Renesas RA Family

RA AWS MQTT/TLS Cloud Connectivity Solution - Wi-Fi DA16600

Introduction

This application note describes IoT Cloud connectivity solutions in general, briefly introduces IoT Cloud providers like Amazon Web Services (AWS), and covers the FSP MQTT/TLS module and its features. The application example provided in the package uses AWS IoT Core. The detailed steps in this document show first-time AWS IoT Core users how to configure the AWS IoT Core platform to run this application example.

This application note enables developers to effectively use the FSP MQTT/TLS modules in end-product design. Upon completion of this guide, developers will be able to add the “AWS Core MQTT”, “Mbed TLS”, and “AWS DA16xxx Wi-Fi sockets wrapper” using the Wi-Fi interface, configure them correctly for the target application, and write code using the included application example code as a reference for an efficient starting point.

The FSP User's Manual (available at https://renesas.github.io/fsp/) contains detailed API descriptions and other application projects that demonstrate more advanced uses of the module. It is a valuable resource for 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

The following resources are needed to build and run the MQTT/TLS application example.

Development tools and software

- Flexible Software Package (FSP) v5.3.0 and required tools (renesas.com/us/en/software-tool/flexible-software-package-fsp)

Hardware

- Renesas CK-RA6M5 v2 kit (renesas.com/ra/ck-ra6m5)
- DA16600 PMOD (US159-DA16600MEVZ)
- PC running Windows® 10 and an installed web browser (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, or Safari)
- 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)*

Prerequisites and Intended Audience

This application note assumes that the user is adept at operating the Renesas e[2] 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 e[2] studio and FSP and validates proper debug connection to the target board. In addition, this application note assumes prior knowledge of MQTT/TLS and its communication protocols and knowledge of Wi-Fi modems.

The intended audience is users who want to develop applications with MQTT/TLS modules using DA16xxx Wi-Fi modules on the Renesas RA6 MCU Series.

Note: If you are a first-time user of e[2] studio and FSP, we highly recommend you install e[2] studio and FSP on your system to run the Blinky Project and to get familiar with the e[2] studio and FSP development environment before proceeding to the next sections.

Note: This Application Project and Application Note can only use version FSP v5.3.0.

Note: If you want to build and run the attached application quickly, please jump to section (2. Running the MQTT/TLS DA16600 Wi-Fi Application Example).

Prerequisites

1. Access to online documentation is available in the Cloud Connectivity References section.
2. Access to the latest documentation for the identified Renesas Flexible Software Package.
3. Prior knowledge of operating e[2] studio and built-in (or standalone) RA Configurator.
4. Access to associated hardware documentation such as User Manuals, Schematics, and other relevant kit information (renesas.com/ra/ck-ra6m5 v2).

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 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.

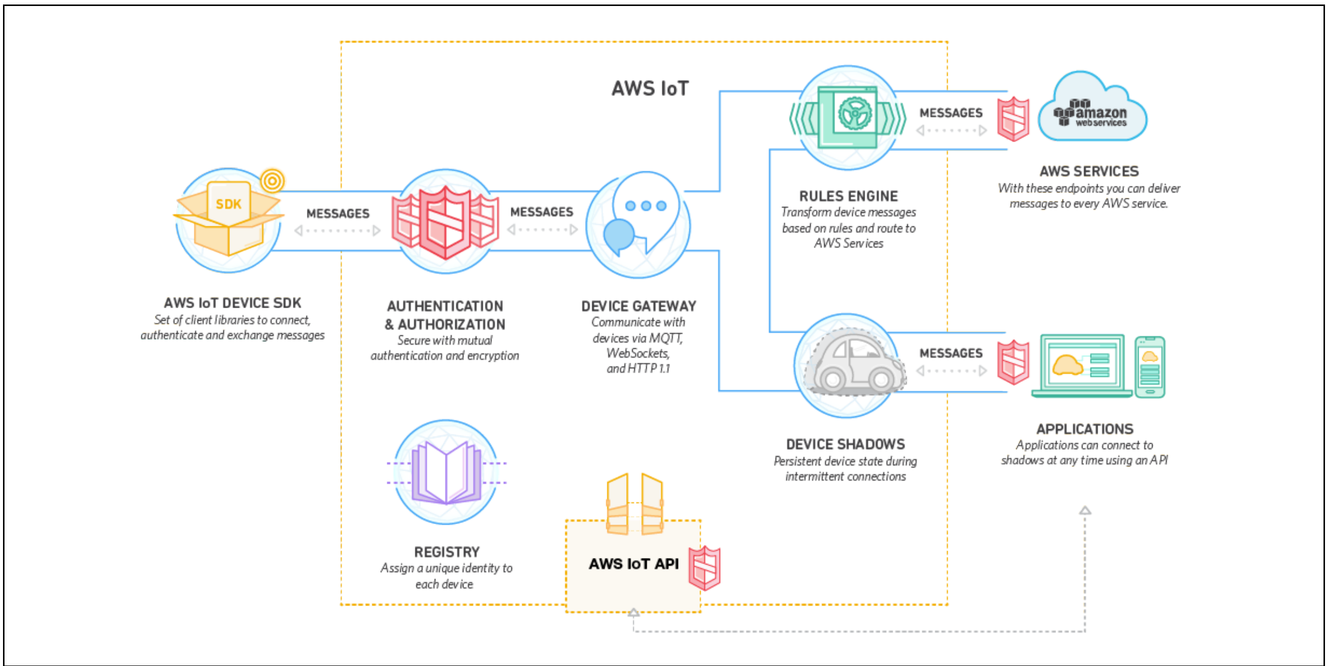

Figure 1. AWS IoT Features, Service Components, and Data Flow Diagram

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 microcontrollers, 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 Cloud Dashboard

A cloud dashboard is a monitoring and controlling GUI for multiple services, that you can build and access on a web browser. It has key advantages over on-premises software such as being easier to deploy, requiring little to no IT support and being accessible on multiple devices.

The **Dashboard** provides a high-level view of your entire fleet of devices and allows you to act on individual devices quickly. You can view graphical representations of relevant device information for your fleet, such as device ownership type, compliance statistics, and platform and OS breakdowns. You can access each set of devices in the presented categories by selecting any of the available data views from the **Device Dashboard**.

1.3.1 Data Monitoring

Data monitoring on the dashboard is a cloud data analytics monitoring solution that lets you track your performance metrics and easily visualize your data sets. You will be able to get a high-level view of your metrics, or you can drill down and analyze the detail.

For instance, it can be sensor data coming from the device in the form of temperature, pressure, and so forth.

1.3.2 Device Management

Device Management provides high-level control to configure the devices in bulk for the entire fleet of devices or to control the individual devices.

Note: All the Dashboard-specific details for this Application Project are discussed in the *(RA AWS Cloud Connectivity on CK-RA6M5v2 with Wi-Fi DA16600 – Getting Started Guide)* document.

1.4 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 to 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 generate asymmetric keys using features of the Secure Cryptography Engine (SCE) and API available through the FSP. The SCE 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.

1.5 MQTT Protocol Overview

This application notes features Message Queuing Telemetry Transport (MQTT), which 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 source 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.

1.6 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. Consequentially, all traffic to and from AWS IoT is sent securely using TLS. TLS protocol

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

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).

1.7 Device Certificates, Certificate Authorities, 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). The 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 certificates that are issued by a CA to itself or to 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 the AWS IoT Core and not another server impersonating as the 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. Running the MQTT/TLS DA16600 Wi-Fi Application Example

Refer to *RA AWS Cloud Connectivity on CK-RA6M5 v2 with Wi-Fi - Getting Started Guide* as part of this project bundle for details on running the project and visualizing the sensor data on the Renesas AWS dashboard.

3. AWS Core MQTT with DA16xxx Wi-Fi Interface

3.1 AWS Core MQTT

The AWS MQTT library included in RA FSP can connect to either AWS MQTT or to any third-party MQTT broker such as Mosquito. The complete library documentation can be found 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.

[2] Public Key length used is 2048 bits.

The AWS Core MQTT can be directly imported into a **Thread** stack. It is configured through the RA Configuration Perspective. To add the AWS Core MQTT 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 Core MQTT.

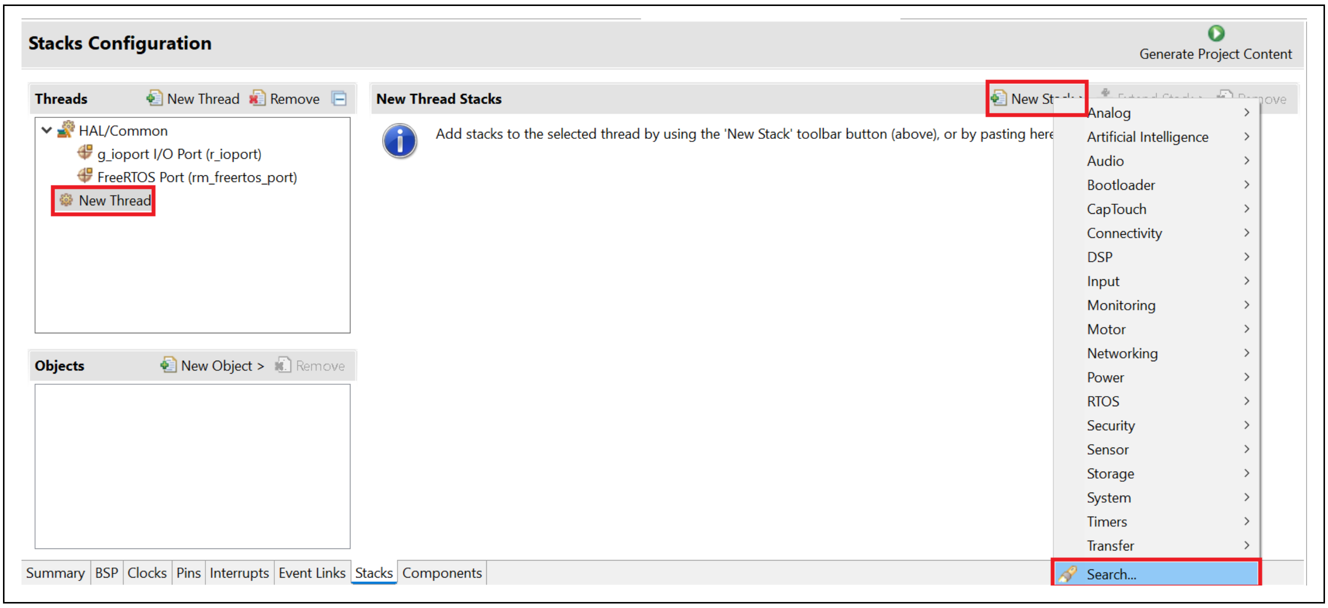

Figure 2. AWS Core MQTT Module Selection

Adding the AWS Core MQTT stack results in the default configuration with *some unmet dependencies*, as shown in Figure 3. FSP offers different Transport interfaces to the users. In this application note, we will be covering the DA16xxx Wi-Fi Interface, which uses the *AWS Transport Interface on MbedTLS/PKCS11*, as shown in the Figure 4.

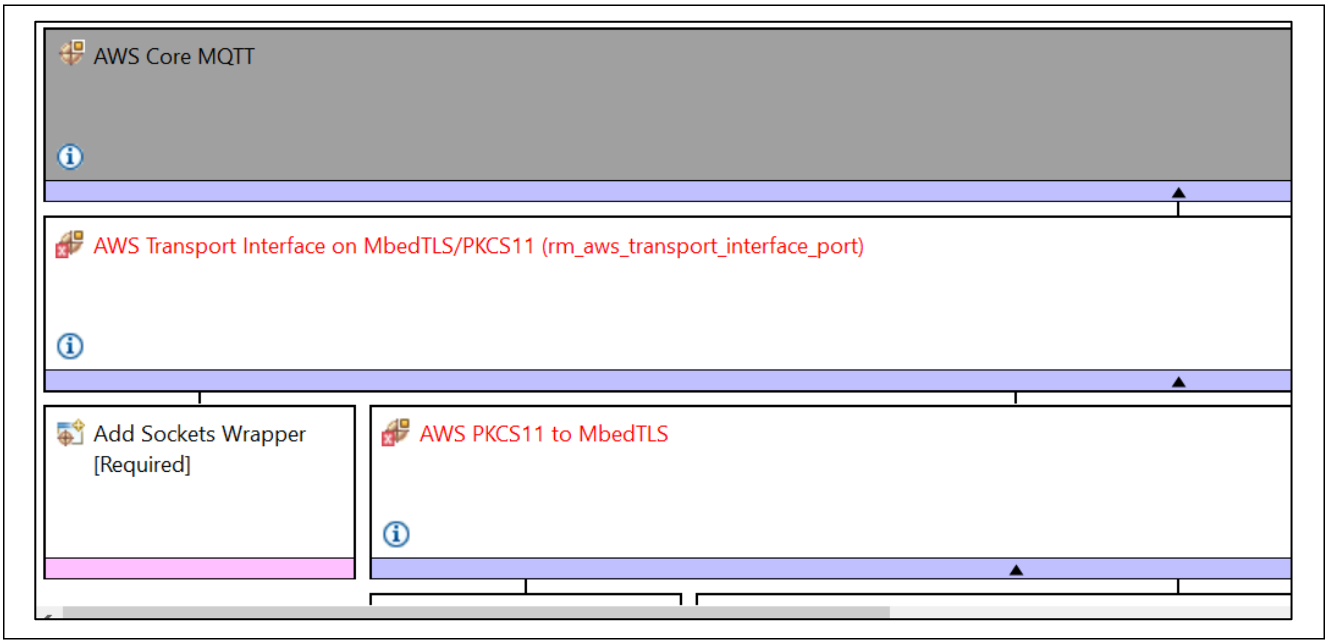

Figure 3. AWS Core MQTT Stack View

While the AWS Core MQTT stack shown contains a lot of 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 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.*

3.2 Transport Layer Implementation

The FSP AWS Transport Interface provides options for Wi-Fi, Cellular, and Ethernet. **AWS Transport Interface on the MbedTLS11** module is used for the Wi-Fi Interface. While the RA FSP contains a Secure Socket Implementation for Wi-Fi, Cellular, and Ethernet, this application and application note focuses on the use of the Wi-Fi Interface.

DA16xxx Wi-Fi Sockets can be added to the Thread Stack by clicking on **Add Sockets Wrapper > New > AWS DA16xxx WiFi Sockets Wrapper**.

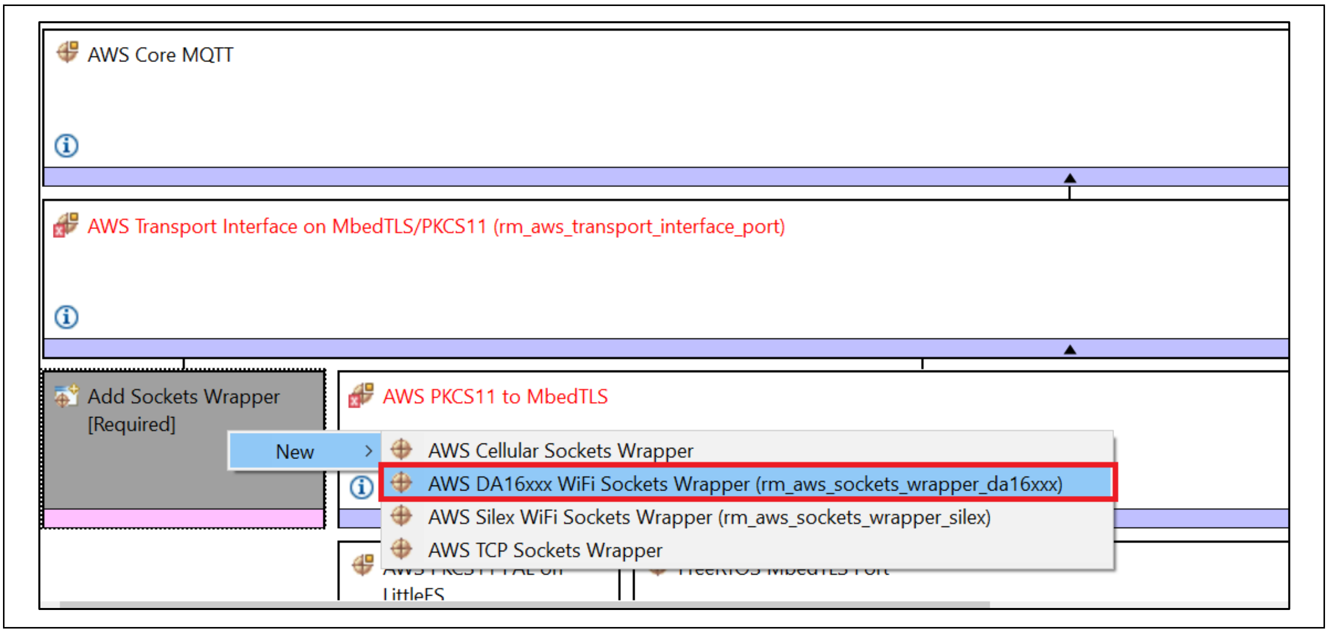

Figure 4. Adding DA16xxx Wi-Fi Interface to the Core MQTT Module

Upon 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 pop up. 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 > Common > Memory Allocation > Support Dynamic Allocation > Enabled**.
  For *error: Mutexes must be enabled in the FreeRTOS thread*, enable mutexes in Thread's 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 > Common > Memory Allocation > Total Heap Size > 0x20000.**
- For **LittleFS** *error: A heap is required to use Malloc*, add heap under **BSP Tab > Properties > RA Common > Heap size (bytes) > 0x20000**.
- 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.**
- UART-specific errors can be resolved by enabling the Flow control and selecting the appropriate RTS and CTS pin selection.

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 errors, as shown in the Figure 5.

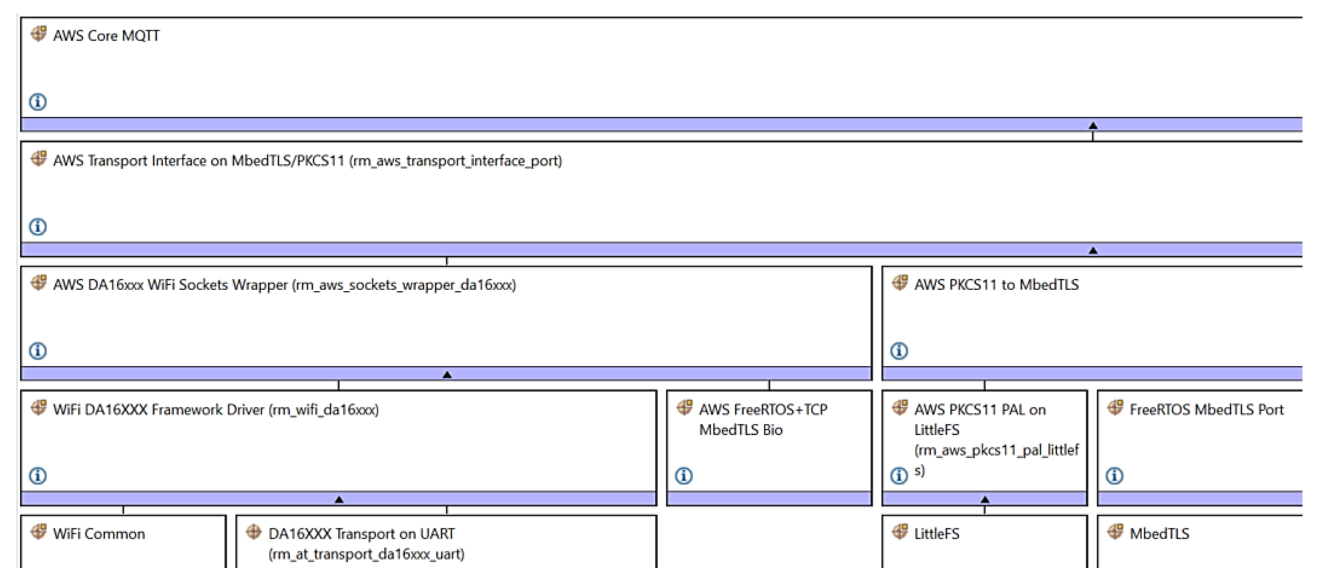

Figure 5. Expanded DA16xxx Wi-Fi Socket Interface Module

3.3 Mbed TLS

Mbed TLS is Arm®'s implementation of the TLS protocols as well as 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 SCE on the MCU.

The following underlying mandatory changes are needed to the project using the DA16xxx WiFi Sockets on the FreeRTOS+Crypto module:

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, which 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. (Note: FreeRTOS requires a Heap size greater than 0x20000 for the AWS MQTT stack and its operation)
7. Add Persistent Storage on LittleFS.

Additional documentation on the Mbed TLS 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).*

3.4 MQTT Module APIs Usage

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

Table 2. MQTT Module APIs

API	Description
MQTT_Init	Initializes an MQTT context
MQTT_Connect	Establishes an MQTT session
MQTT_Subscribe	Sends MQTT SUBSCRIBE for the given list of topic filters to the broker
MQTT_Publish	Publishes a message to the given topic name
MQTT_Ping	Sends an MQTT PINGREQ to the broker
MQTT_Unsubscribe	Sends MQTT UNSUBSCRIBE for the given list of topic filters to the broker
MQTT_Disconnect	Disconnect an MQTT session
MQTT_ProcessLoop	Loop to receive packets from the transport interface. Handles keep-alive
MQTT_ReceiveLoop	Loop to receive packets from the transport interface. Does not handle keep-alive
MQTT_GetPacketId	Get a packet ID that is valid according to the MQTT 3.1.1 specification.
MQTT_MatchTopic	A utility function that determines whether the passed topic filter and topic name match according to the MQTT 3.1.1 protocol specification.
MQTT_GetSubAckStatusCodes	Parses the payload of an MQTT SUBACK packet that contains status codes corresponding to topic filter subscription requests from the original subscribe packet
MQTT_Status_strerror	Error code to string conversion for MQTT statuses.

4. Cloud Connectivity Application Example

4.1 Overview

This application project demonstrates the use of APIs available through the Renesas FSP-integrated modules for Amazon IoT SDK C, Mbed TLS module, Amazon FreeRTOS, and HAL Drivers operating on Renesas RA MCUs. Network connectivity is established using the DA16xxx Wi-Fi module. The application running on a Renesas Cloud Kit also serves as a guide for the operation of Core MQTT, Mbed TLS/Crypto, and Wi-Fi configuration using the FSP configurator. The application may be used as a starting point for inspiring other customized cloud-based solutions using Renesas RA MCUs. In addition, it simply demonstrates the operation and setup of cloud services available through the cloud service provider.

The upcoming sub-sections show the step-by-step creation of a 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.

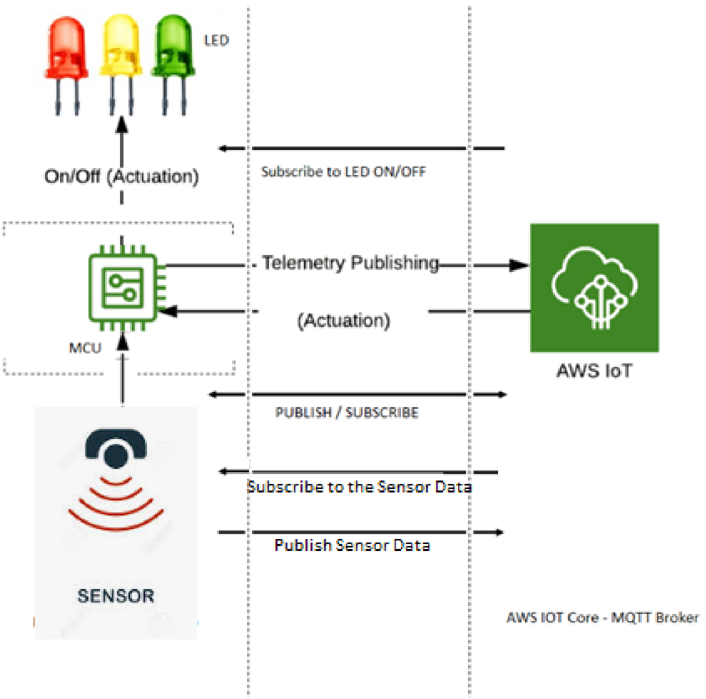

Figure 6. MQTT Publish/Subscribe to/from AWS IoT Core

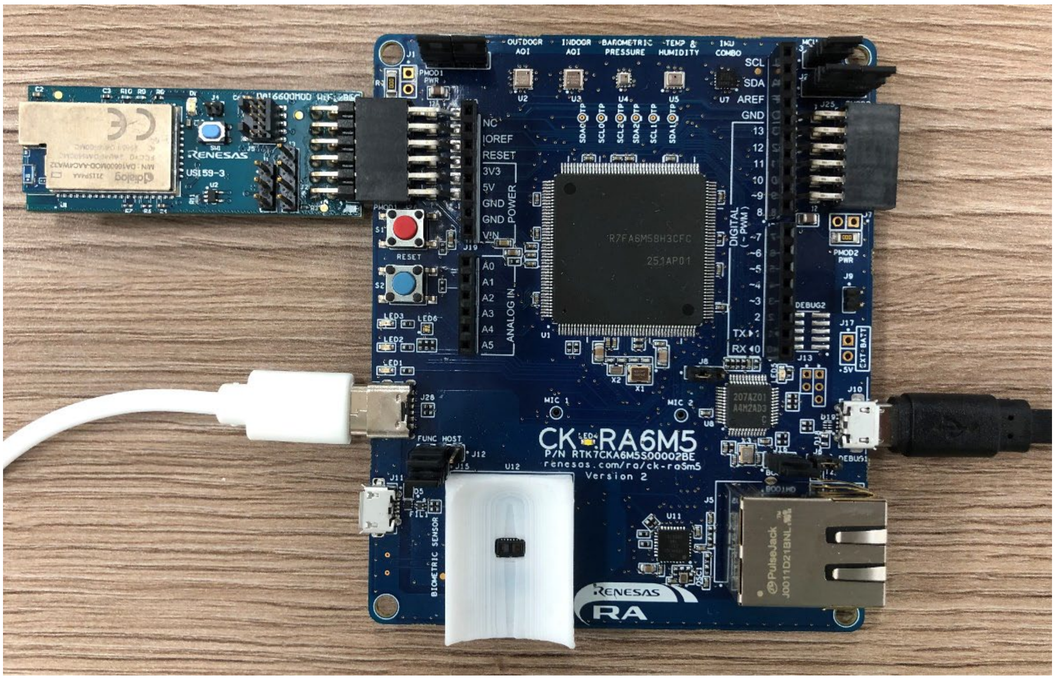

Figure 7. Hardware Setup

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 functions 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 functions prototypes commonly used across the project.
6.	`src/console_thread_entry.c`	Contains the code for command line interface and flash memory operations.
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/mqtt_demo_helpers.c`	Contains code and functions used in the MQTT interface for Cloud Connectivity.
14.	`src/mqtt_demo_helpers.h`	Accompanying header for exposing functionality provided by `mqtt_demo_helpers.c`.
15.	`src/oximeter_thread_entry.c`	Contains codes for oximeter sensor thread's operation.
16.	`src/oximeter.c`	Contains codes for oximeter sensor's initialization and measurement.
17.	`src/oximeter.h`	Contains the Data structure and function prototypes for the oximeter sensor.
18.	`src/r_typedefs.h`	Contains the common derived data types
19.	`src/RA_HS3001.c`	Contains the code and function for Renesas Relative Humidity and Temperature Sensor.
20.	`src/RA_HS3001.h`	Contains the common data structure's function prototypes for the Renesas Relative Humidity and Temperature sensors.

No.	Filename	Purpose
21.	src/RA_ICM42605.c	Contains codes for 6 Axis sensor (Gyroscope, Accelerometer) sensor's initialization and measurement.
22.	src/RA_ICP20100.c	Contains codes for Barometric Pressure and Temperature sensor's initialization and measurement.
23.	src/RA_ZMOD4XXX_Common.c	Contains the common code for the Renesas ZMOD sensors
24.	src/RA_ZMOD4XXX_Common.h	Contains the common data structure's function prototypes for the Renesas ZMOD sensors
25.	src/RA_ZMOD4XXX_IAQ1stGen.c	Contains the common code for the Renesas ZMOD Internal Air Quality sensors
26.	src/RA_ZMOD4XXX_OAQ1stGen.c	Contains the common code for the Renesas ZMOD Outer Air Quality sensors
27.	src/RmcI2C.c	Contains the I2C wrapper functions for the third-party sensors not integrated with FSP
28.	src/RmcI2C.h	Contains the I2C function prototypes for wrapper functions for the third-party sensors not integrated with FSP
29.	src/sensor_thread_entry.c	Contains the Code to access the Sensor data from the different sensors and order topic to publish.
30.	src/ user_choice.c	Contains the code for the user's choice of sensors and user configurations
31.	src/user_choice.h	Contains the Function prototypes for the Sensor and its user configuration for the different sensors and its data accessibility.
32.	src/usr_config.h	To customize the user configuration to run the application.
33.	src/usr_data.h	Accompanying header file for the application thread.
34.	src/usr_hal.c	Contains data structures and functions used for the Hardware Abstraction Layer (HAL) initialization and associated utilities.
35.	src/usr_hal.h	Accompanying header for exposing functionality provided by usr_hal.c.
36.	src/usr_wifi.c	Contains data structures and functions used in Wi-Fi Configs for Cloud Application Project.
37.	src/usr_wifi.h	Contains declarations of data structures and functions used in usr_wifi.c
38.	src/zmod_thread_entry.c	Contains the code for indoor air and outdoor air quality sensors
39.	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.
40.	src/SEGGER_RTT/SEGGER_RTT.h	
41.	src/SEGGER_RTT/SEGGER_RTT_Conf.h	
42.	src/SEGGER_RTT/SEGGER_RTT_printf.c	
43.	src/backoffAlgorithm/backoff_algorithm.c	Retry algorithms with random back off for the next retry attempt
44.	src/backoffAlgorithm/backoff_algorithm.h	Retry algorithms with random back off for the next retry attempt header file
45.	src/subcription_manager/mqtt_subscription_manager.c	MQTT Subscription manager, which handles the callback

No.	Filename	Purpose
46.	src/subcription_manager/mqtt_subscription_manager.h	Associated header file for MQTT Subscription manager, which handles the callback.
47.	src/console_menu/console.c	Contains data structures and functions used to print data on the console using UART
48.	src/console_menu/console.h	Contains the Function prototypes used to print data on the console using UART
49.	src/console_menu/menu_flash.c	Contains data structures and functions used to provide CLI flash memory-related menu
50.	src/console_menu/menu_flash.h	Contains the Function prototypes and macros used to provide CLI flash memory-related menu
51.	src/console_menu/menu_kis.c	Contains functions to get the FSP version, get UUID, and help option for the main menu on the CLI
52.	src/console_menu/menu_kis.h	Contains the function prototypes and macros used to get fsp version, get uuid, and help option for the main menu on CLI
53.	src/console_menu/menu_main.c	Contains data structures and functions used to provide CLI main menu options
54.	src/console_menu/menu_main.h	Contains the Function prototypes and macros used to provide CLI main menu options
55.	Src/flash/ flash_hp.c	Contains data structures and functions used to perform flash memory-related operations
56.	src/flash/ flash_hp.h	Contains the Function prototypes and macros used to perform flash memory-related operations
57.	src/ob1203_bio/KALMAN/kalman.c	Contains algorithm for Heart Rate, Blood Oxygen Concentration, Pulse Oximetry, Proximity, Light and Color Sensor sample calculations
58.	src/ob1203_bio/KALMAN/kalman.h	
59.	src/ob1203_bio/SAVGOL/SAVGOL.c	
60.	src/ob1203_bio/SAVGOL/SAVGOL.h	
61.	src/ob1203_bio/SPO2/SPO2.c	
62.	src/ob1203_bio/SPO2/SPO2.h	
63.	src/ob1203_bio/ob1203_bio.c	Contains codes for the ob1203 sensor's implementation to use with FSP stacks.
64.	src/ob1203_bio/ob1203_bio.h	Contain user data structure and function prototypes used in ob1203_bio.c
65.	src/hal_entry.c	Contains hal level functions used in the application

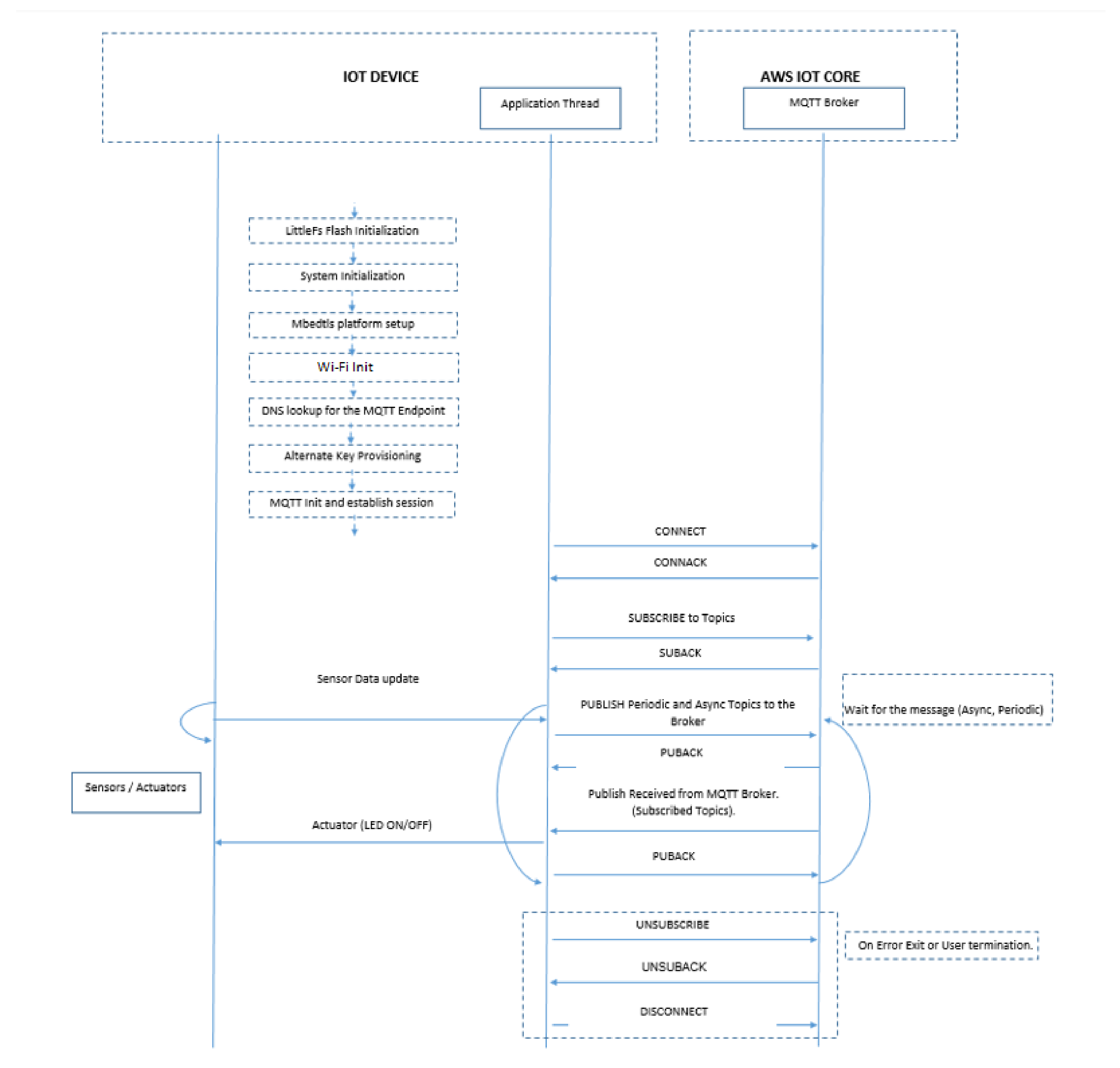

Figure 8. Application Example Implementation Details

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 in 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 Wi-Fi

	Steps	Intermediate Steps
1	**Project Creation:**	**File → New → C/C++ Project**
2	**Project Template:**	**Templates for New RA C/C++ Project → Renesas RA C/C++ Project → Next**
3	**e^2 studio - Project Configuration (RA C Executable Project) →**	**Project Name (Name for the Project)** Note: Input your desired name for the project -> **Next**
4	**Device Selection →**	FSP Version: **5.3.0**
		Board: **CK-RA6M5 V2**
		Device: **R7FA6M5BH3CFC**

	Steps	Intermediate Steps
		Language: C
5	**Select Tools**	Toolchain: **GNU ARM Embedded** (Default)
		Toolchain version: (13.2.1.arm-13-7 or newer)
		Debugger: **J-Link ARM → Next**
5a	**Project Type Selection**	Flat (Non-TrustZone) Project **→ Next**
6	**Build Artifact and RTOS Selection**	Artifact Selection: **Executable**
		RTOS Selection: FreeRTOS(v10.6.1+fsp5.3.0) → **Next**
6a	**Project Template Selection**	**Project Template Selection: FreeRTOS – Minimal – Static Allocation → Finish**
7	**Clock**	**HOCO 20MHz →PLL Src: HOCO → PLL Div/2 →PLL Mul x20.0 → PLL 200MHz**
8	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**
8a	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**
	Common → Memory Allocation	Support Dynamic Allocation: **Enabled**
		Total Heap Size: **0x20000**
9	Add the Heap Implementation in **HAL/Common**	
	New Stack →	**RTOS → FreeRTOS Heap 4**
10	Adding the AWS MQTT Wrapper Module to the Application Thread	
	Note: Now the Newly created thread (Application thread) is ready to add new stack (Here the AWS Core MQTT is added)	
	New Stack →	**Networking → AWS Core MQTT**
10a	Under the **AWS Transport Interface on MbedTLS/PKCS11 → Add Sockets Wrapper**, add	**New → AWS DA16xxx WiFi Sockets Wrapper**
10b	Under the **SCE Compatibility mode → Add Key Injection for PSA Crypto (Optional),** add	**New → Key Injection for PSA Crypto**
10c	**AWS Core MQTT →**	**Common → Retry count for reading CONNACK from network → 10**
11	Adding persistent storage support for AWS PKCS11 and resolve the error in the configurator by selecting the Heap size in the BSP Tab.	
	Under **the MbedTLS (Crypto only) → Add Persistent Storage on LittleFS (Optional) →**	**Use → LittleFS**
	BSP Tab → RA Common →	Heap size (bytes): **0x20000**
11a	**LittleFS on Flash →**	**Block count → (BSP_DATA_FLASH_SIZE_BYTES/256)**
12	Some dependency related to TLS Support are needed 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**

	Steps	Intermediate Steps
	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**
13	**WiFi DA16XXX Framework Driver Configuration** Note: This is only applicable to the Wi-Fi application project. Most of the default settings remain the same, except a few of the default configuration needs to be changed	
	Common →	Enable DA16600 Support: **Enabled**
	Under the **WiFi DA16XXX Framework Driver → g_uart0 UART**	
	g_uart0 UART → Common →	FIFO Support: **Enable**
		DTC Support: **Enable**
		Flow Control Support: **Enable**
	Module g_uart0 UART → General →	Name: **g_uart9**
		Channel: **9**
	Under **UART → Add DTC Driver for Transmission [Recommended but optional] →**	**New → Transfer (r_dtc)**
	Under **UART → Add DTC Driver for Reception [Recommended but optional] →**	**New → Transfer (r_dtc)**
13a	Config for PMOD1 Reset pin (P311)	
	Pins tab → **Pin Selection → Ports → P3 → P311 →**	Mode: **Output mode (Initial High)**
		Drive Capacity: **High**
14	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**
	Interrupts:	Callback: **g_user_timer_cb**
		Overflow/Crest Interrupt Priority: **Priority 5**
14a	Configure Pins for CGC	
	Pins Tab → Pin Selection → Peripherals → System: CGC → CGC0 →	Operation Mode: **Main+Sub Osc**
15	**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**

	Steps	Intermediate Steps
		Subclock Populated: **Not Populated**
16	Adding FreeRTOS Objects for the Application and Sensors	
	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**
	Stacks Tab → Objects →	**New Object → Mutex**
	Property Settings for the Mutex	Symbol: **g_console_out_mutex**
		Type: **Mutex**
		Memory Allocation: **Static**
	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**
	Stacks Tab → Objects →	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_iaq_queue**
		Item Size (Bytes): **12**
		Queue Length (Items): **1**
		Memory Allocation: **Static**
	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**
	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**
	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**
	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**
17	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**
17a	Add the UART module to the Console Thread	

	Steps	Intermediate Steps
	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**
17b	Add Flash module to Console Thread	
	New Stack →	**Storage → Flash**
	Module Flash →	Name: **user_flash**
		Data Flash Background Operation: **Disabled**
		Callback: **flash_callback**
		Flash Ready Interrupt Priority: **Priority 2**
		Flash Error Interrupt Priority: **Priority 2**
18	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**
18a	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**
	Under **I2C Shared Bus → Add I2C Communications Peripheral →**	**New → I2C Master(r_iic_master)**
	Config for I2C Master →	Name: **g_i2c_master0**
		Channel: **0**
		Rate: **Fast-mode**
		Interrupt Priority Level: **Priority 5**
18b	Adding ICP-20100 and ICM-42605 sensors to the Sensor 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, its 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**

	Steps	Intermediate Steps
	New Stack →	**Input → External IRQ**
	Config for External IRQ	Name: **g_external_irq6**
		Channel: **6**
		Trigger: **Falling**
		Callback: **ICP_IRQ_CALLBACK**
18c	Adding I2C Communication Device and External IRQ for ICM-42605 into Sensor 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**
		Channel: **12**
		Trigger: **Falling**
		Callback: **ICM_42605_Callback1**
19	Add Oximeter Thread for OB1203 sensor' 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**
19a	Add the OB1203 sensor module, PPG mode to the Oximeter Thread.	
	New Stack →	**Sensor → OB1203 Light/Proximity/PPG Sensor**
	Config **OB1203 Light/Proximity/PPG Sensor →**	Name: **g_ob1203_sensor0**
	Under the **OB1203 Light/Proximity/PPG Sensor → Add Requires OB1203 Operation mode →**	**New → OB1203 PPG mode**
	Under the **OB1203 PPG mode → I2C Communication Device →**	Name: **g_comms_i2c_device3**
	Under the **I2C Communication Device → Add I2C Share Bus →**	**New → I2C Shared Bus**
	Config **I2C Shared Bus →**	Name: **g_comms_i2c_bus1**
	Under **I2C Shared Bus → Add I2C Communications Peripheral →**	**New → I2C Master (r_iic_master)**
	Config **I2C Master →**	Name: **g_i2c_master1**
		Channel: **1**
		Rate: **Standard**
		Interrupt Priority Level: **Priority 12**

	Steps	Intermediate Steps
	Under the **OB1203 Light/Proximity/PPG Sensor → Add IRQ Driver for measurement →**	**New → External IRQ**
	Config for **External IRQ →**	Name: **g_external_irq14**
		Channel: **14**
		Trigger: **Falling**
19b	Add the OB1203 sensor module, Proximity mode to the Oximeter Thread.	
	New Stack →	**Sensor → OB1203 Light/Proximity/PPG Sensor**
	Config OB1203 Light/Proximity/PPG Sensor →	Name: **g_ob1203_sensor1**
	Under the **OB1203 Light/Proximity/PPG Sensor → Add Requires OB1203 Operation mode →**	**New → OB1203 Proximity mode**
	Under the **OB1203 Proximity mode → I2C Communication Device →**	Name: **g_comms_i2c_device6**
	Under the **I2C Communication Device → Add I2C Share Bus →**	**Use → g_comms_i2c_bus1 I2C Shared Bus**
	Under the **OB1203 Light/Proximity/PPG Sensor → Add IRQ Driver for measurement →**	**Use → g_external_irq14 External IRQ**
20	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**
20a	Adding the ZMOD4XXX Gas Sensor module (ZMOD4410 IAQ) to the Zmod Thread.	
	New Stack →	**Sensor → ZMOD4XXX Gas Sensor**
	Config ZMOD4XXX Gas Sensor→	Name: **g_zmod4xxx_sensor0**
		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**
	Under **I2C Shared Bus → Add I2C Communications Peripheral →**	**New → I2C Master (r_iic_master)**
	Config **I2C Master →**	Name: **g_i2c_master2**
		Channel: **2**
		Rate: **Fast-mode**
		Interrupt Priority Level: **Priority 5**

	Steps	Intermediate Steps
	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**
20b	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**
		Callback: **zmod4xxx_comms_i2c1_callback**
		IRQ Callback: **zmod4xxx_irq1_callback**
	Under the ZMOD4XXX Gas Sensor → Add Requires ZMOD Library →	**New → ZMOD4510 OAQ 1st Generation**
	Under the **ZMOD4510 OAQ 1st 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**
21	Enable "Use float with nano printf" to print float values and add a 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**

The above configuration is a prerequisite to generate the required stack and features for the cloud connectivity application provided with 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 into the `src` folder.

Note: `app_thread_entry.c`, `sensor_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`, `sensor_thread_entry.c`, `oximeter_thread_entry.c`, `zmod_thread_entry.c`, and `console_thread_entry.c` are available parts of this application note bundle 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 paths need to be added to **Project → Properties → C/C++ Build → Settings → Tool Settings tab → GNU Arm Cross C Compiler → Includes → Include paths (-I)**. Refer to the enclosed project for more details.

```
"${workspace_loc:/${ProjName}/src/backoffAlgorithm}"
"${workspace_loc:/${ProjName}/src/subcription_manager}"
"${workspace_loc:/${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}"
```

The details of the configurator from the default settings to changed settings are described in the following sections, including the reason for the change.

4.4 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 5. Application configurations for the CK-RA6M5 that differ from the default

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 → Memory Allocation → Support Dynamic Allocation	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Common → 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 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 2 columns: column 1 – when powered up for the first time and column 2 - after 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 bootup 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 6. Sensor Stabilization Time

Sensor Name	When Powered Up 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 2 hours
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] 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.

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

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

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

[8] 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.

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

[10] 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.

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

[12] 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.

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

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

[15] 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.

8. Known Issues and Troubleshooting

- This section discusses known FSP and tool-related issues. More details can be found at https://github.com/renesas/fsp/issues.
- It is recommended that you use the dashboard with the Microsoft Edge browser; it does not work properly with the Google Chrome browser.
- In case of an error related to wifi_init(): *"[ERR] In Function: wifi_init(), ** Wi-Fi Init Failure **"* is seen on the console every time despite good Wi-Fi connectivity, it is recommended to check the SDK version in DA16600 PMOD. It is required to use DA16600 SDK v3.2.7.1 or later for the application to work. To confirm the DA16600 SDK version and update it, please refer guidelines in DA16200/DA16600 SDK Update Guide
- When the application is running successfully, if the Wi-Fi connection goes down on the AP/router end, the application will not reconnect to the AP/Router. Reset of the Board is required in this case.
- Depending on security levels and configuration of Wi-Fi Router/Modem, sometimes the application may not connect to the network. Please try Simple AP or Hotspot using iOS/Android system.
- When running debug on e^2 studio, if the application is rerun multiple times, it might randomly occur issue with i2c communication of OB1203 sensor. Users need to reconnect the micro-USB cable (J10) to reset OB1203 sensor and run the application again.

9. Debugging

Enable the `USR_LOG_LVL (LOG_DEBUG)` macro in the application project for additional information of 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	Oct.18.23	—	Initial release
1.10	Dec.22.23	—	Updated to FSP 5.1.0
1.20	Oct.03.24	—	Updated to FSP 5.3.0

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

Trademarks

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/.