RE01 1500KB グループ

CMSIS software package R_FLASH ドライバ

R01AN4768JJ0100
Rev.1.00
2019.10.11

要旨

本アプリケーションノートは RE01 1500KB グループ CMSIS software package のフラッシュドライバ R_FLASH について説明します。以降、本ドライバをフラッシュドライバと称します。

フラッシュドライバを使って、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、フラッシュドライバの使用、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。

フラッシュドライバで提供されるソースファイルは、IAR コンパイラ、GNU GCC コンパイラのみに準拠しています。

対象デバイス

- RE01 1500KB 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

- RE01 1500KB グループ CMSIS パッケージを用いた開発スタートアップガイド(R01AN4660)

目次

1. 概要

フラッシュドライバを使って、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。本ドライバはコードフラッシュを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、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

フラッシュドライバ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のイレーズ、プログラム
- アクセスウィンドウによる領域の保護
- スタートアップ領域保護。この機能は、コードフラッシュのブロック 0～7 を安全に書き換えるための機能です。

2. API 情報

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

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- フラッシュ

2.2 ソフトウェアの要求

本ドライバは以下のパッケージに依存しています。

- R-CORE
- R_SYSTEM ドライバ

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）。

2.4 対応ツールチェーン

ドライバは下記ツールチェーンで動作確認を行っています。

- IAR Embedded Workbench for ARM 8.32.1.18631
- GNU GCC ARM(arm-none-eabi) v5.4.1.20160919

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_api.h に記載されています。このファイルは、R_FLASH 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_cfg.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。

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

本ドライバのコンフィギュレーションオプションの設定は、r_flash_cfg.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_cgf.h）

定義	ﾃﾞﾌｫﾙﾄ値	説明
FLASH_CFG_PARAM_CHECKING_ENABLE	1	この定義を“1”に設定するとパラメータチェック処理のコードを生成し、“0”に設定すると生成しません。
FLASH_CFG_CODE_FLASH_ENABLE	1 (固定)	コードフラッシュ領域を書き換えるためのコードを生成します。コードフラッシュを書き換える際は、RAM からコードを実行する必要があります。 RAM からコードを実行するためのコードおよびリンカの設定については、2.11 を参照してください。 本ドライバの BGO モードに関する定義については、2.13 を参照してください。
FLASH_CFG_CODE_FLASH_BGO	0	この定義を“0”に設定すると、処理が完了するまで、コードフラッシュの API 関数はブロックされます。 “1”に設定するとモジュールは 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、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。 2.14 の使用上の注意を参照してください。
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	0	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（2.12 参照）。

2.8 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、最適化なしを前提としたサイズです。ROM (コードおよび定数) と RAM (グローバルデータと RAM 上で実行されるコード)のサイズは、本ドライバ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。

IAR Embedded Workbench for ARM 8.32.1.18631	
PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 CODE_FLASH_BGO 1 > CODE_FLASH_BGO 0	
最小サイズ	ROM: 3772 バイト
	RAM: 3820 バイト (FLASH_CFG_CODE_FLASH_RUN_FROM_ROM が 1 の場合 48 バイト)
最大サイズ	ROM: 4162 バイト
	RAM: 4210 バイト (FLASH_CFG_CODE_FLASH_RUN_FROM_ROM が 1 の場合 48 バイト)

GNU GCC ARM(arm-none-eabi) v5.4.1.20160919	
PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 CODE_FLASH_BGO 1 > CODE_FLASH_BGO 0	
最小サイズ	ROM: 5932 バイト
	RAM: 5930 バイト (FLASH_CFG_CODE_FLASH_RUN_FROM_ROM が 1 の場合 46 バイト)
最大サイズ	ROM: 6372 バイト
	RAM: 6370 バイト (FLASH_CFG_CODE_FLASH_RUN_FROM_ROM が 1 の場合 46 バイト)

2.9 API データ構造体

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

2.10　戻り値

API 関数の戻り値を示します。この列挙型は、r_flash_api.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 など*/
FLASH_ERR_CMD_LOCKED,    /* 周辺機能はコマンドロック状態 */
FLASH_ERR_LOCKBIT_SET,   /* ロックビットに起因するプログラム／イレーズエラー */(非対応)
FLASH_ERR_FREQUENCY,     /* 不正な周波数(1MHz~32MHz 以外) */
FLASH_ERR_ALIGNED,       /* 指定されたアドレスはコードフラッシュにアラインされていません。*/
FLASH_ERR_BOUNDARY,      /* 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,      /* 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,   /* コマンドはサポートされていません。 */
FLASH_ERR_SECURITY,      /* AWS.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;
```

2.11 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、API 関数を RAM にコピーします。RAM は、リセット後に初期化する必要があります。

API 関数を RAM にコピーする場合は、r_flash_cfg.h で FLASH_CFG_CODE_FLASH_RUN_FROM_ROM の設定を”0”にしてください。また、コードフラッシュの書き換えを有効にするために、r_flash_cfg.h で FLASH_CFG_CODE_FLASH_ENABLE を”1”に設定してください。

RAM へのコピーは R_SYSTEM ドライバの API 関数を使用します。以下に、R_SYSTEM ドライバを使用して、API 関数を RAM にコピーする例を示します。

```
int main()
{
    /* ROM から RAM にコードをコピーする */
    R_SYS_CodeCopy();

    R_SYS_Initialize();

    …
}
```

API 関数を RAM にコピーする方法の詳細は、関連アプリケーションノートを参照してください。

2.12 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

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 3 つの領域に分けられます。1 つの領域でコードを実行し、他の領域でプログラム／イレーズを行います。この方法を使用する場合、r_flash_cfg.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。

2.11 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。

2.13 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。

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、"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_api.h”で以下のように定義されます。

```
typedef enum
{
    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_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_END_ENUM
} flash_interrupt_event_t;
```

コードフラッシュを書き換える場合、前もって、ベクタテーブルおよび関連する割り込み処理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。

2.14 使用上の注意

2.14.1 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。フラッシュドライバの 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含まれます。

2.14.2 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、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処理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、ベクタテーブルオフセットレジスタ(VTOR)も変更が必要です。

R_SYSTEM ドライバを使用することにより、ベクタテーブルおよび割り込み処理を再配置することができます。

3. API 関数

3.1 概要

本ドライバには以下の関数が含まれます。

関数	説明
R_FLASH_Open()	フラッシュドライバを初期化します。
R_FLASH_Close()	フラッシュドライバを終了します。
R_FLASH_Erase()	コードフラッシュの指定ブロックをイレーズします。
R_FLASH_Write_8Bytes ()	コードフラッシュを 8 バイト単位で書き換えます。
R_FLASH_Write_256Bytes()	コードフラッシュを 256 バイト単位で書き換えます。
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()	本ドライバのバージョン番号を返します。

3.2 R_FLASH_Open()

フラッシュドライバ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 */* フラッシュドライバ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_api.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ドライバを初期化します。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。

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()

フラッシュドライバを終了する関数です。

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

Parameters
なし

Return Values
FLASH_SUCCESS: */* フラッシュドライバ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_api.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ドライバを終了します。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。

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“r_flash_re01_1500kb.h”で定義されます。ブロックはMCUのUMHの記載と同様にラベル付けされます。例えばUMHではアドレス0x00007000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。

num_blocks

イレーズ対象のブロック数を指定します。

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_BLOCKS */*指定されたブロック数は無効です。*/*

FLASH_ERR_OVERFLOW */*イレーズ対象範囲がコードフラッシュ領域を超えています。*/*

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／ノンブロッキングモードが有効な場合)。*/*

Properties

r_flash_api.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をイレーズします。ブロックサイズ（FLASH_CF_BLOCK_SIZE）は4Kバイトです。第一引数で指定したブロックを基準にブロック番号の大きい方(アドレス加算方向)にイレーズされます。

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;

/* コードフラッシュブロック 2~6 をイレーズ */
err = R_FLASH_Erase((uint32_t)FLASH_CF_BLOCK_2, 5);

/* 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（アクセスウィンドウでアクセス可能設定）でなければなりません。

3.5 R_FLASH_BlankCheck()

本関数は対応していません。

3.6 R_FLASH_Write_8Bytes ()

コードフラッシュを 8 バイト書き込みで書き換えます。

Format

```
flash_err_t R_FLASH_Write_8Bytes(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領域へのポインタ。アドレスには、プログラムサイズ(8 バイト)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、コードフラッシュメモリのプログラムサイズ(8 バイト)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_OVERFLOW	*/*プログラム対象範囲がコードフラッシュ領域を超えています。*/*
FLASH_ERR_BYTES	*/* 指定されたバイト数が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たか、アドレスが */*
	*/*プログラムサイズで割り切れません。*/*
FLASH_ERR_CMD_LOCKED	*/*FCU がコマンドロック状態になり、操作ができません。*/*

Properties

r_flash_api.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、プログラムサイズ(8 バイト)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プログラムサイズ(8 バイト)の倍数としてください。

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_8Bytes((uint32_t)write_buffer, dst_addr, sizeof(write_buffer));

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

Special Notes:

なし

3.7 R_FLASH_Write_256Bytes ()

コードフラッシュを 256 バイト書き込みで書き換えます。

Format

```
flash_err_t R_FLASH_Write_256Bytes(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領域へのポインタ。アドレスには、プログラムサイズ(256 バイト)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、コードフラッシュメモリのプログラムサイズ(256 バイト)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_OVERFLOW	*/*プログラム対象範囲がコードフラッシュ領域を超えています。*/*
FLASH_ERR_BYTES	*/* 指定されたバイト数が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たか、アドレスが */*
	*/*プログラムサイズで割り切れません。*/*

*FLASH_ERR_CMD_LOCKED/*FCU がコマンドロック状態になり、操作ができません。*/*

Properties

r_flash_api.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、プログラムサイズ(256 バイト)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プログラムサイズ(256 バイト)の倍数としてください。

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[256] = “Hello World...”;

/* 内部メモリにデータを書き込む */
err = R_FLASH_Write_256Bytes((uint32_t)write_buffer, dst_addr, sizeof(write_buffer));

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

Special Notes:

なし

3.8 R_FLASH_Control()

プログラム、イレーズ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_FAILURE /*コールバック関数が設定されていないか、ハードウェア操作に*/
/*失敗しました。*/

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

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

FLASH_ERR_PARAM /*無効なコマンド */

FLASH_ERR_FREQUENCY /* FLASH_CMD_CONFIG_CLOCK コマンド実行時に、無効な周波数が*/
/*渡されました。*/

FLASH_ERR_UNSUPPORTED/*サポートされていないコマンドです。*/

Properties

r_flash_api.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、イレーズ以外のシーケンサの機能を組み込むための拡張機能です。コマンドのタイプによって、引数の型も異なります。

コマンド	引数	動作
FLASH_CMD_RESET	NULL	シーケンサをリセットします。処理を中断してリセットするか、処理が完了してからリセットするかは処理内容に依存します。
FLASH_CMD_STATUS_GET	NULL	API の状態（Busy または Idle）を返します。
FLASH_CMD_SET_BGO_CALLBACK	flash_interrupt_config_t *	コールバック関数を登録します。

コマンド	引数	動作
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。
FLASH_CMD_ACCESSWINDOW_SET	flash_access_window_config_t *	コードフラッシュのアクセスウィンドウの境界を設定します。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。**
FLASH_CMD_SWAPFLAG_GET	uint32_t *	スタートアップ領域設定モニタフラグ (BTFLG) の現在の値を読み出します。
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。**
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。
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_api.h で定義して設定します。 #define (value) FLASH_SAS_EXTRA (0) FLASH_SAS_DEFAULT (2) FLASH_SAS_ALTERNATE (3) FLASH_SAS_SWITCH_AREA (4) FLASH_SAS_EXTRA、FLASH_SAS_DEFAULT、または FLASH_SAS_ALTERNATE が設定された場合、値は FISR.SAS に直接設定され、その値によって領域が切り替えられます。 FLASH_SAS_SWITCH_AREA が設定された場合、領域が即座に切り替えられます。切り替えは RAM に配置された関数で行います。リセット後の領域は FLASH_SAS_EXTRA で指定された領域となります。
FLASH_CMD_CONFIG_CLOCK	uint32_t *	ICLK 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。

**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

ブロッキングモードの場合、フラッシュの動作完了待機中に別の動作を実行することはできません。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は BGO モードを使用します。

```
flash_err_t err;

/* 全コードフラッシュをイレーズ */
R_FLASH_Erase(FLASH_CF_BLOCK_0, FLASH_NUM_BLOCKS_CF);

/* 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:割り込みを使って BGO モードを設定する

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、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_function;

    /* ROM から RAM にコードをコピーする */
    R_SYS_CodeCopy();

    /* ベクタテーブルを RAM に再配置する*/
    R_SYS_Initialize();

    /* API を初期設定 */
    err = R_FLASH_Open();
    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_function.pcallback = u_cb_function;
    cb_function.int_priority = 1;

    err = R_FLASH_Control(FLASH_CMD_SET_BGO_CALLBACK, (void *)&cb_function);
    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();

}

__attribute__ ((section(".ramfunc")))
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、イレーズ、
  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)
}
```

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

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 のみ書き換えを許可します。

```
flash_err_t      err;
flash_access_window_config_t access_info;

/* コードフラッシュブロック 3 への書き換え許可 */

access_info.start_addr = (uint32_t) FLASH_CF_BLOCK_3;
access_info.end_addr = (uint32_t) FLASH_CF_BLOCK_4;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
  printf("Control FLASH_CMD_ACCESSWINDOW_SET command failure.");
}
```

以下の例では、ブロック 0 からブロック 2 を書き換え許可に設定しています。

```
flash_err_t err;
flash_access_window_config_t access_info;

/* ブロック 0 からブロック 2 を書き換え許可に設定 */

access_info.start_addr = (uint32_t) FLASH_CF_BLOCK_0;
access_info.end_addr = (uint32_t) FLASH_CF_BLOCK_3;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
   printf("Control FLASH_CMD_ACCESSWINDOW_SET command failure.");
}
```

以下の例では、ブロック 380 からブロック 383 を書き換え許可に設定しています。

```
flash_err_t err;
flash_access_window_config_t access_info;

/* ブロック 380 からブロック 383 を書き換え許可に設定 */

access_info.start_addr = (uint32_t) FLASH_CF_BLOCK_380;
access_info.end_addr = (uint32_t) FLASH_CF_BLOCK_383 + FLASH_CF_BLOCK_SIZE;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
   printf("Control FLASH_CMD_ACCESSWINDOW_SET command failure.");
}
```

Example 5: 選択中のスタートアップ領域の値を取得する

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

```
uint32_t  swap_flag;
flash_err_t err;

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

Example 6: 選択中のスタートアップ領域を切り替える

以下の例では、スタートアップ領域のトグル方法を示します。RAM に配置された関数を使って領域を切り替えます。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, FLASH_NO_PTR);
if(FLASH_SUCCESS != err)
{
  printf("Control FLASH_CMD_SWAPFLAG_TOGGLE command failure.");
}
```

Example 7: スタートアップ領域選択ビットの値を取得する

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

```
uint8_t   swap_area;
flash_err_t err;

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

Example 8: スタートアップ領域選択ビットの値を設定する

以下の例では、スタートアップ領域選択ビット(FSUACR.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.");
}
```

Special Notes:

なし

3.9 R_FLASH_GetVersion ()

この関数は本ドライバ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_api.h にプロトタイプ宣言されています。

Description

この関数は本ドライバ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:

なし

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019.10.11	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。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)

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。