

RX ファミリ

フラッシュモジュール Firmware Integration Technology

要旨

本アプリケーションノートは Firmware Integration Technology (FIT)を使ったフラッシュモジュールについて説明します。以降、本モジュールをフラッシュ FIT モジュールと称します。

本 FIT モジュールを使って、セルフプログラミングを使ったフラッシュの書き換え機能をユーザアプリケーションに容易に組み込むことができます。セルフプログラミングは、シングルチップモードで実行中に内蔵フラッシュメモリを書き換えるための機能です。本アプリケーションノートでは、フラッシュ FIT モジュールの使用、およびユーザアプリケーションへの取り込みについて説明します。

フラッシュ FIT モジュールは、「RX600 & RX200 シリーズ RX 用シンプルフラッシュ API (R01AN0544)」とは異なります。

対象デバイス

- RX110 グループ
- RX111 グループ
- RX113 グループ
- RX130 グループ
- RX13T グループ
- RX230 グループ、RX231 グループ
- RX23E-A グループ
- RX23T グループ
- RX23W グループ
- RX24T グループ
- RX24U グループ
- RX64M グループ
- RX65N グループ、RX651 グループ
- RX66N グループ
- RX66T グループ
- RX71M グループ
- RX72M グループ
- RX72N グループ
- RX72T 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。

対象コンパイラ

- ・ Renesas Electronics C/C++ Compiler Package for RX Family
- ・ GCC for Renesas RX
- ・ IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認内容については 5.1 動作確認環境を参照してください。

関連アプリケーションノート

- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)

目次

1. 概要	5
1.1 機能説明	5
2. API 情報.....	6
2.1 ハードウェアの要求	6
2.2 ソフトウェアの要求	6
2.3 制限事項	6
2.4 サポートされているツールチェーン	6
2.5 使用する割り込みベクタ	7
2.6 ヘッダファイル	7
2.7 整数型.....	7
2.8 フラッシュのタイプと機能	7
2.9 コンパイル時の設定	8
2.10 コードサイズ	9
2.11 引数	14
2.12 戻り値.....	14
2.13 ブロッキングモード、ノンブロッキングモード.....	15
2.13.1 ブロッキングモードで使用する場合	15
2.13.2 ノンブロッキングモードで使用する場合	15
2.14 Flash FIT モジュールの追加方法	16
2.15 既存のユーザプロジェクトと組み合わせた使用方法	16
2.16 RAM からコードを実行してコードフラッシュを書き換える	17
2.17 コードフラッシュからコードを実行してコードフラッシュを書き換える	17
2.18 デュアルバンクの動作	18
2.19 使用上の注意	19
2.19.1 ノンブロッキングモードでのデータフラッシュの動作.....	19
2.19.2 ノンブロッキングモードでのコードフラッシュの動作.....	19
2.19.3 コードフラッシュの動作と割り込み.....	19
2.19.4 エミュレータのデバッグ設定.....	20
3. API 関数.....	21
3.1 概要	21
3.2 R_FLASH_Open()	22
3.3 R_FLASH_Close().....	23
3.4 R_FLASH_Erase()	24
3.5 R_FLASH_BlankCheck().....	26
3.6 R_FLASH_Write()	28
3.7 R_FLASH_Control()	30
3.8 R_FLASH_GetVersion().....	37
4. デモプロジェクト	38
4.1 flash_demo_rskrx113.....	38
4.2 flash_demo_rskrx231	38
4.3 flash_demo_rskrx23t.....	39
4.4 flash_demo_rskrx130.....	39

4.5	flash_demo_rskrx24t	40
4.6	flash_demo_rskrx65n	40
4.7	flash_demo_rskrx24u	41
4.8	flash_demo_rskrx65n2mb_bank0_bootapp / _bank1_otherapp	41
4.9	flash_demo_rskrx64m	42
4.10	flash_demo_rskrx64m_runrom	42
4.11	flash_demo_rskrx66t	43
4.12	flash_demo_rskrx72t	43
4.13	flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp	44
4.14	ワークスペースにデモを追加する	45
4.15	デモのダウンロード方法	45
5.	付録	46
5.1	動作確認環境	46
5.2	トラブルシューティング	49
5.3	コンパイラ依存の設定	51
5.3.1	Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合	51
5.3.1.1	RAM からコードを実行してコードフラッシュを書き換える場合	52
5.3.1.2	デュアルバンク機能を使用してコードフラッシュを書き換える場合	54
5.3.2	GCC for Renesas RX を使用する場合	57
5.3.2.1	RAM からコードを実行してコードフラッシュを書き換える場合	57
5.3.2.2	デュアルバンク機能を使用してコードフラッシュを書き換える場合	59
5.3.3	IAR C/C++ Compiler for Renesas RX を使用する場合	62
5.3.3.1	RAM からコードを実行してコードフラッシュを書き換える場合	62
5.3.3.2	デュアルバンク機能を使用してコードフラッシュを書き換える場合	67
6.	参考ドキュメント	69
	改訂記録	70

1. 概要

フラッシュ FIT モジュールを使って、内蔵フラッシュ領域のプログラムおよびイレーズを簡単に行えます。本モジュールはコードフラッシュおよびデータフラッシュともにサポートしています。ブロッキング、またはノンブロッキングモードでプログラム、イレーズ、ブランクチェックが行えます。ブロッキングモードで、プログラム関数、イレーズ関数、ブランクチェック関数が呼び出された場合、これらの API 関数は動作が完了するまで復帰しません。ノンブロッキングモードでは、これらの API 関数は、処理を開始した直後に復帰します。コードフラッシュでの動作中に、ユーザアプリケーションによってコードフラッシュ領域にアクセスすることはできません。コードフラッシュ領域にアクセスしようとした場合、シーケンサはエラー状態に遷移します。ノンブロッキングモードでは、コードフラッシュとデータフラッシュのいずれで動作している場合でも、ユーザは動作の完了をポーリングするか、フラッシュ割り込みのコールバック関数を提供する必要があります。

1.1 機能説明

フラッシュ FIT モジュールでサポートしている機能を以下に示します。

- ブロッキング、またはノンブロッキングモードでのコードフラッシュおよびデータフラッシュのイレーズ、プログラム、ブランクチェック
- アクセスウィンドウ、あるいはロックビットによる領域の保護
- スタートアップ領域保護。この機能は、コードフラッシュのブロック 0~7 を安全に書き換えるための機能です。

2. API 情報

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- フラッシュ

2.2 ソフトウェアの要求

本 FIT モジュールは以下のパッケージに依存しています。

- ルネサスボードサポートパッケージ (r_bsp) v.5.00 以降のバージョン

2.3 制限事項

- 本 API のコードは複数の API 関数の呼び出しが同時に発生しないように保護します (R_FLASH_Control())の一部のコマンドは対象外)。
- コードフラッシュの書き換え中は、コードフラッシュにアクセスできません。コードフラッシュを書き換えるときは、アプリケーションコードは RAM から実行するようにしてください。
- RAM の配置に関する制限事項

FIT では、API 関数のポインタ引数に NULL と同じ値を設定すると、パラメータチェックにより戻り値がエラーとなる場合があります。そのため、API 関数に渡すポインタ引数の値は NULL と同じ値にしないでください。

ライブラリ関数の仕様で NULL の値は 0 と定義されています。そのため、API 関数のポインタ引数に渡す変数や関数が RAM の先頭番地(0x0 番地)に配置されていると上記現象が発生します。この場合、セクションの設定変更をするか、API 関数のポインタ引数に渡す変数や関数が 0x0 番地に配置されないように RAM の先頭にダミーの変数を用意してください。

なお、CCRX プロジェクト(e2 studio V7.5.0)の場合、変数が 0x0 番地に配置されることを防ぐために RAM の先頭番地が 0x4 になっています。GCC プロジェクト(e2 studio V7.5.0)、IAR プロジェクト(EWRX V4.12.1)の場合は RAM の先頭番地が 0x0 になっていますので、上記対策が必要となります。

IDE のバージョンアップによりセクションのデフォルト設定が変更されることがあります。最新の IDE を使用される際は、セクション設定をご確認の上、ご対応ください。

2.4 サポートされているツールチェーン

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

2.5 使用する割り込みベクタ

コンフィギュレーションオプション(2.9 参照)の FLASH_CFG_DATA_FLASH_BGO または FLASH_CFG_CODE_FLASH_BGO が 1 のとき、表 2.1 に示す割り込みが有効になります。

表 2.1 使用する割り込みベクター一覧

デバイス	割り込みベクタ
RX110、RX111、RX113、RX130、RX13T RX230、RX231、RX23E-A、RX23T、RX23W、RX24T、RX24U	FRDYI 割り込み (ベクタ番号: 23)
RX64M、RX66T、RX71M、RX72T RX651、RX65N、RX66N、RX72M、RX72N	FIFERR 割り込み (ベクタ番号: 21) FRDYI 割り込み (ベクタ番号: 23)

2.6 ヘッドファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は `r_flash_rx_if.h` に記載されています。このファイルは、Flash FIT を使用するすべてのファイルに含める必要があります。

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

2.7 整数型

コードをわかりやすく、また移植が容易に行えるように、本プロジェクトでは ANSI C99 (Exact width integer types (固定幅の整数型)) 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。

2.8 フラッシュのタイプと機能

フラッシュドライバは使用される技術およびシーケンサによって 3 つのタイプに分かれます。コンパイル後のフラッシュドライバのサイズはフラッシュのタイプが基準となります (2.9 参照)。

フラッシュタイプ 1

RX110*、RX111、RX113、RX130、RX13T

RX230、RX231、RX23E-A、RX23T*、RX23W、RX24T、RX24U

* データフラッシュはありません。

フラッシュタイプ 3

RX64M、RX66T、RX71M、RX72T

フラッシュタイプ 4

RX651*、RX65N*、RX66N、RX72M、RX72N

* コードフラッシュメモリ容量が 1M バイト以下の製品ではデータフラッシュはありません。

MCU によってフラッシュタイプが異なるため、すべての MCU ですべてのフラッシュのコマンドや機能が使用できるわけではありません。`r_flash_rx_if.h` で `#define` (下記参照) を使って、各 MCU で使用可能な機能が定義されます。

2.9 コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、`r_flash_rx_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、下表に示します。

コンフィギュレーションオプション (r_flash_rx_config.h) (1/2)		
定義	デフォルト値	説明
FLASH_CFG_PARAM_CHECKING_ENABLE	1	この定義を“1”に設定するとパラメータチェック処理のコードを生成し、“0”に設定すると生成しません。
FLASH_CFG_CODE_FLASH_ENABLE	0	データフラッシュのみを使用する場合は“0”に設定してください。 “1”に設定するとコードフラッシュ領域を書き換えるためのコードを生成します。コードフラッシュを書き換える際は、RAM からコードを実行する必要があります。 RAM からコードを実行するためのコードおよびリンカの設定については、2.16 を参照してください。 フラッシュタイプ 3、4 では制限付きでコードフラッシュからコードを実行することが出来ます。
FLASH_CFG_DATA_FLASH_BGO	0	この定義を“0”に設定すると、処理が完了するまで、データフラッシュの API 関数はブロックされます。 “1”に設定するとモジュールはノンブロッキングモードになります。ノンブロッキングモードでは、API 関数はデータフラッシュでの動作開始後すぐに復帰します。動作完了の通知はコールバック関数を使って行います。 FLASH_CFG_CODE_FLASH_ENABLE が“1”の場合、FLASH_CFG_CODE_FLASH_BGO と同じ設定にしてください。
FLASH_CFG_CODE_FLASH_BGO	0	この定義を“0”に設定すると、処理が完了するまで、コードフラッシュの API 関数はブロックされます。 “1”に設定するとモジュールはノンブロッキングモードになります。ノンブロッキングモードでは、API 関数はコードフラッシュでの動作開始後すぐに復帰します。動作完了の通知はコールバック関数を使って行います。コードフラッシュの書き換え時、可変ベクタテーブルとそれに対応する割り込み処理を、前もってコードフラッシュ以外の場所に配置する必要があります。2.19 の使用上の注意を参照してください。 FLASH_CFG_CODE_FLASH_ENABLE が“1”の場合、FLASH_CFG_DATA_FLASH_BGO と同じ設定にしてください。
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	0	この定義はフラッシュタイプ 3 およびコードフラッシュメモリ容量が 1.5M バイト以上のフラッシュタイプ 4 の製品に適用されます。 また、FLASH_CFG_CODE_FLASH_ENABLE が 1 に設定されている場合のみ有効です。 RAM のコードを実行しながら、コードフラッシュを書き換える場合は“0”に設定してください。 コードフラッシュの別のセグメントでコードを実行しながら、コードフラッシュを書き換える場合は“1”にしてください (2.17 参照)。

2.10 コードサイズ

本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。フラッシュタイプ毎に代表して 1 デバイスずつ掲載しています。

ツールチェーン（セクション 2.4 に記載）でのコードサイズは、最適化レベル 2、およびコードサイズ重視の最適化を前提としたサイズです。ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、本モジュールのコンフィギュレーションヘッダファイルで設定される、ビルド時のコンフィギュレーションオプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_flash_rx Rev.4.30

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(統合開発環境のデフォルト設定に"-lang = c99"オプションを追加)

GCC for Renesas RX 4.08.04.201902

(統合開発環境のデフォルト設定に"-std = gnu99"オプションを追加)

IAR C/C++ Compiler for Renesas RX version 4.12.1

(統合開発環境のデフォルト設定)

コンフィギュレーションオプション: 差分となるコンフィギュレーションオプションの設定は各表に記載
その他のコンフィギュレーションオプションはデフォルト設定

フラッシュタイプ1:ROM、RAM およびスタックのコードサイズ (最大サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX130	ROM	3492 バイト	3167 バイト	7340 バイト	6776 バイト	5222 バイト	4800 バイト
	RAM	2843 バイト		5692 バイト		4187 バイト	
	スタック	112 バイト		—		100 バイト	
コンフィギュレーションオプション: FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

フラッシュタイプ1:ROM、RAM およびスタックのコードサイズ (最小サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX130	ROM	1803 バイト	1688 バイト	3792 バイト	3624 バイト	2496 バイト	2360 バイト
	RAM	73 バイト		76 バイト		53 バイト	
	スタック	44 バイト		－		48 バイト	

コンフィギュレーションオプション:

FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり

FLASH_CFG_CODE_FLASH_ENABLE 0

FLASH_CFG_DATA_FLASH_BGO 0

FLASH_CFG_CODE_FLASH_BGO 0

フラッシュタイプ1:ROM、RAM およびスタックのコードサイズ (最大サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX23T*	ROM	2810 バイト	2545 バイト	5564 バイト	5068 バイト	4000 バイト	3656 バイト
	RAM	2566 バイト		5160 バイト		3733 バイト	
	スタック	108 バイト		—		100 バイト	
コンフィギュレーションオプション: FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

*データフラッシュを搭載しないデバイス

フラッシュタイプ1:ROM、RAM およびスタックのコードサイズ (最小サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX23T*	ROM	2561 バイト	2312 バイト	5076 バイト	4604 バイト	3544 バイト	3208 バイト
	RAM	2319 バイト		4672 バイト		3271 バイト	
	スタック	48 バイト		—		48 バイト	

コンフィギュレーションオプション:
FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり
FLASH_CFG_CODE_FLASH_ENABLE 1
FLASH_CFG_DATA_FLASH_BGO 0
FLASH_CFG_CODE_FLASH_BGO 0

*データフラッシュを搭載しないデバイス

フラッシュタイプ3:ROM、RAM およびスタックのコードサイズ (最大サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX64M	ROM	3536 バイト	3102 バイト	7228 バイト	6476 バイト	5448 バイト	4920 バイト
	RAM	3091 バイト		6416 バイト		4827 バイト	
	スタック	224 バイト		—		180 バイト	

コンフィギュレーションオプション:

FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり

FLASH_CFG_CODE_FLASH_ENABLE 1

FLASH_CFG_DATA_FLASH_BGO 1

FLASH_CFG_CODE_FLASH_BGO 1

フラッシュタイプ3:ROM、RAM およびスタックのコードサイズ (最小サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX64M	ROM	2110 バイト	1967 バイト	4444 バイト	4192 バイト	3068 バイト	2898 バイト
	RAM	65 バイト		68 バイト		48 バイト	
	スタック	76 バイト		－		56 バイト	

コンフィギュレーションオプション:
FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり
FLASH_CFG_CODE_FLASH_ENABLE 0
FLASH_CFG_DATA_FLASH_BGO 0
FLASH_CFG_CODE_FLASH_BGO 0

フラッシュタイプ4:ROM、RAM およびスタックのコードサイズ (最大サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX65N	ROM	3524 バイト	3077 バイト	7068 バイト	6284 バイト	5284 バイト	4736 バイト
	RAM	3177 バイト		5960 バイト		4542 バイト	
	スタック	208 バイト		—		176 バイト	
コンフィギュレーションオプション: FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

フラッシュタイプ4:ROM、RAM およびスタックのコードサイズ (最小サイズ)

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし	パラメータ チェックあり	パラメータ チェックなし
RX65N	ROM	1941 バイト	1798 バイト	3968 バイト	3752 バイト	2827 バイト	2657 バイト
	RAM	61 バイト		92 バイト		47 バイト	
	スタック	72 バイト		－		52 バイト	
コンフィギュレーションオプション: FLASH_CFG_PARAM_CHECKING_ENABLE 0 : パラメータチェックなし、1 : パラメータチェックあり FLASH_CFG_CODE_FLASH_ENABLE 0 FLASH_CFG_DATA_FLASH_BGO 0 FLASH_CFG_CODE_FLASH_BGO 0							

2.11 引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともに `r_flash_rx_if.h` に記載されています。API 関数については 3 章で説明します。

2.12 戻り値

API 関数の戻り値を示します。この列挙型は、`r_flash_rx_if.h` に記載されています。

```
/* Flash FIT エラーコード */
typedef enum _flash_err
{
    FLASH_SUCCESS = 0,
    FLASH_ERR_BUSY,           /* フラッシュモジュールはビジー状態 */
    FLASH_ERR_ACCESSW,        /* アクセスウィンドウのエラー */
    FLASH_ERR_FAILURE,        /* フラッシュの動作失敗; プログラミングエラー、
                               イレーズエラー、ブランクチェックエラーなど */

    FLASH_ERR_CMD_LOCKED,     /* 周辺機能はコマンドロック状態 */
    FLASH_ERR_LOCKBIT_SET,    /* タイプ 3 - ロックビットに起因するプログラム/イレーズエラー */
    FLASH_ERR_FREQUENCY,      /* 不正な周波数 */
    FLASH_ERR_BYTES,          /* 無効なバイト数 */
    FLASH_ERR_ADDRESS,        /* 無効なアドレス */
    FLASH_ERR_BLOCKS,         /* ブロック数を指定する引数が無効です。 */
    FLASH_ERR_PARAM,          /* 不正な引数 */
    FLASH_ERR_NULL_PTR,       /* 要求された引数がありません。 */
    FLASH_ERR_UNSUPPORTED,    /* このフラッシュタイプではコマンドはサポートされていません。 */
    FLASH_ERR_SECURITY,        /* タイプ 4 - FAW.FSPR による保護に起因するプログラム/イレーズエラー。 */
    FLASH_ERR_TIMEOUT,        /* 時間切れです。 */
    FLASH_ERR_ALREADY_OPEN    /* Close() を呼び出さず、Open() を 2 回呼び出した。 */
} flash_err_t;
```

2.13 ブロッキングモード、ノンブロッキングモード

本モジュールはブロッキングモードとノンブロッキングモードに対応しています。

2.13.1 ブロッキングモードで使用する場合

本モジュールをブロッキングモードで使用する場合、コンフィギュレーションオプションを以下のように設定します。

- FLASH_CFG_DATA_FLASH_BGO: 0
- FLASH_CFG_CODE_FLASH_BGO: 0

ブロッキングモードの場合、本モジュールの API 関数は処理が完了するまで復帰しません。

2.13.2 ノンブロッキングモードで使用する場合

本モジュールをノンブロッキングモードで使用する場合、コンフィギュレーションオプションを以下のように設定します。

- FLASH_CFG_DATA_FLASH_BGO: 1
- FLASH_CFG_CODE_FLASH_BGO: 1

ノンブロッキングモードの場合、本モジュールの API 関数はフラッシュに対する処理を開始した直後に復帰します。ユーザは、フラッシュ領域での処理が完了するまで、その領域にアクセスしないでください。アクセスした場合、シーケンサはエラー状態になり、処理は正常に完了しません。

フラッシュに対する処理の結果はコールバック関数を介して行われます。

コールバック関数は R_FLASH_Open() を実行した後、R_FLASH_Control() の引数に FLASH_CMD_SET_BGO_CALLBACK コマンドを指定して登録しておきます。(詳細は 3.7 章を参照ください。)

フラッシュに対する処理が完了すると FRDYI 割り込みが発生します。FRDYI 割り込み内の処理から登録されたコールバック関数が呼び出されます。コールバック関数には完了ステータスを示すイベントが渡されます。これらのイベントは "r_flash_rx_if.h" で以下のように定義されています。

```
typedef enum _flash_interrupt_event
{
    FLASH_INT_EVENT_INITIALIZED,
    FLASH_INT_EVENT_ERASE_COMPLETE,
    FLASH_INT_EVENT_WRITE_COMPLETE,
    FLASH_INT_EVENT_BLANK,
    FLASH_INT_EVENT_NOT_BLANK,
    FLASH_INT_EVENT_TOGGLE_STARTUPAREA,
    FLASH_INT_EVENT_SET_ACCESSWINDOW,
    FLASH_INT_EVENT_LOCKBIT_WRITTEN,
    FLASH_INT_EVENT_LOCKBIT_PROTECTED,
    FLASH_INT_EVENT_LOCKBIT_NON_PROTECTED,
    FLASH_INT_EVENT_ERR_DF_ACCESS,
    FLASH_INT_EVENT_ERR_CF_ACCESS,
    FLASH_INT_EVENT_ERR_SECURITY,
    FLASH_INT_EVENT_ERR_CMD_LOCKED,
    FLASH_INT_EVENT_ERR_LOCKBIT_SET,
    FLASH_INT_EVENT_ERR_FAILURE,
    FLASH_INT_EVENT_TOGGLE_BANK,
    FLASH_INT_EVENT_END_ENUM
} flash_interrupt_event_t;
```

2.14 Flash FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

- (1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator 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「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。
- (2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。
- (3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator 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「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。
- (4) CS+上で FIT モジュールを追加する場合
CS+上で、手動で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 ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.15 既存のユーザプロジェクトと組み合わせた使用方法

本モジュールは、BSP のスタートアップ無効化機能を使用することによって、既存のユーザプロジェクトと組み合わせて使用することが出来ます。

BSP のスタートアップ無効化機能は、新たにプロジェクトを作成せずに、既存のユーザプロジェクトに本モジュールやその他の各周辺の FIT モジュールを追加して使用するための機能です。

既存のユーザプロジェクトに BSP と本モジュール(必要であればその他の各周辺の FIT モジュール)を組み込みます。BSP を組み込む必要はありますが、BSP で行うすべてのスタートアップ処理が無効となるため、既存のユーザプロジェクトのスタートアップ処理と組み合わせて、本モジュールやその他の各周辺の FIT モジュールを使用することができます。

BSP のスタートアップ無効化機能を使用するためには幾つかの設定と注意点があります。詳細は、アプリケーションノート「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)」を参照ください。

2.16 RAM からコードを実行してコードフラッシュを書き換える

コードフラッシュを書き換えるための API 関数を保持するために、RAM およびコードフラッシュにセクションを作成する必要があります。これは、シーケンサがコードフラッシュのコードを実行中にコードフラッシュのプログラム、あるいはイレーズが行えないためです（タイプ 3 では例外の場合あり）。RAM のセクションは、リセット後に初期設定する必要があります。

コードフラッシュの書き換えを有効にするために、`r_flash_rx_config.h` で `FLASH_CFG_CODE_FLASH_ENABLE` を“1”に設定します。これはコードフラッシュを書き換える場合にのみ適用されます。

本モジュールは Rev.4.00 から複数のコンパイラに対応しています。本モジュールを使用するにあたってはコンパイラ毎に異なる設定が必要となります。詳細は 5.3 章を参照いただき、使用するコンパイラにあった設定を行ってください。

2.17 コードフラッシュからコードを実行してコードフラッシュを書き換える

フラッシュタイプ 3 およびコードフラッシュメモリ容量が 1.5M バイト以上のフラッシュタイプ 4 の製品では、一定の条件を満たせば、コードフラッシュからコードを実行中にコードフラッシュを書き換えることができます。コードフラッシュは複数領域に分けられます。コードを実行している領域から、他の領域に対してプログラム／イレーズを行います。領域のサイズは MCU のコードフラッシュの容量によって異なります。領域の境界については、各ユーザズマニュアル ハードウェア編を参照してください。この機能をサポートするのは、コードフラッシュ領域の大きい MCU のみです。コードフラッシュメモリ容量が 1.5M バイト以上のフラッシュタイプ 4 の製品では、境界はバンクの境界と一致します。

この方法を使用する場合、`r_flash_rx_config.h` で `FLASH_CFG_CODE_FLASH_ENABLE`、`FLASH_CFG_CODE_FLASH_RUN_FROM_ROM` の設定を“1”にしてください。

`FLASH_CFG_CODE_FLASH_BGO`（完了待ち）の設定は“0”でも“1”でも構いませんが、`FLASH_CFG_DATA_FLASH_BGO` と一致する必要があります。

2.16 の方法は使用せず、コードの実行元とコードの実行先が異なるように設定してください。

2.18 デュアルバンクの動作

コードフラッシュメモリ容量が 1.5M バイト以上のフラッシュタイプ 4 の製品はリニアとデュアルバンクの 2 種類のモードで動作できます。デュアルバンクモードはコードフラッシュメモリを 2 領域(バンク)に分割して使用するモードです。デュアルバンクモードでは R_FLASH_Control(FLASH_CMD_BANK_TOGGLE)コマンドを使って、起動バンクを切り替えることができます。この切り替えは、次に MCU がリセットされるまで有効にはなりません。

デュアルバンクモードで使用する際は BSP のコンフィギュレーションファイル(r_bsp_config.h)で定義される定数を以下のように変更する必要があります。

- BSP_CFG_CODE_FLASH_BANK_MODE: 1 → 0
デフォルトは“1”となっています。デュアルバンクモードで使用する際は“0”に設定してください。

また、起動バンクに関する設定は以下の定数を使用します。必要に応じて設定してください。

- BSP_CFG_CODE_FLASH_START_BANK: 0、または 1
デフォルトの起動バンクは“0”となっています。起動バンクを“1”にする際は“1”に設定してください。

本モジュールは Rev.4.00 から複数のコンパイラに対応しています。本モジュールを使用するにあたってはコンパイラ毎に異なる設定が必要となります。詳細は 5.3 章を参照いただき、使用するコンパイラにあった設定を行ってください。

2.19 使用上の注意

2.19.1 ノンブロッキングモードでのデータフラッシュの動作

ノンブロッキングモードでデータフラッシュを書き換える場合、コードフラッシュ、RAM、および外部メモリへのアクセスが可能です。ただし、データフラッシュの動作中は、割り込みによるアクセスも含めて、データフラッシュにアクセスしないでください。

2.19.2 ノンブロッキングモードでのコードフラッシュの動作

ノンブロッキングモードでコードフラッシュを書き換える場合、外部メモリ、および RAM へのアクセスが可能です。フラッシュ FIT モジュールの API 関数がコードフラッシュの動作完了前に復帰するため、API 関数を呼び出すコードは RAM に配置します。また、その他のフラッシュコマンドを発行する前に、処理中の動作の完了を確認する必要があります。コマンドにはコードフラッシュのアクセスウィンドウの設定、ブートブロックの切り替え/スタートアップ領域フラグのトグル、コードフラッシュのイレーズ、コードフラッシュのプログラム、また、他の FIT モジュール(R01AN2191)を使ったユニーク ID の呼び出しが含まれます。

2.19.3 コードフラッシュの動作と割り込み

特定のメモリ領域に対してフラッシュが動作中の場合は、コードフラッシュ、またはデータフラッシュのその領域にはアクセスできません。そのため、フラッシュの動作中に割り込みの発生を許可する場合は、可変ベクタテーブルの配置に注意が必要です。

ベクタテーブルは、デフォルトでコードフラッシュに配置されます。コードフラッシュの動作中に割り込みが発生すると、割り込みの開始アドレスを取得するためにコードフラッシュにアクセスし、エラーが発生します。これに対応するために、ベクタテーブルと、発生し得る割り込みをコードフラッシュ以外の場所に配置する必要があります。また、割り込みテーブルレジスタ (INTB)も変更が必要です。

本 FIT モジュールには、ベクタテーブルおよび割り込み処理を再配置するための関数は含まれていません。ユーザシステムに応じて、ベクタテーブルと割り込みを再配置する適切な方法を検討してください。

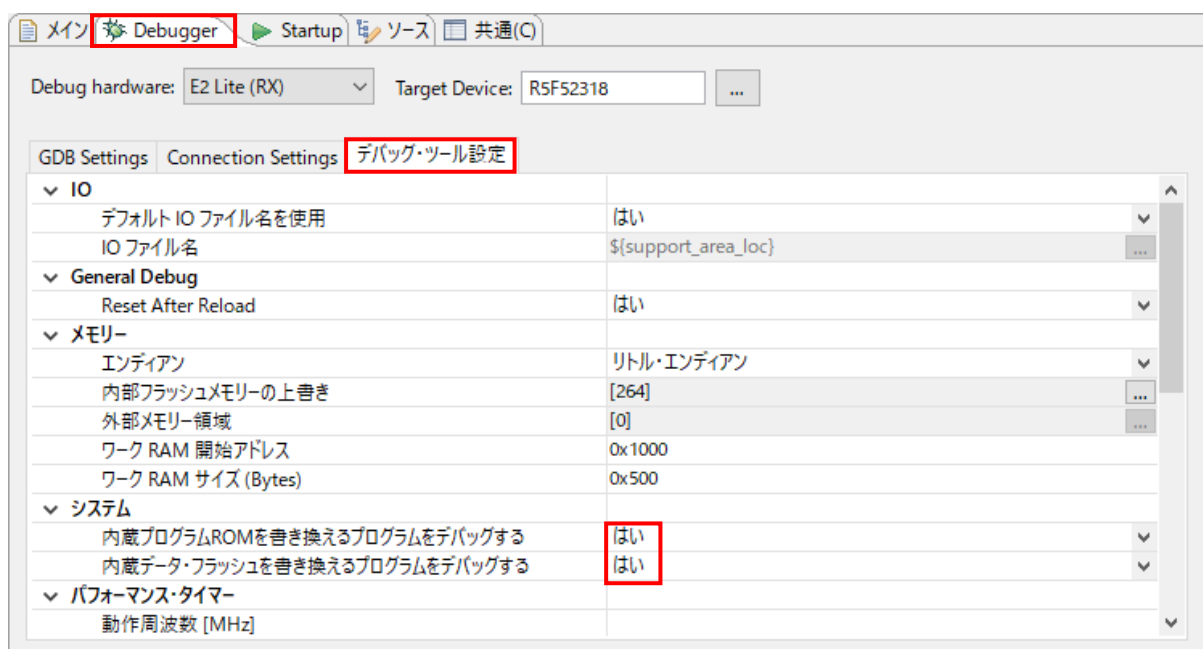
2.19.4 エミュレータのデバッグ設定

デバッグ時にコードフラッシュ、データフラッシュに書き込まれたデータを確認する場合、次のようにデバッグ構成のデバッグ・ツール設定を変更してください。

1. 「プロジェクト・エクスプローラー」においてデバッグ対象のプロジェクトをクリックします。
2. 「実行」→「デバッグの構成...」の順にクリックし、「デバッグ構成」ウィンドウを開きます。
3. 「デバッグ構成」ウィンドウで、“Renesas GDB Hardware Debugging” デバッグ構成の表示を展開し、デバッグ対象のデバッグ構成をクリックしてください。
4. 「Debugger」タブに切り替え、「Debugger」タブの中の「デバッグ・ツール設定」サブタブをクリックし、以下のように設定します。

- システム

- 内蔵プログラム ROM を書き換えるプログラムをデバッグする = “はい”
- 内蔵データ・フラッシュを書き換えるプログラムをデバッグする = “はい”



3. API 関数

3.1 概要

本 FIT モジュールには以下の関数が含まれます。

関数	説明
R_FLASH_Open()	フラッシュ FIT モジュールを初期化します。
R_FLASH_Close()	フラッシュ FIT モジュールを終了します。
R_FLASH_Erase()	コードフラッシュ、またはデータフラッシュの指定ブロックをイ レーズします。
R_FLASH_BlankCheck()	指定したデータフラッシュ、またはコードフラッシュの領域がブラ ンクかどうかを確認します。
R_FLASH_Write()	コードフラッシュ、またはデータフラッシュを書き換えます。
R_FLASH_Control()	状態チェック、および領域保護、スタートアップ領域保護の切り替 えを設定します。
R_FLASH_GetVersion()	本 FIT モジュールのバージョン番号を返します。

3.2 R_FLASH_Open()

フラッシュ FIT モジュールを初期化する関数です。この関数は他の API 関数を使用する前に実行される必要があります。

Format

```
flash_err_t R_FLASH_Open(void);
```

Parameters

なし

Return Values

*FLASH_SUCCESS: /*フラッシュ FIT モジュールが正常に初期化されました。*/*

*FLASH_ERR_BUSY: /*フラッシュに対する別の処理が実行中です。後から再試行してください。*/*

FLASH_ERR_ALREADY_OPEN: / Close() を呼び出さず、Open() を 2 回呼び出した。*/*

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

本関数はフラッシュ FIT モジュールを初期化します。“FLASH_CFG_CODE_FLASH_ENABLE”が“1”の場合、コードフラッシュのプログラム／イレーズに必要な API 関数を RAM にコピーします（ベクタテーブルは含みません）。この関数は他の API 関数を使用する前に実行される必要があります。

Example

```
flash_err_t err;

/* API の初期設定 */
err = R_FLASH_Open();

/* エラーを確認 */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

なし

3.3 R_FLASH_Close()

フラッシュ FIT モジュールを終了する関数です。

Format

```
flash_err_t R_FLASH_Close(void);
```

Parameters

なし

Return Values

FLASH_SUCCESS: */*フラッシュ FIT モジュールを正常に終了しました。*/*

FLASH_ERR_BUSY: */*フラッシュに対する別の処理が実行中です。後から再試行してください。*/*

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

本関数はフラッシュ FIT モジュールを終了します。フラッシュ割り込みが有効な場合はこれを無効化し、ドライバを初期化されていない状態に設定します。

Example

```
flash_err_t err;

/* ドライバの終了 */
err = R_FLASH_Close();

/* エラーを確認 */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

なし

3.4 R_FLASH_Erase()

コードフラッシュまたはデータフラッシュの指定したブロックをイレーズします。

Format

```
flash_err_t R_FLASH_Erase(flash_block_address_t block_start_address,  
                           uint32_t num_blocks);
```

Parameters

block_start_address

イレーズするブロックの開始アドレスを指定します。列挙型“flash_block_address_t”は対応する MCU の“r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h”で定義されます。ブロックは各 MCU の UMH の記載と同様にラベル付けされます。例えば、RX113 の UMH ではアドレス 0xFFFFC000 に配置されているブロックはブロック 7 なので、この引数には“FLASH_CF_BLOCK_7”が渡されます。同様に、アドレス 0x00100000 に配置されているデータフラッシュブロック 0 をイレーズする場合に渡される引数は“FLASH_DF_BLOCK_0”になります。

num_blocks

イレーズ対象のブロック数を指定します。タイプ 1 では、“block_start_address + num_blocks”が、256K の境界を越えないようにしてください。

Return Values

FLASH_SUCCESS	<i>/*正常動作（ノンブロッキングモードが有効な場合、動作が正常に */ /*開始されたことを意味します。） */</i>
FLASH_ERR_BLOCKS	<i>/*指定されたブロック数は無効です。*/</i>
FLASH_ERR_ADDRESS	<i>/*指定されたアドレスは無効です。*/</i>
FLASH_ERR_BUSY	<i>/*フラッシュに対する別の処理が実行中か、*/ /*モジュールが初期化されていません。*/</i>
FLASH_ERR_FAILURE	<i>/*イレーズ失敗。シーケンサがリセットされました。または、*/ /*コールバック関数が登録されていません*/ /*（ノンブロッキングモードの場合）。 */</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

コードフラッシュの隣接するブロック、またはデータフラッシュメモリのブロックをイレーズします。

ブロックサイズは MCU のタイプによって異なります。例えば、RX111 では、コードフラッシュ、データフラッシュともにブロックサイズは 1K バイトです。RX231 および RX23T では、コードフラッシュのブロックサイズは 2K バイト、データフラッシュのブロックサイズは 1K バイトです（RX23T にはデータフラッシュはありません）。これらのサイズを定義するために、コードフラッシュには FLASH_CF_BLOCK_SIZE が、データフラッシュには FLASH_DF_BLOCK_SIZE が提供されます。

列挙型“flash_block_address_t”は、r_bsp モジュールで指定された MCU デバイスのメモリ設定を基に、コンパイル時に設定されます。

API がノンブロッキングモードで使用される場合、指定された番号のブロックがイレーズされた後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。

Example

```
flash_err_t err;

/* データフラッシュブロック 0、1 をイレーズ */
err = R_FLASH_Erase(FLASH_DF_BLOCK_0, 2);

/* エラーの確認 */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

コードフラッシュのブロックをイレーズするためには、イレーズする領域は書き換え可能な領域でなければなりません。フラッシュタイプ 1 は、この指定にアクセスウィンドウを使用します。その他のタイプでは、ロックビットをオフにしてこれに対応してください。

3.5 R_FLASH_BlankCheck()

コードフラッシュ、またはデータフラッシュの指定された領域がブランクかどうかを判定します。

Format

```
flash_err_t R_FLASH_BlankCheck(uint32_t address,  
                                uint32_t num_bytes,  
                                flash_res_t *blank_check_result);
```

Parameters

address

ブランクチェックする領域のアドレス。

num_bytes

この引数でブランクチェックを実施するバイト数を示します。バイト数には、データフラッシュアドレスの場合は FLASH_DF_MIN_PGM_SIZE の倍数を、コードフラッシュアドレスの場合は

FLASH_CF_MIN_PGM_SIZE の倍数を指定します。これらは MCU ごとに

r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h で定義されます。

タイプ 1 では、“address + num_bytes”が、256K の境界を越えないようにしてください。

*blank_check_result

ブロッキングモードの場合、ブランクチェックの結果を格納するメモリのアドレスを指定してください。ノンブロッキングモードの場合、本パラメータは使用しませんので任意の値を指定してください。

Return Values

FLASH_SUCCESS	<i>/*正常動作（ノンブロッキングモードが有効な場合、動作が正常に */ /*開始されたことを意味します。） */</i>
FLASH_ERR_FAILURE	<i>/*ブランクチェック失敗。シーケンサがリセットされました。または、 */ /*コールバック関数が登録されていません */ /*（ノンブロッキングモードの場合）。 */</i>
FLASH_ERR_BUSY	<i>/*フラッシュに対して別の処理が実行中か、 */ /*モジュールが初期化されていません。 */</i>
FLASH_ERR_BYTES	<i>/* “num_bytes”が大きすぎる、最小プログラムサイズの倍数でない、 */ /*最大範囲を超えている、のいずれかのエラーです。 */</i>
FLASH_ERR_ADDRESS	<i>/*無効なアドレスが入力されました。または、アドレスが */ /*最小プログラムサイズで割り切れません。 */</i>
FLASH_ERR_NULL_PTR	<i>/*ブランクチェック結果格納用引数 “blank_check_result” が NULL です。 */</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

プログラム対象のフラッシュ領域はブランクでなければなりません。

ブロッキングモードで動作している場合、ブランクチェックの結果は“blank_check_result”に入ります。この変数は flash_res_t 型で、r_flash_rx_if.h で定義されます。API がノンブロッキングモードで動作している場合、ブランクチェック完了後、ブランクチェックの結果がコールバック関数の引数として渡されます。

Example:

第 2 引数はチェックするバイト数です（FLASH_DF_MIN_PGM_SIZE の倍数でなければなりません）。

```
flash_err_t err;
flash_res_t result;

/* データフラッシュブロック 0 の最初の 64 バイトをブランクチェック */
err = R_FLASH_BlankCheck((uint32_t)FLASH_DF_BLOCK_0, 64, &result);
if (err != FLASH_SUCCESS)
{
    /* エラー処理 */
}
else
{
    /* チェック結果 */
    if (FLASH_RES_NOT_BLANK == result)
    {
        /* ブロックがブランクでない場合の処理 */
        . . .
    }
    else if (FLASH_RES_BLANK == ret)
    {
        /* ブロックがブランクの場合の処理 */
        . . .
    }
}
```

Special Notes:

なし

3.6 R_FLASH_Write()

コードフラッシュ、またはデータフラッシュを書き換えます。

Format

```
flash_err_t R_FLASH_Write(uint32_t    src_address,  
                           uint32_t    dest_address,  
                           uint32_t    num_bytes);
```

Parameters

src_address

フラッシュに書き込むデータを格納したバッファの先頭アドレス。

dest_address

データを書き換えるコードフラッシュ、またはデータフラッシュ領域の先頭アドレス。アドレスには、最小プログラムサイズで割り切れる値を指定します。下記の「Description」に本引数の制限事項を示します。

num_bytes

“src_address”で指定したバッファに含まれるバイト数。この値は、プログラム対象の領域の最小プログラムサイズの倍数となります。

Return Values

FLASH_SUCCESS	<i>/*正常動作（ノンブロッキングモードの場合、動作が正常に */ /*開始されたことを意味します。） */</i>
FLASH_ERR_FAILURE	<i>/*プログラム失敗。書き込み先アドレスが、アクセスウィンドウ、または*/ /*ロックビットで制御されている可能性があります。またはコールバック*/ /*関数が存在しません(ノンブロッキングモードの場合)。*/</i>
FLASH_ERR_BUSY	<i>/*フラッシュに対して別の処理が実行中か、*/ /*モジュールが初期化されていません。*/</i>
FLASH_ERR_BYTES	<i>/* 指定されたバイト数が最小プログラムサイズの倍数でないか、*/ /*最大範囲を超えています。*/</i>
FLASH_ERR_ADDRESS	<i>/*無効なアドレスが入力されました。または、アドレスが */ /*最小プログラムサイズで割り切れません。*/</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

フラッシュメモリを書き換えます。フラッシュ領域に書き込む前に、対象の領域はイレーズしておく必要があります。

書き換えを行う際は、最小プログラムサイズで割り切れるアドレスから開始してください。また、書き込むバイト数は最小プログラムサイズの倍数としてください。最小プログラムサイズは、使用する MCU パッケージ、およびコードフラッシュ、データフラッシュのいずれを書き換えるかによって変わります。

コードフラッシュにデータを書き込む領域は、書き換え可能な領域（アクセスウィンドウ、またはロックビットでアクセス許可設定）でなければなりません。

API がノンブロッキングモードで使用される場合、すべての書き込みが完了すると、コールバック関数が呼び出されます。

Example

```
flash_err_t err;
uint8_t write_buffer[16] = "Hello World...";

/* 内部メモリにデータを書き込む */
err = R_FLASH_Write((uint32_t)write_buffer, dst_addr, sizeof(write_buffer));

/* エラーの確認 */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

FLASH_DF_MIN_PGM_SIZE には、データフラッシュの最小プログラムサイズを定義します。

FLASH_CF_MIN_PGM_SIZE には、コードフラッシュの最小プログラムサイズを定義します。

3.7 R_FLASH_Control()

プログラム、イレーズ、ブランクチェック以外の機能を組み込みます。

Format

```
flash_err_t R_FLASH_Control(flash_cmd_t cmd
                             void *pcfg);
```

Parameters

cmd

実行するコマンド

*pcfg

コマンドに要求される設定用引数。コマンドの要求がない場合は NULL で構いません。

Return Values

FLASH_SUCCESS /*正常動作（ノンブロッキングモードの場合、*/
/*動作が正常に開始されたことを意味します。）*/

FLASH_ERR_ADDRESS /*コード／データフラッシュのブロックの開始アドレスが無効です。*/

FLASH_ERR_NULL_PTR /*設定用構造体を要求するコマンドで使用する引数“pcfg”が NULL です。*/

FLASH_ERR_BUSY /*フラッシュに対して別の処理が実行中か、*/
/*モジュールが初期化されていません。*/

FLASH_ERR_CMD_LOCKED /*フラッシュの制御回路がコマンドロック状態にあり、*/
/*リセットされました。*/

FLASH_ERR_ACCESSW /*アクセスウィンドウエラー: 指定された領域は不正です。*/

FLASH_ERR_PARAM /*無効なコマンド*/

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、プログラム、イレーズ、ブランクチェック以外のシーケンサの機能を組み込むための拡張機能です。コマンドのタイプによって、引数の型も異なります。

コマンド	引数	動作
FLASH_CMD_RESET (フラッシュタイプ 1、3、4 の製品)	NULL	シーケンサをリセットします。処理を中断してリセットするか、処理が完了してからリセットするかは処理内容に依存します。本コマンドはフラッシュが動作中の場合でも使用可能です。
FLASH_CMD_STATUS_GET (フラッシュタイプ 1、3、4 の製品)	NULL	API の状態（Busy または Idle）を返します。本コマンドはフラッシュが動作中の場合でも使用可能です。

コマンド	引数	動作
FLASH_CMD_SET_BGO_CALLBACK (フラッシュタイプ 1、3、4 の製品)	flash_interrupt_config_t *	コールバック関数を登録します。
FLASH_CMD_ACCESSWINDOW_GET (フラッシュタイプ 1、4 の製品)	flash_access_window_config_t *	コードフラッシュのアクセスウィンドウの境界を返します。
FLASH_CMD_ACCESSWINDOW_SET (フラッシュタイプ 1、4 の製品)	flash_access_window_config_t *	コードフラッシュのアクセスウィンドウの境界を設定します。ノンブロッキングモードで使用する場合は、アクセスウィンドウの設定後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。*
FLASH_CMD_SWAPFLAG_GET (フラッシュタイプ 1、4 の製品)	uint32_t *	スタートアップ領域設定モニタフラグ (SASMF (タイプ 1)、BTFLG (タイプ 4)) の現在の値を読み出します。
FLASH_CMD_SWAPFLAG_TOGGLE (フラッシュタイプ 1、4 の製品)	NULL	スタートアッププログラム領域をトグルします。RAM に配置された関数で領域を切り替えます。次のリセットで有効になります。ノンブロッキングモードで使用する場合は、領域の切り替え後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。*
FLASH_CMD_SWAPSTATE_GET (フラッシュタイプ 1、4 の製品)	uint8_t *	スタートアップ領域選択ビットの現在の値(SAS の値)を読み出します。
FLASH_CMD_SWAPSTATE_SET (フラッシュタイプ 1、4 の製品)	uint8_t *	スタートアップ領域選択ビット (FISR.SAS)の値を、r_flash_rx_if.h で定義して設定します。 #define (value) FLASH_SAS_EXTRA (0) FLASH_SAS_DEFAULT (2) FLASH_SAS_ALTERNATE (3) FLASH_SAS_SWITCH_AREA (4) FLASH_SAS_EXTRA、FLASH_SAS_DEFAULT、または FLASH_SAS_ALTERNATE が設定された場合、値は FISR.SAS に直接設定され、その値によって領域が切り替えられます。 FLASH_SAS_SWITCH_AREA が設定された場合、領域が即座に切り替えられます。切り替えは RAM に配置された関数で行います。リセット後の領域は FLASH_SAS_EXTRA で指定された領域となります。
FLASH_CMD_LOCKBIT_READ (フラッシュタイプ 3 の製品)	flash_lockbit_config_t *	指定したブロックのロックビット情報 (FLASH_RES_LOCKBIT_STATE_PROTECTED、または FLASH_RES_LOCKBIT_STATE_NON_PROTECTED) が引数に設定されます。
FLASH_CMD_LOCKBIT_WRITE (フラッシュタイプ 3 の製品)	flash_lockbit_config_t *	指定したブロックアドレスを先頭に、指定したブロック数に対して、ロックビットを設定します。

コマンド	引数	動作
FLASH_CMD_LOCKBIT_ENABLE (フラッシュタイプ 3 の製品)	NULL	ロックビットが設定されたブロックのイレーズ／プログラムを禁止します。
FLASH_CMD_LOCKBIT_DISABLE (フラッシュタイプ 3 の製品)	NULL	ロックビットが設定されたブロックのイレーズ／プログラムを許可します。 注: ブロックをイレーズするとロックビットがクリアされます。
FLASH_CMD_CONFIG_CLOCK (フラッシュタイプ 3、4 の製品)	uint32_t *	FCLK の動作速度 (Hz)。実行時にクロック速度を変更する場合にのみ呼び出しが必要です。
FLASH_CMD_ROM_CACHE_ENABLE (RX24T、RX24U、RX66T、RX72T、フラッシュタイプ 4 の製品)	NULL	コードフラッシュのキャッシュを許可にします。(最初にキャッシュを無効化します。)
FLASH_CMD_ROM_CACHE_DISABLE (RX24T、RX24U、RX66T、RX72T、フラッシュタイプ 4 の製品)	NULL	コードフラッシュのキャッシュを禁止にします。コードフラッシュを書き換える前に呼び出してください。
FLASH_CMD_ROM_CACHE_STATUS (RX24T、RX24U、RX66T、RX72T、フラッシュタイプ 4 の製品)	uint8_t *	キャッシュが許可の場合は引数に“1”が設定されます。キャッシュが禁止の場合は“0”が設定されます。
FLASH_CMD_SET_NON_CACHED_RANGE0 (RX66T、RX66N、RX72T、RX72M、RX72N)	flash_non_cached_t *	コードフラッシュのキャッシュを無効にする領域 0 の範囲を設定します。キャッシュが許可の状態、本コマンドを実行すると、一時的にキャッシュを禁止にします。
FLASH_CMD_SET_NON_CACHED_RANGE1 (RX66T、RX66N、RX72T、RX72M、RX72N)	flash_non_cached_t *	コードフラッシュのキャッシュを無効にする領域 1 の範囲を設定します。キャッシュが許可の状態、本コマンドを実行すると、一時的にキャッシュを禁止にします。
FLASH_CMD_GET_NON_CACHED_RANGE0 (RX66T、RX66N、RX72T、RX72M、RX72N)	flash_non_cached_t *	キャッシュを無効にしている領域 0 の設定を読み出します。
FLASH_CMD_GET_NON_CACHED_RANGE1 (RX66T、RX66N、RX72T、RX72M、RX72N)	flash_non_cached_t *	キャッシュを無効にしている領域 1 の設定を読み出します。
FLASH_CMD_BANK_TOGGLE (コードフラッシュメモリ容量が 1.5M バイト以上のフラッシュタイプ 4 の製品)	NULL	起動バンクを切り替えます。次のリセットで有効になります。ノンブロッキングモードで使用する場合は、バンク選択レジスタ設定後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。*
FLASH_CMD_BANK_GET (コードフラッシュメモリ容量が 1.5M バイト以上のフラッシュタイプ 4 の製品)	flash_bank_t *	現在の BANKSEL 値を読み出します (バンクとアドレスは次のリセットで有効になる)。

**これらのコマンドは、ノンブロッキングモード時でも完了するまでブロックします。これはフラッシュの構成を変える間、必要な処理です。ノンブロッキングモードで完了時もコールバック関数は引き続き呼び出されます。

Example 1:ノンブロッキングモードに設定する

FLASH_CFG_DATA_FLASH_BGO、またはFLASH_CFG_CODE_FLASH_BGO が“1”の場合、ノンブロッキングモードが有効になります。RAM からコードを実行してコードフラッシュの書き換える場合は、可変ベクタテーブルをRAMに再配置します。また、プログラム／イレーズ／ブランクチェックの前にコールバック関数を登録する必要があります。

```
void func(void)
{
    flash_err_t      err;
    flash_interrupt_config_t cb_func_info;
    uint32_t         *pvect_table;

    /* 可変ベクタテーブルを RAM に再配置 */

    /* フラッシュレディ割り込み関数のアドレスを ram_vect_table[23]に直接設定することもできる。ユーザシステムに応じて適切な方法を検討してください。*/
    pvect_table = (uint32_t *)__sectop("C$VECT");
    ram_vect_table[23] = pvect_table[23]; /* FRDYI 割り込み関数コピー */
    set_intb((void *)ram_vect_table);

    /* API の初期設定 */
    err = R_FLASH_Open();
    /* エラーの確認 */
    if (FLASH_SUCCESS != err)
    {
        ... (省略)
    }

    /* コールバック関数と割り込み優先レベルの設定 */
    cb_func_info.pcallback = u_cb_function;
    cb_func_info.int_priority = 1;
    err = R_FLASH_Control(FLASH_CMD_SET_BGO_CALLBACK, (void *)&cb_func_info);
    if (FLASH_SUCCESS != err)
    {
        printf("Control FLASH_CMD_SET_BGO_CALLBACK command failure.");
    }

    /* コードフラッシュ上で動作 */
    do_rom_operations();

    ... (省略)
}

#pragma section FRAM

void u_cb_function(void *event) /* コールバック関数 */
{
    flash_int_cb_args_t *ready_event = event;

    /* ISR コールバック関数の処理 */
}
```

```
void do_rom_operations(void)
{

    /* コードフラッシュアクセスウィンドウの設定、スタートアップ領域フラグのトグル、ブートブロックの切り替え、
    イレーズ、ブランクチェック、またはコードフラッシュのプログラムの処理をここに記載 */

    ... (省略)
}

#pragma section
```

Example 2: API のステータスを確認する

ここではノンブロッキングモードの場合の R_FLASH_Erase()の例を示します。

```
flash_err_t err;

/* 全データフラッシュをイレーズ */
R_FLASH_Erase(FLASH_DF_BLOCK_0, FLASH_NUM_BLOCKS_DF);

/* API のステータスを確認 */
while (R_FLASH_Control(FLASH_CMD_STATUS_GET, NULL) == FLASH_ERR_BUSY)
{
    /* 任意の処理を実施 */
}
```

Example 3: 現在のアクセスウィンドウの範囲を取得する

```
flash_err_t err;
flash_access_window_config_t access_info;

err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_GET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_ACCESSWINDOW_GET command failure.");
}
```

Example 4: コードフラッシュのアクセスウィンドウを設定する（フラッシュタイプ1 および 4）

領域保護は、コードフラッシュのブロックの不正な書き換え、またはイレーズを防ぐために使用されます。以下の例では、ブロック 3 のみに書き換えを許可します（デフォルトではすべてのブロックが書き込み不可の状態です）。

```
flash_err_t err;
flash_access_window_config_t access_info;

/* コードフラッシュブロック 3 への書き込み許可 */

access_info.start_addr = (uint32_t) FLASH_CF_BLOCK_3;
access_info.end_addr = (uint32_t) FLASH_CF_BLOCK_2;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_ACCESSWINDOW_SET command failure.");
}
```

Example 5: 選択中のスタートアップ領域の値を取得する

以下の例では、スタートアップ領域設定モニタフラグ(FSCMR.SASMF)の値の読み込み方法を示します。

```
uint32_t  swap_flag;
flash_err_t err;

err = R_FLASH_Control(FLASH_CMD_SWAPFLAG_GET, (void *)&swap_flag);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPFLAG_GET command failure.");
}
```

Example 6: 選択中のスタートアップ領域を切り替える

以下の例では、スタートアップ領域のトグル方法を示します。RAM に配置された関数を使って領域を切り替えます。

```
flash_err_t err;

/* 2つのアクティブ領域の切り替え */

err = R_FLASH_Control(FLASH_CMD_SWAPFLAG_TOGGLE, FIT_NO_PTR);
if(FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPFLAG_TOGGLE command failure.");
}
```

Example 7: スタートアップ領域選択ビットの値を取得する

以下の例では、スタートアップ領域選択ビット(FISR.SAS)の現在の値を読み込む方法を示します。

```
uint8_t  swap_area;
flash_err_t err;

err = R_FLASH_Control(FLASH_CMD_SWAPSTATE_GET, (void *)&swap_area);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPSTATE_GET command failure.");
}
```

Example 8: スタートアップ領域選択ビットの値を設定する

以下の例では、スタートアップ領域選択ビット(FISR.SAS)の設定方法を示します。RAM に配置された関数を使って領域を切り替えます。リセット後は“FLASH_SAS_EXTRA”で指定された領域が使用されます。

```
uint8_t  swap_area;
flash_err_t err;

swap_area = FLASH_SAS_SWITCH_AREA;
err = R_FLASH_Control(FLASH_CMD_SWAPSTATE_SET, (void *)&swap_area);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPSTATE_SET command failure.");
}
```

Example 9: コードフラッシュの書き換え時にキャッシュの設定をする

以下の例では、コードフラッシュの書き換え時にキャッシュコマンドを使用する方法を示します。

```
uint8_t status;

/* アプリケーションの開始に向けてキャッシュを許可にする */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);

/* メインコードをここに記載（任意でキャッシュが許可かどうかを検証） */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_STATUS, &status);
if (status != 1)
{
    // エラー処理
}

/* コードフラッシュ書き換えの準備（キャッシュを禁止にする） */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_DISABLE, NULL);

/* イレーズ、プログラム、バリファイなどのコードをここに記載 */

/* キャッシュを再度許可にする */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);
```

Example 10: コードフラッシュの特定範囲のキャッシュを無効にする

以下の例では、コードフラッシュの特定範囲のキャッシュを無効にする方法を示します。キャッシュを無効にできる範囲は2つまで設定でき、その2つの範囲は重複していても問題ありません。

```
flash_non_cached_t range;
uint8_t status;

/* FLASH_CF_BLOCK_10 の先頭から 1K バイトの範囲で、命令キャッシュ (IF) とデータキャッシュ (OA)
 * のキャッシュを無効に設定する。 */
range.start_addr = (uint32_t)FLASH_CF_BLOCK_10;
range.size = FLASH_NON_CACHED_1_KBYTE;
range.type_mask = FLASH_NON_CACHED_MASK_IF | FLASH_NON_CACHED_MASK_OA;

R_FLASH_Control(FLASH_CMD_SET_NON_CACHED_RANGE0, &range);

/* キャッシュを許可にする *
 * キャッシュが許可の状態では、本コマンドは実行不要です。 */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);

/* RANGE0 のキャッシュの設定を取得 */
R_FLASH_Control(FLASH_CMD_GET_NON_CACHED_RANGE0, &range);
```

Special Notes:

なし

3.8 R_FLASH_GetVersion()

この関数は本 FIT モジュールのバージョン番号を返します。

Format

```
uint32_t R_FLASH_GetVersion(void);
```

Parameters

なし

Return Values

バージョン番号

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

この関数は本 FIT モジュールのバージョン番号を返します。バージョン番号は符号化され、最上位の 2 バイトがメジャーバージョン番号を、最下位の 2 バイトがマイナーバージョン番号を示しています。バージョンが 4.25 の場合は、“0x00040019”が返されます。

Example

```
uint32_t cur_version;

/* インストールされているフラッシュ FIT のバージョンを取得 */
cur_version = R_FLASH_GetVersion();

/* 本アプリケーションを使用するのに有効なバージョンかどうかを確認 */
if (MIN_VERSION > cur_version)
{
    /* 警告: 本フラッシュ FIT のバージョンでは以降のバージョンでサポートされている xxx 機能を
    サポートしていません。本アプリケーションでは xxx 機能を使用します。*/
    ...
}
```

Special Notes:

なし

4. デモプロジェクト

デモプロジェクトはスタンドアロンプログラムです。デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュール（例：r_bsp）を使用する main()関数が含まれます。デモプロジェクトの標準的な命名規則は、<module>_demo_<board>となり、<module>は周辺の略語(例: s12ad、cmt、sci) 、<board>は標準 RSK（例: rskrx113）です。例えば、RSKRX113 用の s12ad FIT モジュールのデモプロジェクトは s12ad_demo_rskrx113 となります。同様にエクスポートされた.zip ファイルは <module>_demo_<board>.zip となります。例えば、zip 形式のエクスポート/インポートされたファイルは s12ad_demo_rskrx113.zip となります。

また、デモプロジェクトは Renesas Electronics C/C++ Compiler Package for RX Family 以外のコンパイラには対応していません。

4.1 flash_demo_rskrx113

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main()で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX113

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.2 flash_demo_rskrx231

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main()で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX231

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.3 flash_demo_rskrx23t

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンクの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX23T

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.4 flash_demo_rskrx130

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによって、フラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンクの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX130

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.5 flash_demo_rskrx24t

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX24T

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.6 flash_demo_rskrx65n

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX65N

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.7 flash_demo_rskrx24u

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、およびプログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX24U

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.8 flash_demo_rskrx65n2mb_bank0_bootapp / _bank1_otherapp

本デモは下記の対応ボードを使ったデュアルバンク動作のシンプルなデモです。デモでは、ブロッキングモードによって、次のリセットで BANKSEL レジスタの値によって、バンクは切り替えられます。バンク 0 のアプリケーション実行時には LED0 を、バンク 1 のアプリケーション実行時には LED1 をそれぞれ点滅させます。

設定と実行:

1. flash_demo_rskrx65n2mb_bank0_bootapp をビルドして、flash_demo_rskrx65n2mb_bank1_otherapp をビルドします。
2. (HardwareDebug) flash_demo_rskrx65n2mb_bank0_bootapp をダウンロードします（デバッグ設定で他方のアプリケーションもダウンロードされます）。
3. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。
4. LED0 が点滅していることを確認してください。ボードのリセットスイッチを押します。LED1 が点滅していることを確認してください（バンクが切り替えられ、他方のアプリケーションが実行される）。必要に応じて、リセット処理を継続してください。

対応ボード

- RSKRX65N-2MB

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.9 flash_demo_rskrx64m

本デモは下記の対応ボードを使ったシンプルなデモです。デモでは、ブロッキングモードによって、フラッシュのイレーズ、およびプログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX64M

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.10 flash_demo_rskrx64m_runrom

本デモは下記の対応ボードを使ったシンプルなデモです。他のデモとの違いは、RX64M 機能を活用し、片方のコードフラッシュ領域から別の領域へのイレーズ/プログラム中にアプリケーションを実行できる点にあります（他のほとんどの MCU は、コードフラッシュのイレーズ/プログラム中に実行可能なコードを RAM に配置する必要があります）。デモでは、ブロッキングモードによってフラッシュのイレーズ、およびプログラムを行います。プログラム関数の動作は、データのリードバックにより検証します。このデモでは、コードフラッシュのイレーズ/プログラムのサポート用に設定された一般的なリンカ（RAM 配置）は不要で、FLASH_CFG_CODE_FLASH_RUN_FROM_ROM が“r_flash_rx_config.h”で 1 に設定されることに注意してください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX64M

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.11 flash_demo_rskrx66t

本デモは下記の対応ボードを使用したシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、およびプログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「シンボルファイル」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX66T

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.12 flash_demo_rskrx72t

本デモは下記の対応ボードを使用したシンプルなデモです。デモでは、ブロッキングモードによってフラッシュのイレーズ、およびプログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「シンボルファイル」を参照）を事前にご確認ください。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。

対応ボード

- RSKRX72T

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.13 flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp

本デモは下記の対応ボードを使ったデュアルバンク動作のシンプルなデモです。デモでは、ブロッキングモードによって、次のリセットで BANKSEL レジスタの値によって、バンクは切り替えられます。バンク 0 のアプリケーション実行時には LED0 を、バンク 1 のアプリケーション実行時には LED1 をそれぞれ点滅させます。

設定と実行:

1. flash_demo_rskrx72m_bank0_bootapp をビルドして、flash_demo_rskrx72m_bank1_otherapp をビルドします。
2. (HardwareDebug) flash_demo_rskrx72m_bank0_bootapp をダウンロードします（デバッグ設定で他方のアプリケーションもダウンロードされます）。
3. ソフトウェアを実行します。プログラムが main() で停止した場合、F8 を押して再開します。
4. LED0 が点滅していることを確認してください。ボードのリセットスイッチを押します。LED1 が点滅していることを確認してください（バンクが切り替えられ、他方のアプリケーションが実行される）。必要に応じて、リセット処理を継続してください。

対応ボード

- RSKRX72M

評価環境

- 使用バージョン : BSP Rev.5.30、FLASH FIT Rev.4.30

4.14 ワークスペースにデモを追加する

デモプロジェクトは、本アプリケーションノートで提供されるファイルの FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」→「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し、「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「完了」をクリックします。

4.15 デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

5. 付録

5.1 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 5.1 動作確認環境 (Rev.4.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e ² studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.12.1
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 V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99 GCC for Renesas RX 4.08.04.201902 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 IAR C/C++ Compiler for Renesas RX version 4.12.1 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.00
使用ボード	Renesas Starter Kit for RX113 (型名：R0K505113xxxxxx) Renesas Starter Kit for RX130 (型名：RTK5005130xxxxxxxx) Renesas Starter Kit for RX231 (型名：R0K505231xxxxxx) Renesas Starter Kit for RX23T (型名：RTK500523Txxxxxxxx) Renesas Starter Kit for RX24T (型名：RTK500524Txxxxxxxx) Renesas Starter Kit for RX24U (型名：RTK500524Uxxxxxxxx) Renesas Starter Kit+ for RX64M (型名：R0K50564Mxxxxxxxx) Renesas Starter Kit for RX66T (型名：RTK50566Txxxxxxxx) Renesas Starter Kit for RX72T (型名：RTK5572Txxxxxxxx) Renesas Starter Kit+ for RX65N (型名：RTK500565Nxxxxxxxx) Renesas Starter Kit+ for RX65N-2MB (型名：RTK50565Nxxxxxxxx)

表 5.2 動作確認環境 (Rev.4.10)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e ² studio V7.3.0
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 V3.01.00 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.10
使用ボード	Renesas Solution Starter Kit for RX23W (型名：RTK5523Wxxxxxxxx)

表 5.3 動作確認環境 (Rev.4.20)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e ² studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.12.1
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 V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99 GCC for Renesas RX 4.08.04.201902 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 IAR C/C++ Compiler for Renesas RX version 4.12.1 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.20
使用ボード	Renesas Starter Kit+ for RX72M (型名：RTK5572Mxxxxxxxxxx)

表 5.4 動作確認環境 (Rev.4.30)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e ² studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.1
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 V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99 GCC for Renesas RX 4.08.04.201902 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 IAR C/C++ Compiler for Renesas RX version 4.12.1 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.30
使用ボード	RX13T CPU カード (型名：RTK0EMXA10xxxxxxxxxx)

表 5.5 動作確認環境 (Rev.4.40)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e ² studio V7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
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 V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99 GCC for Renesas RX 4.08.04.201902 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 IAR C/C++ Compiler for Renesas RX version 4.12.1 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.40
使用ボード	Renesas Solution Starter Kit for RX23E-A (型名：RTK0ESXB10xxxxxxxxxx)

表 5.6 動作確認環境 (Rev.4.50)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e ² studio V7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
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 V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.08.04.201902 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
	IAR C/C++ Compiler for Renesas RX version 4.12.1 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.50
使用ボード	Renesas Starter Kit+ for RX72N（型名：RTK5572Nxxxxxxxxxx）

5.2 トラブルシューティング

- (1) Q : 本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A : FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

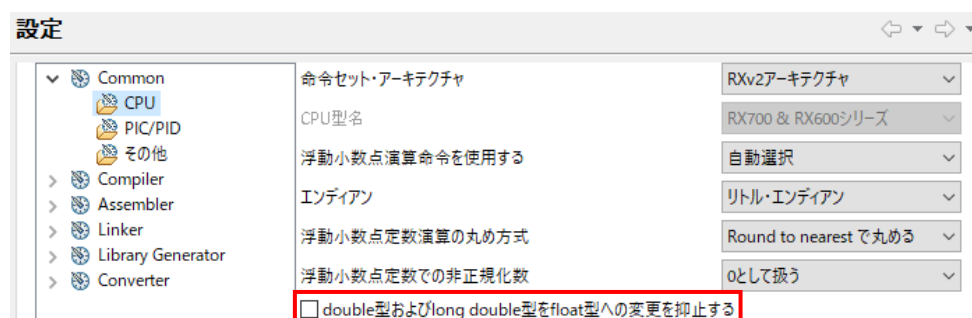
- CS+を使用している場合
アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e² studio を使用している場合
アプリケーションノート RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

- (2) Q : 本 FIT モジュールをプロジェクトに追加し、コンパイラオプションを変更して実行すると ROM アクセス違反が発生します。

A : 本 FIT モジュールを用いて、RAM からコードを実行してコードフラッシュを書き換える場合に使用されるコードは全て RAM に展開されている必要があります。
コンパイラオプションの設定によっては ROM、RAM のどちらが展開先になるかわ変わってくる可能性があります。
コンパイラオプションを変更する必要がある場合はコンパイラオプションを変更することによって、コードが ROM 上に展開されないことをリストファイルに出力する等して確認してください。
以下に、コンパイラオプションの変更が起因して、ROM アクセス違反が発生する例を示します。

A-1 : デフォルトのコンパイラオプション設定



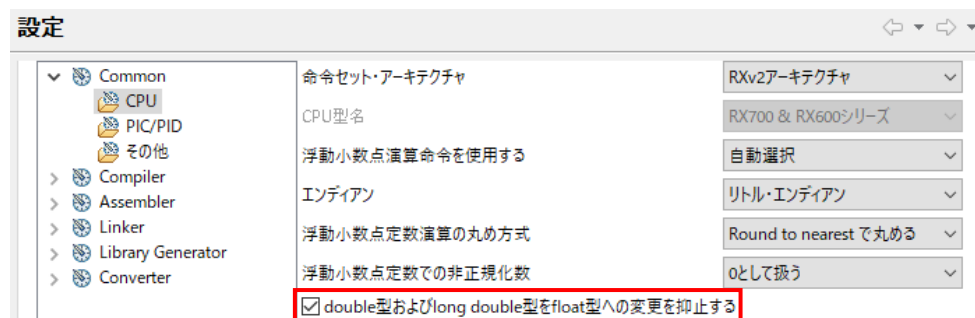
デフォルトのコンパイラオプション設定時のリストファイルの出力結果

```

000003AC FC472E      ^ ^      ITOF R2, R14J
000003AF FD723E005CC149 ^ ^      FMUL #49C15C00H, R14J
000003B6          ^ ^      L127: ^ ; if_break_bb33J
000003B6 92C5      ^ ^      MOV.W R5, 14H[R4]J
000003B8 FCA7E1      ^ ^      FTOU R14, R1J
000003B8 A1C1      ^ ^      MOV.L R1, 18H[R4]J
000003BD 6601      ^ ^      MOV.L #00000000H, R1J

```

A-2 : コンパイラオプション変更



コンパイラオプション変更時のリストファイルの出力結果

```

000003C1 EF21          ^ ^      MOV.L R2, R1J
000003C3 05rrrrrr      A ^ ^      BSR COM_CONV32udJ
000003C7 6603          ^ ^      MOV.L #00000000H, R3J
000003C9 FB42802B3841 ^ ^      MOV.L #41382B80H, R4J
000003CF 05rrrrrr      A ^ ^      BSR COM_MULdJ
000003D3 754740          ^ ^      MOV.L #00000040H, R7J
000003D6          L127: ^ ^      ; if break bb33J
000003D8 05rrrrrr      A ^ ^      BSR COM_CONVd32uJ
000003DA A1E1          ^ ^      MOV.L R1, 18H[R6]J
000003DC 6601          ^ ^      MOV.L #00000000H, R1J
000003DE 92E7          ^ ^      MOV.W R7, 14H[R6]J

```

A-1 はデフォルトのコンパイラオプション設定時のリストファイルの出力結果、
A-3 はコンパイラオプション変更時のリストファイルの出力結果を示しています。

A-1 と A-2 のコンパイラオプションの違いで、リストファイルの出力結果に差分があることがわかります。

A-2 のリストファイルに示す赤枠部分はランタイムライブラリ関数に置き換えられていることを示しています。

このランタイムライブラリ関数はデフォルトでは"P"セクションに配置されるため、RAM には展開されません。このため、実行時に ROM アクセス違反が発生します。

5.3 コンパイラ依存の設定

本モジュールは Rev.4.00 から複数のコンパイラに対応しています。本モジュールを使用するにあたり、コンパイラ毎に異なる設定を以下に示します。

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

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

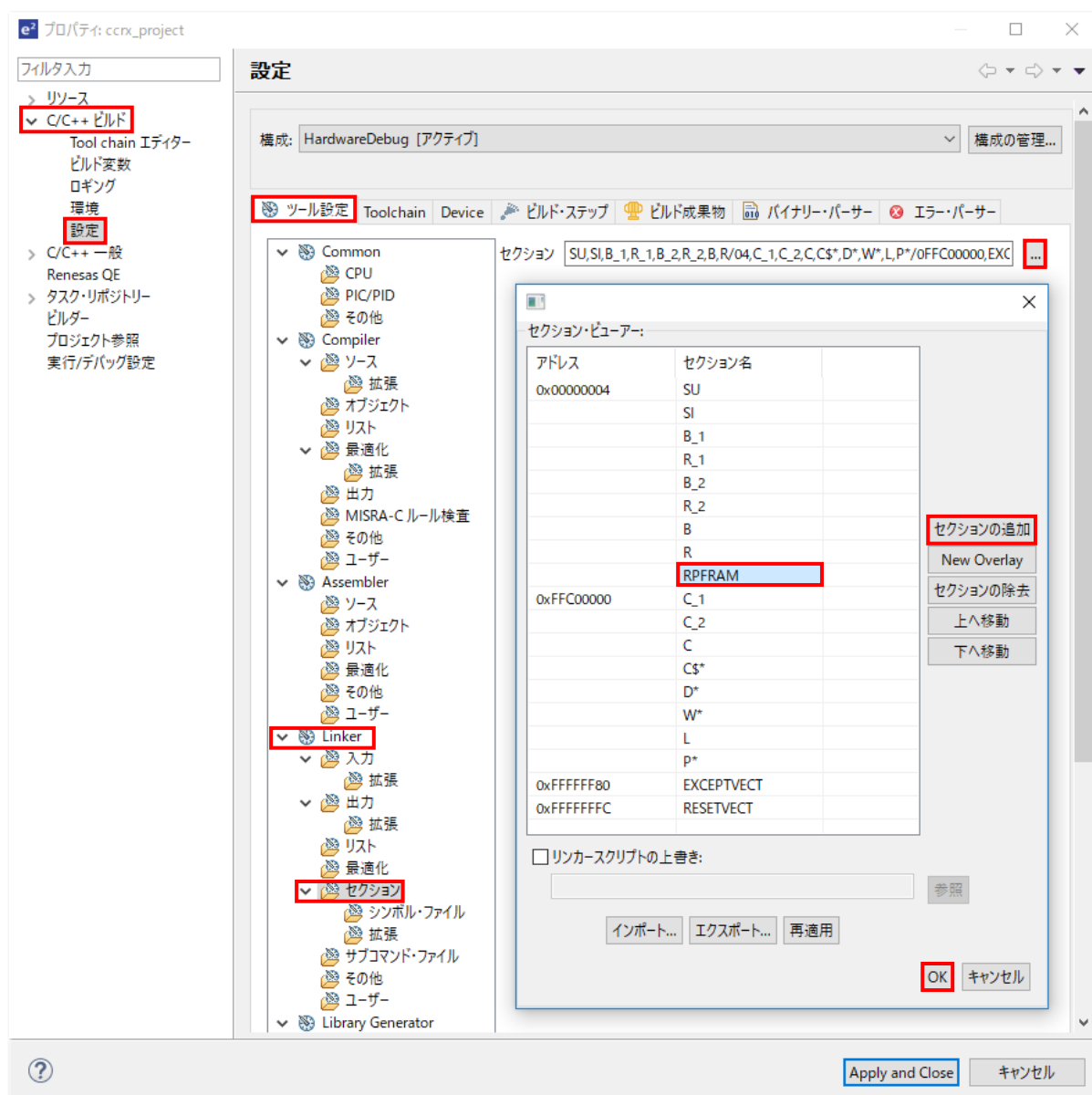
リンカのセクションの設定とコードフラッシュから RAM へのマッピングは e² studio で行う必要があります。

5.3.1.1 RAM からコードを実行してコードフラッシュを書き換える場合

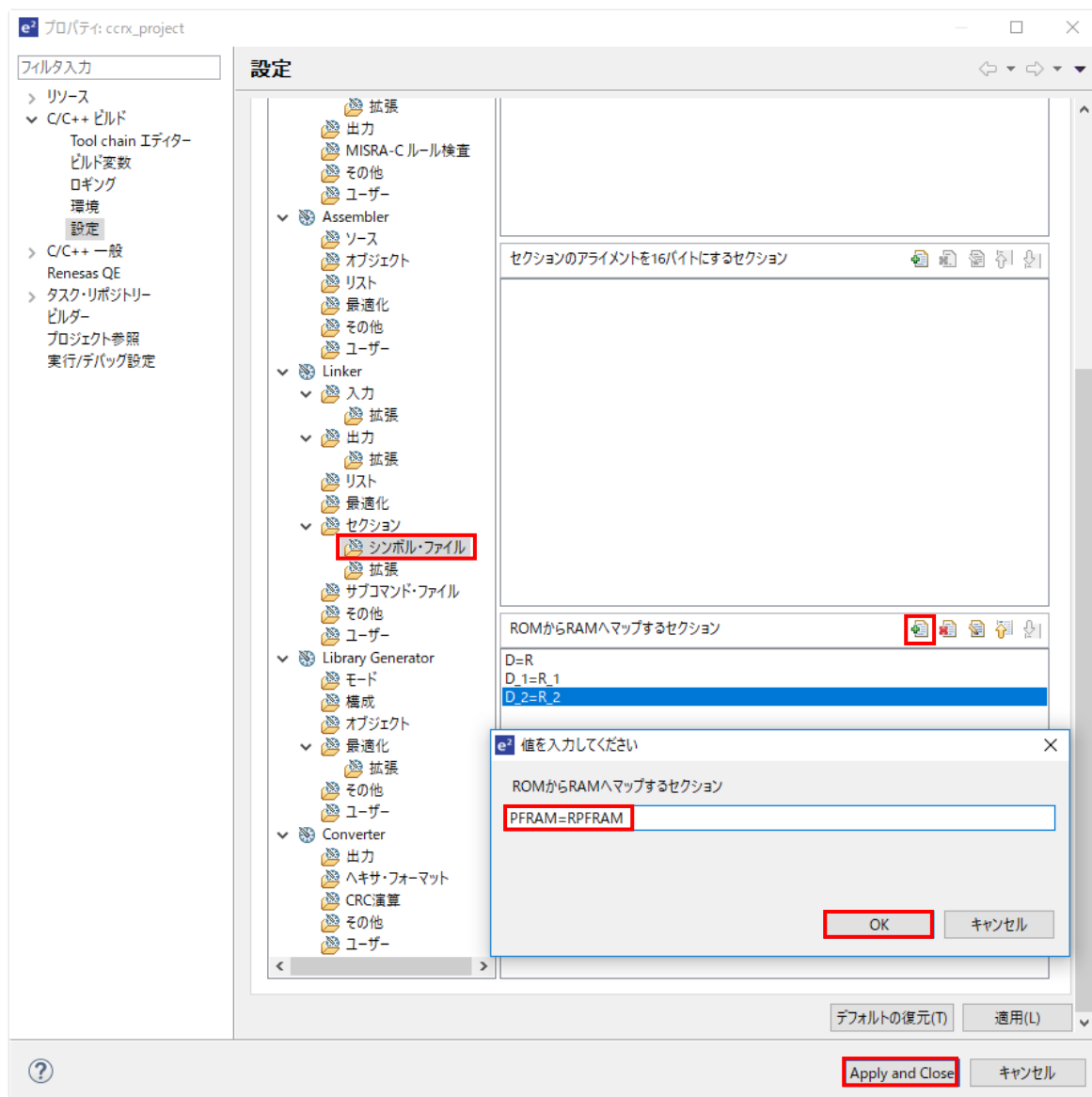
本章では、セクションの追加、コードフラッシュから RAM へのマッピング、コードフラッシュ書き換え中に動作するプログラムの配置について示します。

1. RAM 領域に"RPFRAM"セクションを追加します。

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



2. コードフラッシュのセクション(PFRAM)のアドレスを RAM のセクション(RPFRAM)のアドレスにマッピングします
 - (1) 「シンボル・ファイル」をクリックした後、ROM から RAM へマップするセクションの「追加...」アイコンを押下します。
 - (2) 「値を入力してください」ウィンドウで、"PFRAM=RPFRAM"を入力した後、「OK」ボタンを押下します。
 - (3) 「Apply and Close」ボタンを押下します。



3. 割り込みコールバック関数等コードフラッシュ書き換え中に動作するプログラムは FRAM セクション内に配置する必要があります。

```
#pragma section FRAM
/* コードフラッシュ書き換え中に動作する関数 */
void func(void){...}

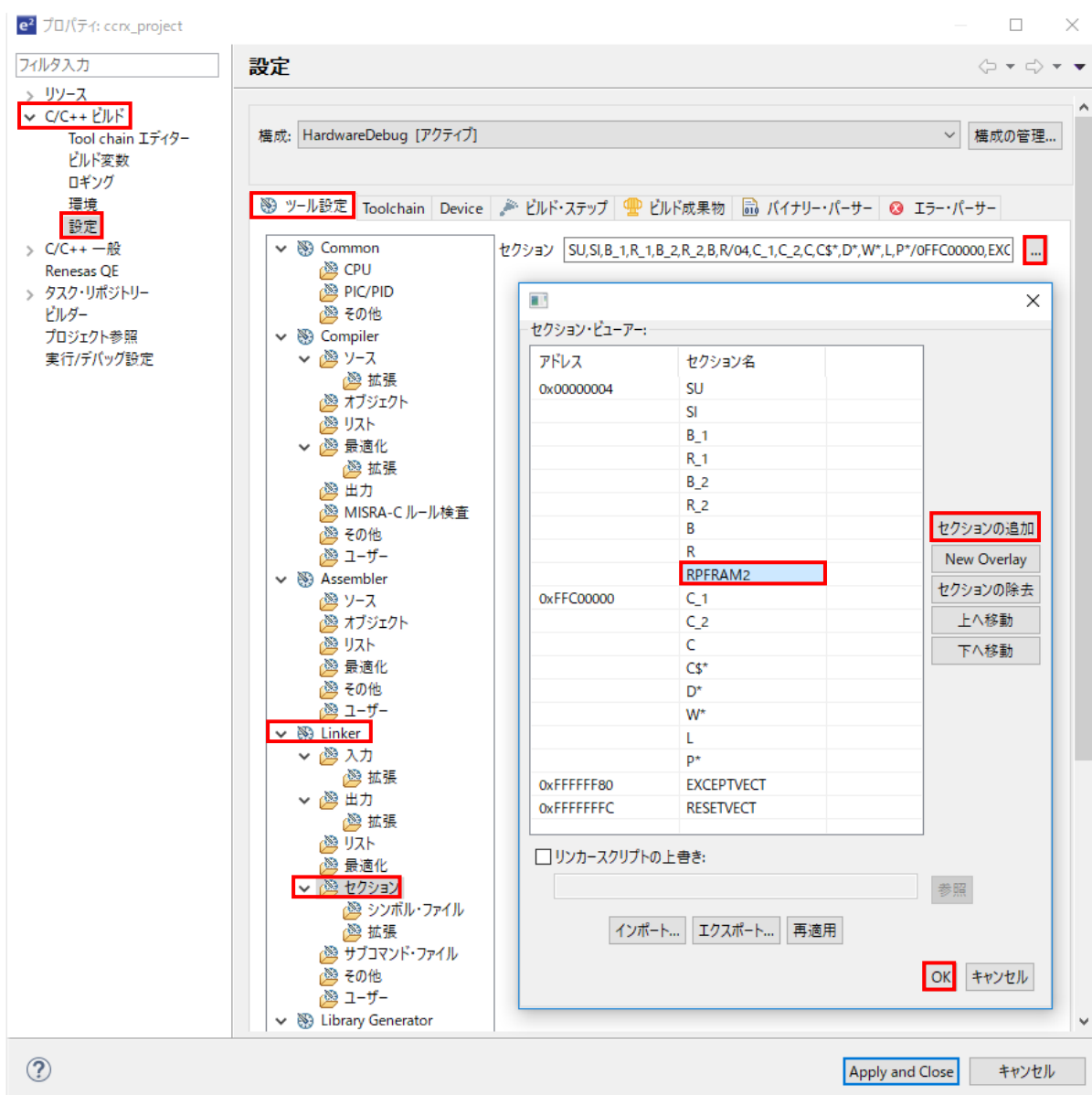
/* コードフラッシュ書き換え中に動作するコールバック関数 */
void cb_func(void){...}
#pragma section
```

5.3.1.2 デュアルバンク機能を使用してコードフラッシュを書き換える場合

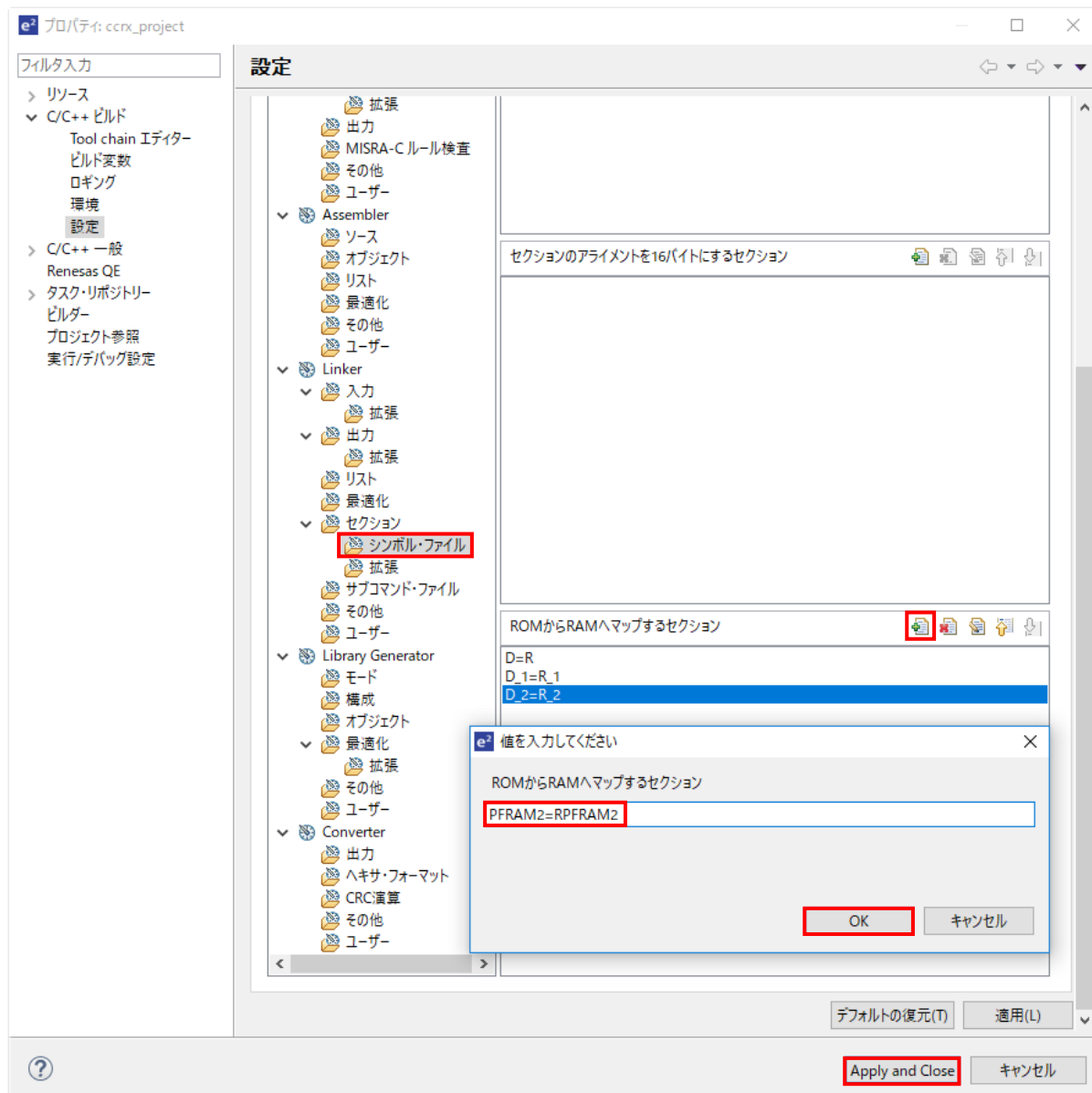
本章では、セクションの追加、コードフラッシュから RAM へのマッピング、デュアルバンク機能のデバッグについて示します。

1. RAM 領域に"RPFRAM2"セクションを追加します。

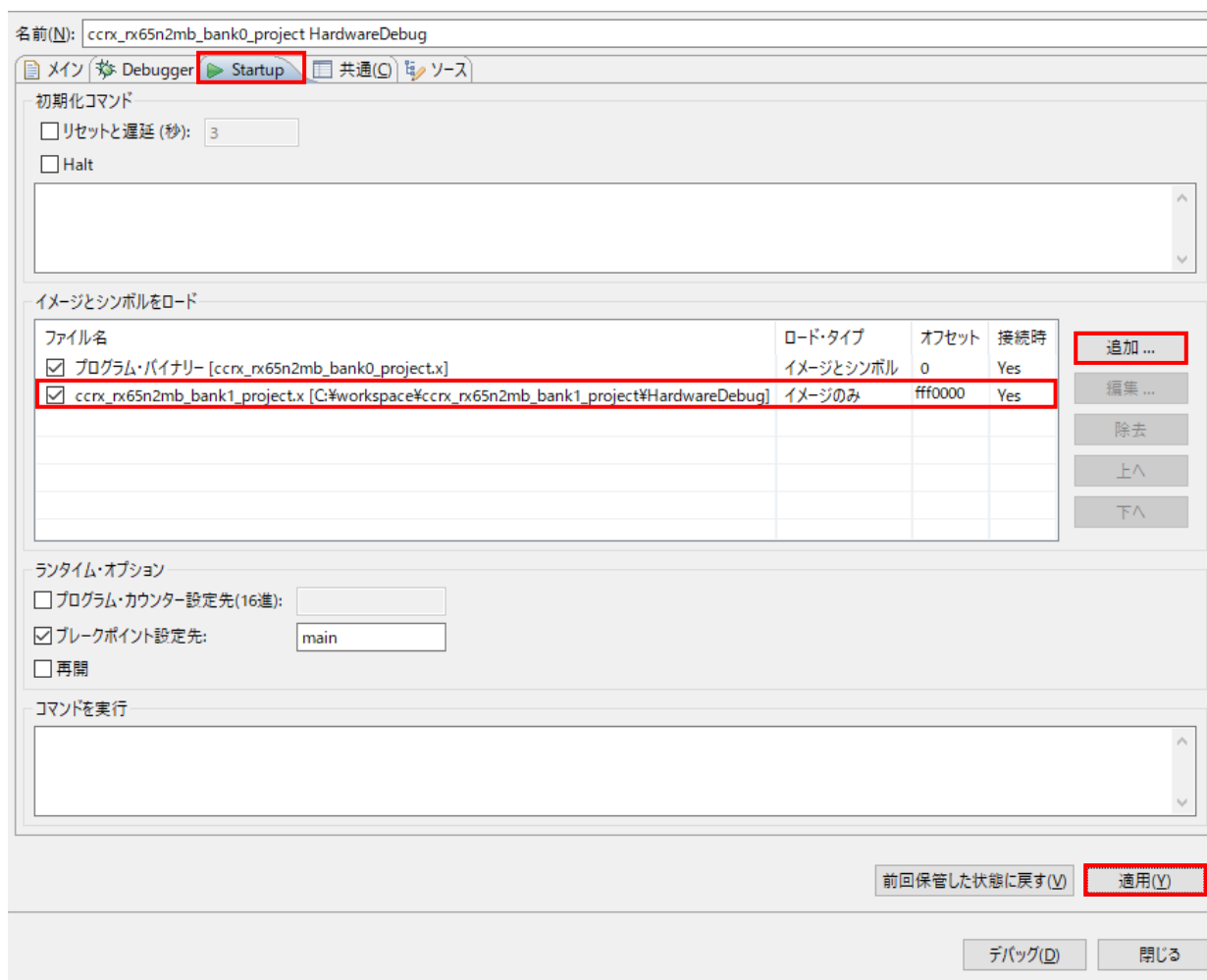
- (1) 「プロジェクト・エクスプローラー」においてデバッグ対象のプロジェクトをクリックします。
- (2) 「ファイル」→「プロパティ」の順にクリックし、「プロパティ」ウィンドウを開きます。
- (3) 「プロパティ」ウィンドウで、「C/C++ビルド」→「設定」の順にクリックします。
- (4) 「ツール設定」タブを押下し、「Linker」→「セクション」の順にクリックして、「...」ボタンを押下し、「セクション・ビューアー」ウィンドウを開きます。
- (5) 「セクション・ビューアー」ウィンドウで、「セクションの追加」ボタンを押下して、RAM 領域に"RPFRAM2"セクションを追加し、「OK」ボタンを押下します。



2. コードフラッシュのセクション(PFRAM2)のアドレスを RAM のセクション(RPFRAM2)のアドレスにマッピングします。
 - (1) 「シンボル・ファイル」をクリックした後、ROM から RAM へマップするセクションの「追加...」アイコンを押下します。
 - (2) 「値を入力してください」ウィンドウで、"PFRAM2=RPFRAM2"を入力した後、「OK」ボタンを押下します。
 - (3) 「Apply and Close」ボタンを押下します。



3. デュアルバンク機能に関連するアプリケーションをターゲットデバイスに接続してデバッグする際、2つのバンクに対するオブジェクトをロードする方法を以下に示します。必要に応じて実施してください。
 - (1) 「プロジェクト・エクスプローラー」においてデバッグ対象のプロジェクトをクリックします。
 - (2) 「実行」→「デバッグの構成…」の順にクリックし、「デバッグ構成」ウィンドウを開きます。
 - (3) 「デバッグ構成」ウィンドウで、“Renesas GDB Hardware Debugging” デバッグ構成の表示を展開し、デバッグ対象のデバッグ構成をクリックしてください。
 - (4) 「Startup」タブに切り替え、「Startup」タブの中の「イメージとシンボルをロード」の「追加…」ボタンを押下します。
 - (5) 「ダウンロード・モジュールの編集」ウィンドウにもう片方の起動バンク用のオブジェクトを指定し、「OK」ボタンを押下します。
 - (6) 「ロード・タイプ」を選択します。
e2studio で 1 度に維持できるデバッグシンボルテーブルは 1 つのみです。そのため、どちらか一方のアプリケーションの「ロード・タイプ」を「イメージとシンボル」にすることができます。
 - (7) 「オフセット」を指定します。
「オフセット」の指定はコードフラッシュメモリの容量によって設定が異なります。詳細は使用されるデバイス毎のユーザーズマニュアルの内容をご確認ください。
以下はターゲットデバイスが RX65N、コードフラッシュの容量が 2MB の場合の例となります。
「オフセット」は 1MB を 2 の補数にした fff00000 を入力してください。これによって、もう片方の起動バンクに割り付けるアプリケーションは、メモリ内で「リンカ／マップファイルで示される値 - 1MB」の位置に読み込まれます。
 - (8) 「適用」ボタンを押下します。



5.3.2 GCC for Renesas RX を使用する場合

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

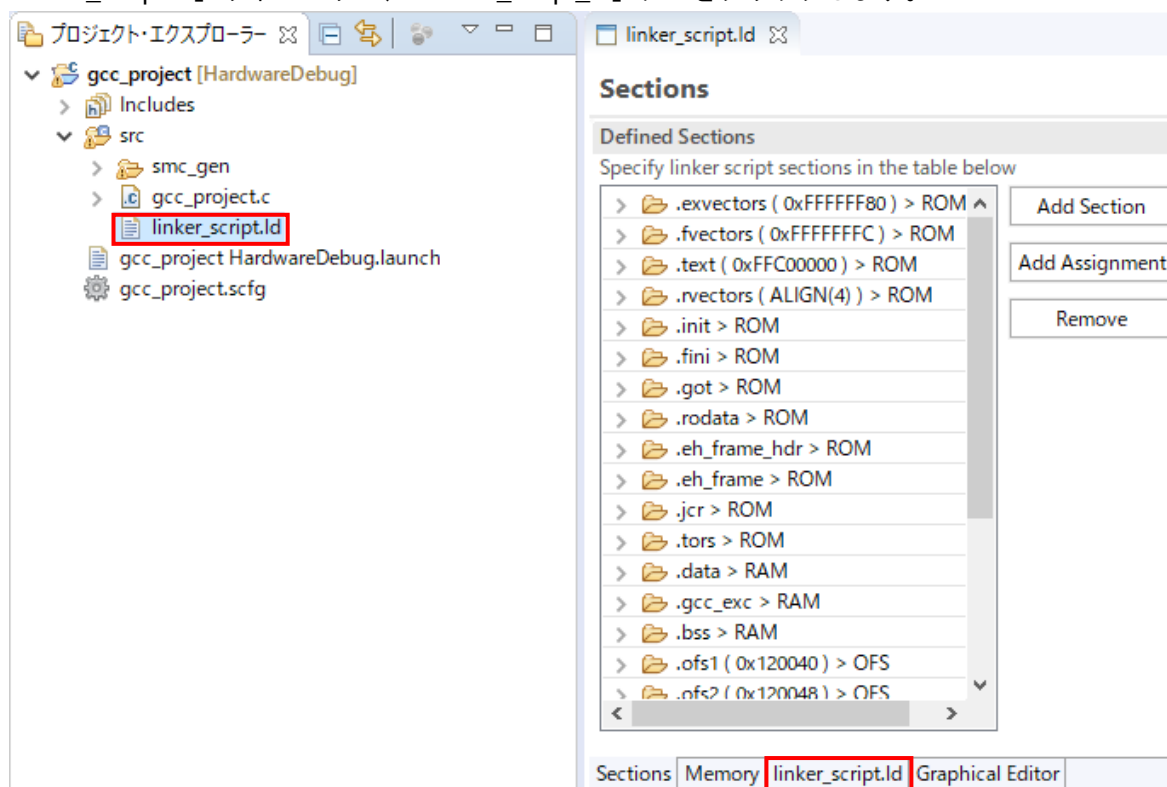
リンカの設定は e² studio で生成されるリンカ設定ファイルを編集する必要があります。

5.3.2.1 RAM からコードを実行してコードフラッシュを書き換える場合

本章では、リンカ設定の追加、コードフラッシュ書き換え中に動作するプログラムの配置について示します。

1. リンカ設定ファイル(linker_script.ld)に設定を追加します。

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



(3) 以下の(a)~(c)をリンカ設定ファイル(linker_script.ld)に追加します。

- (a)

```
. += _edata - _data;
```
- (b)

```
.pfram ALIGN(4):
{
    _PFRAM_start = .;
    . += _RPFRAM_end - _RPFRAM_start;
    _PFRAM_end = .;
} > ROM
```
- (c)

```
.rpfram ALIGN(4): AT(_PFRAM_start)
{
    _RPFRAM_start = .;
    *(PFRAM)
    . = ALIGN(4);
    _RPFRAM_end = .;
} > RAM
```

```

77      .ctors :
78      {
79          __CTOR_LIST__ = .;
80          . = ALIGN(2);
81          __ctors = .;
82          *(.ctors)
83          __ctors_end = .;
84          __CTOR_END__ = .;
85          __DTOR_LIST__ = .;
86          __dtors = .;
87          *(.dtors)
88          __dtors_end = .;
89          __DTOR_END__ = .;
90          . = ALIGN(2);
91          __mdata = .;
92          . += _edata - _data;
93      } > ROM
94      .pfram ALIGN(4):
95      {
96          __PFRAM_start = .;
97          . += _RPFRAM_end - _RPFRAM_start;
98          __PFRAM_end = .;
99      } > ROM
100     .data : AT(__mdata)
101     {
102         __data = .;
103         *(.data)
104         *(.data.*)
105         *(D)
106         *(D_1)
107         *(D_2)
108         __edata = .;
109     } > RAM
110     .rpfram ALIGN(4): AT(__PFRAM_start)
111     {
112         __RPFRAM_start = .;
113         *(PFRAM)
114         . = ALIGN(4);
115         __RPFRAM_end = .;
116     } > RAM
117     .gcc_exc :
118     {
119         *(.gcc_exc)
120     } > RAM

```

2. 割り込みコールバック関数等コードフラッシュ書き換え中に動作するプログラムは関数毎に FRAM セクションを指定して配置する必要があります。

```

__attribute__((section("PFRAM")))
/* コードフラッシュ書き換え中に動作する関数 */
void func(void){...}

```

```

__attribute__((section("PFRAM")))
/* コードフラッシュ書き換え中に動作するコールバック関数 */
void cb_func(void){...}

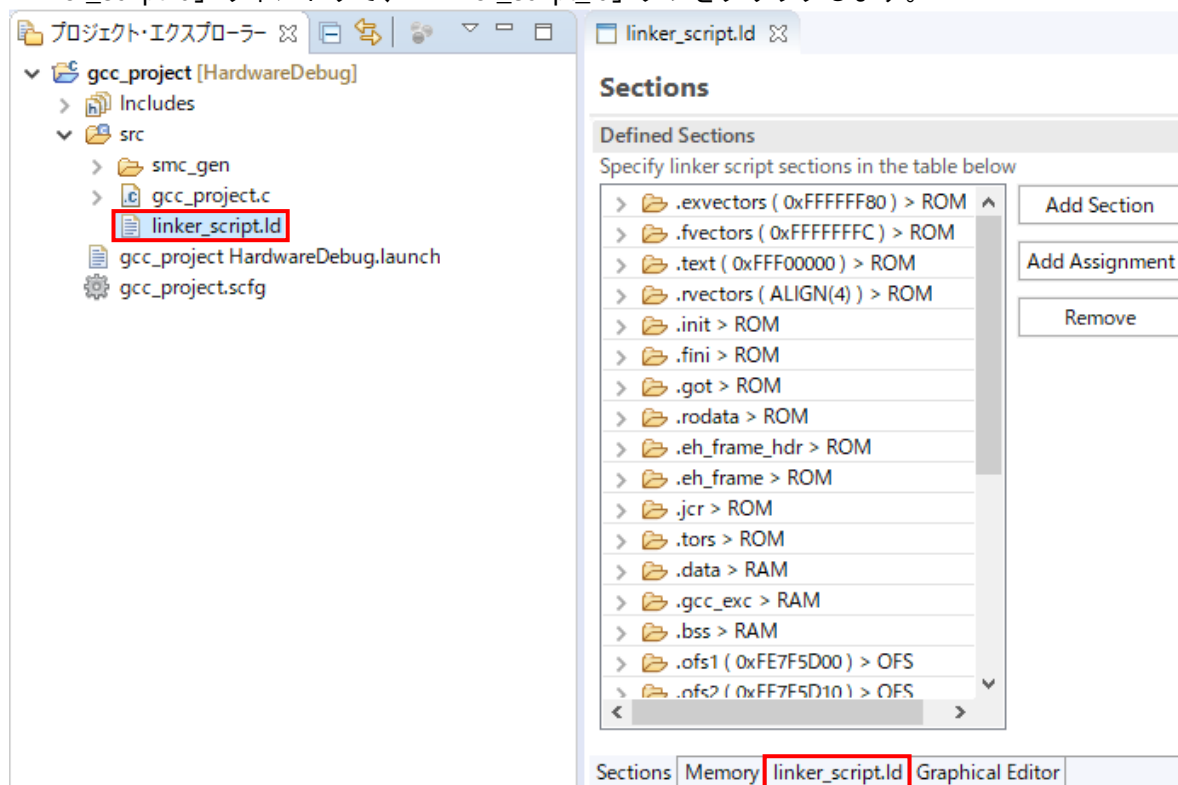
```

5.3.2.2 デュアルバンク機能を使用してコードフラッシュを書き換える場合

本章では、リンカ設定の追加、デュアルバンク機能のデバッグについて示します。

1. リンカ設定ファイル(linker_script.ld)に設定を追加します。

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



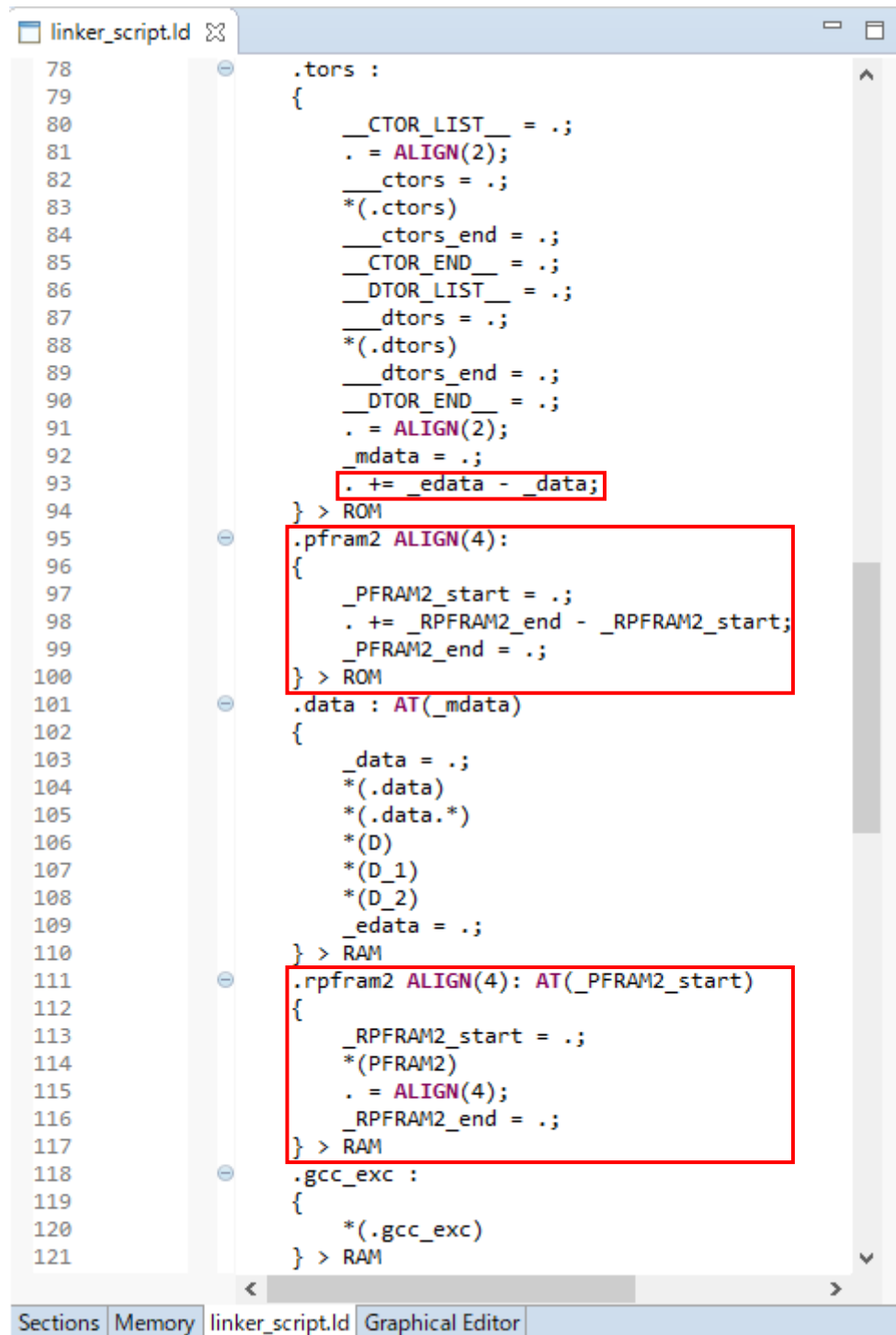
(3) 以下の(a)~(c)をリンカ設定ファイル(linker_script.ld)に追加します。

- (a)

```
. += _edata - _data;
```
- (b)

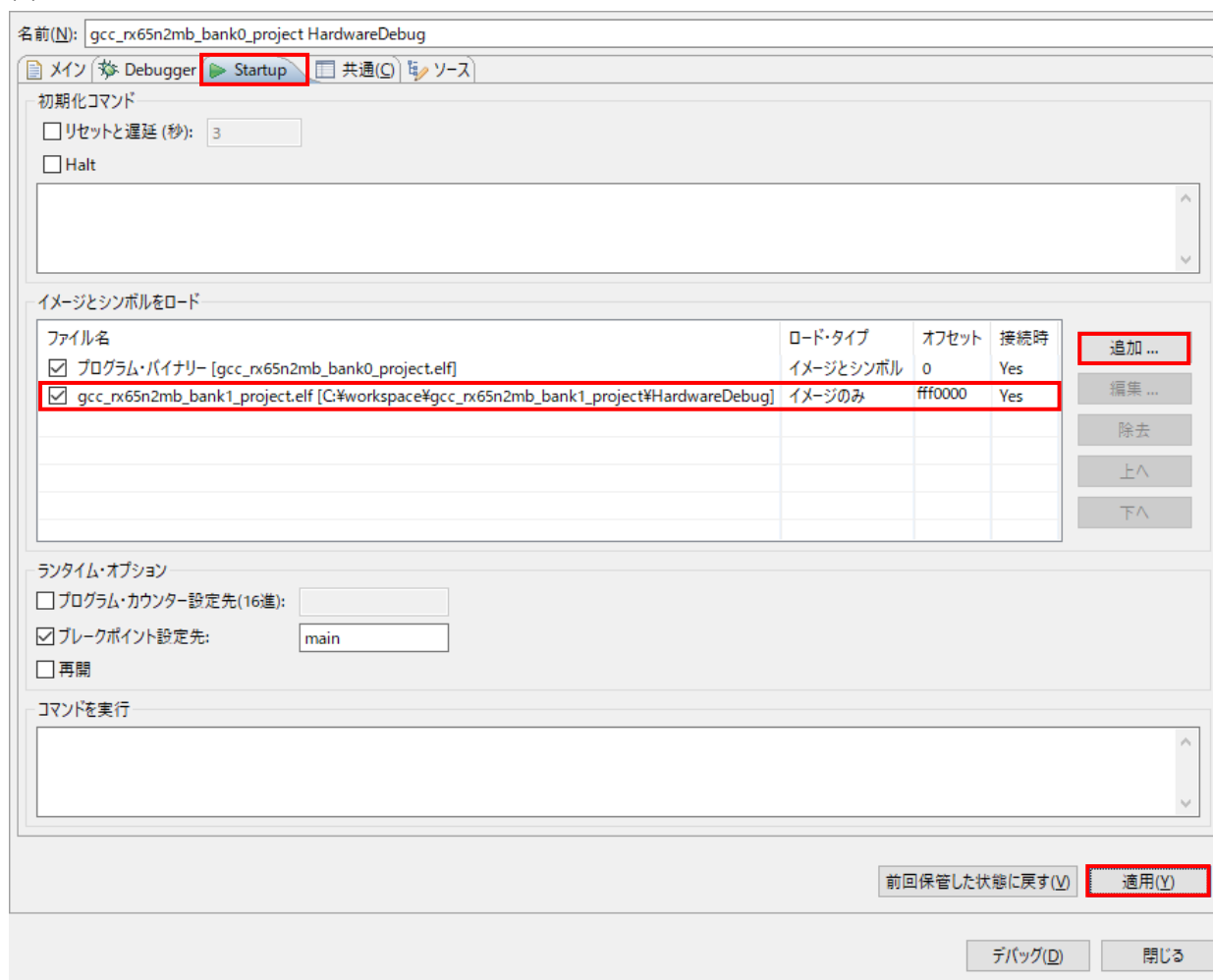
```
.pfram2 ALIGN(4) :
{
    _PFRAM2_start = .;
    . += _RPFRAM2_end - _RPFRAM2_start;
    _PFRAM2_end = .;
} > ROM
```
- (c)

```
.rpfram2 ALIGN(4) : AT(_PFRAM2_start)
{
    _RPFRAM2_start = .;
    *(PFRAM2)
    . = ALIGN(4);
    _RPFRAM2_end = .;
} > RAM
```



```
78     .ctors :
79     {
80         __CTOR_LIST__ = .;
81         . = ALIGN(2);
82         __ctors = .;
83         *(.ctors)
84         __ctors_end = .;
85         __CTOR_END__ = .;
86         __DTOR_LIST__ = .;
87         __dtors = .;
88         *(.dtors)
89         __dtors_end = .;
90         __DTOR_END__ = .;
91         . = ALIGN(2);
92         __mdata = .;
93         . += __edata - __data;
94     } > ROM
95     .pfram2 ALIGN(4):
96     {
97         __PFRAM2_start = .;
98         . += __RPFRAM2_end - __RPFRAM2_start;
99         __PFRAM2_end = .;
100    } > ROM
101    .data : AT(__mdata)
102    {
103        __data = .;
104        *(.data)
105        *(.data.*)
106        *(D)
107        *(D_1)
108        *(D_2)
109        __edata = .;
110    } > RAM
111    .rpfram2 ALIGN(4): AT(__PFRAM2_start)
112    {
113        __RPFRAM2_start = .;
114        *(PFRAM2)
115        . = ALIGN(4);
116        __RPFRAM2_end = .;
117    } > RAM
118    .gcc_exc :
119    {
120        *(.gcc_exc)
121    } > RAM
```

2. デュアルバンク機能に関連するアプリケーションをターゲットデバイスに接続してデバッグする際、2つのバンクに対するオブジェクトをロードする方法を以下に示します。必要に応じて実施してください。
 - (1) 「プロジェクト・エクスプローラー」においてデバッグ対象のプロジェクトをクリックします。
 - (2) 「実行」→「デバッグの構成…」の順にクリックし、「デバッグ構成」ウィンドウを開きます。
 - (3) 「デバッグ構成」ウィンドウで、“Renesas GDB Hardware Debugging” デバッグ構成の表示を展開し、デバッグ対象のデバッグ構成をクリックしてください。
 - (4) 「Startup」タブに切り替え、「Startup」タブの中の「イメージとシンボルをロード」の「追加…」ボタンを押下します。
 - (5) 「ダウンロード・モジュールの編集」ウィンドウにもう片方の起動バンク用のオブジェクトを指定し、「OK」ボタンを押下します。
 - (6) 「ロード・タイプ」を選択します。
e2studio で 1 度に維持できるデバッグシンボルテーブルは 1 つのみです。そのため、どちらか一方のアプリケーションの「ロード・タイプ」を「イメージとシンボル」にすることができます。
 - (7) 「オフセット」を指定します。
「オフセット」の指定はコードフラッシュメモリの容量によって設定が異なります。詳細は使用されるデバイス毎のユーザーズマニュアルの内容をご確認ください。
以下はターゲットデバイスが RX65N、コードフラッシュの容量が 2MB の場合の例となります。
「オフセット」は 1MB を 2 の補数にした fff00000 を入力してください。これによって、もう片方の起動バンクに割り付けるアプリケーションは、メモリ内で「リンカ／マップファイルで示される値 - 1MB」の位置に読み込まれます。
 - (8) 「適用」ボタンを押下します。



5.3.3 IAR C/C++ Compiler for Renesas RX を使用する場合

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

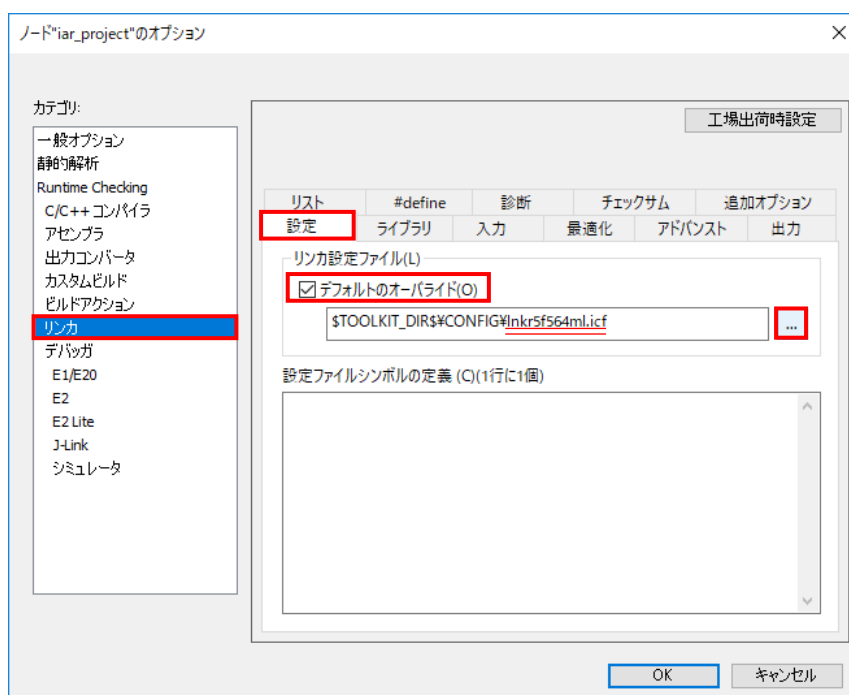
- IAR Embedded Workbench の FIT Module Importer を使用する方法
IAR Embedded Workbench の FIT Module Importer を使用し、本 FIT モジュールや BSP を追加した IAR 用のプロジェクトを生成して使用します。FIT Module Importer の詳細は IAR 社の Web サイトで最新の情報をご確認ください。
- IAR Embedded Workbench の IAR Project Converter を使用する方法
IAR Embedded Workbench の IAR Project Converter を使用し、CCRXX 用に作成したプロジェクトを IAR 用のプロジェクトにコンバートしてから使用します。IAR Project Converter の詳細は IAR 社の Web サイトで最新の情報をご確認ください。また、CCRXX 用に作成したプロジェクトを IAR 用のプロジェクトにコンバートする方法についてはアプリケーションノート「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)」にも記載しています。

本 FIT モジュールを IAR 用のプロジェクトで使用するには、以下の設定が必要となります。

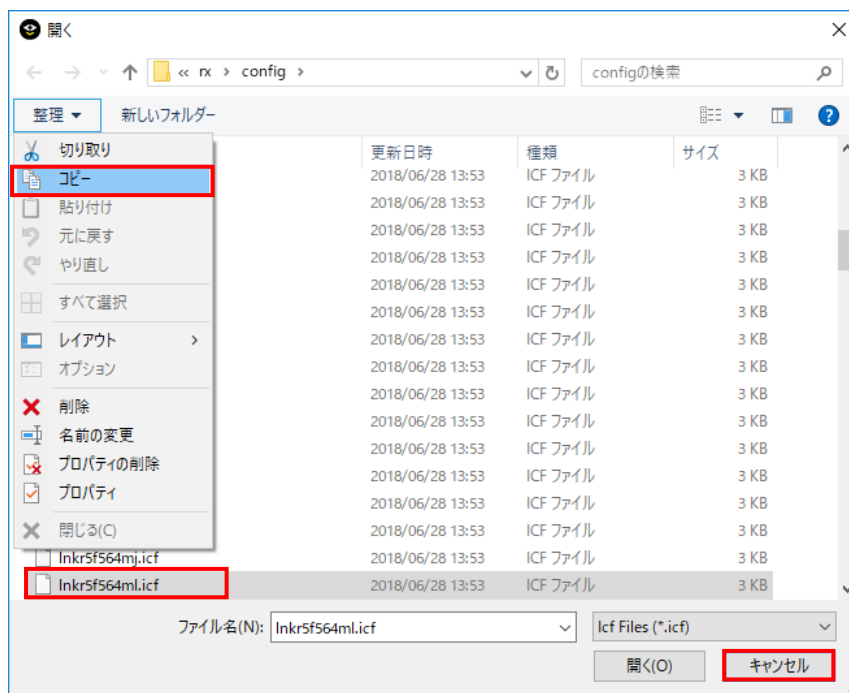
5.3.3.1 RAM からコードを実行してコードフラッシュを書き換える場合

本章では、リンカ設定の追加、コードフラッシュ書き換え中に動作するプログラムの配置について示します。

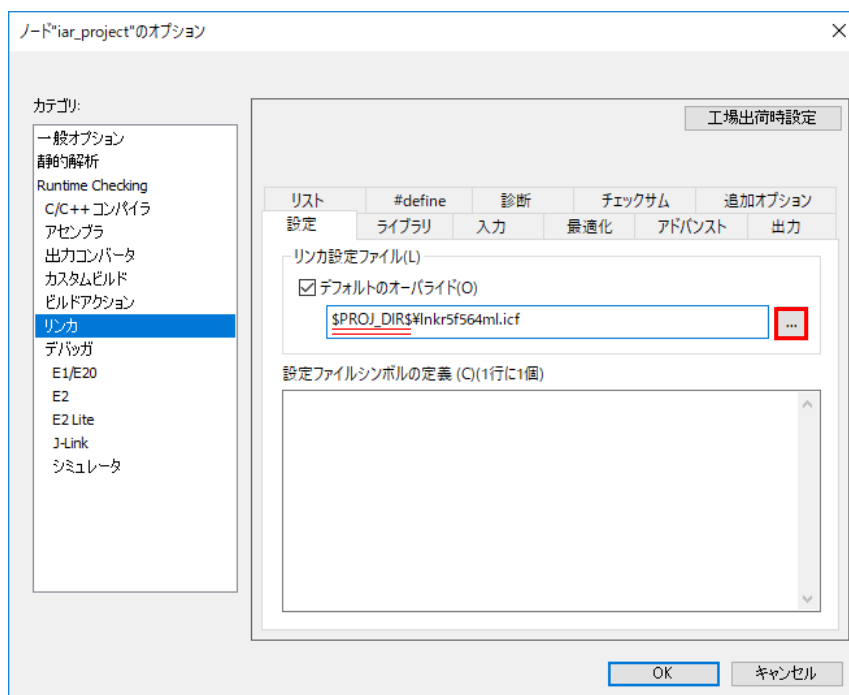
1. IAR 用のプロジェクトの「オプション」ウィンドウを開き、「カテゴリ：」から「リンカ」を選択し、「設定」タブを押下した後、リンカ設定ファイルで「デフォルトのオーバーライド」のチェックボックスがチェックされていることを確認し、「...」ボタンを押下します。



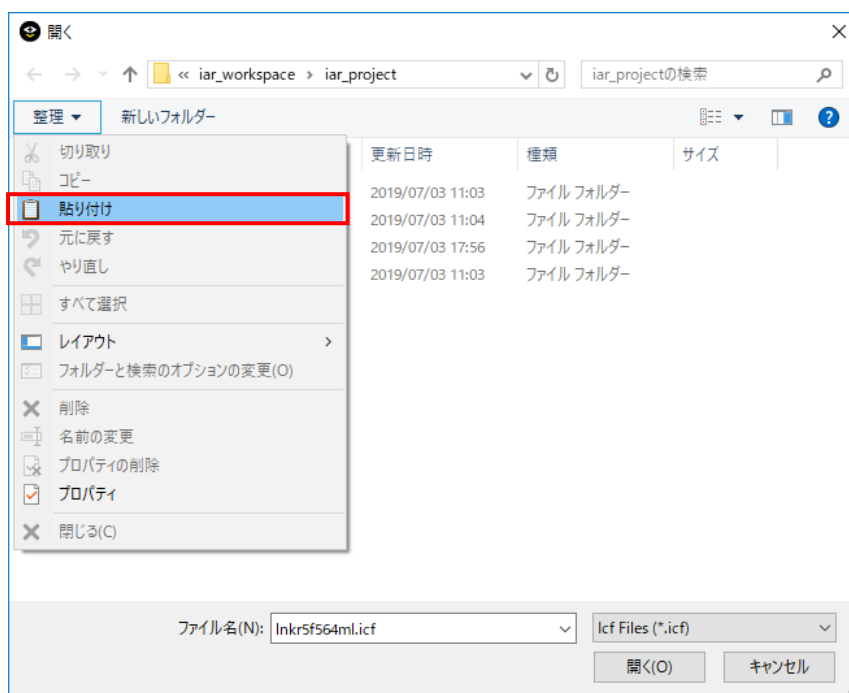
2. 「開く」ウィンドウでターゲットデバイスの.icf ファイル(1.の「オプション」ウィンドウのリンク設定ファイルのテキストボックスの二重下線部分)をコピーし、「キャンセル」ボタンを押下します。



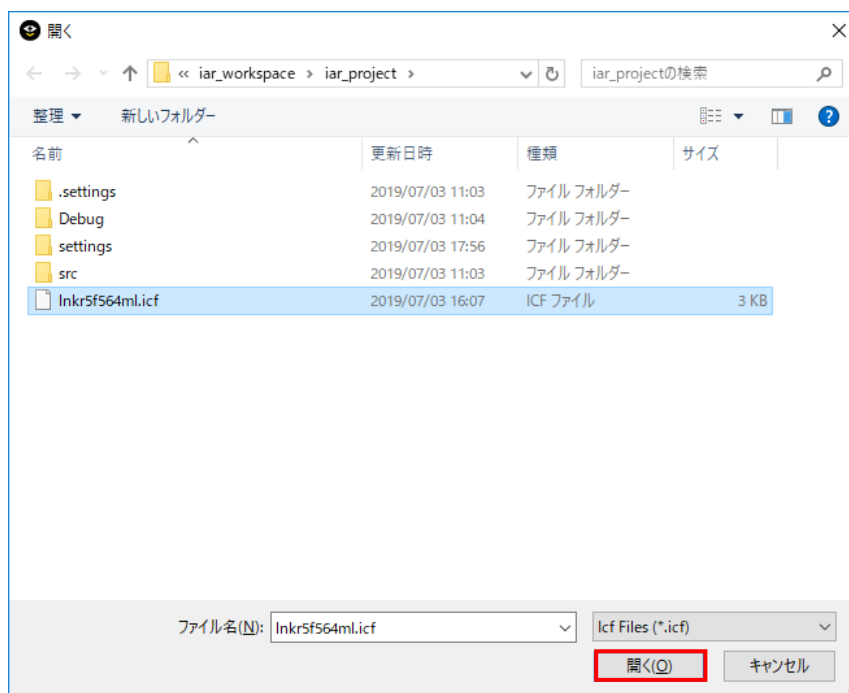
3. 「オプション」ウィンドウのリンク設定ファイルまでのパスを任意の場所に書き換えます。(ここではパス変数として"\$PROJ_DIR\$"を使用し、プロジェクトフォルダの直下となるようにしています) 書き換えた後、再び、「...」ボタンを押下します。



4. 「開く」ウィンドウで 2. でコピーしたターゲットデバイスの .icf ファイルを張り付けます。(ここではプロジェクトフォルダの直下に張り付けています)



「開く」ボタンを押下します。



ここまででデフォルトのリンカ設定ファイルをコピーして、コピーしたリンカ設定ファイルを編集する準備が整いました。

5. 以下の(a)~(d)をコピーして置換えたリンク設定ファイルに追加します。

- (a) `initialize manually { rw section .textrw, section PFRAM };`
- (b) `define block PFRAM with alignment = 4 { section PFRAM };`
`define block PFRAM_init with alignment = 4 { section PFRAM_init };`
- (c) `"ROM32":place in ROM_region32 { ro,`
`block PFRAM_init };`
- (d) `"RAM32":place in RAM_region32 { rw,`
`ro section D,`
`ro section D_1,`
`ro section D_2,`
`block PFRAM,`
`block HEAP };`

Inkr5f564ml.lcf x

```
//-----
// Linker configuration file template for the Renesas RX microcontroller R5F564ML
//-----
// Compatibility check
define exported symbol __link_file_version_4 = 1;

define memory mem with size = 4G;

define region RAM_region16 = mem:[from 0x00000004 to 0x00007FFF];
define region RAM_region24 = mem:[from 0x00000004 to 0x0007FFFF];
define region RAM_region32 = mem:[from 0x00000004 to 0x0007FFFF];

define region ROM_region16 = mem:[from 0xFFFF8000 to 0xFFFFFFFF];
define region ROM_region24 = mem:[from 0xFFC00000 to 0xFFFFFFFF];
define region ROM_region32 = mem:[from 0xFFC00000 to 0xFFFFFFFF];

define region DATA_FLASH = mem:[from 0x00100000 to 0x0010FFFF];

initialize manually { rw section .textrw, section PFRAM };
initialize by copy { rw, ro section D, ro section D_1, ro section D_2 };
initialize by copy with packing = none { section __DLIB_PERTHREAD };
do not initialize { section *.noinit };

define block HEAP with alignment = 4, size = _HEAP_SIZE { };
define block USTACK with alignment = 4, size = _USTACK_SIZE { };
define block ISTACK with alignment = 4, size = _ISTACK_SIZE { };

define block PFRAM with alignment = 4 { section PFRAM };
define block PFRAM_init with alignment = 4 { section PFRAM_init };

define block STACKS with fixed order { block USTACK,
block ISTACK };

place at address mem:0x00120040 { ro section .option_mem };
place at address mem:0xFFFFFFF0 { ro section .resetvect };
place at address mem:0xFFFFF80 { ro section .exceptvect };

"ROM16":place in ROM_region16 { ro section .code16*,
ro section .data16* };
"RAM16":place in RAM_region16 { rw section .data16*,
rw section __DLIB_PERTHREAD };
"ROM24":place in ROM_region24 { ro section .code24*,
ro section .data24* };
"RAM24":place in RAM_region24 { rw section .data24* };
"ROM32":place in ROM_region32 { ro,
block PFRAM_init };
"RAM32":place in RAM_region32 { rw,
ro section D,
ro section D_1,
ro section D_2,
block PFRAM,
block HEAP };
"STACKS":place at end of RAM_region32 { block STACKS };
```

6. 割り込みコールバック関数等コードフラッシュ書き換え中に動作するプログラムは関数毎に FRAM セクションを指定して配置する必要があります。

```
#pragma location="PFRAM"  
/* コードフラッシュ書き換え中に動作する関数 */  
void func(void){...}  
  
#pragma location="PFRAM"  
/* コードフラッシュ書き換え中に動作するコールバック関数 */  
void cb_func(void){...}
```

5.3.3.2 デュアルバンク機能を使用してコードフラッシュを書き換える場合

本章では、リンカ設定の追加について示します。

5.3.3.1 章の 1.~4.項を実施の上、以下の設定を実施してください。

1. 以下の(a)~(d)をコピーして置換えたリンク設定ファイルに変更、追加します。

- (a) デュアルモード時のバンク 0 の先頭アドレスに変更します。

```
define region ROM_region24 = mem:[from 0xFF00000 to 0xFFFFFFFF];
define region ROM_region32 = mem:[from 0xFF00000 to 0xFFFFFFFF];
```
- (b)

```
initialize manually { rw section .textrw, section PFRAM2 };
```
- (c)

```
define block PFRAM2 with alignment = 4 { section PFRAM2 };
define block PFRAM2_init with alignment = 4 { section PFRAM2_init };
```
- (d)

```
"ROM32":place in ROM_region32 { ro,
                                block PFRAM2_init };
```
- (e)

```
"RAM32":place in RAM_region32 { rw,
                                ro section D,
                                ro section D_1,
                                ro section D_2,
                                block PFRAM2,
                                block HEAP };
```

Inkr5f565ne_dual.icf x

```
//-----
// Linker configuration file template for the Renesas RX microcontroller R5F565NE_DUAL
//-----
// Compatibility check
define exported symbol __link_file_version_4 = 1;

define memory mem with size = 4G;

define region RAM_region1 = mem:[from 0x00000004 to 0x0003FFFF];
define region RAM_region2 = mem:[from 0x00800000 to 0x0085FFFF];

define region RAM_region16 = mem:[from 0x00000004 to 0x00007FFF];
define region RAM_region24 = RAM_region1 | RAM_region2;
define region RAM_region32 = RAM_region1 | RAM_region2;

define region STANDBY_RAM = mem:[from 0x000A4000 to 0x000A5FFF];

define region ROM_region16 = mem:[from 0xFFFF8000 to 0xFFFFFFFF];
define region ROM_region24 = mem:[from 0xFF00000 to 0xFFFFFFFF];
define region ROM_region32 = mem:[from 0xFF00000 to 0xFFFFFFFF];

define region DATA_FLASH = mem:[from 0x00100000 to 0x00107FFF];

initialize manually { rw section .textrw, section PFRAM2 };
initialize by copy { rw, ro section D, ro section D_1, ro section D_2 };
initialize by copy with packing = none { section __DLIB_PERTHREAD };
do not initialize { section .*.noinit };

define block HEAP with alignment = 4, size = _HEAP_SIZE { };
define block USTACK with alignment = 4, size = _USTACK_SIZE { };
define block ISTACK with alignment = 4, size = _ISTACK_SIZE { };

define block PFRAM2 with alignment = 4 { section PFRAM2 };
define block PFRAM2_init with alignment = 4 { section PFRAM2_init };

define block STACKS with fixed order { block USTACK,
                                      block ISTACK };

place at address mem:0xFE7F5D00 { ro section .option_mem };
place at address mem:0xFFFFFFF0 { ro section .resetvect };
place at address mem:0xFFFFF800 { ro section .exceptvect };

"ROM16":place in ROM_region16 { ro section .code16*,
                               ro section .data16* };
"RAM16":place in RAM_region16 { rw section .data16*,
                               rw section __DLIB_PERTHREAD };
"ROM24":place in ROM_region24 { ro section .code24*,
                               ro section .data24* };
"RAM24":place in RAM_region24 { rw section .data24* };
"ROM32":place in ROM_region32 { ro,
                               block PFRAM2_init };
"RAM32":place in RAM_region32 { rw,
                               ro section D,
                               ro section D_1,
                               ro section D_2,
                               block PFRAM2,
                               block HEAP };

"STACKS":place at end of RAM_region1 { block STACKS };
```

6. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境

RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）

（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.70	Oct.31.16	—	初版発行
		プログラム	イレーズの境界の問題を修正 (RX64M/71M)
			R_DF_Write_Operation()のビッグエンディアンの問題を修正 (フラッシュタイプ1)
			FLASH_xF_BLOCK_INVALID の値を修正 (フラッシュタイプ3)
			FLASH_CF_BLOCK_INVALID を修正 (RX210/21A/62N/630/63N/63T (フラッシュタイプ2))
			ロックビットの有効/無効コマンドを修正
			ロックビットの書き込み/読み出し、BGO のサポートを追加 (RX64M/71M)
			フラッシュへの大容量の書き込みに失敗しているにも関わらず、正常復帰するという問題 (不正な時間切れ処理) を修正 (RX64M/71M)
			R_FLASH_Control (FLASH_CMD_STATUS_GET, NULL)が常に BUSY を返すという問題に対応 (RX64M/71M)。 #if を追加して、BGO モードでない場合は ISR コードを省くようにしました。
			ブランクチェック結果が不正となる問題を修正 (フラッシュタイプ2)
2.00	Feb.17.17	—	FIT モジュールの RX230 グループ、RX24T グループ (いずれもフラッシュタイプ1) 対応。
		3	「1.2 BSP の使用に関するオプション」: FIT BSP を使用しない場合の説明を追加。
		10	「2.12 Flash FIT モジュールの追加方法」: 内容改訂。
		プログラム	フラッシュタイプ1 対象の定義 "FLASH_CF_LOWEST_VALID_BLOCK"と "FLASH_CF_BLOCK_INVALID"の値を変更。
2.10	Feb.17.17	—	FIT モジュールの RX24T グループ (ROM 512KB 版を含む)、RX24U グループ (いずれもフラッシュタイプ1) 対応。
		プログラム	全フラッシュタイプで小さいバグを修正し、パラメータチェックをさらに追加。変更の詳細は、r_flash_rx_if.h の「History」を参照してください。
3.00	Apr.28.17	7, 8	「2.9 コードサイズ」にて ROM および RAM のサイズを変更
		プログラム	タイプ1、3、4 に共通のコードをまとめ、動作が明確になるようにハイレベルコードの構成を見直し。
3.10	Apr.28.17	— 14 28 37	FIT モジュールでコードフラッシュメモリ容量が 1.5M バイト以上の RX65N グループの製品に対応。 「2.16 デュアルバンクの動作」を追加。 「3.6 R_FLASH_Control()」の Description にコマンド "FLASH_CMD_BANK_xxx"を追加。 「4.8 flash_demo_rskrx65n2mb_bank1_bootapp / _bank0_otherapp」を追加。

Rev.	発行日	改訂内容	
		ページ	ポイント
		プログラム	コマンド "FLASH_CMD_BANK_xxx"を追加。 フラッシュが非常に遅い場合、フラッシュタイプ 1 API 呼び出しから“BUSY”が返る可能性があるため、その問題に対応。 フラッシュタイプ 3 の初期化中に ECC フラグのクリアを追加。
3.20	Aug.11.17	1,4,6,7 9-17 38,39 プログラム	FIT モジュールの RX130-512KB 対応。 e ² studio の変更点を追加。 mcu_config.h が必要なのは BSP を使用しない場合にのみに変更。 RX65N-2M デュアルモードで、バンク 0 での実行時にバンクスワップを実行するとアプリケーションの実行が失敗することがあるバグを修正。
3.30	Dec.08.17	9,20 19,21 35 26	FLASH_ERR_ALREADY_OPEN を追加。 R_FLASH_Close()を追加。 フラッシュタイプ 2 のアクセスウィンドウ設定例を追加。 フラッシュタイプ 2 のブランクチェック例を追加。
3.40	Jul.17.18	1,5,6 15 15-16 44-46 プログラム	RX66T のサポートを追加。 RX111 グループおよび RX24T グループにおいて、ROM サイズが 256K/384K バイト製品のサポートを追加。 セクション 2.14 の表番号を更新。 セクション 2.15 に割り込みイベント列挙型を追加。 RDKRX63N、RSKRX66T 用のデモ、RSKRX64M 用の 2 つのデモを追加。 FPCKAR レジスタが Open()で設定されない RX64M/71M のバグを修正。
3.41	Nov.08.18	34 41 プログラム	R_FLASH_Control()の Description に以下のコマンドを追加 “FLASH_CMD_SET_NON_CACHED_xxxx” “FLASH_CMD_GET_NON_CACHED_xxxx” 追加コマンドの使用例を追加 FIT モジュールのサンプルプログラムをダウンロードするためのアプリケーションノートのドキュメント番号を xml ファイルに追加。
3.42	Feb.12.19	44-47	4.1～4.12 のタイトル誤記修正
3.50	Feb.26.19	1,5,6,34 49 プログラム	RX72T のサポートを追加。 RSKRX72T 用のデモを追加。 RX210 グループにおいて、ROM サイズが 768K/1M バイト製品での書き込みおよびイレーズできないバグを修正。
4.00	Apr.19.19	— 1,6 1 7	GCC/IAR コンパイラのサポートを追加。 対象デバイスから以下のフラッシュタイプ 2 のデバイスを削除。 RX210、RX21A、RX220、RX610、RX621、RX62N、RX62T、RX62G、RX630、RX631、RX63N、RX63T 関連するアプリケーションノートから以下を削除 e ² studio に組み込む方法 Firmware Integration Technology CS+に組み込む方法 Firmware Integration Technology Renesas e ² studio スマート・コンフィグレータ ユーザーガイド FLASH_CFG_USE_FIT_BSP を削除。

Rev.	発行日	改訂内容	
		ページ	ポイント
		9-12 13 14 15 26	FLASH_CFG_FLASH_READY_IPL を削除。 FLASH_CFG_IGNORE_LOCK_BITS を削除。 FLASH_CFG_DATA_FLASH_BGO の説明を追加 FLASH_CFG_CODE_FLASH_BGO の説明を追加 「2.9 コードサイズ」の章を更新。 「2.11 戻り値」の章から不要となった以下の戻り値を削除 FLASH_ERR_ALIGNED FLASH_ERR_BOUNDARY FLASH_ERR_OVERFLOW 「2.12 FLASH FIT モジュールの追加方法」の章を更新。 「2.13 既存のユーザプロジェクトと組み合わせた使用方法」の章を追加。 「2.14 RAM からコードを実行してコードフラッシュを書き換える」の章の構成を以下のように見直し更新。 「2.14.1 Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合」、 「2.14.2 GCC for Renesas RX を使用する場合」、 「2.14.3 IAR C/C++ Compiler for Renesas RX を使用する場合」 「2.18.4 エミュレータのデバッグ設定」の章を追加。
		プログラム	GCC/IAR コンパイラのサポートを追加に伴う変更。 FLASH_CFG_USE_FIT_BSP の削除に伴う変更。 FLASH_CFG_FLASH_READY_IPL の削除に伴う変更。 FLASH_CFG_IGNORE_LOCK_BITS の削除に伴う変更。 対象デバイスからフラッシュタイプ2のデバイスを削除。 FLASH_ERR_ALIGNED を削除。 FLASH_ERR_BOUNDARY を削除。 FLASH_ERR_OVERFLOW を削除。 BSP が Rev.5.00 未満の場合にエラー出力する処理を追加。
4.10	Jun.07.19	1,7 10-14 20-21 56 57-58	RX23W のサポートを追加。 「2.9 コードサイズ」の章を更新。 「2.14.2 GCC for Renesas RX を使用する場合」の章を更新。 「5. 付録」の章を追加。 「5.1 動作確認環境」の章を追加。 「5.2 トラブルシューティング」の章を追加。
		プログラム	RX23W のサポートを追加。 FEARL および FSARL レジスタの設定を修正。 デモプロジェクトの環境を更新。
4.20	Jul.19.19	1,7 55 58	RX72M のサポートを追加。 「4.13 flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp」の章を追加。 「5.1 動作確認環境」の章を更新。
		プログラム	RX72M のサポートを追加。 RX72M のデモプロジェクトを追加。 デモプロジェクトの環境を更新。 ワーニング除去。 使用されていない定義やインクルードの削除。 グローバル変数に volatile 宣言を付与。 デュアルモードとリニアモードのセクションに関する修正。 フラッシュタイプ4のタイムアウト処理を一部見直し。

Rev.	発行日	改訂内容	
		ページ	ポイント
4.30	Sep.09.19	1, 7 6 10-14 17	RX13T のサポートを追加。 「2.5 使用する割り込みベクタ」の章を追加。 「2.10 コードサイズ」の章を更新。 以下の記載を見直し「5.3 コンパイラ依存の設定」へ移動。 「2.14.1 Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合」 「2.14.2 GCC for Renesas RX を使用する場合」 「2.14.3 IAR C/C++ Compiler for Renesas RX を使用する場合」 「2.18 デュアルバンクの動作」を見直しコンパイラに依存する内容を「5.3 コンパイラ依存の設定」へ移動。 「5.1 動作確認環境」の章を更新。 「5.3 コンパイラ依存の設定」の章を追加。
		19 49 52-69 プログラム	RX13T のサポートを追加。 フラッシュタイプ 1 のエラー処理を一部見直し。 R_FlashCodeCopy() の IAR 時のコピー方法を見直し。 r_flash_control() の実装方式を if_then 方式に見直し。
4.40	Sep.27.19	1, 6, 7 27 49	RX23E-A のサポートを追加。 「3.5 R_FLASH_BlankCheck()」の章の Return Values に FLASH_ERR_NULL_PTR を追加。 「5.1 動作確認環境」の章を更新。
		プログラム	RX23E-A のサポートを追加。 r_flash_blankcheck() の第 3 引数の NULL チェックを追加。
4.50	Nov.18.19	1, 7 6 15 18 22-37	RX66N、RX72N のサポートを追加。 「2.3 制限事項」の章に制限事項を追加。 「2.13 ブロッキングモード、ノンブロッキングモード」の章を追加。 「2.17 BGO モードでの動作」の章を削除。 「3.2 R_FLASH_Open()」、「3.3 R_FLASH_Close()」、 「3.4 R_FLASH_Erase()」、「3.5 R_FLASH_BlankCheck()」、 「3.6 R_FLASH_Write()」、「3.7 R_FLASH_Control()」、 「3.8 R_FLASH_GetVersion()」 から Reentrant に関する記載を削除 「3.7 R_FLASH_Control」の Description の記載を見直し。 「5.1 動作確認環境」の章を更新。
		30-32 48 プログラム	RX66N、RX72N のサポートを追加。 Doxygen 対応。 IEN の有効、無効を R_BSP_InterruptRequestEnable()、 R_BSP_InterruptRequestDisable() を使用するよう見直し。

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くと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、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には、お客様の責任において行ってください。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まれます。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、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、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。
 5. 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、当社は一切その責任を負いません。
 6. 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。
 7. 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。
 8. 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。
 9. 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。
 10. 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しします。
 11. 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。
 12. 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.4.0-1 2017.11)

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。