Renesas RA ファミリ

AWS MQTT/TLS クラウド接続ソリューション-Cellular 方式

要旨

本アプリケーションノートでは、IoT クラウド接続ソリューション全般の説明、Amazon Web Services (AWS)などの IoT クラウドプロバイダの簡単な紹介、さらに FSP MQTT/TLS モジュールとその機能についても記述しています。付属のアプリケーション例は、AWS IoT Core を使用しています。このドキュメントの詳細な手順では、AWS IoT Core を初めて使用するユーザが、このアプリケーション例を実行するために AWS IoT Core プラットフォームを構築する方法を示します。

本アプリケーションノートにより、開発者は最終製品の設計で FSP MQTT/TLS モジュールを効果的に使用できるようになります。具体的には、開発者は Cellular インターフェースを使用して"AWS Core MQTT"、"Mbed TLS"、"AWS cellular sockets"を追加して、ターゲットアプリケーション用に正しく設定すること、さらに付属アプリケーションのサンプルコードを参考にすることで、効率的にコードを書き始めることができるようになります。

API の詳細な説明や、より高度なモジュールを使用するための他のアプリケーションプロジェクトは、FSP ユーザーズマニュアル(https://renesas.github.io/fsp/)で参照でき、より複雑な設計を行う際に役に立ちます。

MQTT/TLS AWS クラウド接続ソリューションは、CK-RA6M5 でサポートされています。

対象デバイス

- RA6M5 MCU グループ

要求リソース

MQTT/TLS アプリケーションのサンプルをビルドして実行するには、以下のリソースが必要です。

開発ツールとソフトウェア

- e^2 studio ISDE v 22.4.0 以降(renesas.com/us/en/software-tool/e-studio)
- Flexible Software Package (FSP) 3.7.0 以降(renesas.com/us/en/software-tool/flexible-softwarepackage-fsp)

ハードウェア

- Renesas CK-RA6M5 キット(renesas.com/ra/ck-ra6m5)
- Windows® 10 稼働 PC と、Web ブラウザ(Google Chrome、Internet Explorer、Microsoft Edge、Mozilla Firefox、または Safari)
- Micro USB ケーブル(キットの付属品。CK-RA6M5 ユーザーズマニュアル参照)

前提条件と対象ユーザ

本アプリケーションノートでは、Renesas e^2 studio IDE with Flexible Software Package (FSP)の操作に習熟しているユーザを想定しています。そうでない場合は、、FSP ユーザーズマニュアルの「Starting Development」セクションと「Debug the Blinky Project」セクションを読み、その手順に従うことを推奨します。そうすることで、e^2 studio と FSP に慣れ、ターゲットボードの正しいデバッグ接続を確認することができます。さらに、本アプリケーションノートは、MQTT/TLS とその通信プロトコルに関する予備知識、および Cellular モデムに関する知識を前提としています。

対象ユーザは、Renesas RA6 MCU シリーズで Cellular モジュールを使用した MQTT/TLS モジュールによるアプリケーションを開発したい方々です。

注：　e² studio と FSP を初めて使う場合は、自身のシステムに e² studio と FSP をインストールして Blinky プロジェクトを実行し、e² studio と FSP の開発環境に慣れてから、次のセクションに進むことを強く推奨します。

注：　このアプリケーションプロジェクトとアプリケーションノートは、FSP v3.7.0 以降のバージョンのみ使用可能です。

前提条件

1. クラウド接続例のセクションで利用できるオンラインドキュメントへのアクセス
2. Renesas Flexible Software Package の最新ドキュメントへのアクセス
3. e² studio および内蔵(またはスタンドアロン)RA Configurator の操作に関する事前知識
4. ユーザーズマニュアル、回路図、その他キット関連情報などのハードウェアドキュメントへのアクセス (renesas.com/ra/ck-ra6m5)

目次

1.　クラウド接続用コンポーネントの紹介

1.1　一般的な概要

Internet-of-Things (IoT)は、情報化社会のグローバルインフラであり、既存の進化した相互運用性のある情報と通信技術に基づいて(物理的および仮想的な)「モノ」を相互に接続することにより、高度なサービスを実現します。この定義における「モノ」とは、通信ネットワーク上で識別、統合できる物理世界(実体)または情報世界(仮想)の物体を指します。IoT における「デバイス」とは、通信の必須機能と、オプション機能としてセンシング、アクチュエータ、データ収集、データ保存、データ処理を備えた機器のことを指します。

多くの場合、通信は、ネットワークホストサービス、インフラストラクチャ、生成されたデータを処理/分析しデバイスを管理するビジネスアプリケーションのプロバイダと行われます。このようなプロバイダをクラウドサービスプロバイダと呼びます。デバイスやクラウドサービスプロバイダには多くのメーカーがありますが、本アプリケーションノートにおいては、IoT 向けに Amazon Web Services (AWS)が提供するサービスに接続する Renesas RA マイクロコントローラー(MCU)をデバイスとします。

1.2　クラウドサービスプロバイダ

AWS IoT は、お客様の IoT デバイスを他のデバイスや AWS クラウドサービスと接続するためのクラウドサービスを提供します。クラウドサービスプロバイダとして、AWS IoT は次の機能を提供します。

- デバイスの接続と管理
- デバイスの接続とデータの保護
- デバイスデータの処理と操作
- 恒常的なデバイス状態の取得と設定

図 1 は、AWS IoT によって提供される機能をまとめたものです。

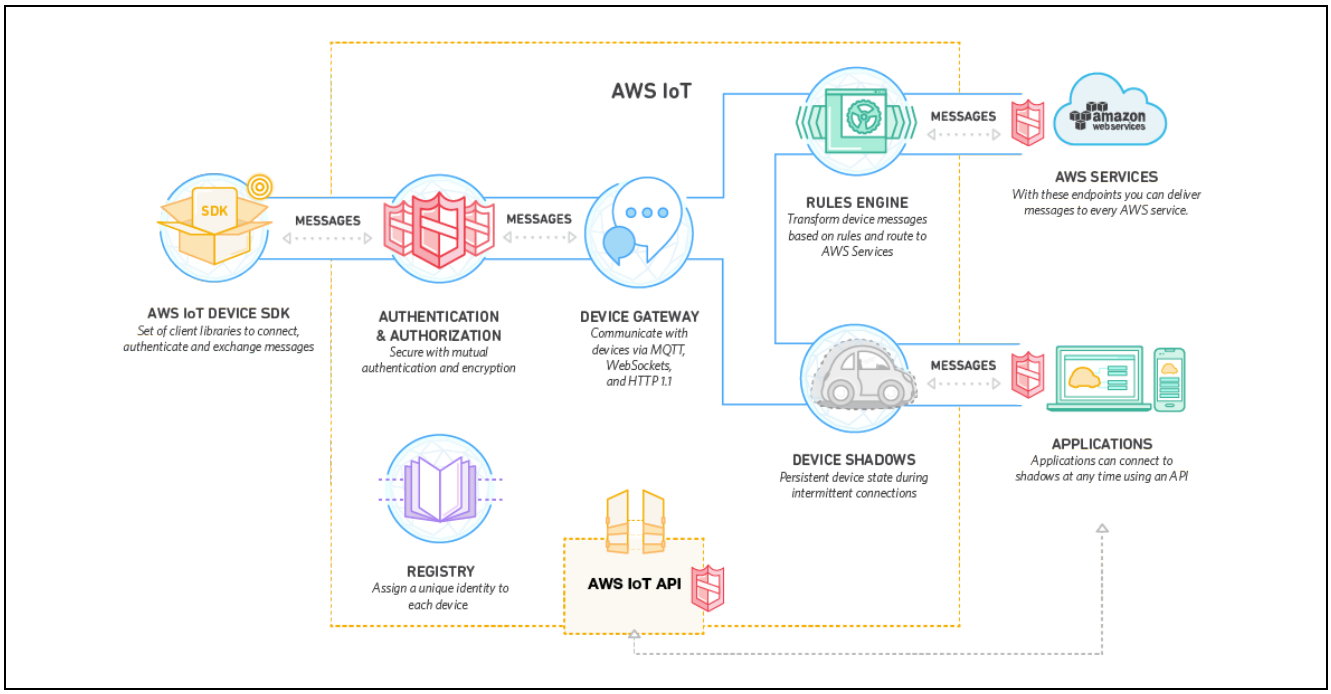

図 1　AWS IoT の機能、サービスコンポーネント、データフローダイアグラム

AWS の主な機能は、C 言語で記述された AWS IoT Software Development Kit (SDK)で提供され、センサ、アクチュエータ、組み込みマイクロコントローラ、またはスマート家電などのデバイスは、MQTT、HTTP、または WebSocket のプロトコルを使用して AWS IoT と接続、認証、およびメッセージ交換を行うことができるようになります。本アプリケーションノートでは、AWS IoT Device SDK と、Renesas RA MCU 用の Renesas Flexible Software Package (FSP)から利用可能な付属の MQTT プロトコルの設定と使用方法について説明します。

1.3　クラウドダッシュボード

クラウドダッシュボードは、複数のサービスを監視、制御する GUI であり、Web ブラウザで構築してアクセスすることができます。導入が容易で、IT サポートをほとんど必要とせず、複数のデバイスからアクセスできるなど、オンプレミスソフトウェアより利点があります。

ダッシュボードは、デバイスのグループ全体の概要が表示され、個々のデバイスをすばやく操作できます。デバイスの所有タイプ、コンプライアンス統計、プラットフォームと OS の内訳など、保有するデバイスの関連情報をグラフィカルに表示することができます。デバイスダッシュボードから利用可能なデータビューのいずれかを選択することで、提示されたカテゴリの各デバイスセットにアクセスすることができます。

1.3.1　データモニタリング

ダッシュボードでのデータモニタリングは、パフォーマンスメトリクスを追跡し、データセットを簡単に視覚化できるクラウドデータ分析モニタリングソリューションです。メトリクスの高度なビューを取得することも、掘り下げて詳細を分析することも可能です。

たとえば、温度、圧力などの形式でデバイスから送られてくるセンサデータなどがあります。

1.3.2　デバイス管理

デバイス管理は、デバイスのグループ全体を一括設定や個々のデバイスを制御するための高度な制御を備えています。

注：　このアプリケーションプロジェクトに関するダッシュボード固有の詳細については、このドキュメントの後半のセクションで説明します。

1.4　AWS IoT Core

AWS IoT Core は、接続されたデバイスがクラウドアプリケーションやその他のデバイスと簡単かつ安全にやり取りできるようにするマネージドクラウドサービスです。AWS IoT Core は数十億のデバイスと膨大な数のメッセージをサポートできます。AWS のエンドポイントや他のデバイスへのメッセージを確実かつ安全に処理し、ルーティングすることもできるので、お客様のアプリケーションは、接続されていない場合でも、すべてのデバイスを常に追跡できます。

AWS IoT Core は、相互認証と暗号化を実装することで、インフラストラクチャのセキュリティの問題に対応しています。AWS IoT Core は、デバイスの AWS IoT Core への初回接続時に自動設定と認証を行い、すべての接続ポイントでエンドツーエンドの暗号化を行うため、デバイスと AWS IoT Core の間で証明された ID なしにデータが交換されることはありません。

本アプリケーションノートでは、オンボードフラッシュに Privacy Enhanced Mail (PEM)形式で x.509 証明書と非対称暗号化鍵を保存する方式で、RA MCU の証明済みの識別子をインストールすることにより、AWS IoT Core のセキュリティニーズを補完することに焦点を当てています。RA MCU は、キーラッピングなどのオンチップセキュリティ機能を備え、公開鍵とデバイスに関連する証明書に関連する秘密鍵を保護します [1]。さらに、RA MCU は、FSP を通じて利用可能なセキュア暗号エンジン(SCE)と API の機能を使用して、非対称鍵を生成することもできます。SCE は、接続デバイスと AWS IoT 間のデータの対称暗号/復号を高速化し、ARM Cortex-M プロセッサが他のアプリケーションの処理を実行できるようにします。

1.5　MQTT プロトコルの概要

本アプリケーションノートでは、Message Queuing Telemetry Transport (MQTT)を取り上げます。MQTT は、断続的な接続を許容し、デバイスのコードフットプリントを最小限に抑え、ネットワーク帯域幅の要件を軽減するように特別に設計された軽量の通信プロトコルです。MQTT は、オープンで簡単に実装できるように設計されたパブリッシュ/サブスクライブ通信モデルを使用しており、1 つのサーバで最大数千のリモートクライアントをサポートすることが可能です。これらの特性により、MQTT は、ネットワーク帯域幅が狭い、またはレイテンシが高い、処理能力やメモリに制限のあるリモートデバイスを使用する環境での使用に適しています。本アプリケーションノートの RA MCU 搭載デバイスは、AWS IoT と通信し、温度、圧力、湿度、加速度計、磁力計、その他多くの種類のセンサデータなどのサンプルテレメトリ情報を交換する Core MQTT を実装しています。

1.6　TLS プロトコルの概要

Transport Layer Security (TLS)プロトコルの主な目的は、通信する 2 つのアプリケーションまたはエンドポイント間での秘匿性とデータの整合性の規定です。AWS IoT は、安全な通信の使用を義務付けています。その結果、AWS IoT との間のすべてのトラフィックは、TLS を使用して安全に送信されます。TLS プロトコルバージョン 1.2 以降は、AWS IoT でサポートされるアプリケーションプロトコルの機密性を保証します。様々な TLS Cipher Suite がサポートされています。本アプリケーションノートでは、MCU ベースのデバイスに RA Flexible Software Package を設定して以下の機能を提供し、AWS IoT が適切な TLS Cipher Suite 構成を組合せてセキュリティを最大化します。

表 1　RA FSP における TLS の機能

セキュア暗号ハードウェアアクセラレーション	サポート
サポートされるキーフォーマット	AES, ECC, RSA
ハッシュ	SHA-256
暗号	AES
公開鍵暗号方式	ECC, ECDSA, RSA
メッセージ認証コード(MAC)	HKDF

[1] 本アプリケーションノートでは、製品リリースされたデバイスの秘密鍵を安全に保存するためのキーラッピングを使用することに焦点を当てていません。

これらのサポートされる機能に加え、Mbed Crypto ミドルウェアは、RA コンフィギュレータから有効にできる様々な機能をサポートしています。Crypto Middleware (rm_psa_crypto)については、FSP ユーザーズマニュアルのセクションを参照してください。

1.7　デバイス証明書、CA、および鍵

デバイス証明書、認証局(CA)、非対称鍵ペアは、安全な環境に必要な信頼の土台を作ります。AWS でー般的に使用されるコンポーネントの背景情報は次のとおりです。

デジタル証明書は、デバイスの身元に関する情報を提供する、既知の形式の文書です。X.509 規格では、公開鍵証明書、属性証明書、証明書失効リスト(CRL)、属性証明書失効リスト(ACRL)のフォーマット定義が含まれています。X.509 で定義された証明書フォーマット(X.509 証明書)は、インターネットや AWS IoT において、リモートエンティティ/エンドポイント、つまりクライアントやサーバを認証するために一般的に使用されています。本アプリケーションノートでは、X.509 証明書と非対称暗号鍵ペア(公開キーと秘密鍵)を AWS IoT から生成され、(バイナリコンパイル時に) Core MQTT を実行している RA MCU デバイスにインストールして既知の ID を確立しています。さらに、ルート認証局(CA)証明書もダウンロードされ、AWS IoT ゲートウェイへの接続を認証するためにデバイスにて使用されます。

認証局(CA)証明書は、2 つの CA[2] 間で定義された関係を作成する目的で、CA が自身または第 2 の CA によって発行する証明書です。ルート CA 証明書は、デバイスが AWS IoT Core と通信しており、AWS IoT Core になりすました別のサーバではないことを確認できます。

AWS IoT からダウンロードした公開鍵と秘密鍵は、暗号化、復号化、署名、検証のために RSA アルゴリズムを使用します [3]。これらの鍵ペア、および証明書は、TLS プロセスで一緒に使用されます。

1. デバイスの識別を確認します。
2. エンドポイント間のデータ転送を暗号化および復号化するための、AES などのアルゴリズムの対称鍵を交換します。

[2] AWS IoT が提供するルート CA 証明書は、Digital Guardian によって署名されています。
[3] 使用される公開鍵の長さは 2048 ビットです。

2.　MQTT/TLS セルラーアプリケーション例の実行

2.1.1　プロジェクトのインポート、ビルド、ロード

2.1.1.1　インポート

このプロジェクトは、RA FSP ユーザーズマニュアルに記載されている手順を使用して e² studio にインポートすることができます。Starting Development > e2 studio User Guide > Importing an Existing Project into e2 studio を参照してください。

2.1.1.2　最新の実行バイナリのビルド

プロジェクトを e² studio IDE にインポートまたは変更したら、FSP ユーザーズマニュアルに記載されている手順に従って、実行可能なバイナリ/hex/mot/elf ファイルを作成してください。Starting Development > Tutorial: Your First RA MCU Project - Blinky > Build the Blinky Project を参照してください。

2.1.2　ターゲット MCU への実行バイナリの書き込み

実行ファイルは、3 つの方法のいずれかによってターゲット MCU にプログラムすることができます。

2.1.2.1　e² studio のデバッグインタフェースを使用する場合

実行バイナリを書き込む手順は、最新の RA FSP ユーザーズマニュアルに記載されています。Starting Development > Tutorial: Your First RA MCU Project - Blinky > Debug the Blinky Project のセクションを参照してください。

この方法は、オンチップデバッガを介して追加のデバッグ機能を使用できるため、書き込みに推奨されます。

ボードの書き込みの説明に従って、セクション 2.1.3 へ進んでください。

2.1.2.2　J-Link ツールを使用する場合

SEGGER J-Link ツール(J-Flash、J-Flash Lite、J-Link Commander)を使用して、ターゲット MCU に実行可能なバイナリを書き込むことができます。www.segger.com のユーザーズマニュアル UM08001 および UM08003 を参照してください。アプリケーションプロジェクトの `.srec` または `.hex` ファイルを使用してボードに書き込み、セクション 2.1.3 に進んでください。

2.1.2.3　Renesas Flash Programmer を使用する場合

Renesas Flash Programmer は、Renesas マイクロコントローラのオンチップフラッシュメモリの書き込みを、開発・量産の各段階において使いやすく、機能的にサポートします。アプリケーションプロジェクトの `.srec` または `.hex` ファイルを使用してボードに書き込み、セクション 2.1.3 に進んでください。

2.1.3　デフォルトのジャンパ設定にリセット

次の一連の手順に進む前に、このアプリケーションノートに関連するボード上のジャンパを CK-RA5M5 ユーザーズマニュアルで指定されているデフォルトのジャンパ設定にリセットします。

2.1.4　ボードの電源投入

ボードへの電源は、USB ケーブルを CK-RA6M5 ボードの J14 コネクタ(USB_DEBUG)に接続し、もう一方を PC の USB ポートに接続します。2 本目の USB ケーブルを CK-RA6M5 ボードの J20 コネクタに接続し、もう一方を PC の 2 番目の USB ポートに接続します(これがアプリケーションのコンソールポートになります)。アプリケーションの設定と実行には、コマンドラインインタフェース(CLI)を使用する必要があります。

次に、以降の手順でデバッグアプリケーションを実行します。

2.1.4.1　電源投入時の動作

設定とターゲット RA MCU へのイメージのダウンロードが成功すると、電源投入時 renesas.com/ra/ck-ra6m5 のウェブサイトにあるクイックスタートサンプルプロジェクトに示されるように、次の動作が確認できます。

1. RA MCU ターゲットボード上の電源 LED が点灯します。
2. J-Link LED は接続時の動きに連動して点滅します。
3. ユーザ LED (青、緑、赤)は、アプリケーションの初期化開始から実行中のステータスを示すために使用されます。

2.1.5　アプリケーションプロジェクトの実行

注：　以下に示す手順は、Window OS にのみ適用されます。プロジェクトは MacOS や Linux ではサポートされていません。

アプリケーションプロジェクトを実行するには、ユーザが次のことを行う必要があります。

- IMEI と ICCID の情報を使って、Launch Pad ポータルでモデムと SIM をアクティベーションします。
- https://renesas.cloud-ra-rx.com/login から Renesas AWS ダッシュボードにサインインして、デバイスの自動的なプロビジョニング、デバイス接続に必要な資格情報の生成を行います。

注：　この情報を取得するための詳細は、以降の手順で説明されています。

2.1.5.1　PC のシリアルポートコンソールへのボードの接続

1. ホスト PC で、Windows デバイスマネージャを開きます。[ポート(COM と LPT)]を展開し、[USB シリアルデバイス(COMxx)]を見つけて、次の手順で参照するための COM ポート番号を書き留めてください。
   注：　CK-RA6M5 ボードとホスト PC 上のターミナルアプリケーションとの通信には、USB Serial Device ドライバが必要です。

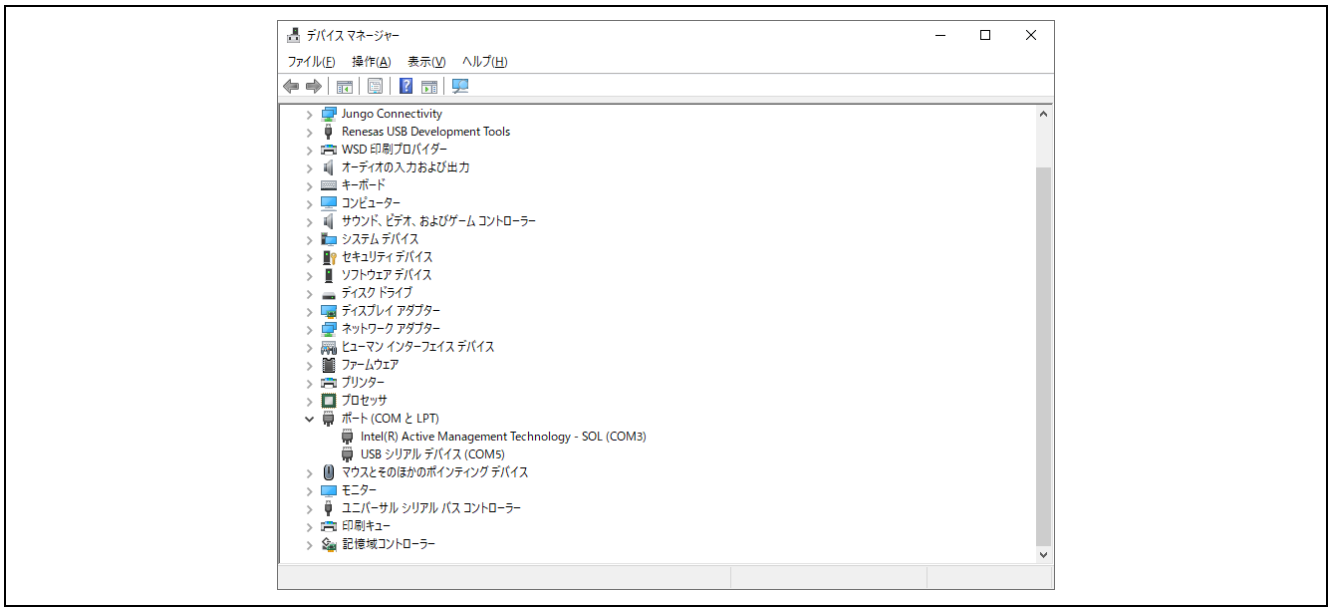

図 2　Windows デバイスマネージャの USB シリアルデバイス

2. Tera Term を起動し[新しい接続]を選択し、[シリアル]と[COMxx: USB シリアルデバイス(COMxx)]を選択し、[OK]をクリックします。

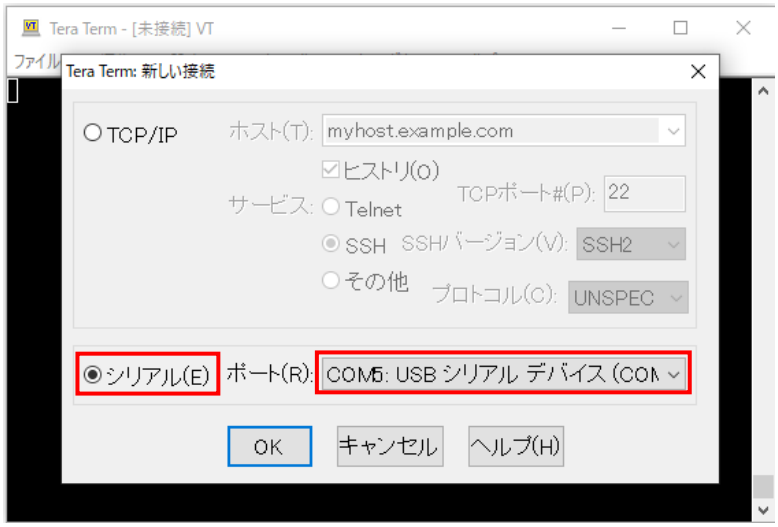

図 3　Tera Term シリアルポート選択

3. [設定]メニューのプルダウンから、[シリアルポート]を選択し、以下のように[ボーレート]を[115200]に設定します。

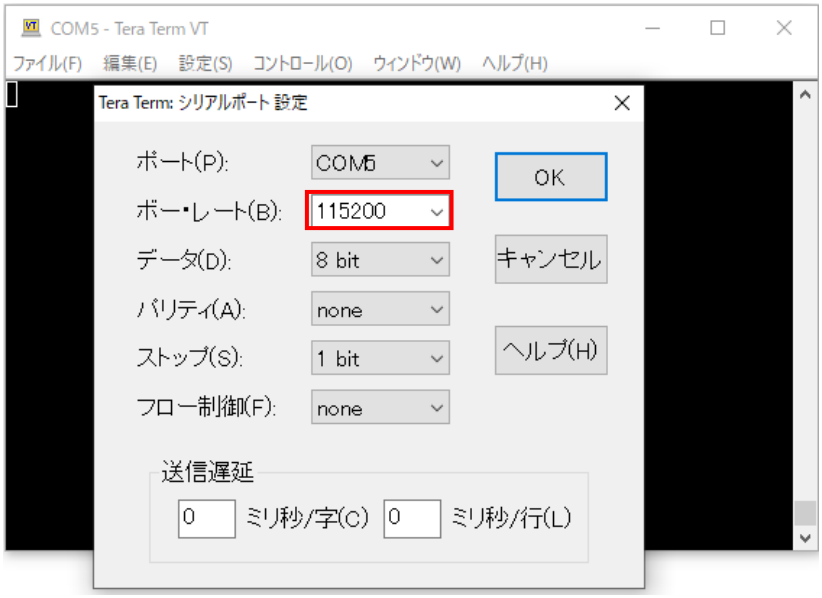

図 4　ボーレート プルダウンで 115200 を選択

4. 接続が完了します。コンソールには、以下に示すように設定 CLI メニューが表示されます。
   注：　メニューが表示されない場合は、S1 ユーザスイッチを押してボードをリセットしてください。

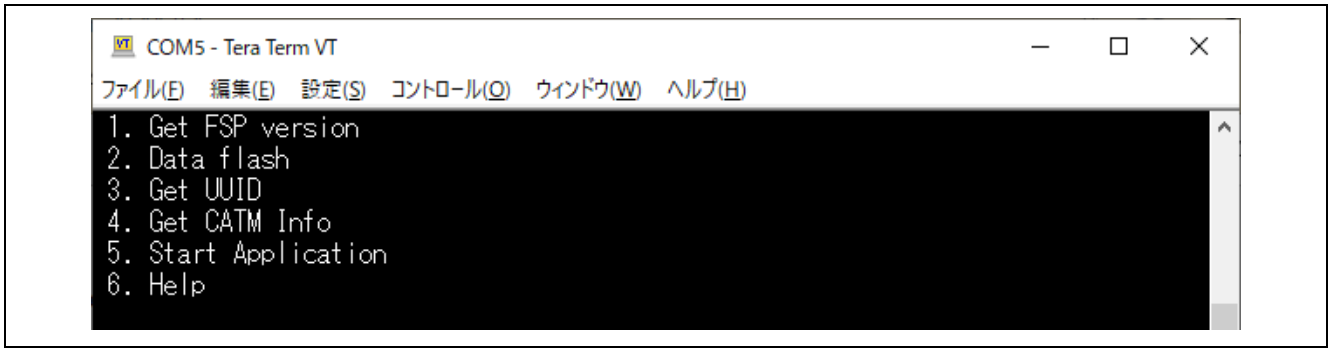

図 5　メインメニュー

5. 上のスクリーンショットに示された CLI で、コマンドを選択するための番号を入力します。例えば、[1]を押すと、下図のように FSP アプリケーションのバージョンが表示されます。[スペースキー]を押すと、いつでも前のメニューに戻ることができます。

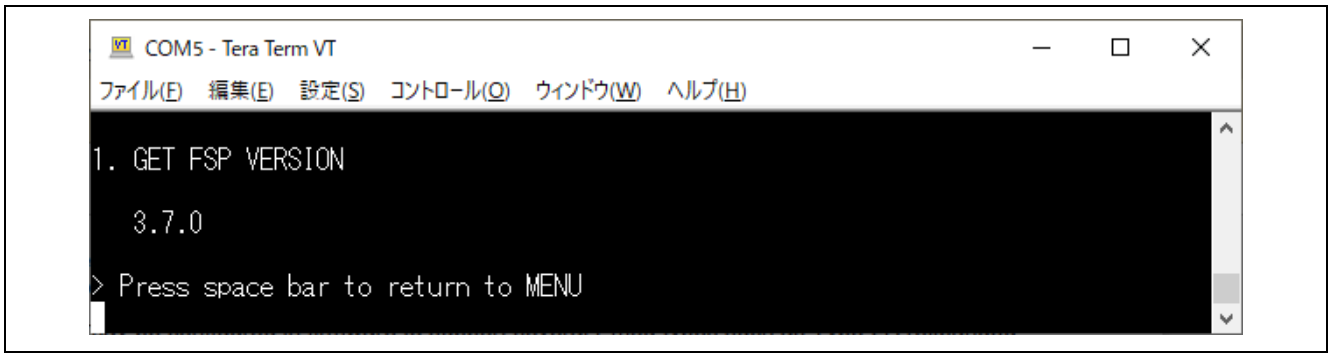

図 6　FSP バージョン情報

2.1.5.2　アクティベーションのための SIM とモデムの情報取得

1. [4]を押して、[CAT-M Information]を表示します。このメニューは、CAT-M モジュールと通信し、SIM カードのアクティベーションに必要な IMEI および ICCID の値を取得します。成功すると、IMEI と ICCID の値が端末の画面に表示されます。プログラムは、正常に接続、またはタイムアウトするまで、CAT-M モジュールとの通信を試みます。IMEI と ICCID の値を取得したら、renesas.micro.ai、MicroAI Launchpad にアクセスし、SIM カードをアクティベーションしてください。

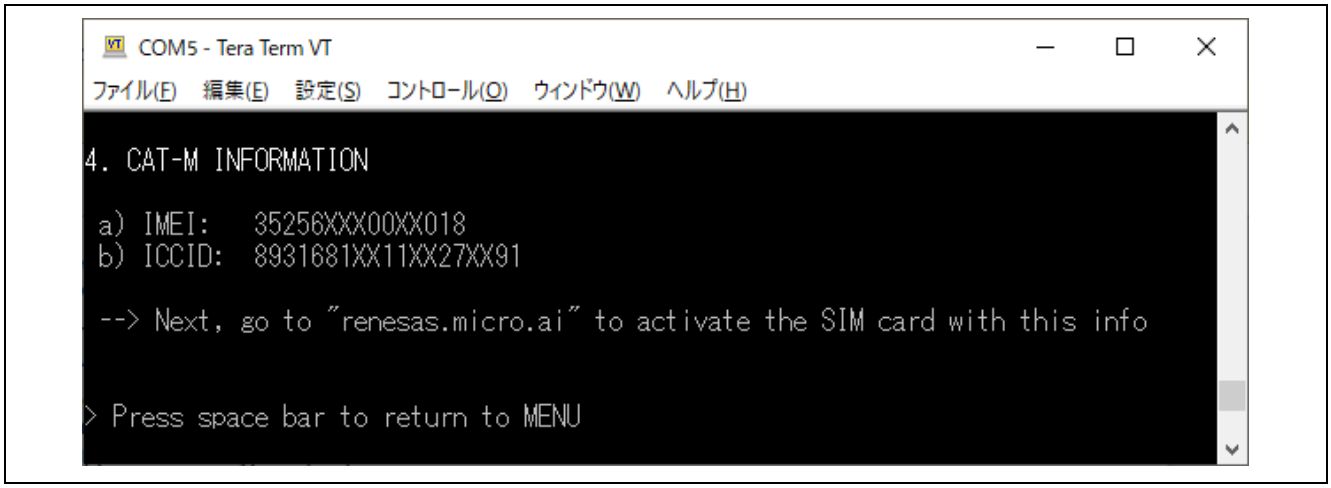

図 7　CAT-M 情報

2.1.5.3　MicroAI Launchpad での SIM カードのアクティベーション

SIM カードのアクティベーションには、MicroAI Launchpad プラットフォームが必要です。SIM カードをアクティベーションするには次の手順を参照してください。

1. Launchpad のアカウントを作成します。サインアップページで登録し、登録後に送信される認証メールでアカウントを確認することで行ってください。
2. Launchpad アカウントにログインし、[Create Device Profile]タイルをクリックします。[Device Type]に[Renesas EK-RA6M5]が、[Connectivity Type]に[CAT-M]を選択します。必要に応じて他のフィールドを入力し、[Next]をクリックします。

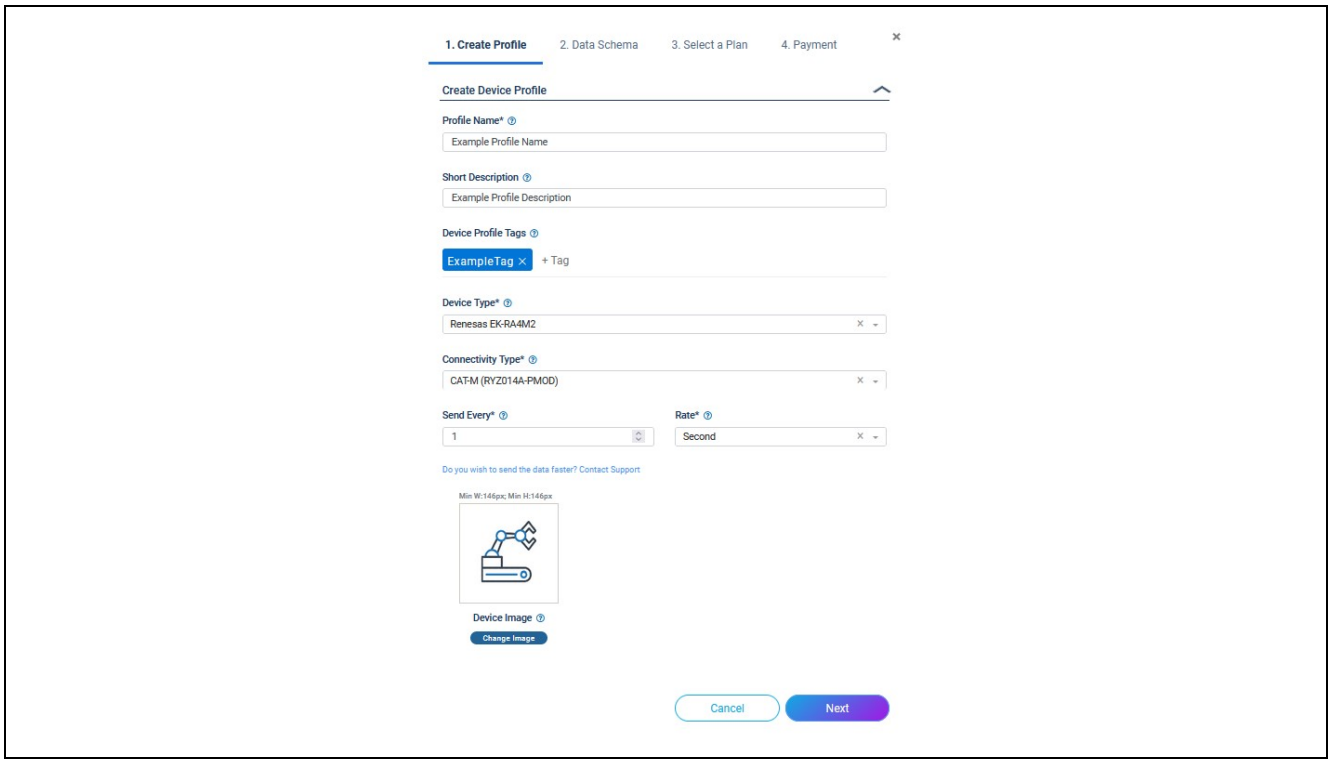

図 8　MicroAI Launchpad でのデバイスプロファイル作成

3. [Data Schema]で、[Next]をクリックします。
4. [Select a Plan]で有効なプロモーションコードを適用し、[Next]をクリックします。
5. [Payment Method]を指定し、利用規約に同意します。[Finish]をクリックします。
6. [Register a Device]タイルをクリックし、CK-RA6M5 キットと SIM カードの登録を開始します。[device profile]のドロップダウンから、作成したデバイスプロファイルを選択します。CLI から取得した IMEI 値を入力します。必要に応じて他のフィールドを入力します。

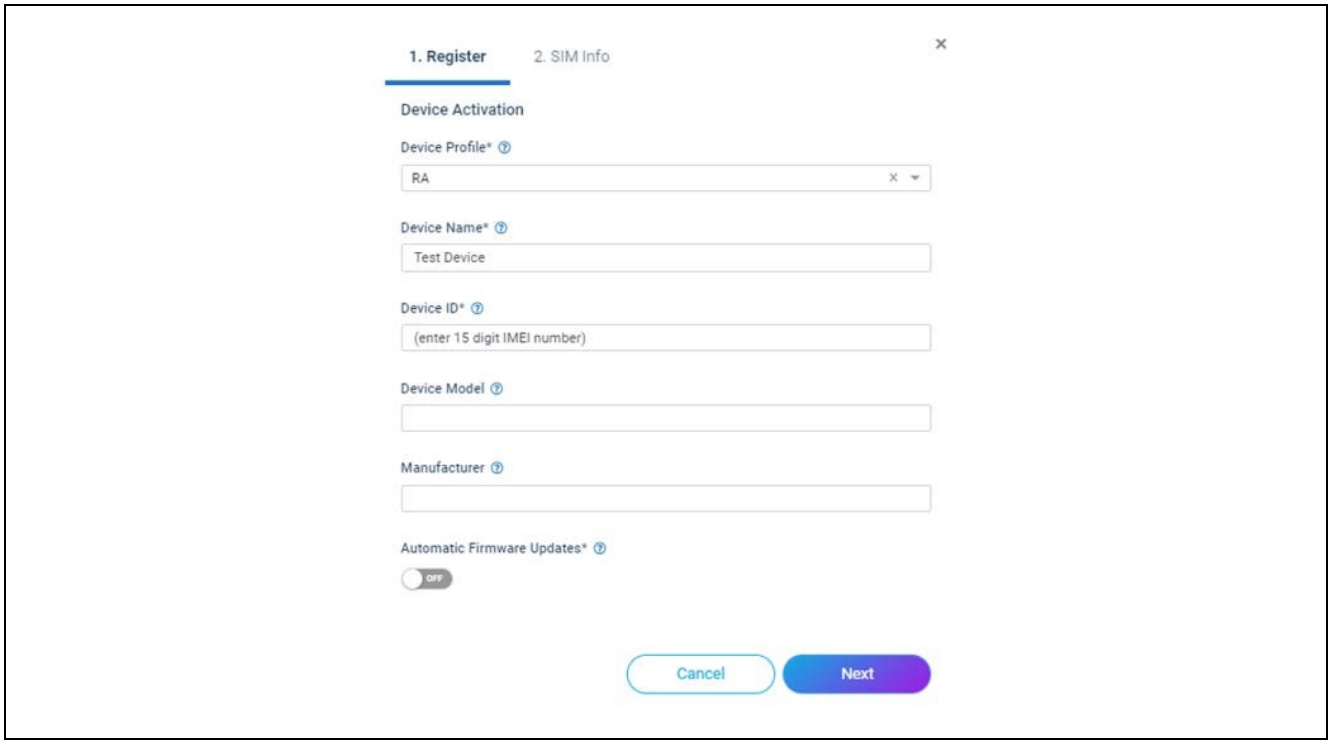

図 9　MicroAI Launchpad での EK-RA6M5 の登録

7. [Next]をクリックし、[Add New SIM]をクリックします。[SIM EID]フィールドに ICCID 値を入力し、[Save SIM]をクリックします。

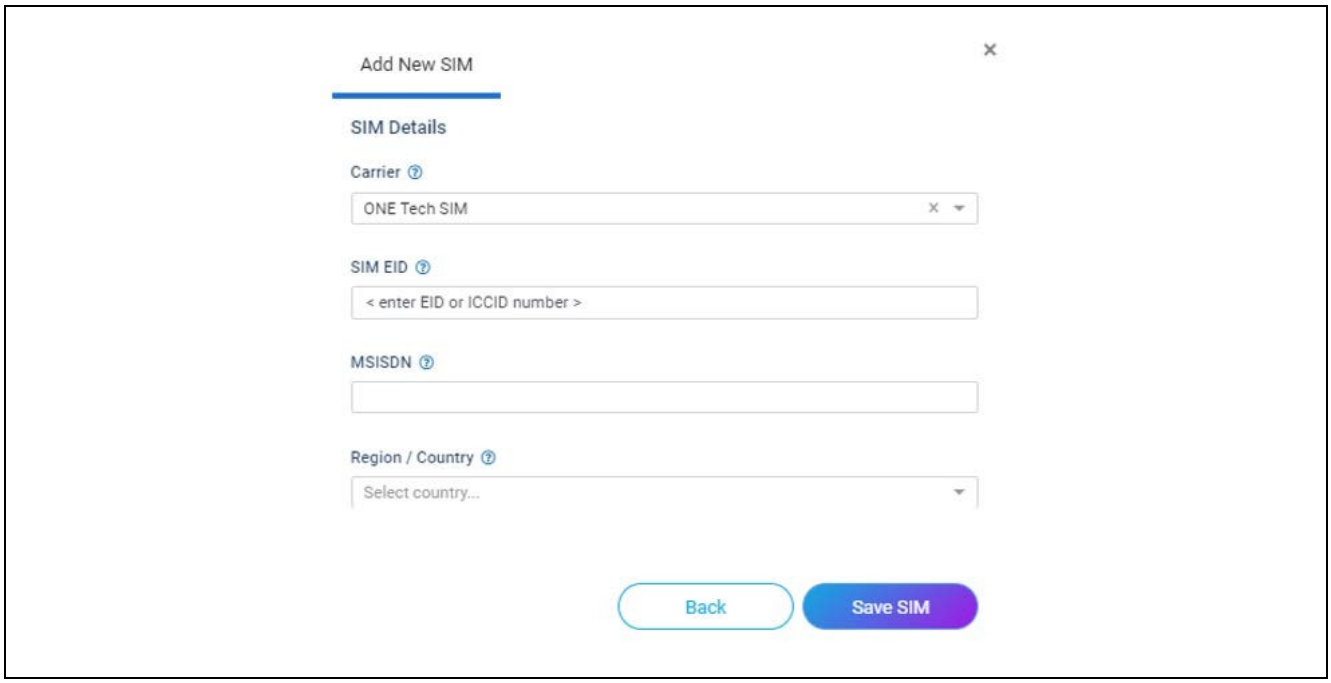

図 10　MicroAI Launchpad での SIM カードアクティベーション

8. デバイスの登録が完了します。CK-RA6M5 キットと SIM カードは、Launchpad プラットフォームによりアクティベーションされており、Tera Term のターミナルで検証できます。

2.1.5.4　ボードの UUID 情報を取得する

1. [メインメニュー]で[3]を押すと、ボードの UUID が表示されます。このコマンドはボードの UUID 情報を取得し、以下のスクリーンショットのようにコンソールに表示します。この情報は、Renesas AWS Cloud Dashboard に登録する際に必要となります。

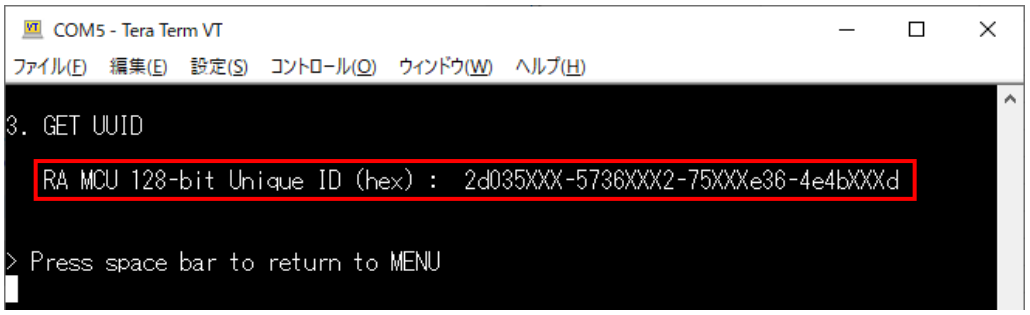

図 11　ボード UUID 情報取得

2.1.5.5　Renesas AWS クラウドダッシュボードへの登録

Renesas AWS クラウドダッシュボードにアクセスするには、キットを renesas.cloud-ra-rx.com で登録する必要があります。*AWS Dashboard for CK-RA6M5 and CK-RX65N Application Note* の AWS ダッシュボードの手順に従って、デバイスをプロビジョニングし、Renesas AWS ダッシュボードにアクセスできるようにします。

登録後、次のステップに進む前に、プロビジョニングプロセスが完了するまで 1 時間ほど待ちます。

2.1.5.6　デバイス証明書、キー、MQTT ブローカーのエンドポイント、モノの名前の保存

アプリケーションを動作させるには、デバイス証明書、デバイスの秘密鍵、MQTT ブローカーのエンドポイント、およびモノの名前をデータフラッシュに保存しておく必要があります。これらは、クラウドダッシュボードに登録した後に取得します。

1. *AWS Dashboard for CK-RA6M5 and CK-RX65N Application Note* の AWS ダッシュボードのセクション 1 の手順 7 で説明されているように、ダッシュボードアカウントページから[certs.zip]をダウンロードします。
2. [メインメニュー]で[2]を押すと、次のスクリーンショットに示すように、[Data Flash]関連のコマンドが表示されます。このサブメニューにはデータの保存、読み出し、検証のコマンドがあります。

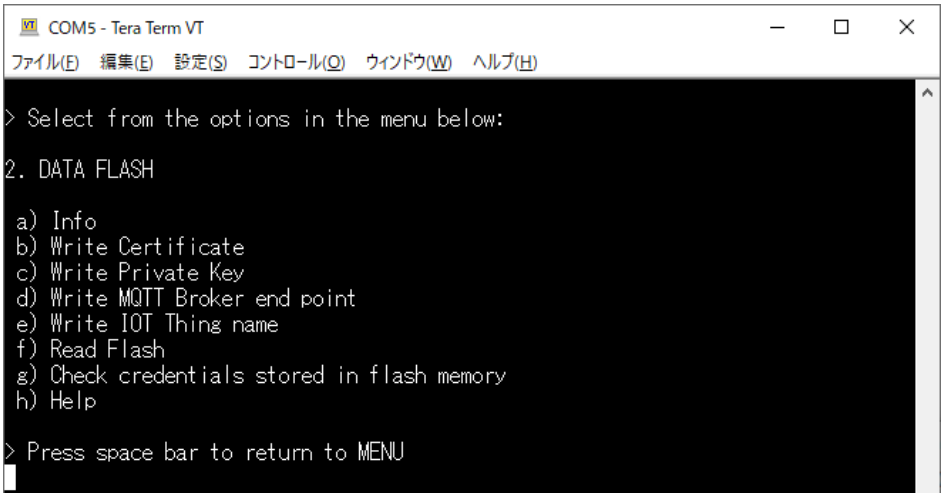

図 12　Data Flash 関連のメニューとコマンド

3. デバイス証明書を保存するために、オプション[b]を押して、Tera Term の[ファイル]タブをクリックします。[ファイル送信]オプションを選択し、手順 1 でダウンロードした[certs.zip]ファイルからデバイス証明書ファイル‘xxxxxcertificate.pem.crt’を選択します。

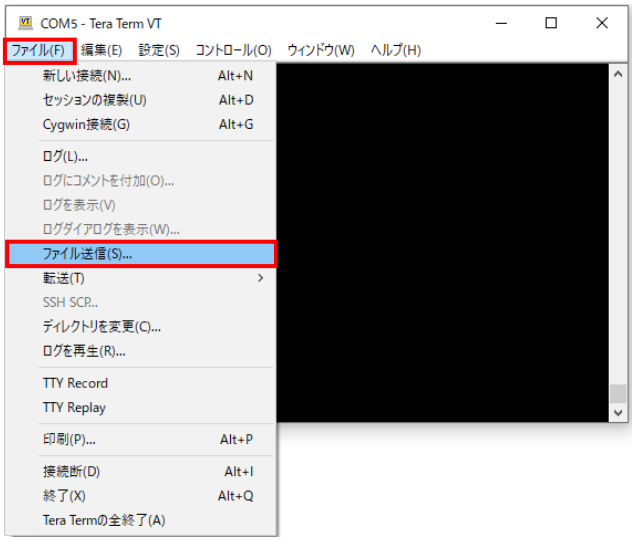

図 13　デバイス証明書へのアクセス

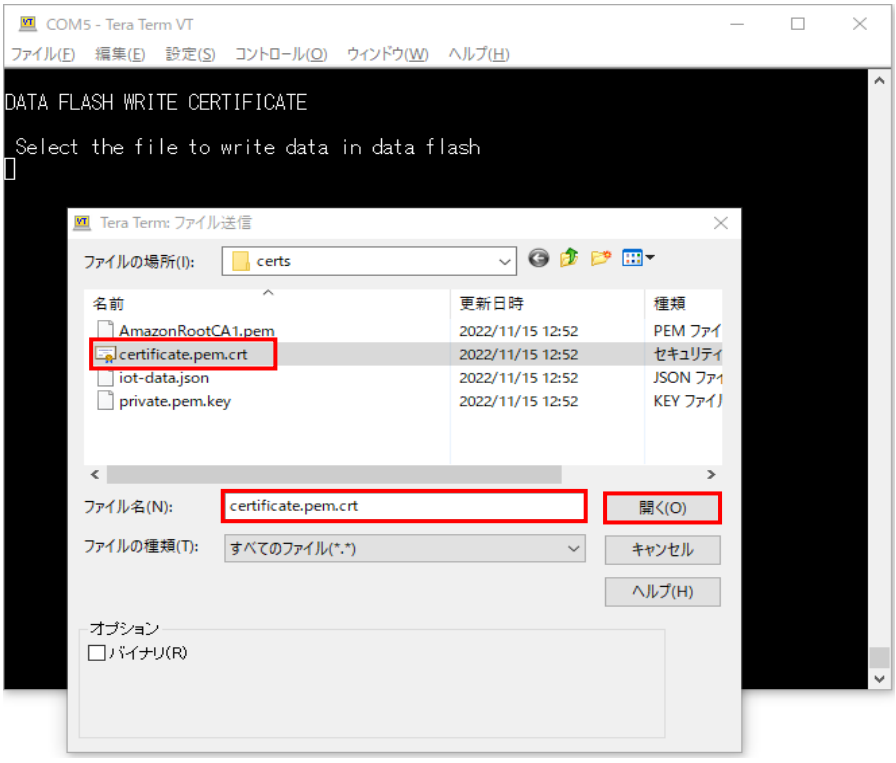

図 14　データフラッシュへのデバイス証明書のダウンロード

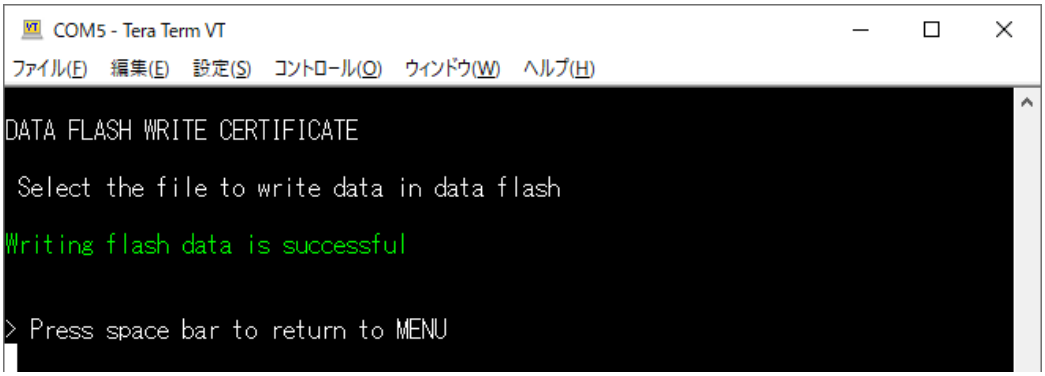

図 15　データフラッシュにダウンロードしたデバイス証明書のステータス

4. デバイスキーを保存するには、オプション[c]を押して、Tera Term の[ファイル]タブをクリックします。[ファイル送信]オプションを選択し、手順 1 でダウンロードした[certs.zip]ファイルからデバイスキー'xxxxxxxprivate.pem.key'を選択してください。
   MQTT ブローカーのエンドポイントを保存するには、手順 1 でダウンロードした[certs.zip]ファイルの[iot-data.json]ファイルから、エンドポイント文字列 xxxxxxxxxxxxx-ats.iot.us-east1.amazonaws.com をコピーします。オプション[d]を押して、Tera Term の[編集]タブをクリックし、[貼り付け<CR>]を実行します。有効な文字列であることを確認して、[OK]を押します。
   注：　MQTT ブローカーのエンドポイントをコピーする際に、ダブルクォーテーションをコピー**しない**でください。

図 16　データフラッシュへの MQTT エンドポイントの保存

5. モノの名前を保存するには、手順 1 でダウンロードした[certs.zip]ファイルの[iot-data.json]ファイルからモノの名前の文字列 xxxxxxxx-5736xxxx-xxxxxxxx-4e4bxxxx をコピーします。オプション[e]を押して、Tera Term の[編集]タブをクリックし、[貼り付け<CR>]を実行します。有効な文字列であることを確認し、[OK]を押します。
   注：　モノの名前をコピーする際に、ダブルクォーテーションを**コピーしない**でください。

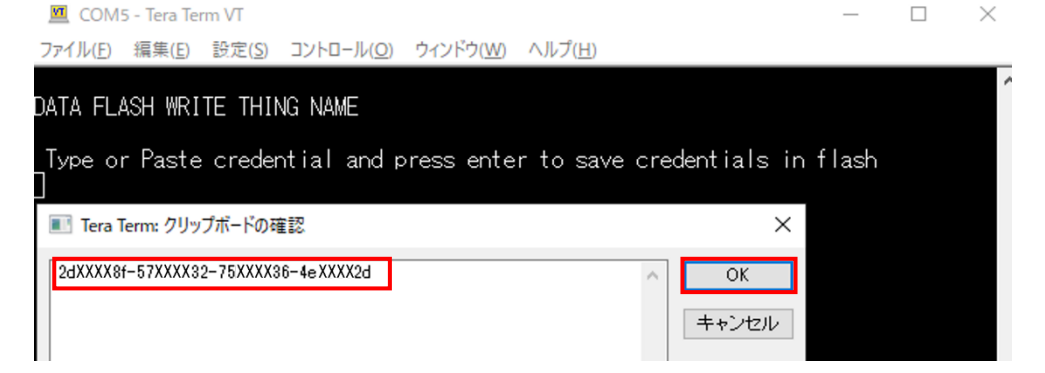

図 17　データフラッシュへのモノの名前の保存

6. オプション[f]と[g]を押して、データフラッシュに保存された情報を読み出し、検証します。[スペースキー]を押して、前のメニューに遷移します。

注：　保存データの検証は非常に限定的で、最小限のデータの組み合わせを検証します。ユーザは、アプリケーションを正しく動作させるために、ダッシュボードから取得した有効なデータをフラッシュに入力する必要があります。

2.2　IoT クラウドの設定と AWS IoT への接続

2.2.1　AWS IoT の利用開始と Renesas クラウドへの登録

Renesas AWS のダッシュボード(https://renesas.cloud-ra-rx.com/login)に、AWS アカウントにサインアップしていないメールアカウントを使用してサインインしてください。

注：　ダッシュボードでは、メールアドレスにリンクした新しい AWS アカウントが作成されるため、AWS アカウントを開設するのに使用していないメールアドレスでサインアップすることが重要です。

ダッシュボードアカウントのステータスに、アカウントのプロビジョニングの完了が表示されるのを待ちます。

サインアップに使用したメールアカウントに、AWS から認証が送信されます。検証後、ユーザはダッシュボードにアクセスし、ダッシュボードから証明書をダウンロードすることができるようになります。

2.3　アプリケーションの起動

SIM カードのアクティベーション、ダッシュボードへの登録、CLI による必要なクラウド認証情報の設定が完了すると、アプリケーションを実行する準備が整います。オプション[5]を押して、アプリケーションを起動します。アプリケーションは、以下のようにデータフラッシュに存在するクラウド認証情報の検証状況とともに、ウェルカムスクリーンを表示します。

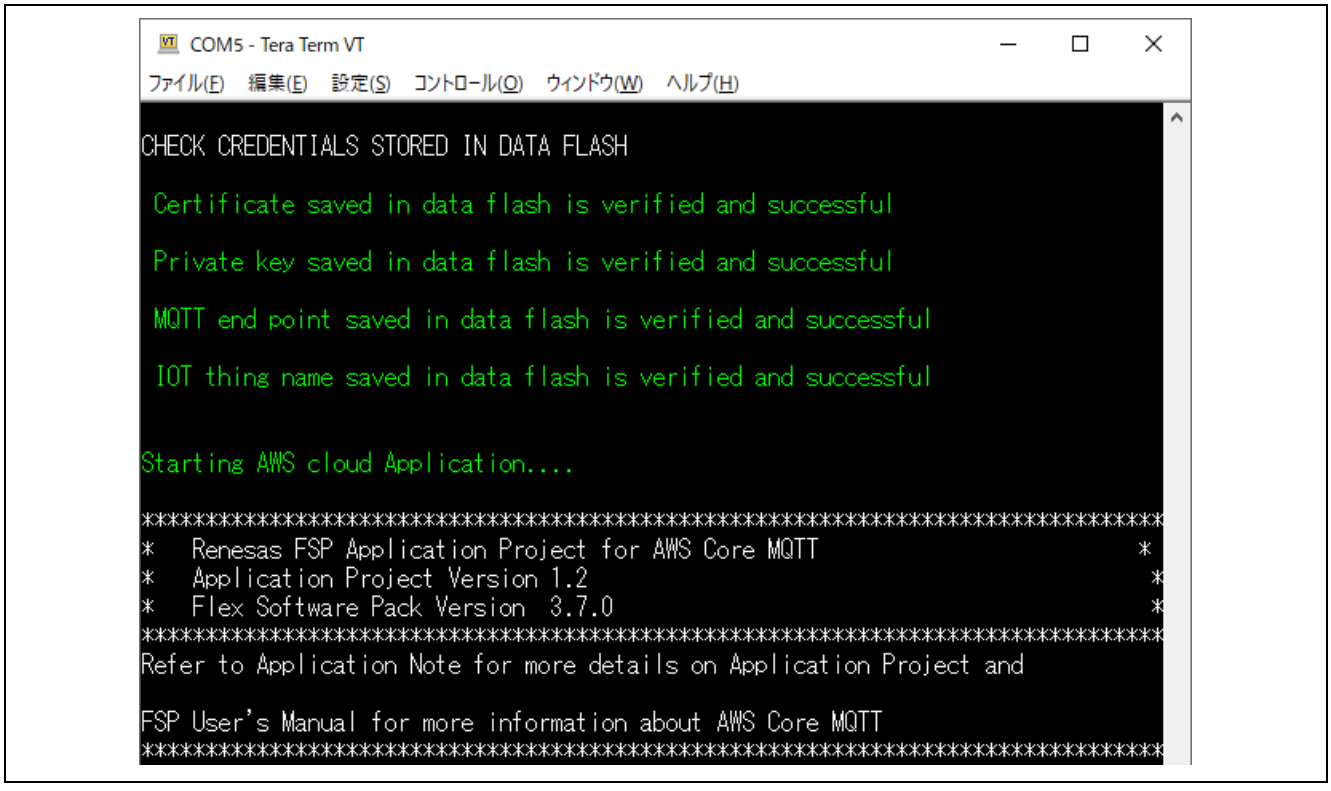

図 18　コンソールのウェルカムスクリーン

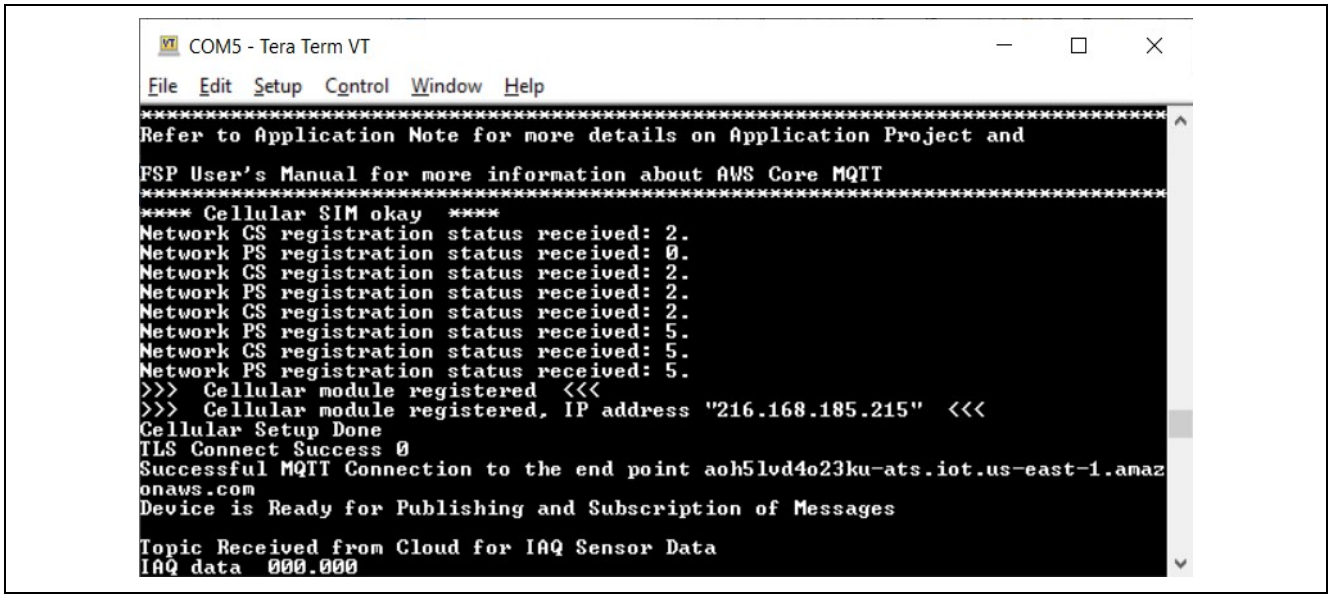

図 19　ネットワークと AWS IoT に接続中のコンソール

2.4　AWS IoT MQTT テストクライアントを使ったアプリケーションプロジェクトの検証

このセクションでは、サンプルアプリケーションの機能を検証する手順を説明します。

注：　Cellular の初期化に成功し、サービスプロバイダからの IP アドレスの取得と、エンドポイントの DNS ルックアップの完了を待ちます。コンソールに成功メッセージ[Successful MQTT Connection to the end point]が表示された後、デバイスはメッセージのパブリッシュとサブスクライブの準備が完了します。

注：　本アプリケーションは、AWS MQTT IOT Core を使用します。ユーザは、AWS IOT MQTT Test Client を使用して検証を行うことができます。また、ルネサスの GUI ベースのダッシュボードを使用して、すべてのセンサデータをカスタマイズして表示することもできます。

検証のために、ユーザは AWS IOT MQTT Test Client を使用して、以下のセクションで説明するようにトピックの購読と公開を設定し、制御することができます。

AWS クラウドダッシュボード側の AWS IoT Console にアクセスして[Test]を選択し、[MQTT test client]を選択します。以下のリストにあるトピックを 1 つずつサブスクライブします。トピックをサブスクライブするサンプルスクリーンショットは以下に示します。

注：　以下のメッセージは、大文字と小文字が区別されます。ユーザはパブリッシュやサブスクライブのメッセージを入力する際には、注意する必要があります。
一度に入力するメッセージは 1 つだけにしてください。メッセージはダブルクォーテーションで囲んで”そのまま”コピーし、余分なスペースを含めないでください。

```
"aws/topic/iaq_sensor_data"
"aws/topic/oaq_sensor_data"
"aws/topic/hs3001_sensor_data"
"aws/topic/icm_sensor_data"
"aws/topic/icp_sensor_data"
"aws/topic/ob1203_sensor_data"
"aws/topic/bulk_sensor_data"
```

注：　トピックのサブスクライブ後に、ダッシュボードはデバイスからパブリッシュされるメッセージを受信できるようになります。

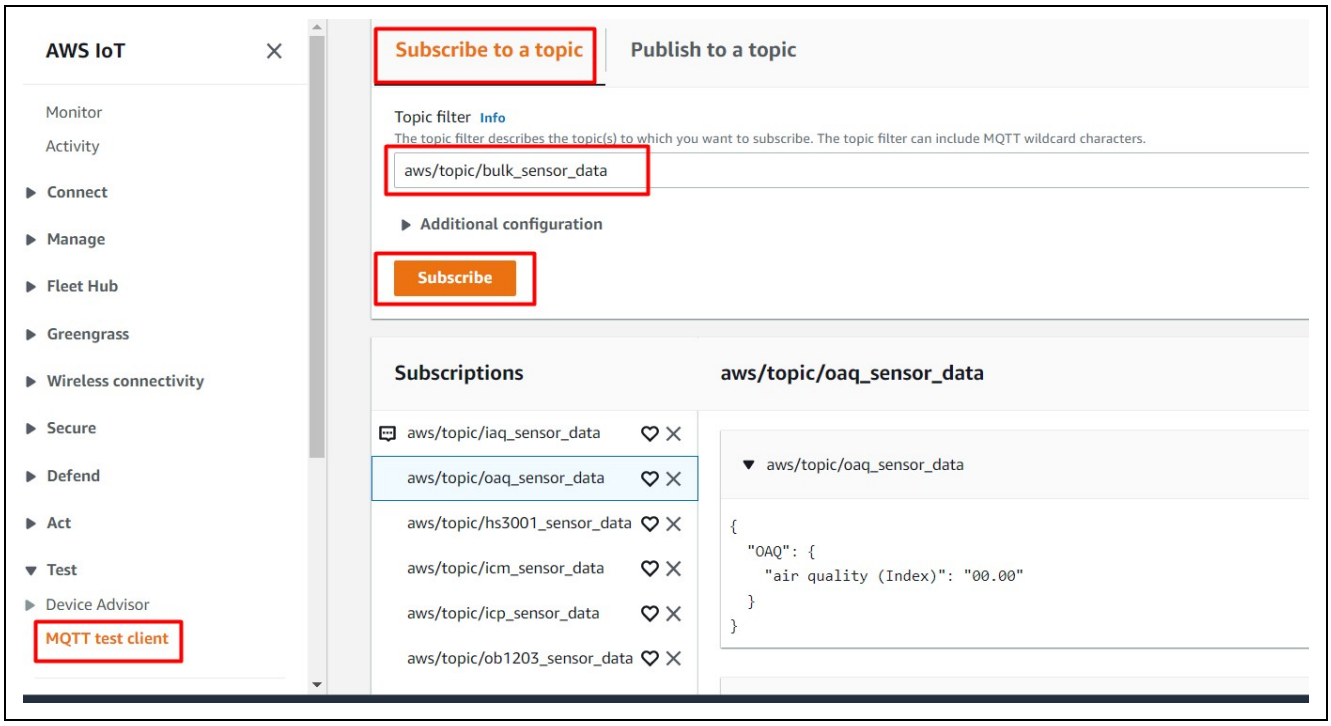

図 20　AWS IoT 画面でのトピックメッセージのサブスクライブ

図 21　AWS IoT 画面でのサブスクライブされたメッセージ

デバイスは、同期またはイベント発生時にメッセージを公開することができます。クラウド側では、メッセージの購読に基づいて、デバイスがパブリッシュしたメッセージを AWS IOT の画面上でサブスクライブメッセージとして確認することができます。

同様に、デバイスがメッセージを受信するには、それらのトピックをサブスクライブする必要があります。デバイスがこれらのトピックをサブスクライブすると、クラウド側のパブリッシュされたメッセージはすべてデバイス側で確認できます。

このアプリケーションでは、センサデータの要求(aws/topic/get_iaq_sensor_data)がクラウドからのパブリッシュアクティビティです。デバイスからの実際のセンサデータは、リクエストに対する応答です。(aws/topic/iaq_sensor_data)

センサデータを要求するには、[Publish to a topic]を選択し、以下のリストにあるトピックを入力します。例えば以下に示すように、ウィンドウに(aws/topic/get_iaq_sensor_data)と入力し、[Publish]をクリックしてください。

```
"aws/topic/get_iaq_sensor_data"
"aws/topic/get_oaq_sensor_data"
"aws/topic/get_hs3001_sensor_data"
"aws/topic/get_icm_sensor_data"
"aws/topic/get_icp_sensor_data"
"aws/topic/get_ob1203_sensor_data"
"aws/topic/get_bulk_sensor_data"
"aws/topic/set_temperature_led_data"
"aws/topic/set_spo2_led_data"
```

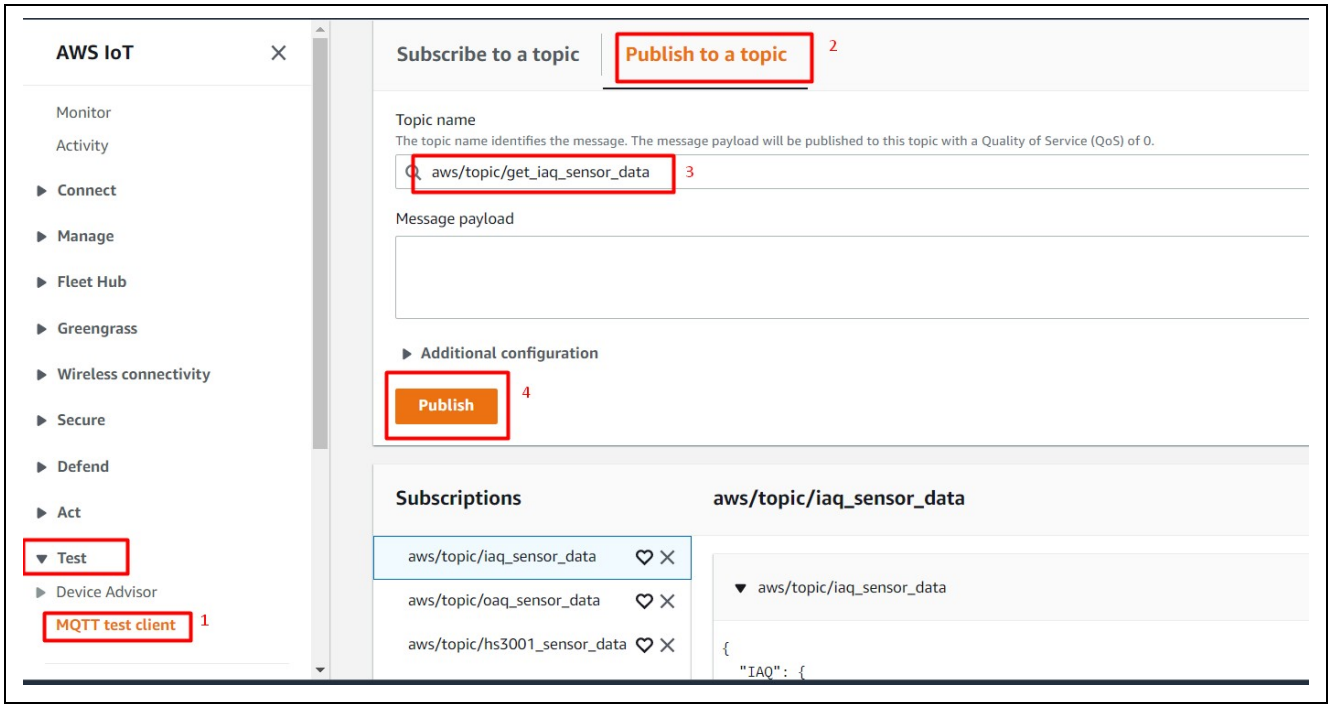

図 22　AWS IoT 画面からのメッセージのパブリッシュ

2.5　Renesas ダッシュボードからアプリケーションプロジェクトを検証する

Renesas AWS ダッシュボードには、renesas.cloud-ra-rx.com から[Go to Grafana]をクリックしてアクセスできます。

注：　ユーザは、デバイスがプロビジョニングされ、デバイスの状態が[Active]の場合にのみ、Grafana ダッシュボードにアクセスできます。

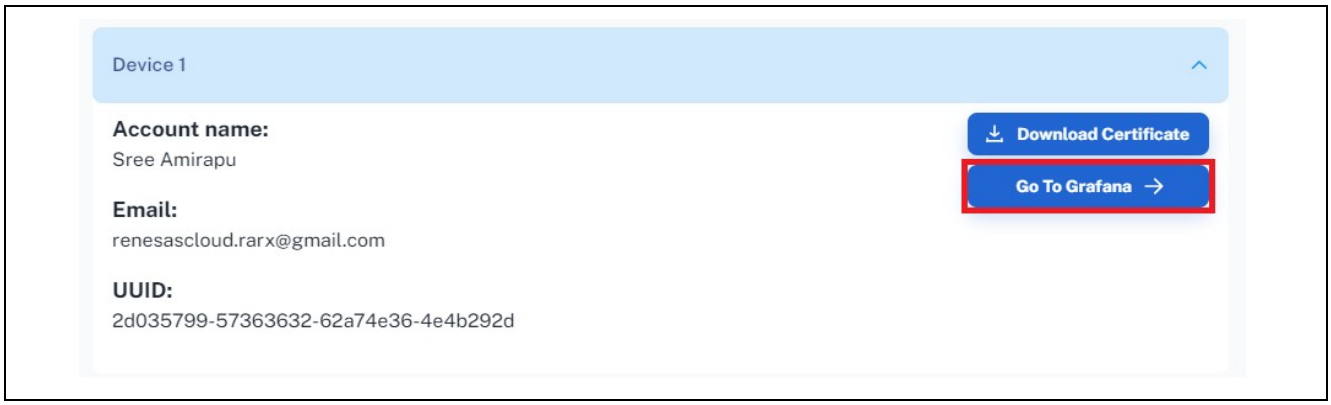

図 23　ルネサス AWS クラウドダッシュボードへのアクセス

Grafana ダッシュボードページが表示されたら、各センサデータタブの横にある「矢印」をクリックすることで、センサデータを表示することができます。ダッシュボードにデータが表示されるまで、最大 60 秒かかります。データが期待通りに表示されない場合は、ページを更新してください。

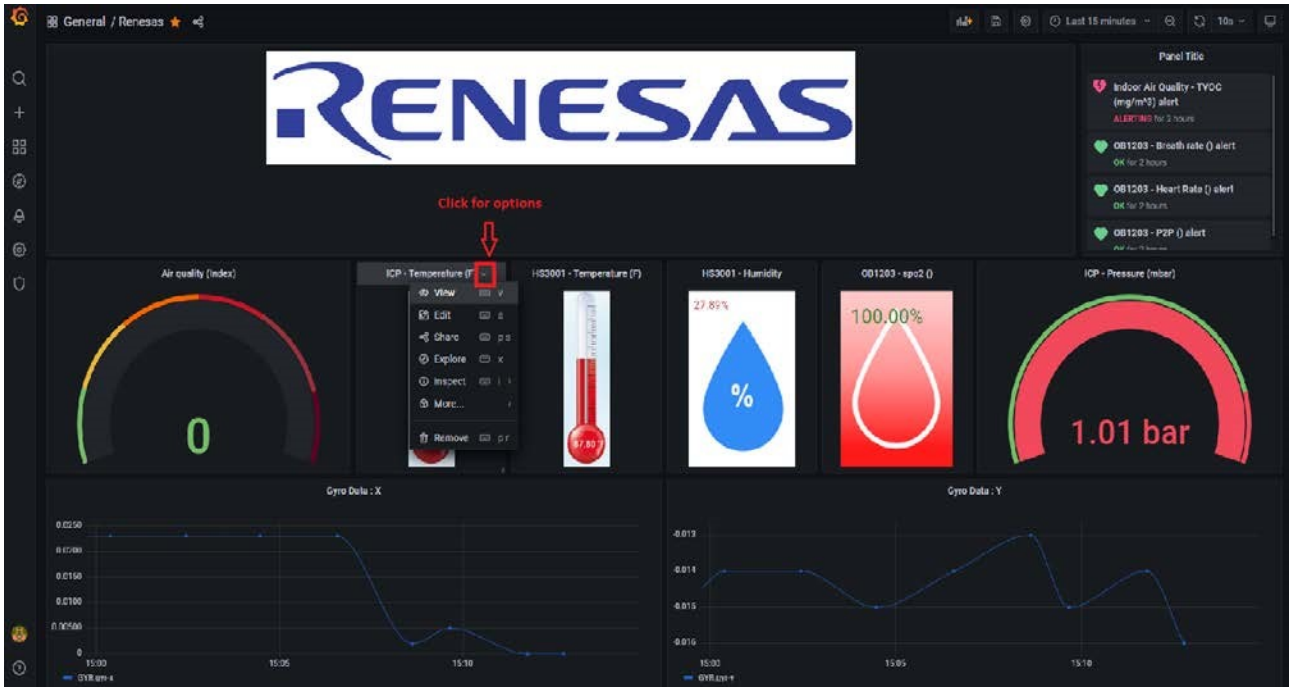

図 24　ルネサス AWS クラウドダッシュボード

3. Cellular インターフェースを備えた AWS Core MQTT

3.1 AWS Core MQTT

RA FSP に含まれる AWS MQTT ライブラリは、AWS MQTT または Mosquitto などのサードパーティ製 MQTT ブローカーに接続することができます。ライブラリの完全なドキュメントは、MQTT のウェブサイト AWS IoT Device SDK C に掲載されています。このライブラリでサポートされている主な機能は以下の通りです。

- AWS IoT エンドポイントか Mosquitto サーバ、または任意の MQTT ブローカーに TLS で MQTT 接続
- Mosquitto サーバへの非セキュア MQTT 接続 [4]

AWS Core MQTT は、[Thread]スタックに直接インポートすることができます。これは、RA Configuration パースペクティブを通じて設定されます。AWS Core MQTT を新しいスレッドに追加するには、RA Configuration で `Configuration.xml` を開きます。正しいスレッドが左側で選択されていることを確認しながら、[Stacks]タブ > [New Stack] > [Search]を使用して、キーワード AWS Core MQTT で検索してください。

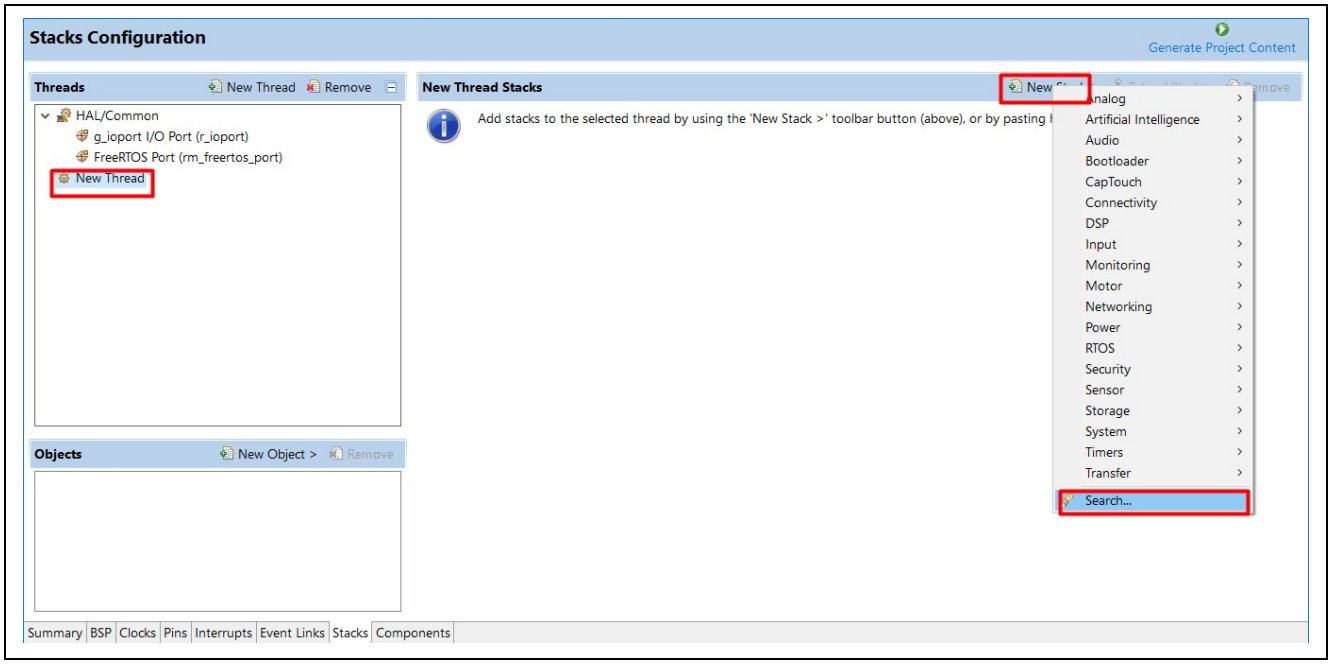

図 25　AWS Core MQTT モジュールの選択

AWS Core MQTT スタックを追加すると、下図のように依存関係が満たされていないデフォルト設定になります。FSP はユーザに様々なトランスポートインターフェースを提供します。このアプリケーションノートでは、図 27 に示すように、*AWS Transport Interface on MbedTLS11* を使用する Cellular インターフェースついて説明します。

[4] 実際の運用ではなく、ローカルサーバのテストに推奨されます。

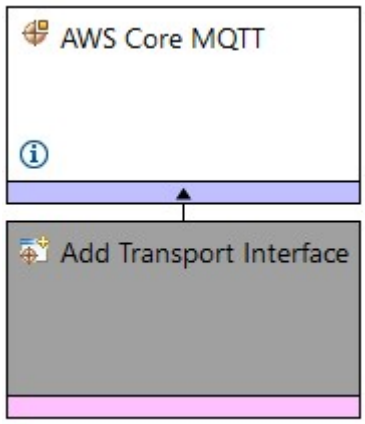

図 26　AWS Core MQTT スタックビュー

表示されている AWS Core MQTT スタックには、多くの依存関係と設定可能なプロパティが含まれていますが、ほぼデフォルト設定のままで使用できます。新しいプロジェクトに追加された AWS Core MQTT スタック(上記参照)に対して、満たされていない依存関係(赤で表示)をすべて満たすには、次の変更が必要です。

- 作成されたスレッドプロパティで、IoT SDK と FreeRTOS が必要とする Mutex と Recursive Mutex の使用サポートの有効化

上記の手順が完了すると、AWS Core MQTT はソケット実装を受け入れる準備ができており、これは、TLS セッションと基礎となる TCP/IP 実装の使用に依存しています。

AWS Core MQTT に関するその他のドキュメントは、FSP ユーザーズマニュアルの以下で入手できます。
RA Flexible Software Package Documentation > API Reference > Modules > AWS Core MQTT

3.2　トランスポート層の実装

FSP が提供する AWS トランスポートインターフェースは、Cellular、イーサネット、Wi-Fi のオプションを提供します。[AWS Transport Interface on MbedTLS11]は、Cellular インターフェースに使用されます。RA FSP には Wi-Fi およびイーサネットのセキュアソケット実装が含まれていますが、本アプリケーションおよびアプリケーションノートでは、Cellular インターフェースの使用に焦点を当てています。

MbedTLS/PKCS11 の[Add Transport Interface] > [New] > [AWS Transport Interface on MbedTLS/PKCS11]をクリックすると、Cellular ソケットをスレッドスタックに追加することができます。

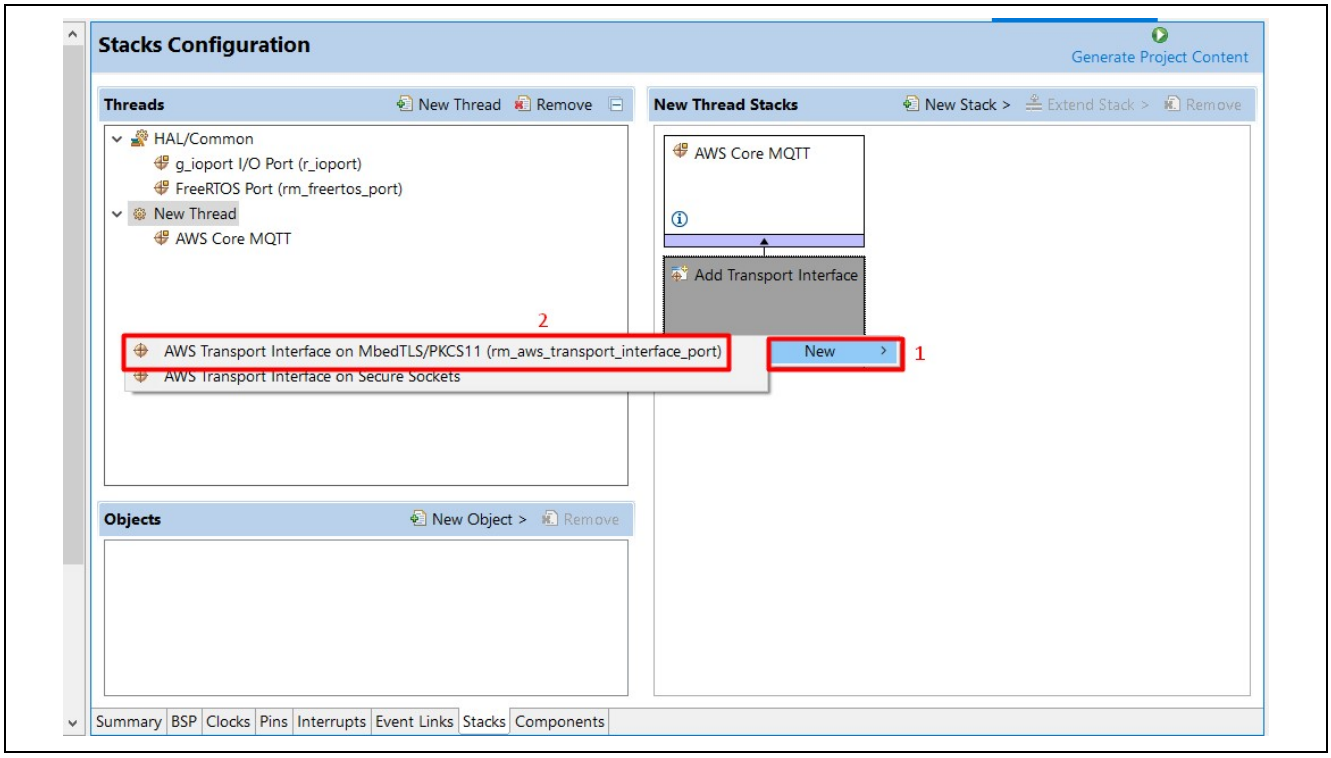

図 27　Core MQTT Module への Cellular インターフェースの追加

追加すると、必要なスタックが完成し、依存モジュールに対して満たされていない依存関係が発生します。

ここで、赤いブロックにカーソルを置くと、エラーがポップアップ表示されます。適切な設定をしてください。

- [Buffer Allocation]エラーの場合、[New Stack] > [FreeRTOS] > [Memory Management] > [Heap 4]を使用してヒープ実装を選択します。また、[Application Thread] > [Properties] > [Memory Allocation] > [Support Dynamic Allocation] > [Enabled]で[Dynamic Memory allocation]を設定します。
- Crypto の場合、MbedTLS の MBEDTLS_ECDH_C は、MbedTLS を使用する際に定義する必要があります。[Using MbedTLS(Crypto Only)] > [Public Key Cryptography(PKC)] > [ECC] > [MBEDTLS_ECDH_C]
- RTOS ヒープメモリエラーの場合は、[Application Thread] > [Properties] > [Memory Allocation] > [Total Heap Size] > [0x40000]でヒープメモリ割り当てを設定します。
- 永続ストレージのフラッシュファイルシステムが必要です。これは、[Add AWS PKCS11 PAL module] > [New] > [AWS PKCS11 PAL on LittleFS]をクリックすることで追加できます。
- LittleFS や他の標準ライブラリ関数で malloc を行うには、0x1000 以上のヒープを[BSP] > [RA Common] > [Heap Size]で追加する必要があります。
- [Application Thread] > [Common] > [General] > [Use Mutexes] > [Enabled]で Mutex を有効にする必要があります。
- [Application Thread] > [Common] > [General] > [Use Recursive Mutexes] > [Enabled]で Mutex を有効にする必要があります。
- [Application Thread] > [Common] > [Optional Functions] > [xTimerPendFunctionCall() Function] > [Enabled]で、xTimerPendFuncionCall を有効にする必要があります。
- UART のエラーは、フロー制御を有効にし、RTS と CTS ピンを適切に選択することで解決できます。

注：　これらは、コンフィギュレータからエラーを取り除くために必要な基本的な設定です。より具体的な設定は、各モジュールとその使用方法に記載されています。

すべての適切な設定により、設定不足によるエラーが解消された後、新しいコンフィギュレータのスクリーンショットは、以下の図 28 に示すように、エラーがなくきれいに表示されます。

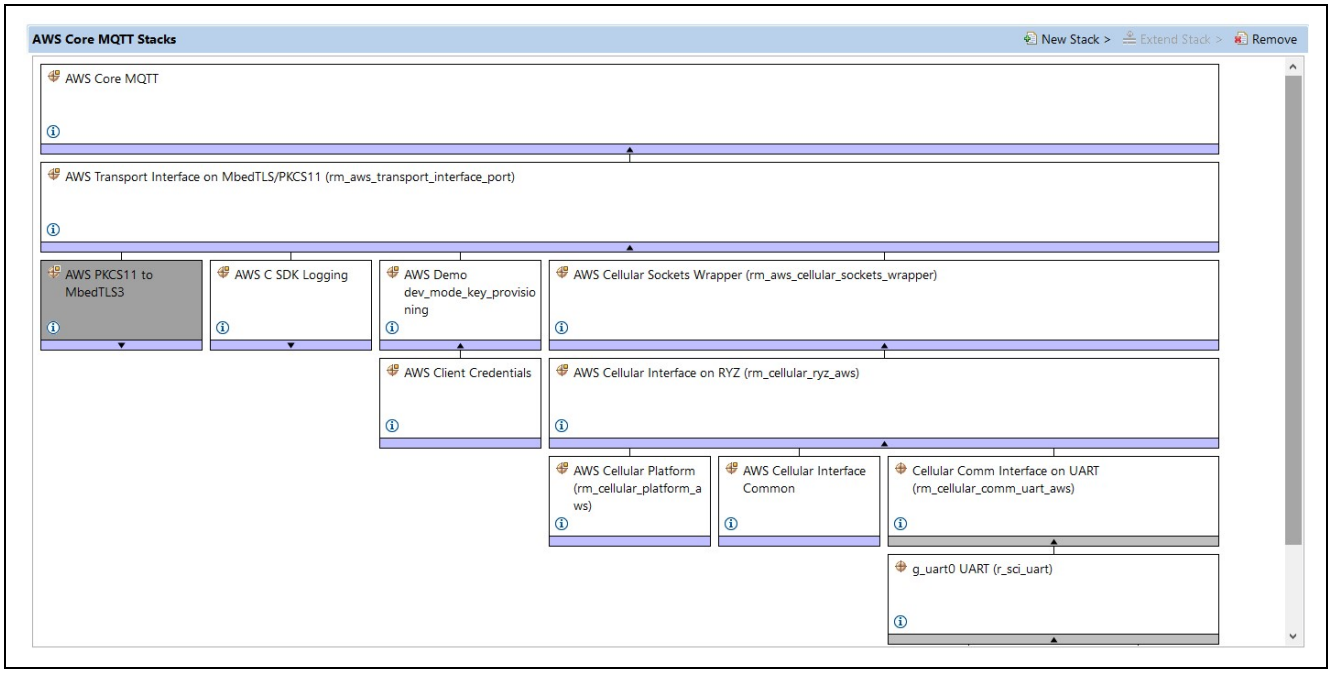

図 28　拡張した Cellular ソケットインターフェースモジュール

MbedTLS の AWS　トランスポートインターフェイスに関するその他のドキュメントは、FSP ユーザーズマニュアルの以下で入手できます。
RA Flexible Software Package Documentation > API Reference > Modules > AWS Transport Interface on MbedTLS/PKCS11

3.3　Mbed TLS

Mbed TLS は、Arm® による TLS プロトコルの実装と、それらの実装に必要な暗号プリミティブです。また、Mbed TLS は、TLS/SSL の部分が使用されていなくても、その暗号機能のみが使用されます。

TLS サポートは、最終的に Mbed TLS を使用する FreeRTOS+Crypto を使用します。Mbed TLS を使用するには、Mbed Crypto モジュールの設定と操作が必要で、そのモジュールが MCU の SCE を操作します。

FreeRTOS+Crypto モジュールで Cellular ソケットを使用するプロジェクトでは、以下の基礎部分の変更が必須です。

1. FreeRTOS ヒープ実装スキーム 4 (ファーストフィットアルゴリズムと結合アルゴリズム)またはスキーム 5 (ファーストフィットアルゴリズムと結合アルゴリズムに加え、隣接していないメモリ領域にまたがることが可能)を使用します。
2. FreeRTOS の動的メモリアロケーションのサポートを有効にします。
3. Mbed TLS プラットフォームのメモリアロケーションレイヤを有効にします。
4. Mbed TLS の汎用スレッドレイヤを有効にします。これはユーザのデフォルトのロックと Mutex を処理し、並列スレッドライブラリを使用するためにスレッドレイヤを抽象化します。
5. Elliptic Curve Diffie Helleman ライブラリを有効にします。
6. FreeRTOS　の合計ヒープサイズを 0x1500 より大きい値に変更します。

Mbed TLS に関するその他のドキュメントは、FSP ユーザーズマニュアルの以下で入手できます。
RA Flexible Software Package Documentation > API Reference > Modules > Crypto Middleware (rm_psa_crypto)

3.4　MQTT モジュール API の使用方法

表 2 に、アプリケーション例の一部として使用される AWS Core MQTT によって提供される API を示します。

表 2　MQTT モジュール API

API	説明
MQTT_Init	MQTT コンテキストを初期化します。
MQTT_Connect	MQTT セッションを確立します。
MQTT_Subscribe	ブローカーに、指定したトピックフィルターのリストに対する MQTT SUBSCRIBE を送信します。
MQTT_Publish	指定したトピック名へメッセージをパブリッシュします。
MQTT_Ping	ブローカーに MQTT PINGREQ を送信します。
MQTT_Unsubscribe	ブローカーに、指定したトピックフィルターのリストに対する MQTT UNSUBSCRIBE を送信します。
MQTT_Disconnect	MQTT セッションを切断します。
MQTT_ProcessLoop	ループを処理して、トランスポートインターフェイスからパケットを受信します。キープアライブを処理します。
MQTT_ReceiveLoop	ループを処理して、トランスポートインターフェイスからパケットを受信します。キープアライブを処理しません。
MQTT_GetSubAckStatusCodes	元のサブスクライブパケットから、トピックフィルターサブスクリプション要求に対応するステータスコードを含む MQTT SUBACK パケットのペイロードを解析します。
MQTT_Status_strerror	MQTT ステータスのエラーコードを文字列に変換します。
MQTT_PublishToResend	再送する次の保留中のパブリッシュのパケット ID を取得します。

4.　クラウド接続アプリケーション例

4.1　概要

このアプリケーションプロジェクトでは、Amazon IoT SDK C、Mbed TLS モジュール、Amazon FreeRTOS、および Renesas RA MCU で動作する HAL ドライバ等の、ルネサス FSP 統合モジュールで利用可能な API の使用方法のデモを行います。ネットワーク接続は、Cellular モジュールを使用して確立されます。Renesas クラウドキットで動作するアプリケーションは、FSP コンフィギュレータを使用した、Core MQTT、Mbed TLS/Crypto、Cellular 設定操作の参考のシステムとしても機能します。このアプリケーションは、Renesas RA MCU を使用して、カスタマイズされたクラウドベースのソリューションを考案するための出発点として使用できます。さらに、クラウドサービスプロバイダから提供されるクラウドサービスの操作とセットアップのデモも行っています。

次のサブセクションでは、エンドデバイスと通信するためにクラウド側の AWS IOT で必要とされるデバイスとセキュリティ認証情報のポリシーを段階的に作成する方法を示しています。このドキュメントに付随する例では、Core MQTT と MQTT ブローカー間のサブスクライブとパブリッシュメッセージング、センサデータのオンデマンドパブリッシュ、MCU からクラウドへの”センサデータ”イベントの非同期パブリッシュのデモを行っています。また、デバイスはクラウドからアクチュエーションイベント(LED 表示)を受信するようにサブスクライブされています。

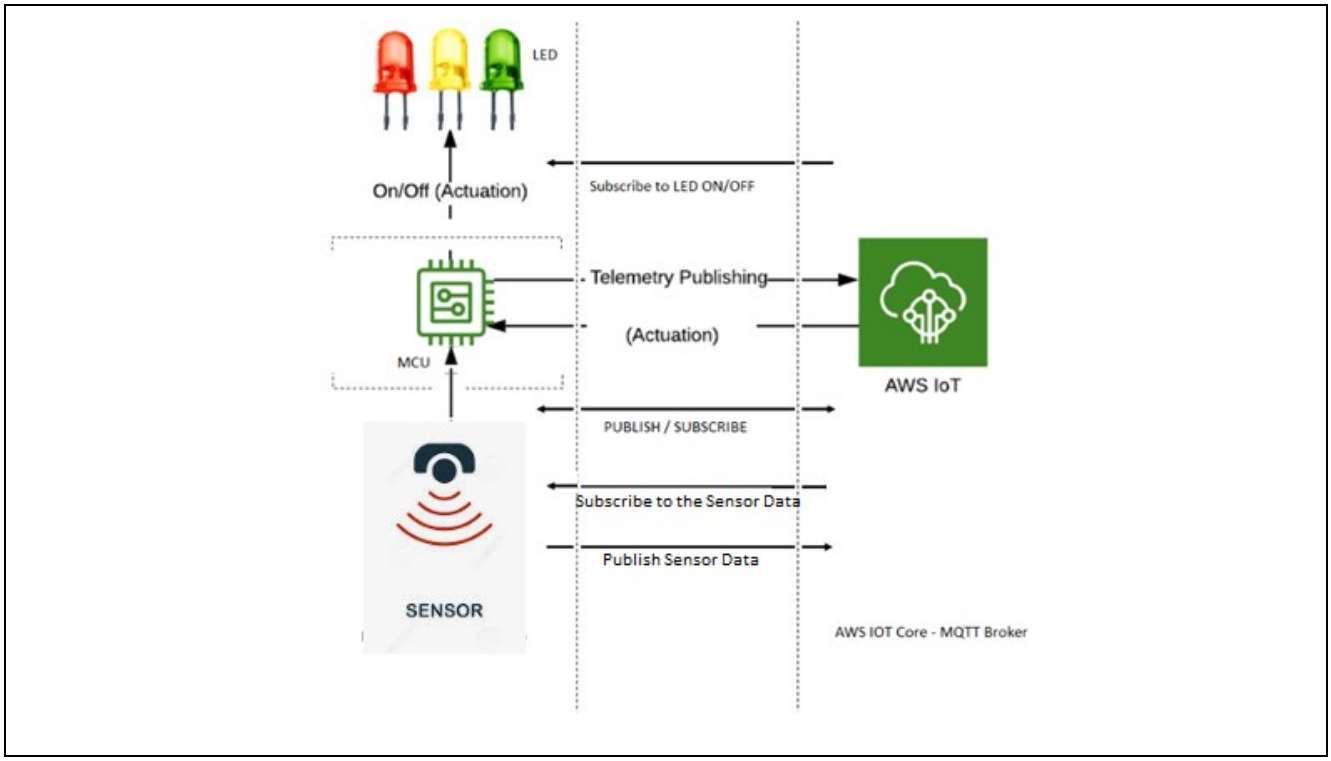

図 29　AWS IoT Core との MQTT のパブリッシュ/サブスクライブ

4.2　MQTT/TLS アプリケーションソフトウェア概要

このアプリケーションプロジェクトのファイルについて、参考情報として表 3 に示します。

表 3　アプリケーションプロジェクトファイル

No.	ファイル名	目的
1.	`src/app_thread_entry.c`	初期化コードと、クラウド接続アプリケーションで使用されるメインスレッドが含まれています。
2.	`src/cellular_setup.c`	Cellular 用の `init` 関数とデータ構造が含まれています。

No.	ファイル名	目的
3.	src/common_init.c	プロジェクト全体で共通の周辺機器を初期化するために使用されるマクロ、データ構造、関数が含まれています。
4.	src/common_init.h	プロジェクト全体で共通の周辺機器を初期化するために使用されるマクロ、データ構造、関数プロトタイプが含まれています。
5.	src/common_utils.c	プロジェクト全体で共通に使用されるマクロ、データ構造、関数が含まれています。
6.	src/common_utils.h	プロジェクト全体で共通に使用されるマクロ、データ構造、関数プロトタイプが含まれています。
7.	src/console_thread_entry.c	コマンドラインインターフェイスとフラッシュメモリ操作のコードが含まれています。
8.	src/ICM_20948.c	9-Axis MEMS Motion Tracking™センサのコードが含まれています。
9.	src/ICM_20948.h	9-Axis MEMS Motion Tracking™センサのデータ構造、関数プロトタイプが含まれています。
10.	src/ICP_10101.c	気圧・温度センサのコードが含まれています。
11.	src/ ICP_10101.h	気圧・温度センサのデータ構造と関数プロトタイプが含まれています。
12.	src/mqtt_demo_helpers.c	クラウド接続用の mqtt インターフェースで使用されるデータ構造と関数が含まれています。
13.	src/mqtt_demo_helpers.h	mqtt_demo_helpers.c に実装された関数を公開するための付属のヘッダです。
14.	src/oximeter_thread_entry.c	心拍数、血中酸素濃度、パルスオキシメトリ、近接、光、カラーセンサのコードを含んでいます。
15.	src/Oximeter.c	酸素濃度計センサに使用されるデータ構造と関数が含まれています
16.	src/Oximeter.h	酸素濃度計センサのデータ構造と関数プロトタイプが含まれています。
17.	src/oximstruct.h	酸素濃度計センサのデータ構造が含まれています
18.	src/r_typedefs.h	一般的な派生データ型が含まれています。
19.	src/RA_HS3001.c	心拍数、血中酸素濃度、パルスオキシメトリ、近接、光、カラーセンサのデータ構造と関数プロトタイプが含まれています。
20.	src/RA_HS3001.c	Renesas 製相対湿度・温度センサのコードが含まれています。
21.	src/RA_ZMOD4XXX_Common.c	Renesas ZMOD センサの共通コードが含まれています。
22.	src/RA_ZMOD4XXX_Common.h	Renesas ZMOD センサの共通データ構造用の関数プロトタイプが含まれています。
23.	src/RA_ZMOD4XXX_IAQ1stGen.c	Renesas ZMOD 室内空気質センサの共通コードが含まれています。
24.	src/RA_ZMOD4XXX_OAQ1stGen.c	Renesas ZMOD 屋外空気質センサの共通コードが含まれています。
25.	src/RmcI2C.c	FSP に統合されていないサードパーティセンサの I2C ラッパー関数が含まれています。

No.	ファイル名	目的
26.	src/RmcI2C.h	FSP に統合されていないサードパーティセンサの I2C ラッパー関数の関数プロトタイプが含まれています。
27.	src/sensor_thread_entry.c	異なるセンサからのセンサデータにアクセスするためのコードが含まれています。
28.	src/uart_CATM1.c	アクティベーションのための SIM 情報へのバックアクセスのために、CATM1 モジュールの UART インターフェースにアクセスするためのコードが含まれています。
29.	src/uart_CATM1.h	アクティベーションのための SIM 情報へのバックアクセスのために、CATM1 モジュールの UART インターフェースにアクセスするための関数プロトタイプが含まれています。
30.	src/ user_choice.c	ユーザが選択したセンサとユーザ設定のコードが含まれています。
31.	src/user_choice.h	センサのための関数プロトタイプと、異なるセンサとそのデータアクセシビリティのためのユーザ設定が含まれています。
32.	src/usr_config.h	アプリケーションを実行するためのユーザ設定をカスタマイズします。
33.	src/usr_hal.c	Hardware Abstraction Layer (HAL)の初期化および関連ユーティリティに使用されるデータ構造と関数が含まれています。
34.	src/usr_hal.h	usr_hal.c に実装された関数を公開するための付属のヘッダです。
35.	src/usr_data.h	アプリケーションスレッドに付属するヘッダファイルです。
36.	src/usr_network.c	FreeRTOS TCP/IP およびイーサネットモジュールの操作に使用されるデータ構造と関数が含まれています。このファイルはイーサネット専用です。
37.	src/usr_network.h	usr_network.c に実装された関数を公開するためのヘッダです。このファイルはイーサネット専用です。
38.	zmod_thread_entry.c	室内空気質センサのコードが含まれています。
39.	src/SEGGER_RTT/SEGGER_RTT.c	CPU 実行中のデバッガのメモリアクセスをサポートするターゲット上でリアルタイム通信を可能にする SEGGER real-time transfer (RTT)の実装です。
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	次の再送までのランダムバックオフによる再送アルゴリズムです。
44.	src/backoffAlgorithm/backoff_algorithm.h	次の再送までのランダムバックオフによる再送アルゴリズムのヘッダファイルです。
45.	src/subcription_manager/mqtt_subscription_manager.c	コールバックを処理する MQTT サブスクリプションマネージャです。
46.	src/subcription_manager/mqtt_subscription_manager.h	コールバックを処理する MQTT サブスクリプションマネージャの関連ヘッダファイルです。

No.	ファイル名	目的
47.	src/console_menu/menu_catm.c	CLI のメインメニューから CATM1 の SIM 情報を取得する関数が含まれています。
48.	src/console_menu/menu_catm.h	CLI のメインメニューから CATM1 の SIM 情報を取得するための関数プロトタイプが含まれています。
49.	src/console_menu/console.c	UART によるコンソールでデータを出力するために使用されるデータ構造と関数が含まれています。
50.	src/console_menu/console.h	UART によるコンソールでデータを出力するために使用される関数プロトタイプが含まれています。
51.	src/console_menu/menu_flash.c	CLI フラッシュメモリ関連のメニューを実装するために使用されるデータ構造と関数が含まれています。
52.	src/console_menu/menu_flash.h	CLI フラッシュメモリ関連のメニューを実装するために使用される関数プロトタイプとマクロが含まれています。
53.	src/console_menu/menu_kis.c	CLI メインメニューで FSP バージョンの取得、UUID の取得、ヘルプオプションを行う関数が含まれています。
54.	src/console_menu/menu_kis.h	CLI メインメニューで FSP バージョンの取得、UUID の取得、ヘルプオプションを行う関数プロトタイプとマクロが含まれています。
55.	src/console_menu/menu_main.c	CLI メインメニューオプションを実装するために使用されるデータ構造と関数が含まれています。
56.	src/console_menu/menu_main.h	CLI メインメニューオプションを実装するために使用される関数プロトタイプとマクロが含まれています。
57.	Src/flash/flash_hp.c	フラッシュメモリ関連の操作を行うために使用されるデータ構造と関数が含まれています。
58.	src/flash/flash_hp.h	フラッシュメモリ関連の操作を行うために使用される関数プロトタイプとマクロが含まれています。
59.	src/i2c.c	I2C 通信に使用されるデータ構造と関数が含まれています。
60.	src/i2c.h	I2C 通信に使用される関数プロトタイプとマクロが含まれています。
61.	src/ob1203_bio/KALMAN/kalman.c	心拍数、血中酸素濃度、パルスオキシメトリ、近接、光、カラーセンサのサンプルの計算アルゴリズムが含まれています。
62.	src/ob1203_bio/KALMAN/kalman.h	
63.	ob1203_bio/OB1203/OB1203.c	
64.	ob1203_bio/OB1203/OB1203.h	
65.	ob1203_bio/SAVGOL/SAVGOL.c	
66.	ob1203_bio/SAVGOL/SAVGOL.h	
67.	ob1203_bio/SPO2/SPO2.c	
68.	ob1203_bio/SPO2/SPO2.h	

注： app_thread_entry.c, sensor_thread_entry.c, oximeter_thread_entry.c, zmod_thread_entry.c, zmod_thread_entry.c はプロジェクト作成時に自動生成されるファイルです。このファイルにはユーザがコードを追加する必要があります。

注： 提供されたコードでアプリケーションを実行するには、app_thread_entry.c, sensor_thread_entry.c, oximeter_thread_entry.c, zmod_thread_entry.c が利用可能で本アプリケーションノートのバンドルの一部は、自動生成ファイルにマージしたり上書きしたりすることができます。

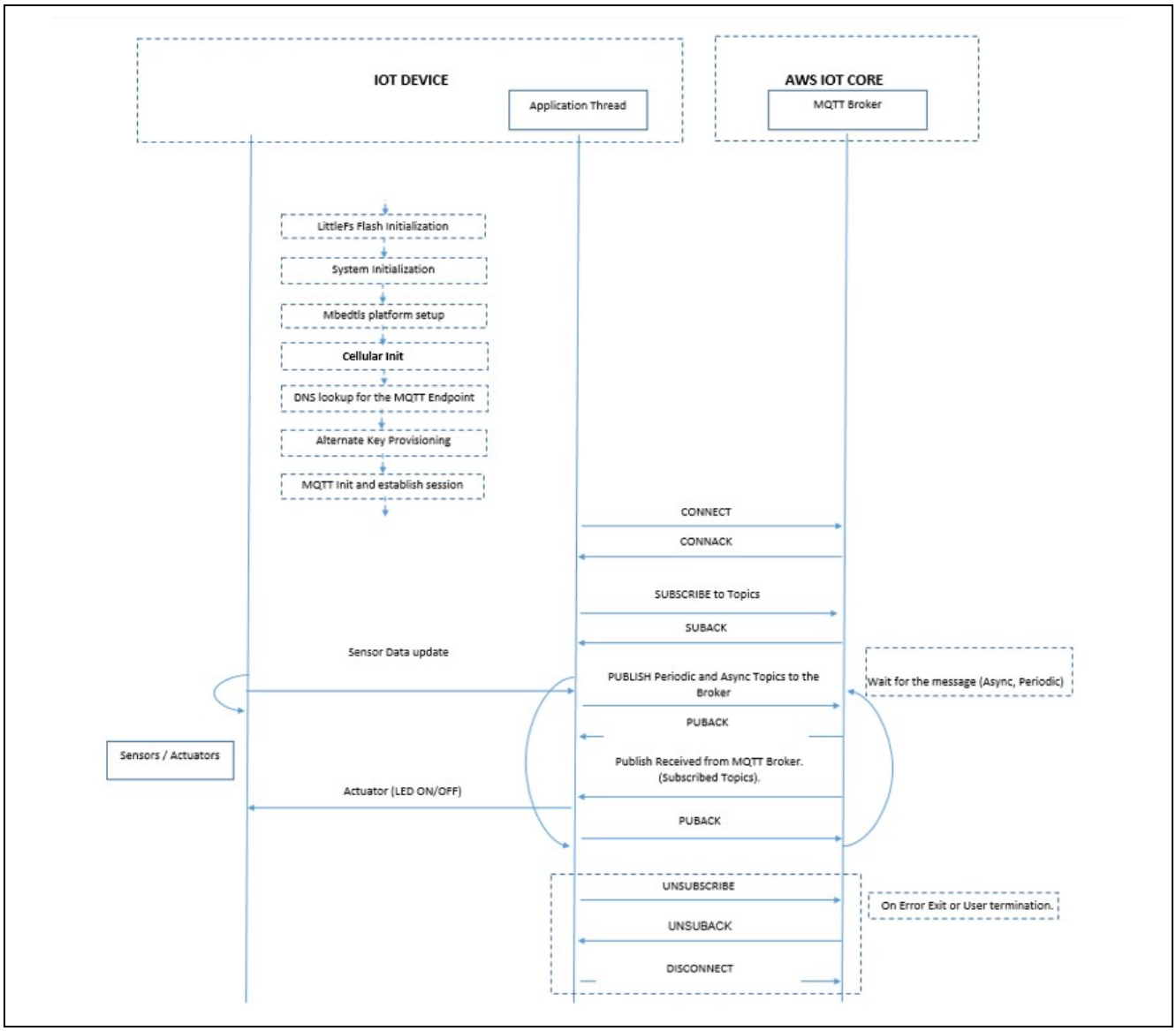

図 30　サンプルアプリケーションの実装の詳細

4.3　FSP コンフィギュレータを使用したアプリケーションプロジェクトの作成

e² studio と FSP コンフィギュレータを使い始めるところから、プロジェクトを作成する手順を完了します。下の表は、プロジェクトを作成するする段階的なプロセスを示しています。ユーザが e² studio と FSP コンフィギュレータに精通していることを前提としています。FSP 用にインストールされている e² studio を起動します。

表 4　Cellular 用アプリケーションプロジェクトの作成手順詳細

No.	手順	手順詳細
1	**プロジェクトの作成：**	**ファイル → 新規 → C/C++ Project**
2	**プロジェクトテンプレート：**	**新規 C/C++プロジェクトのテンプレート → Renesas RA C/C++ Project → 次へ**
3	**e² studio – プロジェクト設定(RA C 実行可能プロジェクト) →**	**Project Name (プロジェクトの名前)** 注：任意のプロジェクト名を入力してください -> **次へ**
4	**Device Selection →**	FSP Version:**3.7.0**
		Board: **CK-RA6M5**
		Device: **R7FA6M5BH3CFC**
		Language: C

No.	手順	手順詳細
5	**Tools Selection**	Toolchains: **GNU ARM Embedded** (Default)
		Toolchain version: **(**10.3.1.20210824**)**
		Debugger: **J-Link ARM**
6	**Build Artifact and RTOS Selection**	Artifact Selection: **Executable**
		RTOS Selection: **FreeRTOS(v3.7.0) → 次へ**
	Project Template	**Project Template Selection: FreeRTOS – Minimal – Static Allocation → 終了**
7	**Stacks タブ**(FSP Configurator の一部) **→**	**Threads → New Thread**
8	**Thread プロパティの設定→**	
		Symbol: **app_thread**
		Name: **Application Thread**
		Stack size: **32768 Bytes**
		Priority: **1**
		Thread Context: **NULL**
		Memory Allocation: **Static**
9	Thread 配下に汎用的な RTOS 設定を行います。(FSP によって提供されるデフォルト設定に追加で設定します)	
	Common → General	Use Mutexs: **Enabled**
		Use Recursive Mutexes: **Enabled**
		Max Task Name Len: **32**
	Common → Memory Allocation	Support Dynamic Allocation: **Enabled**
		Total Heap Size: **0x40000** 注：イーサネットアプリケーションの場合、合計ヒープサイズ: 0x40000 が必要です。
	Common->Optional Functions	xTimerPendFunctionCall() Function: **Enabled**
10	HAL/Common に Heap 実装を追加	
	New Stack →	**RTOS → FreeRTOS Heap 4**
11	**AWS MQTT ラッパーモジュールを Application Thread に追加**	
	注：新規作成されたスレッド(アプリケーションスレッド)は、新規スタックを追加する準備ができています。(ここでは AWS Core MQTT が追加されます)	
	New Stack →	**Networking → AWS Core MQTT**
13a	**Add transport Interface** の下に追加	**New → AWS Transport Interface on MbedTLS/PKCS11**
14	AWS PKCS11 の永続ストレージサポートを追加し、コンフィギュレータで BSP タブの Heap size を選択してエラーを解決します。ピンクで強調されたスタックを右クリックして次の操作を行います。	
	Add AWS PKCS11 PAL module →	**New → AWS PKCS11 PAL on LittleFS**
	BSP Tab → RA Common→	Heap size: **0x1000**
15	**MbedTLS(Crypto Only)**プロパティ設定を修正して FSP コンフィギュレータのエラーを解消するには、TLS サポートに関連するいくつかの依存関係を解決する必要があります。	
	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**
19	**AWS Cellular ソケットラッパー設定** 注：これは、Cellular アプリケーションプロジェクトにのみ適用されます。ほとんどデフォルト設定と同じままですが、一部のデフォルト設定は変更する必要があります。	

No.	手順	手順詳細
	AWS Cellular Interface on RYZ(rm_cellular_ryz_aws) →	**Module Reset Pin (Port Number): 04**
		Module Reset Pin (Pin Number): 09
20	**Cellular Comm Interface on UART**	
	Name →	**g_cellular_comm_interface_on_uart**
	Common →	**Receive Buffer: 512**
		Receive Transfer Size 512
21	**g_uart0 UART**	
	Common →	**FIFO Support: Enable**
		DTC Support: Enable
		Flow Control Support: Enable
	Module g_uart0 UART	
	General	**Name: g_uart0**
		Channel: 0
	Baud	**Baud Rate: 921600**
	Flow Control	**Software RTS Port: 04**
		Software RTS Pin: 12
	Interrupts	**Receive Interrupt Priority: Priority 1**
		Transmit Data Empty Interrupt Priority: Priority 2
		Transmit End Interrupt Priority: Priority 2
		Error Interrupt Priority: Priority 2
22	アプリケーションプロジェクトに必要な HAL モジュールを追加します。ここでは、Timer0、Timer1、外部 IRQ、ELC モジュールを、MCU 温度、30 秒周期タイマ、1 秒周期タイマ、心拍モニタタイマ、押しボタンスイッチ、ADC データ読み込みのイベントリンクにそれぞれ使用します。	
	HAL/Common Stacks → New Stack	**→ System → Clock Generation circuit on r_cgc**
	r_cgc のプロパティ設定	Name: **g_cgc**
	HAL/Common Stacks → New Stack	**→ Input → External IRQ Driver on r_icu**
	r_icu のプロパティ設定	Name: **g_sensorIRQ**
		Channel: **14**
		Trigger: **Falling**
		Digital Filtering: **enabled**
		Digital Filtering Sample Clock (PCLK/64)
		Pin Interrupt Priority: **Priority 2**
		Callback: **sensorIRQCallback**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	r_gpt のプロパティ設定 → General	Name: **g_timer0**
		Channel: **0**
		Mode: **Periodic**
		Period: **10**
		Period Unit: **Milli seconds**
	Interrupts:	Callback: **t_callback**
		Overflow/Crest Interrupt Priority: **Priority 5**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	r_gpt のプロパティ設定 → 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**

No.	手順	手順詳細
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	r_gpt のプロパティ設定→ General	Name: **g_timer2**
		Channel: **1**
		Mode: **Periodic**
		Period: **1**
		Period Unit: **Milli Seconds**
	Interrupts:	Callback: **TimerCallback**
		Overflow/Crest Interrupt Priority: **Priority 5**
24	**BSP 設定の変更 - RA Common (Main stack と Heap 設定)**	
	RA Common のプロパティ設定	Main stack size(bytes): **0x1000**
		Heap size (bytes): **0x1000**
25	アプリケーション用の FreeRTOS オブジェクトの追加(Message Queue アプリケーション用に Topic Queue を作成する必要があります)	
	Stacks Tab → Objects →	**New Object → Queue**
	Queue のプロパティ設定	Symbol: **g_topic_queue**
		Item Size (Bytes): **64**
		Queue Length (Items): **16**
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Mutex**
	Mutex のプロパティ設定	Symbol: **g_sens_data_mutex**
		Type: Mutex
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Mutex**
	Mutex のプロパティ設定	Symbol: **g_console_out_mutex**
		Type: Mutex
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Mutex**
	Mutex のプロパティ設定	Symbol: **g_update_console_event**
		Type: Mutex
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Binary Semaphore**
	Mutex のプロパティ設定	Symbol: **g_ob1203_semaphore**
		Memory Allocation: **Static**
26	**Stacks Tab (FSP Configurator の一部) →**	**Threads → New Thread**
	Thread のプロパティ設定 →	
		Symbol: **sensor_thread**
		Name: **Sensor Thread**
		Stack size: **4096 Bytes**
		Priority: **2**
		Thread Context: **NULL**
		Memory Allocation: **Static**
27	HS300X センサモジュールを Sensor Thread に追加する	
	New Stack →	**Sensor → HS300X Temperature/Humidity Sensor**
	HS300X センサ設定 →	Name: g_**hs300x_sensor0**
		Callback: **hs300x_callback**
28	**Stacks Tab (FSP Configurator の一部)** →	**Threads → New Thread**

No.	手順	手順詳細
	Thread のプロパティ設定 →	
		Symbol: **oximeter_thread**
		Name: **Oximeter Thread**
		Stack size: **2048 Bytes**
		Priority: **4**
		Thread Context: **NULL**
		Memory Allocation: **Static**
29	OB1203 センサモジュールを Oximeter Thread に追加します。 注：OB1203 センサのコードは FSP のコードを使用しません。	
	New Stack →	**Connectivity → I2C Communication Driver**
	HS300X センサ設定 →	Name: **g_comms_i2c_device3**
		Callback: **hs300x_callback**
		Semaphore Timeout (RTOS only): **0xFFFFFFFF**
		Slave Address: **0x53**
		Address Mode: **7-Bit**
		Callback: **comms_i2c_callback**
30	**Stacks Tab** (FSP Configurator の一部) **→**	**Threads → New Thread**
	Config Thread Properties →	
		Symbol: **zmod_thread**
		Name: **Zmod Thread**
		Stack size: **1024 Bytes**
		Priority: **4**
		Thread Context: **NULL**
		Memory Allocation: **Static**
31	ZMOD4XXX センサモジュールを Zmod Thread に追加します。 注：ZMOD4410 IAQ センサが設定されています。(FSP configurator の一部)	
	New Stack →	**Sensor → ZMOD4XXX Gas Sensor**
	Config ZMOD4XXX sensor →	Name: **g_zmod4xxx_sensor0**
		Callback: **zmod4xxx_comms_i2c_callback**
		IRQ Callback: **zmod4xxx_irq0_callback**

上記の構成は、本アプリケーションノートで提供されるクラウド接続アプリケーションに必要なスタックと機能を生成するための前提条件です。[Generate Project Content]ボタンをクリックすると、プロジェクト用のソースコードが生成されます。生成されたソースコードには、必要なドライバ、スタック、ミドルウェアが含まれています。ユーザーアプリケーションのファイルは、src フォルダに追加する必要があります。

注：　FSP が生成したコードはアプリケーションから呼び出して使用する必要がありますが、ミドルウェアの一部は適切な初期化のためのアプリケーションの一部として排他的に呼び出す必要があります。例えば `Mbedtls_platform_setup()`は、SCE と TRNG を初期化します。`CRYPTO_Init()`は Crypto と Socket のライブラリを初期化し、使用可能な状態にします。

作成したプロジェクトの検証のために、MQTT/TLS アプリケーションソフトウェア概要のセクション(表 3 を参照)に記載したものと同じソースファイルを追加することができます。また、コンパイルのために、ディレクトリパスとサブディレクトリを追加する必要があります。詳細については、付属のプロジェクトを参照してください。

デフォルト設定から変更された設定までのコンフィギュレータの詳細は、変更の理由を含めて次のセクションで説明されています。

4.4　MQTT/TLS の設定

このセクションでは、このサンプルアプリケーションの一部として実行される MQTT および TLS モジュールの設定について説明します。

以下の表に、RA コンフィギュレータで入力されたデフォルト設定に加えた変更内容を示します。

表 5　CK-RA6M5 の初期設定値

プロパティ	元の値	変更後の値	変更理由
Application Thread			
Common → General → Use Mutexes	Disabled	Enabled	この要件は、AWS IOT SDK C スタックによって設定されます。
Common → Memory Allocation → Support Dynamic Allocation	Disabled	Enabled	この要件は、AWS IOT SDK C スタックによって設定されます。
Common → Memory Allocation → Total Heap Size	0	0x40000	ヒープは、FreeRTOS、AWS IOT SDK、Mbed TLS に必要です。
AWS IoT Common			
Platform Name	Unknown	AWS Cloud Connectivity	この値はユーザが選択可能であり、任意の値を設定することが可能です。
Mbed TLS (Crypto Only)			
Platform → Mbedtls_platform_memory	Undefine	Define	この選択は、Mbed_tls のサポートに必要です。
General → Mbedtls_threading_alt	Undefine	Define	この選択は、Mbed_tls が任意のスレッドライブラリにプラグインできるようにするために必要です。
General → Mbedtls_threading_c	Undefine	Define	この選択は、Mbed_tls がスレッドレイヤを抽象化し、任意のスレッドライブラリに簡単にプラグインできるようにするために必要です。
Public Key Cryptography → ECC → Mbedtls_ecdh_c	Undefine	Define	この選択は、Mbed_tls が ECDH モジュールを有効にするために必要です。
LittleFS (Heap Selection)			
BSP → RA Common Heap Size	0x0	0x1000	ヒープ 3 以下のヒープ選択はここで行う必要があります。

5. センサ安定時間

センサ名	初回電源投入時	ソフトまたはハードリセット後
ZMOD4410 IAQ	最大 1 分	最大 1 分
ZMOD4510 OAQ	最大 4 時間	最大 2 時間
OB1203	最大 1 分(センサに指を乗せてから、データを検出するまで最大 60 秒かかる場合があります)	最大 10 秒(センサに指を乗せてから、データを検出するまで最大 60 秒かかる場合があります)
HS3001	最大 1 分	最大 10 秒
ICP	最大 1 分	最大 10 秒
ICM	最大 1 分	最大 10 秒

注：　上記のセンサの安定時間は、センサが初期化された時点からのものです

6. MQTT/TLS モジュールの次の手順

- 独自の CA 証明書によって署名されたデバイス証明書を使用してクライアントをセットアップするには、リンクを参照してください。
  https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html
- 自己署名証明書を使用して AWS を設定するには、リンクを参照してください。
  https://developer.amazon.com/docs/custom-skills/configure-web-service-self-signed-certificate.html

7. 参考文献

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

[2] Amazon Web Services, "AWS IoT Core の特徴" [オンライン] https://aws.amazon.com/jp/iot-core/features/

[3] Amazon Web Services, "AWS IoT Core" [オンライン] https://aws.amazon.com/jp/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," [オンライン] https://tools.ietf.org/html/rfc5246

[6] Amazon Web Services, "AWS IoT のセキュリティ" [オンライン] https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/iot-security.html

[7] Amazon Web Services, " AWS IoT でのトランスポートセキュリティ" [オンライン] https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/transport-security.html

[8] International Telecommunication Union, "X.509 (10/19) Summary" 10 2019. [オンライン] 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" [オンライン] https://mosquitto.org/.

[10] Amazon Web Services, "AWS IoT Device SDK C: MQTT," [オンライン] 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" [オンライン] https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/mqtt_functions.html

[13] Amazon, " FreeRTOS デモを設定する" [オンライン] https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/freertos-prereqs.html#freertos-configure

[14] "Amazon FreeRTOS mbedTLS" [オンライン] https://github.com/aws/amazon-freertos/blob/main/libraries/3rdparty/mbedtls_utils/mbedtls_utils.c

[15] Renesas Electronics Corporation, "Renesas Flash Programmer (Programming GUI) – ドキュメント" [オンライン]

https://www.renesas.com/jp/ja/software-tool/renesas-flash-programmer-programming-gui#document
[16] Silex Technology, Inc., "SX-ULPGN-EVK - Evaluation kit for Ultra-Low-Power Hostless Wi-Fi IoT Module," [オンライン]
https://www.silextechnology.com/connectivity-solutions/embedded-wireless/sx-ulpgn-evk

8. 既知の問題

このセクションでは、既知の FSP およびツール関連の問題について記載します。 詳細についてはリンクを参照してください。(https://github.com/renesas/fsp/issues)

9. デバッグ

アプリケーションプロジェクトの USR_LOG_LVL (LOG_DEBUG)マクロを有効にすると、追加のデバッグ情報が出力できます。

10. トラブルシューティング

Cellular 接続が不安定な場合や MQTT 接続が失われた場合は、RYZ014A Pmod の USB を PC に接続して、モジュールに追加の電力を供給します。RYZ014A Pmod Errata を参照してください。

エラーが続く場合は、プロジェクトをイーサネットインタフェースでテストすることで機能の検証ができます。(*RA AWS MQTT/TLS Cloud Connectivity Solution - Ethernet Application Note* を参照してください)

ウェブサイトとサポート

次の URL にアクセスして、RA ファミリの主要な要素について学び、コンポーネントと関連ドキュメントをダウンロードして、サポートを受けることができます。

CK-RA6M5 Kit 設計情報	renesas.com/ra/ck-ra6m5
RA クラウドソリューション	renesas.com/cloudsolutions
RA 製品紹介	renesas.com/ra
RA サポートフォーラム	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas サポート	renesas.com/support

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.01	Mar.27.23	ー	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。