R01AN5824JJ0102
Rev.1.02
2021.10.29

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

要旨

本アプリケーションノートは Firmware Integration Technology (FIT)を使ったファームウェアアップデートモジュールについて説明します。以降、本モジュールをファームウェアアップデート FIT モジュールと称します。

本アプリケーションノードは、「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(R01AN5548)」をベースに記載しておりますので、こちらのアプリケーションノートを先に読まれることをお勧めします。

本 FIT モジュールを使って、ファームウェアアップデート機能をユーザアプリケーションに容易に組み込むことができます。本アプリケーションノートでは、ファームウェアアップデート FIT モジュールの使用、およびユーザアプリケーションへの取り込みについて説明します。

動作確認デバイス

RX130 グループ

RX230、RX231、RX23E-A、RX23W グループ

RX65N、RX651 グループ

RX66N グループ

RX66T グループ

RX671 グループ

RX72M グループ

RX72N グループ

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あわせて変更し、十分評価してください。

関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。あわせて参照してください。

- Renesas 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(R01AN5548)
- 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(R01AN5549)
- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- RX ファミリ e2 studio に組み込む方法 Firmware Integration Technology(R01AN1723)
- RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology(R01AN1685)
- RX ファミリ フラッシュモジュール Firmware Integration Technology(R01AN2184)
- RX ファミリ SCI モジュール Firmware Integration Technology(R01AN1815)
- RX ファミリ イーサネットモジュール Firmware Integration Technology(R01AN2009)
- RX ファミリ CMT モジュール Firmware Integration Technology(R01AN1856)

- RX ファミリ バイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology(R01AN1683)
- RX ファミリ システムタイマモジュール Firmware Integration Technology(R20AN0431)

ターゲットコンパイラ

- ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family
- GCC for Renesas RX

各コンパイラの動作確認環境に関する詳細な内容は、セクション「5.1 動作確認環境」を参照してください。

目次

1. 概要

1.1 ファームウェアアップデートとは

ファームウェアアップデートとは、機器の制御を行うファームウェアを新しいファームウェアに書き換えることです。ファームウェアアップデートは不具合の修正や新機能の追加、性能向上などのために行われます。

RX ファミリの場合、ファームウェアは MCU に内蔵されたフラッシュメモリに書き込まれています。したがって RX ファミリでは、この MCU 内蔵フラッシュメモリの内容を書き換える操作や処理をファームウェアアップデートと呼びます。

通常、MCU に内蔵されたフラッシュメモリの内容を書き換える方法には、以下の 2 種類があります。

- オフボードプログラミング
  外部にフラッシュライタ等のフラッシュ書き換え機器を接続しフラッシュメモリを書き換える方法
- オンボードプログラミング（セルフプログラミング）
  MCU を動作させ MCU 自身で内蔵フラッシュメモリを書き換える方法

ファームウェアアップデートは後者のセルフプログラミング機能を使い、MCU 自身が内蔵しているフラッシュメモリを書き換えます。

フラッシュメモリをセルフプログラミングで書き換える場合、RAM にフラッシュメモリを書き換えるプログラムを配置した後、RAM 上からフラッシュメモリを書き換えるコマンドを実行する必要があります。また、新しいバージョンのファームウェアを各種のインタフェースから取得する必要もあり、お客様のシステムにファームウェアアップデートの機能を構築することは非常に難しい物でした。

本ファームウェアアップデートの FIT モジュールを使うことで、ファームウェアアップデートの機能をお客様のシステムに容易に組み込むことができます。

ファームウェアアップデートモジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.10 FIT モジュールの追加方法」を参照してください。

1.2 ファームウェアアップデートモジュールの構成

ファームウェアアップデートモジュールはファームウェアをアップデートすることを目的としたミドルウェアです。

ファームウェアアップデートモジュールには OS 無しのシステム向けの機能と FreeRTOS（OTA）を用いたシステム向けの機能があります。FreeRTOS（OTA）の詳細については、以下のリンクを参照してください：https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/freertos-ota-dev.html

OS 無しのシステムにおけるファームウェアアップデートのシステム構成を図 1-1 に、FreeRTOS（OTA）を用いたシステムにおけるファームウェアアップデートのシステム構成を図 1-2 に示します。

ブートローダモジュールとは、リセット後に最初に実行され、ユーザプログラム（ブートローダ実行後に実行するプログラム）が改ざんされていないことを検証するモジュールです。

ファームウェアアップデートモジュールとは、ファームウェアアップデートを行うモジュールでユーザプログラムに含まれます。

ファームウェアアップデートで使用する FIT モジュールの一覧を表 1-1 に示します。

各種通信 I/F で取得したアップデート用のファームウェアは、ファームウェアアップデートモジュールとフラッシュ FIT モジュールを介してターゲットデバイス上のコードフラッシュメモリにプログラムされます。

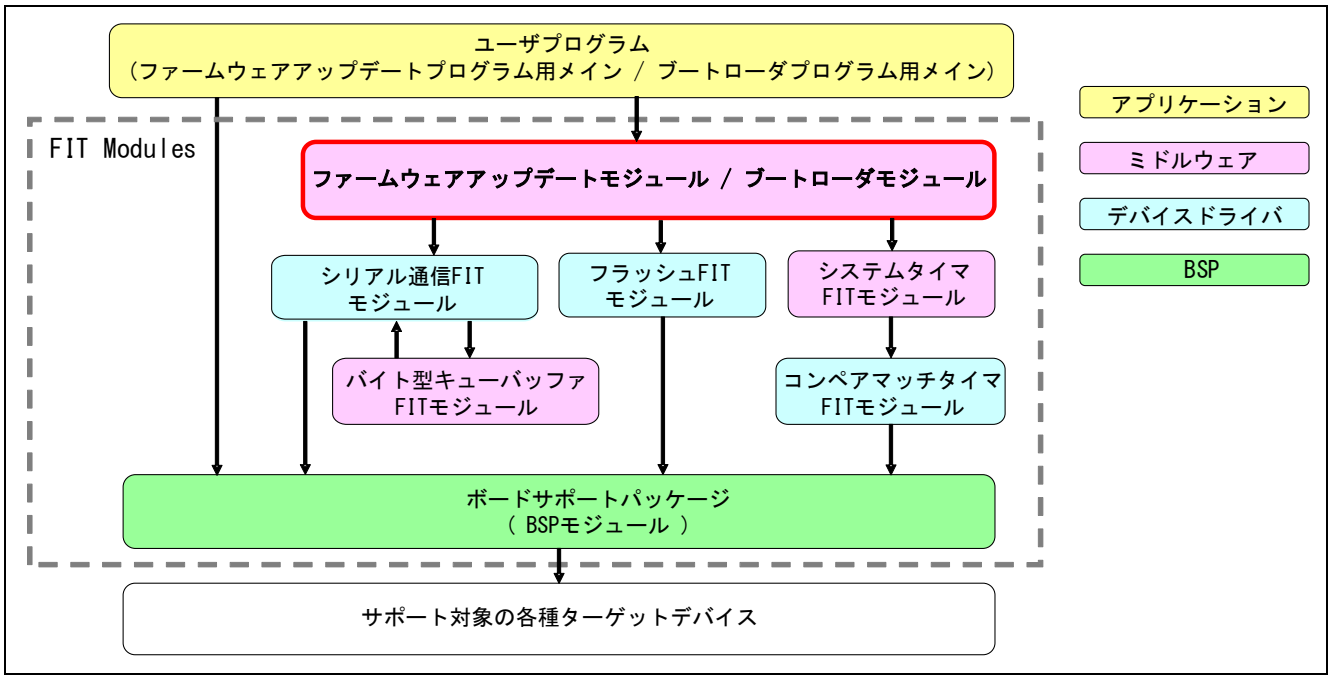


図 1-1 OS 無しのシステムにおけるファームウェアアップデートのシステム構成

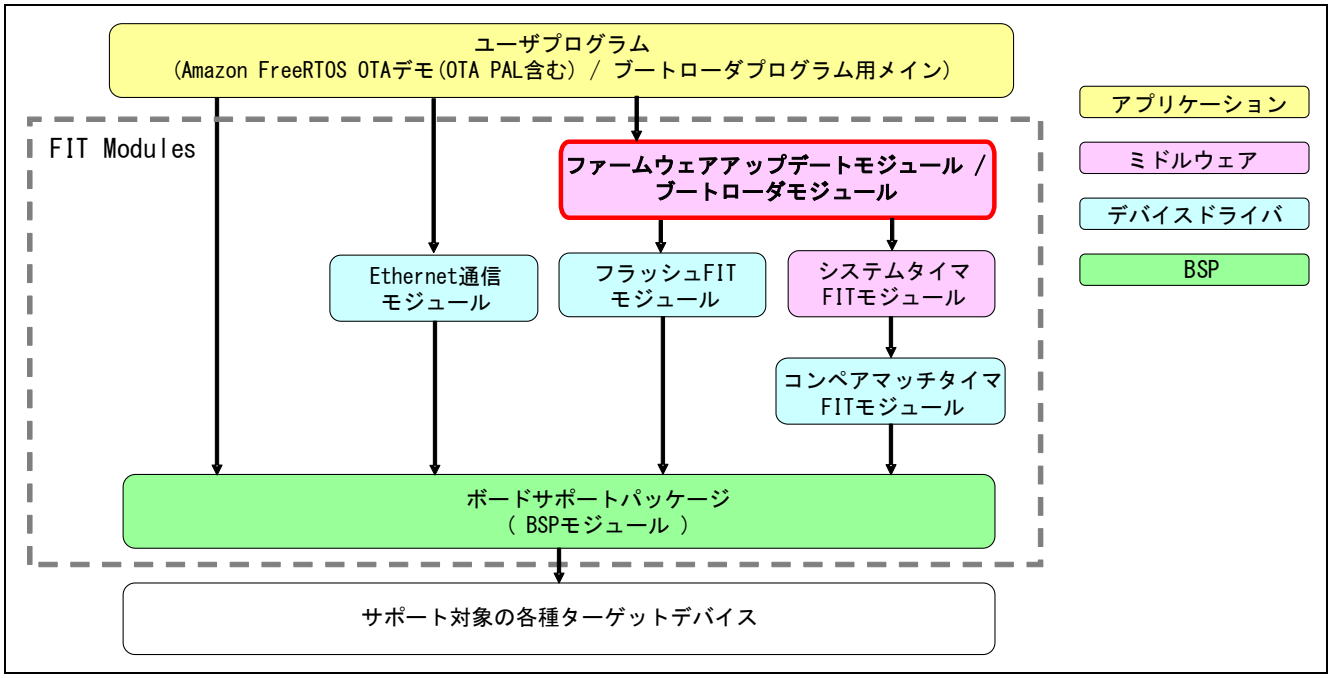


図 1-2 FreeRTOS（OTA）を用いたシステムにおけるファームウェアアップデートのシステム構成

表 1-1 モジュール一覧

種類	アプリケーションノート名（型名）	FIT モジュール名
BSP	RX ファミリボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)	r_bsp
デバイスドライバ	RX ファミリフラッシュモジュール Firmware Integration Technology (R01AN2184)	r_flash_rx
デバイスドライバ	RX ファミリ SCI モジュール Firmware Integration Technology (R01AN1815)	r_sci_rx

デバイスドライバ	RX ファミリ CMT モジュール Firmware Integration Technology (R01AN1856)	r_cmt_rx
ミドルウェア	RX ファミリバイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology (R01AN1683)	r_byteq
ミドルウェア	RX ファミリ システムタイマモジュー Firmware Integration Technology (R20AN0431)	r_sys_time_rx

1.3　ファームウェアアップデートの動作

RX ファミリには、MCU 内蔵のフラッシュメモリにデュアルバンク機能を持つ製品があります。

デュアルバンク機能を持たない製品もしくは、デュアルバンク機能でリニアモードを使用する場合、フラッシュメモリの書き換えを行うためには、RAM にフラッシュメモリを書き換えるプログラムを配置した後、RAM 上からフラッシュメモリを書き換えるコマンドを実行する必要があります。

デュアルバンク機能でデュアルモードを使用する場合、フラッシュメモリの書き換え領域と実行領域が違う領域にある場合、RAM でプログラムを実行して書き換える必要がありません。このため、システム動作を維持したままフラッシュメモリを書き換えることが非常に容易となります。

本ファームウェアアップデートモジュールは、リニアモードでもデュアルモードでも、ファームウェアアップデートが可能です。

表 1-2 デバイス毎のリニアモード／デュアルモードサポート状況

デバイス	リニアモード	デュアルモード
RX130 グループ	○	—
RX231 グループ	○	—
RX65N グループ	○	○
RX66T グループ	○	—
RX671 グループ	○	○
RX72N グループ	○	○

1.3.1　デュアルモードを使用したファームウェアアップデート動作

フラッシュメモリのデュアルモードを使用したファームウェアアップデートの動作を説明します。

ファームウェアアップデートの動作は、ファームウェアアップデートの準備処理としての内蔵フラッシュの初期設定と、ファームウェアアップデート動作に分けられます。

デュアルモードのファームウェアアップデートの初期設定の動作を図 1-3 に示します。

本 FIT モジュールでは、内蔵フラッシュに書き込む初期ファームウェアを生成するためのツール（Renesas Secure Flash Programmer）を提供しています。このツールでは、ユーザプログラムのみの初期ファームウェア、およびブートローダとユーザプログラムで構成される初期ファームウェアを生成することができます。ブートローダとユーザプログラムで構成される初期ファームウェアをフラッシュライタなどで内蔵フラッシュに書き込むことにより、図 1-3 の④の状態にすることができます。

また、ブートローダプログラムをビルドし、生成された mot ファイルを内蔵フラッシュに書き込むことにより、図 1-3 の①の状態にすることができます。内蔵フラッシュにブートローダのみを書き込んだ場合は、ブートローダの機能を使ってユーザプログラムのみの初期ファームウェアを内蔵フラシュに書き込みます。

初期設定を①と④のどちらから開始するかはお客様のシステムに合わせて選択してください。

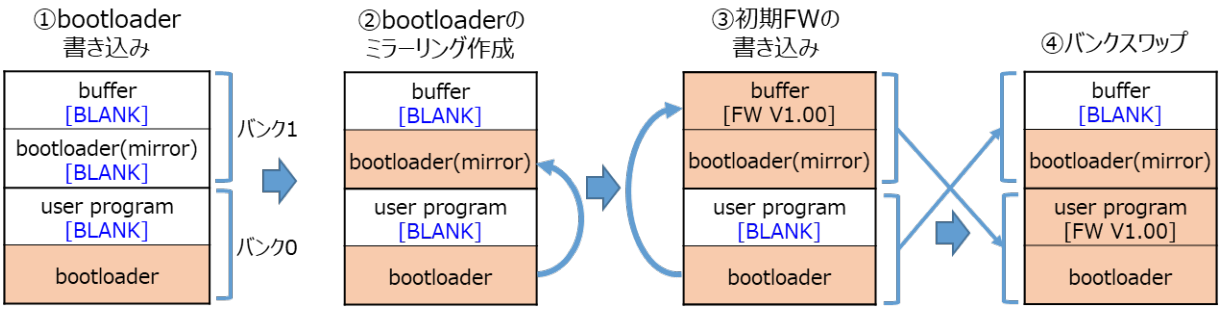


図 1-3 デュアルモードのファームウェアアップデートの初期設定

初期設定を①から開始する場合

① ブートローダをフラッシュライタ等でフラッシュメモリに書き込む。

② ブートローダを実行し、バンク 1 にブートローダをミラーリングする。

③ ブートローダにてユーザプログラムのみの初期ファームウェアを書き込み（外部から入力する必要あり）、書き込んだファームウェアを検証する。

④ 検証に問題なければ、バンクスワップを行う。

初期設定を④から開始する場合

④ ブートローダとユーザプログラムで構成される初期ファームウェアを、フラッシュライタ等でフラッシュメモリに書き込む。

デュアルモードのファームウェアアップデートの動作を図 1-4 に示します（ただし下記「①初期状態」は初回起動時にブートローダを実行し、バンク 1 にブートローダをミラーリングした状態）。

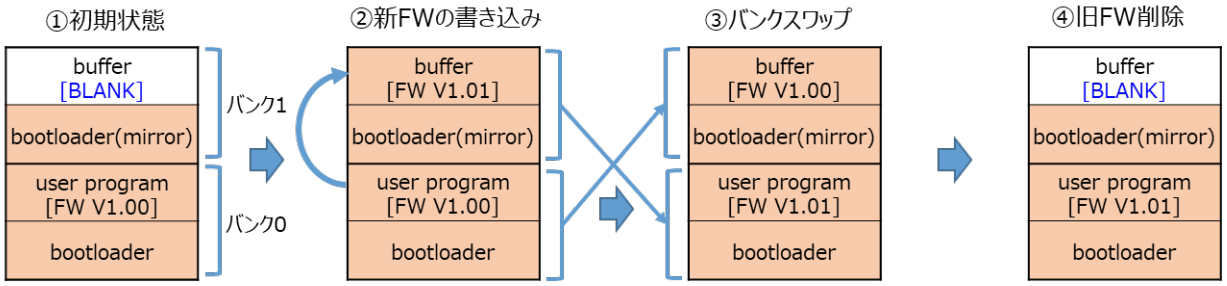


図 1-4 デュアルモードのファームウェアアップデートの動作

① 初期状態。

② ユーザプログラムに含まれるファームウェアアップデートモジュールで新しいバージョンのファームウェアを書き込み（外部から入力する必要あり）、書き込んだファームウェアを検証する。

③ 検証に問題なければ、バンクスワップを行う。

④ バンク 1 の古いファームウェアを削除する。

1.3.2 リニアモードを使用したファームウェアアップデート動作

フラッシュメモリのリニアモードを使用したファームウェアアップデートの動作を説明します。

リニアモードのファームウェアアップデートの初期設定の動作を図 1-5 に示します。

本 FIT モジュールでは、内蔵フラッシュに書き込む初期ファームウェアを生成するためのツール（Renesas Secure Flash Programmer.exe）を提供しています。このツールでは、ユーザプログラムのみの初期ファームウェア、およびブートローダとユーザプログラムで構成される初期ファームウェアを生成することができます。ブートローダとユーザプログラムで構成される初期ファームウェアをフラッシュライタなどで内蔵フラッシュに書き込むことにより、図 1-5 の②の状態にすることができます。

また、ブートローダプログラムをビルドし、生成された mot ファイルを内蔵フラッシュに書き込むことにより、図 1-5 の①の状態にすることができます。内蔵フラッシュにブートローダのみを書き込んだ場合

は、ブートローダの機能を使ってユーザプログラムのみの初期ファームウェアを内蔵フラシュに書き込みます。

初期設定を①と②のどちらから開始するかはお客様のシステムに合わせて選択してください。

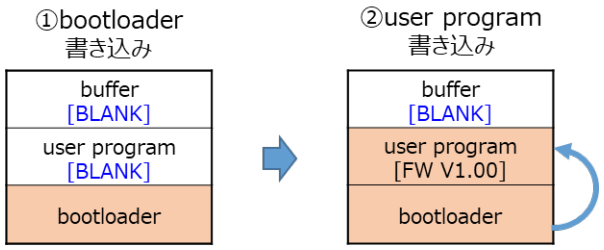


図 1-5 リニアモードのファームウェアアップデートの初期設定

初期設定を①から開始する場合

① ブートローダをフラッシュライタ等でフラッシュメモリに書き込む。

② ブートローダにてユーザプログラムのみの初期ファームウェアを書き込み（外部から入力する必要あり）、書き込んだファームウェアを検証し、検証に問題なければ終了する。

初期設定を②から開始する場合

② ブートローダとユーザプログラムで構成される初期ファームウェアを、フラッシュライタ等でフラッシュメモリに書き込む。

リニアモードのファームウェアアップデートの動作を図 1-6 に示します。

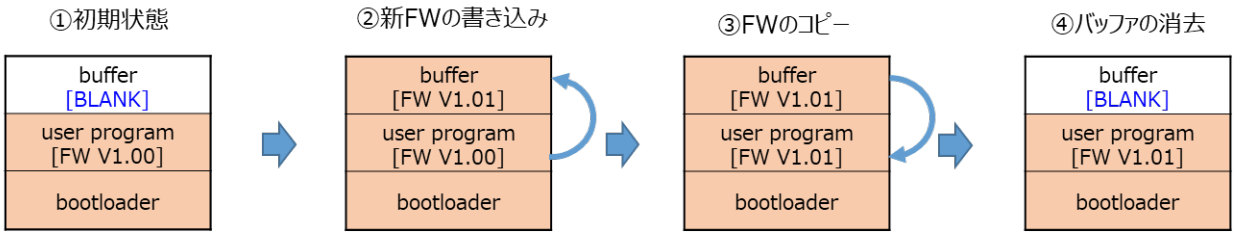


図 1-6 リニアモードのファームウェアアップデートの動作

① 初期状態。

② ユーザプログラムにて新しいバージョンのファームウェアをバッファ領域に書き込み（外部から入力する必要あり）、書き込んだファームウェアを検証する。

③ 検証に問題なければ、バッファ領域からユーザプログラム領域にコピーする。

④ バッファ領域を削除する。

1.4　API の概要

表 **1-3** にファームウェアアップデートモジュールに含まれる API 関数を示します。

表 1-3 API 関数一覧

関数	関数説明	Free RTOS (OTA)	OS レス	ブートローダモジュール
		ファームウェアアップデートモジュール	ファームウェアアップデートモジュール	
R_FWUP_Open	本モジュールのオープン処理行います。	—	○	○
R_FWUP_Close	本モジュールのクローズ処理を行います。	—	○	○
R_FWUP_Operation	ユーザプログラムからのファームウェアアップデート処理を行います。	—	○	—
R_FWUP_SoftwareReset	ソフトウェアリセットを行います。	—	○	○
R_FWUP_SetEndOfLife	ユーザプログラムの保守期間終了時(END OF LIFE)の処理を行います。	○	○	—
R_FWUP_SecureBoot	ブートローダでのセキュアブート処理を行います。	—	—	○
R_FWUP_ExecuteFirmware	インストール、もしくはアップデートしたファームウェアへ処理を移します。	—	—	○
R_FWUP_Abort	OTA 更新処理を中止します。	○	—	—
R_FWUP_CreateFileForRx	OTA の初期設定を行います。	○	—	—
R_FWUP_CloseFile	指定したファイルをクローズします。	○	—	—
R_FWUP_WriteBlock	指定したファイルに、指定したオフセットでデータブロックを書き込みます。	○	—	—
R_FWUP_ActiveNewImage	新しいファームウェアイメージをアクティブ化または起動します。	○	—	—
R_FWUP_SetPlatformImageState	ライフサイクルの状態を、引数で指定された状態に設定します。	○	—	—
R_FWUP_GetPlatformImageState	現在のライフサイクルの状態を返します。	○	—	—
R_FWUP_CheckFileSigunature	指定したファイルの署名を確認します。	○	—	—
R_FWUP_ReadAndAssumeCertificate	指定された署名者証明書をファイルシステムから読み込み返します。	○	—	—
R_FWUP_GetVersion	本モジュールのバージョン番号を返します。	○	○	○

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- フラッシュメモリ
- シリアルコミュニケーションインタフェース：オプション
- イーサネット：オプション
- システムタイマモジュール

2.2 ソフトウェアの要求

本 FIT モジュールは以下のパッケージに依存しています。

- ボードサポートパッケージ（r_bsp）
- バイト型キューバッファモジュール（r_byteq）
- コンペアマッチタイマ（r_cmt_rx）
- フラッシュモジュール（r_flash_rx）
- シリアルコミュニケーション
  インタフェース（SCI：調歩同期式/クロック同期式）（r_sci_rx）：オプション
- イーサネットモジュール（r_ether_rx）：オプション
- システムタイマモジュール（r_sys_time_rx）

2.3 サポートされているツールチェーン

本 FIT モジュールは「5.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.4 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_fwup_if.h に記載しています。

2.5 整数型

このドライバは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

2.6　コンパイル時の設定

本モジュールのコンフィグレーションオプションの設定は、r_fwup_config.h で行います。

オプション名および設定値に関する説明を表 2-1 に示します。

表 2-1 コンフィグレーション設定

Configuration options in r_fwup _config.h	
FWUP_CFG_IMPLEMENTATION_ENVIRONMENT ※デフォルト値は “0”	本 FIT モジュールを実装するユーザプログラムの環境をコンフィグ設定します。 実装対象により、使用可能な API 関数が変わります。 以下のいずれかの値を設定する 0：ブートローダプログラムに実装する（デフォルト） 1：ユーザプログラムのファームウェアアッププログラム（OS 無しのシステム）に実装する 2：FreeRTOS（OTA）プログラムに実装する 3：FreeRTOS 以外の OS を使ったファームウェアアップデートプログラムに実装する 実装環境を追加する場合は、設定値を追加する。
FWUP_CFG_COMMUNICATION_FUNCTION ※デフォルト値は “0”	ユーザプログラムのファームウェアアップデート時に、新しいバージョンのファームウェアを取得する通信経路をコンフィグ設定します。 以下のいずれかの値を設定する 0：SCI 通信を接続（デフォルト） 1：Ethernet 通信を接続 2：USB を接続[注 1] 3：SDHI を接続[注 1] 4：QSPI を接続[注 1] 通信経路を追加する場合は、設定値を追加する。
FWUP_CFG_USE_SERIAL_FLASH_FOR_BUFFER ※デフォルト値は “0”	新しいバージョンのファームウェアの取得に、外付けのシリアルフラッシュを使用するか設定します。 0：外付けシリアルフラッシュを使用しない（デフォルト） 1：外付けシリアルフラッシュを使用する[注 1]
FWUP_CFG_SIGNATURE_VERIFICATION ※デフォルト値は “0”	署名検証のアルゴリズムを指定します。 0：署名検証に ECDSA spcp256r1、ハッシュアルゴリズムとして SHA256 を使用（デフォルト） 検証アルゴリズムを追加する場合は、設定値を追加する。
FWUP_CFG_BOOT_PROTECT_ENABLE ※デフォルト値は “0”	ブート保護設定の有効/無効を切り替えます。 0：ブート保護設定無効（デフォルト） 1：ブート保護設定有効 * *）ブートローダを格納する領域を書き換えられなくするための機能です。 一度ブート保護設定を行うと、環境によっては二度と “ブート保護設定無効” には戻せなくなり、アクセス可能領域やスタートアップ領域保護機能の再設定ができなくなります。 ブート保護設定の取り扱いには十分にご注意ください。
FWUP_CFG_PRINTF_DISABLE ※デフォルト値は “0”	ROM サイズ縮小のため、ターミナルソフトへの printf 文での文字列表示を抑制します。 0：ターミナルソフトへ文字列を表示する（デフォルト） 1：ターミナルソフトへ文字列を表示しない

FWUP_CFG_SERIAL_TERM_SCI ※デフォルト値は　“8”	ファームウェアのダウンロードに使用する SCI チャンネルを設定します。
FWUP_CFG_SERIAL_TERM_SCI_BITRATE ※デフォルト値は　“115200”	ファームウェアのダウンロードに使用する UART のボーレート値を設定します。
FWUP_CFG_SERIAL_TERM_SCI_INTERRUPT_PRIORITY ※デフォルト値は　“15”	ファームウェアのダウンロードに使用する SCI の割り込み優先レベルを設定します。
FWUP_CFG_SCI_RECEIVE_WAIT ※デフォルト値は “300”	送信停止(RTS を HIGH に設定)後の UART 受信待機時間を指定します。マイクロ秒単位で設定します。
FWUP_CFG_PORT_SYMBOL ※デフォルト値は RSK-RX231 用の”PORTC”	UART の受信要求端子である RTS に使用される I/O ポートのポートシンボルを設定します。
FWUP_CFG_BIT_SYMBOL ※デフォルト値は RSK-RX231 用の”B4”	UART の受信要求端子である RTS に使用される I/O ポートのビットシンボルを設定します。

注１）設定しても動作しない未対応の項目

コンフィグレーションオプション設定の“FWUP_CFG_IMPLEMENTATION_ENVIRONMENT”と“FWUP_CFG_COMMUNICATION_FUNCTION”には設定可能/不可能な組み合わせがあり、設定可能な組み合わせは以下のとおり。

（表中：”数字”＝FWUP_ENV_COMMUNICATION_FUNCTION へ設定値、“—”＝無効な組み合わせ）

表 2-2 コンパイル時設定の設定可能組み合わせ

		FWUP_CFG_COMMUNICATION_FUNCTION				
		0 : SCI	1 : Ethernet	2 : USB	3 : SDHI	4 : QSPI
FWUP_CFG_IMPLEMENTATION_ENVIRONMENT	0 : ブートローダプログラム	0	—	—	—	—
	1 : ユーザプログラムのファームウェアアップデートプログラム（OS 無のシステム）	1	—	2	—	3
	2 : FreeRTOS（OTA）プログラム	4	5	6	7	—
	3 : FreeRTOS 以外の OS を使ったファームウェアアップデートプログラム	8	—	9	10	11

実装環境設定と通信経路設定の組み合わせで有効となった組み合わせ条件を、r_fwup_private.h 内にマクロで保持する。

表 2-3 有効組み合わせのマクロ値

マクロ名	値	説明
FWUP_COMM_SCI_BOOTLOADER	0	SCI に PC(COM ポート)を接続し、ブートローダ処理を行う
FWUP_COMM_SCI_PRIMITIVE	1	SCI に PC(COM ポート)を接続し、ターミナルソフト経由で新しいバージョンのファームウェアを取得する
FWUP_COMM_USB_PRIMITIVE	2	USB に PC(COM ポート)を接続し、ターミナルソフト経由で新しいバージョンのファームウェアを取得する
FWUP_COMM_QSPI_PRIMITIVE	3	QSPI に外部ストレージ(SD カード)を接続し、新しいバージョンのファームウェアを取得する
FWUP_COMM_SCI_AFRTOS	4	SCI に無線モジュール(SX-ULPGN, BG96 等)を接続し、FreeRTOS(OTA)で新しいバージョンのファームウェアを取得する
FWUP_COMM_ETHER_AFRTOS	5	Ethernet で接続し、FreeRTOS(OTA)で新しいバージョンのファームウェアを取得する
FWUP_COMM_USB_AFRTOS	6	USB に LTE モデムを接続し、FreeRTOS(OTA)で新しいバージョンのファームウェアを取得する
FWUP_COMM_SDHI_AFRTOS	7	SDHI に無線モジュール(Type 1DX 等)を接続し、FreeRTOS(OTA)で新しいバージョンのファームウェアを取得する
FWUP_COMM_SCI_FS	8	SCI に外部ストレージ(SD カード)を接続し、ファイルシステムで新しいバージョンのファームウェアを取得する
FWUP_COMM_USB_FS	9	USB に外部ストレージ(USB メモリ)を接続し、ファイルシステムで新しいバージョンのファームウェアを取得する
FWUP_COMM_SDHI_FS	10	SDHI に外部ストレージ(SD カード)を接続し、ファイルシステムで新しいバージョンのファームウェアを取得する
FWUP_COMM_QSPI_FS	11	QSPI に外部ストレージ(シリアルフラッシュ)を接続し、ファイルシステムで新しいバージョンのファームウェアを取得する

実装環境設定と通信経路設定の組み合わせ設定が追加となった際には、合わせて本マクロに設定を追加します。

ex.)

```
#define FWUP_COMM_SCI_BOOTLOADER    0    // Used for Bootloader with SCI connection from COM port.
#define FWUP_COMM_SCI_PRIMITIVE     1    // SCI connection from COM port using primitive R/W.
#define FWUP_COMM_USB_PRIMITIVE     2    // USB connection from COM port using primitive R/W.
#define FWUP_COMM_QSP_PRIMITIVE     3    // Connect external storage (SD card) to QSPI using primitive R/W.
#define FWUP_COMM_SCI_AFRTOS        4    // Connect wireless module to SCI with Amazon FreeRTOS.
#define FWUP_COMM_ETHER_AFRTOS      5    // Connect Eathernet with Amazon FreeRTOS.
#define FWUP_COMM_USB_AFRTOS        6    // Connect LTE modem to USB with Amazon FreeRTOS.
#define FWUP_COMM_SDHI_AFRTOS       7    // Connect wireless module to SDHI with Amazon FreeRTOS.
#define FWUP_COMM_SCI_FS            8    // External storage (SD card + file system) connected to SCI.
#define FWUP_COMM_USB_FS            9    // External storage (USB memory + file system) connected to USB.
#define FWUP_COMM_SDHI_FS           10   // External storage (SD card + file system) connected to SDHI.
#define FWUP_COMM_QSPI_FS           11   // External storage (Serial flash + file system) connected to QSPI.
```

2.6.1　RX130 環境でのコンパイル時の注意事項

RSK RX130 で本 FIT を使用する場合は、ボードサポートパッケージ(BSP)のコンフィグレーションオプションで設定されるユーザスタックのサイズ(BSP_CFG_USTACK_BYTES)を、デフォルトの 0x400(1KB)から 0x1000(4KB)に増やしてください。

2.7　コードサイズ

本 FIT モジュールのコードサイズを下表に示します。

フラッシュタイプ[注]毎に代表して 1 デバイスずつ掲載しています。

表 2-4 コードサイズ

ROM、RAM およびスタックのコードサイズ				
デバイス	分類	使用メモリ		備考
		C/C++ Compiler Package for RX Family	GCC for Renesas RX	
RX65N (フラッシュタイプ 4)	ROM	3,294 バイト	6,889 バイト	boot_loader Project
		3,983 バイト	4,329 バイト	fwup_main Project
		3,050 バイト	4,329 バイト	eol_main Project
		5,396 バイト	5,674 バイト	aws_demos Project
	RAM	36,968 バイト	36,957 バイト	boot_loader Project
		3,217 バイト	3,213 バイト	fwup_main Project
		2,193 バイト	3,213 バイト	eol_main Project
		1,256 バイト	1,248 バイト	aws_demos Project
	最大使用スタックサイズ	1,168 バイト	872 バイト	boot_loader Project
		2,192 バイト	2,512 バイト	fwup_main Project
		800 バイト	2,512 バイト	eol_main Project
		1,792 バイト	1,788 バイト	aws_demos Project
RX231 (フラッシュタイプ 1)	ROM	3,665 バイト	7,251 バイト	boot_loader Project
		3,949 バイト	4,137 バイト	fwup_main Project
		2,961 バイト	4,137 バイト	eol_main Project
	RAM	2,961 バイト	6,237 バイト	boot_loader Project
		3,217 バイト	3,213 バイト	fwup_main Project
		2,193 バイト	3,213 バイト	eol_main Project
	最大使用スタックサイズ	1,384 バイト	892 バイト	boot_loader Project
		2,172 バイト	2,496 バイト	fwup_main Project
		772 バイト	2,496 バイト	eol_main Project
RX66T (フラッシュタイプ 3)	ROM	3,726 バイト	7,268 バイト	boot_loader Project
		3,944 バイト	4,134 バイト	fwup_main Project
		2,792 バイト	4,134 バイト	eol_main Project
	RAM	36,969 バイト	36,957 バイト	boot_loader Project
		3,217 バイト	3,213 バイト	fwup_main Project
		2,189 バイト	3,213 バイト	eol_main Project
	最大使用スタックサイズ	1,404 バイト	892 バイト	boot_loader Project
		2,168 バイト	2,496 バイト	fwup_main Project
		736 バイト	2,496 バイト	eol_main Project

注）フラッシュタイプの詳細に関しては、アプリケーションノート「RX ファミリ フラッシュモジュール Firmware Integration Technology(R01AN2184)」を参照してください。

＜条件＞

C/C++ Compiler Package for RX Family

- ・最適化レベル　：レベル 2
- ・リンク時にモジュール間最適化の対象にする　：チェック有り
- ・最適化方法　：コードサイズ重視の最適化を実施する
- ・一度も参照のない変数/関数を削除する　：チェック無し
- ・FWUP_CFG_PRINTF_DISABLE（Config）　：1

GCC for Renesas RX

- ・最適化レベル　：Optimize size (-Os)
- ・デバッグレベル　：None
- ・リンクオプション　：-Wl,--no-gc-sections
- ・FWUP_CFG_PRINTF_DISABLE（Config）　：1

【参考】ブートローダの使用 ROM、RAM サイズ

ブートローダプロジェクトが使用する ROM、RAM のサイズを、参考として製品ごとに以下に示します。

表 2-5 ブートローダの使用 ROM、RAM サイズ

ブートローダの使用 ROM、RAM サイズ			
デバイス	分類	使用メモリ	
		C/C++ Compiler Package for RX Family	GCC for Renesas RX
RX130	ROM	32,310 バイト	29,540 バイト
	RAM	11,273 バイト	10,972 バイト
RX231	ROM	31,664 バイト	28,384 バイト
	RAM	12,358 バイト	12,076 バイト
RX671	ROM	32.934 バイト	33,084 バイト
	RAM	41,271 バイト	41,044 バイト
RX65N	ROM	33,806 バイト	32,157 バイト
	RAM	41,079 バイト	40,836 バイト
RX66T	ROM	33,009 バイト	30,476 バイト
	RAM	43,524 バイト	43,220 バイト
RX72N	ROM	35,138 バイト	33,337 バイト
	RAM	41,195 バイト	41,144 バイト

＜条件＞

C/C++ Compiler Package for RX Family

- ・最適化レベル　：レベル 2
- ・リンク時にモジュール間最適化の対象にする　：チェック有り
- ・最適化方法　：コードサイズ重視の最適化を実施する
- ・一度も参照のない変数/関数を削除する　：チェック無し
- ・入出力関数　：簡易版
- ・FWUP_CFG_PRINTF_DISABLE（Config）　：1

GCC for Renesas RX

- ・最適化レベル　：Optimize size (-Os)
- ・デバッグレベル　：None
- ・リンクオプション　：-Wl,--no-gc-sections
- ・FWUP_CFG_PRINTF_DISABLE（Config）　：1

2.8　引数

API 関数の引数にある構造体は Amazon FreeRTOS(OTA) 202002.00 の環境でのファイルコンテキスト設定を、その他の環境でも流用して使用しています。

流用し使用している構造体は以下の通りです。

【注意】

Amazon FreeRTOS(OTA)の当該設定がバージョンアップなどで変更される可能性があります。バージョンアップされた場合に設定値が変更になっていないか確認が必要です。

OTA 環境での宣言箇所：aws_demos¥libraries¥freertos_plus¥aws¥ota¥include¥aws_iot_ota_agent.h

表 2-6 OTA のファイルコンテキスト

```
typedef struct
{
    uint16_t usSize;                        /* Size, in bytes, of the signature. */
    uint8_t ucData[ kOTA_MaxSignatureSize ]; /* The binary signature data. */
} Sig256_t;

typedef struct OTA_FileContext
{
    uint8_t * pucFilePath; /*!< Local file pathname. */
    union
    {
        int32_t lFileHandle;    /*!< Device internal file pointer or handle.
                                 * File type is handle after file is open for write. */
        #if WIN32
            FILE * pxFile;      /*!< File type is stdio FILE structure after file is open for write. */
        #endif
        uint8_t * pucFile;      /*!< File type is RAM/Flash image pointer after file is open for write. */
    };
    uint32_t ulFileSize;        /*!< The size of the file in bytes. */
    uint32_t ulBlocksRemaining; /*!< How many blocks remain to be received (a code optimization). */
    uint32_t ulFileAttributes;  /*!< Flags specific to the file being received (e.g. secure, bundle, archive). */
    uint32_t ulServerFileID;    /*!< The file is referenced by this numeric ID in the OTA job. */
    uint8_t * pucJobName;       /*!< The job name associated with this file from the job service. */
    uint8_t * pucStreamName;    /*!< The stream associated with this file from the OTA service. */
    Sig256_t * pxSignature;     /*!< Pointer to the file's signature structure. */
    uint8_t * pucRxBlockBitmap; /*!< Bitmap of blocks received (for de-duping and missing block request). */
    uint8_t * pucCertFilepath;  /*!< Pathname of the certificate file used to validate the receive file. */
    uint8_t * pucUpdateUrlPath; /*!< Url for the file. */
    uint8_t * pucAuthScheme;    /*!< Authorization scheme. */
    uint32_t ulUpdaterVersion;  /*!< Used by OTA self-test detection, the version of FW that did the update. */
    bool_t xIsInSelfTest;       /*!< True if the job is in self test mode. */
    uint8_t * pucProtocols;     /*!< Authorization scheme. */
} OTA_FileContext_t;
```

2.9　戻り値

API 関数の戻り値を示します。この列挙型は、API 関数のプロトタイプ宣言とともに r_fwup_if.h に記載されています。

表 2-7 API の戻り値設定

```
typedef enum e_fwup_err
{
    FWUP_SUCCESS = 0,               // Normally terminated.
    FWUP_FAIL,                      // Illegal terminated.
    FWUP_IN_PROGRESS,               // Firmware update is in progress.
    FWUP_END_OF_LIFE,               // End Of Life process finished.
    FWUP_ERR_ALREADY_OPEN,          // Firmware Update module is in use by another process.
    FWUP_ERR_NOT_OPEN,              // R_FWUP_Open function is not executed yet.
    FWUP_ERR_IMAGE_STATE,           // Platform image status not suitable for firmware update.
    FWUP_ERR_LESS_MEMORY,           // Out of memory.
    FWUP_ERR_FLASH,                 // Detect error of r_flash module.
    FWUP_ERR_COMM,                  // Detect error of communication module.
    FWUP_ERR_STATE_MONITORING,      // Detect error of state monitoring module.
} fwup_err_t;
```

2.10 FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、スマート・コンフィグレータを使用した(1)の追加方法を推奨しています。ただし、スマート・コンフィグレータは、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)の方法を使用してください。

(1)　e2 studio 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合

e2 studio のスマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: e2 studio 編 (R20AN0451)」を参照してください。

(2)　e2 studio 上で FIT コンフィグレータを使用して FIT モジュールを追加する場合

e2 studio の FIT コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

2.11 システムタイマによる状態遷移監視の注意事項

本モジュールでは、システムタイマによる状態遷移監視を行っており、状態が設定時間以上遷移しない場合はエラー終了する仕様となっています。default 値は 1 分です。状態が設定時間以上固定されない様に御注意ください。

3. API 関数

3.1 R_FWUP_Open 関数

表 3-1 R_FWUP_Open 関数仕様

Format	fwup_err_t R_FWUP_Open (void)	
Description	ファームウェアアップデートモジュールやブートローダモジュールのオープン処理をおこないます。 ファームウェアアップデートモジュールやブートローダモジュールで使用する資源のオープン処理、OS の初期設定（OS 使用時）、各変数の初期化処理を行います。	
Parameters	なし	
Return Values	FWUP_SUCCESS	：正常終了
	FWUP_ERR_ALREADY_OPEN	：既にオープン済み
	FWUP_ERR_LESS_MEMORY	：メモリ不足
	FWUP_ERR_IMAGE_STATE	：フラッシュのステータスがアップデートできる状態でない
	FWUP_ERR_FLASH	：FLASH モジュールのエラー
	FWUP_ERR_COMM	：通信モジュールのエラー
	FWUP_ERR_STATE_MONITORING	：状態遷移監視モジュールのエラー
Special Notes	ー	

3.2 R_FWUP_Close 関数

表 3-2 R_FWUP_Close 関数仕様

Format	fwup_err_t R_FWUP_Close (void)	
Description	ファームウェアアップデートモジュールやブートローダモジュールのクローズ処理をおこないます。 ファームウェアアップデートモジュールやブートローダモジュールで使用した資源のクローズ処理、OS の終了設定（OS 使用時）を行います。	
Parameters	なし	
Return Values	FWUP_SUCCESS	：正常終了
	FWUP_ERR_NOT_OPEN	：オープンしていない
	FWUP_ERR_FLASH	：FLASH モジュールのエラー
	FWUP_ERR_COMM	：通信モジュールのエラー
	FWUP_ERR_STATE_MONITORING	：状態遷移監視モジュールのエラー
Special Notes	ー	

3.3　R_FWUP_Operation 関数

表 3-3 R_FWUP_Operation 関数仕様

Format	fwup_err_t R_FWUP_Operation (void)	
Description	ユーザプログラムからのファームウェアアップデート処理を行います。 コンフィグ設定で指定された通信経路からアップデート用ファームウェアデータを取り出し、フラッシュへプログラム、署名検証の実施を行います。 ● 更新対象のフラッシュの状態が“VALID”または“INITIAL_FIRM_ INSTALLING”以外の場合は ファームウェアを更新できる状態ではないため、戻り値として”FWUP_ERR_IMAGE_STATE”を返します。 ● 戻り値が“FWUP_IN_PROGRESS”の場合はファームウェアアップデート継続中のため、再度本関数をコールしてください。 ● 戻り値が“FWUP_SUCCESS”の場合はファームウェアアップデート完了です。R_FWUP_SoftwareReset 関数をコールしてください。ソフトウェアリセットを行うことで更新したファームウェアに処理を移します。 ● 戻り値が“FWUP_FAIL”の場合はファームウェアアップデートが失敗を意味します。エラーを解除し再度本関数をコールしてください。	
Parameters	なし	
Return Values	FWUP_SUCCESS	：ファームウェアアップデート正常終了
	FWUP_FAIL	：ファームウェアアップデートエラー発生
	FWUP_IN_PROGRESS	：ファームウェアアップデート継続中
	FWUP_ERR_NOT_OPEN	：オープンしていない
	FWUP_ERR_IMAGE_STATE	：フラッシュのステータスがアップデートできる状態でない
	FWUP_ERR_STATE_MONITORING	：ファームウェアアップデートの状態が一定時間以上変わっていない
Special Notes	ー	

3.4　R_FWUP_SoftwareReset 関数

表 3-4 R_FWUP_SoftwareReset 関数仕様

Format	void R_FWUP_SoftwareReset (void)
Description	ソフトウェアリセットを行います。
Parameters	なし
Return Values	なし
Special Notes	ー

3.5　R_FWUP_SetEndOfLife 関数

表 3-5 R_FWUP_SetEndOfLife 関数仕様

書式	fwup_err_t R_FWUP_SetEndOfLife (void)
Description	ユーザプログラムの保守期間終了時（END OF LIFE）の処理を行います。 [注意] 本関数コールし正常終了（FWUP_SUCCESS）した状態は、END OF LIFE(EOL)の処理は未完了の状態です。 END OF LIFE(EOL)の処理を完了させるためには本関数実施後に R_FWUP_SoftwareReset 関数をコールしバンクスワップを伴うソフトウェアリセットを発生させ、ブートローダで残りの END OF LIFE 処理を実施する必要があります。
Parameters	なし
Return Values	FWUP_SUCCESS　：正常終了
	FWUP_ERR_NOT_OPEN　：オープンしていない
	FWUP_ERR_IMAGE_STATE　：フラッシュのステータスがアップデートできる状態でない
	FWUP_ERR_FLASH　：FLASH モジュールのエラー
	FWUP_ERR_COMM　：通信モジュールのエラー
Special Notes	ー

3.6　R_FWUP_SecureBoot 関数

表 3-6 R_FWUP_SecureBoot 関数仕様

Format	int32_t R_FWUP_SecureBoot (void)
Description	ブートローダでのセキュアブート処理を行う。 ● インストール済みのファームウェアを起動前に、改ざんチェックのため署名検証します。 ● ファームウェアがインストールされていない場合、コンフィグ設定で指定された通信経路からインストール用ファームウェアデータを取り出し、フラッシュへプログラム、署名検証の実施を行います。 ● ユーザプログラムでアップデート用ファームウェアが設定された場合、起動用ファームウェアに置き換えます。 ● ユーザプログラムで保守期間終了時処理（EOL 処理）が設定された場合、本関数でファームウェアを削除します。 ● 戻り値が “FWUP_IN_PROGRESS” の場合はセキュアブート継続中のため、再度本関数をコールしてください。 ● 戻り値が “FWUP_SUCCESS” の場合はセキュアブート完了です。R_FWUP_ExecuteFirmware 関数をコールしインストールし、更新したファームウェアへ処理を移してください。 ● 戻り値が “FWUP_FAIL” の場合はセキュアブートの失敗を意味します。必要であればエラーを解除し再度本関数をコールしてください。
Parameters	なし
Return Values	FWUP_SUCCESS　：セキュアブート正常終了
	FWUP_FAIL　：セキュアブートエラー発生
	FWUP_IN_PROGRESS　：セキュアブート継続中
Special Notes	ー

3.7　R_FWUP_ExecuteFirmware 関数

表 3-7 R_FWUP_ExecuteFirmware 関数仕様

Format	void R_FWUP_ExecuteFirmware (void)
Description	インストール、もしくはアップデートしたファームウェアへ処理を移します。 [注意] 処理を移行するファームウェアの起動アドレスは、MCU のファミリやシリーズにより異なる場合があります。 実装する環境に応じてファームウェアの起動アドレスを取得する処理を実装する必要があります。 [例：RX65N の場合] マクロ USER_RESET_VECTOR_ADDRESS に設定されているアドレスに処理を移す
Parameters	なし
Return Values	なし
Special Notes	ー

3.8　R_FWUP_Abort 関数

表 3-8 R_FWUP_Abort 関数仕様

Format	OTA_Err_t R_FWUP_Abort (OTA_FileContext_t * const C)	
Description	OTA 更新処理を中止します。	
Parameters	* C　　ファイルコンテキスト	
Return Values	kOTA_Err_None	：正常終了
	kOTA_Err_FileClose	：ファイルコンテキストのクローズエラー
Special Notes	ー	

3.9　R_FWUP_CreateFileForRx 関数

表 3-9 R_FWUP_CreateFileForRx 関数仕様

Format	OTA_Err_t R_FWUP_CreateFileForRx (OTA_FileContext_t * const C)	
Description	OTA の初期設定を行います。 受信したデータを保存するファイルを作成します。	
Parameters	* C	：ファイルコンテキスト
Return Values	kOTA_Err_None	：正常終了
	kOTA_Err_RxFileCreateFailed	：ファイル作成エラー
Special Notes	ー	

3.10 R_FWUP_CloseFile 関数

表 3-10 R_FWUP_CloseFile 関数仕様

Format	OTA_Err_t R_FWUP_CloseFile (OTA_FileContext_t * const C)
Description	指定されたファイルをクローズします。 テンポラリエリアのバッファ領域にダウンロードしたファームウェアイメージに対する署名検証を行います。 テンポラリエリアのバッファ領域のヘッダ情報を書き込みます。
Parameters	* C　：ファイルコンテキスト
Return Values	kOTA_Err_None　：正常終了
	kOTA_Err_FileClose　：ファイルのクローズエラー
	kOTA_Err_SignatureCheckFailed　：署名検証エラー
Special Notes	ー

3.11 R_FWUP_WriteBlock 関数

表 3-11 R_FWUP_WriteBlock 関数仕様

Format	int16_t R_FWUP_WriteBlock (OTA_FileContext_t * const C, uint32_t ulOffset, uint8_t * const pacData, uint32_t ulBlockSize)
Description	指定されたファイルに、指定されたオフセットでデータブロックを書き込みます。 成功した場合、書き込まれたバイト数を返します。
Parameters	* C　：ファイルコンテキスト ulOffset　：コードフラッシュの書き込み先オフセット * pacData　：書き込むデータ ulBlockSize　：書き込むデータサイズ
Return Values	R_OTA_ERR_QUEUE_SEND_FAIL (-2)　：コードフラッシュへの書き込みエラー
	上記以外　：コードフラッシュに書き込んだバイト数
Special Notes	ー

3.12 R_FWUP_ActiveNewImage 関数

表 3-12 R_FWUP_ActiveNewImage 関数仕様

Format	OTA_Err_t R_FWUP_ActiveNewImage (void)
Description	新しいファームウェアイメージをアクティブ化または起動します。 R_FWUP_ResetDevice()関数をコールし、ソフトウェアリセットします。
Parameters	なし
Return Values	kOTA_Err_None　：正常終了
Special Notes	ー

3.13 R_FWUP_ResetDevice 関数

表 3-13 R_FWUP_ResetDevice 関数仕様

Format	OTA_Err_t R_FWUP_ResetDevice (void)
Description	本関数コールでソフトウェアリセットが発生し、以降はブートローダの処理を経て新しいファームウェアを起動します。
Parameters	なし
Return Values	kOTA_Err_None　：正常終了
Special Notes	オープンしている周辺回路をすべてクローズしてから本関数を実行すること。

3.14 R_FWUP_SetPlatformImageState 関数

表 3-14 R_FWUP_SetPlatformImageState 関数仕様

Format	OTA_Err_t R_FWUP_SetPlatformImageState (OTA_ImageState_t eState)
Description	ライフサイクルの状態を、Parameters で指定された状態に設定します。 新しいファームウェアへの更新が完了した場合、テンポラリエリアのバッファ領域を消去します。
Parameters	eState　：設定する状態
Return Values	kOTA_Err_None　：正常終了
	kOTA_Err_CommitFailed　：コミットエラー
Special Notes	ー

3.15 R_FWUP_GetPlatformImageState 関数

表 3-15 R_FWUP_GetPlatformImageState 関数仕様

Format	OTA_PAL_ImageState_t R_FWUP_GetPlatformImageState (void)
Description	現在のライフサイクルの状態を返します。
Parameters	なし
Return Values	現在のライフサイクル の状態
Special Notes	ー

3.16 R_FWUP_CheckFileSignature 関数

表 3-16 R_FWUP_CheckFileSignature 関数仕様

Format	OTA_Err_t R_FWUP_CheckFileSignature (OTA_FileContext_t * const C)
Description	指定したファイルの署名を確認します。
Parameters	* C　：ファイルコンテキスト
Return Values	kOTA_Err_None　：正常終了
	kOTA_Err_SignatureCheckFailed　：署名検証エラー
Special Notes	ー

3.17 R_FWUP_ReadAndAssumeCertificate 関数

表 3-17 R_FWUP_ReadAndAssumeCertificate 関数仕様

Format	uint8_t * R_FWUP_ReadAndAssumeCertificate (const uint8_t * const pucCertName uint32_t * const ulSignerCertSize)
Description	指定された署名者証明書をファイルシステムから読み込み返します。
Parameters	* pucCertName　　：証明書ファイル名
	* ulSignerCertSize　　：証明書のサイズ
Return Values	証明書データへのポイン タ
Special Notes	ー

3.18 R_FWUP_GetVersion 関数

表 3-18 R_FWUP_GetVersion 関数仕様

Format	uint32_t R_FWUP_GetVersion (void)
Description	本 FIT モジュールのバージョン番号を返します。
Parameters	なし
Return Values	バージョン番号
Special Notes	ー

4. デモプロジェクト

デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュールを使用する main()関数が含まれます。本 FIT モジュールには以下のデモプロジェクトが含まれます。

4.1 RX65N のシリアル通信インターフェイス(SCI)を用いたファームウェアアップデート

本ファームウェアアップデートのデモは、RSK RX65N スターターキットの RX65N シリアル通信インタフェース（SCI）を用いたデモです。UART として構成された SCI チャネルを介してターミナルと通信を行います。

本ファームウェアアップデートのデモでは、PMOD1 にインタフェースされているシリアルポート SCI6 を使用しています。PMOD1 コネクタにはシリアル変換ボードを接続します。

ターミナルソフトウェアを実行している PC が入出力用に必要となります。

表 4-1 機器構成

No.	機器	補足
1	開発 PC	開発を行う PC です。
2	評価ボード （Renesas Starter Kit for RX65N）	―
3	ホスト PC（TeraTerm 等のターミナルソフトウェア）	XMODEM/SUM 転送プロトコルに対応したシリアル通信ソフトウェア（開発 PC でも代用可能です）
4	USB シリアル変換ボード	Renesas Starter Kit for RX65N のシリアル入出力信号を USB シリアル変換し、ホスト PC と USB 接続します。
5	USB ケーブル	USB シリアル変換ボードとホスト PC を USB 接続します。

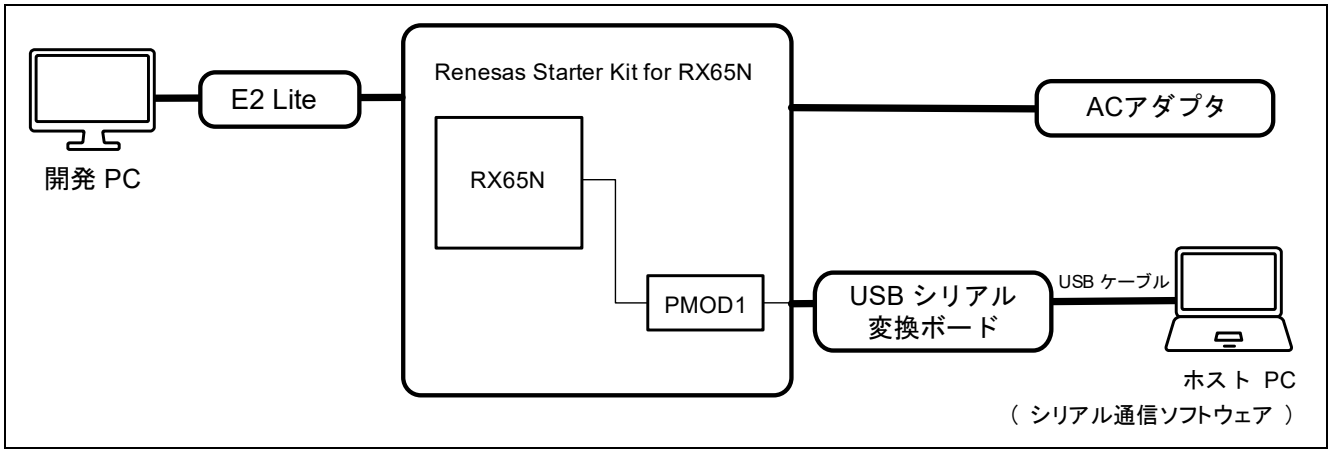


図 4-1 RSK RX65N 機器接続図

表 4-2 通信仕様

項目	内容
通信方式	調歩同期式通信
ビットレート	115200bps
データ長	8 ビット
パリティ	なし
ストップビット	1 ビット
フロー制御	なし

4.1.1　アップデートファームウェアの生成

1. 更新するファームウェアの完全性を確保するために、更新ファームウェアのデジタル署名（ECDSA+SHA256）にて完全性の検証を行います。検証を行うには、下記のコードを fwup_main_RX65N のサンプルアプリケーションに追加します。

   ・ Tinycrypt ライブラリの追加

   ・ Base64 デコード関数の追加

   ・ デジタル署名に使用する Key ファイルの追加

   追加方法は以下の通りです。

   ① Tinycrypt ライブラリの追加方法
   https://github.com/renesas/amazon-freertos/tree/master/libraries/3rdparty/tinycrypt より取得し、lib フォルダを fwup_main_RX65N プロジェクトの src/src/tinycypt/lib に追加します。

   ② Base64 デコード関数の追加方法

   https://github.com/renesas/amazon-freertos/tree/master/projects/renesas/rx65n-rsk/e2studio/boot_loader/src/src より取得し、base64_decode.c と base64_decode.h を fwup_main_RX65N プロジェクトの src/src に追加します。

   ③ Key ファイルの追加方法

   https://github.com/renesas/amazon-freertos/tree/master/projects/renesas/rx65n-rsk/e2studio/boot_loader/src/ より取得し、key フォルダを fwup_main_RX65N プロジェクトの src/key に追加します。追加した後、署名検証用の公開鍵情報を code_signer_public_key.h に入力します。入力方法は以下をリンクの内容を参照してください。
   https://github.com/renesas/amazon-freertos/wiki/OTA の活用#手順まとめ
   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

2. fwup_main_RX65N のサンプルアプリケーションをビルドし、作成された.mot ファイルを.RSU ファイルに変換します。これを initial firmware とします。

.mot ファイルから.RSU ファイルへの変換方法は下記の通りです。

Release mot file converter tool · renesas/mot-file-converter · GitHub から Renesas Secure Flash Programmer.exe をダウンロードし実行します。（一緒に置かれているファイルも必要ですのでダウンロードしてください）

・[Initial Firm]タブを選択して、下図のようにパラメータを設定します。

・Settings の Private Key Path に secp256r1.privatekey のパスを設定します。

・Settings の Select Output format に Bank0 User Program (Binary Format)を設定します。

・Bank 0 User Program の File Path に上記で作成した.mot ファイルのパスを設定します。

・[Generate]をクリックすると.RSU ファイルが生成され init_firmware フォルダに保存されます。

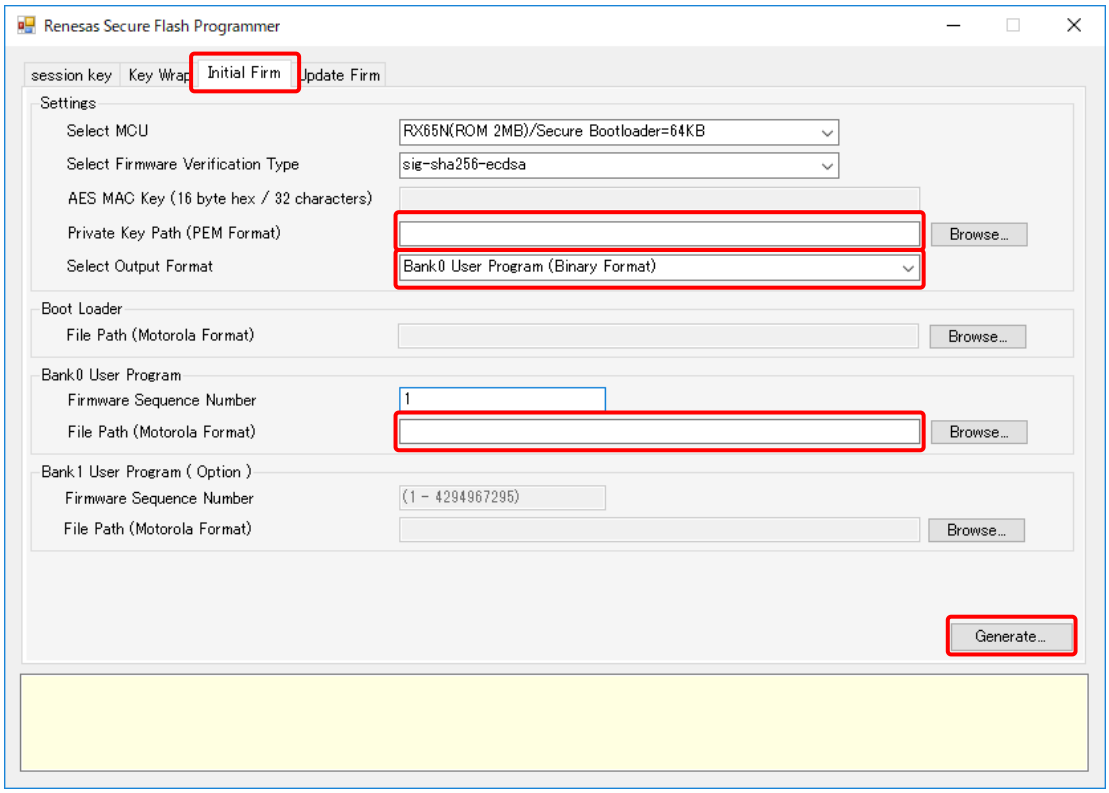


図 4-2 Renesas Secure Flash Programmer Initial Firm 画面

3. src/main.c を開き、以下のコメント行を有効化します。

```
main.c 84 行目～88 行目
//    printf("[FWUP_main DEMO] Firmware update demonstration completed.¥r¥n");
//      while(1)
//      {
//              /* infinity loop */
//      }
```

プロジェクトを再びビルドし、作成された.mot ファイルを.RSU ファイルに変換します。これを next firmware とします。

.mot ファイルから.RSU ファイルへの変換方法は下記の通りです。

・[Update Firm]タブを選択して、下図のようにパラメータを設定します。

・Settings の Private Key Path に secp256r1.privatekey のパスを設定します。

・Bank 0 User Program の File Path に上記で作成した.mot ファイルのパスを設定します。

・[Generate]をクリックすると.RSU ファイルが生成され update_firmware フォルダに保存されます。

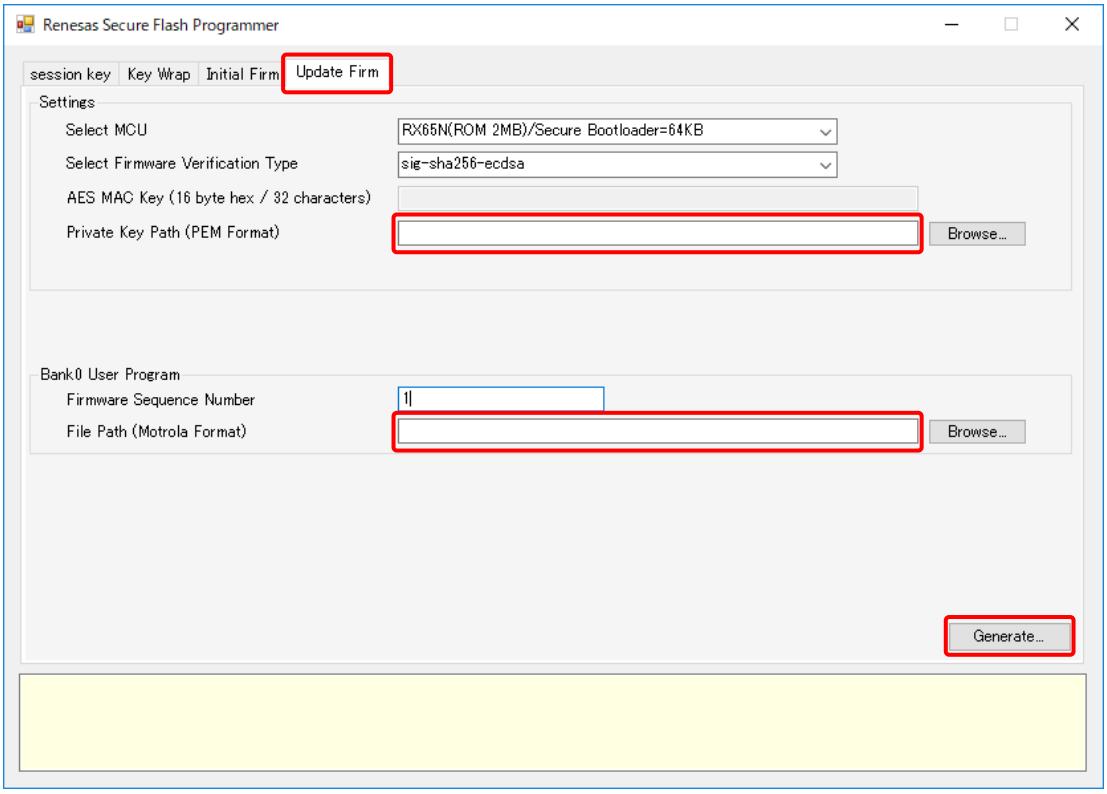


図 4-3 Renesas Secure Flash Programmer Update Firm 画面

4.1.2　ファームウェアのアップデート

1. 「図 4-1 RSK RX65N 機器接続図」の様に PC の USB ポート, USB シリアル変換ボード, RSK ボードの PMOD1 を接続してください。
2. PC 上のターミナルソフトウェア（以下、ターミナル）を開きます。そして、USB シリアル変換ボードに割り当てられたシリアル COM ポートを選択します。
3. このサンプルアプリケーションの設定と一致するように、ターミナルのシリアル設定を行います。115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。
4. ブートローダプログラムをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。
5. ソフトウェアを実行すると以下のメッセージが出力されます。

```
send "userprog.rsu" via UART.
```

ターミナルの機能で「ファイル送信」を選び、生成した initial firmware の.RSU ファイルを送信（送信オプションとしてバイナリの送信を設定してください）します。.RSU ファイルのデータ受信・コードフラッシュへの書き込み中は以下のメッセージが出力されます。

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

6. インストールと署名検証が終了するとファームウェアアップデート用アプリケーションが起動され、ファームウェア アプリケーションの入力を促すメッセージが出力されます。

```
jump to user program
[R_FWUP_GetPlatformImageState] is called.
Function call: R_FWUP_GetPlatformImageState: [2]
[R_FWUP_CreateFileForRx] is called.
[R_FWUP_CreateFileForRx] Receive file created.
[R_FWUP_GetPlatformImageState] is called.
Function call: R_FWUP_GetPlatformImageState: [2]
-------------------------------------------------
Firmware update user program
-------------------------------------------------
Send Update firmware via UART.
```

ターミナルの機能で「ファイル送信」を選び、生成した next firmware の.RSU ファイルを送信（送信オプションとしてバイナリの送信を設定してください）します。.RSU ファイルのデータ受信・コードフラッシュへの書き込み中は以下のメッセージが出力されます。

```
[R_FWUP_WriteBlock] is called.
[R_FWUP_Operation] Flash Write: Address = 0xFFE00000, length = 1024 ... OK
[R_FWUP_WriteBlock] is called.
[R_FWUP_Operation] Flash Write: Address = 0xFFE00400, length = 1024 ... OK
```

7. ファームウェア アプリケーションのインストールと署名検証が終了すると、バンクスワップ等の処理を経て、ファームウェア アプリケーションにジャンプし実行されます。

```
jump to user program
[R_FWUP_GetPlatformImageState] is called.
```

8. ファームウェア アプリケーションで以下のメッセージが出力されるとデモの正常終了です。

```
[FWUP_main DEMO] Firmware update demonstration completed.
```

4.1.3 EOL ファームウェアの生成

1. eol_main_RX65N のサンプルアプリケーションをビルドし、作成された.mot ファイルを.RSU ファイルに変換します。これを eol firmware とします。

   .RSU ファイルへの変換方法は 4.1.1 を参照。

4.1.4 ファームウェアの EOL

1. 「図 4-1 RSK RX65N 機器接続図」の様に PC の USB ポート, USB シリアル変換ボード, RSK ボードの PMOD1 を接続してください。
2. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます。そして、USB シリアル変換ボードに割り当てられたシリアル COM ポートを選択します。
3. このサンプルアプリケーションの設定と一致するように、ターミナルのシリアル設定を行います。115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。
4. ブートローダプログラムをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。
5. ソフトウェアを実行すると以下のメッセージが出力されます。

```
send "userprog.rsu" via UART.
```

   ターミナルの機能で「ファイル送信」を選び、生成した eol firmware の.RSU ファイルを送信（送信オプションとしてバイナリの送信を設定してください）します。.RSU ファイルのデータ受信・コードフラッシュへの書き込み中は以下のメッセージが出力されます。

```
installing firmware...0%(1/960KB).
installing firmware...0%(2/960KB).
installing firmware...0%(3/960KB).
installing firmware...0%(4/960KB).
```

6. インストールと署名検証が終了すると END OF LIFE (EOL)アプリケーションが起動されます。

```
------------------------------------------------
End Of Life (EOL) process of user program
------------------------------------------------
[R_FWUP_SetEndOfLife] erase install area (code flash):OK
[R_FWUP_SetEndOfLife] update bank1 LIFECYCLE_STATE to [LIFECYCLE_STATE_EOL]
[EOL_main] EOL process completely. Bank swap and software reset.
[R_FWUP_ActivateNewImage] Changing the Startup Bank
[R_FWUP_ResetDevice] Resetting the device.
[R_FWUP_ResetDevice] Swap bank...
```

7. END OF LIFE (EOL)アプリケーションの処理が終わるとブートローダに戻り、ブートローダ内の EOL 処理が実行されます。

```
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xe0 [LIFECYCLE_STATE_EOL]
bank 1 status = 0xf8 [LIFECYCLE_STATE_VALID]
```

8. 以下のメッセージが出力されると EOL 処理の正常終了です。

```
End Of Life process finished.
```

5. 付録

5.1 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 5-1 動作確認環境 (Rev.1.02)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio 2021 04
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.02.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202004 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
エンディアン	リトルエンディアン
モジュールのリビジョン	Rev.1.02
使用ボード	Renesas Starter Kit for RX130-512KB（製品型名：RTK5051308SxxxxxBE） Renesas Starter Kit+ for RX231（製品型名：R0K505231SxxxBE） Renesas Starter Kit+ for RX65N（製品型名：RTK50565N2SxxxxxBE） Renesas Starter Kit for RX66T（製品型名：RTK50566T0S00000BE） Renesas Starter Kit+ for RX671（製品型名：RTK55671EHS10000BE） Renesas Starter Kit+ for RX72N（製品型名：RTK5572NNxxxxxxxBE）
USB シリアル変換ボード	Pmod USBUART（DIGILENT 製） https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

ファームウェアアップデートの動作確認のために、デモプロジェクトで使用した FIT モジュールのバージョン一覧を以下に示します。

(1) ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family の環境

表 5-2 FIT モジュールのバージョン一覧(1)

デバイス	プロジェクト	r_bsp	r_byteq	r_flash_rx	r_fwup	r_sys_time_rx	r_sci_rx	r_cmt_rx
RX130	boot_loader fwup_main eol_main	5.61	1.81	4.60	1.01	1.01	3.70	4.70
RX231	boot_loader fwup_main eol_main	5.61	1.81	4.60	1.01	1.01	3.70	4.70
RX65N	boot_loader fwup_main eol_main	5.61	1.81	4.60	1.01	1.01	3.70	4.70
	aws_demos	5.52	1.80	4.50	1.01	1.01	3.50	4.31
RX66T	boot_loader fwup_main eol_main	5.61	1.81	4.60	1.01	1.01	3.70	4.70
RX671	boot_loader fwup_main eol_main	6.11	1.81	4.70	1.02	1.01	3.80	4.80

RX72N	boot_loader fwup_main eol_main	5.61	1.81	4.60	1.01	1.01	3.70	4.70

(2) GCC for Renesas RX の環境

表 5-3 FIT モジュールのバージョン一覧(2)

デバイス	プロジェクト	r_bsp	r_byteq	r_flash_rx	r_fwup	r_sys_time_rx	r_sci_rx	r_cmt_rx
RX130	boot_loader_gcc fwup_main_gcc eol_main_gcc	5.61	1.81	4.60	1.02	1.01	3.70	4.70
RX231	boot_loader_gcc fwup_main_gcc eol_main_gcc	5.61	1.81	4.60	1.02	1.01	3.70	4.70
RX65N	boot_loader_gcc fwup_main_gcc eol_main_gcc	5.52	1.80	4.50	1.02	1.01	3.70	4.70
	aws_demos	5.52	1.80	4.50	1.02	1.01	3.70	4.70
RX66T	boot_loader_gcc fwup_main_gcc eol_main_gcc	5.61	1.81	4.60	1.02	1.01	3.70	4.70
RX671	boot_loader_gcc fwup_main_gcc eol_main_gcc	6.11	1.81	4.70	1.02	1.01	3.80	4.80
RX72N	boot_loader_gcc fwup_main_gcc eol_main_gcc	5.61	1.81	4.60	1.02	1.01	3.70	4.70

5.2　コンパイラ依存の設定

本モジュールは Rev.1.02 から複数のコンパイラに対応しています。本モジュールを使用するにあたり、コンパイラ毎に異なる設定を以下に示します。

5.2.1　Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合

コンパイラとして Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合について示します。

リンカのセクションの設定は e^2 studio で行う必要があります。

5.2.1.1　コンパイルオプション

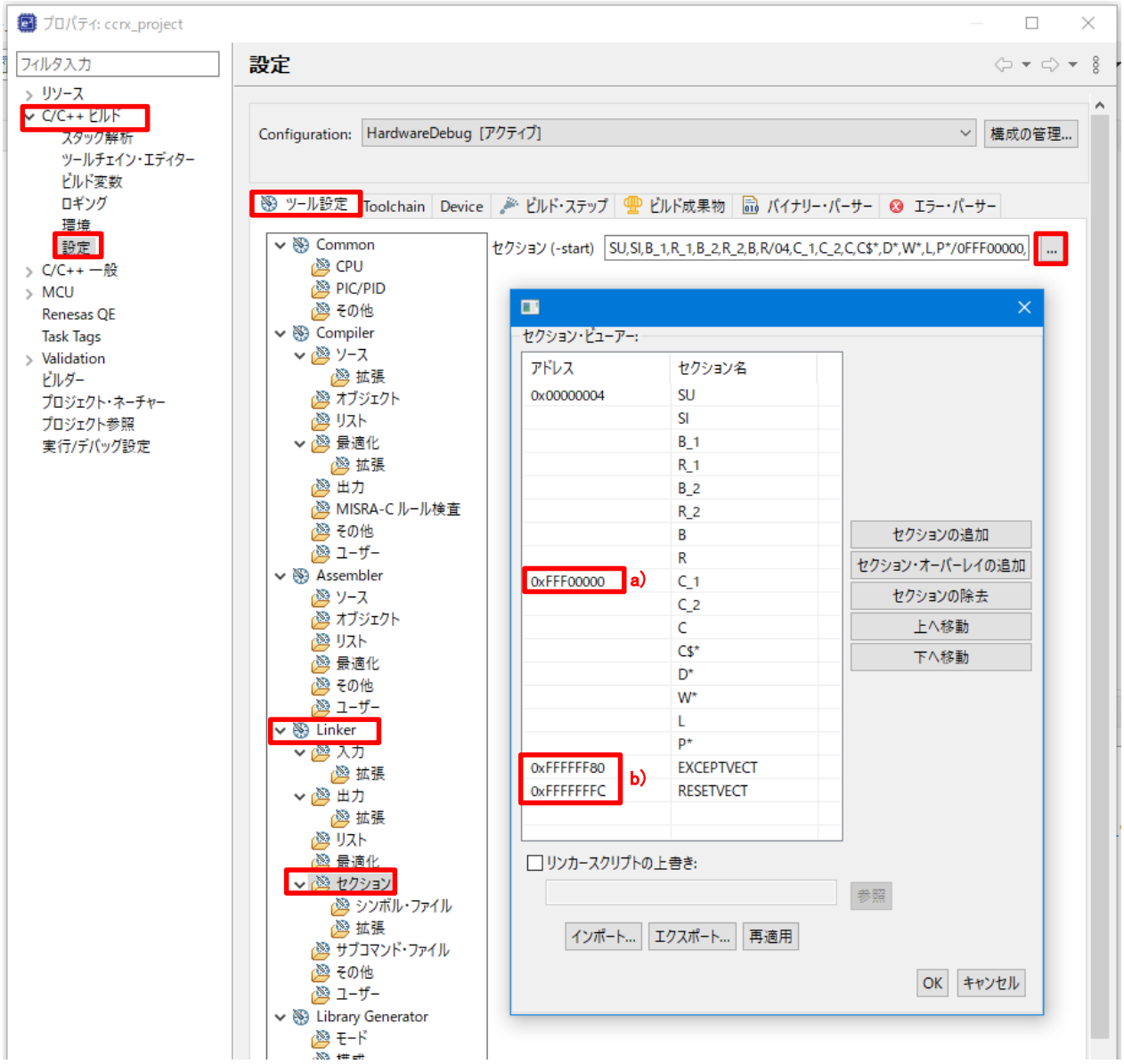
統合開発環境のデフォルト設定に以下のオプションを追加してください。

-lang = c99

5.2.1.2　フラッシュメモリ上の配置アドレスの変更

フラッシュメモリ上の実行領域にブートローダとユーザプログラムを配置するために、リンカのセクション設定を変更します。

1) 「プロジェクト・エクスプローラー」においてデバッグ対象のプロジェクトをクリックします。
2) 「ファイル」→「プロパティ」の順にクリックし、「プロパティ」ウィンドウを開きます。
3) 「プロパティ」ウィンドウで、「C/C++ビルド」→「設定」の順にクリックします。
4) 「ツール設定」タブを押下し、「Linker」→「セクション」の順にクリックして、「…」ボタンを押下し、「セクション・ビューア―」ウィンドウを開きます。



5)　「セクション・ビューアー」ウィンドウの a)～b)を、ユーザの環境に合わせて変更してください。
例) RX65N、デュアルモード、ブートローダのサイズ=64KB の場合の各設定値は以下のとおり。

記号	説明	ブートローダの設定	ユーザプログラムの設定
a)	フラッシュメモリ上での開始アドレス	0xFFFF0000	0xFFF00300
b)	例外ベクタとリセットベクタの配置アドレス	0xFFFFFF80 0xFFFFFFFC	0xFFFEFF80 0xFFFEFFFC

5.2.1.3　フラッシュメモリ書き換えの設定

ユーザプログラムおよびブートプログラムに、フラッシュメモリ書き換えのための設定をする必要があります。設定の詳細は、以下アプリケーションノートを参照してください。

「フラッシュモジュール Firmware Integration Technology (R01AN2184)」の
「5.3.1 Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合」

5.2.2　GCC for Renesas RX を使用する場合

コンパイラとして GCC for Renesas RX を使用する場合について示します。

リンカの設定は e^2 studio で生成されるリンカ設定ファイルを編集する必要があります。

5.2.2.1　コンパイルオプション

1) コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加してください。

   -std=gnu99

2) リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のオプションを追加してください。

   -Wl,--no-gc-sections

   これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。

3) コンパイルオプション：ブートローダをデバッグする場合、統合開発環境のデフォルト設定から以下のオプションを変更してください。

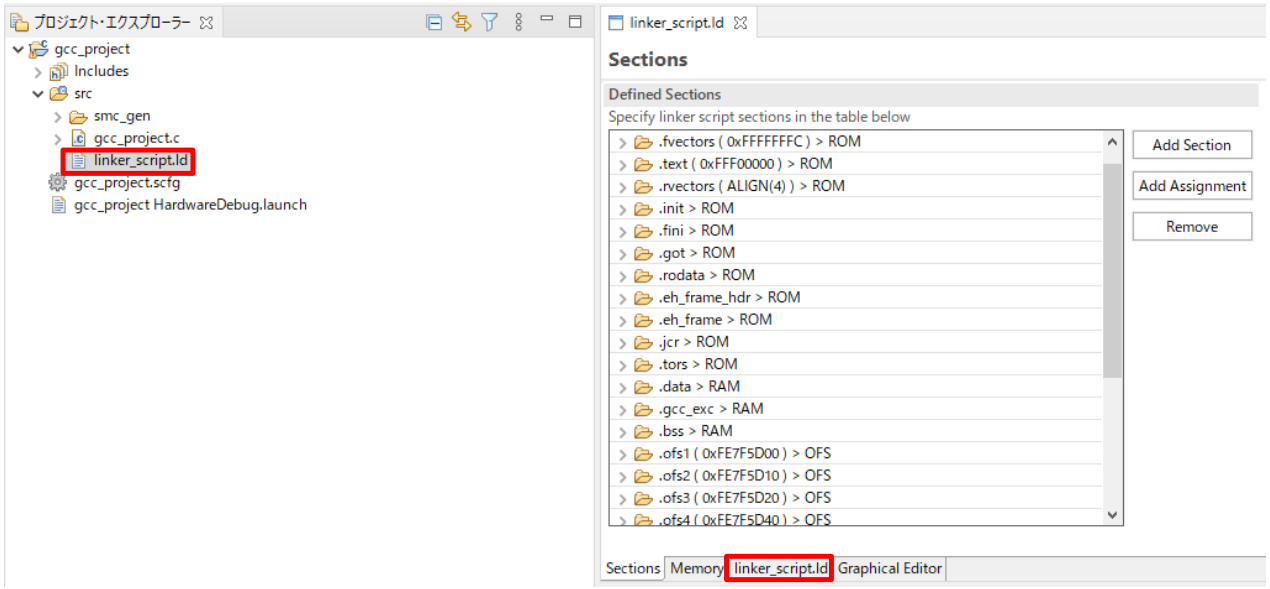
   最適化レベル：Optimize for debug (-Og)

5.2.2.2　フラッシュメモリ上の配置アドレスの変更

内蔵フラッシュ上の実行領域にブートローダとユーザプログラムを配置するために、リンカ設定を変更します。

1) プロジェクト・エクスプローラーからリンカ設定ファイル(linker_script.ld)を右クリックして、「開く」を選択します。

2) 「linker_script.ld」ウィンドウで、「linker_script.ld」タブをクリックします。



3)　以下の(a)～(d)のアドレスを、ユーザの環境に合わせて変更してください。

linker_script.ld

```
1  MEMORY
2  {
3      RAM : ORIGIN = 0x4, LENGTH = 0x3fffc
4      RAM2 : ORIGIN = 0x00800000, LENGTH = 393216
5      ROM : ORIGIN = 0xFFF00000, LENGTH = 1048576   a)
6      OFS : ORIGIN = 0xFE7F5D00, LENGTH = 128
7  }
8  SECTIONS
9  {
10     .exvectors 0xFFFFFF80: AT(0xFFFFFF80)   b)
11     {
12         "_exvectors_start" = .;
13         KEEP(*(.exvectors))
14     "_exvectors_end" = .;
15     } >ROM
16     .fvectors 0xFFFFFFFC: AT(0xFFFFFFFC)   c)
17     {
18         KEEP(*(.fvectors))
19     } > ROM
20     .text 0xFFF00000: AT(0xFFF00000)   d)
21     {
22         *(.text)
23         *(.text.*)
24         *(P)
25         etext = .;
26     } > ROM
27     .rvectors ALIGN(4):
28     {
```

例) RX65N、デュアルモード、ブートローダのサイズ=64KB の場合の各設定値は以下のとおり。

記号	説明	ブートローダの設定	ユーザプログラムの設定
a)	コードフラッシュの開始アドレスと、コードフラッシュのサイズ	ORIGIN = 0xFFFF0000 LENGTH = 65536	ORIGIN = 0xFFF00300 LENGTH = 982272
b)	例外ベクタの配置アドレス	0xFFFFFF80	0xFFFEFF80
c)	リセットベクタの配置アドレス	0xFFFFFFFC	0xFFFEFFFC
d)	コードフラッシュの開始アドレス ＝ a)と同じアドレス	0xFFFF0000	0xFFF00300

5.2.2.3　フラッシュメモリ書き換えの設定

ユーザプログラムおよびブートプログラムに、フラッシュメモリ書き換えのための設定をする必要があります。設定の詳細は、以下アプリケーションノートを参照してください。

「フラッシュモジュール Firmware Integration Technology (R01AN2184)」の
「5.3.2 GCC for Renesas RX を使用する場合」

5.2.2.4　ビルド時のワーニングについて

本 FIT モジュールをビルドする際に、関数のスタック使用量が、-Wstack-usage オプションで指定したバイトサイズを超えた（“warning: stack usage is XXX bytes [-Wstack-usage=]”）ことを示すワーニングが出ます（デフォルト 100 バイト）。問題が有る場合は、ビルドオプションの設定を変更してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021.4.16	-	初版発行
1.01	2021.6.21	表紙	動作確認デバイスに RX72N グループ、RX66T グループ、RX130 グループを追加。
		5	「1.概要」の内容を見直し
		13	「2.6 コンパイル時の設定」に設定するオプションを追記。 - FWUP_CFG_SERIAL_TERM_SCI - FWUP_CFG_SERIAL_TERM_SCI_BITRATE - FWUP_CFG_SERIAL_TERM_SCI_INTERRUPT_PRIORITY 説明の見直し
		16	「2.6.1 RX130 環境でのコンパイル時の注意事項」を追記。
		18	「2.8 引数」に OTA のファイルコンテキストの説明を追記。
		19	該当箇所がないため、「2.12 for 文、while 文、do while 文について」の記載を削除
		25	「3.13 R_FWUP_ResetDevice 関数」に Special Notes を追記。
		33	動作確認環境（Rev.1.01）を追記。
1.02	2021.10.29	表紙	動作確認デバイスに RX671 グループを追加。
		5	「1.2 ファームウェアアップデートモジュールの構成」で、OS なしのシステムと FreeRTOS（OTA）のシステムにおけるファームウェアアップデートモジュールとブートローダモジュールの説明を削除
		6	図 1-1 のシリアル通信モジュールとバイト型キューバッファモジュールの接続を修正
		8	表 1-2 に RX671 グループを追加
		11	表 1-3 の説明を変更 • ブートローダモジュールの説明を変更 ブートローダモジュールで R_FWUP_Open, R_FWUP_Close を使用するよう変更
		17	「2.7 コードサイズ」に GCC for Renesas RX でのコードサイズを追記。
			「2.7 コードサイズ」の GCC for Renesas RX に、RX65N の ”boot_loader Project” と “aws_demos Project” 以外のコードサイズを追記。
		19	ブートローダの使用 ROM、RAM サイズの記述を追加
		21	表 3-1 の説明にブートローダモジュールを追記
			表 3-2 の説明にブートローダモジュールを追記
		34	「5.1 動作確認環境」の表 5-1 を Rev.1.02 の動作確認環境に修正。
			デモプロジェクトで使用した FIT モジュールのバージョン一覧として表 5-2、表 5-3 を追記
		35	「5.2 コンパイラ依存の設定」を追加。
		36	表 5-3 に、RX65N 以外の GCC 環境のデモプロジェクトで使用した FIT モジュールのバージョンを追記
		39	5.2.2.1 コンパイルオプションにブートローダをデバッグする際の最適化レベル設定を追記

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。