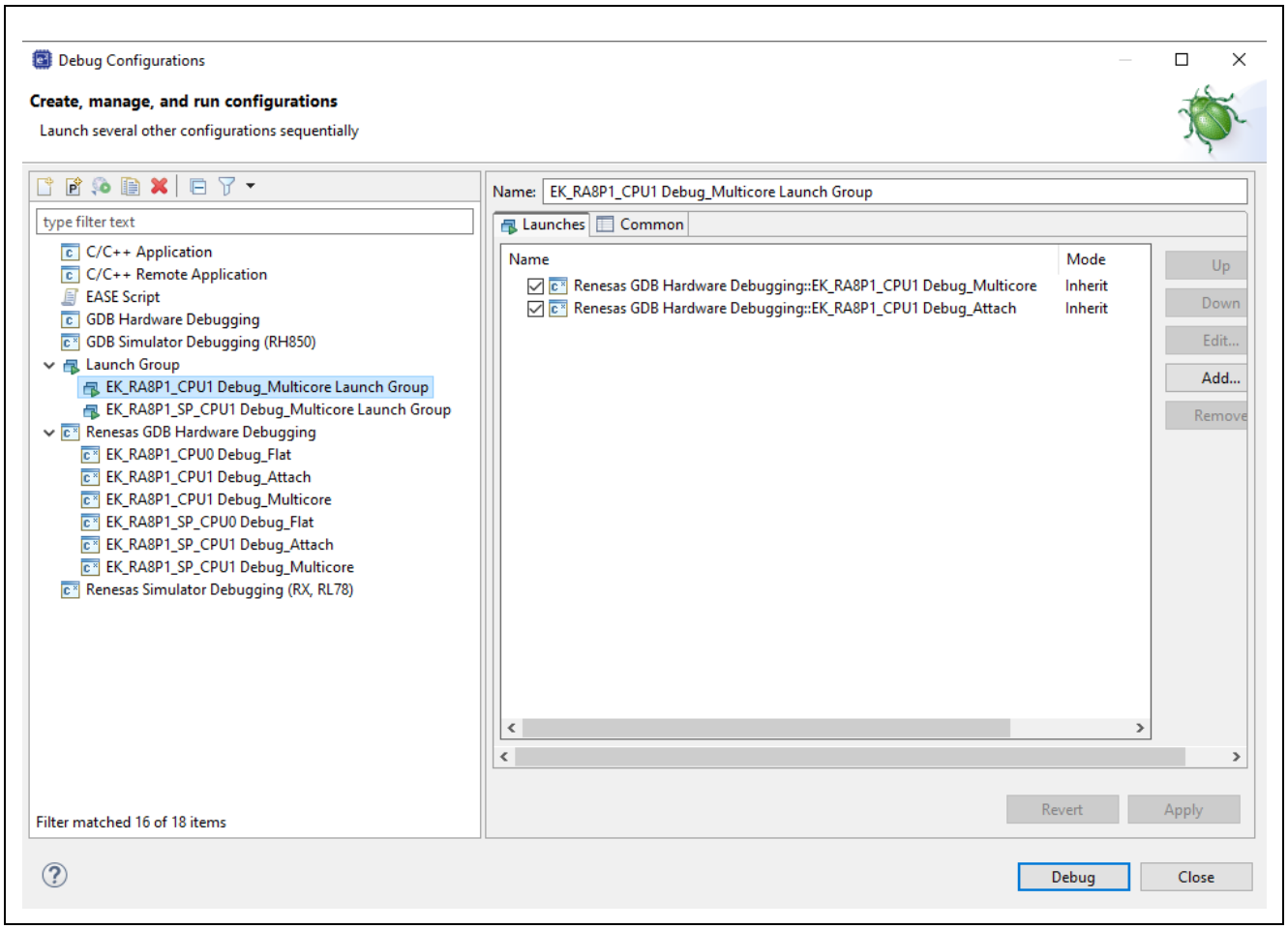

Figure 65. Example of Debug with Multicore Launch Group

The debugging information will appear on the debug console. Clicking Resume to start application execution, the expected output and application behavior can be validated. In this sample application the LED blinking LED1 (Blue) is being driven by CPU0, and LED2 (LED2(Green) is being driven by CPU1.

5. Import, Build, and Verify the FreeRTOS-Based Projects

5.1 Import the Projects

1. Launch the e² studio IDE.
2. Select any workspace in Workspace Launcher.
3. Close the **Welcome** window.
4. Select **File > Import**.
5. Select **Existing Projects into Workspace** from the **Import** dialog box.
6. Select archive file “EK_RA8P1_Dual_core_FreeRTOS.zip”
7. Select the solution project and developed project samples on each core as shown below, and click **Finish.**

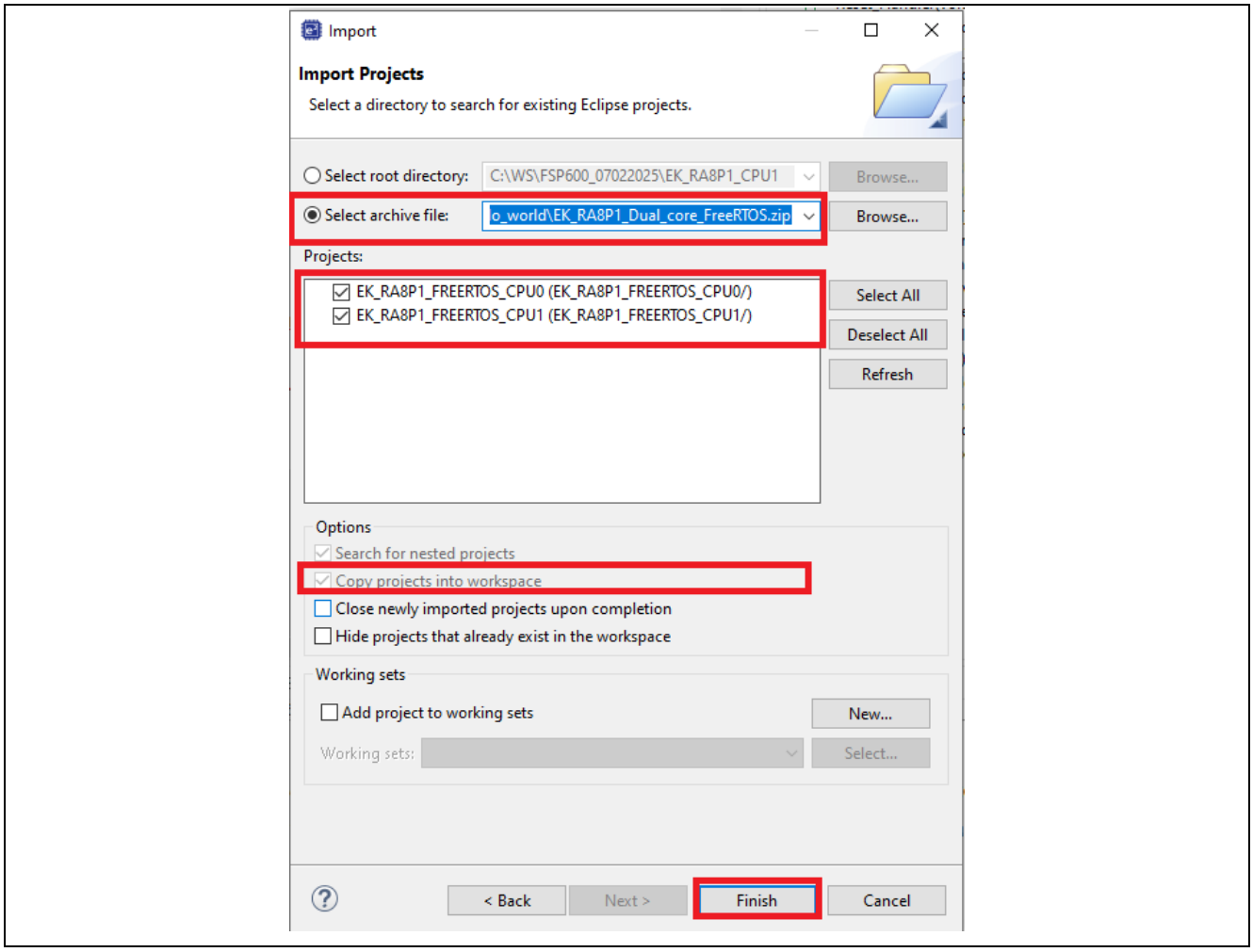

Figure 66. Example of Importing FreeRTOS based Example Project.

5.2 Build Projects

Build the projects sequentially in the order CPU0 → CPU1, as detailed in section 4.3.

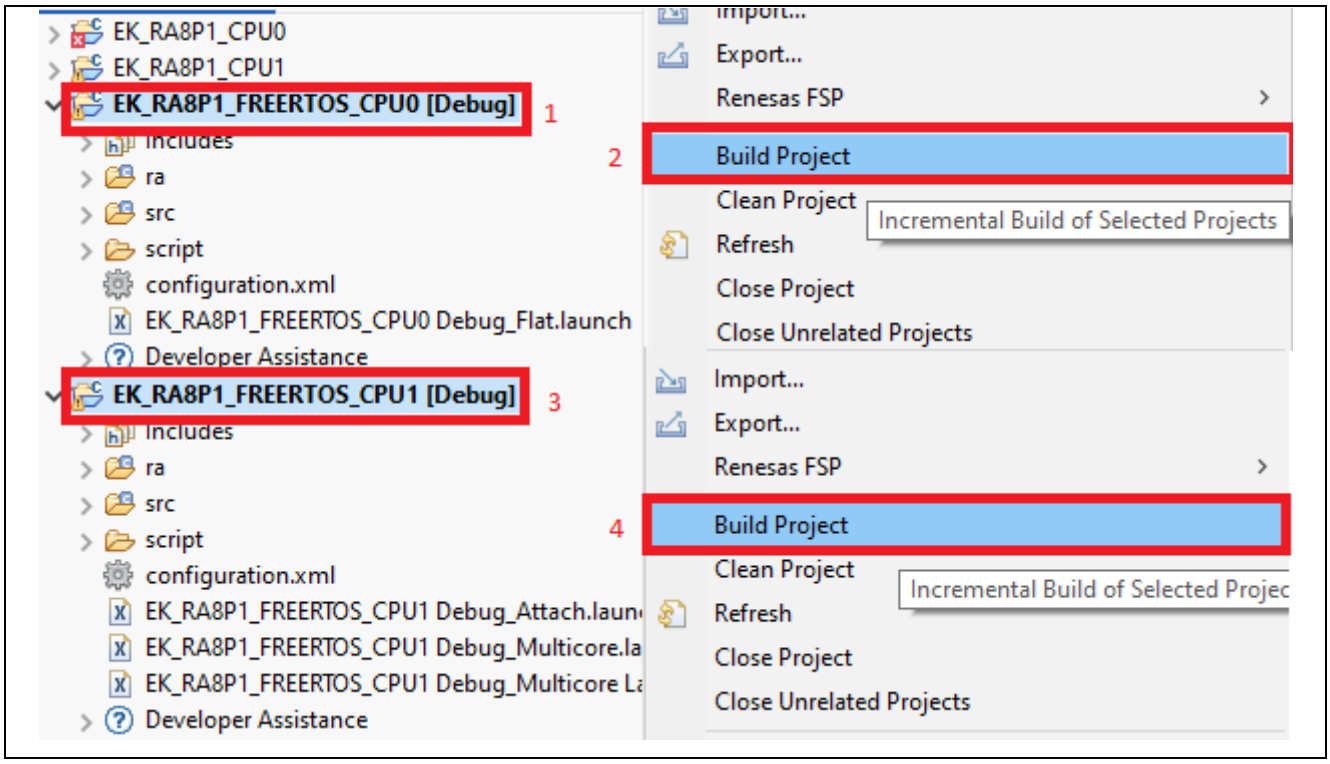

Figure 67. Example of Building FreeRTOS based Project.

Ensure that the build completes successfully for both CPU0 and CPU1 projects by verifying the build status in the Build Log console.

5.3 Download and Run and Verify the Projects

As described in section 4.1, the device must be initialized in the OEM_PL2 state, with no TrustZone boundary settings required.

To start debugging both cores simultaneously:

1. Open Debug Configurations as shown in Figure 68.
2. Select “EK_RA8P1_FREERTOS_CPU1 Debug_Multicore Launch Group,” as shown in Figure 69.
3. Click Debug to launch the dual-core debug session.

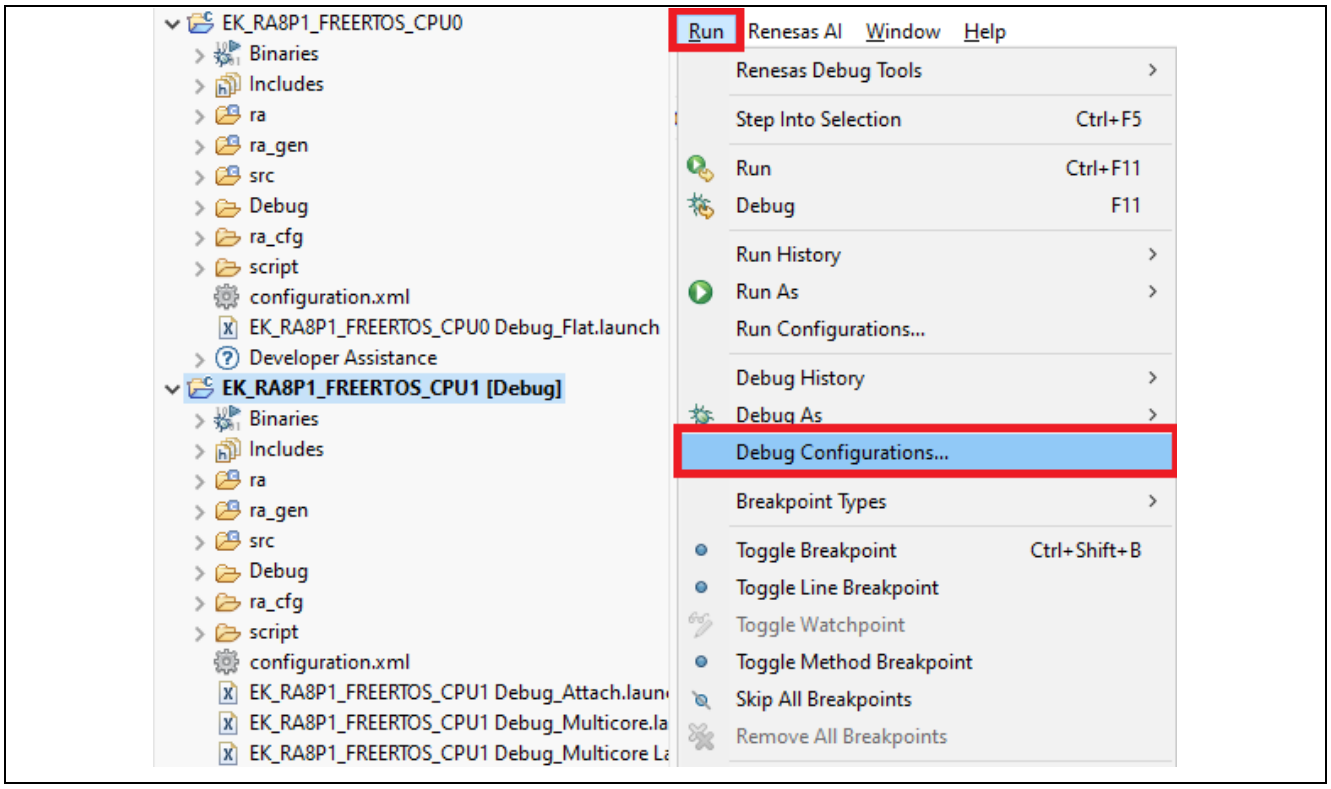

Figure 68. Example of Open Debug Configuration in FreeRTOS Example

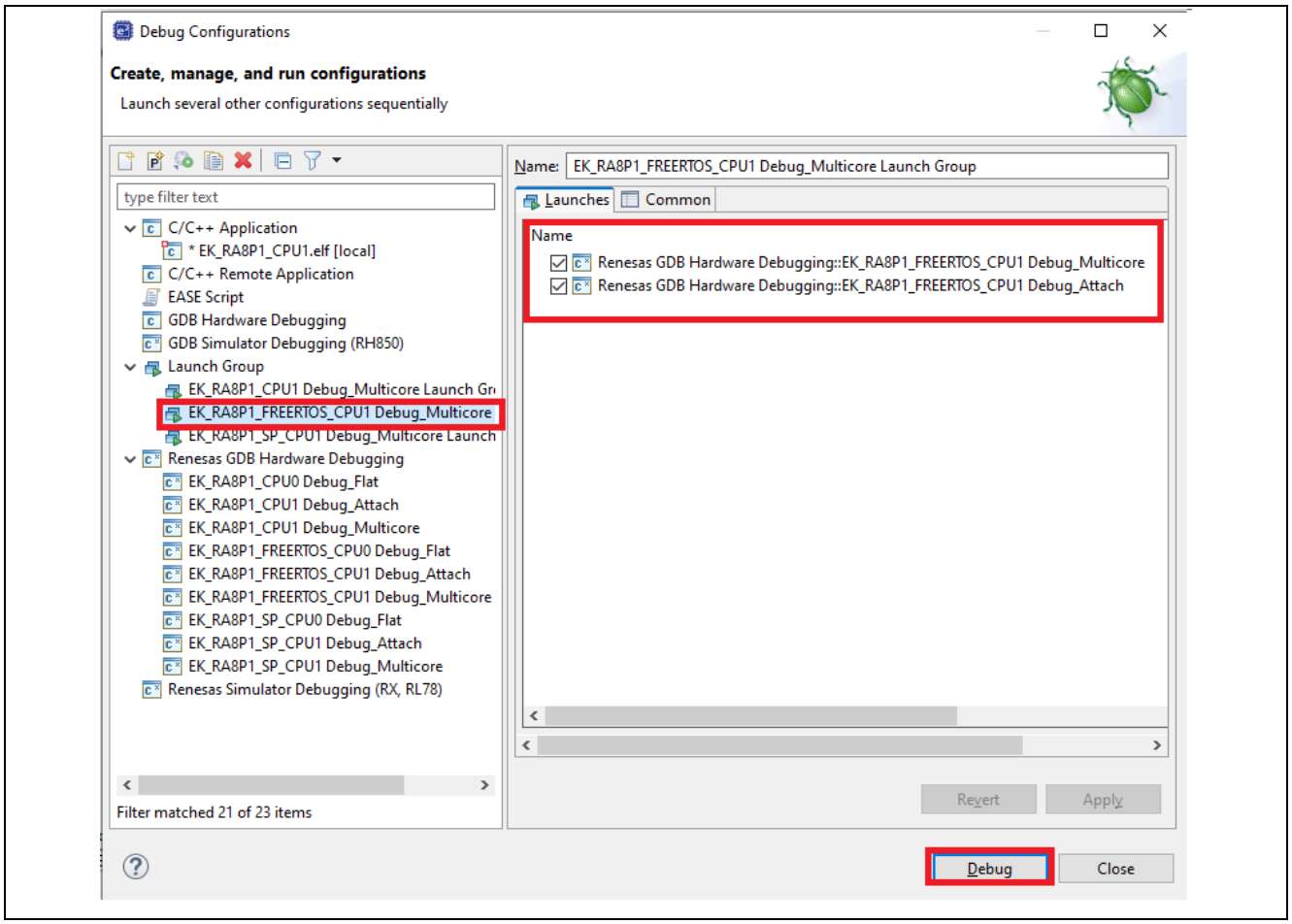

Figure 69. Example of Debug FreeRTOS Dual-Core Example

Return to e² studio and press the Resume button three times to execute the application across both cores. Upon completion, the FreeRTOS example will blink the LED1 (Blue-CPU0) and LED2 (Green-CPU1) successfully.

6. Debugging and Troubleshooting

1. As mentioned in the section 4.1, it is required to go through this step for initial image download and for debugging purpose, not doing so may result in debug error or execution error.
2. Also, using the Launch group, which combines individual launch configurations, helps to run debug the dual-core projects.
3. Additionally, make sure “R_BSP_SecondaryCoreStart()” gets called from the CPU0 project to run the CPU1 core. Sometimes it will not be called. Add a breakpoint to make sure this code is invoked.

7. Next Steps

- To learn more about the EK-RA8P1 kit, refer to the EK-RA8P1 user’s manual and design package available in the Documents and Download tabs respectively of the EK-RA8P1 webpage at renesas.com/ek-ra8p1.

- Renesas provides several example projects that demonstrate different capabilities of the RA the MCUs. These example projects can serve as a good starting point for users to develop custom applications. Example projects (source code and project files) for other kits with RA8D1 are available in the Example Project Bundle and can be reused with AIK-RA8D1. The example projects bundle is available in the Downloads tab of the MCU Evaluation Kit webpage.

- To learn how to create a new e² studio project from scratch, refer to Chapter 2 Starting Development, in the FSP User Manual (renesas.com/ra/fsp). To learn how to use e² studio, refer to the user manual provided on the e² studio webpage (renesas.com/software-tool/e-studio).

8. References

- Renesas FSP User’s Manual: https://renesas.github.io/fsp
- https://www.renesas.com/us/en/document/apn/flash-memory-programming
- Renesas RA MCU Datasheets: See http://renesas.com/ra and select the relevant MCU
- RA8 Example Projects on Renesas RA GitHub: https://github.com/renesas/ra-fsp-examples
- RA8 Quick Design Guide (r01an7087eu0100)
- Getting Started with IPC on Dual-Core RA8P1
- Developing with Developing with RA8 Dual-Core MCU (R01AN7881EU0100)

Website and Support

Visit the following vanity URLs to learn about key elements of the RA family, download components and related documentation, and get support.

EK-RA8P1 Resources	www.renesas.com/ek-ra8p1
RA Product Information	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug.08.25	-	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

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/.

Trademarks

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