Renesas RA Family

RA AWS Cloud Connectivity and Firmware Update OTA on CK-RA6M5 v2 with Ethernet

Introduction

This application note offers a step-by-step guide to creating IoT Cloud connectivity solutions utilizing AWS IoT Core. It includes instructions for implementing firmware updates Over-The-Air (OTA) and step-by-step creation of a Bootloader project using MCUboot for firmware upgrades.

The provided application example is centered around showcasing the Renesas RA Cloud connectivity solution and highlighting Firmware updates. It demonstrates the integration of AWS IoT Core and underscores the firmware update capabilities supported within the Renesas FSP, along with secure MCUboot.

This application note helps developers effectively use FSP MQTT/TLS modules and AWS OTA in their product designs. By following the guide, developers can integrate "AWS Core MQTT," "Mbed TLS," and "AWS TCP Sockets Wrapper" via Ethernet, along with AWS OTA libraries. They'll configure these components for their applications and use the provided example code as a reference for streamlined development.

References to detailed API descriptions and other application projects that demonstrate more advanced uses of the module are in the *FSP User's Manual* (available at: https://renesas.github.io/fsp/), which serves as a valuable resource in creating more complex designs.

This MQTT/TLS AWS Cloud Connectivity solution is supported on the CK-RA6M5 v2 Kit.

Applies to:

RA6M5 MCU Group

Required Resources

To build and run the application project, the following resources are needed.

Development tools and software

- Flexible Software Package (FSP) v6.1.0 and required tools (renesas.com/us/en/software-tool/flexible-software-package-fsp)
- Openssl: v3.0.12 or later (OpenSSL website: https://www.openssl.org/, OpenSSL App for Windows OS: https://slproweb.com/products/Win32OpenSSL.html)
- Python: v3.12.0 or later (https://www.python.org/downloads/)

Hardware

- Renesas CK-RA6M5 v2 kit (renesas.com/ra/ck-ra6m5)
- PC running Windows® 10 or Windows® 11
- Micro USB cables included as part of the kit. See *CK-RA6M5 v2 — User's Manual*)
- USB-C cable for Power supply (See *CK-RA6M5 v2 — User's Manual)*
- CAT5 Ethernet cable
- A router/switch with at least one available 100 Mbps/full duplex Ethernet port with connectivity to the internet

Prerequisites and Intended Audience

This application note assumes that the user is adept at operating the Renesas e² studio IDE with Flexible Software Package (FSP). If not, we recommend reading and following the procedures in the *FSP User's Manual* sections for 'Starting Development,' including 'Debug the Blinky Project.' Doing so enables familiarization with e² studio and FSP and validates proper debug connection to the target board. In addition, this application note assumes prior knowledge of Cloud connectivity and its communication protocols, and knowledge of firmware upgrades over the air.

The intended audience is users who intend to develop a solution for updating firmware over the air using AWS services and the Renesas RA MCU series.

Note: If you are a first-time user of e² studio and FSP, we highly recommend you install e² studio and FSP on your system to run the Blinky Project and to get familiar with the e² studio and FSP development environment before proceeding to the next sections.

Note: This Application Project and Application Note can only use versions FSP v6.1.0.

Prerequisites

1. Access to online documentation is available in the **References** section.
2. Knowledge of the MCU Bootloader and access to the Renesas Bootloader documentation.
3. Knowledge of OTA and access to the AWS OTA documentation.
4. Access to the latest documentation for the identified Renesas Flexible Software Package.
5. Prior knowledge of operating the e² studio and the built-in (or standalone) RA Configurator.
6. Access to associated hardware documentation, such as User Manuals, Schematics, and other relevant kit information (renesas.com/ra/ck-ra6m5).

Contents

1. Introduction to Components for Cloud Connectivity

1.1 General Overview

The Internet-of-Things (IoT) is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. The 'things' in this definition are objects in the physical world (physical objects) or information world (virtual) that can be identified and integrated into communication networks. In the context of the IoT, a 'device' is a piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage, and data processing. Communication is often performed with providers of network-hosted services, infrastructure, and business applications to process/analyze the generated data and manage the devices. Such providers are called Cloud Service Providers. While there are many manufacturers of devices and cloud service providers, for the context of this application note, the device is a Renesas RA Microcontroller (MCU) connecting to services provided by Amazon Web Services (AWS) for IoT.

1.2 Cloud Service Provider

AWS IoT provides the cloud services that connect your IoT devices to other devices and AWS cloud services As a Cloud Service Provider, AWS IoT provides the ability to:

- Connect and manage devices.
- Secure device connections and data.
- Process and act upon device data.
- Read and set the device state at any time.

Figure 1 summarizes the features provided by AWS IoT.

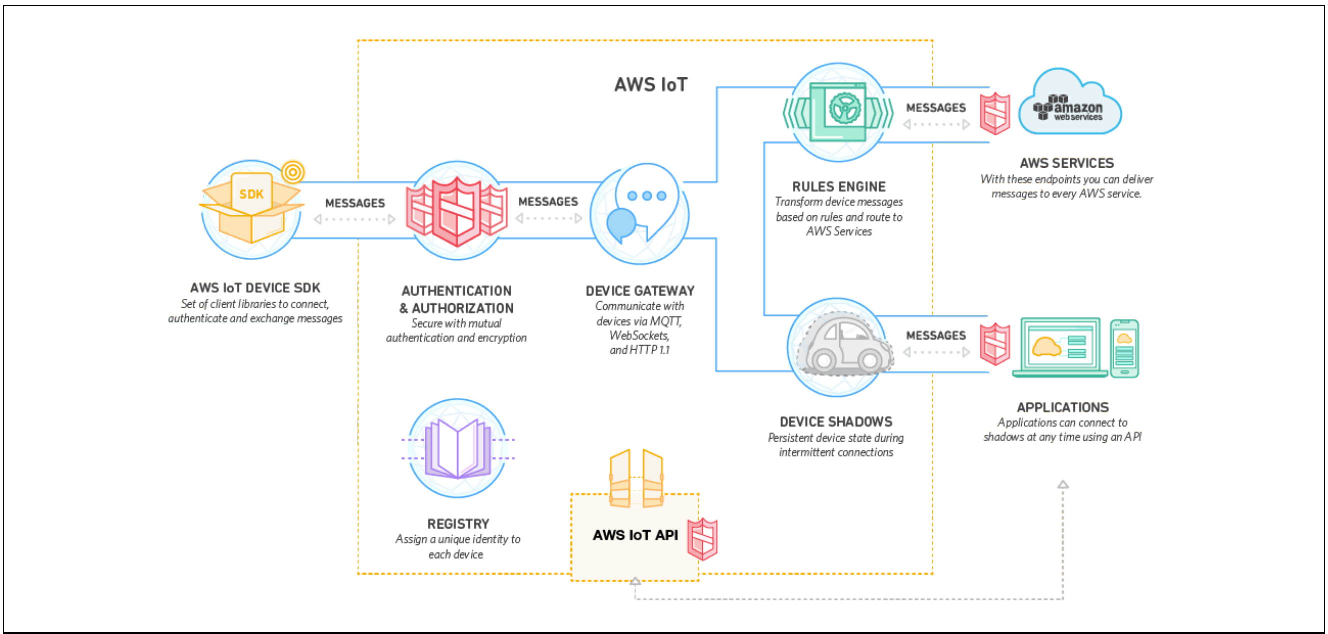

Figure 1. AWS IoT Features, Service Components, and Data Flow Diagram

A key feature provided by AWS is the AWS IoT Software Development Kit (SDK) written in C, which allows devices such as sensors, actuators, embedded micro-controllers, or smart appliances to connect, authenticate, and exchange messages with AWS IoT using the MQTT, HTTP, or WebSocket's protocols. This application note focuses on configuring and using the AWS IoT Device SDK and the MQTT protocol included, which is available through the Renesas Flexible Software Package (FSP) for Renesas RA MCUs.

1.3 AWS IoT Core

AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT Core can support billions of devices and trillions of messages. It can process and route messages to AWS endpoints and other devices reliably and securely. With AWS IoT Core, customer applications can keep track of all devices all the time, even when devices are not connected.

AWS IoT Core addresses security concerns for the infrastructure by implementing mutual authentication and encryption. AWS IoT Core provides automated configuration and authentication upon a device's first connection to AWS IoT Core, as well as end-to-end encryption throughout all points of connection, so that data is only exchanged between devices and AWS IoT Core with proven identity.

This application note focuses on complementing the security needs of AWS IoT Core by installing a proven identity for the RA MCU by storing an X.509 certificate and asymmetric cryptography keys in Privacy Enhanced Mail (PEM) format in the onboard flash. The RA MCU has on-chip security features, such as Key Wrapping, to protect the private key associated with the public key and the certificate associated with the device[1]. Additionally, RA MCUs can also generate asymmetric keys using features of the Secure Engine and API available through the FSP. The Secure Engine accelerates symmetric encryption/decryption of data between the connected device and AWS IoT, allowing the ARM Cortex-M processor to perform other application-specific computations.

1.4 MQTT Protocol Overview

This application note features Message Queuing Telemetry Transport (MQTT) as it is a lightweight communication protocol specifically designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements. MQTT uses a publish/subscribe architecture, which is designed to be open and easy to implement, with up to thousands of remote clients capable of being supported by a single server. These characteristics make MQTT ideal for use in constrained environments where network bandwidth is low or where there is high latency, and with remote devices that might have limited processing capabilities and memory. The RA MCU device in this application note implements a Core MQTT service that communicates with AWS IoT and exchanges example telemetry information, such as temperature, pressure, humidity, accelerometer, magnetometer, and many more types of sensor data.

1.5 TLS Protocol Overview

The primary goal of the Transport Layer Security (TLS) protocol is to provide privacy and data integrity between two communicating applications or endpoints. AWS IoT mandates the use of secure communication. Consequently, all traffic to and from AWS IoT is sent securely using TLS. TLS protocol version 1.2 or later ensures the confidentiality of the application protocols supported by AWS IoT. A variety of TLS Cipher Suites are supported. This application note configures the RA Flexible Software Package for the MCU-based device to provide the following capabilities, and AWS IoT negotiates the appropriate TLS Cipher Suite configuration to maximize security.

Table 1. TLS with Crypto Capabilities in RA FSP

Secure Crypto Hardware Acceleration	Supported
Key Format Supported	AES, ECC, RSA
Hash	SHA-256
Cipher	AES
Public Key Cryptography	ECC, ECDSA, RSA
Message Authentication Code (MAC)	HKDF

On top of these supported features, Mbed Crypto middleware also supports a variety of features that can be enabled through the RA Configurator. Refer to the *FSP User's Manual* section for the Crypto Middleware (rm_psa_crypto).

[1] This application note does not focus on using Key Wrapping for securely storing the private key for devices deployed in a production environment.

1.6 Device Certificates, CA, and Keys

Device Certificates, Certificate Authorities (CA), and Asymmetric Key Pairs create the foundation for trust needed for a secure environment. The background information on these commonly used components in AWS is provided in this section.

A *digital certificate* is a document in a known format that provides information about the identity of a device. The X.509 standard includes the format definition for public-key certificate, attribute certificate, certificate revocation list (CRL), and attribute certificate revocation list (ACRL). X.509-defined certificate formats (X.509 Certificates) are commonly used on the internet and in AWS IoT for authenticating a remote entity/endpoint, that is, a Client and/or Server. In this application note, an X.509 certificate and asymmetric cryptography key pair (public and private keys) are generated from AWS IoT and installed (during binary compilation) into the RA MCU device running the Core MQTT to establish a *known identity*. In addition, a root Certification Authority (CA) certificate is also downloaded and used by the device to authenticate the connection to the AWS IoT gateway.

Certification Authority (CA) certificates are issued by a CA to itself or a second CA for the purpose of creating a defined relationship between the two CAs. The root CA certificate allows devices to verify that they're communicating with AWS IoT Core and not another server impersonating AWS IoT Core.

The public and private keys downloaded from AWS IoT use RSA algorithms for encryption, decryption, signing, and verification[2]. These key pairs and certificates are used together in the TLS process to:

1. Verify device identity.
2. Exchange symmetric keys for algorithms such as AES for encrypting and decrypting data transfers between endpoints.

---

[2] Public Key length used is 2048 bits.

2. Running the MQTT/TLS Ethernet with OTA Application Example

Refer to the ***RA AWS Cloud Connectivity and Firmware Update OTA on CK-RA6M5 v2 with Ethernet - Getting Started Guide*** as part of this project bundle for details on running the project and how to update new firmware using OTA via AWS.

3. AWS IoT Over-the-air Update Library with Ethernet Interface

3.1 AWS IoT Over-the-air Update Library

The AWS IoT Over-the-air Update Library included in RA FSP facilitates firmware updates via AWS IoT, utilizing various protocols such as MQTT or HTTP, with the support of a secure bootloader. This application project and application note focuses on the MQTT protocol.

When users use FreeRTOS project with FSP (can refer to the step-by-step application creation guide in section **4.3, "Set up the Application and Downloader project"**), the AWS IoT Over-the-air Update Library can be directly imported into a **Thread** stack. It is configured through the RA Configuration Perspective. To add the AWS IoT Over-the-air Update Library to a new thread, open **Configuration.xml** with the RA Configuration. While ensuring that the correct thread is selected on the left, use the tab for **Stacks > New Stack > Search** and search for the keyword **AWS IoT Over-the-air Update Library**.

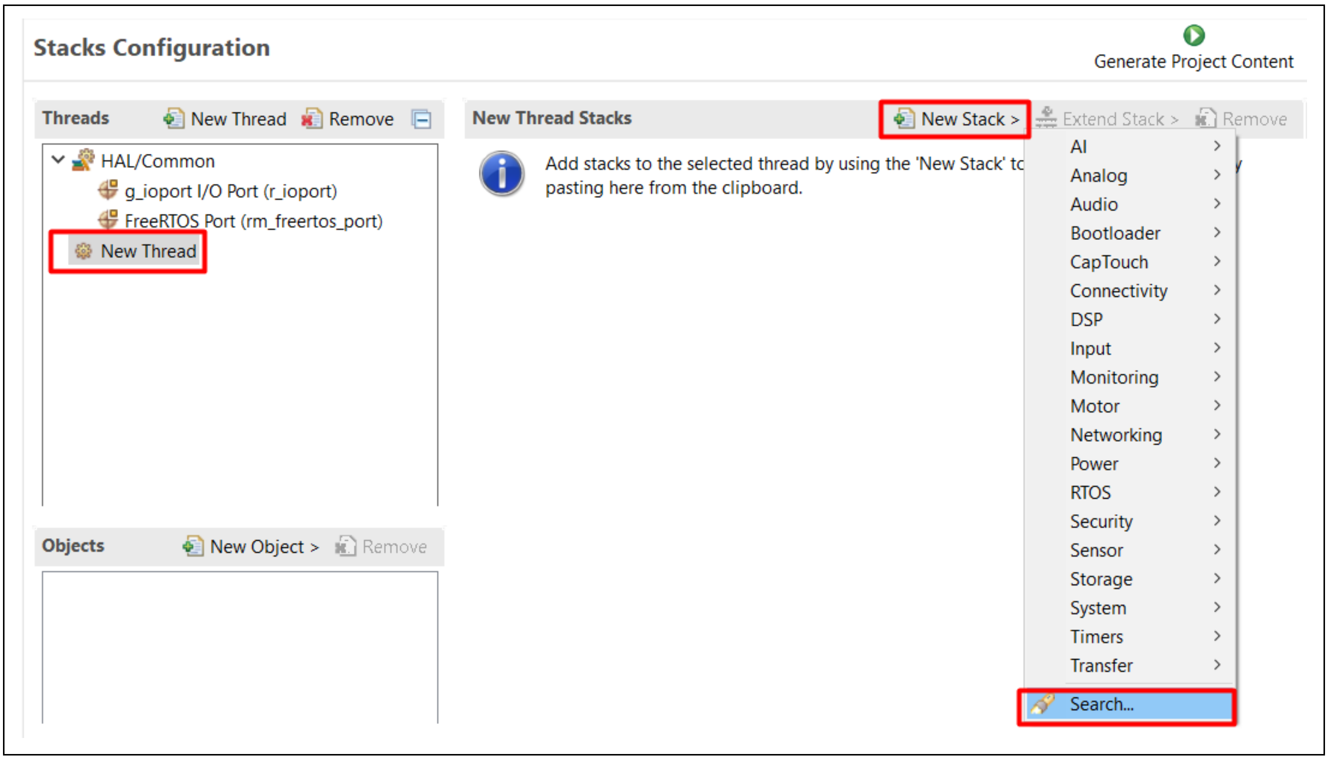

Figure 2. Add AWS IoT Over-the-air Update Library

Adding the AWS IoT Over-the-air Update Library stack results in the default configuration, as shown in Figure 3. AWS IoT Over-the-air Update Library Stack View.

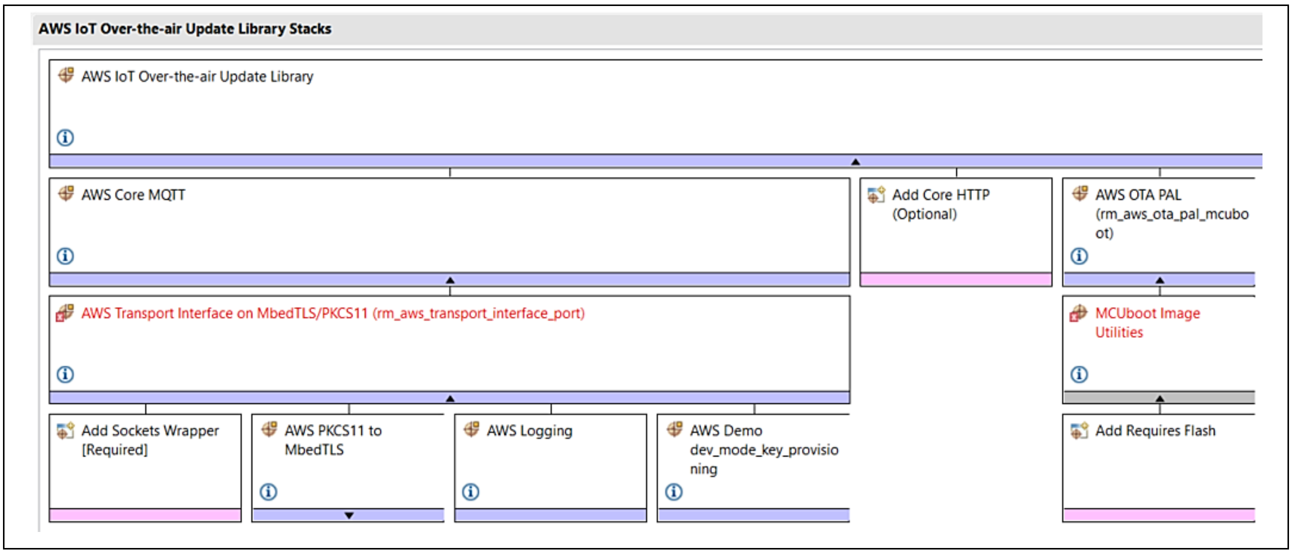

Figure 3. AWS IoT Over-the-air Update Library Stack View

3.2 AWS Core MQTT

The AWS MQTT library included in RA FSP can connect to AWS MQTT. The complete library documentation is available on the AWS IoT Device SDK C: MQTT website. Primary features supported by the library are:

- MQTT connections over TLS to an AWS IoT Endpoint, Mosquitto server, or other MQTT broker.

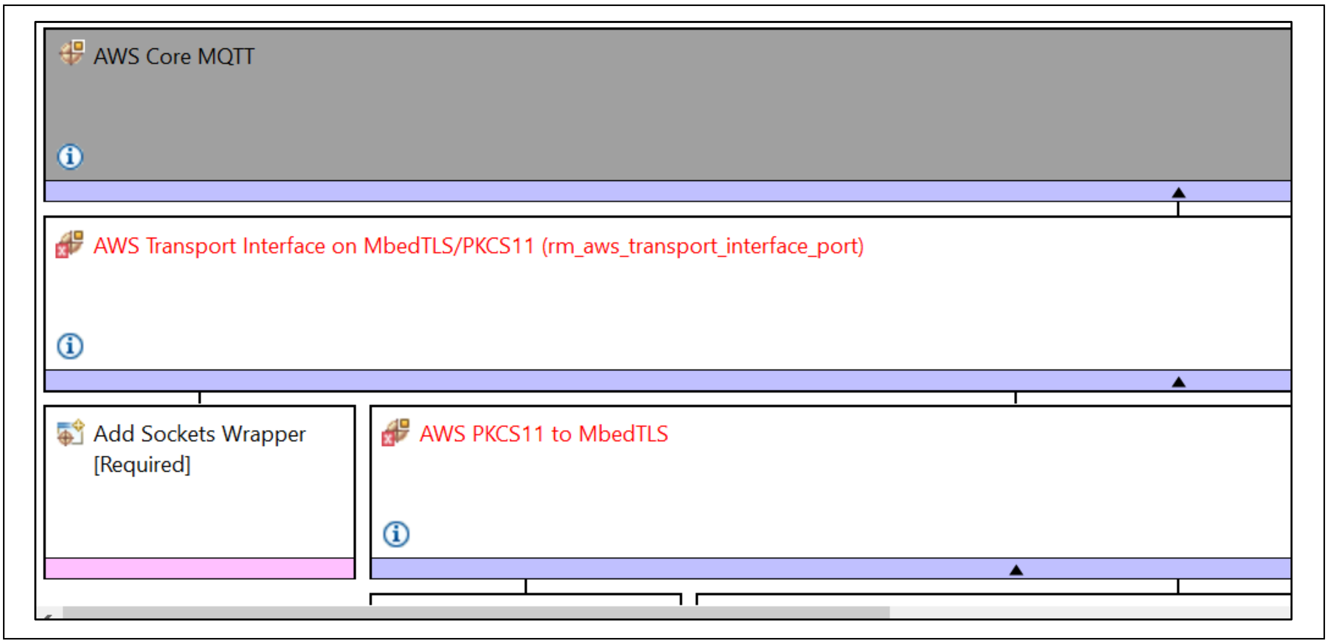

Figure 4. AWS Core MQTT Stack View

While the AWS Core MQTT stack shown includes many dependencies and configurable properties, most default settings can be used as-is. The following change is needed to meet all unmet dependencies (marked in red) for the AWS Core MQTT stack added to a new project (as shown above):

- Enable Mutex and Recursive Mutex usage support as needed by AWS IoT SDK and FreeRTOS in the created Thread properties.

Upon completion of the above step, the AWS Core MQTT is ready to accept a socket implementation, which has dependencies on using a TLS Session and an underlying TCP/IP implementation.

Additional documentation on the AWS Core MQTT is available in the *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > Networking > AWS MQTT.*

3.3 Transport Layer Implementation

The FSP AWS Transport Interface provides options for Wi-Fi, Cellular, and Ethernet. **AWS Transport Interface on MbedTL/PKCS11** module is used for the network communication interface of the AWS Core MQTT. While the RA FSP contains a Secure Socket Implementation for Wi-Fi, Cellular, and Ethernet, this application and application note focus on using the Ethernet Interface.

Ethernet Sockets can be added to the Thread Stack by clicking on **Add Sockets Wrapper > New > AWS TCP Sockets Wrapper**.

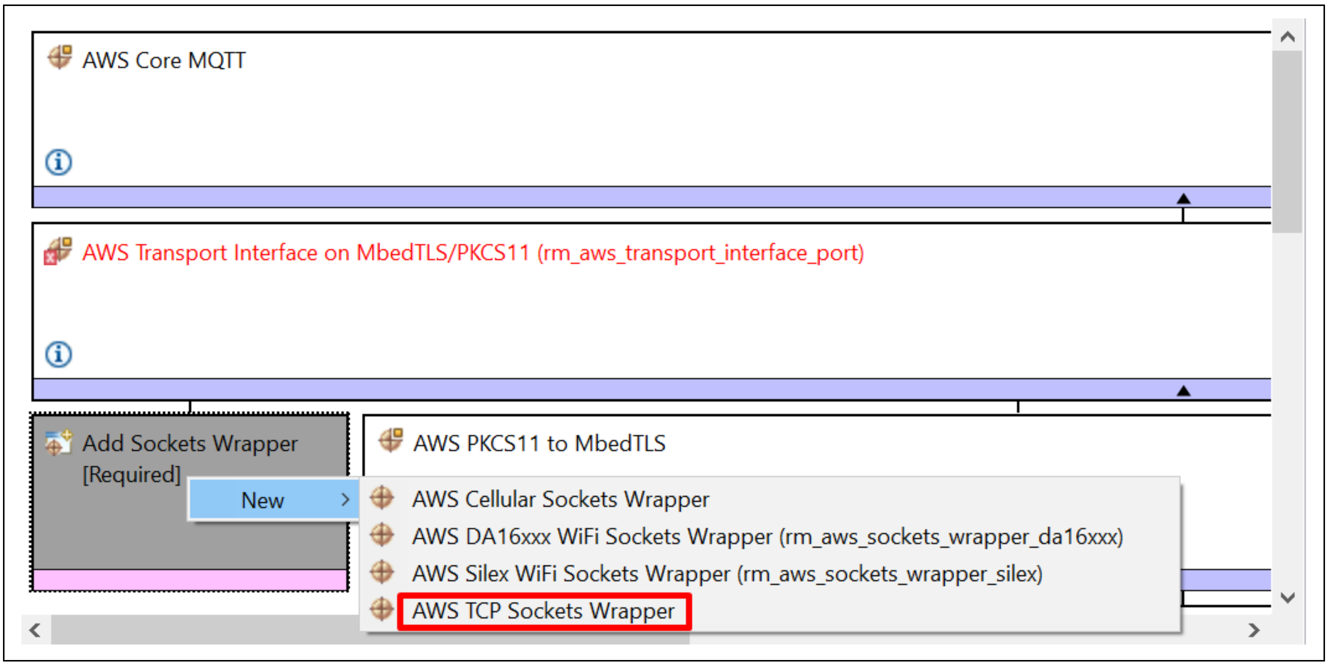

Figure 5. Adding Ethernet Interface to the Core MQTT Module

In addition, the needed stack is complete and has unmet dependencies for the dependent modules.

Now, hover the cursor over the red blocks, and the error will appear. Make the appropriate settings.

- **AWS Transport Interface on MbedTLS/PKCS11** errors:
  For *error: Requires FreeRTOS heap implementation 4 or 5*, choose the heap implementation using **New Stack > RTOS > FreeRTOS Heap 4**. Also, set **Dynamic Memory allocation** in Thread's properties: using **New Thread > Properties > Thread > Memory Allocation > Support Dynamic Allocation > Enabled**.
  For *error: Mutexes must be enabled in the FreeRTOS thread*; enable mutexes in the Thread's common properties using **New Thread > Properties > Common > General > Use Mutexes > Enabled.**
- For **AWS PKCS11 to MbedTLS** *error: MBEDTLS_CMAC_C must be defined*, using **MbedTLS (Crypto Only) > Common > Message Authentication Code (MAC) > MBEDTLS_CMAC_C > Define**.
- For **MbedTLS** *error: MBEDTLS_ECDH_C must be defined*, **using MbedTLS (Crypto Only) > Common > Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_C > Define**.
- **MbedTLS (Crypto Only)** *errors relate to minimum RTOS heap*, set Heap Memory allocation using **New Thread > Properties > Thread > Memory Allocation > Total Heap Size > 0x20000.**
- For **LittleFS** *error: A heap is required to use Malloc*, add a heap under **BSP Tab > Properties > RA Common > Heap size (bytes) > 0x20000**.
- For **LittleFS** *errors:*
  - *A heap is required to use Malloc*; add a heap under **BSP Tab > Properties > RA Common > Heap size (bytes) > 0x20000**.
  - Add a stack for LittleFS: Add **LittleFS Port [Required] > New > LittleFS on Flash (rm_littlefs_flash).**
- Mutexes must be enabled using **New Thread > Common > General > Use Mutexes > Enabled.**
- Mutexes must be enabled using **New Thread > Common > General > Use Recursive Mutexes > Enabled.**

Note: These are the basic settings required to remove the error from the configurator. More specific configurations are listed in the specific module and its usage.

After all the appropriate settings have taken care of the errors due to the missing configuration, the new configurator screenshot looks clean with no mistakes, as shown in Figure 6. Expanded TCP Socket Interface Module.

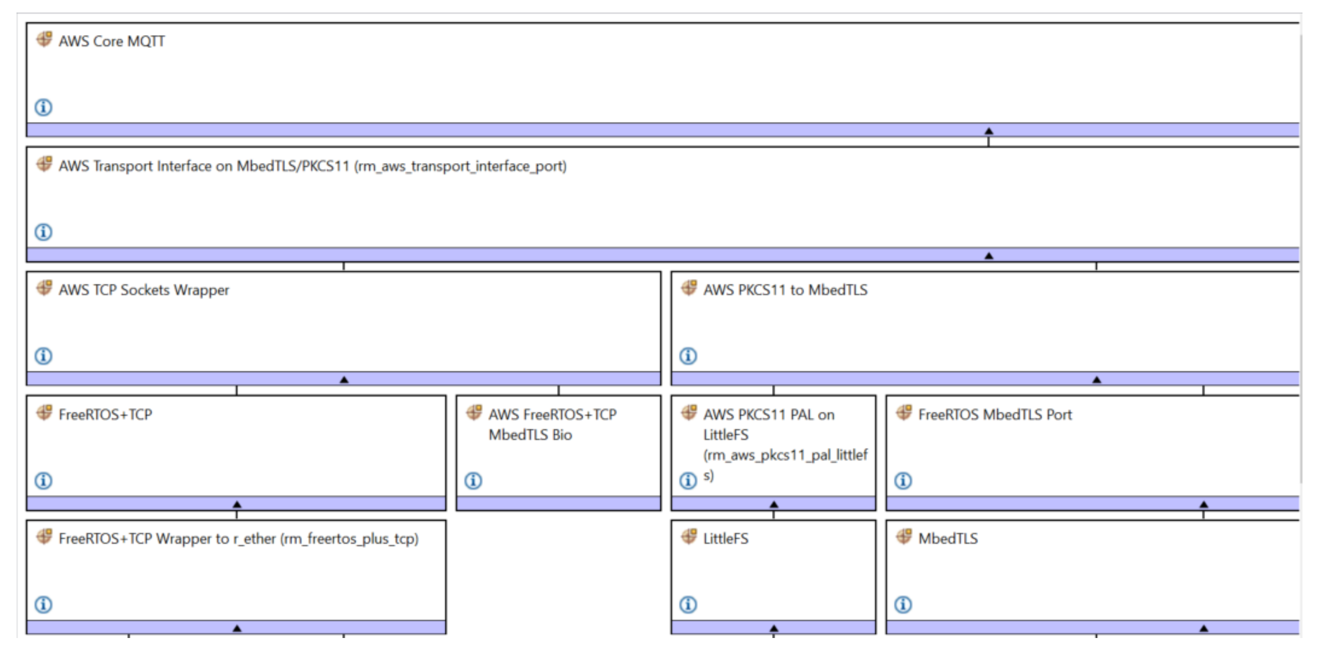

Figure 6. Expanded TCP Socket Interface Module

3.4 Mbed TLS

Mbed TLS is Arm®'s implementation of the TLS protocols and the cryptographic primitives required by those implementations. Mbed TLS is also solely used for its cryptographic features, even if the TLS/SSL portions are not used.

TLS Support uses FreeRTOS+Crypto, which eventually uses Mbed TLS. Use of Mbed TLS requires configuration and operation of the Mbed Crypto module, which in turn operates the Secure IP on the MCU (For RA6M5, it's SCE).

The following underlying mandatory changes are needed to the project using the TCP Sockets on FreeRTOS+Crypto module (Please refer to section 4.3 Creating the Application Project using the FSP Configurator for more detailed configuration):

1. Use FreeRTOS heap implementation scheme 4 (first fit algorithm with coalescence algorithm) or scheme 5 (first fit algorithm with coalescence algorithm with heap spanning over multiple non-adjacent/non-contiguous memory regions.
2. Enable support for dynamic memory allocation in FreeRTOS.
3. Enable Mbed TLS platform memory allocation layer.
4. Enable the Mbed TLS generic threading layer that handles default locks and mutexes for the user and abstracts the threading layer to use an alternate thread library.
5. Enable the Elliptic Curve Diffie-Hellman (ECDH) library.
6. Change FreeRTOS Total Heap Size to a value greater than 0x20000.
7. Add Persistent Storage on LittleFS.

Additional documentation on the Mbed Crypto hardware acceleration port is available in the *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > Security > Mbed Crypto H/W Acceleration (rm_psa_crypto).*

3.5 MQTT Agent Module APIs Usage

Table 2 lists APIs provided by AWS Core MQTT Agent (via AWS Core MQTT Stack) that are used as a part of the Application Example.

Table 2. MQTT Agent Module APIs

API	Description
MQTTAgent_Init	Perform any initialization the MQTT agent requires before it can be used. Must be called before any other function.
MQTTAgent_Connect	Add a command to call MQTT_Connect() for an MQTT connection. If a session is resumed with the broker, it will also resend the necessary QoS1/2 publishes.
MQTTAgent_Subscribe	Add a command to call MQTT_Subscribe() for an MQTT connection.
MQTTAgent_Publish	Add a command to call MQTT_Publish() for an MQTT connection.
MQTTAgent_Ping	Add a command to call MQTT_Ping() for an MQTT connection.
MQTTAgent_Unsubscribe	Add a command to call MQTT_Unsubscribe() for an MQTT connection.
MQTTAgent_Disconnect	Add a command to disconnect an MQTT connection.
MQTTAgent_CommandLoop	Process commands from the command queue in a loop.
MQTTAgent_ResumeSession	Resume a session by resending publishes if a session is present in the broker, or clear state information if not.
MQTTAgent_Terminate	Add a termination command to the command queue.
MQTTAgent_CancelAll	Cancel all enqueued commands and those awaiting acknowledgment while the command loop is not running.

3.6 AWS OTA PAL on MCUboot

AWS OTA PAL layer implementation for programming downloaded firmware images into memory provides the hardware port layer (MCUboot) for AWS IoT Over-the-air Update Library. It allows image signature verification with the sig-sha256-ecdsa method.

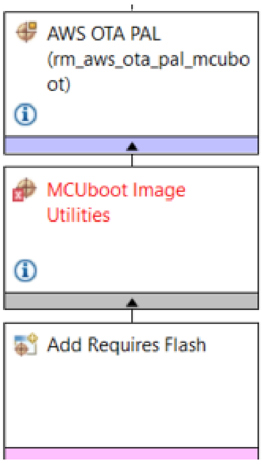

Figure 7. AWS OTA PAL stack view

Note: Currently, RA Flexible Software Package (FSP) v6.1.0 only supports OTA with the sig-sha256-ecdsa signature method.

The following underlying mandatory changes are needed for the project:

- Add Flash to AWS OTA PAL stack by clicking on **Add Requires Flash > New > Flash (r_flash_hp)**

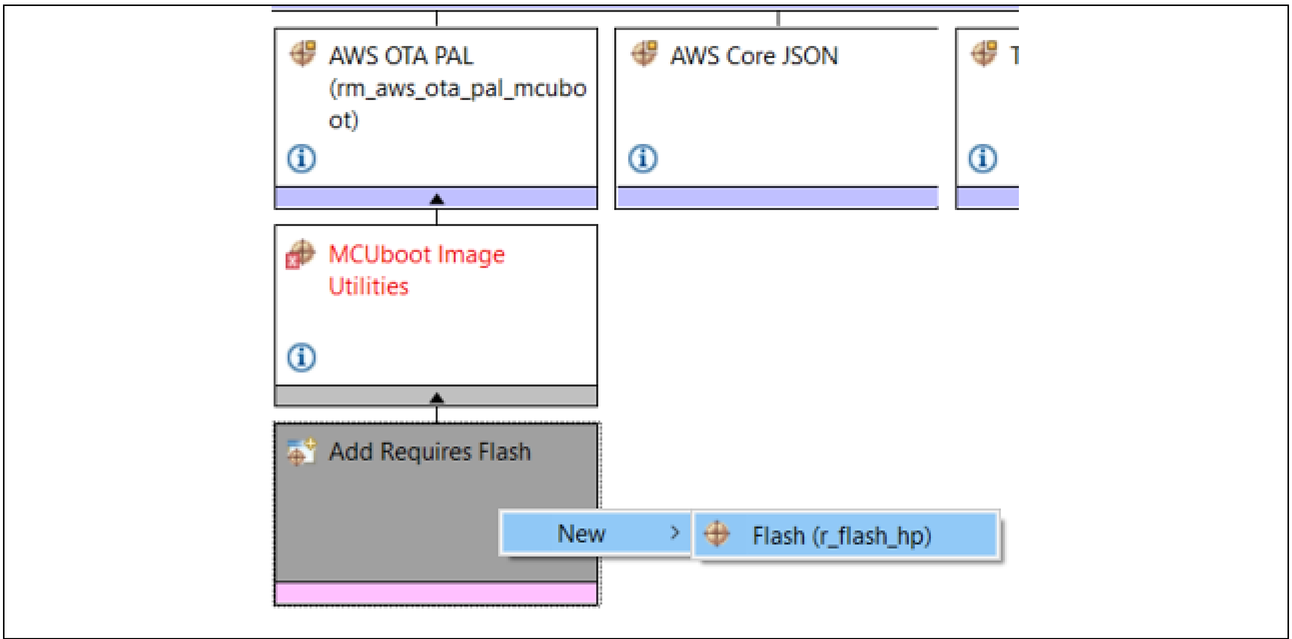

Figure 8. Add Flash to AWS OTA PAL

- **Flash > Property > Module Fash > Data Flash Background Operation** > **Disabled**
- **Flash > Property > Common > Code Flash Programming Enable > Enabled**
- **MCUboot Image Utilities > Property > Common > General:** Fill in the hyperlink paths leading to the MCUboot configuration files generated after building the bootloader project. Please refer to section **3.3 in *RA AWS Cloud Connectivity and Firmware Update OTA on CK-RA6M5 v2 with Ethernet - Getting Started Guide***, or section 4.3 Creating the Application Project using the FSP Configurator in this APN for more detailed configuration.

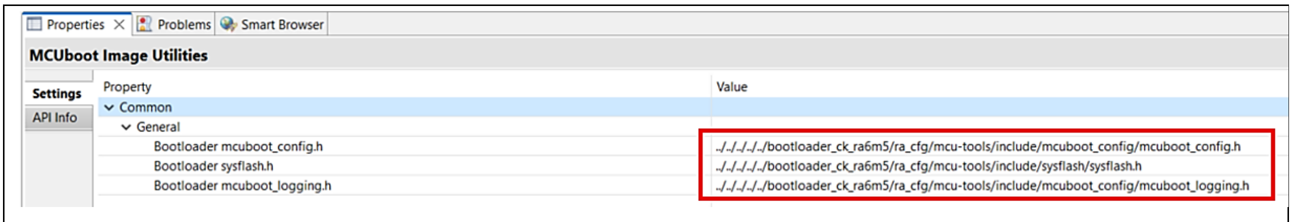

Figure 9. MCUboot Image Utilities Configuration

Note: Regarding the memory settings in the MCUboot section, the application example employs Upgrade Mode as swap mode, with code flash memory in linear mode. The size of MCUboot is 0x20000 Bytes, the size of the Cloud Application image used is 0x90000 Bytes, and the scratch area is 0x8000 Bytes. Users can customize these values depending on their source code. Below is the memory map of the OTA application example:

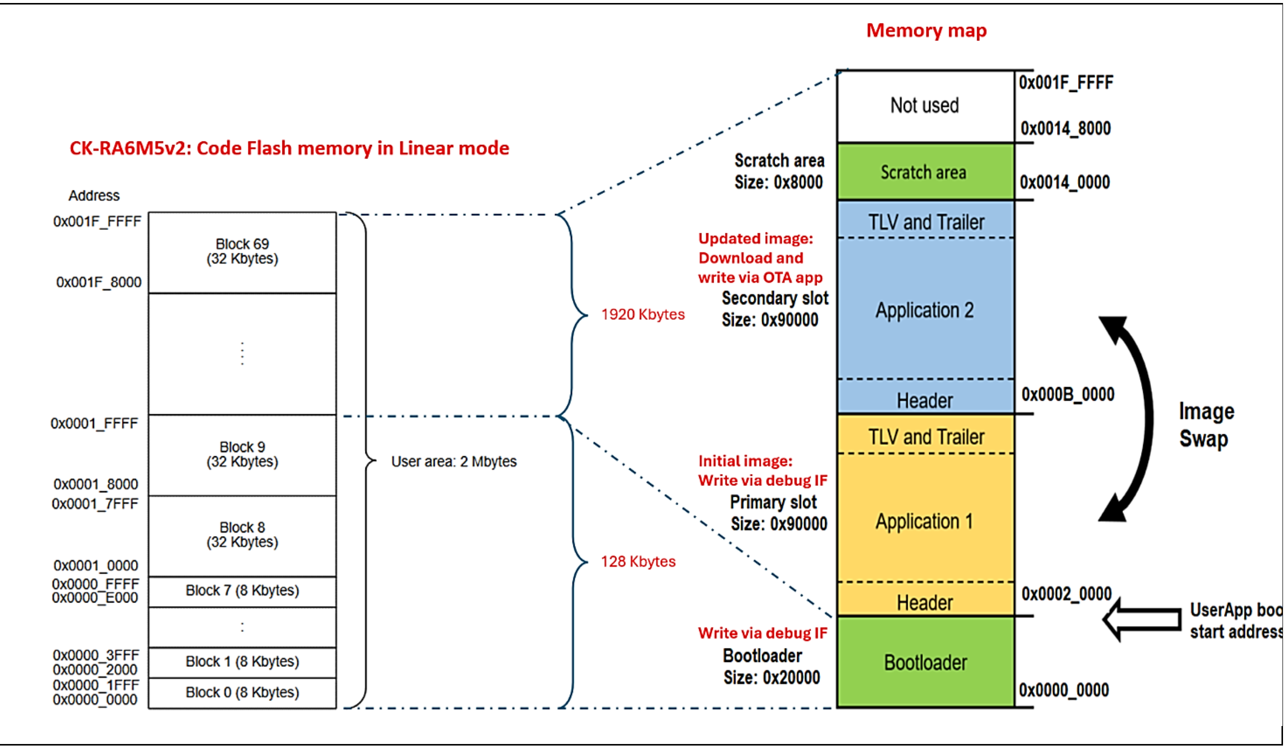

Figure 10. Memory map of the OTA application example

The user application image, once built, will be signed using AWS Code Signing (private key) with a Python tool (imgtool.py – It is included in FSP). The primary (initial) image will be loaded onto the board and verified first by the bootloader (MCUBoot). The secondary (new) image will be verified by the **AWS OTA PAL on the MCUBoot** after it has been downloaded from the cloud. If the verification is successful, the new image will be booted up. At this point, the bootloader will verify the new image once again. Please refer to sections **3, 4, and 5 in RA AWS Cloud Connectivity and Firmware Update OTA on CK-RA6M5 v2 with Ethernet - Getting Started Guide** to get more details about signature installation and OTA operations with AWS OTA PAL on MCUboot.

Additional documentation on the AWS OTA PAL is available in the *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > Networking > AWS OTA PAL on MCUboot.*

4. Cloud Connectivity Application Example

4.1 Overview

This application project showcases the utilization of APIs accessible via Renesas FSP-integrated modules for Amazon IoT SDK C, Mbed TLS module, Amazon FreeRTOS, and HAL Drivers on Renesas RA MCUs. Ethernet module is employed to establish network connectivity. Running on a Renesas Cloud Kit, the application also functions as a tutorial for Core MQTT, Mbed TLS/Crypto, OTA using Ethernet, and its configuration using the FSP configurator. It can serve as a foundation for developing customized cloud-based solutions with OTA using Renesas RA MCUs. Moreover, it offers a straightforward demonstration of firmware OTA service operation and setup provided by the cloud service provider.

The upcoming subsections show step-by-step instructions for creating device and security credentials policies as required by the AWS IOT on the cloud side to communicate with the end devices. The example accompanying this documentation demonstrates the Subscribe and Publish messaging between Core MQTT and MQTT Broker, on-demand publication of sensor data, and asynchronous publication of a “sensor data” event from the MCU to the Cloud. The device is also subscribed to receive actuation events (LED indication) from the Cloud.

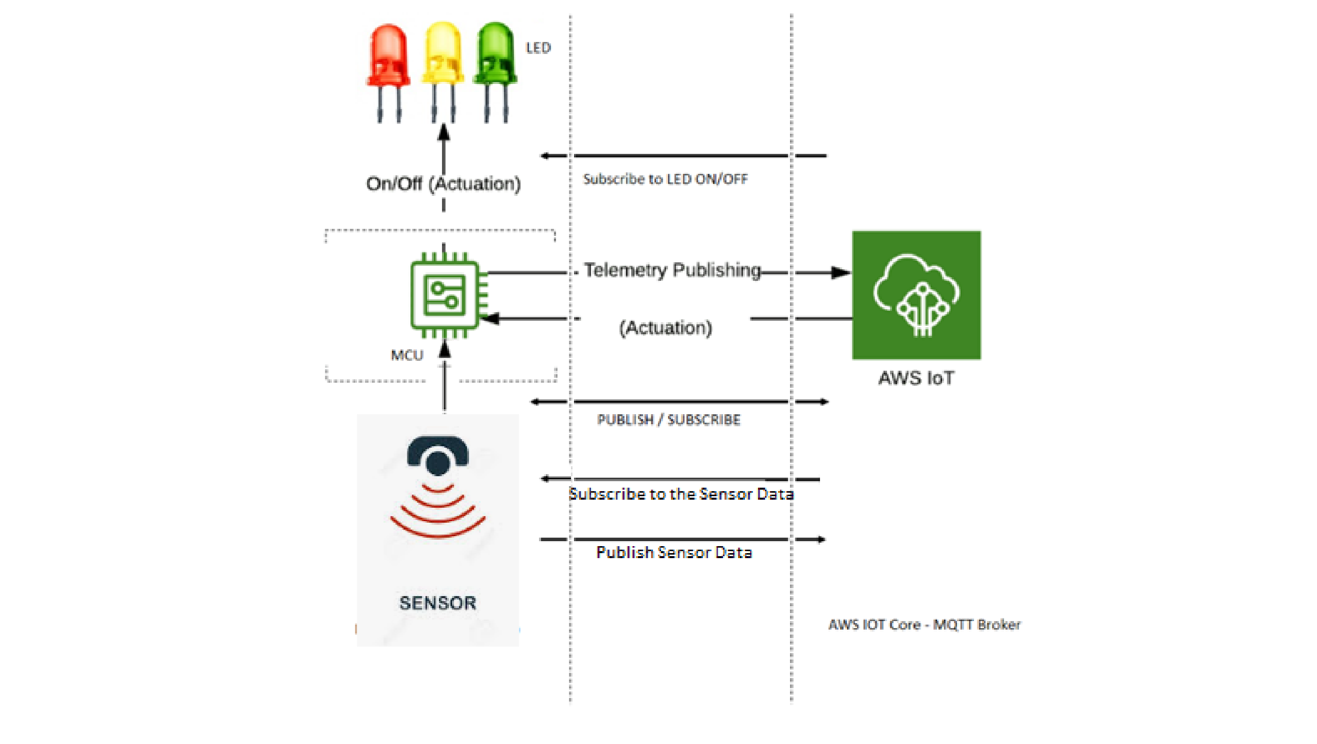

Figure 11. MQTT Publish/Subscribe to/from AWS IoT Core

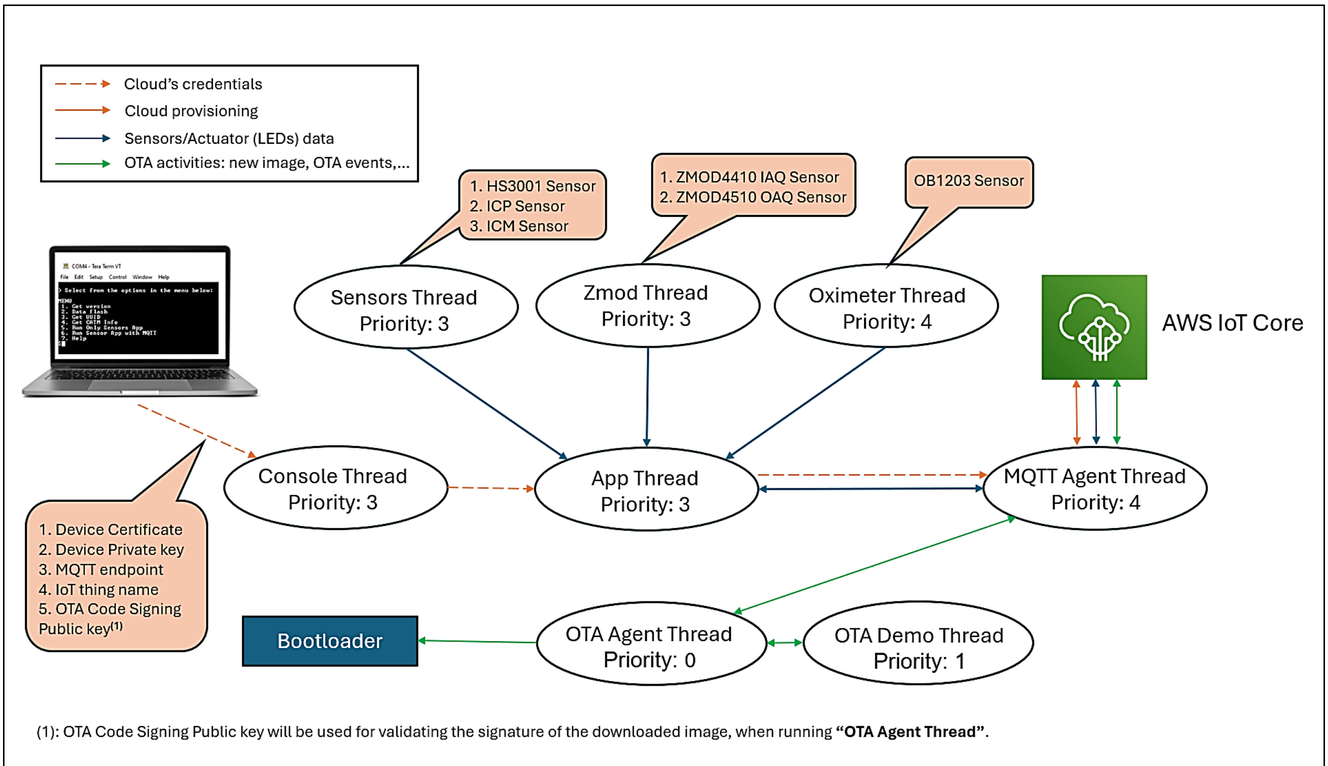

Figure 12. Thread architecture diagram

- Console Thread: To manage user interaction via CLI (Command Line Interface); display information regarding the board's UUID and the current firmware's FSP version, and save Cloud credentials to Data Flash memory.
- Sensors Thread: Collecting data from sensors using I2C channel 0 (HS3001, ICP, ICM).
- Zmod Thread: Collecting data from sensors using I2C channel 2 (ZMOD4410, ZMOD4510).
- Oximeter Thread: Collecting data from sensors using I2C channel 1 (OB1203).
- App Thread: Handles provisioning Cloud credentials from the CLI Thread. This thread aggregates sensor data from the "Sensors Thread," "ZMOD4xxx Thread", and "Oximeter Thread" to publish to

AWS Cloud. Additionally, it subscribes to the topic from IoT Core to send corresponding sensor data or control LEDs on board when message requests are received.

- MQTT Agent Thread: Manage the MQTT protocol operations such as connecting/disconnecting the MQTT broker, publishing, subscribing… and monitoring the MQTT connection status.
- OTA Demo Thread: Manage MQTT connection and OTA implementation status.
- OTA Agent Thread: Manage the OTA firmware update for the device, invoke the callback implementations to publish job-related control information, and receive chunks of pre-signed firmware images from the MQTT broker.
- Bootloader: responsible for initializing the device and loading the main application firmware after verifying the firmware’s signature.

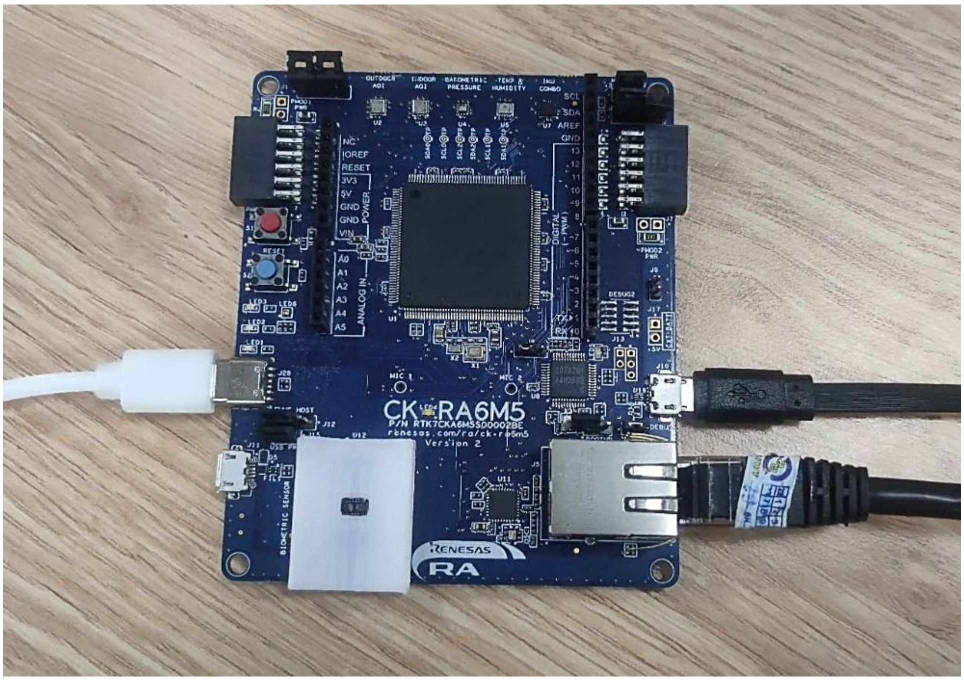

Figure 13. Hardware Setup

4.2 MQTT/TLS Application Software Overview

The following files from this application project serve as a reference, as shown in Table 3.

Table 3. Application Project Files

No.	Filename	Purpose
1.	`src/app_thread_entry.c`	Contains initialization code and has the main thread used in the Cloud Connectivity application.
2.	`src/common_init.c`	Contains code used to initialize common peripherals across the project.
3.	`src/common_init.h`	Contains macros, data structures, and function prototypes used to initialize common peripherals across the project.
4.	`src/common_utils.c`	Contains code commonly used across the project.
5.	`src/common_utils.h`	Contains macros, data structures, and function prototypes commonly used across the project.
6.	`src/console_thread_entry.c`	Contains the code for the command-line interface and flash memory operations.

No.	Filename	Purpose
7.	`src/icm.h`	Contains user-defined data types and function prototypes which have implementation in `RA_ICM42605.c`
8.	`src/ICM42605.c`	Contains driver codes for the 6 Axis sensor (Gyroscope, Accelerometer)
9.	`src/ICM42605.h`	Contains the Data structure function prototypes for the 6 Axis sensor (Gyroscope, Accelerometer)
10.	`src/ICP_20100.c`	Contains the driver codes for Barometric Pressure and Temperature Sensor.
11.	`src/ ICP_20100.h`	Contains the Data structure and function prototypes for Barometric Pressure and Temperature Sensor
12.	`src/icp.h`	Contains user-defined data types and function prototypes which have an implementation in `RA_ICP20100.c`
13.	`src/oximeter_thread_entry.c`	Contains codes for the oximeter sensor thread's operation.
14.	`src/oximeter.c`	Contains codes for oximeter sensor's initialization and measurement.
15.	`src/oximeter.h`	Contains the data structure and function prototypes for the oximeter sensor.
16.	`src/r_typedefs.h`	Contains the common derived data types
17.	`src/RA_HS3001.c`	Contains the code and function for Renesas Relative Humidity and Temperature Sensor.
18.	`src/RA_HS3001.h`	Contains the common data structure's function prototypes for the Renesas Relative Humidity and Temperature sensors.
19.	`src/RA_ICM42605.c`	Contains codes for 6 Axis sensor (Gyroscope, Accelerometer) sensor's initialization and measurement.
20.	`src/RA_ICP20100.c`	Contains codes for Barometric Pressure and Temperature sensor's initialization and measurement.
21.	`src/RA_ZMOD4XXX_Common.c`	Contains the common code for the Renesas ZMOD sensors
22.	`src/RA_ZMOD4XXX_Common.h`	Contains the common data structure's function prototypes for the Renesas ZMOD sensors
23.	`src/RA_ZMOD4XXX_IAQ1stGen.c`	Contains the common code for the Renesas ZMOD Internal Air Quality sensors
24.	`src/RA_ZMOD4XXX_OAQ_NO2_O3.c`	Contains the common code for the Renesas ZMOD Outer Air Quality sensors
25.	`src/RmcI2C.c`	Contains the I2C wrapper functions for the third-party sensors not integrated with FSP
26.	`src/RmcI2C.h`	Contains the I2C function prototypes for wrapper functions for the third-party sensors not integrated with FSP
27.	`src/sensors_thread_entry.c`	Contains the Code to access the Sensor data from the different sensors and order topic to publish.
28.	`src/user_choice.c`	Contains the code for user's choice of sensors and user configurations

No.	Filename	Purpose
29.	src/user_choice.h	Contains the Function prototypes for the Sensor and its user configuration for the different sensors and their data accessibility.
30.	src/usr_config.h	To customize the user configuration to run the application.
31.	src/usr_data.h	Accompanying header file for the application thread.
32.	src/usr_hal.c	Contains data structures and functions used for the Hardware Abstraction Layer (HAL) initialization and associated utilities.
33.	src/usr_hal.h	Accompanying header for exposing functionality provided by usr_hal.c.
34.	src/usr_network.c	Contains data structures and functions used in FreeRTOS TCP/IP and Ethernet Module.
35.	src/usr_network.h	Contains declarations of data structures and functions used in usr_network.c
36.	src/zmod_thread_entry.c	Contains the code for indoor air and outdoor air quality sensors
37.	src/SEGGER_RTT/SEGGER_RTT.c	Implementation of SEGGER real-time transfer (RTT), which allows real-time communication on targets that support debugger memory accesses while the CPU is running.
38.	src/SEGGER_RTT/SEGGER_RTT.h	
39.	src/SEGGER_RTT/SEGGER_RTT_Conf.h	
40.	src/SEGGER_RTT/SEGGER_RTT_printf.c	
41.	src/backoffAlgorithm/backoff_algorithm.c	Retry algorithms with random backoff for the next retry attempt
42.	src/backoffAlgorithm/backoff_algorithm.h	Retry algorithms with random backoff for the next retry attempt header file
43.	src/console_menu/console.c	Contains data structures and functions used to print data on the console using the UART
44.	src/console_menu/console.h	Contains the Function prototypes used to print data on the console using UART
45.	src/console_menu/menu_flash.c	Contains data structures and functions used to provide a CLI flash memory-related menu
46.	src/console_menu/menu_flash.h	Contains the Function prototypes and macros used to provide the CLI flash memory-related menu
47.	src/console_menu/menu_kis.c	Contains functions to get the FSP version, get UUID, and a help option for the main menu on CLI
48.	src/console_menu/menu_kis.h	Contains the function prototypes and macros used to get the FSP version, get uuid and the help option for the main menu on the CLI
49.	src/console_menu/menu_main.c	Contains data structures and functions used to provide CLI main menu options
50.	src/console_menu/menu_main.h	Contains the Function prototypes and macros used to provide CLI main menu options
51.	Src/flash/flash_hp.c	Contains data structures and functions used to perform flash memory-related operations
52.	src/flash/flash_hp.h	Contains the Function prototypes and macros used to perform flash memory-related operations
53.	src/ob1203_bio/KALMAN/kalman.c	
54.	src/ob1203_bio/KALMAN/kalman.h	
55.	src/ob1203_bio/SAVGOL/SAVGOL.c	

No.	Filename	Purpose
56.	src/ob1203_bio/SAVGOL/SAVGOL.h	Contains an algorithm for Heart Rate, Blood Oxygen Concentration, Pulse Oximetry, Proximity, Light, and Color Sensor sample calculations
57.	src/ob1203_bio/SPO2/SPO2.c	
58.	src/ob1203_bio/SPO2/SPO2.h	
59.	src/ob1203_bio/ob1203_bio.c	Contain codes for the OB1203 sensor's implementation to use with FSP stacks.
60.	src/ob1203_bio/ob1203_bio.h	Contain user data structure and function prototypes used in ob1203_bio.c
61.	src/ob1203_bio/OB1203_Config.c	Contain configuration used for OB1203 driver.
62.	src/ob1203_bio/OB1203_Config.h	Contains the Function prototypes and macros used in OB1203_Config.c
63.	src/ob1203_bio/OB_driver/rm_ob1203/ppg_mode/rm_ob1203_ppg_mode.c	Contain codes for the OB1203 driver.
64.	src/ob1203_bio/OB_driver/rm_ob1203/proximity_mode/rm_ob1203_proximity_mode.c	
65.	src/ob1203_bio/OB_driver/rm_ob1203/rm_ob1203_ra_driver.c	
66.	src/ob1203_bio/OB_driver/rm_ob1203/rm_ob1203.c	
67.	src/ob1203_bio/OB_driver/rm_ob1203_api.h	
68.	src/ob1203_bio/OB_driver/rm_ob1203_cfg.h	
69.	src/ob1203_bio/OB_driver/rm_ob1203.h	
70.	src/hal_entry.c	Contains HAL-level functions used in the application
71.	src/hal_warmstart.c	
72.	src/OtaOverMqttDemoExample.c	Contain AWS OTA Demo code
73.	src/agent/demo_config.h	Defines AWS OTA demo common configuration options
74.	src/agent/mqtt_agent/mqtt_agent_task.c	This file contains the initial task created after the TCP/IP stack connects to the network
75.	src/agent/mqtt_agent/mqtt_agent_task.h	Contains declarations of the structures and functions used in mqtt_agent_task.c

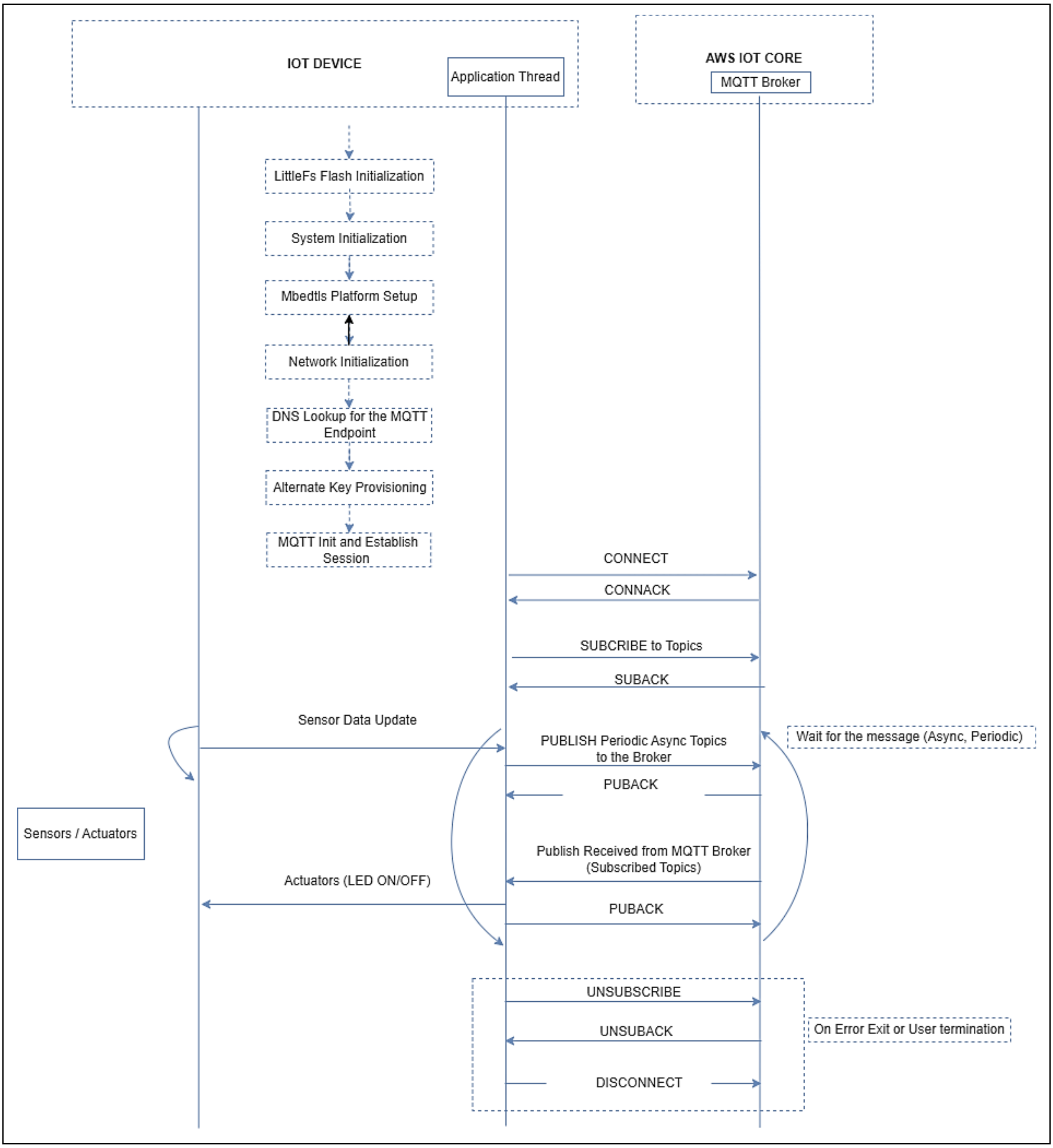

Figure 14. Application Example Implementation Details

4.3 Creating the Application Project using the FSP Configurator

Complete the steps to create the project from the start using the e^2 studio and FSP configurator. Table 4 shows the step-by-step process of creating the project. It is assumed that the user is familiar with the e^2 studio and FSP configurator. Launch the installed e^2 studio for the FSP.

Table 4. Step-by-step Details for Creating the Application Project for Ethernet with OTA feature

No	Steps	Intermediate Steps
Set up the Bootloader project		
1	**Project Creation:**	**File → New → Renesas C/C++ Project → Renesas RA**

No	Steps	Intermediate Steps
2	**Project Template:**	**Templates for Renesas RA Project → Renesas RA C/C++ Project → Next**
3	**e² studio - Project Configuration (RA C/C++ Project) →**	Project Name: **bootloader_ck_ra6m5 → Next**
4	**Device Selection →**	FSP Version: **6.1.0**
		Board: **CK-RA6M5 V2**
		Device: **R7FA6M5BH3CFC**
		Language: **C**
5	**Select Tools**	Toolchain: **GNU ARM Embedded**
		Toolchain version: **13.2.1.arm-13-7**
		Debugger: **J-Link ARM → Next**
5a	**Project Type Selection**	Flat (Non-TrustZone) Project **→ Next**
5b	**Preceding Project or Smart Bundle Selection**	None **→ Next**
6	**Build Artifact and RTOS Selection**	Artifact Selection: **Executable**
		RTOS Selection: **No RTOS → Next**
6a	**Project Template Selection**	**Project Template Selection: Bare Metal – Minimal → Finish**
7	**Clock**	**HOCO 20MHz →PLL Src: HOCO → PLL Div/2 →PLL Mul x20.0 → PLL 200MHz**
8	Update the pin configuration file	
	Pins tab	
	Select Pin Configuration: RA6M5_2 CK	→**Uncheck the box**: Generate data
	Select Pin Configuration: R7FA6M5BH3CFC.pincfg	→**Check the box:** ✓ Generate data
		Generate data: **g_bsp_pin_cfg**
	Note: Bootloader does not use the extra peripheral or GPIO pins configured in the RA6M5_2 CK configuration. This also reduces some memory usage for the bootloader project.	
9	Add MCUboot stack	
	Stacks tab → New Stack →	**Bootloader → MCUboot**
9a	**Under MCUboot Port for RA → Add Requires a crypto stack →**	**New → MbedTLS (Crypto Only)**
	Under MCUboot Port for RA → Add Requires Flash →	**New → Flash (r_flash_hp)**
9b	Flash configuration	
	Property → Common →	Code Flash Programming Enable: **Enabled**
		Data Flash Programming Enable: **Disabled**
	Property → Module Flash →	Name: **g_flash0**
		Data Flash Background Operation: **Disabled**
		Callback: **NULL**
		Flash Ready Interrupt Priority: **Disabled**
		Flash Error Interrupt Priority: **Disabled**
10	**Modifying the BSP Settings – RA Common for (Main stack, Heap, and Sub clock Settings); RA6M5 Family for Dual bank mode settings**	
	BSP Tab → Settings → RA Common	Main stack size (bytes): **0x1000**
		Heap size (bytes): **0x400**
		Subclock Populated: **Not Populated**
10a	**BSP Tab → Settings → RA6M5 Device Options → OFS Registers →**	DUALSEL (Dual Mode Select Register) Settings: **Enabled**
		Bank Mode: **Linear**
11	Some dependencies related to TLS Support need to be resolved to remove the error in the FSP configurator by modifying the **MbedTLS (Crypto Only)** property settings.	
	Common → General →	MBEDTLS_THREADING_ALT: **Undefine**

No	Steps	Intermediate Steps
	Common → General →	MBEDTLS_THREADING_C: **Undefine**
	Common → General →	MBEDTLS_MEMORY_BUFFER_ALLOC_C**: Define**
	Common → Public Key Cryptography (PKC) → RSA →	MBEDTLS_RSA_C: **Undefine**
12	MCUboot configuration	
	Common → General →	Upgrade Mode: **Swap**
	Common → General →	Validate Primary Image: **Enabled**
	Common → Signing and Encryption Options →	Signature Type: **ECDSA P-256** (RA FSP (v6.1.0) currently only supports OTA with the sig-sha256-ecdsa signature method)
	Common → Signing and Encryption Options →	Custom: **--pad**
	Additional documentation on the ***MCUboot Port*** is available in the ***FSP User's Manual*** under ***RA Flexible Software Package Documentation > API Reference > Modules > Bootloader > MCUboot Port***	
13	Add the user application files to folder `bootloader_ck_ra6m5/src`: file `hal_entry.c`, file `key.c`	
14	Follow the section "*Setting Up the Device*" in the (*RA AWS Cloud Connectivity and Firmware Update OTA on CK-RA6M5 v2 with Ethernet - Getting Started Guide)* document to generate key pairs, a certificate, and add `secp256r1.privatekey` to the Bootloader project.	
15	Build project `bootloader_ck_ra6m5` to generate file `bootloader_ck_ra6m5.sbd` in folder `bootloader_ck_ra6m5/Debug`	
Set up the Application and Downloader project		
16	**Project Creation:**	**File → New → C/C++ Project**
17	**Project Template:**	**Templates for New C/C++ Project → Renesas RA C/C++ Project → Next**
18	**e² studio - Project Configuration (RA C/C++ Project) →**	**Project Name (Name for the Project)** Note: Input your desired name for the project → **Next**
19	**Device Selection →**	FSP Version: **6.1.0**
		Board: **CK-RA6M5 V2**
		Device: **R7FA6M5BH3CFC**
		Language: C
20	**Toolchains**	Toolchain: **GNU ARM Embedded**
		Toolchain version: **13.2.1.arm-13-7**
		Debugger: **J-Link ARM → Next**
20a	**Project Type Selection**	Flat (Non-TrustZone) Project **→ Next**
20b	**Preceding Project or Smart Bundle Selection**	Preceding Project: **bootloader_ck_ra6m5** **→ Next**
21	**Build Artifact and RTOS Selection**	Artifact Selection: **Executable**
		RTOS Selection: FreeRTOS(v11.1.0+fsp6.1.0) → **Next**
21a	**Project Template Selection**	**Project Template Selection: FreeRTOS – Minimal – Static Allocation → Finish**
22	**Modifying the BSP Settings – RA Common for (Main stack, Heap, and Subclock Settings)**	
	BSP Tab → Property Settings for RA Common	Main stack size (bytes): **0x2000**
		Heap size (bytes): **0x20000**
		Subclock Populated: **Not Populated**
23	**Clocks**	**HOCO 20MHz →PLL Src: HOCO → PLL Div/2 →PLL Mul x20.0 → PLL 200MHz**
24	Add the Heap Implementation in **HAL/Common**	
	Stacks tab → New Stack →	**RTOS → FreeRTOS Heap 4**

No	Steps	Intermediate Steps
25	Adding the HAL Modules as required for the Application Project: Clock Generation Circuit and GPT Timer1 for control publishing sensor value into MQTT	
	HAL/Common Stacks → New Stack	**→ System → Clock Generation Circuit (r_cgc)**
	Module g_cgc0 Clock Generation Circuit (r_cgc)	Name: **g_cgc0**
	HAL/Common Stacks → New Stack	**→ Timers → Timer, General PWM (r_gpt)**
	Module g_timer0 Timer, General PWM (r_gpt) → General	Name: **g_timer1**
		Channel: **1**
		Mode: **Periodic**
		Period: **1**
		Period Unit: **Seconds**
	Module g_timer0 Timer, General PWM (r_gpt) → Interrupts	Callback: **g_user_timer_cb**
		Overflow/Crest Interrupt Priority: **Priority 5**
25a	Configure Pins for CGC	
	Pins Tab → Pin Selection → Peripherals → System: CGC → CGC0 →	Operation Mode: **Main+Sub Osc**
26	Create and configure for App Thread	
	Stacks Tab→	**Threads → New Thread**
	Config Thread Properties→	Symbol: **app_thread**
		Name: **App Thread**
		Stack size (bytes): **0x12000**
		Priority: **3**
		Thread Context: **NULL**
		Memory Allocation: **Static**
26a	Generic RTOS configs under thread (Additional configuration on top of the Default Config provided by FSP)	
	Common → General	Use Mutexes: **Enabled**
		Use Recursive Mutexes: **Enabled**
	Thread → Memory Allocation	Support Dynamic Allocation: **Enabled**
		Total Heap Size: **0x20000**
27	Adding the AWS MQTT Wrapper Module to the Application Thread	
	Note: Now the newly created thread (Application thread) is ready to add a new stack (Here, the AWS IoT Over-the-air Update Library is added)	
	New Stack →	**Networking → AWS IoT Over-the-air Update Library**
	AWS IoT Over-the-air Update Library	Common → Log2 File Block Size: **12**
27a	Under the **AWS Transport Interface on MbedTLS/PKCS11 → Add Sockets Wrapper [Required]**, add	**New → AWS TCP Sockets Wrapper**
27b	Under the **SCE Compatibility Mode → Add Key Injection for PSA Crypto (Optional)**, add	**New → Key Injection for PSA Crypto**
27c	**AWS Core MQTT →**	**Common → Retry count for reading CONNACK from network → 10**
28	Adding persistent storage support for AWS PKCS11	
	Under **the MbedTLS (Crypto only) → Add Persistent Storage on LittleFS (Optional) →**	**Use → LittleFS**
28a	Under **AWS PKCS11 PAL on LittleFS → LittleFS → Add LittleFS Port [Required]**, add	**New → LittleFS on Flash**

No	Steps	Intermediate Steps
28b	**LittleFS on Flash → Module LittleFS on Flash (rm_littlefs_flash) →**	**Block Count → (BSP_DATA_FLASH_SIZE_BYTES/256)**
29	Some dependencies related to TLS Support need to be resolved to remove the error in the FSP configurator by modifying the **MbedTLS (Crypto Only)** property settings.	
	Common → Platform →	MBEDTLS_PLATFORM_MEMORY: **Define**
	Common → General →	MBEDTLS_THREADING_C: **Define**
	Common → General →	MBEDTLS_THREADING_ALT: **Define**
	Common → Public Key Cryptography (PKC) →	ECC → MBEDTLS_ECDH_C: **Define**
	Common →Hardware Acceleration → Public Key Cryptography (PKC)	RSA 3072 → Verification: **Enabled**
	Common →Hardware Acceleration → Public Key Cryptography (PKC)	RSA 4096 → Verification: **Enabled**
	Common → Storage →	MBEDTLS_FS_IO: **Define**
	Common → Storage →	MBEDTLS_PSA_CRYPTO_STORAGE_C: **Define**
	Common → Storage →	MBEDTLS_PSA_ITS_FILE_C: **Define**
	Common → Message Authentication Code (MAC)→	MBEDTLS_CMAC_C: **Define**
30	**MCUboot Image Utilities** Configuration	
	Under AWS OTA PAL (rm_aws_ota_pal_mcuboot) →MCUboot Image Utilities → Property → Common → General →	Bootloader mcuboot_config.h: **../../../../../bootloader_ck_ra6m5/ra_cfg/mcu-tools/include/mcuboot_config/mcuboot_config.h**
		Bootloader sysflash.h: **../../../../../bootloader_ck_ra6m5/ra_cfg/mcu-tools/include/sysflash/sysflash.h**
		Bootloader mcuboot_logging.h: **../../../../../bootloader_ck_ra6m5/ra_cfg/mcu-tools/include/mcuboot_config/mcuboot_logging.h**
	In case of using another bootloader folder name, please update the above configurations accordingly.	
30a	**Under MCUboot Image Utilities → Add Requires Flash →**	**New → Flash (r_flash_hp)**
31	**Under MCUboot Image Utilities → Flash →**	
	Property → Module Fash (r_flash_hp) →	Name: **g_flash1**
		Data Flash Background Operation: **Disabled**
	Property → Common →	Code Flash Programming Enable: **Enabled**
		Data Flash Programming Enable: **Enabled**

No	Steps	Intermediate Steps
32	**FreeRTOS + TCP Configuration** Note: This is only applicable to the Ethernet application project. Most of the default settings remain the same, except a few of the default configurations need to be changed.	
	Common →	DNS Request Attempts: **20**
		Stack size in words (not bytes): **configMINIMAL_STACK_SIZE * 16**
		DHCP callback function: **Enable**
		Set the maximum number of events: **ipconfigNUM_NETWORK_BUFFER_DESCRIPTORS + 16**
		Size of Rx buffer for TCP sockets: **8192**
		Size of Tx buffer for TCP sockets: **8192**
	Under **FreeRTOS+TCP Wrapper to ethernet driver (rm_freertos_plus_tcp) → Add Ethernet Driver,** add	**New → Ethernet (r_ther)**
	Properties setting for **g_ether0 Ethernet → Module g_ether0 Ethernet (r_ether) → General →**	Name: **g_ether0** MAC address: **<User needs to define the valid values for their network>**
	Properties setting for **g_ether0 Ethernet → Module g_ether0 Ethernet (r_ether) → Buffers →**	Number of TX buffers: **8**
		Number of RX buffers: **8**
	Properties setting for g_ether0 Ethernet → Module g_ether0 Ethernet (r_ether) → Interrupts →	Interrupt priority: **Priority 5**
	Properties setting for **g_ether_phy0 Ethernet (r_ether_phy) → Common**	ICS1894 target: **Enabled**
		Reference clock: **Enabled**
	g_ether_phy0 Ethernet (r_ether_phy) → Module g_ether_phy0 Ethernet	PHY-LSI Address: **5**
	g_ether_phy0 Ethernet (r_ether_phy) → Pins	ET0_LINKSTA: **None**
		ET0_WOL: **None**
	Properties setting for **g_ether_phy_lsi0 Ethernet PHY-LSI**	PHY-LSI Address: **5**
33	Adding FreeRTOS Objects for the Application and Sensors	
33a	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_topic_queue**
		Item Size (Bytes): **65**
		Queue Length (Items): **16**
		Memory Allocation: **Static**
33b	**Stacks Tab → Objects →**	**New Object → Mutex**
	Property Settings for the Mutex	Symbol: **g_console_out_mutex**
		Type: **Mutex**
		Memory Allocation: **Static**
33c	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_hs3001_queue**
		Item Size (Bytes): **8**
		Queue Length (Items): **1**
		Memory Allocation: **Static**
33d	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_iaq_queue**
		Item Size (Bytes): **12**
		Queue Length (Items): **1**

No	Steps	Intermediate Steps
		Memory Allocation: **Static**
33e	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_oaq_queue**
		Item Size (Bytes): **4**
		Queue Length (Items): **1**
		Memory Allocation: **Static**
33f	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_icm_queue**
		Item Size (Bytes): **72**
		Queue Length (Items): **1**
		Memory Allocation: **Static**
33g	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_icp_queue**
		Item Size (Bytes): **16**
		Queue Length (Items): **1**
		Memory Allocation: **Static**
33h	**Stacks Tab → Objects →**	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_ob1203_queue**
		Item Size (Bytes): **10**
		Queue Length (Items): **1**
		Memory Allocation: **Static**
34	Add Console Thread	
	Stacks Tab → Threads	**New Thread**
	Config **Thread Properties →**	Symbol: **console_thread**
		Name: **Console Thread**
		Stack size (bytes): **4096**
		Priority: **3**
		Thread Context: **NULL**
		Memory Allocation: **Static**
34a	Add the UART module to Console Thread	
	New Stack →	**Connectivity → UART**
	Common →	FIFO Support: **Enable**
		DTC Support: **Enable**
		Flow Control Support: **Enable**
	Module UART → General →	Name: **g_console_uart**
		Channel: **5**
		Data Bits: **8bits**
		Parity: **None**
		Stop Bits: **1bit**
	Module UART → Baud→	Baud rate: **115200**
	Module UART → Interrupts →	Callback: **user_uart_callback**
	Pins →	TXD5: **P501**
		RXD5: **P502**
		CTS5: **P500**
		CTSRTS5: **P508**
34b	Add Flash module to Console Thread	
	New Stack →	**Storage → Flash**
	Module Flash →	Name: **user_flash**
		Data Flash Background Operation: **Disabled**
		Callback: **flash_callback**

No	Steps	Intermediate Steps
		Flash Ready Interrupt Priority: **Priority 2**
		Flash Error Interrupt Priority: **Priority 2**
35	Add Sensors Thread, this thread is used to access the sensor's values of HS3001, ICP-20100, and ICM-42605; and prepare topics to publish messages using timer1 and g_topic_queue.	
	Stacks Tab → Threads	**New Thread**
	Config **Thread Properties →**	Symbol: **sensors_thread**
		Name: **Sensors Thread**
		Stack size (bytes): **8192**
		Priority: **3**
		Thread Context: **NULL**
		Memory Allocation: **Static**
35a	Adding the HS300X Temperature/Humidity Sensor Module to the Sensors Thread	
	New Stack →	**Sensor → HS300X Temperature/Humidity Sensor**
	Config **HS300X Temperature/Humidity sensor→**	Name: **g_hs300x_sensor0**
		Callback: **hs300x_callback**
	Config **g_comms_i2c_bus0 I2C Shared Bus**	Name: **g_comms_i2c_bus0**
		Channel: **0**
		Rate: **Fast-mode**
	Under **I2C Shared Bus → Add I2C Communications Peripheral →**	**New → I2C Master(r_iic_master)**
	Config for I2C Master →	Name: **g_i2c_master0**
		Interrupt Priority Level: **Priority 5**
35b	Adding ICP-20100 and ICM-42605 sensors to the Sensors Thread. Note: FSP doesn't provide an integrated module for ICP-20100 and ICM-42605 sensors. This needs to be integrated via the I2C communication device and external IRQ manually. Also, the related sensor driver code needs to be added to the src folder.	
	New Stack →	**Connectivity → I2C Communication Device**
	Config I2C Communication Device →	Name: **g_comms_i2c_device4**
		Slave Address: **0x63**
		Callback: **ICP_comms_i2c_callback**
	Under the **I2C Communication Device → Add I2C Shared Bus →**	**Use → g_comms_i2c_bus0 I2C Shared Bus**
	New Stack →	**Input → External IRQ**
	Config for External IRQ	Name: **g_external_irq6**
		Channel: **6**
		Trigger: **Falling**
		Callback: **ICP_IRQ_CALLBACK**
35c	Adding I2C Communication Device and External IRQ for ICM-42605 into Sensors Thread	
	New Stack →	**Connectivity → I2C Communication Device**
	Config I2C Communication Device →	Name: **g_comms_i2c_device5**
		Slave Address: **0x68**
		Callback: **ICM_comms_i2c_callback**
	Under the **I2C Communication Device → Add I2C Shared Bus →**	**Use → g_comms_i2c_bus0 I2C Shared Bus**
	New Stack →	**Input → External IRQ**
	Config for External IRQ	Name: **g_external_irq3**
		Channel: **3**
		Trigger: **Falling**
		Callback: **ICM_42605_Callback2**
	New Stack →	**Input → External IRQ**
	Config for External IRQ	Name: **g_external_irq12**

No	Steps	Intermediate Steps
		Channel: **12**
		Trigger: **Falling**
		Callback: **ICM_42605_Callback1**
36	Add Oximeter Thread for OB1203 sensor's handling.	
	Stacks Tab → Threads	**New Thread**
	Config **Thread Properties →**	Symbol: **oximeter_thread**
		Name: **Oximeter Thread**
		Stack size (bytes): **2048**
		Priority: **4**
		Thread Context: **NULL**
		Memory Allocation: **Static**
36a	Adding I2C Communication Device and External IRQ for OB1203 sensor module into Oximeter Thread. Note: Beginning with FSP v6.0.0, FSP doesn't provide an integrated module for the OB1203 sensor. This needs to be integrated via the I2C communication device and external IRQ manually. Also, the sensor driver code related to it needs to be added to the src folder.	
	New Stack →	**Connectivity→ I2C Communication Device**
	Config **I2C Comm Device →**	Name: **g_comms_i2c_device3**
		Slave Address:**0x53**
		Callback: **rm_ob1203_comms_i2c_callback**
	Under the **I2C Communication Device → Add I2C Shared Bus →**	New **→ I2C Shared Bus**
	Config **I2C Shared Bus →**	Name: **g_comms_i2c_bus1**
		Channel: **1**
		Rate: **Standard**
	Under **I2C Shared Bus → Add I2C Communications Peripheral →**	**New → I2C Master (r_iic_master)**
	Config **I2C Master →**	Name: **g_i2c_master1**
		Interrupt Priority Level: **Priority 12**
	New Stack →	**Input → External IRQ**
	Config for **External IRQ →**	Name: **g_external_irq14**
		Channel: **14**
		Trigger: **Falling**
		Callback: **rm_ob1203_irq_callback**
		Pins: **IRQ14: P403**
	New Stack →	**Connectivity→ I2C Communication Device**
	Config **I2C Comm Device →**	Name: **g_comms_i2c_device6**
		Slave Address: **0x53**
		Callback: **rm_ob1203_comms_i2c_callback**
	Under the **I2C Communication Device → Add I2C Shared Bus →**	**Use → g_comms_i2c_bus1 I2C Shared Bus**
37	Add Zmod Thread for ZMOD4410 IAQ and ZMOD4510 OAQ sensors' handling.	
	Stacks Tab → Threads	**New Thread**
	Config **Thread Properties →**	Symbol: **zmod_thread**
		Name: **Zmod Thread**
		Stack size (bytes): **1024**
		Priority: **3**
		Thread Context: **NULL**
		Memory Allocation: **Static**
37a	Adding the ZMOD4XXX Gas Sensor module (ZMOD4410 IAQ) to the Zmod Thread.	
	New Stack →	**Sensor → ZMOD4XXX Gas Sensor**

No	Steps	Intermediate Steps
	Config ZMOD4XXX Gas Sensor→	Name: **g_zmod4xxx_sensor0**
		Comms I2C Callback: **zmod4xxx_comms_i2c_callback**
		IRQ Callback: **zmod4xxx_irq0_callback**
	Under the ZMOD4XXX Gas Sensor → Add Requires ZMOD Library →	**New → ZMOD4410 IAQ 1st Generation**
	Under the **ZMOD4410 IAQ 1st Generation → I2C Communication Device →**	Name: **g_comms_i2c_device1**
	Under the **I2C Communication Device → Add I2C Share Bus →**	**New → I2C Shared Bus**
	Config **I2C Shared Bus →**	Name: **g_comms_i2c_bus2**
		Channel: **2**
		Rate: **Fast-mode**
	Under **I2C Shared Bus → Add I2C Communications Peripheral →**	**New → I2C Master (r_iic_master)**
	Config **I2C Master →**	Name: **g_i2c_master2**
		Interrupt Priority Level: **Priority 5**
	Under the **ZMOD4XXX Gas Sensor → Add IRQ Driver for measurement [optional] →**	**New → External IRQ**
	Config **External IRQ**	Name: **g_external_irq4**
		Channel: **4**
		Trigger: **Falling**
		Pin Interrupt Priority: **Priority 3**
37b	Adding the ZMOD4XXX Gas Sensor module (ZMOD4510 OAQ) to the Zmod Thread.	
	New Stack →	**Sensor → ZMOD4XXX Gas Sensor**
	Config ZMOD4XXX Gas Sensor→	Name: **g_zmod4xxx_sensor1**
		Comms I2C Callback: **zmod4xxx_comms_i2c1_callback**
		IRQ Callback: **zmod4xxx_irq1_callback**
	Under the ZMOD4XXX Gas Sensor → Add Requires ZMOD Library →	**New → ZMOD4510 NO2 O3**
	Under the **ZMOD4510 NO2 O3 Generation → I2C Communication Device →**	Name: **g_comms_i2c_device2**
	Under the **I2C Communication Device → Add I2C Share Bus →**	**Use → g_comms_i2c_bus2 I2C Shared Bus**
	Under the ZMOD4XXX Gas Sensor → Add IRQ Driver for measurement →	**New → External IRQ**
	Config **External IRQ**	Name: **g_external_irq15**
		Channel: **15**
		Trigger: **Falling**
		Pin Interrupt Priority: **Priority 12**
38	**Pins** tab **→ Pin Selection → Peripherals → System:DEBUG → DEBUG0**	Operation Mode: **SWD**
39	Enable "Use float with nano printf" to print float values and add flag.	
	Project → Properties → C/C++ Build → Settings → Tool Settings tab → GNU ARM Cross C Linker → Miscellaneous	**→Check the box**: ✓ Use float with nano printf (-u _printf_float)
		Other linker flags: **--specs=rdimon.specs**

No	Steps	Intermediate Steps
40	Add environment variables	
	Project → Properties → C/C++ Build → Environment → Add	Name: **MCUBOOT_IMAGE_SIGNING_KEY**
		Value: **${workspace_loc:bootloader_ck_ra6m5}/src/secp256r1.privatekey**
		→Check the box: ✓ Add to all configurations
40a	**Project → Properties → C/C++ Build → Environment → Add**	Name: **MCUBOOT_IMAGE_VERSION**
		Value: **0**
		→Check the box: ✓ Add to all configurations
	Select **Apply and Close** to apply the settings and close the configuration window	

Note: Please add the folder `aws_ck_ra6m5_v2_ethernet_ota_app/ra/fsp` from the package released to this project with the corresponding directory due to the issue of FSP.

The above configuration is a prerequisite to generating the required stack and features for the cloud connectivity application provided in this application note. Once the **Generate Project Content** button is clicked, it generates the source code for the project. The generated source code contains the required drivers, stack, and middleware. The user application files must be added to the `src` folder.

Note: `app_thread_entry.c`, `sensors_thread_entry.c`, `oximeter_thread_entry.c`, `zmod_thread_entry.c` and `console_thread_entry.c` are the auto-generated files as part of the project creation. Users are required to add code to this file.

Note: To run the application with the supplied code, `app_thread_entry.c`, `sensors_thread_entry.c`, `oximeter_thread_entry.c`, `zmod_thread_entry.c` and `console_thread_entry.c` are available parts of this application note bundle that can be merged or overwritten to the auto-generated files.

Note: FSP-generated code must be called/used from the application, while some of the middleware needs to be called exclusively as part of the application for proper initialization. For instance, the `Mbedtls_platform_setup()` call initializes the SCE and TRNG.

For validation of the created project, the same source files listed in section MQTT/TLS Application Software Overview (as shown in Table 3) may be added. Users are required to add the directory path and subdirectory for proper compilation. The following includes paths that need to be added to **Project → Properties → C/C++ Build → Settings → Tool Settings tab → GNU Arm Cross C Compiler → Includes → Include paths (-I)** (Please choose **Add subdirectories**). Refer to the enclosed project for more details.

```
"${workspace_loc:/${ProjName}/src/"
"${workspace_loc:/${ProjName}/src/backoffAlgorithm}"
"${workspace_loc:/${ProjName}/src/agent}"
"${workspace_loc:/${ProjName}/src/SEGGER_RTT}"
"${workspace_loc:/${ProjName}/src/ob1203_bio/KALMAN}"
"${workspace_loc:/${ProjName}/src/ob1203_bio/SAVGOL}"
"${workspace_loc:/${ProjName}/src/ob1203_bio}"
"${workspace_loc:/${ProjName}/src/ob1203_bio/SPO2}"
"${workspace_loc:/${ProjName}/ra/aws/FreeRTOS/FreeRTOS-Plus/Source/AWS/ota/source/portable}"
```

Example: Add *“${workspace_loc:/${ProjName}/src/agent}”* directory path:

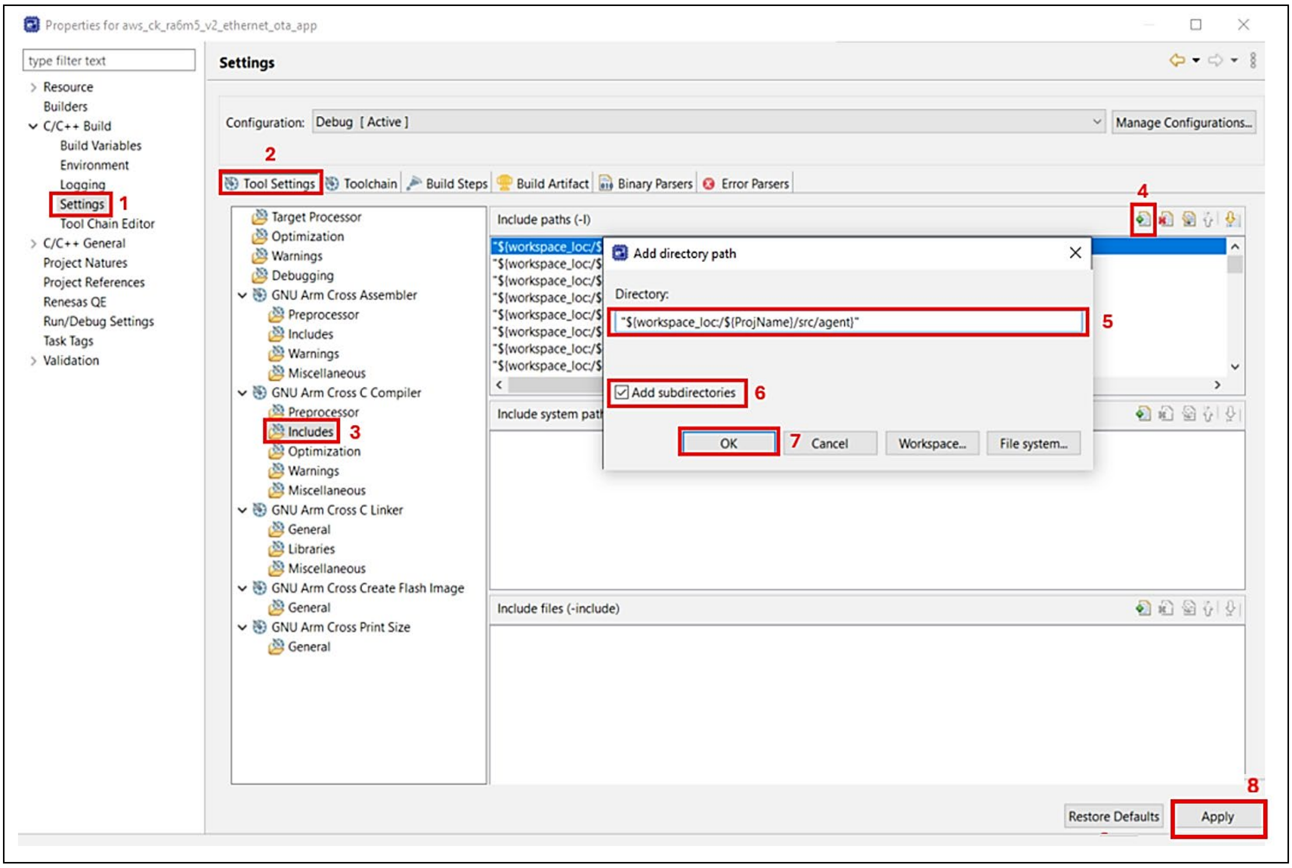

Figure 15. Add a directory path before building project

Note: In FSP v6.1.0, there is a limitation with the switching image flow due to the swap type of MCUboot. It affects the operation of OTA. Please add "`case BOOT_SWAP_TYPE_REVERT:`" to **otaPal_GetPlatformImageState** function, at line 359, in **rm_aws_ota_pal_mcuboot.c** file (aws_ck_ra6m5_v2_ethernet_ota_app/ra/fsp/src/rm_aws_ota_pal_mcuboot/rm_aws_ota_pal_mcuboot.c) as below:

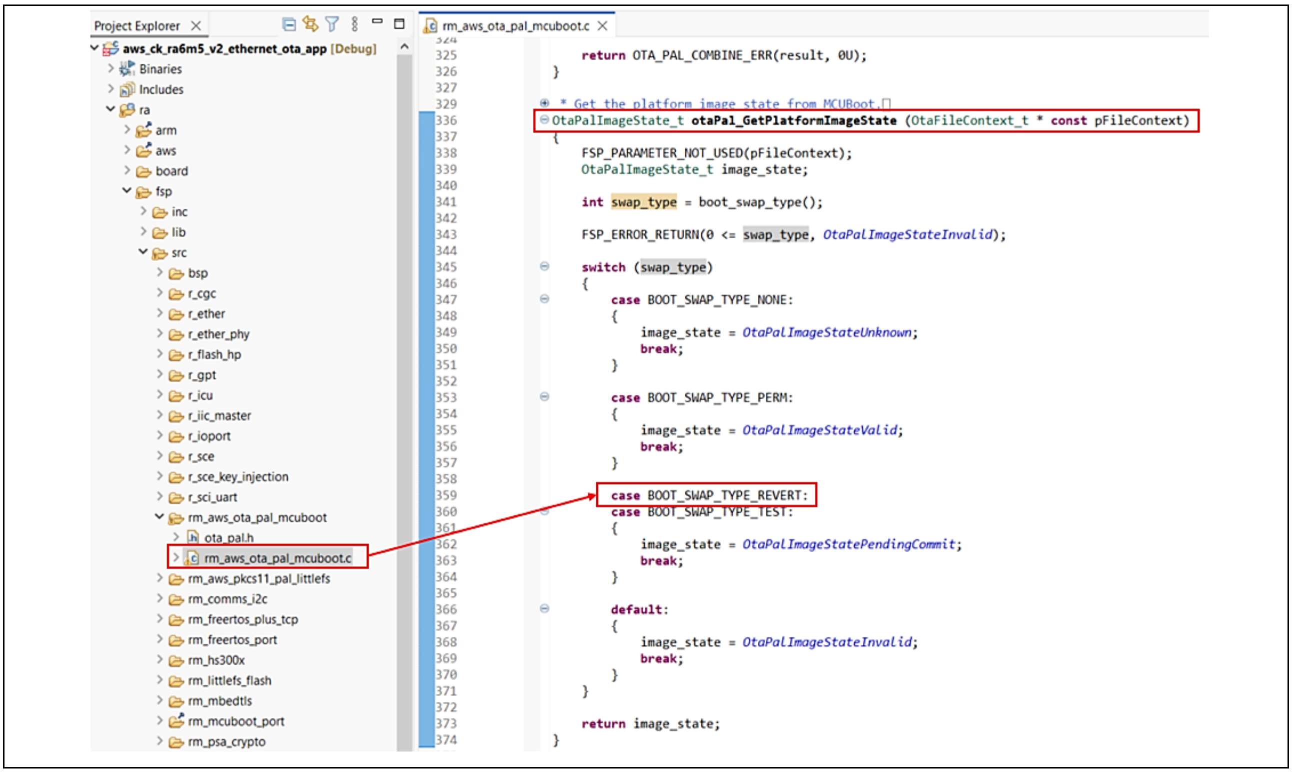

Figure 16. Modify otaPal_GetPlatformImageStage function

Right-click the application project → **Build Project.**

Note: The details of the configurator, from the default settings to the changed settings, are described in section 4.5 MQTT/TLS Configuration, including the reason for the change.

Note: After building the project, you might encounter the error: 'ERROR: Could not find HEADER_SIZE macro in bsp_linker_info.h.' Starting with FSP 6.0.0 onwards, the MCUboot project and the application project must be linked using an RA FSP solution project

The following section will show the steps to create an FSP solution project.

4.4 Creating the FSP Solution Project

Table 5 shows the step-by-step process of creating the FSP solution project, after the user completes creating the bootloader project and the application project.

Table 5 Step-by-step Details for Creating the FSP Solution Project

<table>
<tr><th>No</th><th>Steps</th><th>Intermediate Steps</th></tr>
<tr><td>1</td><td>Project Creation:</td><td>File → New → C/C++ Project</td></tr>
<tr><td>2</td><td>Project Template:</td><td>Templates for New RA C/C++ Project →
Renesas FSP Solution Project (Advanced) → Next</td></tr>
<tr><td>3</td><td>FSP Solution Project →</td><td>Project Name:
Note: Input your desired name for the project → Next</td></tr>
<tr><td>4</td><td>FSP Solution Project → Provide an existing FSP C/C++ executable project chain:</td><td>Project → Choose the application that you created at 4.3 – No 18: The project chain will display the bootloader project and the application project (a total of 2 applications).
→ Finish</td></tr>
<tr><td colspan="3">Create a memory layout for the bootloader, application image (primary and secondary), as shown in Figure 10. Memory map of the OTA application example. About the symbol conventions for memory partitioning, the user can refer to the guide at FSP User's Manual under RA Flexible Software Package Documentation > API Reference > Modules > Bootloader > MCUboot Port (rm_mcuboot_port).</td></tr>
<tr><td>5</td><td>solution.xml → Memories
Choose FLASH →</td><td>Choose FLASH_CM33_B
Size: 0x20000</td></tr>
<tr><td>6</td><td>Choose FLASH →</td><td>Add Partition
Name: __BL_0_P_H
Start: 0x00020000
Size: 0x200
Security: Secure
→ OK</td></tr>
<tr><td>7</td><td>Choose FLASH →</td><td>Choose FLASH_CM33_S
Start: 0x00020200
Size: 0x8FE00</td></tr>
<tr><td>8</td><td>Choose FLASH →</td><td>Choose FLASH_CM33_C
Start: 0x000B0000
Size: 0x8000</td></tr>
<tr><td>9</td><td>Choose FLASH →</td><td>Add Partition
Name: __BL_0_S_H
Start: 0x000B8000
Size: 0x200
Security: Non-secure
→ OK</td></tr>
<tr><td>10</td><td>Choose FLASH →</td><td>Choose FLASH_CM33_C
Start: 0x000B0000
Size: 0x0</td></tr>
<tr><td>11</td><td>Choose FLASH →</td><td>Choose __BL_0_S_H
Start: 0x000B0000
Size: 0x200</td></tr>
</table>

12	Choose **FLASH** →	**Add Partition** Name: **__BL_0_S_I** Start: **0x000B0200** Size: **0x8FE00** Security: **Non-secure** → **OK**
13	Choose **FLASH** →	**Add Partition** Name: **__BL_S** Start: **0x00140000** Size: **0x8000** Security: **Non-secure** → **OK**
14	Choose **RAM** →	Choose **RAM_CM33_S** Size: **0x74000**
15	Choose **RAM** →	Choose **RAM_CM33_C** Start: **0x20078000** Size: **0x2000**
16	Choose **RAM** →	**Add Partition** Name: **NS_BUFFER** Start: **0x2007A000** Size: **0x6000** Security: **Non-secure** → **OK**
17	Choose **RAM** →	Choose **RAM_CM33_C** Start: **0x20078000** Size: **0x0**
18	Choose **RAM** →	Choose **NS_BUFFER** Start: **0x20078000** Size: **0x8000**

After setting the solution project, right-click the FSP Solution Project → **Build Project.**

The bootloader project and application project will be built again with the new memory partition setting.

Note: Follow section "*3. Importing, Setting, Building and Loading the Project*" in the (*RA AWS Cloud Connectivity and Firmware Update OTA on CK-RA6M5 v2 with Ethernet – Getting Started Guide)* document to set up the device, build and load the application projects, refer to the downloader application's name, which users created at **4.3** – No 18 is **aws_ck_ra6m5_v2_ethernet_ota_app**, and the FSP solution project name is **aws_ck_ra6m5_v2_ethernet_ota_solution.**

4.5 MQTT/TLS Configuration

This section describes the MQTT and TLS module configuration settings that are included in this application example.

The following table lists changes made to a default configuration populated by the RA Configurator.

Table 6. Default Configuration for CK-RA6M5v2

Property	Original Value	Changed Value	Reason for Change
Application Thread			
Common → General → Use Mutexes	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Common → General → Use Recursive Mutexes	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Thread → Memory Allocation → Support Dynamic Allocation	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Thread → Memory Allocation → Total Heap Size	0	0x20000	Heap required for the FreeRTOS, AWS IOT SDK, Mbed TLS
Mbed TLS (Crypto Only)			
Platform → MBEDTLS_PLATFORM_MEMORY	Undefine	Define	This selection is required to support the MbedTLS.
General → MBEDTLS_THREADING_ALT	Undefine	Define	This selection is required to support the MbedTLS to plug in any thread library.
General → MBEDTLS_THREADING_C	Undefine	Define	This selection is required to support the MbedTLS to abstract the threading layer to allow easy plugging in of any thread library.
Public Key Cryptography (PKC) → ECC → MBEDTLS_ECDH_C	Undefine	Define	This selection is required to support the MbedTLS to enable the ECDH module.
LittleFS (Heap Selection)			
BSP → RA Common → Heap Size (bytes)	0	0x20000	Heap selection for Heap 4 and other usages with malloc.

5. Sensor Stabilization Time

This table gives the time required for the sensors to sense and provide valid data to the users. Here, you will see two columns: column 1 – when powered up for the first time, and column 2 - after a software or hard reset. If the system boots up from a cold start, the time for the sensors to provide the valid data is up to (1 min – 4 hours), whereas if the system boots up from a warm start, the time for the sensors to provide the valid data is up to (10 sec – 2 hours). For more details, refer to the specific sensor datasheet.

Table 7. Sensor Stabilization Time

Sensor Name	When Powered Up for the First Time	After Soft or Hard Reset
ZMOD4410 IAQ	Up to 1 minute	Up to 1 minute
ZMOD4510 OAQ	Up to 4 hours	Up to 5 minutes
OB1203	Up to 1 minute (after placing the index finger on the sensor, it may take up to 60 seconds to sense data)	Up to 10 seconds (after placing the index finger on the sensor, it may take up to 60 seconds to sense data)
HS3001	Up to 1 minute	Up to 10 seconds
ICP	Up to 1 minute	Up to 10 seconds
ICM	Up to 1 minute	Up to 10 seconds

Note: Stabilization time of the sensor provided above is from the point of sensor initialization.

6. MQTT/TLS Module Next Steps

- For setting up a client using a device certificate signed by a preferred CA certificate, refer to the link: https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html
- For using a self-signed certificate to configure AWS, refer to the link: https://developer.amazon.com/docs/custom-skills/configure-web-service-self-signed-certificate.html

7. References

[1] International Telecommunication Union, "ITU-T Y.4000/Y.2060 (06/2012)," 15 06 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559.

[2] Amazon Web Services, "AWS IoT Core Features," [Online]. Available: https://www.amazonaws.cn/en/iot-core/features/.

[3] Amazon Web Services, "AWS IoT Core," [Online]. Available: https://www.amazonaws.cn/en/iot-core/.

[4] I. E. T. Force, "The Transport Layer Security (TLS) Protocol Version 1.2," [Online]. Available: https://tools.ietf.org/html/rfc5246.

[5] Amazon Web Services, "AWS IoT Security," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-security.html.

[6] Amazon Web Services, "Transport Security in AWS IoT," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html.

[7] International Telecommunication Union, "X.509 (10/19) Summary," 10 2019. [Online]. Available: https://www.itu.int/dms_pubrec/itu-t/rec/x/T-REC-X.509-201910-I!!SUM-HTM-E.htm.

[8] Eclipse Foundation, "Eclipse Mosquitto™ - An open source MQTT broker," [Online]. Available: https://mosquitto.org/.

[9] Amazon Web Services, "AWS IoT Device SDK C: MQTT," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/index.html.

[10] R. Barry, "Mastering the FreeRTOS™ Real Time Kernel," in *A Hands-On Tutorial Guide*, 2016.

[11] A. I. D. S. C. Documentation, "AWS IoT Device SDK C: MQTT Functions," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/mqtt_functions.html.

[12] Amazon, "Configuring the FreeRTOS Demos," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/userguide/freertos-configure.html.

[13] "Amazon FreeRTOS mbedTLS," [Online]. Available: https://github.com/aws/amazon-freertos/blob/master/libraries/3rdparty/mbedtls/utils/mbedtls_utils.c.

[14] Renesas Electronics Corporation, "Renesas Flash Programmer (Programming GUI) - Documentation," [Online]. Available: https://www.renesas.com/us/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html#documents.

[15] Renesas Electronics Corporation, "AWS OTA PAL on MCUBoot," [Online]. Available: https://renesas.github.io/fsp/group___a_w_s___o_t_a___p_a_l___m_c_u_b_o_o_t.html.

[16] Renesas Electronics Corporation, "Booting Encrypted Image using MCUboot and QSPI," [Online]. Available: https://www.renesas.cn/cn/zh/document/apn/booting-encrypted-image-using-mcuboot-and-qspi-application-project.

[17] Nordic Semiconductor, "Bootloader," [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/mcuboot/design.html.

[18] Amazon Web Services, "FreeRTOS Over-the-Air Updates," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/userguide/freertos-ota-dev.html.

[19] Renesas Electronics Corporation, "How to implement FreeRTOS OTA using Amazon Web Services in RX65N," [Online]. Available: https://www.renesas.com/us/en/document/apn/rx-family-how-implement-freertos-ota-using-amazon-web-services-rx65n-v20221001-lts-rx-110-or-later.

[20] Renesas Electronics Corporation, "RA6 Basic Secure Bootloader Using MCUboot and Internal Code Flash," [Online]. Available: https://www.renesas.com/us/en/document/apn/ra6-basic-secure-bootloader-using-mcuboot-and-internal-code-flash.

[21] W. T. L. L. O. S. R. N. S. R. X. G. K. N. K. S. F. M. K. D. L. I. R. Valerie Lampkin, Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, IBM Redbooks, 2012.

8. Known Issues and Troubleshooting

- This section talks about the known FSP and tool-related issues. More details can be found at the link: https://github.com/renesas/fsp/issues.
- When running debug on e^2 studio, if the application is rerun multiple times, it might randomly cause an issue with the I2C communication of the OB1203 sensor. Users need to reconnect the USB cable (J10) to reset the OB1203 sensor and run the application again.

9. Debugging

Enable the `USR_LOG_LVL (LOG_DEBUG)` macro in the application project for additional information on the error during debugging.

Website and Support

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

CK-RA6M5v2 Kit Information	renesas.com/ra/ck-ra6m5
RA Cloud Solutions	renesas.com/cloudsolutions
RA Product Information	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug.22.24	—	Initial release
1.10	Oct.01.25	—	Migrated to FSP 6.1.0

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Contact information

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

Trademarks

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