Renesas RA ファミリ

MCUboot とフラッシュデュアルバンクを使用した RA6 セキュアファームウェアアップデート

要旨

MCUboot は 32 ビットマイコン向けのセキュアブートローダです。ブートローダの共通インフラストラクチャとマイクロコントローラシステムのシステム上のフラッシュレイアウトを定義し、簡単にソフトウェア更新を可能にするセキュアブートローダを提供します。MCUboot は、オペレーティングシステムとハードウェアに依存せず、オペレーティングシステムが動作するハードウェアポーティングレイヤに依存します。Renesas Flexible Software Package (FSP)は、FSP v3.0.0 以降の MCUboot Port を統合しています。ユーザは、FSP MCUboot モジュールを使用することで、Root of Trust (RoT)による安全な起動とフェイルセーフなアプリケーションの更新が容易に実現可能です。

MCUboot は GitHub mcu-tools のページ https://github.com/mcu-tools/mcuboot 内の Linaro によってメンテナンスされています。ここには MCUboot のドキュメントを.md ファイル形式で保持する`\docs` フォルダがあります。本アプリケーションノートでは、可能な限り本フォルダ内のドキュメントを参照し、FSP MCUboot モジュールの使用に関する情報を提供することを目的としています。

RA ファミリ MCU の RA6M4 および RA6M5 グループの場合、内蔵コードフラッシュメモリのデュアルバンク機能によりファームウェアの更新が簡単かつ高速に可能です。このデュアルバンク機能は、FSP v3.6.0 からサポートされています。本アプリケーションノートでは、デュアルバンク機能を使用したセキュアブートローダの設計について説明します。

本アプリケーションプロジェクトは、EK-RA6M4 を使用したプロジェクトの例で説明します。さらに、ブートローダを使用したアプリケーションを使いこなす方法と、新しいアプリケーションに更新する方法の手順についても説明します。ユーザはこれらの手順に従って参考のブートローダを再現し、本アプリケーションプロジェクトに付属のサンプルアプリケーションプロジェクトをリンクさせてブートローダを使用することができます。

RA6 内部フラッシュリニアモードにおける MCUboot モジュールを使用したセキュアブートローダの設計については、アプリケーションプロジェクト R11AN0497 を参照してください。

開発環境

開発ツールとソフトウェア

- e² studio ISDE v2022-01 以降
- Renesas Flexible Software Package (FSP) v3.6.0 以降
- SEGGER J-link®用 USB ドライバ

上記 3 つがバンドルされたコンポーネントは renesas.com/ra/fsp より入手可能です。

- Python v3.8 以降 https://www.python.org/downloads/
- Renesas Flash Programming (RFP) v3.09 以降
  https://www.renesas.com/jp/ja/software-tool/renesas-flash-programmer-programming-gui

ハードウェア

- EK-RA6M4, RA6M4 MCU グループ評価キット http://www.renesas.com/ra/ek-ra6m4
- Windows®10 搭載 PC
- USB ケーブル×2 (type-A メス to micro-B メス)
- USB to TTL Serial 3.3-V UART 変換ケーブル×1

前提条件と対象ユーザ

本アプリケーションノートは、Renesas 製の IDE e^2 studio の使用経験があることを前提としています。また、ユーザは事前に FSP ユーザーズマニュアルの MCUboot Port セクションと MCU ユーザーズマニュアルハードウェア編のフラッシュメモリセクションを理解いただく必要があります。さらに、暗号に関する知識があることも前提です。Python の使用に関する予備知識も役に立ちます。

本アプリケーションノートの対象者は、製品開発者、製品製造者、製品サポート担当者、エンドユーザなどで、セキュアブートローダを使用するアプリケーションシステムの設計に携わる方々です。

さらに、RA MCU にセキュアキーをインストールする、またはインストールしたセキュアキーを使用する段階に携わる製品開発者、製品メーカー、製品サポート、エンドユーザです。

本アプリケーションノートの構成

以下の 8 つのセクションで構成されます。

セクション 1：RA6M4 と RA6M5 のコードフラッシュデュアルバンク機能の概要です。
MCU デュアルバンク機能に精通しているユーザはこのセクションをスキップできます。

セクション 2：FSP MCUboot モジュールを使用したシステム設計の一般的な流れについて説明します。
たとえば、MCUboot を使用したコードフラッシュデュアルバンクをベースとしたブートローダのメモリ設定については、このセクションで紹介します。MCUboot モジュールに精通しているユーザは、このセクションをスキップできます。

セクション 3：このアプリケーションプロジェクトに含まれるサンプルプロジェクトの概要について説明します。ユーザは、このセクションを参照して、サンプルプロジェクトの使用方法を理解する必要があります。

セクション 4：コードフラッシュデュアルバンク機能と MCUboot モジュールを使用してセキュアブートローダを作成する手順について説明します。ブートローダをカスタマイズするユーザは、このセクションを確認してブートローダがどのように構成されているかを理解する必要があります。

セクション 5：セクション 4 で作成されたブートローダを使用するようにアプリケーションを設定および署名する手順を説明します。このセクションでは、いくつかのサンプルプロジェクトを使用しています。初めてデュアルバンク機能を使用するユーザは、このセクションを理解する必要があります。

セクション 6：以下の 3 つに関して説明します。初めてデュアルバンク機能を使用するユーザは、このセクションを確認してください。

- プライマリアプリケーションのデバッグと起動
- プライマリイメージダウンローダーを使用して新しいイメージをダウンロード
- 新しいイメージの起動

セクション 7：ブートローダと最初のアプリケーションを使用して新しい MCU をプロビジョニングする製品化の方法について説明します。

セクション 8：付属のサンプルプロジェクトの実行方法について説明します。
MCUboot を使用したブートローダの設計に精通しているユーザは、このセクションにアクセスして、付属のサンプルプロジェクトを簡単に評価できます。

目次

1.　コードフラッシュメモリのデュアルバンク機能

RA6M4 および RA6M5 MCU グループの場合、内蔵のフラッシュメモリはリニアモードまたはデュアルモードで動作できます。リニアモードでは、コードフラッシュメモリを 1 つの領域として使用します。デュアルモードでは、コードフラッシュメモリは 2 つの領域に分割され、バンクスワップ機能を使用して、ブートローダを含むシステム用の新しいアプリケーションを起動することができます。

1.1　RA6M4 MCU グループ コードフラッシュ構成

1Mbyte 製品を使用する場合を例とすると、RA6M4 のリニアモードのコードフラッシュメモリには、図 1 に示すブロックが含まれます。

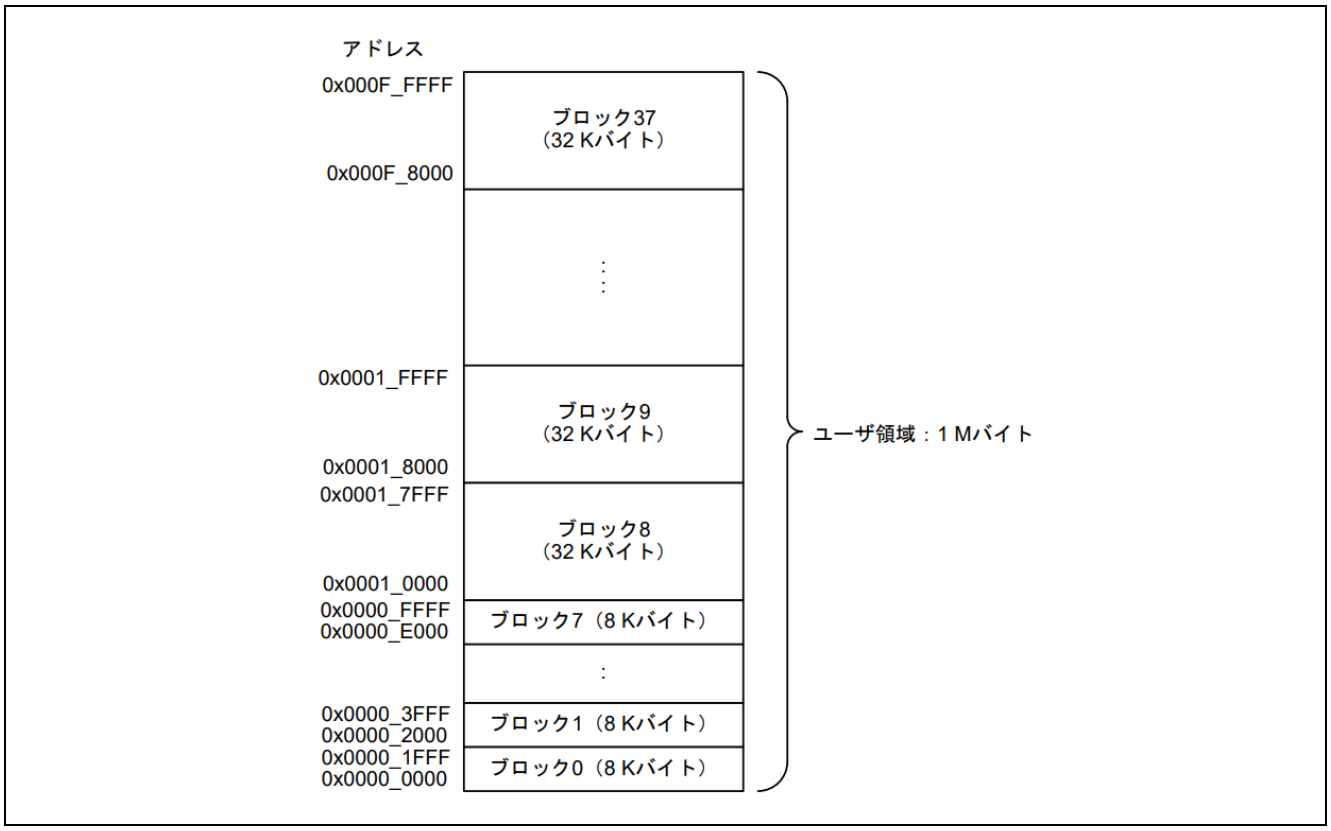

図 1　リニアモード時 RA6M4 コードフラッシュメモリ

1Mbyte 製品を使用する場合を例とすると、RA6M4 のデュアルモードのコードフラッシュメモリには、図 2 に示すブロックが含まれます。

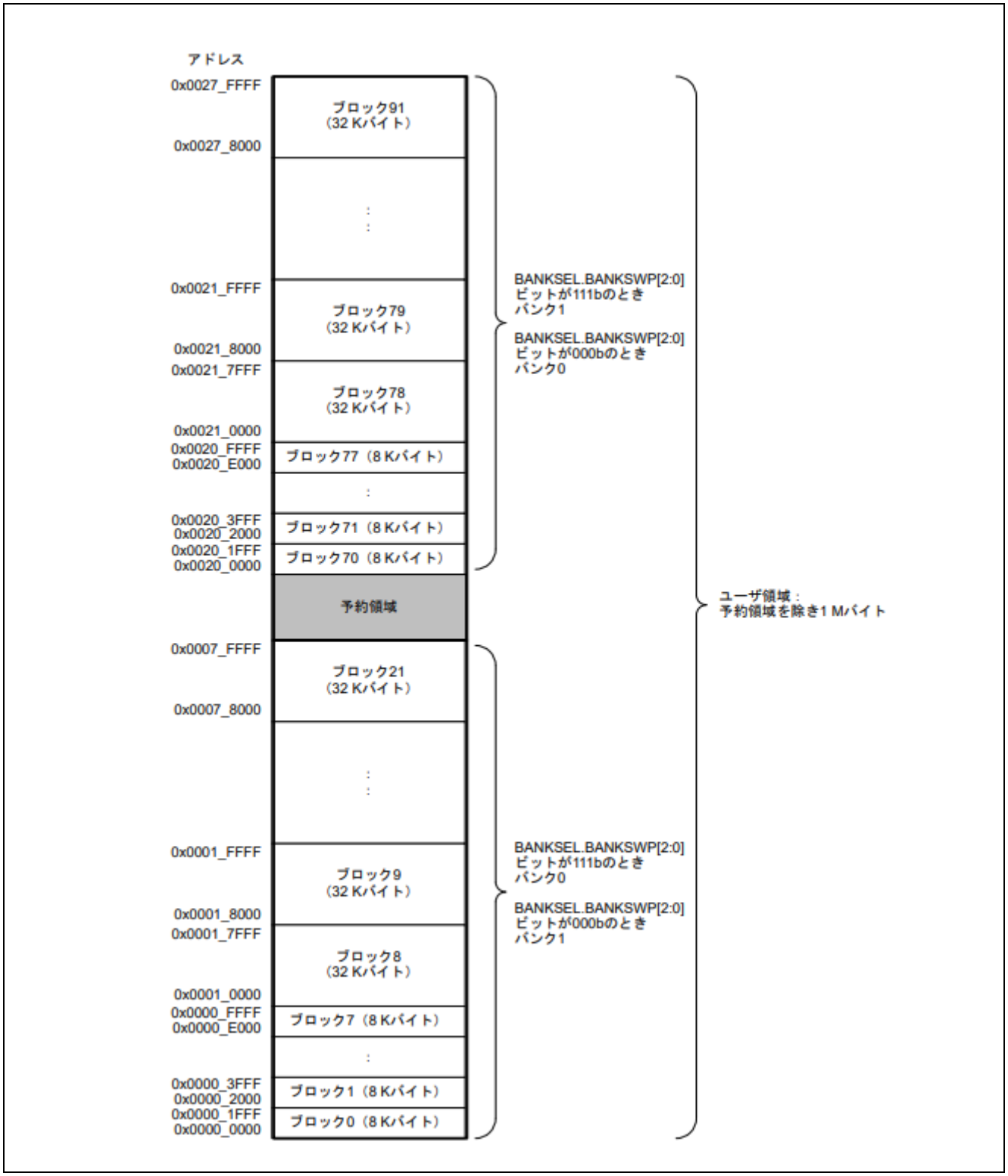

図 2　デュアルモード時 RA6M4 コードフラッシュメモリ

表 1 は、リニアおよびデュアルモードでのコードフラッシュブロックのアドレスです。

表 1　RA6M4 コードフラッシュ

製品	コードフラッシュ範囲アドレス	
	リニア	デュアル
1M バイト製品	0x0000_0000～0x000F_FFFF	下側バンク: 0x0000_0000～0x0007_FFFF
		上側バンク: 0x0020_0000～0x0027_FFFF
768K バイト製品	0x0000_0000～0x000B_FFFF	下側バンク: 0x0000_0000～0x0005_FFFF
		上側バンク: 0x0020_0000～0x0025FFFF
512K バイト製品	0x0000_0000～0x0007_FFFF	下側バンク: 0x0000_0000～0x0003_FFFF
		上側バンク: 0x0020_0000～0x0023_FFFF

図 3 は、RA6M4 のコードフラッシュのブロック構造です。コードフラッシュの消去と書き込みの最小単位は、コードフラッシュのブロックサイズです。ブロックの番号割り当ては、ブロック保護設計で使用されます。

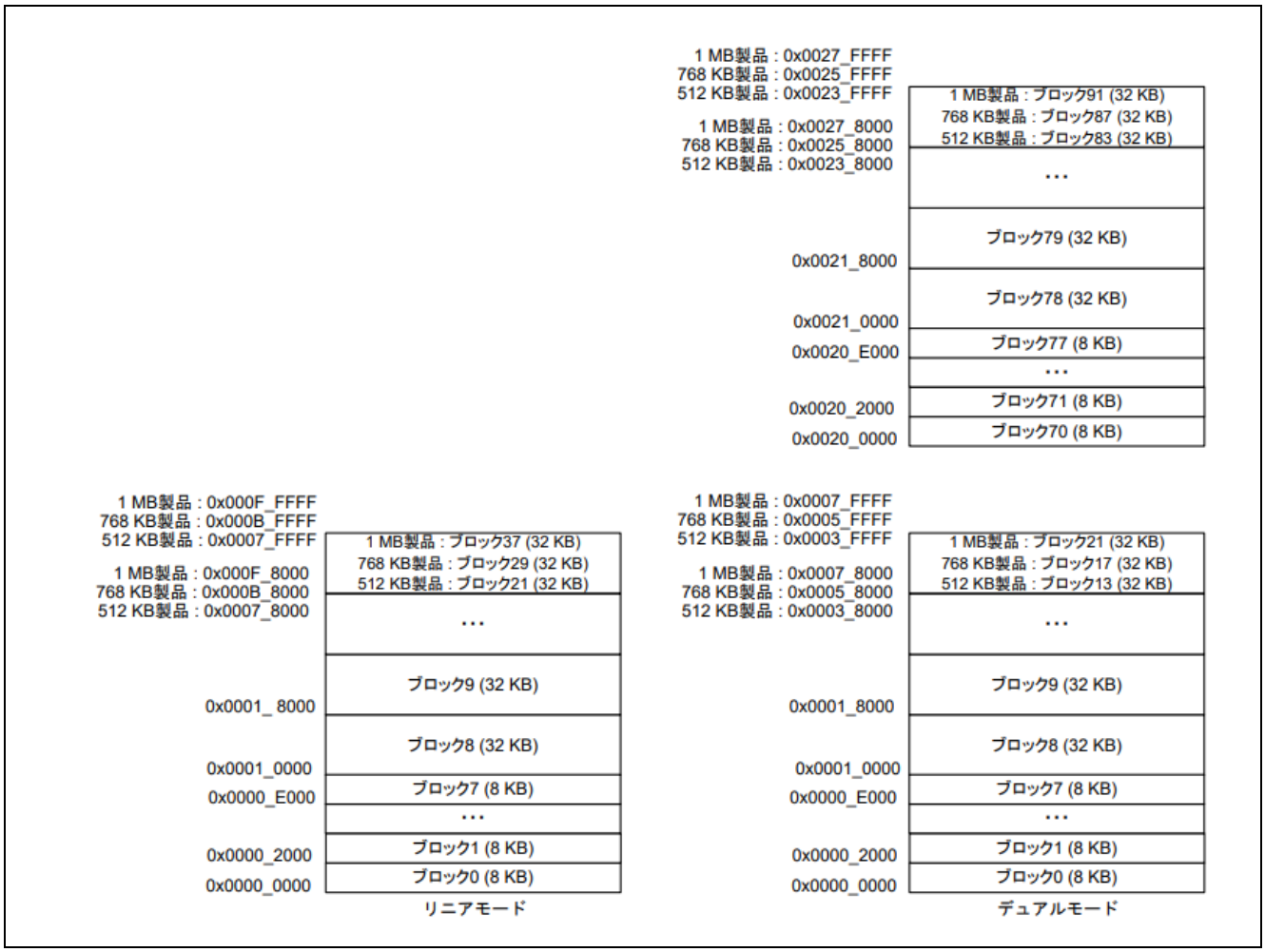

図 3　RA6M4 コードフラッシュブロック構造

1.2　RA6M5 MCU グループ コードフラッシュ構成

2Mbyte 製品を使用する場合を例とすると、RA6M5 のリニアモードのコードフラッシュメモリには、図 4 に示すブロックが含まれます。

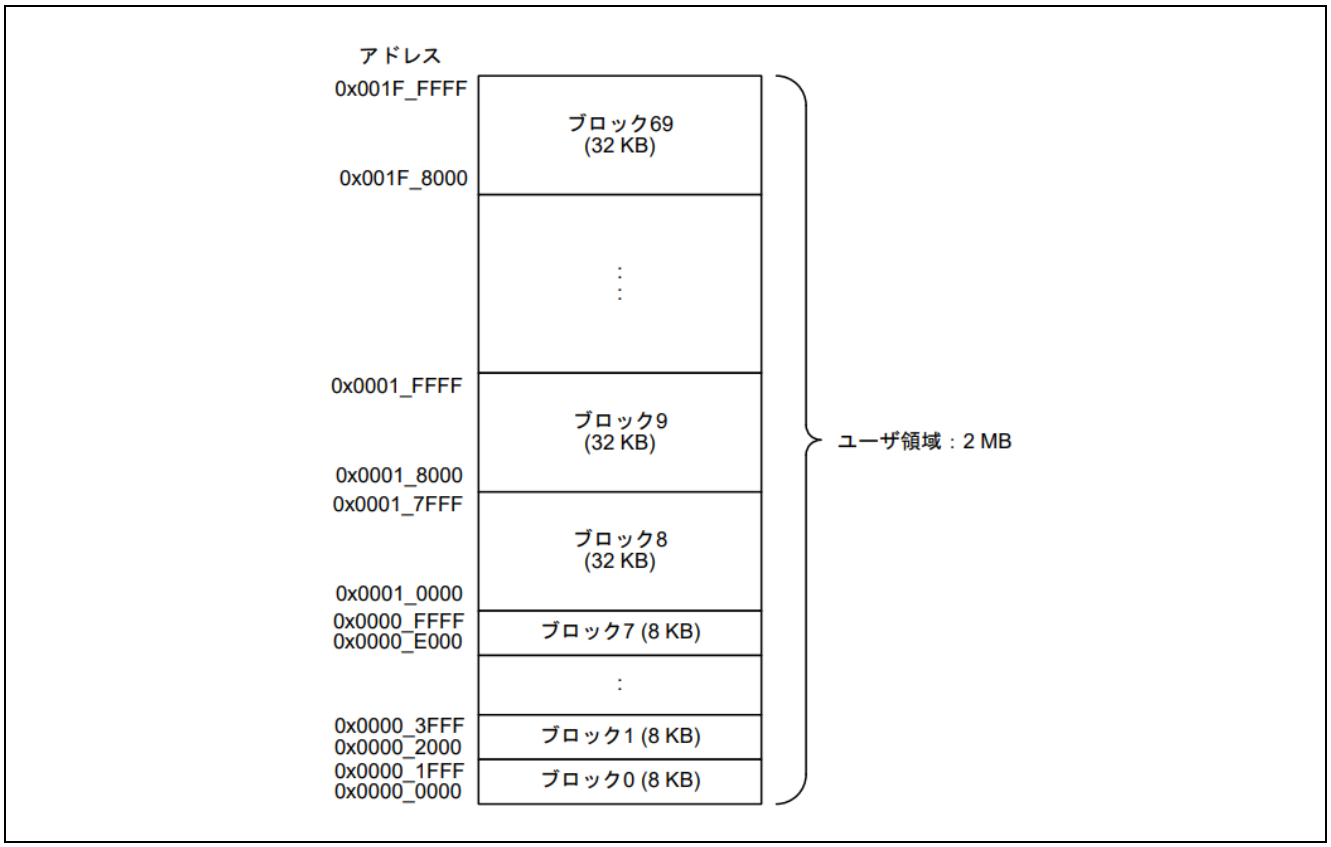

図 4　リニアモード時 RA6M5 コードフラッシュメモリ

2Mbyte 製品を使用する場合を例とすると、RA6M5 のデュアルモードのコードフラッシュメモリには、図 5 に示すブロックが含まれます。

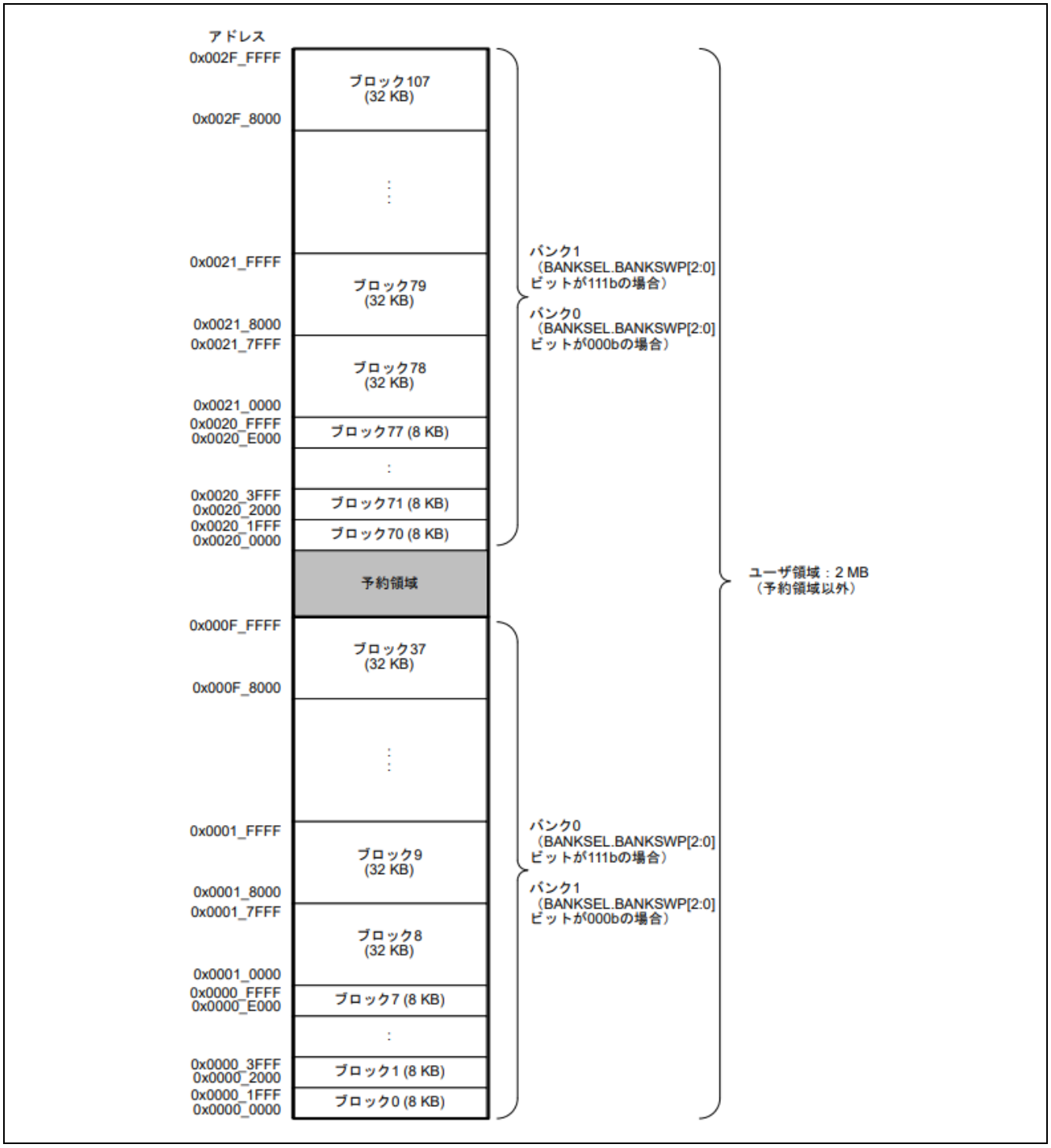

図 5　デュアルモード時 RA6M5 コードフラッシュメモリ

表 2 は、RA6M5 のリニアおよびデュアルモードのコードフラッシュブロックの概要です。

表 2　RA6M5 コードフラッシュ

製品	コードフラッシュ範囲アドレス	
	リニア	デュアル
2M バイト製品	0x0000_0000～0x001F_FFFF	下側バンク: 0x0000_0000～0x000F_FFFF
		上側バンク: 0x0020_0000～0x002F_FFFF
1M バイト製品	0x0000_0000～0x000F_FFFF	下側バンク: 0x0000_0000～0x0007_FFFF
		上側バンク: 0x0020_0000～0x0027_FFFF

図 6 は、RA6M5 のコードフラッシュのブロック構造です。コードフラッシュメモリの消去と書き込みの最小単位は、コードフラッシュのブロックサイズです。ブロックの番号割り当ては、ブロック保護設計で使用されます。

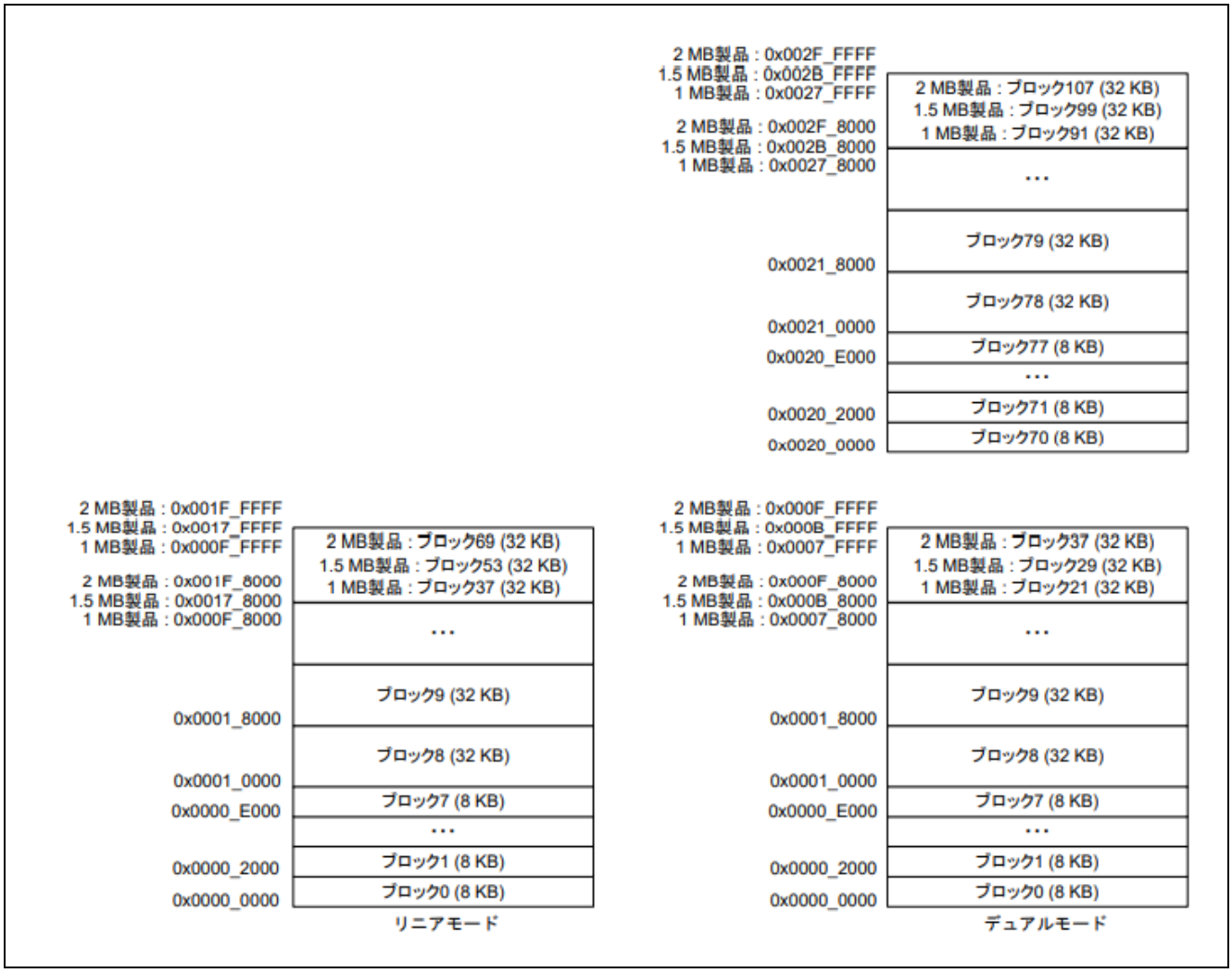

図 6　RA6M5 コードフラッシュブロック構造

1.3　オプション設定メモリ

このセクションは RA6M4 と RA6M5 の両方に適用されます。オプション設定メモリはリセット後の MCU の状態を決定します。このセクションではコードフラッシュのモードに関連するいくつかのプロパティ設定について説明します。

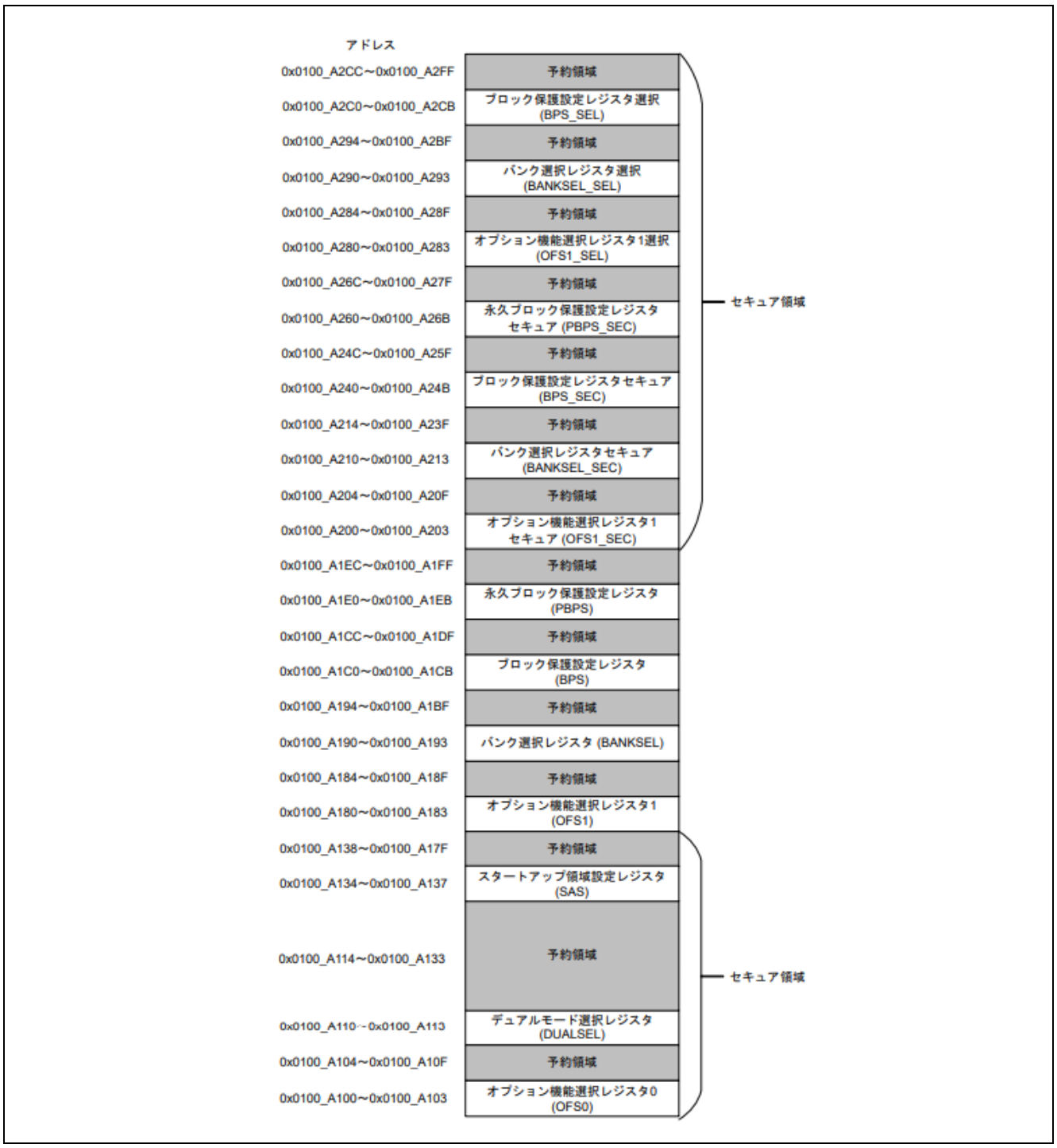

図 7　オプション設定メモリ

1.3.1　コードフラッシュバンクモード

コードフラッシュバンクモードを設定するレジスタは、オプション設定メモリにあります。図 7 に示すように、デュアルモード選択レジスタ DUALSEL は 0x0100A110 にあります。

DUALSEL レジスタは、コードフラッシュがリニアモードかデュアルモードかを定義します。ブランク MCU の場合、コードフラッシュはリニアモードです。ユーザアプリケーションは、リニア/デュアルモードの変更ができます。現在の FSP では、このレジスタはコンパイル時に[BSP]タブでプロパティを構成することによって設定されます。

6.2.2　DUALSEL : デュアルモード選択レジスタ

address: 0x0100_A110

Bit position:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset:	ユーザー設定値(注1)															

Bit position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	BANKMD[2:0]		
Value after reset:	ユーザー設定値(注1)															

ビット	シンボル	機能	R/W
2:0	BANKMD[2:0]	バンクモード選択 000: デュアルモード 111: リニアモード その他: 設定禁止	R
31:3	—	読んだ場合は、プログラムした値が読めます。書く場合、1 としてください。	R

注 1.　ブランク品は、0xFFFF_FFFF です。ユーザーがプログラムした値になります。

BANKMD[2:0]ビット（バンクモード選択）

BANKMD[2:0]ビットは、コードフラッシュメモリのデュアルバンク機能のバンクモードを選択します。

図 8　コードフラッシュデュアルモードレジスタ設定

1.3.2　起動バンクの選択

このセクションは RA6M4 と RA6M5 の両方に適用されます。バンク 0 はブランクの RA6M4 および RA6M5 MCU 用の下側バンクで、以下のレジスタで定義されています。

6.2.5　BANKSEL, BANKSEL_SEC, BANKSEL_SEL : バンク選択レジスタ

Address: BANKSEL: 0x0100_A190
BANKSEL_SEC: 0x0100_A210
BANKSEL_SEL: 0x0100_A290

Bit position:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit field:	—	—	—	—	—	—	—	—	BLCKSWP[7:0]							
Value after reset:	ユーザー設定値(注1)															

Bit position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	BANKSWP[2:0]		
Value after reset:	ユーザー設定値(注1)															

ビット	シンボル	機能	R/W
2:0	BANKSWP[2:0]	スタートアップバンク切り替え 本設定はデュアルモードで有効です。 000: デュアルモードでは、Bank0 の開始アドレスは 0x0020_0000、Bank1 の開始アドレスは 0x0000_0000 111: デュアルモードでは、Bank0 の開始アドレスは 0x0000_0000、Bank1 の開始アドレスは 0x0020_0000 その他: 設定禁止	R/W
15:3	—	読んだ場合は、プログラムした値が読めます。書く場合、1 としてください。	R/W
23:16	BLCKSWP[7:0]	ブロックスワップ選択 全ビットとも 1 の場合ブロックスワップは無効です。1 つでも 0 に設定されているビットがあると、ブロックスワップは有効で、コードフラッシュメモリの対応ブロックがスワップされます。	R/W
31:24	—	読んだ場合は、プログラムした値が読めます。書く場合、1 としてください。	R/W

注 1.　ブランク品は、0xFFFF_FFFF です。ユーザーがプログラムした値になります。

図 9　バンク 0 開始アドレス(デフォルト 0x00000000)

セキュア開発者のみが BANKSEL_SEC レジスタおよび BANKSEL_SEL レジスタを書き換え可能です。BANKSEL_SEC レジスタはセキュア開発者用で、BANKSEL レジスタは非セキュア開発者用です。

BANKSEL_SEL は、BANKSEL と BANKSEL_SEC のどちらの設定を適用するかを制御します。BANKSEL_SEL が 0xFFFFFFF8 の場合、BANKSEL の設定が使用されます。BANKSEL_SEL が 0xFFFFFFFF の場合、BANKSEL_SEC の設定が使用されます。Non-Trust Zone ベースの Flat プロジェクトの場合、BANKSEL_SEL は BANKSEL_SEC レジスタの対応するビットを選択します。

1.3.3　バンクスワップ

デュアルモード時にバンク 0,1 どちらで起動するかは、リセット時の BANKSWP[2:0]により決定され、プログラムを安全にアップデートできます。

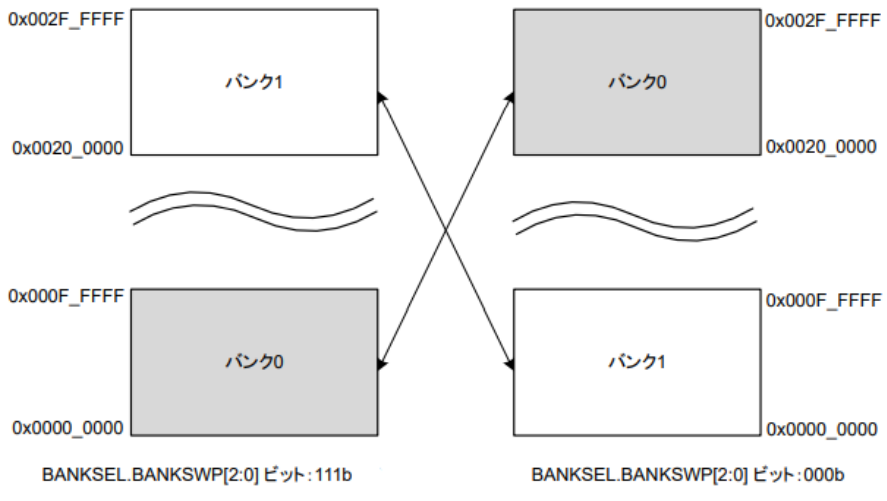

図 10　起動バンク選択例(1M バイトのコードフラッシュメモリを搭載した製品の場合)

バンクの選択は、FSP API を介してプログラム実行時に変更できます。BANKSEL レジスタの BANKSWP ビットは、アプリケーションレベルで変更できます。FSP フラッシュドライバは、このアクションを容易にするために `R_FLASH_HP_BankSwap()` API を提供します。この API は、FSP MCUboot モジュールから自動的に呼び出されます。バンクの切り替えは、次のリセット後に有効になります。

1.3.4　コードフラッシュブロック保護

RA6M4 および RA6M5 は、コードフラッシュを不正な改ざんやフラッシュメモリのデータの読み出しから保護するためのセキュリティ機能を実装しています。このセキュリティ機能を定義するレジスタは、オプション設定メモリにあります。本設定により、コードフラッシュメモリは書き込み/消去操作から一時的または永久に保護することができます。

一時的にコードフラッシュのブロック保護を行うレジスタは、オプション設定メモリにあります。

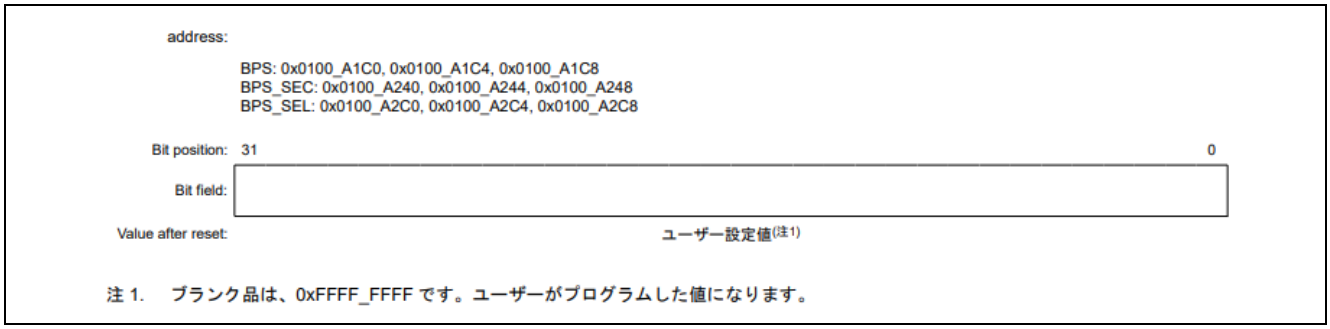

図 11　コードフラッシュブロック一時保護関連設定レジスタ

セキュア開発者のみが BPS_SEC レジスタおよび BPS_SEL レジスタを書き換え可能です。BPS_SEC レジスタはセキュア開発者用で、BPS レジスタは非セキュア開発者用です。適用される設定値は BPS_SEL レジスタの対応ビットの設定値によって決まります。BPS_SEL は、BPS と BPS_SEC のどちらの設定を適用するかを制御します。BPS_SEL が 0xFFFFFFF8 の場合は、BPS の設定が使用されます。BPS_SEL が 0xFFFFFFFF の場合は、BPS_SEC の設定が使用されます。Non-Trust Zone ベースの Flat プロジェクトの場合、BSP_SEL は BSP_SEC レジスタの対応するビットを選択します。BPS および BPS_SEC レジスタは、コードフラッシュメモリへの書き込みと消去を無効にします。レジスタのビットが 0 に設定されている場合、対応するブロックへの書き込みと消去は無効です。

これらのレジスタは、図 87 および図 88 に示すように、RA コンフィギュレータで BSP プロパティを設定することによって設定できます。

永久にコードフラッシュのブロック保護を行うレジスタは、オプション設定メモリにあります。

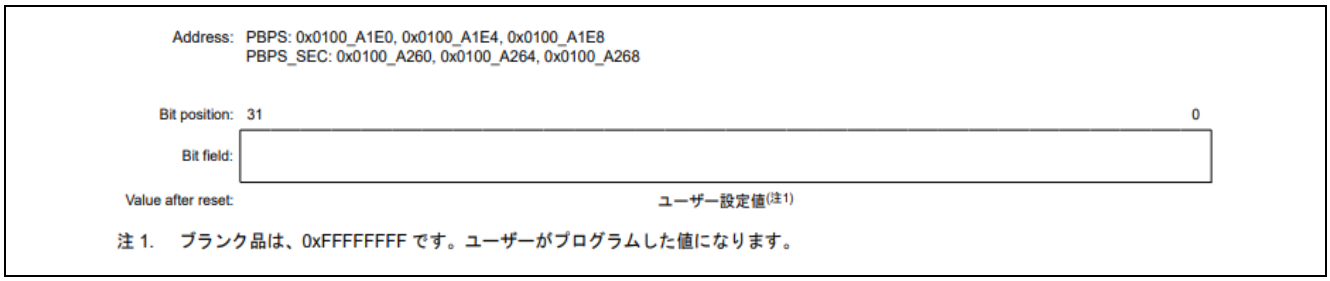

図 12　コードフラッシュブロック永久保護関連レジスタ

セキュア開発者のみが PBPS_SEC と PBPS_SEL レジスタを書き換え可能です。PBPS_SEC レジスタはセキュア開発者用で、PBPS レジスタは非セキュア開発者用です。適用された設定値は、PBPS_SEL レジスタの対応するビットの設定値によって決定され、PBPS_SEL は、BPS または BPS_SEC 設定を適用するかどうかを制御します。PBPS_SEL が 0xFFFFFFF8 の場合、PBPS の設定が使用されます。PBPS_SEL が 0xFFFFFFFF の場合、PBPS_SEC の設定が使用されます。Non-Trust Zone ベースの Flat プロジェクトの場合、PBSP_SEL は PBSP_SEC レジスタの対応するビットを選択します。PBPS および PBPS_SEC レジスタは、コードフラッシュメモリへの書き込みと消去を無効にします。このレジスタのビットが 0 に設定されている場合、対応するブロックへの書き込みと消去は無効です。

これらのレジスタの設定は、図 89 および図 90 に示すように、RA コンフィギュレータで BSP プロパティを設定することによって実現できます。

2.　コードフラッシュデュアルバンク機能と MCUboot の使用方法の概要

MCUboot は、runtime.io によって作成された Apache Mynewt ブートローダから進化しました。MCUboot はその後、2018 年 11 月に JuulLabs により買収されました。MCUboot の github リポジトリは、後に JuulLabs から mcu-tools github project に移行されました。2020 年、MCUboot はオープンソースプロジェクトとして Linaro Community Project の傘下に移されました。

2.1　MCUboot の概要

MCUboot は、起動後のファームウェアの信頼性チェックとファームウェアの更新プロセスのファームウェア切り替えの一部を処理します。新しいバージョンのファームウェアをダウンロードすることは、MCUboot の範囲外です。通常、新しいバージョンのファームウェアのダウンロードは、アプリケーションプロジェクト自体に実装される機能です。このアプリケーションプロジェクトは、アプリケーションプロジェクトから XModem プロトコルを使用して新しいイメージをダウンロードする例を実装しています。

起動および更新中の MCUboot の機能は、次のプロセスに従います。

CPU がリセットから解除されると、ブートを開始します。更新するように指定されたセカンダリアプリメモリにイメージがある場合、ブートローダは次のアクションを実行します。

1. ブートローダはセカンダリイメージを認証します。
2. 認証に成功すると、ブートローダは選択されたアップデート方法に基づいて新しいイメージを更新します。FSP でサポートされている更新方法は、上書き、スワップ、Direct XIP です。
3. ブートローダが新しいイメージを起動します。

ここで、セカンダリアプリメモリ領域に新しいイメージがない場合、ブートローダはプライマリアプリケーションを認証し、プライマリイメージを起動します。

アプリケーションの認証は、認証の方法と、MCUboot を使用して認証を実行するかを設定できます。認証で使用可能なアルゴリズムは、RSA または ECDSA です。ファームウェアイメージは、ハッシュ(SHA-256)とデジタル署名の検証によって認証されます。デジタル署名の検証に使用される公開鍵は、ブートローダに組み込むことも、製造時に MCU に書き込み、提供することもできます。このアプリケーションプロジェクトに含まれている例では、公開鍵はブートローダに組み込まれています。

MCUboot には署名ツールの `imgtool.py` が含まれています。このツールは、ルートキーの作成、キー管理、およびバージョンコントロールを使用したイメージの署名とパッケージ化のためのサービスを提供します。これらの操作を使用および理解するには、MCUboot のドキュメントを参照してください。

2.2　コードフラッシュデュアルモードでの MCUboot の使用方法

FSP は、上書き、スワップ、および Direct XIP (execute-in-place)更新モードをサポートしています。フラッシュデュアルモードでは、Direct XIP モードのみがサポートされます。ブートローダを含むシステムでコードフラッシュデュアルモードを使用する利点は、セクション 2.2.1 で説明されているように、MCUboot モジュールに実装されている安全機能に加えて、新しいイメージの同時ダウンロードと新しいイメージへのより高速な切り替えです。

2.2.1　Direct XIP ファームウェア更新モードの使用

リニアモードのコードフラッシュで Direct XIP モードを使用する場合、アクティブなイメージスロットはファームウェアの更新ごとに切り替わります。この更新方法を使用する場合、2 つのファームウェアを更新するイメージを生成する必要があります。1 つはプライマリスロットメモリ領域から実行されるようにリンクされ、もう一方はセカンダリスロットから実行されるようにリンクされます。Direct XIP は FSP v3.6.0 からサポートされています。

- 長所
  - アプリケーションイメージの上書きや交換が不要なため、起動時間が短縮されます。
  - フェイルセーフかつ電源断に耐性があります。
- 短所
  - どのファームウェアイメージをダウンロードする必要があるかを判断するため、アプリケーションのレベルが複雑化します。
  - 暗号化されたイメージはサポート対象外です。

他の更新モードの概要と使用法については、R11AN0497 と以下の MCUboot 設計のページを参照してください。

https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md

デュアルモードのコードフラッシュで Direct XIP モードを使用する場合、プライマリイメージとセカンダリイメージの両方がリンクされ、プライマリスロットメモリ領域から実行されます。なお、フラッシュデュアルモードで RA6M4 および RA6M5 MCU を使用する場合、現在の FSP v3.6 でサポートされているのは、Non-Trust Zone ベースの Flat プロジェクトのみです。

Direct XIP モードの場合、MCUboot 側でダウングレードの防止がサポートされていることに注意してください。フラッシュデュアルモードを使用する場合、更新イメージのバージョン番号は現在のプライマリイメージよりも高くする必要があります。

2.2.2　デュアルバンクおよび MCUboot を使用したメモリ構成の概要

図 13 に示すように、フラッシュデュアルモードの FSP MCUboot モジュールには、下側バンクと上側バンクの両方にブートローダが必要です。さらに、ブートローダとアプリケーションイメージのメモリ割り当ては同一である必要があります。

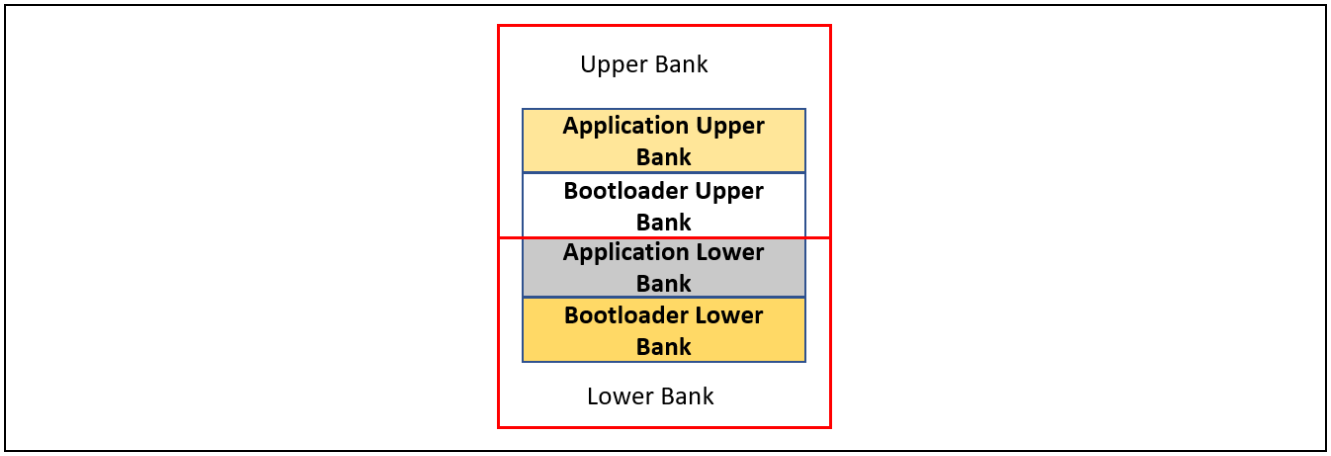

図 13　フラッシュデュアルモード及び MCUboot 使用時のメモリアーキテクチャ

2.3　ブートローダの設計とプライマリアプリケーションの概要

ブートローダは通常、最初に書き込みするプライマリアプリケーションと一緒に設計されます。以下は、ブートローダとプライマリアプリケーションを設計するための一般的なガイドラインです。

- ブートローダを開発し、ブートローダとアプリケーションに必要な MCU メモリリソースの割り当てを検討します。ブートローダのメモリ使用量は、ブートローダのメモリ使用量は、アプリケーションイメージの更新モード、署名の種類、プライマリイメージを検証するかどうか、および使用する暗号化ライブラリによって影響を受けます。
- 最初に書き込みするプライマリアプリケーションを開発し、メモリ使用量を検討して、ブートローダのメモリ割り当てと比較して整合性を確認し、必要に応じて調整してください。
- イメージ認証と新しいイメージ更新モードの観点から、ブートローダの構成を決定します。これにより、ブートローダプロジェクトでメモリ割り当ての定義が調整される可能性があります。
- アプリケーションイメージに署名します。署名コマンドは、<bootloader project>\Debug\>bootloader project>.bld ファイルに出力されます。アプリケーションイメージは Build Variable を使用して、この.bld

ファイルにアクセスできます。IDE ツールは署名コマンドを使用してアプリケーションに署名し、MCU にダウンロードするためのバイナリファイルを生成します。

- ブートローダと最初のプライマリアプリケーションをテストします。

上記のガイドラインは、本アプリケーションノートのウォークスルーセクションに示されています。

3.　付属のサンプルプロジェクトの使用に関するガイドライン

RA6_Secure_Bootloader_DualBank.zip を解凍し、本アプリケーションノートのサンプルプロジェクトを展開します。

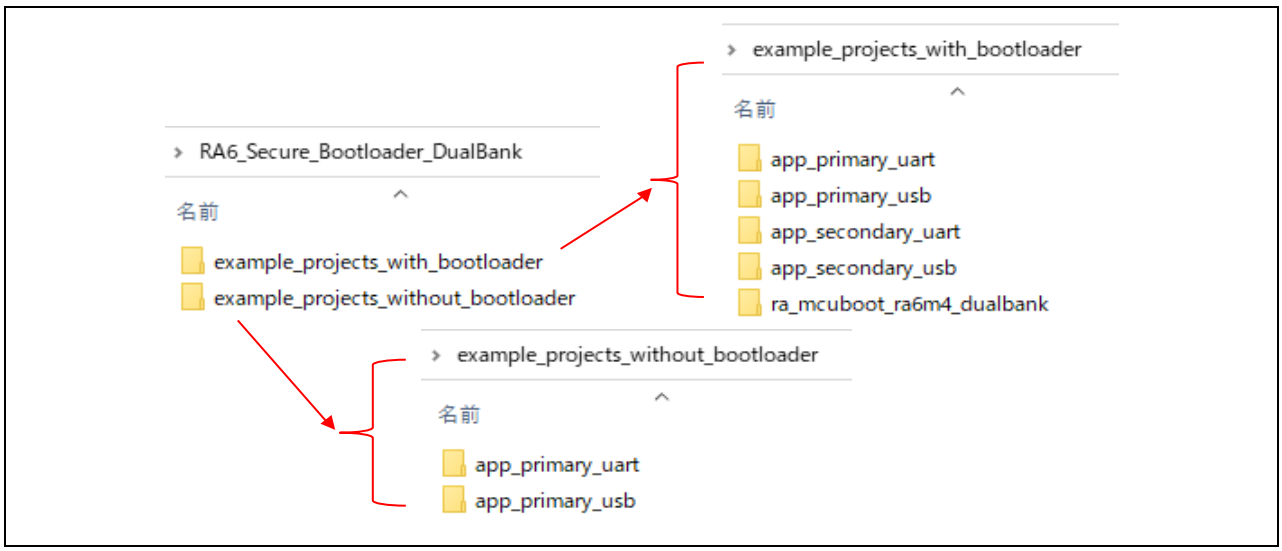

図 14　付属のサンプルプロジェクト

3.1　ブートローダを使用したプロジェクト例

example_projects_with_bootloader フォルダには、フラッシュデュアルバンク機能をサポートするブートローダと、このフォルダに含まれるブートローダを使用するように構成された新しいアプリケーションイメージをダウンロードするための通信チャネルとして USB または UART を使用するサンプルアプリケーションが含まれています。MCUboot モジュールを使用した経験を持つユーザは、セクション 8 に従って、これらのサンプルプロジェクトを直接実行できます。

- ra_mcuboot_ra6m4_dualbank:デュアルモードと Direct XIP 更新モードを有効にするブートローダです。
- app_primary_usb:USB VCOM 経由で新しいアプリケーションイメージをダウンロードするため、XModem を実装しているプライマリアプリケーションです。FreeRTOS は 2 つのスレッドと使用され、1 つのスレッドが EK-RA6M4 の 3 つの LED を点滅させ、もう一方のスレッドは新しいアプリケーションイメージを並行してダウンロードします。
- app_secondary_usb:app_primary_usb と同じ機能を実装したセカンダリアプリケーションで、青と緑の LED のみを点滅させます。
- app_primary_uart:UART 経由で新しいアプリケーションイメージをダウンロードするため、XModem を実装しているプライマリアプリケーションです。FreeRTOS は 2 つのスレッドと使用され、1 つのスレッドが EK-RA6M4 の 3 つの LED を点滅させ、もう一方のスレッドは新しいアプリケーションイメージを並行してダウンロードします。
- app_secondary_uart:app_primary_uart と同じ機能を実装したセカンダリアプリケーションで、青と緑の LED のみを点滅させます。

3.2　ブートローダを使用しないプロジェクト例

example_projects_without_bootloader フォルダには、セクション 5 に従って、ユーザがブートローダプロジェクトを使用するように構成できるスタンドアロンのサンプルプロジェクトが含まれています。フラッシュデュアルモードが有効でない場合、これらのアプリケーションプロジェクトは正しく実行されないことに注意してください。これは、付属のイメージダウンローダールーチンが新しいイメージの場所を RA6M4 の上位バンクにあると想定しているためです。

- app_primary_usb:\example_projects_with_bootloader\app_primary_usb と同じ機能ですが、ブートローダで動作するように設定されていない点が異なります。
- app_primary_uart:\example_projects_with_bootloader\app_primary_uart と同じ機能ですが、ブートローダで動作するように設定されていない点が異なります。

ブートローダを使用するために、イメージのダウンロードを実装し、セクション 5 に従って独自のアプリケーションプロジェクトを使用することもできます。

4.　コードフラッシュデュアルモードを使用したブートローダプロジェクト作成

このセクションでは、MCUboot と RA6M4 が Non-Trust Zone モードで動作しているフラッシュデュアルモードを使用したブートローダプロジェクトの作成手順について説明します。

4.1　ブートローダプロジェクトに MCUboot モジュールを組み込む

以下の手順でブートローダプロジェクトの作成を開始し、MCUboot モジュールをプロジェクトに組み込みます。

1. e² studio を起動し、C/C++プロジェクトの新規作成を開始します。[ファイル] > [新規] > [C/C++ Project]をクリックします。

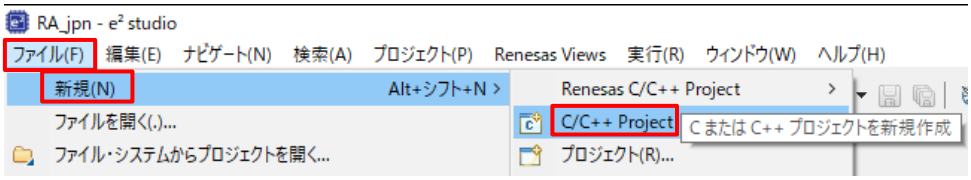

図 15　新規プロジェクト開始

2. [Renesas RA] > [Renesas RA C/C++ Project]を選択します。[次へ]をクリックします。

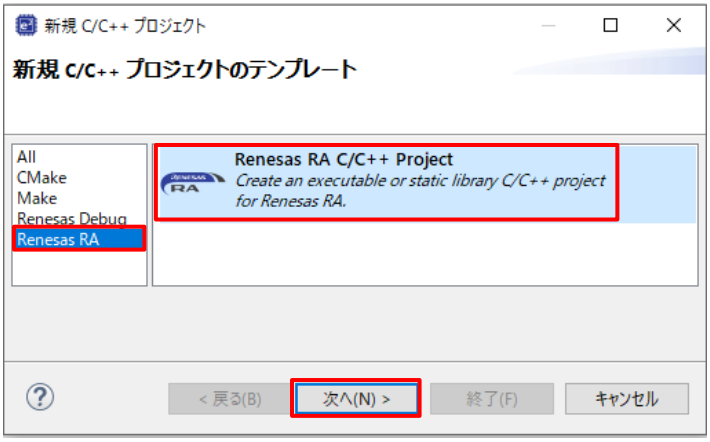

図 16　Renesas RA C/C++ Project 選択

3. 次の画面で[Project name]に `ra_mcuboot_ra6m4_dualbank` を入力し、[次へ]をクリックします。
4. 次の画面で[Board]で[EK-RA6M4]を選択し、[次へ]をクリックします。

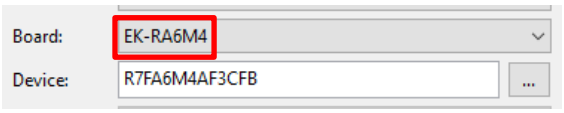

図 17　ボード選択

5. [Build Artifact]で[Executable]と、[No RTOS]を選択します。[次へ]をクリックします。

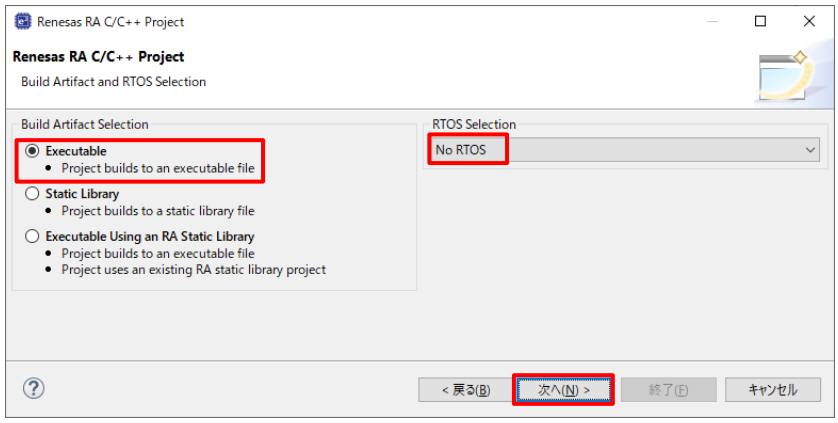

図 18　Build Executable と No RTOS 選択

6. 次の画面で、[Project Template]で[Bare Metal - Minimal]を選択し、[終了]をクリックして初期プロジェクトを作成します。

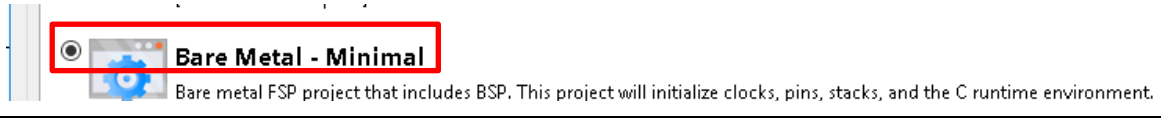

図 19　Project Template 選択

7. 以下のポップアップが表示されたら、[パースペクティブを開く]をクリックします。

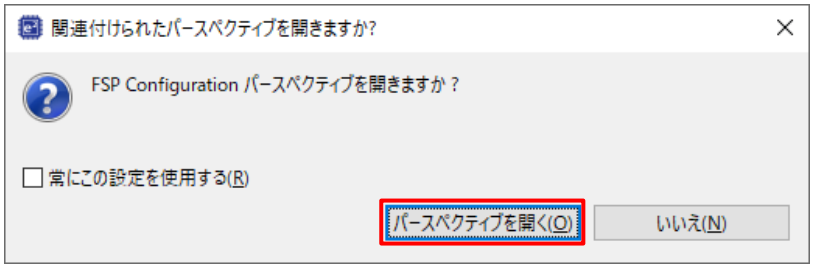

図 20　FSP Configuration パースペクティブ表示選択

これでプロジェクトが作成され、ブートローダプロジェクトの構成が表示されます。

8. [Pins]タブを選択し、[RA6M4 EK]の[Generate data]のチェックを外します。

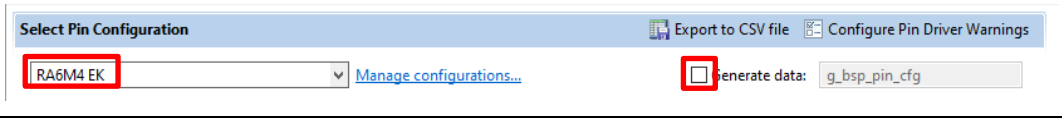

図 21　RA6M4 EK Pin Configuration Generate data 非選択

プルダウンメニューで[Select Pin Configuration]を[RA6M4 EK]から[R7FA6M4AF3CFB.pincfg]に切り替え、[Generate data]のチェックを入れて[g_bsp_pin_cfg]と入力してください。ブートローダは RA6M4 EK コンフィギュレーションで設定された余分なペリフェラルや GPIO ピンを使用しないため、ここでは設定されたペリフェラル/ピンの数が少ないこの構成を選択することに注意してください。これにより、ブートローダプロジェクトのメモリ使用量も多少削減されます。

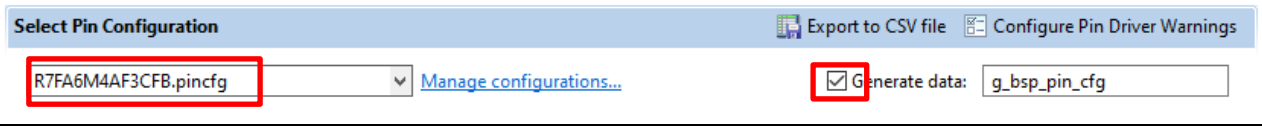

図 22　g_bsp_pin_cfg 生成

9. プロジェクトが作成されたら、RA コンフィギュレータの[Stacks]タブをクリックします。[New Stack] > [Bootloader] > [MCUboot]を追加します。

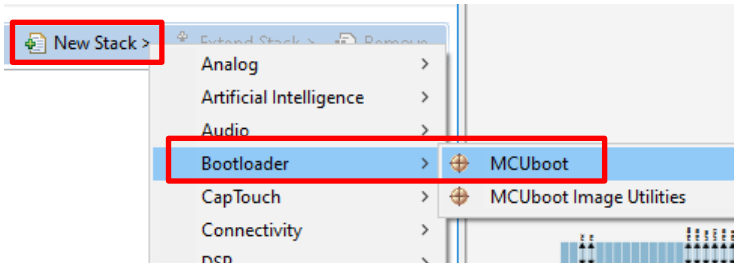

図 23　MCUboot Port 追加

10. 次に、[MCUboot]の[General]プロパティを設定します。次の手順でコンフィギュレータのエラーを解決していきます。
[MCUboot]モジュールの場合、[Upgrade Mode]を[Direct XIP]に、[Number of Images Per Application]を[1]に設定します。

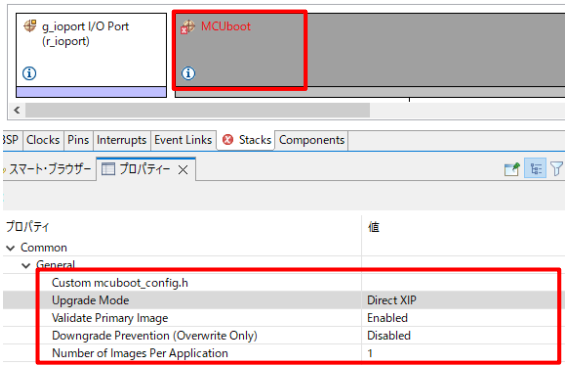

図 24　MCUboot モジュール一般設定

以下に、設定される各種プロパティについて説明します。

- **Custom mcuboot_configh**：デフォルトの mcuboot_config.h ファイルには、ユーザが RA コンフィギュレータから選択した MCUboot モジュールの設定が含まれています。このファイルのカスタムバージョンを作成して、MCUboot で利用可能な追加のブートローダ機能を実現することができます。
- **Upgrade Mode**：このプロパティは、アプリケーションイメージのアップグレード方法を設定します。利用可能なオプションは、Overwrite Only、Overwrite Only Fast、Swap、および Direct XIP です。フラッシュデュアルバンク動作では、Direct XIP のみがサポートされます。
- **Validate Primary Image**：有効にすると、ブートローダは、MCUboot マジックナンバーベースのサニティチェックに加えて、選択した検証方法に応じてハッシュまたは署名の検証を実行します。無効の場合、MCUboot マジックナンバーベースのサニティチェックのみが実行されます。
- **Number of Images Per Application**：このプロパティでは、Non-Trust Zone ベースのアプリケーションには 1 つのイメージを、Trust Zone ベースのアプリケーションには 2 つのイメージを選択することができます。このプロパティは 1 に設定します。
- **Downgrade Prevention (Overwrite Only)**：このプロパティは、Overwrite によるアップグレードモードにのみ適用されます。このプロパティが有効の場合、バージョン番号の低い新しいファームウェアは、既存のアプリケーションを上書きしません。

Direct XIP モードの場合、MCUboot 側でダウングレード防止がサポートされていることに注意してください。フラッシュデュアルモードを使用する場合、更新イメージは現在のプライマリイメージよりも高いバージョン番号にする必要があります。

4.2　メモリ構成と認証方法の設定

MCUboot モジュールの署名オプションとフラッシュレイアウトを設定します。EK-RA6M4 では、コードフラッシュデュアルモードのデフォルトメモリは下図の通りです。このデフォルトメモリマップは、ブートローダの設計例で使用します。

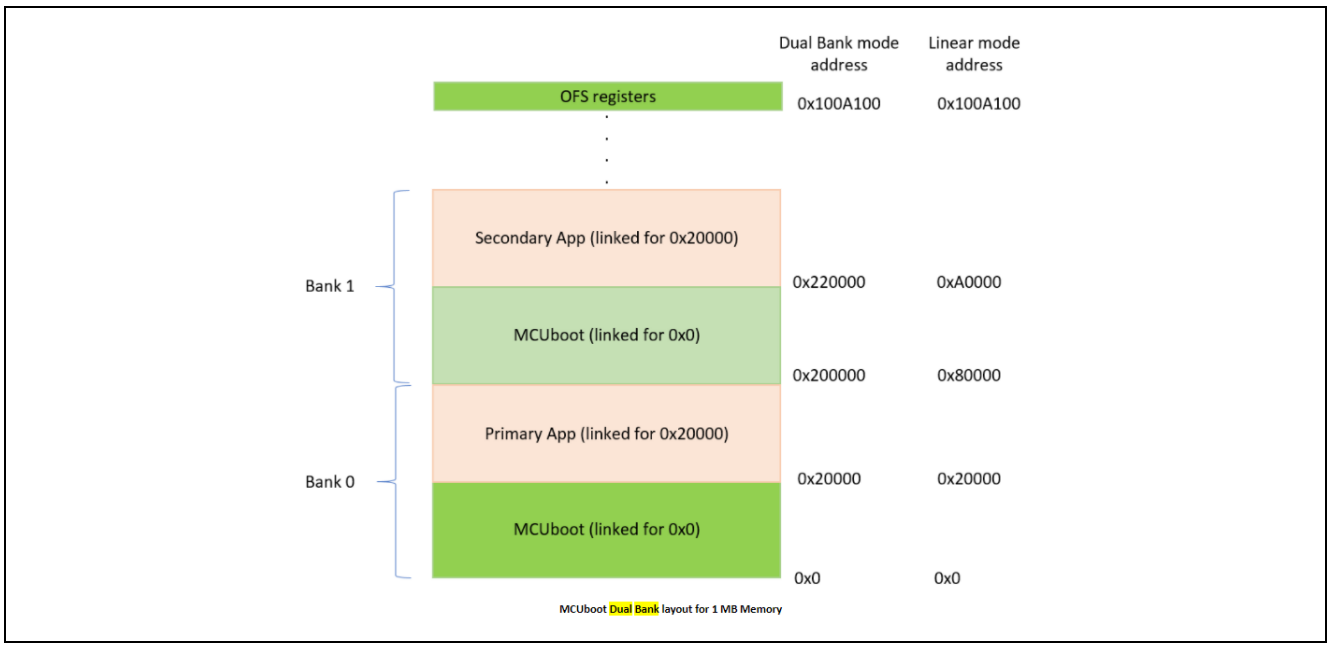

図 25　MCUboot デュアルバンクメモリマップ

コンフィギュレータの観点から、Flash レイアウトのプロパティは図 25 に示すメモリマップと既に一致しているため、変更する必要はありません。

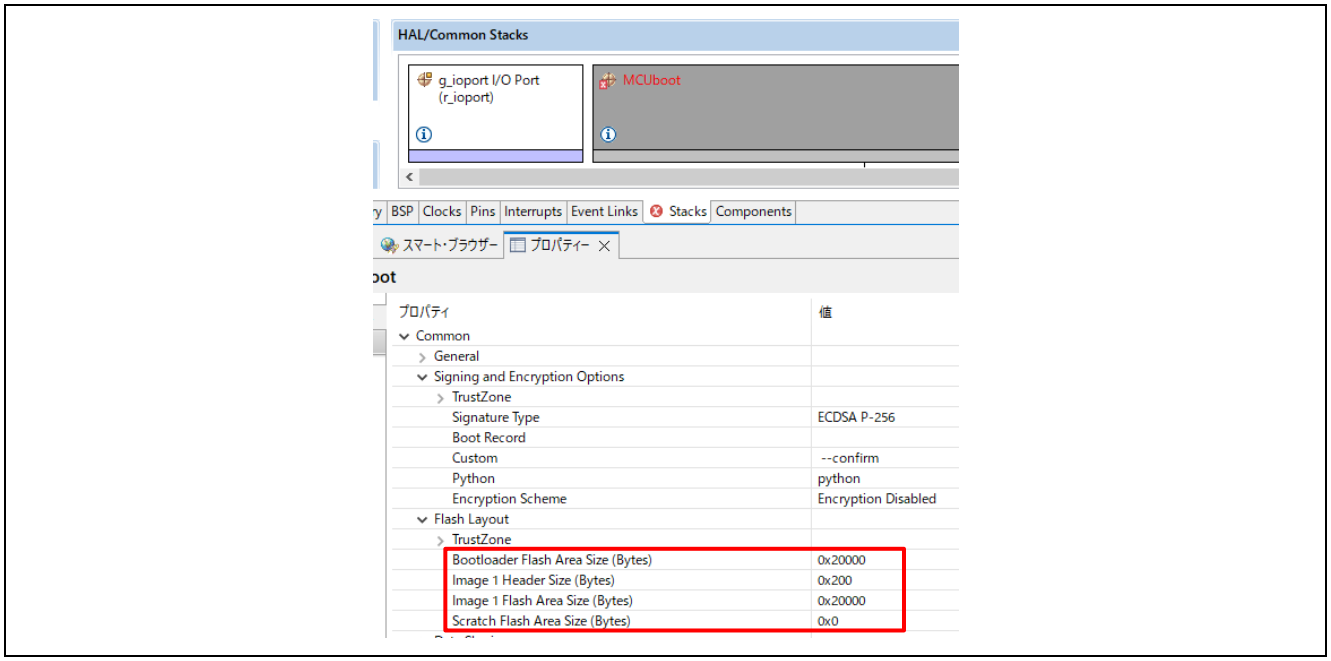

図 26　Flash レイアウトと署名オプション設定

以下に、設定される各種プロパティについて説明します。

- **Bootloader Flash Area**：RA6M4 コードフラッシュの最小消去サイズが 0x8000 であるため、ブートローダ用に割り当てられたフラッシュ領域のサイズは 0x8000 を境界とします。
- **Image 1 Header Size**：アプリケーションイメージヘッダー用に予約されているコードフラッシュのサイズです。RA6M4 に実装されている割り込みの数に応じて、VTOR アライメントの最小要件を満たす必要があります。RA6M4 の場合、すべての割り込みをサポートするには、このプロパティは最小 0x200 で設定する必要があります。
- **Image 1 Flash Area Size**：ヘッダーとトレーラーを含むアプリケーションイメージ 1 のサイズです。RA6M4 の場合、このサイズはコードフラッシュの最小消去サイズである 0x8000 区切りにする必要があります。
- **Scratch Flash Area Size**：このプロパティは、スワップモードにのみ必要です。このプロパティは、フラッシュデュアルバンクブートローダの設計には使用されません。
- **Signature Type**：署名アルゴリズムを選択します。選択肢は次のとおりです。
  - **None**：署名検証をサポートしていないブートローダには、このオプションを選択します。
  - **ECDSA P-256**：このブートローダ設計例では、このオプションを選択します。
  - **RSA 2048** と **RSA 3072**：
    MCUboot を使用するアプリケーションイメージは、MCUboot で動作するよう署名する必要があります。これには最低限、イメージのトレーラーにハッシュと MCUboot 固有の定数値を追加する必要があります。
- **Custom**：このブートローダの設計には、デフォルトの--confirm を使用します。新しいイメージへの切り替えは常に確認され、その後のシステムリセット後に新しいイメージが起動されます。Direct XIP を使用したイメージの復元は、この FSP リリースバージョンではサポートされていません。

4.3　MbedTLS 暗号専用モジュールとフラッシュドライバの設定

以下の手順に従って、MbedTLS モジュールとフラッシュドライバの設定を行います。

1. [Add Crypto Stack]を右クリックし、[MbedTLS (Crypto Only)]モジュールの追加を選択します。

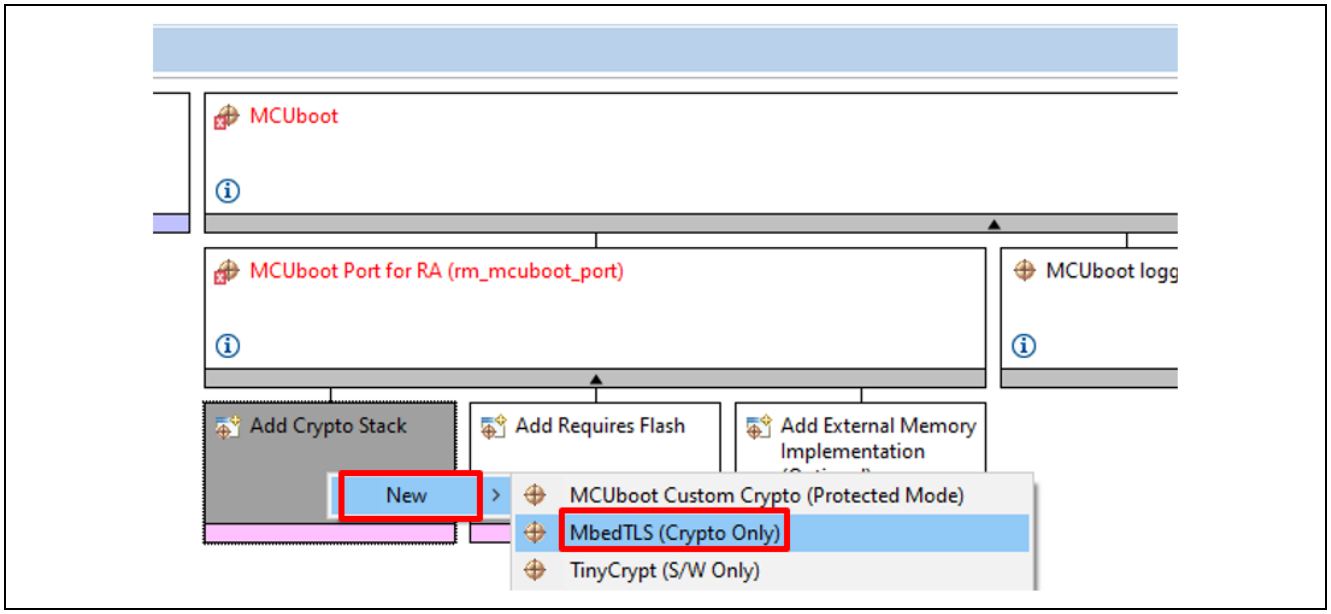

図 27　MbedTLS 暗号専用モジュール選択

2. [Add Requires Flash]を右クリックし、[Flash (r_flash_hp)]スタックの追加を選択します。

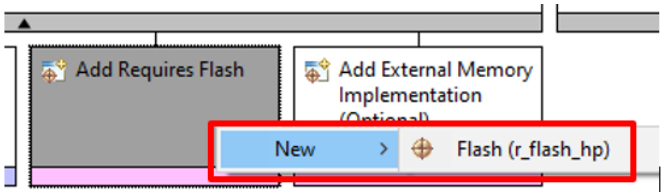

図 28　フラッシュドライバ追加

3. 次に、[Code Flash Programming]を[Enabled]に設定します。[Data Flash Programming]がブートローダで使用されていないので、ブートローダのメモリ使用量を減らすために[Data Flash Programming]を[Disabled]に設定します。

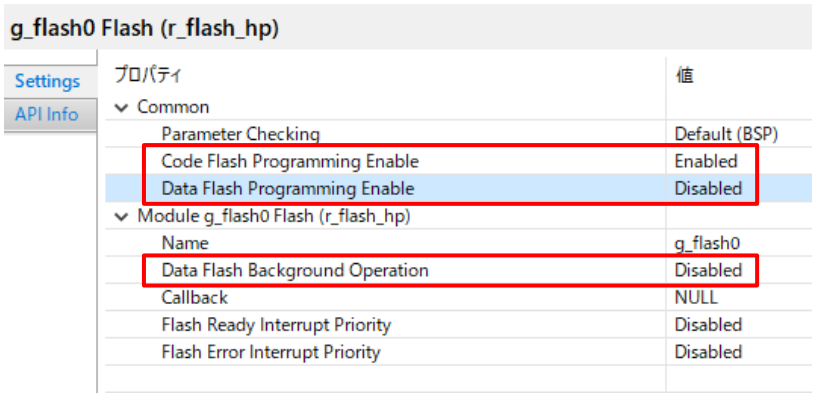

図 29　フラッシュドライバ設定

4. MCUboot Port for RA (rm_mcuboot_port)のプロンプトに従って、次のプロパティを設定します。

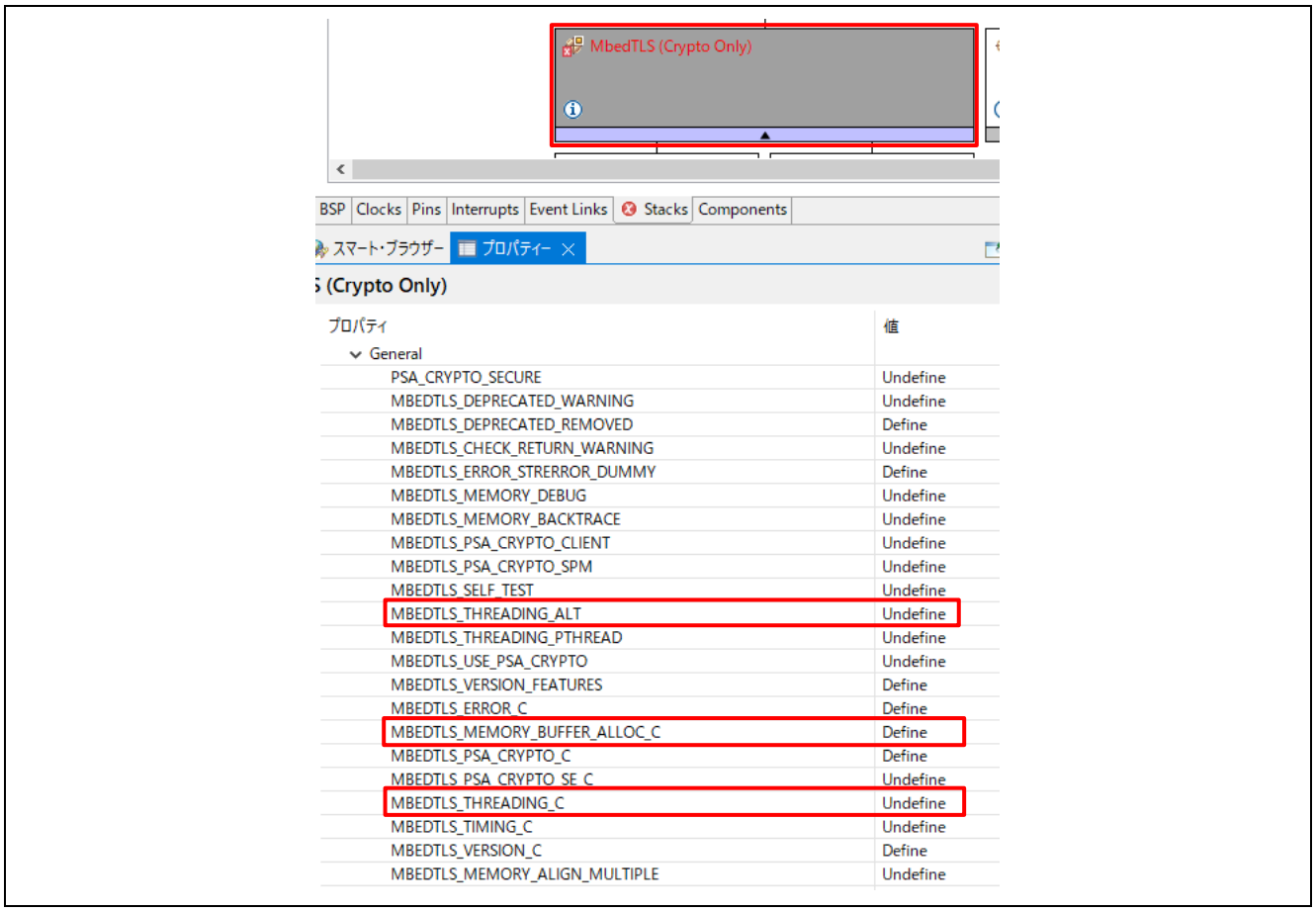

図 30　MbedTLS (Crypto Only)モジュール設定

5. RSA を無効にして、メモリ使用量を節約します。

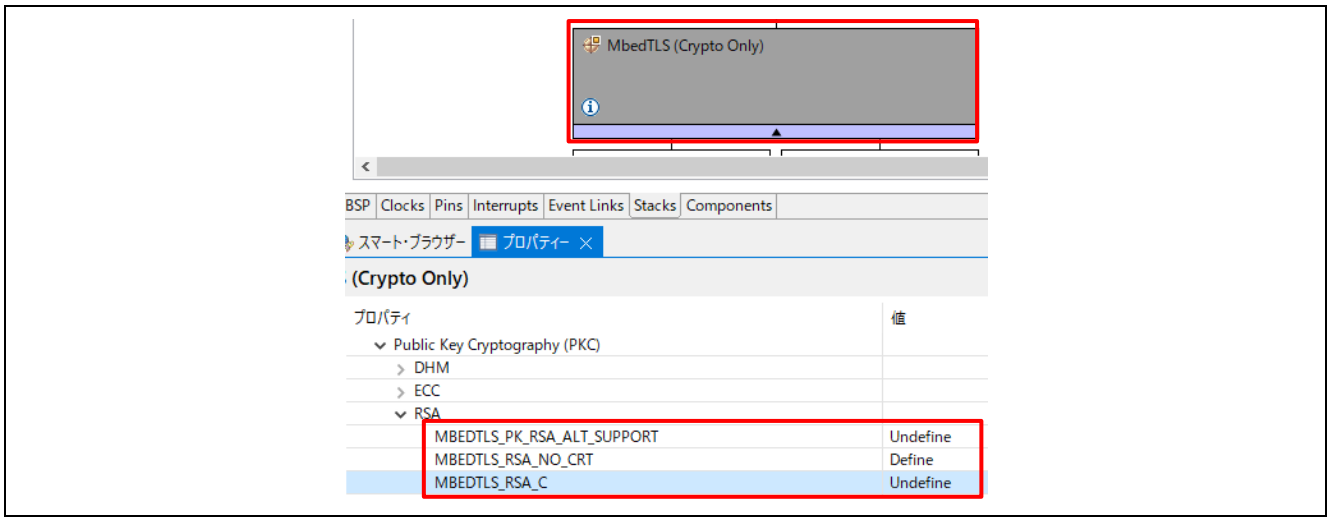

図 31　RSA 無効

6. 認証モードに基づいて、ブートローダによって使用されるスタックとヒープを設定します。
MbedTLS (Crypto Only)モジュールからのプロンプトに従って、[BSP]タブからスタックとヒープサイズを変更します。

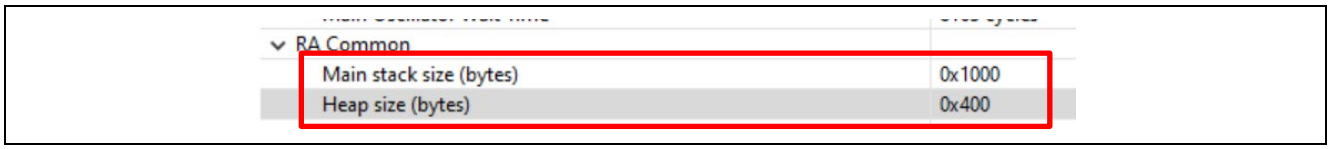

図 32　BSP スタックとヒープ使用量設定

7. サンプルプロダクションキーモジュールを追加します。

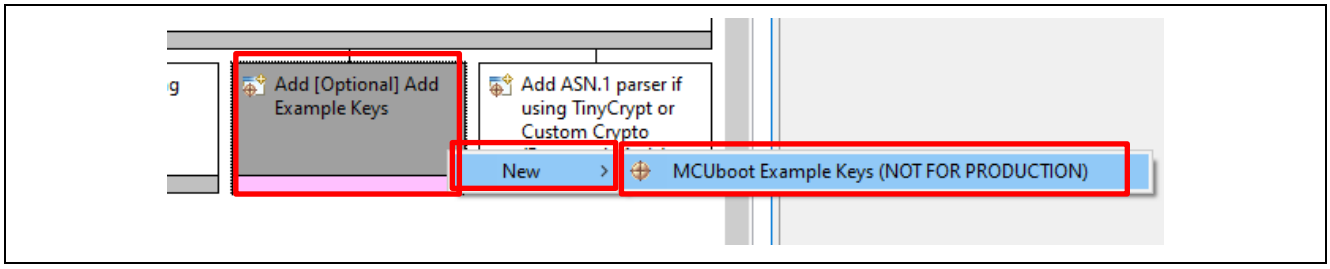

図 33　サンプルプロダクションキーモジュール追加

8. [BSP]タブで[Dual Bank Mode]を[Enabled]にします。

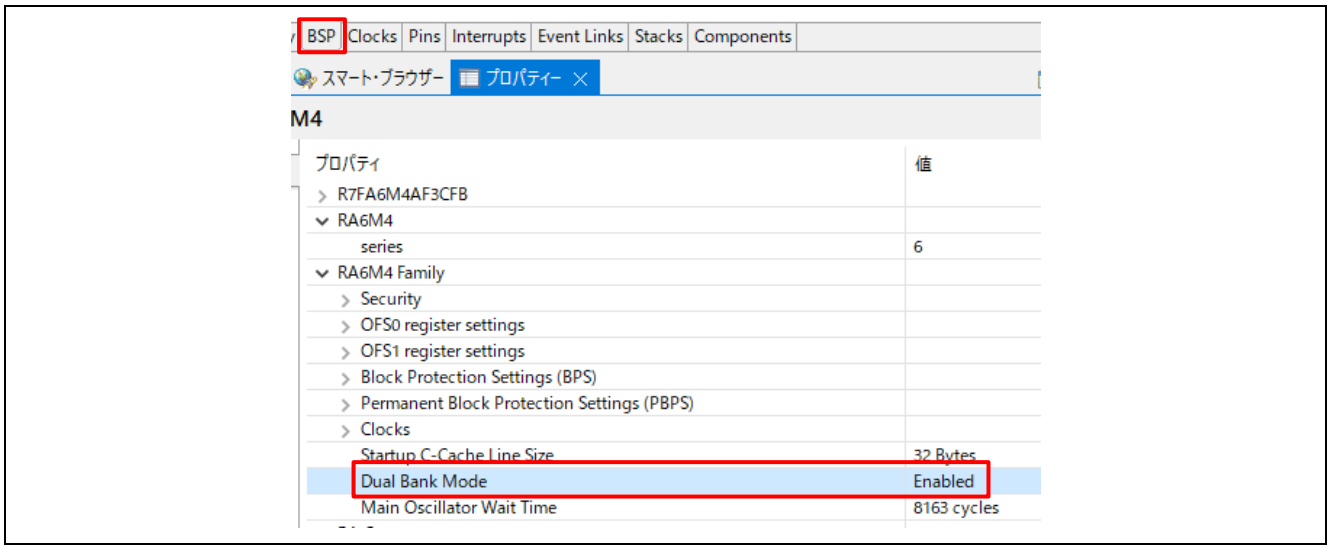

図 34　フラッシュデュアルモード有効化

4.4　ブートコードの追加

`Configuration.xml` を保存し、[Generate Project Content]をクリックします。次に、[Developer Assistance] > [HAL/Common] > [MCUboot] > [Quick Setup]を展開し、[Call Quick Setup]をブートローダプロジェクトの `hal_entry.c` の一番上にドラッグして追加します。

`hal_entry()`関数の先頭に以下の関数呼び出しを追加します。

```
mcuboot_quick_setup();
```

4.5　ブートローダプロジェクトのコンパイル

RA コンフィギュレータで、[Generate Project Content]をクリックしプロジェクトをコンパイルします。

```
'Invoking: GNU Arm Cross Create Flash Image'
arm-none-eabi-objcopy -O srec "ra_mcuboot_ra6m4_dualbank.elf"  "ra_mcuboot_ra6m4_dualbank.srec"
'Invoking: GNU Arm Cross Print Size'
arm-none-eabi-size --format=berkeley "ra_mcuboot_ra6m4_dualbank.elf"
   text    data     bss     dec     hex filename
  60376       0    6356   66732   104ac ra_mcuboot_ra6m4_dualbank.elf
'Finished building: ra_mcuboot_ra6m4_dualbank.srec'
'Finished building: ra_mcuboot_ra6m4_dualbank.siz'
' '
' '

01:12:01 Build Finished. 0 errors, 190 warnings. (took 57s.212ms)
```

図 35　ブートローダ ra_mcuboot_ra6m4_dualbank コンパイル

サードパーティコードにワーニングがあります。

4.6　Python 署名ツールの環境構築

アプリケーションイメージへの署名は、MCUboot に含まれるイメージ署名ツール `Imgtool.py` を用いて、e² studio のビルド後のステップで行うことができます。このツールは、アプリケーションイメージに署名するために e² studio のポストビルドツールとして統合されています。Python スクリプト署名ツールを使用するのが初めてではない場合は、セクション 5 までスキップできます。

Python スクリプト署名ツールを初めて使用する場合、スクリプトが動作するために必要な依存関係をインストールする必要があります。[プロジェクト・エクスプローラー]で[ra_mcuboot_ra6m4_dualbank] > [ra] > [mcu-tools] > [MCUboot]フォルダに移動し、右クリックして[Command Prompt]を選択します。これによりパスが`\mcu-tools\MCUboot` フォルダに設定されたコマンドウィンドウが表示されます。

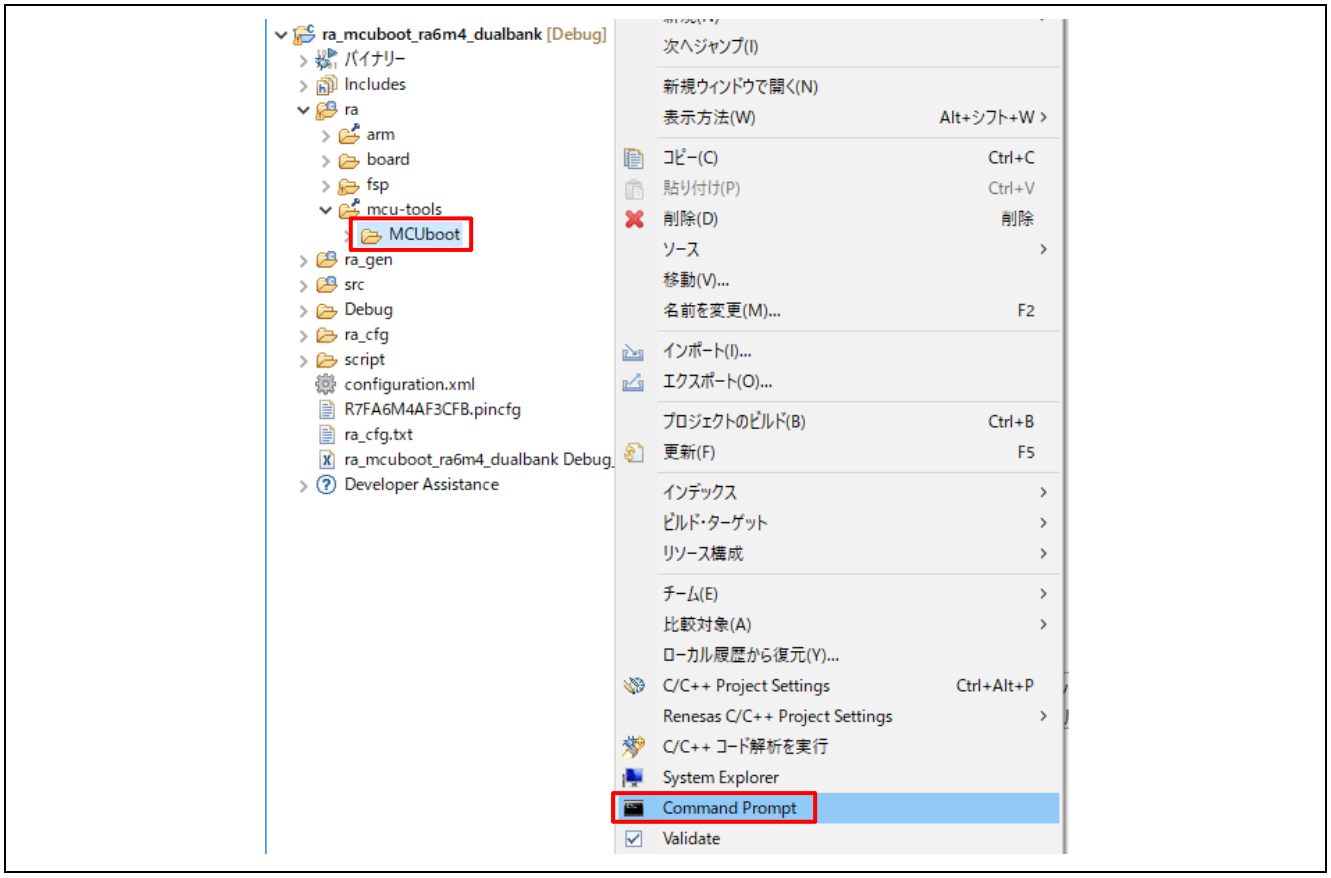

図 36　コマンドプロンプト起動

依存関係をインストールする前に、pip をアップデートすることをお勧めします。以下のコマンドを入力して、pip をアップデートしてください。

```
python -m pip install --upgrade pip
```

次に、コマンドウィンドウで、以下のコマンドを入力してすべての MCUboot 依存関係をインストールしてください。

```
pip3 install --user -r scripts/requirements.txt
```

これにより、必要な依存関係が検証され、インストールされます。

署名コマンドの確認

ブートローダがコンパイルされると、アプリケーションイメージの署名コマンドが自動的に生成されます。[プロジェクト・エクスプローラー]で、`ra_mcuboot_ra6m4_dualbank\debug\ra_mcuboot_ra6m4_dualbank.bld` ファイルを開きます。署名コマンドはセクション <images> の下にあります。

アプリケーションイメージは、[ビルド変数]を使用して、.bld ファイルにリンクします。このプロセスについては、セクション 5.1 で詳しく説明します。アプリケーションイメージには、.bld ファイルへのアクセス権があり、アプリケーションイメージがコンパイルされると、署名コマンドが自動的に実行されます。

```
<images>
    <image path="${BuildArtifactFileBaseName}.bin.signed">python ${workspace_loc:ra_mcuboot_
    0x20000 --max-sectors 4  --confirm --pad-header ${BuildArtifactFileName} ${BuildArtifac
```

図 37　.bld ファイル署名コマンド

4.7　製品化の準備

製品化のため、上側バンクにロードされるブートローダの `.srec` ファイルを生成します。これは、ブートローダプロジェクトのために e² studio 内のカスタムビルダーを設定することで実行できます。

このアプリケーションプロジェクトには、`srec_cat.exe` を使用してスクリプトを実行するバッチファイル `process_bootloader.bat` が含まれており、`.srec` ファイル `ra_mcuboot_ra6m4_dualbank_offset.srec` を生成します。このファイルは、ブートローダのオフセットをフラッシュリニアモードの上側バンクアドレス 0x80000 にオフセットします。

オプション設定メモリはバンクの範囲外にあるため、このプロセスではブートローダもバンクサイズ 0x80000 に切り捨てられます。

```
srec_cat Debug\ra_mcuboot_ra6m4_dualbank.srec -crop 0 0x80000 -offset 0x80000 -o
ra_mcuboot_ra6m4_dualbank_offset.srec
```

図 38　上側バンクへのブートローダロード処理：process_bootloader.bat

以下の手順に従って、作成したブートローダプロジェクトのカスタムビルダーを設定します。

1. RA6_Secure_Bootloader_Dualbank.zip を解凍し、同じフォルダにある \ra_mcuboot_ra6m4_dualbank\process_bootloader.bat と srec_cat.exe を作成したブートローダプロジェクトのルートフォルダ直下にコピーしてください。
2. ブートローダプロジェクトを右クリックし、[プロパティ]を開き、[ビルダー]に移動します。[新規]をクリックし、カスタムビルダーの作成を開始します。

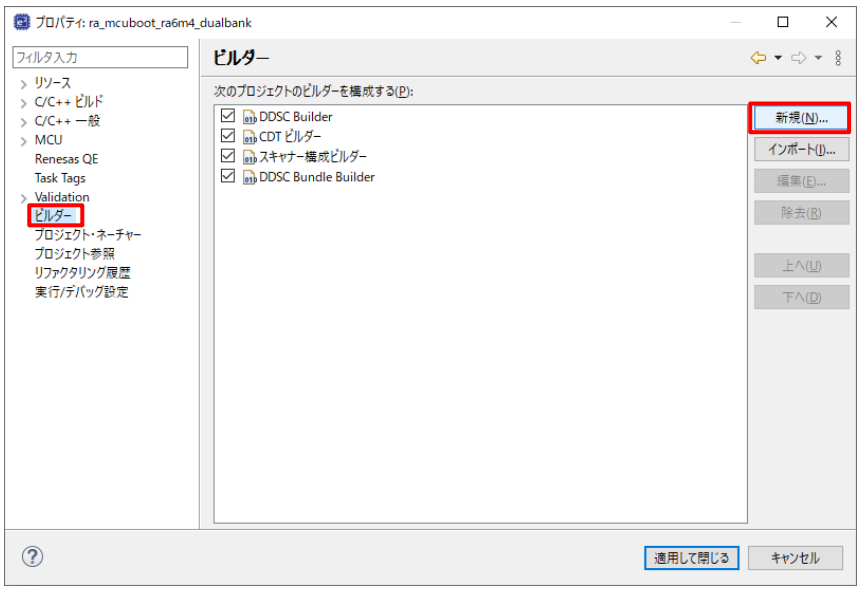

図 39　カスタムビルダーエントリー新規作成

3. 次の画面で[プログラム]を選択し、[OK]をクリックします。

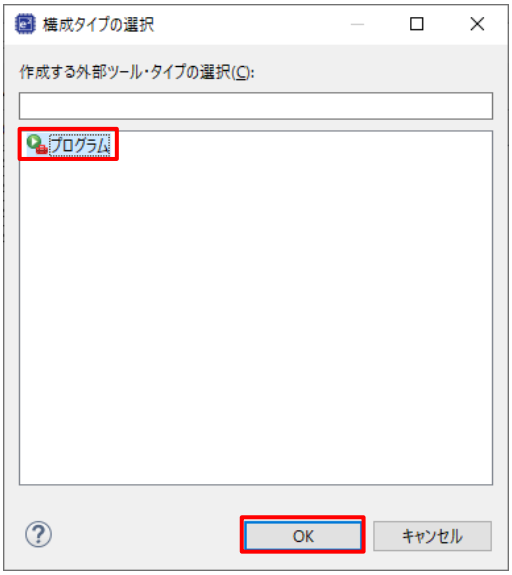

図 40　ビルダーの種類をプログラムとして選択

4. 次に、[名前]に `Process Bootloader` を入力し、[ワークスペースの参照]をクリックしてビルダーの[ロケーション]に `process_bootloader.bat` ファイルを選択します。また、[ワークスペースの参照]をクリックして、[作業ディレクトリー]を以下のように設定し、[適用]をクリックします。

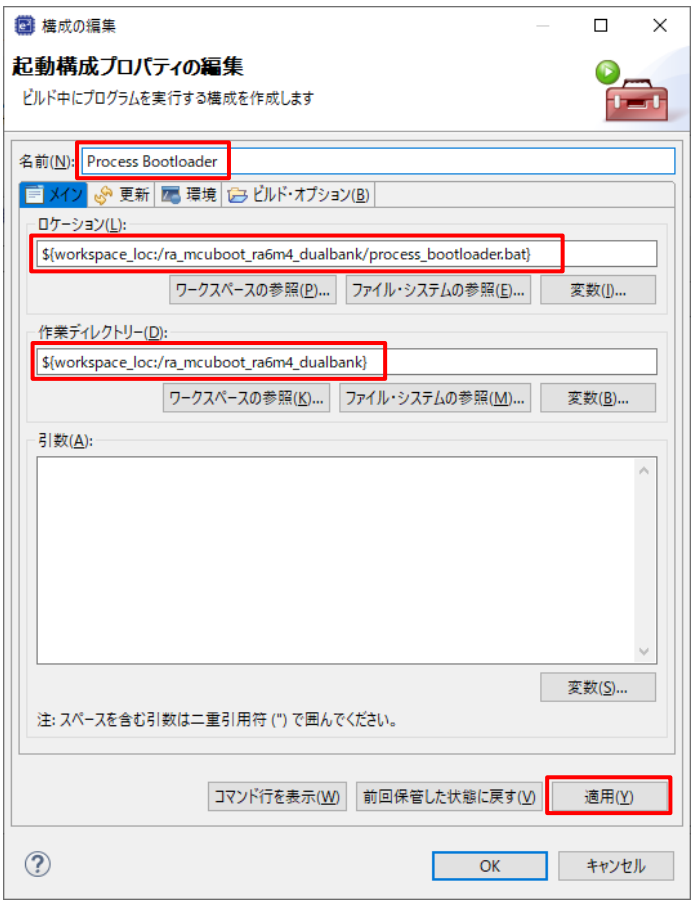

図 41　カスタムビルダー設定

5. [OK]をクリックし、次の画面で[適用して閉じる]をクリックします。

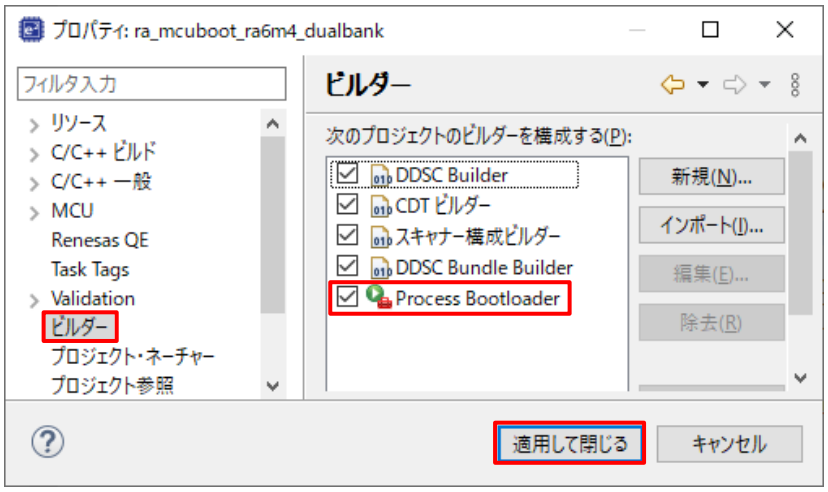

図 42　カスタムビルダー

6. ブートローダプロジェクトを再コンパイルし、ブートローダプロジェクトのルート・ディレクトリーの下に `ra_mcuboot_ra6m4_dualbank_offset.srec` が作成されたことを確認します。

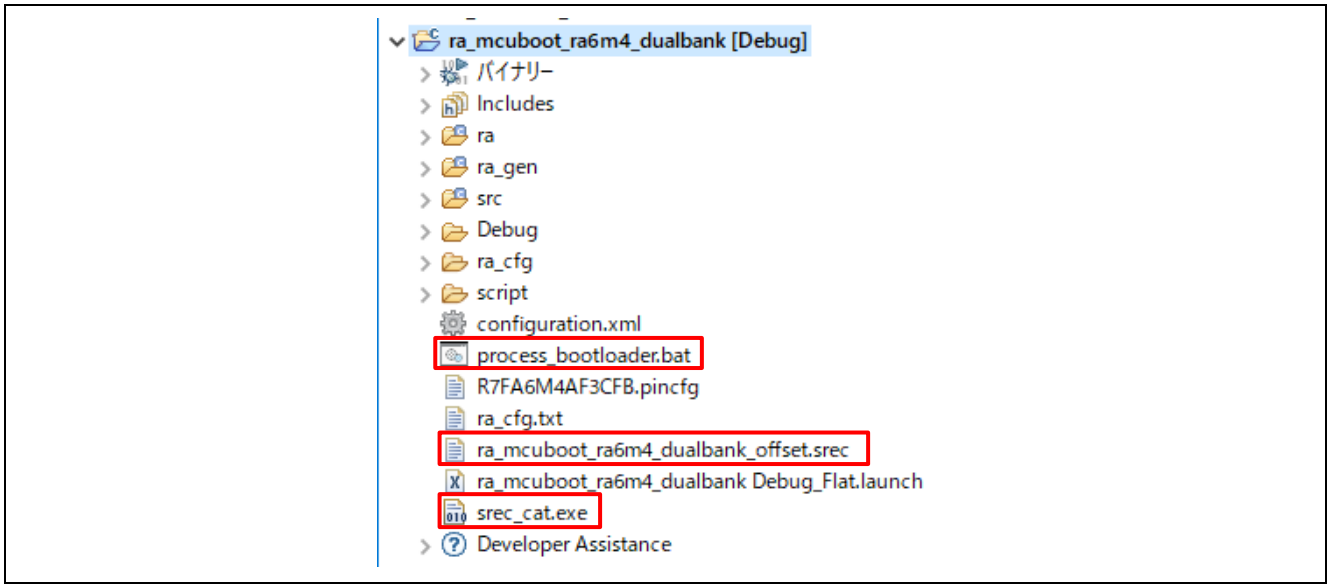

図 43　カスタムビルダーによるブートローダの再ビルド

5.　アプリケーションプロジェクトの設定と署名

ブートローダを使用する最初のアプリケーションの開発は、アプリケーションとブートローダを個別に開発およびテストすることから始まります。ブートローダを既存のアプリケーションで使用したり、ブートローダを使用する新しいアプリケーションを開発するには、次のような共通の手順があります。

- アプリケーションとブートローダが利用可能な MCU メモリ領域に収まるようにブートローダのメモリマップを調整します。
- ブートローダを使用するようにアプリケーションを設定します。
- アプリケーションイメージに署名します。
- ブートローダを使用するアプリケーションを開発するには、通常、新しいアプリケーションをダウンロードする機能を持ったアプリケーションが必要です。このアプリケーションプロジェクトでは、USB インタフェースと UART インタフェースの両方を使用して新しいアプリケーションをダウンロードする方法を示しています。イメージのダウンロード方法は、お客様の環境に合わせてカスタマイズしていただくのが一般的です。

5.1　ブートローダを使用するためのアプリケーションプロジェクトの設定

ユーザは、FSP ユーザーズマニュアルのチュートリアルセクション：Your First RA MCU Project - Blinky に従って新しいプロジェクトを作成できます。本アプリケーションノートでは、付属のサンプルプロジェクトを最初のアプリケーションプロジェクトとして使用し、セクション 4 で作成したブートローダを使用するように、サンプルプロジェクトを構築する手順を説明しています。

このセクションで説明する手順を別の既存のアプリケーションプロジェクトに適用し、ブートローダを使用するようにアプリケーションプロジェクトを構築できますが、注意が必要です。アプリケーションプロジェクトのサイズを考慮する必要があります。別のアプリケーションプロジェクトでブートローダを使用する場合は、[Image 1 Flash Area Size]プロパティを適宜調整する必要があります。

`\example_projects_without_bootloader` フォルダの下にある目的のアプリケーションプロジェクトを、ブートローダを作成したワークスペースにインポートします。例えば、目的のファームウェアアップデートチャネルが USB の場合、ワークスペースに `app_primary_usb` をインポートします。

このセクションの図では、USB インタフェースが使用されています。UART インタフェースの使用手順は、USB の使用手順と同様です。

[プロジェクト・エクスプローラー]でアプリケーションプロジェクトフォルダー`app_primary_usb` を右クリックし、[プロパティ]を選択します。[C/C++ビルド] > [ビルド変数]を選択し、[追加]をクリックして[変数名]に `BootloaderDataFile` を入力し、[すべての構成に適用]をチェックします。[タイプ]を[ファイル]に変更し、ブートローダプロジェクト `ra_mcuboot_ra6m4_dualbank` の bld ファイルへのパスを入力します。

- ${workspace_loc:ra_mcuboot_ra6m4_dualbank}/Debug/ra_mcuboot_ra6m4_dualbank.bld を[値]に設定してください。

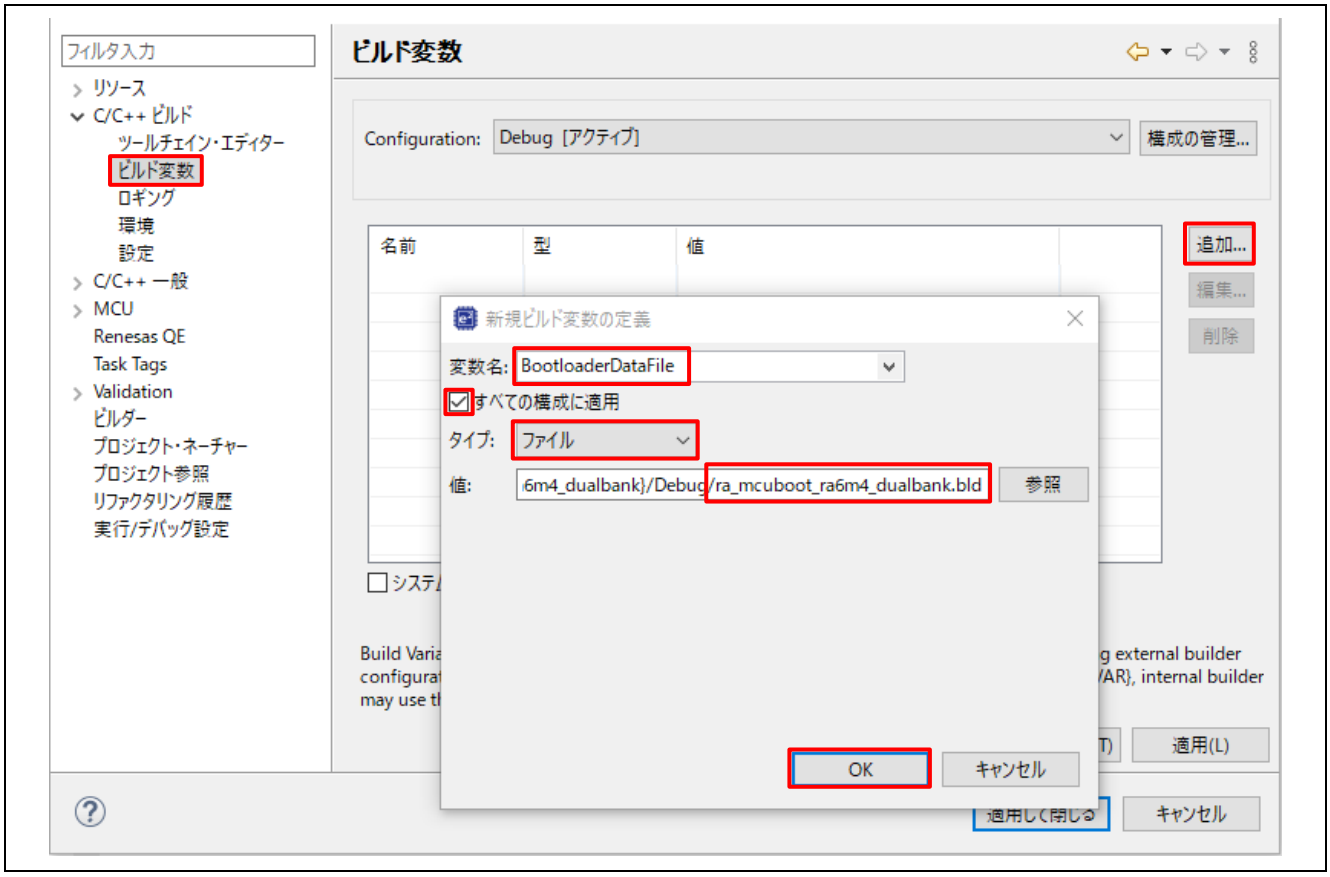

図 44　ブートローダ用のビルド変数設定

[OK]をクリックし、次の画面で[適用]、[適用して閉じる]の順にクリックします。

5.2　アプリケーションイメージへの署名

MCUboot モジュールの署名と、署名の[プロパティ]を変更後にブートローダプロジェクトを再ビルドする場合、更新した.bld ファイルを取り込むために再度[Generate Project Content]を実行する必要があります。

Direct XIP モードを使用する場合、各アプリケーションはバージョン番号を定義できます。これは、環境変数 MCUBOOT_IMAGE_VERSION を定義することで実現されます。

署名の検証をサポートするアプリケーションでは、別の環境変数 MCUBOOT_IMAGE_SIGNING_KEY を使用して署名鍵を設定することができます。署名検証を行わない場合は、環境変数 MCUBOOT_IMAGE_SIGNING_KEY を設定する必要はありません。

プロジェクト app_primary_usb の[プロパティ]ページを開き、[環境]で[追加]をクリックして、MCUBOOT_IMAGE_VERSION を設定します。

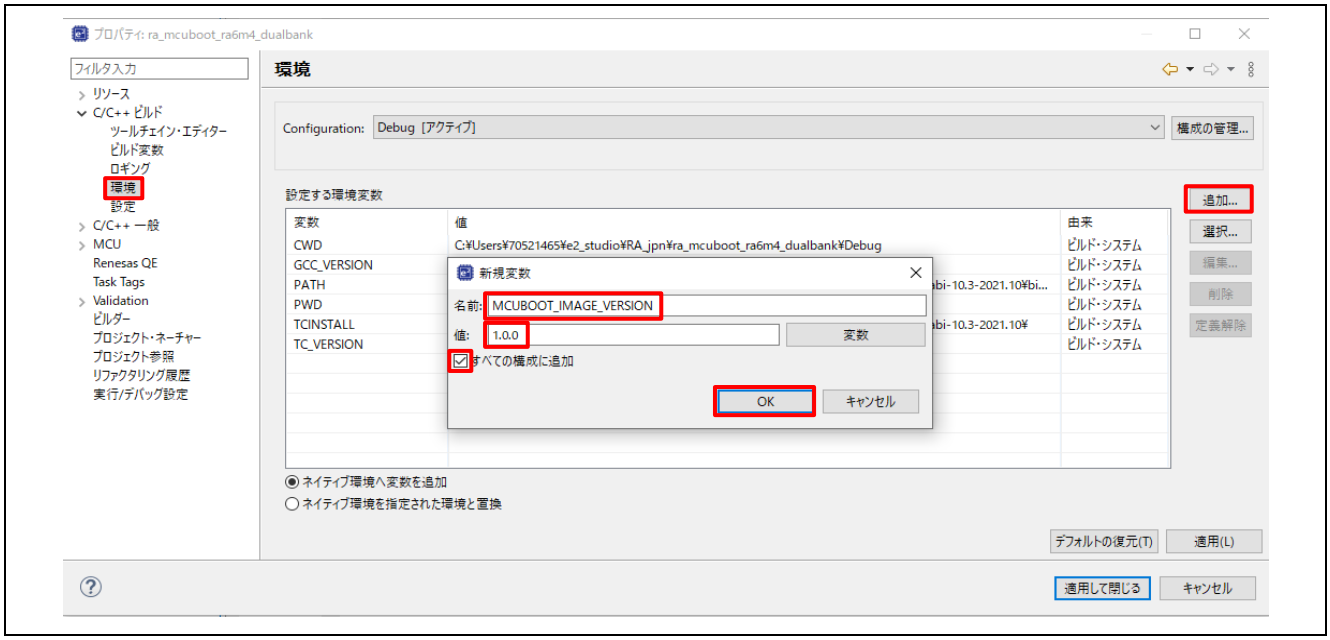

図 45　アプリケーションバージョン設定

同様に、新しい変数として MCUBOOT_IMAGE_SIGNING_KEY を追加します。

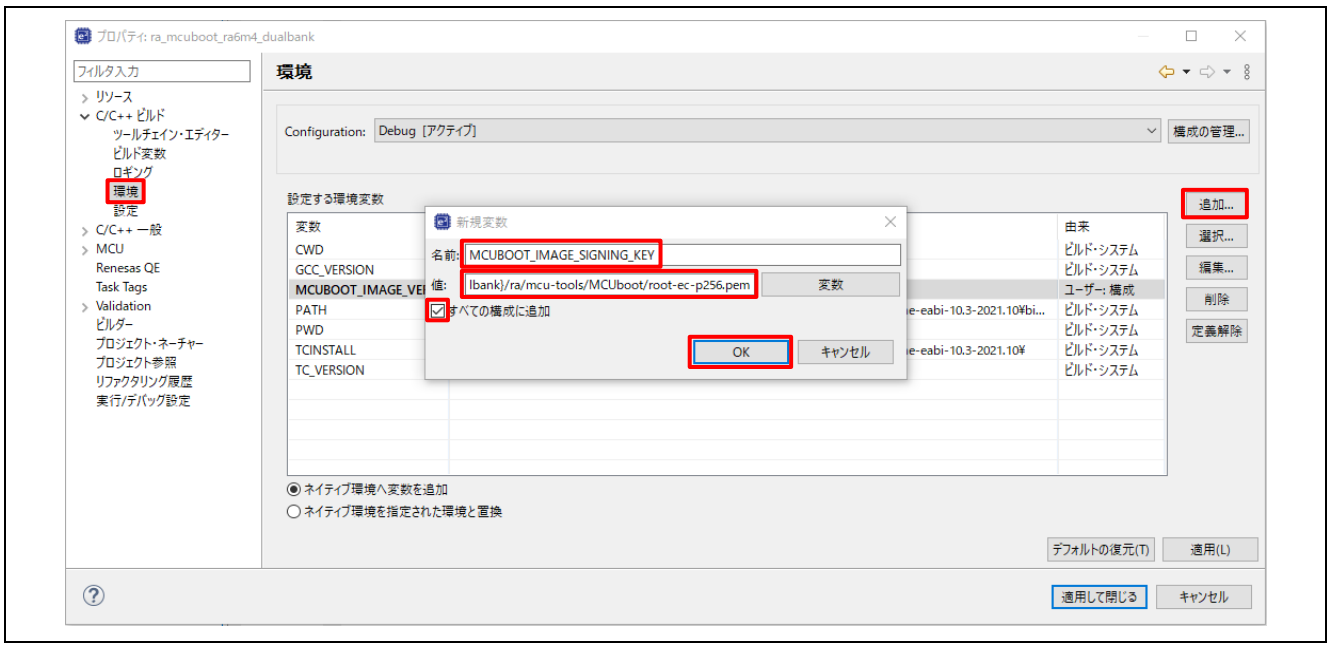

図 46　秘密署名鍵設定

アプリケーションイメージの署名に使用される秘密鍵は、署名コマンドで表示されることに注意してください。

`/ra/mcu-tools/MCUboot/root-ec-p256.pem` はサンプルブートローダで使用されています。この鍵はテスト用途専用です。実際のユースケースや製品化においては、ユーザは任意の秘密鍵に**必ず**変更しなければなりません。

図 47 は、上記の設定の結果です。[適用して閉じる]をクリックします。

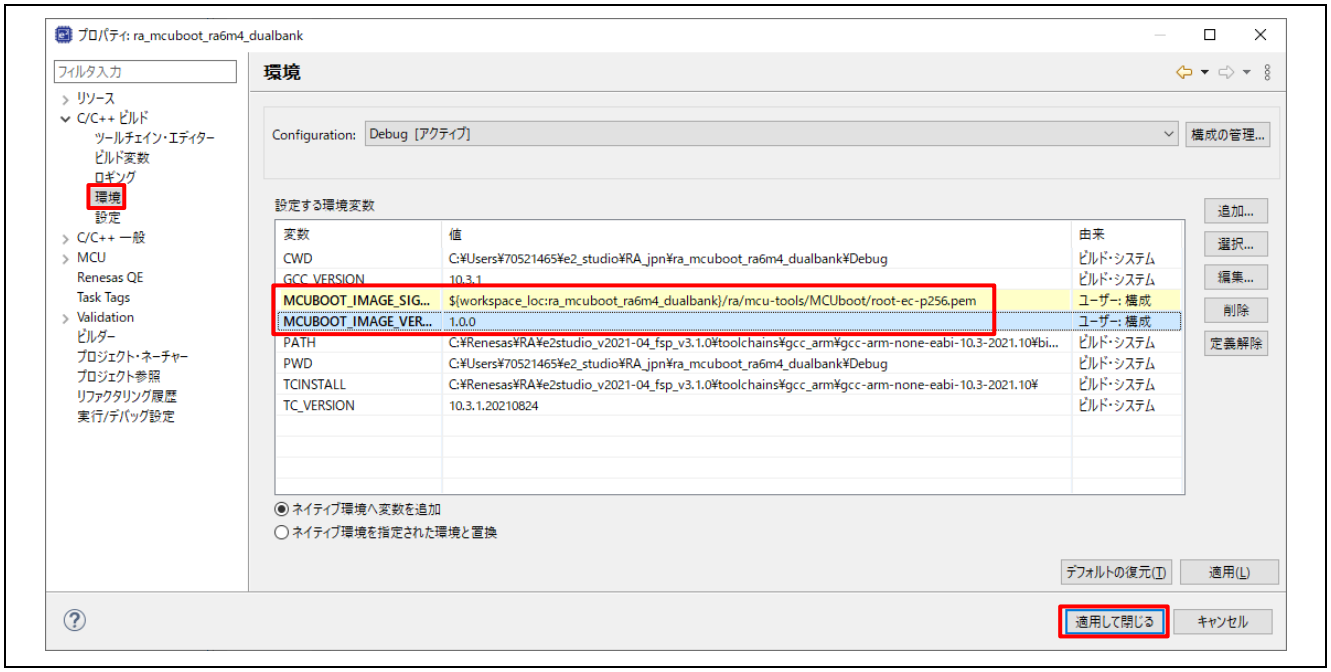

図 47　アプリケーションイメージバージョン番号及び署名鍵設定

環境変数が更新されるたびにプロジェクトを再コンパイルできるようにするには、[ビルド前のステップ]を追加して、以下に示すように.elf ファイルを常に削除し、アプリケーションプロジェクトが常に再コンパイルされるようにすることを推奨します。

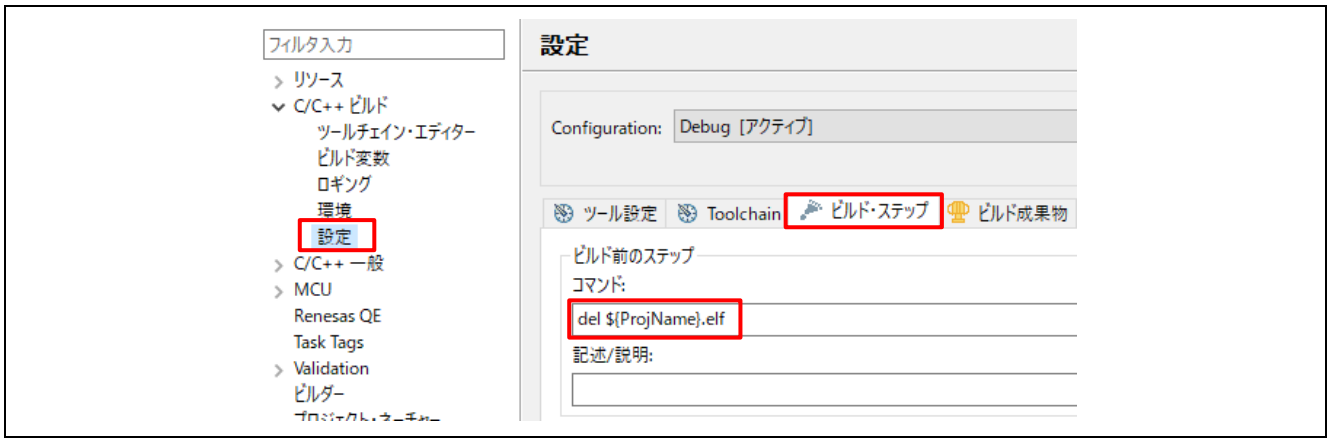

図 48　ビルド前コマンド設定

この時点で、ユーザは[Generate Project Content]をクリックし、新しく作成されたアプリケーションプロジェクトをコンパイルして\Debug\app_primary_usb.bin.signed が生成されることを確認できます。

5.3　製品化の準備

製品化のために、署名済みアプリケーションイメージに基づく.srec ファイルを生成する必要があります。この.srec ファイルは、図 25 に基づいて、プライマリアプリケーションの開始アドレス 0x20000 にアプリケーションをオフセットしています。

```
srec_cat Debug\app_primary_usb.bin.signed -binary -offset 0x20000 -o
app_primary_usb_singed_offset.srec
```

図 49　app_primary_usb_signed_offset.srec 作成

セクション 4.7 と同様の手順に従って、カスタムビルダーを追加し、プライマリアプリケーションをコンパイルします。

1. まず、\example_projects_with_bootloader\app_primary_usb\srec_cat.exe と process_signed_binary_primary.bat を app_primary_usb プロジェクトのルートにコピーします。
2. 次にセクション 4.7 に従って新しい[ビルダー]を作成します。設定したものは図 50 のようになります。

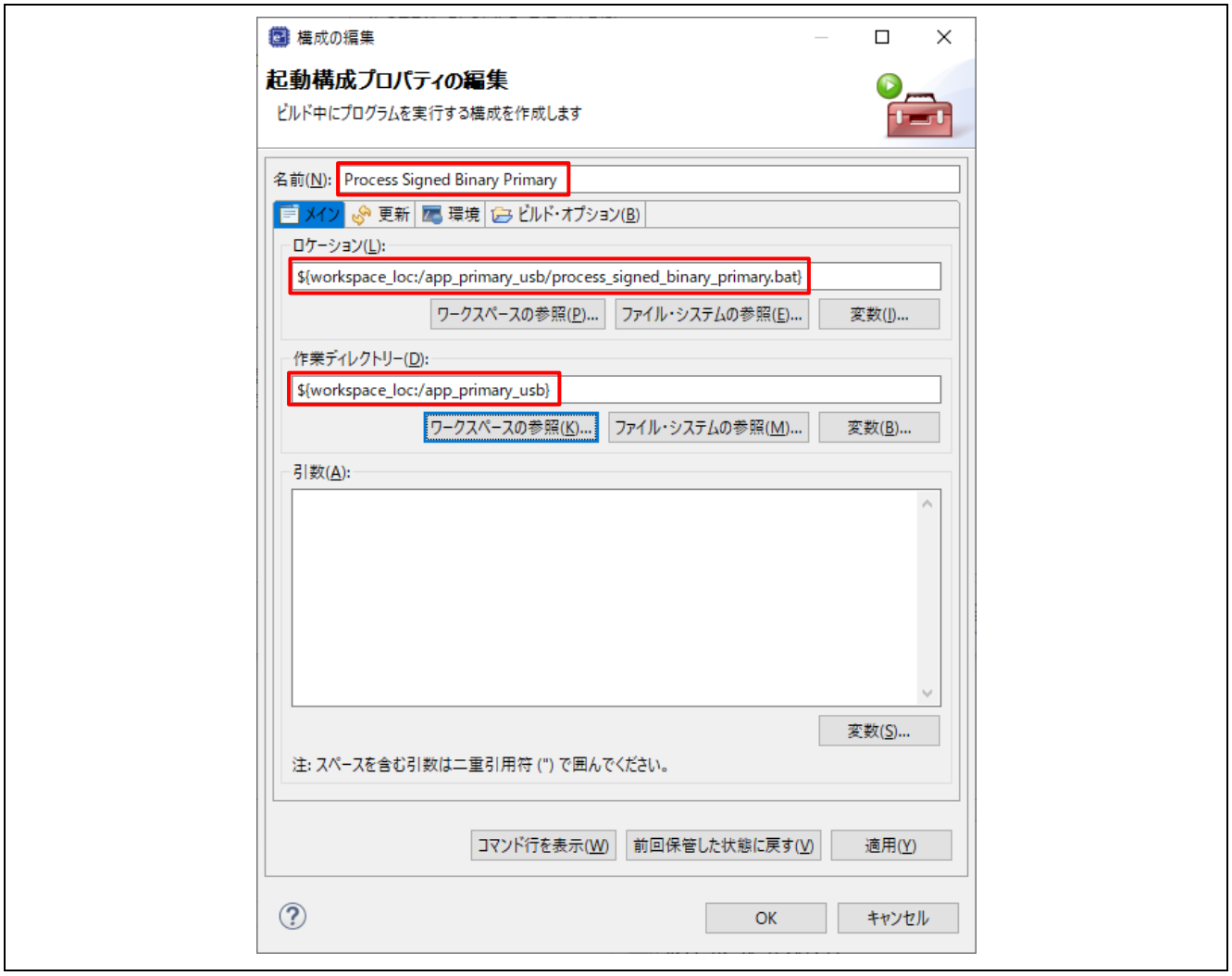

図 50　プライマリアプリケーションカスタムビルダー設定

3. [Generate Project Content] をクリックし、app_primary_usb プロジェクトをコンパイルします。app_primary_usb_signed_offset.srec が app_primary_usb プロジェクトのルートに生成されていることを確認します。

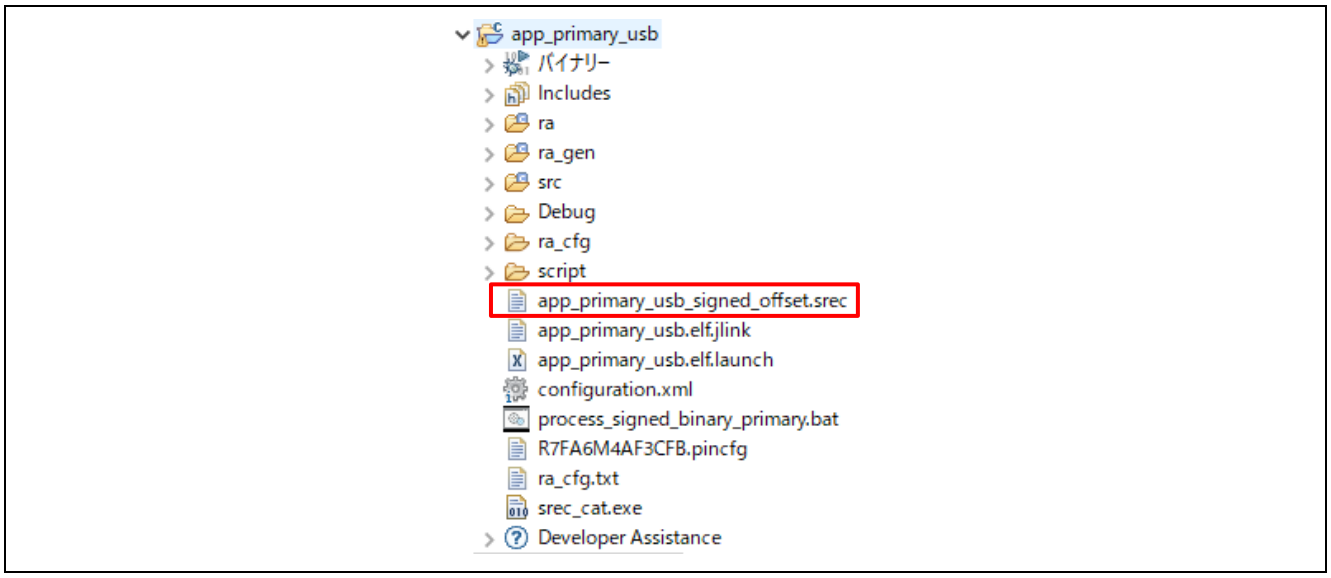

図 51　署名付きプライマリイメージのプライマリスロットへのオフセット

6.　プライマリアプリケーションの起動と新しいイメージへの更新

アプリケーションを更新するためには、プライマリアプリケーションにイメージダウンローダを実装する必要があります。また、イメージダウンローダの機能をテストするために、新しいイメージを準備する必要があります。

6.1　セカンダリイメージの準備

このプロジェクトでは、プライマリアプリケーションのダウンロード機能をテストするためにセカンダリイメージが作成されます。新しいアプリケーションは、既存のアプリケーションを変更するか、新規にアプリケーションプロジェクトを作成することによって作成できます。新規にアプリケーションプロジェクトを作成する場合は、セクション 5 に従ってブートローダへのリンクを確立する必要があります。また、新規に作成されたアプリケーションプロジェクトは、新しいアプリケーションを上側バンクにダウンロードする方法を実装する必要があります。

このアプリケーションプロジェクトでは、最初のアプリケーションプロジェクトを同じワークスペースにインポートし、新しいプロジェクトの名前を変更し、簡単な変更を行います。

[プロジェクト・エクスプローラー]の空白部分を右クリックして[インポート]を選択し、[Rename & Import Existing C/C++ Project into Workspace]を選択します。

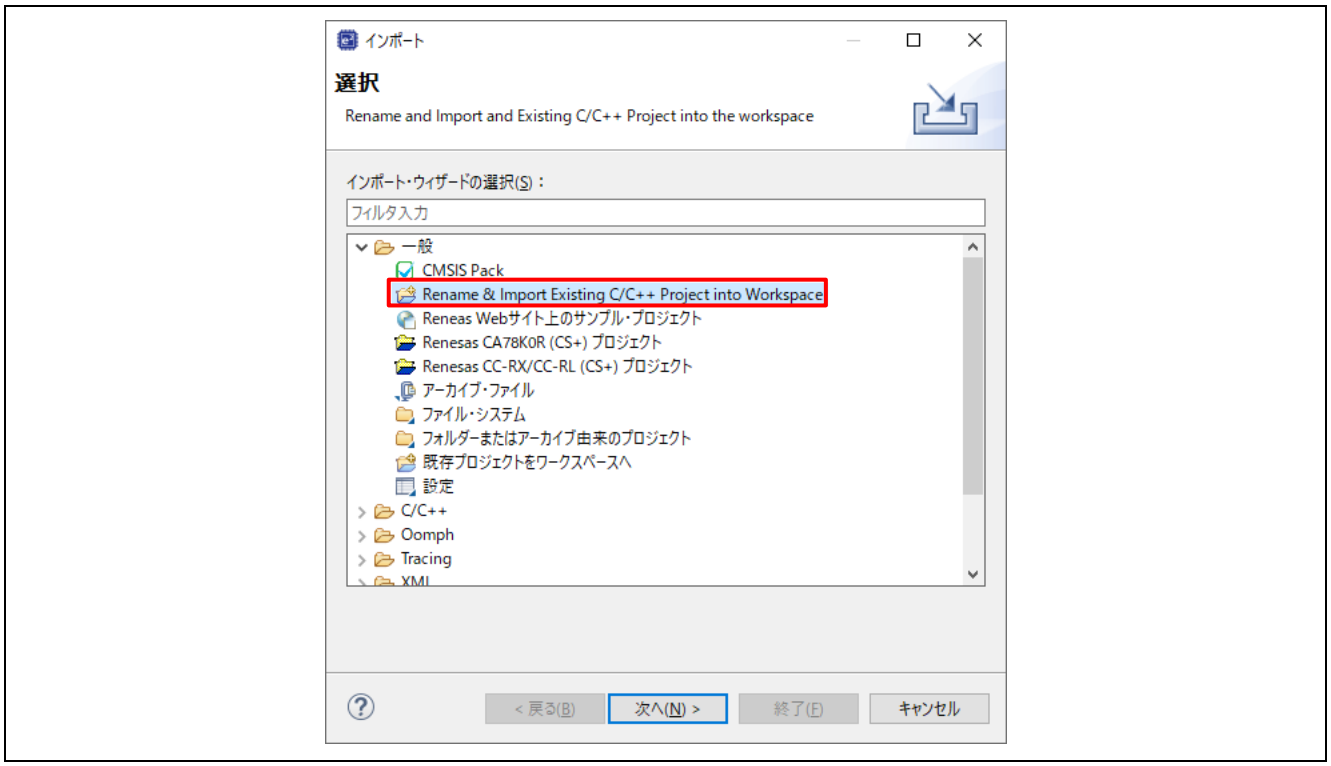

図 52　最初のアプリケーションのインポート

[インポート]ウィンドウが開いたら、プロジェクト名を `app_secondary_usb` とし、[ルート・ディレクトリーの選択]にチェックを入れて、[参照]をクリックします。

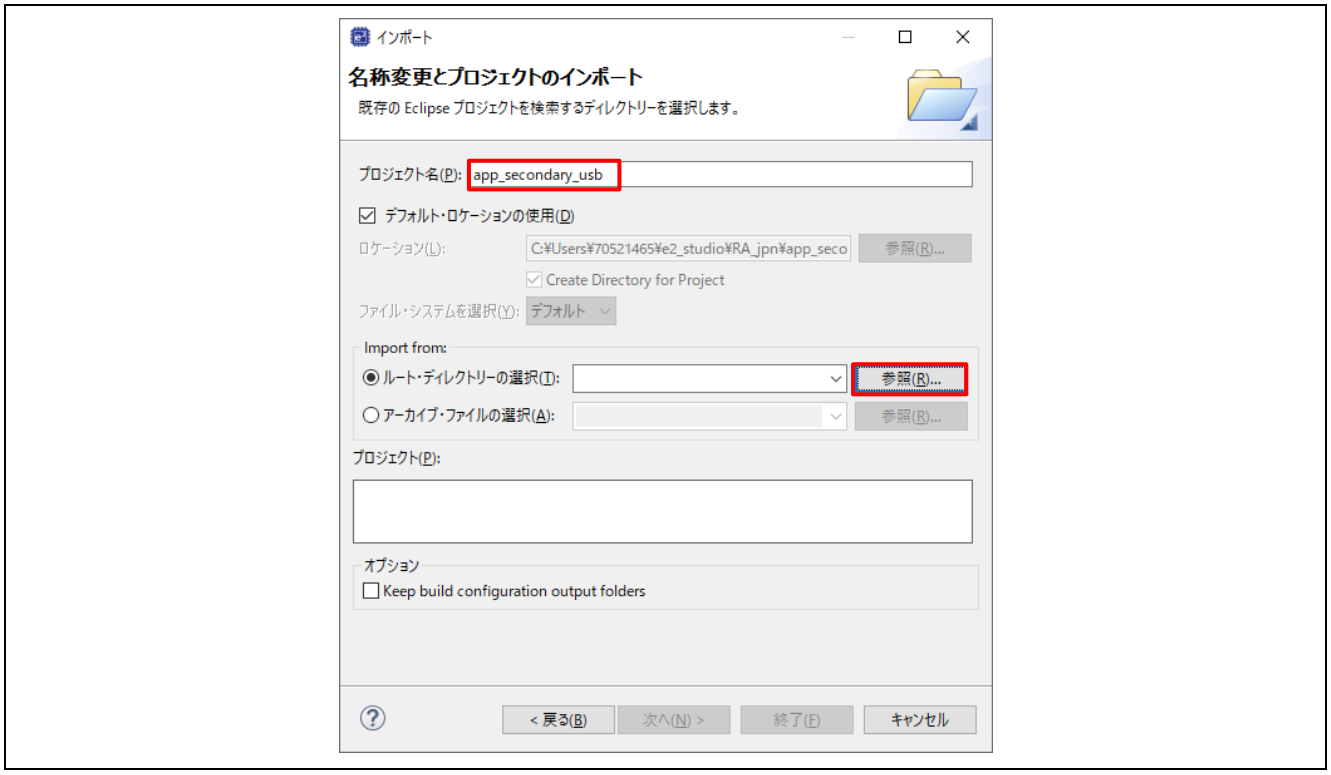

図 53　新規アプリケーション名

ワークスペースのフォルダを参照し、app_primary_usb を選択します。

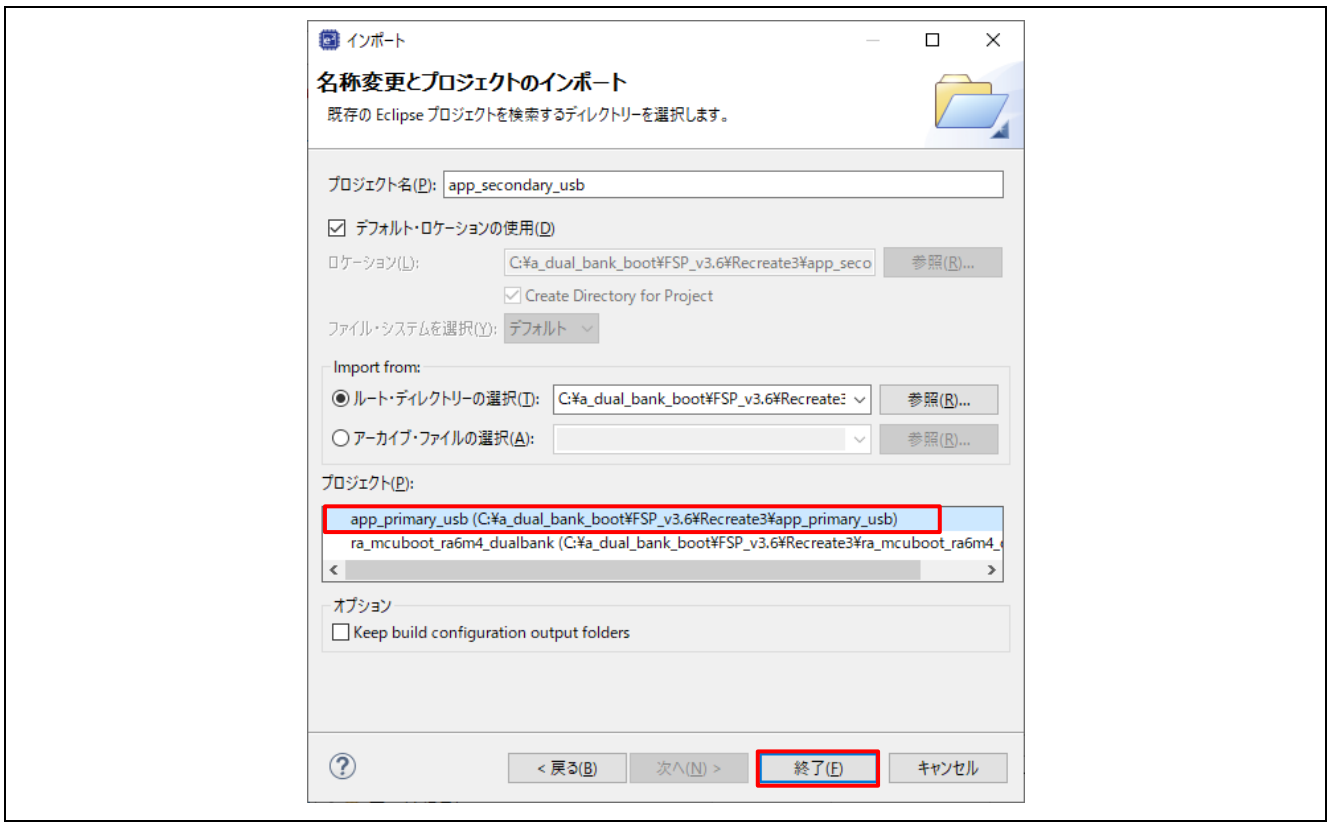

図 54　最初のプライマリアプリケーション選択

[終了]をクリックすると、新しいアプリケーションプロジェクトが作成されます。

- プライマリアプリケーションをインポートする際、[ビルド変数]と[環境変数]が自動的にインポートされます。
- カスタムビルダー”Process Signed Binary Primary”もインポートされます。クリーンなプロジェクトにするためには、ユーザはこのビルダーと対応するサポートファイルをセカンダリプロジェクトから手動で削除する必要があります。
- 通常の XIP モード動作とは異なり、デュアルモードではリンカースクリプトシンボル XIP_SECONDARY_SLOT_IMAGE は未定義である必要があります。デフォルトでは、XIP_SECONDARY_SLOT_IMAGE はリンカースクリプトシンボルに定義されていないので、ここでは何もする必要はありません。

既存のアプリケーションを新しいアプリケーションに更新する

アプリケーションの更新のデモとして、青と緑の LED のみを点滅させるようにアプリケーションを更新します。

`blinky_thread_entry.c` のコードを次のように変更します。

blinky_thread_entry 関数の以下のコードを:

```
/* Update all board LEDs */
for (uint32_t i = 0; i < leds.led_count; i++)
{
    /* Get pin to toggle */
    uint32_t pin = leds.p_leds[i];

    /* Write to this pin */
    R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);
}
```

次のように変更します:

```
/* update the blue led */
R_BSP_PinWrite(leds.p_leds[0], pin_level);

/* update the green led */
R_BSP_PinWrite(leds.p_leds[1], pin_level);
```

図 55　LED 制御の更新

更新したソースファイルを保存し、[Generate Project Content]をクリックして、新しいプロジェクトをコンパイルします。

新規アプリケーションプロジェクトを作成し、ブートローダを使用してデバッグする場合は、セクション 6.3 からセクション 6.7 の手順で行います。ブートローダで更新したイメージをデバッグする場合、更新したイメージをプライマリアプリケーションとして扱うことができます。

6.2　ハードウェアのセットアップ

最初のアプリケーションプロジェクトとして app_primary_usb を使用した場合：

- USB Micro-B ケーブルを使用して EK-RA6M4 の J10 (USB デバッグ)を開発用 PC に接続し、電源供給とオンボードデバッガによるデバッグ接続を行います。
- USB Micro-B ケーブルを使用して EK-RA6M4 の J11 (USBFS)を開発用 PC に接続し、USB デバイス接続を行います。

最初のアプリケーションプロジェクトとして app_primary_uart を使用する場合：

- USB Micro-B ケーブルを使用して EK-RA6M4 の J10 を開発用 PC に接続し、電源供給とオンボードデバッガによるデバッグ接続を行います。
- UART-USB 変換器の表 3 の 3 つのピンを EK-RA6M4 に接続します。

表 3　UART インタフェースによる接続

UART-USB 変換器	RA6M4
RX	P101 (TX)
TX	P100 (RX)
GND	GND

6.3　MCU の消去

MCUboot がフラッシュデュアルモードで使用される場合、コードフラッシュモードはリニアモードで起動する必要があります。MCU のオプション設定メモリの設定を消去すると、コードフラッシュモードがリニアモードに設定されます。MCU メモリ全体を消去することを推奨します。MCU は、さまざまな方法で消去できます。Renesas Device Partition Manager、Renesas Flash Programmer、または J-Flash Lite などのサードパーティツールを使用して、MCU のフラッシュを消去することができます。

6.3.1　Renesas Flash Programmer の使用

Renesas Flash Programmer (RFP)は、新規 RFP プロジェクト作成時にフラッシュモードを検出することができます。

RFP で接続する前に、開発ボードの電源を入れ直します。

J10 USB デバッグ経由で EK-RA6M4 を PC に接続します。RFP を起動し、RFP プロジェクトの新規作成を開始します。[ファイル] > [新しいプロジェクトを作成]をクリックします。

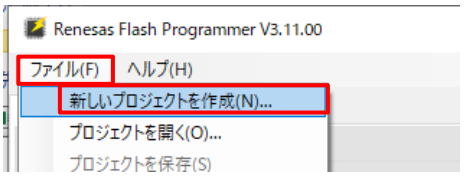

図 56　新しい RFP プロジェクト作成

[マイクロコントローラ]の選択と通信に使用する[ツール]を設定します。 次に、[接続]をクリックします。

図 57　新規プロジェクト設定

接続に成功すると、ユーザは[ブロック設定]ページを開いてコードフラッシュの設定を確認することができます。

RA6M4 のコードフラッシュがリニアモードの場合、[ブロック設定]は図 58 のように表示されます。

図 58　リニアモードのフラッシュ

RA6M4 のコードフラッシュがデュアルモードの場合、[ブロック設定]は図 59 のように表示されます。

図 59　デュアルモードのフラッシュ

MCU がフラッシュデュアルモードまたはフラッシュリニアモードのいずれであっても、[デバイスを初期化する]コマンドは設定領域を含むフラッシュ全体を消去することができます。また、MCU のコードフラッシュをリニアモードに戻すことができます。

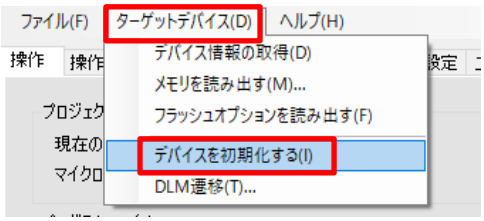

図 60　デバイスの初期化コマンド

MCU がコードフラッシュデュアルモードの場合、次のセクションに進む前に、必ず[デバイスを初期化する]を使用してリニアモードに復元してください。以降の操作説明は、デバイスがコードフラッシュリニアモードであることを前提としています。デバイスが既にコードフラッシュデュアルモードになっている場合は、以降の操作は機能しません。

デバイスの初期化が成功すると、図 61 のメッセージがステータスウィンドウに表示されます。

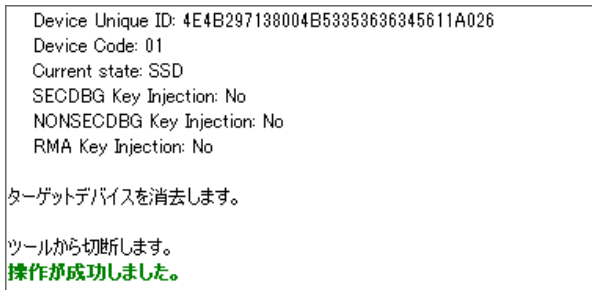

図 61　デバイス初期化成功

6.3.2　SEGGER J-Flash Lite の使用方法

J-Flash Lite は、ターゲットシステムのフラッシュメモリにダウンロードすることができる、無料のシンプルなグラフィカルユーザインタフェースです。J-Flash Lite は、J-Link software & documentation pack をインストール時に、インストールされる J-Link ソフトウェアおよびドキュメントパッケージの一部です。

J-Flash Lite を使用するには、USB デバッグポート J10 を PC に接続し、J-Flash Lite を起動します。[Device]とデバッグ[Interface]と通信速度を選択します。

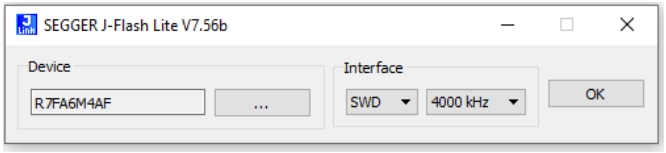

図 62　J-Flash Lite 起動

[OK]をクリックします。次の画面で、[Erase Chip]を選択します。

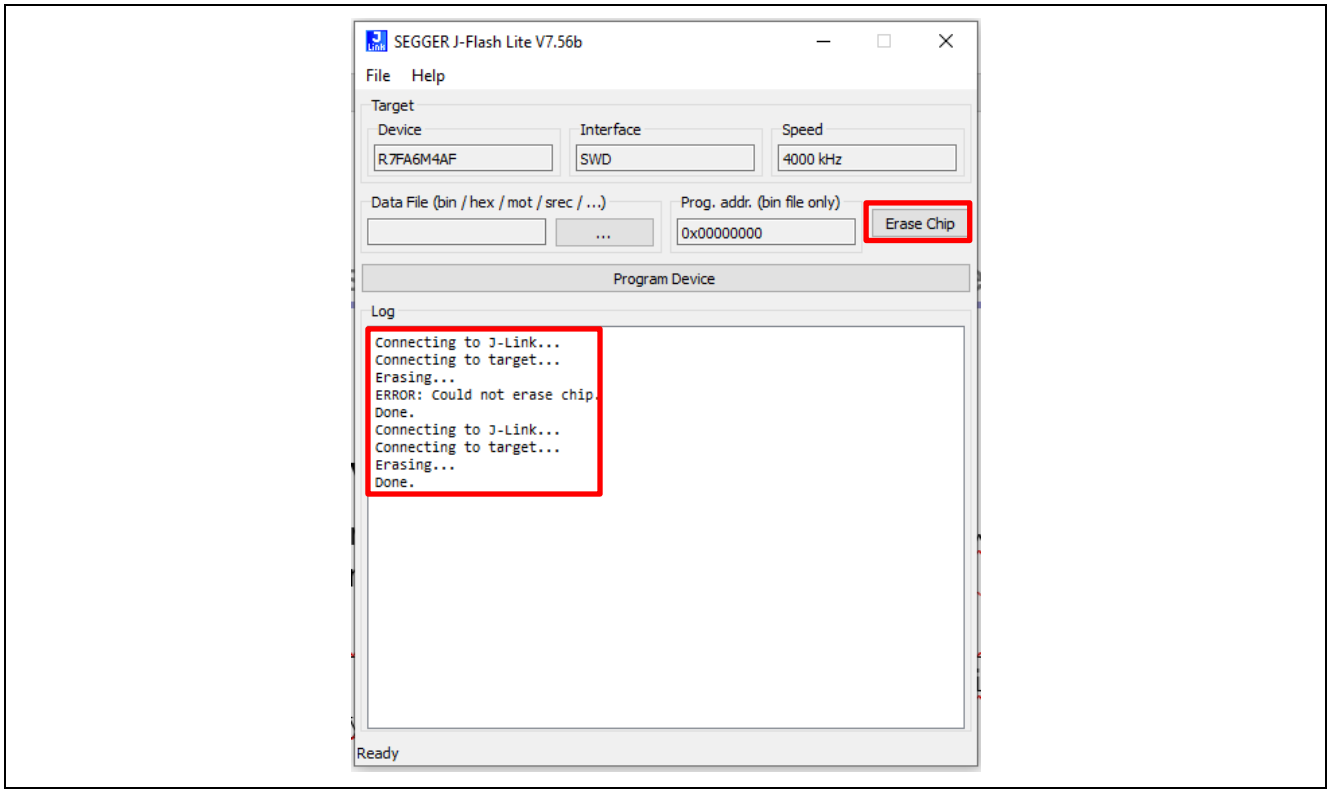

図 63　J-Flash Lite による MCU 消去

Segger J-Flash Lite 7.56b 以前のバージョンを使用する際に、すでにデュアルモードになっている場合は、消去操作を 2 回実行する必要がありますのでご注意ください。この問題は以降の J-Flash Lite のバージョンで修正される可能性があります。

6.3.3　ルネサスデバイスパーティションマネージャの使用方法

Renesas Device Partition Manager の使用にあたって、まずデバッグセッション終了後に評価ボード EK-RA6M4 の電源を入れ直します。e² studio 内で、[実行] > [Renesas Debug Tools] > [Renesas Device Partition Manager]に移動します。接続方法として[J-Link]を選択し、[Action]から[Initialize device back to factory default]を選択します。

[Run]をクリックすると、MCU は消去されます。

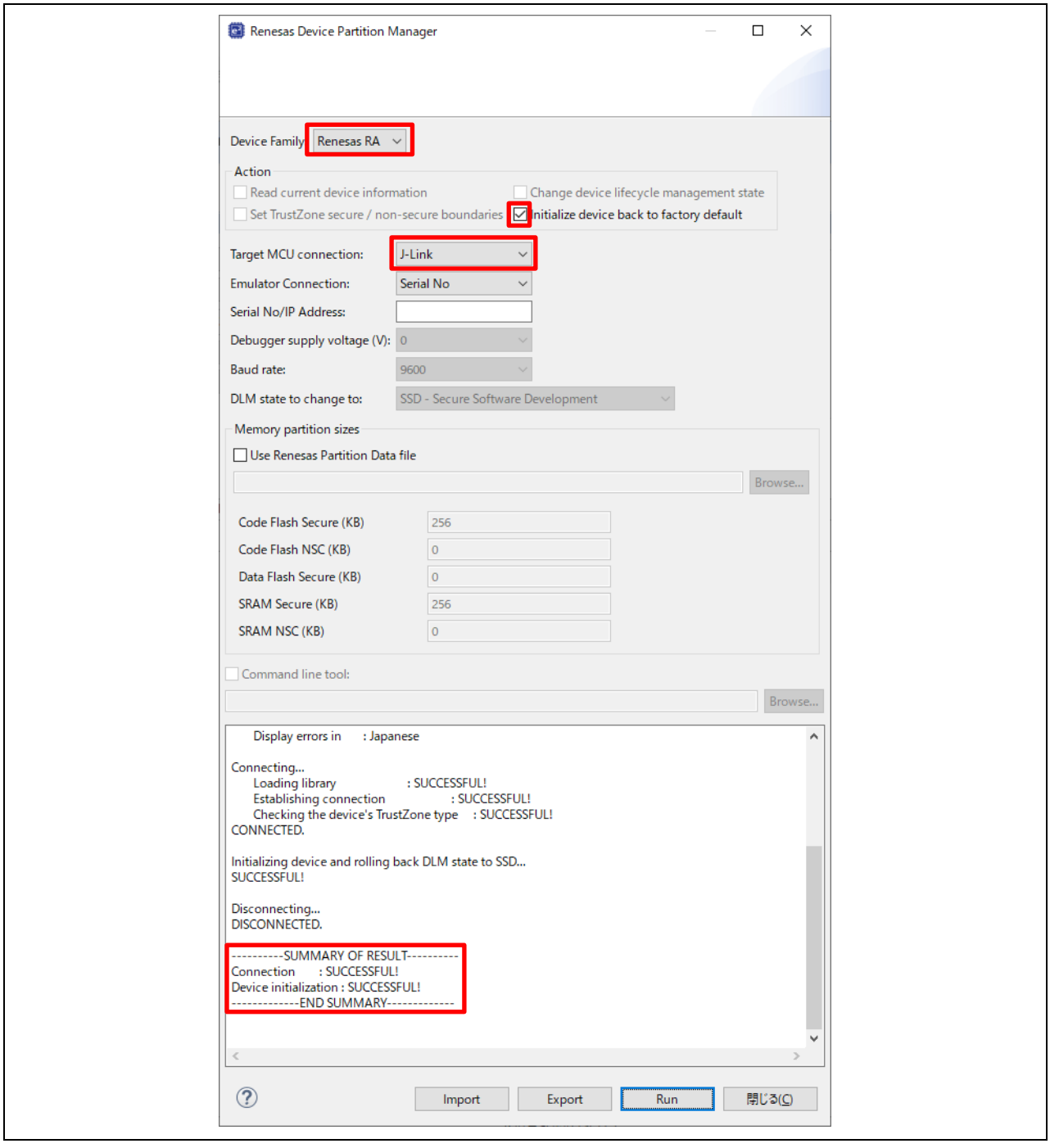

図 64　Renesas Device Partition Manager による MCU 消去

6.4　上側バンクのブートローダの書き込み

上側バンクのブートローダを最初に書き込みできます。MCU は 0x0 番地から実行を開始するので、上側バンクのブートローダは書き込みされた後、実行されません。従って、MCU はリニアモードのままになります。上側バンクのブートローダは、RFP または J-Flash Lite を使用して書き込みできます。

6.4.1　RFP を用いた上側バンクのブートローダの書き込み

セクション 6.3 に従って MCU を消去した後、セクション 6.3.1 に従い、新規 RFP プロジェクトを起動して、[参照]からセクション 4.7 で生成した `ra_mcuboot_ra6m4_dualbank_offset.srec` を選択して MCU に書き込むことが可能です。

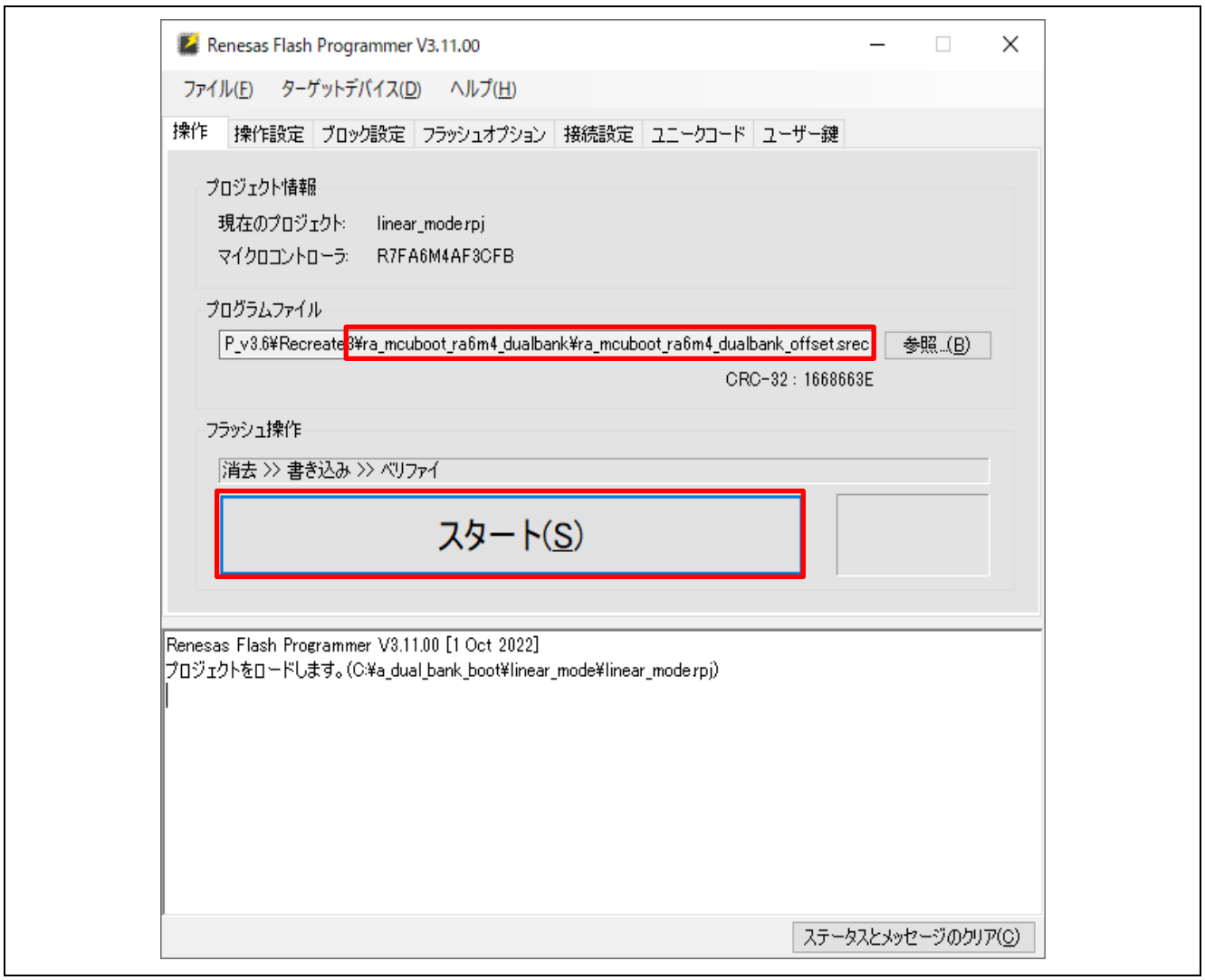

図 65　RFP による上側バンクブートローダ書き込み

[スタート]をクリックすると、上側バンクのブートローダに書き込まれます。

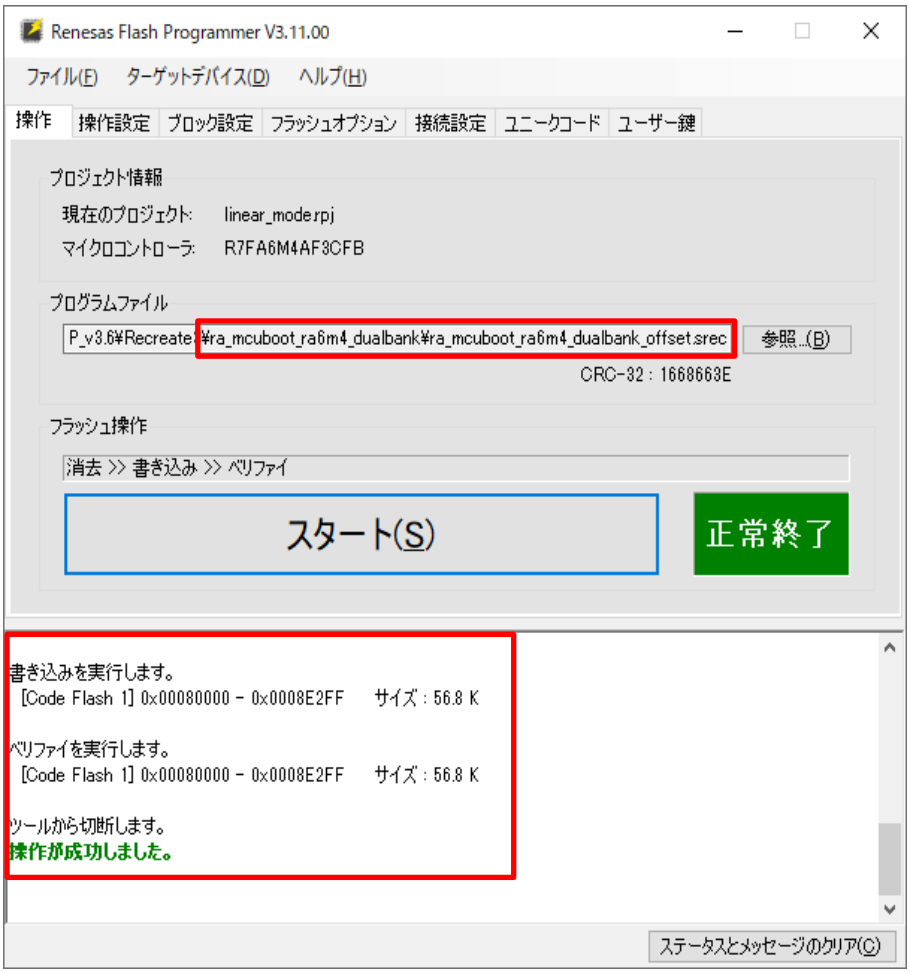

図 66　RFP により正常に書き込まれた上側バンクブートローダ

6.4.2　J-Flash Lite を用いた上側バンクのブートローダの書き込み

セクション 6.3 に従って MCU を消去した後、セクション 6.3.2、6.4.1 に従って J-Flash Lite を起動して、[参照]からセクション 4.7 で生成した `ra_mcuboot_ra6m4_dualbank_offset.srec` を選択して MCU に書き込むことができます。

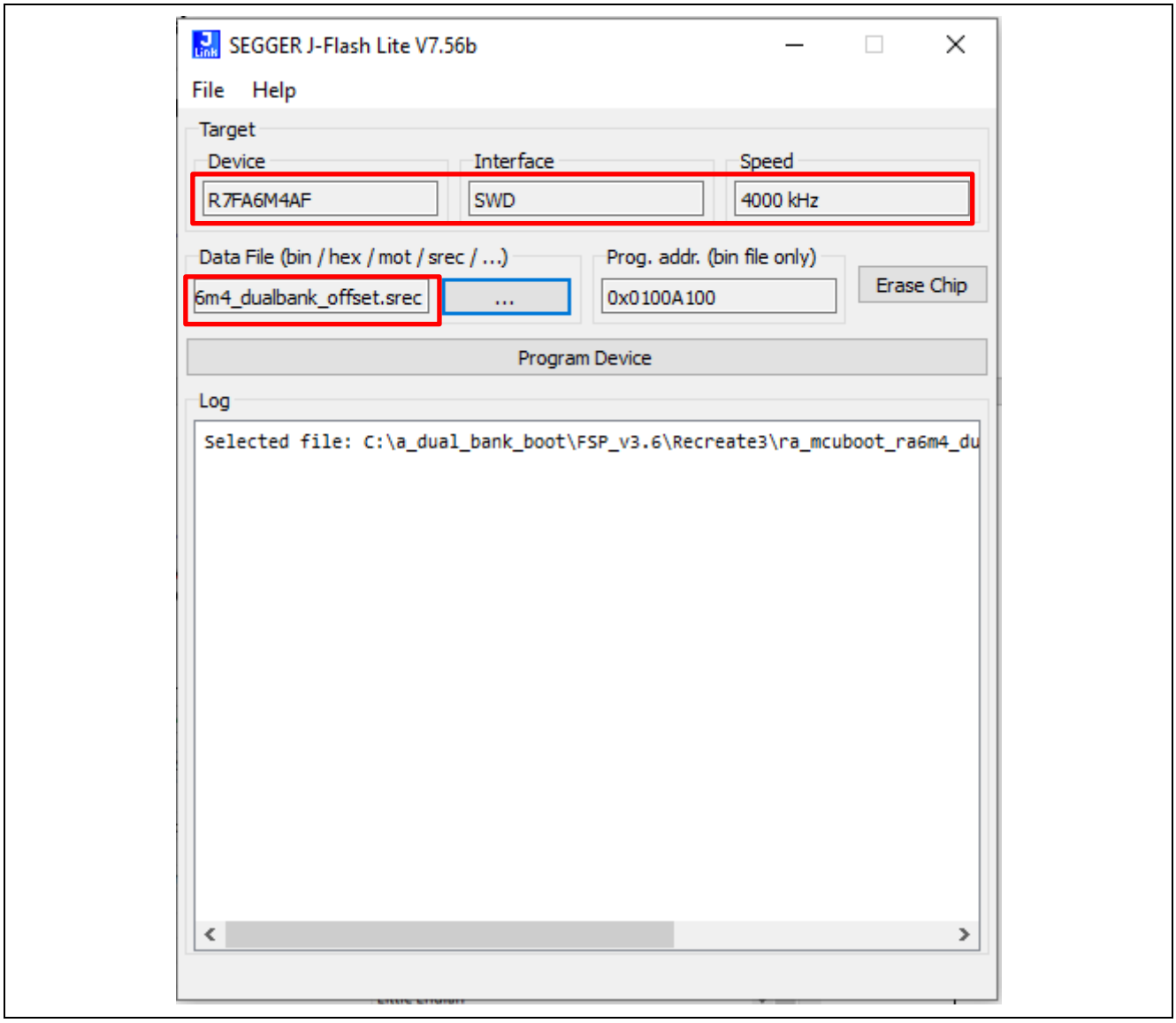

図 67　J-Flash Lite による上側バンクブートローダ書き込み

[Program Device]をクリックして、上側バンクのブートローダを書き込みします。

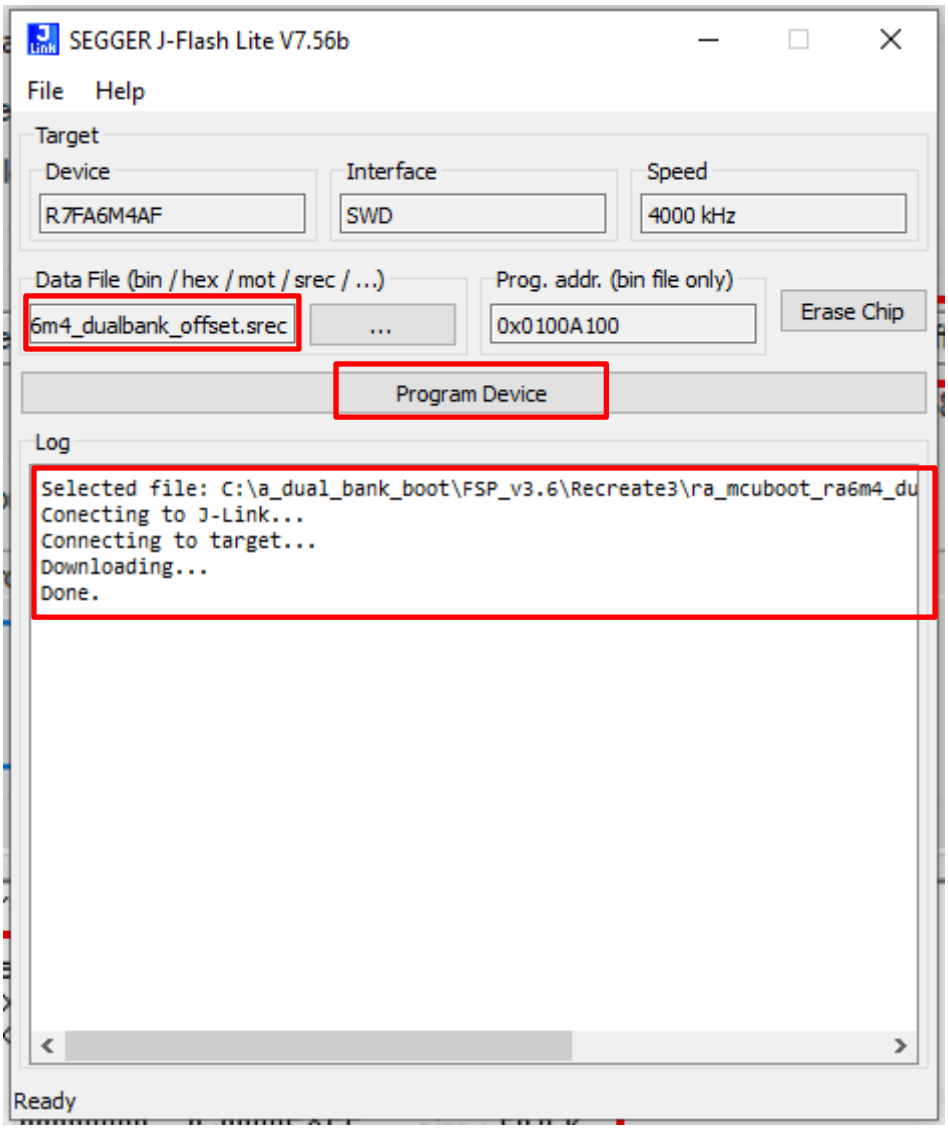

図 68　J-Flash Lite により正常に書き込まれた上側バンクブートローダ

6.5　デバッグセッションの開始

以下の手順に従って、デバッグセッションを開始します。

1. デバッガの設定からフラッシュコンテンツキャッシュを無効にする。
   プロジェクト[app_primary_usb]を右クリックし、[デバッグ] > [デバッグの構成]、[Debugger]->[Debug Tool Settings]に移動し、[Allow caching of flash contents]のチェックを外してください。この設定を行わなかった場合、ブートローダアプリケーションのメモリウィンドウ情報をデバッグ時、誤った情報が表示される場合があります。

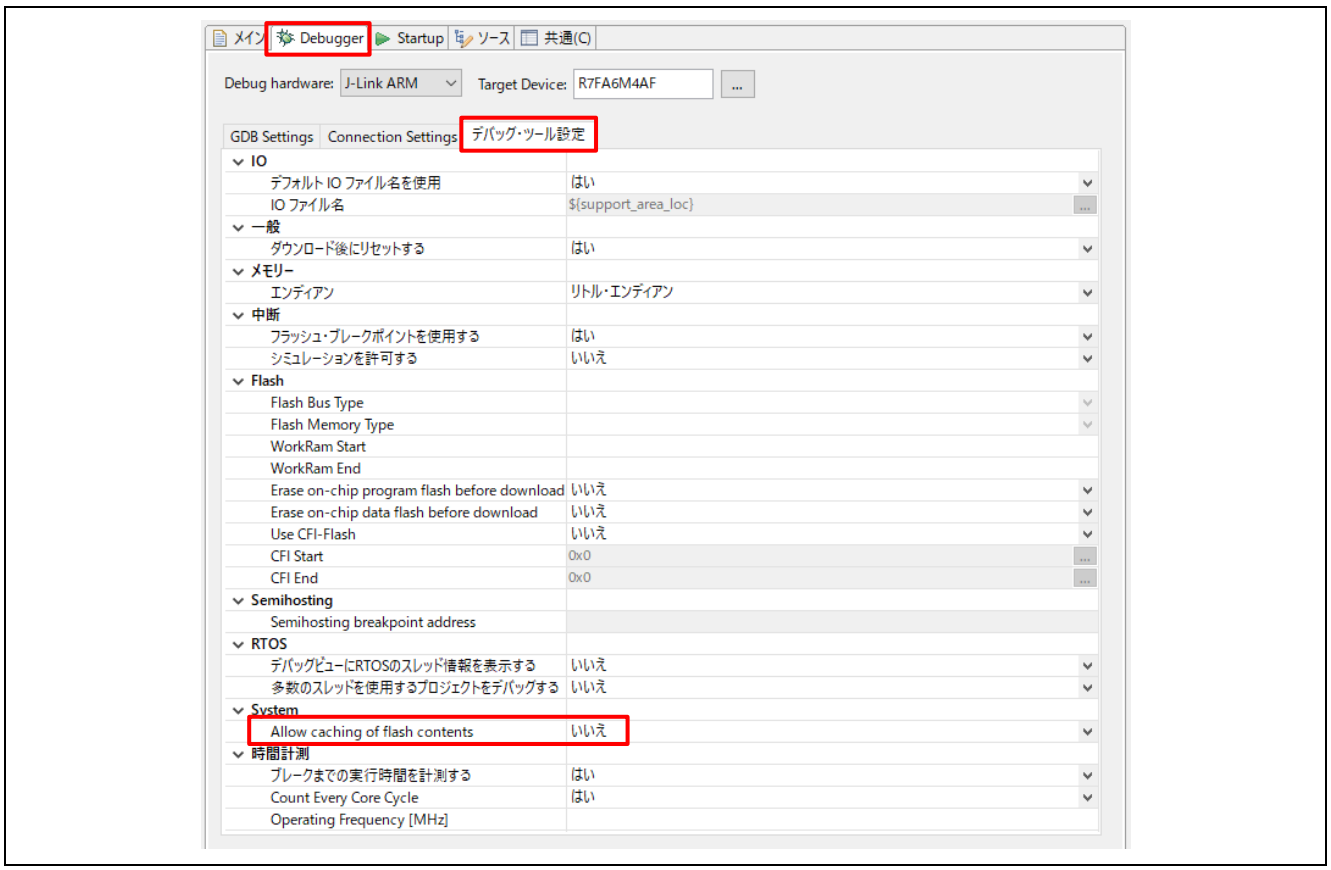

図 69　フラッシュコンテンツキャッシュ無効

2. ロードイメージとシンボルのプロパティの設定
   [デバッグの構成]を開く：[app_primary_usb] > [デバッグ] > [デバッグの構成]
   [App_primary_usb] [Debug_Flat]が選択されていることを確認し、[Startup]タブを選択します。
   [追加]をクリックして、[ワークスペース]を選択して、[ra_mcuboot_ra6m4_dualbank]プロジェクトに移動し、[Debug]フォルダから[ra_mcuboot_ra6m4_dualbank.elf]ファイルを選択します。[OK]をクリックします。

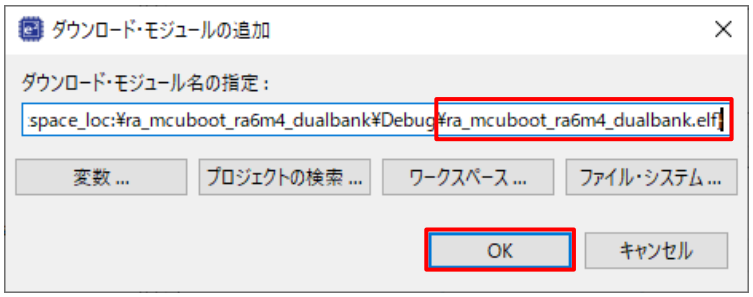

図 70　デバッグ設定にブートローダプロジェクト追加

[ロード・タイプ]のセルをクリックし、ドロップダウンメニューから[シンボルのみ]を選択して、[app_primary_usb]プロジェクトのプログラム・バイナリーのロードタイプを[シンボルのみ]に変更します。

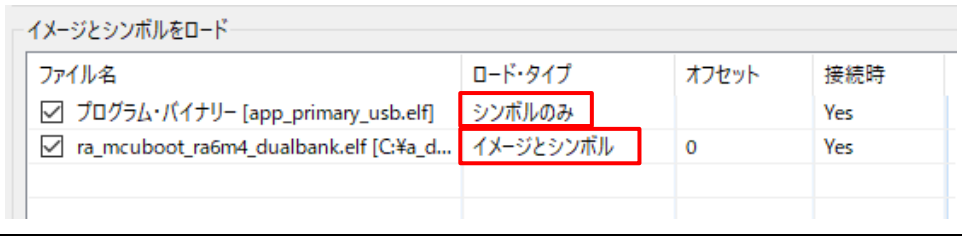

図 71　アプリケーションプロジェクトシンボルのみ読み込み選択

[デバッグ]をクリックします。デバッガは、ブートローダのリセットハンドラーに到達していることが確認できます。

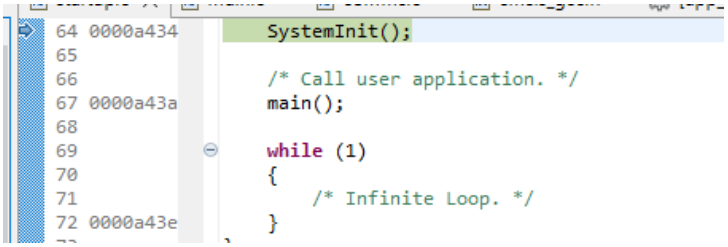

図 72　アプリケーション実行開始

[常にこの設定を使用する]を選択し、ポップアップが表示されたら[切り替え]をクリックしてパースペクティブを切り替えます。

図 73　パースペクティブ切り替え

6.6　プライマリアプリケーションのダウンロード

この時点では、ブートローダのイメージのみがフラッシュにダウンロードされています。次に、[ファイルのロード]ボタンを使用してアプリケーションイメージをダウンロードします。e² studio ツールバーの上部の アイコンをクリックし、署名付きイメージ`\Debug\app_primary_usb.bin.signed` file ファイルを選択します。

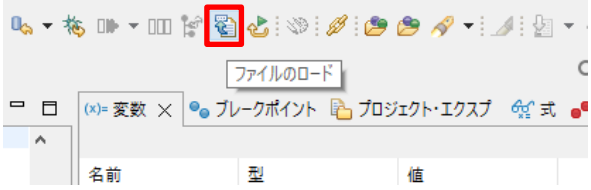

図 74　ファイルロード

[バイナリー・イメージとしてロードする]をチェックし、図 26 の[Bootloader Flash Area Size]とブートローダプロジェクトが使用するサイズに基づいて、アドレスを設定します。図 75 に示すように、プライマリイメージを 0x20000 にダウンロードします。[OK]押してイメージをダウンロードします。

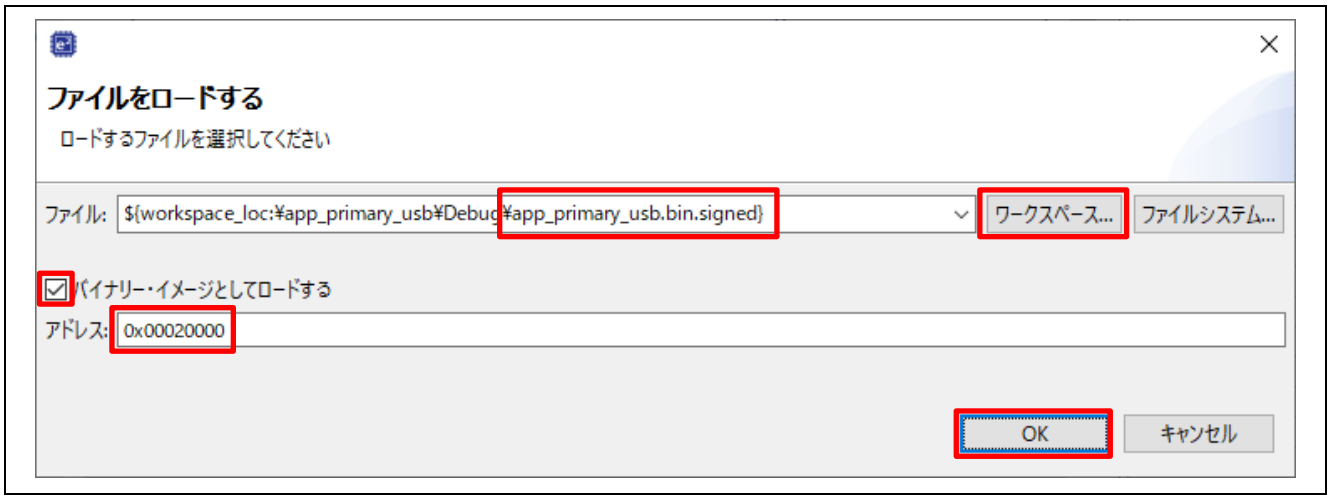

図 75　アプリケーションイメージダウンロードアドレス設定

6.7　プライマリアプリケーションの起動

[再開] をクリックしてプロジェクトを実行します。

プログラムは、ブートローダ `main` 関数の `hal_entry()` 呼び出しで一時停止していることが確認できます。

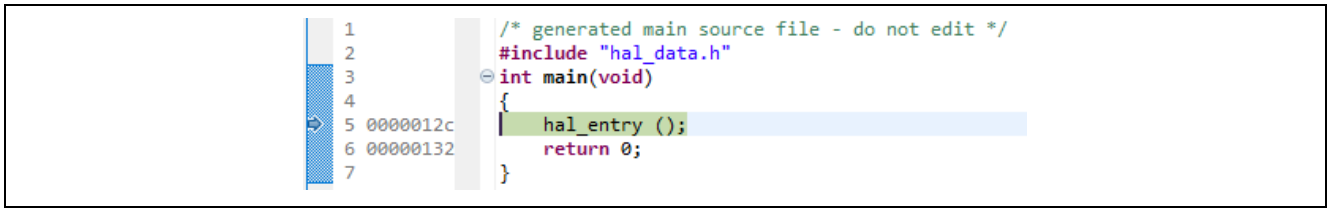

図 76　アプリケーション実行開始

をクリックして再度実行します。本アプリケーションにより、EK-RA6M4 上の赤、青、緑の LED が点滅しているはずです。

デバッグセッションを停止し、ボードの電源を入れ直してください。

6.8　プライマリアプリケーションのダウンローダによる新しいアプリケーションの更新

次の手順に従って、セクション 6.1 で作成した新しいアプリケーションを書き込みます。

1. Tera Term を開き、USB シリアルポートを選択し(COM 番号は環境により異なります)、[OK]をクリックします。

図 77　COM ポート選択

図 78 のメニューが、Tera Term に表示されます。

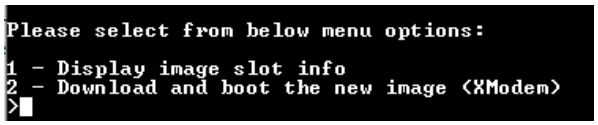

図 78　Tera Term メニュー

UART インタフェースを使用する場合は、シリアルターミナルを選択し、[ボー・レート]を 115200 に設定することに注意してください。USB インタフェースを使用する場合は、この手順をスキップしてください。

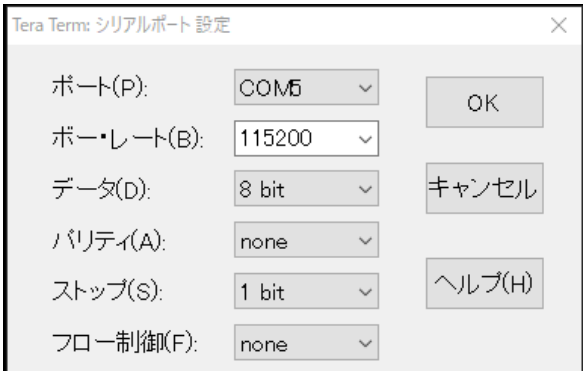

図 79　UART インタフェース使用時のボー・レート設定

2. オプション 1 を選択すると、イメージスロット情報が表示されます。

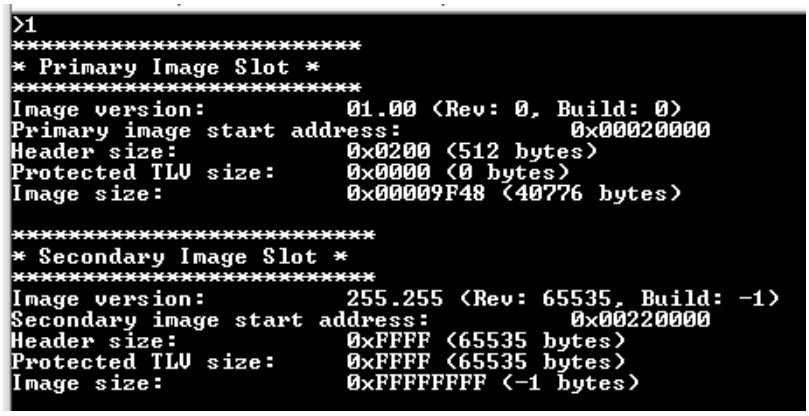

図 80　イメージスロット情報表示

3. オプション 2 を選択し、プライマリイメージのダウンローダを使用してセカンダリイメージをダウンロードします。

図 81　オプション 2 選択 XModem 経由の新規イメージダウンロード

ここで、Tera Term の[転送]を開きます。

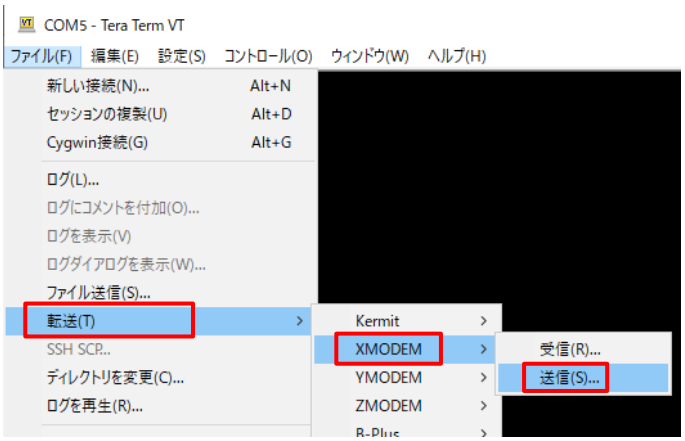

図 82　Tera Term 転送開始

\app_secondary_usb\Debug\app_secondary_usb.bin.signed を選択し、[開く]をクリックします。

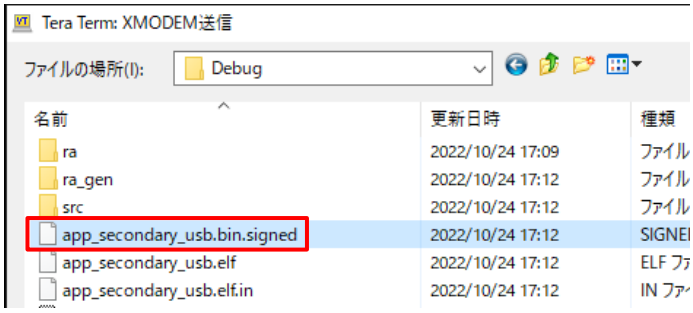

図 83　署名付きセカンダリイメージ選択

その後、セカンダリイメージをダウンロードし、上位バンクに書き込みます。

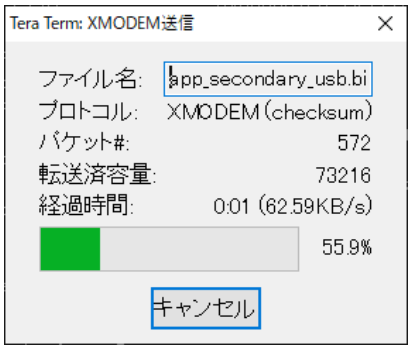

図 84　Xmodem による新規イメージダウンロード

6.9　新規アプリケーションの起動

新しいイメージがダウンロードされた後、システムは自動的に再起動します。

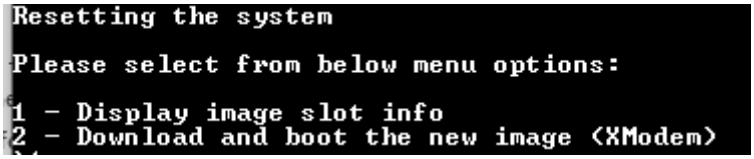

図 85　新規イメージ起動

オプション 1 を選択すると、スワップされたメモリレイアウトを読み込むことができます。

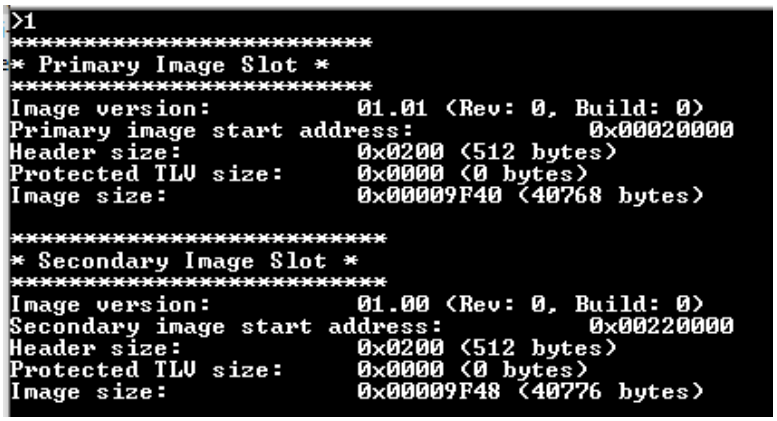

図 86　新規イメージ起動後スロットレイアウト

セカンダリイメージが起動されても、デバッガにダウンロードされたシンボルはプライマリイメージ用であるため、デバッグできないことに注意してください。

また、ユーザがさらに更新を行う場合、新しいイメージはプライマリスロットの現在のイメージより新しいバージョンにする必要があります。

7.　製品化に関する留意点

このセクションでは、製品化プロセスの 1 例を説明します。お客様は、可能な範囲内でこの手順を必要に応じて変更してください。

7.1　フラッシュのブロック保護によるブートローダの保護

セキュアブートローダは、システムの Root of Trust 保護のため、アプリケーションによる変更から保護されている必要があります。図 35 によると、ブートローダは最初の 64KB の領域に配置されています。図 3 によると、保護が必要なブロックは、下側バンクが 0～8 ブロック、上側バンクが 70～78 ブロックとなります。

ユーザは、`ra_mcuboot_ra6m4_dualbank` プロジェクトの[BSP]タブで、これらのブロックを一時的に保護するように設定することができます。ブロックが一時的に保護されている場合、セクション 6.3 で説明されている MCU 消去操作を行うことで、ブロックの保護設定をリセットすることができます。

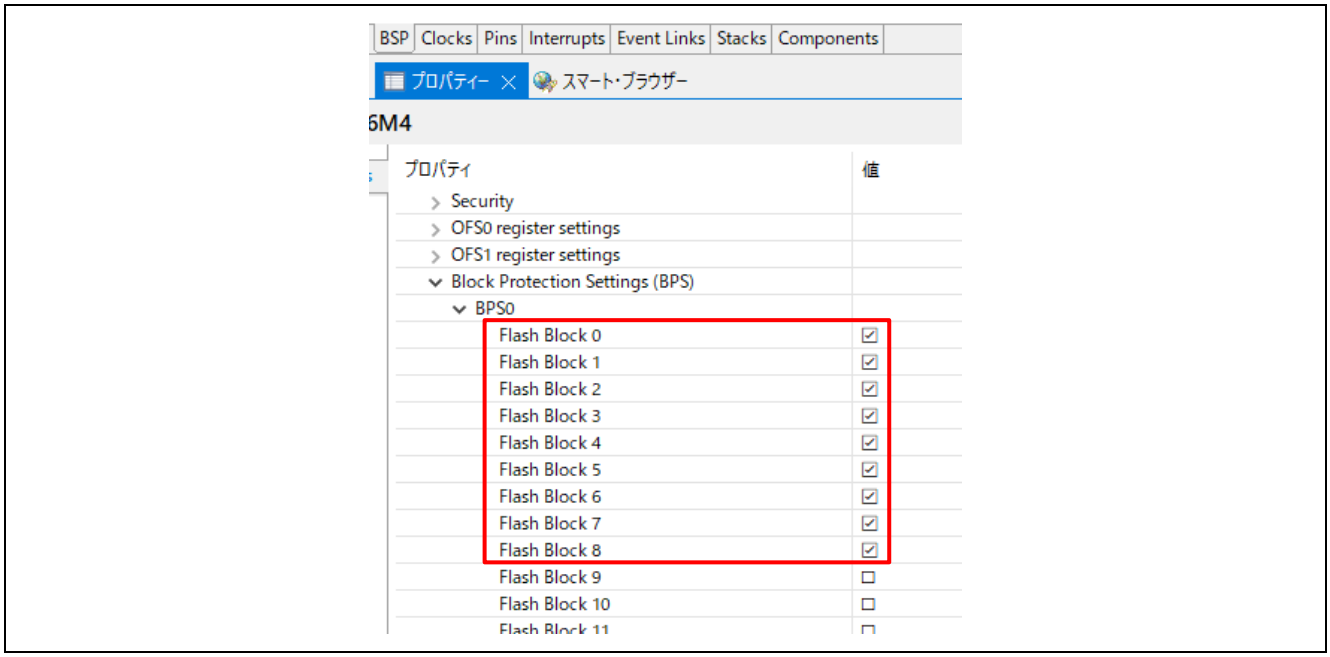

図 87　下側バンクブートローダ領域一時保護

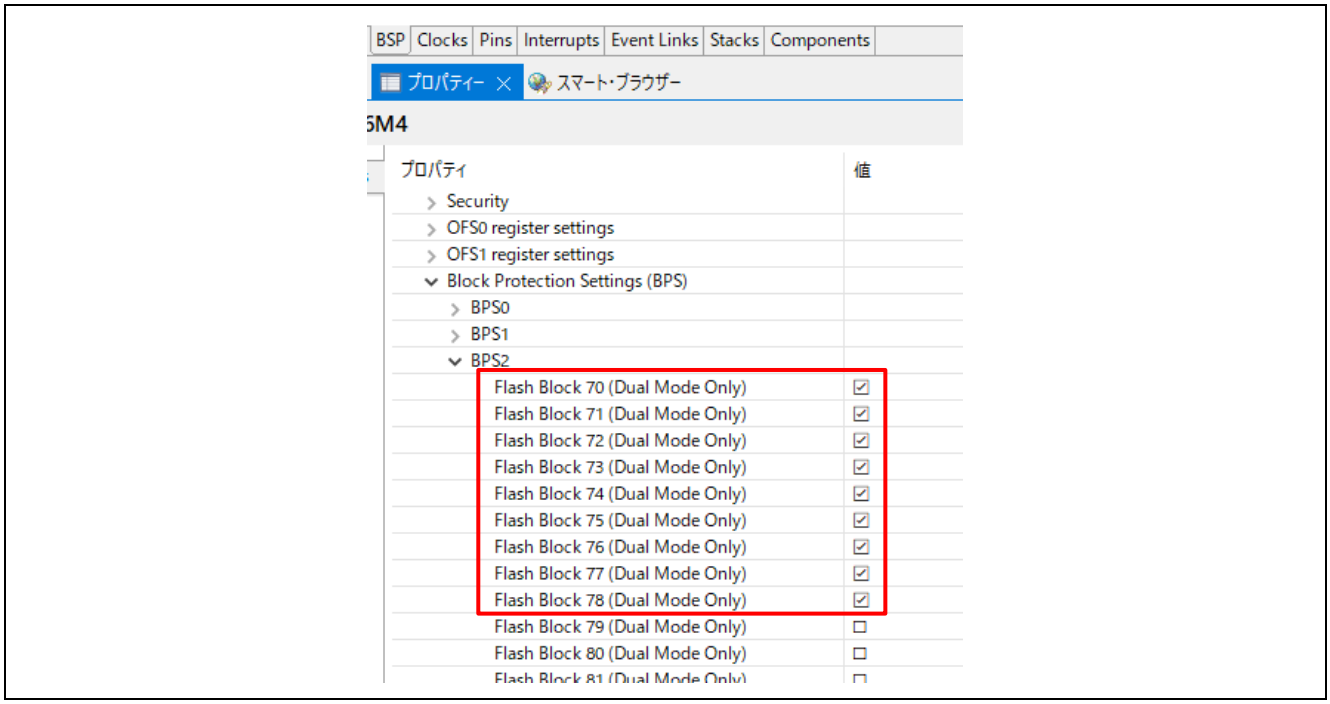

図 88　上側バンクブートローダ領域一時保護

ユーザは、ra_mcuboot_ra6m4_dualbank プロジェクトの[BSP]タブで、これらのブロックを永久に保護するようにも設定できます。

これらのブロックが永久保護されている場合、これらのブロックは MCU の寿命を通じて消去および再書き込みできません。永久保護を設定するときは十分に注意してください。セクション 6.3 で説明されている MCU の消去方法でも、これらのブロックを消去することはできません。

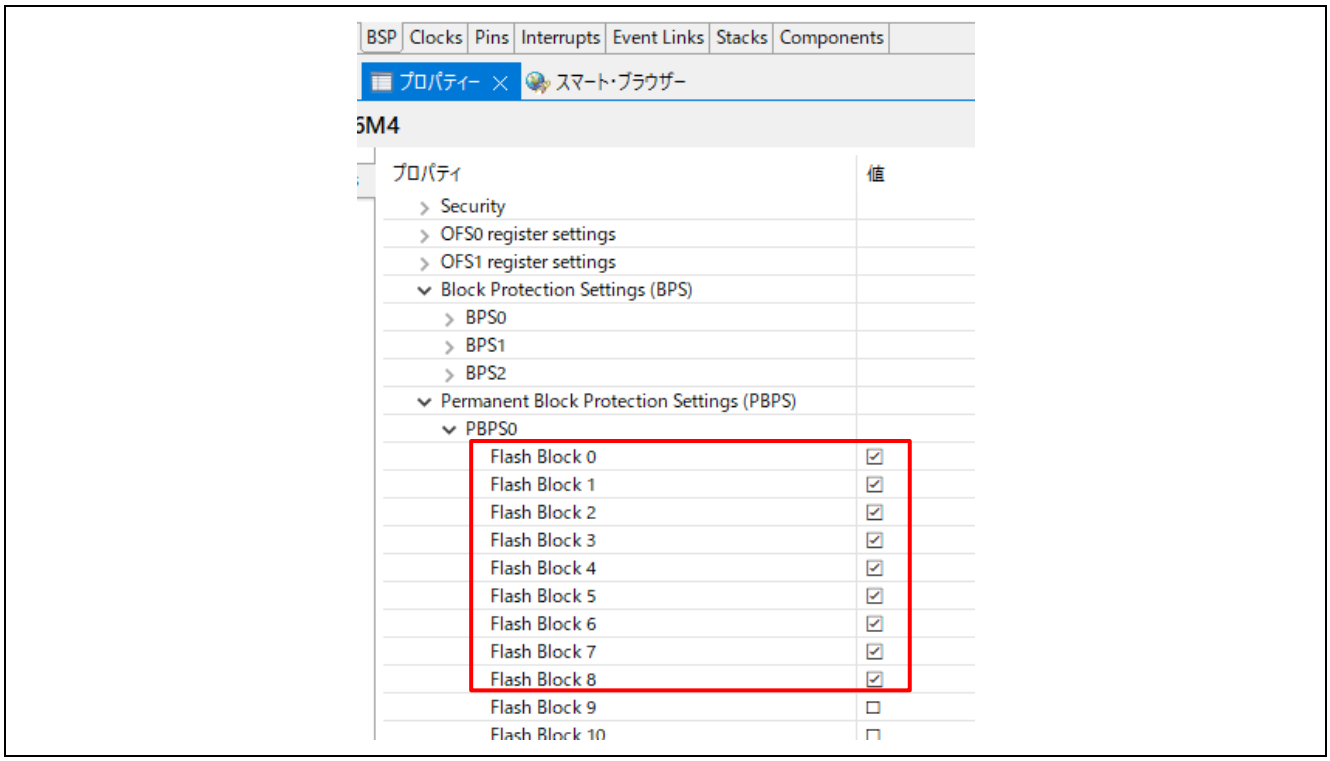

図 89　下側バンクブートローダ領域永久保護

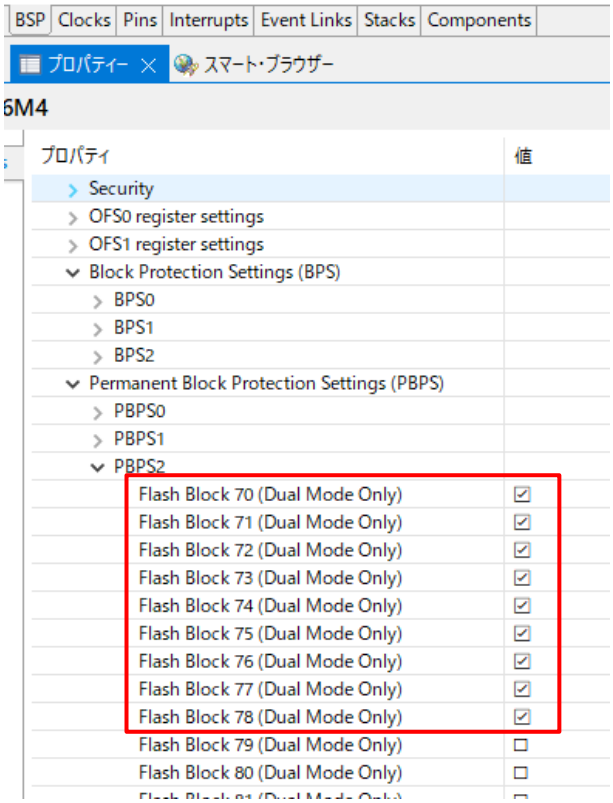

図 90　上側バンクブートローダ領域永久保護

付属のサンプルブートローダには、ブロックの保護を有効にするブロック設定が含まれていないので、ユーザは、リリース前に有効にすることができます。

7.2　ブートローダと最初のアプリケーションの MCU への実装

ユーザは、上記のセクションで生成された.srec ファイルを 1 つに結合し、製造時に MCU に書き込みすることができます。

以下の 3 つのイメージを含む合成.srec ファイルを作成します。

1. 下側バンク用ブートローダ　　: ra_mcuboot_ra6m4_dualbank.srec
2. 上側バンク用ブートローダ　　: ra_mcuboot_ra6m4_dualbank_offset.srec
3. 下側バンク用アプリケーション　: app_primary_usb_signed_offset.srec

以下のコマンドは、上記 3 つの.srec ファイルを 1 つに結合した.srec を生成するために使用できます。

```
srec_cat ra_mcuboot_ra6m4_dualbank.srec ra_mcuboot_ra6m4_dualbank_offset.srec
app_primary_usb_signed_offset.srec -o combined.srec
```

セクション 6.4 で説明した ra_mcuboot_ra6m4_dualbank_offset の書き込みと同じ方法で、RFP または J-Flash Lite を使用して、combined.srec を MCU にダウンロードします。

デバイスのリリース後は、アプリケーションプロジェクトに実装されたイメージダウンローダを介してアプリケーションの更新ができます。

8. 付属のブートローダとアプリケーションサンプルプロジェクトのコンパイルと実行

8.1 ダウンロードインタフェースとして USB を使用する場合

USB インタフェースでは、次の 3 つのプロジェクトが必要です。

- ra_mcuboot_ra6m4_dualbank
- app_primary_usb
- app_secondary_usb

ユーザは、次の手順に従い以下のフォルダにあるサンプルプロジェクトを実行できます。

\RA6_Secure_Bootloader_Dualbank\example_projects_with_bootloader

1. セクション 6.2 の説明に従って、ハードウェアをセットアップします。
2. 前述した 3 つのプロジェクトをワークスペースにインポートします。
3. プロジェクト ra_mcuboot_ra6m4_dualbank から Configuration.xml ファイルを開きます。
4. [Generate Project Content]をクリックします。
5. プロジェクト ra_mcuboot_ra6m4_dualbank をコンパイルします。
6. プロジェクト app_primary_usb の Configuration.xml ファイルを開きます。
7. [Generate Project Content]をクリックします。
8. app_primary_usb をコンパイルします。
9. プロジェクト app_secondary_usb の Configuration.xml ファイルを開きます。
10. [Generate Project Content]をクリックします。
11. app_secondary_usb プロジェクトをコンパイルします。
12. セクション 6.3 に従って、チップ全体を消去します。
13. セクション 6.4 に従って、ra_mcuboot_ra6m4_dualbank_offset.srec を RFP または J-Flash Lite を使って MCU にダウンロードします。
14. e^2 studio 環境で、プロジェクト app_primary_usb からアプリケーションをデバッグします。
15. デバッグセッションを一時停止します。
16. [ファイルのロード]メニューオプションを使用して、0x20000 番地に app_primary_usb.bin.signed をダウンロードします。これには数秒程度かかります。
17. プログラムの実行を再開します。3 つの LED がすべて点滅することが確認できます。
18. デバッグセッションを終了し、EK-RA6M4 の電源を入れ直します。
19. COM ポート(USB シリアルデバイス)を選択して、Tera Term を起動する。
20. Tera Term を使用して、セクション 6.8 の手順に従い、MCU へ \app_secondary_usb\Debug\app_secondary_usb.bin.signed を送信します。
21. ダウンロード後、自動的にシステムがリセットされます。
22. 青と緑の LED が点滅することが確認できます。
23. メニュー項目 1 を入力し、バージョン 1.1.0 のイメージがプライマリスロット(下側バンク)に、バージョン 1.0.0 のイメージがセカンダリスロット(上側バンク)に配置されていることを確認します。

8.2　ダウンロードインタフェースとして UART を使用する場合

UART インタフェースでは、次の 3 つのプロジェクトが必要です。

- ra_mcuboot_ra6m4_dualbank
- app_primary_uart
- app_secondary_uart

ユーザは、次の手順に従い以下のフォルダにあるサンプルプロジェクトを実行できます。

\RA6_Secure_Bootloader_Dualbank\example_projects_with_bootloader.

1. セクション 6.2 の指示に従って、ハードウェアを設定します。
2. 前述の 3 つのプロジェクトをワークスペースにインポートします。
3. プロジェクト ra_mcuboot_ra6m4_dualbank の Configuration.xml ファイルを開きます。
4. [Generate Project Content]をクリックします。
5. プロジェクト ra_mcuboot_ra6m4_dualbank をコンパイルします。
6. プロジェクト app_primary_uart の Configuration.xml ファイルを開きます。
7. [Generate Project Content]をクリックします。
8. app_primary_uart をコンパイルします。
9. プロジェクト app_secondary_uart の Configuration.xml ファイルを開く。
10. [Generate Project Content]をクリックします。
11. app_secondary_uart プロジェクトをコンパイルします。
12. セクション 6.3 に従って、チップ全体を消去します。
13. セクション 6.4 に従って、ra_mcuboot_ra6m4_dualbank_offset.srec を RFP または J-Flash Lite を使って MCU にダウンロードします。
14. e² studio 環境で、プロジェクト app_primary_uart からアプリケーションをデバッグします。
15. デバッグセッションを一時停止します。
16. [ファイルのロード]メニューオプションを使用して、0x20000 番地に app_primary_uart.bin.signed をダウンロードします。
17. プログラムの実行を再開します。3 つの LED がすべて点滅することが確認できます。
18. デバッグセッションを終了し、EK-RA6M4 の電源を入れ直します。
19. COM ポートを選択して、Tera Term を起動して、[ボー・レート]を[115200]に設定します。
20. Tera Term を使用して、セクション 6.8 の手順に従い、MCU へ \app_secondary_uart\Debug\app_secondary_uart.bin.signed を送信します。これには 20 秒程度かかります。
21. ダウンロード後、自動的にシステムがリセットされます。
22. 青と緑の LED の点滅を確認してください。
23. メニュー項目 1 を入力し、バージョン 1.1.0 のイメージがプライマリスロット(下側バンク)に、バージョン 1.0.0 のイメージがセカンダリスロット(上段バンク)に配置されていることを確認します。

9.　参考文献

1. *Renesas RA Family MCU Securing Data at Rest using Security MPU Application Project* (R11AN0416)
2. *Using MCUboot with RA Family MCUs Application Project* (R11AN0497)

10. ウェブサイトとサポート

RA マイクロコントローラファミリの詳細、ツールやドキュメントのダウンロード、サポートについては、以下の URL を参照してください

EK-RA6M4 設計情報	renesas.com/ra/ek-ra6m4
RA 製品紹介	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA サポートフォーラム	renesas.com/ra/forum
Renesas サポート	renesas.com/support

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Mar.27.23	ー	初版発行
1.01	Feb.13.25	ー	誤記修正 （本資料は文書番号 R11AN0570JJ0100 の誤記修正版です）

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。