RX ファミリ
フラッシュモジュール Firmware Integration Technology

R01AN2184JU0340
Rev.3.40
2018.07.17

要旨

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

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

フラッシュ FIT モジュールは RX600 シリーズ、RX200 シリーズ対象のシンプルフラッシュ API とは異なります(R01AN0544JU)。

フラッシュ FIT モジュールで提供されるソースファイルは、ルネサス RX コンパイラのみに準拠しています。

対象デバイス

- RX110、RX111、RX113 グループ
- RX130 グループ
- RX210、RX21A グループ
- RX220 グループ
- RX231、RX230 グループ
- RX23T、RX24T グループ
- RX24U グループ
- RX610 グループ
- RX621、RX62N、RX62T、RX62G グループ
- RX630、RX631、RX63N、RX63T グループ
- RX64M グループ
- RX651、RX65N グループ
- RX66T グループ
- RX71M グループ

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

関連アプリケーションノート

- 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)
- e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)
- CS+に組み込む方法 Firmware Integration Technology (R01AN1826)

目次

1. 概要

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

1.1 機能説明

フラッシュ FIT モジュールでサポートしている機能を以下に示します。

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

1.2 BSP の使用に関するオプション

フラッシュ FIT モジュールは、rev. 2.00 より、BSP の使用有無に関わらずビルドすることが可能になりました。BSP を使用しない場合、クロック速度やメモリサイズといった r_bsp_config.h で設定されるフラッシュ依存の設定は、r_mcu_config.h で行います。

2. API 情報

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

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- フラッシュ

2.2 ソフトウェアの要求

本 FIT モジュールは以下のパッケージに依存しています。

- ルネサスボードサポートパッケージ（r_bsp）v3.91

2.3 制限事項

- 本 API のコードは再入不可で、複数関数の呼び出しが同時に発生しないように保護します（RESET は対象外）。
- コードフラッシュの書き換え中は、コードフラッシュにアクセスできません。コードフラッシュを書き換えるときは、アプリケーションコードは RAM から実行するようにしてください。

2.4 対応ツールチェーン

本 FIT モジュールは下記ツールチェーンで動作確認を行っています。

- Renesas RX Toolchain v.2.07.00

2.5 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_flash_rx_if.h に記載されています。このファイルは、Flash API 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選択あるいは定義できます。

BSP なしでビルドする場合、追加のコンフィギュレーションオプションが r_mcu_config.h ファイルで選択あるいは定義されます。

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 フラッシュのタイプと機能

フラッシュドライバは使用される技術およびシーケンサによって 4 つのタイプに分かれます。コンパイル後のフラッシュドライバのサイズはフラッシュのタイプが基準となります（2.9 参照）。

フラッシュタイプ 1

RX110*、RX111、RX113、RX130

RX230、RX231、RX23T*、RX24T、RX24U

* データフラッシュはありません。

フラッシュタイプ 2

RX210、RX220、RX21A

RX610

RX62G、RX62N、RX62T

RX630、RX631、RX63N、RX63T

フラッシュタイプ 3

RX64M、RX66T、RX71M

フラッシュタイプ 4

RX651*、RX65N**

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

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

```
#define FLASH_HAS_ISR_CALLBACK_CMD
#define FLASH_NO_BLANK_CHECK
#define FLASH_HAS_CF_BLANK_CHECK
#define FLASH_ERASE_ASCENDING_BLOCK_NUMS
#define FLASH_ERASE_ASCENDING_ADDRESSES
#define FLASH_HAS_ROM_CACHE
#define FLASH_HAS_DIFF_CF_BLOCK_SIZES
#define FLASH_HAS_BOOT_SWAP
#define FLASH_HAS_APP_SWAP
#define FLASH_HAS_CF_ACCESS_WINDOW
#define FLASH_HAS_DF_ACCESS_WINDOW
#define FLASH_HAS_INDIVIDUAL_CF_BLOCK_LOCKS
#define FLASH_HAS_SEQUENTIAL_CF_BLOCKS_LOCK
#define FLASH_HAS_ERR_ISR
```

2.8 コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_flash_rx_config.h で行います。
オプション名および設定値に関する説明を、下表に示します。

コンフィギュレーションオプション（r_flash_rx_config.h）(1/2)

定義	ﾃﾞﾌｫﾙﾄ値	説明
FLASH_CFG_USE_FIT_BSP	1	この定義を“1”に設定すると、r_bsp_config.h の定数を使ってビルドします。 “0”に設定すると、r_mcu_config.h の定数を使ってビルドします。
FLASH_CFG_PARAM_CHECKING_ENABLE	1	この定義を“1”に設定するとパラメータチェック処理のコードを生成し、“0”に設定すると生成しません。
FLASH_CFG_CODE_FLASH_ENABLE	0	データフラッシュのみを使用する場合は“0”に設定してください。 “1”に設定するとコードフラッシュ領域を書き換えるためのコードを生成します。コードフラッシュを書き換える際は、RAM からコードを実行する必要があります。ただし、フラッシュタイプ 3（ユーザーズマニュアル　ハードウェア編の表 63.18 参照）と RX65N-2M（ユーザーズマニュアル　ハードウェア編の表 59.15 参照）には制限付きで例外があります。 RAM からコードを実行するためのコードおよびリンカの設定については、2.13 を参照してください。 本 FIT モジュールの BGO モードに関する定義については、2.15 を参照してください。
FLASH_CFG_DATA_FLASH_BGO	0	この定義を“0”に設定すると、処理が完了するまで、データフラッシュの API 関数はブロックされます。 “1”に設定するとモジュールは BGO モード（BGO 動作／割り込み）になります。BGO モードでは、API 関数はデータフラッシュでの動作開始後すぐに復帰します。動作完了の通知はコールバック関数を使って行います。
FLASH_CFG_CODE_FLASH_BGO	0	この定義を“0”に設定すると、処理が完了するまで、コードフラッシュの API 関数はブロックされます。 “1”に設定するとモジュールは BGO モード（BGO 動作／割り込み）になります。BGO モードでは、API 関数はコードフラッシュでの動作開始後すぐに復帰します。動作完了の通知はコールバック関数を使って行います。コードフラッシュの書き換え時、可変ベクタテーブルとそれに対応する割り込み処理を、前もってコードフラッシュ以外の場所に配置する必要があります。2.17 の使用上の注意を参照してください。
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	0	この定義はフラッシュタイプ 3 およびコードフラッシュメモリ容量が 1.5M バイト以上の RX65N グループの製品に適用されます。 また、FLASH_CFG_CODE_FLASH_ENABLE が 1 に設定されている場合のみ有効です。 RAM のコードを実行しながら、コードフラッシュを書き換える場合は“0”に設定してください。 コードフラッシュの別のセグメントでコードを実行しながら、コードフラッシュを書き換える場合は“1”にしてください（2.14 参照）。

コンフィギュレーションオプション（r_flash_rx_config.h）(2/2)		
定義	ﾃﾞﾌｫﾙﾄ値	説明
FLASH_CFG_FLASH_READY_IPL	5	この定義はフラッシュタイプ 2 に適用され、割り込み優先レベルを定義します。
FLASH_CFG_IGNORE_LOCK_BITS	1	この定義はフラッシュタイプ 2 に適用されます。また、データフラッシュはロックビットに対応していないため、コードフラッシュのみに該当します。 イレーズ単位の各ブロックには対応するロックビットが備えられており、ロックビットが設定されたブロックにプログラム／イレーズが実行されないように保護します。 本定義を“1”に設定すると、ロックビットは無視され、ブロックへのプログラム／イレーズは制限されません。 “0”に設定すると、コントロールコマンドのユーザ設定に従ってロックビットが使用されます。

コンフィギュレーションオプション（r_mcu_config.h）		
定義	ﾃﾞﾌｫﾙﾄ値	説明
MCU_CFG_ICLK_HZ	(FIT BSP ﾃﾞﾌｫﾙﾄ)	MCU の ICLK の速度を設定します。 例: 80MHz の場合、80000000 を設定
MCU_CFG_FCLK_HZ	(FIT BSP ﾃﾞﾌｫﾙﾄ)	MCU のフラッシュクロックの速度を設定します。 例: 20MHz の場合、20000000 を設定
MCU_CFG_PART_MEMORY_SIZE	(FIT BSP ﾃﾞﾌｫﾙﾄ)	パート番号に示されているメモリサイズ（0x0～0xF）を設定します。設定可能な値は、r_mcu_config.h ファイルの本定義の直下でも確認できます。

2.9 コードサイズ

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

フラッシュタイプ 1: ROM および RAM の使用	
ROM の使用: PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 DATA_FLASH_BGO 1 > DATA_FLASH_BGO 0 CODE_FLASH_ENABLE 1 > CODE_FLASH_ENABLE 0 CODE_FLASH_BGO 1 > CODE_FLASH_BGO 0	
最小サイズ	ROM: 2198 バイト（データフラッシュがない場合は 2098 バイト）
	RAM: 84 バイト （データフラッシュがない場合は 84 + 1944 = 2028 バイト）
最大サイズ	ROM: 3386 バイト (データフラッシュがない場合は 2567 バイト)
	RAM: 84 + 2690 = 2774 バイト (データフラッシュがない場合は 84 + 2384 = 2468 バイト)

フラッシュタイプ 2: ROM および RAM の使用	
ROM の使用: PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 DATA_FLASH_BGO 1 > DATA_FLASH_BGO 0 CODE_FLASH_ENABLE 1 > CODE_FLASH_ENABLE 0 CODE_FLASH_BGO 1 > CODE_FLASH_BGO 0 IGNORE_LOCK_BITS 0 > IGNORE_LOCK_BITS 1	
最小サイズ	ROM: 2179 バイト
	RAM: 32 バイト
最大サイズ	ROM: 2953 バイト
	RAM: 44 + 2626 = 2670 バイト

フラッシュタイプ 3: ROM および RAM の使用	
ROM の使用: PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 CODE/DATA_FLASH_BGO 1 > CODE/DATA_FLASH_BGO 0 CODE_FLASH_ENABLE 1 > CODE_FLASH_ENABLE 0	
最小サイズ	ROM: 1797 バイト
	RAM: 64 バイト
最大サイズ	ROM: 3262 バイト
	RAM: 64 + 2900 = 2964 バイト

フラッシュタイプ 4: ROM および RAM の使用	
ROM の使用: PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 CODE/DATA_FLASH_BGO 1 > CODE/DATA_FLASH_BGO 0	
最小サイズ	ROM: 1967 バイト
	RAM: 64 + 1707 = 1771 バイト
最大サイズ	ROM: 2656 バイト
	RAM: 64 + 2368 = 2432 バイト

2.10 API データ構造体

API で使用されるデータ構造体は r_flash_rx_if.h に記載されています。API については 3 章で説明します。

2.11 戻り値

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

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

FLASH_ERR_CMD_LOCKED,    /* タイプ 3 – 周辺機能はコマンドロック状態 */
FLASH_ERR_LOCKBIT_SET,   /* タイプ 3 – ロックビットに起因するプログラム／イレーズエラー */
FLASH_ERR_FREQUENCY,     /* タイプ 3 – 不正な周波数(4-60Mhz) */
FLASH_ERR_ALIGNED,       /* タイプ 2 – 指定されたアドレスはコードフラッシュ、またはデータフラッシュに
                            アラインされていません。*/
FLASH_ERR_BOUNDARY,      /* タイプ 2 – 1M バイトの境界を越えて書き込めない箇所があります。*/
FLASH_ERR_OVERFLOW,      /* タイプ 2 – この動作の「アドレス+バイト数」がメモリ領域の */
                         /* 終端を超えました。 */
FLASH_ERR_BYTES,         /* 無効なバイト数 */
FLASH_ERR_ADDRESS,       /* 無効なアドレス */
FLASH_ERR_BLOCKS,        /* ブロック数を指定する引数が無効です。*/
FLASH_ERR_PARAM,         /* 不正な引数 */
FLASH_ERR_NULL_PTR,      /* 要求された引数がありません。 */
FLASH_ERR_UNSUPPORTED,   /* このフラッシュタイプではコマンドはサポートされていません。 */
FLASH_ERR_SECURITY,      /* タイプ 4 – FAW.FSPR による保護に起因するプログラム／イレーズエラー。 */
FLASH_ERR_TIMEOUT,       /* 時間切れです。 */
FLASH_ERR_ALREADY_OPEN /* Close()を呼び出さず、Open()を 2 回呼び出した。 */
} flash_err_t;
```

2.12 Flash FIT モジュールの追加方法

FIT モジュールのプロジェクトへの詳細な追加方法については、「e² studio に組み込む方法(R01AN1723)」を参照してください。

2.12.1 ソースツリーとプロジェクトのインクルードパスを追加する

1. e² studio で「ファイル > 新規 > Renesas FIT Module (v5.3.0 より前のバージョン)」、「Renesas Views > e2 ソリューション・ツールキット > FIT Configurator (v5.3.0 以降)」のような FIT ツールを使用するか、プロジェクトを生成するとき選択したスマート・コンフィグレータ(v5.3.0 以降)を使用します。これで、モジュールとプロジェクトにインクルードパスが追加されます。
2. e² studio で、プロジェクトのコンテキストメニューから「ファイル > インポート > 一般 > アーカイブファイル」を選択する。
3. Windows で ZIP ファイルをプロジェクトディレクトリに直接解凍して配置する。

上記方法 2 または 3 を使用する場合、インクルードパスはマニュアルでプロジェクトに追加する必要があります。インクルードパスを追加するには、「プロパティ > C/C++ビルド > 設定」を選択し、「Tool Settings」タブで「Compiler > ソース」を選択します。ウィンドウの上部にある「追加（緑色の“+”）」をクリックすると、インクルードパスの追加に使用するダイアログボックスが表示されます。ダイアログボックスで「ワークスペース」ボタンをクリックして、表示されるプロジェクトツリーから必要なディレクトリを選択します。本モジュールに必要なディレクトリを以下に示します。

- ${workspace_loc:/${ProjName}/r_flash_rx
- ${workspace_loc:/${ProjName}/r_flash_rx/src
- ${workspace_loc:/${ProjName}/r_flash_rx/src/targets
- ${workspace_loc:/${ProjName}/r_flash_rx/src/flash_type_1
- ${workspace_loc:/${ProjName}/r_flash_rx/src/flash_type_2
- ${workspace_loc:/${ProjName}/r_flash_rx/src/flash_type_3
- ${workspace_loc:/${ProjName}/r_flash_rx/src/flash_type_4
- ${workspace_loc:/${ProjName}/r_config

2.12.2 本 FIT モジュールおよび FIT BSP 使用上のオプションを設定する

フラッシュ特定のオプションは¥r_config¥r_flash_rx_config.h にあり、編集が可能です。
r_flash_rx_config_reference.h は、このファイルのデフォルト設定を含む参照用ファイル（編集用ではない）で、¥r_flash_rx¥ref に格納されています。
FIT BSP を使ってアプリケーションをビルドする場合、必要な作業は以上になります。
FIT BSP を使わずにアプリケーションをビルドする場合、以下に従ってこの参照用ファイルのコピーを適切な場所に置き、改名します。

コピー元: ¥r_flash_rx¥src¥targets¥<*mcu*>¥r_mcu_config_reference.h
コピー先: ¥r_config¥r_mcu_config_reference.h
改名後のファイル名: r_mcu_config.h

次に r_flash_rx_config.h で FLASH_CFG_USE_FIT_BSP を“0”に設定します。また、必要に応じて、r_mcu_config.h でクロック速度とメモリサイズを変更します。
API 関数の呼び出しを実行するアプリケーションファイルには、インタフェースファイル“r_flash_rx_if.h”（このファイルには “r_flash_rx_config.h”と必要に応じて間接的に“r_mcu_config.h”がインクルードされている）をインクルードする必要があります。このファイルには API 関数の宣言、および本 FIT モジュールを使用するために必要な全構造体と列挙型が含まれます。

2.12.3　プロジェクト生成ファイル（FIT BSP 以外）

プロジェクトジェネレータを使用している場合で、アプリケーションでフラッシュ割り込みを使用する場合、プロジェクトで生成される ISR テンプレートをコメント化する必要があります。

以下は必ずコメント化してください。

src¥vect.h:

```
#pragma interrupt (Excep_FCUIF_FRDYI(vect=23))
#pragma interrupt (Excep_FCUIF_FIFERR(vect=21))
```

src¥interrupt_handlers.c

```
void Excep_FCUIF_FRDYI(void){ }
void Excep_FCUIF_FIFERR(void){ }
```

2.12.4　バージョン 1.x からバージョン 2.x への移行

FIT BSP 使用時にバージョン 1.x から 2.x に移行する場合、新規のソースツリーをインストール後、以下のファイルをコピーして改名するのみで移行できます。

コピーするファイル: ¥r_flash_rx¥src¥targets¥<*mcu*>¥r_mcu_config_reference.h

コピー先: ¥r_config¥r_mcu_config_reference.h

改名後のファイル名: ¥r_mcu_config.h

2.12.5　バージョン 2.x からバージョン 3.20 への移行

FIT BSP 使用時にバージョン 2.x から 3.20 に移行する場合、新規のソースツリーをインストール後、¥r_config¥r_mcu_config.h ファイルを削除する必要があります。

FIT BSP を使用せずにバージョン 2.x から 3.20 に移行する場合、新規のソースツリーをインストール後、¥r_mcu_config_reference.h ファイルのコピーと改名は必要なままです。

コピーするファイル: ¥r_flash_rx¥src¥targets¥<*mcu*>¥r_mcu_config_reference.h

コピー先: ¥r_config¥r_mcu_config_reference.h

改名後のファイル名: ¥r_config¥r_mcu_config.h

これにより、以前の "r_mcu_config.h"ファイルが上書きされます。 他の操作は必要ありません。

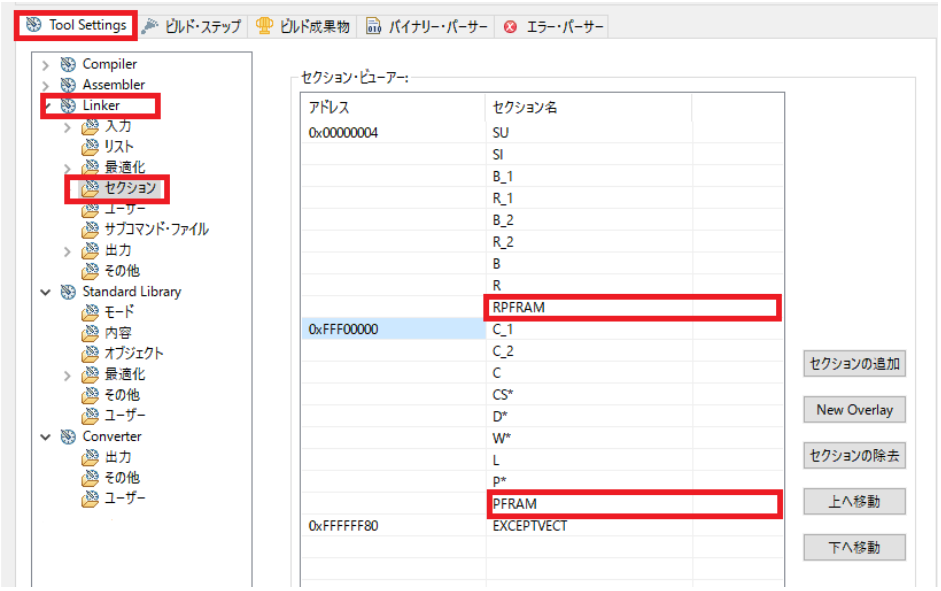
2.13　RAM からコードを実行してコードフラッシュを書き換える

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

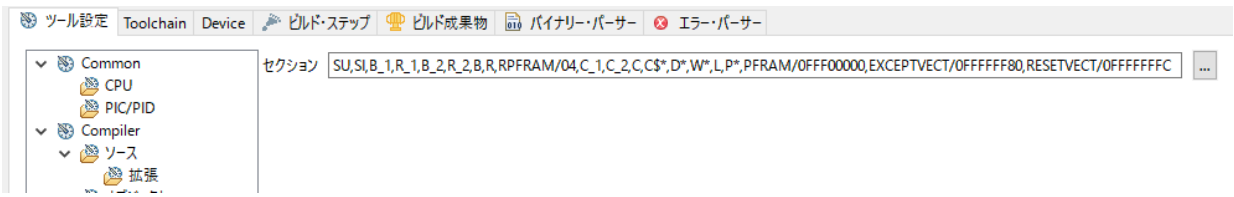
コードフラッシュの書き換えを有効にするために、r_flash_rx_config.h で“FLASH_CFG_CODE_FLASH_ENABLE”を“1”に設定します。これはコードフラッシュを書き換える場合にのみ適用されます。以下にコードフラッシュのプログラム／イレーズ手順を示します。

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

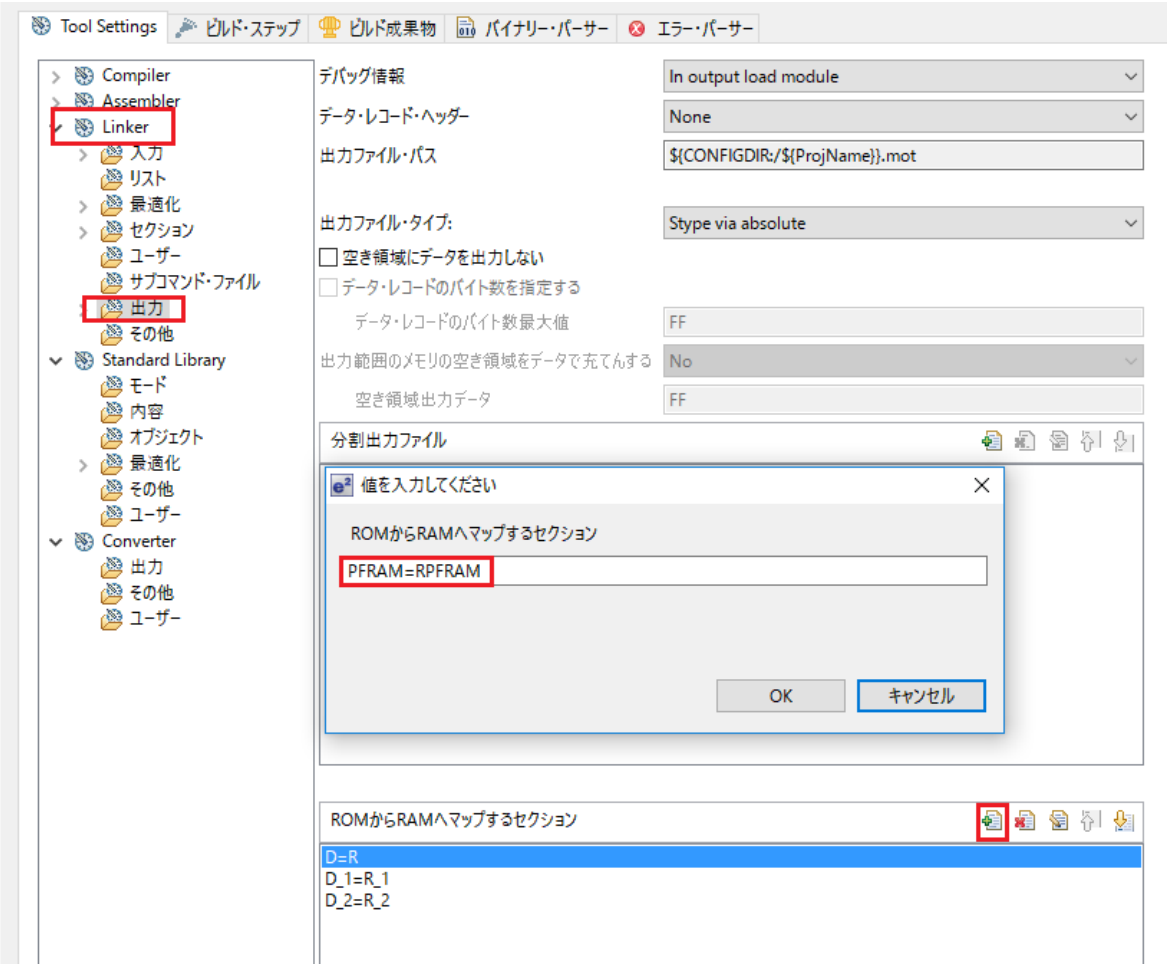
1. RAM 領域に新規セクション“RPFRAM”を追加します。
2. コードフラッシュ領域に新規セクション“PFRAM”を追加します。



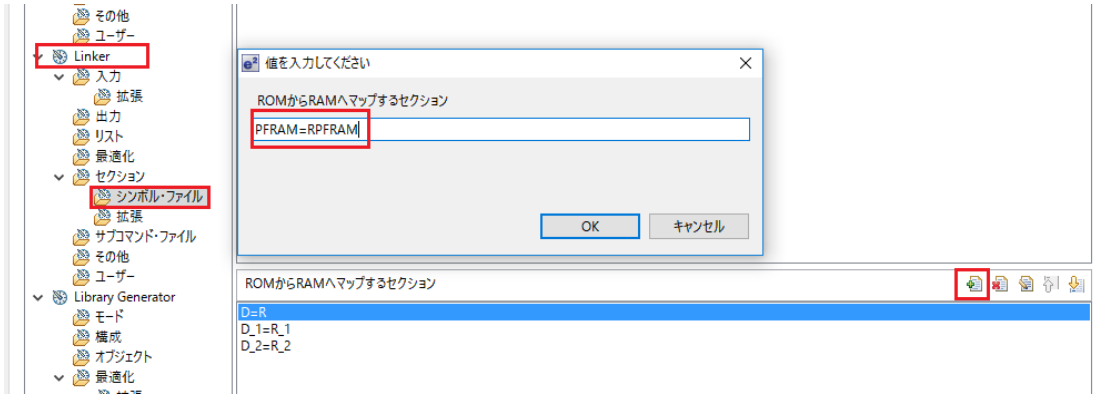
注: ご使用の e² studio バージョンにより、P セクションまたは P*セクションが存在するものがあります。P*セクションがある場合、PFRAM セクションを別で指定する必要はありません。
e² studio v6.0.0 以降を使用している場合、そのままではセクション・ビューアーは表示されません。セクション・ビューアーを表示するには、[セクション]エントリの右側にある[...]ボタンをクリックします。



3. リンカ出力オプションに “PFRAM=RPFRAM”を追加して（以下参照）、コードフラッシュのセクション (PFRAM) アドレスを RAM のセクション (RPFRAM) アドレスにマップします。これは v6.0.0 より前の e² studio では「ツール設定」タブで「Linker」の「出力」セクションから行います。



e² studio v6.0.0 以降を使用している場合、リンカ > セクション > シンボルファイルで “PFRAM = RPFRAM”を追加してください。



4. ここまででリンカが正しく設定され、適切な API コードが RAM に割り当てられました。コードフラッシュから RAM へのコードのコピーは、R_FLASH_Open()関数の呼び出し時に自動的に行われます。これが API 関数を呼び出す前に行われていないと、MCU は初期化していない RAM にジャンプすることになります。

5. 割り込みコールバック関数とコードフラッシュで動作するコードは FRAM セクション内に配置する必要があります。

```
#pragma section FRAM

/* コードフラッシュ上で動作する関数（および割り込みコールバック）はここに置く */
#pragma
```

2.14　コードフラッシュからコードを実行してコードフラッシュを書き換える

フラッシュタイプ 3 およびコードフラッシュメモリ容量が 1.5M バイト以上の RX65N グループの製品では、一定の条件を満たせば、コードフラッシュからコードを実行中にコードフラッシュを書き換えることができます。コードフラッシュは基本的に 2 つの領域に分けられます。片方の領域でコードを実行し、他方の領域でプログラム／イレーズを行います。領域のサイズは MCU のコードフラッシュの容量によって異なります。領域の境界については、それぞれのユーザーズマニュアル　ハードウェア編で表 63.16（RX64M）および表 63.18（RX71M）を参照してください。この機能をサポートするのは、ROM 領域の大きい MCU のみです。コードフラッシュメモリ容量が 1.5M バイト以上の RX65N グループの製品では、境界はバンクの境界と一致します。

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

FLASH_CFG_CODE_FLASH_BGO（完了待ち）の設定は“0”でも“1”でも構いませんが、データフラッシュの BGO 設定と一致する必要があります。

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

2.15　BGO モードでの動作

BGO モードは、通常、ドライバがノンブロッキングモードになることを言います。ノンブロッキングモードとは、コードフラッシュの動作がバックグラウンドで実行中に、RAM からの命令を実行できるモードです。ですが、フラッシュタイプ 3 と RX65N-2M のデバイスのユーザーズマニュアル　ハードウェア編では、BGO モードは、コードフラッシュの一方のセクションでコードを実行し、他方のセクションでプログラムが実行できる（2.14 参照）モードとして定義されています。

r_flash_config.h 内の#define 定義は、割り込みの使用やブロッキングモードでの動作に関わらず、通常の BGO の定義を使用しますので、フラッシュタイプ 3 と RX65N-2M デバイスに対応するため、BGO 機能への新規定義 “FLASH_CFG_CODE_FLASH_RUN_FROM_ROM”が提供されます。

BGO モードで動作している場合、API 関数はブロックされず、すぐに復帰します。ユーザは、フラッシュ領域で処理中の動作が完了するまで、その領域にアクセスしないでください。アクセスした場合、シーケンサはエラー状態になり、動作は正常に完了しません。

動作の完了は FRDYI 割り込みによって示されます。FRDYI 割り込み処理で処理の完了が確認され、コールバック関数が呼び出されます。コールバック関数を登録するには（フラッシュタイプ 2 以外）、"FLASH_CMD_SET_BGO_CALLBACK"コマンドを使って、R_FLASH_Control 関数を呼び出します。完了のステータスを示すイベントがコールバック関数に渡されます。一部の MCU 固有のイベントは“r_flash_rx_if.h”で以下のように定義されます。

```
typedef enum _flash_interrupt_event
{
    FLASH_INT_EVENT_INITIALIZED,
    FLASH_INT_EVENT_ERASE_COMPLETE,
    FLASH_INT_EVENT_WRITE_COMPLETE,
```

```
    FLASH_INT_EVENT_BLANK,
    FLASH_INT_EVENT_NOT_BLANK,
    FLASH_INT_EVENT_TOGGLE_STARTUPAREA,
    FLASH_INT_EVENT_SET_ACCESSWINDOW,
    FLASH_INT_EVENT_LOCKBIT_WRITTEN,
    FLASH_INT_EVENT_LOCKBIT_PROTECTED,
    FLASH_INT_EVENT_LOCKBIT_NON_PROTECTED,
    FLASH_INT_EVENT_ERR_DF_ACCESS,
    FLASH_INT_EVENT_ERR_CF_ACCESS,
    FLASH_INT_EVENT_ERR_SECURITY,
    FLASH_INT_EVENT_ERR_CMD_LOCKED,
    FLASH_INT_EVENT_ERR_LOCKBIT_SET,
    FLASH_INT_EVENT_ERR_FAILURE,
    FLASH_INT_EVENT_TOGGLE_BANK,
    FLASH_INT_EVENT_END_ENUM
} flash_interrupt_event_t;
```

フラッシュタイプ 2 では、単一のコールバック関数にイベントを渡すのではなく、ドライバがイベントカテゴリごとに予め定義されたコールバック関数を提供します（古いシンプルフラッシュ API と後方互換性あり）。

- FlashEraseDone(void)
- FlashBlankCheckDone(result)
- FlashWriteDone(void)
- FlashError(void)

コードフラッシュを書き換える場合、前もって、可変ベクタテーブルおよび関連する割り込みをコードフラッシュ以外の領域に配置する必要があります（フラッシュタイプ 3 と RX65N-2M では例外あり。2.14 参照。）

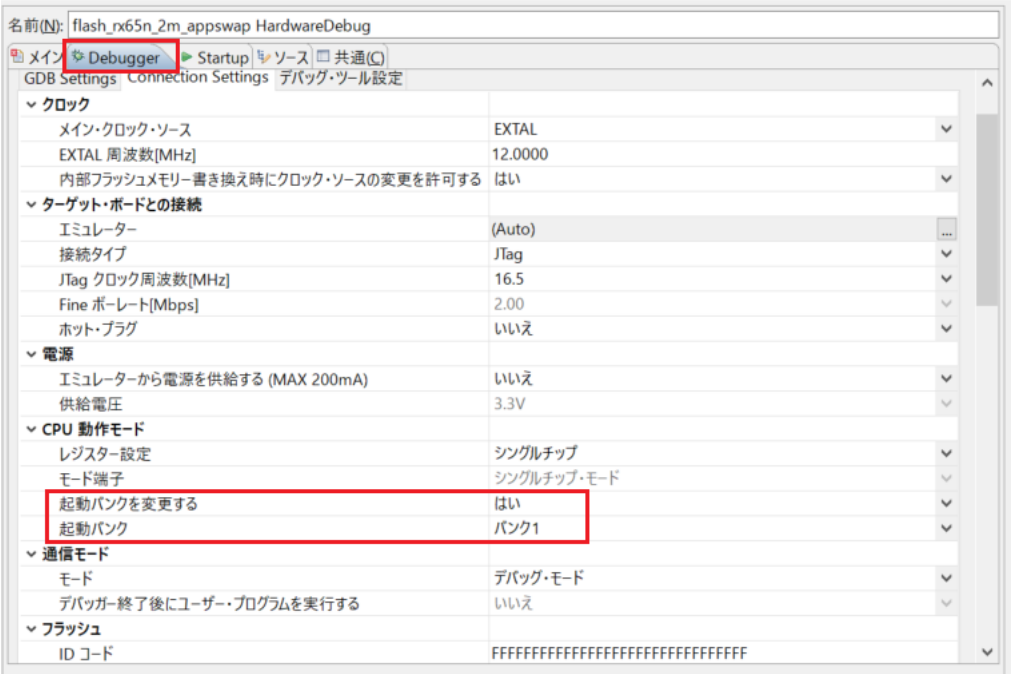
2.16　デュアルバンクの動作

コードフラッシュメモリ容量が 1.5M バイト以上の RX65N グループの製品はリニアとデュアルバンクの 2 種類のモードで動作できます。リニアモードは標準モードで、単一のアプリケーションがコードフラッシュ外で実行されます。デュアルバンクモードでは、2 つのアプリケーションをコードフラッシュに同時に読み込むことができます。コードフラッシュの上半分（固定ベクタテーブルが含まれる部分）に読み込まれたアプリケーションが実行されます。アプリケーションは R_FLASH_Control(FLASH_CMD_BANK_TOGGLE)コマンドを使って実行時に切り替えることができます。この切り替えは、次に MCU がリセットされるまで有効にはなりません。

アプリケーション開発時は、bsp_config.h で定義される以下の 2 つの定数を変更する必要があります。

- BSP_CFG_CODE_FLASH_BANK_MODE: 0 // デュアルモードの場合、“0”に設定（デフォルトではありません）
- BSP_CFG_CODE_FLASH_START_BANK: 1 // アプリケーションごとで値が異なる

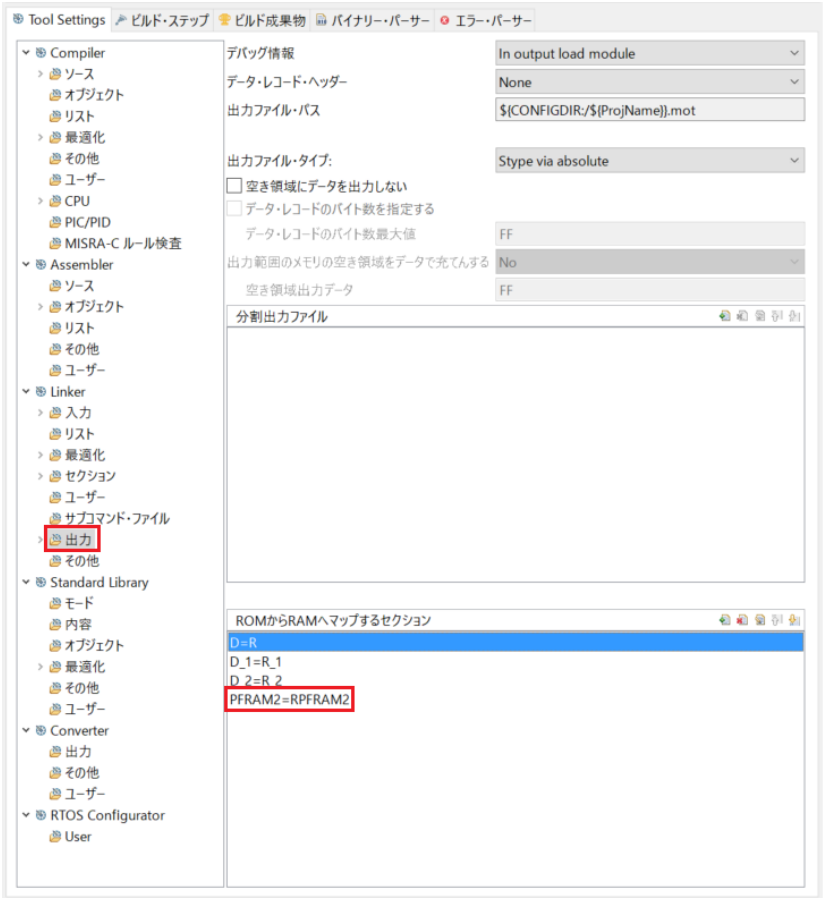
モードの定数はいずれのアプリケーションでも“0”に設定してください。スタートバンクは一方のアプリケーションで“1”に、もう一方では“0”に設定します。いずれのバンクもブートバンクにできます。ブート元のバンクがデバッグ設定で選択されます。



上記の例では、BSP_CFG_CODE_FLASH_START_BANK を“1”に設定してビルドしたアプリケーションがリセット時に実行されます。

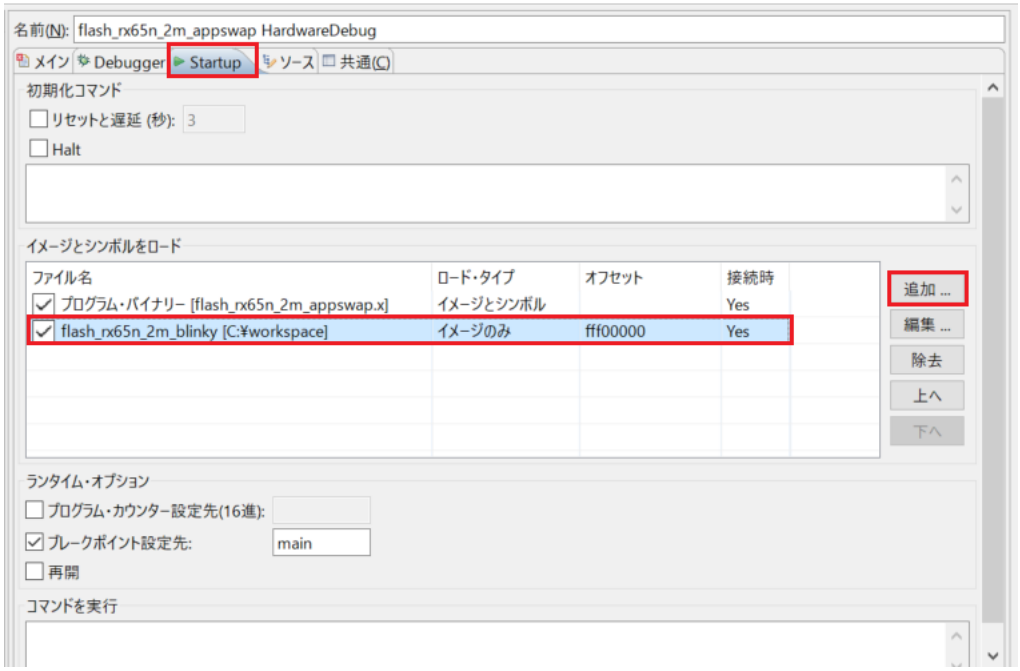
セクション 2.14 で述べたように、フラッシュ FIT モジュールは、RAM から実行せずとも他方のバンクのコードフラッシュに対してイレーズおよびプログラムが行えます（r_flash_config.h の FLASH_CFG_CODE_FLASH_RUN_FROM_ROM を“1”に設定）。ただし、バンクの切り替えを行うコードは RAM から実行する必要があります。これを可能にするために、リンカのセクションテーブルに“RPFRAM2”を追加し、マッピングを次ページで示すように出力します。

注: e^2 studio v6.0.0 以降では、右上に[...]ボタンが表示されます。このボタンは、以前のバージョンのようなセクション・ビューアーを表示するためには押す必要があります。



注: e² studio v6.0.0 以降では、[ROM から RAM へマップするセクション]はリンカ ＞ セクション ＞ シンボルファイルにありリンカ ＞ 出力ではありません。

エミュレータを使用する場合は、BSP_CFG_CODE_FLASH_START_BANK を“0”に設定してビルドしたアプリケーションをダウンロードします。これを実施すると、以下のようにスタートアップタブに追加されます。



2 つ目のアプリケーションのオフセットは-1Mb のオフセットを付ける必要があります。-1Mb のオフセットは 1Mb の 2 の補数(FFF00000)にしてください（開始アドレスではありません）。これによって、2 つ目の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。

現状の 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みのロード・タイプが「イメージとシンボル」になります。デバッグセッションが停止した場合、e²studio はこのシンボルテーブルを使ってソースコードと変数を表示します。ただし、表示される内容は、その時に実行していたアプリケーションのものとは限りません。そのため、ユーザは LED を使って実行中のアプリケーションが示されるようにすることが推奨されます。そうすることで、プログラムが一時停止した場合、表示されているソースコードが実行中のコードかどうかを判断できます。

これは、最初にアプリケーションが完璧にデバッグされ、またシンボルテーブルが必要に応じていつでも変更可能な場合、わずかな制限に過ぎません。

2.17 使用上の注意

2.17.1 BGO モードでのデータフラッシュの動作

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

2.17.2 BGO モードでのコードフラッシュの動作

BGO／ノンブロッキングモードでコードフラッシュを書き換える場合、外部メモリ、および RAM へのアクセスが可能です。フラッシュ FIT モジュールの API 関数がコードフラッシュの動作完了前に復帰するため、API 関数を呼び出すコードは RAM に配置します。また、その他のフラッシュコマンドを発行する前に、処理中の動作の完了を確認する必要があります。コマンドにはコードフラッシュのアクセスウィンドウの設定、ブートブロックの切り替え／スタートアップ領域フラグのトグル、コードフラッシュのイレーズ、コードフラッシュのプログラム、また、他の FIT モジュール(R01AN2191JJ)を使ったユニーク ID の呼び出しが含まれます。

2.17.3 コードフラッシュの動作と割り込み

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

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

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

3. API 関数

3.1 概要

本 FIT モジュールには以下の関数が含まれます。

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

3.2 R_FLASH_Open()

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

Format

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

Parameters

なし

Return Values

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

*FLASH_ERR_BUSY　　/*他のフラッシュ動作が処理中です。後から再試行してください。 */*

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

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

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

Reentrant

この関数は再入不可です。

Example

```
flash_err_t err;

/* API の初期設定 */
err = R_FLASH_Open();

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

Special Notes:

なし

3.3 R_FLASH_Close()

フラッシュ FIT モジュールを終了する関数です。

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

Parameters
なし

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

FLASH_ERR_BUSY:　　*/*他のフラッシュ動作が処理中です。後から再試行してください。*/*

Properties
r_flash_rx_if.h にプロトタイプ宣言されています。

Description
本関数はフラッシュ FIT モジュールを終了します。フラッシュ割り込みが有効な場合はこれを無効化し、ドライバを初期化されていない状態に設定します。この関数が必要なのは、VEE（仮想 EEPROM）ドライバを使用するときのみです。その場合、R_VEE_Open()を呼び出す前に、フラッシュドライバを終了しておく必要があります（動作させてはいけません）。

Reentrant
この関数は再入不可です。

Example
```
flash_err_t err;

/* ドライバの終了 */
err = R_FLASH_Close();

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

Special Notes:
なし

3.4 R_FLASH_Erase()

コードフラッシュまたはデータフラッシュの指定したブロックをイレーズします。

Format

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

Parameters

block_start_address

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

num_blocks

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

Return Values

FLASH_SUCCESS */*正常動作（BGO モードが有効な場合、動作が正常に */*
*/*開始されたことを意味します。）*/*
FLASH_ERR_BLOCKS */*指定されたブロック数は無効です。*/*
FLASH_ERR_ADDRESS */*指定されたアドレスは無効です。*/*
FLASH_ERR_BUSY */*別のフラッシュ動作が処理中か、モジュールが初期化されていません。*/*
FLASH_ERR_FAILURE */*イレーズ失敗。シーケンサがリセットされました。または、*/*
*/*コールバック関数が登録されていません（BGO／ポーリングモードが有効な場合）。 */*

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

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

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

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

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

Reentrant

この関数は再入不可です。

Example

```
flash_err_t err;

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

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

Special Notes:

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

3.5 R_FLASH_BlankCheck()

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

Format

```
uint8_t R_FLASH_BlankCheck(uint32_t address,
                           uint32_t num_bytes,
                           flash_res_t *blank_check_result);
```

Parameters

address

ブランクチェックする領域のアドレス。この機能は MCU によって、データフラッシュ、コードフラッシュのいずれか、または両方に対応するもの、あるいは、いずれにも対応しないものがあります。

num_bytes

フラッシュタイプ 1、3、4 では、この引数でブランクチェックを実施するバイト数を示します。バイト数には、データフラッシュアドレスの場合は FLASH_DF_MIN_PGM_SIZE の倍数を、コードフラッシュアドレスの場合は FLASH_CF_MIN_PGM_SIZE の倍数を指定します。これらは MCU ごとに r_flash_rx¥src¥targets¥<*mcu*>¥r_flash_<*mcu*>.h で定義されます。

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

フラッシュタイプ 2 では、num_bytes は BLANK_CHECK_SMALLEST、または BLANK_CHECK_ENTIRE_BLOCK のいずれかになります。これらの値は、シンプルフラッシュ API のコードと互換性があります。BLANK_CHECK_SMALLEST は、FLASH_DF_MIN_PGM_SIZE がチェックされることを意味します。

*blank_check_result

API で設定されるポインタ。設定内容はブロッキングモード（BGO モードでない）でのブランクチェックの結果です。

Return Values

FLASH_SUCCESS	*/*正常動作（BGO モードが有効な場合、動作が正常に */*
	*/*開始されたことを意味します。）*/*
FLASH_ERR_FAILURE	*/*ブランクチェック失敗。シーケンサがリセットされました。または、*/*
	*/*コールバック関数が登録されていません（BGO モード、かつ */*
	*/*フラッシュ割り込みが有効な場合）。 */*
FLASH_ERR_BUSY	*/*別のフラッシュ動作が処理中か、モジュールが初期化されていません。*/*
FLASH_ERR_BYTES	*/* “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。*/*
FLASH_ERR_ADDRESS	*/*無効なアドレスが入力されました。または、アドレスが */*
	*/*最小プログラムサイズで割り切れません。*/*

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

プログラム対象のフラッシュ領域はブランクでなければなりません。

ブロッキングモードで動作時、ブランクチェックの結果は“blank_check_result”に入ります。この変数は flash_res_t 型で、r_flash_rx_if.h で定義されます。API が BGO／ノンブロッキングモードで使用される場合、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

この関数は再入不可です。

Example: フラッシュタイプ 1、3、4

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

```
flash_err_t err;
flash_res_t result;

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

Example: フラッシュタイプ 2

第 2 引数は BLANK_CHECK_SMALLEST（FLASH_DF_MIN_PGM_SIZE バイトをチェック）または BLANK_CHECK_ENTIRE_BLOCK でなければなりません。

```
flash_err_t err;
flash_res_t result;

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

Special Notes:

なし

3.6 R_FLASH_Write()

コードフラッシュ、またはデータフラッシュを書き換えます。

Format

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

Parameters

src_address

フラッシュに書き込むデータを格納したバッファへのポインタ。

dest_address

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

num_bytes

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

Return Values

FLASH_SUCCESS	*/*正常動作（BGO/ノンブロッキングモードの場合、動作が正常に */*
	*/*開始されたことを意味します。）*/*
FLASH_ERR_FAILURE	*/*プログラム失敗。書き込み先アドレスが、アクセスウィンドウ、または */*
	*/*ロックビットで制御されている可能性があります。またはコールバック*/*
	*/*関数が存在しません(BGO モード、 かつフラッシュ割り込み有効時)。*/*
FLASH_ERR_BUSY	*/*別のフラッシュ動作が処理中か、モジュールが初期化されていません。*/*
FLASH_ERR_BYTES	*/* 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。*/*
FLASH_ERR_ADDRESS	*/*無効なアドレスが入力されました。または、アドレスが */*
	*/*最小プログラムサイズで割り切れません。*/*

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

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

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

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

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

Reentrant

この関数は再入不可です。

Example

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

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

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

Special Notes:

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

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

3.7 R_FLASH_Control()

プログラム、イレーズ、ブランクチェック以外の機能を組み込みます。

Format

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

Parameters

cmd

実行するコマンド

*pcfg

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

Return Values

FLASH_SUCCESS	*/*正常動作（BGO モードの場合、動作が正常に開始されたことを */*
	*/*意味します。）*/*
FLASH_ERR_BYTES	*/*ブロック数が最大範囲を超えています。*/*
FLASH_ERR_ADDRESS	*/*コード／データフラッシュのブロックの開始アドレスが無効です。*/*
FLASH_ERR_NULL_PTR	*/*設定用構造体を要求するコマンドで使用する引数 “pcfg”が NULL です。*/*
FLASH_ERR_BUSY	*/*別のフラッシュ動作が処理中か、API が初期化されていません。*/*
FLASH_ERR_LOCKED	*/*フラッシュの制御回路がコマンドロック状態にあり、*/*
	*/*リセットされました。*/*
FLASH_ERR_ACCESSW	*/*アクセスウィンドウエラー: 指定された領域は不正です。*/*
FLASH_ERR_PARAM	*/*無効なコマンド */*

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

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

コマンド	引数	動作
フラッシュタイプ 1、2、3、4 FLASH_CMD_RESET	NULL	処理中の動作を中断し、シーケンサをリセットします。
フラッシュタイプ 1、2、3、4 FLASH_CMD_STATUS_GET	NULL	API の状態（Busy または Idle）を返します。
フラッシュタイプ 1、3、4 FLASH_CMD_SET_BGO_CALLBACK	flash_interrupt_config_t *	コールバック関数を登録します。
フラッシュタイプ 1、4 FLASH_CMD_ACCESSWINDOW_GET	flash_access_window_config_t *	コードフラッシュのアクセスウィンドウの境界を返します。

コマンド	引数	動作
フラッシュタイプ 1、2、4 FLASH_CMD_ACCESSWINDOW_SET	flash_access_window_config_t * (フラッシュタイプ 2 では構造体が異なります。)	（タイプ 1、4）コードフラッシュのアクセスウィンドウの境界を設定します。BGO／ノンブロッキングモードで使用する場合は、アクセスウィンドウの設定後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。** （タイプ 2）データフラッシュの読み出し／書き込みブロックイネーブルマスクを設定します。
フラッシュタイプ 1、4 FLASH_CMD_SWAPFLAG_GET	uint32_t *	スタートアップ領域設定モニタフラグ (SASMF (タイプ 1)、BTFLG (タイプ 4)) の現在の値を読み出します。
フラッシュタイプ 1、4 FLASH_CMD_SWAPFLAG_TOGGLE	NULL	スタートアッププログラム領域をトグルします。RAM に配置された関数で領域を切り替えます。領域の切り替え後は、コードフラッシュには戻らずに MCU をリセットします。BGO／ノンブロッキングモードで使用する場合は、領域の切り替え後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。**
フラッシュタイプ 1、4 FLASH_CMD_SWAPSTATE_GET	uint8_t *	スタートアップ領域選択ビットの現在の値 (SAS の値)を読み出します。
フラッシュタイプ 1、4 FLASH_CMD_SWAPSTATE_SET	uint8_t *	スタートアップ領域選択ビット (FISR.SAS)の値を、r_flash_rx_if.h で定義して設定します。 #define (value) FLASH_SAS_EXTRA (0) FLASH_SAS_DEFAULT (2) FLASH_SAS_ALTERNATE (3) FLASH_SAS_SWITCH_AREA (4) FLASH_SAS_EXTRA、FLASH_SAS_DEFAULT、または FLASH_SAS_ALTERNATE が設定された場合、値は FISR.SAS に直接設定され、その値によって領域が切り替えられます。FLASH_SAS_SWITCH_AREA が設定された場合、領域が即座に切り替えられます。切り替えは RAM に配置された関数で行います。リセット後の領域は FLASH_SAS_EXTRA で指定された領域となります。
フラッシュタイプ 2 FLASH_CMD_LOCKBIT_PROTECTION	flash_lockbit_enable_t *	引数を“false”に設定すると、ロックビットが設定されたブロックのイレーズ／プログラムを許可します。引数を“true”に設定すると、ロックビットが設定されたブロックのイレーズ／プログラムを禁止します。 注: ブロックをイレーズするとロックビットをクリアします。
フラッシュタイプ 2 FLASH_CMD_LOCKBIT_PROGRAM	flash_program_lockbit_config_t *	引数で指定したアドレスのブロックに対してロックビットを設定します。

コマンド	引数	動作
フラッシュタイプ 2、3 FLASH_CMD_LOCKBIT_READ	2) flash_read_lockbit_config_t * 3) flash_lockbit_config_t *	タイプ 2: 指定したブロックのロックビット情報（FLASH_LOCKBIT_SET、または FLASH_LOCKBIT_NOT_SET）が引数に設定されます。 タイプ 3: 指定したブロックのロックビット情報（FLASH_RES_LOCKBIT_STATE_PROTECTED、または FLASH_RES_LOCKBIT_STATE_NON_PROTECTED）が引数に設定されます。
フラッシュタイプ 3 FLASH_CMD_LOCKBIT_WRITE	flash_lockbit_config_t *	指定したブロックアドレスを先頭に、指定したブロック数に対して、ロックビットを設定します。
フラッシュタイプ 3 FLASH_CMD_LOCKBIT_ENABLE	NULL	ロックビットが設定されたブロックのイレーズ／プログラムを禁止します。
フラッシュタイプ 3 FLASH_CMD_LOCKBIT_DISABLE	NULL	ロックビットが設定されたブロックのイレーズ／プログラムを許可します。 注: ブロックをイレーズするとロックビットがクリアされます。
フラッシュタイプ 3、4 FLASH_CMD_CONFIG_CLOCK	uint32_t *	FCLK の動作速度 (Hz)。実行時にクロック速度を変更する場合にのみ呼び出しが必要です。
RX24T、RX24U、RX65x FLASH_CMD_ROM_CACHE_ENABLE	NULL	コードフラッシュのキャッシュを有効にします。(最初にキャッシュを無効にします。)
RX24T、RX24U、RX65x FLASH_CMD_ROM_CACHE_DISABLE	NULL	コードフラッシュのキャッシュを無効にします。コードフラッシュを書き換える前に呼び出してください。
RX24T、RX24U、RX65x FLASH_CMD_ROM_CACHE_STATUS	uint8_t *	キャッシュが有効な場合、“1”を設定してください(キャッシュが無効な場合は“0”)。
RX65N（コードフラッシュメモリ容量が 1.5M バイト以上の製品） FLASH_CMD_BANK_TOGGLE	NULL	スタートアップバンクを切り替えます。次のリセットで有効になります。BGO/ノンブロッキングモードで使用する場合は、バンク選択レジスタ設定後に FRDYI 割り込みが発生し、コールバック関数が呼び出されます。**
RX65N（コードフラッシュメモリ容量が 1.5M バイト以上の製品） FLASH_CMD_BANK_GET	flash_bank_t *	現在の BANKSEL 値を読み出します（バンクとアドレスは次のリセットで有効になる）。

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

Reentrant

本関数は再入不可です。ただし、FLASH_CMD_RESET コマンドは例外で、いつでも実行が可能です。

Example 1: BGO モードで、ポーリングを行う

フラッシュの動作完了待機中に、別の動作を行わずにループすることは、ブロッキングモードの通常動作と機能的に同じことです。フラッシュ動作の完了待機中に、別の処理を実行する必要がある場合に BGO モードを使用します。

```
flash_err_t err;

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

/* 動作完了の待機 */
while (R_FLASH_Control(FLASH_CMD_STATUS_GET, NULL) == FLASH_ERR_BUSY)
{
/* 重要なシステムチェックをここで実施 */
}
```

Example 2: フラッシュタイプ 1、3、4 で割り込みを使って BGO モードを設定する

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

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

  /* 可変ベクタテーブルを RAM に再配置 */

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


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

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

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

  ... (省略)
}

#pragma section FRAM

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

  /* ISR コールバック関数の処理 */
}
```

```
void do_rom_operations(void)
{

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

  ... (省略)
}

#pragma section
```

Example 3: 現在のアクセスウィンドウの範囲を取得する

```
flash_err_t          err;
flash_access_window_config_t access_info;

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

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

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

```
flash_err_t      err;
flash_access_window_config_t access_info;

/* コードフラッシュブロック 3 への書き込み許可 */

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

Example 5: データフラッシュのアクセスウィンドウを設定する（フラッシュタイプ 2）

領域保護は、データフラッシュブロックの不正な読み取り、プログラミングまたはイレーズを防ぐために使用されます。以下の例では、RX63N でブロック 64-127 のみに読み取り／書き換えを許可します。

```
flash_err_t                 err;
flash_access_window_config_t df_access;

/* アクセスウィンドウビットとデータフラッシュブロックの対応
 *      RX62*   RX63*       RX21*
 * b0   0       0-63        0-15
 * b1   1       64-127      16-31
 * b2   2       128-191     32-47
 * b3   3       192-255     48-63
 * b4   4       256-319
 * b5   5       320-383
 * b6   6       384-447
 * b7   7       448-511
 * b8   8       512-575
 * b9   9       576-639
 * b10  10      640-703
 * b11  11      704-767
 * b12  12      768-831
 * b13  13      832-895
 * b14  14      896-959
 * b15  15      960-1023
 */
/* RX63N でデータフラッシュブロック 64-127 の読み取り／書き換えを許可 */

df_access.read_en_mask = 0x0002;
df_access.write_en_mask = 0x0002;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&df_access);
if (FLASH_SUCCESS != err)
{
   printf("Control FLASH_CMD_ACCESSWINDOW_SET command failure.");
}
```

Example 6: 選択中のスタートアップ領域の値を取得する

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

```
uint32_t  swap_flag;
flash_err_t err;

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

Example 7: 選択中のスタートアップ領域を切り替える

以下の例では、スタートアップ領域のトグル方法を示します。RAM に配置された関数を使って領域を切り替えます。領域の切り替え後、コードフラッシュに戻らずに MCU をリセットします。

```
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)
{
  printf("Control FLASH_CMD_SWAPFLAG_TOGGLE command failure.");
}
```

Example 8: スタートアップ領域選択ビットの値を取得する

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

```
uint8_t   swap_area;
flash_err_t err;

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

Example 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(FISR.SAS)の設定方法を示します。RAM に配置された関数を使って領域を切り替えます。リセット後は“FLASH_SAS_EXTRA”で指定された領域が使用されます。

```
uint8_t   swap_area;
flash_err_t err;

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

Example 10: コードフラッシュの書き換えにキャッシュを使用する

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

```
uint8_t status;

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

/* メインコードをここに記載; 任意でフラッシュが有効かどうかを検証 */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_STATUS, &status);
if (status != 1)
{
  // エラー処理
}

/* コードフラッシュ書き換えの準備 */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_DISABLE, NULL);

/* イレーズ、プログラム、新規コードフラッシュ認証のコードをここに記載 */

/* キャッシュを再度有効にする */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);
```

Special Notes:

なし

3.8 R_FLASH_GetVersion ()

この関数は本 FIT モジュールのバージョン番号を返します。

Format

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

Parameters

なし

Return Values

バージョン番号

Properties

r_flash_rx_if.h にプロトタイプ宣言されています。

Description

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

Reentrant

この関数は、再入可能（リエントラント）です。

Example

```
uint32_t cur_version;

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

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

Special Notes:

この関数は“#pragma inline”を使用してインライン化されています。

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。

4.1 flash_demo_rskrx113

本デモは RSKRX113 を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、データフラッシュおよびコード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

4.2 flash_demo_rskrx231

本デモは RSKRX231 を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、データフラッシュおよびコード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

4.3 flash_demo_rskrx23T

本デモは RSKRX23T を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、データフラッシュおよびコード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

- RSKRX23T

4.4 flash_demo_rskrx130

本デモは RSKRX130 を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、データフラッシュおよびコード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

- RSKRX130

4.5 flash_demo_rskrx24T

本デモは RSKRX24T を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、データフラッシュおよびコードフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

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

対応ボード

- RSKRX24T

4.6 flash_demo_rskrx65N

本デモは RSKRX65N を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、データフラッシュおよびコードフラッシュのイレーズ、ブランクチェック、プログラムを行います。プログラム関数では、データをリードしてプログラムしたデータと比較します。コードフラッシュを書き換えるときは、“pragma section FRAM”と、リンカの対応セクションの定義（プロジェクトの「プロパティ」→「C/C++ビルド」→「設定」を選択し、「ツール設定」タブで「Linker」→「セクション」→「出力」を参照）を事前にご確認ください。

設定と実行:

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

対応ボード

- RSKRX65N-1

4.7 flash_demo_rskrx24U

本デモは RSKRX24U を使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、コード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

4.8 flash_demo_rx65n2mb_bank1_bootapp / _bank0_otherapp

本デモは RX65N-2MB デモ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、r_flash_rx API をブロッキングモードで使用して、次のリセットで BANKSEL レジスタを読み出して、バンク／アプリケーションを切り替えます。バンク 1 のアプリケーション実行時には LED1 を、バンク 0 のアプリケーション実行時には LED0 をそれぞれ点滅させます。

設定と実行:

1. flash_demo_rx65n2mb_bank1_bootapp をビルドして、flash_demo_rx65n2mb_bank0_otherapp をビルドします。
2. (HardwareDebug) flash_demo_rx65n2mb_bank1_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. LED1 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。LED0 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

4.9 flash_demo_rdkrx63n

本デモは RDKRX63N デモンストレーションキットを使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、コード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

- RDKRX63N

4.10 flash_demo_rskrx64m

本デモは RSKRX64M スターターキットを使ったシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、コード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

4.11 flash_demo_rskrx64m_romrun

本デモは RSKRX64M スターターキット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、r_flash_rx API をブロッキングモードで使用して、コード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:

3. サンプルコードをコンパイルしてダウンロードします。
4. 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

4.12 flash_demo_rskrx66T

本デモは e2 studio v6.2.0 用 RSKRX66T スターターキットを使用したシンプルなデモです。デモでは、r_flash_rx API をブロッキングモードで使用して、コード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

4.13 ワークスペースにデモを追加する

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デモの zip ファイルを選択して「完了」をクリックします。

e² studio v6.0.0 以降を使用している場合、プロジェクトを正しくビルドするためには、インポートした後にプロジェクトを更新する必要があります。これは、プロジェクトフォルダを右クリックし、[古い e2studio プロジェクトを更新]を選択することで行います。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.70	2016.10.31	—	初版発行
		ﾌﾟﾛｸﾞﾗﾑ	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	2017.02.17	—	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。
		ﾌﾟﾛｸﾞﾗﾑ	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	2017.02.17	—	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。
		ﾌﾟﾛｸﾞﾗﾑ	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	2017.04.28	7, 8	「2.9 コードサイズ」にて ROM および RAM のサイズを変更
		ﾌﾟﾛｸﾞﾗﾑ	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	2017.04.28	—	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。
		ﾌﾟﾛｸﾞﾗﾑ	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。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.20	2017.08.11	1,4,6,7 9-17 38,39	FIT モジュールの RX130-512KB 対応。 e² 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	2017.12.08	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	2018.07.17	1,5,6 15 15-16 44-46	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。

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部ROM、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みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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といたします。
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)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口　http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/