RX ファミリ

MCUboot Firmware Integration Technology

要旨

本アプリケーションノートは Firmware Integration Technology (FIT)を使用した、MCUboot モジュールについて説明します。以降、本モジュールを MCUboot FIT モジュールと称します。

MCUboot FIT モジュールは、GitHub の mcu-tools のページ（https://github.com/mcu-tools/mcuboot）にて公開されている MCUboot V2.1.0 を FIT モジュール化したものです。

本アプリケーションノートでは、MCUboot FIT モジュールの使用方法、およびユーザアプリケーションへの組み込み方法について説明します。

また、本アプリケーションノートのリリースパッケージにはデモプロジェクトが含まれています。「4.デモプロジェクト」に記載する手順に沿ってデモの実行環境を構築することで、MCUboot FIT モジュールの基本的な動作を確認することができます。

動作確認デバイス

RX261 グループ

RX65N、RX651 グループ

RX66N グループ

RX671 グループ

RX72M グループ

RX72N グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

関連アプリケーションノート

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

- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- RX ファミリ e² studio に組み込む方法 Firmware Integration Technology(R01AN1723)
- RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology(R01AN1685)
- RX ファミリ フラッシュモジュール Firmware Integration Technology(R01AN2184)
- RX ファミリ TSIP(Trusted Secure IP) モジュール Firmware Integration Technology(R20AN0371)
- RX ファミリ RSIP(Renesas Secure IP)モジュール Protected Mode Firmware Integration Technology (R20AN0748)
- RX ファミリ SCI モジュール Firmware Integration Technology(R01AN1815)
- RX ファミリ バイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology (R01AN1683)

ターゲットコンパイラ

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

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

目次

1. 概要 5
1.1 MCUboot とは 5
1.2 MCUboot FIT モジュールとは 5
1.3 システム構成 6
1.4 MCUboot の動作 7
1.5 MCUboot の各アップデート方式について 8
1.5.1 Overwrite Only/Only Fast 方式 8
1.5.1.1 Overwrite Only 方式 8
1.5.1.2 Overwrite Only Fast 方式 8
1.5.2 Swap 方式 9
1.5.3 DirectXIP 方式 10
1.5.3.1 リニアモードの DirectXIP 方式 10
1.5.3.2 デュアルモードの DirectXIP 方式 10
1.6 パッケージ構成 11
1.7 API の概要 13

2. API 情報 14
2.1 ハードウェアの要求 14
2.2 ソフトウェアの要求 14
2.3 サポートされているツールチェーン 14
2.4 ヘッダファイル 14
2.5 整数型 14
2.6 コンパイル時の設定 15
2.7 サンプルプロジェクトのコードサイズ 18
2.8 引数 22
2.9 戻り値 22
2.10 FIT モジュールの追加方法 23
2.11 for 文、while 文、do while 文について 24
2.12 API の実装例について 25

3. API 関数 26
3.1 boot_go 関数 26
3.2 RM_MCUBOOT_BootApp 関数 26
3.3 RM_MCUBOOT_GetVersion 関数 26

4. デモプロジェクト 27
4.1 デモプロジェクトの構成 27
4.1.1 初期イメージの詳細 28
4.2 動作環境準備 31
4.2.1 Imgtool の入手 31
4.2.2 ターミナルソフトのインストール 31
4.2.3 Python 実行環境のインストール 31
4.2.4 OpenSSL の実行環境のインストール 32
4.2.5 フラッシュライタのインストール 32
4.2.6 Security Key Management Tool のインストール 32

4.2.7 USB シリアル変換ボード ... 32
4.3 デモプロジェクトの実行手順 ... 33
4.3.1 鍵のインジェクション ... 33
4.3.1.1 Security Key Management Tool による鍵データの生成 ... 33
4.3.1.2 鍵インジェクションの準備 ... 35
4.3.1.3 鍵インジェクションプログラムの実行 ... 36
4.3.2 署名検証用公開鍵の埋め込み ... 38
4.3.3 デモプロジェクトのイメージの準備 ... 39
4.3.3.1 ブートローダのイメージを生成 ... 39
4.3.3.2 初期イメージを生成 ... 40
4.3.3.3 更新イメージを生成 ... 41
4.3.4 デモプロジェクトの書き込み ... 43
4.3.5 デモプロジェクトの実行 ... 47

5. 付録 ... 48
5.1 動作確認環境 ... 48
5.2 デモプロジェクトの動作環境 ... 50
5.2.1 RX261 の動作確認環境 ... 50
5.2.1.1 機器接続情報 ... 50
5.2.1.2 メモリ配置およびコンフィグ設定値 ... 51
5.2.2 RX65N の動作確認環境 ... 52
5.2.2.1 リニアモードを使用した方式の機器接続情報 ... 52
5.2.2.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 53
5.2.2.3 デュアルモードを使用した方式の機器接続情報 ... 54
5.2.2.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 55
5.2.3 RX671 の動作確認環境 ... 56
5.2.3.1 リニアモードを使用した方式の機器接続情報 ... 56
5.2.3.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 57
5.2.3.3 デュアルモードを使用した方式の機器接続情報 ... 58
5.2.3.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 59
5.2.4 RX72M の動作確認環境 ... 60
5.2.4.1 リニアモードを使用した方式の機器接続情報 ... 60
5.2.4.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 61
5.2.4.3 デュアルモードを使用した方式の機器接続情報 ... 62
5.2.4.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 63
5.2.5 RX72N の動作確認環境 ... 64
5.2.5.1 リニアモードを使用した方式の機器接続情報 ... 64
5.2.5.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 65
5.2.5.3 デュアルモードを使用した方式の機器接続情報 ... 66
5.2.5.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値 ... 67

6. 注意事項 ... 68
6.1 ブートローダ（MCUboot）からアプリケーションへの遷移時の注意事項 ... 68

改訂記録 ... 69

1. 概要

1.1 MCUboot とは

MCUboot は 32 ビットマイクコントローラ向けのセキュアブートローダです。

マイクロコントローラシステム上にブートローダとフラッシュレイアウトの共通インフラストクチャを定義し、アプリケーションのアップデートを簡単に可能にするセキュアブートローダを提供します。

MCUboot は、特定のオペレーティングシステムやハードウェアに依存せず、動作するオペレーティングシステムからのハードウェアポーティングレイヤーに依存します。

MCUboot は GitHub mcu-tools のページ（https://github.com/mcu-tools/mcuboot）にて公開されています。

MCUboot の主なサポート機能は以下の通りです。

・アプリケーションを起動する機能
・アプリケーションを署名検証する機能
・アプリケーションをアップデートまたは切り替える機能
・暗号化されたアプリケーションイメージを復号しアップデートする機能

1.2 MCUboot FIT モジュールとは

MCUboot FIT モジュールは上記の MCUboot V2.1.0 を FIT モジュール化したものです。本 FIT モジュールを使うことで、ユーザにて作成するブートローダに MCUboot を容易に組み込むことができます。

また、MCUboot FIT では、ルネサス製のハードウェアセキュリティ IP（TSIP、RSIP）を利用しているため、ユーザ鍵のセキュアな秘匿やアプリケーションイメージの高速な復号が可能です。

MCUboot FIT モジュールでは以下のアップデート方式に対応しています。

・Overwrite Only 方式（リニアモード）
・Overwrite Only Fast 方式（リニアモード）
・Swap 方式（リニアモード）
・DirectXIP 方式（リニア／デュアルモード）

MCUboot FIT モジュールでは以下の署名検証方式に対応しています。

・ECDSA NIST P-256
・RSA 2048（RSASSA-PSS）：RX261 は未対応
・署名検証無し

1.3 システム構成

MCUboot FIT モジュールを使用したブートローダおよびデモアプリケーションのシステム構成図を図 1-1 に、使用するモジュールの一覧を表 1-1 に示します。

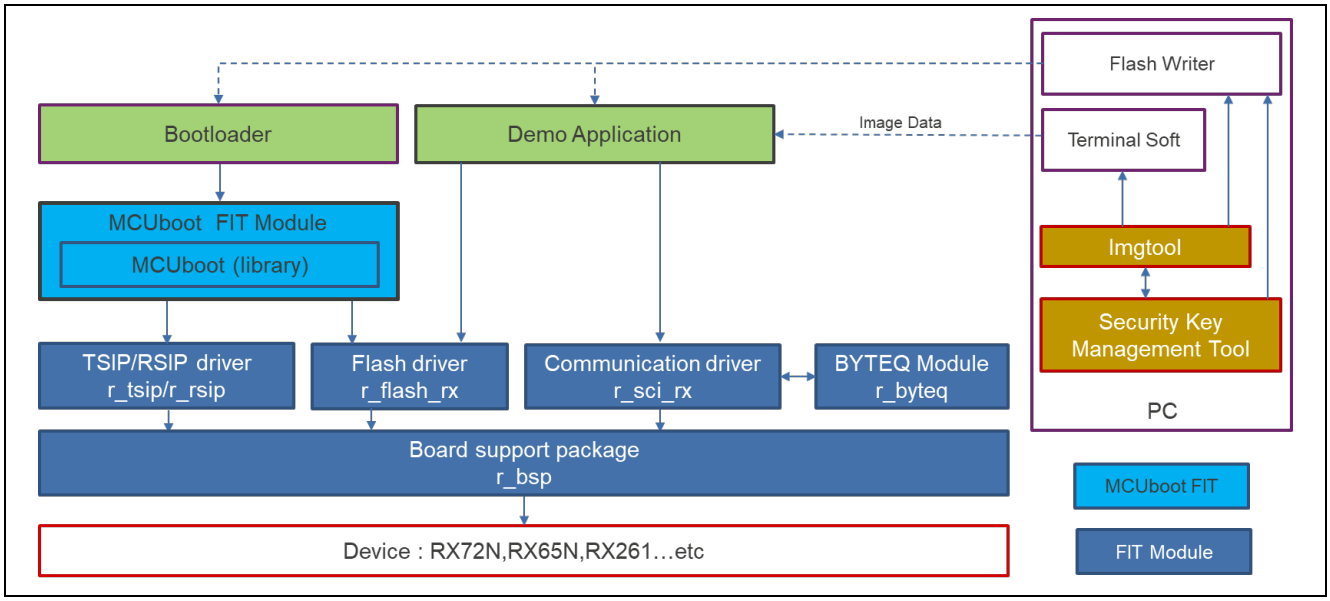

図 1-1 ブートローダおよびデモアプリケーションのシステム構成

表 1-1 ブートローダおよびデモアプリケーションで使用するモジュール一覧

種類	アプリケーションノート名（型名）	FIT モジュール名	備考
BSP	RX ファミリボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)	r_bsp	
デバイスドライバ	RX ファミリフラッシュモジュール Firmware Integration Technology (R01AN2184)	r_flash_rx	
デバイスドライバ	RX ファミリ TSIP(Trusted Secure IP) モジュール Firmware Integration Technology (R20AN0371)	r_tsip	RX65N/RX651/ RX66N/RX671/ RX72M/RX72N で使用
デバイスドライバ	RX ファミリ RSIP(Renesas Secure IP)モジュール Protected Mode Firmware Integration Technology (R20AN0748)	r_rsip_protected_rx	RX261 で使用
デバイスドライバ	RX ファミリ SCI モジュール Firmware Integration Technology (R01AN1815)	r_sci_rx	
ミドルウェア	RX ファミリバイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology (R01AN1683)	r_byteq	

1.4 MCUboot の動作

MCUboot では、マイクコントローラに搭載されるフラッシュメモリを図 1-2 に示すように分割して使用します。

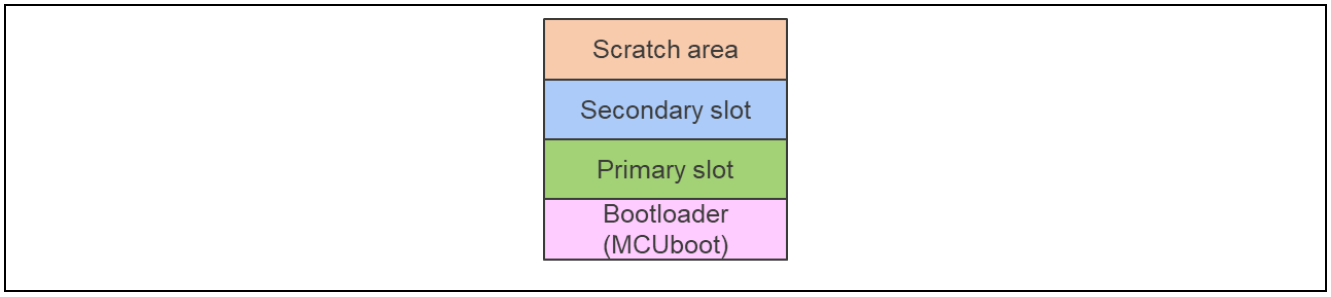

図 1-2 MCUboot のフラッシュメモリのマップイメージ

・Bootloader：MCUboot を使ったブートローダ格納領域

・Primary slot：起動可能イメージ格納領域
（MCUboot により起動されるユーザアプリケーション）

・Secondary slot：アップデート用イメージ格納領域
（Primary slot のイメージをアップデートしたい場合、この領域にアップデート用イメージを配置）

・Scratch area：Swap 方式のアップデート時のみ使用するバッファ領域
（Swap 以外のアップデート方式の場合には不要）

MCUboot の動作は以下の通りです。

① リセット解除後 Bootloader（MCUboot）が起動

② Secondary slot にアップデート用イメージが格納されているか確認

③ Secondary slot にアップデート用イメージが無い場合 → ⑥へ

④ Secondary slot にアップデート用イメージが有る場合、Secondary slot のアップデート用イメージの署名検証を実施

⑤ 検証結果に問題が無ければ選択されたアップデート方式により Primary slot を Secondary slot のアップデート用イメージでアップデートします。（Overwrite Only / Only Fast 方式のみ、アップデート後に Secondary slot を消去します）

⑥ Primary slot のイメージの署名検証を実施

⑦ 検証結果に問題が無ければ Primary slot のイメージ（ユーザアプリケーション）を起動

1.5 MCUboot の各アップデート方式について

MCUboot FIT モジュールでは、MCUboot のアップデート方式の中で以下の方式に対応しています。

・Overwrite Only/Only Fast 方式

・Swap 方式

・DirectXIP 方式

フラッシュメモリのモードとアップデート方式の関係としては、リニアモードは Overwrite Only/Only Fast 方式、Swap 方式、DirectXIP 方式に対応し、デュアルモードは DirectXIP 方式に対応します。

各方式の具体的なアップデート方法に詳細は 1.5.1～1.5.3 を参照してください。

1.5.1 Overwrite Only/Only Fast 方式

Overwrite Only/Only Fast 方式では、起動可能なイメージは常に Primary slot に格納され、その slot から実行されます。また Secondary slot には更新用のイメージが格納されます。

更新用イメージが Secondary slot に格納された場合、Secondary slot の内容を署名検証し問題無ければ、Secondary slot の内容を Primary slot にコピーすることでアップデートを行い、その後 Secondary slot を消去します。

1.5.1.1 Overwrite Only 方式

Overwrite Only 方式では、Secondary slot 全域を Primary slot 全域にコピーすることによりアップデートを行います。

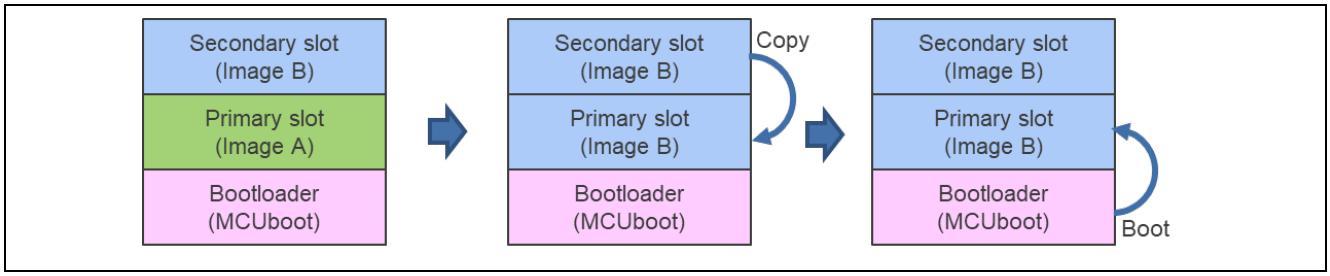

図 1-3 Overwrite Only 方式のアップデート動作

1.5.1.2 Overwrite Only Fast 方式

Overwrite Only Fast 方式では、Secondary slot を Primary slot にコピーすることは Overwrite Only 方式と同じですが、コピーする際、Secondary slot 全域ではなく、更新用イメージのサイズ分のみ Primary slot にコピーし、未使用の領域はコピーの対象としません。これにより、アップデートサイズが小さい場合はアップデートのコピー処理を短縮することができます。

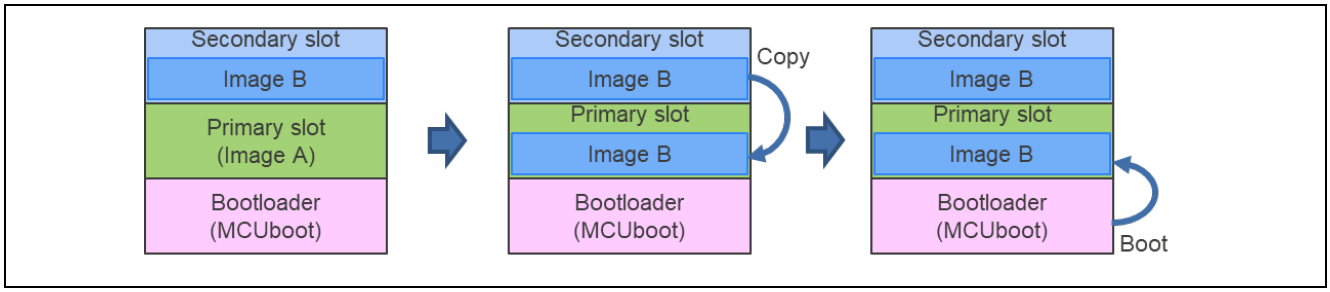

図 1-4 Overwrite Only Fast 方式のアップデート動作

1.5.2 Swap 方式

Swap 方式では、起動可能なイメージは常に Primary slot に格納され、その slot から実行されます。また Secondary slot には更新用のイメージが格納されます。

更新用イメージが Secondary slot に格納された場合、Secondary slot の内容を署名検証し問題無ければ、Secondary slot のイメージを Scratch area に退避し、Primary slot のイメージを Secondary slot にコピーします。その後 Scratch area に退避したイメージを Primary slot にコピーすることでアップデートを行います。

Scratch area を介してイメージをスワップするため、元々Primary slot に有ったイメージがそのまま、Secondary slot に確保されます。

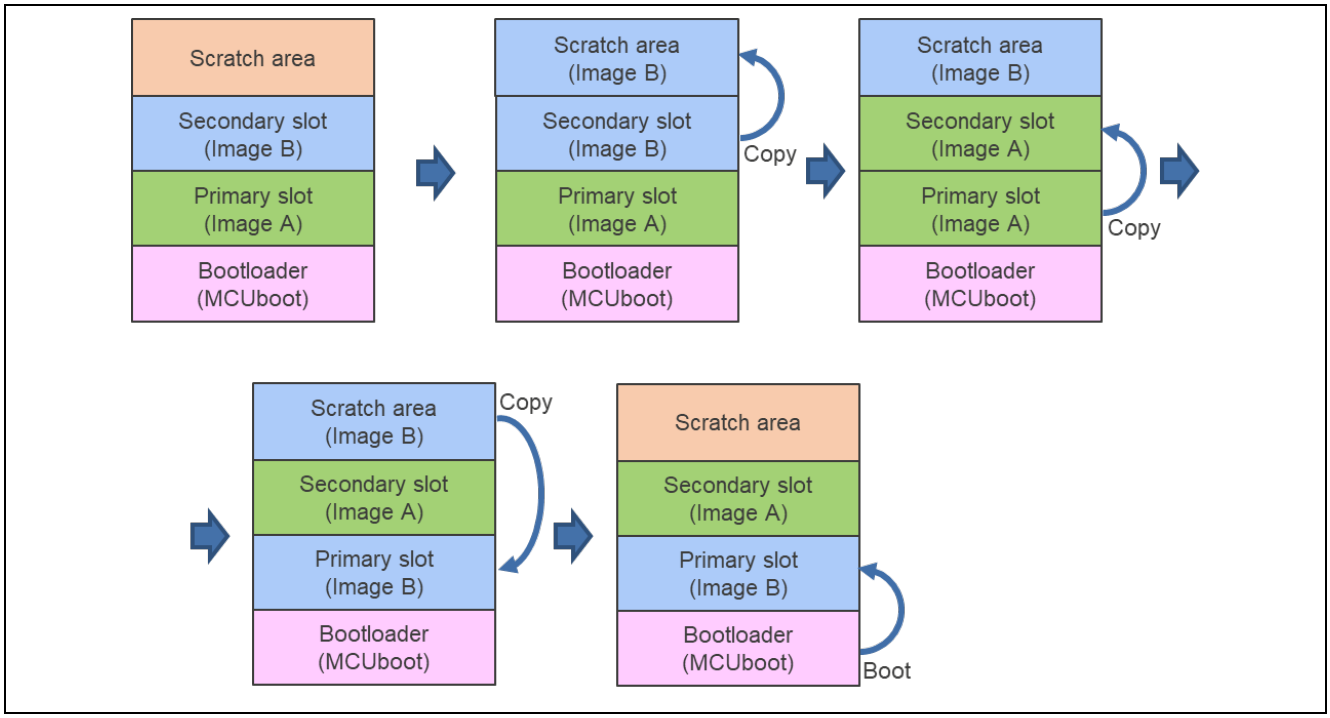

図 1-5 Swap 方式のアップデートの動作

1.5.3 DirectXIP 方式

DirectXIP 方式では、Overwrite Only/Only Fast 方式や Swap 方式と異なり、Primary slot と Secondary slot 間でコピーなどの操作は行わず、どちらの slot からも直接起動することが可能です。

MCUboot FIT では、フラッシュメモリのモードにより、動作が異なります。

Primary slot と Secondary slot どちらに実行可能なイメージを存在させることが可能です。

1.5.3.1 リニアモードの DirectXIP 方式

リニアモードの DirectXIP 方式では、MCUboot が起動する有効 slot を切り替えることによりアップデートを行います。

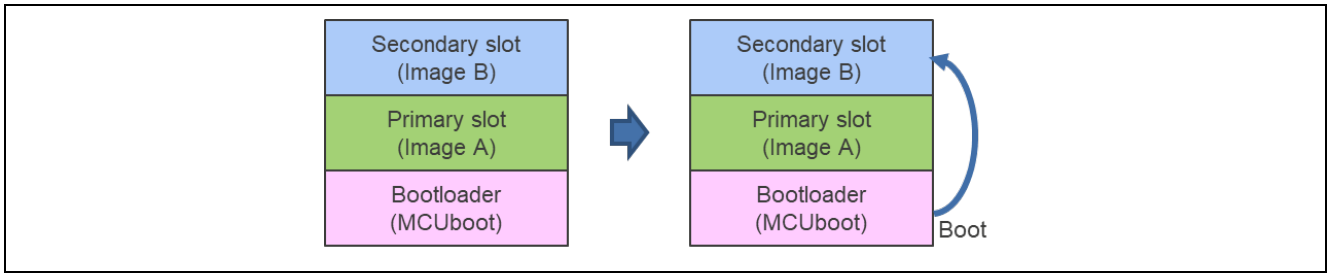

図 1-6 リニアモードの DirectXIP 方式のアップデートの動作

1.5.3.2 デュアルモードの DirectXIP 方式

デュアルモードの DirectXIP 方式では、フラッシュメモリのデュアルバンク機能を使用し 2 分割されたバンクをスワップさせることにより、Secondary slot のイメージを Primary slot のイメージと入れ替えます。

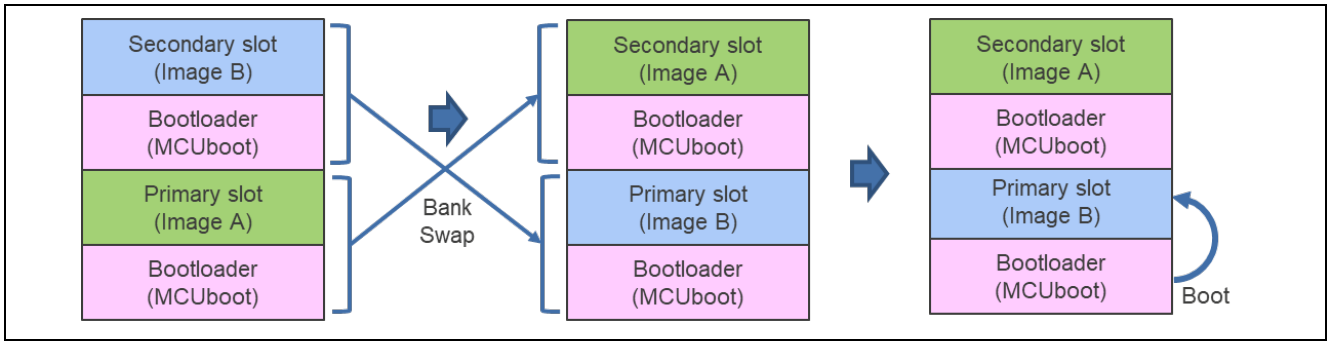

図 1-7 デュアルモード DirectXIP 方式のアップデートの動作

1.6 パッケージ構成

MCUboot FIT モジュールのパッケージには、ソフトウェアやツールを含むいくつかのファイルが含まれています。次の表は、それらの内容を示しています。

表 1-2 MCUboot FIT モジュールパッケージのフォルダ構成

フォルダ名	説明
rm_mcuboot_v1.00	FIT モジュール
├─rm_mcuboot	MCUboot モジュール
│ ├─doc	アプリケーションノート
│ ├─src	
│ │ ├─rm_mcuboot_port	MCUboot FIT
│ │ └─mcu-tools	MCUboot & imgtool
│ └─rm_mcuboot_if.h	インターフェースヘッダファイル
└─r_config	
└─rm_mcuboot_config.h	コンフィグレーション定義ファイル
fitdemos	FIT デモ
├─common	サンプル共通ファイル
├─e2_ccrx	CC-RX 向け
│ ├─rx261-ek	EK-RX261 向け
│ │ └─linear	
│ │ ├─application_primary	初期イメージのアプリケーション
│ │ ├─application_primary_another_slot	更新イメージのアプリケーション（DirectXIP 用）
│ │ ├─boot_loader	ブートローダ
│ │ └─key_injection	キーインジェクションのアプリケーション
│ └─rx###-rsk	RSK-RX###向け
│ ├─dual_bank	デュアルモード向け
│ │ ├─application_primary	初期イメージのアプリケーション
│ │ ├─boot_loader	ブートローダ
│ │ └─key_injection	キーインジェクションのアプリケーション
│ └─linear	リニアモード向け
│ ├─application_primary	初期イメージのアプリケーション
│ ├─application_primary_another_slot	更新イメージのアプリケーション（DirectXIP 用）
│ ├─boot_loader	ブートローダ
│ └─key_injection	キーインジェクションのアプリケーション
├─e2_gcc	GCC 向け
│ ├─rx261-ek	EK-RX261 向け
│ │ └─linear	
│ │ ├─application_primary	初期イメージのアプリケーション
│ │ ├─application_primary_another_slot	更新イメージのアプリケーション（DirectXIP 用）
│ │ ├─boot_loader	ブートローダ
│ │ └─key_injection	キーインジェクションのアプリケーション
│ └─rx###-rsk	RSK-RX###向け
│ ├─dual_bank	デュアルモード向け
│ │ ├─application_primary	初期イメージのアプリケーション
│ │ ├─boot_loader	ブートローダ
│ │ └─key_injection	キーインジェクションのアプリケーション
│ └─linear	リニアモード向け
│ ├─application_primary	初期イメージのアプリケーション
│ ├─application_primary_another_slot	更新イメージのアプリケーション（DirectXIP 用）

フォルダ名	説明
│ ├─boot_loader	ブートローダ
│ └─key_injection	キーインジェクションのアプリケーション
└─iar	IAR 向け
├─rx261-ek	EK-RX261 向け
│ └─linear	
│ ├─application_primary	初期イメージのアプリケーション
│ ├─application_primary_another_slot	更新イメージのアプリケーション（DirectXIP 用）
│ ├─boot_loader	ブートローダ
│ └─key_injection	キーインジェクションのアプリケーション
└─rx###-rsk	RSK-RX###向け
├─dual_bank	デュアルモード向け
│ ├─application_primary	初期イメージのアプリケーション
│ ├─boot_loader	ブートローダ
│ └─key_injection	キーインジェクションのアプリケーション
└─linear	リニアモード向け
├─application_primary	初期イメージのアプリケーション
├─application_primary_another_slot	更新イメージのアプリケーション（DirectXIP 用）
├─boot_loader	ブートローダ
└─key_injection	キーインジェクションのアプリケーション

(###：65N / 671 / 72M / 72N)

1.7 API の概要

本モジュールに含まれる API 関数を表 1-3 に示します。

表 1-3 API 関数一覧

関数	関数説明
boot_go	起動するイメージ情報を取得する。イメージ情報の確認中に Secondary slot で更新イメージが確認された場合はイメージの署名検証が行われ、検証が成功するとアップデート方式に応じて更新イメージが Primary slot に格納される。
RM_MCUBOOT_BootApp	本モジュールに関連するドライバのクローズ処理を行ったあと、引数で指定されたイメージを起動します。
RM_MCUBOOT_GetVersion	モジュールのバージョンを取得する。

2. API 情報

本モジュールは下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用の MCU が以下の機能をサポートしている必要があります。

- 内蔵フラッシュメモリ
- ハードウェア暗号処理（TSIP/RSIP）

2.2 ソフトウェアの要求

本モジュールは以下のドライバに依存しています。

- ボードサポートパッケージ（r_bsp）
- フラッシュモジュール（r_flash_rx）
- TSIP/RSIP モジュール（r_tsip / r_rsip_protected_rx）
- シリアルコミュニケーション インタフェース（SCI：調歩同期式/クロック同期式）（r_sci_rx）
- バイト型キューバッファモジュール（r_byteq）

2.3 サポートされているツールチェーン

本モジュールは「5.1 動作確認環境」に示すツールチェーンで動作確認しています。

2.4 ヘッダファイル

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

2.5 整数型

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

2.6 コンパイル時の設定

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

オプション名および設定値に関する説明を表 **2-1** コンフィグレーション設定に示します。

表 2-1 コンフィグレーション設定

コンフィグレーション設定	
RM_MCUBOOT_CFG_UPGRADE_MODE	アップデート方式の設定 MCUboot のアップデート方式を選択する。 0：Overwrite Only【Default】 1：Overwrite Only Fast 2：Swap 3：DirectXIP
RM_MCUBOOT_CFG _VALIDATE_PRIMARY_SLOT	Primary イメージの署名検証設定 起動前に Primary イメージの署名検証を実施する場合は有効にする。 0：Disable 1：Enable【Default】
RM_MCUBOOT_CFG _DOWNGRADE_PREVENTION	アップデート実行時のダウングレード防止設定 Overwrite Only 方式か Overwrite Only Fast 方式を使用する際に変更する。 0：Disable【Default】 1：Enable
RM_MCUBOOT_CFG _WATCHDOG_FEED_ENABLED	ユーザ定義の watchdog feed を使用する際に有効にする設定 MCUboot の処理の途中に watchdog によるシステムのリセットが起こることを防ぐ。 0：Disable【Default】 1：Enable
RM_MCUBOOT_CFG _WATCHDOG_FEED_FUNCTION	MCUBOOT_WATCHDOG_FEED にユーザ定義の watchdog 関数を登録する。
RM_MCUBOOT_CFG_SIGN	署名検証方式の設定 イメージに対する署名検証の方式を設定する。Primary イメージの署名検証が有効に設定されている場合、ここで設定された方式で検証を行う。 0：None 1：ECDSA P-256【Default】 2：RSA 2048
RM_MCUBOOT_CFG _ APPLICATION_ENCRYPTION_SCHEME	更新イメージを暗号化する場合に Enable に設定する。 Overwrite Only/Only Fast 方式および Swap 方式にて設定可能。 0：Encryption Disabled【Default】 1：Key Wrap
RM_MCUBOOT_CFG _DER_PUB_USER_KEY_ENABLE	ユーザが用意する DER 形式の公開鍵データを使用する際に Enable に設定する。 署名検証の認証方式が設定されていない場合は変更できない。 0：Disable【Default】 1：Enable

RM_MCUBOOT_CFG _VERIFY_KEY_ADDRESS	署名検証に使用する公開鍵のアドレスを設定。 【Default：NULL】
RM_MCUBOOT_CFG _ENCRYPT_KEY_ADDRESS	イメージ復号機能で使用する鍵暗号化鍵のアドレスを設定 【Default：NULL】
RM_MCUBOOT_CFG _MCUBOOT_AREA_SIZE	MCUboot に割り当てられた領域のサイズを設定 （注：使用するデバイスに搭載されているフラッシュのブロックサイズを考慮して設定してください） 【Default：0x10000】
RM_MCUBOOT_CFG _APPLICATION_AREA_SIZE	Application Image に割り当てられた領域のサイズを設定 （注：使用するデバイスに搭載されているフラッシュのブロックサイズを考慮して設定してください） 【Default: RX65N, RX671：0xF0000 RX66N, RX72M, RX72N：0x1F0000 RX261：0x30000】
RM_MCUBOOT_CFG _SCRATCH_AREA_SIZE	Scratch area のサイズを設定 アップデート方式を Swap 方式に指定した際に設定が必要。 （注：使用するデバイスに搭載されているフラッシュのブロックサイズを考慮して設定してください。Scratch area として使用する Flash のセクタサイズの整数倍を設定する必要があります） 【Default：0x10000】
RM_MCUBOOT_CFG_LOG_LEVEL	ログの設定 指定されたレベルのログが MCUboot より出力される。 0：Off【Default】 1：Error 2：Warning 3：Info 4：Debug

表 2-1、表 2-2 に示したコンフィグレーション設定は、特定の設定値により他のパラメータ設定が無効になる組み合わせがあります。以下に無効となるパラメータを示します。

表 2-2 アップデート方式の設定とダウングレード防止設定の対応

RM_MCUBOOT_CFG _UPGRADE_MODE	RM_MCUBOOT_CFG _DOWNGRADE_PREVENTION
1（Overwrite Only）	0 (Disable) / 1 (Enable)
2（Overwtite Only Fast）	0 (Disable) / 1 (Enable)
3（Swap）	設定無効
4（DirectXIP）	設定無効

表 2-3 ユーザ定義の watchdog feed に関連するマクロ定義の対応

RM_MCUBOOT_CFG _WATCHDOG_FEED_ENABLED	RM_MCUBOOT_CFG _WATCHDOG_FEED_FUNCTION
0（Disable）	設定無効
1（Enable）	ユーザ定義の watch dog 関数登録

表 2-4 署名検証の認証方式の設定に対応するパラメータの組み合わせ

RM_MCUBOOT_CFG_SIGN	RM_MCUBOOT_CFG _VERIFY_KEY_ADDRESS	RM_MCUBOOT_CFG _VALIDATE_PRIMARY_SLOT
0（None）	設定無効	設定無効
1（ECDSA P-256）	公開鍵のアドレス設定	0 (Disable)
		1 (Enable)
2（RSA 2048）	公開鍵のアドレス設定	0 (Disable)
		1 (Enable)

表 2-5 暗号化イメージの復号機能の設定と鍵暗号化鍵のアドレス設定の対応

RM_MCUBOOT_CFG _APPLICATION_ENCRYPTION_SCHEME	RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS
0（Encryption Disable）	設定無効
1（Key Wrap）	鍵暗号化鍵のアドレス設定

2.7 サンプルプロジェクトのコードサイズ

本アプリケーションノートのパッケージに含まれるサンプルプロジェクトの ROM、RAM、最大使用スタックサイズを表 2-7 に示します。この表の値は以下の条件で確認しています。

モジュールリビジョン：MCUboot モジュール for RX v1.0.1

コンパイラバージョン：Renesas Electronics C/C++ Compiler for RX Family V3.07.00
GCC for Renesas RX 8.3.0.202411
IAR C/C++ Compiler for Renesas RX 5.10.1

CC-RX

・最適化レベル：サイズ＆実行速度(-Odefault)

・一度も参照のない変数／関数を削除する(-optimize=symbol_delete)

・複数の同一命令をサブルーチン化する(-optimize=same_code)

・コードサイズがより小さくなる命令に置き換える(-optimize=short_format)

・プログラムの配置に基づいて、分岐命令サイズの最適化を行う(-optimize=branch)

・機能縮小版の入出力関数を生成する(はい：最大縮小版)

GCC

・最適化レベル：サイズ(-Os)

・Use newlib-nano (--specs=nano.specs)

・RX65N/RX671/RX72M/RX72N,dual-bank,application_primary のみ
User defined options, -Wl,--no-gc-sections を設定

IAR

・最適化レベル：高(バランス)

表 2-6 サンプルプロジェクトの ROM、RAM、スタックサイズ

compiler	device	bank mode	sample		ROM	RAM	Stack
CC-RX	RX261	linear	application_primary		17673	9903	192
			key_injection		17673	9903	192
			boot_loader	overwrite_only	59876	12236	456
				overwrite_only_fast	60189	12268	456
				swap	62922	13052	457
				directXIP	55123	9696	458
	RX65N	linear	application_primary		16811	10817	192
			key_injection		23066	11303	284
			boot_loader	overwrite_only	43815	19474	320
				overwrite_only_fast	44681	19622	320
				swap	49363	19878	316
				directXIP	33414	18106	248
		dual	application_primary		17511	9421	188
			key_injection		21441	11464	188
			boot_loader	directXIP	34342	16694	248
	RX671	linear	application_primary		17243	10940	192
			key_injection		23457	11442	284
			boot_loader	overwrite_only	44244	19597	320
				overwrite_only_fast	45113	19745	320
				swap	49791	20001	316
				directXIP	33839	18229	248
		dual	application_primary		17888	9437	192
			key_injection		21833	11604	192
			boot_loader	directXIP	33955	16710	248
	RX72M	linear	application_primary		17806	11088	192
			key_injection		24021	11590	284
			boot_loader	overwrite_only	45056	20257	320
				overwrite_only_fast	45932	20437	320
				swap	50600	20917	316
				directXIP	34647	18377	248
		dual	application_primary		18445	9585	192
			key_injection		22391	11752	192
			boot_loader	directXIP	34725	16853	248
	RX72N	linear	application_primary		17667	11044	192
			key_injection		23882	11564	284
			boot_loader	overwrite_only	44920	20213	320
				overwrite_only_fast	45796	20361	320
				swap	50467	20873	316
				directXIP	34514	18333	248
		dual	application_primary		18312	9541	192
			key_injection		22258	11708	192
			boot_loader	directXIP	34598	16814	248

(byte)

compiler	device	bank mode	sample		ROM	RAM	Stack
GCC	RX261	linear	application_primary		14640	12436	68
			key_injection		21400	11796	260
			boot_loader	overwrite_only	54408	14752	700
				overwrite_only_fast	52915	14764	700
				swap	57274	15532	700
				directXIP	49023	12200	684
	RX65N	linear	application_primary		17796	12696	68
			key_injection		22484	12952	928
			boot_loader	overwrite_only	42384	21284	928
				overwrite_only_fast	43035	21424	928
				swap	47406	21680	928
				directXIP	33452	20012	928
		dual	application_primary		21380	10784	68
			key_injection		20988	12952	928
			boot_loader	directXIP	33267	17964	928
	RX671	linear	application_primary		18208	12824	68
			key_injection		22904	12952	928
			boot_loader	overwrite_only	42829	21412	928
				overwrite_only_fast	43480	21552	928
				swap	47851	21808	928
				directXIP	33896	20140	928
		dual	application_primary		22044	10784	68
			key_injection		21392	13080	928
			boot_loader	directXIP	33699	18092	928
	RX72M	linear	application_primary		18992	12952	68
			key_injection		23680	13208	928
			boot_loader	overwrite_only	45148	22052	928
				overwrite_only_fast	45800	22320	928
				swap	50172	22832	928
				directXIP	36212	20268	928
		dual	application_primary		22564	10912	68
			key_injection		22160	13208	928
			boot_loader	directXIP	36008	18220	928
	RX72N	linear	application_primary		18776	12952	68
			key_injection		23464	13080	928
			boot_loader	overwrite_only	42384	21284	928
				overwrite_only_fast	45584	22192	928
				swap	49956	22704	928
				directXIP	36004	20140	928
		dual	application_primary		26188	11040	48
			key_injection		21944	13080	928
			boot_loader	directXIP	35800	18092	928

(byte)

compiler	device	bank mode	sample		ROM	RAM	Stack
IAR	RX261	linear	application_primary		11516	7259	920
			key_injection		18406	6618	1604
			boot_loader	overwrite_only	49243	9578	3044
				overwrite_only_fast	49615	9620	3164
				swap	54867	10404	3148
				directXIP	45378	7044	3012
	RX65N	linear	application_primary		15384	8507	1332
			key_injection		20985	8996	1536
			boot_loader	overwrite_only	41047	17156	2604
				overwrite_only_fast	41716	17307	2604
				swap	46812	17563	2588
				directXIP	31151	15787	2332
		dual	application_primary		16112	6838	1448
			key_injection		19400	9006	1472
			boot_loader	directXIP	31236	14097	2332
	RX671	linear	application_primary		15936	8630	1332
			key_injection		21533	9120	1536
			boot_loader	overwrite_only	41640	17279	2604
				overwrite_only_fast	42310	17432	2604
				swap	47422	17688	2588
				directXIP	31704	15910	2332
		dual	application_primary		16669	6857	1448
			key_injection		19973	9142	1472
			boot_loader	directXIP	31800	14113	2332
	RX72M	linear	application_primary		16550	8782	1400
			key_injection		22205	9271	1605
			boot_loader	overwrite_only	43107	17941	2692
				overwrite_only_fast	43785	18094	2692
				swap	48897	18606	2676
				directXIP	33171	16060	2420
		dual	application_primary		17285	7005	1516
			key_injection		20589	9293	1540
			boot_loader	directXIP	33251	14261	2420
	RX72N	linear	application_primary		16375	8737	1400
			key_injection		21996	9228	1604
			boot_loader	overwrite_only	42924	17900	2692
				overwrite_only_fast	43605	18050	2692
				swap	48717	18562	2676
				directXIP	32991	16145	2420
		dual	application_primary		17109	6961	1516
			key_injection		20416	9250	1540
			boot_loader	directXIP	33087	14217	2420

(byte)

2.8 引数

API 関数の引数を示します。この構造体は API 関数のプロトタイプ宣言とともに bootutil.h に記載されています。

```
struct boot_rsp {
    /** A pointer to the header of the image to be executed. */
    const struct image_header *br_hdr;

    /**
     * The flash offset of the image to execute.  Indicates the position of
     * the image header within its flash device.
     */
    uint8_t br_flash_dev_id;
    uint32_t br_image_off;
};
```

表 2-7 引数一覧

構造体名	メンバ	説明
boot_rsp	image_header br_hdr	実行するイメージヘッダ用のポインタ
	uint8_t br_flash_dev_id	フラッシュデバイスの ID
	uint32_t br_image_off	実行するイメージのオフセット

2.9 戻り値

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

```
#define FIH_POSITIVE_VALUE 0
#define FIH_NEGATIVE_VALUE -1

extern fih_ret FIH_SUCCESS;
extern fih_ret FIH_FAILURE;
```

表 2-8 戻り値一覧

定数定義	数値	説明
FIH_SUCCESS	0	API 関数の戻り値。API 関数の処理が成功した際に使用される。
FIH_FAILURE	-1	API 関数の戻り値。API 関数の処理が失敗した際に使用される。

2.10 FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。

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

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

(2) IAR Embedded Workbench for Renesas RX の環境でスマート・コンフィグレータを使用して FIT モジュールを追加する場合

IAR Embedded Workbench for Renesas RX の環境で FIT モジュールを追加する場合は、RX スマート・コンフィグレータを使用して、ユーザプロジェクトに FIT モジュールを追加します。
詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: IAREW 編 (R20AN0535)」を参照してください。

2.11 for 文、while 文、do while 文について

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

以下に記述例を示します。

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

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

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

2.12 API の実装例について

MCUboot FIT の実装例を示します。

詳細は本アプリケーションノートのパッケージに含まれるデモプロジェクトのソースコードをご確認ください。

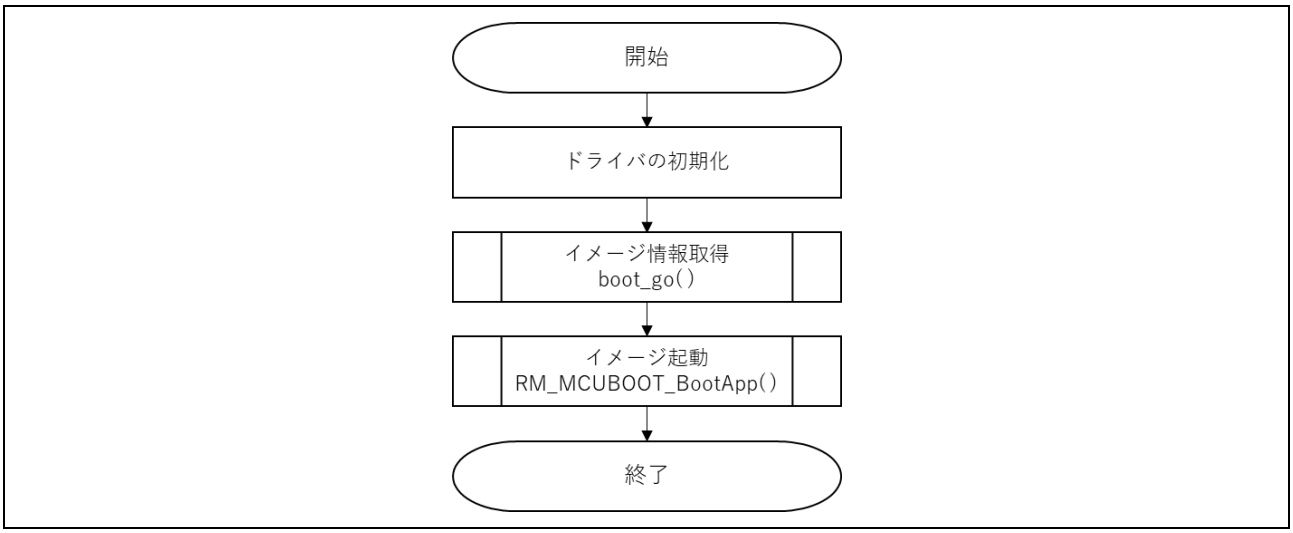

図 2-1 MCUboot FIT の実装例

3. API 関数

3.1 boot_go 関数

表 3-1 boot_go 関数仕様

Format	fih_ret boot_go(struct boot_rsp *rsp)
Description	以下のように起動するイメージの情報を取得する。 ① 各 Slot で有効なイメージヘッダを含むイメージの確認。Secondary slot に更新イメージがない場合、②はスキップする。 ② 更新イメージの署名検証と、アップデート方式に従ったイメージの更新を実施。検証の方式とアップデートの方式はコンフィグレーション設定で変更する。 起動するイメージ情報を返す。
Parameters	struct boot_rsp *rsp
Return Values	FIH_SUCCESS イメージ情報の取得成功
	FIH_FAILURE イメージ情報の取得失敗
Special Notes	詳細は以下を参照。 https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md

3.2 RM_MCUBOOT_BootApp 関数

表 3-2 RM_MCUBOOT_BootApp 関数仕様

Format	void RM_MCUBOOT_BootApp (struct boot_rsp * rsp)
Description	本モジュールに関連するドライバのクローズ処理を行ったあと、引数で指定されたイメージを起動します。
Parameters	struct boot_rsp * rsp
Return Values	なし
Special Notes	デュアルモードで DirectXIP 設定時、起動アドレスが Secondary slot である場合、バンクの交換とソフトウェアリセットの処理が RM_MCUBOOT_BootApp で行われる。

3.3 RM_MCUBOOT_GetVersion 関数

表 3-3 RM_MCUBOOT_GetVersion 関数仕様

Format	uint32_t RM_MCUBOOT_GetVersion(void)
Description	MCUboot FIT モジュールのバージョンを取得する。
Parameters	なし
Return Values	MCUboot FIT モジュールのバージョン番号
Special Notes	MCUboot FIT モジュールのメジャーバージョン番号とマイナーバージョン番号は、インターフェースヘッダファイルで管理する。

4. デモプロジェクト

本デモプロジェクトは、MCUboot とシリアル通信インタフェース（SCI）を用いたファームウェアアップデートを実施するためのサンプルプログラムです。

4.1 デモプロジェクトの構成

デモプロジェクトは、MCUboot FIT モジュールとその他の依存するモジュールを含むブートローダと、ファームウェアアップデートを実施するための機能を含む初期イメージで構成されます。本パッケージでは、1.6 に示すデバイスとコンパイラに対応したデモプロジェクトを提供しています。

MCUboot を用いたファームウェアアップデートのデモは以下のプロジェクトで構成されています。

- ブートローダ（MCUboot）：リセット後に最初に実行され、MCUboot FIT のコンフィグレーションで設定された検証方式に基づきイメージの検証を行います。また、更新イメージが存在する場合は、アップデート方式に基づきアップデートを行います。
- 初期イメージ：ブートローダ（MCUboot）により起動され、アップデートを行うための更新イメージを通信インタフェースからダウンロードし、Secondary slot に書き込みます。
- 更新イメージ：初期イメージとバージョンのみが異なるイメージです。イメージの暗号化に関しては更新イメージのみ暗号化することが可能です。
- 鍵インジェクションプログラム：署名検証用公開鍵とイメージ復号用鍵をインジェクションする際に使用します。

4.1.1 初期イメージの詳細

ブートローダにより起動され、Primary slot に配置されるイメージです。初期イメージは通信インタフェース経由で更新イメージを受信し、Secondary slot に書き込みます。

図 4-1～図 4-5 に初期イメージのフローを記載します。

詳細は本アプリケーションノートのパッケージに含まれるソースコードをご確認ください。

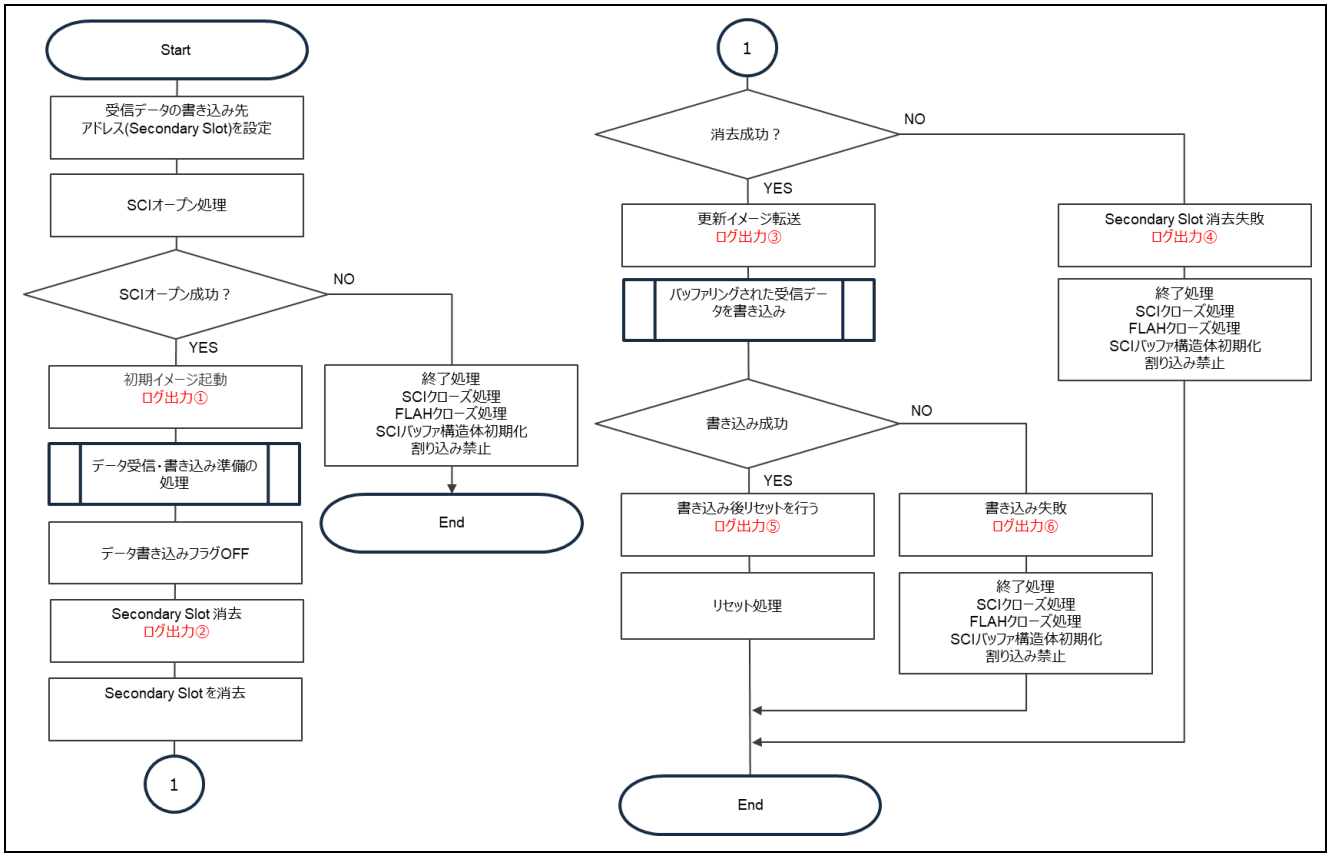

図 4-1 初期イメージの処理フロー（1/5）

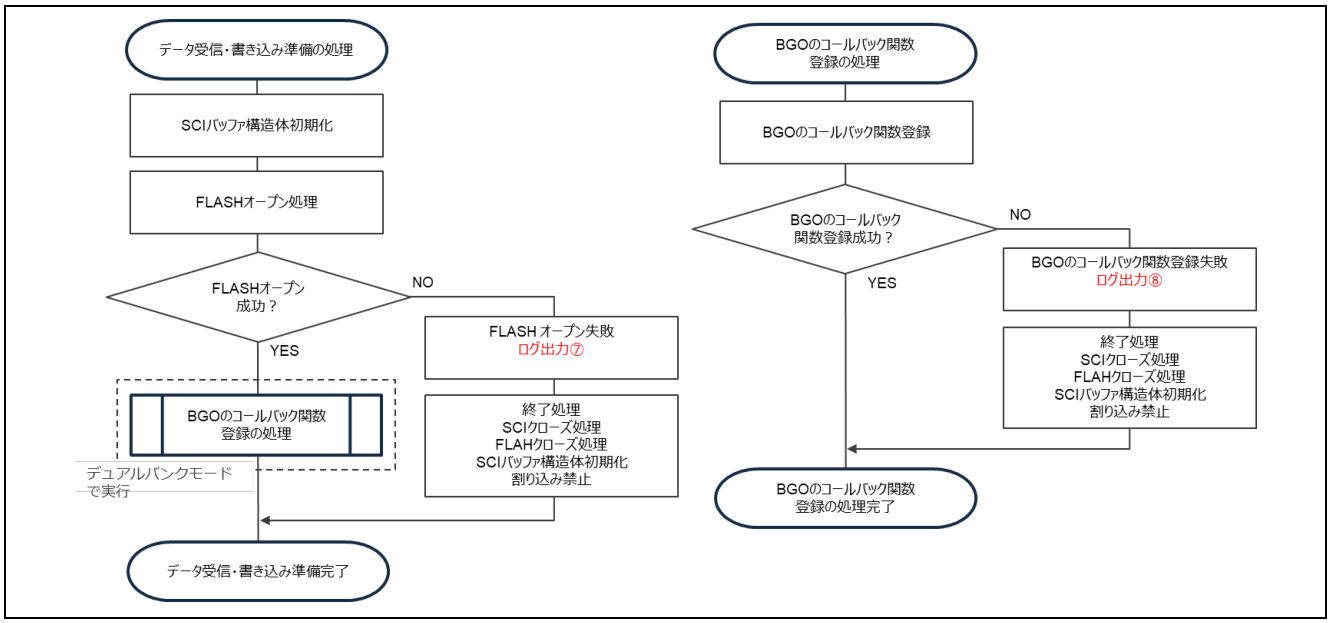

図 4-2 初期イメージの処理フロー（2/5）

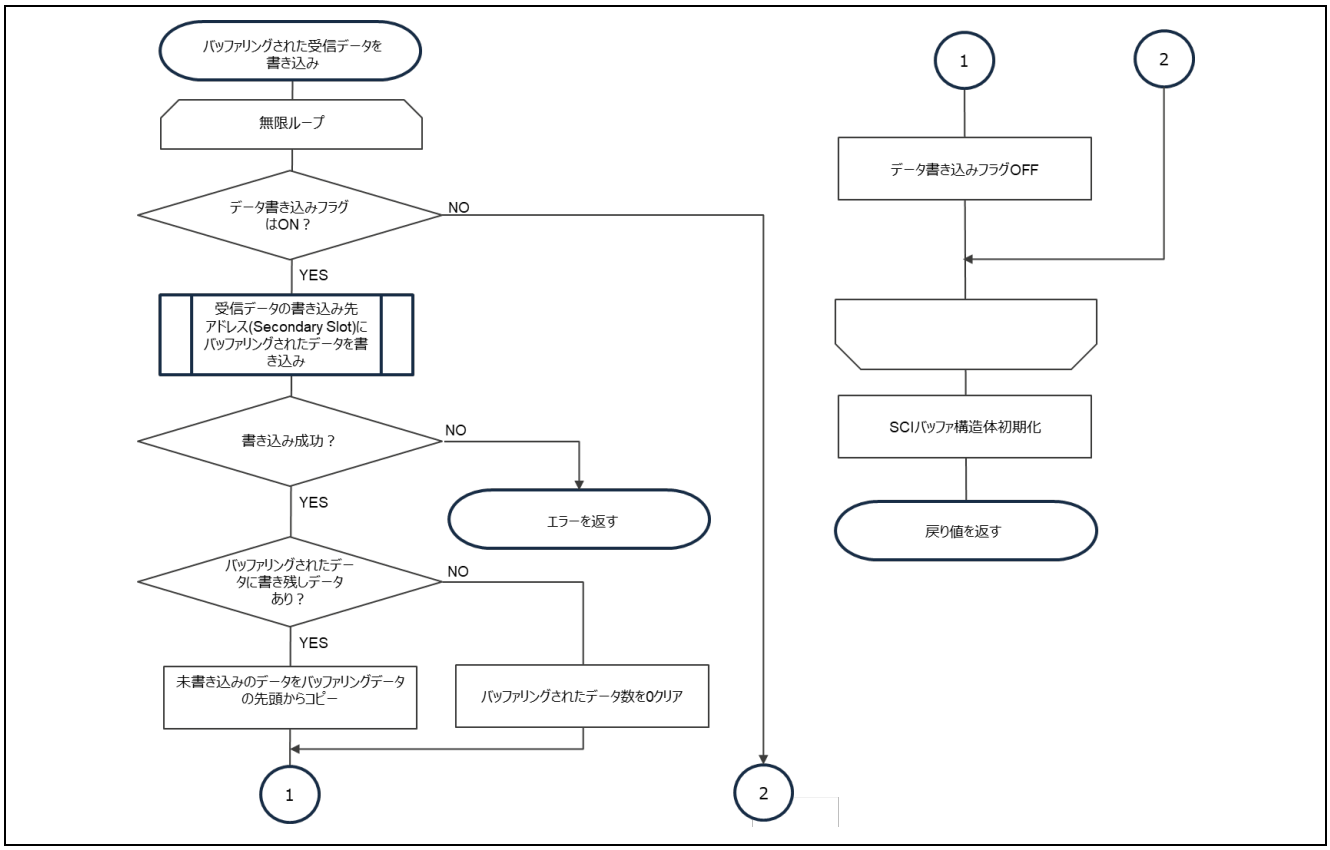

図 4-3 初期イメージの処理フロー（3/5）

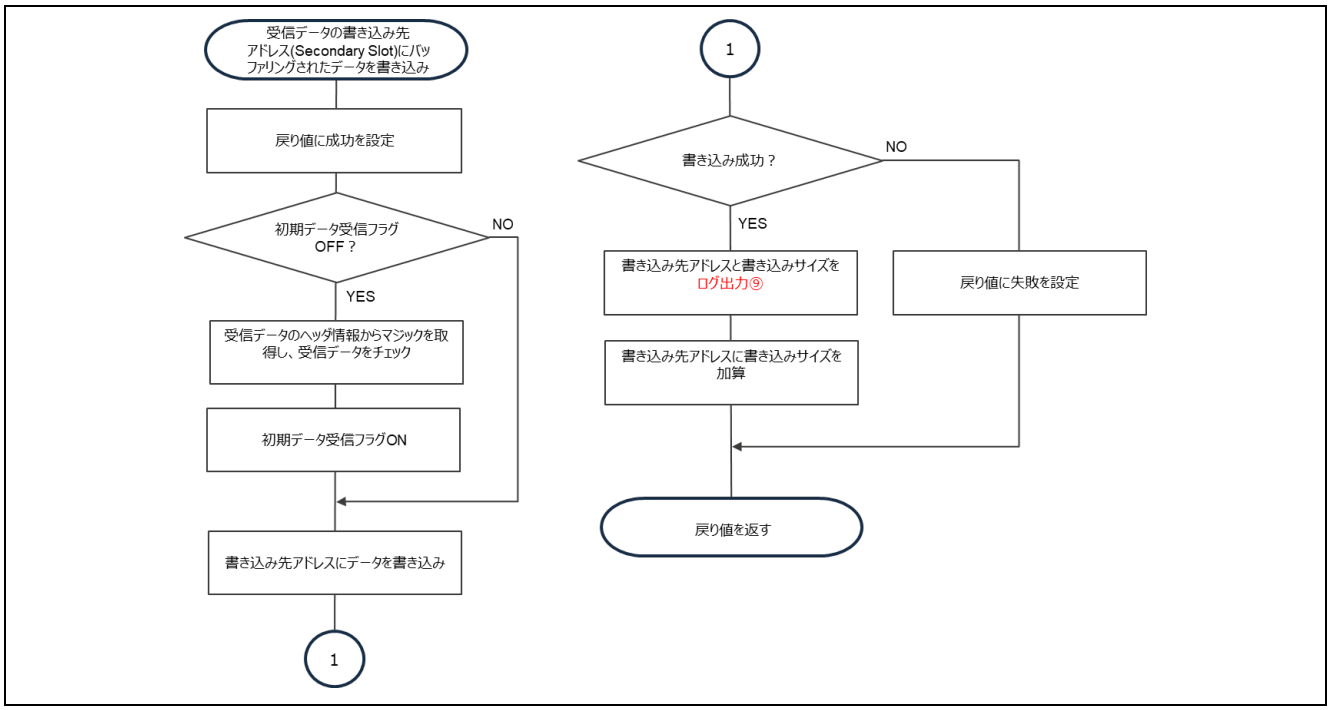

図 4-4 初期イメージの処理フロー（4/5）

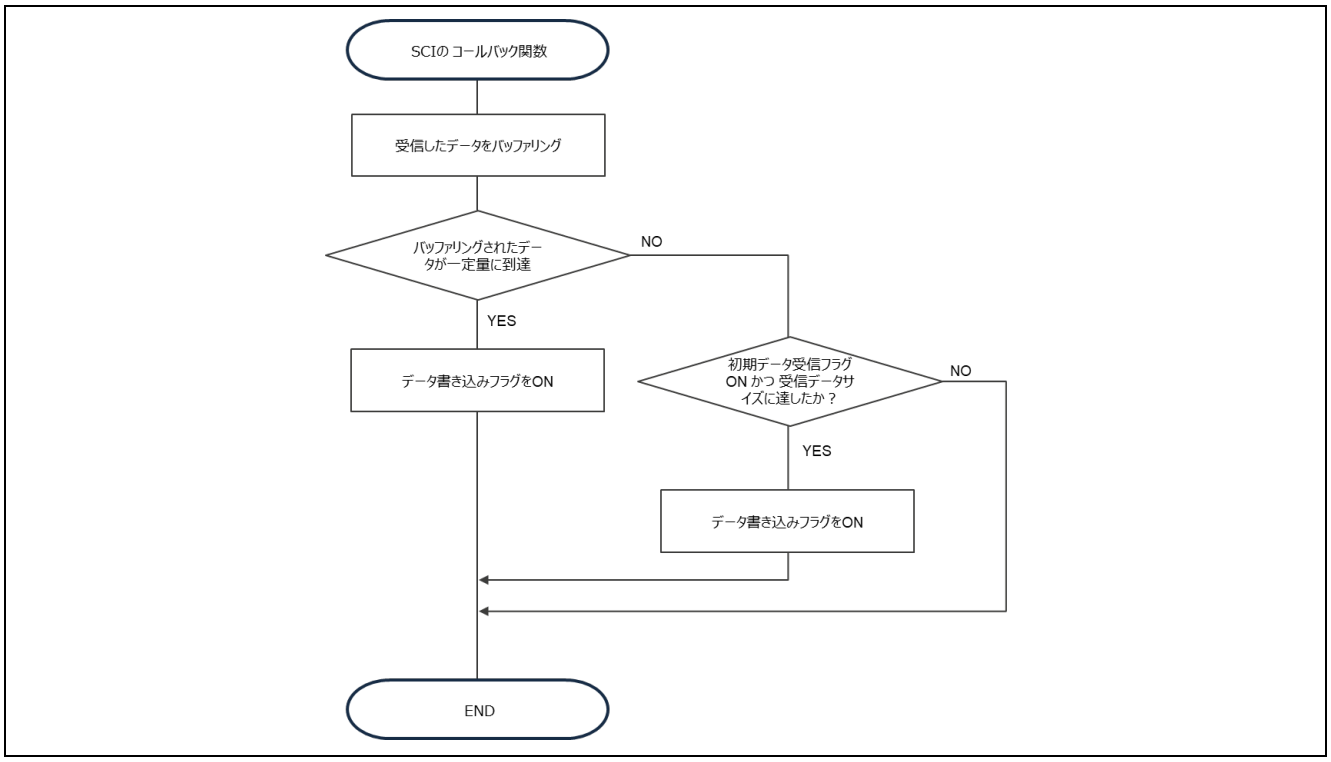

図 4-5 初期イメージの処理フロー（5/5）

表 4-1 初期イメージのログ出力内容

ログ出力箇所	ログ出力内容
初期イメージ起動 （ログ出力①）	---------- Primary Slot Application Image Start (ver 1.0.0) ----------
Secondary slot 消去 （ログ出力②）	Erase the code flash of the Secondary Slot.
更新イメージ転送 （ログ出力③）	send user program (MCUboot image) via UART.
Secondary slot 消去失敗 （ログ出力④）	Erase the code flash of the Secondary Slot failed.
書き込み後リセットを行う （ログ出力⑤）	software reset...
書き込み失敗 （ログ出力⑥）	Failed to write code flash.
FLASH オープン失敗 （ログ出力⑦）	Flash driver open failure.
BGO のコールバック関数登録失敗（ログ出力⑧）	BGO callback function set failure.
書き込み先アドレスと書き込みサイズ（ログ出力⑨）	例） W 0xFFF00000, 256 ... OK

4.2 動作環境準備

MCUboot のデモプロジェクトを実行するには、Windows PC にツールをインストールする必要があります。

4.2.1 Imgtool の入手

Imgtool は初期イメージおよび更新イメージの生成を行います。イメージへの署名や、ヘッダやトレーラー情報などを付加します。また、署名検証用の鍵ペアを生成することも可能です。

Imgtool は MCUboot FIT 向けに一部変更を行っておりますので、パッケージ内に格納されている imgtool.py を使用してください。Imgtool の v2.1.0 をベースに変更を行っています。

詳細は、以下の MCUboot の imgtool に関する URL を参照して下さい。

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

4.2.2 ターミナルソフトのインストール

Windows PC からターゲットボードへのシリアル通信により、アップデートを行うためのファームウェアイメージを転送するために使用します。

ここでは TeraTerm v5.2 で動作確認を行っています。

シリアルポートの通信設定を表 4-2 のように設定してください。

表 4-2 通信仕様

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

4.2.3 Python 実行環境のインストール

MCUboot に含まれる imgtool を使用するためには Python 実行環境が必要となります。

ここでは Python 3.11.4 で動作確認を行っています。

Python の暗号化ライブラリ（pycryptodome）を使用しますので、Python をインストールし最初に以下のコマンドで pip をして下さい。

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

また、以下のコマンドで依存関係をインストールして下さい。

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

注）`scripts/requirements.txt` はパッケージ内の `mcu-tools` に同梱しています。

4.2.4 OpenSSL の実行環境のインストール

イメージの暗号化に必要な鍵を生成するために OpenSSL を使用します。以下の URL から OpenSSL インストーラをダウンロードして、インストールします。Light 版で問題ありません。

ここでは OpenSSL 3.4.1 で動作確認を行っています。

https://slproweb.com/products/Win32OpenSSL.html

4.2.5 フラッシュライタのインストール

MCUboot モジュール、初期イメージ、鍵ラッピングデータ等をフラッシュメモリに書き込むために使用するツールです。詳細は以下 URL を参照してインストールして下さい。

デモプロジェクトでは Renesas Flash Programmer v3.14.00（RFP）を使用しています。

Renesas Flash Programmer (Programming GUI) | Renesas

4.2.6 Security Key Management Tool のインストール

鍵ラッピングデータを生成するために使用するツールです。詳細は以下 URL を参照してインストールして下さい。

Security Key Management Tool | Renesas

4.2.7 USB シリアル変換ボード

RSK-RX65N/RSK-RX72N にはオンボードに USB シリアル変換回路が実装されていますが、リニアモードで使用する場合は使用できないため、ここに記載した外付けの USB シリアル変換ボードを使用してください。

EK-RX261 ではオンボードの USB シリアル変換回路を使用します。

ターゲットボードとの接続方法については、5.2 デモプロジェクトの動作環境を参照してください。

外付けの USB シリアル変換ボードは Pmod USBUART（DIGILENT 製）を使用します。

https://reference.digilentinc.com/reference/pmod/pmodusbuart/start

4.3 デモプロジェクトの実行手順

本章では、デモプロジェクトを実行するための手順について記載します。

アドレス等の値は RSK-RX65N を使用する場合の例となります。製品毎の設定値は 5.2 デモプロジェクトの動作環境を参照して下さい。

鍵に関しては、デモプロジェクトにサンプルの鍵を同梱しておりますので、そちらをご使用頂く事も可能です。ただし、製品版では必ず鍵を新たに生成頂くようお願いします。

4.3.1 鍵のインジェクション

デモプロジェクトでは、TSIP モジュールまたは RSIP モジュールを使用して、署名検証やイメージの復号を行います。TSIP または RSIP を使用するため、署名検証用公開鍵、イメージ復号用鍵とイメージ暗号鍵をラップするための鍵を Hardware Unique Key（HUK）でラップし、デバイスにインジェクションする必要があります。

鍵の生成およびインジェクションの詳細について以下に示す手順を参照して下さい。

注）以降 TSIP と記載している場合、TSIP もしくは RSIP を指します。

4.3.1.1 Security Key Management Tool による鍵データの生成

Security Key Management Tool（SKMT）を使い、User Factory Programming Key（UFPK）ファイルを生成し、この UFPK ファイルにて署名検証用公開鍵とイメージ暗号鍵をラップするための鍵（AES-KeyWrap 鍵）をラップします。

また、Renesas Key Wrap Service により、UFPK ファイルを Hardware Root Key（HRK）でラップした W-UFPK を取得します。

SKMT の詳細については、Security Key Management Tool のページ（Security Key Management Tool | Renesas）や、Security Key Management Tool ユーザーズマニュアル（R20UT5349）を参照して下さい。

Renesas Key Wrap Service（https://dlm.renesas.com/keywrap）の詳細については、FAQ や操作マニュアルを参照して下さい。

以下の手順に従って生成して下さい。

Step1) SKMT で UFPK ファイルを生成します。

Step2) Step1 で作成したファイルを Renesas Key Wrap Service に送信します。

Step3) Renesas Key Wrap Service を使用し UFPK ファイルを HRK でラップされた W-UFPK を入手します。

Step4) imgtool で署名検証用の鍵ペアを生成します。（4.3.3.2、4.3.3.3 のイメージ生成時および、4.3.2 の公開鍵の埋め込み時にも使用します）

mcu-tools¥MCUboot¥scripts の imgtool.py を Python 環境で実行します。

【ECDSA P-256】

```
python imgtool.py keygen -k ecc_sign_key_pair.pem -t ecdsa-p256
```

【RSA 2048】

```
python imgtool.py keygen -k rsa_sign_key_pair.pem -t rsa-2048
```

Step5) OpenSSL で乱数を使いイメージ暗号鍵をラップするための AES-KeyWrap 鍵を生成します。（4.3.3.3 の更新イメージの生成に使用します）

```
openssl rand 32 -out AES-KeyWrap.bin
```

Step6) OpenSSL で乱数を使いイメージ暗号鍵を生成します。（4.3.3.3 の更新イメージの生成に使用します）

```
openssl rand 32 -out AES-CTR.bin
```

Step7) SKMT により、Step4 で生成した署名検証用公開鍵と Step5 で生成した AES-KeyWrap 鍵を Step1 で生成した UFPK ファイルでラップします。

Step8) SKMT により UFPK で暗号化された鍵データのファイル（バイナリ形式）を生成します。

生成した鍵データファイルは、4.3.1.2 でコードフラッシュに書き込みます。

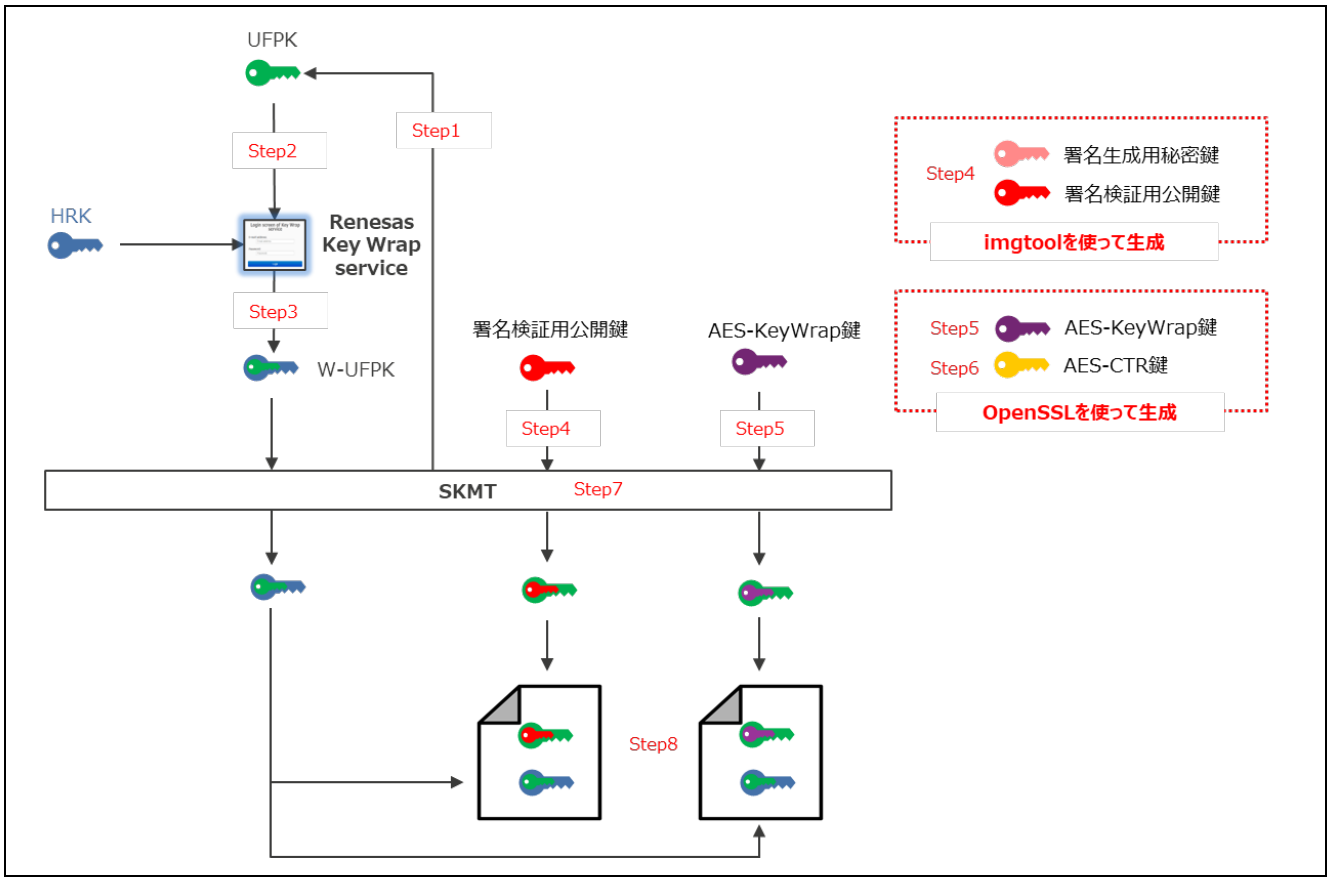

図 4-6 SKMT による鍵データの生成

4.3.1.2 鍵インジェクションの準備

4.3.1.1 で生成したラップされた鍵データ（W-UFPK、署名検証用公開鍵、AES-KeyWrap 鍵）と鍵インジェクションプログラム（サンプルとして提供）を Renesas Flash Programmer（RFP）でフラッシュメモリに書き込みます。

Step1) 4.3.1.1 で生成した鍵データのファイル（バイナリ形式）を RFP でデータフラッシュ（0x00100000）に書き込みます。
RFP の操作の詳細については、Renesas Flash Programmer フラッシュ書き込みソフトウェアユーザーズマニュアル（R20UT5517）を参照して下さい。

Step2) 鍵インジェクションプログラムを RFP でコードフラッシュ（0xFFFF2000）に書き込みます。

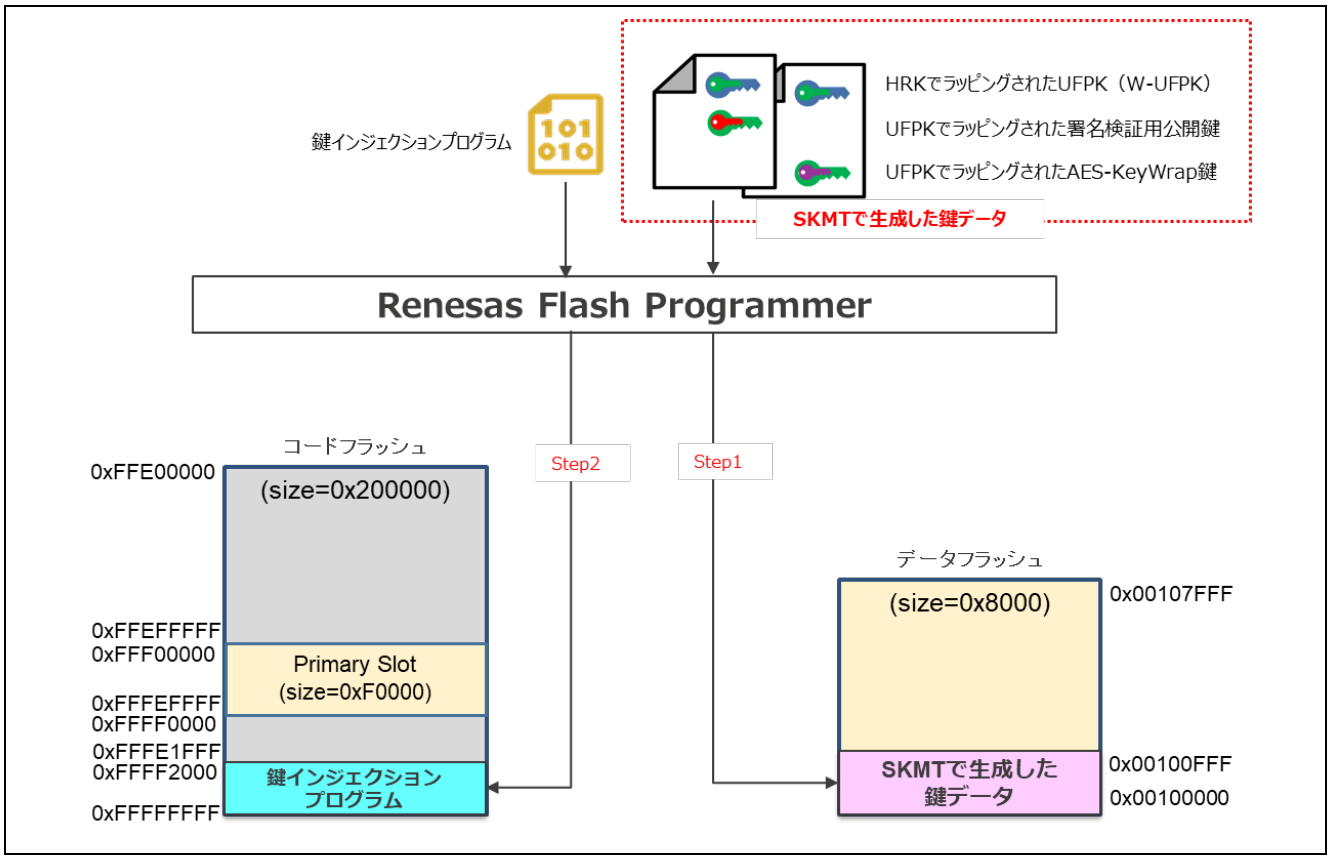

図 4-7 鍵のインジェクション準備

4.3.1.3 鍵インジェクションプログラムの実行

鍵インジェクションプログラムを実行し鍵をインジェクションします。

フラッシュメモリがリニアモードの場合：

Step1) ボードのリセットを行い、鍵インジェクションプログラムを実行します。

Step2) 鍵インジェクションプログラムにより、データフラッシュに格納された鍵データを使用し、TSIP の HUK で署名検証用公開鍵と AES-KeyWrap 鍵をラップしたデータを生成し、コードフラッシュ（0xFFFF0000）に Wrapped Key を書き込みます。

Step3) データフラッシュに格納された鍵データを消去します。

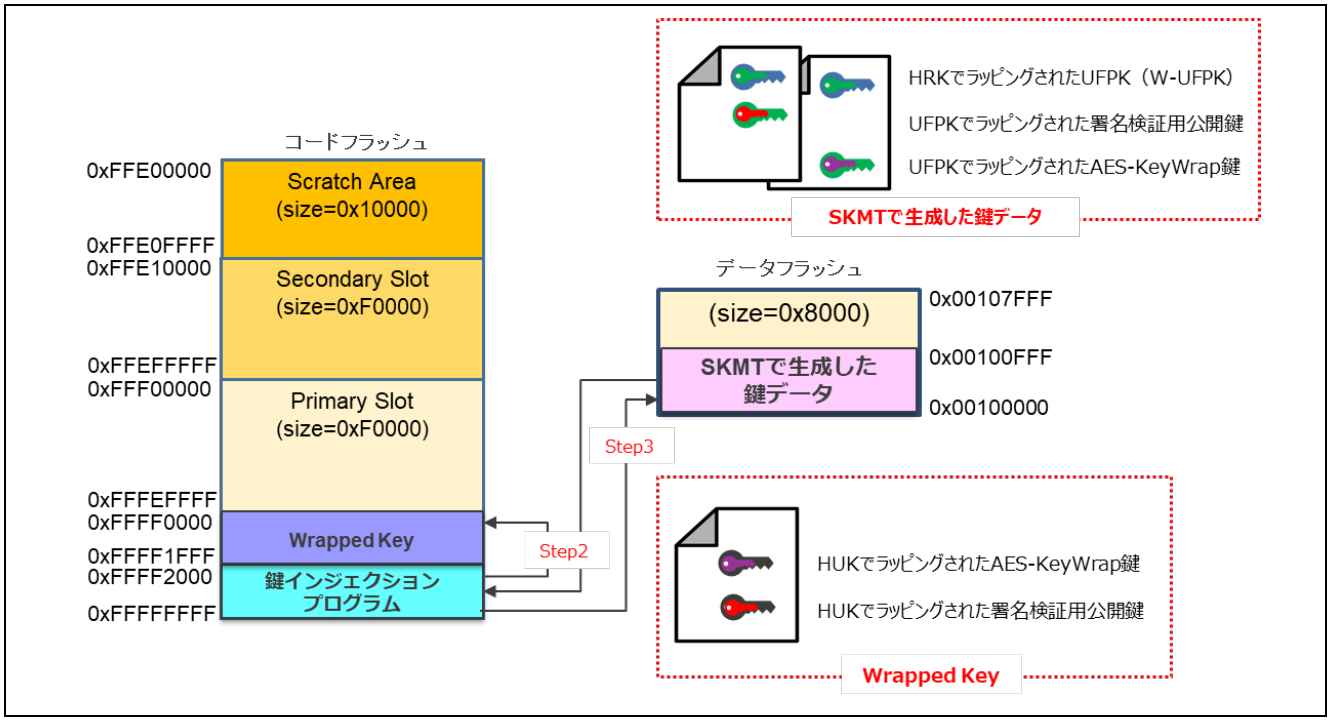

図 4-8 鍵インジェクションプログラムの実行（リニアモード）

フラッシュメモリがデュアルモードの場合：

Step1) ボードのリセットを行い、鍵インジェクションプログラムを実行します。

Step2) 鍵インジェクションプログラムにより、データフラッシュに格納された鍵データを使用し、TSIP の HUK で署名検証用公開鍵と AES-KeyWrap 鍵をラップしたデータを生成し、コードフラッシュ（0xFFFF0000 と 0xFFEF0000）に Wrapped Key を書き込みます。

Step3) データフラッシュに格納された鍵データを消去します。

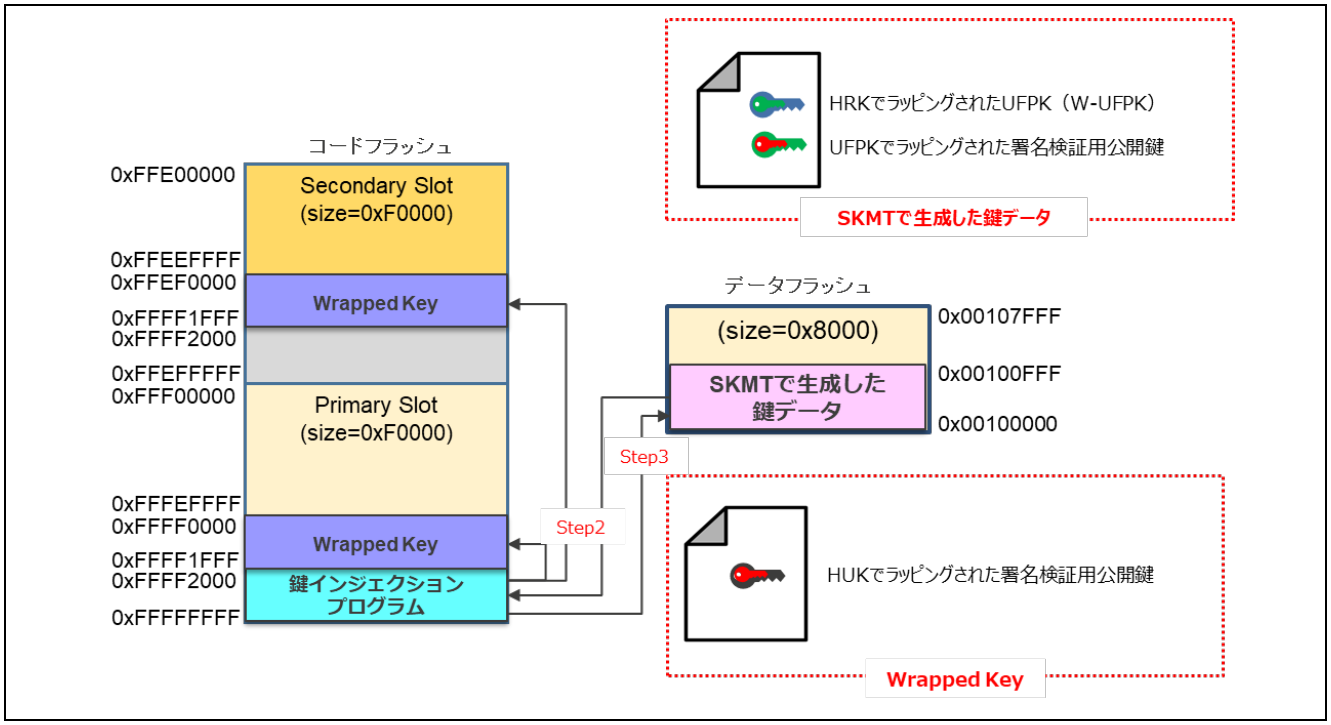

図 4-9 鍵インジェクションプログラムの実行（デュアルモード）

4.3.2 署名検証用公開鍵の埋め込み

デモプロジェクト内の keys.c に、署名検証用公開鍵を埋め込みます。

Step1) imgtool でブートローダに埋め込む署名検証用公開鍵を抽出します。

imgtool で 4.3.1.1 の Step4 で生成した鍵ペアの pem ファイルから公開鍵データが抽出され、コンソールに表示される。

【ECDSA P-256】

```
python imgtool.py getpub -k ecc_sign_key_pair.pem
```

【RSA 2048】

```
python imgtool.py getpub -k rsa_sign_key_pair.pem
```

【出力例】

```
/* Autogenerated by imgtool.py, do not edit. */
const unsigned char ecdsa_pub_key[] = {
    0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2a, 0x86,
    0x48, 0xce, 0x3d, 0x02, 0x01, 0x06, 0x08, 0x2a,
    0x86, 0x48, 0xce, 0x3d, 0x03, 0x01, 0x07, 0x03,
    0x42, 0x00, 0x04, 0x53, 0x5a, 0x25, 0x70, 0xe6,
    0xa4, 0xd1, 0x0b, 0xaa, 0x25, 0x52, 0x14, 0xf7,
    0xa2, 0x69, 0x3b, 0xc5, 0x02, 0xe0, 0xe7, 0x96,
    0x0c, 0xa8, 0x59, 0x5f, 0x28, 0x04, 0x95, 0x52,
    0x05, 0x3d, 0xea, 0x46, 0x75, 0xd6, 0xa9, 0xd5,
    0x0b, 0x99, 0x5d, 0x1a, 0x2f, 0x10, 0x31, 0x01,
    0xc9, 0x1e, 0x67, 0x42, 0x6d, 0xea, 0xec, 0x77,
    0x3d, 0x23, 0xd4, 0x23, 0x75, 0x28, 0x67, 0x29,
    0xd1, 0x4f, 0x4a,
};
const unsigned int ecdsa pub key len = 91;
```

Step2) コンソールに表示されたデータを、デモプロジェクト内の keys.c に埋め込みます。

```
#include <bootutil/sign_key.h>

const unsigned char root_pub_der[] = {
    /* 生成した DER 形式の署名検証用の公開鍵データを埋め込んで下さい */
};
const unsigned int root_pub_der_len = 0; /* len を埋め込んで下さい */

const struct bootutil_key bootutil_keys[] = {
    {
        .key = root_pub_der,
        .len = &root_pub_der_len,
    },
};
const int bootutil_key_cnt = 1;
```

4.3.3 デモプロジェクトのイメージの準備

デモプロジェクト（ブートローダ、初期イメージ、更新イメージ）を準備します。アップデート方式により、イメージを格納するコードフラッシュのアドレスや一部手順が異なりますので、ご注意下さい。

4.3.3.1 ブートローダのイメージを生成

デモプロジェクトのブートローダをビルドし、バイナリ形式のイメージを生成します。

4.3.3.2 初期イメージを生成

デモプロジェクトの初期イメージをビルドしバイナリ形式のイメージを生成し、imgtool で MCUboot が管理するための情報を付加します。

Step1) デモプロジェクトの初期イメージ（UART 通信により更新イメージを受信し、Secondary slot に書き込むアプリケーションプログラム）をビルドし、バイナリ形式のイメージを生成します。

Step2) imgtool により、step1 で生成したイメージに、MCUboot が管理する Trailer 情報を付加した初期イメージを生成します。

例）初期イメージの生成方法

```
imgtool.py sign --version 1.0.0 --header-size 0x200 --align 128
--max-align 128 --slot-size 0xF0000 --max-sectors 16 --confirm
--pad-header --key ecc_sign_key_pair.pem
input_image.bin input_image.bin.ecc_sign
```

--version #.#.# : バージョンを指定してください。

--slot-size 0x#### : slot のサイズを指定してください。

--key ########.pem : 4.3.1.1 の Step4 で生成した署名検証用の鍵ペアを設定してください。

########.bin : Step1 で生成したイメージのバイナリファイルを設定してください。

########.bin.ecc_sign : 初期イメージが出力されます。

注）初期イメージは暗号化することはできません。

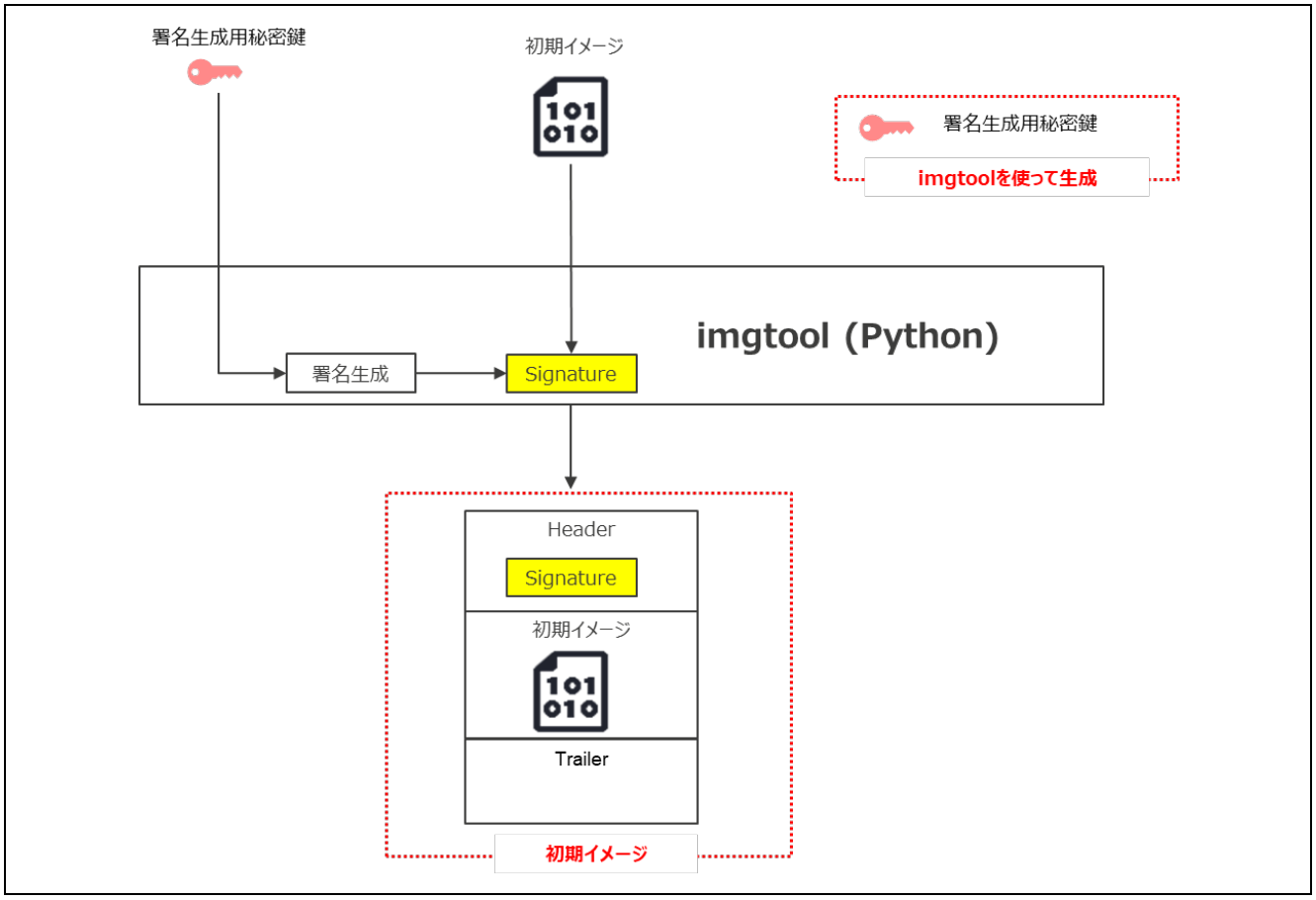

図 4-10 初期イメージを生成

4.3.3.3 更新イメージを生成

デモプロジェクトでは、更新イメージに関しては初期イメージを流用します。

Imgtool で情報を付加する際に、バージョンの指定（--version）を初期イメージより上げて設定してください。

またイメージを暗号化したい場合は、イメージ暗号化鍵の指定（-kw--enckey）および、イメージ暗号化鍵をラップするための鍵を指定（-kw--kek）して、更新イメージを生成してください。

イメージの暗号化に関しては、DirectXIP 方式では対応できません。

Step1) デモプロジェクトの初期イメージをビルドし、バイナリ形式のイメージを生成します。

Step2) imgtool により、Step1 で生成したイメージに、MCUboot が管理する Trailer 情報を付加した更新イメージを生成します。

例）暗号化した更新イメージの生成方法

```
imgtool.py sign --version 1.1.0 --header-size 0x200 --align 128
--max-align 128 --slot-size 0xF0000 --max-sectors 16 -confirm
--pad-header --key ecc_sign_key_pair.pem
-kw--enckey AES-CTR.bin -kw--kek AES-KeyWrap.bin
input_image.bin input_image.bin.ecc_sign.enc
```

--version #.#.#：バージョンを指定してください。（初期イメージよりバージョンを上げて設定することにより、更新イメージとなります）

--slot-size 0x####：slot のサイズを指定してください。

--key ########.pem：4.3.1.1 の Step4 で生成した署名検証用の鍵ペアを設定してください。

-kw--enckey ########.bin：4.3.1.1 の Step6 で生成したイメージ暗号化鍵を設定してください。

-kw--kek ########.bin：4.3.1.1 の Step5 で生成したイメージ暗号化鍵をラップするための鍵を設定してください。

########.bin：Step1 で生成したイメージのバイナリファイルを設定してください。

########.bin.ecc_sign.enc：暗号化された更新イメージが出力されます。

例）暗号化しない更新イメージの生成方法

```
imgtool.py sign --version 1.1.0 --header-size 0x200 --align 128
--max-align 128 --slot-size 0xF0000 --max-sectors 16 -confirm
--pad-header --key ecc_sign_key_pair.pem
input_image.bin input_image.bin.ecc_sign
```

暗号化しない更新イメージに関しては、初期イメージと同様の生成方法となり、--version の指定（バージョンを上げたもの）のみ異なります。

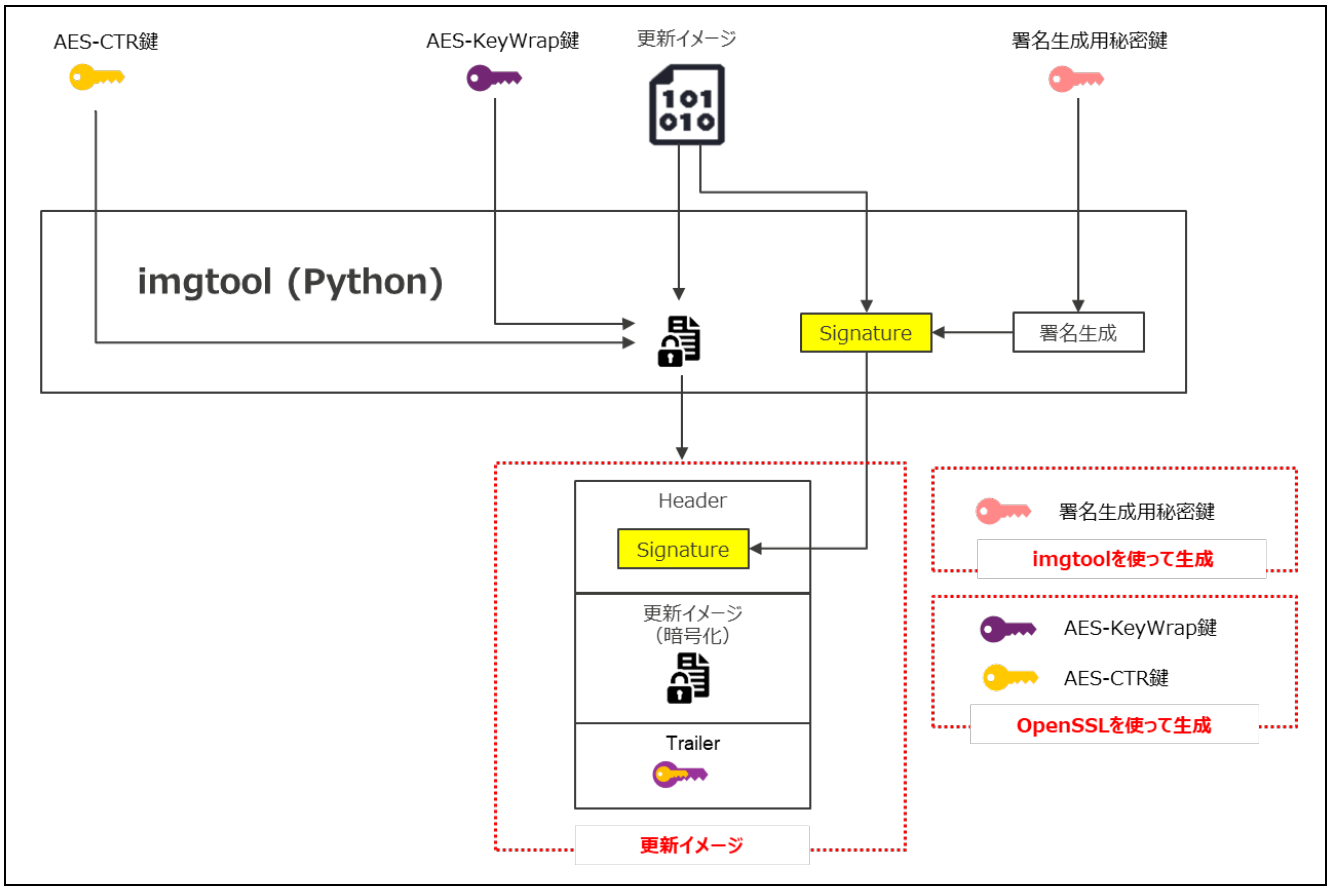

図 4-11 更新イメージを生成（イメージの暗号化あり）

4.3.4 デモプロジェクトの書き込み

4.3.3 で生成したデモプロジェクトに使用するイメージを RFP で書き込みます。

なお、Step1 のフラッシュ消去と Step2 以降の書き込みについては、RFP での消去後の書き込みの操作で、複数のファイルを同時に指定して一度に実施してください。

アップデート方式およびフラッシュメモリの構成によって書き込み先アドレスが異なります。書き込み先は、5.2 デモプロジェクトの動作環境の各方式のメモリマップを参照して下さい。

リニアモードの場合：

Step1) 4.3.1.3 で生成した HUK でラッピングした鍵データ以外のコードフラッシュを RFP で消去します。

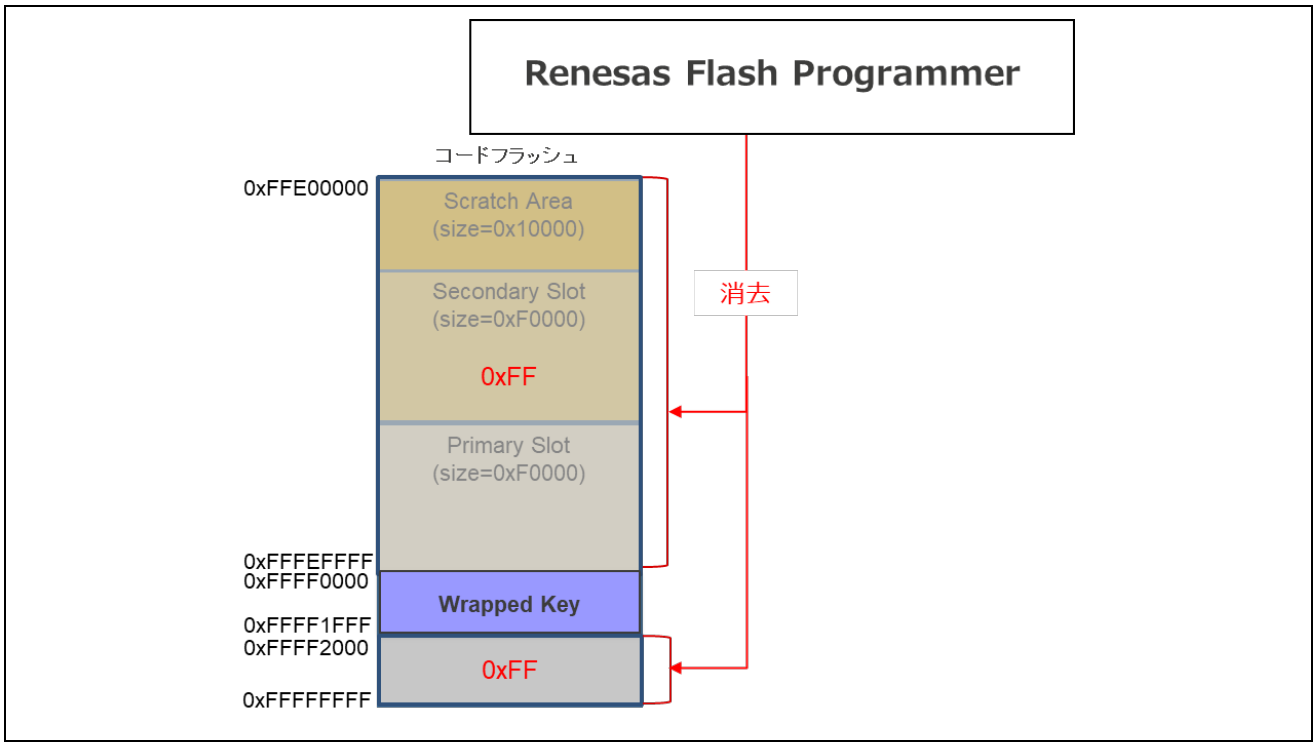

図 4-12 書き込み先の消去（リニアモード）

Step2) 4.3.3.1 で生成したブートローダ（MCUBoot）を Bootloader 領域に、4.3.2.2 で生成した初期イメージを Primary slot に RFP で書き込みます。

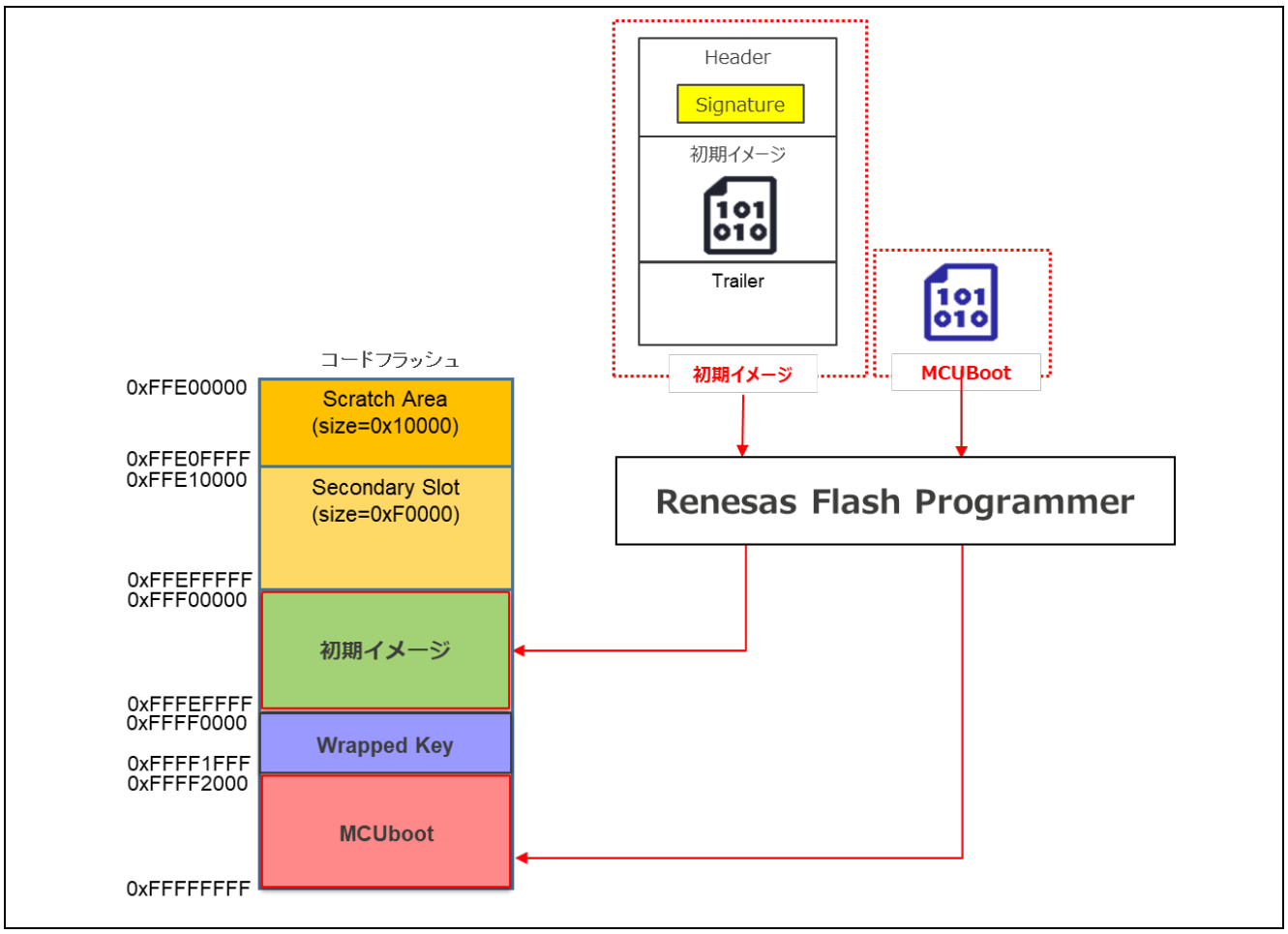

図 4-13 デモプロジェクトの書き込み（リニアモード）

デュアルモードの場合：

Step1) 4.3.1.3 で生成した HUK でラッピングした鍵データ以外のコードフラッシュを RFP で消去します。

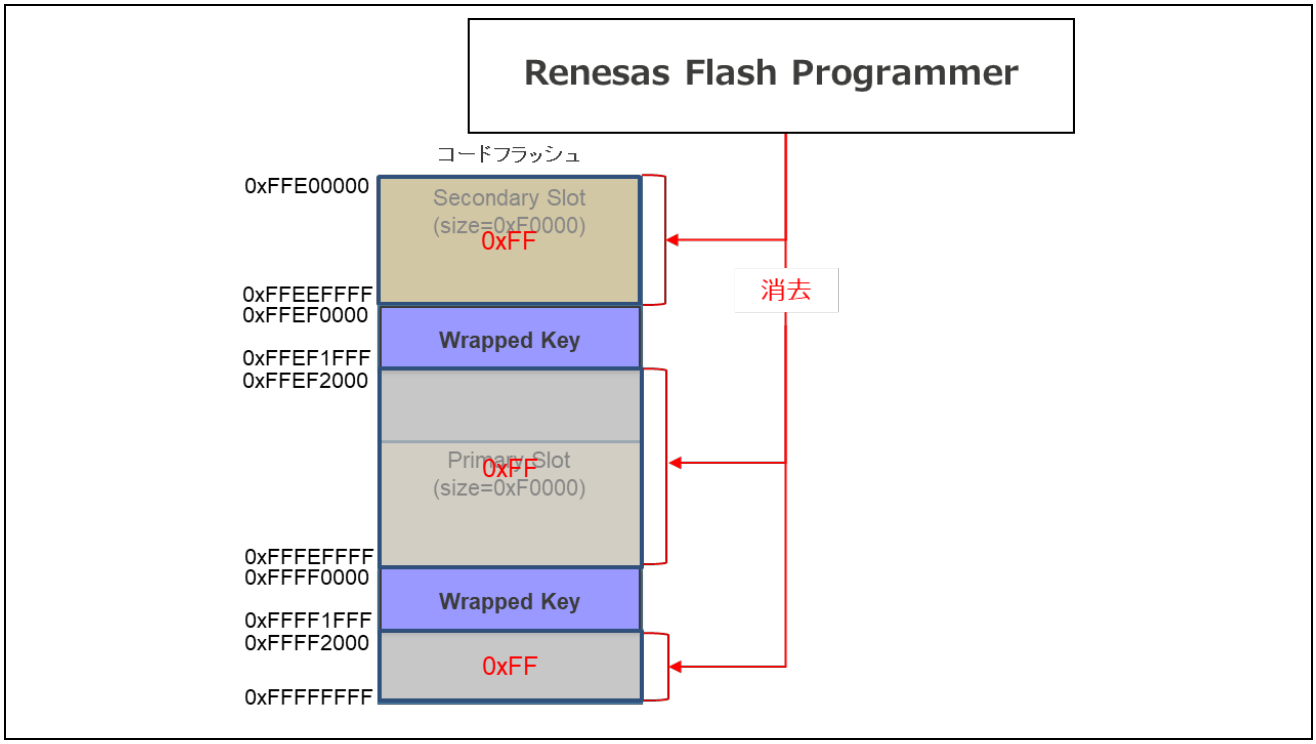

図 4-14 書き込み先の消去（デュアルモード）

Step2) 4.3.3.1 で生成したブートローダ（MCUboot）を Bank0 および Bank1 の Bootloader 領域に、
4.3.3.2 で生成した初期イメージを Primary slot に RFP で書き込みます。

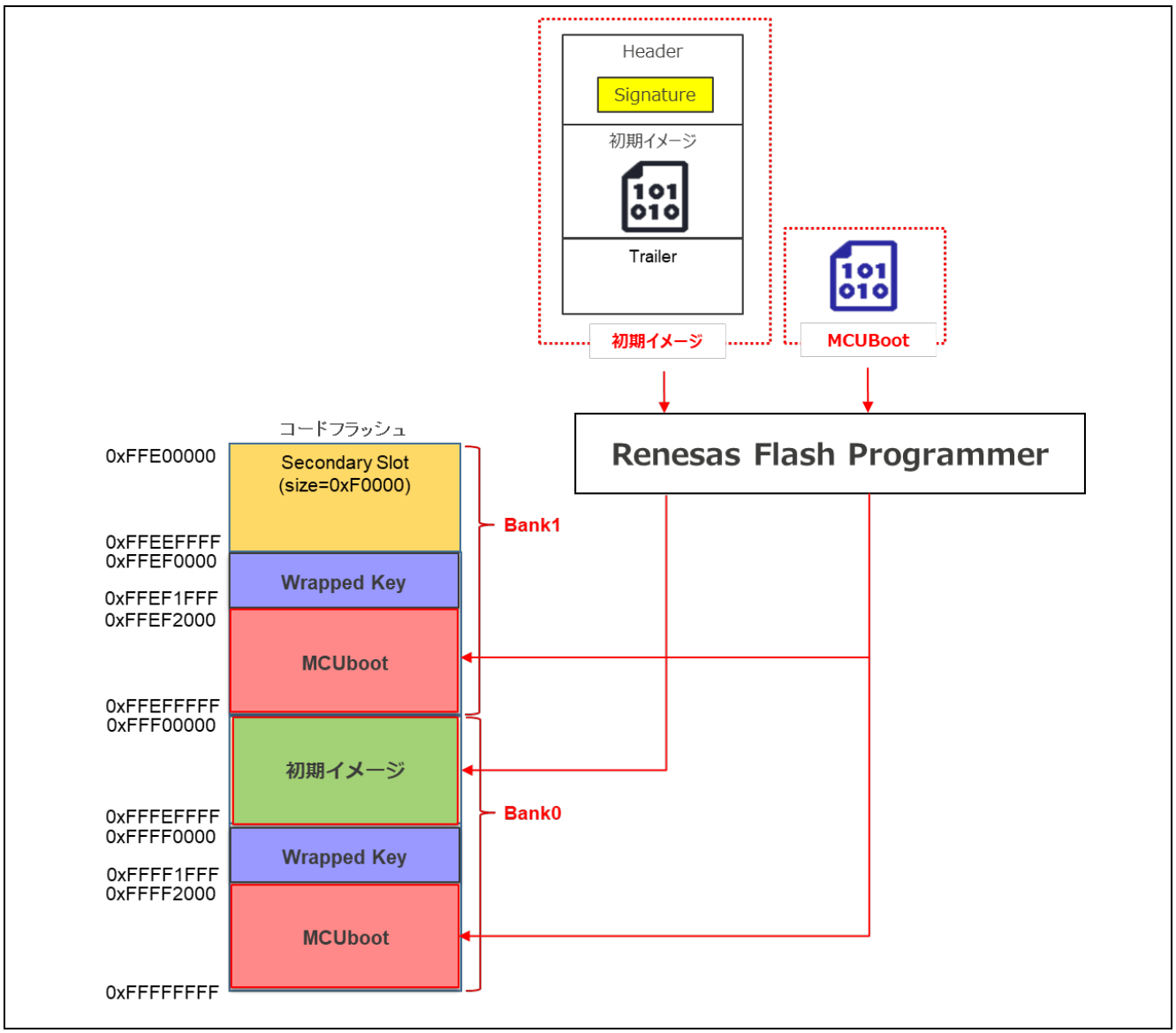

図 4-15 デモプロジェクトの書き込み（デュアルモード）

4.3.5 デモプロジェクトの実行

4.3.4 で書き込んだデモプロジェクトを実行します。

デモプロジェクトを起動すると、ブートローダ（MCUboot）が起動し primary slot に書き込んだ初期イメージを起動します。このアプリケーションはターミナル経由で更新イメージを受信し Secondary slot に書き込みます。

書き込み完了後にソフトウェアリセットを行い、再度ブートローダ（MCUboot）を起動します。

以下の手順により、デモプロジェクトを実行してください。

① 「5.2.2 RX65N の動作確認環境」を参考に機器を接続してください。

② PC のターミナルソフトを起動しシリアル COM ポートを選択し接続設定を行います。

③ ボードの電源を投入します。ブートローダ（MCUboot）が起動し Primary slot に書き込んだ初期イメージが起動します。

```
Update start
[INF] Primary   slot: version=1.0.0+0
[INF] Image 0 Secondary slot: Image not found
[INF] Image 0 loaded from the primary slot
------------------------------------------------------------------
Primary Slot Application Image Start (ver 1.0.0)
------------------------------------------------------------------
Erase the code flash of the Secondary slot.
```

④ 初期イメージでは更新イメージの受信待ち状態になりますので、ターミナルから更新イメージを送信します。

受信した更新イメージを Secondary slot に書き込み中は以下のメッセージが出力されます。

```
send user program (MCUboot image) via UART.
W 0xffe10000, 512 ... OK
W 0xffe10200, 512 ... OK
...
W 0xffeffc00, 512 ... OK
W 0xffeffe00, 512 ... OK
```

⑤ 更新イメージの書き込みが完了すると、ソフトウェアリセットを実行します。

```
software reset..
```

⑥ 再度ブートローダ（MCUboot）が起動され、アップデートモードにしたがって、MCUboot がアップデートを行います。アップデートが完了すると更新されたイメージを起動します。

```
Update start
[INF] Swap type: perm
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] Copying the secondary slot to the primary slot: 0xf0000 bytes
------------------------------------------------------------------
Primary Slot Application Image Start (ver 1.1.0)
------------------------------------------------------------------
Erase the code flash of the Secondary slot.
send user program (MCUboot image) via UART.
```

5. 付録

5.1 動作確認環境

本モジュールの動作確認環境を以下に示します。

表 5-1 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 2025-04
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.07.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	リトルエンディアン
モジュールのリビジョン	Rev.1.01
使用ボード	Evaluation Kit for RX261（製品型名：RTK5EK2610SxxxxxBE） Renesas Starter Kit+ for RX65N（製品型名：RTK50565N2SxxxxxBE） Renesas Starter Kit+ for RX671（製品型名：RTK55671EHSxxxxxBE） Renesas Starter Kit+ for RX72M（製品型名：RTK5572MNHSxxxxxBE） Renesas Starter Kit+ for RX72N（製品型名：RTK5572NNHSxxxxxBE）

表 5-2 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 2025-04
C コンパイラ	GCC for Renesas RX 8.3.0.202411 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
エンディアン	リトルエンディアン
モジュールのリビジョン	Rev.1.01
使用ボード	Evaluation Kit for RX261（製品型名：RTK5EK2610SxxxxxBE） Renesas Starter Kit+ for RX65N（製品型名：RTK50565N2SxxxxxBE） Renesas Starter Kit+ for RX671（製品型名：RTK55671EHSxxxxxBE） Renesas Starter Kit+ for RX72M（製品型名：RTK5572MNHSxxxxxBE） Renesas Starter Kit+ for RX72N（製品型名：RTK5572NNHSxxxxxBE）

表 5-3 動作確認環境 (IAR)

項目	内容
統合開発環境	IAR Systems 製 IAR Embedded Workbench for Renesas RX 5.10.1 RX スマート・コンフィグレータ V2.25.0
C コンパイラ	IAR Systems 製 IAR C/C++ Compiler for Renesas RX 5.10.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	リトルエンディアン
モジュールのリビジョン	Rev.1.01
使用ボード	Evaluation Kit for RX261（製品型名：RTK5EK2610SxxxxxBE） Renesas Starter Kit+ for RX65N（製品型名：RTK50565N2SxxxxxBE） Renesas Starter Kit+ for RX671（製品型名：RTK55671EHSxxxxxBE） Renesas Starter Kit+ for RX72M（製品型名：RTK5572MNHSxxxxxBE） Renesas Starter Kit+ for RX72N（製品型名：RTK5572NNHSxxxxxBE）

MCUboot の動作確認のために、デモプロジェクトで使用した FIT モジュールのバージョン一覧を以下に示します。

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

表 5-4 FIT モジュールのバージョン一覧（CC-RX）

デバイス	プロジェクト	r_bsp	r_flash_rx	r_tsip	r_rsip_protected_rx	r_sci_rx	r_byteq	rm_mcuboot
RX65N, RX66N, RX671, RX72M, RX72N	application_primary	7.53	5.22	1.22	—	5.41	2.11	1.01
	boot_loader	7.53	5.22	1.22	—	5.41	2.11	1.01
	key_injection	7.53	5.22	1.22	—	5.41	2.11	1.01
RX261	application_primary	7.53	5.22	—	1.00	5.41	2.11	1.01
	boot_loader	7.53	5.22	—	1.00	5.41	2.11	1.01
	key_injection	7.53	5.22	—	1.00	5.41	2.11	1.01

(2) GCC for Renesas RX の環境

表 5-5 FIT モジュールのバージョン一覧（GCC）

デバイス	プロジェクト	r_bsp	r_flash_rx	r_tsip	r_rsip_protected_rx	r_sci_rx	r_byteq	rm_mcuboot
RX65N, RX66N, RX671, RX72M, RX72N	application_primary	7.53	5.22	1.22	—	5.41	2.11	1.01
	boot_loader	7.53	5.22	1.22	—	5.41	2.11	1.01
	key_injection	7.53	5.22	1.22	—	5.41	2.11	1.01
RX261	application_primary	7.53	5.22	—	1.00	5.41	2.11	1.01
	boot_loader	7.53	5.22	—	1.00	5.41	2.11	1.01
	key_injection	7.53	5.22	—	1.00	5.41	2.11	1.01

(3) IAR C/C++ Compiler for RX の環境

表 5-6 FIT モジュールのバージョン一覧(IAR)

デバイス	プロジェクト	r_bsp	r_flash_rx	r_tsip	r_rsip_protected_rx	r_sci_rx	r_byteq	rm_mcuboot
RX65N, RX66N, RX671, RX72M, RX72N	application_primary	7.53	5.22	1.22	—	5.41	2.11	1.01
	boot_loader	7.53	5.22	1.22	—	5.41	2.11	1.01
	key_injection	7.53	5.22	1.22	—	5.41	2.11	1.01
RX261	application_primary	7.53	5.22	—	1.00	5.41	2.11	1.01
	boot_loader	7.53	5.22	—	1.00	5.41	2.11	1.01
	key_injection	7.53	5.22	—	1.00	5.41	2.11	1.01

5.2 デモプロジェクトの動作環境

デモプロジェクトは複数の製品に対応しています。デモプロジェクトを使用するにあたり、製品毎に異なる設定を以下に示します。

5.2.1 RX261 の動作確認環境

5.2.1.1 機器接続情報

EK-RX261 の機器接続情報を以下に示します。

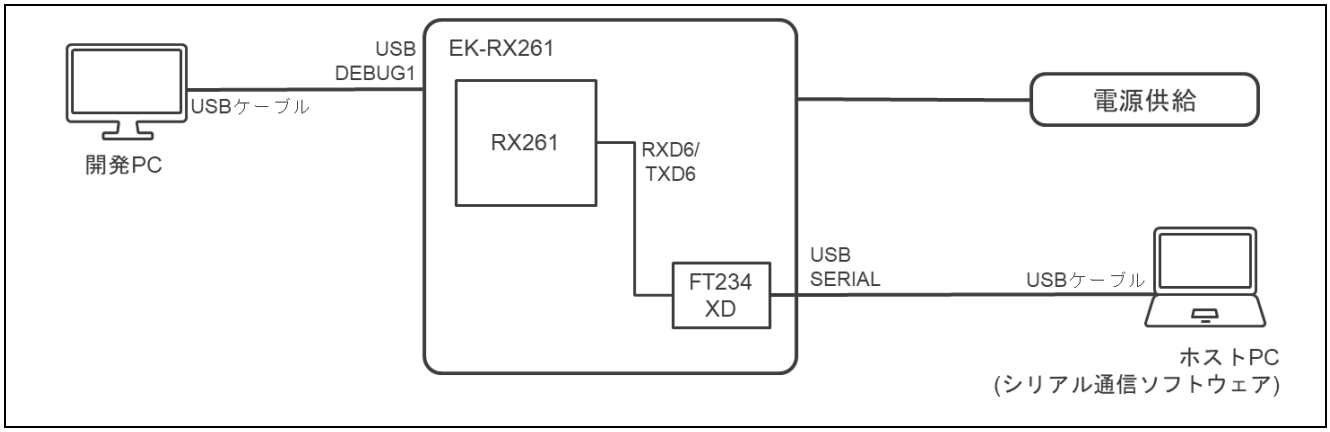

図 5-1 EK-RX261 機器接続図

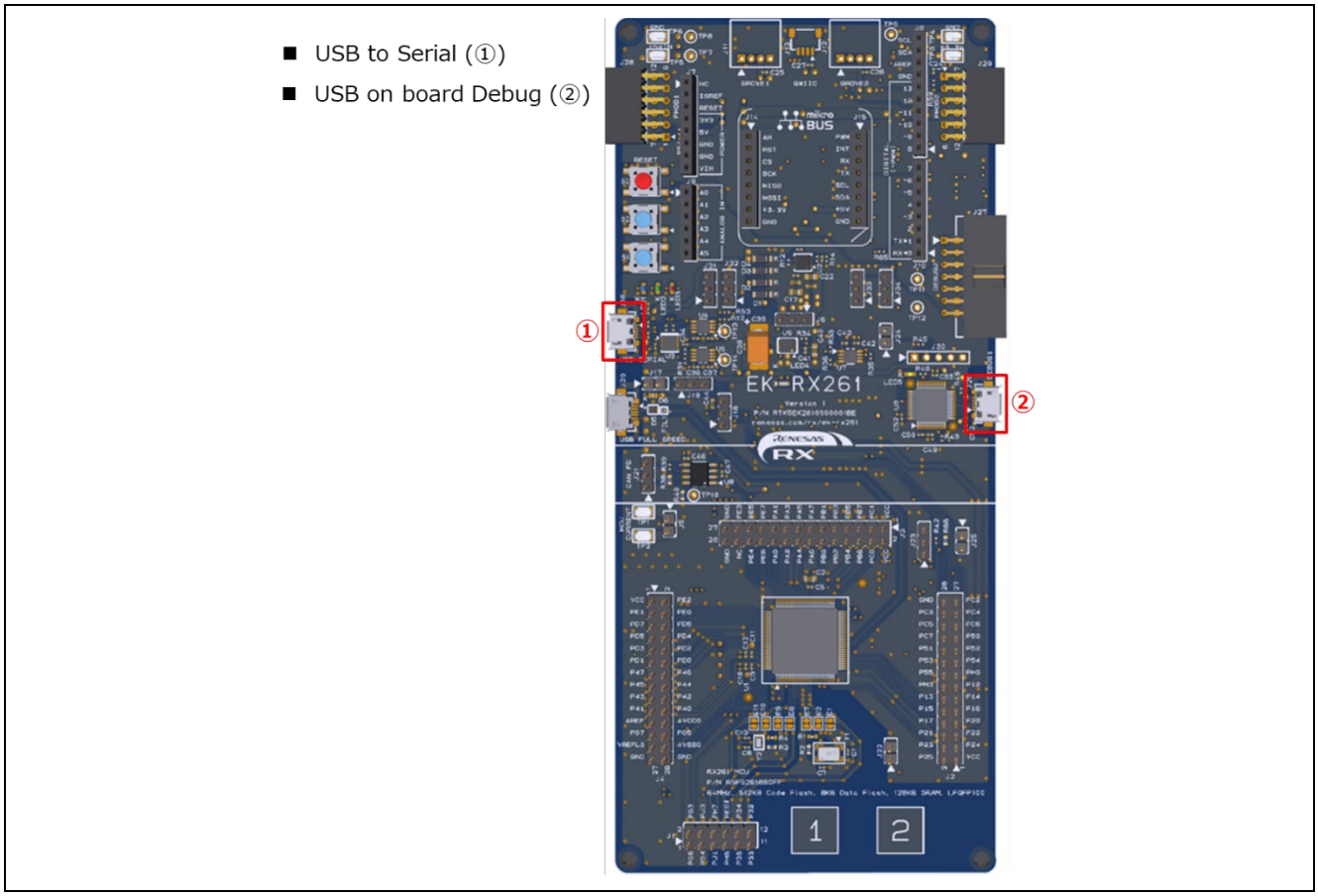

図 5-2 EK-RX261 ボード接続情報

5.2.1.2 メモリ配置およびコンフィグ設定値

EK-RX261 のデモプロジェクトのメモリ配置を示します。

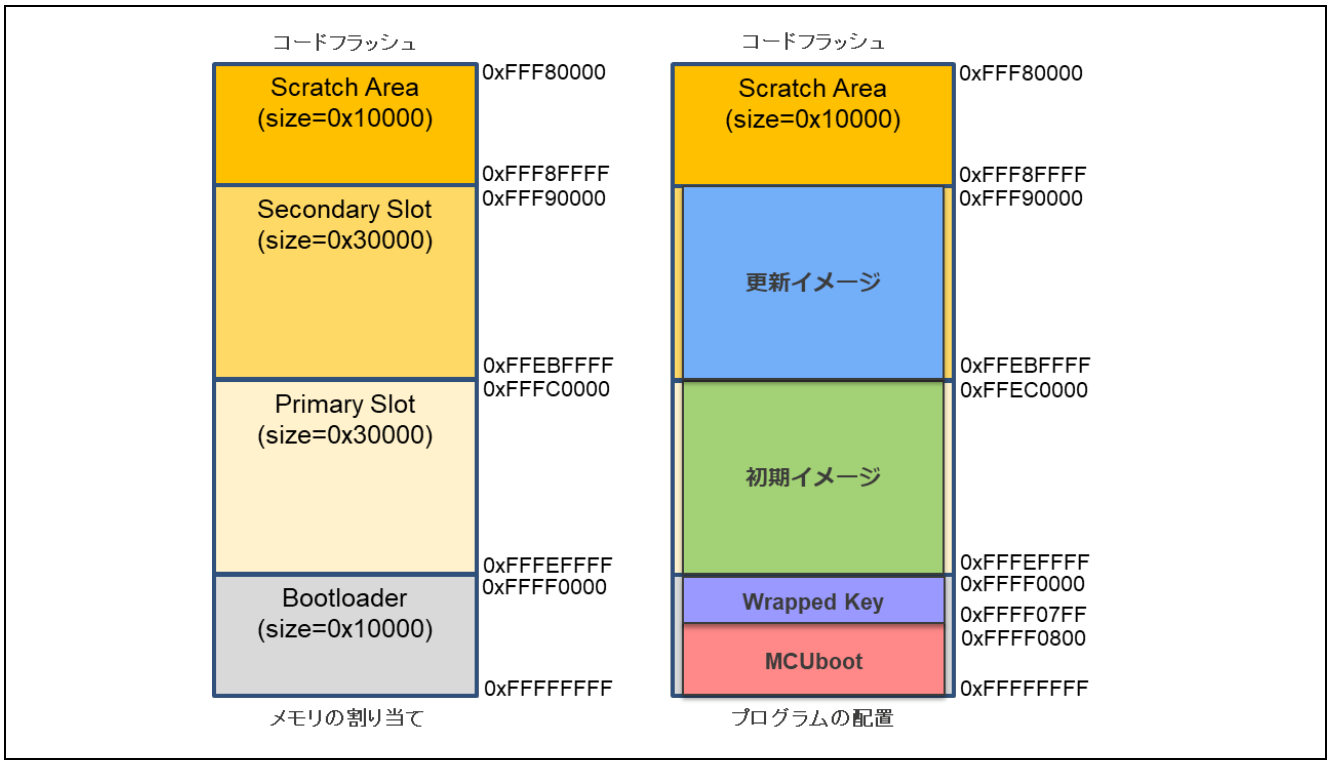

図 5-3 EK-RX261 デモプロジェクトのメモリマップ

表 5-7 EK-RX261 デモプロジェクトのコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	0～3 を選択
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x30000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.2 RX65N の動作確認環境

5.2.2.1 リニアモードを使用した方式の機器接続情報

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式の RSK-RX65N の機器接続情報を以下に示します。

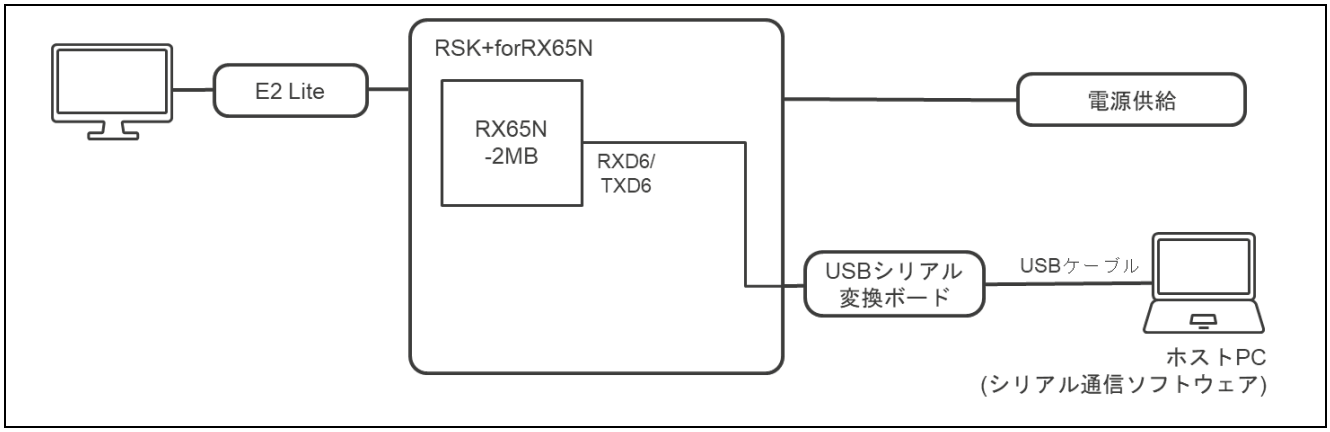

図 5-4 RSK-RX65N リニアモードを使用した方式の機器接続図

■ USB Serial (①)

PMOD1		USB-Serial
2	TXD6	RX
3	RXD6	TX
4	P02(RTS)	CTS

■ Debugger (②)

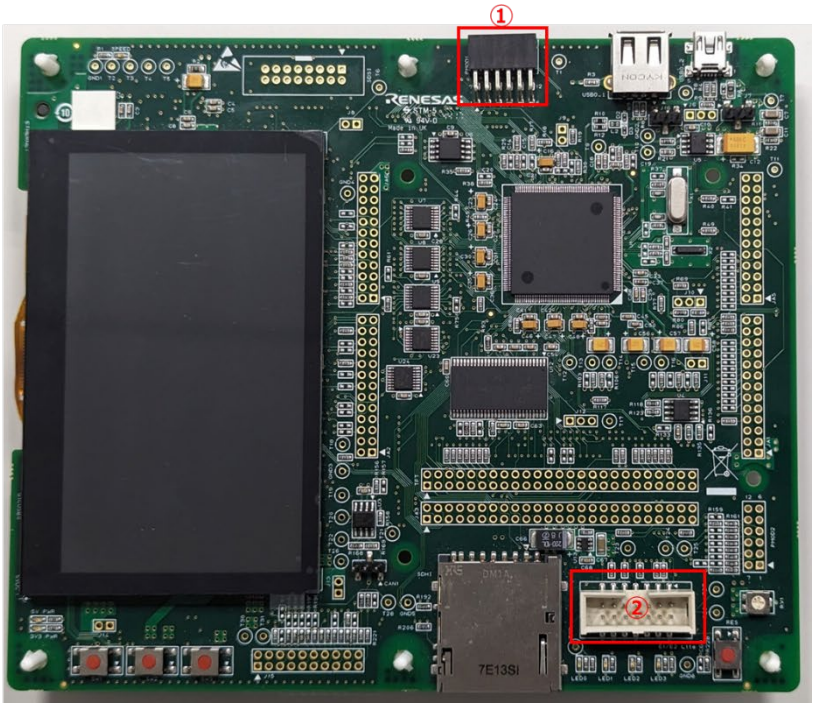

図 5-5 RSK-RX65N リニアモードを使用した方式の接続情報

5.2.2.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

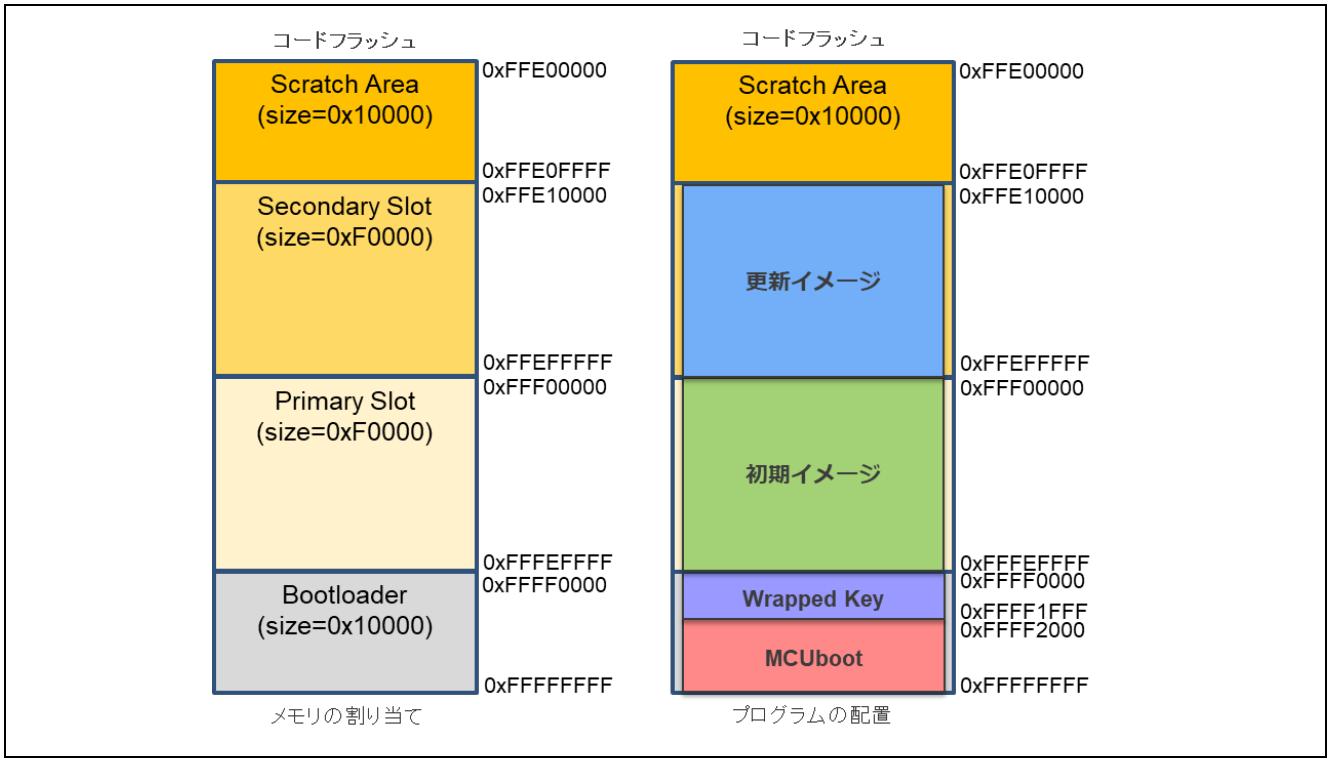

図 5-6 RSK-RX65N(2MB) リニアモードを使用した方式のデモプロジェクトのメモリマップ

表 5-8 RSK-RX65N(2MB) リニアモードを使用した方式のコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	0～3 を選択
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.2.3 デュアルモードを使用した方式の機器接続情報

フラッシュメモリのデュアルモードを使用した DirectXIP 方式の機器接続情報を以下に示します。

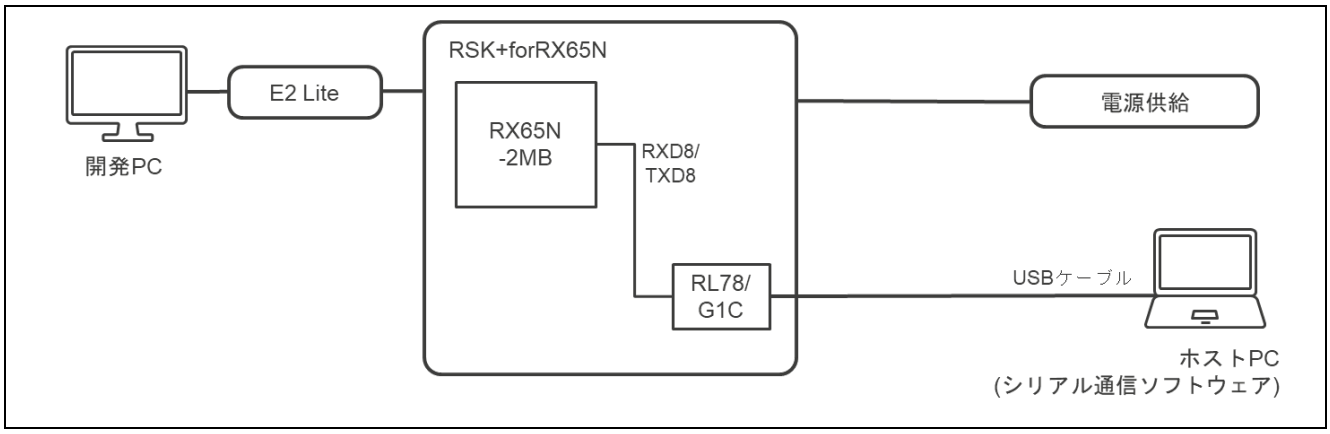

図 5-7 RSK-RX65N デュアルモードを使用した方式の機器接続図

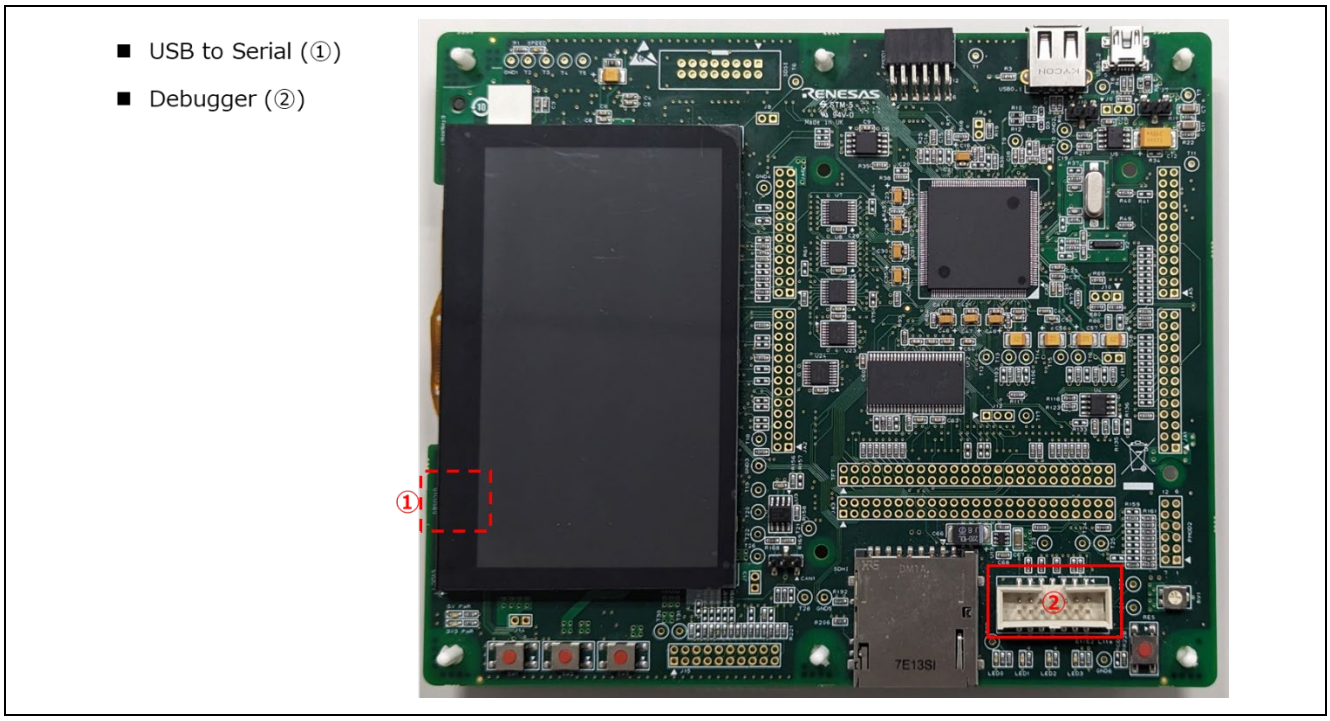

図 5-8 RSK-RX65N デュアルモードを使用した方式の接続情報

5.2.2.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのデュアルモードを使用した DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

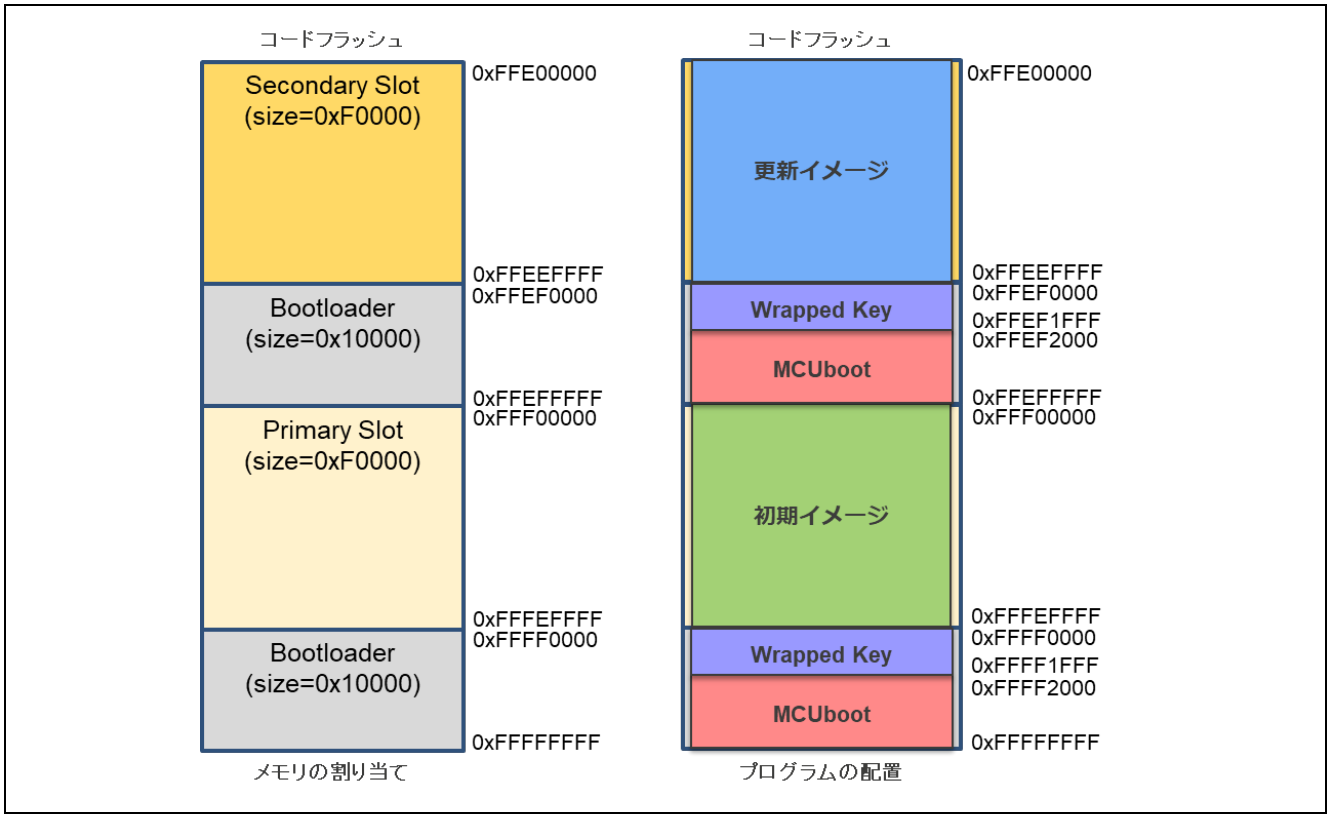

図 5-9 RSK-RX65N(2MB) デュアルモードを使用した方式のデモプロジェクトのメモリマップ

表 5-9 RSK-RX65N(2MB) デュアルモードを使用した方式のデモプロジェクトのコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3（DirectXIP）
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0（本方式では無効な設定）
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	0（本方式では無効な設定）
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0（本方式では無効な設定）
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.3 RX671 の動作確認環境

5.2.3.1 リニアモードを使用した方式の機器接続情報

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式の RSK-RX671 の機器接続情報を以下に示します。

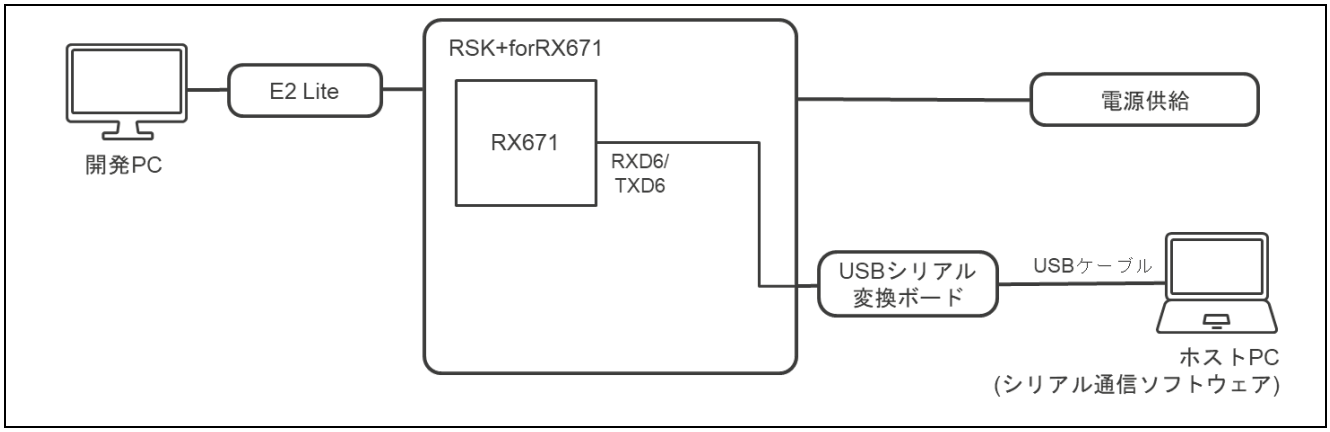

図 5-10 RSK-RX671 リニアモードを使用した方式の機器接続図

■ USB Serial (①)

PMOD1		USB-Serial
2	TXD6	RX
3	RXD6	TX
4	P02(RTS)	CTS

■ Debugger (②)

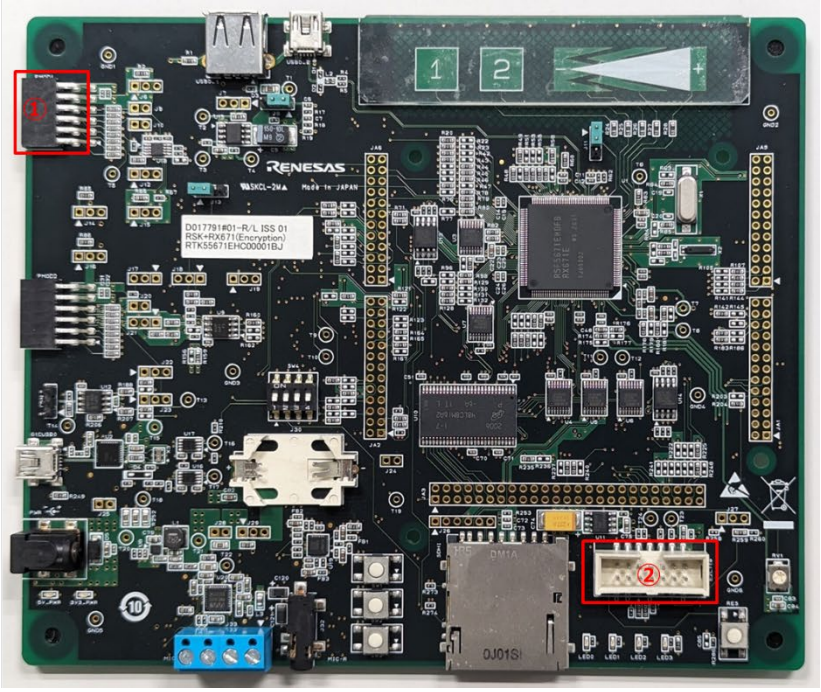

図 5-11 RSK-RX671 リニアモードを使用した方式の接続情報

5.2.3.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

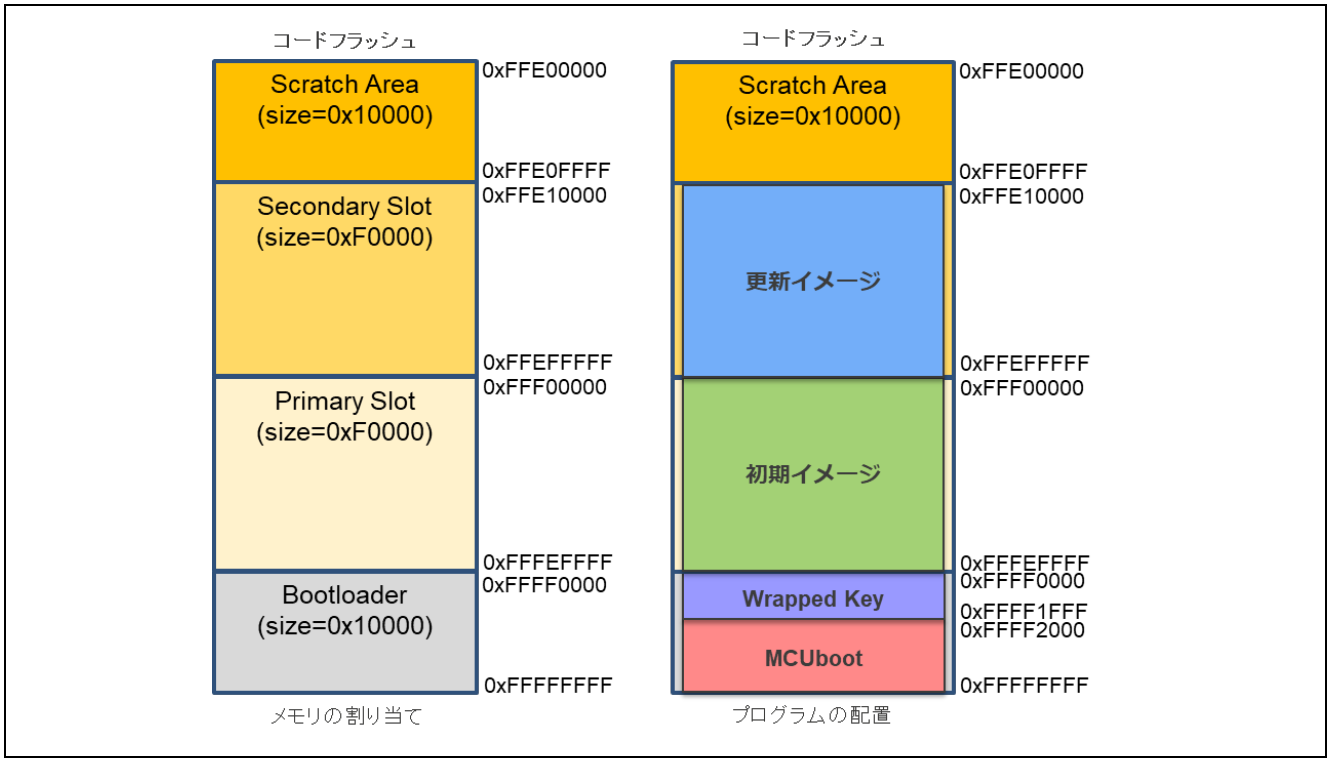

図 5-12 RSK-RX671 リニアモードを使用した方式のデモプロジェクトのメモリマップ

表 5-10 RSK-RX671 リニアモードを使用した方式のコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	0～3 を選択
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.3.3 デュアルモードを使用した方式の機器接続情報

フラッシュメモリのデュアルモードを使用した DirectXIP 方式の機器接続情報を以下に示します。

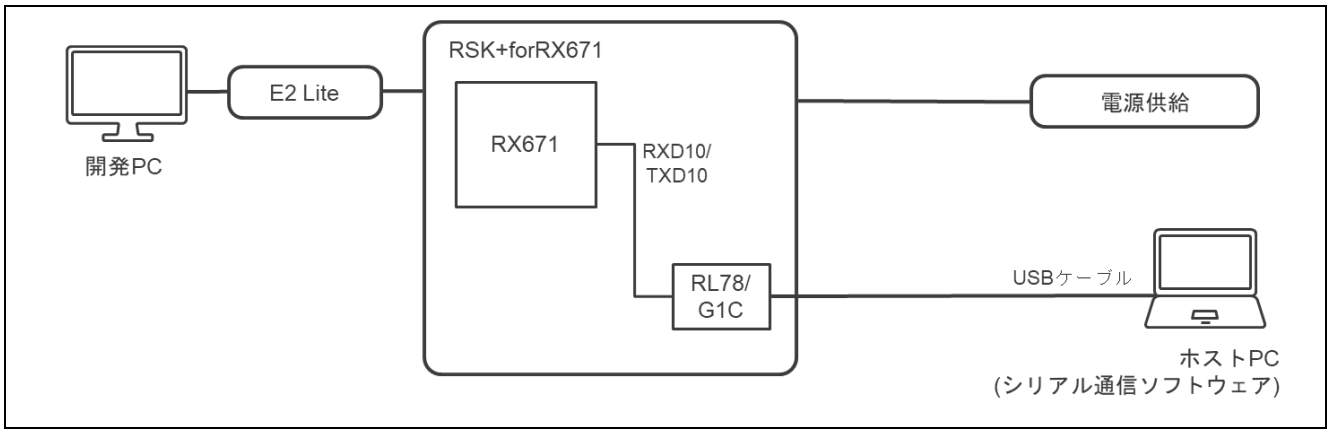

図 5-13 RSK-RX671 デュアルモードを使用した方式の機器接続図

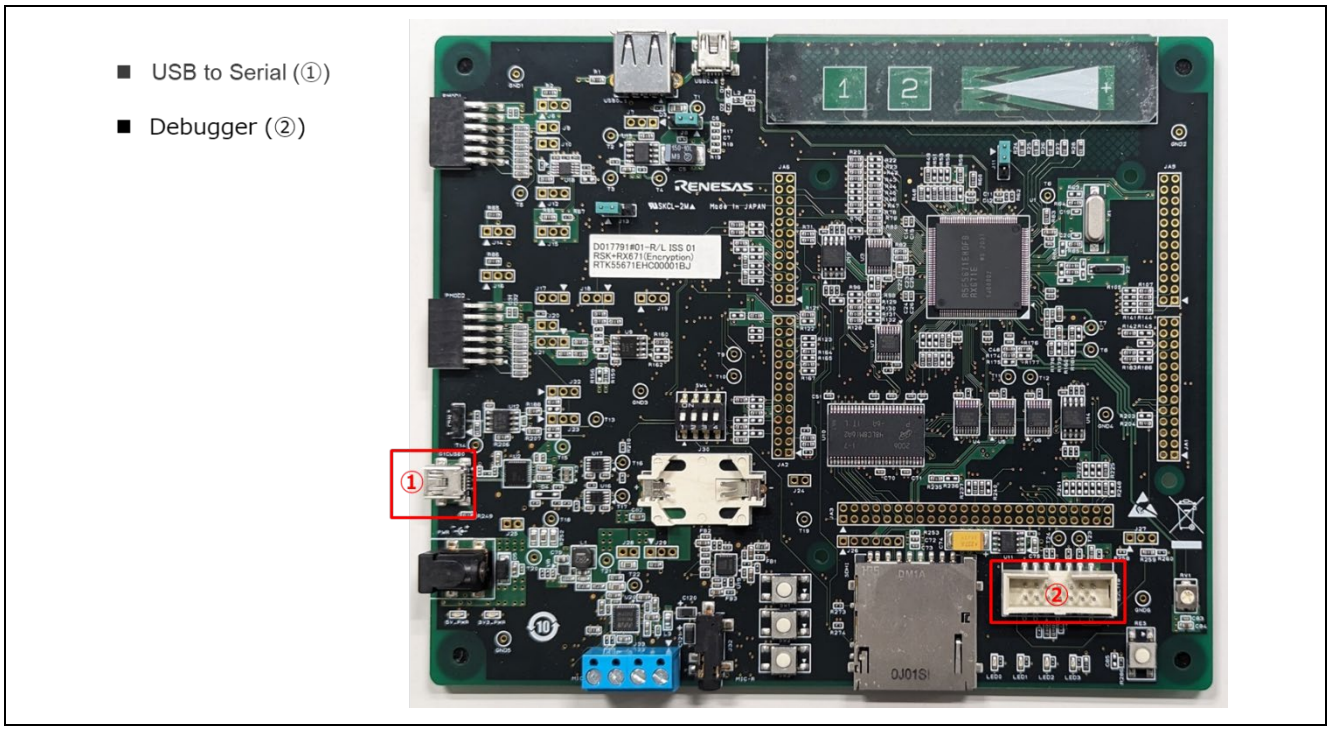

図 5-14 RSK-RX671 デュアルモードを使用した方式の接続情報

5.2.3.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのデュアルモードを使用した DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

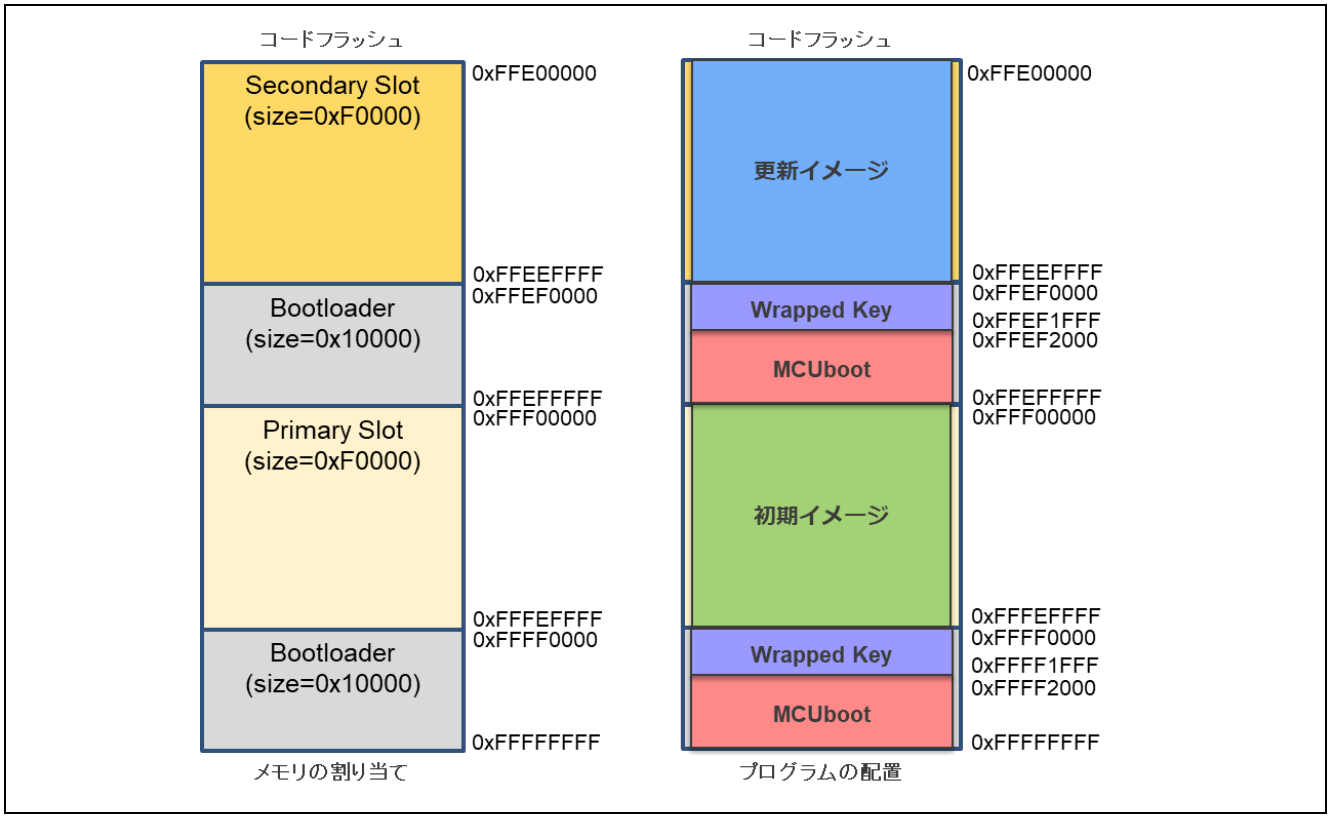

図 5-15 RSK-RX671 デュアルモードを使用した方式のデモプロジェクトのメモリマップ

表 5-11 RSK-RX671 デュアルモードを使用した方式のデモプロジェクトのコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3（DirectXIP）
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0（本方式では無効な設定）
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	0（本方式では無効な設定）
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0xF0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0（本方式では無効な設定）
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.4 RX72M の動作確認環境

5.2.4.1 リニアモードを使用した方式の機器接続情報

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式の RSK-RX72M の機器接続情報を以下に示します。

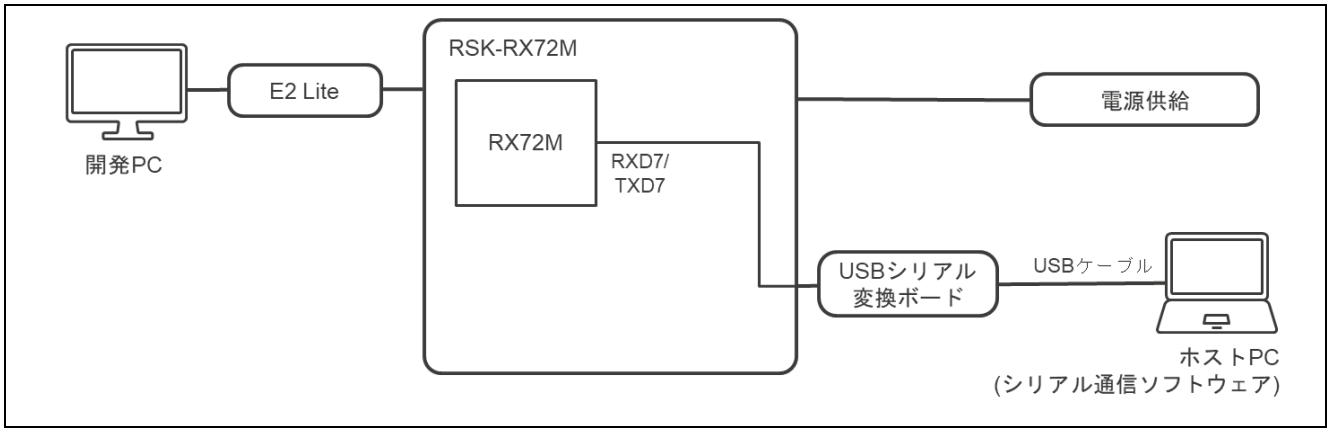

図 5-16 RSK-RX72M リニアモードを使用した方式の機器接続図

■ USB Serial (①)

PMOD1		USB-UART
2	TXD7	RX
3	RXD7	TX
4	PH0(RTS)	CTS

■ Debugger (②)

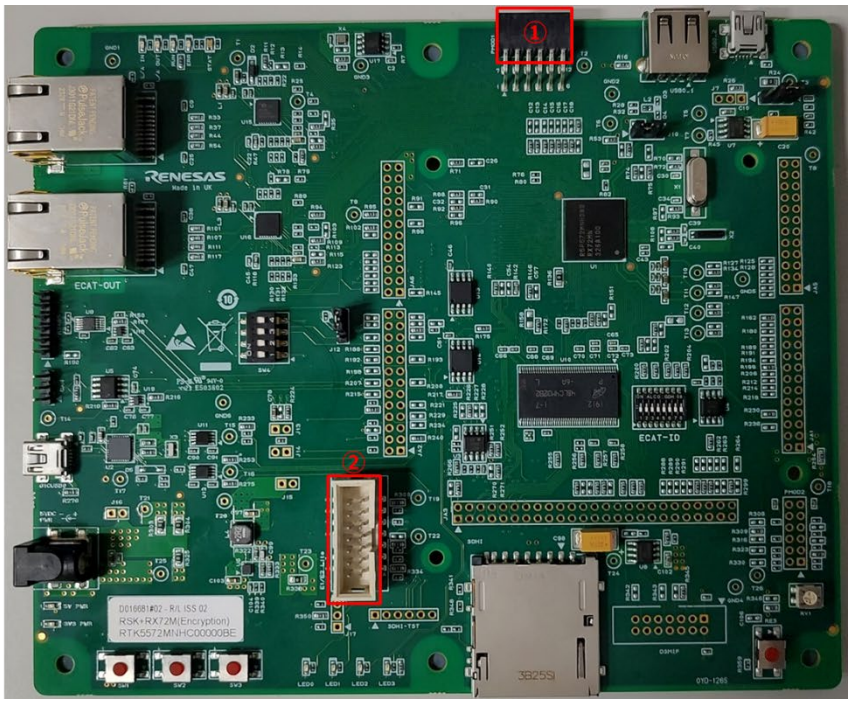

図 5-17 RSK-RX72M リニアモードを使用した方式の接続情報

5.2.4.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

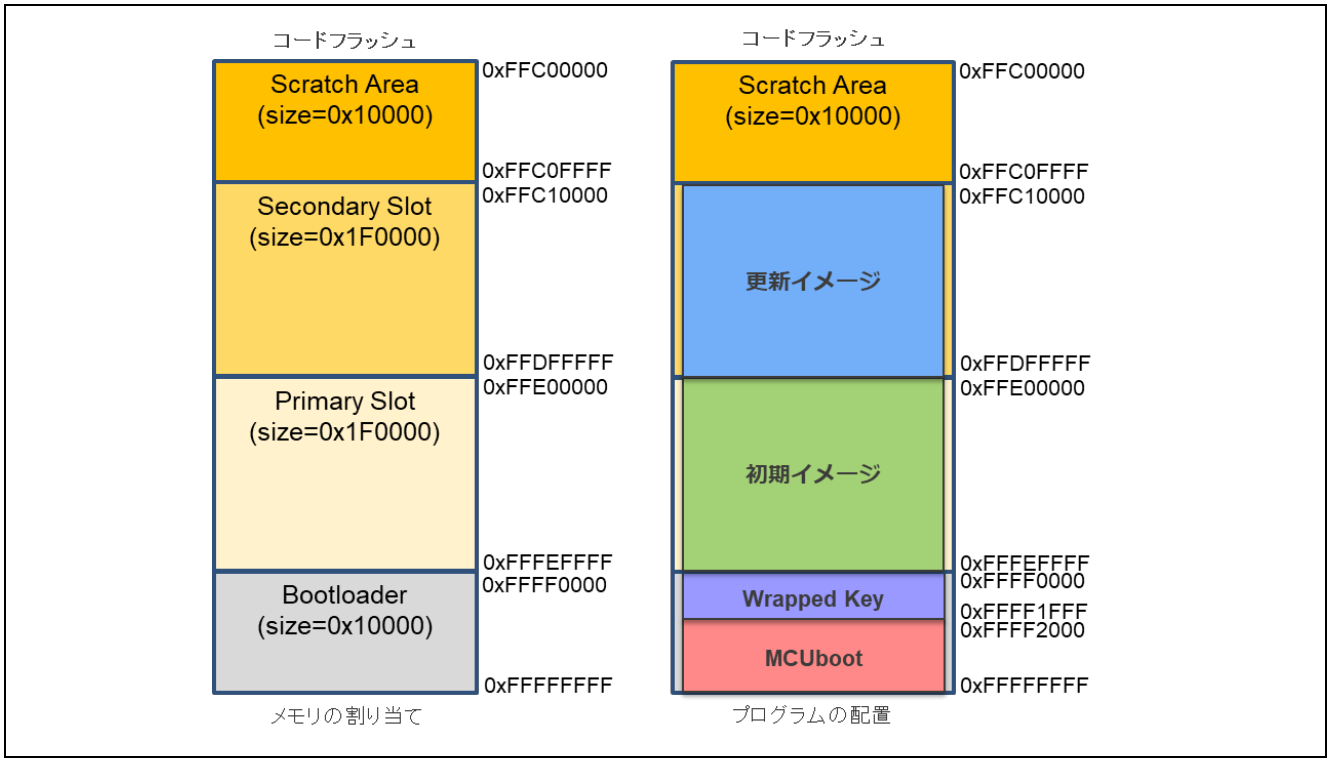

図 5-18 RSK-RX72M リニアモードを使用した方式のデモプロジェクトのメモリマップ

表 5-12 RSK-RX72M リニアモードを使用した方式のコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	0～3 を選択
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.4.3 デュアルモードを使用した方式の機器接続情報

フラッシュメモリのデュアルモードを使用した DirectXIP 方式の機器接続情報を以下に示します。

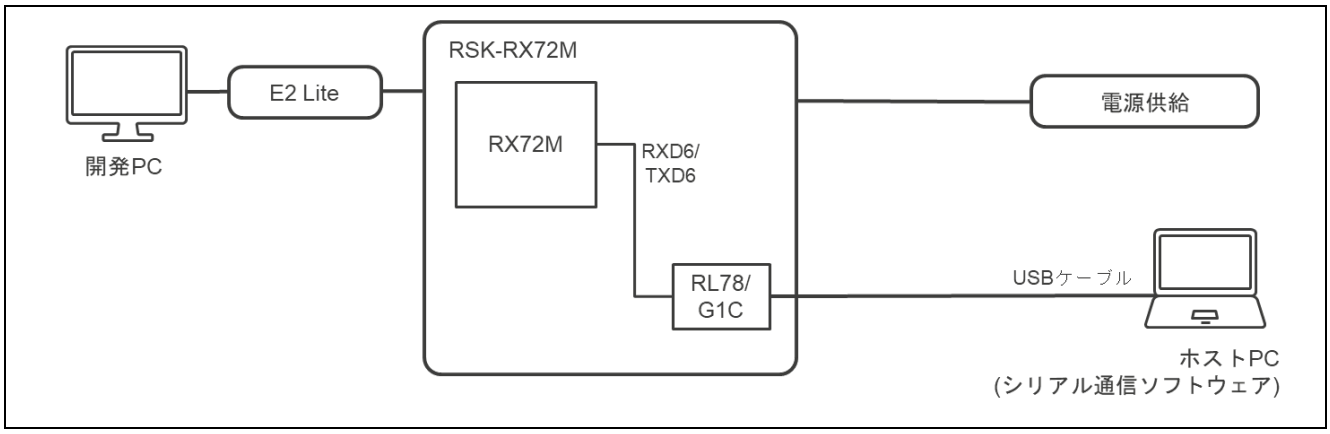

図 5-19 RSK-RX72M デュアルモードを使用した方式の機器接続図

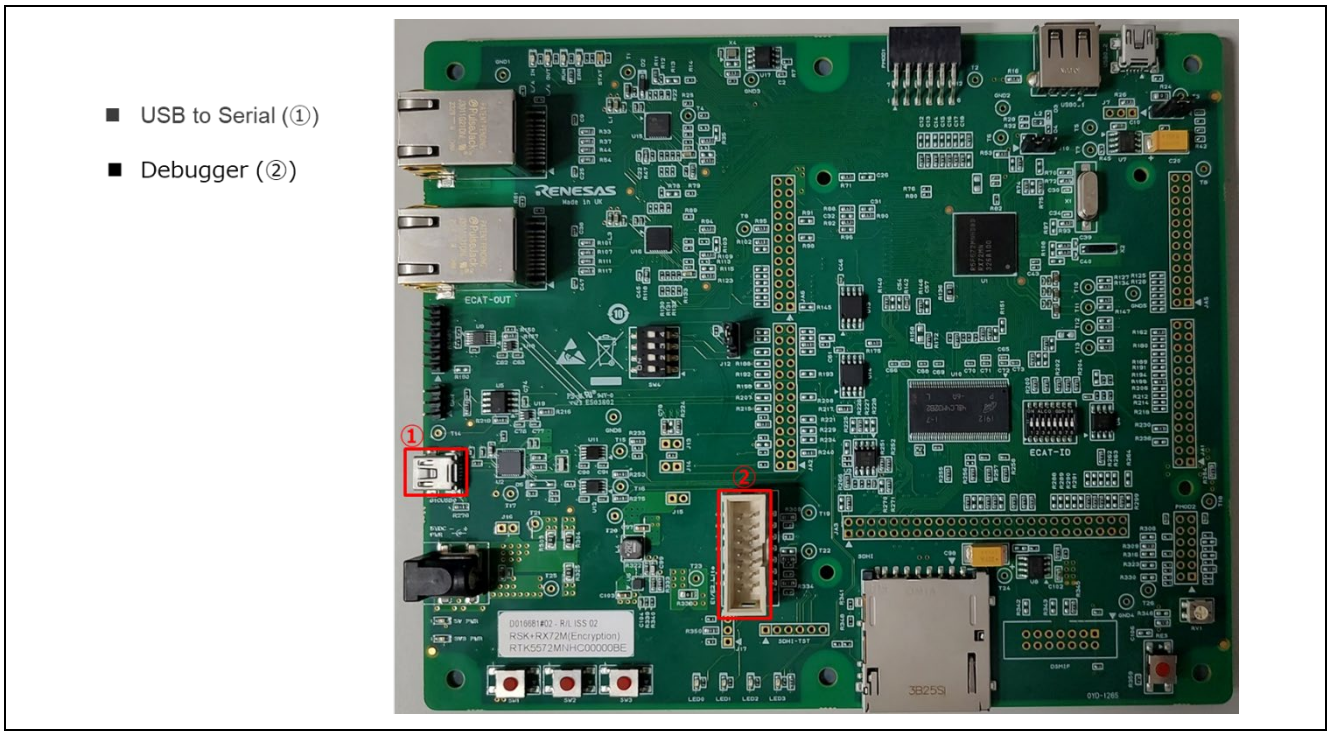

図 5-20 RSK-RX72M デュアルモードを使用した方式の接続情報

5.2.4.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのデュアルモードを使用した DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

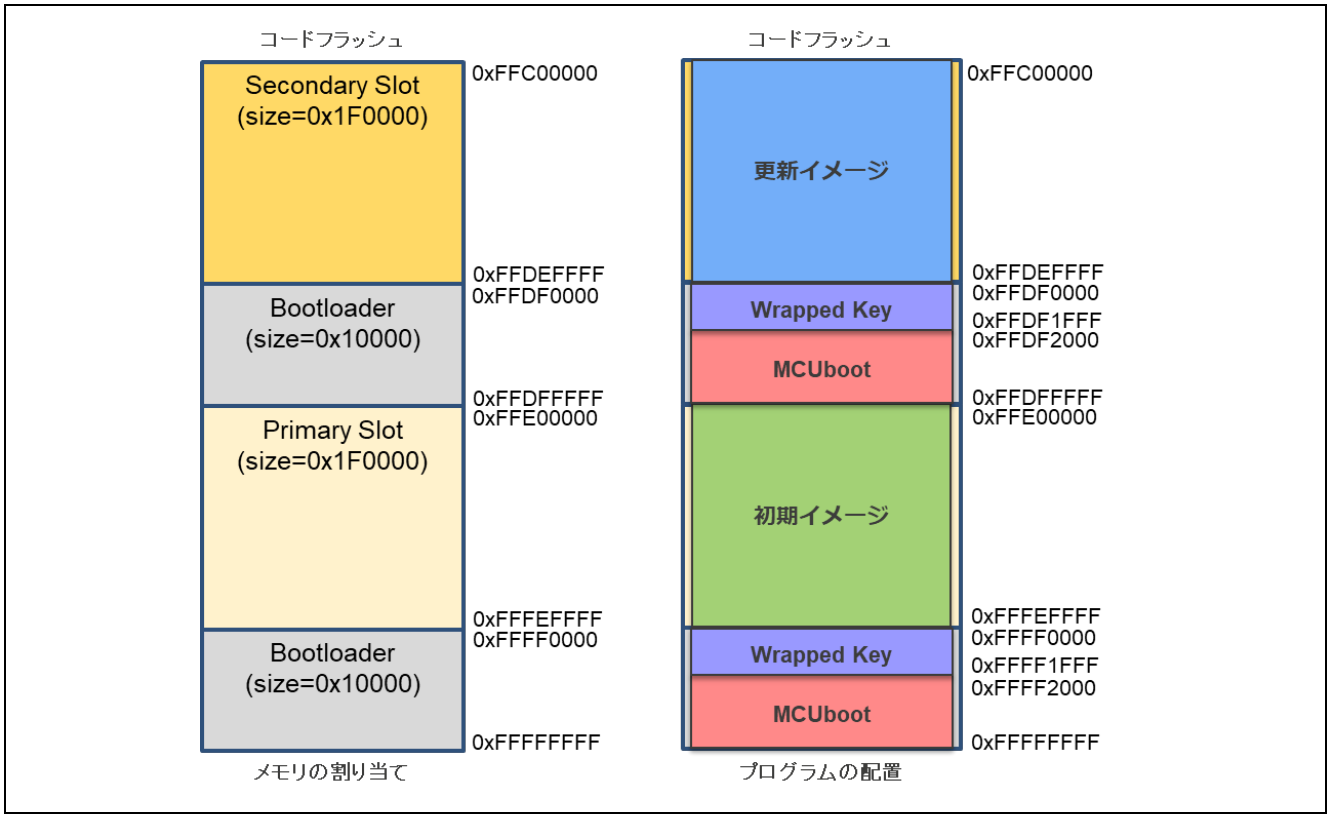

図 5-21 RSK-RX72M デュアルモードを使用した方式のデモプロジェクトのメモリマップ

表 5-13 RSK-RX72M デュアルモードを使用した方式のデモプロジェクトのコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3（DirectXIP）
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0（本方式では無効な設定）
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	0（本方式では無効な設定）
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0（本方式では無効な設定）
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.5 RX72N の動作確認環境

5.2.5.1 リニアモードを使用した方式の機器接続情報

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式の RSK-RX72N の機器接続情報を以下に示します。

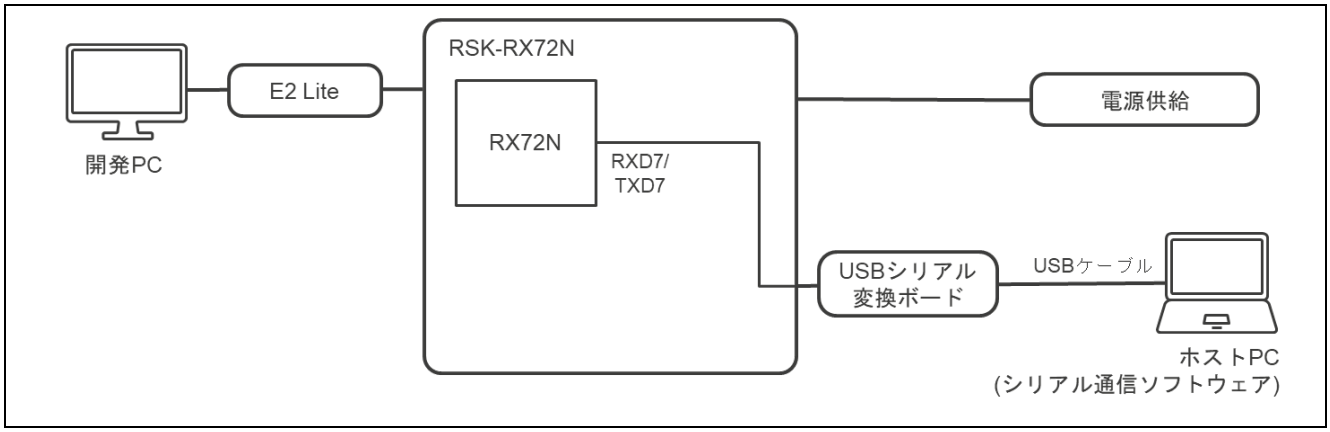

図 5-22 RSK-RX72N リニアモードを使用した方式の機器接続図

■ USB Serial (①)

PMOD1		USB-UART
2	TXD7	RX
3	RXD7	TX
4	PH0(RTS)	CTS

■ Debugger (②)

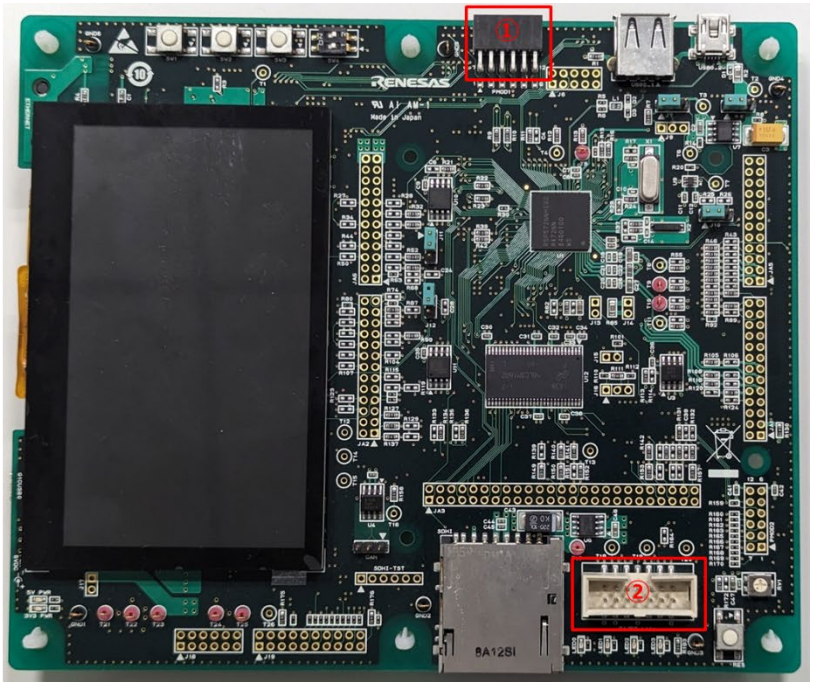

図 5-23 RSK-RX72N リニアモードを使用した方式の接続情報

5.2.5.2 リニアモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのリニアモードを使用した Overwrite Only／Overwrite Only Fast／Swap／DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

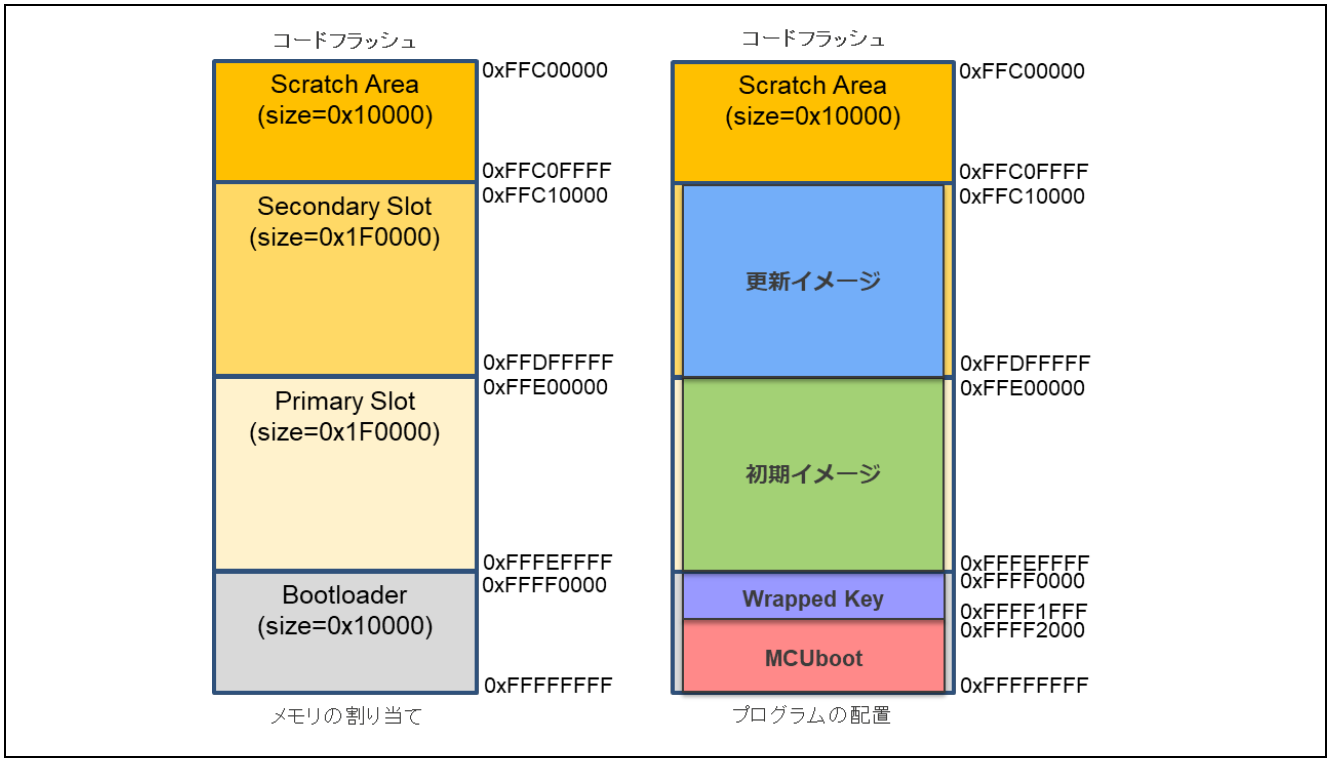

図 5-24 RSK-RX72N リニアモードを使用した方式のデモプロジェクトのメモリマップ

表 5-14 RSK-RX72N リニアモードを使用した方式のコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	0～3 を選択
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	1
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	0xFFFF1000
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_LOG_LEVEL	3

5.2.5.3 デュアルモードを使用した方式の機器接続情報

フラッシュメモリのデュアルモードを使用した DirectXIP 方式の機器接続情報を以下に示します。

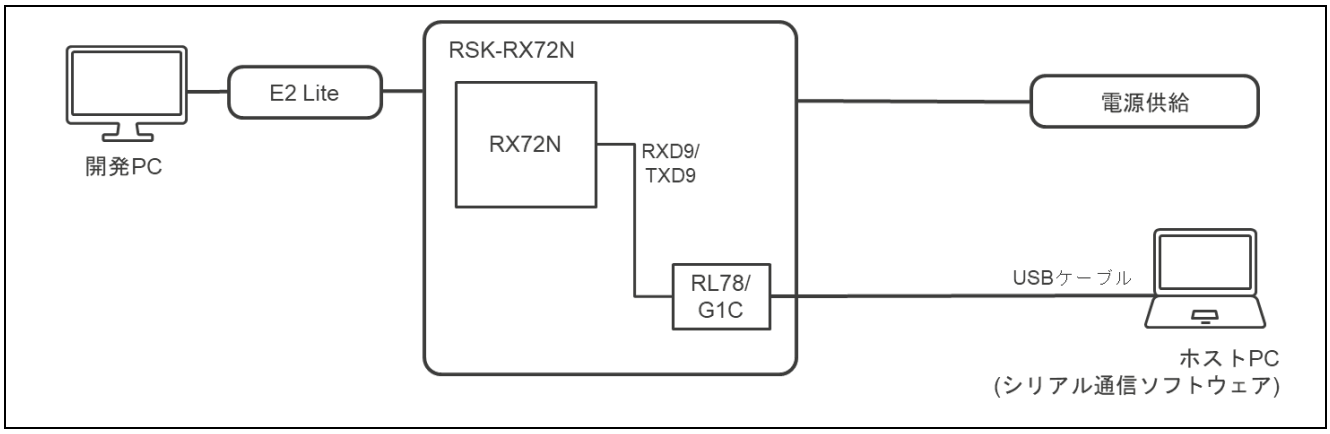

図 5-25 RSK-RX72N デュアルモードを使用した方式の機器接続図

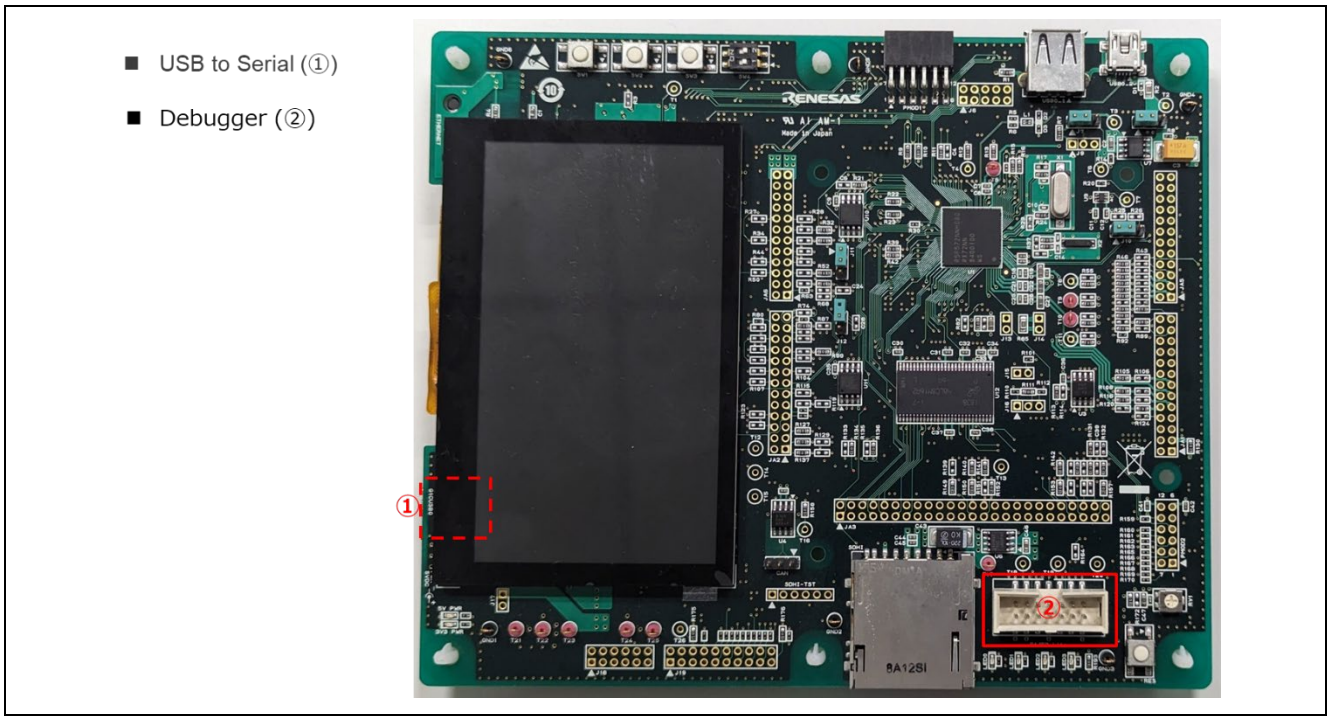

図 5-26 RSK-RX72N デュアルモードを使用した方式の接続情報

5.2.5.4 デュアルモードを使用した方式のメモリ配置およびコンフィグ設定値

フラッシュメモリのデュアルモードを使用した DirectXIP 方式のデモプロジェクトのメモリマップとコンフィグ設定を示します。

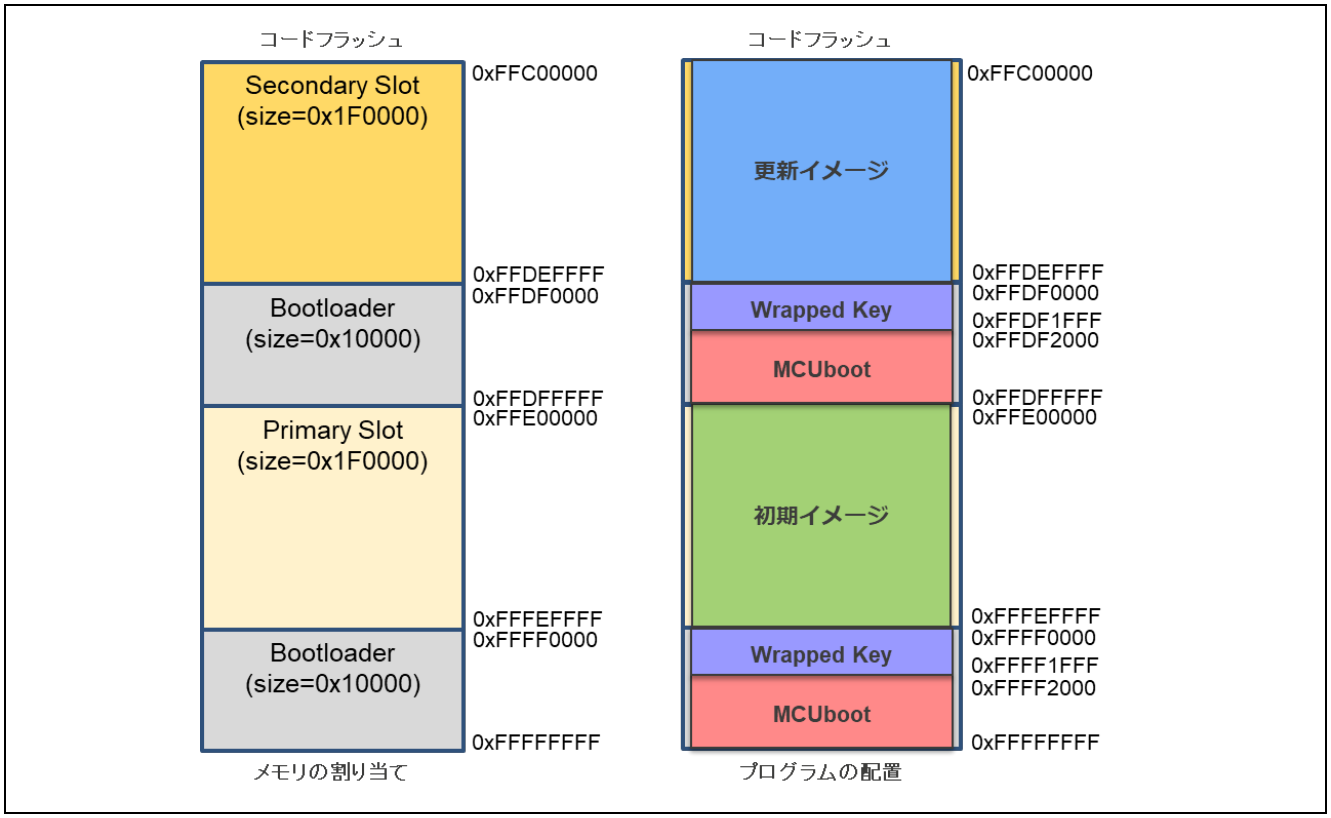

図 5-27 RSK-RX72N デュアルモードを使用した方式のデモプロジェクトのメモリマップ

表 5-15 RSK-RX72N デュアルモードを使用した方式のデモプロジェクトのコンフィグ設定

Configuration options in rm_mcuboot_config.h	
パラメータ名	mcu_boot
RM_MCUBOOT_CFG_UPGRADE_MODE	3（DirectXIP）
RM_MCUBOOT_CFG_VALIDATE_PRIMARY_SLOT	1
RM_MCUBOOT_CFG_DOWNGRADE_PREVENTION	0（本方式では無効な設定）
RM_MCUBOOT_CFG_WATCHDOG_FEED_ENABLED	0
RM_MCUBOOT_CFG_WATCHDOG_FEED_FUNCTION	NULL
RM_MCUBOOT_CFG_SIGN	1
RM_MCUBOOT_CFG_APPLICATION _ENCRYPTION_SCHEME	0（本方式では無効な設定）
RM_MCUBOOT_CFG_ DER_PUB_USER_KEY_ENABLE	1
RM_MCUBOOT_CFG_VERIFY_KEY_ADDRESS	0xFFFF0000
RM_MCUBOOT_CFG_ENCRYPT_KEY_ADDRESS	NULL
RM_MCUBOOT_CFG_MCUBOOT_AREA_SIZE	0x10000
RM_MCUBOOT_CFG_APPLICATION_AREA_SIZE	0x1F0000
RM_MCUBOOT_CFG_SCRATCH_AREA_SIZE	0（本方式では無効な設定）
RM_MCUBOOT_CFG_LOG_LEVEL	3

6. 注意事項

6.1 ブートローダ（MCUboot）からアプリケーションへの遷移時の注意事項

ブートローダ（MCUboot）のサンプルプログラムからアプリケーションへの遷移時には、ブートローダの周辺機能の設定がアプリケーションに引き継がれることになります。

サンプルのブートローダで使用する周辺機能（表 6-1）に関しては、ブートローダ終了時に各 FIT モジュールの API 関数を close した状態にします。また、その他の設定に関してはスマート・コンフィグレータを使用した際の初期値となります。

お客様にて、ブートローダのサンプルプログラムを改造して使用される場合は、ブートローダにて設定した周辺機能の設定がアプリケーション側に引き継がれる事になりますので、ブートローダからアプリケーションに遷移する前に周辺機能の設定を初期化するか、アプリケーション側と周辺機能の設定を共通化されることを推奨します。

アプリケーションを作成される際は、ブートローダの実装を考慮して開発頂きますようお願いします。

表 6-1 ブートローダで使用する周辺機能の注意事項

周辺機能	関連 FIT	ブートローダでの設定および注意事項
ボードに関する機能	r_bsp	スマート・コンフィグレータにて BSP FIT モジュールを組み込んだ際の初期値となります。ブートローダでは設定を変更しておりません。 PMR, PFS レジスタもボードに合わせて設定されますので、ご注意ください。
フラッシュメモリに関する機能	r_flash	フラッシュメモリに関する周辺機能に関しては、Flash FIT の API にて Close 処理を行いアプリケーションに遷移します。
シリアル通信に関する機能	r_sci	シリアル通信に関する周辺機能に関しては、SCI FIT の API にて Close 処理を行いアプリケーションに遷移します。 ブートローダで使用する SCI のチャンネルは 5.2 デモプロジェクトの動作環境の製品毎の機器接続図を参照ください。
オプション設定メモリ	―	オプション設定メモリに関してはブートローダとアプリケーションプログラムで一意の値を設定してください。
その他の機能	―	その他の機能の設定に関してはスマート・コンフィグレータを使用した際の初期値となります。 また、PSW の割り込み許可フラグを 0 の割り込み禁止にし、アプリケーションに遷移します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2025.4.21	-	初版発行
1.01	2025.6.30	1	動作確認デバイス RX66N, RX671, RX72M を追加
		6	表 1-1 の備考を見直し
		11,12	表 1-2 の内容を見直し
		16	表 2-1 の Default 値を見直し
		18-21	2.7 のコンパイラバージョンおよび ROM,RAM,Stack を見直し
		41,42	4.3.3.3 の内容を見直し
		48,49	5.1 のコンパイラおよびモジュールのバージョンを見直し
		56-59	5.2.3 に RX671 の動作確認環境を追加
		60-63	5.2.4 に RX72M の動作確認環境を追加

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。