Renesas RA Family

Security Design with Arm® TrustZone® using Cortex-M33

Introduction

Arm® TrustZone® technology for ARMv8-M is an optional security extension that is designed to provide a foundation for improved system-level security in a wide range of embedded applications. This application note explains the various RA MCU TrustZone technology-enabled hardware and software features and provides guidelines for using these features. In addition, this application project provides step-by-step instructions to kickstart TrustZone technology-enabled secure system design with Renesas RA Family MCUs.

For fundamentals of Arm® TrustZone® Technology, users are encouraged to read the document *Arm® TrustZone Technology for the Armv8-M Architecture* from Arm. This application project focuses on the implementation of TrustZone technology and features for RA Family MCUs with TrustZone support. At the time of release, the RA MCU groups covered by this application project will include the MCU groups with the support of both TrustZone and Device Lifecycle Management.

Creating a secure design involves using hardware enforcement, software development for security, and tooling support. For TrustZone-based security design, tooling plays a critical role in the development, production, and deployment of a product. For the tools support, refer to the *FSP User's Manual section: Primer: TrustZone Project Development* prior to proceeding to TrustZone-based development.

An EK-RA6M4-based application project implementing an IP protection use case for TrustZone technology is provided as a reference project to start application development with the RA Family MCU TrustZone feature. Implementations with e² studio, IAR EWARM, and Keil MDK IDEs are provided with instructions on how to import and run the example projects.

Required Resources

Target Devices

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

- RA4T1
- RA4M2
- RA4M3
- RA4E1
- RA6E1
- RA6E2
- RA6T2
- RA6T3
- RA6M4
- RA6M5

Software and development tools

- e² studio IDE v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- Renesas Advanced Smart Configurator v2024-07

The links to download the above software are available at https://github.com/renesas/fsp.

- IAR Embedded Workbench for Arm version v9.50.2
- (https://www.iar.com/products/architectures/arm/iar-embedded-workbench-for-arm/)
- Keil MDK v5.39.0.0
  (https://www.keil.com/download/product/)
- SEGGER J-Link® USB driver 7.98b (SEGGER J-Link)
- Renesas Flash Programmer (RFP) v3.16

Hardware

- EK-RA6M4, Evaluation Kit for RA6M4 MCU Group (renesas.com/ra/ek-ra6m4)
- Workstation running Windows® 10 and the Tera Term console or similar application.
- One USB device cable (type-A male to micro-B male)

Prerequisites and Intended Audience

This application project assumes that you have some experience with the Renesas e^2 studio IDE, IAR EWARM, and Keil MDK IDEs. In addition, the user is expected to understand how to extract the generated content from FSP and Renesas RA Smart Configurator. In addition to reading the two reference documents mentioned in the Introduction section, we recommend reading the first two chapters of the application note *Renesas RA Family Installing and Utilizing the Device Lifecycle Management Keys* to understand the Device Lifecycle States of RA TrustZone technology-enabled MCUs. Furthermore, users must know how to enter MCU boot mode using the EK-RA6M4 and create a basic RFP project to communicate with the MCU. This application project only provides the necessary settings for the specific functions used in this application project. For more information on the MCU boot mode and RFP, refer to the *Renesas RA6M4 Group User's Manual: Hardware* and *Renesas Flash Programmer User's Manual*.

The intended audience is all users who are or will be developing Arm® TrustZone® applications using Renesas RA Family MCUs.

Contents

1. Introduction to Arm® TrustZone® and its Security Features

1.1 TrustZone Technology Overview

Arm® TrustZone® technology is a hardware-enforced separation of MCU features. Arm® TrustZone® technology enables the system and the software to be partitioned into Secure and Non-secure worlds. Secure software can access both Secure and Non-secure memories and resources, while Non-secure software can only access Non-secure memories and resources. These security states are orthogonal to the existing Thread and Handler modes, enabling both a Thread and Handler mode in both Secure and Non-secure states.

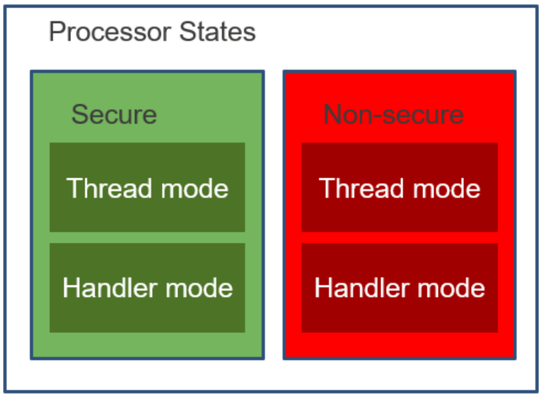

Figure 1. Processor States

The Armv8-M architecture with Security Extension is an optional architecture extension. If the Security Extension is implemented, the system starts up in the Secure state by default. If the Security Extension is not implemented, the system will always be in a Non-secure state. Arm® TrustZone® technology does not cover all aspects of security. For example, it does not include cryptography.

In designs with Armv8-M architecture with Security Extension, components that are critical to the security of the system can be placed in the Secure world. These critical components include:

- A Secure boot loader
- Secret keys
- Flash programming support
- High-value assets

The remaining applications are placed in the Non-secure world.

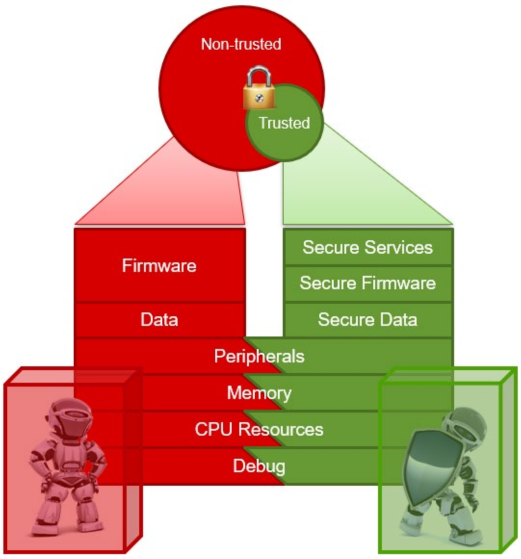

Figure 2. Secure and Non-secure Worlds

As mentioned in the Introduction section, for more details on the definition and usage of TrustZone®, see the Arm document, *Arm® TrustZone® Technology for the Armv8-M Architecture.*

1.2 RA MCU Hardware Enforced Security using Arm® TrustZone®

To build a Secure hardware platform, the security considerations need to go beyond the processor level. Renesas RA Arm® TrustZone® enabled MCUs to extend the security arrangement to the entire system, including:

- Memory system
- Bus system
- Access control to Secure and Non-secure peripherals
- Debug system

Note that the RA6M4 MCU Groups are used as a reference in this section. Other TrustZone technology-enabled MCUs may have some variations in terms of the details of the hardware features.

1.2.1 Memory Separation

Code flash, data flash, and SRAM on RA TrustZone technology-enabled RA MCUs are divided into Secure (S), Non-secure (NS), and Non-secure Callable (NSC) regions by way of the IDAU (Implementation Defined Attribution Unit). These memory security attributes are programmed into the nonvolatile memory using serial programming commands when the device lifecycle is in the Secure Software Development (SSD) state. For the Device Lifecycle State definition and transitions, see the *Renesas RA6M4 Group User's Manual: Hardware* section, Security Features.

Figure 3 shows a summary of the 8 available regions.

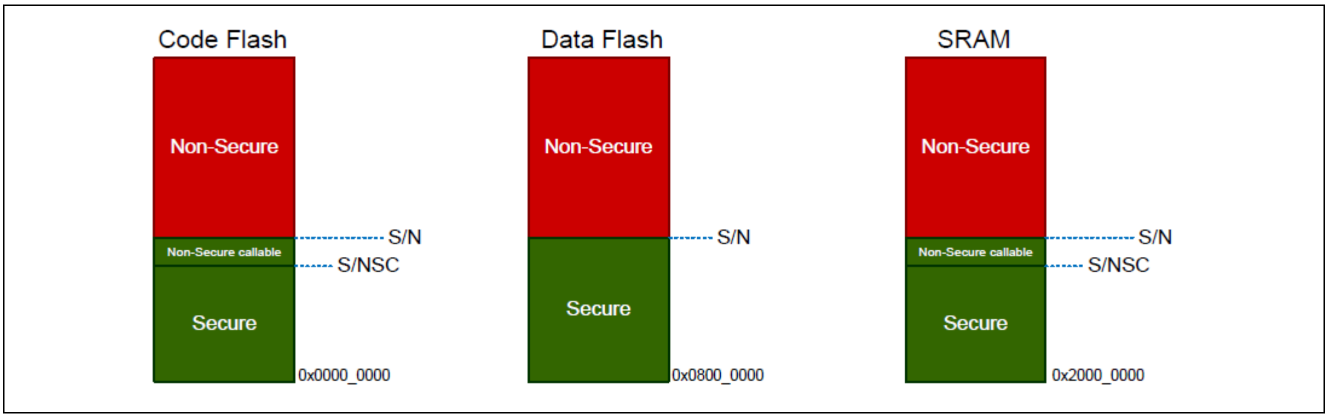

Figure 3. IDAU Regions

Code and Data Flash TrustZone® Based Security Features

Code and Data flash regions read from a Non-secure region will generate a TrustZone Secure Fault. Per the MCU design, the Code and Data Flash Programming and Erasing (P/E) mode entry can be configured to be available only from secure software or from both Secure and Non-secure software.

By default, the MCU configures the Code and Data Flash P/E functionality, which is available only from Secure software. The flash driver may be placed in the Secure partition and may be configured as Non-secure Callable through the FSP to allow the Non-secure application to perform flash P/E operations.

Table 1. Secure Flash Region Read/Write Protection

Access Violation	Error Report
Flash read	TrustZone Secure Fault: Reset or Non-Maskable Interrupt (NMI).
Flash P/E mode entry	Flash P/E Error Flag: Handled by FSP flash driver.

RA Family MCUs support temporary and permanent Flash Block Protections for both the Secure region and the Non-secure region. For more details on the Code and Data Flash TrustZone technology-enabled hardware features, see the *Renesas RA6M4 Group User's Manual: Hardware*, Flash Memory section.

SRAM

SRAM memory, such as SRAM0, which includes an ECC region and Parity, can be divided into Secure/Non-secure Callable/Non-secure regions with Memory Security Attribution (MSA) and can be protected from Non-secure access. When MSA indicates that an SRAM memory region is of Secure or Non-secure Callable status, Non-secure access cannot overwrite them.

Table 2. Secure SRAM Region Read/Write Protection

Access Violation	Error Report
SRAM read	Arm® TrustZone® Secure Fault: Reset or NMI
SRAM write	Arm® TrustZone® Secure Fault: Reset or NMI

1.2.2 Bus System Separation

The IDAU region setup is consistent for the CPU, Direct Memory Access Controller (DMAC), and Data Transfer Controller (DTC). Master TrustZone filters are implemented for the DMAC and DTC.

1.2.2.1 Master TrustZone Filter for DMA Controller and Data Transfer Controller

The DMAC and DTC are supervised by the Master TrustZone Filter. The TrustZone violation area of Flash and SRAM is detected before accessing the bus. The Master TrustZone Filter in the DMAC or DTC can detect the security areas of Flash area (code Flash and data Flash) and SRAM area (ECC/Parity RAM) defined by IDAU. When a Non-secure channel accesses those addresses, the Master TrustZone Filter detects the security violation. Access to the address in violation is not granted. For both DMAC and DTC, the detected access violation is handled as the “Master TrustZone Filter error”. A DMA_TRANSERR interrupt will be generated in response to the “Master TrustZone Filter error”.

Below are some additional comments on the DMAC security attribute:

- The Security Attribution can be configured individually for each channel. Each DMA channel can assume a Secure or Non-secure attribute.
- Only Secure code can configure whether the DMAC can be started by Secure or Non-secure code.
  - If the DMAC is used in the Secure project, the FSP will start DMA in Secure mode and prevent a Non-secure project from accidentally stopping the DMAC by setting up the corresponding registers.

1.2.2.2 Ethernet DMA Controller (EDMAC)

The RA6M4 MCU requires EDMAC RAM buffers to be placed in TrustZone Non-secure RAM. The EDMAC is hard-coded as a TrustZone Non-secure bus master. These hardware features allow the following Ethernet code partitioning options:

- Run Ethernet code as Secure and EDMAC RAM buffer in Non-secure RAM.
- Run Ethernet code and EDMAC RAM buffer in a Non-secure region.

The FSP supports implementations with both options.

1.2.2.3 Bus Master MPU TrustZone® Feature

The Bus Master MPU is available for memory protection function for each bus master except the CPU. Secure software can set up the security attributes of the Bus Master MPU.

Refer to the *Renesas RA6M4 User’s Manual: Hardware* and *FSP User’s Manual* for more details of the security attribute control for the bus systems.

1.2.3 IO and Peripheral Separation

Most peripherals in the MCU can be configured to be Secure or Non-secure with several exceptions as shown in Table 3.

Peripherals are divided into two types:

- Type-1 peripherals have one security attribute. Access to all registers is controlled by one security attribute. Type-1 peripheral security attributes are written to the Peripheral Security Attribution Registers (PSARx: x = B to E) by the Secure application.
  - e² studio and the FSP provide a convenient way to assign the PSARx.
  - Different channels for the peripheral can assume different security attributes. For example, UART Channel 0 and Channel 1 can have different Secure or Non-secure attributes.
- Type-2 peripherals have the security attributes for each register or for each bit. Access to each register or bit field is controlled according to these security attributes. Type-2 peripheral security attributes are written to the Security Attribution register in each module by the Secure application. For the Security Attribution register, see sections in the user manual for each peripheral.
  - e² studio and the FSP provide configurability for most of these peripherals with several exceptions where sensible default settings have been made to provide a better development experience.
  - See the latest *FSP User's Manual* for details for each peripheral.

Table 3. List of Type-1 and Type-2 Peripherals

Type	Peripheral
Type 1	SCI, SPI, USBFS, CAN, IIC, SCE9, DOC, SDHI, SSIE, CTSU, CRC, CAC, TSN, ADC12, DAC12, POEG, AGT, GPT, RTC, IWDT, WDT
Type 2	System control (Resets, LVD, Clock Generation Circuit, Low Power Modes, Battery Backup Function), Flash Cache, SRAM controller, CPU Cache, DMAC, DTC, ICU, MPU, BUS, Security setting, ELC, I/O ports
Always Non-secure	CS Area Controller, QSPI, OSPI, ETHERC, EDMAC

The access permissions of type-2 peripherals are different by peripheral. See the Register Description section of each peripheral.

Table 4. Peripheral Access Control Based on Arm® TrustZone®

Permission	Secure access	Non-secure access
Peripheral configured as Secure	Allowed	Write is ignored; read is ignored. TrustZone Access error is generated.
Peripheral configured as Non-secure	Allowed	Allowed

Notes on Clock Generation Circuit (CGC)

The Clock Generation Circuit has individual security attributes for each of the clock tree controls. The current release of the tooling and FSP provides flexibility for the following clock control schemes:

- Entire clock tree is controlled from the Secure project only and locked down in the Non-secure project.
- Entire clock tree is controllable from the Non-secure project as well as the Secure project.

Refer to *Notes on Clock Control* for the operational details.

Peripherals that Support Non-secure Partition Operation Only

As shown in Table 3, the following three peripherals have limitations in terms of their security attributes:

- **Ethernet:** See Section 1.2.2 for the limitations on Ethernet application development.
- **CS Area Controller, QSPI, OSPI:** These peripherals are Non-secure peripherals only. The FSP has support for them to be used from all three project types. Refer to section 4 for the definitions of project types based on the Project Configurator.

1.2.4 Debug Interface

For the Arm® TrustZone® technology-enabled RA Family MCUs, the debug function is considered in three levels (DBG0, DBG1, and DBG2) to support TrustZone technology-enabled debugging and provide security in development, production, and deployed products:

- DBG2: The debugger connection is allowed, and there is no restriction on accessing memories and peripherals.
- DBG1: The debugger connection is allowed and restricted to access only Non-secure memory regions and peripherals.
- DBG0: The debugger connection is not allowed.

The debug level is determined to correspond to the device lifecycle state of the product. See the *Renesas RA6M4 Group User's Manual: Hardware* chapter on Security Feature section Device Lifecycle Management for more details.

Debug-level regression is possible through the Device Lifecycle Management system. See the application note for the *Renesas RA Family Installing and Utilizing the Device Lifecycle Management Key* for the corresponding operational flows.

For Renesas RA TrustZone technology-enabled MCUs, J-Link, E2, and E2 Lite debuggers are supported.

1.3 Device Lifecycle Management

The RA Family TrustZone technology enabled MCUs to incorporate an enhanced Device Lifecycle Management System using TrustZone technology features and Secure Crypto Engine 9 (SCE9). Device Lifecycle Management is important during TrustZone technology-enabled application development, production, and deployment stages.

For Device Lifecycle State definition and transitions, see the *Renesas RA6M4 Group Hardware User's Manual*. For creation, installation, and use of the Device Lifecycle Management keys during the development and production stages, see the application note *Renesas RA Family Installing and Utilizing the Device Lifecycle Management Keys*.

1.4 Example TrustZone Use Cases

This application project explains two specific use cases for TrustZone technology and provides an example software project for the IP Protection use case.

For additional attack scenarios where an attacker may attempt to access protected information and how the TrustZone technology for ARMv8-M can prevent them, see Chapter 2, Security of *Arm® TrustZone Technology for the Armv8-M Architecture*.

1.4.1 Intellectual Property (IP) Protection

IP protection is a common need for proprietary software algorithms and data protection. TrustZone technology provides good hardware isolation for IP protection. TrustZone technology creates separation between two regions: Secure ("trusted") and Non-secure ("non-trusted") code/data. Users who create building blocks for others to integrate can take advantage of the TrustZone technology feature by storing their software IP in the Secure ("trusted") region.

Business Model

Not all software developers create end products. Some create building blocks, such as algorithms, for others to integrate into an end product. One difficulty they face is the protection of their software IP. Their end customers would prefer to receive source code, but source code can easily be copied and redistributed. Even binary libraries are not complete protection, as there are tools that can disassemble binaries to assembly and even C source code.

TrustZone® technology enables new business models for these developers in which they can program their algorithms into the secure region of a TrustZone-enabled MCU and sell a value-added MCU, with their IP protected by TrustZone and the Device Lifecycle Management (DLM) system of the RA MCU.

RA MCU Device Lifecycle Management Feature for IP Protection

During development, DLM state regression allows erasing the protected areas of flash (unless permanently locked). This prevents reading of the protected area of the flash, hence protecting the IP and eliminating scrappage of devices in case the algorithms need to be modified.

In production, if the algorithm developer would like to retain the potential to debug algorithms with the application in place, they can install DLM keys for the NSECSD to SSD and DPL to NSECSD transitions. Refer to the *Renesas RA Family Installing and Utilizing the Device Lifecycle Management Keys* application note for the definition of the device lifecycle states and state regression operational flow.

- SSD: Secure Software Development
- NSECSD: Non-secure Software Development
- DPL: DePLoyed

RA MCU Flash Block Locking Feature for IP Protection

RA MCUs support temporary and permanent Flash Block Protections. This allows customer IP and Root of Trust to be protected from accidental erasure and alteration.

IP Protection Development, Production, and Deployment Flow

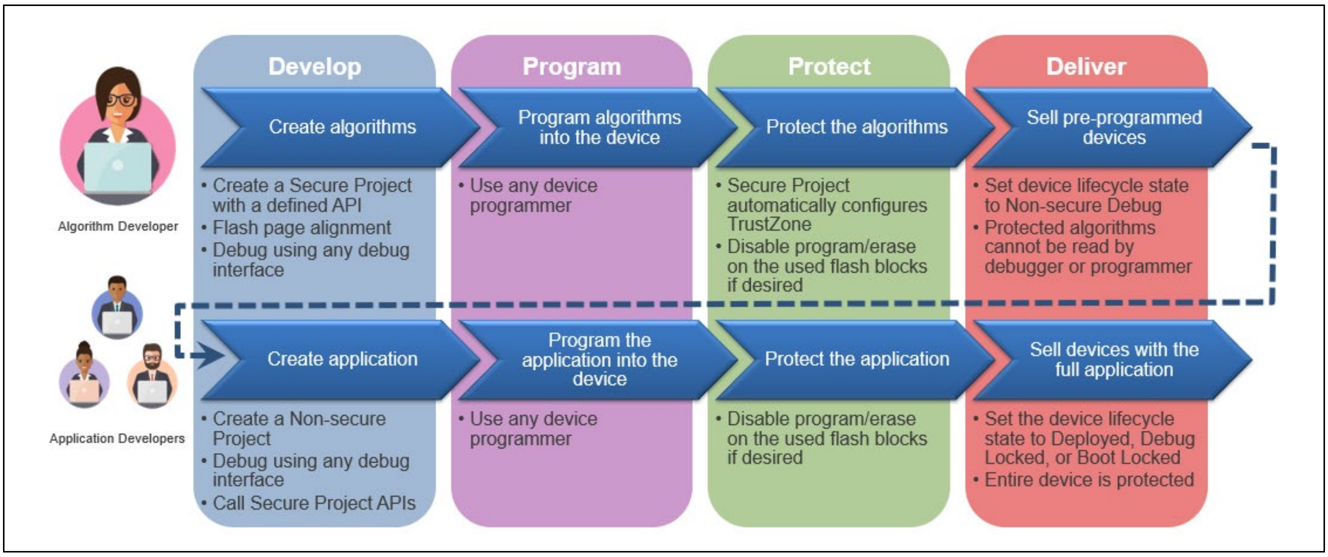

Figure 4. IP Protection using Arm® TrustZone®

Designing for IP protection uses the **Split Project Development** model. See section 4.2 for the operational details.

1.4.2 Root of Trust Protection

The Root of Trust (RoT) is a product's security foundation. All higher-level security is built on top of the RoT. The RoT also implements recovery features for higher-level security breaches. When the Root of Trust is breached, recovery is not possible and can lead to serious consequences. For IoT applications, Root of Trust may encapsulate authenticated firmware updates and secure internet communication.

To reduce the attack surface, the functionality included in the RoT should be as little as possible. Typical services in the RoT are described in Figure 5.

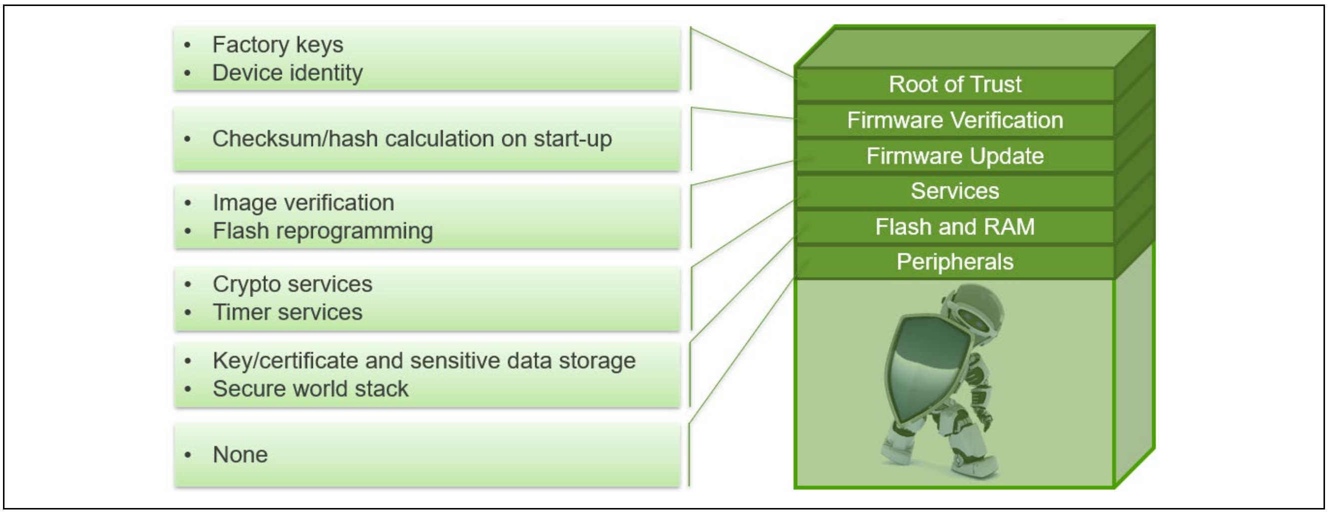

Figure 5. Root of Trust Protection – Put as Little as Possible in the Secure Region

All other application codes and device drivers should be considered to be allocated to the Non-secure region.

2. Arm® TrustZone® Application Design Support

This chapter introduces several IDE features that are established to simplify software development when using the TrustZone hardware isolation with support from other MCU hardware components, FSP software, or tooling.

2.1 Renesas Advanced Smart Configurator

The Renesas Advanced Smart Configurator (RASC) implements a project generator that allows TrustZone and Non-TrustZone template projects to be conveniently generated.

2.1.1 Using RASC with Renesas e² studio

RASC is natively integrated with Renesas e² studio IDE.

Section 4 explains how to use the Smart Configurator to start TrustZone development.

2.1.2 Using RASC with IAR Embedded Workbench for Arm

Create the initial secure project using RASC and choose the IAR Compiler. This process will generate the initial secure project for IAR EWARM. Once the initial IAR EWARM project is generated, the user can open this project from the IAR EWARM IDE.

Next, the user should follow the `rasc_quick_start.html` file, which is installed under `\<RASC installation root>\eclipse\`. Refer to the `rasc_quick_start.html` section, *Adding tools to a third-party IDE* to integrate RASC and the Renesas Device Partition Manager into the IAR EWARM IDE.

Once RASC is integrated into IAR EWARM, you can open RASC within the IAR EWARM IDE to further develop the TrustZone-based secure and non-secure project application project following the operations explained in section 4.

2.1.3 Using RASC with Arm Keil MDK

The operation of using RASC as well as the Device Partition Manager with Arm Keil MDK to create a TrustZone-based application is identical to the development process for using RASC with IAR EWARM in terms of the general flow. Section 5.6.2 demonstrated the usage of RASC as well as the Renesas Device Partition Manager (RDPM).

2.2 Transitioning from CM State to SSD State

There are some prerequisites prior to setting up the MCU IDAU regions. From the factory, RA MCUs are delivered to the developer in the CM (Chip Manufacturing) lifecycle state. The MCU must be transitioned to SSD (Secure Software Development) lifecycle state prior to setting up the IDAU regions.

Transitioning from CM State to SSD State and setting up the IDAU region can only be achieved using the MCU's boot mode, which can only be accessed using an SCI/USB connection. To benefit from the tools' support, developers need to bring the MCU Mode pin (MD) and SCI pins to the Debug interface. Special debugger firmware has been developed to manage to bring the device up in SCI boot mode to set up the IDAU registers (automatically drives MD pin) and then switch back to debug mode as needed.

Hardware design must reference the EK-RA6M4 debug interface design (signals in red) to provide proper connections to support the above functionality.

Pin No.	SWD	JTAG	Serial Programming using SCI
1	VCC	VCC	VCC
2	P108/SWDIO	P108/TMS	NC
4	P300/SWCLK Wired OR with MD	P300/TCK Wired OR with MD	P201/MD
6	P109/SWO/TXD9	P109/TDO/TXD9	P109/TXD9
8	P110/RXD9	P110/TDI/RXD9	P110/RXD9
9	GNDdetect	GNDdetect	GNDdetect
10	nRESET	nRESET	nRESET

Figure 6. Debug Connection to Support TrustZone® Design

The operational flow when using this feature differs between e² studio and the EWARM IDE.

2.2.1 Developing with e² studio

When developing with e² studio and using Renesas evaluation kits for TrustZone MCUs, the MCU is automatically transitioned from the CM state to the SSD state when the first secure program is downloaded to the MCU if the above required connection is provided.

2.2.2 Developing with IAR EWARM

When developing with IAR EWARM, transitioning from CM to SSD needs to be performed manually using Renesas Device Partition Manager or Renesas Flash Programmer. This is achieved by using the **Initialize device back to the factory default** option, as shown in Figure 7.

2.2.3 Developing with Keil MDK

When developing with Keil MDK, transitioning from CM to SSD needs to be performed manually using Renesas Device Partition Manager or Renesas Flash Programmer. This is achieved by using the **Initialize device back to factory default** option, as shown in Figure 7.

2.3 Setting up the IDAU Region

Whether you are using e² studio or a third-party IDE like Keil MDK or IAR EWARM, you can manually set up the IDAU region using RDPM. As shown in Figure 7, the functionalities of the RDPM are under the **Action** area. To set up the IDAU region, select **Set TrustZone® secure / non-secure boundaries** and provide the IDAU region sizes in the IDAU region configuration area.

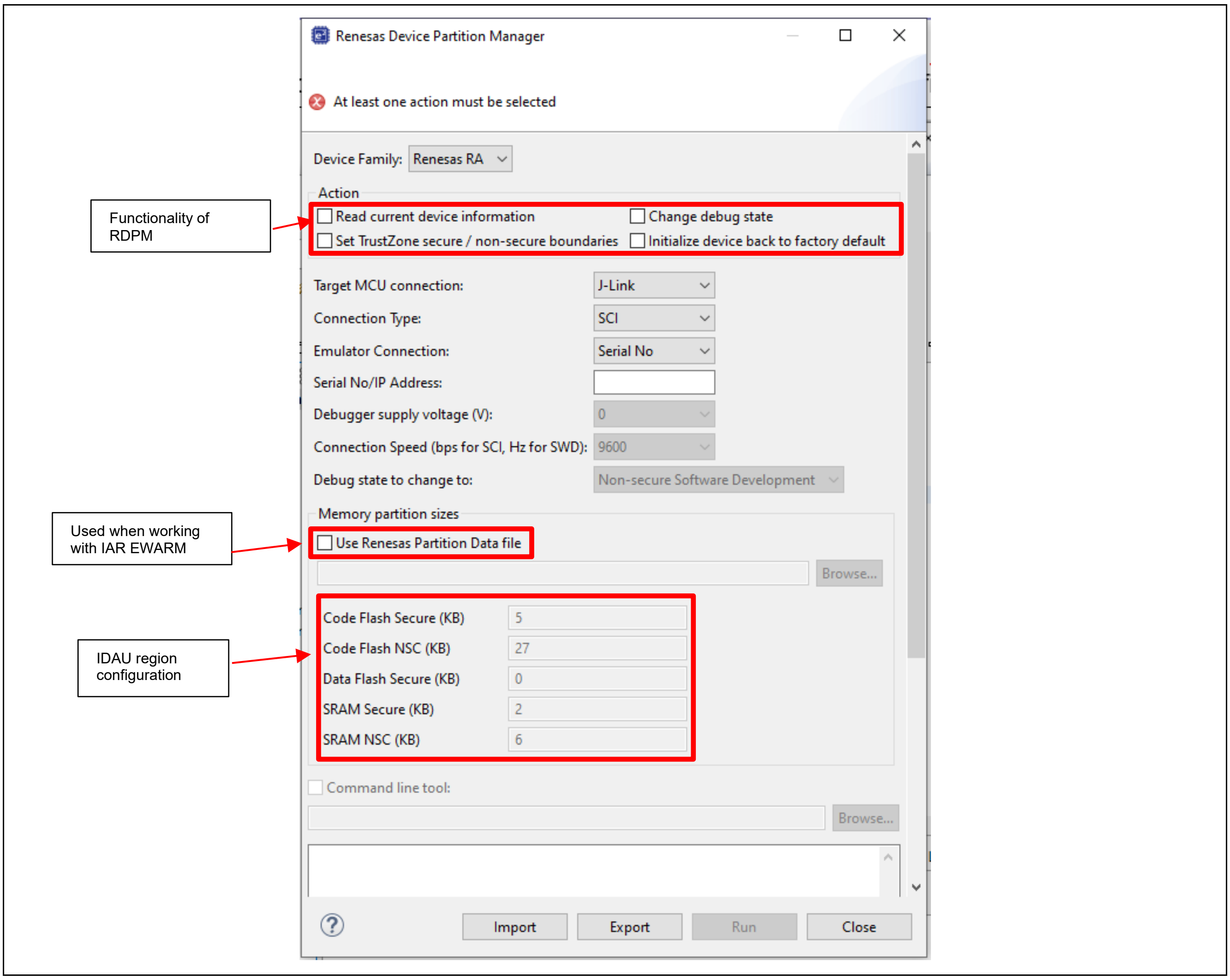

Figure 7. Functionality of RDPM

The RDPM also provides the following functionalities:

- Use **Read current device information** to read out the DLM and IDAU region setup information.
- Use **Change device lifecycle management state** to transition to a different state.
- Use **Initialize device back to factory default** to transition the DLM state to SSD if the device is in NSECSD or DPL state.

When using e² studio, the IDAU region configuration is automatically loaded in the dialog box, and no additional actions are needed to fill in the configuration data.

Pay special attention to the check box for the **Use Renesas Partition Data file**. This check box is used when setting up the IDAU region using IAR EWARM. You must use the generated `.rpd` fie to configure the IDAU region. This usage is described in section 5.5. Once an `.rpd` file is selected, the new IADU region configuration information will be updated automatically based on the `.rpd` file.

Note: The `.rpd` filename is stored for future runs. When switching to another project, you must reselect the `.rpd` file.

The operational flow for using the RDPM differs between e² studio, EWARM IDE, and Keil MDK, as detailed in the following sections.

2.3.1 Developing with e² studio

When using e² studio, the necessary values to set up the TrustZone® memory partition (IDAU registers) are calculated after the binary code to program into the Secure region is created by building the Secure project. The regions are set up to ensure that they match the code and data sizes and keep the attack surface as small as possible. If the hardware connection mentioned in Figure 6 is provided in the PCB design, there is no need to use the RDPM manually to set up the IDAU region. Setting up the IDAU region when developing with e² studio is a transparent process for most applications.

2.3.2 Developing with IAR EWARM

Unlike e² studio, setting up the IDAU when developing with IAR EWARM needs to be performed semi-manually using the RDPM. As part of the debug configuration generated when the RASC creates a project for EWARM, there is the invocation of a C-SPY macro file called `partition_device.mac`, as shown in Figure 8.

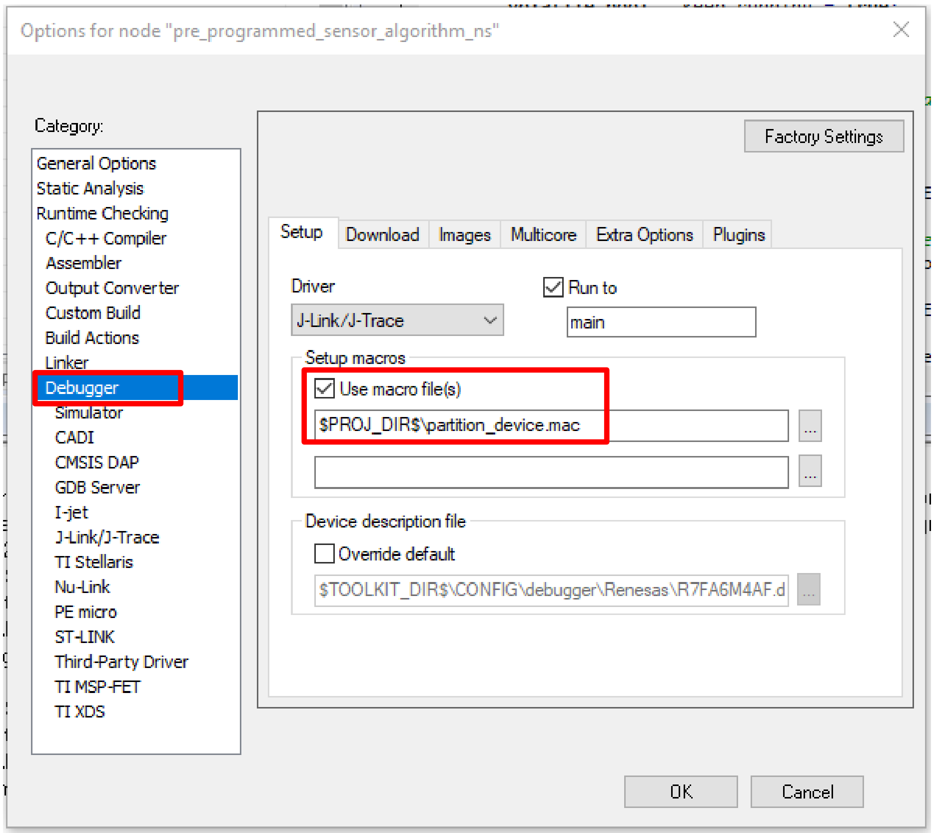

Figure 8. Debug Configuration for IDAU Region Setup

As part of the debug startup sequence, this file will invoke the RDPM integrated to check the target MCU's TrustZone partition boundaries and compare them against the settings calculated as part of the project build sequence. If a mismatch is found, a dialog is displayed asking you whether to reconfigure the IDAU region. You can then choose to launch the RDPM and set up the IDAU regions.

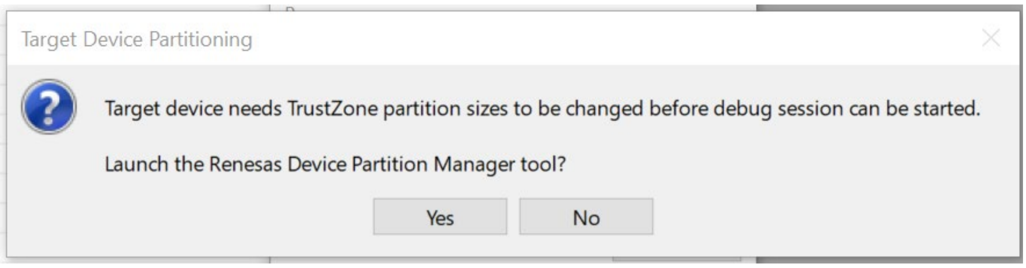

Figure 9. Prompt to Launch the Renesas Device Partition Manager

2.3.3 Developing with Keil MDK

Unlike e[2] studio, setting up the IDAU when developing with Keil MDK needs to be performed manually using the RDPM. The walk-through of setting up the IDAU region when working with Keil MDK is demonstrated in section 5.6.1.

3. General Considerations in TrustZone® Application Design

3.1 Non-secure Callable Modules

Some driver and middleware stacks in the Secure project may need to be accessed by the Non-secure partition. To enable the generation of NSC veneers, set **Non-secure Callable** from the right-click context menu for the selected modules in the Configurator.

Note: It is only possible to configure the top of stacks as NSC.

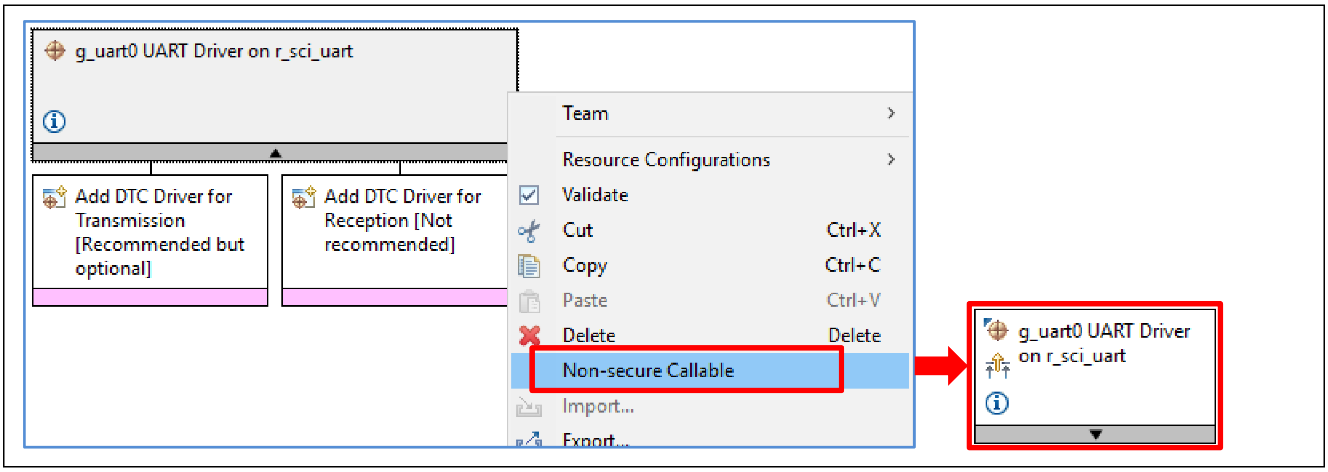

Figure 10. Generate NSC Veneers

3.2 Guard Function for Non-secure Callables

Access to NSC drivers from a Non-secure project is possible through the Guard APIs. The FSP automatically generates guard functions for all the top-of-stack/driver APIs configured in the Secure project as Non-secure Callable.

Some best practices and guidelines for using the guard functions are listed as follows:

3.2.1 Limit Access to Selected Configurations and Controls

The default guard functions generated ignore `p_ctrl` and `p_cfg` arguments sent in from the NS side. Instead, the guard function provides static Secure region instances of these data structures based on the module Instance.

```
BSP_CMSE_NONSECURE_ENTRY fsp_err_t g_uart0_open_guard(
            uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) {
      /* TODO: add your own security checks here */

      FSP_PARAMETER_NOT_USED(p_api_ctrl);
      FSP_PARAMETER_NOT_USED(p_cfg);

      return R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
}
```

Figure 11. Example Guard Function

3.2.2 Test for Non-secure Buffer Locations

- If the Non-secure region is providing input (such as by calling the `write()` function with the data buffer), then the guard functions should check that the data buffer is entirely within an NS area.
- If the Non-secure region is providing a pointer to store output (such as by calling the `read()` function with a pointer of where to store), then the guard functions should check that the data buffer is entirely within an NS area.

See section 3.5.1 for examples of using the CMSE library to handle this requirement.

3.2.3 Handle Non-secure Data Input Structure as Volatile

If a Non-secure region is providing a data structure as input (for example, a `typedef'd` structure with 3 members), then guard functions should make a copy of the data structure in the Secure region before passing it to the Secure function. This is done because the Non-secure data structure should be seen as volatile, and the Non-secure region could alter contents after invoking the NSC function.

See section 3.5.2 for an example of how to handle this requirement.

3.2.4 Limit the Number of Arguments in an NSC Function

The compiler uses registers R0 to R3 to pass parameters and return values. Registers R4 to R12 are used during function execution. The called function restores registers R4 to R12. Therefore, if an NSC API is being used for a Secure function with more than 4 arguments, the guard function should define a function with a different prototype that will be a funnel to handle all of the arguments. The new function prototype should take a data structure that has members to cover all parameters in the Secure function. This means that Non-secure code will need to put the function arguments into the structure. The guard function will then expand the data structure into separate arguments and pass them to the Secure function.

Figure 12 shows an FSP example for funneling the 5 arguments from the `R_SPI_WriteRead` function to 4 arguments in the NSC API guard function.

```
/** Non-secure arguments for write-read guard function */
typedef struct st_spi_write_read_guard_args
{
    void const          * p_src;
    void                * p_dest;
    uint32_t const        length;
    spi_bit_width_t const bit_width;
} spi_write_read_guard_args_t;


/* This function has been modified to reduce the number of arguments. */
BSP_CMSE_NONSECURE_ENTRY fsp_err_t g_spi0_write_read_guard_fanin(spi_ctrl_t *const p_api_ctrl,
        spi_write_read_guard_args_t *args)
{
    /* Verify all pointers are in non-secure memory. */
    spi_write_read_guard_args_t *args_checked = cmse_check_pointed_object (args, CMSE_AU_NONSECURE);
    FSP_ASSERT (args == args_checked);
    void const *p_src_checked = cmse_check_address_range ((void*) args_checked->p_src, args_checked->length,
                                                            CMSE_AU_NONSECURE);
    FSP_ASSERT (args_checked->p_src == p_src_checked);
    void *p_dest_checked = cmse_check_address_range (args_checked->p_dest, args_checked->length, CMSE_AU_NONSECURE);
    FSP_ASSERT (args_checked->p_dest == p_dest_checked);

    /* TODO: add your own security checks here */

    FSP_PARAMETER_NOT_USED (p_api_ctrl);

    return R_SPI_WriteRead (&g_spi0_ctrl, p_src_checked, p_dest_checked, args_checked->length, args_checked->bit_width);
}
```

Figure 12. Handling Secure Functions with More than 4 Arguments

3.3 Creating User-Defined Non-secure Callable Functions

For IP protection purposes, you can create a customized NSC API in the Secure project to expose only the top-level control of your algorithms and store the IP in the Secure Arm® TrustZone® region. Precautions mentioned previously should be exercised during the creation of the user-defined NSC API.

The steps to create a customized NSC API are:

1. Create the Non-secure Callable custom function by declaring the function with `BSP_CMSE_NONSECURE_ENTRY`.
2. Create a header file that includes all the customized NSC function prototypes, for example, `my_nsc_api.h`.
3. Include the path to the NSC header using the Build Variable as shown in Figure 13.
4. Compile the Secure project to create the Secure bundle. The NSC header will be automatically extracted for use in the Non-secure project.

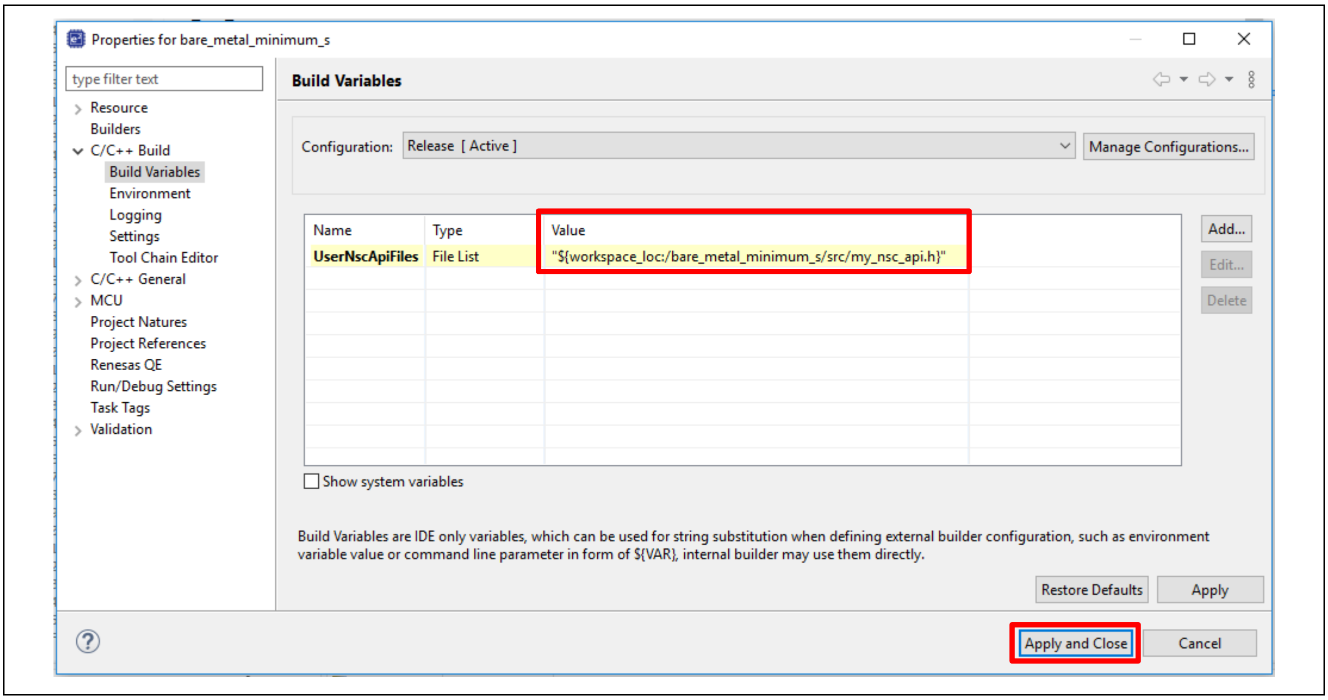

Figure 13. Link User-Defined Non-secure Callable API Header File

3.4 RTOS Support

Renesas tooling and the FSP support Non-secure partition RTOS integration with Secure region access through Non-secure callable APIs. Secure projects can use the **Secure TrustZone Support – Minimum** project type to add the Arm® TrustZone® Context RA port. For operation details, see section 4.1.1, Step 3 for Secure Project handling and section 4.1.2, Step 5 Non-secure Project Handling.

3.5 Writing TrustZone Technology-Enabled Software

Security design using TrustZone technology has some specific challenges that secure developers should bear in mind and take corresponding actions when writing secure application software.

This section provides several guidelines that secure software developers should consider following in order to avoid Secure information leaks to the Non-secure region.

3.5.1 Benefitting from CMSE Functions to Enhance System-Level Security

This subsection discusses how to benefit from the CMSE library to improve the secure software design. Some examples of the CMSE functions are:

- `cmse_check_address_range`: For example, this function can be used to confirm the address range is entirely in the Non-secure region.
- `cmse_check_pointed_object`: For example, this function can be used to confirm the memory pointed to by the pointer is entirely in the Non-secure region.

```
BSP_CMSE_NONSECURE_ENTRY fsp_err_t g_uart0_read_guard(uart_ctrl_t *const p_api_ctrl,
uint8_t *const p_dest,
        uint32_t const bytes)
{
    /* Verify all pointers are in non-secure memory. */
    uint8_t *const p_dest_checked = cmse_check_address_range ((void*) p_dest, bytes,
CMSE_AU_NONSECURE);
    FSP_ASSERT (p_dest == p_dest_checked);

    /* TODO: add your own security checks here */

    FSP_PARAMETER_NOT_USED (p_api_ctrl);

    return R_SCI_UART_Read (&g_uart0_ctrl, p_dest_checked, bytes);
}
```

Figure 14. Non-secure Buffer Address Range Check

3.5.2 Avoid Asynchronous Modifications to Currently Processed Data

An example of handling is shown in Figure 15. When the pointer `p` points to Non-secure memory, it is possible for its value to change after the memory access is used to perform the array bounds check but before the memory access is used to index the array. Such an asynchronous change to Non-secure memory would render this array bounds check useless.

```
int array[N];
void foo(volatile int *p)
{
      int i = *p;
      if (i >= 0 && i < N) { array[i] = 0; }
}
```

Figure 15. Treat Non-secure Data as Volatile in Secure Code

3.5.3 Utilize the Armv8-M Stack Pointer Stack Limit Feature

The Armv8-M architecture introduces stack limit registers that trigger an exception on a stack overflow.

CM23 with Arm® TrustZone® technology has two stack limit registers in the Secure state:

- Stack Limit Register for Main Stack: MSPLIM_S
- Stack Limit Register for Process Stack: PSPLIM_S

CM33 with TrustZone technology has two stack limit registers in the Secure state and two stack limit registers in the Non-secure state:

- Stack Limit Register for Main Stack in Secure state: MSPLIM_S
- Stack Limit Register for Process Stack in Secure state: PSPLIM_S
- Stack Limit Register for Main Stack in Non-secure state: MSPLIM_NS
- Stack Limit Register for Process Stack in Non-secure state: PSPLIM_NS

Users can implement customized fault handlers to catch the stack limit overflow error.

Refer to the *Arm®v8-M Architecture Reference Manual* section, *The Armv8-M Architecture Profile,* for more information on the functionality of the stack limit registers.

4. Using Renesas RA Project Generator for TrustZone Development

The RASC is designed for TrustZone technology-based applications. It provides ease of use based on the following implementation features from the tools and FSP point of view:

- RA Project Generator guides you through the TrustZone project creation process.
- TrustZone IDAU region setup during Secure program download, calculated automatically based on the Secure project. See section 2.1 for more details.
- The FSP provides a quick and versatile way to build secure connected IoT devices using Renesas RA MCUs.

Note: FSP version information is removed from the following screen captures because these instructions apply to all FSP versions 5.0.0 or later.

RA Project Generator

The RA Project Generator provides three project types to create the initial template projects for developing with Arm® TrustZone® technology-enabled MCUs:

- A **Secure Project** and **Non-secure Project Type** pair, which work with the Secure and Non-secure partitions, respectively.
- A **Flat Project** with which an application can be developed with no TrustZone partition awareness.
- Whether developing with a TrustZone-enabled project or with a Flat project, the MCU needs to transit from the CM state to the SSD state prior to proceeding with the development.

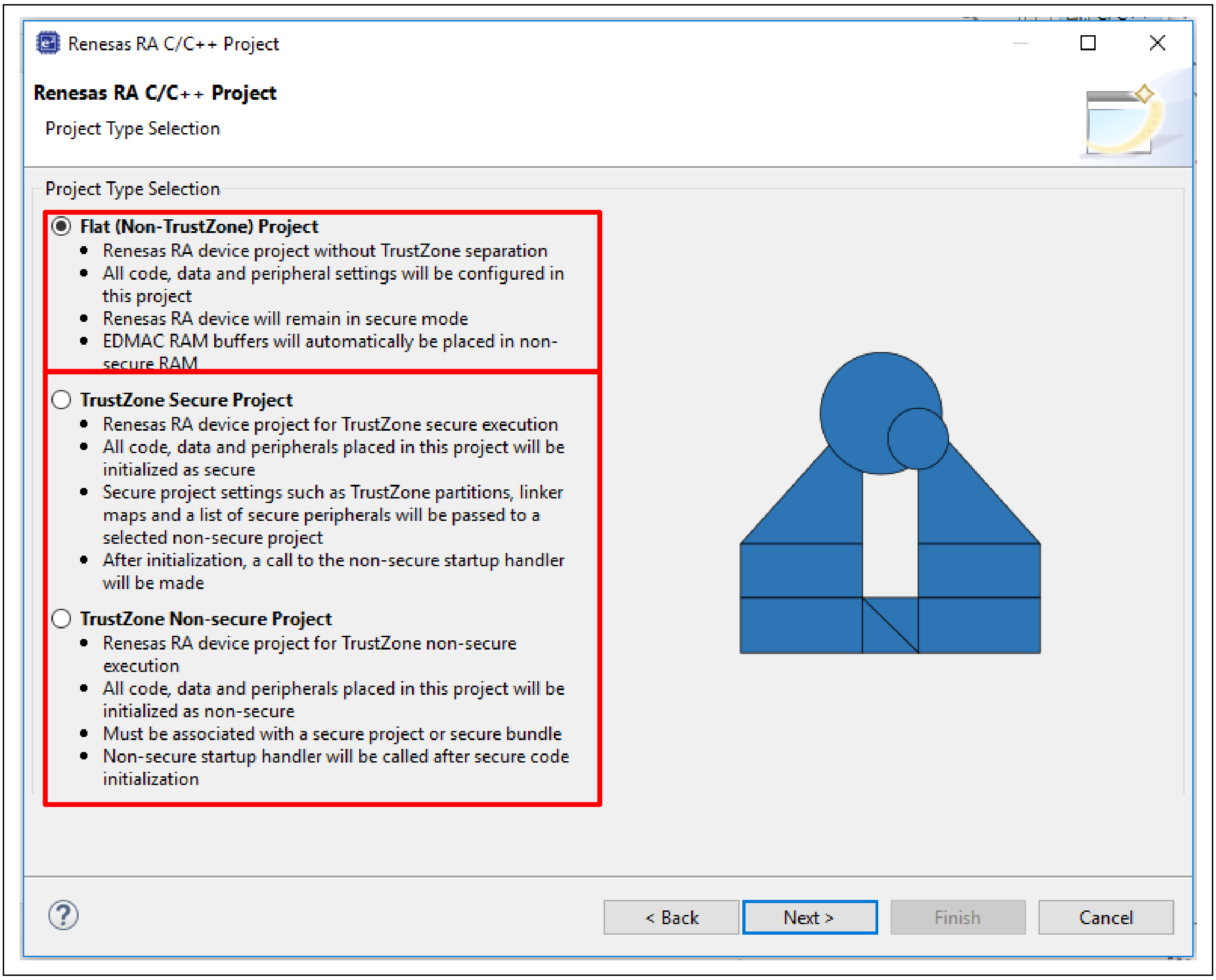

Figure 16. RA Project Generator

For RA TrustZone technology-enabled MCUs, there are two development models:

- Combined Project Development
  - — Secure and Non-secure applications are developed by one trusted team.
- Split Project Development
  - — Secure and Non-secure applications are developed by two different teams.
  - — The Non-secure application team does not have direct access to Secure partition assets. Access to a Secure partition is only possible via Non-secure Callable APIs.

The design process, based on each of these two development models, is introduced in the subsequent subsections. The design process based on the Flat Project type is introduced in section 4.3.

4.1 Combined Project Development

With the Combined Project Development Model, Secure and Non-secure projects are developed by a single trusted team. A Secure project must reside in the same workspace as the Non-secure project and is typically used when a design engineer has access to both the Secure and Non-secure project sources.

In addition, a Secure `.elf` file is referenced and included in the debug configuration for the Debug build for download to the target device. The development engineer has visibility of Secure and Non-secure project source code and configuration.

4.1.1 Developing the Secure Project

Most peripherals and IO defined in the Secure project are configured as Secure with the exceptions of Clock, QSPI, OSPI, and the CS Area. These peripherals can be used in the Secure project and be configured as Non-secure.

The major IDE operational steps in developing the Secure project are explained in the following steps.

Step 1: Create a new project using the RA Project Generator template.

Renesas RA MCU tooling provides several project templates to help kickstart development.

Figure 17 to Figure 21 show some common steps when creating a new project with e[2] studio regardless of whether Secure or Non-secure projects are to be created with either the Split Project Development Model or Combined Project Development Model.

- This step will be referenced in the context of Non-secure Project Development for the Combined Project Development Model.
- This step will be referenced in the context of Secure and Non-secure Project Development for the Split Project Development Model.

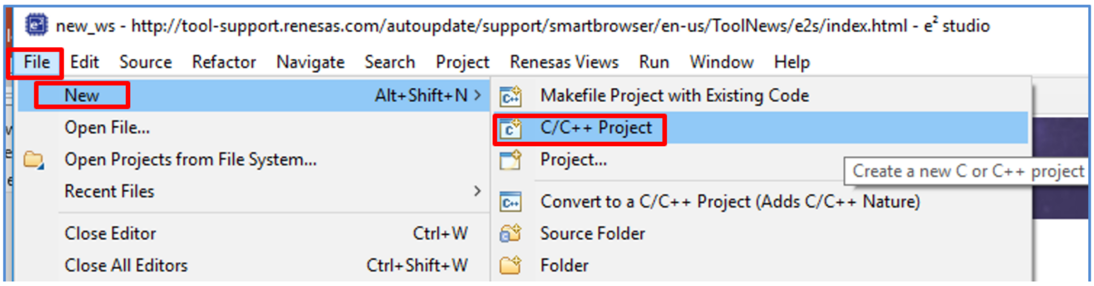

Figure 17. Create New Project

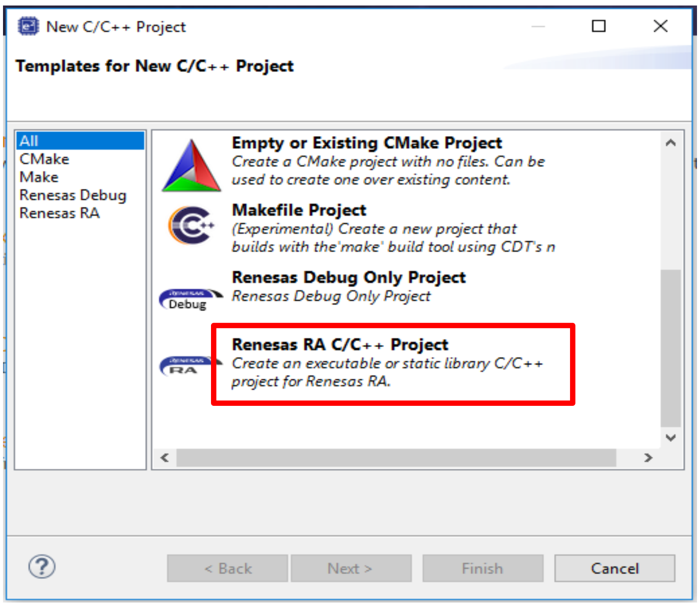

Figure 18. Select "Renesas RA C/C++ Project"

Click **Next**, then provide the Secure project name. It is helpful to attach "_s" (for Secure") and "_ns" (for Non-secure) to the end of the project name as a reminder of the security nature of this project.

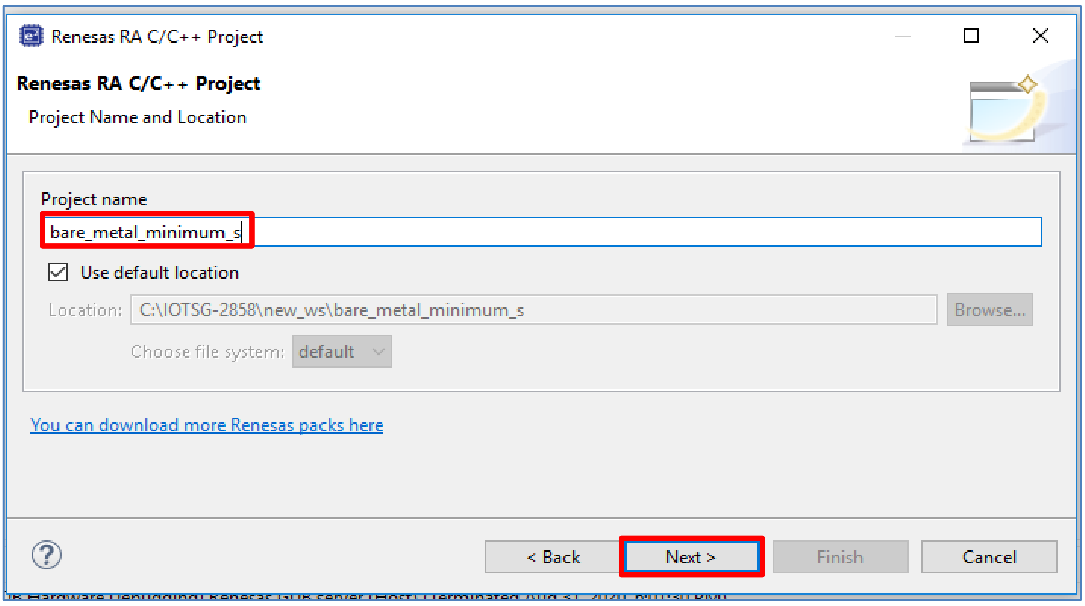

Figure 19. Define the Name of the Secure Project

Click **Next**, then select the EK-RA6M4 BSP.

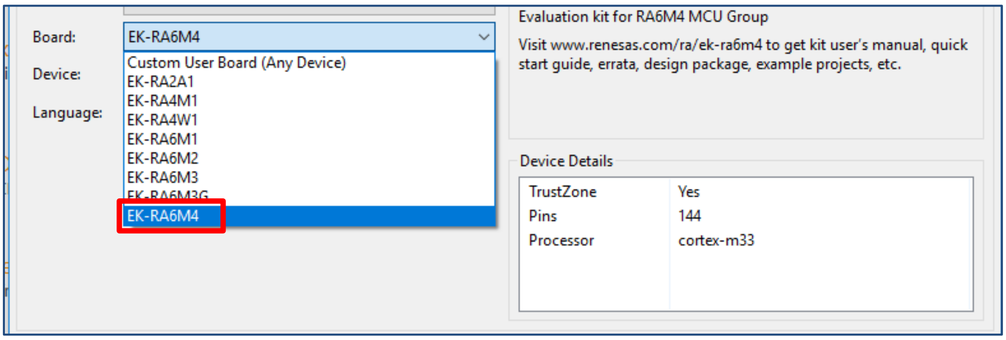

Figure 20. Select the BSP

Note: By default, the BSP functionality with regard to security control is only enabled in the Secure project.

Once the BSP is selected, click **Next** to view the summary for the hardware setup page.

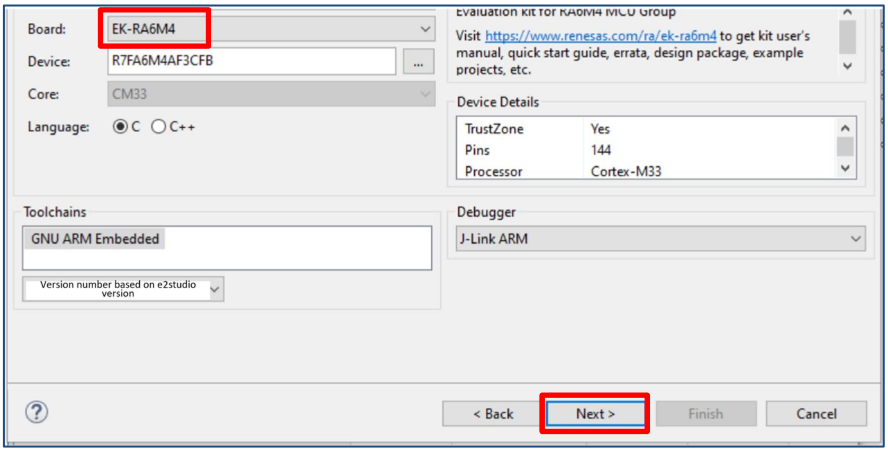

Figure 21. Review the Configurations Prior to Proceeding to Next Step

Click **Next** and proceed to the following steps.

Note: Step 2 to Step 7 below are common for the Split Project Development Model and Combined Project Development Model. These steps are referred to in the context of the Secure Project development for the Split Project Development Model.

Step 2: Choose the TrustZone Secure Project as the Project Type.

Choose **TrutZone Secure Project** as the project type, and take a moment to read the description on this project type. All peripherals initialized in this project will be assumed to have the Secure attribute with the exceptions indicated in Table 3 as **Always Non-secure**. All code and data placed in this project will be initialized as Secure by the FSP BSP, and control will be passed to the Non-secure project reset handler at the end of the Secure project execution.

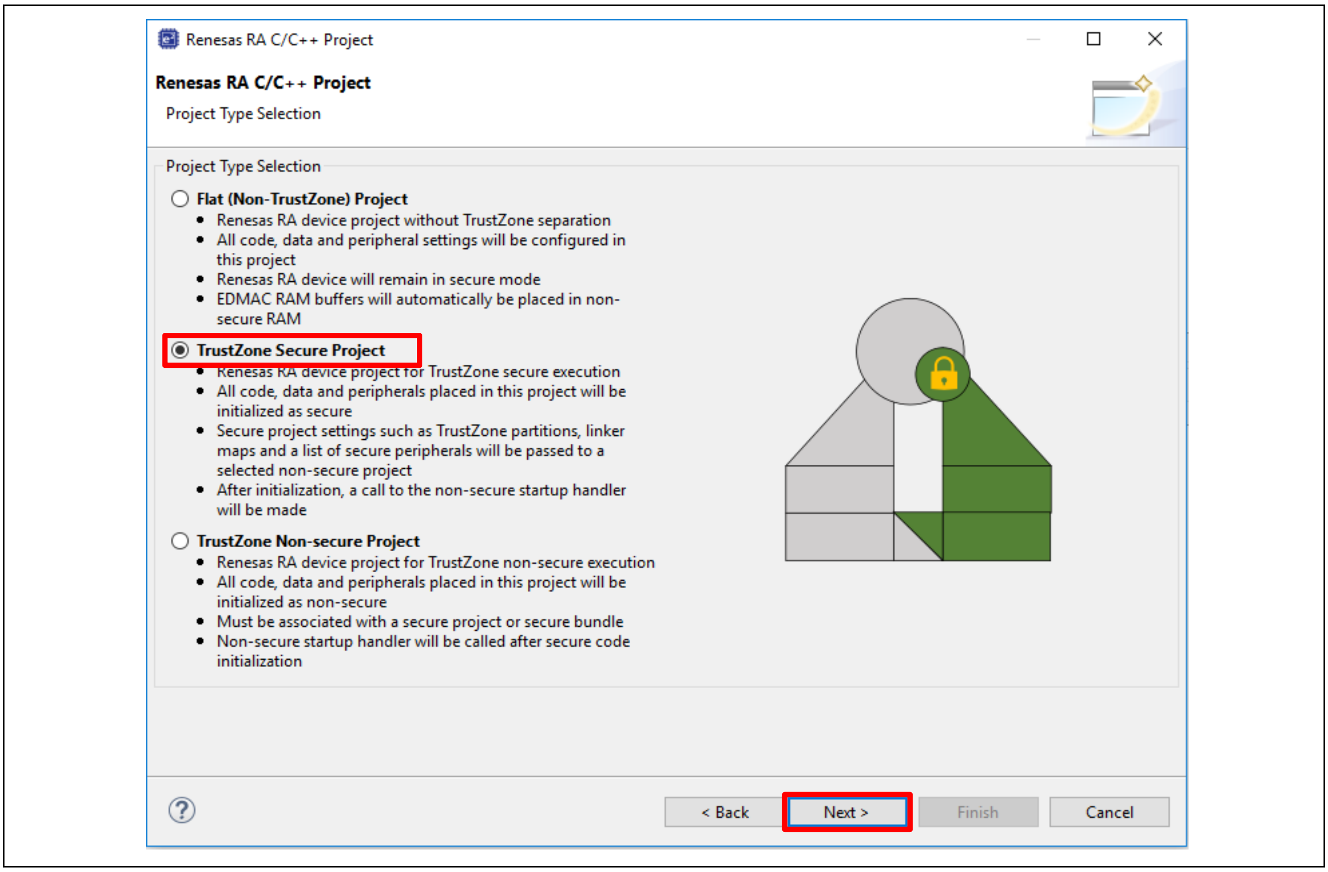

Figure 22. Choose the Secure Project Type

Click **Next** and choose the Project Template.

Step 3: Choose the project template.

As shown in Figure 23, there are two Secure project templates. You can choose which template to use based on whether an RTOS is used in the Non-secure project.

- **Bare Metal – Minimal**
  Secure project with MCU Initialization functions with support for transitioning to a Non-secure partition. This application note uses the **Bare Metal – Minimal** project template as an example to explain the general steps of creating a secure project.
- **TrustZone Secure RTOS – Minimal**
  - Secure projects will add the required RTOS context in the Secure region for the Thread that needs to access the NSC APIs in an RTOS-enabled project. When this project type is selected, the Arm® TrustZone® Context RA Port will be added, as shown in Figure 24.
  - The RTOS kernel and user tasks will reside in the Non-secure partition.

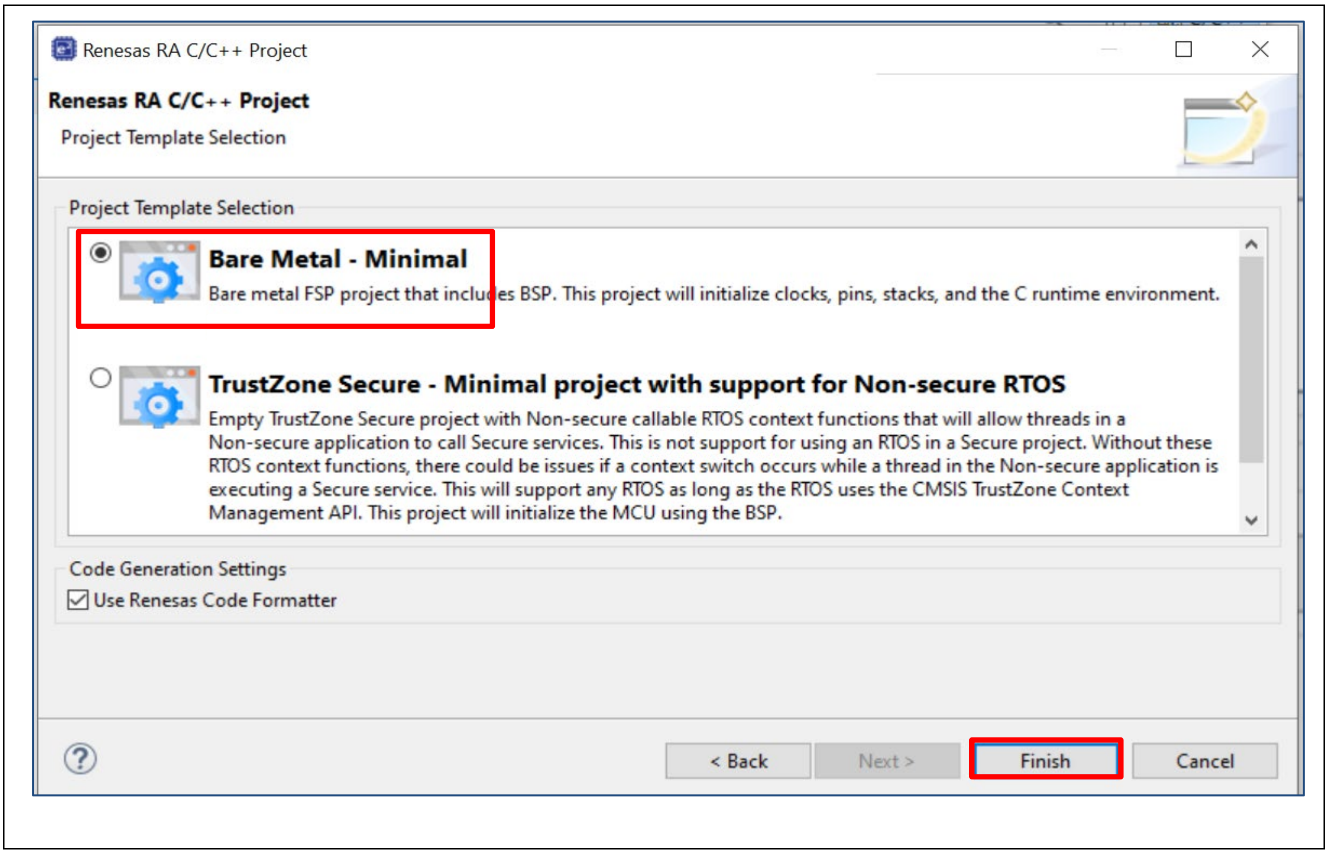

Figure 23. Choose the Project Template

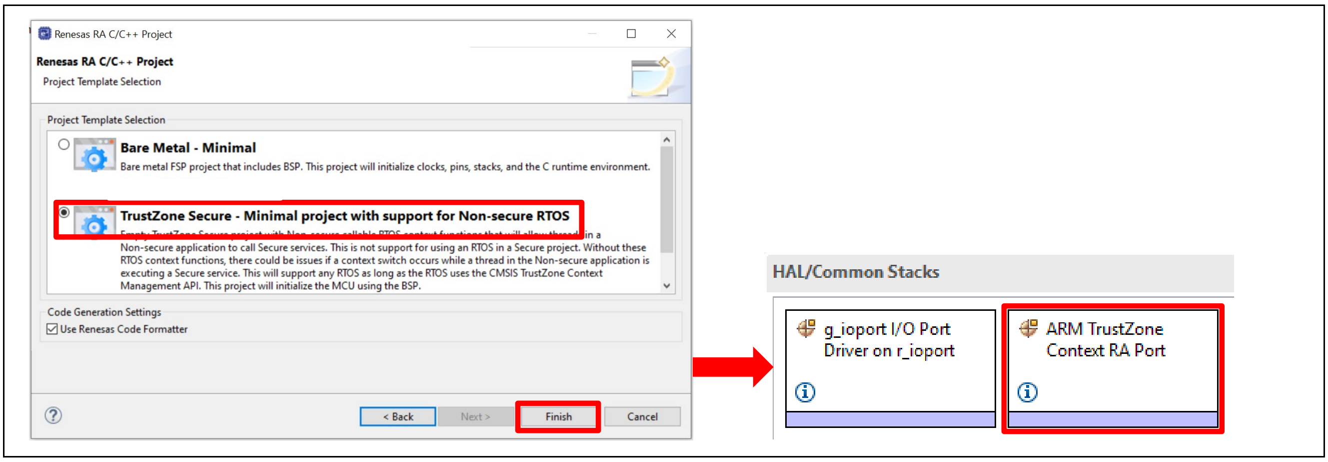

Figure 24. Adding the TrustZone Context RA Port

Click **Finish** to allow the Project Generator to populate the project template.

Notes on Clock Control

The clock is initialized in the Secure project to allow faster start-up. By default, the FSP sets all the security attributes of the Clock Generation Circuit (CGC) to be Non-secure, as shown in Figure 25. Therefore, both Secure and Non-secure projects can change the clock setting.

Users have the option to set all the security attributes of CGC as Secure; thus the Non-secure project developer cannot override the secure project setting, as shown in Figure 26.

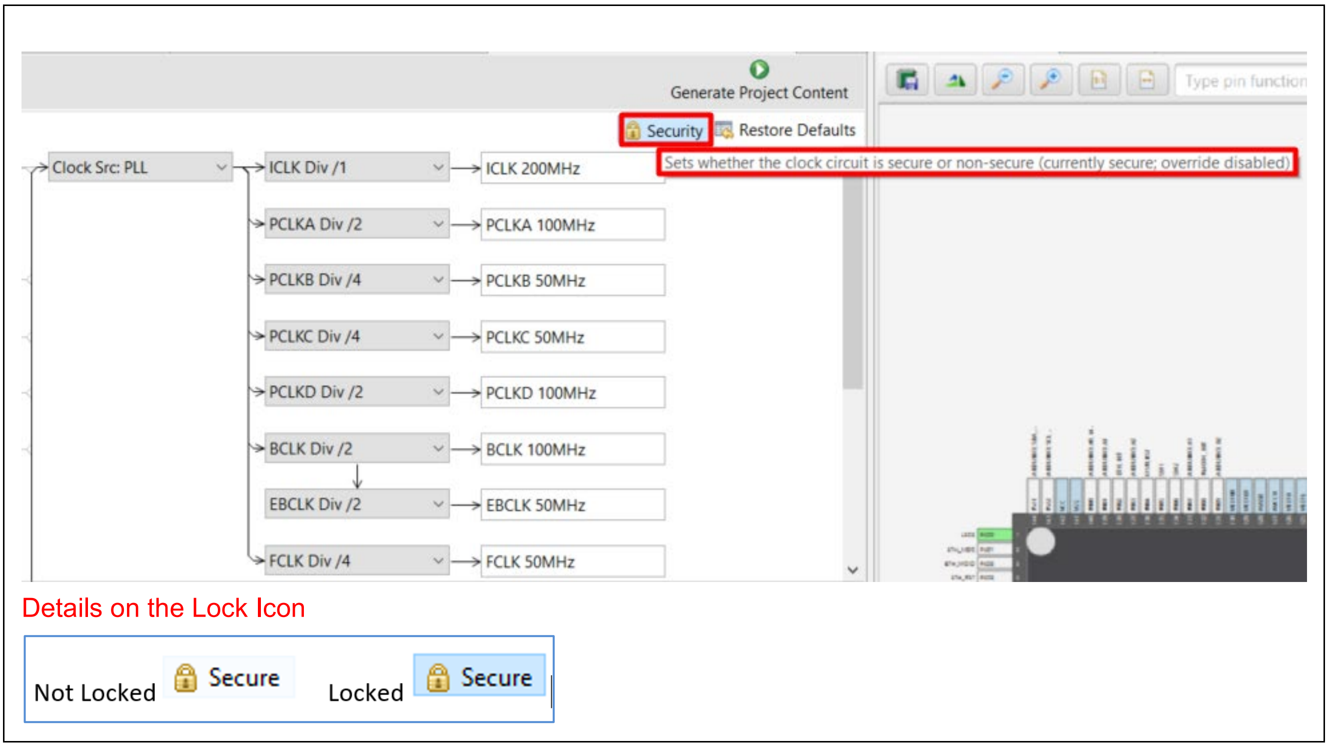

Figure 25. Secure Project Sets Clock as Secure

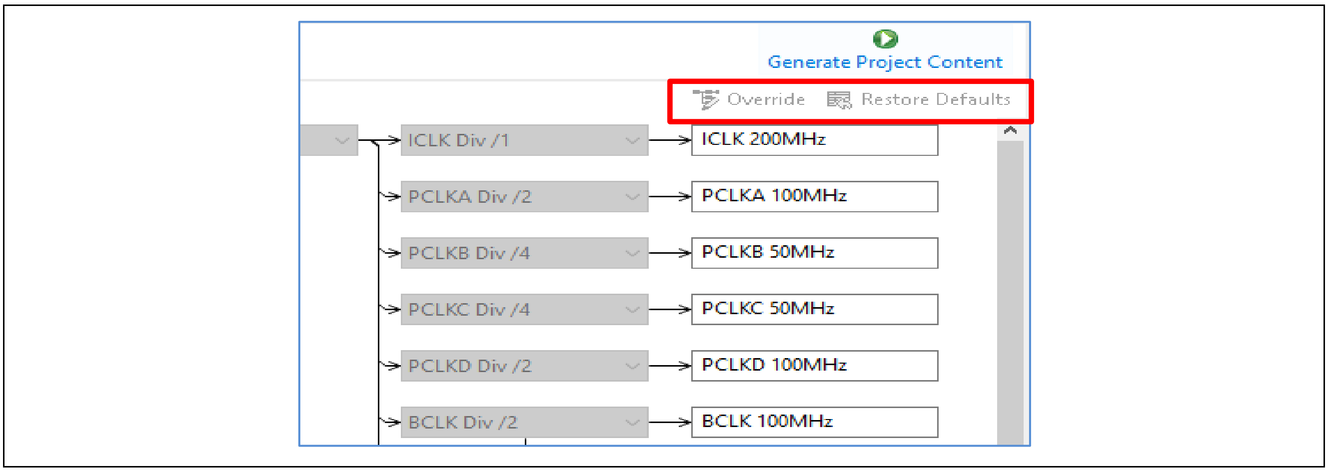

Figure 26. Non-secure Project Clock control "Override and Restore Default" Disabled

Step 4: Generate Project Content and compile the project template.

Double-click `configuration.xml` to open the configurator. Click **Generate Project Content** as shown in Figure 27.

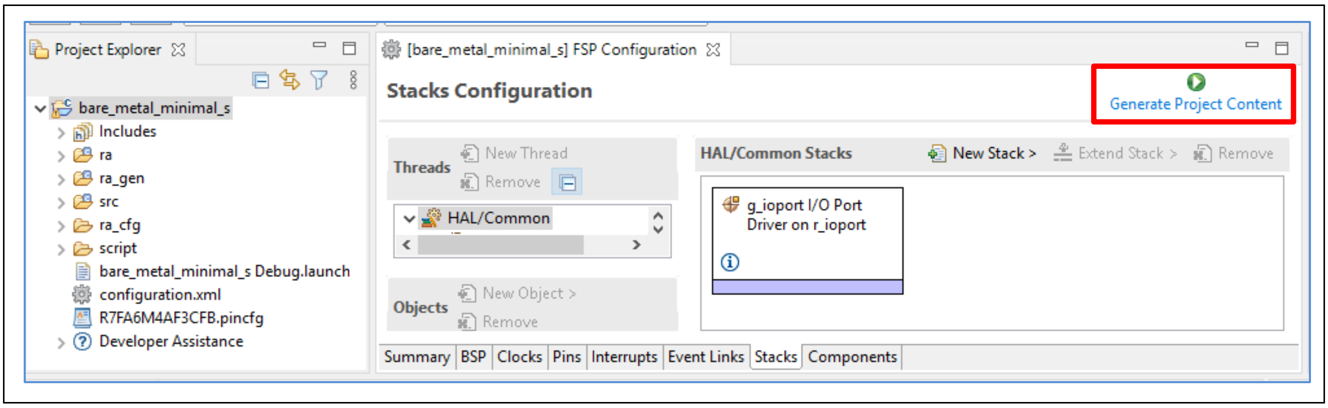

Figure 27. Generate Project Content

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

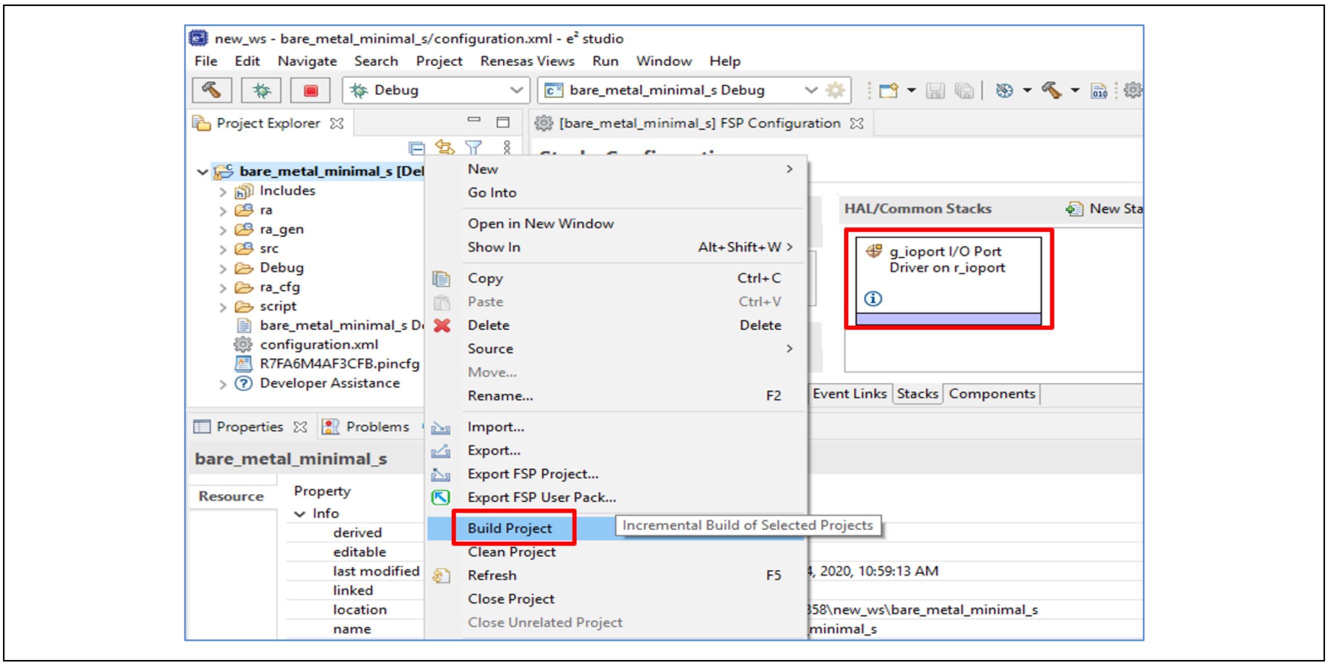

Figure 28. Compile the Template Project

Note: By default, the GPIO driver to control the Secure GPIO pins is included in the template. You can remove the GPIO driver if it is not needed to reduce the project footprint.

Figure 29 is an example of the compilation result based on the **Bare-Metal Minimum** project template.

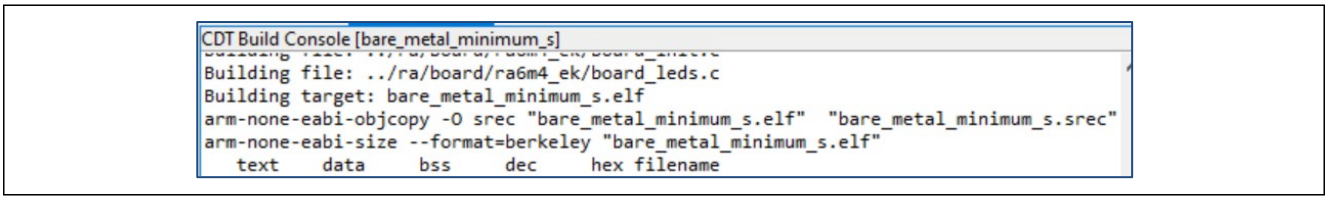

Figure 29. Compilation Result of the Bare-Metal Minimum Secure Template Project

Step 5: Review the initial Secure bundle generated.

After successful compilation, the Secure bundle <project_name>.sbd is generated, as shown in Figure 30.

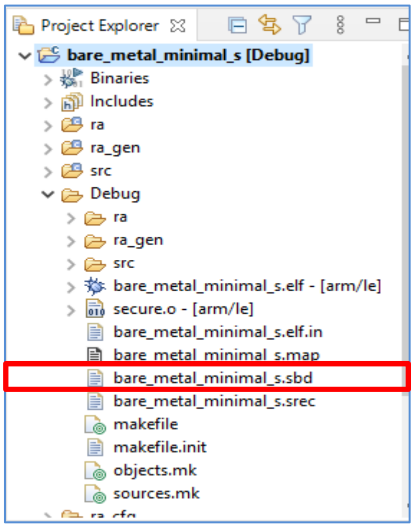

Figure 30. Secure Bundle Generated

Step 6: Develop the Secure application.

During the product development, it is likely that you will go through the following steps iteratively prior to completing development:

- Add Needed FSP Modules:
  - Define NSC Modules if needed. See Section 3.1 for details.
  - Note: Ethernet cannot be used in the Secure Project. It is only available for Non-secure Projects.
- Create user-defined Non-secure Callable Functions if needed. See section 3.3 for details.
- Develop the Secure applications:
  - Design the code flow such that the Secure applications that are not Non-secure Callable are executed prior to starting the Non-secure project execution: prior to function call `R_BSP_NonSecureEnter();`
- Recompile and test the application.

Step 7: Debug the Secure project in isolation.

With the Combined Project Development Model, the Secure project is typically not debugged in isolation from the Non-secure project. To debug a Secure project on its own, you can use the following options:

- Prepare a "dummy/test" Non-secure project. This approach offers the benefits of allowing the Non-secure Callable APIs to be debugged in the test Non-secure project.
- Replace `R_BSP_NonSecureEnter();` with `while(1);` in `hal_entry.c` and debug the Secure project by itself. Be sure to restore the `R_BSP_NonSecureEnter()` after debugging the Secure project prior to provisioning the Secure project to the MCU.

Step 8: Debug the Secure project with the Non-secure project.

For the Combined Project Development Model, Secure and Non-secure project development can be debugged in one workspace. Debugging the Secure project typically does not happen in an isolated manner for the Combined Project Development Model. See Section 4.1.2, Step 7 for operational details.

4.1.2 Developing the Non-secure Project

Once the Secure template project is established and compiled, you can start the Non-secure template project creation in the same workspace where the Secure project resides.

Step 1: Follow Step 1 in section 4.1.1 to start a new Non-secure project.

It is helpful to attach "`_ns`" to the end of the project name as a reminder of the security configuration of this project.

Step 2: Choose Non-secure project as the Project Type.

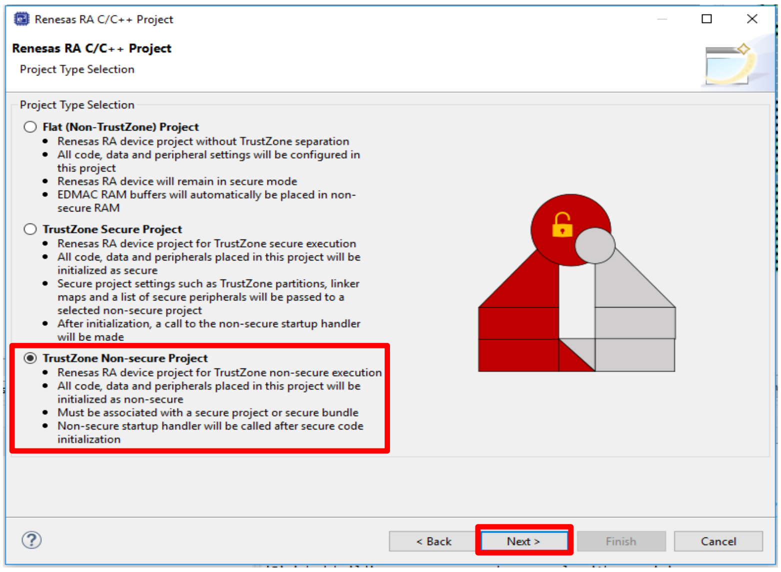

Figure 31. Choose Non-secure Project as Project Type

Step 3: Establish linkage to the Secure project, which resides in the same e² studio workspace.

Click the down arrow and select the secure project **bare_metal_minimum_s** created in section 4.1.1.

Note: The Secure project must exist in the same workspace AND be open for it to be referenced in the selection box. The Secure project must also be built to create the information used to set up the Non-secure project.

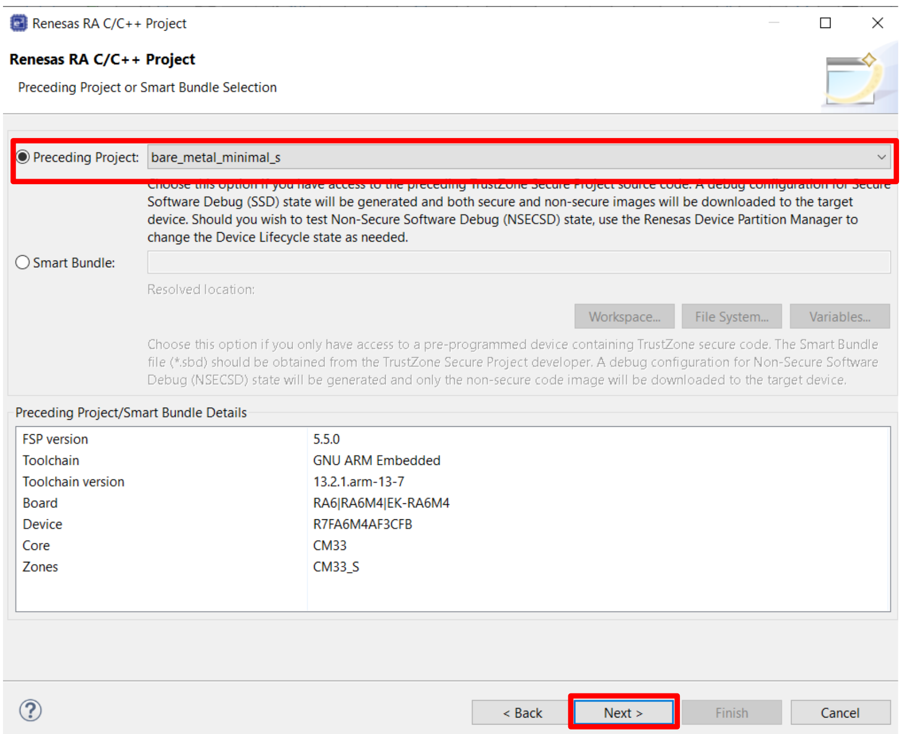

Figure 32. Establish Linkage to the Secure Project

Click **Next** to proceed.

Step 4: Follow the prompt as shown below to choose whether the Non-secure project will have RTOS support.

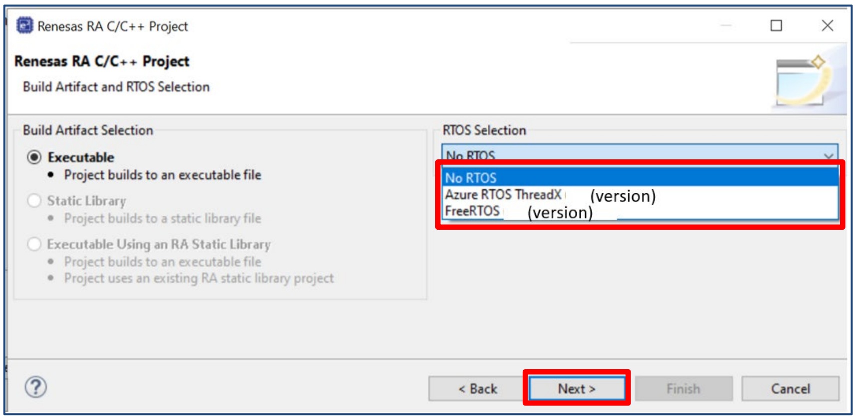

Figure 33. Choose Whether to Use FreeRTOS in the Non-secure Project

Click **Next** to proceed.

Step 5: Select the project template to finish creating the Non-secure template project.

- If FreeRTOS is selected, the Project Generator provides the following two project templates. Choose the project template based on the application needs. An example of FreeRTOS is shown as follows. Azure RTOS has similar options.

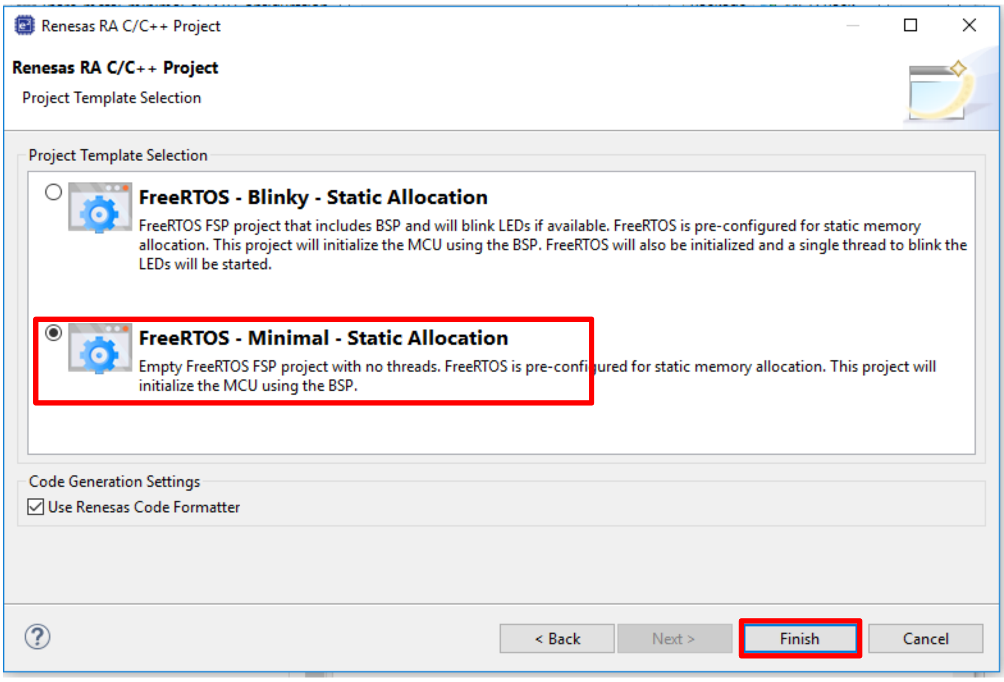

Figure 34. Template Options for FreeRTOS Enabled Projects

Note: If FreeRTOS is selected and there is access to NSC functions from a Thread in the Non-secure project, it is necessary to enable **Allocate secure context for this thread** in the configurator for that Thread.

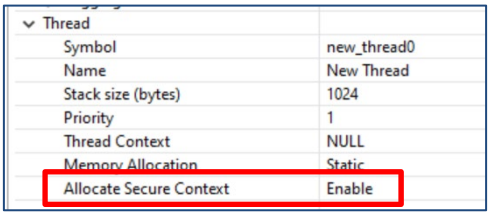

Figure 35. Enable Secure Context Allocation

- If **No RTOS** is selected, the Project Generator provides the following two project templates.
  Note: The **No RTOS** selection must be selected if a new RTOS other than FreeRTOS is to be integrated into the Non-secure project.

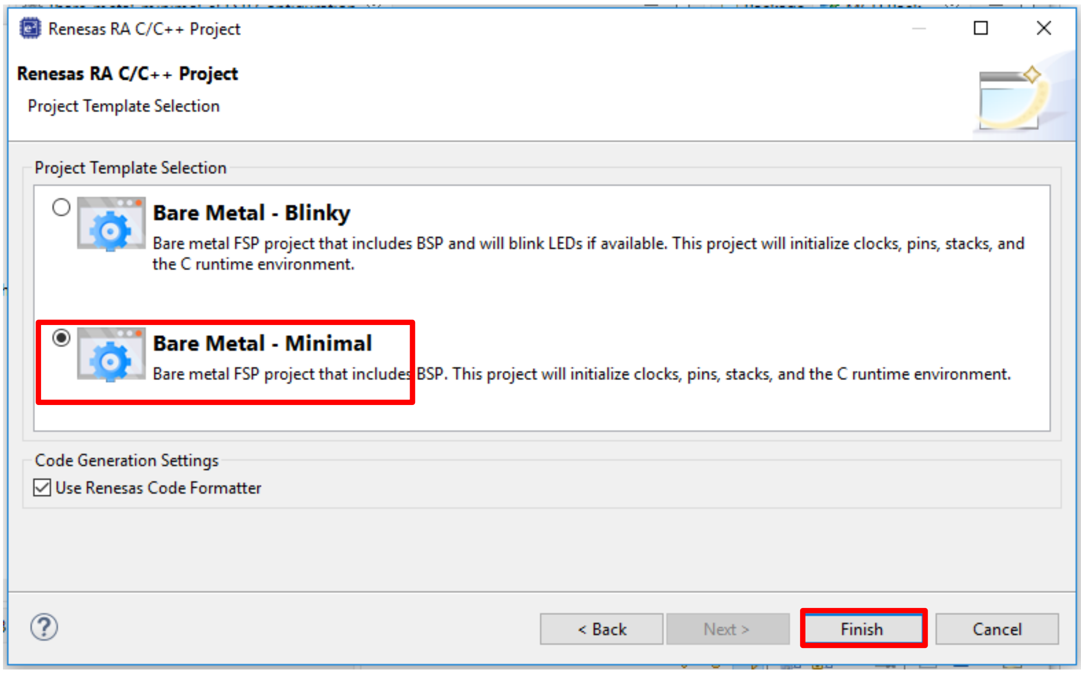

Figure 36. Template Options for Non-FreeRTOS Usage

- Click **Finish** to create the corresponding template project.
  Note: Even though there are security properties allowed for configuration on the BSP **Properties** page, they are not being enabled with the current IDE support. The following attributes cannot be configured from the Non-secure project:

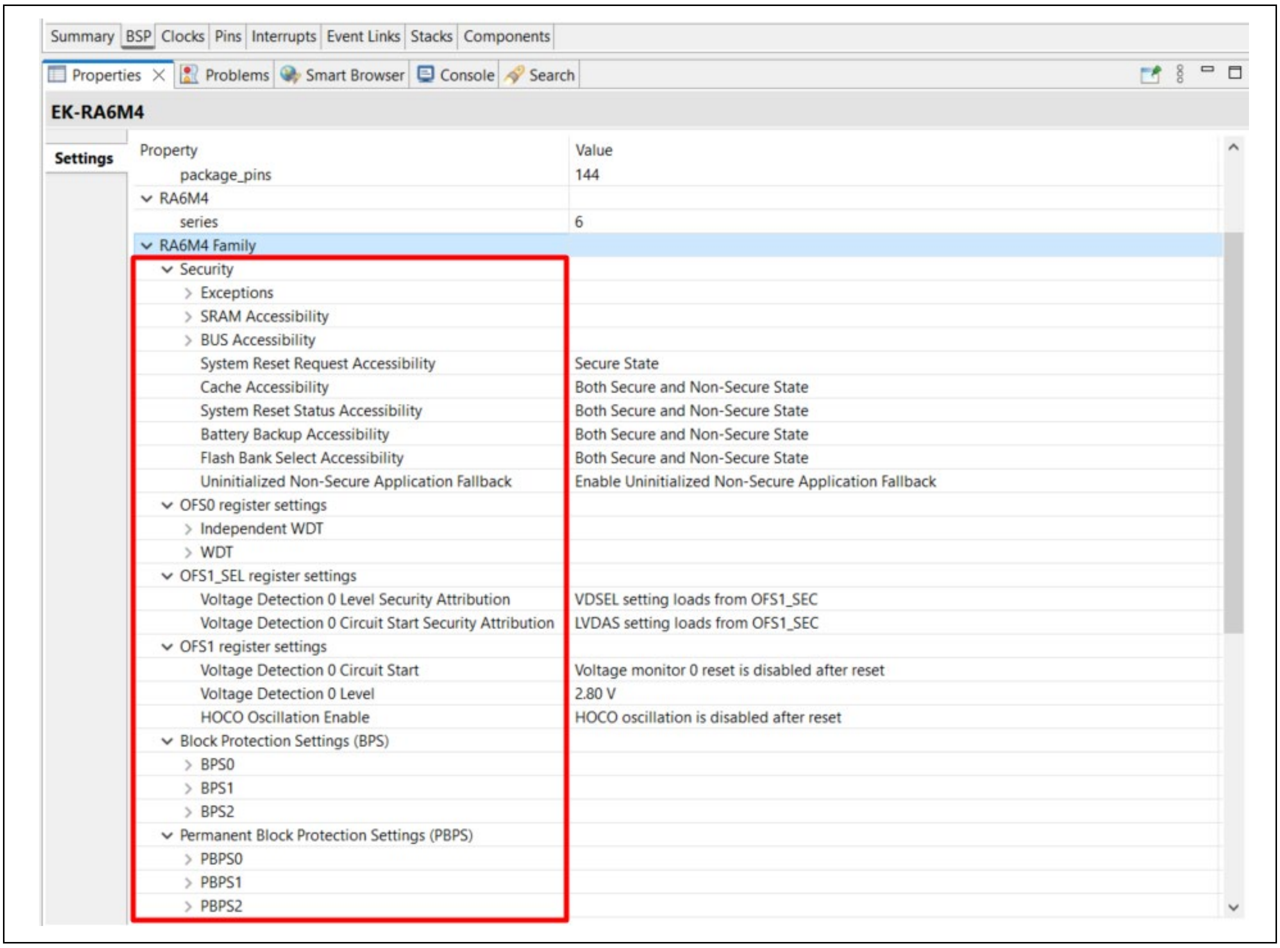

Figure 37. Attributes That Are Not Configurable from a Non-secure Project

- By default, the Non-secure project BSP can reconfigure the MCU clock. Refer to Notes on Clock Control.

Step 6: Follow the Instructions from Step 1, Section 4.1.1 to Generate Project Content and compile Non-secure projects.

Notice that both the Secure project `bare_metal_minimun_s` and `bare_metal_minimum_ns` reside in the same workspace.

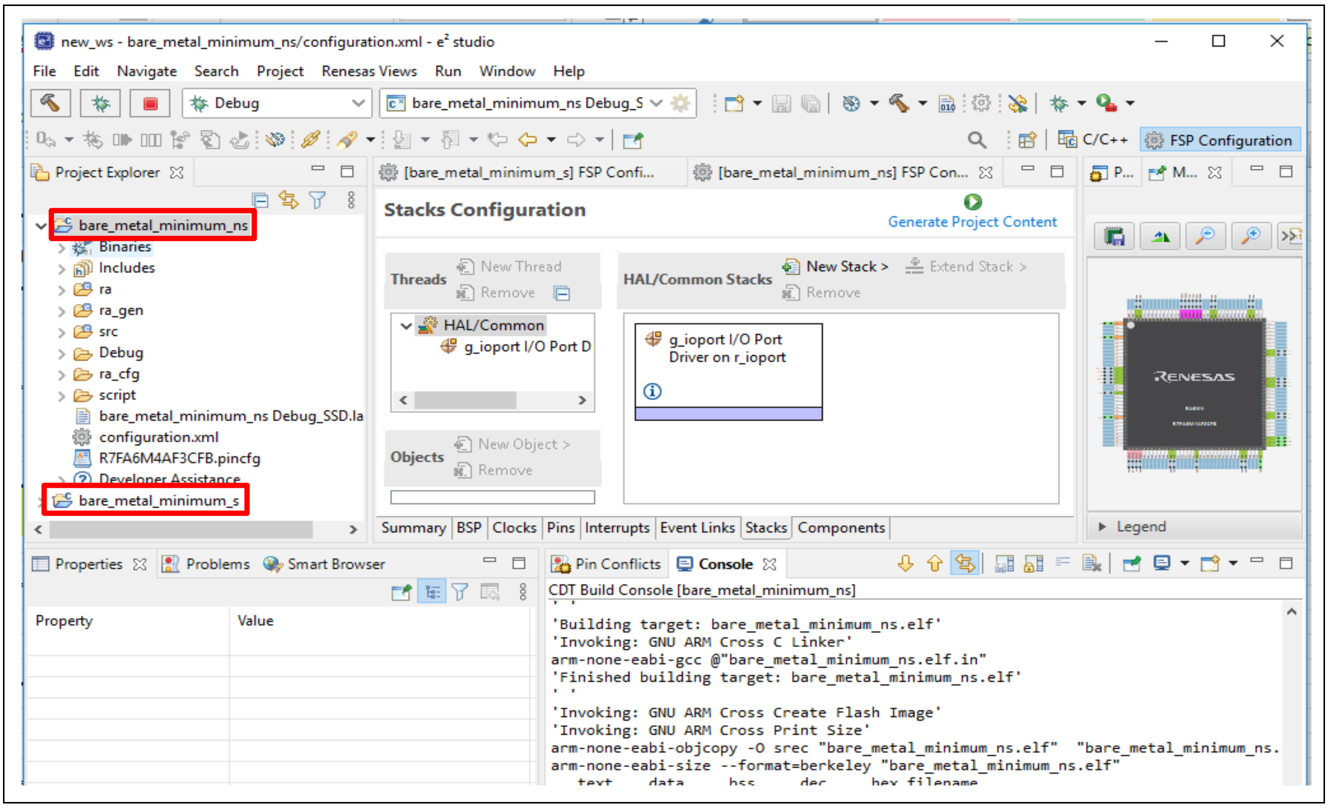

Figure 38. Compile the Non-secure Project (No RTOS, Bare-Metal Minimum)

Step 7: Debug both the Secure and Non-secure projects.

As shown in Figure 39, the debug configuration of the Non-secure project programs, including both the Secure and Non-secure `.elf` files to the MCU by default to allow a unified debug session of both the Secure and Non-secure projects.

Notice that <project_name> <build_configuration>_SSD.launch is generated, as debugging both Secure and Non-secure projects are performed in device lifecycle state SSD.

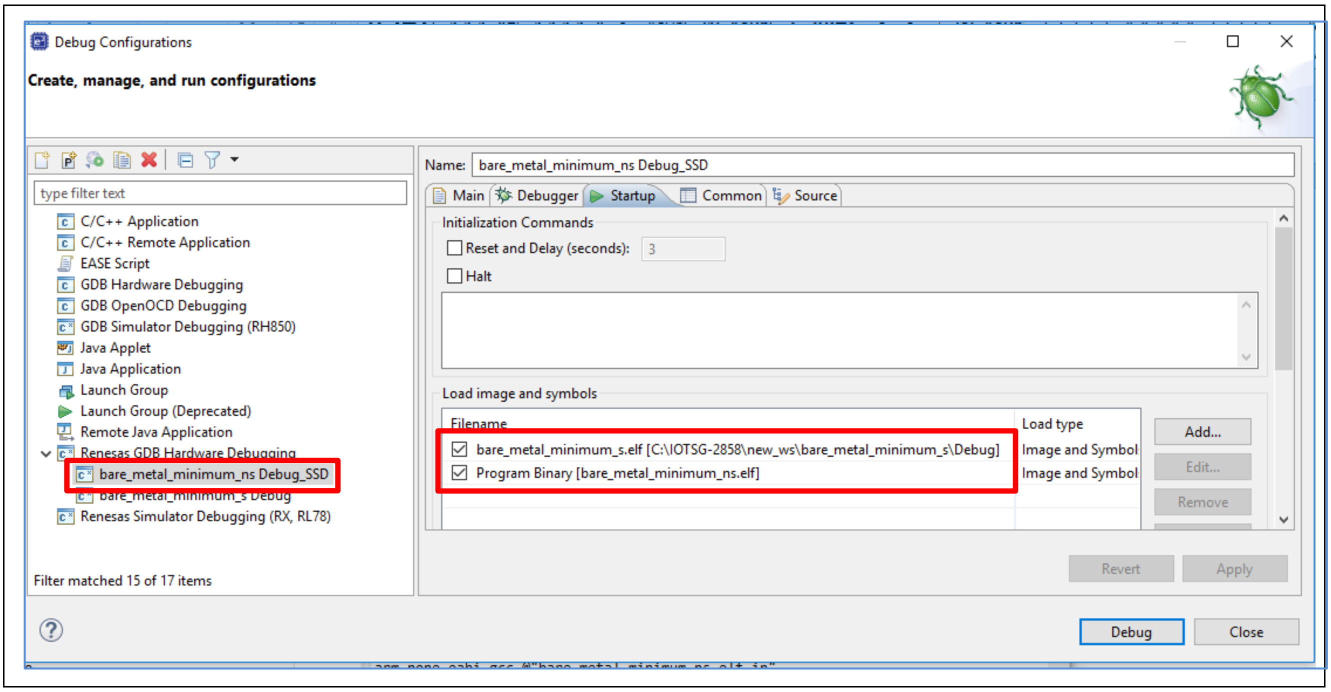

Figure 39. Debug Both the Secure and Non-secure Projects

Note: The Secure project must be built each time it is changed to ensure that the connection to the Non-Secure project is maintained. When the Secure bundle changes, there will be a popup window asking you to take the latest Secure bundle. Click **Yes**, then recompile the Non-secure project so that the updated <project_name>.sbd will be used.

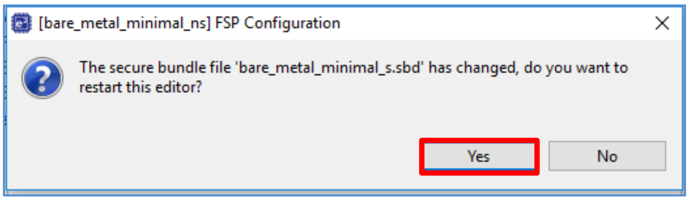

Figure 40. Secure Bundle Update Notification

Tips on Ensuring Synchronization between Secure and Non-secure Projects

To avoid accidental updates from the Secure Project being missed, you can also define the Secure Project as a reference to the Non-secure Project so that compiling the Non-secure Project will automatically trigger a compilation to the Secure Project.

Open the **Properties** page of the Non-secure project, click **Project References,** and choose the corresponding Secure project as the Reference project. Once this is set up, compiling the Non-secure project will always trigger the Secure project to be recompiled.

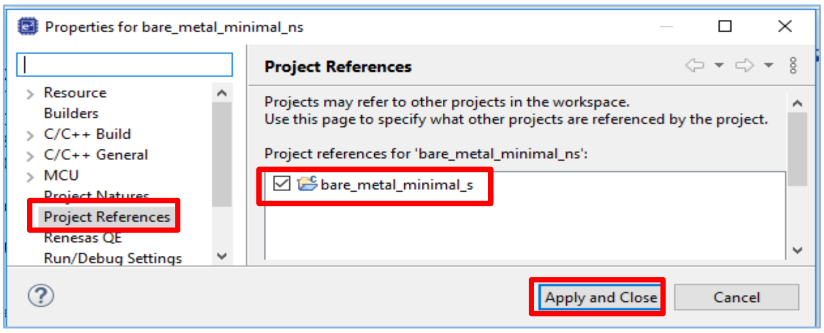

Figure 41. Create Project Reference

4.1.3 Production Flow Overview

This step is for production flow; it is not a step needed during development. Once both Secure and Non-secure project development is finished, you can send the following information to the production line for the MCU to be provisioned prior to selling:

- Secure binary
- Non-secure binary
- IDAU region configuration

Refer to section 6.2 to program the Secure binary and section 6.3 to program the Non-secure binary and transition the MCU state to one of the following device lifecycle states:

- DPL (DePLoyed): The debug interface is disabled temporarily. The serial programming interface is available, but it cannot access the code and data flash.
- LCK_DBG (LoCKed DeBuG): The debug interface is permanently disabled. The serial programming interface is available, but it cannot access the code and data flash.
- LCK_BOOT (LoCKed BOOT interface): The debug interface and the serial programming interface are permanently disabled.

4.2 Split Project Development

Characteristics of the Split Project Development Model include:

- The Secure project and Non-secure project are developed separately by two different teams.
- The Secure project will be developed first by the IP provider. The IP provider creates a Secure bundle.
- The Secure bundle is pre-programmed on the device prior to the Non-secure developer starting their development. Only the Non-secure project and Non-secure partition are visible to the Non-secure developer.

4.2.1 Developing the Secure Bundle and Provisioning the MCU

Developing the Secure project using the Split Project Development Model is very similar to the Combined Project Development Model. However, several key differences are explained in this section.

Step 1: Follow Step 1 to Step 6 from section 4.1.1 to establish the Secure template project and create the applications.

Debugging the Secure project with the Split Project Development Model will not happen with the Non-secure project for the product. As explained in Step 7, section 4.1.1, you can create a dummy Non-secure project for the purpose of Secure project testing, for example, to test the Non-secure callable APIs.

Step 2: Provision the MCU with the Secure project and change the device lifecycle state to NSECSD.

A major difference between Split Project Development and Combined Project Development is that the Secure binary associated with the Secure bundle needs to be provisioned to the MCU prior to the Non-secure project development for the Split Project Development. The Secure bundle contains the Secure project IP in binary format and the NSC API interface from the Secure project. In addition, the MCU device lifecycle state needs to transition from SSD to NSECSD to protect the Secure content.

4.2.2 Limitations and Workarounds for Developing in NSECSD State

There is a limitation with the current version of the tools in that a dummy Non-secure project must be provisioned on the device in addition to the Secure binary prior to changing the MCU device lifecycle from SSD to NSECSD with the Split Project Development Model. This is necessary to allow the Non-secure development to resume in the NSEDSD state.

- In the development stage, follow the Combined Project Development Model to prepare a dummy Non-secure project paired with the intended Secure project. Program the Secure binary and the dummy Non-secure binary first and then change the device lifecycle state to NSECSD.
- In the production stage, send the following items to the production team:
  - Secure binary
  - IDAU region setup information

  RFP will be used to program the Secure binary and set up the IDAU region. See section 6.2 for the operational details.
- Note that the Secure developer also needs to provide the Secure bundle (`<project_name>.sbd`) to the Non-secure developer to allow the Non-secure project to proceed to development.
- See Figure 42 for details on the general flow to support Non-secure project development in the NSECSD state.

4.2.3 Developing the Non-secure Project in NSECSD State

Developing a Non-secure project using the Split Project Development Model has some key differences when compared with the Combined Project Development Model.

For the Split Project Development Model, the Non-secure application developer receives the MCU in the NSECSD state. As mentioned towards the end of the last section, special handling is needed to enable development in the NSECSD state. Figure 42 is a summary of the general flow of development in the NSECSD state.

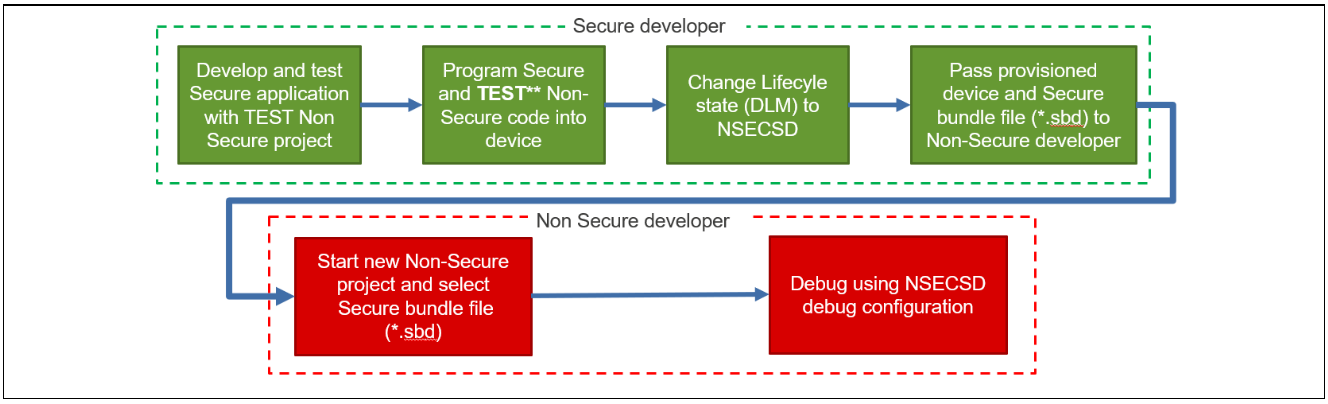

Figure 42. Development Flow for Developing in NSECSD State

Once the Non-secure developers receive the MCU provisioned with the Secure binary, IDAU region, and the Non-secure dummy binary in the NSECSD state, they can use the following steps to proceed to the Non-secure project development:

1. Follow step 1 and step 2 in section 4.1.2 to start Non-secure project development.
   Typically, the Non-secure project will be created in a different workspace from the Secure project as the Secure project source file and `.elf` file will not be available for the Non-secure developer.
2. When the Secure Bundle Selection window opens, choose the secure bundle obtained from the Secure developer.
   This step is a key difference between the Combined Project Development and the Split Project Development process.
   The Secure Bundle contains the following information to allow Non-secure project development:
   — MCU startup code
   — IDAU region setup
   — Details of locked Secure peripherals configuration settings
   — User-defined Non-secure Callable API interface header file (refer to section 3.3)

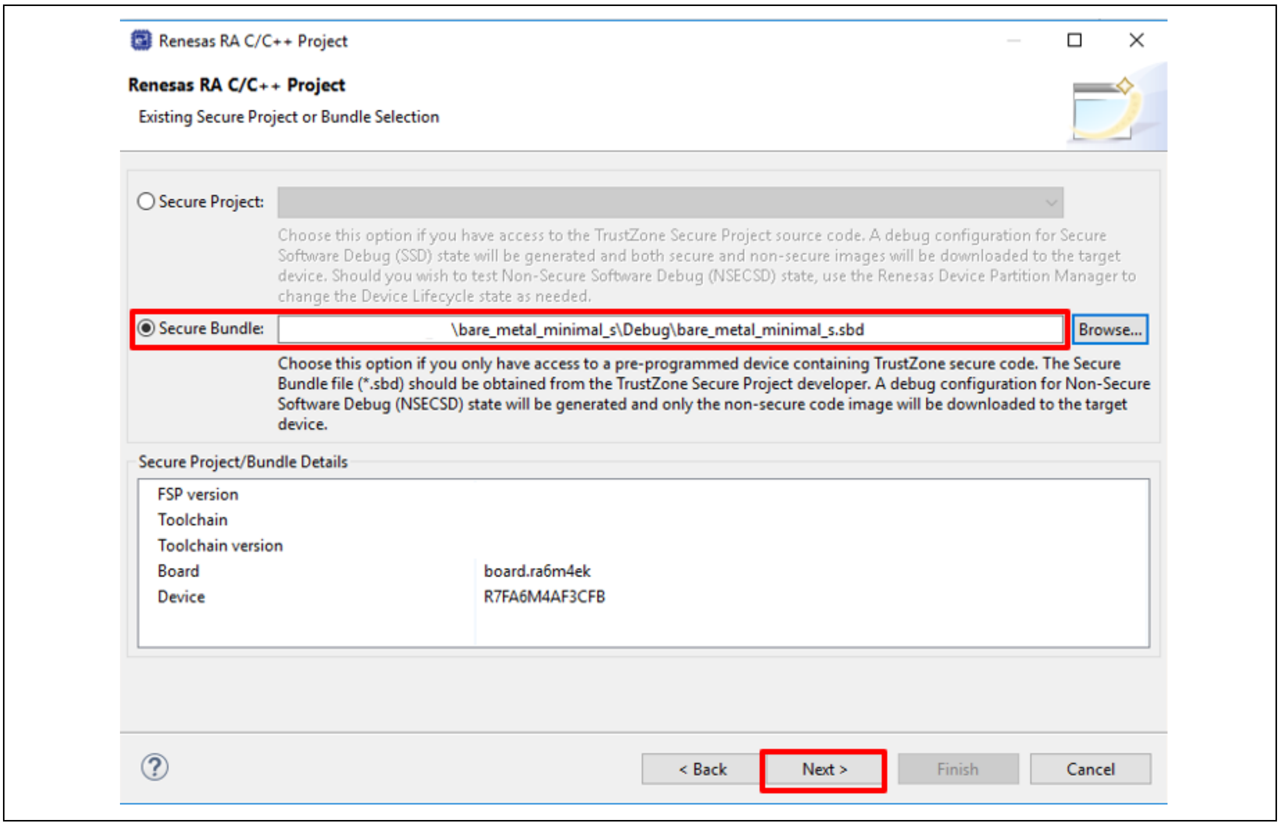

Figure 43. Create Linkage to Secure Bundle

Note: The Secure Bundle is linked in with an absolute path. Verify the Secure Bundle linkage whenever the folder location of the `<project_name>.sbd` changes.

Follow the prompts to define RTOS usage and select the template project. Once the project is generated, double-click `configuration.xml` to open the smart configurator. Click **Generate Project Content** and compile the project.

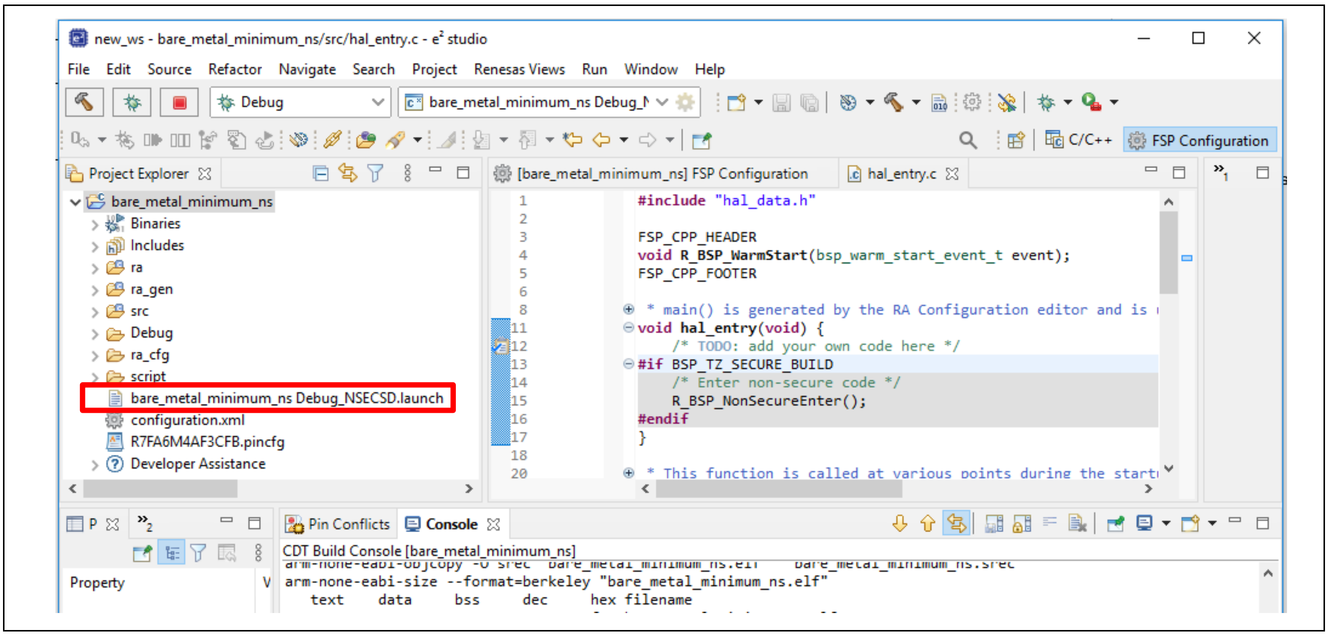

Figure 44. Compilation Result of Non RTOS Bare-Metal Minimum Non-secure Project Template

Notice that `<project_name> <build_configuration>_NSECSD.launch` is generated as the development is carried out in the NSECSD state.

4.2.3.1 Debug the Non-secure Project

Prior to debugging the Non-secure project, ensure that the Secure binary and the dummy Non-secure binary are programmed on the MCU.

During Non-secure project debugging, only the Non-secure `.elf` file will be downloaded. There is only the Non-secure project visible in the workspace for the Non-secure developer as opposed to both Secure and Non-secure projects being visible with the Combined Project Development.

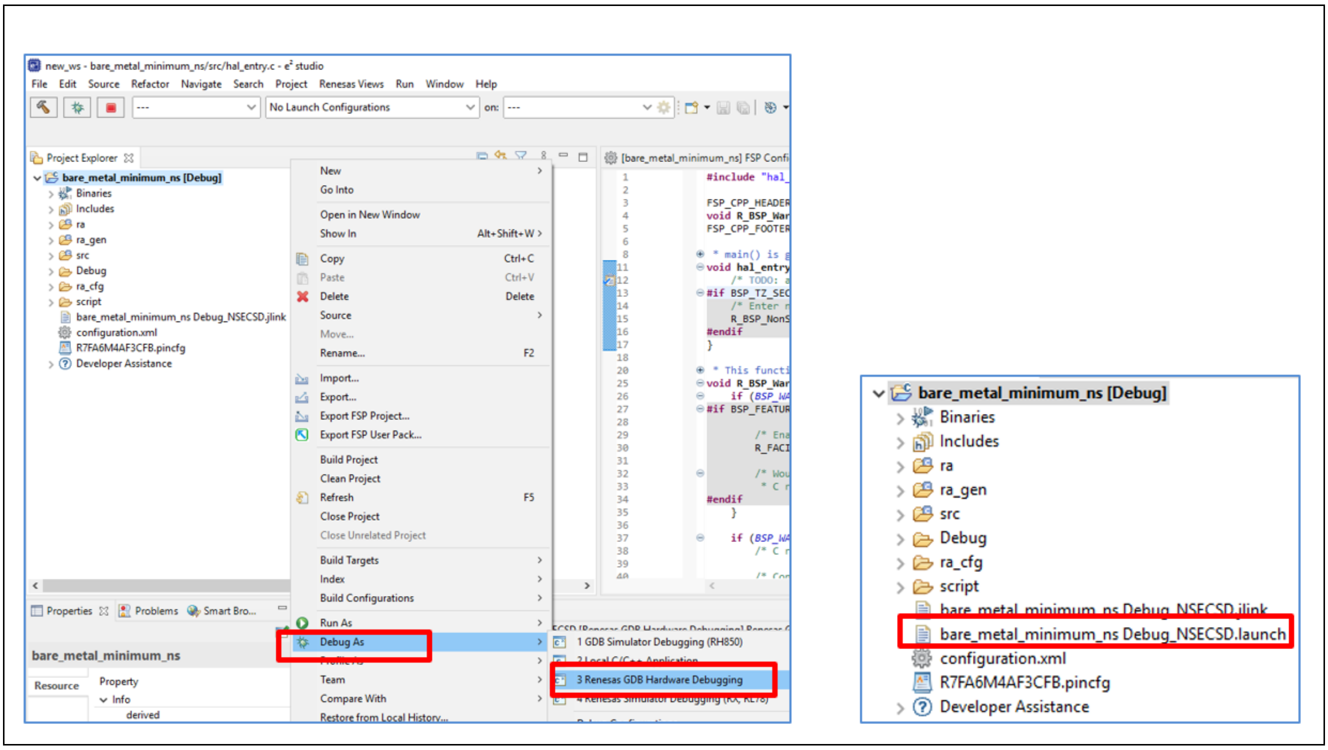

Figure 45. Debug the Non-secure Project

Notes on updating the Secure Bundle:

- If during Non-secure project development, the Secure Bundle needs to be updated, the Non-secure Developer would need to return the MCU to the Secure Development team for MCU update.
- See section ***Non-secure Debug*** in the document *FSP User's Manual* *section: Primer: Arm® TrustZone® Project Development* section *Non-secure Debug* to understand how the tools handle protection of the Secure region when debugging the Non-secure project in the NSECSD Device Lifecycle State.

4.2.3.2 Program the Non-secure Project and Transition to DPL Device Lifecycle State

This step is for the production flow. It is not normally needed during Non-secure project development.

Once the Non-secure project is fully debugged, the Non-secure binary can be sent to the production line to program the MCU and transition to the DPL device lifecycle state. Refer to section 6.3 for operational details.

See the application note, *Installing and Utilizing the Device Lifecycle Management Keys* (*R11AN0469)* for information about other possible deployment mechanisms (LCK_DBG, LCK_BOOT) as well as the state regression methods utilizing the DLM key through an authenticated procedure.

4.2.4 Production Flow Overview

Refer to section 6 to understand the example production flow. For the Split Project Development Model, there can be multiple vendors involved in the production flow:

- Secure image handling vendor: the production team programs the Secure image, sets up the IDAU boundary, injects the desired DLM and User Keys, and transitions the MCU to the NSECSD state. The production team also needs to provide the `.sbd` bundle to the Non-secure application production team.
- Non-secure image handling vendor: the production team programs the Non-secure image and transitions the MCU to a deployment device lifecycle. See section 4.1.3 for the different possible states.

4.3 Flat Project Development

The Flat Project type in the RA Project Generator refers to the development model in which the developer does not need to develop the application with TrustZone® technology awareness:

- One single project handles the entire application.
- Development flow is identical to the Non-TrustZone technology part.
- The MCU operates in the SSD device lifecycle state.
- All peripherals that support Secure and Non-secure attributes will operate in Secure mode.
- Peripherals as identified as Non-secure only in Table 3 will operate in Non-secure mode.

4.3.1 Operational Flow

1. Follow Step 1 and Step 2 from section 4.2.1 to start creating the Flat Project template project.
2. Select **Flat Project** as the project type from the Project Generator.
3. Choose the **Build Artifact Selection** and **RTOS Selection** (same interface as in Figure 33).
4. The rest of the development is the same as the development for Non-TrustZone technology-enabled MCUs and is out of the scope of this application project.
5. Debug Flat Project.
   Debugging the Flat Project follows the Non-TrustZone RA MCU Debugging model. The launch file is named: `<program_name> <build_configuration>_Flat.launch.`

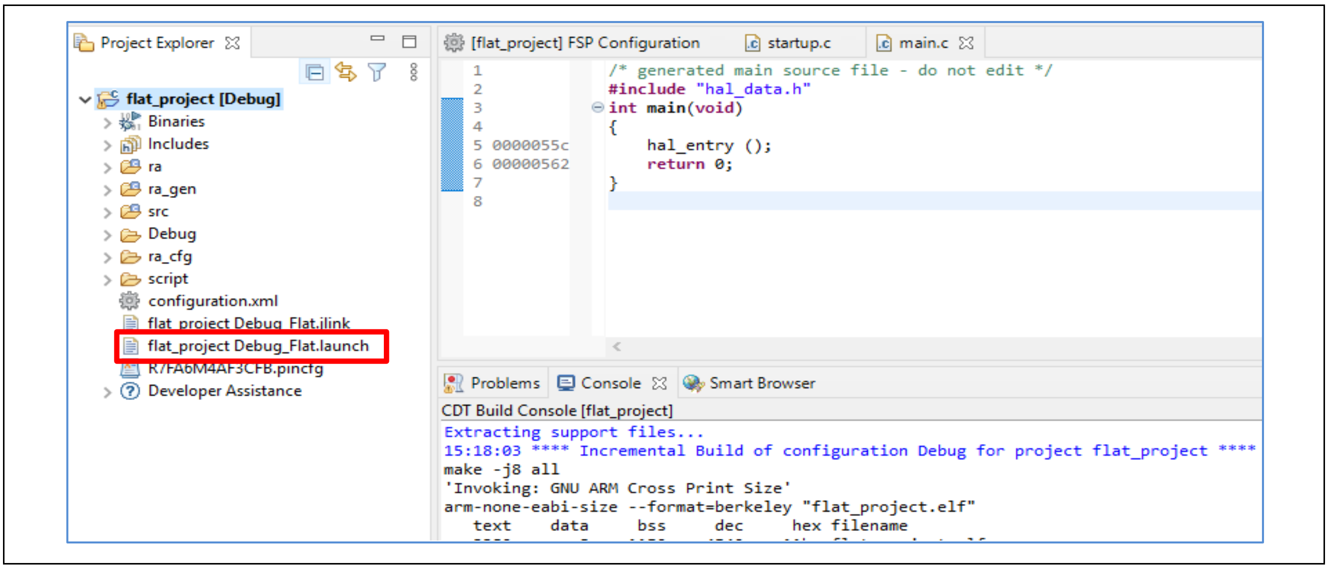

Figure 46. Debug the Flat Project

4.3.2 Ethernet Application

In the case of using Ethernet with a Flat Project, the IDE will calculate the size of the SRAM buffer on an 8KB boundary based on the application to cover all the Ethernet buffer usage. The IDE will then allocate this region to the Non-secure SRAM region in the .rdp file. This entire process is automatically handled by the IDE and FSP, the operation is transparent to users.

4.3.3 Production Flow Overview

Production of the Flat Project development model will bring in TrustZone technology awareness. The Flat Project development is carried out in the MCU lifecycle state SSD. For production deployment, you have the same options as the TrustZone technology aware development model: Split Project Development Model or Combined Project Development Model.

- Option one is to transition the MCU lifecycle state from SSD to NSECSD, then transition to DPL.
  - If desired, the MCU lifecycle state can then be transitioned further to LCK_DBG or LCK_BOOT.
- Option two is to transition the MCU state from SSD directly to LCK_DBG or LCK_BOOT.

Refer to section 4.1.3 for the different possible states.

5. Example Project for IP Protection

As discussed in section 1.4.1, IP Protection is a strong use case for TrustZone® technology. The project accompanying this document utilizes the Split Project Development Model to provide an IP protection example of a TrustZone use case with EK-RA6M4 using the e² studio IDE. The Combined Project Development Model is used for the IAR EWARM and Keil MDK projects.

5.1 Overview

RA6M4 MCUs can be configured to use an ADC peripheral to monitor the on-chip temperature sensor. This application project defines an algorithm to control the LED blinking pattern based on the temperature read from the ADC. The following hardware components are configured as Secure by the Secure project:

- ADC channel for on-chip temperature sensor reading.
- GPIO 400, 404, and 415.
- The IDAU sets up the secure flash and SRAM.

The following software components are configured as Secure by the Secure project:

- The FSP ADC HAL driver.
- The FSP GPIO HAL driver for the corresponding LED driving pins.
- The application code that starts, scans, and stops the ADC.
- The application code that controls the LED blinking pattern based on the temperature reading.
- The API that starts the monitoring and reacting algorithm.
  - This API is defined as a Non-secure Callable API, and its veneer is exposed to the Non-secure partition.
- The API that stops the monitoring and reacting algorithm.
  - This API is defined as a Non-secure Callable API, and its veneer is exposed to the Non-secure partition.

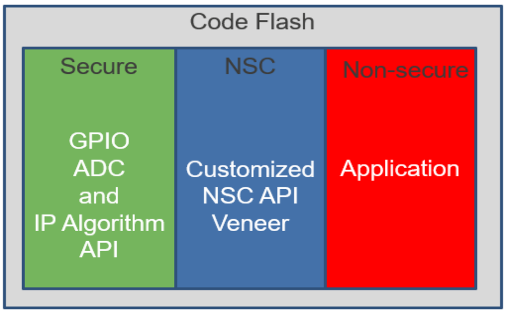

Figure 47. Sensor Algorithm IP Protection

5.2 System Architecture

5.2.1 Software Components

Figure 48 shows the Secure, Non-secure, and Non-secure Callable hardware and software partition scheme in this example project.

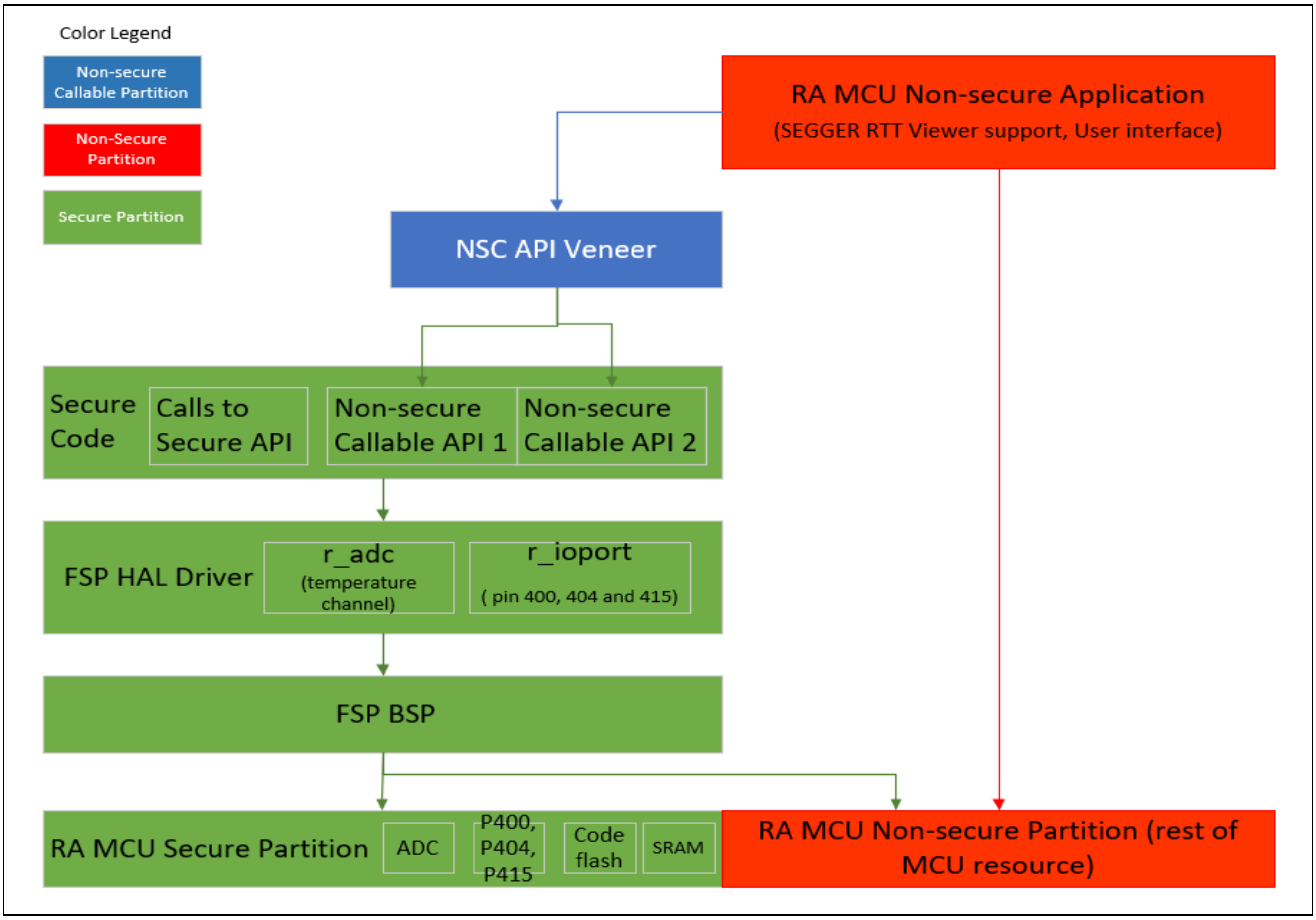

Figure 48. Software Architecture Block Diagram

5.2.2 Operational Flow

Figure 49 shows the system-level operational flow of the example project.

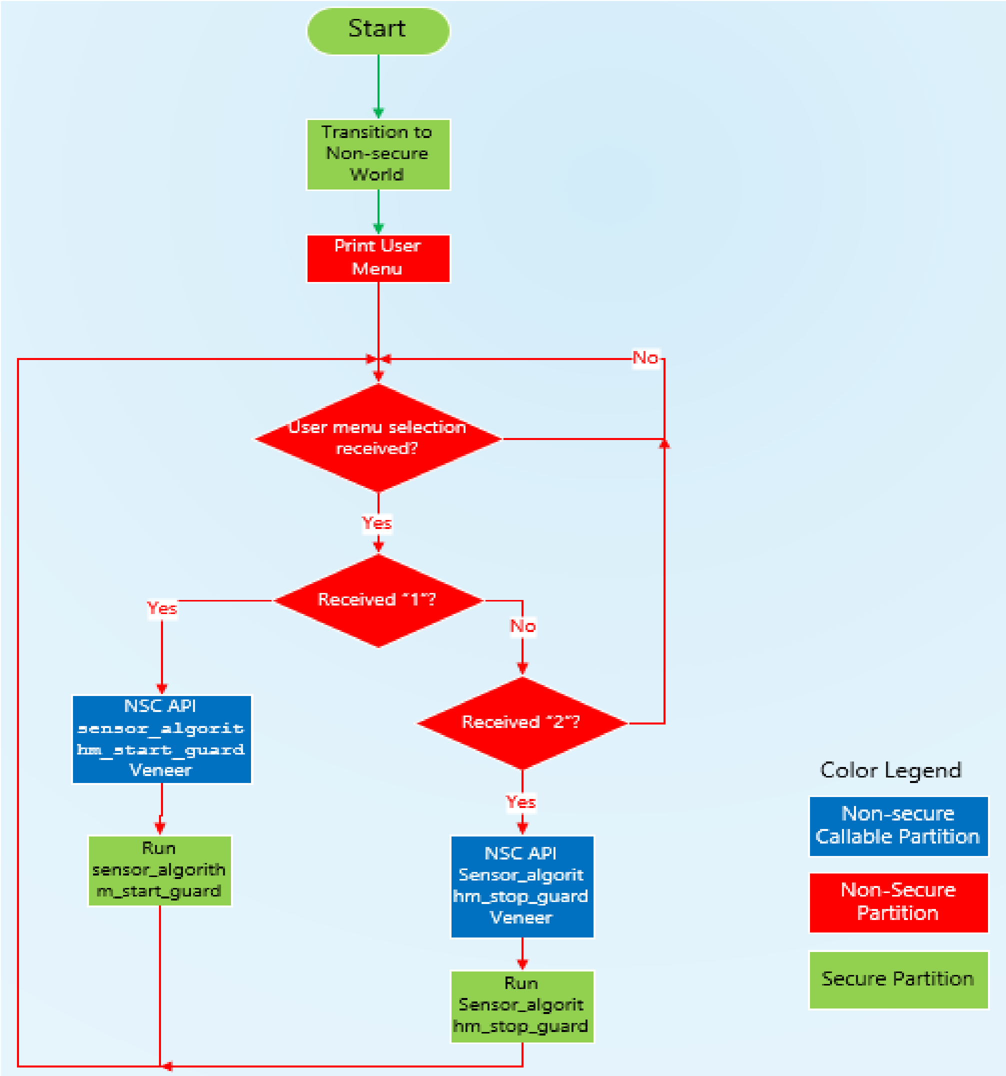

Figure 49. Operational Flow

5.2.3 Simulated User's IP Algorithm

The simulated user's IP algorithm is described in Figure 50.

Note: In Figure 50, TSN means on-chip Temperature Sensor.

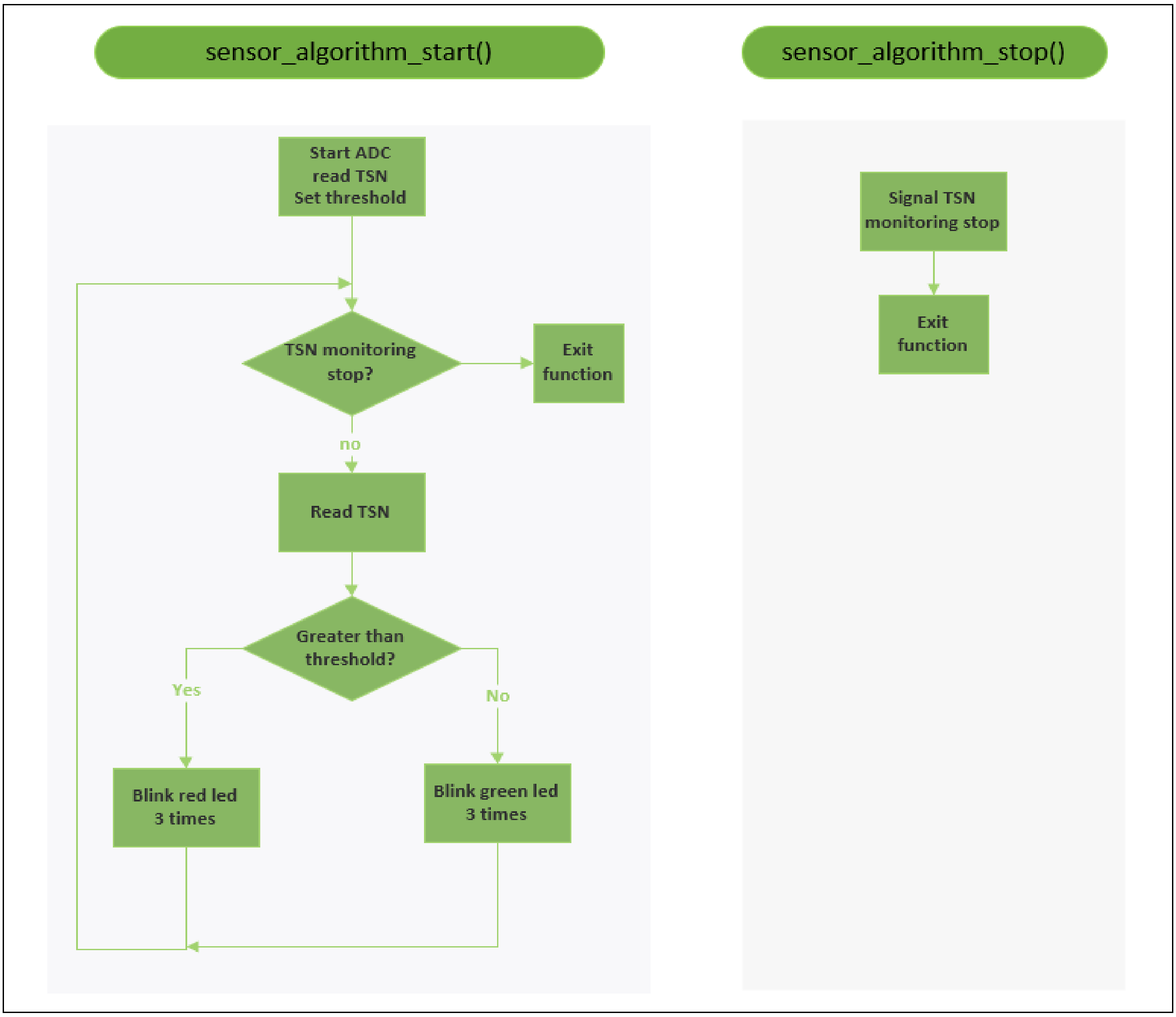

Figure 50. Simulated Sensor IP Algorithms (Running in Secure Partition)

5.2.4 User-Defined Non-secure Callable APIs

The Non-secure callable functions exposed to the Non-secure partition are defined in `sensor_algorithm_nsc.h` from the Secure project.

```
 * File Name    : sensor_algorithm_nsc.h
 * DISCLAIMER
#ifndef SENSOR_ALGORITHM_NSC_H_
#define SENSOR_ALGORITHM_NSC_H_

#include <hal_data.h>

BSP_CMSE_NONSECURE_ENTRY void sensor_algorithm_start_guard(void);
BSP_CMSE_NONSECURE_ENTRY void sensor_algorithm_stop_guard(void);

#endif /* SENSOR_ALGORITHM_NSC_H_ */
```

Figure 51. User-Defined NSC APIs

To share the user-defined NSC calls, this header file is linked to e² studio by a Build Variable.

The path to this header file is added using the Build Variable `UserNscApiFiles`, as shown in Figure 52.

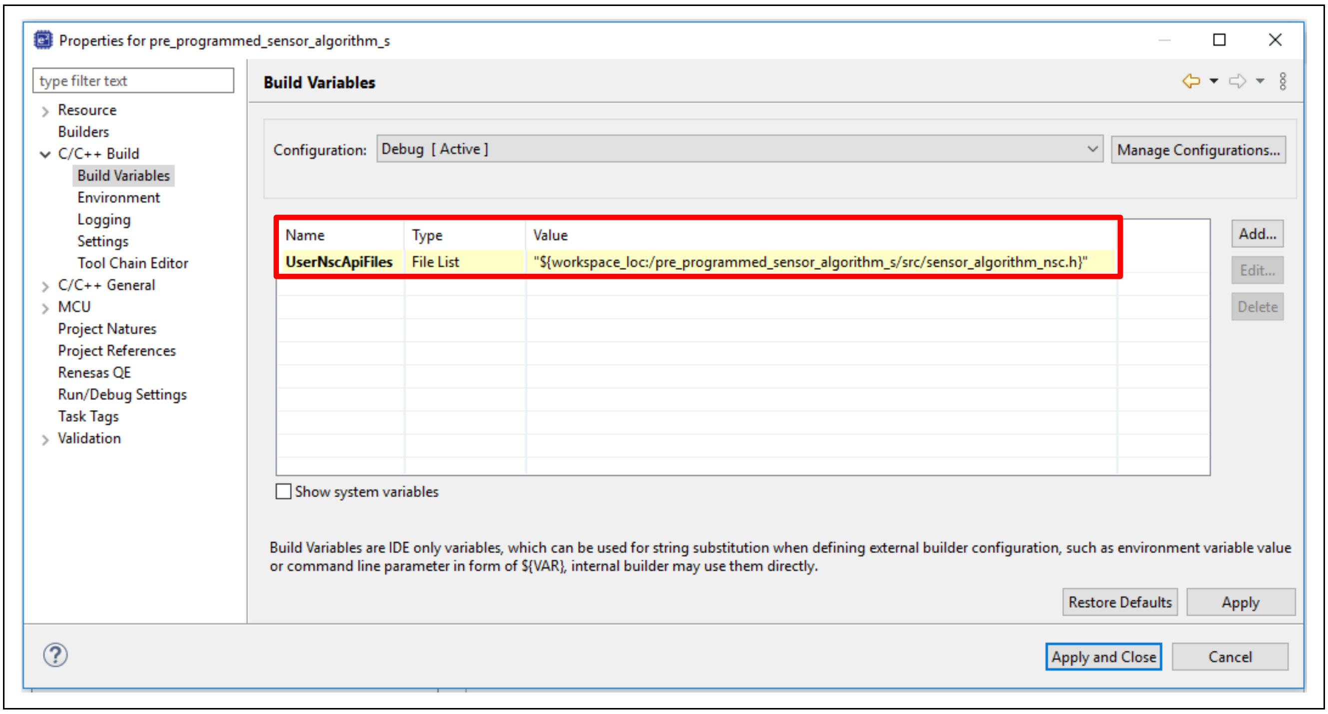

Figure 52. User Build Variable to Link User NSC Header File (Secure Project Setting) in e² studio

The Build Variable approach does not exist when using IAR EWARM and Keil MDK; you need to manually share this header file with the Non-secure project. This is demonstrated in the IAR EWARM and Keil MDK example projects that were included.

5.3 Setting up Hardware

- Jumper setting – default EK-RA6M4 setting.
  - See *EK-RA6M4 User's Manual*.
- Connect J10 using USB macro to B cable from EK-RA6M4 to the development PC to provide power and debugging capability using the onboard debugger.

Initialize the MCU

This step is optional but recommended. Prior to downloading the example application, it is recommended that the device be initialized to the SSD state. Unlocked flash content will be erased during this process. This step can be achieved using the Renesas Device Partition Manager or RFP. This is particularly helpful if the device was previously used in the NSECSD state or has a certain flash block locked up temporarily.

For instructions on how to use RFP to perform this function, see section 6.1.

Use Renesas Device Partition Manager and J-Link Debugger to initialize the MCU.

Establish the following connection prior to using the Renesas Device Partition Manager and the Onboard J-Link debugger to Initialize the **device back to factory default**. Note that **Initialize device back to factory default** performs the same functionality as **Initialize Device** when using RFP:

- EK-RA6M4 jumper setting: J6 closed, J9 open. Other jumpers keep out-of-box settings.
- USB cable connected between J10 and development PC.

Note: You must power cycle the board prior to working with the Renesas Device Partition Manager after a debug session if using J-Link as the connection interface.

Open Renesas Device Partition Manager

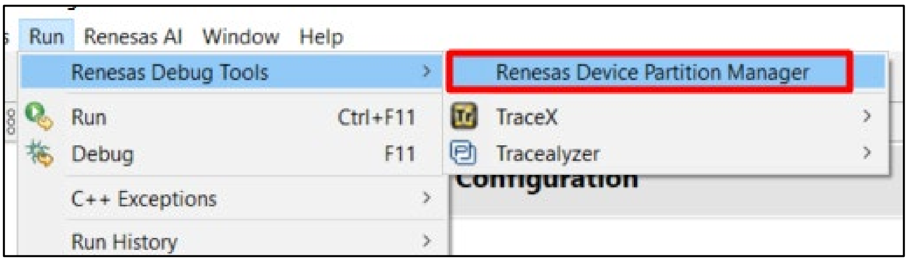

Figure 53. Open the Renesas Device Partition Manager

Next, check **Initialize device back to factory default**, choose the connection method, then click **Run**.

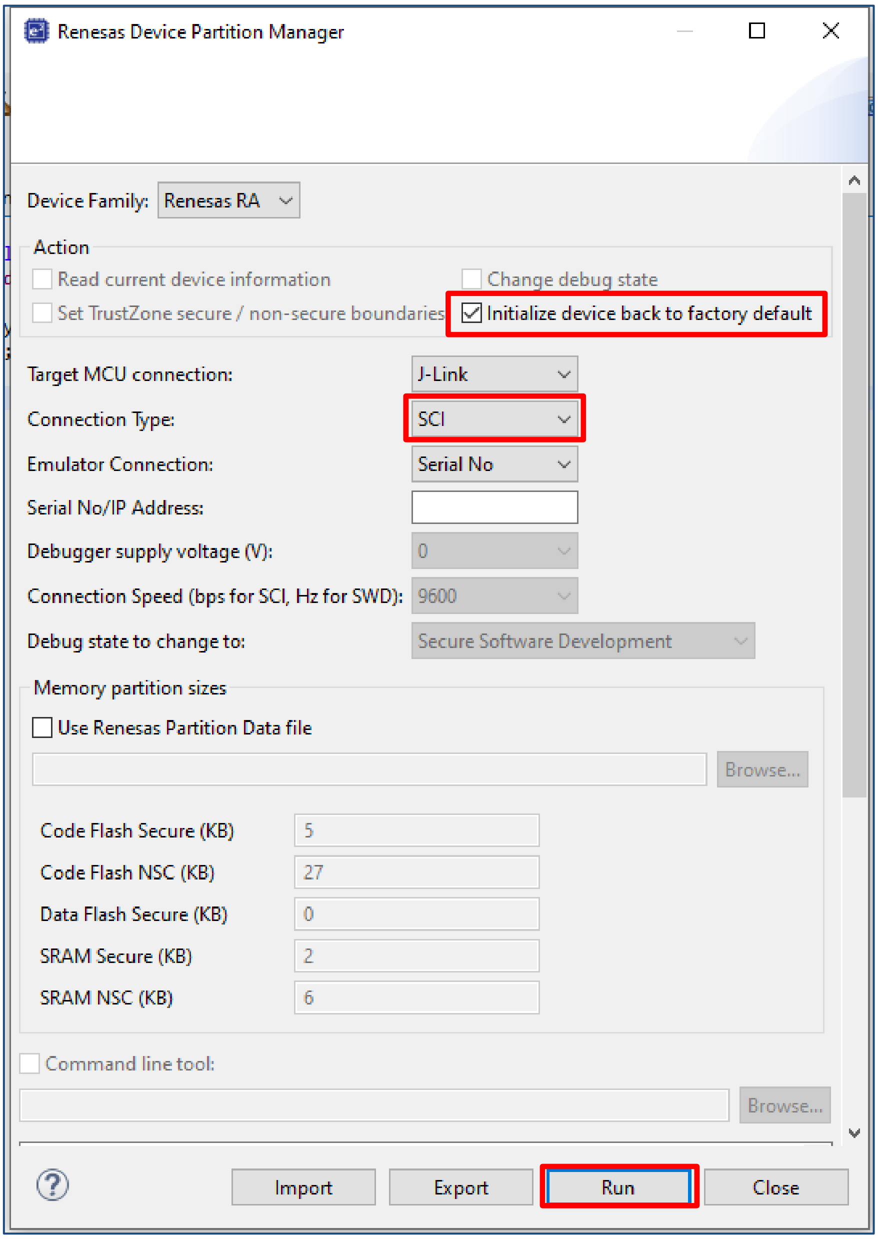

Figure 54. Initialize RA6M4 using Renesas Device Partition Manager

After the MCU is initialized, proceed to the project importing and verification based on the IDE selected.

5.4 Example Application with e² studio IDE using Split Project Development Model

The e² studio project utilizes the Split Project Development Model to establish an application for IP protection. The assumption is that the Secure and Non-secure applications are developed by separate teams.

5.4.1 Import, Build, and Program the Secure Binary and Dummy Non-secure Binary

Use the following steps to provision the MCU with the Secure binary and a dummy Non-secure binary.

5.4.1.1 Import the Secure Project and Dummy Non-secure Project

Unzip `e2studio.zip`, which is included in this application project, to reveal the folders shown in Figure 55.

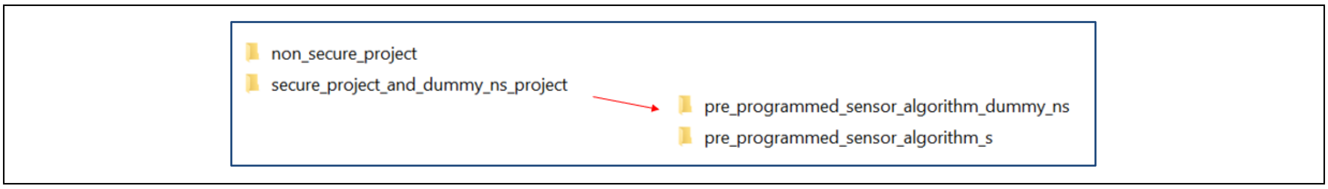

Figure 55. e² studio Software Project Content

Next, follow the *FSP User's Manual* section, *Importing an Existing Project into e² studio* to import the Secure project and the dummy Non-secure project into the same workspace.

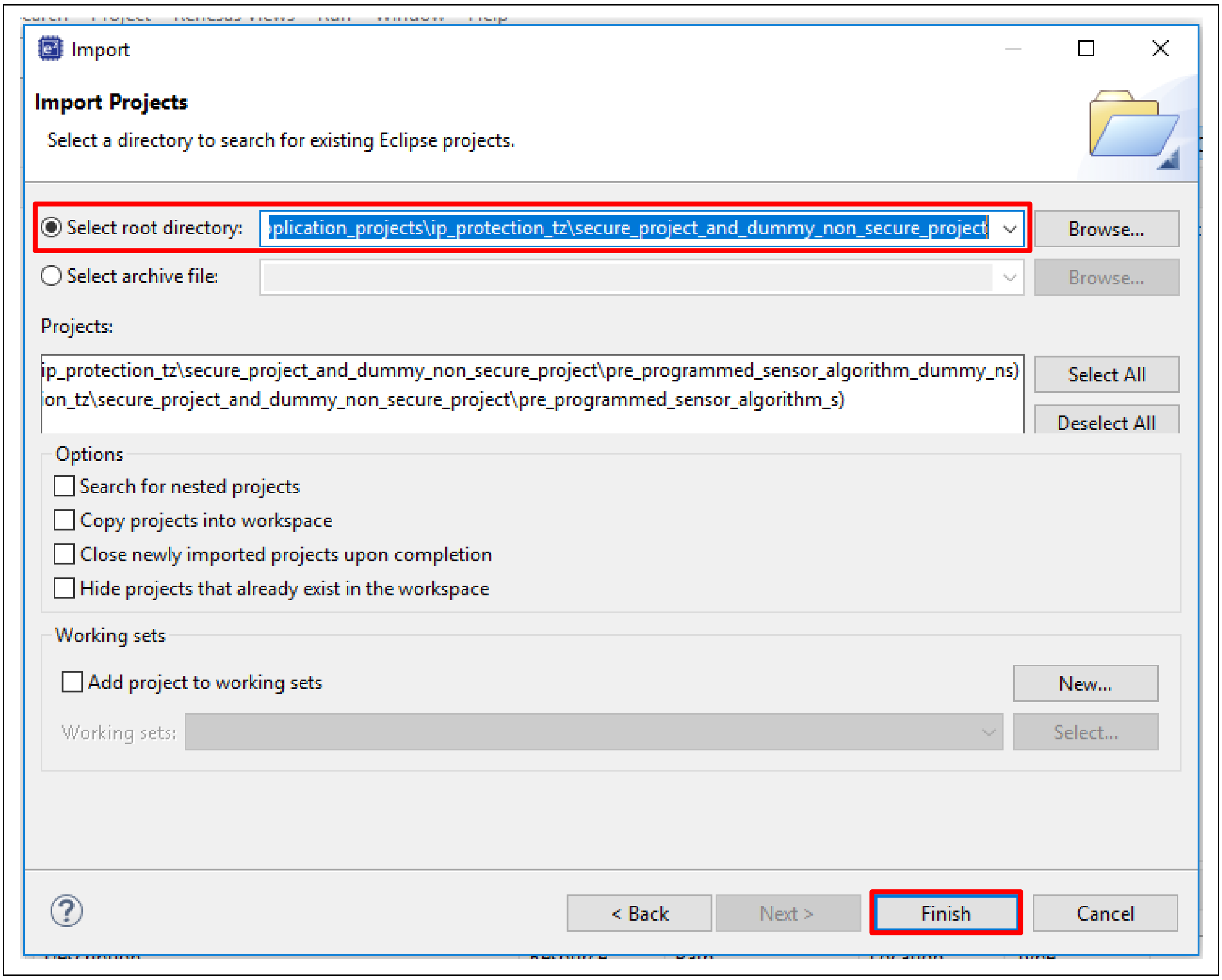

Figure 56. Import the Secure Project and Dummy Non-secure Project

Click **Finish**.

5.4.1.2 Compile the Secure Binary and Dummy Non-secure Binary using e² studio

- Compile the Secure project first. Double-click to open the `configuration.xml` in the Secure project. Click **Generation Project Content**. Compile the Secure project. Ensure `pre_programmed_sensor_algorithm_s.srec` and `pre_programmed_sensor_algorithm_s.sbd` are generated.
- Next, compile the Dummy Non-secure project. Double-click to open the `configuration.xml` in the Dummy Non-secure project. Click **Generate Project Content**. Compile the Non-secure project. Ensure `pre_programmed_sensor_algorithm_dummy_ns.srec` is generated.

5.4.1.3 Download the Secure Binary and Dummy Non-secure Binary using e² studio

Prior to downloading and running the example project, the user should first follow section 5.3 to set up the MCU.

Right-click on the `pre_programmed_sensor_algorithm_dummy_ns` project and select **Debug As > Renesas GDB Hardware Debug**. Click **Resume** twice to run the Secure and dummy Non-secure project. Click **Pause** and confirm the execution pauses at the `while(true)` loop in the `hal_entry()` function in `hal_entry.c` of the dummy Non-secure project.

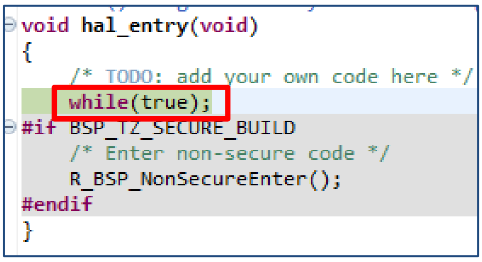

Figure 57. Program and Run the Secure and Dummy Non-secure Projects

Stop the debug session.

5.4.1.4 Transition MCU Device Lifecycle State to NSECSD

After both the Secure binary and dummy Non-secure binary are downloaded to the MCU, you can use the **Renesas Device Partition Manager** (RDPM) to transition the MCU from the SSD device lifecycle state to the NSECSD device lifecycle state.

First, power cycle the board. Next, launch RDPM and configure it to transit to NSECSD.

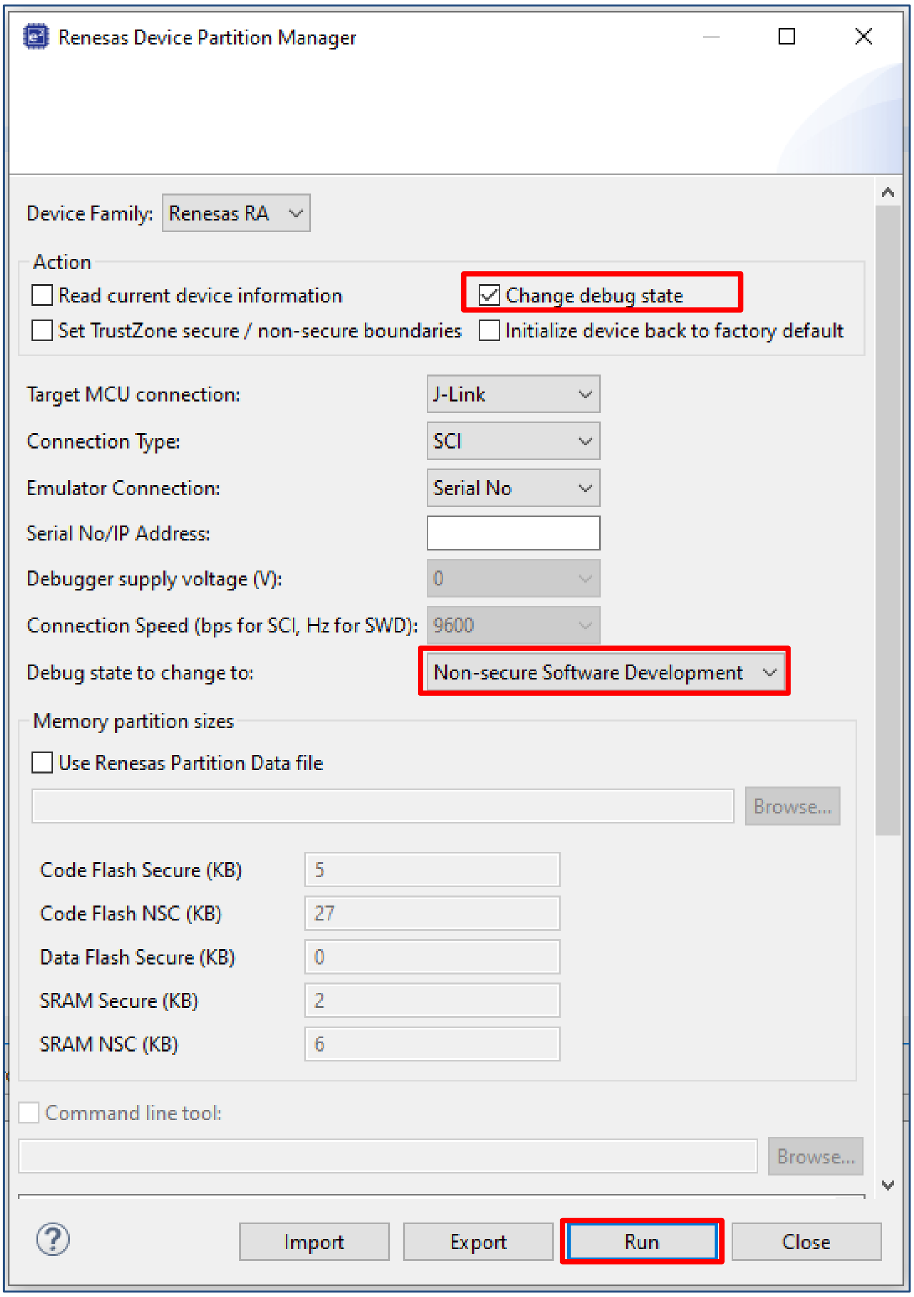

Figure 58. The transition from SSD to NSECSD using Renesas Device Partition Manager

Click **Run** to ensure the transition is successful.

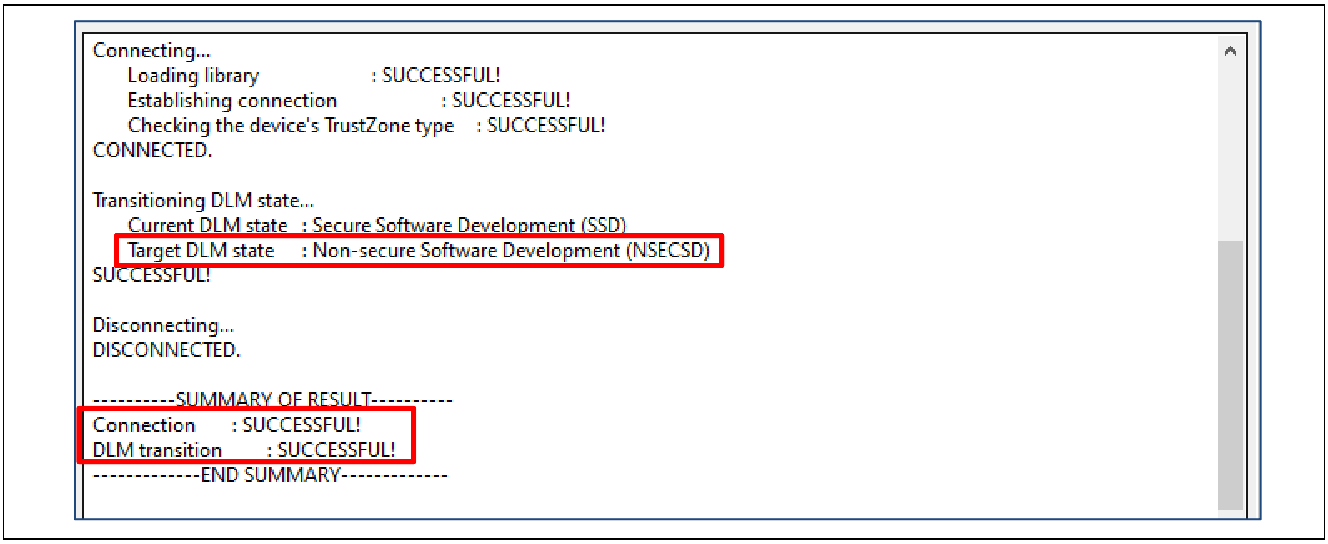

Figure 59. Result: Transition from SSD to NSECSD

Refer to section 6.1 and section 6.2 for the operational steps of downloading the Secure binary and setting up the IDAU region using RFP during the production stage.

5.4.2 Import, Build, and Program the Non-secure Project

Once the DLM transitions to NSECSD, you can download the real Non-secure project.

5.4.2.1 Import the Non-secure Project

Follow the *FSP User's Manual* section, Importing an Existing Project into e² studio to import the Non-secure project into the workspace. You can import it into the workspace where the Secure project is imported to verify the example project.

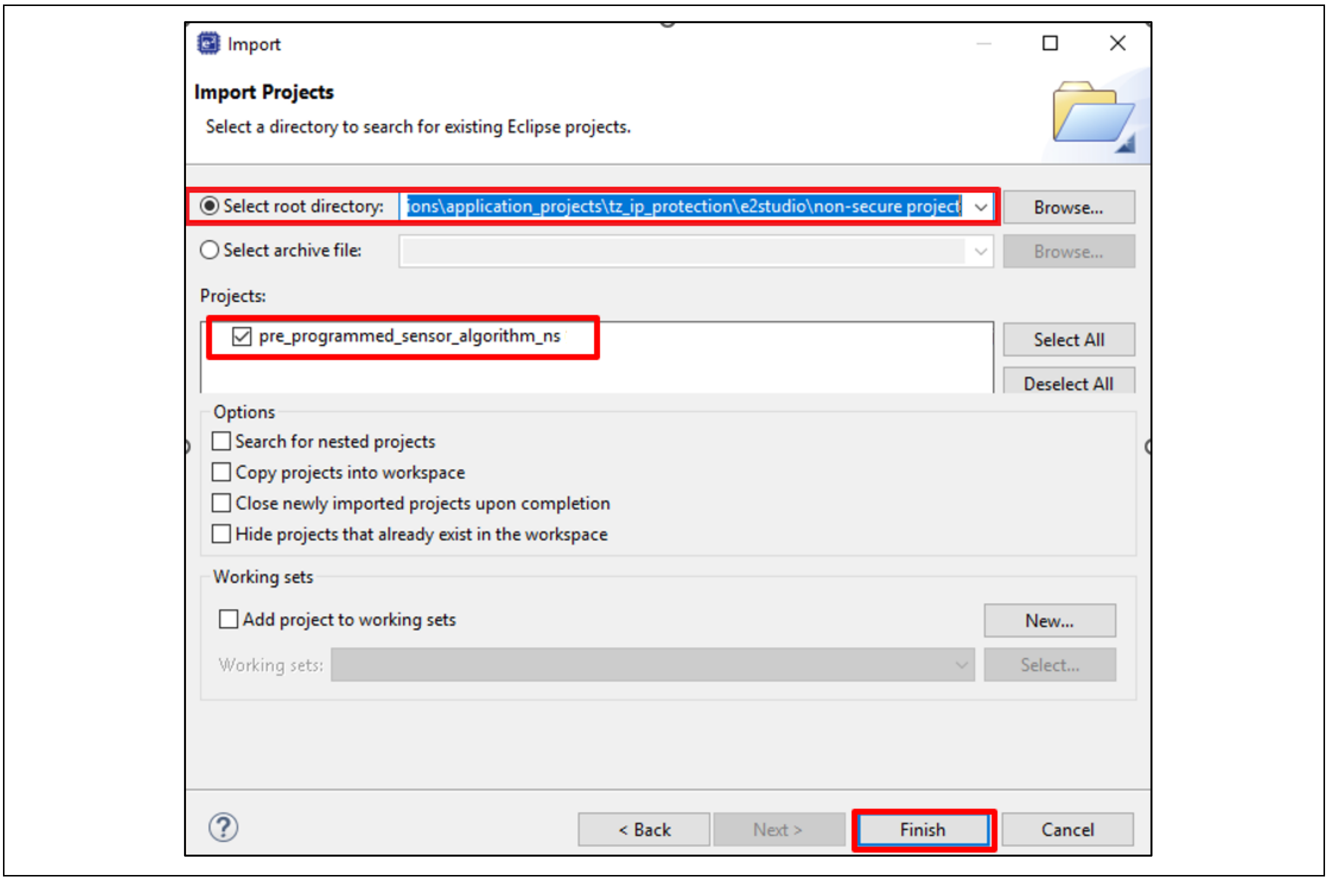

Figure 60. Import the Non-secure Project

Note: You must update the Build Variable **SmartBundle** by selecting the `pre_programmed_sensor_algorithm_s.sbd` based on your local file structure prior to moving forward to the other steps. This is a limitation of the current tools.

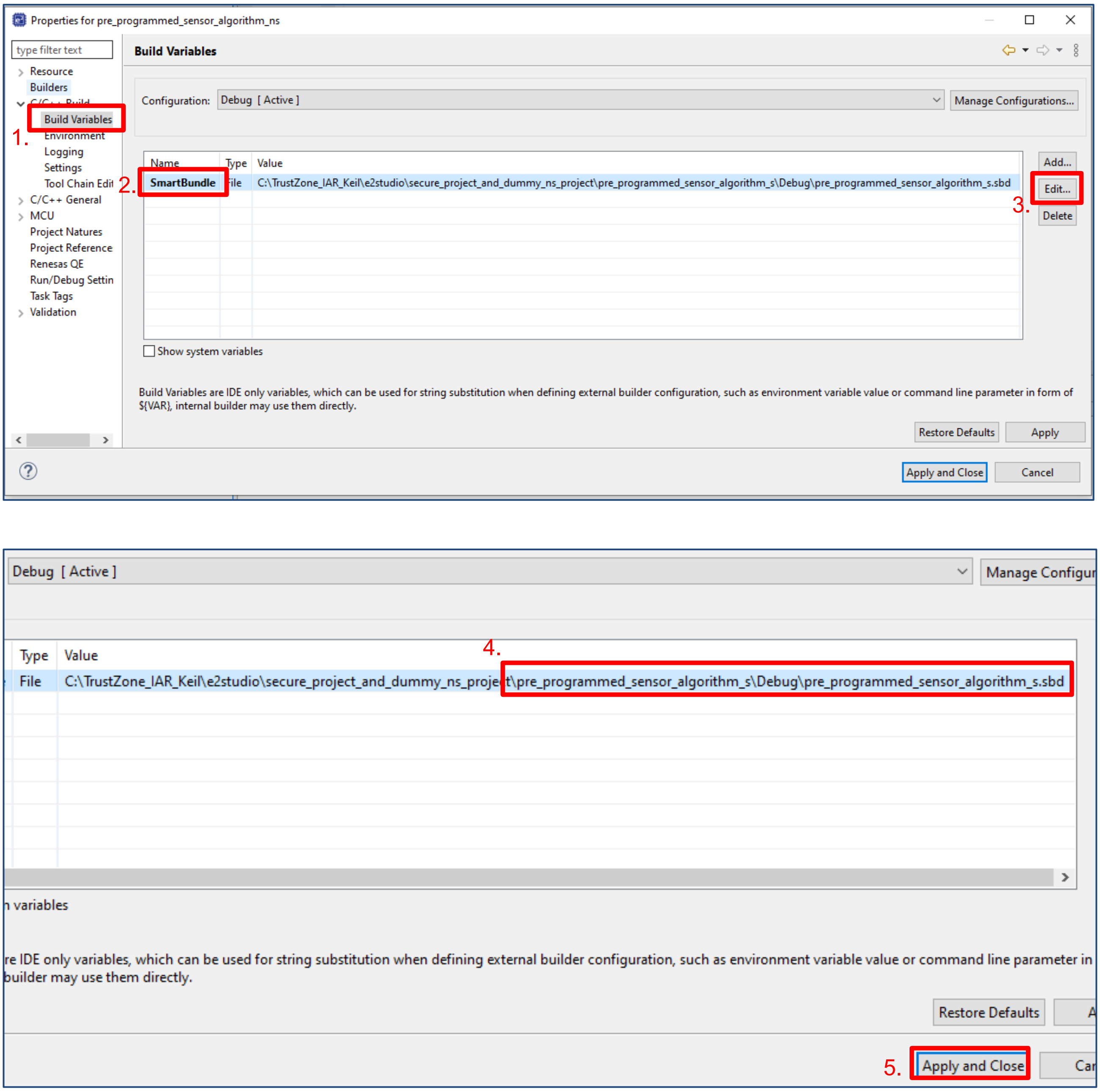

Figure 61. Referencing the Secure Bundle

5.4.2.2 Compile and Download the Non-secure Project

- Double-click to open the `configuration.xml` in the Non-secure project. Click **Generation Project Content**. Compile the Non-secure project.
- Download and run the Non-secure project.

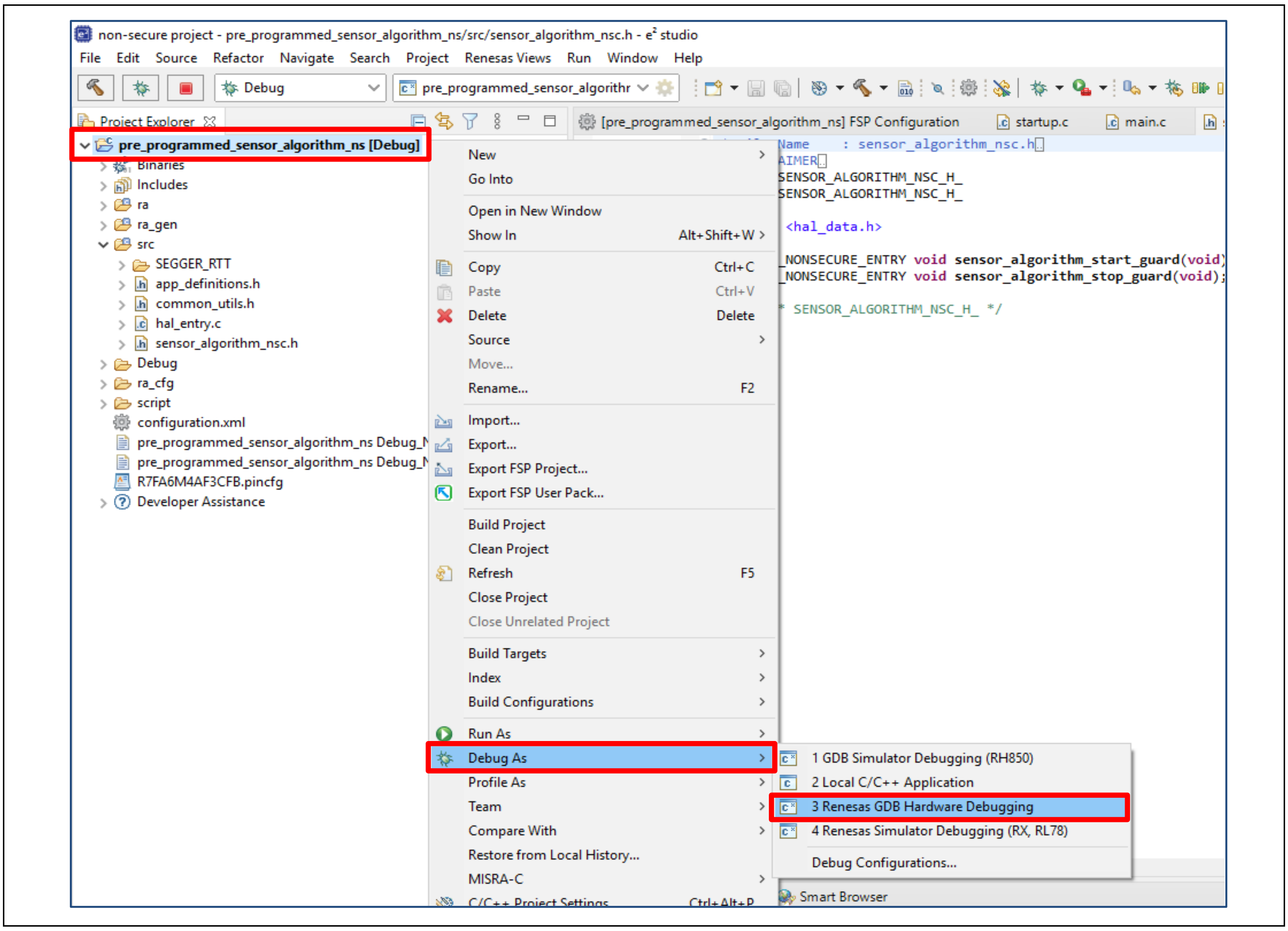

Figure 62. Download and Run the Non-secure Project

Note: For the Split Project Development model, the debug session of the Non-secure project created by referencing the Secure Bundle rather than the Secure Project (as with the case for the dummy Non-secure project) only downloads the `.elf` file of the Non-secure project.

5.4.3 Verify the Example Application

The projects are now loaded, and the debugger should be paused in the `Reset_Handler()` at the `SystemInit()` call for the Non-secure project.

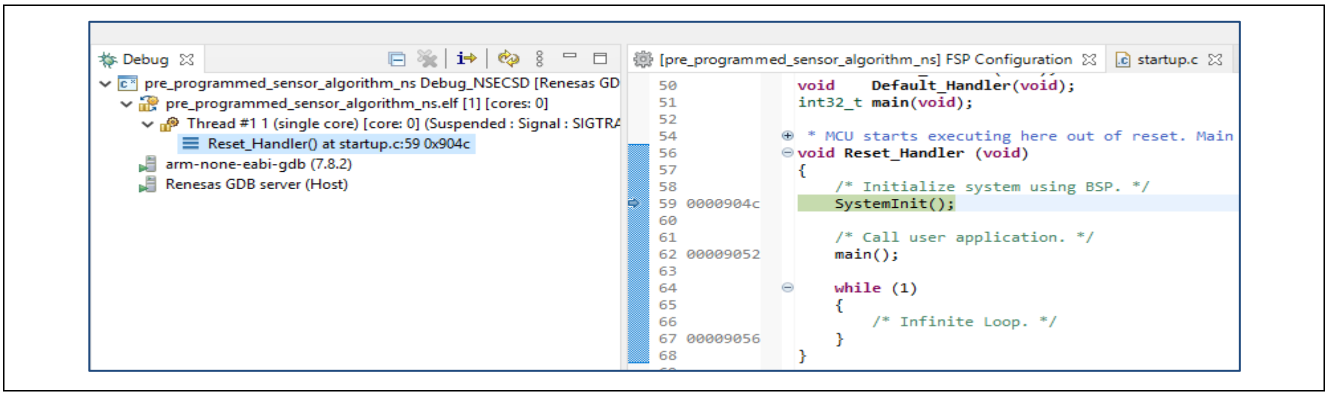

Figure 63. Running the Non-secure Project

Open the J-Link RTT Viewer 7.98b or later. First, click "**…**" and select **R7FA6M4AF** from **Renesas** as the Target Device. Next, set the connection to J-Link to **the Existing Session and the RTT Control Block to the Search Range**. Set the search range to 0x20000000 0x8000 and then click **OK** to start RTT Viewer.

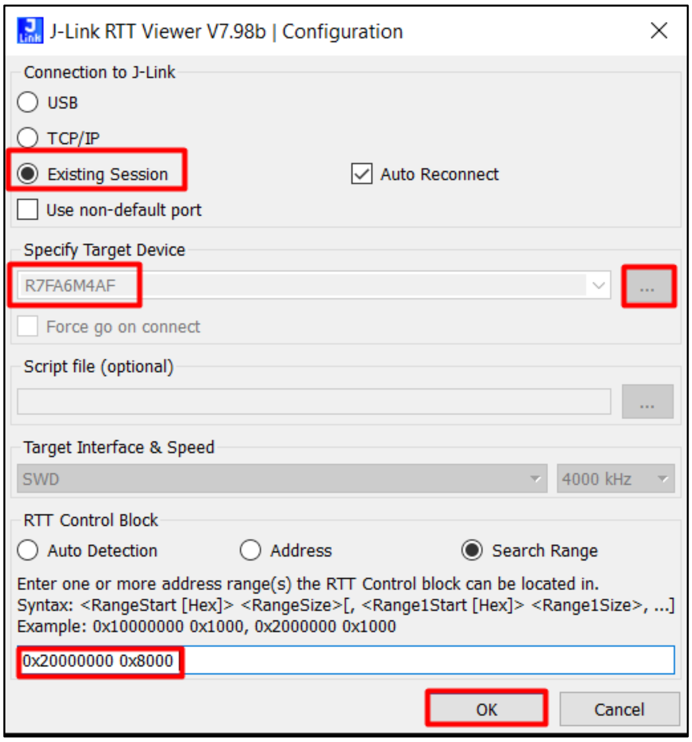

Figure 64. Start the RTT Viewer

Next, click  twice to run the project.

The user menu is then output, and the system waits for user input.

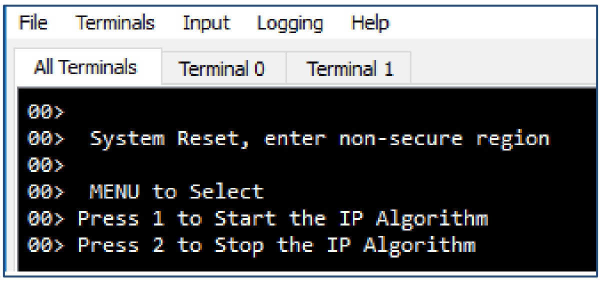

Figure 65. User Menu

Input **1** to start the IP algorithm. You will see the green LED start to blink after a couple of seconds.

You can warm up the MCU (for example, touch the MCU using grouped fingers) and see that the green LED stops blinking and the red LED starts to blink after about 5-10 seconds.

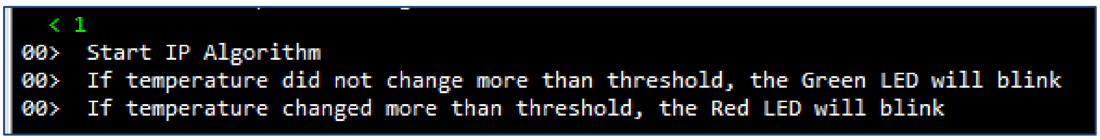

Figure 66. User Input ‘1’

Input **2** to stop the IP algorithm. The green or red LED stops blinking. The blue LED blinks twice and stops blinking.

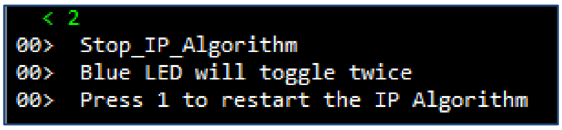

Figure 67. User Input ‘2’

You can repeatedly input **1** to restart the IP algorithm and input **2** to stop.

Notes on Running the Application in Standalone Mode

After the MCU is programmed with the application code, you can run the application in standalone mode (with no debugging session). In this case, choose **USB** as the **Connection to J-Link**.

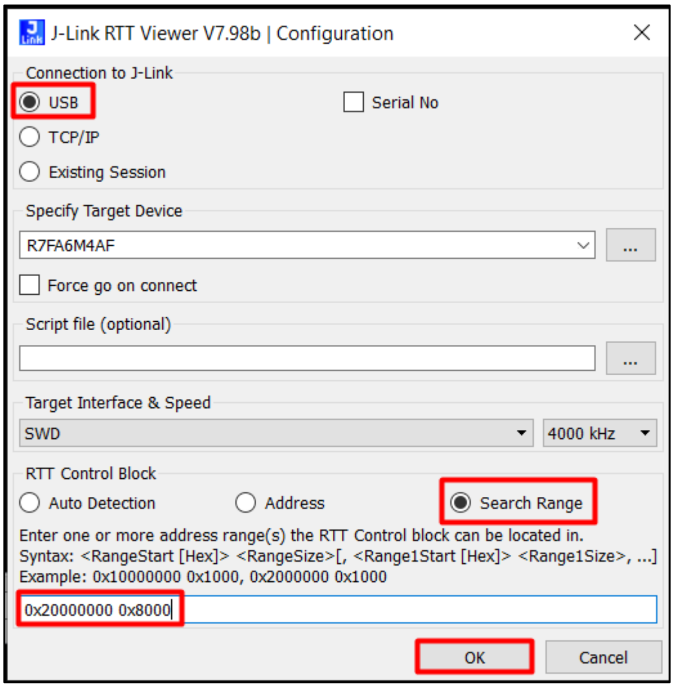

Figure 68. SEGGER RTT Viewer Connection Setup when MCU Running in Standalone Mode

5.5 Example Application with IAR EWARM using Combined Development Model

The IAR-based projects use the Combined Development model. The assumption is that the Secure and Non-secure applications are developed by one team.

Unzip `IAR.zip` to explore the IAR project contents.

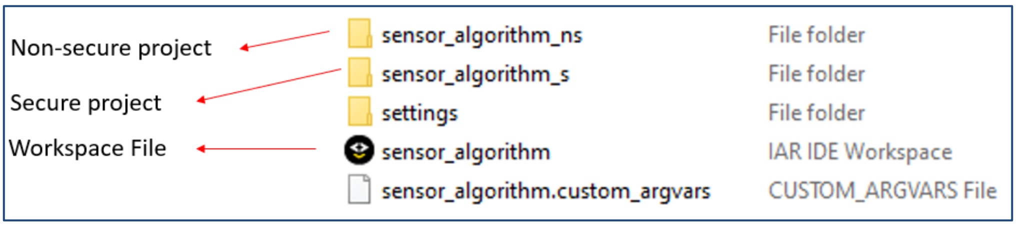

Figure 69. IAR EWARM Software Project Content

5.5.1 Import and Build the Example Projects

Use the following steps to build the IAR example project:

1. Double-click on `\IAR\sensor_trustzone.eww` to launch the IAR EWARM. There are two projects in this workspace. Click on the Secure project `sensor_s` to make it the active project.
2. Notice that the header file `sensor_algorithm_nsc.h`, which includes the user-defined NSC functions, is included in both the Secure project and the Non-secure project.
3. Select **Tools > RA Smart Configurator**.

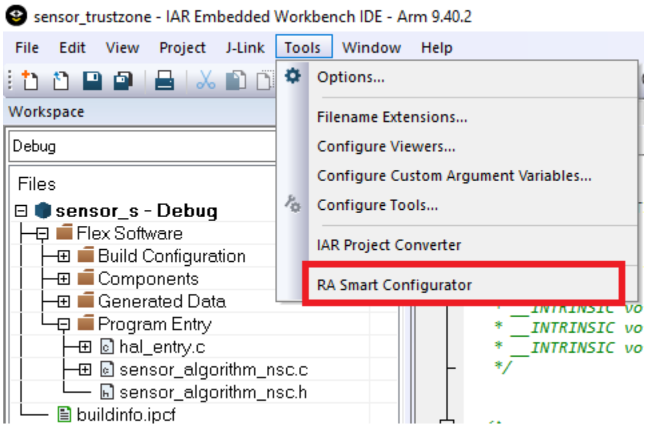

Figure 70. Launch RA Smart Configurator from IAR

4. Once the RA Smart Configurator is launched, click **Generate Project Content**.
5. Close the RA Smart Configurator.
6. Return to the EWARM IDE, right-click on `sensor_s,` and select **Rebuild All**. The Secure project will be compiled.
7. Select the Non-secure project `sensor_ns` to make it the active project.
8. Select **Tools > RA Smart Configurator**.
9. Click **Generate Project Content**.
10. Return to the EWARM IDE and check if there is a `\Objects` folder under `\Flex_Software` and `secure.o` exists in the `\Objects` folder. If yes, the non-secure project will be compiled with no issue. If not, then the non-secure project will need to be compiled twice. The first compile will issue an error message similar to Figure 72. The second compile process will succeed. This is because there is a timing issue between EWARM and RSAC operations.

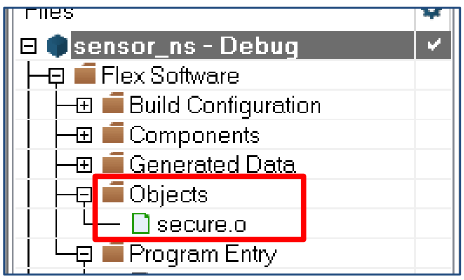

Figure 71. Check that the secure.o is included in the project

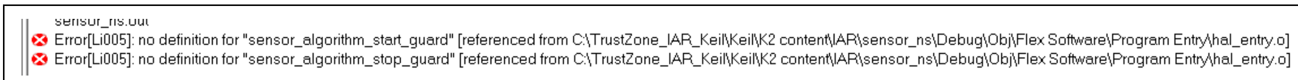

Figure 72. Potential Error Message

5.5.2 Download and Debug the Application Projects

Prior to downloading and running the example project, the user should first follow section 5.3 to set up the MCU.

Then, use the following steps:

1. Click on the Project tab **Project > Options > Debugger > Setup** and notice that `partition_device.mac` is selected. This macro defines the IDAU boundary setting generated.
2. Switch to the **Debugger > Images** window and notice that the Secure image is also downloaded.

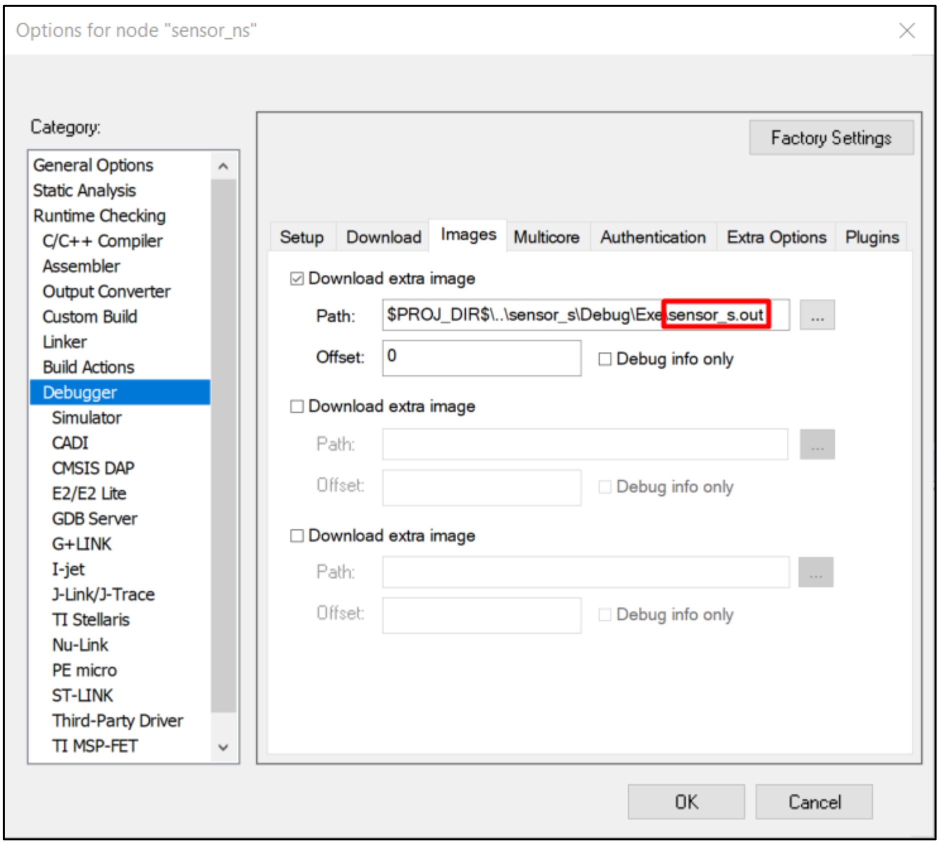

Figure 73. Non-secure Project Debug Configuration to Download the Secure Project

3. Click Download and Debug .
4. If the current MCU IDAU region setup differs from the boundary calculated from the Secure project, the window shown in Figure 74 will appear, prompting you to set up the IDAU region.

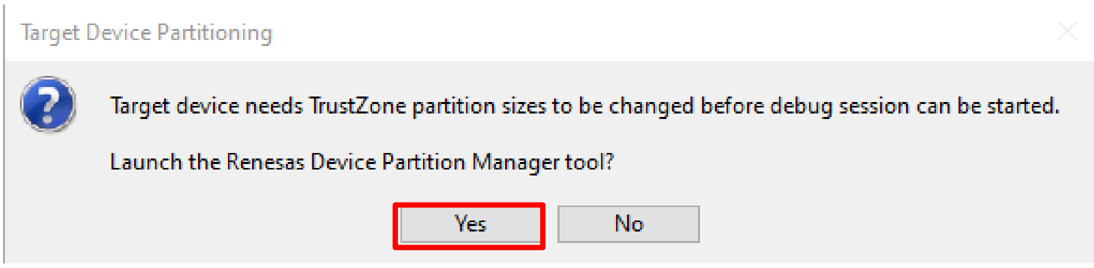

Figure 74. Choose to Launch Renesas Device Partition Manager

Once the Renesas Device Partition Manager is launched, configure the settings as shown in Figure 75. Use **Browse** to select the .rpd file generated from the secure project (`sensor_s.rpd`) as the input for the **User Renesas Partition Data file** entry.

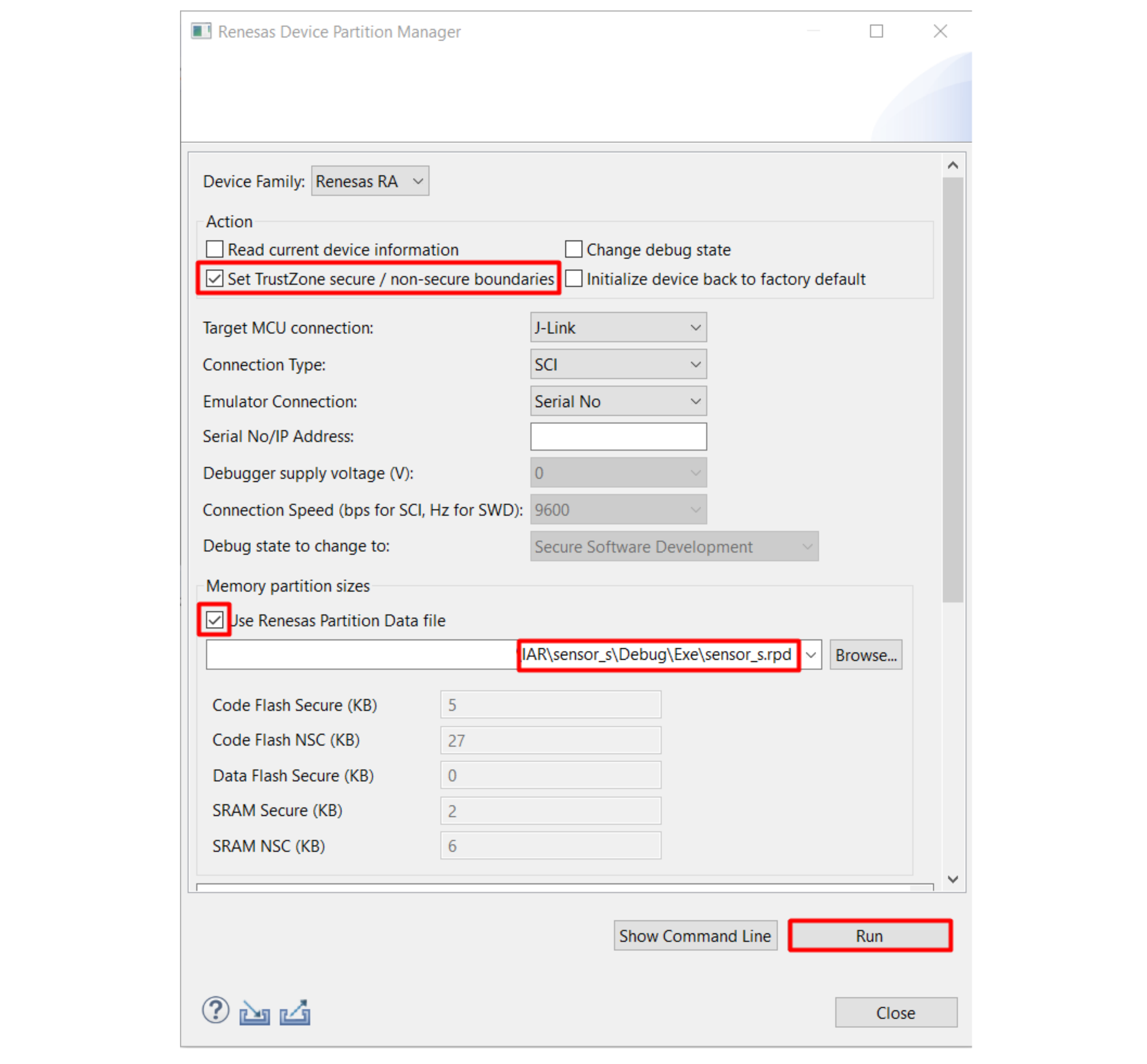

Figure 75. Configure the Renesas Device Partition Manager

5. Click **Run** to set up the IDAU region.

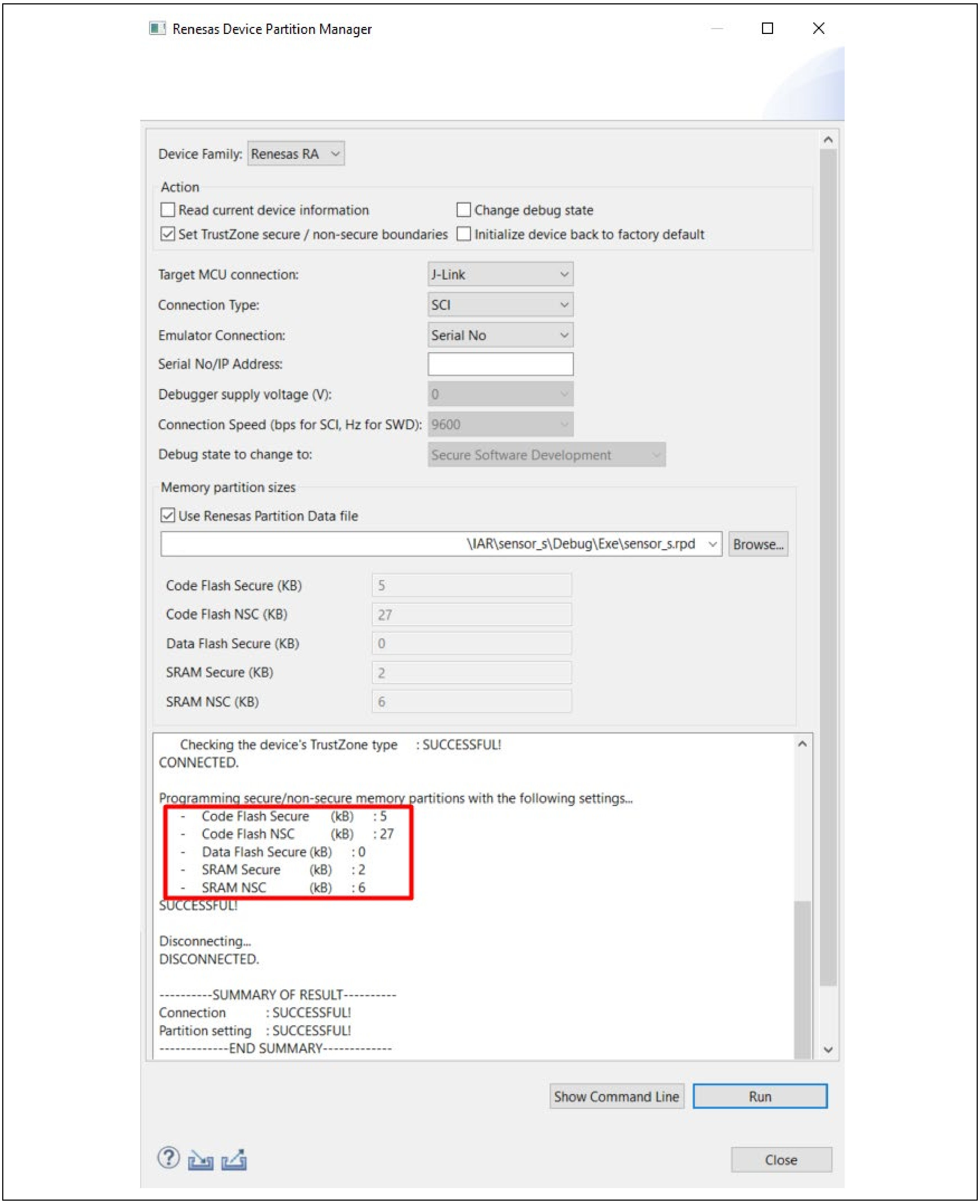

Figure 76. Renesas Device Partition Manager IDAU Result

6. Click **Close** to close the RDPM.
7. Navigate to the EWARM IDE, click Download and Debug, to program the Secure and Non-secure applications. When the execution stops at Reset_Handler, click the Go button to resume the execution.
8. See section 5.4.3 to verify the functionality of the project.

5.6 Example Application with Keil MDK using Combined Development Model

The Keil MDK-based projects utilize the Combined Development model. The assumption is that the Secure and Non-secure applications are developed by one team.

Unzip `Keil.zip` to explore the IAR project contents.

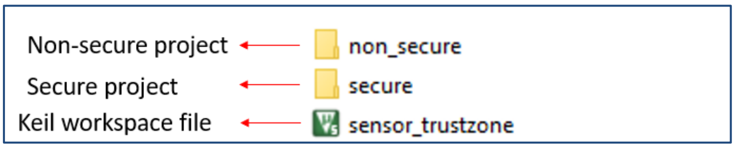

Figure 77. Keil MDK Software Project Content

5.6.1 Import and Build the Example Projects

Follow the steps below to build the Keil example projects:

1. Launch Keil MDK with Administrator authority. Right-click on UV4 and select Run as administrator.
2. Open the multi-project Workspace `sensor_trustzone.uvmpw`.

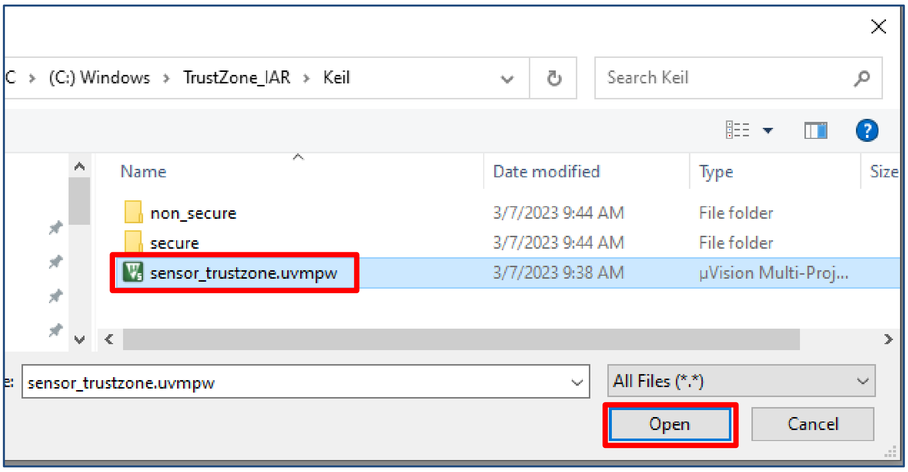

Figure 78. Open the Keil Multi-project Workspace

3. Set the `sensor_s` as the Active Project and then launch the RA Smart Configurator.

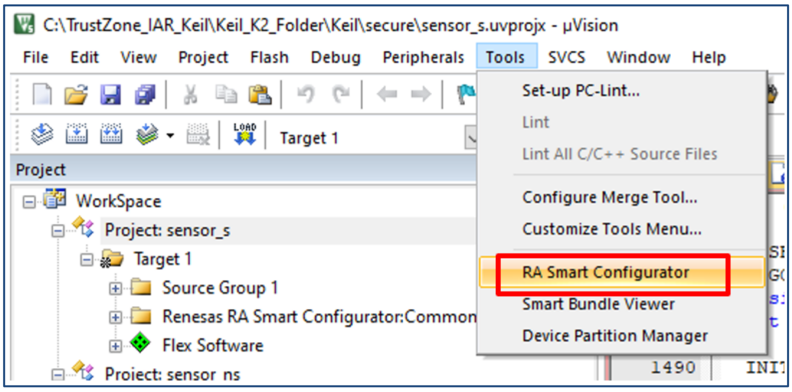

Figure 79. Launch RA Smart Configurator from Keil MDK

4. Once the RA Smart Configurator is launched, click **Generate Project Content**.
5. Close the RA Smart Configurator.
6. Return to the MDK IDE, and click **Project**->**Build 'sensor_s'**.

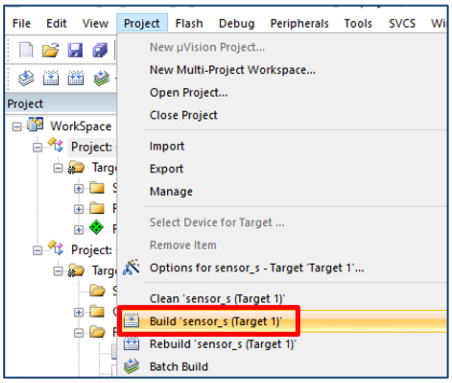

Figure 80. Build the Secure Project

7. The Secure project will be compiled.
8. Follow section 5.3 to set up the MCU.
9. Launch Device Partition Manager and set up the IDAU region using the sensor_s.rdp in a similar way as in Figure 75.
10. Ensure that the IDAU region is successfully set up.

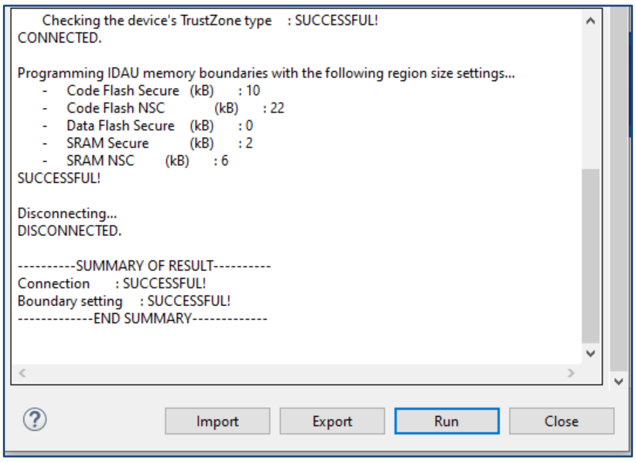

Figure 81. IDAU Region is Configured Correctly

11. Close the Device Partition Manager.

12. Right-click on the Non-secure project `sensor_ns` and set it as the Active Project.

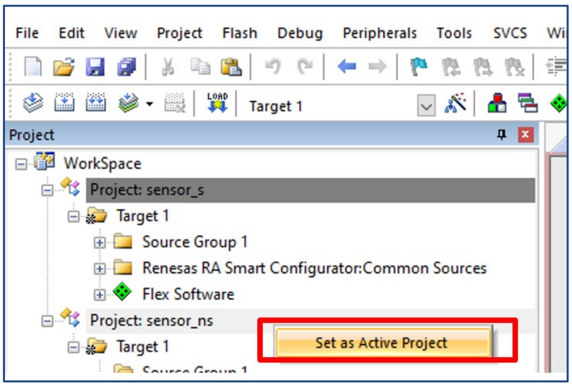

Figure 82. Set the Non-secure Project as the Active Project

13. Select **Tools > RA Smart Configurator**.
14. Click **Generate Project Content**.
15. Close the RA Smart Configurator
16. Return to the Keil MDK IDE and select **Project -> Build 'sensor_ns'**.

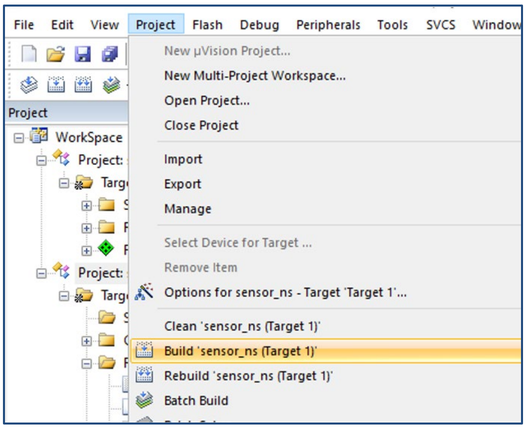

Figure 83. Build the Non-secure Project

17. The non-secure project will compile successfully with no issues.

5.6.2 Download and Debug the Application Project

Follow the steps in this section to debug the system.

1. With `sensor_ns` as the Active Project, click the Start/Stop Debug Session button.

Figure 84. Start Debug with Keil MDK

2. Click **Run** and then follow section 5.4.3 to verify the functionality of the application project.

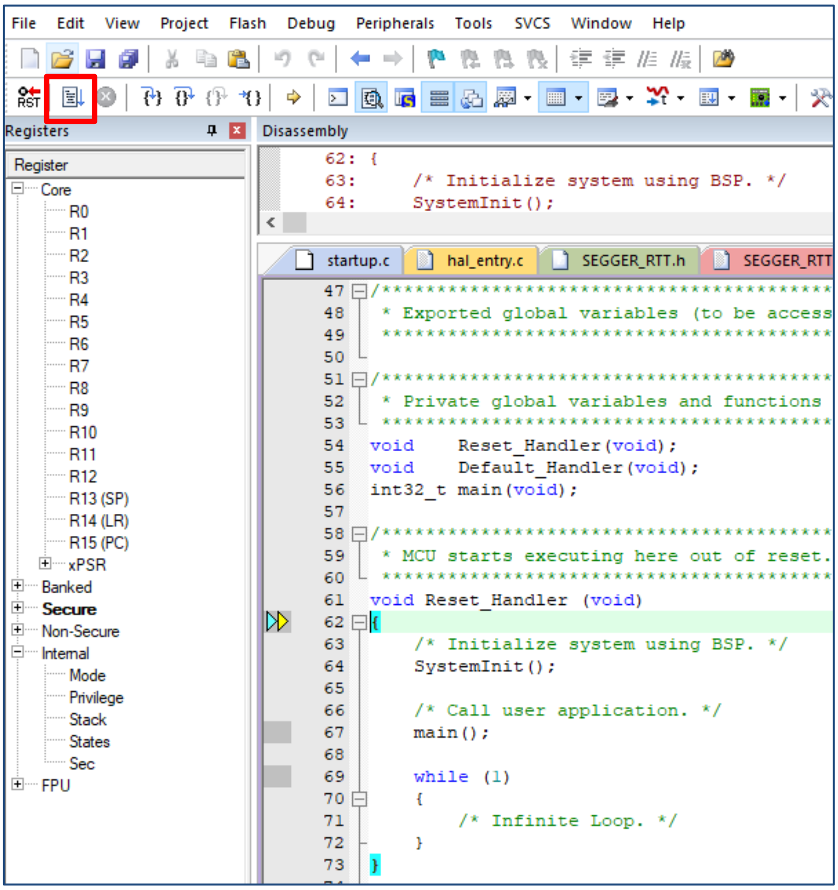

Figure 85. Run the Application Project

3. Follow section 5.4.3 to verify the functionality of the example projects.

6. Appendix A: Using Renesas Flash Programmer for Production Flow

- All instructions in this section are based on connection to RFP using a J-Link debugger over USB. For other connections, refer to the *RFP User's Manual* for instructions.
- All the instructions provided in this section are for supporting the production flow of the e² studio example application explained in section 5.4. The difference in the production operation between the Combined Project Development model and the Split Project Development model will be pointed out. However, providing detailed instructions on the production flow of the Combined Project Development model is out of the scope of this application project. Users need to adjust these RFP projects with the IDAU region setup if different projects are used.

6.1 Initialize the MCU

Follow the steps in section 5.3 to establish the hardware connections. Then, launch RFP, open "`\RFP_projects\initialize_mcu\initialize_mcu.rpj`", go to the tab **Device Information**, and select **Initialize Device**.

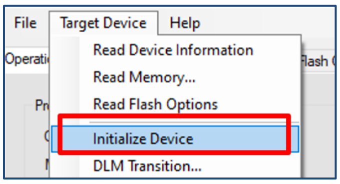

Figure 86. Initialize using RFP

Figure 87. MCU is Successfully Initialized

6.2 Download the Secure Binary

Open the attached RFP project `\RFP_projects\pre_programmed_sensor_algorithm_s\pre_programmed_sensor_algorithm_s.rpj` to perform the following functions:

- Program the Secure binary.
- Set up IDAU regions.
- Transition to NSECSD.

Note that the demonstration in this section is based on the configuration in the e² studio projects demonstrated in section 5.4.

Figure 88 shows the settings for the **Operation Settings** tab:

- Choose **Program** and **Verify** so that the Secure binary can be programmed and verified.
- Choose **Program Flash Option** and **Verify Flash Option** so that the IDAU and device lifecycle state can be set up and verified.
- **Erase** is not selected, as this has been taken care of with the Initialize command, as shown in section 6.1.

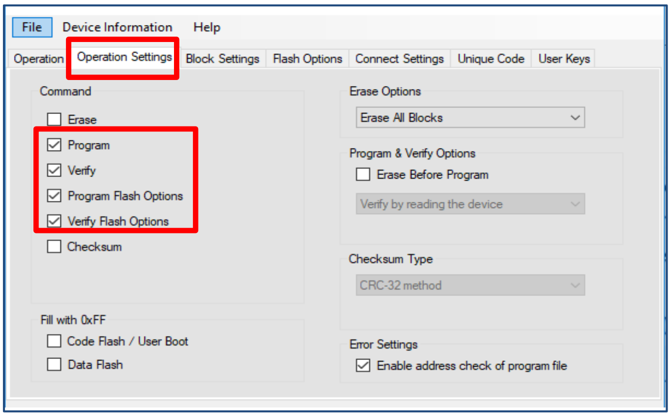

Figure 88. Set up Operation Settings (RFP)

Figure 89 shows the setup for the DLM state transition and IDAU region setup for this example application.

Note: With RFP, you can directly transition the MCU device lifecycle state from SSD to NSECSD without needing to download the dummy Non-secure binary. The dummy Non-secure binary is only needed to start the development of the Non-secure project.

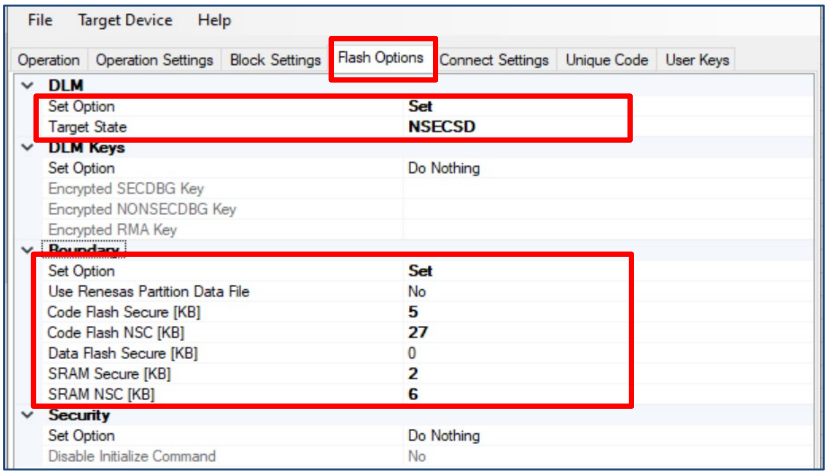

Figure 89. Setup for the IDAU Region

Settings for the connection interface are shown in Figure 90.

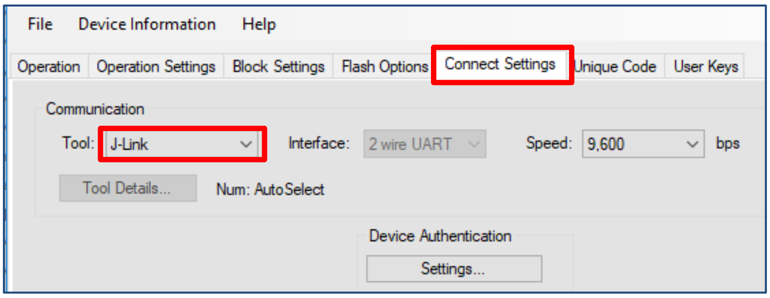

Figure 90. Setup for the Connection

Select the Secure project binary (`.srec` or `.hex`) generated to be programmed into the MCU. Select the binary generated from section 5.4.1.2.

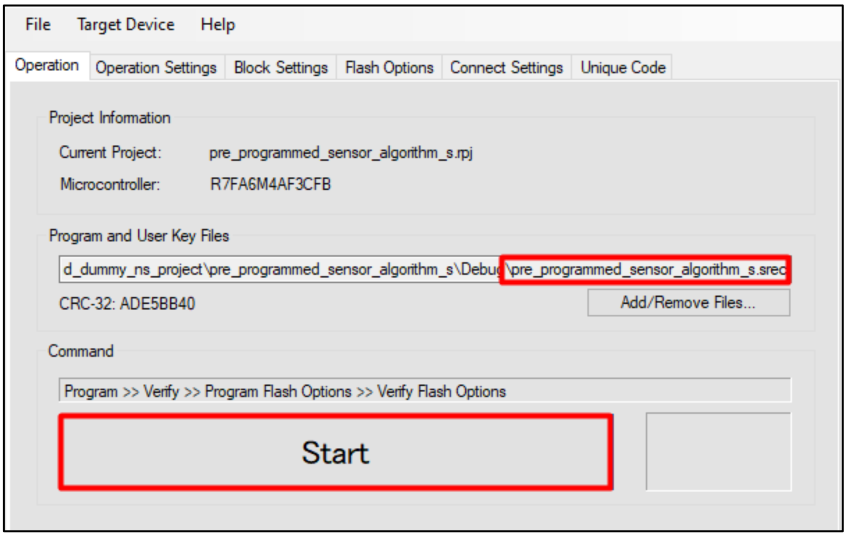

Figure 91. Select the Secure Binary to Program into the MCU

With all settings in place, click **Start** to download the Secure binary and set up the IDAU region.

6.3 Download the Non-secure Binary

Use RFP to download the Non-secure project binaries using the provided RFP project: `\RFP_projects\pre_programmed_sensor_algorithm_ns\pre_programmed_sensor_algorithm_ns.rpj`.

Check **Program Flash** and **Verify Flash**, uncheck **Program Flash Option** and **Verify Flash Option** from the **Operation Settings** tab.

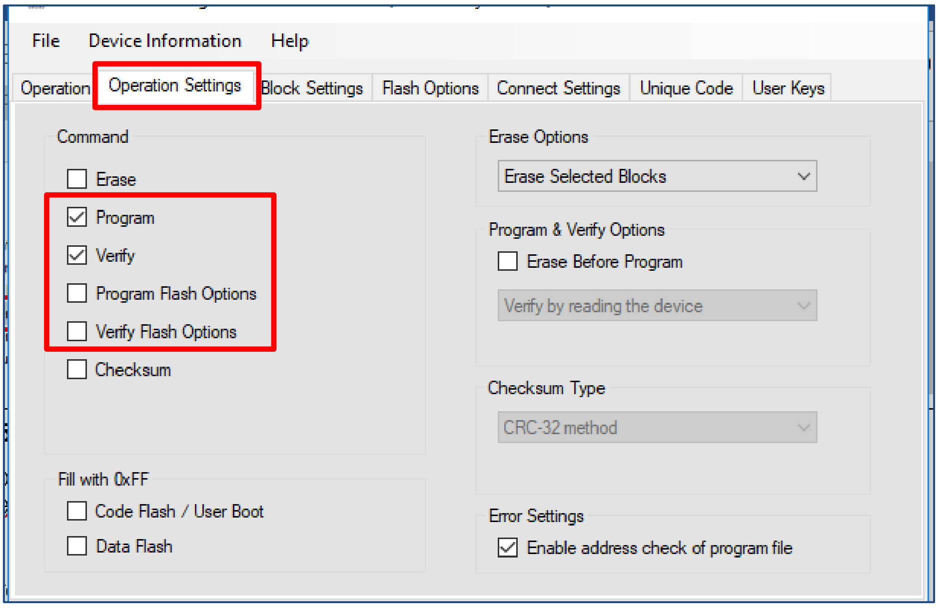

Figure 92. Operation Settings for Non-secure Project Binary Download

Transition to DPL is not selected. Change from **Do Nothing** to **Set** in production flow. Once the device lifecycle state is transitioned to DPL, the JTAG interface will be disabled (no SEGGER RTT Viewer input/output functionality).

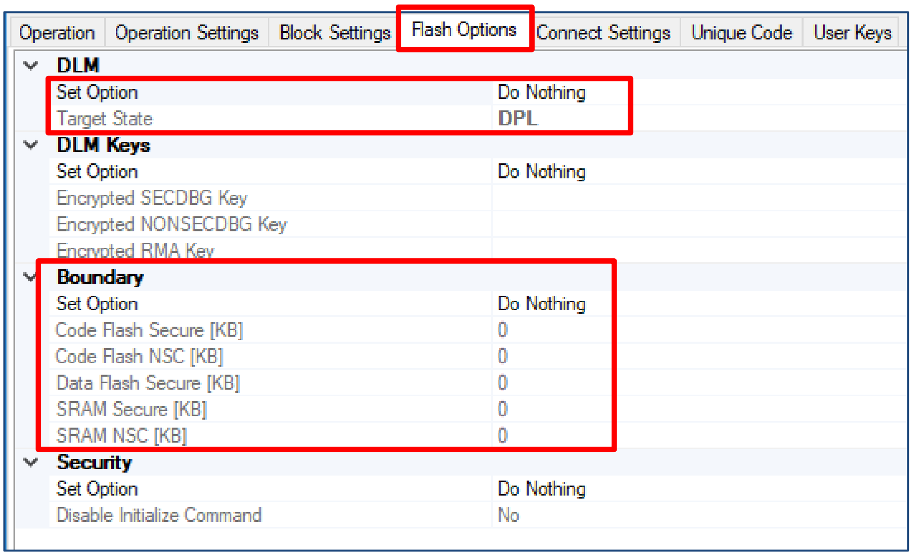

Figure 93. Operation Settings for Non-secure Project Binary Download

The **Connect Settings** should use the same setup as shown in Figure 90.

Select the Non-secure binary generated from section 5.4.2.2.

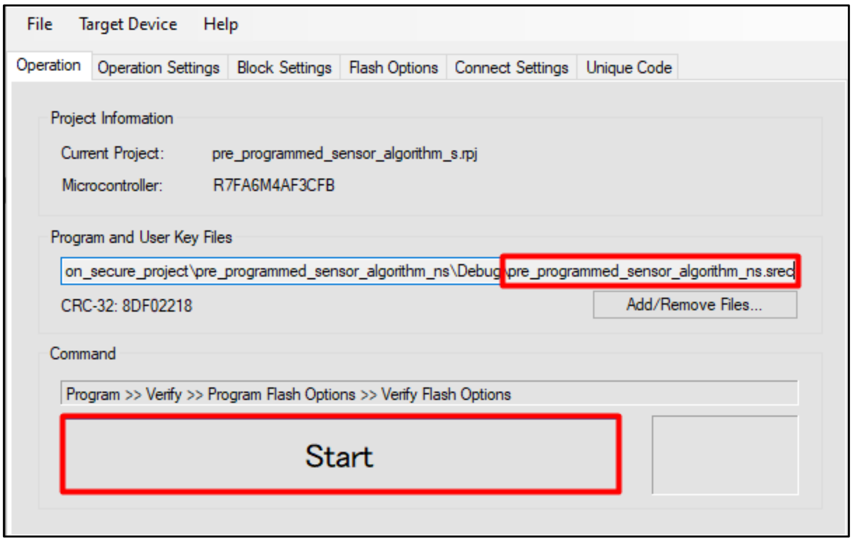

Figure 94. Select the Non-secure Binary

With all the above settings, click **Start** to download the Non-secure binary.

The production flow of the IP protection use case also requires advancing the device lifecycle state from DPL to LCK_DBG or LCK_BOOT. However, once the device lifecycle state advances to LCK_DBG, the debug interface will be permanently locked. Once the device lifecycle state advances to LCK_BOOT, the serial programming interface will be permanently locked. To avoid accidental MCU debug and serial programming interface locking, do not transition the device lifecycle state to LCK_DBG or LCK_BOOT unless you are doing so for production usage.

6.4 Specific Instructions to Support IAR EWARM Development Path

6.4.1 IAR I-jet and TrustZone® Partition Boundary Setup

IAR's I-jet debug probe does not provide support for setting the TrustZone partition boundaries, as it does not have the ability to interface with the RA MCU's boot mode through the debug header.

It is, therefore, necessary to set the TrustZone partition boundaries appropriately using alternative means before debugging through I-jet. Typically, this will need to be done using an SCI connection to the board/MCU and the Renesas Flash Programmer (RFP) application available from:

https://www.renesas.com/us/en/software-tool/renesas-flash-programmer-programming-gui

Figure 95 shows RFP configured to read the TrustZone partition boundaries from a `.rpd` file.

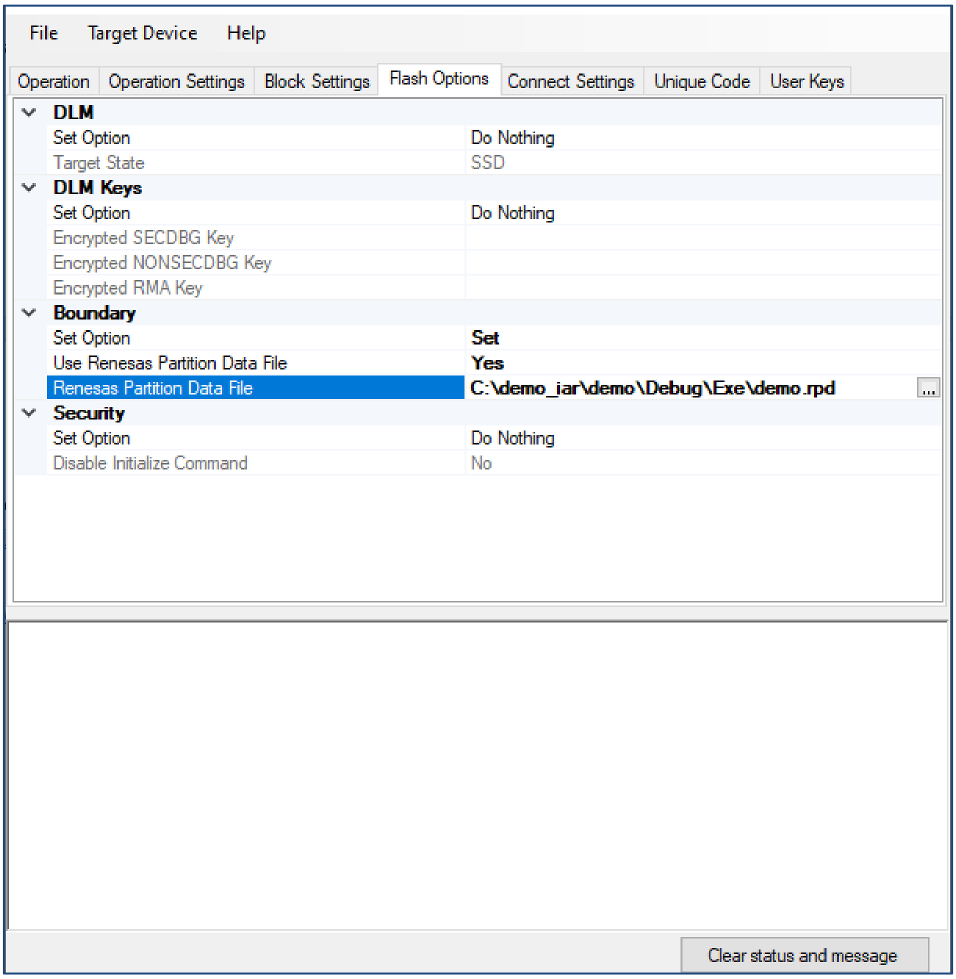

Figure 95. Configure TrustZone® Partition

6.4.2 CMSIS-DAP and Trust Zone Partition Boundary Setup

EWARM also supports the use of CMSIS-DAP-based debug probes. These do not have the ability to interface with the RA MCU's boot mode through the debug header.

7. Appendix B: Glossary

Term	Meaning
SSD	Device Lifecycle State: **S**ecure **S**oftware **D**evelopment. The debugging level is DBG2. IDAU region can be set up in this state.
NSECSD	Device Lifecycle State: **N**on-**SEC**ure **S**oftware **D**evelopment. The debugging level is DBG1.
DPL	Device Lifecycle State: **D**e**PL**oyed. The debugging Level is DBG0.
SCE9	**S**ecure **C**rypto **E**ngine **9**: An isolated subsystem within the MCU protected by an Access Management Circuit. Performs Cryptographic operations.

8. References

1. Renesas RA6M4 Group User's Manual: Hardware
2. Flexible Software Package (FSP) User's Manual
3. Arm® TrustZone® Technology for the Armv8-M Architecture
4. Renesas RA Family Device Lifecycle Management Key Installation (R11AN0469EU)
5. Renesas RA Family Securing Data at Rest using Arm® TrustZone® (R11AN0468EU)
6. Arm®v8-M Architecture Reference Manual
7. Arm® Cortex®-M33 Processor Technical Reference Manual
8. Arm® Cortex®-M33 Devices Generic User Guide

9. Website and Support

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

EK-RA6M4 Resources	renesas.com/ra/ek-ra6m4
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.01.20	—	Initial release
1.10	Jun.2.21	—	Update to FSP v3.0.0 and remove usage instructions with E2
1.20	Feb.15.23	—	Add IAR Support and Update to FSP v4.0.0
1.30	Apr.10.23	—	Add Keil Support and Update to FSP v4.2.0
1.40	Jan.24.24	—	Update to FSP v5.0.0
1.50	Mar.29.24	Section 4.3.2	Update Ethernet buffer Non-secure region allocation in flat project.
1.60	Apr.17.24	IAR project	Add debug configuration files
1.70	Sep.20.24	—	Update to FSP v5.5.0

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Corporate Headquarters

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

Contact information

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

Trademarks