RX ファミリ

ファームウェアアップデート通信モジュール Firmware Integration Technology

要旨

本アプリケーションノートは Firmware Integration Technology(FIT)を使用したファームウェアアップデート通信モジュールについて説明します。

本モジュールを使用して、プライマリ MCU—セカンダリ MCU 構成のシステムで、セカンダリ MCU のファームウェア更新を実現できます。本アプリケーションノートでは、本モジュールの使用方法、ユーザアプリケーションへの組込み方法、および拡張方法について説明します。

また、本アプリケーションノートのリリースパッケージにはデモプロジェクトが含まれています。「5 デモプロジェクト」に記載する手順に沿ってデモの実行環境を構築することで、本モジュールを使用したセカンダリ MCU のファームウェア更新の基本的な動作を確認することができます。

動作確認デバイス

RX140 グループ

RX23E-B グループ

RX261 グループ

RX66T グループ

RX660 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

- 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 SCI モジュール Firmware Integration Technology(R01AN1815)
- 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(R01AN6850)

ターゲットコンパイラ

- Renesas Electronics 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ついては 「6.1 動作確認環境」を参照してください。

目次

1. 概要 4
1.1 ファームウェアアップデート通信モジュールとは 4
1.2 サポートする通信 IP とハードウェア構成 4
1.3 ソフトウェア構成 4
1.3.1 UART 通信設定 5
1.4 パケット通信 5
1.5 データフォーマット 6
1.5.1 パケットのデータフォーマット 6
1.6 コマンド仕様 7
1.6.1 Common コマンド 7
1.6.2 FWUP コマンド 8
1.6.2.2 FWUP コマンドの通信フロー 10
1.7 エラーハンドリング 11
1.8 API の概要 11

2. API 情報 12
2.1 ハードウェアの要求 12
2.2 ソフトウェアの要求 12
2.3 サポートされているツールチェーン 12
2.4 ヘッダファイル 12
2.5 整数型 12
2.6 コンパイル時の設定 13
2.7 サンプルプロジェクトのコードサイズ 15
2.8 引数 17
2.9 戻り値 19
2.10 FIT モジュールの追加方法 19
2.11 for 文、while 文、do while 文について 19

3. API 関数 21
3.1 R_FWUPCOMM_Open 関数 21
3.2 R_FWUPCOMM_Close 関数 21
3.3 R_FWUPCOMM_CmdSend 関数 22
3.4 R_FWUPCOMM_ProcessCmdLoop 関数 23

4. 本モジュールの拡張方法 24
4.1 コマンドの追加 24
4.2 通信方式の変更 27
4.2.1 通信インターフェース 27
4.2.1.1 fwupcomm_err_t (*open)(void) 27
4.2.1.2 void (*close)(void) 27
4.2.1.3 fwupcomm_err_t (*send)(uint8_t *src, uint16_t size) 28
4.2.1.4 fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size) 28
4.2.1.5 void (*rx_flush)(void) 28
4.2.2 通信方式の変更方法 29

5. デモプロジェクト 30
5.1 デモプロジェクトの構成 30
5.1.1 プライマリ MCU 30
5.1.2 セカンダリ MCU 30
5.2 動作環境準備 31
5.2.1 TeraTerm のインストール 31
5.2.2 Python 実行環境のインストール 31
5.2.3 フラッシュライタのインストール 31
5.3 プロジェクトの実行手順 32
5.3.1 実行環境 32
5.3.2 デモプロジェクトの構築 32
5.3.2.1 プライマリ MCU 32
5.3.2.2 セカンダリ MCU 用の初期イメージと更新イメージを作成 32
5.3.3 初期イメージの書き込み 33
5.3.4 ファームウェアアップデートの実行 34

6. 付録 36
6.1 動作確認環境 36
6.2 UART 通信設定 36
6.3 デモプロジェクトの動作環境 37
6.3.1 RX140 の動作確認環境 37
6.3.2 RX23E-B の動作確認環境 38
6.3.3 RX261 の動作確認環境 39
6.3.4 RX66T の動作確認環境 40
6.3.5 RX660 の動作確認環境 41

改訂記録 42

1. 概要

1.1 ファームウェアアップデート通信モジュールとは

ファームウェアアップデート通信モジュールとは、図 1-1 に示すようなプライマリ MCU—セカンダリ MCU 構成のシステムで、セカンダリ MCU がプライマリ MCU から更新ファームウェアを受け取ってファームウェア更新する際に、MCU 間の通信を制御するミドルウェアです。プライマリ MCU とセカンダリ MCU の両方に本モジュールを組み込むことで、容易にセカンダリ MCU のファームウェア更新を実現できます。

1.2 サポートする通信 IP とハードウェア構成

本モジュールは、通信インターフェースにシリアルコミュニケーションインターフェース(SCI)を用いた UART 通信に対応しています。本モジュールが想定するハードウェア構成を図 1-1 に示します。プライマリ MCU とセカンダリ MCU は 2 線 UART(TXD, RXD)で同一バス上に 1 対 1 接続します。

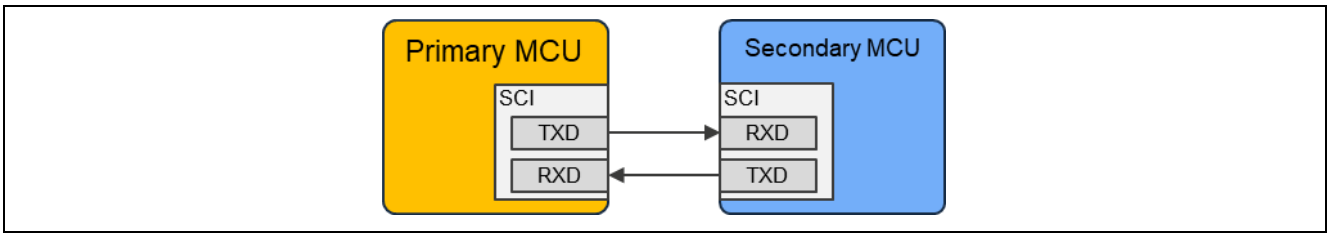

図 1-1 ハードウェア構成図

1.3 ソフトウェア構成

ソフトウェアモジュール構成を図 1-2(プライマリ MCU)、図 1-3(セカンダリ MCU)に示します。本モジュールは、ベアメタル、FreeRTOS のプロジェクトで利用可能です。

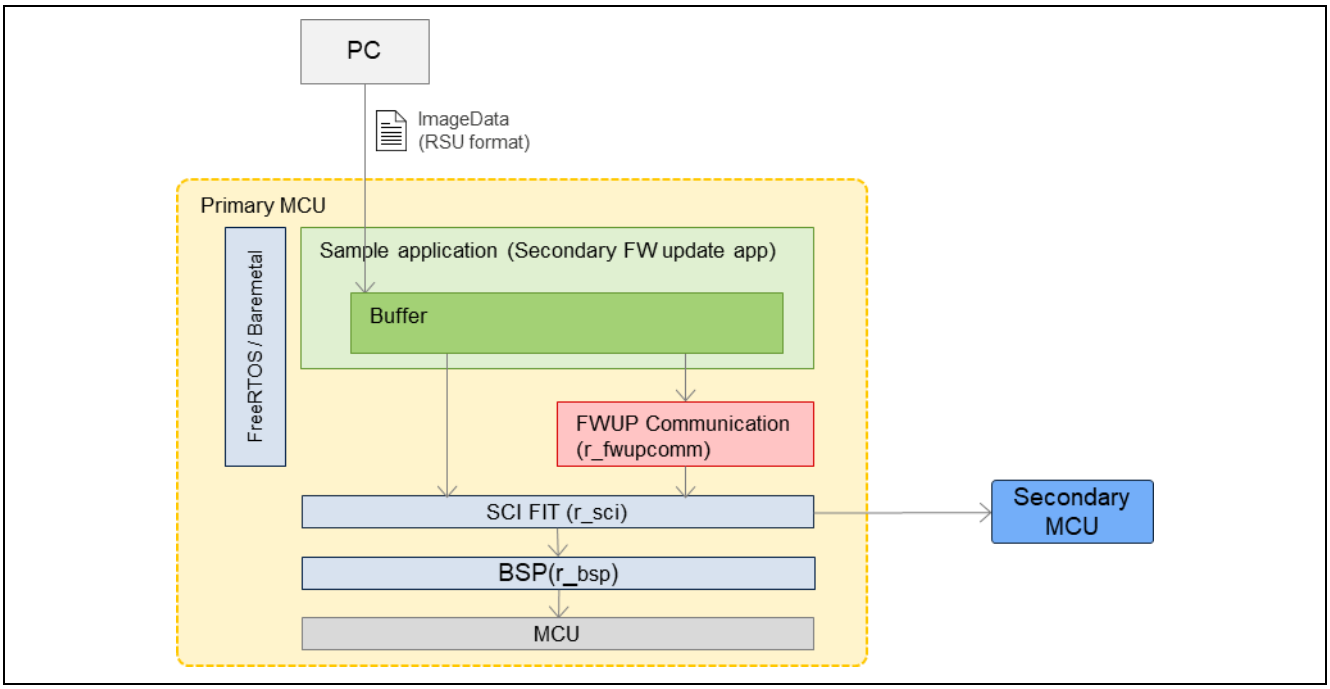

図 1-2 プライマリ MCU のソフトウェアモジュール構成

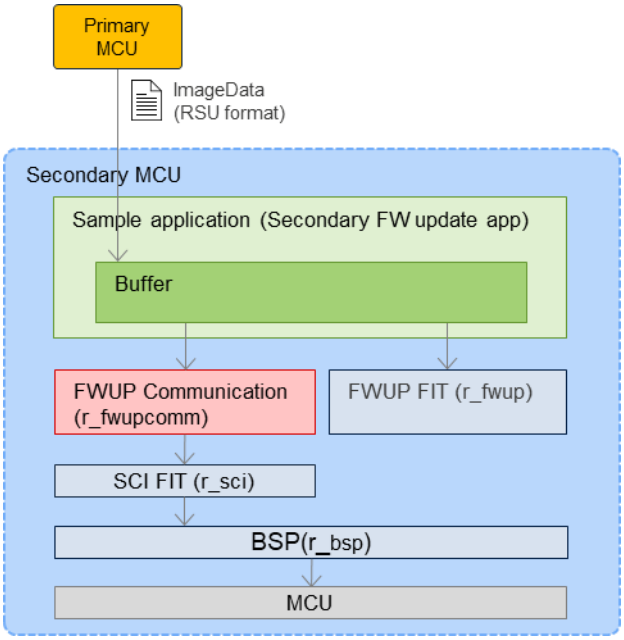

図 1-3 セカンダリ MCU のソフトウェアモジュール構成

1.3.1　UART 通信設定

本モジュールは「6.2 UART 通信設定」に示す UART 通信設定で動作確認しています。

1.4　パケット通信

プライマリ MCU とセカンダリ MCU 間は通信インターフェース上でパケット通信を行います。プライマリ MCU は、セカンダリ MCU に対してリクエストパケットを送信します。セカンダリ MCU はリクエストパケットを受信すると、そのコマンドに応じた処理を実施し、結果をレスポンスパケットとしてプライマリ MCU に対して送信します。パケット通信のフローを図 1-4 に示します。

プライマリ MCU		セカンダリ MCU
リクエストパケット送信		
	----->	
		リクエストパケット受信
		コマンドに応じた処理を実行
		レスポンスパケット送信
	<-----	
レスポンスパケット受信		

図 1-4 パケット通信のフロー図

全てのコマンドは、そのコマンドの目的ごとに分類されており、Command class と呼んでいます。

1.5　データフォーマット

プライマリ MCU とセカンダリ MCU 間で行うパケット通信の仕様を説明します。MCU 間の物理的な通信方式に依存しない形でデータフォーマットを規定しています。

1.5.1　パケットのデータフォーマット

リクエストパケットのデータフォーマットを図 1-5 に示します。Command header と Command data で構成されています。

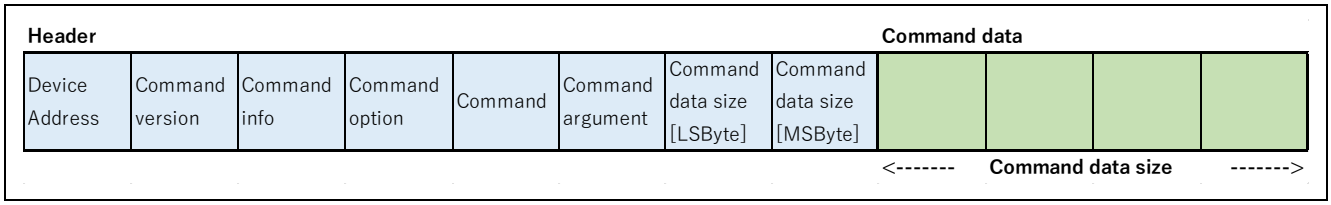

図 1-5 リクエストパケットのデータフォーマット

レスポンスパケットのデータフォーマットを図 1-6 に示します。

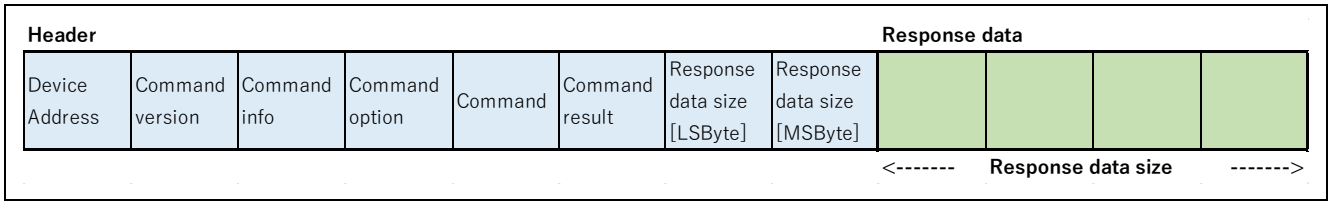

図 1-6 レスポンスパケットのデータフォーマット

パケットの Header 仕様を表 1-1 に示します。

表 1-1 パケットの Header 仕様

項目	内容
Device address	コマンド送信先のセカンダリ MCU のデバイスアドレスです。 セカンダリ MCU は自身宛の場合のみ、コマンドに対する処理を行います。 ● 0x00 – 0xFE: セカンダリ MCU のデバイスアドレス ● 0xFF: Reserved
Command version	コマンドのバージョン。セカンダリ MCU は自身のコマンドのバージョンと等しい場合のみ、コマンドに対する処理を行います。 0x00 – 0xFF
Command info	● b7:　0: Command / 1: Response ● b4 – b6: Command class。「1.6 コマンド仕様」参照。 ● b0 – b3: Command ID。対応するコマンドとレスポンスパケットで同一の ID 値。
Command option	● b7: 0: レスポンスを送信する / 1: レスポンスを送信しない ● b0 – b6: Reserved
Command	コマンドを表す値です。「1.6 コマンド仕様」参照。
Command argument / result	コマンド送信時は、コマンドの引数です。 レスポンス送信時は、コマンドの実行結果です。 「1.6 コマンド仕様」参照。
Command / Response data size	Command data もしくは Response data のサイズです。 単位はバイトで、4 の倍数である必要があります。

1.6　コマンド仕様

本モジュールでは Command class として、セカンダリ MCU のファームウェア更新を制御する FWUP コマンドと、汎用的なデータ通信に利用可能な Common コマンドを定義しています。

表 1-2 Command class リスト

Command class	内容	値
Common コマンド	汎用コマンド群	0x00
FWUP コマンド	セカンダリ MCU のファームウェア更新制御用コマンド群	0x01

1.6.1　Common コマンド

汎用的な目的で使用可能なコマンド群です。表 1-3 にコマンド一覧を示します。

表 1-3　Common コマンドリスト

Command	内容	コマンド値
DATA_SEND：データ送信コマンド	任意サイズのデータをセカンダリ MCU に送信	0x01
DATA_RECV：データ受信コマンド	任意サイズのデータの送信をセカンダリ MCU に要求	0x02

(1)　DATA_SEND：データ送信コマンド

セカンダリ MCU にデータを送信します。

表 1-4 COMMON DATA_SEND コマンド仕様

項目	値
Command	0x01
Command argument	0x00
Command result	0x00: 処理に成功 / 0x01: 処理に失敗
Command data size	任意のデータ長（「2.6 コンパイル時の設定」で設定可能）
Response data size	0
Command data	任意のデータ
Response data	なし

(2)　DATA_RECV：データ受信コマンド

セカンダリ MCU にデータ送信を要求します。

表 1-5 COMMON DATA_RECV コマンド仕様

項目	値
Command	0x02
Command argument	0x00
Command result	0x00: 処理に成功 / 0x01: 処理に失敗
Command data size	0
Response data size	任意のデータ長（「2.6 コンパイル時の設定」で設定可能）
Command data	なし
Response data	任意のデータ

1.6.2　FWUP コマンド

ファームウェア更新時に使用するコマンド群です。表 1-6 にコマンド一覧を示します。

表 1-6　FWUP コマンドリスト

Command	内容	コマンド値
START：FW 更新開始コマンド	FW 更新開始	0x01
WRITE：更新 FW 書き込みコマンド	更新 FW の書き込み	0x02
INSTALL：更新 FW インストールコマンド	更新 FW のインストールと実行	0x03
CANCEL：FW 更新キャンセルコマンド	FW 更新の中止	0x04

(1)　START：FW 更新開始コマンド

セカンダリ MCU にファームウェア更新開始を要求します。

Command data には任意のデータ長を設定できます。ファームウェア更新開始時のユーザ側での初期化処理に必要なデータの送信に利用可能です。

本コマンドを受信したセカンダリ MCU は、更新ファームウェアデータを受信可能な状態にします。

ファームウェア更新開始時は、最初にこのコマンドを送信します。

表 1-7 FWUP START コマンド仕様

項目	値
Command	0x01
Command argument	0x00
Command result	0x00: 処理に成功 / 0x02: 処理に失敗
Command data size	任意のデータ長（「2.6 コンパイル時の設定」で設定可能）
Response data size	0
Command data	任意のデータ
Response data	なし

(2) WRITE：更新 FW 書き込みコマンド

セカンダリ MCU に更新 FW データを送信し、FW 書き込みを要求します。

セカンダリ MCU は書き込み処理を実行します。更新 FW データが最終ブロックの場合、さらに署名検証処理を実行します。

表 1-8 FWUP WRITE コマンド仕様

項目	値
Command	0x02
Command argument	0x00
Command result	0x00: 処理に成功 / 0x01: 署名検証に成功 / 0x02: 処理に失敗
Command data size	セカンダリ MCU の ROM 書き込み単位の整数倍であること （「2.6 コンパイル時の設定」で設定可能）
Response data size	0x04
Command data	更新 FW データ
Response data	残りの更新 FW サイズ

(3) INSTALL：更新 FW インストールコマンド

セカンダリ MCU に書き込まれた更新 FW のインストールと実行を要求します。

表 1-9 FWUP INSTALL コマンド仕様

項目	値
Command	0x03
Command argument	0x00
Command result	0x00: 処理に成功 / 0x02: 処理に失敗
Command data size	0
Response data size	0
Command data	なし
Response data	なし

(4) CANCEL：FW 更新キャンセルコマンド

セカンダリ MCU に FW 更新中止を要求します。

セカンダリ MCU は更新を中断し、書き込んだ更新 FW の消去処理を実行します。

表 1-10 FWUP CANCEL コマンド仕様

項目	値
Command	0x04
Command argument	0x00
Command result	0x00: 処理に成功 / 0x02: 処理に失敗
Command data size	0
Response data size	0
Command data	なし
Response data	なし

1.6.2.2 FWUP コマンドの通信フロー

FWUP コマンドを用いたセカンダリ MCU のファームウェア更新時のコマンド通信フローを図 1-7 に示します。

プライマリ MCU		セカンダリ MCU
FWUP START コマンド送信		
	----->	
		FWUP START コマンド受信
		更新ファームウェアを受信できる状態に遷移
		FWUP START レスポンス送信
	<-----	
FWUP START レスポンス受信		
FWUP WRITE コマンド送信		
	----->	
		FWUP WRITE コマンド受信
		FWUP FIT の API を利用し、受信した更新ファームウェアデータを ROM に書き込む
		FWUP WRITE レスポンス送信
	<-----	
FWUP WRITE レスポンス受信		
全ての更新ファームウェアデータを受信するまで FWUP WRITE コマンドの通信を繰り返す		
FWUP INSTALL コマンド送信		
	----->	
		FWUP INSTALL コマンド受信
		更新ファームウェアをインストールし、レスポンスを送信後に更新ファームウェアを実行する準備を行う
		FWUP INSTALL レスポンス送信
	<-----	
FWUP INSTALL レスポンス受信		
		更新ファームウェアを実行

図 1-7 FWUP コマンドの通信フロー図

1.7　エラーハンドリング

セカンダリ MCU は、受信したリクエストパケットの Header 解析に失敗した場合、受信した Command header をプライマリ MCU に送信します。但し、Command version はセカンダリ MCU で設定された Command version に上書きされます。また、Command data size は 0 に上書きされます。この時、コマンドに対応する処理は実行されません。リクエストパケットの Header 解析に失敗するのは次のようなケースです。

- 受信したリクエストパケットの Header が定義されている仕様と異なる
- 受信したリクエストパケットの Command version がセカンダリ MCU で設定された Command version と異なる
- Command class または Command が未定義値である
- Command data size で指定されたデータサイズ分の Command data を受信できなかった

プライマリ MCU 側は、受信したパケットの Command info の最上位ビットが 0: Command であることを確認することで、セカンダリ MCU 側での Header 解析失敗を検出できます。

1.8　API の概要

本モジュールに含まれる API 関数を表 1-11 に示します。

表 1-11 API 関数一覧

関数	関数説明
R_FWUPCOMM_Open()	本モジュール及び本モジュール内で使用する通信チャネルをオープンします。
R_FWUPCOMM_Close()	本モジュール及び本モジュール内で使用する通信チャネルをクローズします。
R_FWUPCOMM_CmdSend()	セカンダリ MCU に対してコマンドを送信し、それに対する応答を受信します。
R_FWUPCOMM_ProcessCmdLoop()	プライマリ MCU からのコマンドを受信し、対応するハンドラを実行します。その後、コマンドの実行結果を送信します。

2. API 情報

本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。

- SCI

2.2 ソフトウェアの要求

本モジュールは以下のドライバに依存しています。

- ボードサポートパッケージ(r_bsp)
- シリアルコミュニケーションインターフェース(r_sci)

2.3 サポートされているツールチェーン

本モジュールは「6.1 動作確認環境」に示すツールチェーンで動作確認しています。

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ーフェース定義は r_fwupcomm_if.h ファイルに記載されています。

r_fwupcomm_config.h ファイルに、ビルド時に設定可能なコンフィギュレーションオプションを定義します。

2.5 整数型

このモジュールは 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_fwupcomm_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 コンフィグレーション設定に示します。

表 2-1 コンフィグレーション設定(r_fwupcomm_config.h)

コンフィグレーションオプション(r_fwupcomm_config.h)	
FWUPCOMM_CFG_PARAM_CHECKING_ENABLE ※デフォルトは “0”	0：ビルド時にパラメータチェックの処理をコードから省略します。 1：ビルド時にパラメータチェックの処理をコードに含めます。 このオプションに BSP_CFG_PARAM_CHECKING_ENABLE を設定すると、システムのデフォルト設定が使用されます。
FWUPCOMM_CFG_DEVICE_PRIMARY ※デフォルトは “0”	0：セカンダリ MCU 1：プライマリ MCU
FWUPCOMM_CFG_SCI_UART_CHANNEL ※デフォルトは “1”	通信で使用する SCI チャネル番号を設定します。
FWUPCOMM_CFG_SCI_UART_BITRATE ※デフォルトは “115200”	UART 通信のビットレートを設定します。
FWUPCOMM_CFG_SCI_UART_INT_PRIORITY ※デフォルトは “15”	通信で使用する SCI チャネルの割込み優先度を設定します。
FWUPCOMM_CFG_SEND_PACKET_BUFFER_SIZE ※デフォルトは “1500”	コマンドの送信バッファサイズを設定します。
FWUPCOMM_CFG_RECV_PACKET_BUFFER_SIZE ※デフォルトは “1500”	コマンドの受信バッファサイズを設定します。
FWUPCOMM_CFG_DEVICE_ADDRESS ※デフォルトは “0xA0”	このデバイスの固有アドレスを設定します。
FWUPCOMM_CFG_CMD_SEND_TIMEOUT ※デフォルトは “500”	通信の送信タイムアウト時間を設定します。単位はミリ秒です。
FWUPCOMM_CFG_CMD_RECV_TIMEOUT ※デフォルトは “500”	通信の受信タイムアウト時間を設定します。単位はミリ秒です。
FWUPCOMM_CFG_CMD_COMMON_ENABLE ※デフォルトは “1”	Common コマンドを有効にするか選択します。
FWUPCOMM_CFG_CMD_HANDLER_COMMON ※デフォルトは “R_FWUPCOMM_CmdHandler_Common”	Common コマンドを受信したときに呼び出されるハンドラ関数名を設定します。
FWUPCOMM_CFG_CMD_FWUP_ENABLE ※デフォルトは “1”	FWUP コマンドを有効にするか選択します。
FWUPCOMM_CFG_CMD_HANDLER_FWUP ※デフォルトは “R_FWUPCOMM_CmdHandler_FWUP”	FWUP コマンドを受信したときに呼び出されるハンドラ関数名を設定します。
FWUPCOMM_CFG_CMD_VER ※デフォルトは “1”	コマンドのバージョンを設定します。
FWUPCOMM_CFG_CMD_FWUP_START_DATA_SIZE ※デフォルトは “0”	FWUP_START コマンドに付加するデータサイズを設定します。
FWUPCOMM_CFG_CMD_FWUP_WRITE_FW_BLOCK_SIZE ※デフォルトは “1024”	FWUP_WRITE コマンドに付加する FW ブロックサイズを設定します。
FWUPCOMM_CFG_CMD_COMMON_MAX_DATA_SIZE ※デフォルトは “10”	COMMON コマンドに付加するデータサイズの最大値を設定します。

本モジュールが使用する SCI FIT 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_sci_rx_config.h で行います。

SCI FIT モジュールに対する設定オプション名および設定値に関する説明を表 2-2 に示します。オプションの詳細は、「RX ファミリ SCI モジュール Firmware Integration Technology (R01AN1815)」を参照してください。

表 2-2 コンフィグレーション設定 r_sci rx_config.h)

コンフィグレーションオプション(r_sci_rx_config.h)	
SCI_CFG_CHx_INCLUDED ※1. CHx = CH0～CH12 ※2. 各デフォルト値は以下のとおり: CH0=1、CH1～CH12: 0	チャネルごとに送受信バッファ、カウンタ、割り込み、その他のプログラム、RAM などのリソースを持ちます。このオプションを“1”に設定すると、そのチャネルに関連したリソースが割り当てられます。 FWUPCOMM_CFG_SCI_UART_CHANNEL で指定した SCI チャネル番号を指定してください。
SCI_CFG_CHx_TX_BUFSIZ ※1. CHx = CH0～CH12 ※2. 各デフォルト値は 80	調歩同期式モードで、各チャネルの送信キューに使用されるバッファサイズを指定します。 FWUPCOMM_CFG_SEND_PACKET_BUFFER_SIZE で指定したバッファサイズを設定してください。
SCI_CFG_CHx_RX_BUFSIZ ※1. CHx = CH0～CH12 ※2. 各デフォルト値は 80	調歩同期式モードで、各チャネルの受信キューに使用されるバッファサイズを指定します。 FWUPCOMM_CFG_RECV_PACKET_BUFFER_SIZE で指定したバッファサイズを設定してください。
SCI_CFG_TEI_INCLUDED ※デフォルト値は"0"	シリアル送信の送信完了割り込みを有効にします。 本 FIT モジュールではシリアル送信完了割り込みを使用するため、”1”を設定してください。

2.7 サンプルプロジェクトのコードサイズ

本アプリケーションノートのパッケージに含まれるサンプルプロジェクトの ROM、RAM サイズを表 2-3 に示します。この表の値は以下の条件で確認しています。

モジュールリビジョン : r_fwupcomm rev.1.00

コンパイラバージョン : Renesas Electronics C/C++ Compiler for RX Family V3.07.00

GCC for Renesas RX 8.3.0.202411

CC-RX

- 最適化レベル(-optimize) : Level 2: Performs whole module optimization
- 最適化タイプ(-speed/-size): Optimizes with emphasis on code size
- 一度も参照のない変数／関数を削除する(-optimize=symbol_delete)

GCC

- 最適化レベル : サイズ(-Os)

表 2-3 サンプルプロジェクト（半面更新）の ROM、RAM サイズ

ROM、RAM のコードサイズ				
デバイス	分類	使用メモリ(単位: byte)		プロジェクト名
		CC-RX	GCC	
RX140	ROM	29665	31248	app_rx140_fpb_w_buffer
		30018	28128	bootloader_rx140_fpb_w_buffer
	RAM	9040	14332	app_rx140_fpb_w_buffer
		6975	6908	bootloader_rx140_fpb_w_buffer
RX23E-B	ROM	35253	31024	app_rx23eb_rssk_w_buffer
		29838	27996	bootloader_rx23eb_rssk_w_buffer
	RAM	10412	10364	app_rx23eb_rssk_w_buffer
		7096	7036	bootloader_rx23eb_rssk_w_buffer
RX261	ROM	36037	31968	app_rx261_fpb_w_buffer
		30401	28544	bootloader_rx261_fpb_w_buffer
	RAM	10364	10364	app_rx261_fpb_w_buffer
		7356	7292	bootloader_rx261_fpb_w_buffer
RX66T	ROM	37396	35220	app_rx66t_rsk_w_buffer
		31587	31024	bootloader_rx66t_rsk_w_buffer
	RAM	10785	14588	app_rx66t_rsk_w_buffer
		7581	11388	bootloader_rx66t_rsk_w_buffer
RX660	ROM	38419	34108	app_rx660_tb_w_buffer
		31987	30664	bootloader_rx660_tb_w_buffer
	RAM	10640	14716	app_rx660_tb_w_buffer
		7436	11516	bootloader_rx660_tb_w_buffer

表 2-4 サンプルプロジェクト（全面更新）の ROM、RAM サイズ

ROM、RAM のコードサイズ				
デバイス	分類	使用メモリ(単位: byte)		プロジェクト名
		CC-RX	GCC	
RX140	ROM	25681	21308	app_rx140_fpb_wo_buffer
		29328	31184	bootloader_rx140_fpb_wo_buffer
	RAM	13380	13308	app_rx140_fpb_wo_buffer
		8869	10108	bootloader_rx140_fpb_wo_buffer
RX23E-B	ROM	25783	21392	app_rx23eb_rssk_wo_buffer
		13823	13820	bootloader_rx23eb_rssk_wo_buffer
	RAM	14629	14629	app_rx23eb_rssk_wo_buffer
		8986	10236	bootloader_rx23eb_rssk_wo_buffer
RX261	ROM	26574	22356	app_rx261_fpb_wo_buffer
		29830	32232	bootloader_rx261_fpb_wo_buffer
	RAM	13777	13692	app_rx261_fpb_wo_buffer
		9107	10492	bootloader_rx261_fpb_wo_buffer
RX66T	ROM	28155	24788	app_rx66t_rsk_wo_buffer
		37654	34804	bootloader_rx66t_rsk_wo_buffer
	RAM	14350	13564	app_rx66t_rsk_wo_buffer
		10641	14460	bootloader_rx66t_rsk_wo_buffer
RX660	ROM	28139	24708	app_rx660_tb_wo_buffer
		38048	34420	bootloader_rx660_tb_wo_buffer
	RAM	13729	14204	app_rx660_tb_wo_buffer
		10496	14588	bootloader_rx660_tb_wo_buffer

2.8 引数

API 関数の引数で使用する構造体、列挙体の定義を示します。これらは API 関数のプロトタイプ宣言とともに r_fwupcomm_if.h で記載されています。

```
/* タイマーインターフェースを登録する際に使用する構造体 */
typedef struct r_fwupcomm_timer
{
    r_fwupcomm_start_timer_t start; // タイマーのカウント開始関数へのポインタ
    r_fwupcomm_stop_timer_t stop;   // タイマーのカウント停止関数へのポインタ
} r_fwupcomm_timer_t;
```

```
/* 初期化時に Open 関数の引数として使用する構造体 */
typedef struct r_fwupcomm_cfg
{
    r_fwupcomm_timer_t timer; // タイマーインターフェース
} r_fwupcomm_cfg_t;
```

```
/* コマンド情報を指定する構造体 */
struct r_fwupcomm_cmd_info
{
    uint8_t device_address; // コマンド送信先のデバイスアドレス
    uint8_t class;          // Command class
    uint8_t type;           // Command
    uint8_t arg;            // Command argument
    uint16_t data_size;     // Command data サイズ
    const void *data;       // Command data へのポインタ
    uint8_t id;             // Command ID
};
```

```
/* レスポンス情報を格納する構造体 */
struct r_fwupcomm_resp_info
{
    int8_t result;       // Command result
    void *data;          // Response data の格納先へのポインタ
    uint16_t data_size;  // Response data の格納先のサイズ
};
```

```
/* コマンド送信時に CmdSend 関数の引数として使用する構造体 */
struct r_fwupcomm_cmd_instr
{
    uint16_t timeout_ms;          // コマンド送信からレスポンス受信までのタイムアウト時間
    r_fwupcomm_cmd_info_t cmd;    // コマンド情報
    r_fwupcomm_resp_info_t resp;  // レスポンス情報格納先
};
```

```
/* Command class を定義する列挙型 */
typedef enum
{
    FWUPCOMM_CMD_CLS_COMMON = 0,    // Common コマンド
    FWUPCOMM_CMD_CLS_FWUP,          // FWUP コマンド
    FWUPCOMM_CMD_NUM_CLS            // 定義されている Command class の数
} r_fwupcomm_cmd_class_t;
```

```
/* Common command class のコマンドを定義する列挙型 */
typedef enum
{
    FWUPCOMM_CMD_COMMON_DATA_SEND  = 0,   // DATA_SEND コマンド
    FWUPCOMM_CMD_COMMON_DATA_RECV,        // DATA_RECV コマンド
    FWUPCOMM_CMD_COMMON_NUM_COMMANDS      // 定義されている Common コマンドの数
} r_fwupcomm_cmd_type_common_t;
```

```
/* FWUP command class のコマンドを定義する列挙型 */
typedef enum
{
    FWUPCOMM_CMD_FWUP_START = 0,          // START コマンド
    FWUPCOMM_CMD_FWUP_WRITE,              // WRITE コマンド
    FWUPCOMM_CMD_FWUP_INSTALL,            // INSTALL コマンド
    FWUPCOMM_CMD_FWUP_CANCEL,             // CANCEL コマンド
    FWUPCOMM_CMD_FWUP_NUM_COMMANDS        // 定義されている FWUP コマンドの数
} r_fwupcomm_cmd_type_fwup_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_fwupcomm_if.h で記載されています。

```
typedef enum
{
    FWUPCOMM_SUCCESS = 0,
    FWUPCOMM_ERR_INVALID_PTR,        // 引数で渡されたポインタ変数が不正です。
    FWUPCOMM_ERR_INVALID_ARG,        // 引数で渡されたパラメータが不正です。
    FWUPCOMM_ERR_NOT_OPEN,           // モジュールが初期化されていません。
    FWUPCOMM_ERR_ALREADY_OPEN,       // モジュールは既に初期化されています。
    FWUPCOMM_ERR_INVALID_CMD,        // 引数で渡されたコマンドが不正です。
    FWUPCOMM_ERR_INVALID_RESP,       // 受信した応答が不正です。
    FWUPCOMM_ERR_RECV_RESP_TIMEOUT,  // 応答を受信する前にタイムアウトしました。
    FWUPCOMM_ERR_NO_CMD,             // コマンドを受信していません。
    FWUPCOMM_ERR_CH_ALREADY_OPEN,    // 通信チャネルが別のモジュールによって使用されています。
    FWUPCOMM_ERR_CH_SEND,            // 通信チャネルがデータ送信に失敗しました。
    FWUPCOMM_ERR_CH_SEND_BUSY,       // 通信チャネルがビジー状態のためデータ送信に失敗しました。
    FWUPCOMM_ERR_CH_RECV,            // 通信チャネルが受信に失敗しました。
    FWUPCOMM_ERR_CH_RECV_NO_DATA,    // 通信チャネルに十分な受信データがありません。
} fwupcomm_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、e^2 studio の環境では、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) e^2 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

e^2 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: e^2 studio 編 (R20AN0451)」を参照してください。

(2) e^2 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

e^2 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 e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

2.11 for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

while 文の例：

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

for 文の例：

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

do while 文の例：

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET));
/* WAIT_LOOP */
```

3. API 関数

3.1 R_FWUPCOMM_Open 関数

表 3-1 R_FWUPCOMM_Open 関数仕様

Format	fwupcomm_err_t R_FWUPCOMM_Open(r_fwupcomm_hdl_t *hdl, void *cfg)	
Description	本モジュール及び本モジュール内で使用する通信チャネルをオープンします。この関数は他の API 関数を使用する前に実行される必要があります。	
Parameters	hdl：モジュールのハンドラ cfg：モジュールの初期化に必要な情報を持った構造体変数	
Return Values	FWUPCOMM_SUCCESS	正常に初期化されました。
	FWUPCOMM_ERR_INVALID_PTR	引数で入力されたポインタが NULL です。
	FWUPCOMM_ERR_ALREADY_OPEN	既にオープン済です。
	FWUPCOMM_ERR_CH_ALREADY_OPEN	通信チャネルが既にオープン済です。
	FWUPCOMM_ERR_NOT_OPEN	通信チャネルの初期化に失敗しました。
Special Notes	—	

例:

```
fwupcomm_err_t fwupcomm_err;
r_fwupcomm_hdl_t fwupcomm_hdl = {0};
r_fwupcomm_cfg_t fwupcomm_cfg;
fwupcomm_cfg.timer.start = demo_start_timer;
fwupcomm_cfg.timer.stop = demo_stop_timer;

fwupcomm_err = R_FWUPCOMM_Open(&fwupcomm_hdl, &fwupcomm_cfg);
```

3.2 R_FWUPCOMM_Close 関数

表 3-2 R_FWUPCOMM_Close 関数仕様

Format	fwupcomm_err_t R_FWUPCOMM_Close(r_fwupcomm_hdl_t *hdl)	
Description	本モジュール及び本モジュール内で使用する通信チャネルをクローズします。	
Parameters	hdl：モジュールのハンドラ	
Return Values	FWUPCOMM_SUCCESS	正常にクローズされました。
	FWUPCOMM_ERR_NOT_OPEN	モジュールはオープンされていません。
	FWUPCOMM_ERR_INVALID_PTR	引数で入力されたポインタが NULL です。
Special Notes	—	

例:

```
fwupcomm_err = R_FWUPCOMM_Close(&fwupcomm_hdl);
```

3.3 R_FWUPCOMM_CmdSend 関数

表 3-3 R_FWUPCOMM_CmdSend 関数仕様

Format	fwupcomm_err_t R_FWUPCOMM_CmdSend(r_fwupcomm_hdl_t *hdl, r_fwupcomm_cmd_instr_t *cmd_instr)	
Description	セカンダリ MCU に対してコマンドを送信し、それに対する応答を受信します。	
Parameters	hdl：モジュールのハンドラ cmd_instr：送信するコマンド情報、レスポンスの格納先情報を持った構造体変数	
Return Values	FWUPCOMM_SUCCESS	正常終了しました。
	FWUPCOMM_ERR_NOT_OPEN	モジュールはオープンされていません。
	FWUPCOMM_ERR_INVALID_PTR	引数で入力されたポインタが NULL です。
	FWUPCOMM_ERR_INVALID_ARG	引数で入力されたパラメータが不正です。
	FWUPCOMM_ERR_CH_SEND	通信チャネルが送信処理に失敗しました。
	FWUPCOMM_ERR_CH_RECV	通信チャネルが受信処理に失敗しました。
	FWUPCOMM_ERR_RECV_RESP_TIMEOUT	コマンドの応答待ちがタイムアウトしました。
Special Notes	—	

例:

```
r_fwupcomm_cmd_info_t cmd = {0};
r_fwupcomm_resp_info_t resp = {0};
uint8_t resp_data[4] = {0};


cmd.device_address = 0xA0;
cmd.class = FWUPCOMM_CMD_CLS_FWUP;
cmd.type = FWUPCOMM_CMD_FWUP_START;
cmd.arg = 0;
cmd.data = NULL;
cmd.data_size = 0;

resp.data = resp_data;

r_fwupcomm_cmd_instr_t cmd_instruction =
{
    .timeout_ms = 500U,
    .cmd = cmd,
    .resp = resp
};

fwupcomm_err = R_FWUPCOMM_CmdSend(&fwupcomm_hdl, &cmd_instruction);
```

3.4 R_FWUPCOMM_ProcessCmdLoop 関数

表 3-4 R_FWUPCOMM_ProcessCmdLoop 関数仕様

Format	fwupcomm_err_t R_FWUPCOMM_ProcessCmdLoop(r_fwupcomm_hdl_t *hdl)	
Description	プライマリ MCU からのコマンドを受信し、対応するハンドラを実行します。その後、コマンドの実行結果を送信します。コマンドを待ち受けしているセカンダリ MCU は、この関数を定期的に実行してください。	
Parameters	hdl：モジュールのハンドラ	
Return Values	FWUPCOMM_SUCCESS	正常終了しました。
	FWUPCOMM_ERR_NOT_OPEN	モジュールはオープンされていません。
	FWUPCOMM_ERR_INVALID_PTR	引数で入力されたポインタが NULL です。
	FWUPCOMM_ERR_INVALID_ARG	引数で入力されたパラメータが不正です。
	FWUPCOMM_ERR_NO_CMD	コマンドを受信しませんでした。
	FWUPCOMM_ERR_INVALID_CMD	不正なコマンドを受信しました。
	FWUPCOMM_ERR_CH_SEND	通信チャネルが送信処理に失敗しました。
	FWUPCOMM_ERR_CH_RECV	通信チャネルが受信処理に失敗しました。
Special Notes	—	

例:

```
do
{
    fwupcomm_err = R_FWUPCOMM_ProcessCmdLoop(&fwupcomm_hdl);
}while((FWUPCOMM_SUCCESS == fwupcomm_err)||(FWUPCOMM_ERR_NO_CMD == fwupcomm_err));
```

4. 本モジュールの拡張方法

本モジュールのコマンドの追加方法と、通信方式の変更方法について説明します。

4.1 コマンドの追加

本モジュールに予め定義されている FWUP コマンド、Common コマンドに追加して任意のコマンドを定義する方法を説明します。ここでは、「UserDefined」という Command class 名の「ADDITIONAL1」、「ADDITIONAL2」の 2 つのコマンドを追加します。

(1) UserDefined コマンドを定義するソースファイル(例: r_fwupcomm_cmd_user_defined.c)とヘッダファイル(例: r_fwupcomm_cmd_user_defined.h)を作成します。

ヘッダファイルには r_fwupcomm_if.h をインクルードし、ソースファイルには作成した UserDefined コマンドのヘッダファイルをインクルードしてください。

(2) ヘッダファイルに、UserDefined コマンドを定義する列挙型(例: r_fwupcomm_cmd_class_user_defined_t)を作成し、ADDITIONAL1, ADDITIONAL2 コマンドを表す列挙子を定義します。列挙型の最後に、要素数を表す列挙子を定義します。

```
typedef enum
{
    FWUPCOMM_CMD_USERDEFINED_ADDITIONAL1,
    FWUPCOMM_CMD_USERDEFINED_ADDITIONAL2,
    FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS
} r_fwupcomm_cmd_class_user_defined_t;
```

(3) ソースファイルに、r_fwupcomm_cmd_table_t 型の配列を定義し、配列の各要素に ADDITIONAL1, ADDITIONAL2 コマンドの情報を定義します。

```
const r_fwupcomm_cmd_table_t
r_fwupcomm_user_defined_cmd_table[FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS] =
{
  { FWUPCOMM_CMD_USERDEFINED_ADDITIONAL1, 0x01, 0U, 0U },
  { FWUPCOMM_CMD_USERDEFINED_ADDITIONAL2, 0x02, 0U, 0U }
};
```

r_fwupcomm_cmd_table_t 型は r_fwupcomm_if.h に定義されており、各メンバの定義は以下の通りです。

```
typedef struct r_fwupcomm_cmd_table
{
    uint8_t type;                  // このコマンドを表す値(列挙子)
    uint8_t value;                 // このコマンドの通信で使われる実際の値
    uint16_t cmd_data_max_size;    // このコマンドの Command data の最大サイズ
    uint16_t resp_data_max_size;   // このコマンドの Response data の最大サイズ
} r_fwupcomm_cmd_table_t;
```

(4) ソースファイルに、セカンダリ MCU が UserDefined コマンドを受信した時に実行する処理を記述したハンドラ関数を定義します。
引数の r_fwupcomm_cmd_info_t 型のポインタ変数には、受信したコマンドの情報(Command argument や Command data へのポインタ等)が入っており、このコマンド情報を参照してこのハンドラ関数内で処理を実施し、同じく引数の r_fwupcomm_resp_info_t 型のポインタ変数に、プライマリ MCU に送信するレスポンスの情報(Command result、Response data へのポインタ、Response data サイズ)を格納します。

```
void R_FWUPCOMM_CmdHandler_UserDefined(r_fwupcomm_cmd_info_t *cmd,
                                       r_fwupcomm_resp_info_t *resp)
{
    if((NULL == cmd)||(NULL == resp))
    {
        return;
    }

    if(cmd->type >= FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS)
    {
        return;
    }

    switch(cmd->type)
    {
        case FWUPCOMM_CMD_USERDEFINED_ADDITIONAL1:
            /* ADDITIONAL1 コマンド受信時に実行する処理を記述 */
            break;
        case FWUPCOMM_CMD_USERDEFINED_ADDITIONAL2:
            /* ADDITIONAL2 コマンド受信時に実行する処理を記述 */
            break;
}
```

(5) 先ほどソースファイルで定義した UserDefined コマンドの r_fwupcomm_cmd_table_t 型の配列を、ヘッダファイルに extern 宣言します。また、同様に UserDefined コマンドのハンドラ関数をプロトタイプ宣言します。

```
extern const r_fwupcomm_cmd_table_t r_fwupcomm_user_defined_cmd_table
[FWUPCOMM_CMD_COMMON_NUM_COMMANDS];

#if FWUPCOMM_CFG_DEVICE_PRIMARY == (0)  // セカンダリ MCU のみ有効にするマクロ
void R_FWUPCOMM_CmdHandler_UserDefined (r_fwupcomm_cmd_info_t *cmd,
r_fwupcomm_resp_info_t *resp);
#endif
```

(6) r_fwupcomm¥src¥commands¥r_fwupcomm_cmd.h ファイルに、UserDefined コマンドのヘッダファイルをインクルードします。

```
#include "r_fwupcomm_cmd_common.h"
#include "r_fwupcomm_cmd_fwup.h"
#include " r_fwupcomm_cmd_user_defined.h"
```

(7) r_fwupcomm_cmd.h ファイルに定義されている r_fwupcomm_cmd_class_t 列挙型に、UserDefined コマンドを表す列挙子を追加します。

```
typedef enum
{
    FWUPCOMM_CMD_CLS_COMMON = 0,
    FWUPCOMM_CMD_CLS_FWUP,
    FWUPCOMM_CMD_CLS_USERDEFINED,
    FWUPCOMM_CMD_NUM_CLS
} r_fwupcomm_cmd_class_t;
```

(8) r_fwupcomm_cmd.c ファイルに定義されている r_fwupcomm_cmd_def_table_t 型の配列に、UserDefined コマンドを追加します。

```
const r_fwupcomm_cmd_def_table_t r_fwupcomm_cmd_def_table_list[] =
{
  [FWUPCOMM_CMD_CLS_COMMON] = {r_fwupcomm_common_cmd_table, FWUPCOMM_CMD_COMMON_NUM_COMMANDS},
  [FWUPCOMM_CMD_CLS_FWUP] = {r_fwupcomm_fwup_cmd_table, FWUPCOMM_CMD_FWUP_NUM_COMMANDS},
  [FWUPCOMM_CMD_CLS_USERDEFINED] = {r_fwupcomm_user_defined_cmd_table,
                                    FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS}
};
```

r_fwupcomm_cmd_def_table_t 型は r_fwupcomm_cmd.h に定義されており、table メンバには、ソースファイルで定義した r_fwupcomm_cmd_table_t 型の配列を、num_cmd メンバにはその Command class のコマンド数を指定します。

```
typedef struct
{
    const r_fwupcomm_cmd_table_t *table;
    uint8_t num_cmd;
} r_fwupcomm_cmd_def_table_t;
```

(9) r_fwupcomm_cmd.c ファイルに定義されている R_FWUPCOMM_CmdHandler_t 型の配列に、ソースファイルで定義した UserDefined コマンドのハンドラ関数を追加します。

```
#if FWUPCOMM_CFG_DEVICE_PRIMARY == (0)    // セカンダリ MCU のみ有効にするマクロ
const R_FWUPCOMM_CmdHandler_t r_fwupcomm_cmd_handler_list[FWUPCOMM_CMD_NUM_CLS]
=
{
     [FWUPCOMM_CMD_CLS_COMMON] = FWUPCOMM_CFG_CMD_HANDLER_COMMON,
     [FWUPCOMM_CMD_CLS_FWUP] = FWUPCOMM_CFG_CMD_HANDLER_FWUP,
     [FWUPCOMM_CMD_CLS_USERDEFINED] = R_FWUPCOMM_CmdHandler_UserDefined
};
#endif
```

コマンド追加手順は以上です。r_fwupcomm¥src¥commands フォルダ内に FWUP コマンド用定義ファイル(r_fwupcomm_cmd_fwup.c, r_fwupcomm_cmd_fwup.h)と Common コマンド用定義ファイル(r_fwupcomm_cmd_common.c, r_fwupcomm_cmd_common.h)がありますので、参考にしてください。

4.2 通信方式の変更

本モジュールは、SCI を用いた UART 通信にのみ対応しています。ここでは、他の通信方式に変更する方法を説明します。

4.2.1 通信インターフェース

本モジュールではパケット通信を行う際の通信インターフェースを規定しており、r_fwupcomm¥src¥connectivity¥r_fwupcomm_ch.h で以下のように定義されています。

```
typedef struct r_fwupcomm_ch_api
{
    fwupcomm_err_t (*open)(void);
    void (*close)(void);
    fwupcomm_err_t (*send)(uint8_t *src, uint16_t size);
    fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size);
    void (*rx_flush)(void);
} r_fwupcomm_ch_api_t;
```

4.2.1.1 fwupcomm_err_t (*open)(void)

表 4-1 open 関数仕様

Format	fwupcomm_err_t (*open)(void)	
Description	通信チャネルをオープンします。	
Parameters	—	
Return Values	FWUPCOMM_SUCCESS	正常に初期化されました。
	FWUPCOMM_ERR_CH_ALREADY_OPEN	通信チャネルが既にオープン済です。
	FWUPCOMM_ERR_NOT_OPEN	通信チャネルの初期化に失敗しました。
Special Notes	—	

4.2.1.2 void (*close)(void)

表 4-2 close 関数仕様

Format	void (*close)(void)
Description	通信チャネルをクローズします。
Parameters	—
Return Values	—
Special Notes	—

4.2.1.3 fwupcomm_err_t (*send)(uint8_t *src, uint16_t size)

表 4-3 send 関数仕様

Format	fwupcomm_err_t (*send)(uint8_t *src, uint16_t size)	
Description	通信チャネル使用してデータを送信します。	
Parameters	src：送信データの格納先へのポインタ size：送信データサイズ	
Return Values	FWUPCOMM_SUCCESS	正常に初期化されました。
	FWUPCOMM_ERR_INVALID_PTR	src ポインタが NULL です。
	FWUPCOMM_ERR_INVALID_ARG	size が 0 です。
	FWUPCOMM_ERR_NOT_OPEN	通信チャネルがオープンされていません。
	FWUPCOMM_ERR_CH_SEND_BUSY	通信チャネルがビジー状態のためデータ送信に失敗しました。
	FWUPCOMM_ERR_CH_SEND	通信チャネルがデータ送信に失敗しました。
Special Notes	—	

4.2.1.4 fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size)

表 4-4 recv 関数仕様

Format	fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size)	
Description	通信チャネル使用してデータを受信します。	
Parameters	dest：受信データを格納するバッファへのポインタ size：必要受信データサイズ	
Return Values	FWUPCOMM_SUCCESS	正常に初期化されました。
	FWUPCOMM_ERR_INVALID_PTR	dest ポインタが NULL です。
	FWUPCOMM_ERR_INVALID_ARG	size が 0 です。
	FWUPCOMM_ERR_NOT_OPEN	通信チャネルがオープンされていません。
	FWUPCOMM_ERR_CH_RECV_NO_DATA	通信チャネルに十分な受信データがありません。
Special Notes	—	

4.2.1.5 void (*rx_flush)(void)

表 4-5 rx_flush 関数仕様

Format	void (*rx_flush)(void)
Description	通信チャネルの受信バッファを空にします。
Parameters	—
Return Values	—
Special Notes	—

4.2.2　通信方式の変更方法

(1)　変更したい通信方式を使って、4.2.1 の通信インターフェースの関数を実装します。

(2)　「const r_fwupcomm_ch_api_t」型の r_fwupcomm_ch_api 変数を定義し、作成した通信インターフェースの関数で以下のように初期化します。

```
const r_fwupcomm_ch_api_t r_fwupcomm_ch_api =
{ .open = r_fwupcomm_rx_sci_uart_open,          // open
  .close = r_fwupcomm_rx_sci_uart_close,        // close
  .send = r_fwupcomm_rx_sci_uart_send,          // send
  .recv = r_fwupcomm_rx_sci_uart_recv,          // recv
  .rx_flush = r_fwupcomm_rx_sci_uart_rx_flush   // rx_flush
};
```

(3)　ヘッダファイル(例: r_fwupcomm_ch_user_defined.h)を作成し、r_fwupcomm_ch_api 変数を extern 宣言します。

```
extern r_fwupcomm_ch_api_t const r_fwupcomm_ch_api;
```

(4)　r_fwupcomm¥src¥r_fwupcomm_private.h ファイルに通信インターフェースの定義を追加し、作成したヘッダファイルが代わりにインクルードされるようにします。

```
#define FWUPCOMM_CH_RX_SCI_UART            (1)
#define FWUPCOMM_CH_USERDEFINED            (2)

#define FWUPCOMM_USE_CH                    (FWUPCOMM_CH_USERDEFINED)

#if (FWUPCOMM_USE_CH == FWUPCOMM_CH_RX_SCI_UART)
  #include "r_fwupcomm_rx_sci_uart.h"
#elif (FWUPCOMM_USE_CH == FWUPCOMM_CH_USERDEFINED)
  #include "r_fwupcomm_ch_user_defined.h"
#endif
```

通信方式の変更方法は以上です。

5. デモプロジェクト

本デモプロジェクトは、以下の構成図のように、PC と接続されたプライマリ MCU がシリアル通信によってセカンダリ MCU 用の更新ファームウェアを受け取り、FWUP Comm.モジュールを用いてセカンダリ MCU に更新ファームウェアを転送し、セカンダリ MCU のファームウェアアップデートを実施するためのサンプルプログラムです。

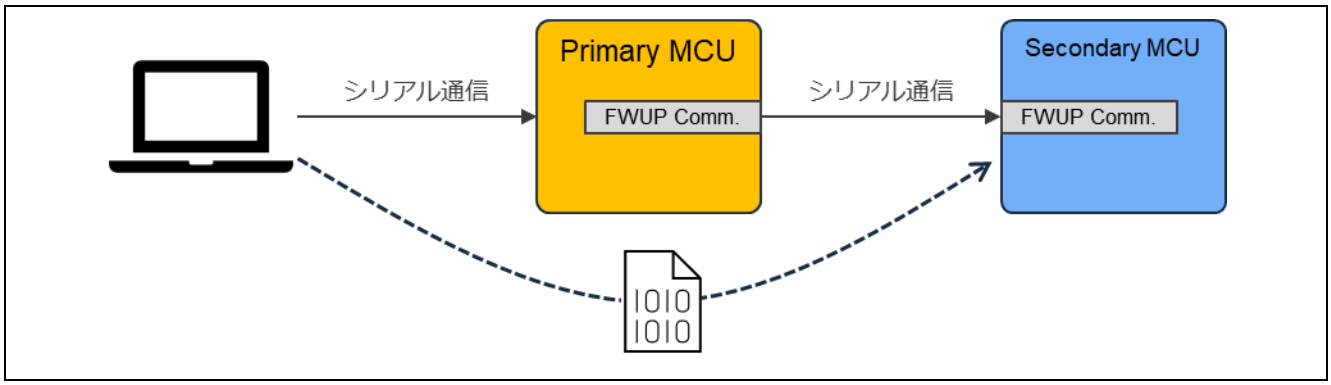

図 5-1 デモの構成図

5.1 デモプロジェクトの構成

5.1.1 プライマリ MCU

プライマリ MCU 側のデモプロジェクトは RX65N のみです。

- FreeRTOS 環境：FITDemos¥ck-rx65n¥app_rx65n_ck_primary_frtos
- ベアメタル環境：FITDemos¥ck-rx65n¥app_rx65n_ck_primary

5.1.2 セカンダリ MCU

セカンダリ MCU 側のデモプロジェクトは以下のようにデバイス毎にフォルダ分けされています。

- リニアモードの半面更新方式: FITDemos¥(デバイス名)¥w_buffer¥(コンパイラ名)¥(プロジェクト名)
- リニアモードの全面更新方式: FITDemos¥(デバイス名)¥wo_buffer¥(コンパイラ名)¥(プロジェクト名)

ブートローダプロジェクト:

- リニアモードの半面更新方式: bootloader_(デバイス名)_w_buffer
- リニアモードの全面更新方式: bootloader_(デバイス名)_wo_buffer

アプリケーションプロジェクト:

- リニアモードの半面更新方式: app_(デバイス名)_w_buffer
- リニアモードの全面更新方式: app_(デバイス名)_wo_buffer

5.2 動作環境準備

セカンダリ MCU のファームウェアアップデートにはファームウェアアップデートモジュールを使用します。デモプロジェクトの実行には Windows PC にツールをインストールする必要があります。

5.2.1 TeraTerm のインストール

Windows PC からプライマリ MCU へのシリアル通信により、更新ファームウェアのイメージを転送するために使用します。デモプロジェクトでは、TeraTerm 4.106 で動作確認を実施しています。

インストール後は、シリアルポートの通信設定を表 5-1 の様に設定してください。

表 5-1 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	RTS/CTS

5.2.2 Python 実行環境のインストール

Renesas Image Generator (image-gen.py)で初期イメージと更新イメージを作成するために使用します

Renesas Image Generator は ECDSA により署名データを生成します。デモプロジェクトでは、Python 3.10.4 で動作確認を実施しています。

また、Python の暗号化ライブラリ(pycryptodome)を使用しますので、Python をインストール後、コマンドプロンプトから以下の pip コマンドを実行し、ライブラリのインストールを行ってください。

```
pip install pycryptodome
```

5.2.3 フラッシュライタのインストール

初期イメージを書き込むためのフラッシュライタが必要です。

デモプロジェクトでは、Renesas Flash Programmer V3.18.00 を使用しています。

Renesas Flash Programmer (Programming GUI) | Renesas ルネサス

5.3 プロジェクトの実行手順

本章では、RX140 を例にデモプロジェクトの実行手順を記載しています。他の MCU 製品においてもデモプロジェクトの実行手順は共通ですが、動作確認環境のみ MCU 毎に異なりますので、該当する MCU 製品の「6.1 動作確認環境」を確認してください。また、CC-RX コンパイラと GCC コンパイラにおいてもデモプロジェクトの実行手順は共通です。

5.3.1 実行環境

RX140 の動作確認環境(6.3.1)を準備します。RX140 以外の MCU 製品の場合は、該当する MCU 製品の動作確認環境を参照してください。

5.3.2 デモプロジェクトの構築

プライマリ MCU 用のプロジェクト、セカンダリ MCU 用のプロジェクトを構築します。

5.3.2.1 プライマリ MCU

(1) e² studio に app_rx65n_ck_primary プロジェクトをインポートし、ビルドします。全面更新方式の場合、ビルド前に app_rx65n_ck_primary¥src¥app_rx65n_ck_primary.h の「FWUP_FULL_UPDATE」マクロ定義を(1)に変更してください。

```
49
50  #define FWUP_FULL_UPDATE          (1)
51
```

(2) プロジェクトの HardwareDebug フォルダ内に、以下の MOT ファイルが生成されていることを確認します。

- app_rx65n_ck_primary.mot

5.3.2.2 セカンダリ MCU 用の初期イメージと更新イメージを作成

初期イメージ名を initial_firm.mot、更新イメージ名を update_firm.mot として、初期イメージと更新イメージの作成手順を説明します。半面更新方式の手順を記載しますが、全面更新方式も手順は共通ですので、使用するプロジェクトを全面更新方式のものに置き換えてください。

(1) e² studio に bootloader_rx140_fpb_w_buffer, app_rx140_fpb_w_buffer プロジェクトをインポートし、ビルドします。

(2) 各プロジェクトの HardwareDebug フォルダ内に、以下の MOT ファイルが生成されていることを確認します。

- bootloader_rx140_fpb_w_buffer.mot
- app_rx140_fpb_w_buffer.mot

(3) bootloader_rx140_fpb_w_buffer¥src¥smc_gen¥r_fwup¥tool フォルダにビルドしたデモプロジェクトの MOT ファイルを格納します。また、同様に FITDemos¥keys¥secp256r1.privatekey ファイルを格納します。

```
image-gen.py
RX140_Linear_Full_ImageGenerator_PRM.csv
RX140_Linear_Half_ImageGenerator_PRM.csv
secp256r1.privatekey
bootloader_rx140_fpb_w_buffer.mot
app_rx140_fpb_w_buffer.mot
```

(4) bootloader_rx140_fpb_w_buffer¥src¥smc_gen¥r_fwup¥tool フォルダで以下のコマンドを実行し、初期イメージを作成します。全面更新方式の場合は、RX140_Linear_Half_ImageGenerator_PRM.csv ではなく RX140_Linear_Full_ImageGenerator_PRM.csv を使用します。

```
python .¥image-gen.py -iup ".¥app_rx140_fpb_w_buffer.mot" -
ip .¥RX140_Linear_Half_ImageGenerator_PRM.csv -o initial_firm -ibp
".¥bootloader_rx140_fpb_w_buffer.mot" -vt ecdsa -key ".¥secp256r1.privatekey"
```

(5) app_rx140_fpb_w_buffer¥src¥fwupcomm_demo_main.h ファイルを開き、DEMO_VER_MAJOR の定義を(1)から(2)に変更し、再度 app_rx140_fpb_w_buffer プロジェクトをビルドします。ビルドしたプロジェクトの MOT ファイルを同様に tool フォルダに格納します。

```
51  #define DEMO_VER_MAJOR      (2)
52  #define DEMO_VER_MINOR      (0)
53  #define DEMO_VER_BUILD      (0)
```

(6) 以下のコマンドを実行し、更新イメージを作成します。全面更新方式の場合は、RX140_Linear_Half_ImageGenerator_PRM.csv ではなく RX140_Linear_Full_ImageGenerator_PRM.csv を使用します。

```
python .¥image-gen.py -iup ".¥app_rx140_fpb_w_buffer.mot" -
ip .¥RX140_Linear_Half_ImageGenerator_PRM.csv -o update_firm -vt ecdsa -key
".¥secp256r1.privatekey"
```

tool フォルダに初期イメージと更新イメージが生成されていることを確認してください。

Image-gen.py
RX140_Linear_Full_ImageGenerator_PRM.csv
RX140_Linear_Half_ImageGenerator_PRM.csv
secp256r1.privatekey
bootloader_rx140_fpb_w_buffer.mot
app_rx140_fpb_w_buffer.mot
initial_firm.mot
update_firm.rsu

5.3.3 初期イメージの書き込み

app_rx65n_ck_primary.mot をフラッシュライタで CK-RX65Nv2 ボードに書き込みます。

同様に、初期イメージ(initial_firm.mot)をフラッシュライタで FPB-RX140 ボードに書き込みます。書き込み後はボードの電源を OFF にし、デバッガ(E2 Lite)の接続を外しておいてください。

5.3.4　ファームウェアアップデートの実行

初期イメージが起動するとプライマリ MCU 経由で更新イメージの転送を待ちます。受信した更新イメージをフラッシュに書き込み、転送完了後に署名検証を経て更新イメージのファームウェアを起動します。

以下の手順により、ファームウェアアップデートを実施してください。

(1) PC で TeraTerm を 2 画面起動し、プライマリ MCU(CK-RX65Nv2)とセカンダリ MCU(FPB-RX140)のシリアル COM ポートを選択し接続設定を行います。

(2) ボードの電源を投入します。TeraTerm に以下のメッセージが出力されます。

プライマリ MCU 側

```
==== RX65N : FWUPCOMM DEMO [Primary] ====
Send image(*.rsu) via UART.
```

セカンダリ MCU 側

```
==== RX140 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX140 : FWUPCOMM DEMO [Secondary][with buffer]  ver. 1.0.0 ====
```

(3) TeraTerm から更新イメージを送信します。

プライマリ MCU 側の TeraTerm の[ファイル]メニューから[ファイル送信]をクリックします。update_firm.rsu を選択し、オプションの「バイナリ」にチェックを入れ、[Open]をクリックします。

更新イメージの転送中は以下のメッセージが出力され、インストールと署名検証が終了するとソフトウェアリセットし、更新イメージのファームウェアが実行されます。

セカンダリ MCU 側のメッセージに出力されるバージョンがインクリメントされていれば成功です。

プライマリ MCU 側

```
Send FWUP_START command... OK.
Send FWUP_WRITE command... OK. (1024 bytes sent, remaining 38912 bytes.)
Send FWUP_WRITE command... OK. (1024 bytes sent, remaining 37888 bytes.)
...
Send FWUP_WRITE command... OK. (1024 bytes sent, remaining 2048 bytes.)
Send FWUP_WRITE command... OK. (1024 bytes sent, remaining 1024 bytes.)
Send FWUP_INSTALL command... OK.
Firmware update for the device(0xA0) is successful.
```

セカンダリ MCU 側

```
Received FWUPCOMM_CMD_FWUP_START command.
Received FWUPCOMM_CMD_FWUP_WRITE command. size=1024
W 0xFFF78000, 512 ... OK
W 0xFFF78200, 256 ... OK
W 0xFFF78300, 256 ... OK
Received FWUPCOMM_CMD_FWUP_WRITE command. size=1024
W 0xFFF78400, 1024 ... OK
...
Received FWUPCOMM_CMD_FWUP_WRITE command. size=1024
W 0xFFF81400, 1024 ... OK
Received FWUPCOMM_CMD_FWUP_WRITE command. size=1024
W 0xFFF81800, 768 ... OK
W 0xFFFEFF00, 256 ... OK
verify install area buffer [sig-sha256-ecdsa]...OK
Received FWUPCOMM_CMD_FWUP_INSTALL command.
software reset...

==== RX140 : BootLoader [with buffer] ====
verify install area buffer [sig-sha256-ecdsa]...OK
activating image ... OK
software reset...
==== RX140 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ...
==== RX140 : FWUPCOMM DEMO [Secondary][with buffer]  ver. 2.0.0 ====
```

6. 付録

6.1 動作確認環境

本モジュールの動作確認環境を以下に示します。

表 6-1 動作確認環境(CC-RX)

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 2025-04
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 V3.07.00 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
エンディアン	リトルエンディアン
モジュールリビジョン	Rev.1.00
使用ボード	Fast Prototyping Board for RX140 MCU Group（製品型名：RTK5FP1400S00001BE） Renesas Solution Starter Kit for RX23E-B（製品型名：RTK0ES1001C00001BJ） Fast Prototyping Board for RX261 MCU Group（製品型名：RTK5FP2610S00001BE） Target Board for RX660（製品型名：(RTK5RX6600C00000BJ） Renesas Starter Kit for RX66T（製品型名：RTK50566T0S00000BE） CK-RX65N v2 cloud kit（製品型名：RTK5CK65N0S08001BE）
USB シリアル変換ボード	Pmod USBUART（DIGILENT 製） https://digilent.com/reference/pmod/pmodusbuart/start

表 6-2 動作確認環境(GCC)

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 2025-04
C コンパイラ	GCC for Renesas RX 8.3.0.202411 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
モジュールリビジョン	Rev.1.00
使用ボード	Fast Prototyping Board for RX140 MCU Group（製品型名：RTK5FP1400S00001BE） Renesas Solution Starter Kit for RX23E-B（製品型名：RTK0ES1001C00001BJ） Fast Prototyping Board for RX261 MCU Group（製品型名：RTK5FP2610S00001BE） Target Board for RX660（製品型名：(RTK5RX6600C00000BJ） Renesas Starter Kit for RX66T（製品型名：RTK50566T0S00000BE） CK-RX65N v2 cloud kit（製品型名：RTK5CK65N0S08001BE）
USB シリアル変換ボード	Pmod USBUART（DIGILENT 製） https://digilent.com/reference/pmod/pmodusbuart/start

6.2 UART 通信設定

本モジュールの UART 通信設定を以下に示します。

表 6-3 UART 通信設定

項目	内容
Data Length	8-bit
Parity	None
Stop Bits	1-bit
Flow Control	None
Bitrate	1Mbps

6.3　デモプロジェクトの動作環境

本デモプロジェクトのデバイス毎の接続構成を以下に示します。

なお、図中の評価ボードの PMOD 端子と USB シリアル変換ボードは、PMOD 端子の Pin1~6 と USB シリアル変換ボード(Pmod USBUART)の Pin1~6 を接続します。

6.3.1　RX140 の動作確認環境

接続構成を以下に示します。

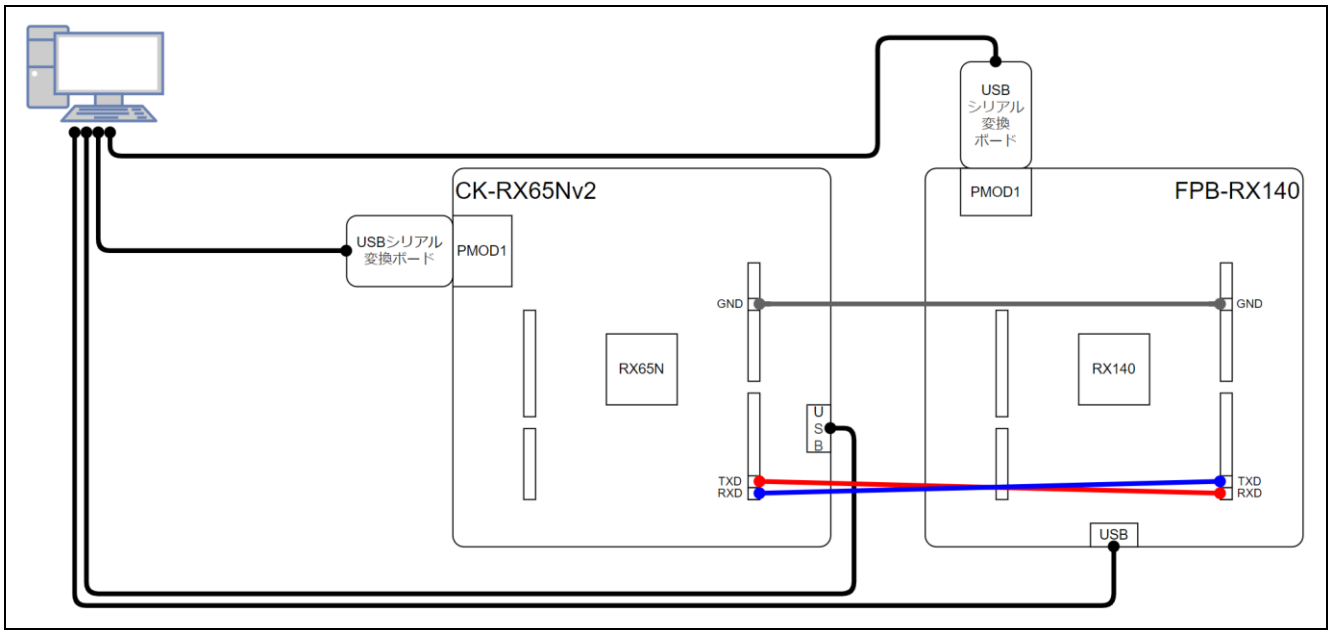

図 6-1 FPB-RX140 接続構成図

表 6-4 CK-RX65Nv2 - FPB-RX140 間 UART 通信の接続端子対応

CK-RX65Nv2		FPB-RX140
J24 Pin7: GND	⇔	J10 Pin7
J23 Pin2: D1/TX	⇔	J12 Pin1 D0/RX
J23 Pin1: D0/RX	⇔	J12 Pin2 D1/TX

6.3.2　RX23E-B の動作確認環境

接続構成を以下に示します。

RSSK-RX23E-B には USB シリアル変換ボードから電源を供給するため、以下の設定をしてください。

- RSSK-RX23E-B のジャンパ JP1 の「1-2 番ピン」を接続
- RSSK-RX23E-B のジャンパ JP3 の「1-2 番ピン」を接続
- USB シリアル変換ボード(Pmod USBUART)のジャンパ JP1 の「VCC-SYS ピン」を接続

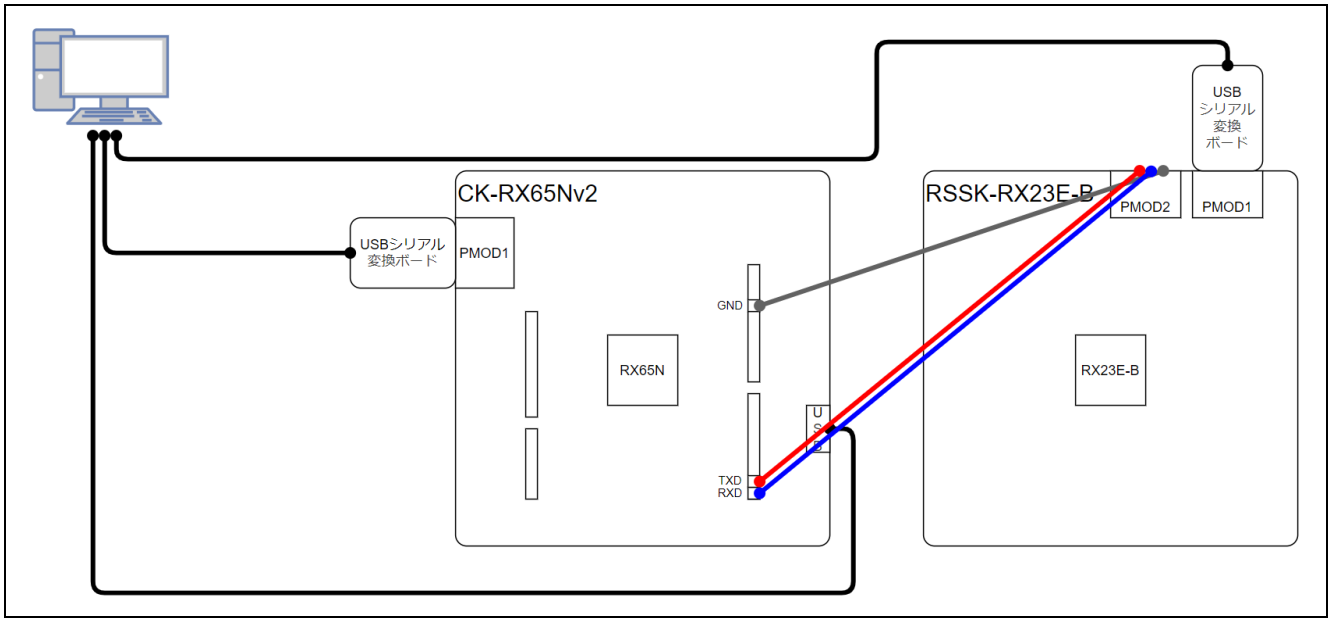

図 6-2 RSSK-RX23E-B 接続構成図

表 6-5 CK-RX65Nv2 – RSSK-RX23E-B 間 UART 通信の接続端子対応

CK-RX65Nv2		RSSK-RX23E-B
J24 Pin7: GND	⇔	PMOD2 Pin5
J23 Pin2: D1/TX	⇔	PMOD2 Pin3
J23 Pin1: D0/RX	⇔	PMOD2 Pin4

6.3.3　RX261 の動作確認環境

接続構成を以下に示します。

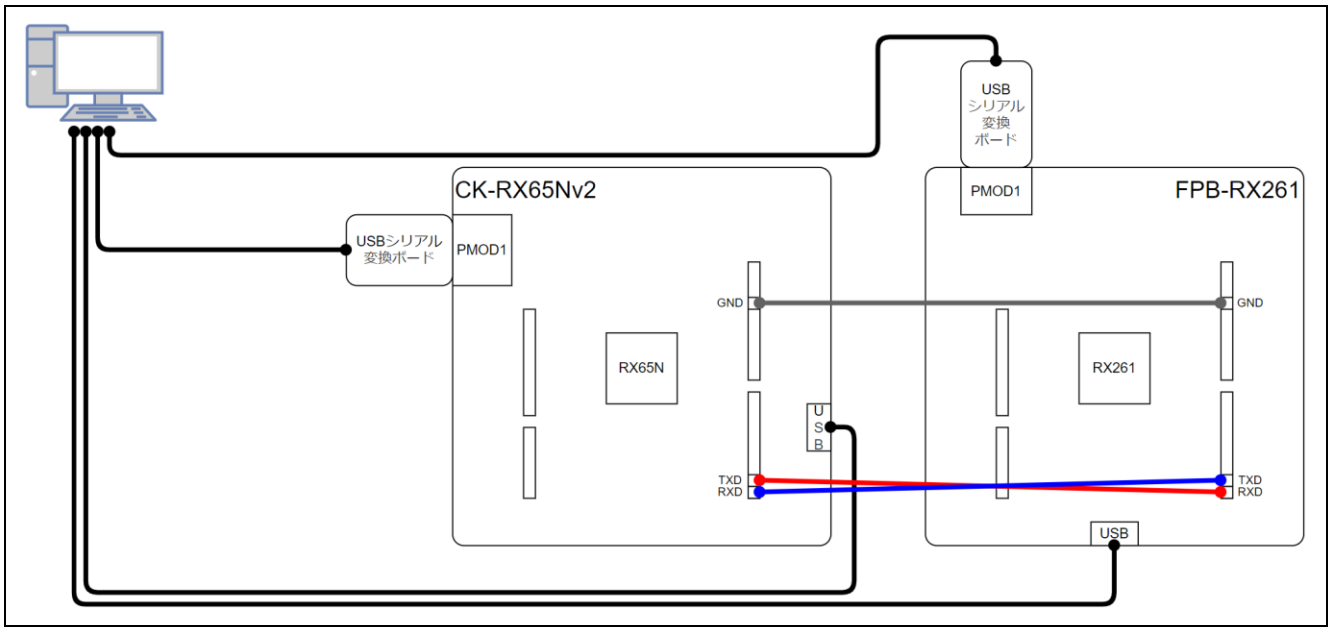

図 6-3 FPB-RX261 接続構成図

表 6-6 CK-RX65Nv2 - FPB-RX261 間 UART 通信の接続端子対応

CK-RX65Nv2		FPB-RX261
J24 Pin7: GND	⇔	J10 Pin7
J23 Pin2: D1/TX	⇔	J12 Pin1 D0/RX
J23 Pin1: D0/RX	⇔	J12 Pin2 D1/TX

6.3.4　RX66T の動作確認環境

接続構成を以下に示します。

RSK-RX66T には DC 電源コネクタ(5V)から電源を供給するため、以下の設定をしてください。

- RSK-RX66T のジャンパ J7 を短絡

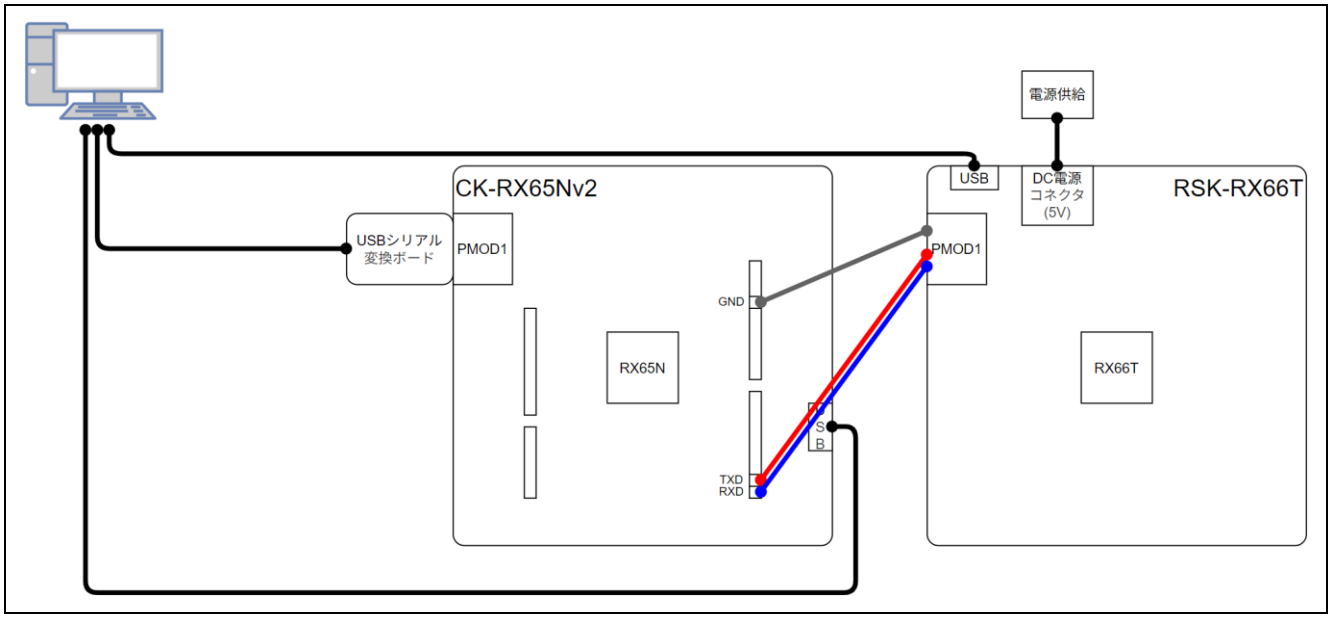

図 6-4 RSK-RX66T 接続構成図

表 6-7 CK-RX65Nv2 – RSK-RX66T 間 UART 通信の接続端子対応

CK-RX65Nv2		RSK-RX66T
J24 Pin7: GND	⇔	PMOD1 Pin5
J23 Pin2: D1/TX	⇔	PMOD1 Pin3
J23 Pin1: D0/RX	⇔	PMOD1 Pin2

6.3.5　RX660 の動作確認環境

接続構成を以下に示します。

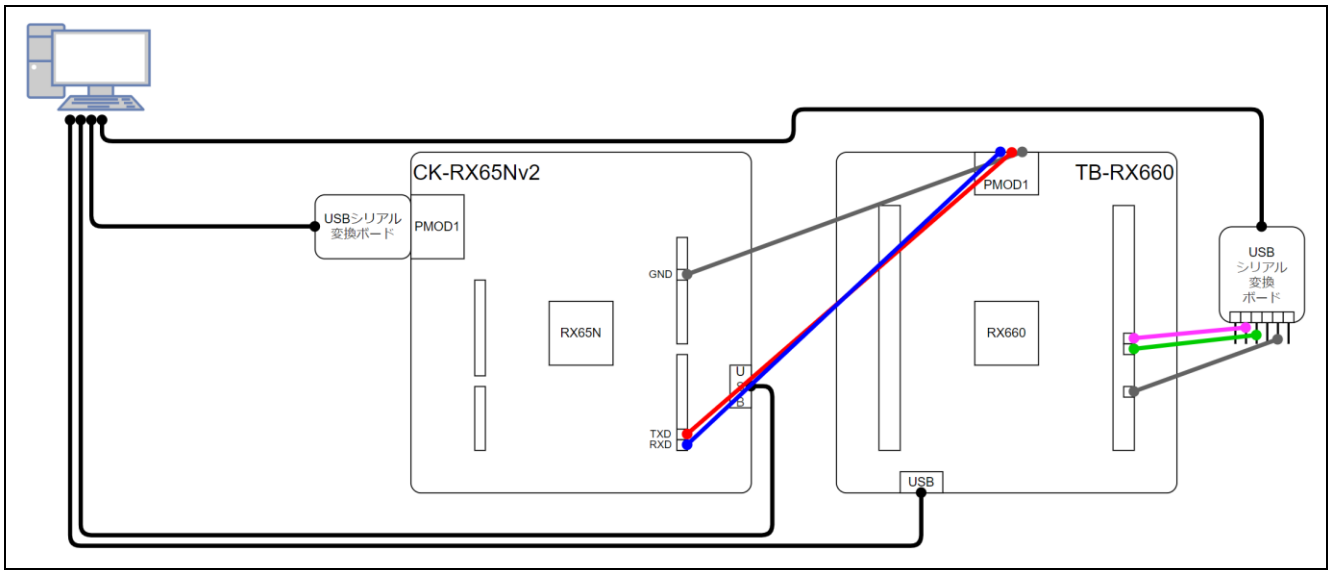

図 6-5 TB-RX660 接続構成図

表 6-8 CK-RX65Nv2 – TB-RX660 間 UART 通信の接続端子対応

CK-RX65Nv2		TB-RX660
J24 Pin7: GND	⇔	PMOD1 Pin11
J23 Pin2: D1/TX	⇔	PMOD1 Pin10
J23 Pin1: D0/RX	⇔	PMOD1 Pin9

表 6-9 TB-RX660 – USB シリアル変換ボード(Pmod USBUART)間 UART 通信の接続端子対応

TB-RX660		PmodUSBUART
MCU Header CN2 Pin22	⇔	Pin2
MCU Header CN2 Pin20	⇔	Pin3
MCU Header CN2 Pin12	⇔	Pin5

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2025.5.20	-	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、ノイズ輻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

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。

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/

© 2025 Renesas Electronics Corporation. All rights reserved.

© 2025 Renesas Electronics Corporation. All rights reserved.