RX ファミリ
ファームウェアアップデートモジュールを使用した差分ファームウェア更新

要旨

本アプリケーションノートは、RX ファミリ向けのファームウェアアップデートモジュールを使用した差分ファームウェア更新の手法について説明します。現行ファームウェアとの差分のみを転送することで、更新イメージサイズ、データ転送量、および更新時間を削減し、効率的なファームウェア更新を実現します。

本アプリケーションノートでは、差分ファームウェア更新の仕組み、処理手順、および RX65N を用いたデモプロジェクトの動作確認手順を示します。

動作確認デバイス

RX65N 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

- 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)
- RX ファミリ デュアルモードの使用ガイド (R01AN6894)
- RX65N Group CK-RX65N v2 User's Manual (R20UT5366)

（最新版をルネサス エレクトロニクスホームページから入手してください。）

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 「3 動作確認環境」を参照してください。

目次

1. 概要

1.1 差分ファームウェア更新とは

差分ファームウェア更新とは、ファームウェア更新時において、現行ファームウェアとの差分が存在するアドレス領域のデータのみを更新イメージに含める方式です。

この方式を採用することで、更新イメージのサイズを削減することができ、ファームウェア更新時間の短縮やデータ転送量の削減が可能です。これにより、通信帯域や書き込み時間に制約のある組込みシステムにおいて、効率的なファームウェア更新を実現できます。

1.2 差分ファームウェア更新の仕組み

本アプリケーションノートが実現する差分ファームウェア更新の仕組みについて説明します。

ルネサスは、RX マイコン向けにファームウェア更新機能を提供する「ファームウェアアップデートモジュール」を提供しています。

ファームウェアアップデートモジュールは、更新対象 MCU のコードフラッシュ上にあらかじめ確保されたバッファ領域へ更新ファームウェアを書き込み、その正当性を検証した後に新しいファームウェアを有効化する仕組みを提供します。差分ファームウェア更新も、このファームウェア更新機構（デュアルバンク方式）の枠組みの中で動作します。

ファームウェアアップデートモジュールが生成する更新イメージは、ブランク領域を除き、対象となるファームウェアのすべてのデータを含んでいます。一方、差分ファームウェア更新では、現行ファームウェアと更新ファームウェアとの差分を比較し、変更があった部分のみを含む差分更新イメージを生成します。

差分更新イメージを受信した更新対象 MCU は、ファームウェアアップデートモジュールが提供する API 関数を用いて、差分データを適切なアドレスに書き込みます。さらに、差分が存在しない領域については、現行ファームウェアのデータをコピーすることで補完し、最終的な更新ファームウェアを構築します。

1.3 差分ファームウェア更新イメージの作成方法

差分ファームウェア更新イメージ（以降、差分イメージ）の作成方法について説明します。

差分イメージの作成手順は以下の通りです。

(1) Step 1: 現行ファームウェアと更新ファームウェアを準備する。

(2) Step 2: 現行ファームウェアと更新ファームウェアの差分を抽出する。

(3) Step 3: 抽出した差分データと更新ファームウェアを、Renesas Image Generator を用いてそれぞれ更新イメージファイル（RSU 形式）に変換する。

(4) Step 4: 差分データと更新ファームウェアの更新イメージファイルをマージし、差分イメージを作成する。

それぞれの Step の処理の詳細を以降で説明します。

1.3.1 Step 1: 現行ファームウェアと更新ファームウェアの準備

現行ファームウェアと更新ファームウェアを準備します。

現行ファームウェアは、更新対象 MCU でファームウェア更新開始時に動作しているファームウェアです。Renesas Image Generator を用いて作成した初期イメージファイルではなく、ビルド直後の MOT ファイルを使用します。更新ファームウェアも同様に、ビルド直後の MOT ファイルを使用します。

1.3.2 Step 2: 差分データの抽出

更新ファームウェアと現行ファームウェアの差分を抽出します。

まず、現行ファームウェアと更新ファームウェアを、ファームウェア更新対象 MCU のコードフラッシュのプログラム単位でブロック分割し、両ファームウェアの差分をブロック単位で比較します（下図左）。例えばコードフラッシュのプログラム単位が 128 バイトの MCU の場合は、128 バイトのブロック単位で現行ファームウェアと更新ファームウェアの差分を比較します。

次に、ブロック内に差分が含まれる場合はそのブロック全体を差分ブロックとして抽出します。（下図中）

最後に、抽出した差分ブロックを結合し、最終的な差分データとします。

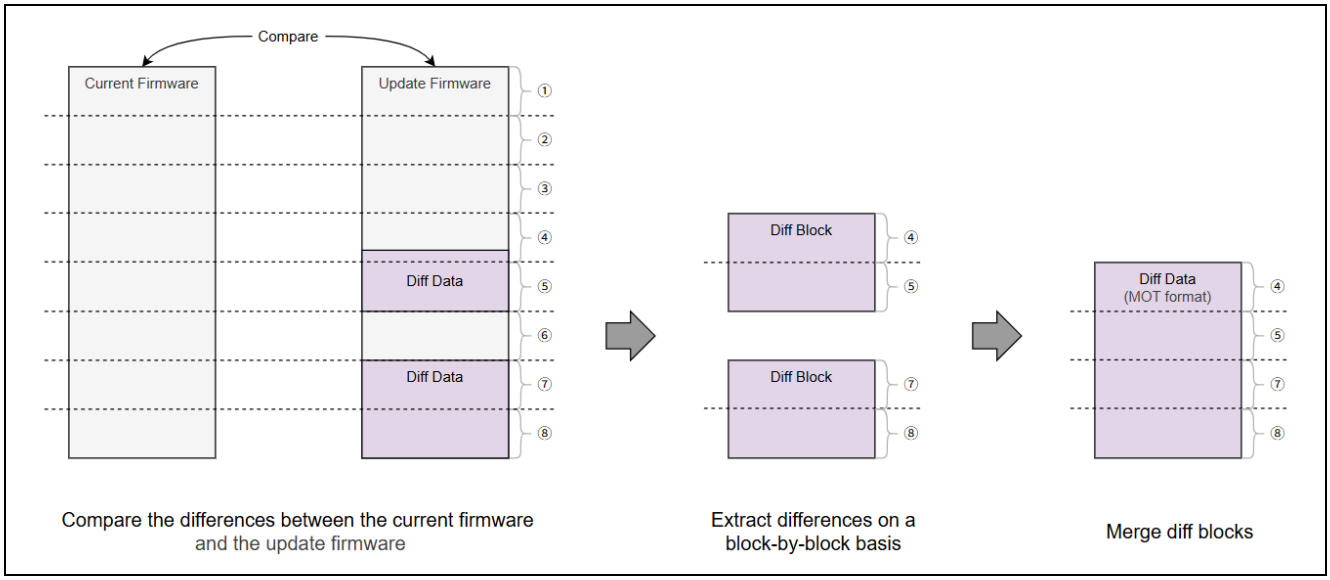


図 1-1 差分データの抽出手順

1.3.3 Step 3: 更新イメージファイル（RSU 形式）への変換

更新ファームウェアと Step 2 で作成した差分データを、それぞれ Renesas Image Generator に入力して更新イメージファイル（RSU 形式）を生成します。

Renesas Image Generator の更新イメージファイル生成処理では、ファームウェア情報の MOT 形式からバイナリデータへの変換と、RSU ヘッダ（署名情報、アドレス情報）を付与します。

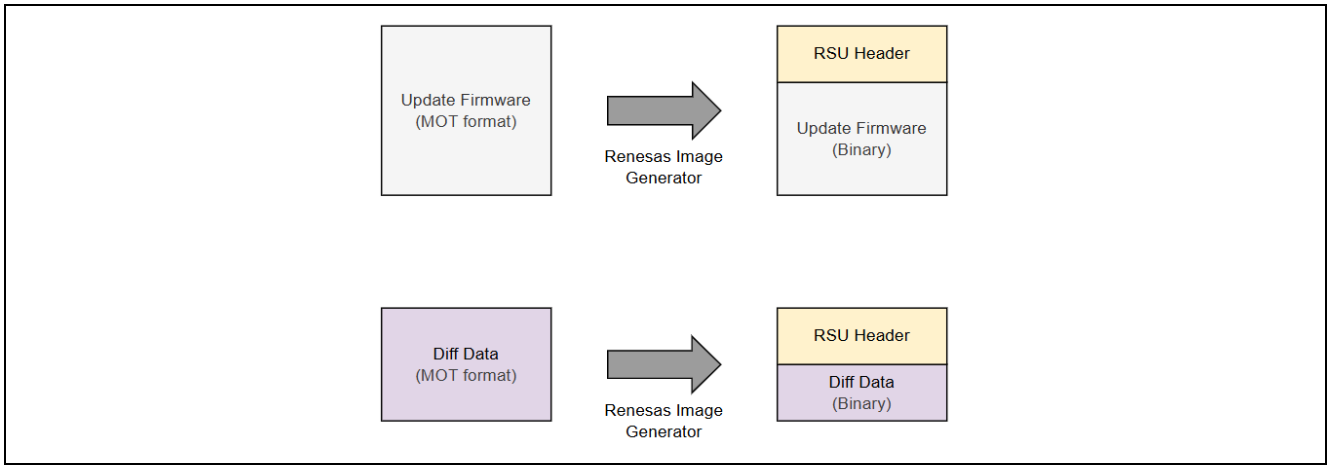


図 1-2 更新イメージファイルへの変換

1.3.4 Step 4: 差分イメージファイルの作成

Step 3 で生成した差分データと更新ファームウェアの更新イメージファイルをマージし、差分イメージを作成します。

差分イメージは、データの先頭から順に次の 4 つの領域で構成されます。

- RSU ヘッダ（署名情報）
- RSU ヘッダ（差分データのアドレス情報）
- 差分データ
- RSU ヘッダ（更新ファームウェアのアドレス情報）

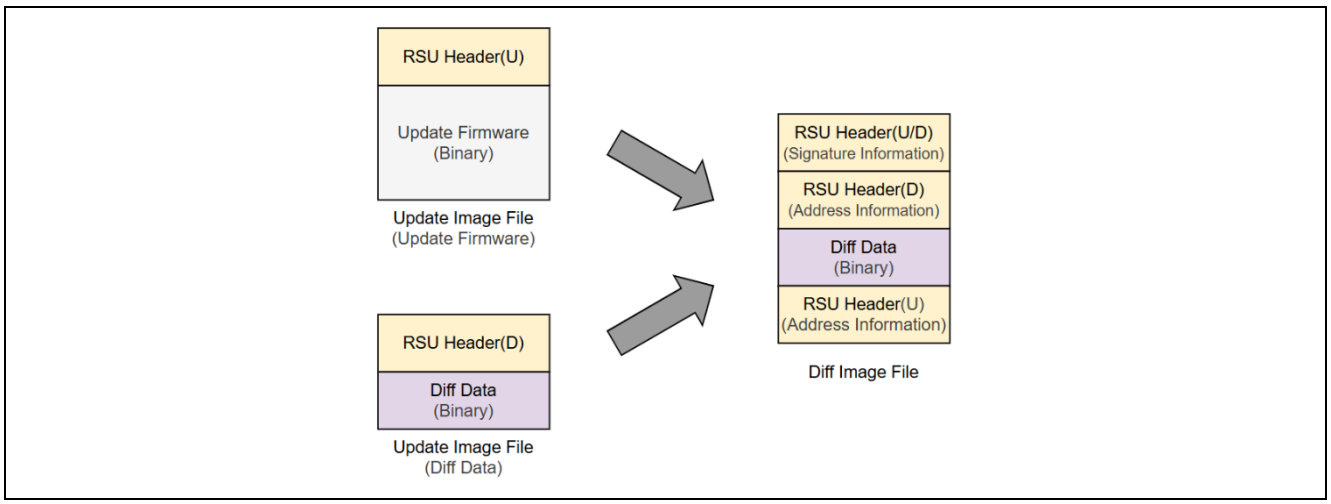


図 1-3 差分イメージファイルの作成

(1) RSU ヘッダ（署名情報） — 768 bytes

差分イメージファイルの RSU ヘッダの各項目は、更新ファームウェアの更新イメージファイルの RSU ヘッダ（上図の RSU Header(U)）と、差分データの更新イメージファイルの RSU ヘッダ（上図の RSU Header(D)）のどちらかを以下の表 1-1 の通りに使用し構築します。

表 1-1 差分イメージファイルの RSU ヘッダの構築方法

オフセット	項目	長さ (byte)	使用	理由
0x00000000	Magic Code	7	-	-
0x00000007	Reserved	1	-	-
0x00000008	Firmware Verification Type	32	更新ファームウェア	コード署名検証は更新イメージの全領域を対象に実行するため
0x00000028	Signature size	4	更新ファームウェア	同上
0x0000002C	Signature	64	更新ファームウェア	同上
0x0000006C	RSU File Size	4	差分データ	RSU ファイルサイズはファームウェア更新時に入力される更新イメージファイルのサイズを指定するため
0x00000070	Reserved	400	-	-

(2) RSU ヘッダ（差分データのアドレス情報） — 256 bytes

ファームウェア更新時に、バイナリ形式の差分データをコードフラッシュ上の正しいアドレスに配置するために、差分データのアドレス情報を使用します。

(3) 差分データ

差分イメージファイルの差分データは、Step 3 で生成した差分データの更新イメージファイルのバイナリデータを使用します。

(4) RSU ヘッダ（更新ファームウェアのアドレス情報） — 256 bytes

更新ファームウェアの更新イメージファイルの RSU ヘッダ（RSU Header(U)）のアドレス情報を使用します。これは最終的な署名検証対象となる更新イメージの領域に、RSU ヘッダのアドレス情報が含まれるためです。

1.4 MCU での差分ファームウェア更新処理

1.3 で作成した差分イメージを用いて差分ファームウェア更新を実行する処理手順を図 1-4、図 1-5 に示します。

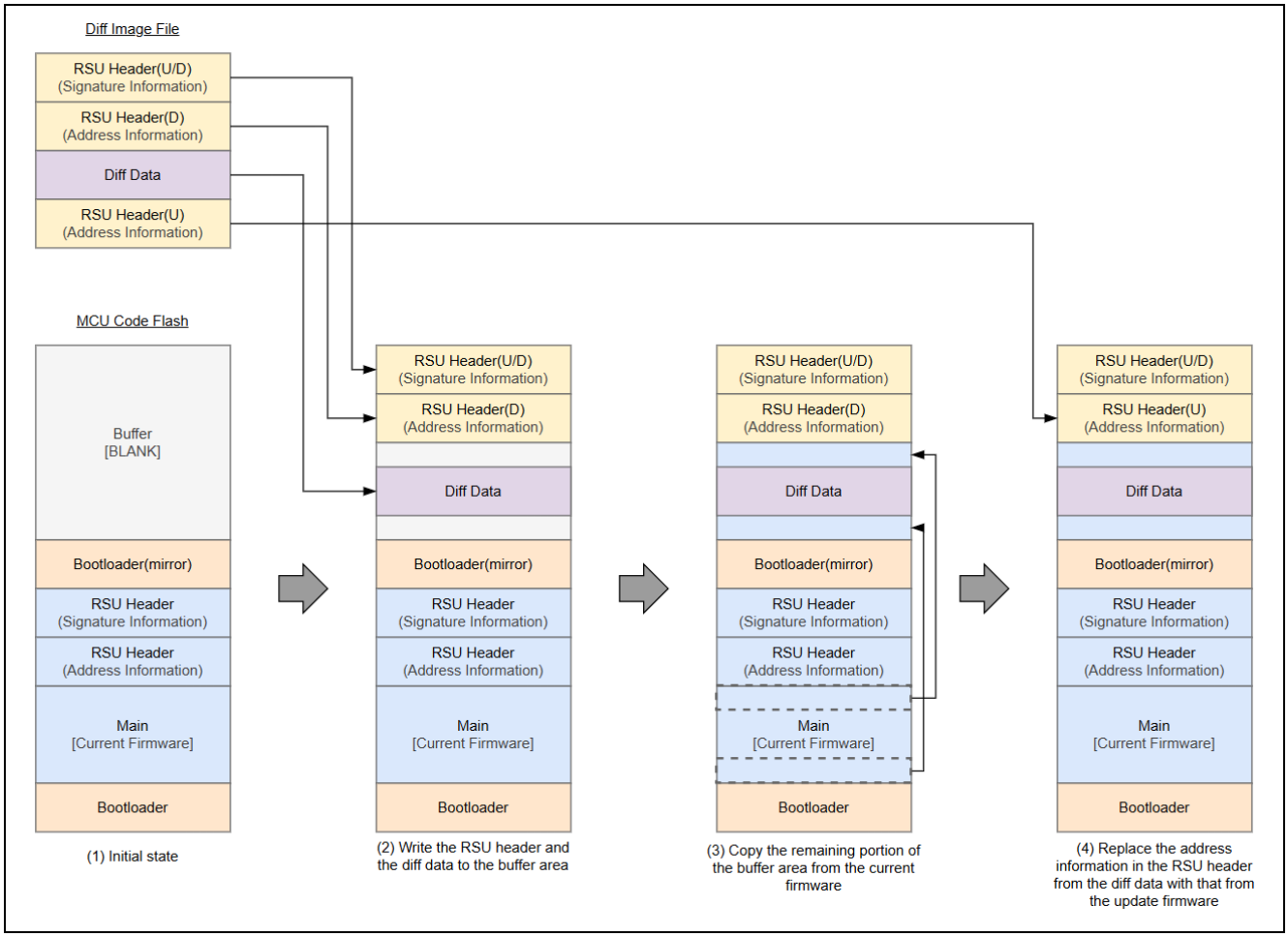


図 1-4 MCU での差分ファームウェア更新処理手順(1)

(1) デュアルバンク方式の初期状態です。差分ファームウェア更新のための特別なセットアップはありません。

(2) ファームウェアアップデートモジュールの R_FWUP_WriteImage 関数を使用して、差分イメージの RSU ヘッダと差分データをバッファエリアに書き込みます。

(3) 更新ファームウェアの差分ではない領域に現行ファームウェアのデータをコピーします。

(4) (2)で書き込んだ RSU ヘッダ(差分データのアドレス情報)を、差分イメージの RSU ヘッダ（更新ファームウェアのアドレス情報）に置き換えます。

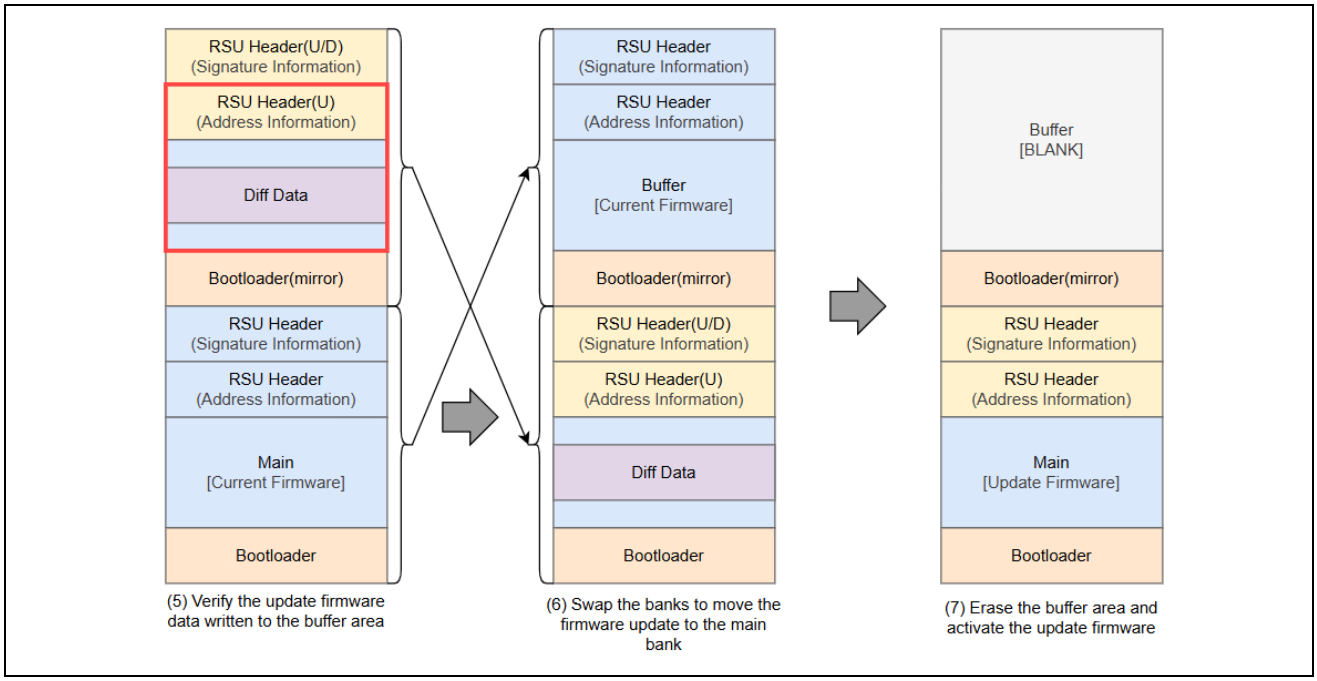


図 1-5 MCU での差分ファームウェア更新処理手順(2)

(5) バッファエリアに書き込んだ更新ファームウェアのデータの正当性を検証します。

(6) バンクスワップを実施し、更新ファームウェアをメインエリアに移動します。

(7) バッファエリアを消去し、更新ファームウェアを有効化します。

1.5 例外処理

差分更新が実行できないケースについて説明します。

1.5.1 Non-FF ブロックから All-FF ブロックへの差分の場合

現行ファームウェアでは 0xFF 以外の何らかのデータが 1 バイトでも書き込まれた状態のブロック（Non-FF ブロック）が、更新ファームウェアでは全て 0xFF のブランク状態（All-FF ブロック）に変化する場合、Renesas Image Generator による RSU ファイル生成処理の過程で、All-FF ブロックはデータ無し領域として除去され、RSU ファイルには出力されません。そのため、本差分更新方式では Non-FF ブロックから All-FF ブロックへの差分は反映されません。

本アプリケーションノートのサンプルプログラムとして提供する差分イメージ生成スクリプト（Delta Image Generator）では、Non-FF ブロックから All-FF ブロックへの差分を含む場合は差分イメージを出力しません。

1.5.2 現行ファームウェアと更新ファームウェアの差分ブロック数が 0 個の場合

差分が無いため、同一ファイルと判断します。そのため、差分イメージファイルは出力されません。

1.5.3 現行ファームウェアと更新ファームウェアの差分ブロック数が 32 個以上の場合

Renesas Image Generator の仕様上、RSU ヘッダのオフセット 0x200 で管理するアドレス情報の最大値 31 を超え、エラーが発生します。そのため、差分イメージファイルは出力されません。

2. Delta Image Generator

Delta Image Generator (delta-image-gen.py)は、本アプリケーションノートの差分ファームウェア更新で使用するファームウェアイメージを生成する Python スクリプトです。

Delta Image Generator は内部で Renesas Image Generator を実行します。Renesas Image Generator は、ファームウェアアップデートモジュールに付属するファームウェア更新対応の初期イメージ、更新イメージを生成するツールです。Renesas Image Generator の詳細や動作環境に関しては「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)」を参照してください。

2.1 使用方法

delta-image-gen.py のコマンド形式は以下の通りです。

```
python delta-image-gen.py <options>
```

表 2-1 delta-image-gen.py のオプション一覧

オプション	必須	説明
--before, -b	Yes	現行ファームウェアの MOT ファイルを絶対/相対パスで指定します。
--after, -a	Yes	更新ファームウェアの MOT ファイルを絶対/相対パスで指定します。
--output, -o	Yes	出力する差分イメージのファイル名を指定します。 出力されるファイル名は、「入力文字列.delta_rsu」となります。
--block-size, -bs	No	差分ブロックのサイズ（バイト）を指定します。 ファームウェア更新対象の MCU の ROM のプログラム単位の倍数値である必要があります。 デフォルト値: 128
--tool-dir, -t	No	Renesas Image Generator(image-gen.py)が存在するフォルダを絶対/相対パスで指定します。 デフォルト値: ./tool
-ip	Yes	Delta Image Generator が内部で Renesas Image Generator を実行する際の-ip オプションを指定します。（注）
-ff	No	Delta Image Generator が内部で Renesas Image Generator を実行する際の-ff オプションを指定します。（注） デフォルト値: BareMetal
-vt	No	Delta Image Generator が内部で Renesas Image Generator を実行する際の-vt オプションを指定します。（注） デフォルト値: ecdsa
-key	No	Delta Image Generator が内部で Renesas Image Generator を実行する際の-key オプションを指定します。（注）

注：詳細は「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) 5.1 イメージの生成方法」を参照してください。

2.2 使用例

Delta Image Generator を使用して、差分イメージを生成するコマンド入力例を以下に示します。

- 現行ファームウェア: ./fwup_main_v1.mot
- 更新ファームウェア: ./fwup_main_v2.mot
- 出力差分イメージ名: update_firm
- 差分ブロックサイズ: 128
- コード署名に使用する秘密鍵: ./key/secp256r1.privatekey
- Renesas Image Generator のパラメータファイル: ./tool/RX65N_DualBank_ImageGenerator_PRM.csv

コマンド入力例

```
python ./delta-image-gen.py -b ./fwup_main_v1.mot -a ./fwup_main_v2.mot -o
update_firm -bs 128 -key ./key/secp256r1.privatekey -
ip ./tool/RX65N_DualBank_ImageGenerator_PRM.csv
```

出力ファイル

- update_firm.delta_rsu: 差分イメージファイル
- fw_diff.mot: 現行ファームウェアに対する更新ファームウェアの差分をブロック単位で抽出した MOT ファイル
- update_output.rsu: Renesas Image Generator を用いて作成した、更新ファームウェアの RSU ファイル
- diff_output.rsu: Renesas Image Generator を用いて作成した、fw_diff.mot の RSU ファイル

3. 動作確認環境

本デモの動作確認環境を以下に示します。

表 3-1 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 e² studio 2025-12
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 (CC-RX) 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
使用 FIT モジュール	RX Driver Package V1.50 同梱物
使用ボード	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 3-2 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 e² studio 2025-12
C コンパイラ	GCC for Renesas RX 14.2.0.202505 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
使用 FIT モジュール	RX Driver Package V1.50 同梱物
使用ボード	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 3-3 動作確認環境(その他)

項目	内容
Python	3.12.6
Renesas Image Generator	V3.05 （FWUP FIT v2.06 に同梱）
Renesas Flash Programmer	V3.22.00
Tera Term	5.5.0

4. デモ

本デモでは差分ファームウェア更新方式でのファームウェア更新を実行します。

4.1 デモの動作説明

本デモの流れは次の通りです。

(1) 本アプリケーションノートで提供しているサンプルプロジェクトを使用して、差分ファームウェア更新用の初期イメージと更新イメージを作成します。

(2) CK-RX65Nv2 ボードに初期イメージを書き込みます。初期イメージでは、TeraTerm 上に出力されるログに「ver 1.0.0」と表示され、ボード上の LED4（青色）が点灯します。

(3) TeraTerm のシリアル通信機能を使用して CK-RX65Nv2 ボードに更新イメージを送信します。RX65N は差分ファームウェア更新方式の更新イメージを受信すると、更新処理を実行します。

(4) 更新処理完了後、更新イメージが実行され、TeraTerm 上に出力されるログが「ver 2.0.0」に変化します。また、ボード上の LED4（青色）が点滅動作に変わります。

4.2 デモプロジェクトのフォルダ構成

デモプロジェクトのフォルダ構成を以下に示します。

Demos フォルダには、CC-RX 用と GCC 用のデモプロジェクトを格納しています。

DeltaImageGenerator フォルダには、差分更新イメージを作成するための Python スクリプト（delta-image-gen.py）を格納しています。

tool フォルダには、FWUP FIT で提供している初期イメージおよび更新イメージを作成するための RenesasImageGenerator を格納しています。

key フォルダには、デモでファームウェアのデジタル署名に使用するサンプルの秘密鍵と公開鍵を格納しています。

```
r01an8321xx0100-rx-delta-fwup
├─Demos
│   ├─ccrx
│   │   ├─boot_loader
│   │   └─fwup_main
│   └─gcc
│       ├─boot_loader
│       └─fwup_main
├─DeltaImageGenerator
│   ├─key
│   │   ├─secp256r1.privatekey
│   │   └─secp256r1.publickey
│   ├─tool
│   │   ├─image-gen.py
│   │   ├─RX65N_DualBank_ImageGenerator_PRM.csv
│   │   └─...
│   └─delta-image-gen.py
├─r01an8321ej0100-rx-delta-fwup.pdf
└─r01an8321jj0100-rx-delta-fwup.pdf
```

4.3 デモプロジェクトの設定

既存プロジェクトを利用して、差分ファームウェア更新ができます。

本アプリケーションノートは、さらに差分更新の利点を活かすために、セクション追加による更新対象 ROM 領域を小さくしたプロジェクトを提供しています。

ユーザアプリに相当する fwup_main.c ファイルをファイル単位でセクション設定し、かつ、このセクションを他セクションから離して配置します。そのため、fwup_main.c のコード変更に伴う ROM 領域の差分情報を最小限にすることができます。

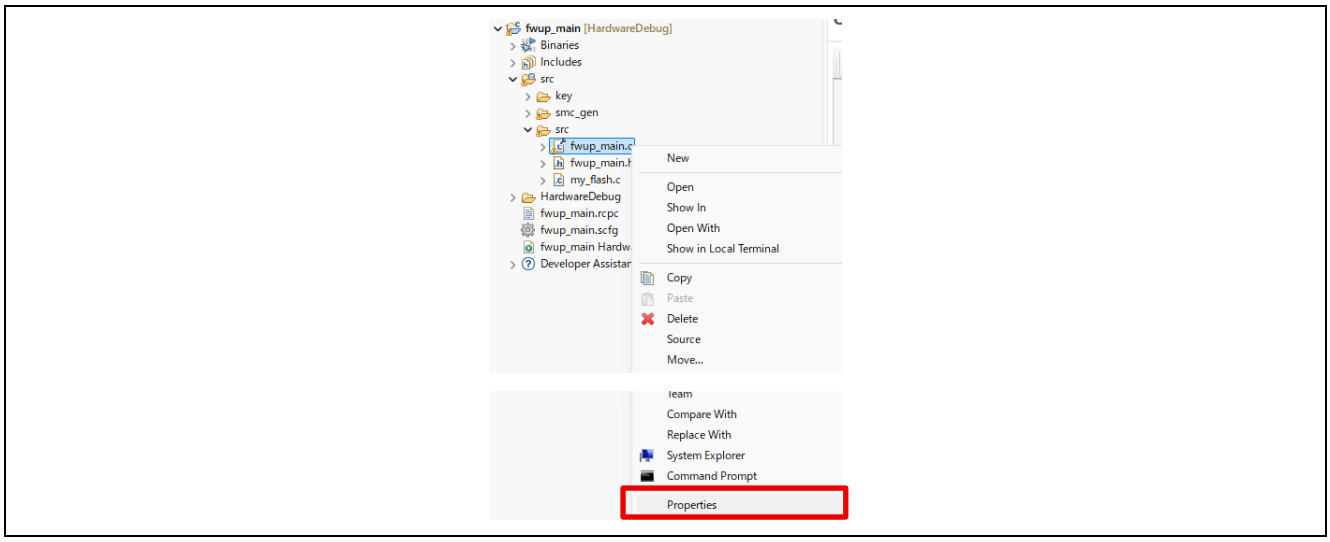
両デモプロジェクトは、fwup_main.c のセクションを 0xFFF40000 に配置しています。

4.3.1 CC-RX プロジェクト

4.3.1.1 ファイル単位でのセクション変更手順

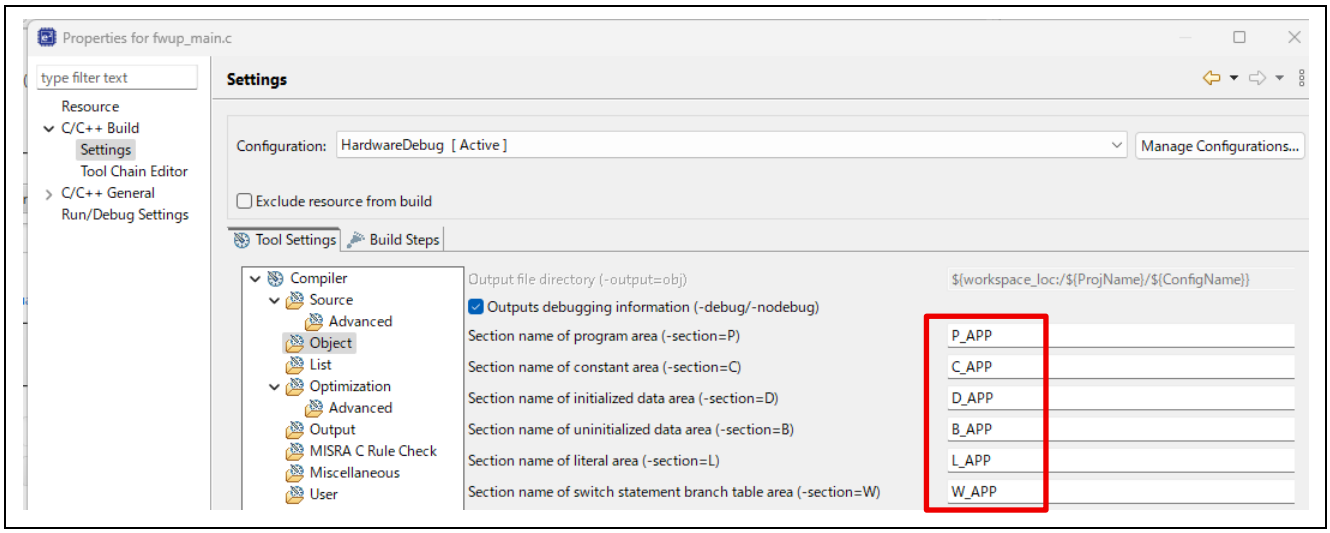
以下の手順で、ファイル単位でのセクション変更ができます。

(1) Project Explorer 上の“**fwup_main.c**“**ファイル**を右クリックし、“Properties“を選択



(2) “C/C++ Build“の“Settings“を選択

(3) “Tool Settings“内の“Compiler“→“Source“→“Object “をクリックし、セクション名を “X_APP“（X= P, C, D, B, L, W）に変更



4.3.1.2 プロジェクトレベルでのセクションの配置手順

以下の手順で、上記で追加したセクション配置アドレスを設定します。

(1) Project Explorer 上の”**fwup_main**”**プロジェクト**を右クリックし、“Properties“を選択

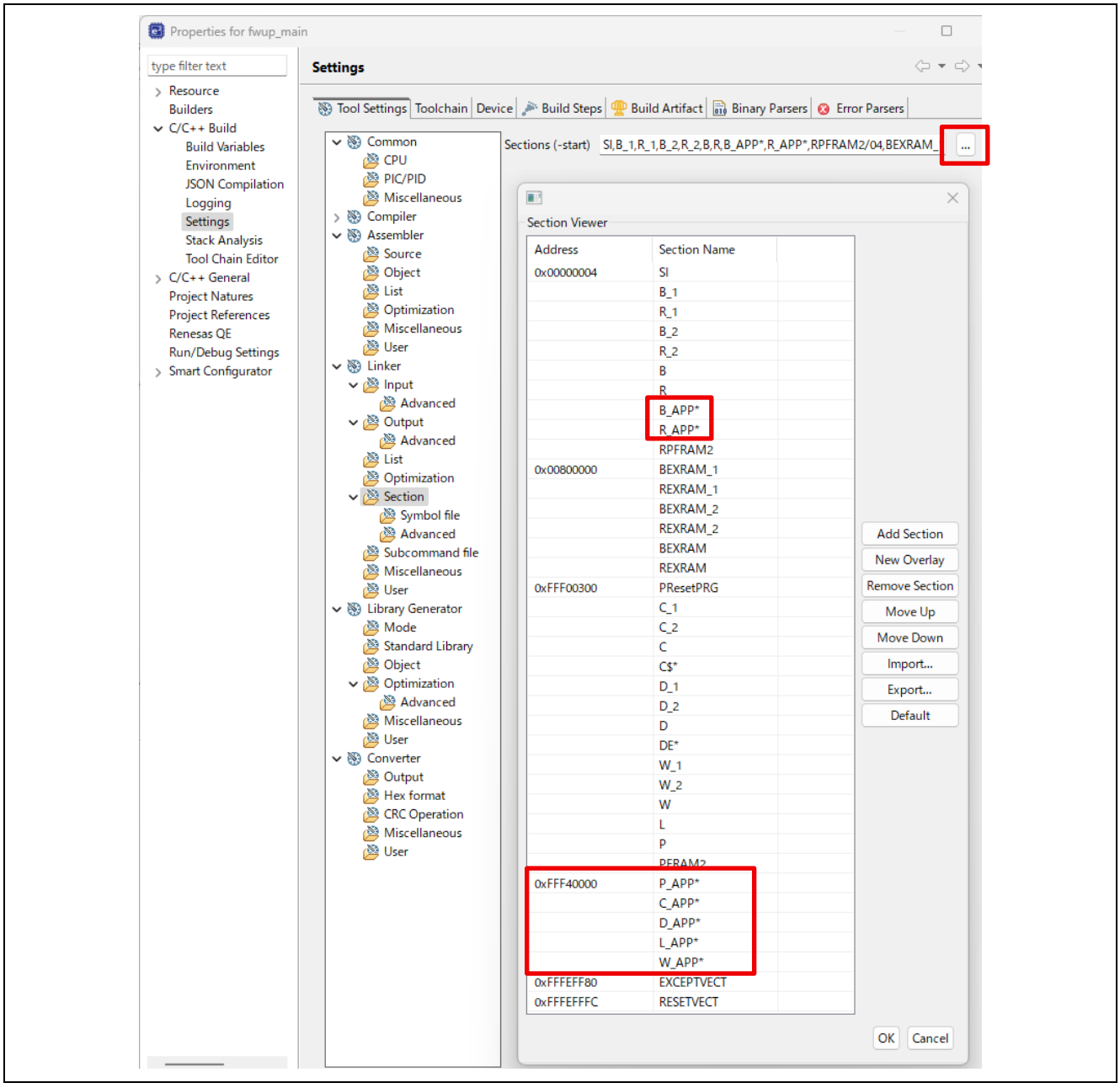
(2) “C++ Build“の“Settings“を選択

(3) “Tool Settings“内の“Linker“→“Section“をクリック。さらに“Sections (-start)“の右にある“...“をクリック

(4) セクションを追加／変更

下記は、デモプロジェクトの設定です。

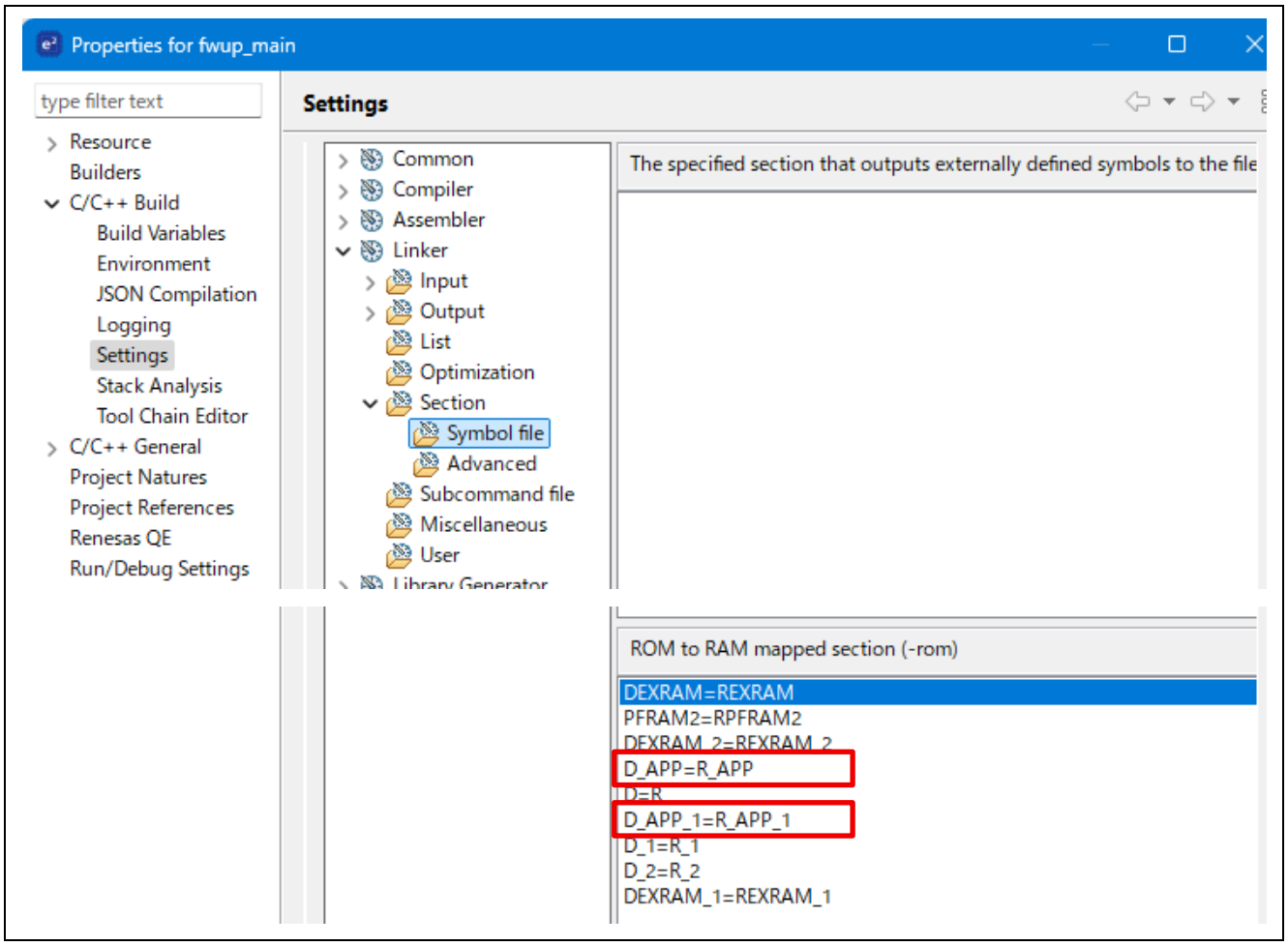
— 追加した B_APP/R_APP ：既存の B/R セクション領域の後方に配置
— 追加した P_APP/C_APP/D_APP/L_APP/W_APP ：新規の 0xFFF40000 に配置



(5) ROM to RAM Mapped セクションに ROM→RAM へのコピーセクションを追加

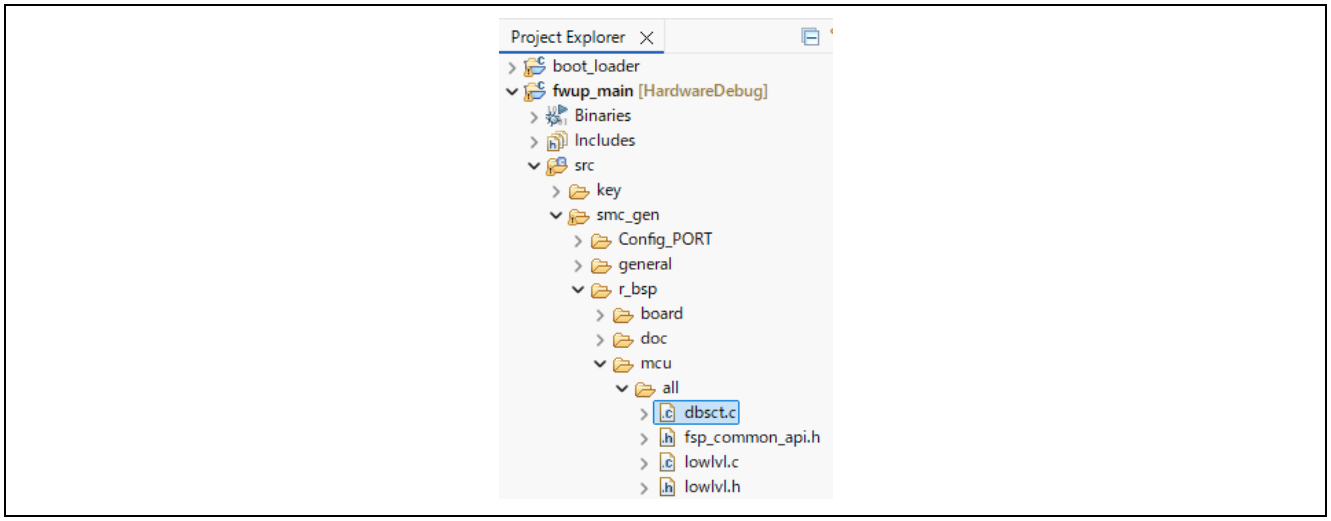
デモプロジェクトでは、以下を追加しています。

— D_APP = R_APP
— D_APP_1 = R_APP_1



(6) dbsct.c にセクションを追加

ROM→RAM へのコピー処理と未初期化領域の初期化を dbsct.c に追記します。



```
/* Section start */
#pragma section C C$DSEC

extern st_dtbl_t const _DTBL[] = {
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
    { __sectop("D_8"), __secend("D_8"), __sectop("R_8") },
#endif
    { __sectop("D"), __secend("D"), __sectop("R") },
    { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
    { __sectop("D_1"), __secend("D_1"), __sectop("R_1") },
    { __sectop("D_APP"), __secend("D_APP"), __sectop("R_APP") },
    { __sectop("D_APP_1"), __secend("D_APP_1"), __sectop("R_APP_1") }
#if (defined(BSP_CFG_EXPANSION_RAM_ENABLE) && (BSP_CFG_EXPANSION_RAM_ENABLE == 1))
#ifdef BSP_EXPANSION_RAM
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
   ,{ __sectop("DEXRAM_8"), __secend("DEXRAM_8"), __sectop("REXRAM_8") }
#endif
   ,{ __sectop("DEXRAM"), __secend("DEXRAM"), __sectop("REXRAM") },
    { __sectop("DEXRAM_2"), __secend("DEXRAM_2"), __sectop("REXRAM_2") },
    { __sectop("DEXRAM_1"), __secend("DEXRAM_1"), __sectop("REXRAM_1") }
#endif
#endif
#if (BSP_CFG_RTOS_USED == 4) && (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600PX)
   ,{ __sectop("DRI_ROM"), __secend("DRI_ROM"), __sectop("RRI_RAM") }
#endif /* Renesas RI600PX */
};

/* Section start */
#pragma section C C$BSEC

extern st_btbl_t const _BTBL[] = {
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
    { __sectop("B_8"), __secend("B_8") },
#endif
    { __sectop("B"), __secend("B") },
    { __sectop("B_2"), __secend("B_2") },
    { __sectop("B_1"), __secend("B_1") },
    { __sectop("B_APP"), __secend("B_APP") },
    { __sectop("B_APP_1"), __secend("B_APP_1") }
#if (defined(BSP_CFG_EXPANSION_RAM_ENABLE) && (BSP_CFG_EXPANSION_RAM_ENABLE == 1))
#ifdef BSP_EXPANSION_RAM
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
   ,{ __sectop("BEXRAM_8"), __secend("BEXRAM_8") }
#endif
   ,{ __sectop("BEXRAM"), __secend("BEXRAM") },
    { __sectop("BEXRAM_2"), __secend("BEXRAM_2") },
    { __sectop("BEXRAM_1"), __secend("BEXRAM_1") }
#endif
#endif
#if (BSP_CFG_RTOS_USED == 4) && (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600V4)
   ,{ __sectop("BRI_RAM"), __secend("BRI_RAM") }
#endif /* Renesas RI600V4 */
};
```

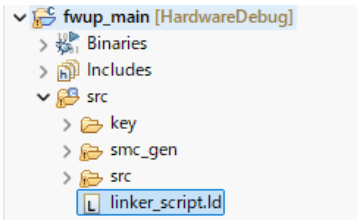
4.3.1.3 ビルド時の Warning について

追加した W_APP セクションに関する以下の Warning が出力されますが、動作上問題ありません。無視してください。

```
W0561100:Cannot find "W_APP*" specified in option "start"
```

4.3.2 GCC プロジェクト

Linker_script.ld ファイルを編集し、セクションを追加します。



(1) MEMORY 領域：変更不要

MEMORY 領域を編集せず、ファイル単位でセクションを設定するため、変更不要です。

```
MEMORY
{
        RAM : ORIGIN = 0x4, LENGTH = 0x3fffc
        RAM2 : ORIGIN = 0x00800000, LENGTH = 393216
        ROM : ORIGIN = 0xFFF00000, LENGTH = 960K
        OFS : ORIGIN = 0xFE7F5D00, LENGTH = 128
}
```

(2) SECTIONS：ROM 領域 text セクション設定

EXCLUDE_FILE(* fwup_main.o)設定により、fwup_main.c の text セクションと RX 固有の P, P_1, P_2 セクションを ROM 領域の text セクションの対象外に設定します。

```
SECTIONS
{
        .exvectors 0xFFFEFF80 : AT(0xFFFEFF80)
        {
                "_exvectors_start" = .;
                KEEP(*(.exvectors))
                "_exvectors_end" = .;
        } >ROM
        .fvectors 0xFFFEFFFC : AT(0xFFFEFFFC)
        {
                KEEP(*(.fvectors))
        } >ROM
        .text 0xFFF00300 : AT(0xFFF00300)
        {
                /* Exclude fwup_main.o */
                *(EXCLUDE_FILE(*fwup_main.o) .text*)
                /* P* is not used because there are PFRAM* sections */
                *(EXCLUDE_FILE(*fwup_main.o) P)
                *(EXCLUDE_FILE(*fwup_main.o) P_1)
                *(EXCLUDE_FILE(*fwup_main.o) P_2)
                KEEP(*(.text.*_isr))
                KEEP(*(.text.*isr))
                KEEP(*(.text.Excep_*))
                KEEP(*(.text.*ISR))
                KEEP(*(.text.*interrupt))
                etext = .;
        } >ROM
```

(3) SECTIONS：ROM 領域 rodata セクション設定

EXCLUDE_FILE(* fwup_main.o)設定により、fwup_main.c の rodata セクションと RX 固有の C セクションを ROM 領域の rodata セクションの対象外に設定します。

```
.rodata :
{
        /* Exclude fwup_main.o */
        *(EXCLUDE_FILE(*fwup_main.o) .rodata*)
        *(EXCLUDE_FILE(*fwup_main.o) C*)
        _erodata = .;
} >ROM
```

(4) SECTIONS：fwup_app 用 text セクション、rodata セクション配置設定

ROM 領域に fwup_main.c 専用の app セクションを追加ます。

設定を以下に示します。

— app セクション開始アドレスを 0xFFF40000 に設定
— ROM 上に連続して格納するため、text 関連、P 関連、rodata 関連、C 関連のセクションを設定

```
.pfram2 ALIGN(4) :
{
        _PFRAM2_start = .;
        . += _RPFRAM2_end - _RPFRAM2_start;
        _PFRAM2_end = .;
} >ROM
/* Add the app section for fwup_main.c in ROM */
.app 0xFFF40000 : AT(0xFFF40000)
{
        . = ALIGN(4);
        _fwup_main_start = .;

        KEEP(*fwup_main.o(.text*))
        KEEP(*fwup_main.o(P*))

        *fwup_main.o(.rodata*)
        *fwup_main.o(C*)

        . = ALIGN(4);
        _fwup_main_end = .;
} >ROM
```

(5) SECTIONS：ROM to RAM へのコピー処理と未初期化領域の初期化処理：変更不要

ROM→RAM へのコピー処理と未初期化領域の初期化は reset_program.S の__INITSCT で処理されます。

(a) ROM→RAM へのコピー処理

コピー処理では、アドレス_mdata を初期値付き変数のコピー元として扱っています。

アドレス_data から_edata まで、初期値付き変数の全セクションが含まれた設定のため、変更不要です。

```
.data : AT(_mdata)
{
        _data = .;
        *(.data)
        *(.data.*)
        *(D)
        *(D_1)
        *(D_2)
        _edata = .;
} >RAM
```

アドレス_mdata から初期値付き変数用の ROM 領域を設定します。アドレス_mdata から、(_edata - _data)分、.data セクション分の ROM が確保された設定のため、変更不要です。

```
.tors :
{
        __CTOR_LIST__ = .;
                …
        . = ALIGN(2);
        _mdata = .;
        . += _edata - _data;
} >ROM
```

(b) 未初期化領域の初期化

初期化処理では、アドレス_bss を初期化開始アドレスとして扱っています。

アドレス_bss から_ebss までに、初期値無し変数の全セクションが含まれた設定のため、変更不要です。

```
.bss :
{
        _bss = .;
        *(.bss)
        *(.bss.**)
        *(COMMON)
        *(B)
        *(B_1)
        *(B_2)
        _ebss = .;
        . = ALIGN(128);
        _end = .;
} >RAM AT>RAM
```

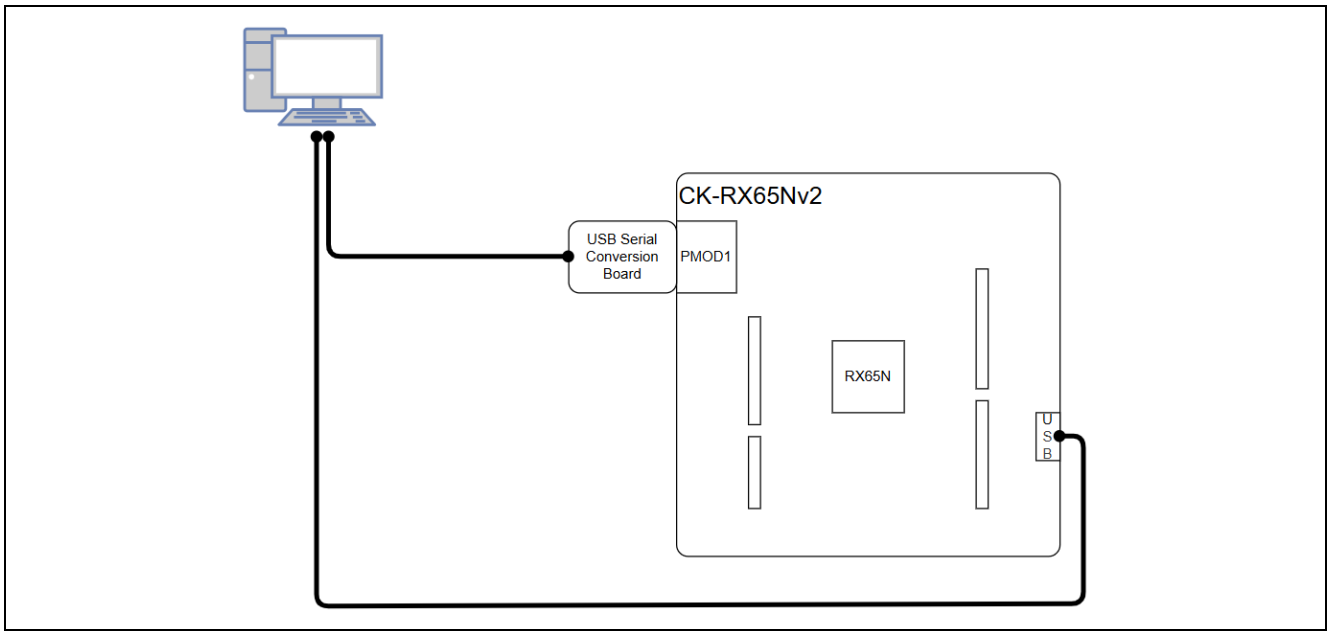
4.3.2.2 ビルド時の Warning について

スタック使用に関する以下のような Warning が計 7 件出力されます。現行プロジェクトでは十分なスタックサイズを確保済です。必要に応じてスタックサイズを見直してください。

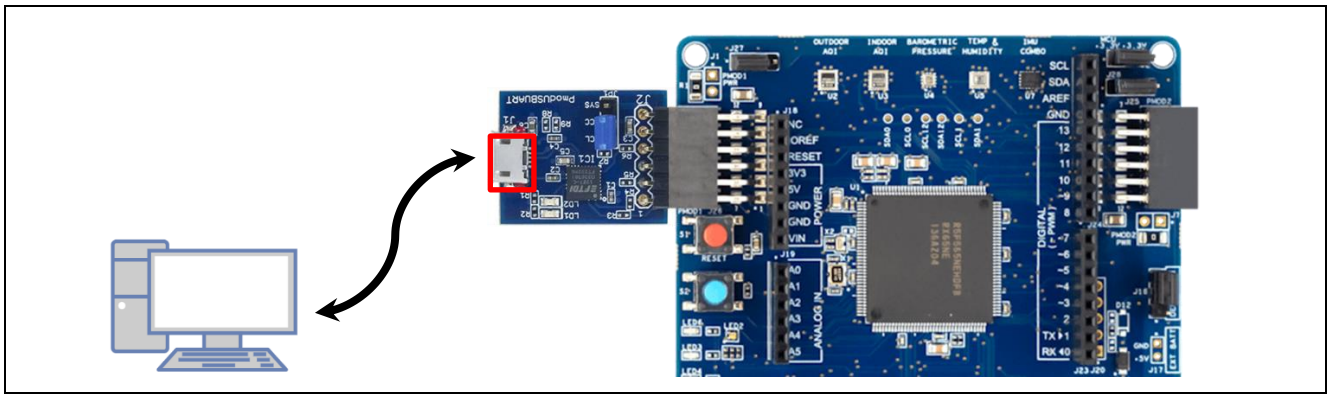
```
warning: stack usage is XXX bytes [-Wstack-usage=]

warning: stack usage computation not supported for this target
```

4.4 ハードウェアのセットアップ

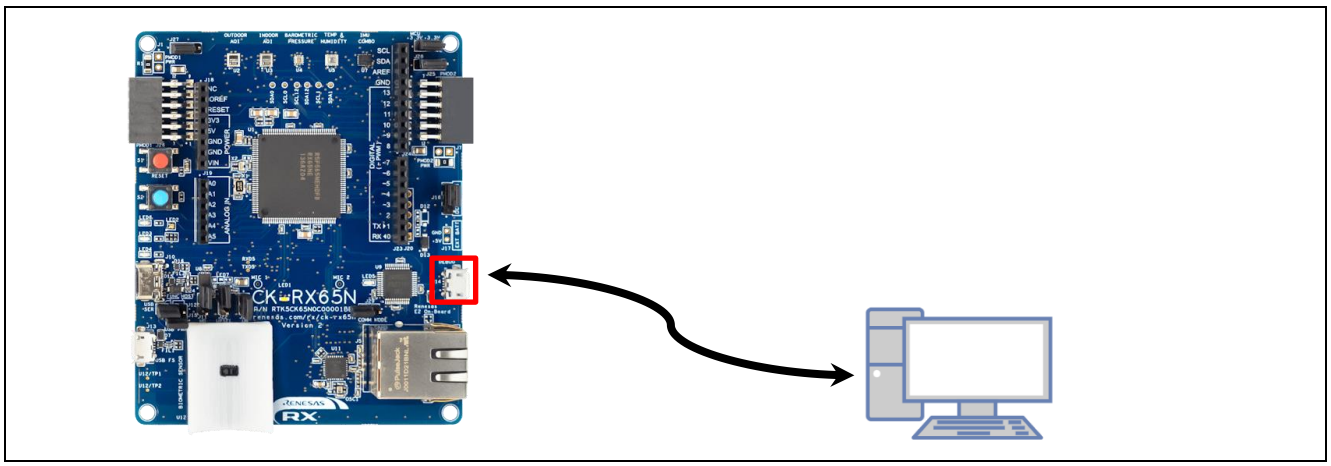
デモのハードウェア構成を以下に示します。



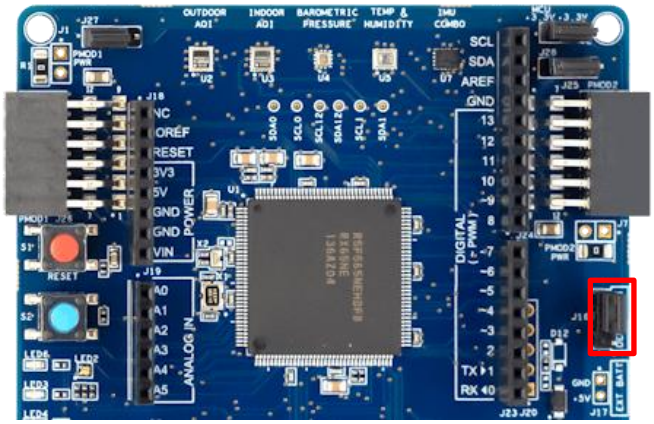
(1) USB シリアル変換ボードを CK-RX65Nv2 ボードの PMOD1 端子の Pin1-6 に接続します。USB シリアル変換ボードの USB コネクタを PC に接続します。



(2) CK-RX65Nv2 ボードの E2OB Debugger コネクタ(microUSB Type-B)を PC に接続します。



(3)　CK-RX65Nv2 ボードのジャンパ J16 を DEBUG 側(pin 1-2)に短絡します。



4.5　ソフトウェアのセットアップ

4.5.1　動作環境準備

デモの実行に必要なアプリを PC にインストールします。

4.5.1.1　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 5.5.0 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 4-1 に示すように設定してください。

表 4-1 シリアル通信仕様

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

4.5.1.2　Python 実行環境のインストール

Delta Image Generator(delta-image-gen.py)で差分イメージを作成するために使用します。デモプロジェクトでは、Python 3.12.6 で動作確認を実施しています。

また、Delta Image Generator が内部で実行する Renesas Image Generator (image-gen.py)は Python の暗号化ライブラリ(pycryptodome)を使用するため、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
```

4.5.1.3 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 (RFP) V3.22.00 を使用しています。

Renesas Flash Programmer (Programming GUI) | Renesas

デモプロジェクトでは、デュアルバンク方式を利用するため、FW 更新を繰り返すとバンクの位置が分からなくなります。また、e² studio が出力する MOT 形式ファイルでは、Bank 起動設定は Bank0 に固定されます。そのため、RFP で MOT ファイルを書き込む際に、Bank0/1 不整合が発生する可能性があります。

その解決方法として「RX ファミリ デュアルモードの使用ガイド (R01AN6894)」の「2.1 バンクの位置が分からない場合」を参照し、MCU を出荷状態に戻してください。

RFP による初期イメージの書き込みフローを以下に示します。

1. MCU 出荷状態で Linear 用 RFP プロジェクトを使用し、初期イメージを書き込む。
「Dual/Bank0 起動」設定に切り替わるため、デモプロジェクトの FW 更新が可能。
2. RFP による初期イメージを再度書き込む場合、以下のどちらかの RFP プロジェクトを使用し「チップ消去」を実行する。（注）

- FW 更新済状態で新規作成した RFP プロジェクト
- 「Dual」用 RFP プロジェクト

「Linear」設定に切り替わり、MCU 出荷状態になるため、初期イメージの書き込みが可能。

注：詳細は「RX ファミリ デュアルモードの使用ガイド (R01AN6894)」を参照してください。

4.6 デモの実行手順

本章では、CC-RX 用のデモプロジェクトを例に実行手順を記載します。GCC 用のデモプロジェクトにおいても手順は共通です。

4.6.1 デモプロジェクトの構築

4.6.1.1 初期イメージの作成

初期イメージ名を initial_firm.mot として、作成手順を説明します。

(1) e² studio に boot_loader, fwup_main 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、ビルドされた MOT ファイルが生成されていることを確認し、DeltaImageGenerator フォルダにコピーします。DeltaImageGenerator フォルダ内は以下のようなファイル構成となります。

```
key
tool
delta-image-gen.py
boot_loader.mot
fwup_main.mot
```

(3) RenesasImageGenerator を使用して初期イメージを作成します。DeltaImageGenerator フォルダで以下のコマンドを実行します。

```
python ./tool/image-gen.py -iup "./fwup_main.mot" -
ip ./tool/RX65N_DualBank_ImageGenerator_PRM.csv -o initial_firm -ibp
"./boot_loader.mot" -vt ecdsa -key "./key/secp256r1.privatekey"
```

DeltaImageGenerator フォルダ内に initial_firm.mot という名前のファイルが生成されます。

4.6.1.2 差分更新イメージの作成

差分更新イメージ名を update_firm.delta_rsu として、作成手順を説明します。

(1) 「4.6.1.1 初期イメージの作成」で fwup_main プロジェクトをビルドして作成した fwup_main.mot ファイルを fwup_main_v1.mot と名前を変更します。

(2) fwup_main¥src¥src¥fwup_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 fwup_main プロジェクトをビルドします。

```
58  /* FW version definition */
59  #define DEMO_VER_MAJOR          (1)
60  #define DEMO_VER_MINOR          (0)
61  #define DEMO_VER_BUILD          (0)
```

(3) 生成された MOT ファイルを fwup_main_v2.mot と名前を変更し、DeltaImageGenerator フォルダにコピーします。DeltaImageGenerator フォルダ内は以下のようなファイル構成となります。

```
key
tool
delta-image-gen.py
initial_firm.mot
boot_loader.mot
fwup_main_v1.mot
fwup_main_v2.mot
```

(4) DeltaImageGenerator を使用して差分更新イメージを作成します。DeltaImageGenerator フォルダで以下のコマンドを実行します。

```
python ./delta-image-gen.py -b ./fwup_main_v1.mot -a ./fwup_main_v2.mot -o
update_firm -bs 128 -key ./key/secp256r1.privatekey -
ip ./tool/RX65N_DualBank_ImageGenerator_PRM.csv
```

以下のように実行ログが出力され、DeltaImageGenerator フォルダに update_firm.delta_rsu ファイルが生成されます。

```
=== Step 1: Parsing S-record files ===
Parsing: ./fwup_main_v1.mot
  Address range: 0xFE7F5D00 - 0xFFFEFFFF
  Data bytes: 37184
Parsing: ./fwup_main_v2.mot
  Address range: 0xFE7F5D00 - 0xFFFEFFFF
  Data bytes: 37184

=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
Diff blocks: 1
Block size: 128 bytes
Total diff size: 128 bytes
Addresses:
  0xFFF40680 - 0xFFF406FF
Generated diff MOT file: C:¥delta_fwup¥DeltaImageGenerator¥fw_diff.mot

=== Step 3: Generating RSU files ===
Using parameter file:
C:¥delta_fwup¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv
Using private key   : C:¥delta_fwup¥DeltaImageGenerator¥key¥secp256r1.privatekey

Generating RSU for diff data...
Running: C:¥Users¥taro¥.pyenv¥pyenv-win¥versions¥3.12.6¥python.exe
C:¥delta_fwup¥DeltaImageGenerator¥tool¥image-gen.py -iup
C:¥delta_fwup¥DeltaImageGenerator¥fw_diff.mot -ip
C:¥delta_fwup¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv -vt
ecdsa -ff BareMetal -o diff_output -key
C:¥delta_fwup¥DeltaImageGenerator¥key¥secp256r1.privatekey


Successfully generated the diff_output.rsu file.


Generating RSU for update firmware...
Running: C:¥Users¥taro¥.pyenv¥pyenv-win¥versions¥3.12.6¥python.exe
C:¥delta_fwup¥DeltaImageGenerator¥tool¥image-gen.py -iup
C:¥delta_fwup¥Demos¥ccrx¥fw¥fwup_main_v2.mot -ip
C:¥delta_fwup¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv -vt
ecdsa -ff BareMetal -o update_output -key
C:¥delta_fwup¥DeltaImageGenerator¥key¥secp256r1.privatekey


Successfully generated the update_output.rsu file.


=== Step 4: Merging RSU files ===
Merged file created: update_firm.delta_rsu
File size: 1152 bytes

=== Complete ===
Output file: update_firm.delta_rsu
Total file size: 1152 bytes
Diff blocks: 1
Total diff size: 128 bytes
```

4.6.1.3 差分ファイル(update_firm.delta_rsu)が生成されない特殊ケース

「1.5 例外処理」に示す差分更新ファイルが生成されない場合の DeltaImageGenerator の実行結果を以下に示します。ログで実行結果を判断できます。

(1) Non-FF ブロックから All-FF ブロックへの差分の場合

DeltaImageGenerator 使用時、Step 2 にて、以下の All-FF ブロック検出のログが出力されます。
差分ファイルは生成されず、update_firm.rsu ファイル（注）が生成されます。このファイルを使用して FW 更新できます。
注：update_firm.rsu ファイル: RenesasImageGenerator を使用して生成された RSU ファイルと同一

```
=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
WARNING: Erased block (all 0xFF) detected at 0xFFFXXXXX - 0xFFFXXXX
Aborting diff generation. Will generate RSU from update firmware directly.

=== Erased block detected: generating RSU from update firmware directly ===
Using parameter file: D:¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv
Using private key   : D:¥DeltaImageGenerator¥key¥secp256r1.privatekey

Generating RSU for update firmware...
Running: C:¥Program Files¥Python313¥python.exe D:¥DeltaImageGenerator¥tool¥image-gen.py
-iup D:¥DeltaImageGenerator¥fwup_main_v2.mot -ip
D:¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv -vt ecdsa -ff
BareMetal -o update_output -key D:¥DeltaImageGenerator¥key¥secp256r1.privatekey
Successfully generated the update_output.rsu file.

=== Complete ===
Output file: update_firm.rsu
Note: Diff generation was aborted due to erased block detection in the update firmware.
The output file contains the entire update firmware RSU.
```

(2) 現行ファームウェアと更新ファームウェアの差分ブロック数が 0 個（同一）の場合

DeltaImageGenerator 使用時、Step2 にて、以下のログが出力され、終了します。
この場合、更新不要です。

```
=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
No differences found between the two firmware files.
```

(3) 現行ファームウェアと更新ファームウェアの差分ブロック数が 32 個以上の場合

DeltaImageGenerator 使用時、以下のログが出力されます。
Step 2 にて、以下の”Diff block: NN”（NN > 31）検出のログ
Step 3 にて、以下の” The number of program data exceeds 31.”と”Error: ”のログ

差分ファイル、および update_firm.rsu ファイルは、いずれも生成されません。

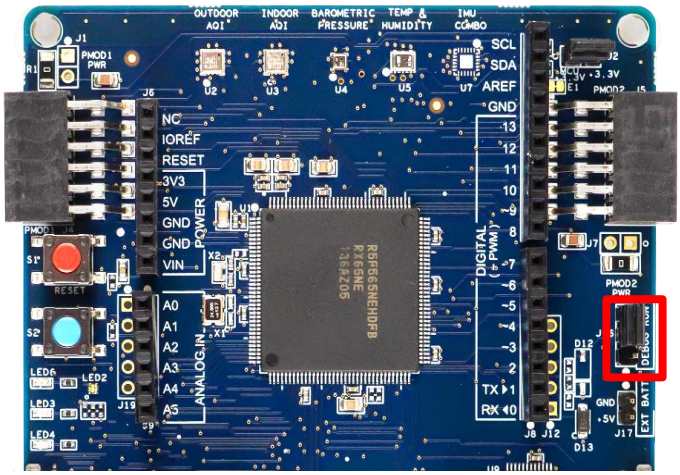
```
=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
Diff blocks: 32    <- Detected Number of Blocks
Block size: 128 bytes
Total diff size: XXXX bytes
Addresses:
  0xFFFXXXXX - 0xFFFXXXX
  …

=== Step 3: Generating RSU files ===
Using parameter file: D:¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv
Using private key   : D:¥DeltaImageGenerator¥key¥secp256r1.privatekey

Generating RSU for diff data...
Running: C:¥Program Files¥Python313¥python.exe D:¥DeltaImageGenerator¥tool¥image-gen.py
-iup D:¥DeltaImageGenerator¥fw_diff.mot -ip
D:¥DeltaImageGenerator¥tool¥RX65N_DualBank_ImageGenerator_PRM.csv -vt ecdsa -ff
BareMetal -o diff_output -key D:¥DeltaImageGenerator¥key¥secp256r1.privatekey
The number of program data exceeds 31.

Error: RSU file not created: D:¥DeltaImageGenerator¥diff_output.rsu
Traceback (most recent call last):
  File "D:¥DeltaImageGenerator¥delta-image-gen.py", line 739, in main
    diff_rsu_path = runner.generate_rsu(
        diff_mot_path,
    ...<2 lines>...
        args.ff
    )
  File "D:¥DeltaImageGenerator¥delta-image-gen.py", line 453, in generate_rsu
    raise RuntimeError(f"RSU file not created: {rsu_path}")
RuntimeError: RSU file not created: D:¥DeltaImageGenerator¥diff_output.rsu
```

4.6.2 初期イメージの書き込みと実行

(1) 「4.6.1.1 初期イメージの作成」で作成した initial_firm.mot をフラッシュライタで CK-RX65Nv2 ボードに書き込みます。

(2) TeraTerm を起動し、CK-RX65Nv2 ボードに接続した USB シリアル変換ボードのシリアル COM ポートを選択し接続設定を行います。シリアル通信の設定は「表 4-1 シリアル通信仕様」を参照してください。

CK-RX65Nv2 ボードのジャンパ J16 を RUN 側(pin 2-3)に短絡します。



(3) TeraTerm に初期イメージ(ver 1.0.0)のログが以下のように出力されます。また、CK-RX65Nv2 ボード上の LED4（青色）が点灯します。

```
==== RX65N : BootLoader [dual bank] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX65N : Update from User [dual bank]  ver 1.0.0 ====
send image(*.rsu) via UART.
```

もし、以下のように、**「ver 1.0.0」が表示されない場合、**「BootLoader」から「Update form User (Main)」に遷移していないことを示しており、Bank0/1 不整合が発生しています。

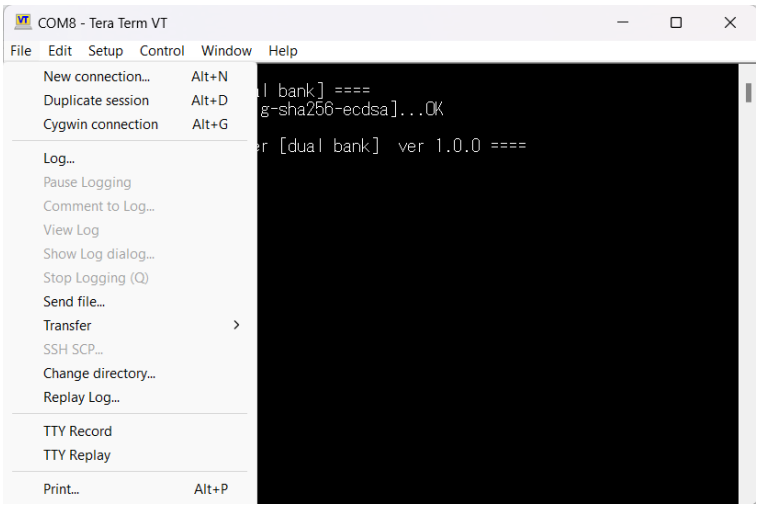
「4.5.1.3 フラッシュライタのインストール」を参照し、initial_firm.mot を書き直してください。

```
==== RX65N : BootLoader [dual bank] ====
send image(*.rsu) via UART.
```

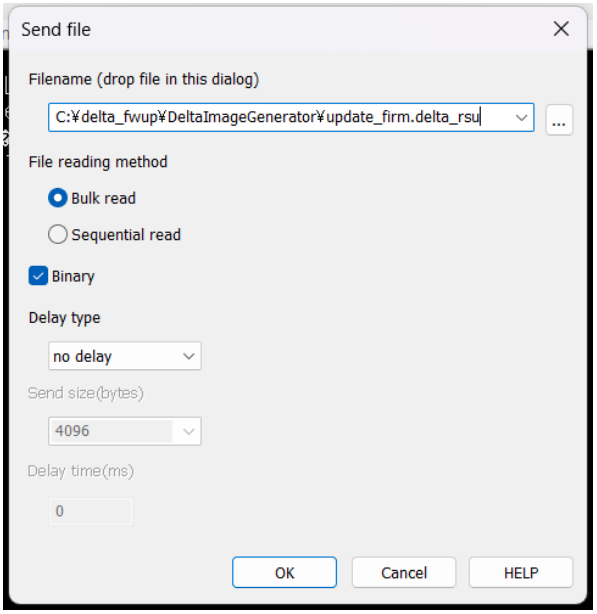
s

4.6.3 ファームウェアアップデートの実行

(1) TeraTerm の「File」メニューから「Send file…」をクリックします。



(2) Filename に「4.6.1.2 差分更新イメージの作成」で作成した差分更新イメージファイルを選択し、「Binary」オプションにチェックを入れて「OK」をクリックします。



(3) 更新イメージの転送中および更新処理中は進捗が以下のようにログ出力され、更新処理が完了すると自動的に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。

(4) ログ出力されたファームウェアバージョンが「ver 2.0.0」になっていれば成功です。また、CK-RX65Nv2 ボード上の LED4（青色）が点灯から点滅に変わります。

```
==== RX65N : BootLoader [dual bank] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX65N : Update from User [dual bank]  ver 1.0.0 ====
send image(*.rsu) via UART.
W 0xFFE00000, 256 ... OK
W 0xFFE00100, 256 ... OK
W 0xFFE00200, 256 ... OK
W 0xFFE40680, 128 ... OK
copy current data ... OK
rewrite addr info ... OK
verify install area buffer [sig-sha256-ecdsa]...OK
bank swap ...
software reset...
==== RX65N : BootLoader [dual bank] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX65N : Update from User [dual bank]  ver 2.0.0 ====
send image(*.rsu) via UART.
```

デモの実行手順の説明は以上です。

4.7 本デモで生成した差分更新イメージの特性

本デモの「4.6.1.2 差分更新イメージの作成」で作成した差分更新イメージの特性を以下に示します。

表 4-2 本デモプログラムの差分更新イメージの特性

項目	内容
更新前後の動作変更	• ログ出力されるバージョンの変更(1.0.0 → 2.0.0) • LED の点灯パターンの変更（常時点灯 → 点滅）
差分更新イメージファイルサイズ	1152 bytes
差分ブロック数	1
差分データサイズ（注）	128 bytes

注：差分ブロックサイズ×差分ブロック数

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2026/07/03	—	新規作成

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。