RX65Nグループ

AWS FreeRTOS 2nd MCUのOTAファームウェアアップデート設計ガイド

要旨

クラウドを利用した多くの組込みシステムは、図 1-1のようにインターネットと直接接続され、Amazon Web Services™ (以後 AWS)と通信する機能を持った MCU(以後 1st MCU)と、1st MCU とローカルな通信路(UART や BLE 等)で接続されている MCU(以後 2nd MCU)で構成されています。

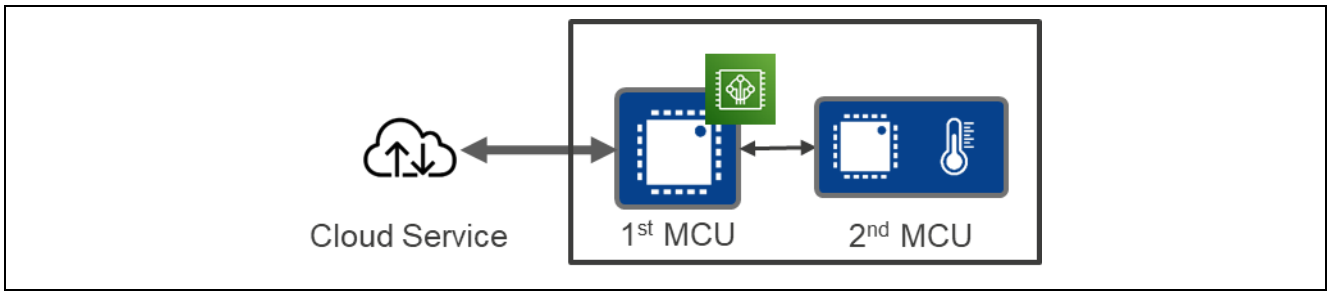


図 1-1 想定するクラウドを利用した組込みシステム概念図

本アプリケーションノートでは、このようなシステム上で 1st MCU を経由して 2nd MCU のファームウェアを OTA アップデート(以後 2nd MCU OTA アップデート)する方法を解説します。本アプリケーションノートを読むことで以下の知識を習得できます。

- AWS + FreeRTOS™ 環境で、OTA Library を利用した 2nd MCU OTA アップデートを実現する方法
- デバイス側での 2nd MCU OTA アップデートに必要なソフトウェア実装例
- AWS クラウド側での 1st MCU と 2nd MCU の OTA アップデート実行指示の切り替え方

これにより、AWS IoT Device Management サービスが提供している OTA アップデートの仕組みを用いて、2nd MCU の OTA アップデートを行うことができます。

本設計ガイドに基づいた 2nd MCU OTA アップデートのデモは以下を参照してください。

RX65N グループ FreeRTOS を用いたAmazon Web Service によるセカンダリデバイスのOTAアップデートサンプルコード

また、ルネサス MCU におけるファームウェアアップデートの設計方針についての基礎情報は以下のアプリケーションノートを参照してください。

ルネサスMCUにおけるファームウェアアップデートの設計方針

動作確認デバイス

- 1st MCU：
  RX65N グループ
- 2nd MCU：
  RX23W グループ

本アプリケーションノートおよび 2nd MCU OTA アップデートのデモアプリケーションノートは上記のマイコンを使用して動作を確認しています。本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様に合わせて変更し、十分評価を行ってください。

関連文書

本アプリケーションノートは、以下の文書を参照し、説明しています。文書更新の場合に章構成等が変わる場合があります。参照の際に注意してください。

文書名	文書番号
RXファミリ RX65NにおけるAmazon Web Servicesを利用したFreeRTOS OTAの実現方法	R01AN5549JJ0102
RX65Nグループ FreeRTOS を用いたAmazon Web Service によるセカンダリデバイスのOTAアップデートサンプルコード	R01AN6220JJ0110
ルネサスMCUにおけるファームウェアアップデートの設計方針	R01AN5548JJ0100
RXファミリ ファームウェアアップデートモジュール Firmware Integration Technology	R01AN5824JJ0102

その他リンク

- Secondary Device OTA デモンストレーション動画
  Secondary Device OTA Update using FreeRTOS and Amazon Web Services | Renesas
- RX ファミリコネクティビティソリューション WEB
  https://www.renesas.com/rx-cloud

目次

1. 概要

1.1 AWS IoT サービスと 2nd MCU OTA アップデート

AWS ブログの以下の記事では、既存のファームウェアを一部変更することで 2nd MCU OTA アップデートを実行する方法が紹介されています。

FreeRTOSでセカンダリプロセッサのOTAアップデートを実行する方法 | Amazon Web Services ブログ

また、以下の FreeRTOS に関する AWS 発行のユーザガイドに、OTA アップデートを作成する際に指定できる `fileType` パラメータの使い方について以下のように説明されており、2nd MCU の OTA アップデートへの利用が挙げられています。

> *Under File Type, enter an integer value in the range 0-255. The file type you enter will be added to the Job document that is delivered to the MCU. The MCU firmware/software developer has full ownership on what to do with this value. Possible scenarios include an MCU that has a secondary processor whose firmware can be updated independently from the primary processor. When the device receives an OTA update job, it can use the File Type to identify which processor the update is for.*

Creating an OTA update (AWS IoT console) - FreeRTOS (amazon.com)

これらの情報をもとに、本アプリケーションノートでは RX ファミリ MCU で 2nd MCU OTA アップデートを実現するために必要な手順をまとめています。

1.2 AWS IoT サービスが提供する OTA アップデートの仕組み

AWS IoT サービスは、クラウド側では AWS IoT Jobs を、FreeRTOS を搭載したデバイス側では AWS IoT Over-the-air Update Library(以後 OTA Library)を用いて OTA アップデートを実現しています。

AWS IoT Jobs とは、AWS IoT サービスで管理されているデバイスに対して、ジョブをリモートで実行可能な機能です。ジョブとは、ユーザが事前に定義した任意の動作を示します。ユーザは、ジョブの仕様をジョブドキュメントに定義します。ジョブドキュメントは JSON フォーマットであり、ユーザが自由に設計できます。

あるデバイスに対してジョブを作成すると、ジョブドキュメントを含むメッセージが、そのデバイスの AWS IoT Jobs 用トピックに MQTT で発行されます。デバイス側は自身の AWS IoT Jobs 用トピックを購読してメッセージを受け取ると、そこに含まれるジョブドキュメントを解析し、記述されている処理を実行します。そのため、ユーザはジョブドキュメントの解析処理と実行処理を事前に実装しておく必要があります。

このように AWS IoT Jobs の機能を実現するためには、ジョブドキュメントの定義と、デバイス側のジョブドキュメントの解析処理と実行処理の実装が、ユーザ側で必要となります。

そこで、AWS IoT サービスが提供する OTA アップデートでは、AWS が OTA アップデートを実行するジョブ用のジョブドキュメントの定義を用意し、OTA Library がデバイス側で実装が必要なジョブドキュメントの解析と実行処理を行うようになっています。このようにして、ユーザが簡単に OTA アップデートができるような仕組みが整備されています。

1.3 OTA Library

OTA Library は AWS によって GitHub 上で開発されている OSS プロジェクトであり、MIT License で公開されています。

aws/ota-for-aws-iot-embedded-sdk (github.com)

AWS による FreeRTOS のリファレンス実装であるFreeRTOS AWS Reference Integrationsの Version 202107.00 では OTA Library v3.0.0 が搭載されており、デモアプリケーションが提供されています。

無線通信経由更新デモアプリケーション - FreeRTOS (amazon.com)

自身の AWS IoT Jobs 用トピックにメッセージが発行されると、OTA Library はジョブドキュメントを解析し OTA アップデート処理を開始します。OTA アップデート処理の中には、ROM の特定領域にデータをセルフプログラミングする動作や、新ファームウェアへ更新するためのソフトウェアリセットを実行する動作など、ハードウェアに作用する処理が含まれます。このような処理を含みつつ、ライブラリのポータビリティを実現するために、OTA Library には OTA Platform Abstraction Layer(以後 OTA PAL)という API インターフェースが用意されています。この OTA PAL によってライブラリから見たハードウェアを抽象化しています。

1.4 2nd MCU OTA アップデートへの拡張

AWS IoT Jobs と OTA Library による OTA アップデートの仕組みを利用して、2nd MCU OTA アップデートを実現するためには、1st MCU OTA アップデートと 2nd MCU OTA アップデートを区別する方法を用いて、処理を切り替える必要があります。

まず、OTA アップデートの対象を区別するため、AWS が用意している OTA アップデートジョブ用のジョブドキュメントに定義されている `fileType` パラメータを利用します。OTA Library は `fileType` の値が 0 の場合、デフォルトでは 1st MCU の OTA アップデートを実行します。そのため、0 以外の値(例えば 1)を指定し OTA アップデートの対象を区別します。

次に、OTA 処理の切り替えには OTA PAL を利用します。OTA PAL はファームウェアアップデートに必要なハードウェア処理を抽象化しているため、2nd MCU OTA アップデート用の OTA PAL(以後 OTA PAL2)関数群を用意し、`fileType` の値によって呼ぶ OTA PAL 関数を切り替えることで、アップデートの対象を切り替えることができます。

1.5 動作確認環境

2nd MCU OTA アップデートは以下に示す条件で動作を確認しています。また、以降の説明はこの環境を想定しています。

表 1-1 2nd MCU OTA アップデート動作確認環境

項目	概要
1st MCU	RX65N
1st MCU ボード	Renesas Starter Kit+ for RX65N-2MB（RSK+RX65N-2MB ボード）
RTOS	FreeRTOS AWS Reference Integrations Version 202107.00
2nd MCU	RX23W
2nd MCU ボード	Target Board for RX23W（TB-RX23W ボード）
FWUP FITモジュール	V1.02
IDE	e2 studio 2022-01
Toolchain	CC-RX V3.04.00
ファームウェア結合ツール	Renesas Secure Flash Programmer 1.01 ※ 1.x.x 系を使用してください。2.x.x 系は現在 Amazon FreeRTOS プロジェクトを未サポートです。
ファームウェア書き込みツール	Renesas Flash Programmer V3.09.00

2. システム構成

2nd MCU OTA アップデートを実行するシステムの構成を説明します。

2.1 ハードウェア構成

以下にハードウェア構成の模式図を示します。

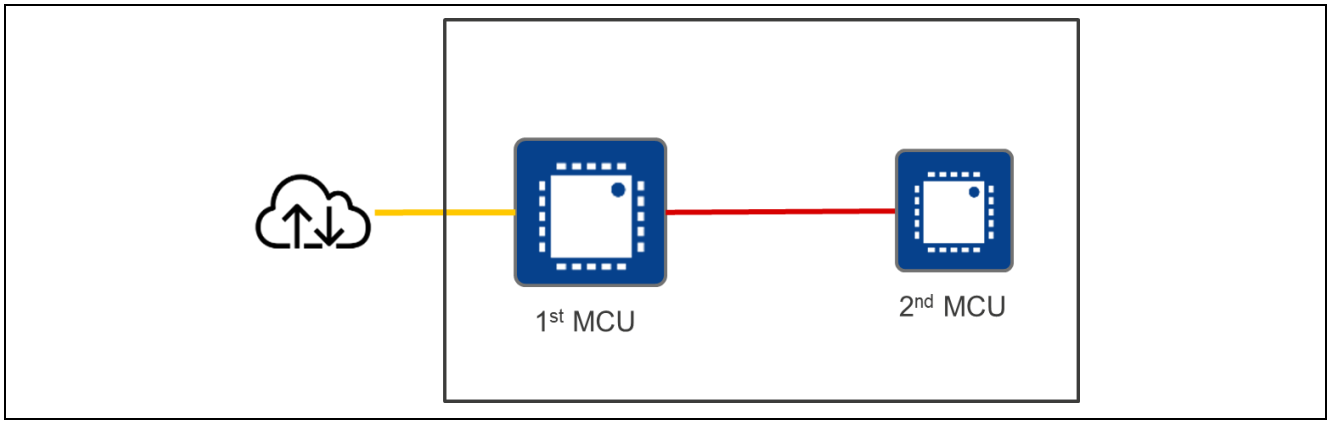


図 2-1 ハードウェア構成の模式図

まず、1st MCU はインターネットと直接接続されるため、イーサネットや Wi-Fi 等のインターフェースを備えている必要があります。(図 2-1黄色線)

本アプリケーションノートでは、OTA アップデートの機能まで含めて AWS のデバイス認定を取得している Renesas Starter Kit+ for RX65N-2MB(RSK+RX65N-2MB ボード)を使用し、イーサネットでインターネットに接続しています。

Renesas Starter Kit+ for RX65N-2MB | AWS Qualified (amazonaws.com)

次に、1st MCU と 2nd MCU はデータを相互に通信する必要があるため、通信路が必要です。(図 2-1 赤線)

本アプリケーションノートでは、UART 通信を行います。

2.2 ソフトウェア構成

以下に 1st MCU および 2nd MCU のソフトウェア構成を示します。

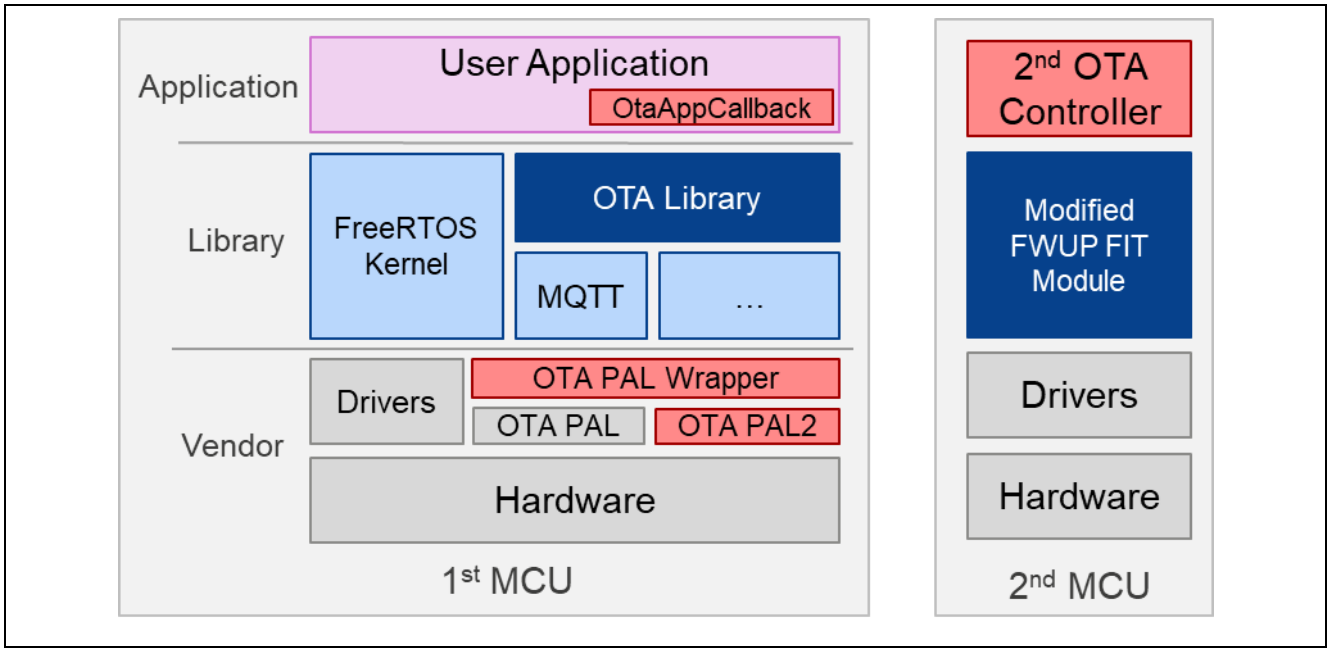


図 2-2 ソフトウェア構成

2.2.1 1st MCU のソフトウェア構成

FreeRTOS AWS Reference Integrations Version 202107.00 をベースに作成します。図 2-2のソフトウェア構成図の**赤箱で示した部分が新規で作成、または追記が必要な箇所**になります。

User Application として、OTA アップデートのデモアプリケーションとして提供されている `RunOtaCoreMqttDemo` を使用します。OtaAppCallback は OTA Library 初期化時に登録するコールバック関数です。このコールバック関数内に、アップデート完了時のイベント(*`OtaJobEventUpdateComplete`*)に実行する処理を追記します。

OTA PAL2 には 2nd MCU にコマンドを送信する処理を実装します。OTA PAL Wrapper には `fileType` の値によって OTA PAL と OTA PAL2 の切り替えを行う処理を実装します。

2.2.2 2nd MCU のソフトウェア構成

2nd MCU のソフトウェアは Bootloader と User Program の 2 つのプロジェクトで構成され、ファームウェアアップデート(FWUP) FIT モジュールのデモプロジェクトをベースに作成します。

Bootloader は起動時に最初に実行され、User Program のコード署名の検証や新ファームウェアへの切り替えを行います。

User Program では、1st MCU から SCI チャネルを介して UART 通信で送信されるコマンドによるイベントドリブンなステートマシンとして、2nd OTA Controller を実装し FWUP FIT モジュールを制御します。ファームウェアアップデート処理は、一部変更を加えた FWUP FIT モジュールが行います。

2.2.3 AWS 構成

AWS 上に構築が必要な環境は、通常の AWS IoT サービスを用いた OTA アップデートと同様です。以下のアプリケーションノートを参考に AWS の環境を構築してください。

RXファミリ RX65NにおけるAmazon Web Servicesを利用したFreeRTOS OTAの実現方法

3. 2nd MCU OTA アップデートの仕組み

3.1 全体の流れ

2nd MCU OTA アップデートの動作を以下に示します。

① User Application は OTA Library を初期化し、OTA Agent タスクを作成します。以降、OTA Agent が OTA アップデート処理を行います。

② OTA Agent はジョブを受信すると、ジョブドキュメントを解析し OTA アップデート処理を開始します。

③ OTA Agent は OTA アップデート処理内で OTA PAL Wrapper 関数を呼びます。

④ OTA PAL Wrapper 関数内で、`fileType` の値を参照し、実行する OTA PAL 関数を切り替えます。

⑤ 2nd MCU OTA アップデートの場合は OTA PAL2 の関数が実行され、UART で 2nd MCU にコマンドを送信します。

⑥ 2nd MCU が 1st MCU からコマンドを受信すると、2nd OTA Controller が FWUP FIT モジュールの API 関数を実行します。

⑦ FWUP FIT モジュールがファームウェアアップデート処理を実行します。

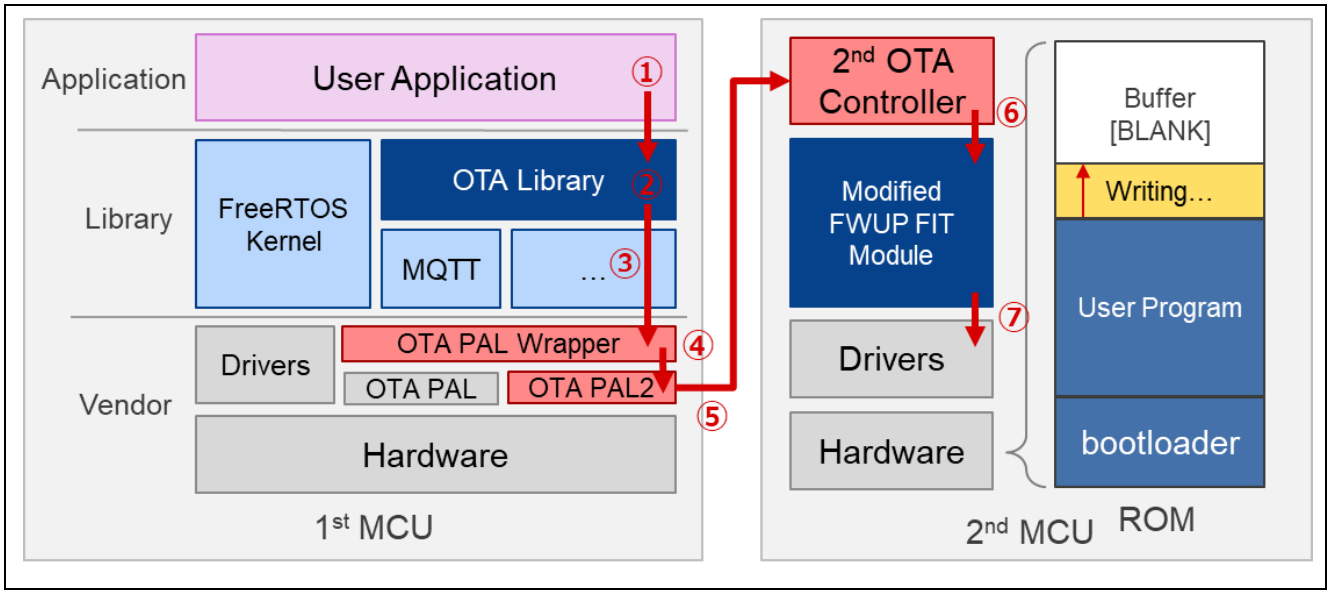


図 3-1 2nd MCU OTA アップデート処理の流れ

3.2 fileType の値と OTA Agent の動作の関係

OTA Library は fileType が configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID と一致する/しないで異なる動作をします。configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID はデフォルトでは 0U が定義されています。

- filetype = configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID の場合
  1st MCU の OTA アップデート処理の動作をします。処理の大まかな流れを図 3-2に示します。

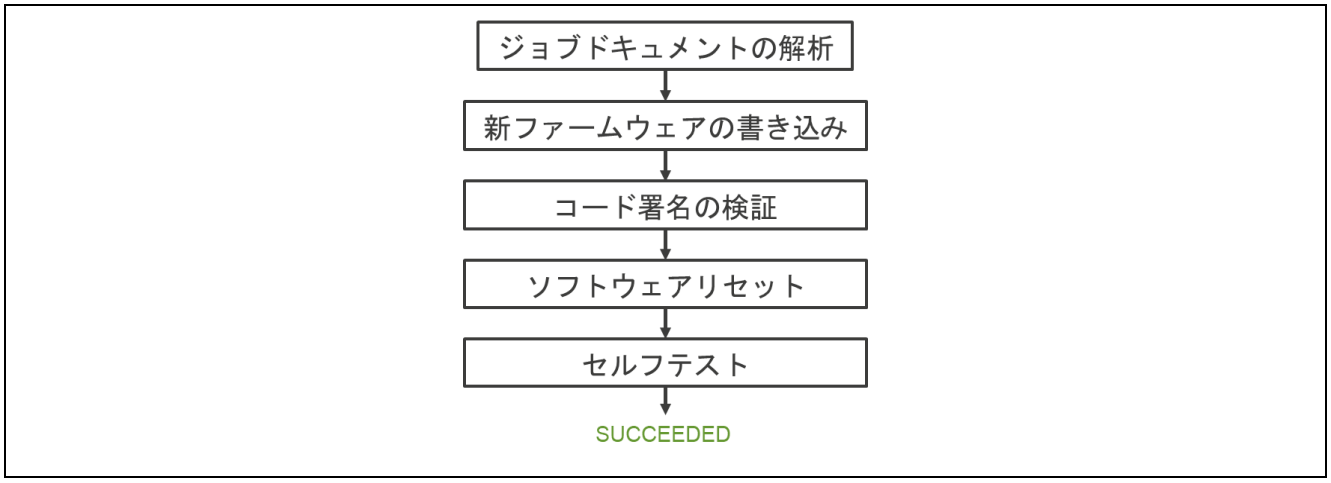


図 3-2 filetype = configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID の場合の OTA Agent 動作

- filetype ≠ configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID の場合
  新ファームウェアの書き込みが終了し、コード署名の検証後、AWS IoT サービスにジョブの SUCCEEDED / FAILED を送信し、*OtaJobEventActivate* のコールバック関数を実行することなくジョブを完了します。その後、*OtaJobEventUpdateComplete* のコールバック関数を実行します。処理の大まかな流れを図 3-3に示します。

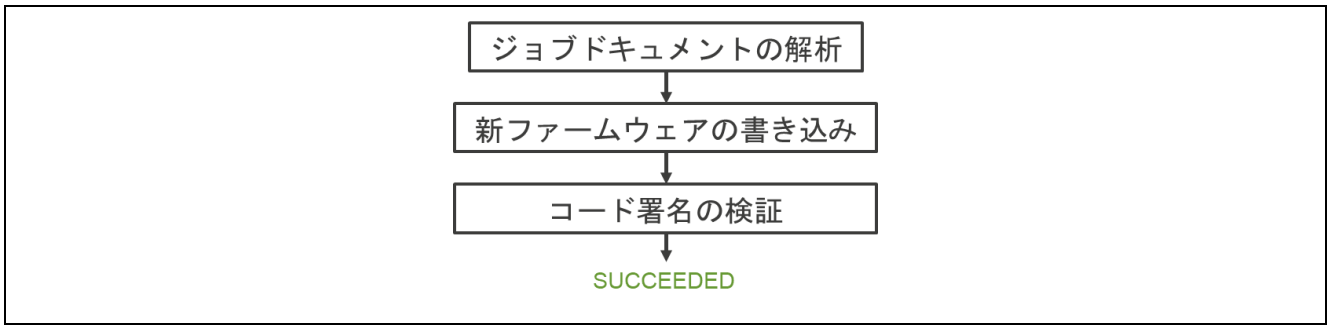


図 3-3 filetype ≠ configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID の場合の OTA Agent 動作

3.3 1st MCU ⇔ 2nd MCU 間の通信

1st MCU は、OTA 処理の中で OTA PAL2 の関数が呼ばれるタイミングで 2nd MCU にコマンドを送信します。そのため、各コマンド名を OTA PAL の API インターフェース名と同じにしています。正常に OTA アップデートが進行した場合、1st MCU ⇔ 2nd MCU 間の通信は図 3-4のようなシーケンス図となります。

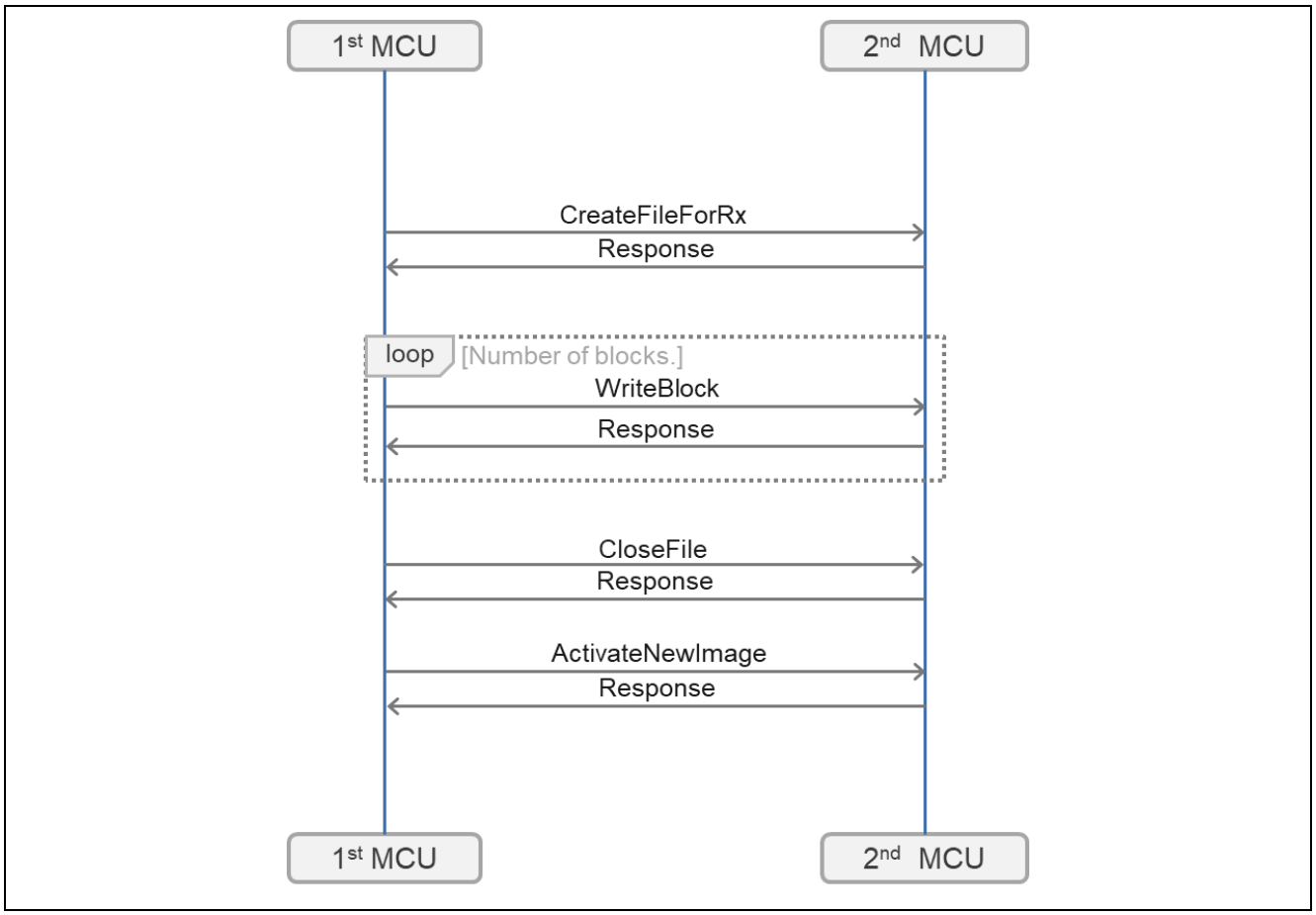


図 3-4 マイコン間の通信のシーケンス図

また、これとは別に、OTA アップデート処理中には GetPlatformImageState が複数回呼ばれます。

図 3-5 2nd MCU の ImageState を問い合わせる通信

更に、OTA アップデート処理中にエラーで処理を終了する場合、Abort が呼ばれます。

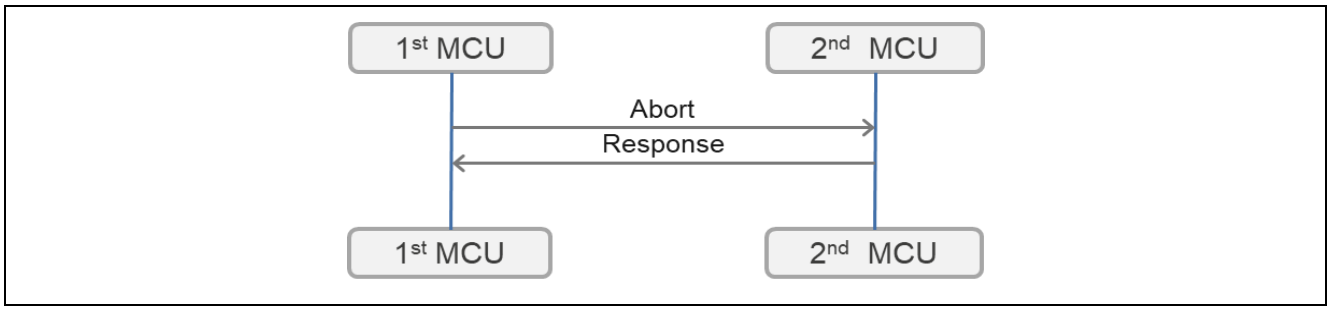


図 3-6 Abort 発生時の通信

2nd MCU OTA アップデートにおける、それぞれのコマンドの役割を以下に示します。

表 3-1 CreateFileForRx コマンドの説明

コマンド名	CreateFileForRx
説明	2nd MCU に OTA 処理の開始を知らせます。
2nd MCU からのレスポンス	成功 or 失敗

表 3-2 WriteBlock コマンドの説明

コマンド名	WriteBlock
説明	AWS から送信されてくる新ファームウェアイメージを 2nd MCU に転送します。 このコマンドには新ファームウェアイメージ用の 1024byte のペイロードが付属します。
2nd MCU からのレスポンス	成功 or 失敗

表 3-3 CloseFile コマンドの説明

コマンド名	CloseFile
説明	書き込みが完了した 2nd MCU の新ファームウェアに対してコード署名の検証を指示します。
2nd MCU からのレスポンス	成功 or 失敗

表 3-4 ActivateNewImage コマンドの説明

コマンド名	ActivateNewImage
説明	新ファームウェアに更新するためのソフトウェアリセットを 2nd MCU に命令します。
2nd MCU からのレスポンス	成功 or 失敗

表 3-5 Abort コマンドの説明

コマンド名	Abort
説明	OTA 処理の中断を 2nd MCU に知らせます。
2nd MCU からのレスポンス	成功 or 失敗

表 3-6 GetPlatformImageState の説明

コマンド名	GetPlatformImageState
説明	2nd MCU のファームウェアの状態を要求します。
2nd MCU からのレスポンス	現在のファームウェアの状態

3.4 FWUP FIT モジュールの一部変更について

FWUP FIT モジュールを使用することで、SCI チャネルを介して UART 通信で更新用ファームウェアを送り、ファームウェアアップデートを実行することができます。

しかし、FWUP FIT モジュールは SCI の 1 チャネルを占有するため、他のアプリケーション(センサデータの送信等)が該当の SCI チャネルを利用できない設計になっています。

そこで、他のアプリケーションも 1st MCU 間の通信路を利用可能にするために FWUP FIT モジュールに一部変更を加えます。

3.5 UART 通信のデータ通信プロトコルの導入

同じ SCI チャネルで「ファームウェアアップデート用の通信」と「他のアプリケーション(センサデータの送信等)」の 2 系統の通信が行われるため、データ通信プロトコルを規定する必要があります。例として、サンプルコードでは以下の図 3-7のようなプロトコルを規定し、更新用ファームウェアデータとセンサデータを通信しています。

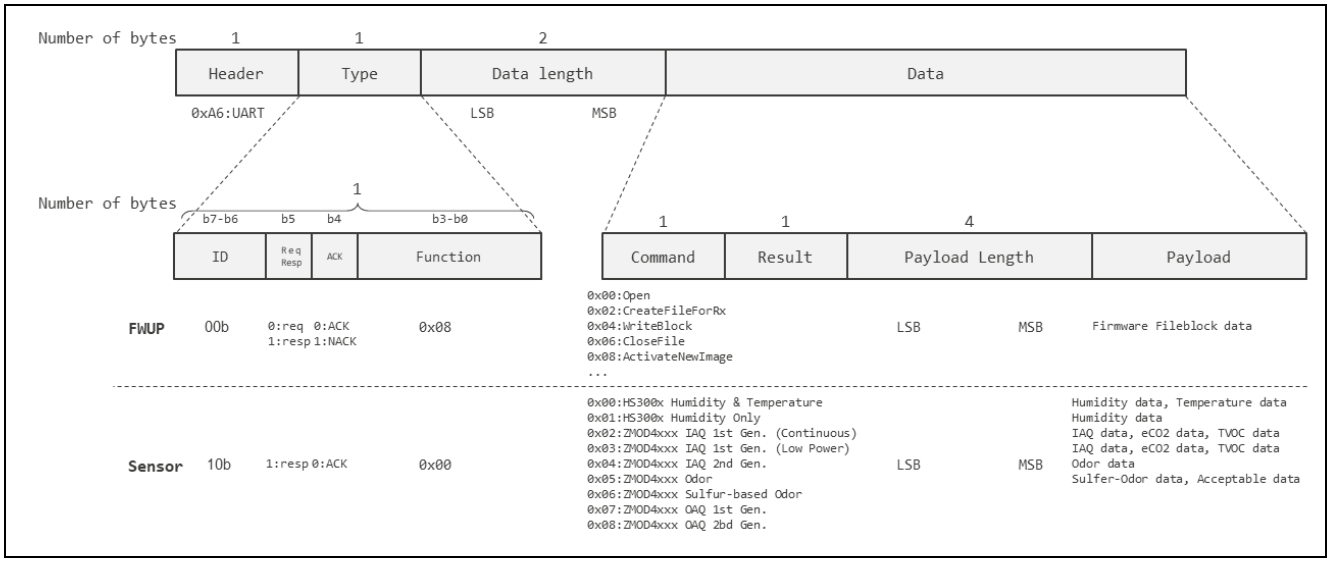


図 3-7 データ通信プロトコル例

4. 1st MCU 側のソフトウェア実装例

FreeRTOS AWS Reference Integrations Version 202107.00 を使用します。1st MCU OTA 新たに実装が必要な箇所または既存のコードに修正が必要な箇所は以下の 3 点です。

1. vendors/renesas/boards/rx65n-rsk/ports/ota_pal_for_aws/ota_second_pal.c (新規作成)
2. vendors/renesas/boards/rx65n-rsk/ports/ota_pal_for_aws/ota_pal_wrapper.c (新規作成)
3. demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c (修正)

4.1 ota_second_pal.c ファイルの新規作成

ota_second_pal.c ファイルを作成し、OTA PAL2 関数群を定義します。

OTA PAL2 関数には、

1. 2nd MCU にコマンドを送信する
2. 2nd MCU からのレスポンスを待機する
3. 2nd MCU からレスポンスが返ってきたら内容を確認して戻り値を返す

の一連の処理を実装します。

実装の一例は以下のアプリケーションノートの OTA Library v3.x.x の場合の RSK+RX65N-2MB ボードのサンプルコードを参考にしてください。

RX65N グループ FreeRTOS を用いたAmazon Web Service によるセカンダリデバイスのOTAアップデートサンプルコード

ここでは CreateFileForRx の OTA PAL2 関数 otaPal_Second_createFileForRx()の例を示します。

```
OtaPalStatus_t otaPal_Second_CreateFileForRx( OtaFileContext_t * const pFileContext )
{
    OtaPalStatus_t eResult     = OtaPalUninitialized;
    uint8_t   recv_pkt[ 8 ] = { 0 };

    create_send_packet( s_send_packet, FWUP_2nd_Command_LoadJob, 0, 0 );

    send_packet_to_secondary();

    if( SUCCESS_ARRIVE_PACKET == wait_arrive_packet( recv_pkt ) )
    {
        if( is_received_packet_intended_content( recv_pkt, FWUP_2nd_Command_LoadJob_Finish ) )
        {
            eResult = OtaPalSuccess;
        }
        else
        {
            eResult = OtaPalRxFileCreateFailed;
        }

        pFileContext->pFile = ( uint8_t * )pFileContext; /* Casting to uint8_t * type is valid */
        return eResult;
    }
    else
    {
        return OtaPalRxFileCreateFailed;
    }
}
```

以下に各 OTA PAL2 関数の処理内容を示します。

4.1.1 OTA PAL2 関数仕様

表 4-1 otaPal_Second_CreateFileForRx 関数仕様

関数名		otaPal_Second_CreateFileForRx
引数		OtaFileContext_t *const pFileContext
戻り値	型	OtaPalStatus_t
	成功時	OtaPalSuccess
	失敗時	OtaPalRxFileCreateFailed
備考		引数として受け取った pFileContext のメンバ pFile に自身のポインタを代入する必要があります。これを行わない場合、以降の処理でエラーとなります。 `C->pFile = (uint8_t *)C;`

表 4-2 otaPal_Second_WriteBlock 関数仕様

関数名		otaPal_Second_WriteBlock
引数		OtaFileContext_t *const pFileContext, uint32_t offset, uint8_t *const pData, uint32_t blockSize
戻り値	型	OtaPalStatus_t
	成功時	blockSize
	失敗時	-2
備考		引数として受け取った pData には、書き込む新ファームウェアイメージの断片が格納されているメモリの先頭アドレスが入っています。また、同じく引数として受け取った blockSize には書き込む新ファームウェアイメージの断片のファイルサイズ(byte)が入っています。 よって、pData の先頭から blockSize 分だけ取り出し、2nd MCU に送信します。 成功時は書き込んだ新ファームウェアイメージの断片のファイルサイズ(blockSize)を、失敗時は負の整数値を返します。

表 4-3 otaPal_Second_CloseFile 関数仕様

関数名		otaPal_Second_CloseFile
引数		OtaFileContext_t *const pFileContext
戻り値	型	OtaPalStatus_t
	成功時	OtaPalSuccess
	失敗時	OtaPalSignatureCheckFailed

表 4-4 otaPal_Second_ActivateNewImage 関数仕様

関数名		otaPal_Second_ActivateNewImage
引数		OtaFileContext_t *const pFileContext
戻り値	型	OtaPalStatus_t
	成功時	OtaPalSuccess
	失敗時	OtaPalActivateFailed

表 4-5 otaPal_Second_Abort 関数仕様

関数名		otaPal_Second_Abort
引数		OtaFileContext_t *const pFileContext
戻り値	型	OtaPalStatus_t
	失敗時	OtaPalSuccess
	成功時	OtaPalAbortFailed
備考		引数として受け取った pFileContext のメンバ pFile に 0 を代入します。 `C->pFile = (uint8_t *)0;`

表 4-6 otaPal_Second_SetPlatformImageState 関数仕様

引数		OtaFileContext_t *const pFileContext, OtaImageState_t eState
戻り値	型	OtaPalStatus_t
	成功時	OtaPalSuccess
	失敗時	OtaPalBadImageState
備考		2nd MCU OTA アップデート時には呼ばれませんが、PAL として定義されているため用意します。

表 4-7 otaPal_Second_GetPlatformImageState 関数仕様

引数		OtaFileContext_t *const pFileContext
戻り値	型	OtaPalImageState_t
	成功時	現在の ImageState
	失敗時	OtaPalImageStateUnknown
備考		2nd MCU OTA アップデート時は処理が正常であれば常に OtaPalImageStateValid を返します。

表 4-8 otaPal_Second_ResetDevice 関数仕様

引数		OtaFileContext_t *const pFileContext
戻り値	型	OtaPalStatus_t
	成功時	OtaPalSuccess
	失敗時	OtaPalAbortFailed
備考		2nd MCU OTA アップデート時は 1st MCU のリセットは行わないため、ただ OtaPalSuccess を返します。

4.2 ota_pal_wrapper.c, ota_pal_wrapper.h ファイルの新規作成

ota_pal_wrapper.c ファイルを作成し、OTA PAL の関数に渡される OtaFileContext_t 型の引数の fileType メンバの値によって 1st MCU 自身の OTA PAL 関数と 2nd MCU OTA アップデート用の OTA PAL2 関数を切り替えるラッパー関数を定義します。

ここでは CreateFileForRx 用のラッパー関数 otaPalWrap_createFileForRx()の例を示します。

filetype が 0 の場合は OTA PAL 関数、1 の場合は OTA PAL2 関数を実行するようにしています。

```
OtaPalStatus_t otaPalWrap_CreateFileForRx(OtaFileContext_t * const pFileContext)
{
    OtaPalStatus_t eResult = OtaPalUninitialized;
    uint32_t otaTargetID = pFileContext->fileType;

    if (otaTargetID == 0)
    {
        eResult = otaPal_CreateFileForRx( pFileContext );
    }
    else if (otaTargetID == 1)
    {
        eResult = otaPal_Second_CreateFileForRx( pFileContext );
    }
    else
    {
        eResult = OtaPalRxFileCreateFailed;
    }
    return eResult;
}
```

他の OTA PAL についても同様に fileType を参照し、実行する関数を切り替えるようなラッパー関数を作成してください。

続いて、ota_pal_wrapper.h ファイルを作成し、上記で定義したラッパー関数の宣言を記述します。

```
OtaPalStatus_t otaPalWrap_Abort( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_CreateFileForRx( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_CloseFile( OtaFileContext_t * const pFileContext );
int16_t otaPalWrap_WriteBlock( OtaFileContext_t * const pFileContext,
                               uint32_t ulOffset,
                               uint8_t * const pData,
                               uint32_t ulBlockSize );
OtaPalStatus_t otaPalWrap_ActivateNewImage( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_SetPlatformImageState( OtaFileContext_t * const pFileContext,
                                                 OtaImageState_t eState );
OtaPalImageState_t otaPalWrap_GetPlatformImageState( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_ResetDevice( OtaFileContext_t * const pFileContext );
```

4.3 ota_demo_core_mqtt.c ファイルの修正

ota_demo_core_mqtt.c ファイル内の以下の 3 箇所を修正、追記します。

1. ota_pal_wrapper.h をインクルードします。

```
#include "ota_pal_wrapper.h"
```

2. prvSetOtaInterfaces 関数内で OTA Library 内で呼ばれる OTA PAL インターフェースの初期化が行われています。ここで先ほど作成した OTA PAL ラッパー関数を設定します。

```
    /* Initialize the OTA library PAL Interface.*/
```

```
    pxOtaInterfaces->pal.getPlatformImageState = otaPalWrap_GetPlatformImageState;
    pxOtaInterfaces->pal.setPlatformImageState = otaPalWrap_SetPlatformImageState;
    pxOtaInterfaces->pal.writeBlock = otaPalWrap_WriteBlock;
    pxOtaInterfaces->pal.activate = otaPalWrap_ActivateNewImage;
    pxOtaInterfaces->pal.closeFile = otaPalWrap_CloseFile;
    pxOtaInterfaces->pal.reset = otaPalWrap_ResetDevice;
    pxOtaInterfaces->pal.abort = otaPalWrap_Abort;
    pxOtaInterfaces->pal.createFile = otaPalWrap_CreateFileForRx;
```

3. prvOtaAppCallback 関数内で switch 文の case に以下を追加します。OtaJobEventUpdateComplete は fileType≠0 の OTA アップデートジョブ完了時に呼ばれます。ここで OTA_ActivateNewImage 関数を実行し、2nd MCU のファームウェアを更新するためにソフトウェアリセットのコマンドを送信します。

```
        case OtaJobEventUpdateComplete:
            if(pData != NULL)
            {
                const OtaJobDocument_t *pJobDoc = (const OtaJobDocument_t *)pData;
                if(pJobDoc->fileTypeId == 1)
                {
                    OTA_ActivateNewImage();
                }
            }

            break;
```

5. 2nd MCU 側のソフトウェア実装例

2nd MCU 側のプログラムは Bootloader と User Program の 2 つのプロジェクトで構成されます。両プロジェクト共に FWUP FIT モジュールのデモプロジェクトを利用します。

5.1 Bootloader プロジェクトの作成

まず、Bootloader プロジェクトは次の手順で作成してください。

1. FWUP FIT モジュールのサンプルコードから rx231-rsk 用の boot_loader プロジェクトをインポートします。
2. スマートコンフィグレータを用いてターゲットデバイスを TB-RX23W ボードに変更します。
3. プロジェクトフォルダ内の boot_loader/src/src/tinycrypt フォルダは空になっているため、以下 URL より Tinycrypt ライブラリを取得し lib フォルダを boot_loader/src/src/tinycrypt に追加します。

   amazon-freertos/libraries/3rdparty/tinycrypt at master · renesas/amazon-freertos (github.com)
4. boot_loader/src/key/code_signer_public_key.h ファイルの CODE_SIGNER_PUBLIC_KEY_PEM にデジタル署名に使用する公開鍵情報を入力します。入力方法は以下のリンクの

   4. ファームウェア検証に使用する鍵を OpenSSL で作成する

   5. ファームウェア検証に ECDSA+SHA256 を使用するため、ブートローダに署名検証用の公開鍵 =secp256r1.publickey を仕込む

   を参照してください。

   OTAの活用 · renesas/amazon-freertos Wiki (github.com)

5.2 User Program プロジェクトの作成

次に、User Program プロジェクトは次の手順で作成してください。

1. FWUP FIT モジュールのサンプルコードから rx231-rsk 用の fwup_main プロジェクトをインポートします。
2. 上記の boot_loader プロジェクトの作成で行ったターゲットデバイスの変更、Tinycript の追加、公開鍵情報の入力を同様に行います。
3. FWUP FIT モジュールのソースコードを 2nd MCU OTA アップデートに合わせて一部変更します。

   ① UART 通信で受信したファームウェアデータを FWUP FIT モジュールが受け取るための関数を作成します。r_fwup.c ファイル内に以下の関数を追加します。

```
void fwup_receive_fileblock(uint8_t received_packet[])
{
    if (s_sci_receive_control_block.p_sci_buffer_control->buffer_full_flag ==
FWUP_SCI_RECEIVE_BUFFER_EMPTY)
    {
        for (uint16_t i = 0; i < 1024; ++i)
        {
            s_sci_receive_control_block.p_sci_buffer_control->buffer[i] = received_packet[8 + i];
            s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size++;
        }
#if (FLASH_CFG_CODE_FLASH_BGO == 0)
                /* RTS HIGH */
        FWUP_CFG_PORT_SYMBOL.PODR.BIT.FWUP_CFG_BIT_SYMBOL = 1;  /* Set RTS to HIGH */
        FWUP_CFG_PORT_SYMBOL.PDR.BIT.FWUP_CFG_BIT_SYMBOL = 1;
        FWUP_CFG_PORT_SYMBOL.PMR.BIT.FWUP_CFG_BIT_SYMBOL = 0;   /* Change to general I/O port */
#endif /* FLASH_CFG_CODE_FLASH_BGO == 0 */
        s_sci_receive_control_block.total_byte_size +=
            s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size;
        s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size = 0;
```

```
        s_sci_receive_control_block.p_sci_buffer_control->buffer_full_flag =
FWUP_SCI_RECEIVE_BUFFER_FULL;

        if (FWUP_SCI_CONTROL_BLOCK_A == s_sci_receive_control_block.current_state)
        {
            s_sci_receive_control_block.current_state = FWUP_SCI_CONTROL_BLOCK_B;
            s_sci_receive_control_block.p_sci_buffer_control =
&s_sci_buffer_control[FWUP_SCI_CONTROL_BLOCK_B];
        }
        else
        {
            s_sci_receive_control_block.current_state = FWUP_SCI_CONTROL_BLOCK_A;
            s_sci_receive_control_block.p_sci_buffer_control =
&s_sci_buffer_control[FWUP_SCI_CONTROL_BLOCK_A];
        }

    }
}
```

上のソースコード内の、

```
        for (uint16_t i = 0; i < 1024; ++i)
        {
            s_sci_receive_control_block.p_sci_buffer_control->buffer[i] = received_packet[8 + i];
            s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size++;
        }
```

には受信した WriteBlock コマンドに付加されている新ファームウェアの断片(1024byte)を

s_sci_receive_control_block.p_sci_buffer_control->buffer に格納し、

s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size をインクリメントします。

② FWUP FIT モジュール外部から Abort 関数を呼べるように、引数なしの Abort 関数のラッパー関数を作成します。

```
fwup_err_t R_FWUP_Abort_global( void )
{
    return (fwup_err_t)R_FWUP_Abort( &g_file_context );
}
```

③ r_fwup_if.h ファイルに、①, ②で追加した 2 つの関数の宣言を追加します。

```
void fwup_receive_fileblock(uint8_t []);
fwup_err_t R_FWUP_Abort_global(void);
```

5.2.1 2nd OTA Controller の実装

1st MCU からのコマンドをイベントとして、FWUP FIT モジュールの API 関数を呼ぶ 2nd OTA Controller の状態遷移の設計例を以下に示します。

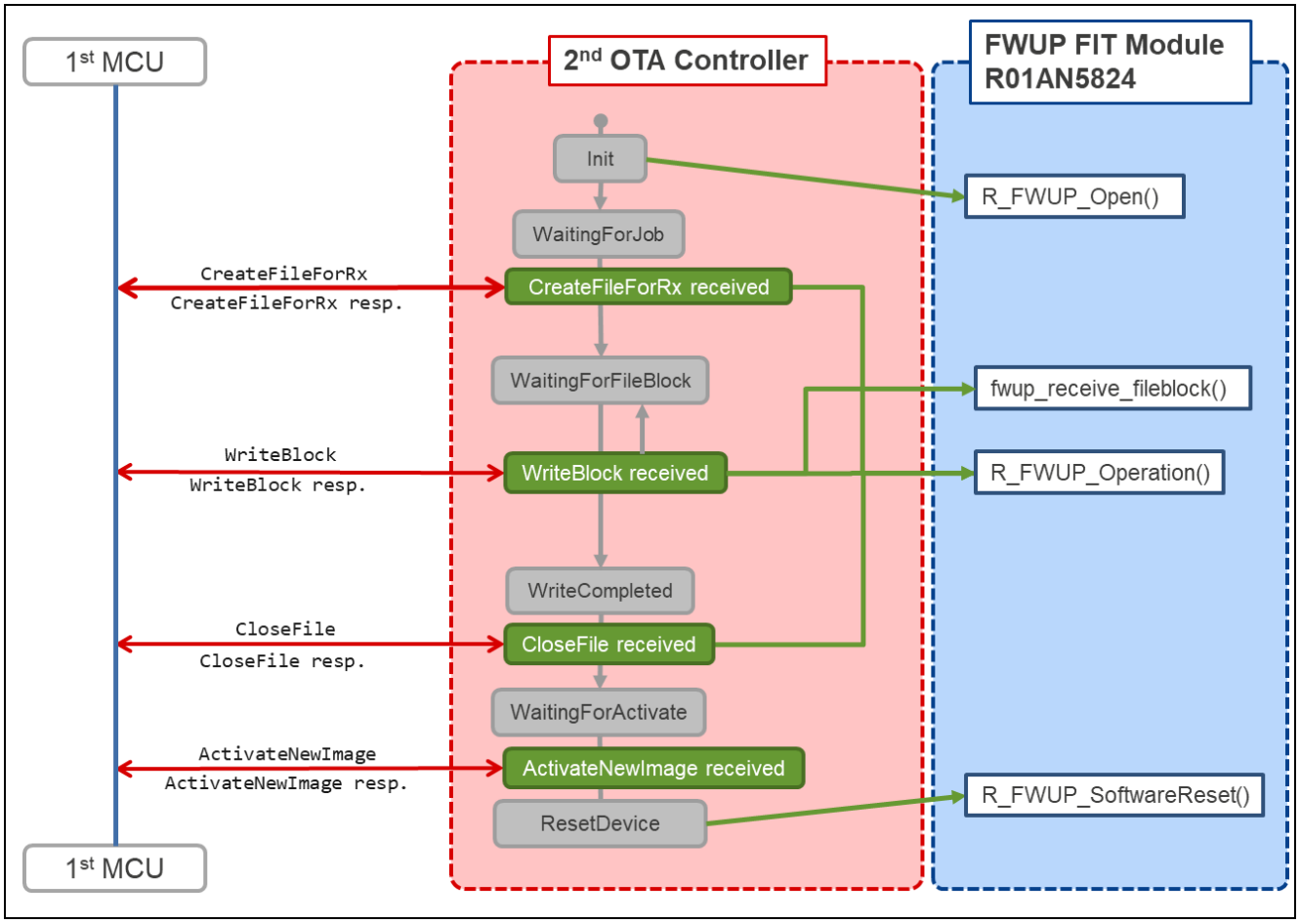


図 5-1 2nd OTA Controller の状態遷移設計例

図中右の青い四角内が FWUP FIT モジュールの API 関数、赤い四角内が 2nd OTA Controller の状態遷移図です。2nd OTA Controller の状態遷移図は、灰色の箱が状態、緑色の箱がイベントを表しており、緑色の矢印はそのイベント発生時に実行する FWUP FIT モジュールの API 関数を示しています。

図 5-1の 2nd OTA Controller の状態遷移表は以下のようになります。

表 5-1 2nd OTA Controller の状態遷移表

現在の状態	イベント	実行する関数	次の状態	
Init	-	R_FWUP_Open()	WaitingForJob	
WaitingForJob	“CreateFileForRx” コマンド受信	R_FWUP_Operation()	WaitingForFileBlock	
WaitingForFileBlock	“WriteBlock” コマンド受信	fwup_receive_fileblock() R_FWUP_Operation() R_FWUP_Operation()	※①	※②
			WriteCompleted	WaitingForFileBlock
WriteCompleted	“CloseFile” コマンド受信	R_FWUP_Operation()	WaitingForActivate	
WaitingForActivate	“ActivateNewImage” コマンド受信	R_FWUP_SoftwareReset()	-	
-	“Abort”コマンド受信 2nd OTA Controller 処理中にエラー発生	R_FWUP_Abort_global() fwup_communication_close() fwup_state_monitoring_close() fwup_flash_close() R_FWUP_Close() R_FWUP_Open()	WaitingForJob	
-	“GetPlatformImageState” コマンド受信	R_FWUP_GetPlatformImageState()	-	

※ ① fwup_get_status() == FWUP_STATE_CHECK_SIGNATURE の場合

※ ② fwup_get_status() == FWUP_STATE_DATA_RECEIVE の場合

実装の一例は以下のアプリケーションノートの OTA Library v3.x.x の場合の TB-RX23W ボードのサンプルコードを参考にしてください。

RX65N グループ FreeRTOS を用いたAmazon Web Service によるセカンダリデバイスのOTAアップデートサンプルコード

6. 2nd MCU の初期ファームウェアと更新用ファームウェアの作成方法

6.1 初期ファームウェアの作成方法

作成した bootloader プロジェクトと User Program プロジェクトをビルドし、.mot ファイルを生成します。生成した 2 つの.mot ファイルを Renesas Secure Flash Programmer を用いて結合し、初期ファームウェアの.mot ファイルを作成します。パラメータは以下の通りです。

表 6-1 Renesas Secure Flash Programmer で初期ファームウェア作成時のパラメータ設定

Select MCU	RX23W(ROM 512KB)/Secure Bootloader=64KB
Select Firmware Verification Type	sig-sha256-ecdsa
Private Key Path(PEM Format)	上で作成した秘密鍵のパス
Select Output Format	Bank 0 User Program + Boot Loader (Motorola S Format)
Boot Loader File Path (Motorola Format)	生成した bootloader の.mot ファイルのパス
Bank0 User program Firmware Sequence Number	1
Bank0 File Path (Motorola Format)	生成した User Program の.mot ファイルのパス

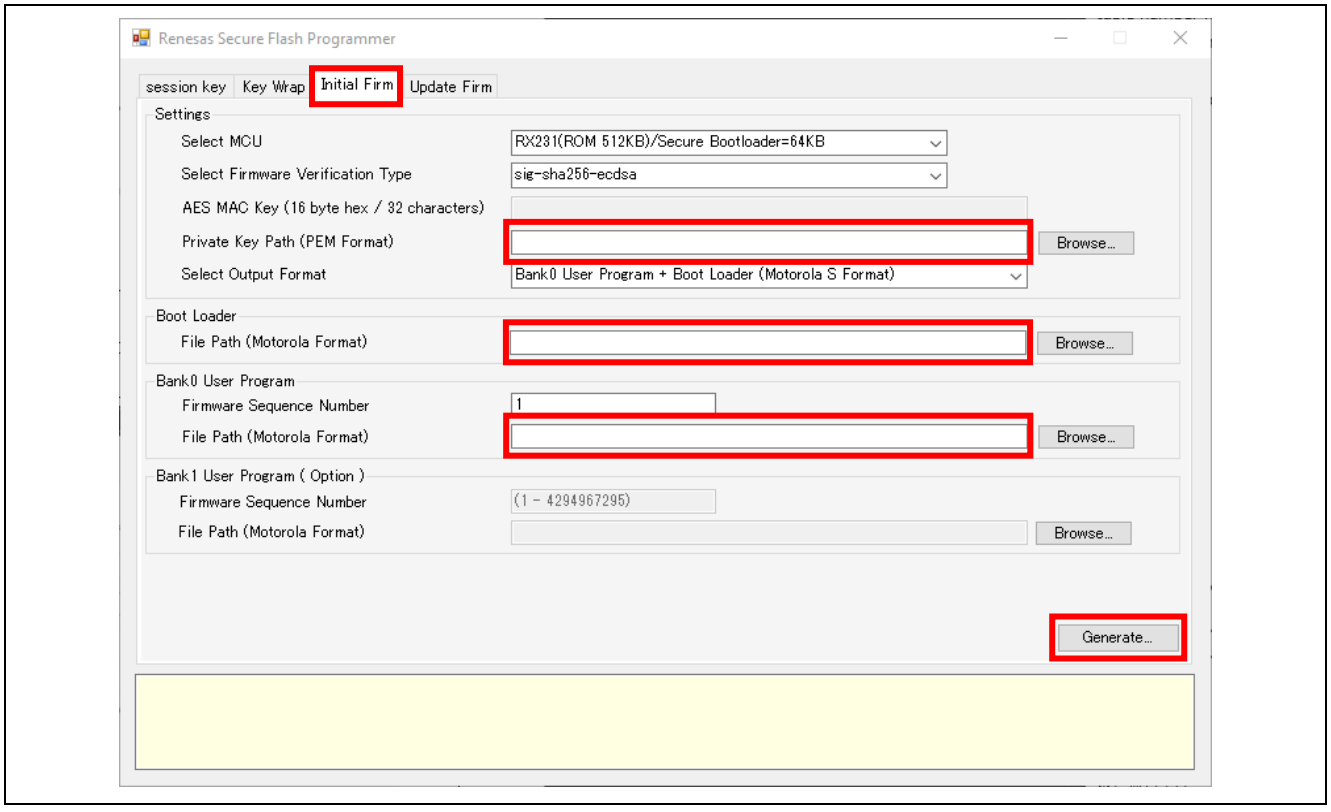


図 6-1 Renesas Secure Flash Programmer で初期ファームウェア作成時の設定画面

6.2 更新用ファームウェアのアップデート用ファイルの作成方法

作成した更新用 User Program プロジェクトをビルドし、.mot ファイルを生成します。生成した.mot ファイルを Renesas Secure Flash Programmer を用いてルネサス独自のバイナリ形式のファームウェアデータフォーマットである.rsu ファイルに変換します。RSU(Renesas Secure Update)形式に関しての詳細は、

ルネサスMCUにおけるファームウェアアップデートの設計方針 Rev.1.00 (renesas.com)

の第 7 章を参照してください。

表 6-2 Renesas Secure Flash Programmer で更新用ファームウェアの
アップデート用ファイル作成時のパラメータ設定

Select MCU	RX23W(ROM 512KB)/Secure Bootloader=64KB
Select Firmware Verification Type	sig-sha256-ecdsa
Private Key Path(PEM Format)	上で作成した秘密鍵のパス
Bank0 User program Firmware Sequence Number	1
Bank0 File Path (Motorola Format)	生成した更新用ファームウェアの MOT ファイルのパス

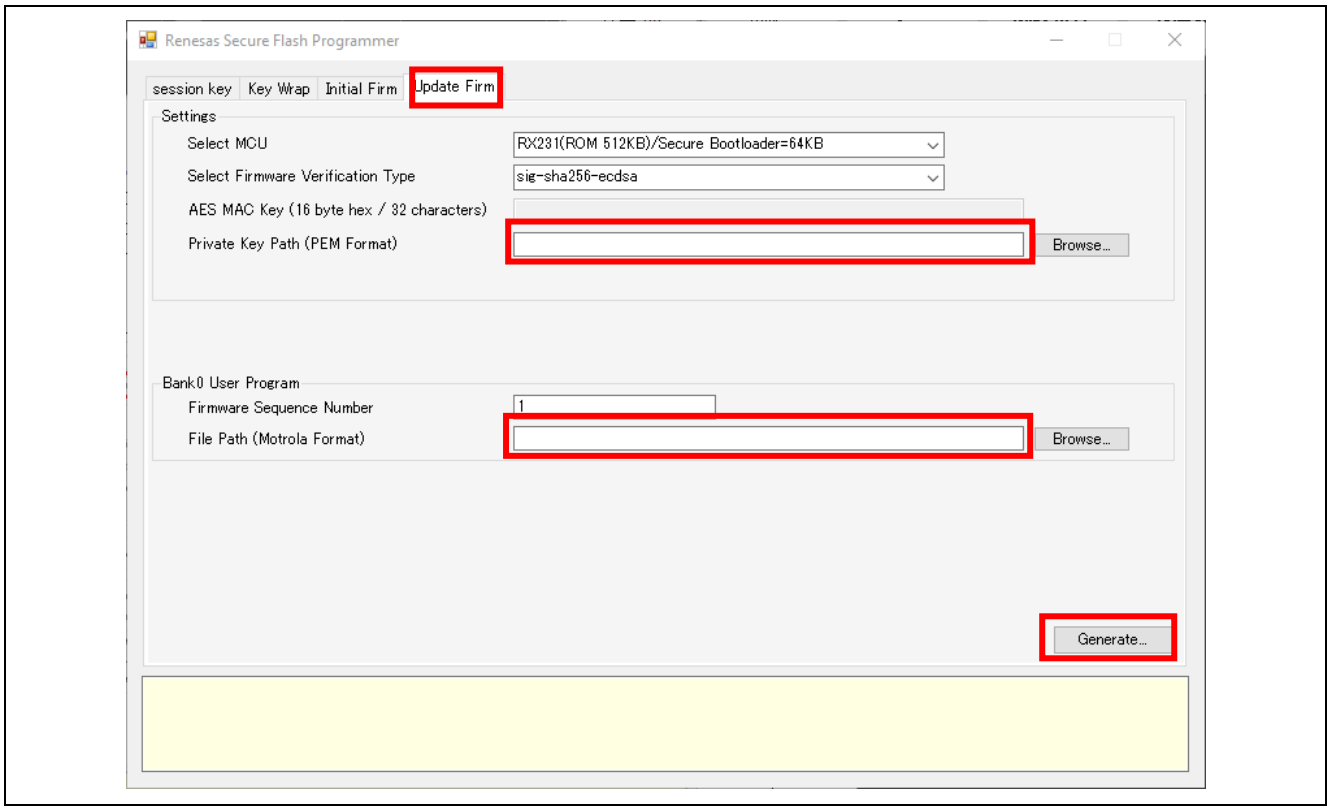
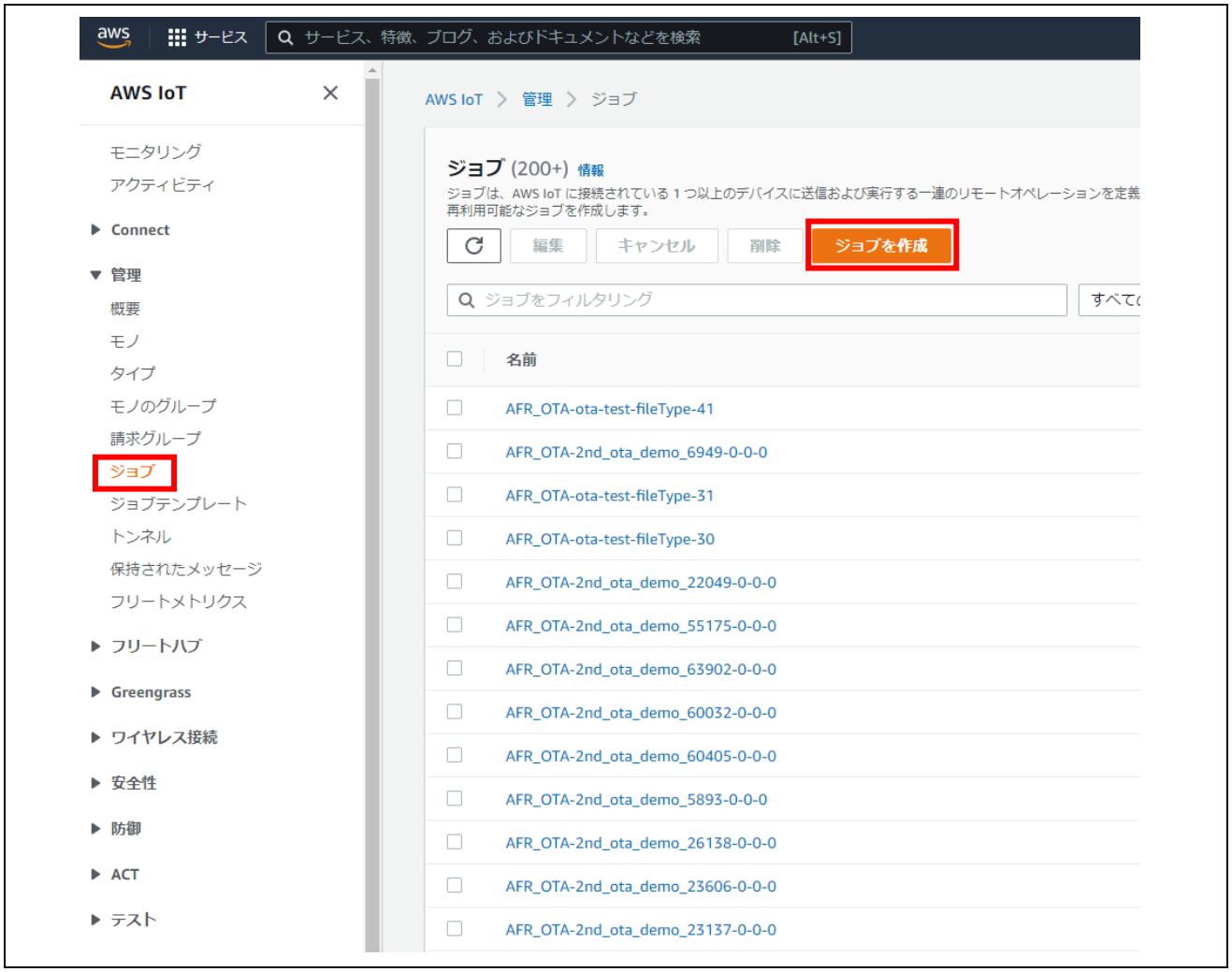


図 6-2 Renesas Secure Flash Programmer で更新用ファームウェアの
アップデート用ファイル作成時の設定画面

7. 2nd MCU OTA アップデートの実行方法

AWS IoT Core ブラウザコンソール上で 2nd MCU OTA アップデートジョブを発行する方法について説明します。

1. AWS IoT Core ブラウザコンソールを開き、左のメニューから**管理 ⇒ ジョブ**を選択し、**[ジョブの作成]**をクリックします。



2. 「ジョブを作成」画面で**「FreeRTOS OTA 更新ジョブを作成」**を選択して**[次へ]**をクリックします。

3. 「OTA ジョブのプロパティ」画面で**「ジョブ名」**を入力し、**[次へ]**をクリックします。

4. 「OTA ファイル設定」画面で各種項目を入力します。

① **「更新するデバイス」**は AWS IoT サービスに登録されている 1st MCU のモノ名を入力してください。

② **「コード署名プロファイル」**は任意のプロファイルを指定します。
※ ここで指定したコード署名プロファイルは 2nd MCU のファームウェアのコード署名検証には使用されません。コード署名は Renesas Secure Flash Programmer で更新用 rsu ファイルを作成した際にファイル内に記述されます。そのため、2nd MCU の新ファームウェア作成時に使用した証明書とプライベートキーを用いてコード署名プロファイルを作成する必要はありません。

③ **「アップロードするファイル」**は 2nd MCU の新ファームウェア(.rsu 形式)を選択してください。

④ **「ファイルタイプ」**にはファームウェア作成時に定義した 2nd MCU OTA アップデート用の値を入力してください。

上記以外の項目は通常の OTA ジョブ作成時と同じ設定を入力し、**[次へ]**をクリックします。


aws
サービス
サービス、特徴、ブログ、およびドキュメントなどを検索 [Alt+S]
AWS IoT > ジョブ > ジョブの作成 > OTA ジョブ
Step 1
OTA ジョブのプロパティ
Step 2
OTA ファイル設定
Step 3
OTA ジョブ設定
OTA ファイル設定 情報
デバイス 情報
この OTA 更新ジョブは、MQTT または HTTP 経由で、FreeRTOS ベースのモノや選択したモノのグループにファイルを安全に送信します。
更新するデバイス
モノおよび/またはモノのグループを選択する
test_mono
ファイル転送のプロトコルを選択します
デバイスがサポートするプロトコルを選択します。
MQTT
HTTP
ファイル 情報
ファイルに署名して選択する
コード署名により、信頼された作成者によって発行されたコードのみがデバイスで実行され、署名後にコードが変更または破損していないことが保証されます。コード署名には 3 つのオプションがあります。
新しいファイルに署名します。
以前に署名したファイルを選択します。
カスタム署名ファイルを使用します。
コード署名プロファイル
このプロファイルには、コード署名ジョブの作成に必要な情報が含まれています。このプロファイルは、デバイスのハードウェアプラットフォーム、AWS Certificate Manager からの証明書、およびデバイス上のコード署名証明書のパスの場所を指定します。
既存のコード署名プロファイル
新しいプロファイルの作成
ファイル
新しいファイルをアップロードします。
既存のファイルを選択します。
アップロードするファイル
ファイルを選択
rx23w_tb_2ndota_demo_update.rsu
229376 バイト
S3 でのファイルのアップロード先
これは、S3 内でファイルが保存されるロケーションです。
S3 の URL
表示
S3 をブラウズ
S3 バケットを作成
形式: s3://bucket/prefix/object
デバイス上のファイルのパス名
これは、FreeRTOS デバイス上のファイルの保存先の名前と場所です。
hoge
▼ ファイルタイプ - オプション
ファイルタイプ
これはジョブドキュメントに含まれ、AWS クラウドから受信したファイルのタイプをデバイスが識別できるようにします。
1
IAM ロール 情報
ロール
S3、AWS IoT ジョブ、AWS コード署名リソースへの AWS IoT アクセス権限を付与するロールを選択します。
OtaDemoServiceRole
キャンセル
戻る
次へ

5. 「OTA ジョブ設定」画面は変更を加える必要はないため、そのまま**[ジョブの作成]**をクリックします。

以上の手順で 2nd MCU OTA アップデート用の OTA ジョブが作成され、指定したデバイスに向けて OTA ジョブが配信されます。

作成した OTA ジョブのジョブドキュメントを確認すると、「”filetype”: 1」というメンバが追加されていることが確認できます。このジョブドキュメントを 1st MCU が受け取り、2nd MCU OTA アップデート用の処理に分岐します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2022/03/31	—	新規作成
1.01	2022/06/30	1	要旨 本アプリケーションノートで習得可能な内容に関して、文章を見直し
		3	目次を更新
		4	1.1 文章の表現を修正
		5	1.4 見出しの誤記を修正
		7	2.2.1 文章中の「赤枠」を「赤箱」に修正
		8	2.2.3 OTA Library v1.x.x 系についての記述を削除
		13	図 3-7 図中の誤記を修正
		14	4 実装例であることを明示するため見出しを修正
		19	5 実装例であることを明示するため見出しを修正
		21	図 5-1 図中のオブジェクトの配色の変更と本文中の説明の修正

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。