

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)を使用したフラッシュモジュール^{*1}について説明します。

本モジュールを使用することによって、セルフプログラミング^{*2}による内蔵フラッシュメモリの書き換え処理を、ユーザアプリケーションに対して容易に実装することができます。

本アプリケーションノートでは、本モジュールの使用方法、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示します。

^{*1} 本モジュールは、「RX 用シンプルフラッシュ API (R01AN0544)」とは異なります。

^{*2} セルフプログラミングとは、ユーザプログラムを使用してフラッシュメモリを書き換える方式です。

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
- ・ RX671 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。

関連ドキュメント

- 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 フラッシュモジュールの概要	5
1.1.1 フラッシュタイプの概要	5
1.1.2 サポートしている機能	6
1.2 API の概要	7
1.3 制限事項	8
1.3.1 フラッシュメモリのアクセスに関する制限事項	8
1.3.2 RAM の配置に関する制限事項	8
1.3.3 エミュレータのデバッグ設定に関する制限事項	9
2. API 情報	10
2.1 ハードウェアの要求	10
2.2 ソフトウェアの要求	10
2.3 サポートされているツールチェーン	10
2.4 使用する割り込みベクタ	10
2.5 ヘッダファイル	10
2.6 整数型	10
2.7 コンパイル時の設定	11
2.8 コードサイズ	12
2.9 引数	17
2.9.1 共通で使用する引数の定義	17
2.9.2 フラッシュメモリの機能や容量の違いによって異なる引数の定義	20
2.10 戻り値	24
2.11 コールバック関数	25
2.12 FIT モジュールの追加方法	28
2.13 ブロッキングモード、ノンブロッキングモード	29
2.13.1 ブロッキングモードで使用する場合	29
2.13.2 ノンブロッキングモードで使用する場合	29
2.14 アクセスウィンドウ、ロックビットによる領域の保護	30
2.14.1 アクセスウィンドウによる領域の保護	30
2.14.2 ロックビットによる領域の保護	30
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	31
2.16 フラッシュメモリの書き換え	32
2.16.1 RAM からコードを実行してコードフラッシュメモリを書き換える	33
2.16.2 コードフラッシュメモリからコードを実行してコードフラッシュメモリを書き換える	34
2.16.3 デュアルバンク機能と組み合わせてコードフラッシュメモリを書き換える	35
3. API 関数	37
3.1 R_FLASH_Open()	37
3.2 R_FLASH_Close()	40
3.3 R_FLASH_Erase()	41
3.4 R_FLASH_BlankCheck()	44
3.5 R_FLASH_Write()	47
3.6 R_FLASH_Control()	50
3.7 R_FLASH_GetVersion()	68
4. デモプロジェクト	69
4.1 flash_demo_rskrx113	69
4.2 flash_demo_rskrx231	69
4.3 flash_demo_rskrx23t	70
4.4 flash_demo_rskrx130	70
4.5 flash_demo_rskrx24t	71
4.6 flash_demo_rskrx65n	71
4.7 flash_demo_rskrx24u	72
4.8 flash_demo_rskrx65n2mb_bank0_bootapp / _bank1_otherapp	72
4.9 flash_demo_rskrx64m	73

4.10	flash_demo_rskrx64m_runrom	73
4.11	flash_demo_rskrx66t	74
4.12	flash_demo_rskrx72t	74
4.13	flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp	75
4.14	ワークスペースにデモを追加する	76
4.15	デモのダウンロード方法	76
5.	付録	77
5.1	動作確認環境	77
5.2	トラブルシューティング	80
5.3	コンパイラ依存の設定	82
5.3.1	Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合	82
5.3.1.1	Renesas Electronics C/C++ Compiler Package for RX Family を使用する場合	83
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	85
5.3.2	GCC for Renesas RX を使用する場合	88
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	88
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	90
5.3.3	IAR C/C++ Compiler for Renesas RX を使用する場合	93
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	93
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	98
6.	参考ドキュメント	100
	改訂記録	101

1. 概要

1.1 フラッシュモジュールの概要

本モジュールでは、MCU に内蔵されているフラッシュメモリ(コードフラッシュメモリ、データフラッシュメモリ)を書き換えの対象としています。

本モジュールでは、フラッシュメモリを書き換える API 関数を用意しています。

1.1.1 フラッシュタイプの概要

フラッシュメモリは MCU によってサポートされている機能に違いがあるため、本モジュールでは便宜上、表 1.1 のような分類としています。

表 1.1 サポート対象としている MCU グループとフラッシュタイプとの関係

フラッシュタイプ	サポート対象の MCU グループ
1	RX110 ^{*1} 、RX111、RX113、RX130、RX13T RX230、RX231、RX23E-A、RX23T ^{*1} 、RX23W、RX24T、RX24U
3	RX64M、RX66T、RX71M、RX72T
4	RX651 ^{*2} 、RX65N ^{*2} 、RX66N、RX671、RX72M、RX72N

^{*1} データフラッシュメモリはありません。

^{*2} 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メモリはありません。

1.1.2 サポートしている機能

本モジュールでサポートしている機能と各フラッシュタイプとの関係を表 1.2 に示します。

表 1.2 サポートしている機能とフラッシュタイプとの関係

機能	概要	フラッシュタイプ		
		1	3	4
プログラム	指定された領域をプログラムします。	✓	✓	✓
イレース	指定された領域をイレースします。	✓	✓	✓
ブランクチェック	指定された領域にプログラムされていないことを確認します。	✓ *1	✓ *1	✓ *1
アクセスウィンドウ	指定された領域のみを書き換え可能な領域とし、その他の領域を保護します。	✓ *2	—	✓ *2
スタートアッププログラム保護	リセット後に起動するプログラム(スタートアッププログラム)の領域を切り替えることにより、スタートアップ領域を保護します。	✓ *3	—	✓
ロックビット	指定された領域の書き換えの有効/無効を切り替えることにより、指定された領域を保護します。	—	✓*4	—
ROM キャッシュ	コードフラッシュメモリのキャッシュの許可/禁止を設定します。	✓ *5	✓ *6	✓
キャッシュ無効化	キャッシュを無効にする領域を設定します。	—	✓ *6	✓ *7
デュアルバンク	起動バンクを切り替えます。	—	—	✓
フラッシュシーケンサリセット	フラッシュシーケンサをリセットします。	✓	✓	✓
フラッシュシーケンサ使用周波数通知	フラッシュシーケンサに使用する周波数を通知します。	—	✓	✓

*1 コードフラッシュメモリに対してブランクチェック出来るのはフラッシュタイプ 1 のみです。

*2 アクセスウィンドウの対象領域はコードフラッシュメモリのみです。

*3 コードフラッシュメモリの容量が 32K バイト以上の製品のみ対象となります。

*4 ロックビットの対象領域はコードフラッシュメモリのみです。

*5 RX24T、RX24U のみ対象となります。

*6 RX66T、RX72T のみ対象となります。

*7 RX66N、RX671、RX72M、RX72N のみ対象となります。

1.2 API の概要

表 1.3 に本モジュールに含まれる API 情報を示します。

表 1.3 API 関数一覧

関数	関数説明
R_FLASH_Open()	本モジュールを初期化します。
R_FLASH_Close()	本モジュールを終了します。
R_FLASH_Erase()	データフラッシュメモリ、またはコードフラッシュメモリの指定されたブロックをイレーズします。
R_FLASH_BlankCheck()	データフラッシュメモリ、またはコードフラッシュメモリの指定された領域にプログラムされていないことを確認します。
R_FLASH_Write()	データフラッシュメモリ、またはコードフラッシュメモリの指定された領域に指定されたデータをプログラムします。
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実行します。
R_FLASH_GetVersion()	本モジュールのバージョン番号を返します。

1.3 制限事項

1.3.1 フラッシュメモリのアクセスに関する制限事項

フラッシュシーケンサにはフラッシュメモリを読み出すリードモードとフラッシュメモリに対する書き換えを行う P/E モードが存在します。

P/E モード中の場合、表 1.4 のようにリードアクセス禁止領域と可能領域があります。

表 1.4 P/E モード中のリードアクセス禁止領域と可能領域

P/E モード中の領域	リードアクセス禁止領域	リードアクセス可能領域 ^{*1}
コードフラッシュメモリ	コードフラッシュメモリ	データフラッシュメモリ RAM 外部メモリ 他のコードフラッシュメモリ ^{*2}
データフラッシュメモリ	データフラッシュメモリ	コードフラッシュメモリ RAM 外部メモリ

^{*1} 書き換え用のコードと割り込みベクタテーブルはデータフラッシュメモリを除く、リードアクセス可能領域に配置してください。

^{*2} コードフラッシュメモリの領域が複数ある製品の場合。

書き換え用のコードを RAM から実行させる方法については 2.16.1 章を参照してください。

他のコードフラッシュメモリからコードフラッシュメモリを書き換える方法については 2.16.2 章を参照してください。

割り込みベクタテーブルや割り込み処理の再配置については 3.6 章の Example 1 を参照してください。

1.3.2 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。

標準ライブラリの仕様で 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。

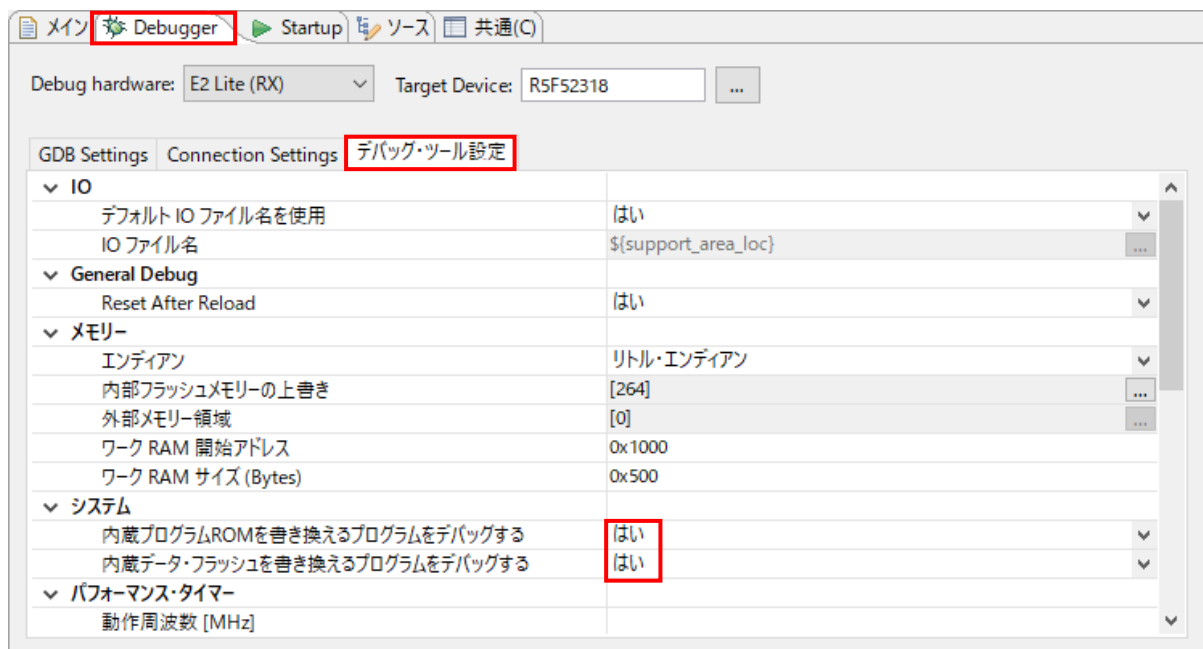
1.3.3 エミュレータのデバッグ設定に関する制限事項

デバッグ時にコードフラッシュメモリ、データフラッシュメモリ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"



2. API 情報

本モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- フラッシュメモリ(コードフラッシュメモリ、データフラッシュメモリ)

2.2 ソフトウェアの要求

このドライバは以下の FIT モジュールに依存しています。

- ボードサポートパッケージ (r_bsp) v.5.20 以降のバージョン

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を行っています。

2.4 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(2.7 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

フラッシュタイプ	割り込みベクタ
1	FRDYI 割り込み (ベクタ番号: 23)
3、4	FRDYI 割り込み (ベクタ番号: 23)、FIFERR 割り込み (ベクタ番号: 21)

2.5 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、フラッシュモジュール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。

2.6 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。

2.7 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。

Configuration options in r_flash_rx_config.h	
FLASH_CFG_PARAM_CHECKING_ENABLE *デフォルト値は"1"	パラメータチェック処理をコードに含めるか選択できます。 "0"の場合、パラメータチェック処理をコードから省略します。 "1"の場合、パラメータチェック処理をコードに含めます。
FLASH_CFG_CODE_FLASH_ENABLE *デフォルト値は"0"	コードフラッシュメモリの領域をプログラムするためのコードを含めるかを指定します。 "0"の場合、データフラッシュメモリの領域のみをプログラムするためのコードを含みます。(コードフラッシュメモリの領域をプログラムするためのコードは含みません。) "1"の場合、コードフラッシュメモリの領域をプログラムするためのコードを含みます。(データフラッシュメモリの領域をプログラムするためのコードも含みます。)
FLASH_CFG_DATA_FLASH_BGO *デフォルト値は"0"	データフラッシュメモリに関する処理の仕方を指定します。 "0"の場合、ブロッキングモードでデータフラッシュメモリに関する処理を行います。 "1"の場合、ノンブロッキングモードでデータフラッシュメモリに関する処理を行います。 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。 ブロッキングモード、ノンブロッキングモードの詳細に関しては 2.13 を参照ください。
FLASH_CFG_CODE_FLASH_BGO *デフォルト値は"0"	コードフラッシュメモリに関する処理の仕方を指定します。 "0"の場合、ブロッキングモードでコードフラッシュメモリに関する処理を行います。 "1"の場合、ノンブロッキングモードでコードフラッシュメモリに関する処理を行います。 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。 ブロッキングモード、ノンブロッキングモードの詳細に関しては 2.13 を参照ください。
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM* ¹ *デフォルト値は"0"	フラッシュメモリに対してプログラムやイレーズを実行するためのコードの配置を指定します。 このオプションは、FLASH_CFG_CODE_FLASH_ENABLE が"1"に設定されている場合のみ有効です。 "0"の場合、フラッシュメモリに対してプログラムやイレーズを実行するためのコードを RAM 上に配置し、RAM 上でコードを実行します。詳細は 2.16.1 章を参照ください。 "1"の場合、フラッシュメモリに対してプログラムやイレーズを実行するためのコードをコードフラッシュメモリ上に配置し、コードフラッシュメモリ上でコードを実行します。詳細は 2.16.2 章を参照ください。

*¹ コードフラッシュメモリが複数の領域に分かれている製品のみ対象。

2.8 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 1 の製品についてはデータフラッシュメモリを搭載する製品と搭載しない製品について、フラッシュタイプ 3 と 4 の製品についてはそれぞれ一例ずつ掲載しています。

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 ^{*1}	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

^{*1} データフラッシュメモリ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 ^{*1}	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

^{*1} データフラッシュメモリ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

2.9 引数

API 関数の引数で使用する構造体、列挙体の定義を示します。これらは本モジュールにおいて共通で使用する定義と、フラッシュメモリの機能や容量の違いによって異なる定義があります。

2.9.1 共通で使用する引数の定義

本モジュールの引数として共通に使用する構造体、列挙体については `r_flash_rx_if.h` に定義されています。

```
/* コールバック関数のイベントタイプ */
typedef enum _flash_interrupt_event
{
    FLASH_INT_EVENT_INITIALIZED,           // この値は返りません
    FLASH_INT_EVENT_ERASE_COMPLETE,        // イレーズ完了
    FLASH_INT_EVENT_WRITE_COMPLETE,        // プログラム完了
    FLASH_INT_EVENT_BLANK,                 // ブランクチェックの結果、ブランク状態
    FLASH_INT_EVENT_NOT_BLANK,             // ブランクチェックの結果、非ブランク状態
    FLASH_INT_EVENT_TOGGLE_STARTUPAREA,    // スタートアップ領域の切り替え
    FLASH_INT_EVENT_SET_ACCESSWINDOW,      // アクセスウィンドウの設定
    FLASH_INT_EVENT_LOCKBIT_WRITTEN,        // ロックビットの設定
    FLASH_INT_EVENT_LOCKBIT_PROTECTED,      // ロックビットのプロテクトが有効
    FLASH_INT_EVENT_LOCKBIT_NON_PROTECTED,  // ロックビットのプロテクトが無効
    FLASH_INT_EVENT_ERR_DF_ACCESS,          // データフラッシュメモリのアクセス違反
    FLASH_INT_EVENT_ERR_CF_ACCESS,          // コードフラッシュメモリのアクセス違反
    FLASH_INT_EVENT_ERR_SECURITY,           // アクセスウィンドウの書き込み保護違反
    FLASH_INT_EVENT_ERR_CMD_LOCKED,         // コマンドがロック状態
    FLASH_INT_EVENT_ERR_LOCKBIT_SET,        // ロックビットで保護されている領域でのエラー
    FLASH_INT_EVENT_ERR_FAILURE,            // プログラムやイレーズ中のエラー
    FLASH_INT_EVENT_TOGGLE_BANK,           // 起動バンクの切り替え
    FLASH_INT_EVENT_END_ENUM               // この値は返りません
} flash_interrupt_event_t;
```

```
/* コールバック関数を登録する際に使用する定義 */
typedef struct _flash_interrupt_config
{
    void      (*pcallback)(void *);        // コールバック関数へのポインタ
    uint8_t   int_priority;                 // 割り込み優先度
} flash_interrupt_config_t;
```

```
/* コールバック関数の引数として使用する定義 */
typedef struct
{
    flash_interrupt_event_t event;          // 割り込みの要因となったイベント
} flash_int_cb_args_t;
```

```

/* R_FLASH_Control 関数のコマンド定義 */
typedef enum _flash_cmd
{
    FLASH_CMD_RESET,                // フラッシュシーケンサをリセット
    FLASH_CMD_STATUS_GET,           // FLASH FIT モジュールの API の状態を取得
    FLASH_CMD_SET_BGO_CALLBACK,     // コールバック関数を登録
    FLASH_CMD_SWAPFLAG_GET,         // 現在のスタートアップ領域の設定を取得
    FLASH_CMD_SWAPFLAG_TOGGLE,     // スタートアップ領域の切り替え
    FLASH_CMD_SWAPSTATE_GET,        // スタートアップ領域選択ビットの設定を取得
    FLASH_CMD_SWAPSTATE_SET,        // スタートアップ領域選択ビットを設定
    FLASH_CMD_ACCESSWINDOW_SET,     // アクセスウィンドウの境界を設定
    FLASH_CMD_ACCESSWINDOW_GET,     // アクセスウィンドウの境界を取得
    FLASH_CMD_LOCKBIT_READ,         // 指定したブロックのロックビット情報を取得
    FLASH_CMD_LOCKBIT_WRITE,        // 指定したブロックにロックビットを設定
    FLASH_CMD_LOCKBIT_ENABLE,       // ロックビットによるプロテクトを有効
    FLASH_CMD_LOCKBIT_DISABLE,      // ロックビットによるプロテクトを無効
    FLASH_CMD_CONFIG_CLOCK,         // フラッシュシーケンサに動作周波数を通知
    FLASH_CMD_ROM_CACHE_ENABLE,     // ROM キャッシュを有効
    FLASH_CMD_ROM_CACHE_DISABLE,    // ROM キャッシュを無効
    FLASH_CMD_ROM_CACHE_STATUS,     // ROM キャッシュの状態 (有効/無効) を取得
    FLASH_CMD_SET_NON_CACHED_RANGE0, // キャッシュを無効にする領域 0 の範囲を設定
    FLASH_CMD_SET_NON_CACHED_RANGE1, // キャッシュを無効にする領域 1 の範囲を設定
    FLASH_CMD_GET_NON_CACHED_RANGE0, // キャッシュを無効にしている領域 0 の設定を取得
    FLASH_CMD_GET_NON_CACHED_RANGE1, // キャッシュを無効にしている領域 1 の設定を取得
    FLASH_CMD_BANK_TOGGLE,          // 起動バンクの切り替え
    FLASH_CMD_BANK_GET,              // 現在のバンク選択レジスタの設定を取得
    FLASH_CMD_END_ENUM              // 本定義は使用できません
} flash_cmd_t;

```

```

/* R_FLASH_Control 関数、R_FLASH_BlankCheck 関数の結果の定義
typedef enum _flash_res
{
    FLASH_RES_LOCKBIT_STATE_PROTECTED, // FLASH_CMD_LOCKBIT_READ の結果、プロテクト状態
    FLASH_RES_LOCKBIT_STATE_NON_PROTECTED, // FLASH_CMD_LOCKBIT_READ の結果、非プロテクト状態
    FLASH_RES_BLANK,                    // R_FLASH_BlankCheck の結果、ブランク状態
    FLASH_RES_NOT_BLANK                 // R_FLASH_BlankCheck の結果、非ブランク状態
} flash_res_t;

```

```

/* R_FLASH_Control 関数の FLASH_CMD_BANK_GET コマンドにおいて使用する定義 */
typedef enum _flash_bank
{
    FLASH_BANK1      = 0,                // BANKSEL.BANKSWP is 000
    FLASH_BANK0      = 1,                // BANKSEL.BANKSWP is 111
    FLASH_BANK0_FFE00000 = 0,            // BANKSEL.BANKSWP is 000
    FLASH_BANK1_FFF00000 = 0,            // BANKSEL.BANKSWP is 000
    FLASH_BANK0_FFF00000 = 1,            // BANKSEL.BANKSWP is 111
    FLASH_BANK1_FFE00000 = 1,            // BANKSEL.BANKSWP is 111
} flash_bank_t;

```

```

/* R_FLASH_Control 関数の FLASH_CMD_ACCESSWINDOW_SET/GET コマンドにおいて使用する定義 */
typedef struct _flash_access_window_config
{
    uint32_t start_addr;           // アクセスウィンドウの開始アドレス
    uint32_t end_addr;           // アクセスウィンドウの終了アドレス
} flash_access_window_config_t;

```

```

/* R_FLASH_Control 関数の FLASH_CMD_LOCKBIT_READ/WRITE コマンドにおいて使用する定義 */
typedef struct _flash_lockbit_config
{
    flash_block_address_t block_start_address; // 開始アドレス*1
    flash_res_t result; // ロックビット情報の取得結果*2
    uint32_t num_blocks; // ロックビットを設定するブロック数*3
} flash_lockbit_config_t;

```

*1 flash_block_address_t は使用する MCU によって定義の内容が異なります。

*2 FLASH_CMD_LOCKBIT_READ コマンドを使用する場合に使用します。

*3 FLASH_CMD_LOCKBIT_WRITE コマンドを使用する場合に使用します。

```

/* 無効化するキャッシュサイズの指定に使用する定義 */
typedef enum _flash_no_cache_size
{
    FLASH_NON_CACHED_16_BYTES = 0x10, // 16 バイト
    FLASH_NON_CACHED_32_BYTES = 0x20, // 32 バイト
    FLASH_NON_CACHED_64_BYTES = 0x40, // 64 バイト
    FLASH_NON_CACHED_128_BYTES = 0x80, // 128 バイト
    FLASH_NON_CACHED_256_BYTES = 0x100, // 256 バイト
    FLASH_NON_CACHED_512_BYTES = 0x200, // 512 バイト
    FLASH_NON_CACHED_1_KBYTE = 0x400, // 1K バイト
    FLASH_NON_CACHED_2_KBYTES = 0x800, // 2K バイト
    FLASH_NON_CACHED_4_KBYTES = 0x1000, // 4K バイト
    FLASH_NON_CACHED_8_KBYTES = 0x2000, // 8K バイト
    FLASH_NON_CACHED_16_KBYTES = 0x4000, // 16K バイト
    FLASH_NON_CACHED_32_KBYTES = 0x8000, // 32K バイト
    FLASH_NON_CACHED_64_KBYTES = 0x10000, // 64K バイト
    FLASH_NON_CACHED_128_KBYTES = 0x20000, // 128K バイト
    FLASH_NON_CACHED_256_KBYTES = 0x40000, // 256K バイト
    FLASH_NON_CACHED_512_KBYTES = 0x80000, // 512K バイト
    FLASH_NON_CACHED_1_MBYTE = 0x100000, // 1M バイト
    FLASH_NON_CACHED_2_MBYTE = 0x200000 // 2M バイト
} flash_non_cached_size_t;

```

```

/* R_FLASH_Control 関数の FLASH_CMD_SET_NON_CACHED_RANGE0/RANGE1、
FLASH_CMD_GET_NON_CACHED_RANGE0/RANGE1 コマンドにおいて使用する定義 */
typedef struct _flash_non_cached
{
    uint32_t type_mask; // 無効化するキャッシュのタイプ
    uint32_t start_addr; // 無効化するキャッシュの開始アドレス
    flash_non_cached_size_t size; // 無効化するキャッシュのサイズ
} flash_non_cached_t;

```

2.9.2 フラッシュメモリの機能や容量の違いによって異なる引数の定義

フラッシュメモリの機能や容量の違いによって、引数の定義の内容が異なります。

引数の定義の内容が異なる例として、MCU が RX231、RX64M、RX72M の場合を以下に示します。

ファイル名 : r_flash_rx¥src¥targets¥rx231¥r_flash_rx231.h

```
/* フラッシュメモリのブロック数、ブロックサイズ、最小プログラムサイズ、ブロック番号、アドレス等に関する定義 */

~中略~

#define FLASH_NUM_BLOCKS_DF          (8)
#define FLASH_DF_MIN_PGM_SIZE        (1)
#define FLASH_CF_MIN_PGM_SIZE        (8)

#define FLASH_CF_BLOCK_SIZE           (2048)
#define FLASH_DF_BLOCK_SIZE           (1024)
#define FLASH_DF_FULL_SIZE            (FLASH_NUM_BLOCKS_DF*FLASH_DF_BLOCK_SIZE)
#define FLASH_DF_FULL_PGM_SIZE        (FLASH_DF_FULL_SIZE-FLASH_DF_MIN_PGM_SIZE)
#define FLASH_DF_BLOCK_INVALID_ADDR   (FLASH_DF_BLOCK_INVALID-1)
#define FLASH_DF_HIGHEST_VALID_BLOCK  (FLASH_DF_BLOCK_INVALID-FLASH_DF_BLOCK_SIZE)

#define FLASH_NUM_BLOCKS_CF           (MCU_ROM_SIZE_BYTES / FLASH_CF_BLOCK_SIZE)
#define FLASH_CF_FULL_SIZE            (FLASH_NUM_BLOCKS_CF*FLASH_CF_BLOCK_SIZE)
#define FLASH_CF_LOWEST_VALID_BLOCK   (FLASH_CF_BLOCK_INVALID + 1)
#define FLASH_CF_LAST_VALID_ADDR      (FLASH_CF_LOWEST_VALID_BLOCK)

~中略~

typedef enum _flash_block_address
{
    FLASH_CF_BLOCK_END      = 0xFFFFFFFF, /* Top of the CS */
    FLASH_CF_BLOCK_0        = 0xFFFFF800, /* 2KB: 0xFFFFF800 - 0xFFFFFFFF */

~中略~

    FLASH_CF_BLOCK_255      = 0xFFF80000, /* 2KB: 0xFFF80000 - 0xFFF807FF */
    FLASH_CF_BLOCK_INVALID = (FLASH_CF_BLOCK_255 - 1),
#endif

~中略~

    FLASH_DF_BLOCK_0        = 0x00100000, /* 1KB: 0x00100000 - 0x001003ff */

~中略~

    FLASH_DF_BLOCK_7        = 0x00101C00, /* 1KB: 0x00101C00 - 0x00101fff */
    FLASH_DF_BLOCK_INVALID = 0x00102000 /* 1KB: Can't write beyond 0x00101fff */
} flash_block_address_t;

~中略~
```

ファイル名 : r_flash_rx¥src¥targets¥rx64m¥r_flash_rx64m.h

/* フラッシュメモリのブロック数、ブロックサイズ、最小プログラムサイズ、ブロック番号、アドレス等に関する定義 */

～中略～

```
#if (MCU_CFG_PART_MEMORY_SIZE == 0x15 )
#define FLASH_NUM_BLOCKS_CF (134)
#elif (MCU_CFG_PART_MEMORY_SIZE == 0x13 )
#define FLASH_NUM_BLOCKS_CF (102)
#elif (MCU_CFG_PART_MEMORY_SIZE == 0x10 )
#define FLASH_NUM_BLOCKS_CF (86)
#elif (MCU_CFG_PART_MEMORY_SIZE == 0xF )
#define FLASH_NUM_BLOCKS_CF (70)
#endif

#define FLASH_NUM_BLOCKS_DF (1024)
#define FLASH_DF_MIN_PGM_SIZE (4)
#define FLASH_CF_MIN_PGM_SIZE (256)

#define FLASH_CF_SMALL_BLOCK_SIZE (8192)
#define FLASH_CF_MEDIUM_BLOCK_SIZE (32768)
#define FLASH_DF_BLOCK_SIZE (64)
#define FLASH_DF_HIGHEST_VALID_BLOCK (FLASH_DF_BLOCK_INVALID - FLASH_DF_BLOCK_SIZE)
```

～中略～

```
typedef enum _flash_block_address
{
    FLASH_CF_BLOCK_END = 0xFFFFFFFF, /* End of Code Flash Area */
    FLASH_CF_BLOCK_0 = 0xFFFFE000, /* 8KB: 0xFFFFE000 - 0xFFFFFFFF */

```

～中略～

```
    FLASH_CF_BLOCK_133 = 0xFFC00000, /* 32KB: 0xFFC00000 - 0xFFC07FFF */
    FLASH_CF_BLOCK_INVALID = (FLASH_CF_BLOCK_133 - 1), /* 0x15 parts 4M ROM */
}
#endif
```

～中略～

```
    FLASH_DF_BLOCK_0 = 0x00100000, /* 64B: 0x00100000 - 0x0010003F */

```

～中略～

```
    FLASH_DF_BLOCK_1023 = 0x0010FFC0, /* 64B: 0x0010FFC0 - 0x0010FFFF */
    FLASH_DF_BLOCK_INVALID = 0x00110000 /* Block 1023 + 64 bytes */
} flash_block_address_t;
```

～中略～

ファイル名 : r_flash_rx¥src¥targets¥rx72m¥r_flash_rx72m.h

/* フラッシュメモリのブロック数、ブロックサイズ、最小プログラムサイズ、ブロック番号、アドレス等に関する定義 */

～中略～

```
#if (MCU_CFG_PART_MEMORY_SIZE == 0xD)
    #if FLASH_IN_DUAL_BANK_MODE
        #define FLASH_NUM_BLOCKS_CF (30+8)    // 1 Mb per bank dual mode
    #else
        #define FLASH_NUM_BLOCKS_CF (62+8)    // 2 Mb linear mode
    #endif
#elif (MCU_CFG_PART_MEMORY_SIZE == 0x17)
    #if FLASH_IN_DUAL_BANK_MODE
        #define FLASH_NUM_BLOCKS_CF (62+8)    // 2 Mb per bank dual mode
    #else
        #define FLASH_NUM_BLOCKS_CF (126+8)   // 4 Mb linear mode
    #endif
#endif

#define FLASH_NUM_BLOCKS_DF          (512)
#define FLASH_DF_MIN_PGM_SIZE        (4)
#define FLASH_CF_MIN_PGM_SIZE        (128)

#define FLASH_CF_SMALL_BLOCK_SIZE    (8192)
#define FLASH_CF_MEDIUM_BLOCK_SIZE   (32768)
#define FLASH_CF_LO_BANK_SMALL_BLOCK_ADDR (FLASH_CF_BLOCK_77)
#define FLASH_CF_LOWEST_VALID_BLOCK  (FLASH_CF_BLOCK_INVALID + 1)
#define FLASH_DF_BLOCK_SIZE          (64)
#define FLASH_DF_HIGHEST_VALID_BLOCK (FLASH_DF_BLOCK_INVALID - FLASH_DF_BLOCK_SIZE)
```

～中略～

～中略～

```
typedef enum _flash_block_address
{
#ifdef FLASH_IN_DUAL_BANK_MODE
    FLASH_CF_BLOCK_END      = 0xFFFFFFFF, /* End of Code Flash Area */
    FLASH_CF_BLOCK_0        = 0xFFFFE000, /* 8KB: 0xFFFFE000 - 0xFFFFFFFF */

```

～中略～

```
    FLASH_CF_BLOCK_69      = 0xFFE00000, /* 32KB: 0xFFE00000 - 0xFFE07FFF */
#if MCU_CFG_PART_MEMORY_SIZE == 0x0D /* 'D' parts 2 Mb ROM */
    FLASH_CF_BLOCK_INVALID = (FLASH_CF_BLOCK_69 - 1),
#else
    FLASH_CF_BLOCK_70      = 0xFFDF8000, /* 32KB: 0xFFDF8000 - 0xFFDFFFFF */

```

～中略～

```
    FLASH_CF_BLOCK_133     = 0xFFC00000, /* 32KB: 0xFFC00000 - 0xFFC07FFF */
    FLASH_CF_BLOCK_INVALID = (FLASH_CF_BLOCK_133 - 1), /* 'N' parts 4 Mb ROM */
#endif // > 2M

```

```
#else /* DUAL MODE */
    FLASH_CF_BLOCK_END      = 0xFFFFFFFF, /* End of Code Flash Area */
    FLASH_CF_HI_BANK_HI_ADDR = FLASH_CF_BLOCK_END,
    FLASH_CF_BLOCK_0        = 0xFFFFE000, /* 8KB: 0xFFFFE000 - 0xFFFFFFFF */

```

～中略～

```
    FLASH_CF_BLOCK_69      = 0xFFE00000, /* 32KB: 0xFFE00000 - 0xFFE07FFF */
    FLASH_CF_HI_BANK_LO_ADDR = FLASH_CF_BLOCK_69,
#endif
    FLASH_CF_LO_BANK_HI_ADDR = 0xFFDFFFFF, /* START OF NEXT BANK */

    FLASH_CF_BLOCK_70      = 0xFFDFE000, /* 8KB: 0xFFDFE000 - 0xFFDFFFFF */

```

～中略～

```
    FLASH_CF_BLOCK_139     = 0xFFC00000, /* 32KB: 0xFFC00000 - 0xFFC07FFF */
    FLASH_CF_LO_BANK_LO_ADDR = FLASH_CF_BLOCK_139,
    FLASH_CF_BLOCK_INVALID = (FLASH_CF_BLOCK_139 - 1),
#endif // 32 blocks for 4M only
#endif // DUAL MODE

```

```
    FLASH_DF_BLOCK_0        = 0x00100000, /* 64B: 0x00100000 - 0x0010003F */

```

～中略～

```
    FLASH_DF_BLOCK_511     = 0x00107FC0, /* 64B: 0x00107FC0 - 0x00107FFF */
    FLASH_DF_BLOCK_INVALID = 0x00108000 /* Block 511 + 64 bytes */
} flash_block_address_t;

```

～中略～

これらの定義は本モジュールの API 関数の引数として使用します。実際の使用方法については 3 章の各 API 関数の説明および Example を参照してください。

2.10 戻り値

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

```
/* FLASH FIT モジュールの戻り値の定義 */
typedef enum _flash_err
{
    FLASH_SUCCESS = 0,
    FLASH_ERR_BUSY,
    FLASH_ERR_ACCESSW,
    FLASH_ERR_FAILURE,
    FLASH_ERR_CMD_LOCKED,
    FLASH_ERR_LOCKBIT_SET,
    FLASH_ERR_FREQUENCY,
    FLASH_ERR_BYTES,
    FLASH_ERR_ADDRESS,
    FLASH_ERR_BLOCKS,
    FLASH_ERR_PARAM,
    FLASH_ERR_NULL_PTR,
    FLASH_ERR_UNSUPPORTED,
    FLASH_ERR_SECURITY,
    FLASH_ERR_TIMEOUT,
    FLASH_ERR_ALREADY_OPEN
} flash_err_t;

// フラッシュモジュールはビジー状態
// アクセスウィンドウのエラー
// フラッシュの動作、プログラミング、イレーズ等のエラー
// フラッシュモジュールはコマンドロック状態
// ロックビットに起因するプログラム、イレーズ等のエラー
// 不正な周波数が指定された
// 無効なバイト数が指定された
// 無効なアドレス、プログラム境界でないアドレスが指定された
// ブロック数を指定する引数が無効
// 不正なパラメータが指定された
// NULL が指定された
// サポートされていないコマンドが指定された
// アクセスウィンドウの保護に起因するエラー
// タイムアウト発生
// Close () を呼び出さず、Open () を 2 回呼び出した。
```


2.11 コールバック関数

本モジュールでは、FRDYI 割り込み、FIFERR 割り込みが発生したタイミングで、ユーザが設定したコールバック関数を呼び出します。

コールバック関数は、「2.9 引数」に記載された構造体メンバ“pcallback”に、ユーザの関数のアドレスを格納することで設定されます。コールバック関数が呼び出されると、表 2.2～表 2.4 に示す定数が格納された変数が引数として渡されます。

引数の型は void ポインタ型で渡されるため、コールバック関数の引数は void 型のポインタ変数としてください。

コールバック関数内部で値を使うときはキャストして値を使用してください。

コールバック関数の実装例については 3.6 章の Example 1 を参照してください。

表 2.2 フラッシュタイプ 1 のコールバック関数の引数一覧(enum flash_interrupt_event_t)

定数定義	説明
FLASH_INT_EVENT_ERASE_COMPLETE	FRDYI 割り込みの割り込み処理から呼ばれ、イレーズの完了を示します。
FLASH_INT_EVENT_WRITE_COMPLETE	FRDYI 割り込みの割り込み処理から呼ばれ、プログラムの完了を示します。
FLASH_INT_EVENT_BLANK	FRDYI 割り込みの割り込み処理から呼ばれ、ブランクチェックの結果がブランク状態であることを示します。
FLASH_INT_EVENT_NOT_BLANK	FRDYI 割り込みの割り込み処理から呼ばれ、ブランクチェックの結果が非ブランク状態であることを示します。
FLASH_INT_EVENT_TOGGLE_STARTUPAREA	FRDYI 割り込みの割り込み処理から呼ばれ、スタートアップ領域の切り替え完了を示します。
FLASH_INT_EVENT_SET_ACCESSWINDOW	FRDYI 割り込みの割り込み処理から呼ばれ、アクセスウィンドウの設定完了を示します。
FLASH_INT_EVENT_ERR_FAILURE	FRDYI 割り込みの割り込み処理から呼ばれ、プログラムやイレーズのエラーを示します。

表 2.3 フラッシュタイプ 3 のコールバック関数の引数一覧(enum flash_interrupt_event_t)

定数定義	説明
FLASH_INT_EVENT_ERASE_COMPLETE	FRDYI 割り込みの割り込み処理から呼ばれ、イレーズの完了を示します。
FLASH_INT_EVENT_WRITE_COMPLETE	FRDYI 割り込みの割り込み処理から呼ばれ、プログラムの完了を示します。
FLASH_INT_EVENT_BLANK*	FRDYI 割り込みの割り込み処理から呼ばれ、ブランクチェックの結果ブランク状態であることを示します。
FLASH_INT_EVENT_NOT_BLANK* ¹	FRDYI 割り込みの割り込み処理から呼ばれ、ブランクチェックの結果非ブランク状態であることを示します。
FLASH_INT_EVENT_LOCKBIT_WRITTEN	FRDYI 割り込みの割り込み処理から呼ばれ、ロックビットの設定を示します。
FLASH_INT_EVENT_LOCKBIT_PROTECTED	FRDYI 割り込みの割り込み処理から呼ばれ、ロックビットのプロテクトが有効であることを示します。
FLASH_INT_EVENT_LOCKBIT_NON_PROTECTED	FRDYI 割り込みの割り込み処理から呼ばれ、ロックビットのプロテクトが無効であることを示します。
FLASH_INT_EVENT_ERR_DF_ACCESS	FIFERR 割り込みの割り込み処理から呼ばれ、データフラッシュメモリのアクセス違反を示します。
FLASH_INT_EVENT_ERR_CF_ACCESS	FIFERR 割り込みの割り込み処理から呼ばれ、コードフラッシュメモリのアクセス違反を示します。
FLASH_INT_EVENT_ERR_CMD_LOCKED	FIFERR 割り込みの割り込み処理から呼ばれ、コマンドがロック状態であることを示します。
FLASH_INT_EVENT_ERR_LOCKBIT_SET	FIFERR 割り込みの割り込み処理から呼ばれ、ロックビットで保護されている領域に対するエラーを示します。
FLASH_INT_EVENT_ERR_FAILURE	FIFERR 割り込みの割り込み処理から呼ばれ、プログラムやイレーズ中のエラーを示します。

*¹ ブランクチェックの対象はデータフラッシュメモリのみ。

表 2.4 フラッシュタイプ 4 のコールバック関数の引数一覧(enum flash_interrupt_event_t)

定数定義	説明
FLASH_INT_EVENT_ERASE_COMPLETE	FRDYI 割り込みの割り込み処理から呼ばれ、イレーズの完了を示します。
FLASH_INT_EVENT_WRITE_COMPLETE	FRDYI 割り込みの割り込み処理から呼ばれ、プログラムの完了を示します。
FLASH_INT_EVENT_BLANK ^{*1}	FRDYI 割り込みの割り込み処理から呼ばれ、ブランクチェックの結果ブランク状態であることを示します。
FLASH_INT_EVENT_NOT_BLANK ^{*1}	FRDYI 割り込みの割り込み処理から呼ばれ、ブランクチェックの結果非ブランク状態であることを示します。
FLASH_INT_EVENT_TOGGLE_STARTUPAREA	FRDYI 割り込みの割り込み処理から呼ばれ、スタートアップ領域の切り替え完了を示します。
FLASH_INT_EVENT_SET_ACCESSWINDOW	FRDYI 割り込みの割り込み処理から呼ばれ、アクセスウィンドウの設定完了を示します。
FLASH_INT_EVENT_TOGGLE_BANK	FRDYI 割り込みの割り込み処理から呼ばれ、起動バンクの切り替え完了を示します。
FLASH_INT_EVENT_ERR_DF_ACCESS	FIFERR 割り込みの割り込み処理から呼ばれ、データフラッシュメモリのアクセス違反を示します。
FLASH_INT_EVENT_ERR_CF_ACCESS	FIFERR 割り込みの割り込み処理から呼ばれ、コードフラッシュメモリのアクセス違反を示します。
FLASH_INT_EVENT_ERR_SECURITY	FIFERR 割り込みの割り込み処理から呼ばれ、アクセスウィンドウの書き込み保護領域に対する書き換えを示します。
FLASH_INT_EVENT_ERR_CMD_LOCKED	FIFERR 割り込みの割り込み処理から呼ばれ、コマンドがロック状態であることを示します。
FLASH_INT_EVENT_ERR_FAILURE	FIFERR 割り込みの割り込み処理から呼ばれ、プログラムやイレーズのエラーを示します。

^{*1} ブランクチェックの対象はデータフラッシュメモリのみ。

2.12 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、スマート・コンフィグレータを使用した(1)、(3)、(5)の追加方法を推奨しています。ただし、スマート・コンフィグレータは、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

- (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) e² studio 上で FIT コンフィグレータを使用して FIT モジュールを追加する場合
e² studio の FIT コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ 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
CS+上で、スタンドアロン版スマート・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: CS+編 (R20AN0470)」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。
- (5) IAREW 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
スタンドアロン版スマート・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: IAREW 編 (R20AN0535)」を参照してください。

2.13 ブロッキングモード、ノンブロッキングモード

本モジュールの API 関数はブロッキングモードとノンブロッキングモードで動作します。

ブロッキングモードは、API 関数のフラッシュメモリに対する処理が完了するまで復帰しません。

ノンブロッキングモードは、API 関数のフラッシュメモリに対する処理の完了を待たずに復帰します。

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 と FLASH_CFG_CODE_FLASH_BGO は同じ値に設定してください。

- FLASH_CFG_DATA_FLASH_BGO: 0
- FLASH_CFG_CODE_FLASH_BGO: 0

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 と FLASH_CFG_CODE_FLASH_BGO は同じ値に設定してください。

- FLASH_CFG_DATA_FLASH_BGO: 1
- FLASH_CFG_CODE_FLASH_BGO: 1

ユーザは、フラッシュメモリに対する処理が完了するまで、フラッシュメモリの領域に対してはアクセスしないでください。アクセスした場合、フラッシュシーケンサはエラー状態となり、正しく処理することができません。

フラッシュメモリに対する処理の結果はコールバック関数を介して通知されます。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 3.6 章を参照ください。)

コールバック関数を介して処理の結果を通知する API 関数を表 2.5 示します。

表 2.5 コールバック関数を介して処理結果を通知する API 関数

API 関数	コールバック関数を介した処理結果の通知
R_FLASH_Open()、R_FLASH_Close()、R_FLASH_GetVersion()	通知しない
R_FLASH_Erase()、R_FLASH_BLankCheck()、R_FLASH_Write()	通知する
R_FLASH_Control()	以下のコマンドに関しては通知する ・ FLASH_CMD_SWAPFLAG_TOGGLE ・ FLASH_CMD_ACCESSWINDOW_SET ・ FLASH_CMD_LOCKBIT_READ ・ FLASH_CMD_LOCKBIT_WRITE ・ FLASH_CMD_BANK_TOGGLE

フラッシュメモリに対する処理が完了すると FRDYI 割り込みや FIFERR 割り込みが発生します。それぞれの割り込み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。コールバック関数の詳細については 2.11 章を参照ください。

2.14 アクセスウィンドウ、ロックビットによる領域の保護

各 MCU のフラッシュメモリはコードフラッシュメモリ上の領域が意図せずに書き換られてしまうことを防止するため、アクセスウィンドウ、もしくはロックビットを使用して領域を保護することができます。本モジュールの API 関数はそれらの機能をサポートしています。

2.14.1 アクセスウィンドウによる領域の保護

フラッシュタイプ 1、4 の製品についてはアクセスウィンドウを使用して領域を保護することができます。

アクセスウィンドウの設定はアクセスウィンドウの対象領域として扱うブロックの開始アドレスと終了アドレスを指定します。

アクセスウィンドウの対象領域として扱うブロックの開始アドレスから終了アドレスまでの領域が書き換えのできる領域となります。その他の領域が書き換えのできない保護された領域となりますのでご注意ください。

なお、出荷時にはアクセスウィンドウは設定されていないため、全ての領域が書き換え可能な領域となっています。

アクセスウィンドウは `R_FLASH_Control()` を使用して設定します。詳細は 3.6 章を参照してください。

2.14.2 ロックビットによる領域の保護

フラッシュタイプ 3 の製品についてはロックビットを使用して領域を保護することができます。

ロックビットの設定はロックビットの対象領域として扱うブロックの先頭アドレスとブロック数、およびロックビットによるプロテクトの有効／無効を指定します。

ロックビットの対象領域として扱うブロックの先頭アドレスから指定されたブロック数分の領域が書き換えのできない保護された領域となります。その他の領域は書き換えのできる領域となりますのでご注意ください。

なお、出荷時にはロックビットは設定されていないため、全ての領域が書き換え可能な領域となっています。

ロックビットは `R_FLASH_Control()` を使用して設定します。詳細は 3.6 章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、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 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 フラッシュメモリの書き換え

フラッシュメモリの書き換えに必要なコードは図 2.1(左図)に示すようにコードフラッシュメモリ上に配置されます。図 2.1(右図)に示すようにコードフラッシュメモリ上のコードを実行することで、書き換え対象のフラッシュメモリの領域を書き換えることができます。

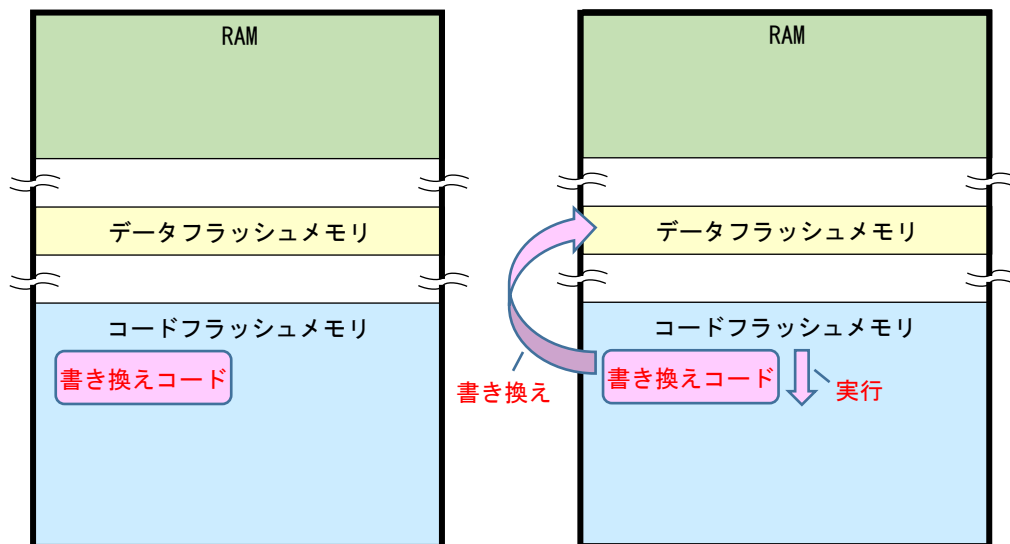


図 2.1 フラッシュメモリの書き換えに必要なコードの配置と書き換え動作

しかしながら、図 2.2 に示すようにフラッシュメモリの書き換えに必要なコードが配置されている領域と同じ領域を書き換えることはできません。

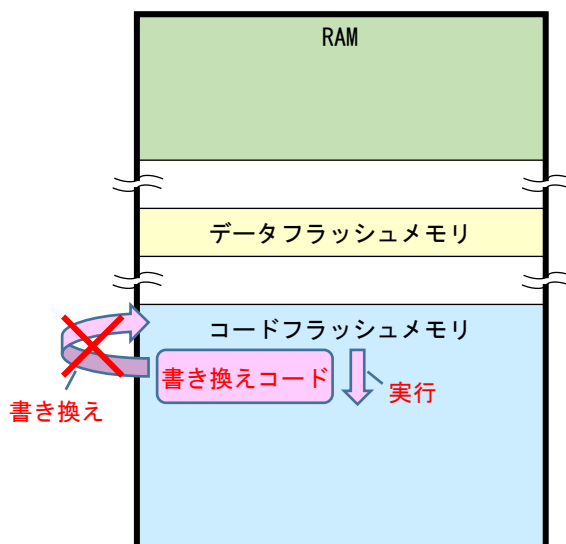


図 2.2 フラッシュメモリの書き換えに必要なコードが配置されている領域と同じ領域の書き換え

2.16.1 ~ 2.16.3 章ではコードフラッシュメモリを書き換える方法を示します。

2.16.1 RAM からコードを実行してコードフラッシュメモリを書き換える

図 2.3 に示すようにフラッシュメモリの書き換えに必要なコードを RAM 上にコピーした後、RAM 上にコピーされたコードを実行することにより、コードフラッシュメモリ上の領域を書き換えることができます。

*1*2

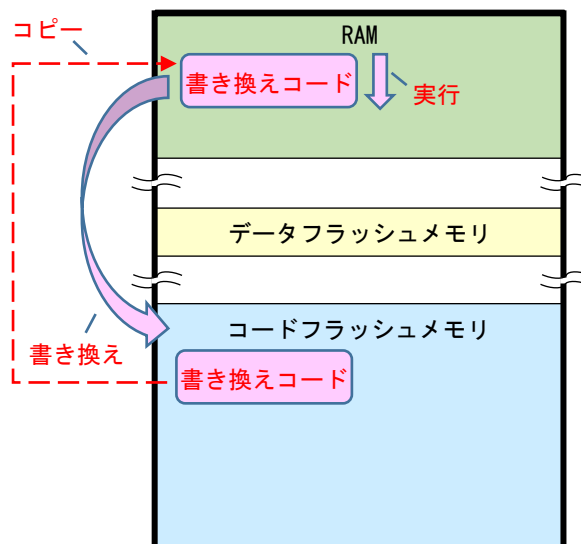


図 2.3 RAM からコードを実行してコードフラッシュメモリを書き換える

本モジュールのコンフィギュレーションオプションは以下のように設定してください。

- FLASH_CFG_CODE_FLASH_ENABLE: 1
- FLASH_CFG_CODE_FLASH_RUN_FROM_ROM: 0

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。

*1 フラッシュメモリの書き換えに必要なコードは本モジュールの R_FLASH_Open()関数によって、RAM 上にコピーされます。

*2 コードフラッシュメモリが複数の領域に分かれている製品については RAM を使用せずにコードフラッシュメモリ上の領域を書き換えることができます。詳細は 2.16.2 章を参照してください。

2.16.2 コードフラッシュメモリからコードを実行してコードフラッシュメモリを書き換える

コードフラッシュメモリからコードを実行してコードフラッシュメモリを書き換えることが可能な製品を6に示します。これらの製品ではコードフラッシュメモリが複数の領域に分かれています。

表 2.6 コードフラッシュメモリが複数の領域に分かれている製品

フラッシュタイプ	コードフラッシュメモリが複数の領域に分かれている製品
3	RX64M ^{*1} 、RX71M ^{*1}
4	RX651 ^{*2} 、RX65N ^{*2} 、RX66N、RX671、RX72M、RX72N

^{*1} コードフラッシュメモリの容量が 2.5M バイト以上の製品

^{*2} コードフラッシュメモリの容量が 1.5M バイト以上の製品

コードフラッシュメモリの領域については MCU によって搭載されている容量が異なります。このため、コードフラッシュメモリの領域のサイズや境界は MCU に依存します。詳細は各ユーザーズマニュアルのハードウェア編を参照してください。

図 2.4 に示すようにコードフラッシュメモリが複数の領域に分かれている製品の場合、フラッシュメモリの書き換えに必要なコードが配置されている領域とは異なるコードフラッシュメモリ上の領域を書き換えることができます。

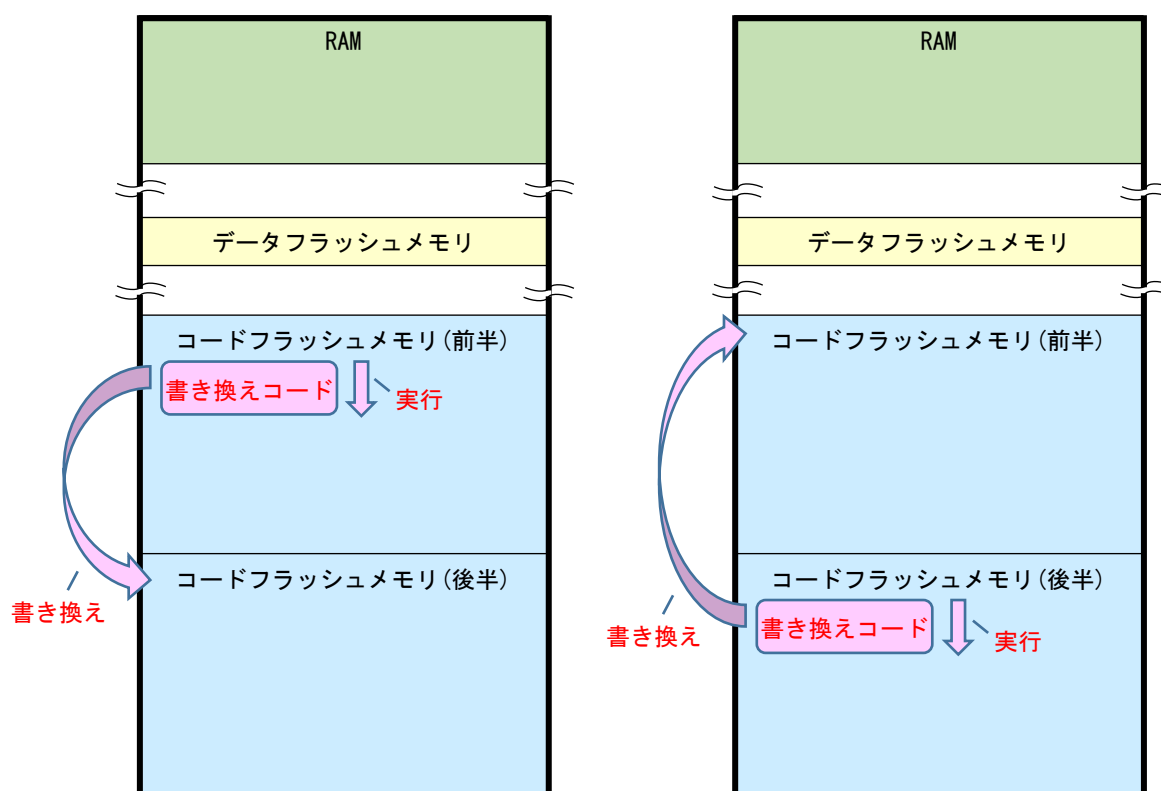


図 2.4 コードフラッシュメモリからコードを実行してコードフラッシュメモリを書き換える

本モジュールのコンフィギュレーションオプションは以下のように設定してください。

- FLASH_CFG_CODE_FLASH_ENABLE: 1
- FLASH_CFG_CODE_FLASH_RUN_FROM_ROM: 1

2.16.3 デュアルバンク機能と組み合わせてコードフラッシュメモリを書き換える

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 の製品はデュアルバンク機能を搭載しています。

デュアルバンク機能はバンクモード切り替え機能とバンク選択機能によりユーザプログラムを実行しながらプログラムを更新出来る機能となります。

バンクモード切り替え機能とはコードフラッシュメモリのユーザ領域を 1 つの領域として扱うリニアモードと、2 つのバンク領域として扱うデュアルモードを切り替える機能となります。

バンク選択機能とはデュアルモード選択時にプログラムを起動するバンク領域を選択する機能となります。

図 2.5(左図)に示すようにフラッシュメモリの書き換えに必要なコードが配置されているバンク 0 側の領域とは異なるバンク 1 側の領域を書き換えることができます。また、バンクを切り替えることにより、図 2.5 (右図)に示すようにフラッシュメモリの書き換えに必要なコードが配置されているバンク 1 側の領域とは異なるバンク 0 側の領域を書き換えることができます。

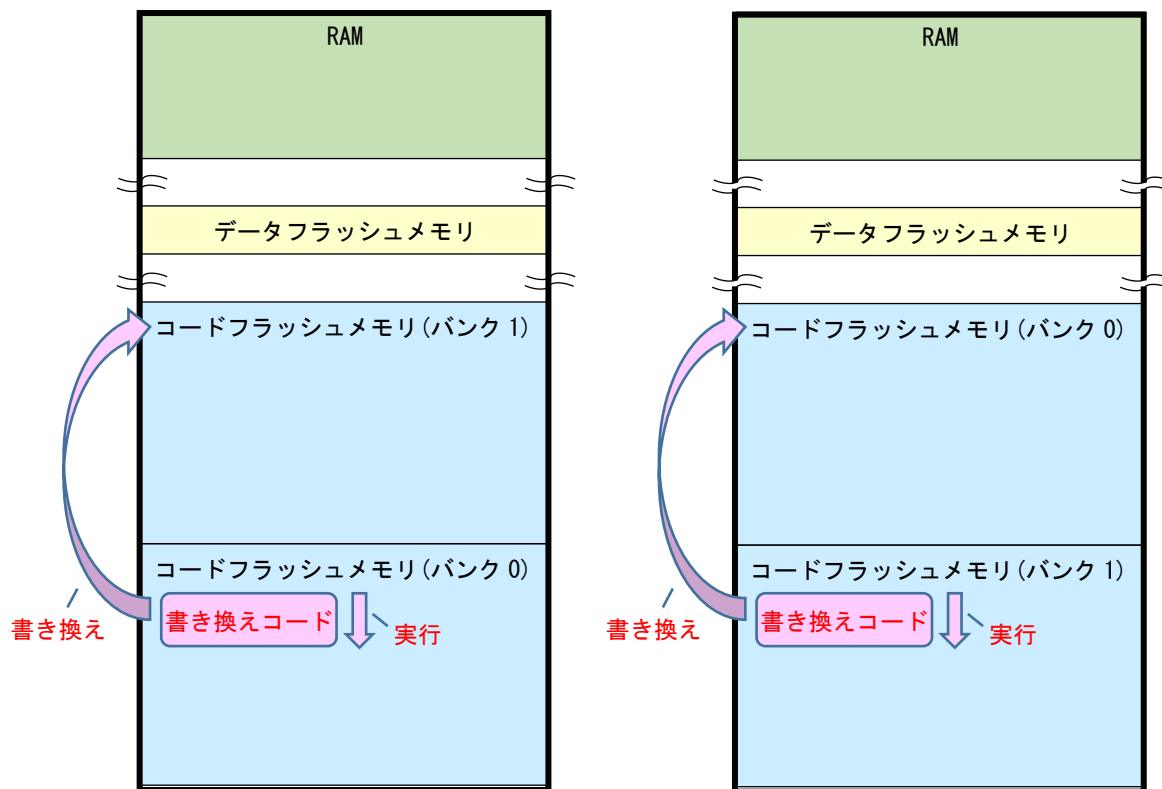


図 2.5 デュアルバンク機能と組み合わせてコードフラッシュメモリを書き換える

デュアルバンク機能を使用する場合、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。デュアルモードで使用する際は"0"に設定してください。

デュアルモードでは、R_FLASH_Control()関数の第一引数に FLASH_CMD_BANK_TOGGLE コマンドを指定することにより、起動バンクを切り替えることができます。起動バンクの切り替えは、次に MCU がリセットされるまで有効にはなりません。

本モジュールのコンフィギュレーションオプションは以下のように設定してください。

- FLASH_CFG_CODE_FLASH_ENABLE: 1
- FLASH_CFG_CODE_FLASH_RUN_FROM_ROM: 1

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。

3. API 関数

3.1 R_FLASH_Open()

本 API 関数はフラッシュ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

<code>FLASH_SUCCESS</code>	<i>/* 正常に初期化されました。 */</i>
<code>FLASH_ERR_BUSY</code>	<i>/* フラッシュメモリ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。 */</i>
<code>FLASH_ERR_ALREADY_OPEN</code>	<i>/* 既にオープン済です、R_FLASH_Close() を実行ください。 */</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

本 API 関数では以下の処理を行います。

1. フラッシュメモリの書き換えに必要なコードの準備

フラッシュメモリの書き換えに必要なコードはコンフィギュレーションオプションの設定に応じて、表 3.1 のように配置されます。

表 3.1 コンフィギュレーションオプションの設定とコードの配置の関係

コンフィギュレーションオプション	設定	コードの配置
FLASH_CFG_CODE_FLASH_ENABLE	0	フラッシュメモリを扱うコードはコードフラッシュメモリ上に配置されます。ただし、コードフラッシュメモリを扱うコードは含まれません。
FLASH_CFG_CODE_FLASH_ENABLE	1	フラッシュメモリを扱うコードは RAM 上にコピーされます。 ^{*1}
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	0	
BSP_CFG_CODE_FLASH_BANK_MODE	0	
FLASH_CFG_CODE_FLASH_ENABLE	1	フラッシュメモリを扱うコードはコードフラッシュメモリ上に配置されます。
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	1	
BSP_CFG_CODE_FLASH_BANK_MODE	0	
FLASH_CFG_CODE_FLASH_ENABLE	1	フラッシュメモリを扱うコードはコードフラッシュメモリ上に配置されます。 デュアルバンク機能を扱うコードは RAM 上にコピーされます。 ^{*1}
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	1	
BSP_CFG_CODE_FLASH_BANK_MODE	1	

^{*1} 割り込みベクタテーブルや割り込み処理を再配置する機能は本 API 関数には含まれません。

2. フラッシュシーケンサの初期設定

フラッシュタイプ 3、4 の製品についてはフラッシュシーケンサに対する設定として、BSP のコンフィギュレーションオプション(BSP_FCLK_HZ)の設定値をフラッシュシーケンサ処理クロック通知レジスタ(FPCKAR)に設定します。

さらに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 の製品については同様の設定をデータフラッシュメモリアクセス周波数設定レジスタ(EEPFCLK)に対して行います。

また、RX64M、RX71M についてはフラッシュシーケンサを使用するために必要な FCU ファームウェアを専用の RAM(FCURAM)にコピーします。

3. 割り込みの初期設定

2.4 章に記載している割り込みを禁止にします。

Reentrant

- 不可

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

3.2 R_FLASH_Close()

本 API 関数はフラッシュモジュールの終了処理を行う関数です。

Format

```
flash_err_t R_FLASH_Close(void)
```

Parameters

なし

Return Values

<code>FLASH_SUCCESS</code>	<i>/* 正常にフラッシュモジュールの終了処理は完了しました。*/</i>
<code>FLASH_ERR_BUSY</code>	<i>/* フラッシュメモリ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。*/</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

本 API 関数はフラッシュモジュールの終了処理として、2.4 章に記載している割り込みを禁止にし、本モジュールを初期化されていない状態に設定します。

Reentrant

- 不可

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

3.3 R_FLASH_Erase()

本 API 関数はコードフラッシュメモリまたはデータフラッシュメモリの指定したブロックのイレーズを行う関数です。

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先頭アドレスを指定します。

"flash_block_address_t"はブロックの先頭アドレスとブロック番号を紐づけた定義です。

"flash_block_address_t"は"r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h"に定義されています。

num_blocks

イレーズ対象のブロック数を指定します。

RX111、RX113、RX130 の製品の場合、"block_start_address"と"num_blocks"で指定された領域が、256K バイトの境界をまたがないようにしてください。

Return Values

FLASH_SUCCESS	<i>/* 正常にイレーズ処理は完了しました。ノンブロッキングモードの場合はイレーズ処理が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>/* フラッシュメモリ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>/* イレーズ処理に失敗しました。ノンブロッキングモードの場合はコールバック関数が登録されていません。*/</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

コードフラッシュメモリおよびデータフラッシュメモリをブロック単位でイレーズします。
 ブロックサイズは表 3.2 に示すように MCU グループによって異なります。

表 3.2 MCU グループとブロックサイズ

MCU グループ	コードフラッシュメモリ	データフラッシュメモリ ^{*3}
RX110	1K バイト ^{*1}	— ^{*4}
RX111	1K バイト ^{*1}	1K バイト
RX113	1K バイト ^{*1}	1K バイト
RX130	1K バイト ^{*1}	1K バイト
RX13T	1K バイト ^{*1}	1K バイト
RX230、RX231	2K バイト ^{*1}	1K バイト
RX23E-A	2K バイト ^{*1}	1K バイト
RX23T	2K バイト ^{*1}	— ^{*4}
RX23W	2K バイト ^{*1}	1K バイト
RX24T	2K バイト ^{*1}	1K バイト
RX24U	2K バイト ^{*1}	1K バイト
RX64M	8K バイト、32K バイト ^{*2}	64 バイト
RX65N、RX651	8K バイト、32K バイト ^{*2}	64 バイト ^{*5}
RX66N	8K バイト、32K バイト ^{*2}	64 バイト
RX66T	8K バイト、32K バイト ^{*2}	64 バイト
RX671	8K バイト、32K バイト ^{*2}	64 バイト
RX71M	8K バイト、32K バイト ^{*2}	64 バイト
RX72M	8K バイト、32K バイト ^{*2}	64 バイト
RX72N	8K バイト、32K バイト ^{*2}	64 バイト
RX72T	8K バイト、32K バイト ^{*2}	64 バイト

^{*1} MCU 毎の定義ファイル("r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h")に FLASH_CF_BLOCK_SIZE として定義されています。

^{*2} 8K バイトのブロックと 32K バイトのブロックが存在します。
 MCU 毎の定義ファイル("r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h")に 8K バイトのブロックは FLASH_CF_SMALL_BLOCK_SIZE、32K バイトのブロックは FLASH_CF_MEDIUM_BLOCK_SIZE として定義されています。

^{*3} MCU 毎の定義ファイル("r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h")に FLASH_DF_BLOCK_SIZE として定義されています。

^{*4} データフラッシュメモリは搭載されていません。

^{*5} コードフラッシュメモリの容量が 1M バイト以下の製品の場合、データフラッシュメモリは搭載されていません。

ノンブロッキングモードで本 API 関数が使用された場合、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。

Reentrant

- 不可

Example

第 1 引数はイレーズを開始するブロックアドレスを指定します。

第 2 引数はイレーズを開始するブロックアドレスを起点にイレーズするブロック数を指定します。

複数のブロックを指定したフラッシュメモリのイレーズの例を以下に示します。

データフラッシュメモリとコードフラッシュメモリ、またフラッシュタイプの違いによって、イレーズされるブロックの方向が異なることに注意してください。

```
flash_err_t err;

/* フラッシュタイプ 1、3、4 の製品において共通 */
/* データフラッシュメモリのブロック 5 を起点にブロック番号が大きくなる方向にイレーズされる */
/* 以下の場合、データフラッシュメモリのブロック 5、6 がイレーズされる */
err = R_FLASH_Erase(FLASH_DF_BLOCK_5, 2);

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

/* フラッシュタイプ 1 の製品の場合 */
/* コードフラッシュメモリのブロック 5 を起点にブロック番号が小さくなる方向にイレーズされる */
/* 以下の場合、コードフラッシュメモリのブロック 4、5 がイレーズされる */
err = R_FLASH_Erase(FLASH_CF_BLOCK_5, 2);

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

/* フラッシュタイプ 3、4 の製品の場合 */
/* コードフラッシュメモリのブロック 5 を起点にブロック番号が大きくなる方向にイレーズされる */
/* 以下の場合、コードフラッシュメモリのブロック 5、6 がイレーズされる */
err = R_FLASH_Erase(FLASH_CF_BLOCK_5, 2);

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

Special Notes:

なし

3.4 R_FLASH_BlankCheck()

本 API 関数はコードフラッシュメモリまたはデータフラッシュメモリの指定された領域がブランクかどうかを判定する関数です。

Format

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

Parameters

address

ブランクチェックを行う領域の先頭アドレスを指定します。

本パラメータには対象となるフラッシュメモリ上の領域の最小プログラムサイズの倍数を指定する必要があります。

num_bytes

ブランクチェックを行うバイト数を指定します。

本パラメータには対象となるフラッシュメモリ上の領域の最小プログラムサイズの倍数を指定する必要があります。また、RX111、RX113、RX130 の製品の場合、“address”と“num_bytes”で指定された領域が、256K バイトの境界をまたが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に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ます。

ブランクチェックの結果は以下の内容が格納されます。

- FLASH_RES_BLANK: ブランク状態
- FLASH_RES_NOT_BLANK: 非ブランク状態

ノンブロッキングモードの場合、本パラメータは使用しないため任意の値を指定してください。

Return Values

FLASH_SUCCESS	<i>/* 正常にブランクチェック処理は完了しました。ノンブロッキングモードの場合はブランクチェック処理が開始されたことを意味します。*/</i>
FLASH_ERR_FAILURE	<i>/* ブランクチェック処理に失敗しました。ノンブロッキングモードの場合はコールバック関数が登録されていません。*/</i>
FLASH_ERR_BUSY	<i>/* フラッシュメモリ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_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>/* 無効なアドレスが指定されました。アドレスが*/ /* 最小プログラムサイズの倍数になっていないか、ブランクチェック対象外のフラッシュタイプのアドレスが指定されました。*/</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

ブランクチェックをサポートしている MCU グループを表 3.3 に示します。

表 3.3 ブランクチェックをサポートしている MCU グループ

MCU グループ	コードフラッシュメモリ	データフラッシュメモリ
RX110	●	— ^{*1}
RX111	●	●
RX113	●	●
RX130	●	●
RX13T	●	●
RX230、RX231	●	●
RX23E-A	●	●
RX23T	●	— ^{*1}
RX23W	●	●
RX24T	●	●
RX24U	●	●
RX64M	— ^{*2}	●
RX65N、RX651	— ^{*2}	● ^{*3}
RX66N	— ^{*2}	●
RX66T	— ^{*2}	●
RX671	— ^{*2}	●
RX71M	— ^{*2}	●
RX72M	— ^{*2}	●
RX72N	— ^{*2}	●
RX72T	— ^{*2}	●

^{*1} データフラッシュメモリは搭載されていません。

^{*2} ブランクチェックはサポートしていません。

^{*3} コードフラッシュメモリの容量が 1M バイト以下の製品の場合、データフラッシュメモリは搭載されていません。

本 API 関数の第 1 引数で指定するアドレス、および第 2 引数で指定するバイト数は最小プログラムサイズの倍数となります。最小プログラムサイズは MCU のタイプや対象となるフラッシュメモリの種類によって異なります。詳細は 3.5 章の表 3.4 を参照してください。

本 API 関数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。

Reentrant

- 不可

Example

第 1 引数はブランクチェックを行う先頭のアドレスを指定します。

第 2 引数はブランクチェックを行うバイト数を指定します。

共に最小プログラムサイズの倍数でなければなりません。

```
flash_err_t err;
flash_res_t result;

/* データフラッシュメモリのブロック 0 の最初の 64 バイトのブランクチェックを行う */
err = R_FLASH_BlankCheck((uint32_t)FLASH_DF_BLOCK_0, 64, &result);
if (FLASH_SUCCESS != err)
{
    /*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

3.5 R_FLASH_Write()

本 API 関数はコードフラッシュメモリ、またはデータフラッシュメモリの書き換えを行う関数です。

Format

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

Parameters

src_address

フラッシュメモリに書き込むデータを格納したバッファの先頭アドレスを指定します。

dest_address

書き換え対象となるフラッシュメモリ上の領域の先頭アドレスを指定します。

本パラメータには対象となるフラッシュメモリ上の領域の最小プログラムサイズの倍数を指定する必要があります。

num_bytes

フラッシュメモリに対して書き込みを行うバイト数を指定します。

本パラメータには対象となるフラッシュメモリ上の領域の最小プログラムサイズの倍数を指定する必要があります。

Return Values

FLASH_SUCCESS	<i>/* 正常にプログラム処理は完了しました。ノンブロッキングモードの場合はプログラム処理が開始されたことを意味します。*/</i>
FLASH_ERR_FAILURE	<i>/* フラッシュシーケンサに関連する要因によりプログラム処理に失敗しました。ノンブロッキングモードの場合はコールバック関数が登録されていません。*/</i>
FLASH_ERR_BUSY	<i>/* フラッシュメモリ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_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>/* 指定されたアドレスは無効です。*/</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

フラッシュメモリの書き換え対象の領域は書き込む前にイレーズしておく必要があります。

本 API 関数の第 2 引数で指定するアドレス、および第 3 引数で指定するバイト数は最小プログラムサイズの倍数となります。最小プログラムサイズは MCU や対象となるフラッシュメモリの種類によって表 3.4 のように異なります。

表 3.4 MCU グループと最小プログラムサイズ

MCU グループ	コードフラッシュメモリ ^{*1}	データフラッシュメモリ ^{*2}
RX110	4 バイト	— ^{*3}
RX111	4 バイト	1 バイト
RX113	4 バイト	1 バイト
RX130	4 バイト	1 バイト
RX13T	4 バイト	1 バイト
RX230、RX231	8 バイト	1 バイト
RX23E-A	8 バイト	1 バイト
RX23T	8 バイト	— ^{*3}
RX23W	8 バイト	1 バイト
RX24T	8 バイト	1 バイト
RX24U	8 バイト	1 バイト
RX64M	256 バイト	4 バイト
RX65N、RX651	128 バイト	4 バイト ^{*4}
RX66N	128 バイト	4 バイト
RX66T	256 バイト	4 バイト
RX671	128 バイト	4 バイト
RX71M	256 バイト	4 バイト
RX72M	128 バイト	4 バイト
RX72N	128 バイト	4 バイト
RX72T	256 バイト	4 バイト

^{*1} MCU 毎の定義ファイル("r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h")に FLASH_CF_MIN_PGM_SIZE として定義されています。

^{*2} MCU 毎の定義ファイル("r_flash_rx¥src¥targets¥<mcu>¥r_flash_<mcu>.h")に FLASH_DF_MIN_PGM_SIZE として定義されています。

^{*3} データフラッシュメモリは搭載されていません。

^{*4} コードフラッシュメモリの容量が 1M バイト以下の製品の場合、データフラッシュメモリは搭載されていません。

本 API 関数がノンブロッキングモードで 사용되는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。

Reentrant

- 不可

Example

第2引数はフラッシュメモリ上の書き換え対象のアドレスを指定します。

第3引数はフラッシュメモリ上に書き込むバイト数を指定します。

共に最小プログラムサイズの倍数でなければなりません。

```
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));

if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

なし

3.6 R_FLASH_Control()

本 API 関数はプログラム、イレーズ、ブランクチェック以外の処理を行う関数です。

Format

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

Parameters

cmd

実行するコマンドを指定します。

*pcfg

第 1 引数の実行するコマンドに依存して必要な引数を指定します。実行するコマンドが引数を必要としない場合は NULL を指定してください。

Return Values

FLASH_SUCCESS	<i>/* 正常に完了しました。ノンブロッキングモードの場合は処理が正常に開始されたことを意味します。*/</i>
FLASH_ERR_ADDRESS	<i>/* 指定されたアドレスは無効です。*/</i>
FLASH_ERR_NULL_PTR	<i>/* 第 2 引数が必要とされていますが NULL が指定されています。*/</i>
FLASH_ERR_BUSY	<i>/* フラッシュメモリ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_CMD_LOCKED	<i>/* フラッシュシーケンサがコマンドロック状態にあり、*/ /* 強制終了コマンドを発行し復帰処理を行いました。*/</i>
FLASH_ERR_ACCESSW	<i>/* アクセスウィンドウに関するエラーが発生しました。指定された 領域は不正です。*/</i>
FLASH_ERR_PARAM	<i>/* 無効なパラメータが渡されました */</i>

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

本 API 関数は引数で指定されたコマンドに応じた処理を行います。フラッシュタイプの違いによってサポートしているコマンドは表 3.5 のように異なります。

表 3.5 サポートしているコマンドとフラッシュタイプとの関係

コマンド区分	コマンド	フラッシュタイプ		
		1	3	4
フラッシュタイプ共通				
フラッシュモジュールの API 関数実行状態取得	FLASH_CMD_STATUS_GET	✓	✓	✓
コールバック関数の登録	FLASH_CMD_SET_BGO_CALLBACK	✓	✓	✓
フラッシュシーケンサリセット	FLASH_CMD_RESET	✓	✓	✓
フラッシュシーケンサ使用周波数通知				
フラッシュシーケンサ使用周波数の通知	FLASH_CMD_CONFIG_CLOCK	—	✓	✓
アクセスウィンドウ				
アクセスウィンドウの設定取得	FLASH_CMD_ACCESSWINDOW_GET	✓	—	✓
アクセスウィンドウの設定	FLASH_CMD_ACCESSWINDOW_SET			
スタートアッププログラム保護				
スタートアップ領域の設定取得	FLASH_CMD_SWAPFLAG_GET	✓	—	✓
スタートアップ領域の切り替え	FLASH_CMD_SWAPFLAG_TOGGLE			
スタートアップ領域選択ビットの設定取得	FLASH_CMD_SWAPSTATE_GET			
スタートアップ領域選択ビットの設定	FLASH_CMD_SWAPSTATE_SET			
ロックビット				
ロックビットの設定取得	FLASH_CMD_LOCKBIT_READ	—	✓	—
ロックビットの設定	FLASH_CMD_LOCKBIT_WRITE			
ロックビットの設定有効	FLASH_CMD_LOCKBIT_ENABLE			
ロックビットの設定無効	FLASH_CMD_LOCKBIT_DISABLE			
ROM キャッシュ				
ROM キャッシュの設定有効	FLASH_CMD_ROM_CACHE_ENABLE	✓ ^{*1}	✓ ^{*2}	✓
ROM キャッシュの設定無効	FLASH_CMD_ROM_CACHE_DISABLE			
ROM キャッシュの設定取得	FLASH_CMD_ROM_CACHE_STATUS			
キャッシュ無効化				
ノンキャッシュ領域 0 の設定	FLASH_CMD_SET_NON_CACHED_RANGE0	—	✓ ^{*2}	✓ ^{*3}
ノンキャッシュ領域 1 の設定	FLASH_CMD_SET_NON_CACHED_RANGE1			
ノンキャッシュ領域 0 の設定取得	FLASH_CMD_GET_NON_CACHED_RANGE0			
ノンキャッシュ領域 1 の設定取得	FLASH_CMD_GET_NON_CACHED_RANGE1			
デュアルバンク				
バンクの切り替え	FLASH_CMD_BANK_TOGGLE	—	—	✓
バンクの設定取得	FLASH_CMD_BANK_GET			

^{*1} RX24T、RX24U のみ対象となります。

^{*2} RX66T、RX72T のみ対象となります。

^{*3} RX66N、RX671、RX72M、RX72N のみ対象となります。

フラッシュタイプ毎にサポートしているコマンドの詳細を表 3.6～表 3.8 に示します。

表 3.6 フラッシュタイプ 1 でサポートしているコマンドの詳細

コマンド	内容
FLASH_CMD_STATUS_GET (引数には NULL を設定してください) *使用例は Example 3 に記載	フラッシュメモリに対するフラッシュシーケンサの実行状態を取得します。 本コマンドはフラッシュメモリに対する処理が実行中の場合でも使用可能です。 FLASH_SUCCESS : フラッシュシーケンサは実行されていない FLASH_ERR_BUSY : フラッシュシーケンサは実行されている
FLASH_CMD_SET_BGO_CALLBACK (引数の型 : flash_interrupt_config_t *) *使用例は Example 1、Example 2 に記載	コールバック関数を登録します。本コマンドはノンブロッキングモードで使用する場合に必要です。
FLASH_CMD_RESET (引数には NULL を設定してください)	フラッシュシーケンサをリセットします。 本コマンドはフラッシュメモリに対する処理が実行中の場合でも使用可能です。
FLASH_CMD_ACCESSWINDOW_GET (引数の型 : flash_access_window_config_t *) *使用例は Example 4 に記載	コードフラッシュメモリのアクセスウィンドウの対象領域として扱うブロックの開始アドレスと終了アドレスを取得します。
FLASH_CMD_ACCESSWINDOW_SET (引数の型 : flash_access_window_config_t *) *使用例は Example 5 に記載	コードフラッシュメモリのアクセスウィンドウの対象領域として扱うブロックの開始アドレスと終了アドレスを指定します。 アクセスウィンドウの設定において、開始アドレスは終了アドレスよりも小さい値とする必要があります。 開始アドレスと終了アドレスで指定された範囲以外のブロックに対してはプログラムやイレーズすることができません。 また、開始アドレスと終了アドレスで指定された範囲を複数指定することはできません。 アクセスウィンドウの設定を削除する場合は、開始アドレスと終了アドレスに同じ値を指定します。 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 (引数の型 : uint32_t *) *使用例は Example 6 に記載	スタートアップ領域の設定を取得します。 0 : 代替領域から起動 1 : デフォルト領域から起動

<p>FLASH_CMD_SWAPFLAG_TOGGLE (引数には NULL を設定してください) *使用例は Example 7 に記載</p>	<p>スタートアップ領域を切り替えます。 切り替えられたスタートアップ領域は次のリセットで有効となります。ノンブロッキングモードで使用する場合は、スタートアップ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。 本コマンドはコンフィギュレーションオプションの FLASH_CFG_CODE_FLASH_ENABLE を"1"に設定している状態で使用してください。</p>
<p>FLASH_CMD_SWAPSTATE_GET (引数の型 : uint8_t*) *使用例は Example 8 に記載</p>	<p>スタートアップ領域選択ビット(FISR.SAS)の値を取得します。 FLASH_SAS_EXTRA : スタートアップ領域選択ビットはスタートアップ領域の設定に従う FLASH_SAS_DEFAULT : スタートアップ領域選択ビットはデフォルト領域に設定されている FLASH_SAS_ALTERNATE : スタートアップ領域選択ビットは代替え領域に設定されている</p>
<p>FLASH_CMD_SWAPSTATE_SET (引数の型 : uint8_t*) *使用例は Example 9 に記載</p>	<p>スタートアップ領域選択ビット(FISR.SAS)の値を設定します。 即座に設定されたスタートアップ領域となります。 リセット後の初期値は FLASH_SAS_EXTRA です。 FLASH_SAS_EXTRA : エクストラ領域内のスタートアップ領域の設定に従う FLASH_SAS_DEFAULT : 一時的にスタートアップ領域をデフォルト領域に切り替える FLASH_SAS_ALTERNATE : 一時的にスタートアップ領域を代替え領域に切り替える FLASH_SAS_SWITCH_AREA : スタートアップ領域を切り替える</p>
<p>FLASH_CMD_ROM_CACHE_ENABLE (引数には NULL を設定してください) *使用例は Example 10 に記載</p>	<p>コードフラッシュメモリのキャッシュを有効にします。</p>
<p>FLASH_CMD_ROM_CACHE_DISABLE (引数には NULL を設定してください) *使用例は Example 10 に記載</p>	<p>コードフラッシュメモリのキャッシュを無効にします。 コードフラッシュメモリを書き換える前に呼び出してください。</p>
<p>FLASH_CMD_ROM_CACHE_STATUS (引数の型 : uint8_t*) *使用例は Example 10 に記載</p>	<p>コードフラッシュメモリのキャッシュの状態を取得します。 0 : コードフラッシュメモリのキャッシュは無効 1 : コードフラッシュメモリのキャッシュは有効</p>

表 3.7 フラッシュタイプ 3 でサポートしているコマンドの詳細

コマンド	内容
FLASH_CMD_STATUS_GET (引数には NULL を設定してください) *使用例は Example 3 に記載	フラッシュメモリに対するフラッシュシーケンサの実行状態を取得します。 本コマンドはフラッシュメモリに対する処理が実行中の場合でも使用可能です。 FLASH_SUCCESS : フラッシュシーケンサは実行されていない FLASH_ERR_BUSY : フラッシュシーケンサは実行されている
FLASH_CMD_SET_BGO_CALLBACK (引数の型 : flash_interrupt_config_t *) *使用例は Example 1、Example 2 に記載	コールバック関数を登録します。本コマンドはノンブロッキングモードで使用する場合に必要です。
FLASH_CMD_RESET (引数には NULL を設定してください)	フラッシュシーケンサをリセットします。 本コマンドはフラッシュメモリに対する処理が実行中の場合でも使用可能です。
FLASH_CMD_LOCKBIT_READ (引数の型 : flash_lockbit_config_t *) *使用例は Example 12 に記載	コードフラッシュメモリ上の指定したブロックのロックビットの設定状態を取得します。 ノンブロッキングモードで使用する場合は、ロックビットの設定状態を取得後 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。 ^{*1} FLASH_RES_LOCKBIT_STATE_PROTECTED : プロテクト状態 FLASH_RES_LOCKBIT_STATE_NON_PROTECTED : 非プロテクト状態
FLASH_CMD_LOCKBIT_WRITE (引数の型 : flash_lockbit_config_t *) *使用例は Example 12 に記載	コードフラッシュメモリ上にロックビットの対象領域として扱うブロックの先頭アドレスとブロック数を設定します。 ロックビットの設定はブロックの先頭アドレスとブロック数で指定された対象領域を複数設定することができます。 ノンブロッキングモードで使用する場合は、ロックビットの設定後 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。 ^{*1}
FLASH_CMD_LOCKBIT_ENABLE (引数には NULL を設定してください) *使用例は Example 12 に記載	ロックビットの対象領域として設定されているコードフラッシュメモリ上のブロックに対するプログラムやイレーズを禁止します。

FLASH_CMD_LOCKBIT_DISABLE (引数には NULL を設定してください) *使用例は Example 12 に記載	ロックビットの対象領域として設定されているコードフラッシュメモリ上のブロックに対するプログラムやイレーズを許可します。 ロックビットが設定されたブロックは本コマンドを使用してからイレーズすることが出来ます。 ロックビットが設定されているブロックをイレーズすることにより、イレーズされブロックのロックビットの設定もクリアされますのでご注意ください。
FLASH_CMD_ROM_CACHE_ENABLE (引数には NULL を設定してください) *使用例は Example 10 に記載	コードフラッシュメモリのキャッシュを有効にします。
FLASH_CMD_ROM_CACHE_DISABLE (引数には NULL を設定してください) *使用例は Example 10 に記載	コードフラッシュメモリのキャッシュを無効にします。 コードフラッシュメモリを書き換える前に呼び出してください。
FLASH_CMD_ROM_CACHE_STATUS (引数の型 : uint8_t*) *使用例は Example 10 に記載	コードフラッシュメモリのキャッシュの状態を取得します。 0 : コードフラッシュメモリのキャッシュは無効 1 : コードフラッシュメモリのキャッシュは有効
FLASH_CMD_SET_NON_CACHED_RANGE0 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	コードフラッシュメモリ上の指定した範囲をノンキャッシュابل領域 0 に設定します。指定された範囲はキャッシュが無効になります。 キャッシュが有効な状態で、本コマンドが実行された場合、キャッシュは一時的に無効となりますのでご注意ください。
FLASH_CMD_SET_NON_CACHED_RANGE1 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	コードフラッシュメモリ上の指定した範囲をノンキャッシュابل領域 1 に設定します。指定された範囲はキャッシュが無効になります。 キャッシュが有効な状態で、本コマンドが実行された場合、キャッシュは一時的に無効となりますのでご注意ください。
FLASH_CMD_GET_NON_CACHED_RANGE0 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	ノンキャッシュابل領域 0 の設定を取得します。
FLASH_CMD_GET_NON_CACHED_RANGE1 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	ノンキャッシュابل領域 1 の設定を取得します。
FLASH_CMD_CONFIG_CLOCK (引数の型 : uint32_t*)	フラッシュシーケンサに使用する周波数を通知します。 プログラム動作中に BSP で設定した周波数から Flash クロック (FLCK) の速度を変更した場合に使用します。Flash クロック (FCLK) を変更しない場合は使用する必要はありません。

*1 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。

表 3.8 フラッシュタイプ 4 でサポートしているコマンドの詳細

コマンド	内容
FLASH_CMD_STATUS_GET (引数には NULL を設定してください) *使用例は Example 3 に記載	フラッシュメモリに対するフラッシュシーケンサの実行状態を取得します。 本コマンドはフラッシュメモリに対する処理が実行中の場合でも使用可能です。 FLASH_SUCCESS : フラッシュシーケンサは実行されていない FLASH_ERR_BUSY : フラッシュシーケンサは実行されている
FLASH_CMD_SET_BGO_CALLBACK (引数の型 : flash_interrupt_config_t *) *使用例は Example 1、Example 2 に記載	コールバック関数を登録します。本コマンドはノンブロッキングモードで使用する場合に必要です。
FLASH_CMD_RESET (引数には NULL を設定してください)	フラッシュシーケンサをリセットします。 本コマンドはフラッシュメモリに対する処理が実行中の場合でも使用可能です。
FLASH_CMD_ACCESSWINDOW_GET (引数の型 : flash_access_window_config_t *) *使用例は Example 4 に記載	コードフラッシュメモリのアクセスウィンドウの対象領域として扱うブロックの開始アドレスと終了アドレスを取得します。
FLASH_CMD_ACCESSWINDOW_SET (引数の型 : flash_access_window_config_t *) *使用例は Example 5 に記載	コードフラッシュメモリのアクセスウィンドウの対象領域として扱うブロックの開始アドレスと終了アドレスを指定します。 アクセスウィンドウの設定において、開始アドレスは終了アドレスよりも小さい値とする必要があります。 開始アドレスと終了アドレスで指定された範囲以外のブロックに対してはプログラムやイレーズすることができません。 また、開始アドレスと終了アドレスで指定された範囲を複数指定することはできません。 アクセスウィンドウの設定を削除する場合は、開始アドレスと終了アドレスに同じ値を指定します。 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。 ^{*1}
FLASH_CMD_SWAPFLAG_GET (引数の型 : uint32_t *) *使用例は Example 6 に記載	スタートアップ領域の設定を取得します。 0 : スタートアップ領域 0 と 1 の設定は入れ替わっている 1 : スタートアップ領域 0 と 1 の設定はデフォルトのまま
FLASH_CMD_SWAPFLAG_TOGGLE (引数には NULL を設定してください) *使用例は Example 7 に記載	スタートアップ領域を切り替えます。 切り替えられたスタートアップ領域は次のリセットで有効となります。ノンブロッキングモードで使用する場合は、スタートアップ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。 ^{*1}

<p>FLASH_CMD_SWAPSTATE_GET (引数の型 : uint8_t*) *使用例は Example 8 に記載</p>	<p>リニアモードの場合、スタートアップ領域選択ビット (FSUACR.SAS)の値を取得します。</p> <p>デュアルモードの場合、本コマンドは使用できません。</p> <p>FLASH_SAS_SWAPFLG : スタートアップ領域選択ビットはスタートアップ領域の設定に従う</p> <p>FLASH_SAS_DEFAULT : スタートアップ領域選択ビットはスタートアップ領域 0 に設定されている</p> <p>FLASH_SAS_ALTERNATE : スタートアップ領域選択ビットはスタートアップ領域 1 に設定されている</p>
<p>FLASH_CMD_SWAPSTATE_SET (引数の型 : uint8_t*) *使用例は Example 9 に記載</p>	<p>リニアモードの場合、スタートアップ領域選択ビット (FSUACR.SAS)の値を設定します。</p> <p>即座に設定されたスタートアップ領域となります。</p> <p>リセット後の初期値は FLASH_SAS_SWAPFLG です。</p> <p>デュアルモードの場合、本コマンドは使用できません。</p> <p>FLASH_SAS_SWAPFLG : オプション設定メモリ内のスタートアップ領域の設定に従う</p> <p>FLASH_SAS_DEFAULT : 一時的にスタートアップ領域をスタートアップ領域 0 に切り替える</p> <p>FLASH_SAS_ALTERNATE : 一時的にスタートアップ領域をスタートアップ領域 1 に切り替える</p> <p>FLASH_SAS_SWITCH_AREA : スタートアップ領域を切り替える</p>
<p>FLASH_CMD_ROM_CACHE_ENABLE (引数には NULL を設定してください) *使用例は Example 10 に記載</p>	<p>コードフラッシュメモリのキャッシュを有効にします。</p>
<p>FLASH_CMD_ROM_CACHE_DISABLE (引数には NULL を設定してください) *使用例は Example 10 に記載</p>	<p>コードフラッシュメモリのキャッシュを無効にします。</p> <p>コードフラッシュメモリを書き換える前に呼び出してください。</p>
<p>FLASH_CMD_ROM_CACHE_STATUS (引数の型 : uint8_t*) *使用例は Example 10 に記載</p>	<p>コードフラッシュメモリのキャッシュの状態を取得します。</p> <p>0 : コードフラッシュメモリのキャッシュは無効</p> <p>1 : コードフラッシュメモリのキャッシュは有効</p>

FLASH_CMD_SET_NON_CACHED_RANGE0 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	コードフラッシュメモリ上の指定した範囲をノンキャッシュابل領域 0 に設定します。指定された範囲はキャッシュが無効になります。 キャッシュが有効な状態で、本コマンドが実行された場合、キャッシュは一時的に無効となりますのでご注意ください。
FLASH_CMD_SET_NON_CACHED_RANGE1 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	コードフラッシュメモリ上の指定した範囲をノンキャッシュابل領域 1 に設定します。指定された範囲はキャッシュが無効になります。 キャッシュが有効な状態で、本コマンドが実行された場合、キャッシュは一時的に無効となりますのでご注意ください。
FLASH_CMD_GET_NON_CACHED_RANGE0 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	ノンキャッシュابل領域 0 の設定を取得します。
FLASH_CMD_GET_NON_CACHED_RANGE1 (引数の型 : flash_non_cached_t*) *使用例は Example 11 に記載	ノンキャッシュابل領域 1 の設定を取得します。
FLASH_CMD_BANK_TOGGLE ² (引数には NULL を設定してください) *使用例は Example 13 に記載	リニアモードの場合、本コマンドは使用できません。 デュアルモードの場合、起動バンクを切り替えます。 起動バンクの切り替えは次のリセット後に有効となります。 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 (BANKSEL) 設定後 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。 ^{*1}
FLASH_CMD_BANK_GET ² (引数の型 : flash_bank_t*) *使用例は Example 13 に記載	リニアモードの場合、本コマンドは使用できません。 デュアルモードの場合、バンク選択レジスタ (BANKSEL) から起動バンクに関する現在の設定を取得します。 FLASH_BANK0 : 1 FLASH_BANK1 : 0
FLASH_CMD_CONFIG_CLOCK (引数の型 : uint32_t*)	フラッシュシーケンサに使用する周波数を通知します。 また、データフラッシュメモリのリード速度を設定します。 ^{*2} プログラム動作中に BSP で設定した周波数から Flash クロック (FLCK) の速度を変更した場合に使用します。Flash クロック (FCLK) を変更しない場合は使用する必要はありません。

^{*1} 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。

^{*2} コードフラッシュメモリの容量が 1.5M バイト以上の製品のみが対象となります。

Example 1: コードフラッシュメモリに対してノンブロッキングモードで書き込む

フラッシュモジュールの API 関数をノンブロッキングモードで使用する場合は、コンフィギュレーションオプションの FLASH_CFG_DATA_FLASH_BGO、FLASH_CFG_CODE_FLASH_BGO を"1"にします。

RAM からコードを実行してコードフラッシュメモリに対してプログラムする場合は、コンフィギュレーションオプションの FLASH_CFG_CODE_FLASH_ENABLE を"1"にします。また、発生し得る割り込みのベクタテーブルについては RAM に再配置する必要があります。

コールバック関数については R_FLASH_Open()を実行した後、R_FLASH_Control()を使用してコールバック関数を登録し、フラッシュモジュールの API 関数(R_FLASH_Write()、R_FLASH_Erase()、R_FLASH_BlankCheck())を実行することにより、登録されたコールバック関数を使用することができます。

```
/* ベクタテーブルを保持する RAM の領域 */
static uint32_t ram_vect_table[256];

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

    /* 割り込みベクタテーブルを RAM に再配置 */
    /* FRDYI 割り込み関数のアドレスを ram_vect_table[23]に直接設定。*/
    /* 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)
    {
        /* エラー処理 */
    }

    /* 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)
    {
        /* エラー処理 */
    }

    /* コードフラッシュメモリ上で動作 */
    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 */
    ... (省略)
}

void do_rom_operations(void)
{
    /* コードフラッシュメモリの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、プログラムの処理をここに記載 */
    ... (省略)
}
#pragma section
```

Example 2: データフラッシュメモリに対してノンブロッキングモードで書き込む

フラッシュモジュールの API 関数をノンブロッキングモードで使用する場合は、コンフィギュレーションオプションの FLASH_CFG_DATA_FLASH_BGO、FLASH_CFG_CODE_FLASH_BGO を"1"にします。

データフラッシュメモリに対してプログラムする場合は、フラッシュメモリを書き換えるためのコードはコードフラッシュメモリ上で実行することができます。

コールバック関数については R_FLASH_Open()を実行した後、R_FLASH_Control()を使用してコールバック関数を登録し、フラッシュモジュールの API 関数(R_FLASH_Write()、R_FLASH_Erase()、R_FLASH_BlankCheck())を実行することにより、登録されたコールバック関数を使用することができます。

```
void func(void)
{
    flash_err_t err;
    flash_interrupt_config_t cb_func_info;

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

    /* 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)
    {
        /* エラー処理 */
    }

    /* データフラッシュメモリのイレーズ、ブランクチェック、プログラムの処理をここに記載 */
    ... (省略)
}

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 */
    ... (省略)
}
```

Example 3: フラッシュモジュールの API 関数の実行状態を確認する

ノンブロッキングモードで R_FLASH_Erase()を使用した場合を例として以下に示します。

```
flash_err_t err;

/* 全データフラッシュをイレーズ */
err = R_FLASH_Erase(FLASH_DF_BLOCK_0, FLASH_NUM_BLOCKS_DF);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* フラッシュモジュールの API 関数の実行状態を確認 */
while (FLASH_ERR_BUSY == R_FLASH_Control(FLASH_CMD_STATUS_GET, NULL))
{
    /* 任意の処理を実施 */
}
```

Example 4: コードフラッシュメモリに対するアクセスウィンドウの設定範囲を取得する

```
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)
{
    /* エラー処理 */
}
```

Example 5: コードフラッシュメモリに対するアクセスウィンドウの範囲を設定する

アクセスウィンドウによる領域の保護は、コードフラッシュメモリの設定された範囲に対して、誤ったプログラムやイレーズを防ぐために使用されます。

```
flash_err_t err;
flash_access_window_config_t access_info;

/* コードフラッシュメモリのブロック 3 に対するプログラムやイレーズを許可 */
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)
{
    /* エラー処理 */
}

/* コードフラッシュメモリのブロック 2、ブロック 1、ブロック 0 を含むプログラムやイレーズを許可 */
/* 設定範囲にブロック 0 を含む場合、終了アドレスの指定に FLASH_CF_BLOCK_END を使用します。 */
access_info.start_addr = (uint32_t) FLASH_CF_BLOCK_2;
access_info.end_addr = (uint32_t) FLASH_CF_BLOCK_END;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}
```

Example 6: スタートアップ領域の設定を取得する

```
flash_err_t err;
uint32_t swap_flag;

err = R_FLASH_Control(FLASH_CMD_SWAPFLAG_GET, (void *)&swap_flag);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}
```

Example 7: スタートアップ領域の設定を切り替える

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。

```
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)
{
    /* エラー処理 */
}
```

Example 8: スタートアップ領域選択ビットの値を取得する

```
flash_err_t err;
uint8_t swap_area;

err = R_FLASH_Control(FLASH_CMD_SWAPSTATE_GET, (void *)&swap_area);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}
```

Example 9: 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の設定方法を示します。リセット後、スタートアップ領域選択ビットで指定された領域が使用されます。

```
flash_err_t err;
uint8_t swap_area;

swap_area = FLASH_SAS_SWITCH_AREA;
err = R_FLASH_Control(FLASH_CMD_SWAPSTATE_SET, (void *)&swap_area);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}
```

Example 10: コードフラッシュメモリに対するキャッシュの有効／無効を設定する

コードフラッシュメモリのキャッシュを有効にした状態から、イレーズ、プログラムする際にキャッシュを無効にした後、再度、キャッシュを有効にする場合を例として以下に示します。

```
flash_err_t err;
uint8_t status;

/* キャッシュを有効にする */
err = R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* キャッシュが有効となっていることを確認 */
err = R_FLASH_Control(FLASH_CMD_ROM_CACHE_STATUS, &status);
if ((FLASH_SUCCESS != err) || (1 != status))
{
    /* エラー処理 */
}

... (省略)

/* プログラムする準備としてキャッシュを無効する */
err = R_FLASH_Control(FLASH_CMD_ROM_CACHE_DISABLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* イレーズ、プログラム、ベリファイなどのコードをここに記載 */

/* キャッシュを再度有効にする */
err = R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}
```


Example 11: コードフラッシュメモリの特定範囲のキャッシュを無効にする

コードフラッシュメモリの特定範囲のキャッシュを無効にする方法を以下に示します。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_err_t err;
flash_non_cached_t range;

/* 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;

err = R_FLASH_Control(FLASH_CMD_SET_NON_CACHED_RANGE0, &range);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* 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。 */
err = R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* RANGE0 のキャッシュの設定を取得 */
err = R_FLASH_Control(FLASH_CMD_GET_NON_CACHED_RANGE0, &range);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}
```

Example 12: コードフラッシュメモリに対するロックビットによるプロテクトを設定する

コードフラッシュメモリの指定した範囲のブロックに対する、ロックビットの設定、およびロックビットの取得を行った後、ロックビットによるプロテクトの無効化、プロテクトの有効化を行う場合の例を以下に示します。

```
flash_err_t err;
flash_lockbit_config_t lockbit_info;

/* コードフラッシュメモリのブロック 3 に対するロックビットを設定する */
lockbit_info.block_start_address = FLASH_CF_BLOCK_3;
lockbit_info.num_blocks = 1;
err = R_FLASH_Control(FLASH_CMD_LOCKBIT_WRITE, (void *)&lockbit_info);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* コードフラッシュメモリのブロック 3 に対するロックビットを取得する */
err = R_FLASH_Control(FLASH_CMD_LOCKBIT_READ, (void *)&lockbit_info);
if ((FLASH_SUCCESS != err) ||
    (lockbit_info.result != FLASH_RES_LOCKBIT_STATE_PROTECTED))
{
    /* エラー処理 */
}

/* ロックビットによるプロテクトを無効化することにより、                               */
/* ロックビットが設定されているブロックのイレーズ／プログラムは許可されている状態となる */
err = R_FLASH_Control(FLASH_CMD_LOCKBIT_DISABLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* ロックビットが設定されているブロックのイレーズ／プログラムは許可されている状態 */

/* ロックビットによるプロテクトを有効化することにより、                               */
/* ロックビットが設定されているブロックのイレーズ／プログラムは禁止されている状態となる */
err = R_FLASH_Control(FLASH_CMD_LOCKBIT_ENABLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* ロックビットが設定されているブロックのイレーズ／プログラムは禁止されている状態 */
```

Example 13: 起動バンクの切り替え

起動バンクの切り替えを行います。起動バンクの切り替えは次のリセットで有効となります。

次のリセットで有効となる起動バンクは R_FLASH_Control() の第 1 引数に FLASH_CMD_BANK_GET コマンドを指定することにより、第 2 引数で取得することができます。

第 2 引数の取得結果が FLASH_BANK0 の場合、次のリセットで有効となる起動バンクはバンク 0 となります。FLASH_BANK1 の場合、次のリセットで有効となる起動バンクはバンク 1 となります。

```
flash_err_t err;
flash_bank_t bank_info;

/* 起動バンクとして選択されているバンクの切り替えを行う */
err = R_FLASH_Control(FLASH_CMD_BANK_TOGGLE, NULL);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* 起動バンクとして選択されているバンクを取得する */
err = R_FLASH_Control(FLASH_CMD_BANK_GET, (void *)&bank_info);
if (FLASH_SUCCESS != err)
{
    /* エラー処理 */
}

/* 起動バンクの切り替えは次のリセットで有効となる */
```

Special Notes:

なし

3.7 R_FLASH_GetVersion()

本 API 関数はフラッシュモジュールのバージョン番号を取得する関数です。

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

本 API 関数はフラッシュ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モジュールのバージョンを取得 */
cur_version = R_FLASH_GetVersion();

/* バージョンの判別処理 */
if (0x00040019 > cur_version)
{
    /* バージョンに応じた個別処理 */
}
```

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

- 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

- 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 e2 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
モジュールのバージョン	Rev.4.00
使用ボード	Renesas Starter Kit for RX113 (型名：R0K505113xxxxxx) Renesas Starter Kit for RX130 (型名：RTK5005130xxxxxxxxx) Renesas Starter Kit for RX231 (型名：R0K505231xxxxxx) Renesas Starter Kit for RX23T (型名：RTK500523Txxxxxxxxx) Renesas Starter Kit for RX24T (型名：RTK500524Txxxxxxxxx) Renesas Starter Kit for RX24U (型名：RTK500524Uxxxxxxxxx) Renesas Starter Kit+ for RX64M (型名：R0K50564Mxxxxxx) Renesas Starter Kit for RX66T (型名：RTK50566Txxxxxxxxx) Renesas Starter Kit for RX72T (型名：RTK5572Txxxxxxxxxx) Renesas Starter Kit+ for RX65N (型名：RTK500565Nxxxxxxxxx) Renesas Starter Kit+ for RX65N-2MB (型名：RTK50565Nxxxxxxxxxx)

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 e2 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
モジュールのバージョン	Rev.4.10
使用ボード	Renesas Solution Starter Kit for RX23W (型名：RTK5523Wxxxxxxxxxx)

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 e2 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
モジュールのバージョン	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 e2 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
モジュールのバージョン	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 e2 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
モジュールのバージョン	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 e2 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
モジュールのバージョン	Rev.4.50
使用ボード	Renesas Starter Kit+ for RX72N（型名：RTK5572Nxxxxxxxxxx）

表 5.7 動作確認環境 (Rev.4.70)

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 2021-01
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 V3.02.00 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
モジュールのバージョン	Rev.4.70
使用ボード	Renesas Starter Kit+ for RX671（型名：RTK55671xxxxxxxxxx）

5.2 トラブルシューティング

- (1) Q : 本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、ビルド実行すると以下のエラーが発生します。

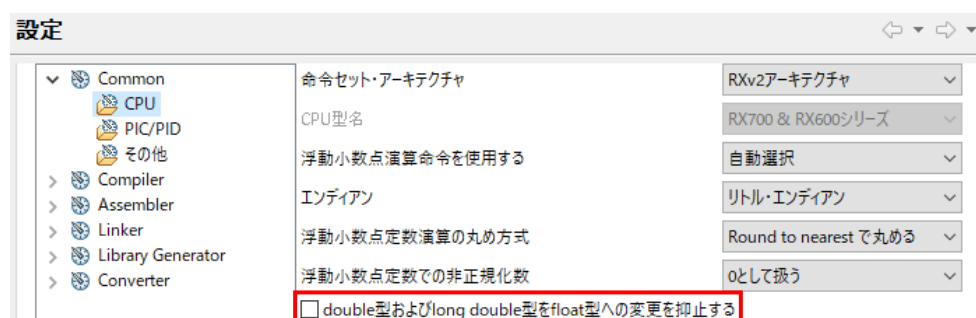
“No data flash on this MCU. Set FLASH_CFG_CODE_FLASH_ENABLE to 1 in r_flash_rx_config.h.”

A : “r_flash_rx_config.h”ファイルの設定値が間違っている可能性があります。
“r_flash_rx_config.h” ファイルを確認して正しい値を設定してください。詳細は「2.7 コンパイル時の設定」を参照してください。

- (3) Q : 本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



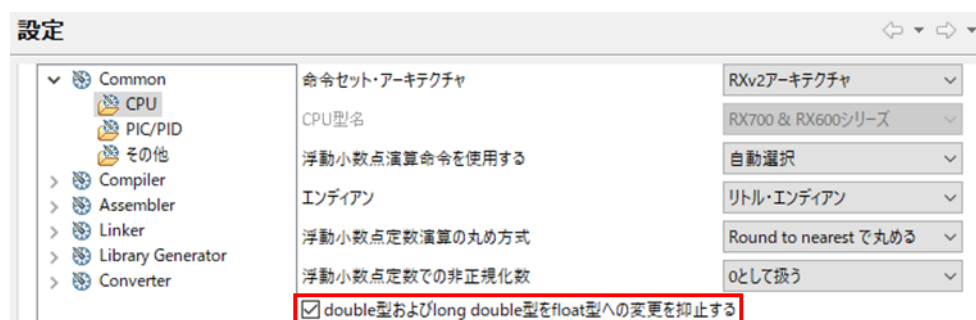
デフォルトのコンパイラオプション設定時のリストファイルの出力結果

```

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 FB4280283841      ^ ^      MOV.L #41382880H, R4J
000003CF 05rrrrrr      A ^ ^      BSR COM_MULdJ
000003D3 754740      ^ ^      MOV.L #00000040H, R7J
000003D6      L127: ^ ^      ; if_break_bb33J
000003D6 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-2 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。

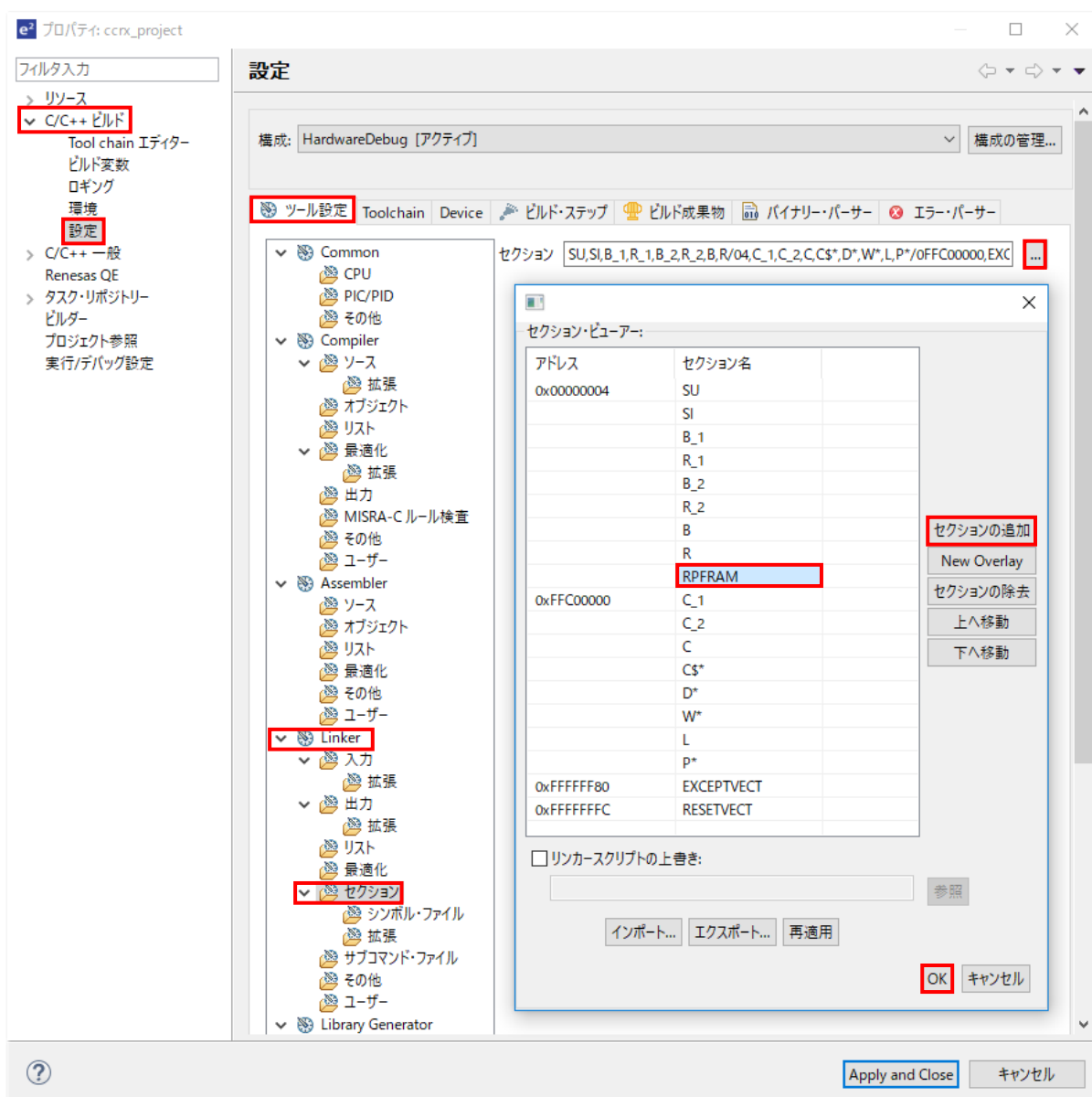
リンクのセクションの設定とコードフラッシュから RAM へのマッピングは e2 studio で行う必要があります。

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

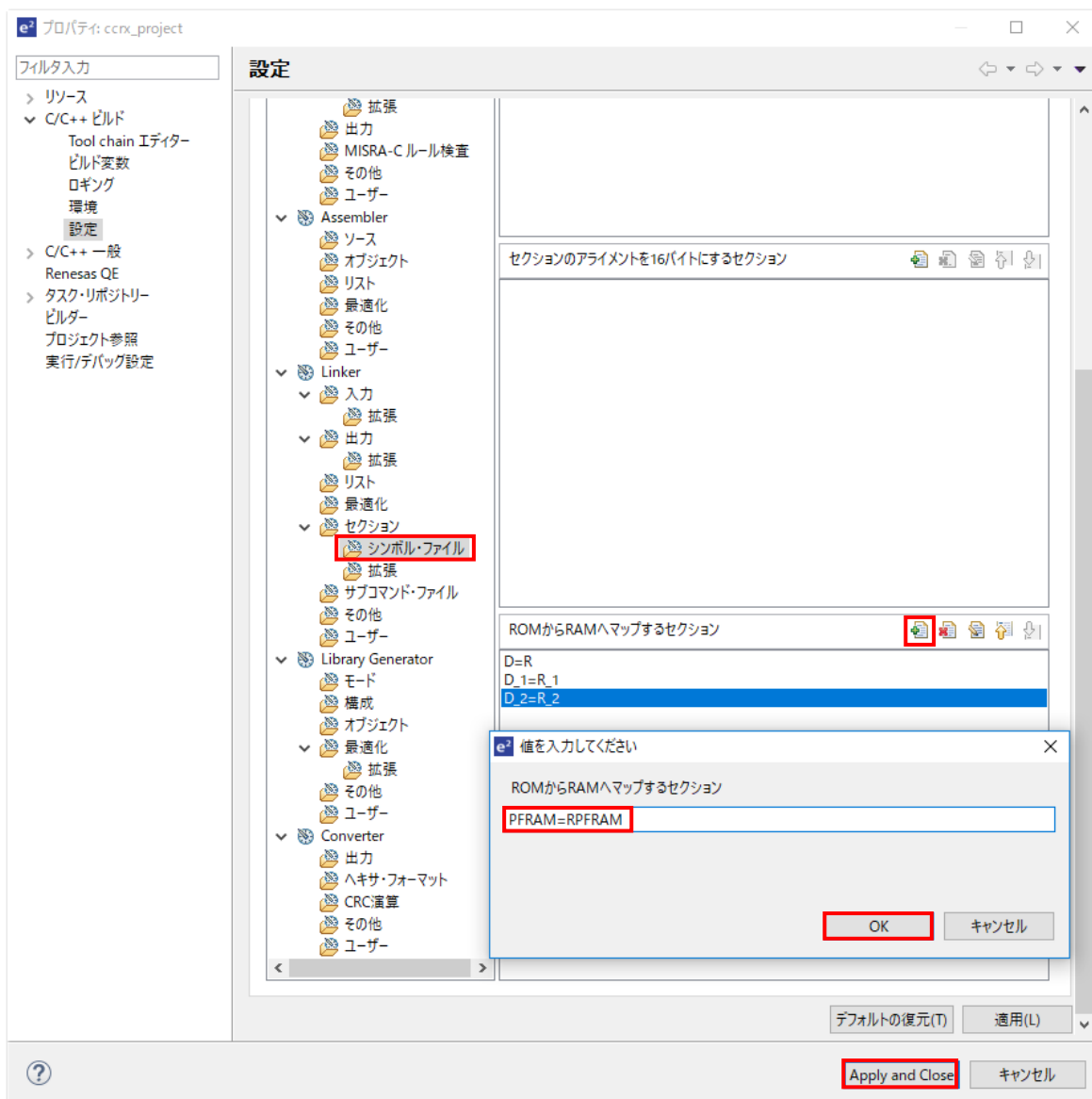
本章では、セクションの追加、コードフラッシュから RAM へのマッピング、コードフラッシュ書き換え中に動作するプログラムの配置について示します。

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) {...}

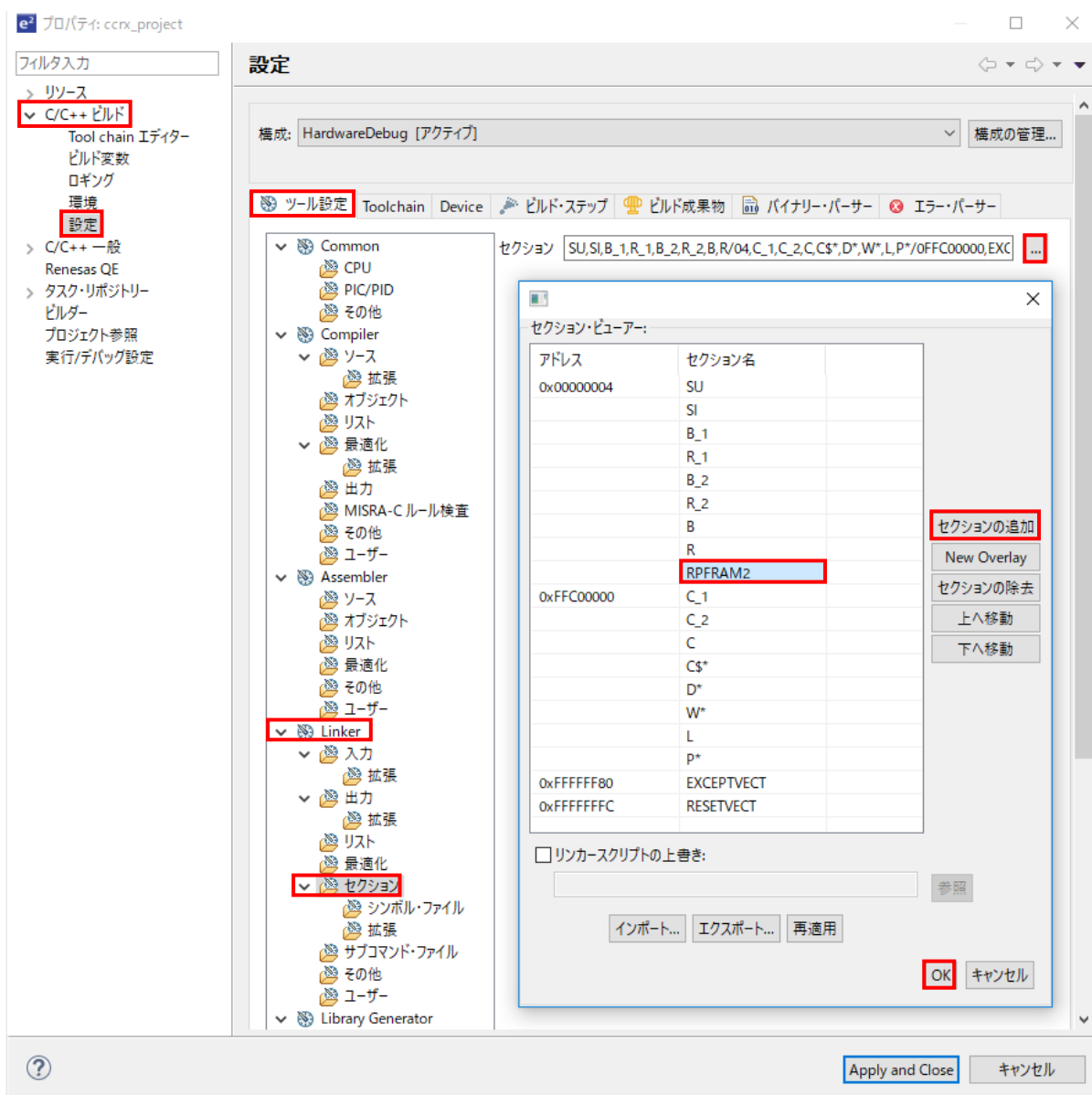
/* コードフラッシュ書き換え中に動作するコールバック関数 */
void cb_func(void) {...}
#pragma section
```

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

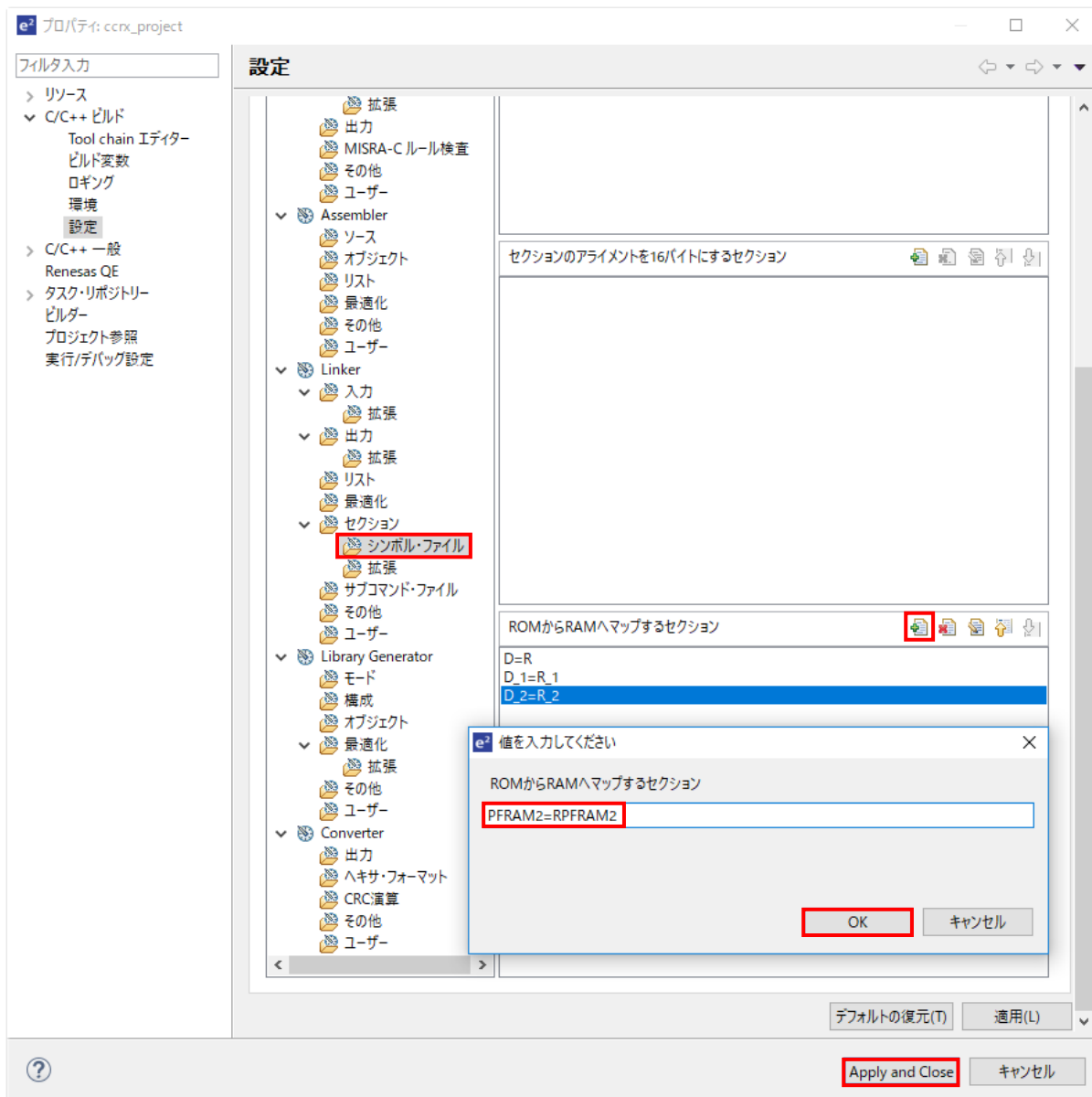
本章では、セクションの追加、コードフラッシュから RAM へのマッピング、デュアルバンク機能のデバッグについて示します。

1. RAM 領域に"RPFRAM2"セクションを追加します。

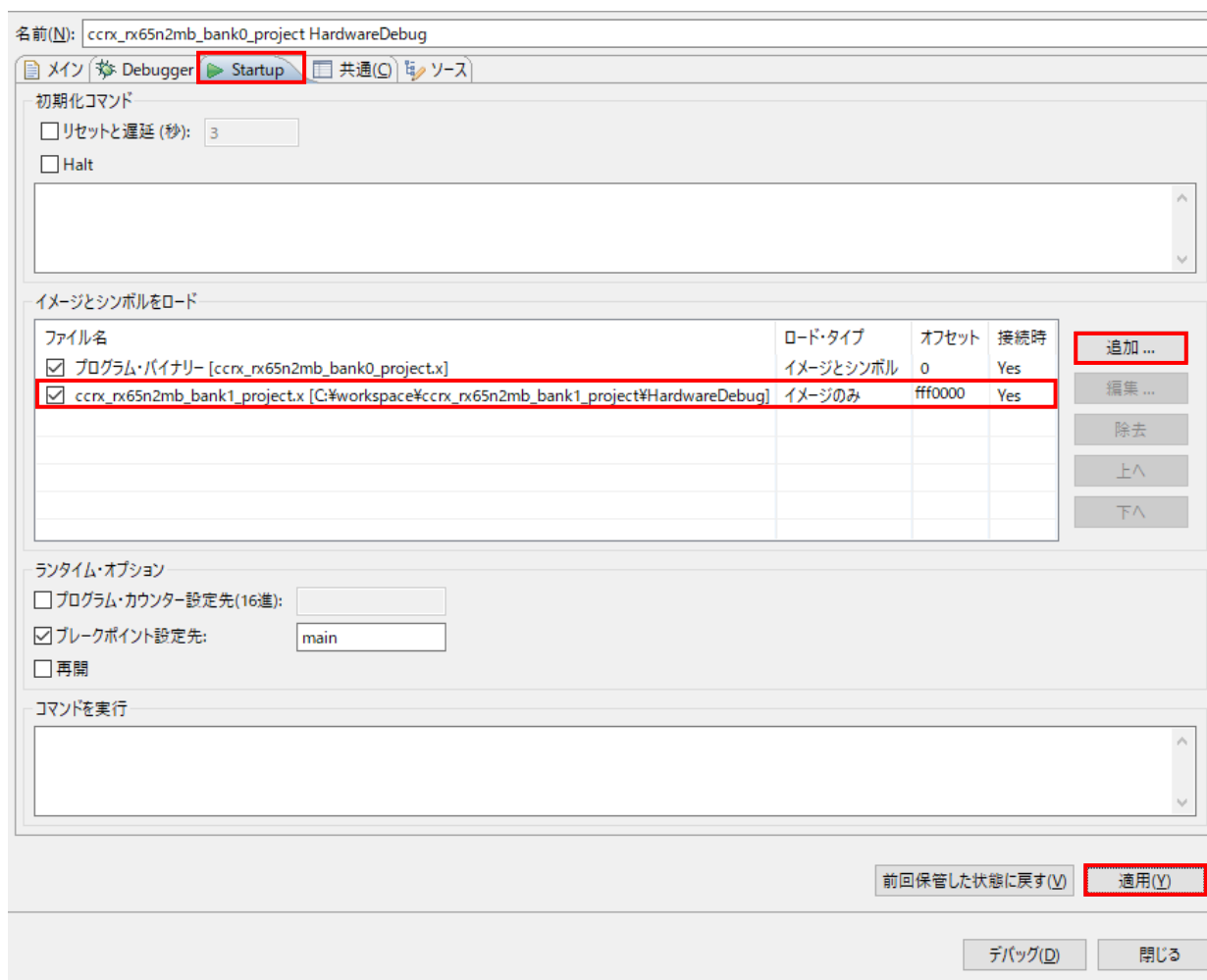
- (1) 「プロジェクト・エクスプローラー」においてデバッグ対象のプロジェクトをクリックします。
- (2) 「ファイル」→「プロパティ」の順にクリックし、「プロパティ」ウィンドウを開きます。
- (3) 「プロパティ」ウィンドウで、「C/C++ビルド」→「設定」の順にクリックします。
- (4) 「ツール設定」タブを押下し、「Linker」→「セクション」の順にクリックして、「...」ボタンを押下し、「セクション・ビューアー」ウィンドウを開きます。
- (5) 「セクション・ビューアー」ウィンドウで、「セクションの追加」ボタンを押下して、RAM 領域に"RPFRAM2"セクションを追加し、「OK」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

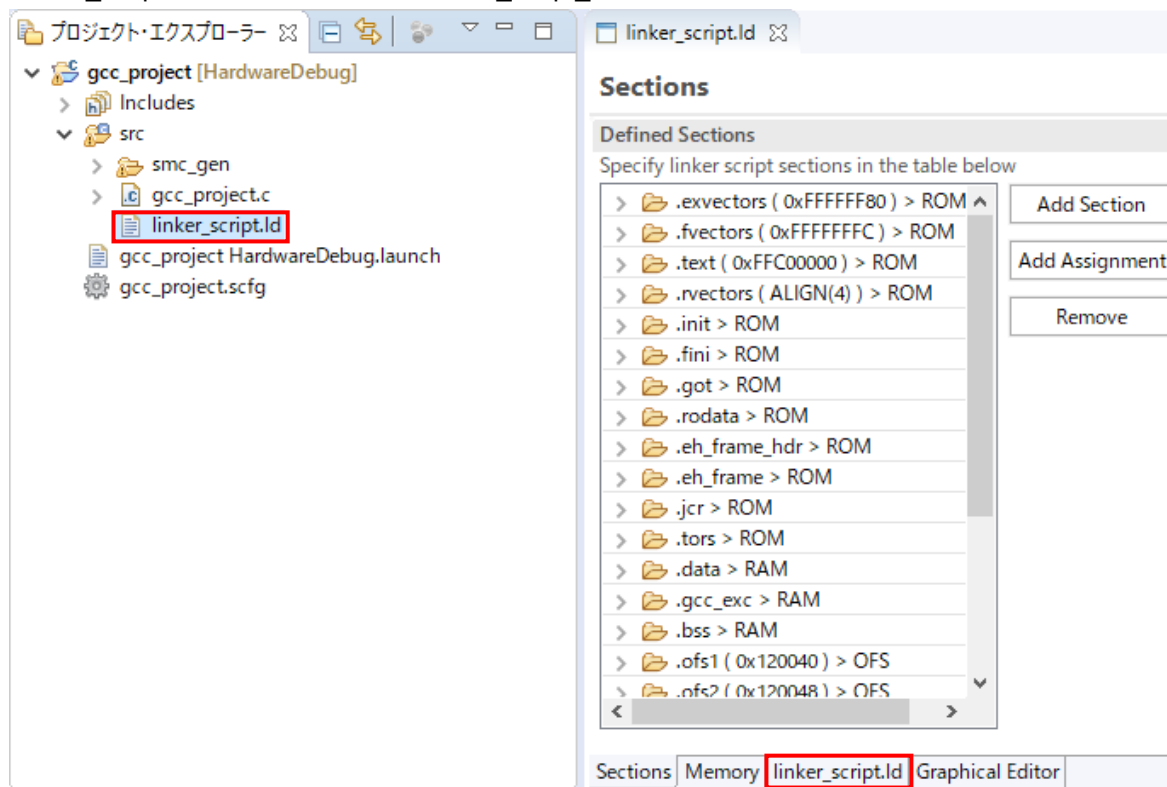
リンカの設定は e² studio で生成されるリンカ設定ファイルを編集する必要があります。

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.id」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     .tors :
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

```

The screenshot shows a linker script editor with tabs for 'Sections', 'Memory', 'linker_script.ld', and 'Graphical Editor'. The script defines several memory sections: `.tors` (ROM), `.pfram` (ROM), `.data` (RAM), `.rpfram` (RAM), and `.gcc_exc` (RAM). Red boxes highlight the `.tors` section, the `.pfram` section, and the `.rpfram` section. The `.tors` section includes code for handling constructors and destructors. The `.pfram` section is aligned to 4 bytes and is located in ROM. The `.rpfram` section is also aligned to 4 bytes and is located in RAM. The `.data` section is located in RAM and contains data for the program. The `.gcc_exc` section is located in RAM and contains exception handling code.

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) {...}

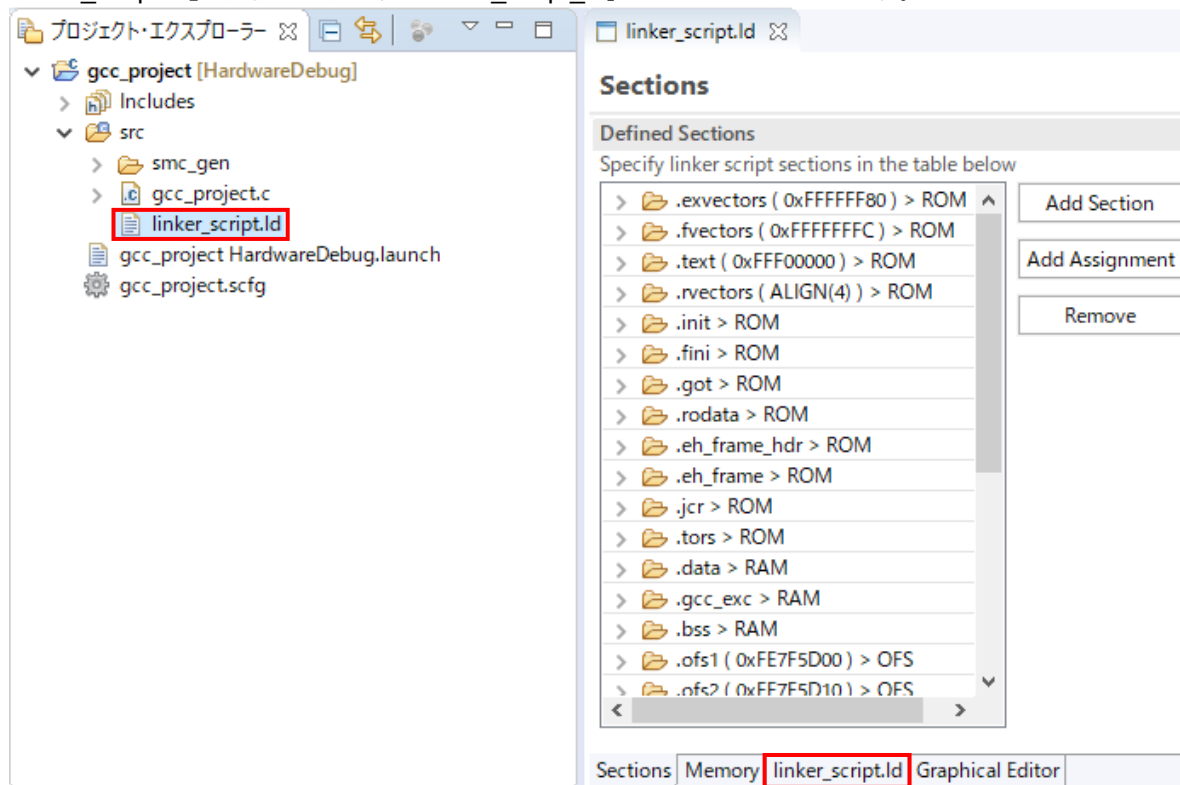
```

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。

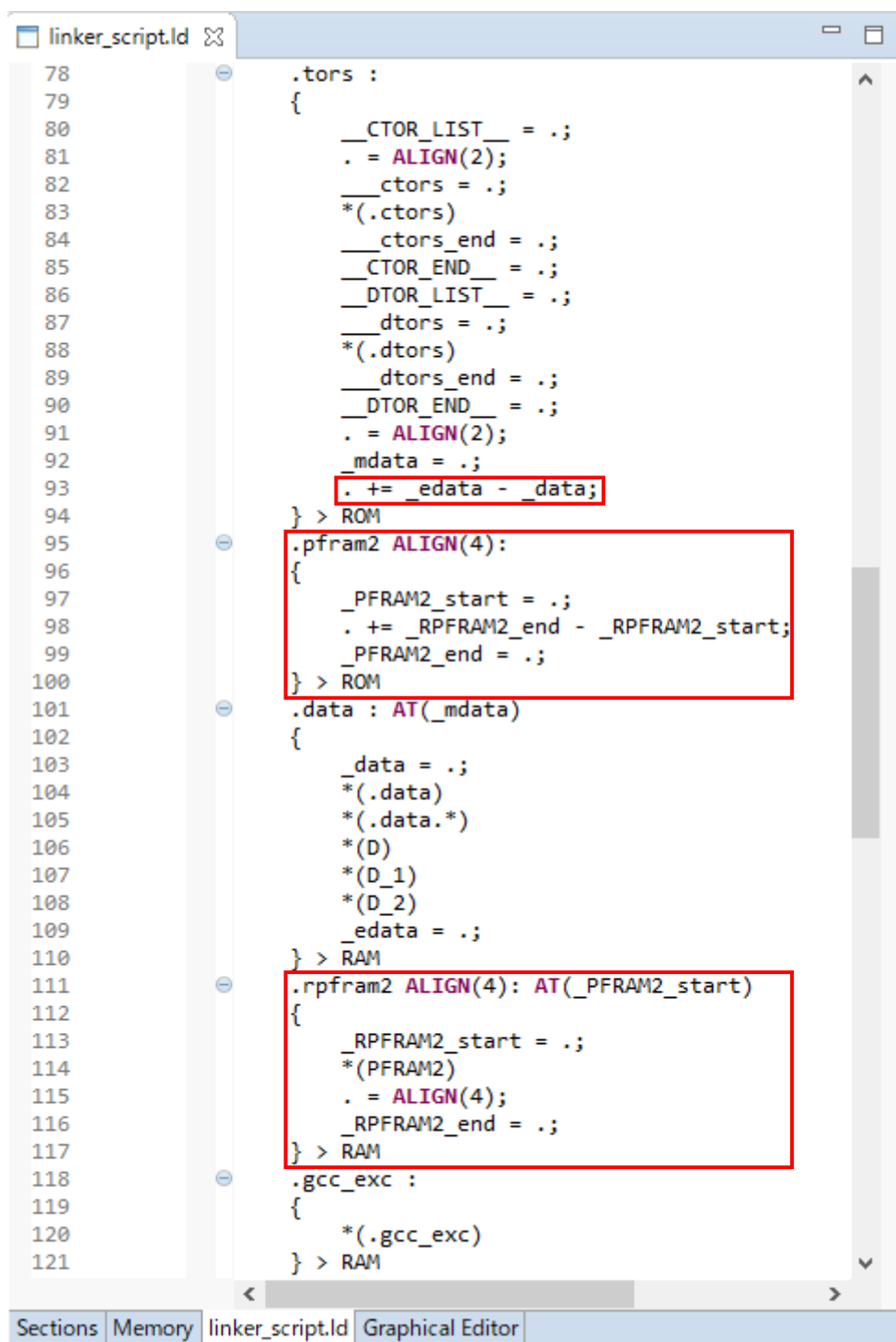
1. リンカ設定ファイル(linker_script.ld)に設定を追加します。

- (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。



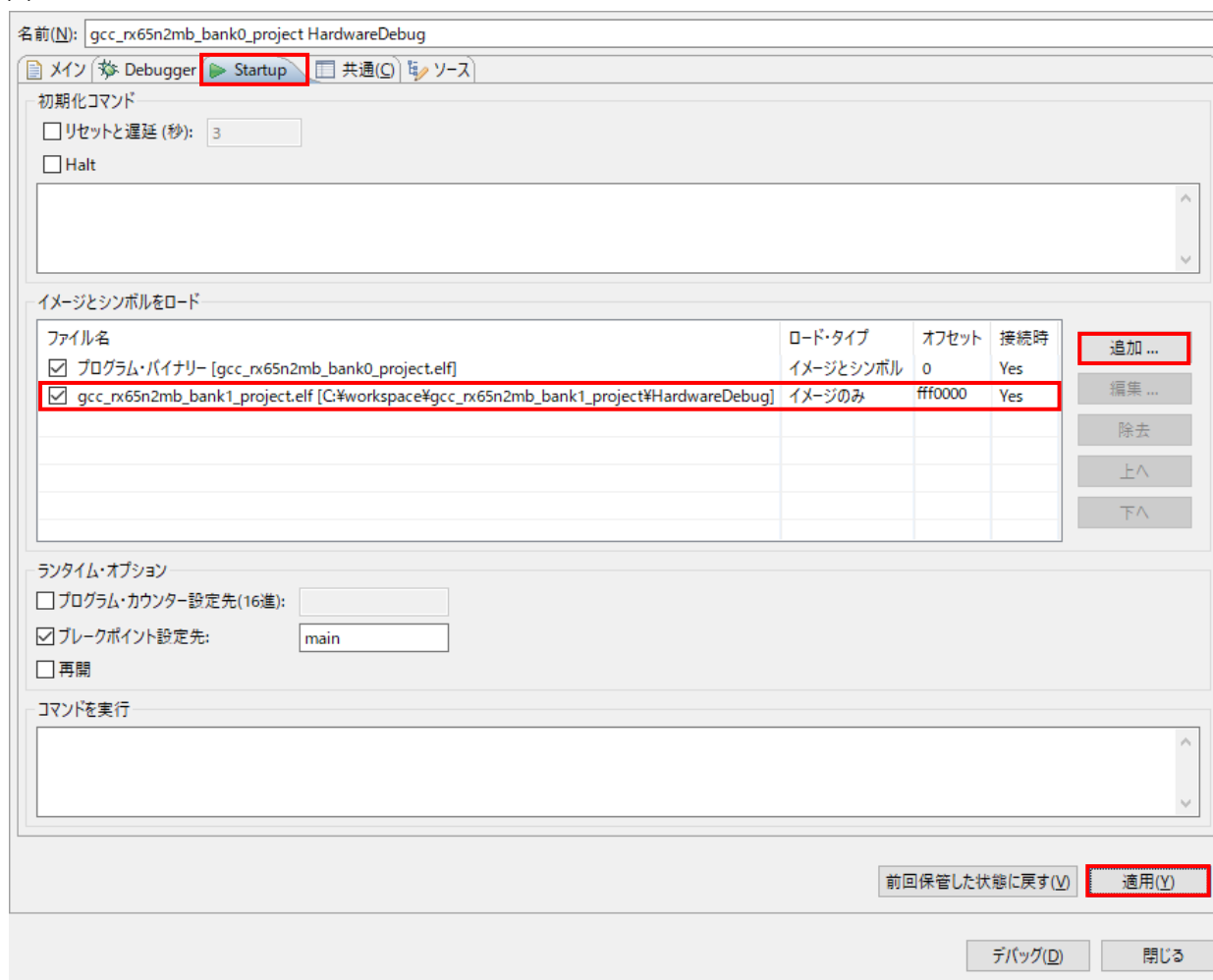
(3) 以下の(a)~(c)をリンカ設定ファイル(linker_script.ld)に追加します。

- (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。

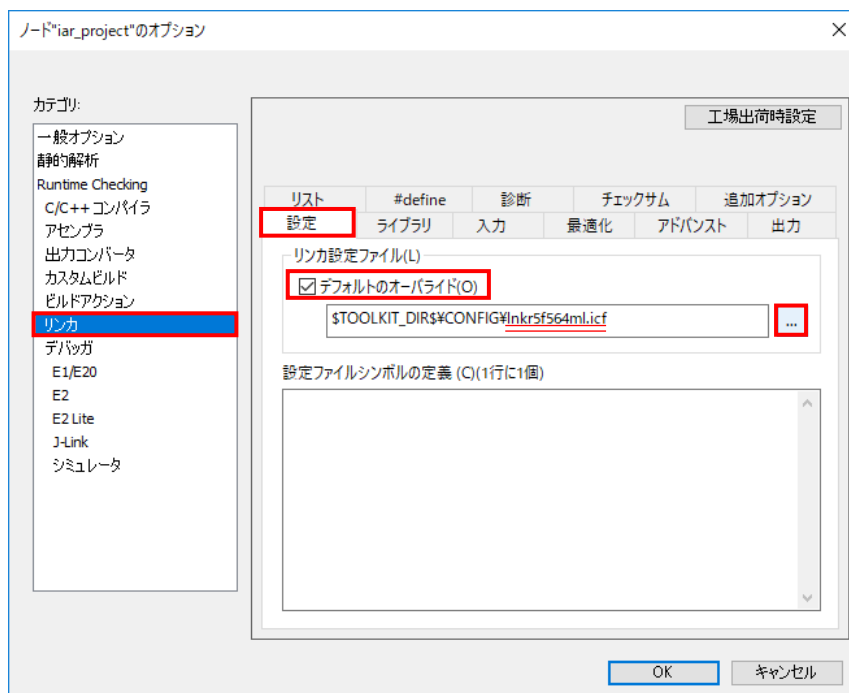
- スタンドアロン版のスマート・コンフィグレータを使用する方法
スタンドアロン版のスマート・コンフィグレータを使用し、本モジュールや BSP を追加した IAR 用のプロジェクトを生成して使用します。スタンドアロン版のスマート・コンフィグレータの詳細はアプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: IAREW 編 (R20AN0535)」に記載しています。
- IAR Embedded Workbench の FIT Module Importer を使用する方法
IAR Embedded Workbench の FIT Module Importer を使用し、本モジュールや 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。

本モジュールを 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。

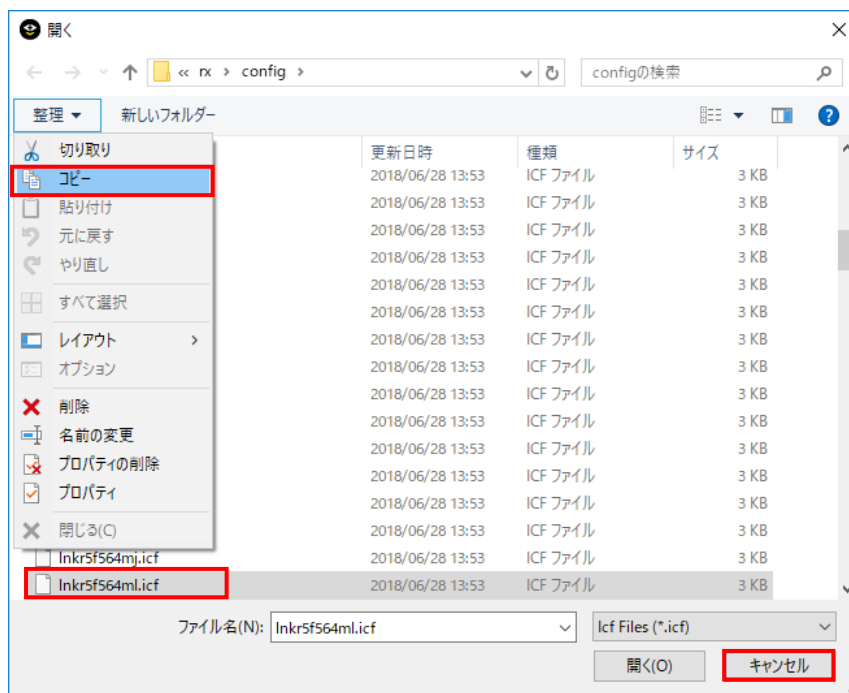
5.3.3.1 RAM からコードを実行してコードフラッシュを書き換える場合

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。

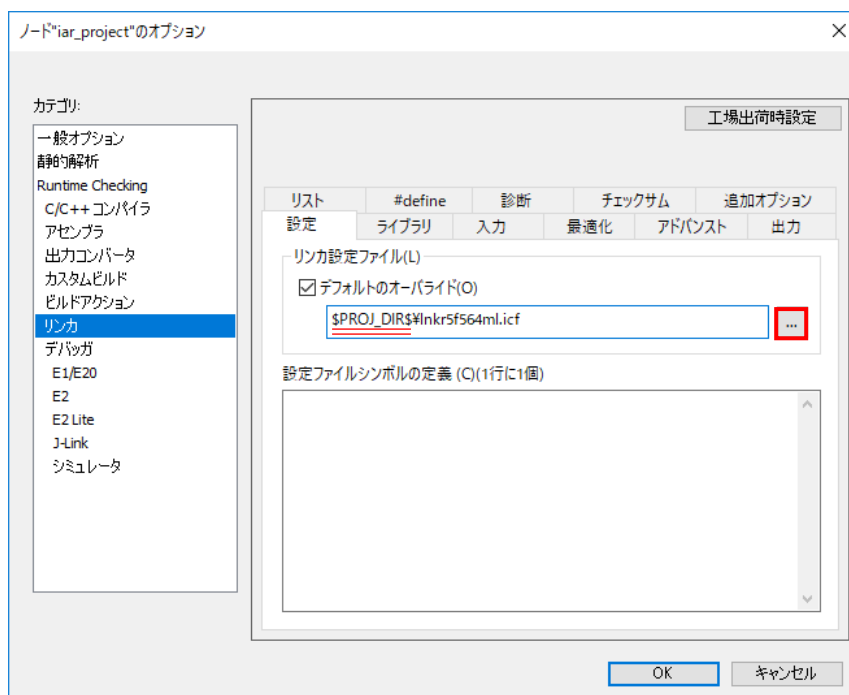
1. IAR 用のプロジェクトの「オプション」ウィンドウを開き、「カテゴリ:」から「リンカ」を選択し、「設定」タブを押下した後、リンカ設定ファイルで「デフォルトのオーバーライド」のチェックボックスがチェックされていることを確認し、「...」ボタンを押下します。



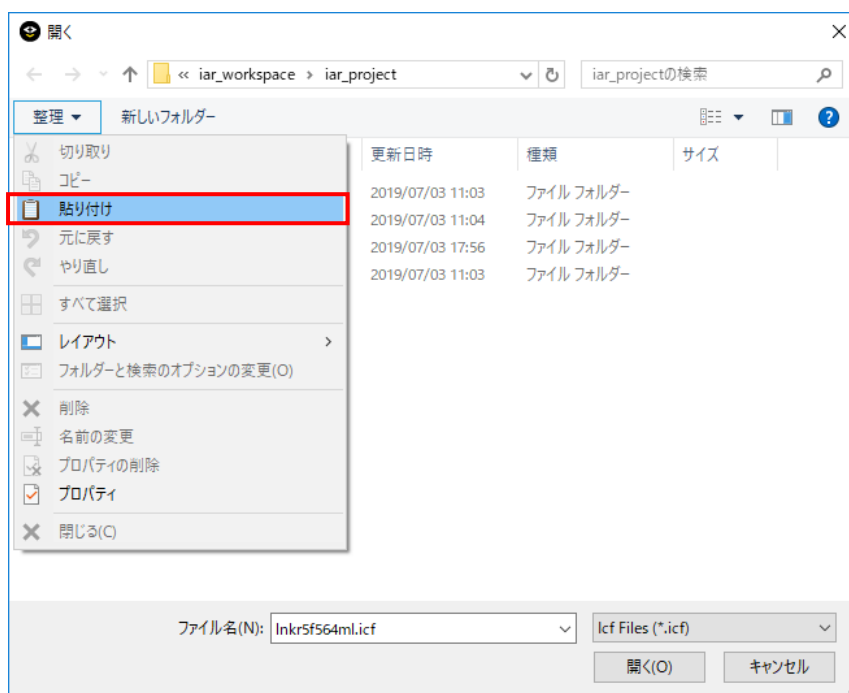
2. 「開く」ウィンドウでターゲットデバイスの.icf ファイル(1.の「オプション」ウィンドウのリンカ設定ファイルのテキストボックスの二重下線部分)をコピーし、「キャンセル」ボタンを押下します。



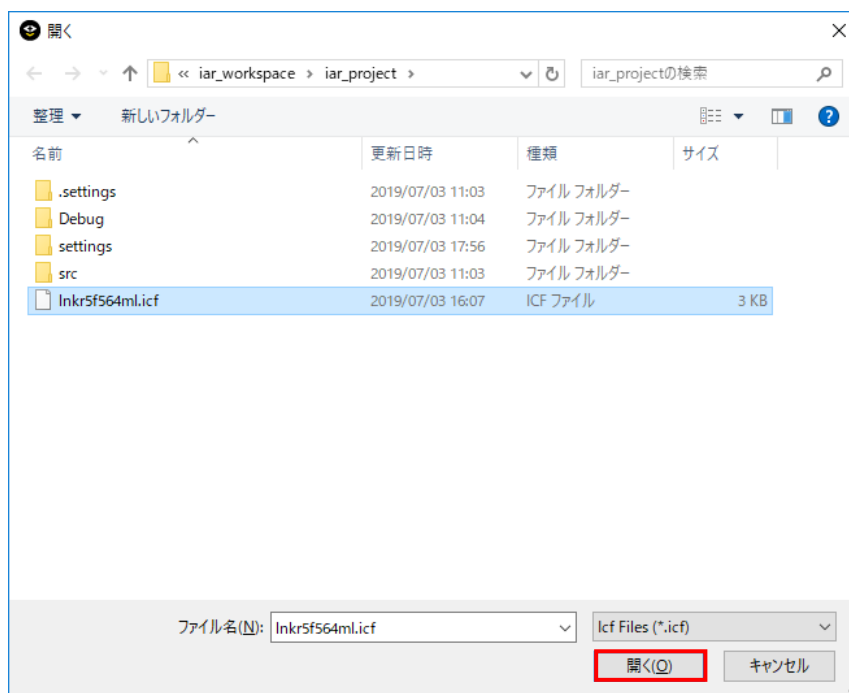
3. 「オプション」ウィンドウのリンカ設定ファイルまでのパスを任意の場所に書き換えます。(ここではパス変数として"\$PROJ_DIR\$"を使用し、プロジェクトフォルダの直下となるようにしています) 書き換えた後、再び、「...」ボタンを押下します。



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カ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カ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 .textw, 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 .textw, 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: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 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	—	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。
		14	「2.16 デュアルバンクの動作」を追加。
		28	「3.6 R_FLASH_Control()」の Description にコマンド "FLASH_CMD_BANK_xxx"を追加。
		37	「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。
4.60	Jun.24.20	5-9 11 17-23 25-27 29 30 32-36 37-68 80	「1. 概要」の章の構成および記載内容の見直し。 「2.7 コンパイル時の設定」の章の記載内容の見直し。 「2.9 引数」の章の構成および記載内容の見直し。 「2.11 コールバック関数」の章を追加。 「2.13 ブロッキングモード、ノンブロッキングモード」の記載内容の見直し。 「2.14 アクセスウィンドウ、ロックビットによる領域の保護」の章を追加。 「2.16 フラッシュメモリの書き換え」の章の構成および記載内容の見直し。 「3. API 関数」の章の構成および記載内容の見直し。 「5.2 トラブルシューティング」の章を更新。

Rev.	発行日	改訂内容	
		ページ	ポイント
		93	「5.3.3 IAR C/C++ Compiler for Renesas RX を使用する場 合」の章を更新
		ブ ロ ッ ク	R_FLASH_Open()が実行済か否かの判定処理を追加。 ブロック 0 を含めたアクセスウィンドウの処理を見直し。 IEN の有効、無効をフラッシュモジュール内で処理するよう 見直し。 不要な定義の削除等、軽微な内容についての見直し。
4.70	Oct.23.20	1、5、6、 34、42、 45、48、51 79	RX671 のサポートを追加。 「5.1 動作確認環境」の章を更新。
		ブ ロ ッ ク	RX671 のサポートを追加。

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も、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いたします。
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。