RX ファミリ

RX 用シンプルフラッシュ API

要旨

RX ファミリにはコード格納用フラッシュメモリ（ROM）とデータ格納用フラッシュメモリ（データフラッシュ）が搭載されています。これらのフラッシュメモリはユーザプログラム上で書き換えることができます。

本アプリケーションノートでは、ユーザプログラム上でコード格納用フラッシュメモリ（ROM）とデータ格納用フラッシュメモリ（データフラッシュ）を書き換えるためアプリケーションプログラミングインタフェース（API）を提供します。本ドキュメントでは、API 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を説明します。

対象デバイス

- RX610 グループ
- RX621、RX62N、RX62T、RX62G グループ
- RX630、RX631、RX63N、RX63T グループ
- RX210、RX21A、RX220 グループ

目次

1. 概要

フラッシュメモリを書き換えるためには、内蔵されているフラッシュコントロールユニット（FCU）を制御して複雑な手順を実行する必要があります。本アプリケーションノートが提供する API はその複雑な手順をユーザが気にすることなくフラッシュメモリを書き換えることができます。

1.1 サポートしている機能

本 API では以下の機能をサポートしています。

- ROM およびデータフラッシュの書き換え
- API の動作モード（ブロッキングモードとノンブロッキングモード）
- ROM から ROM、データフラッシュからデータフラッシュへの書き換え
- ロックビットによるプロテクト

1.2 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 1.1（左図）に示すように ROM 上に配置されます。図 1.1（右図）に示すように ROM 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（この例ではデータフラッシュ）の領域を書き換えることができます。

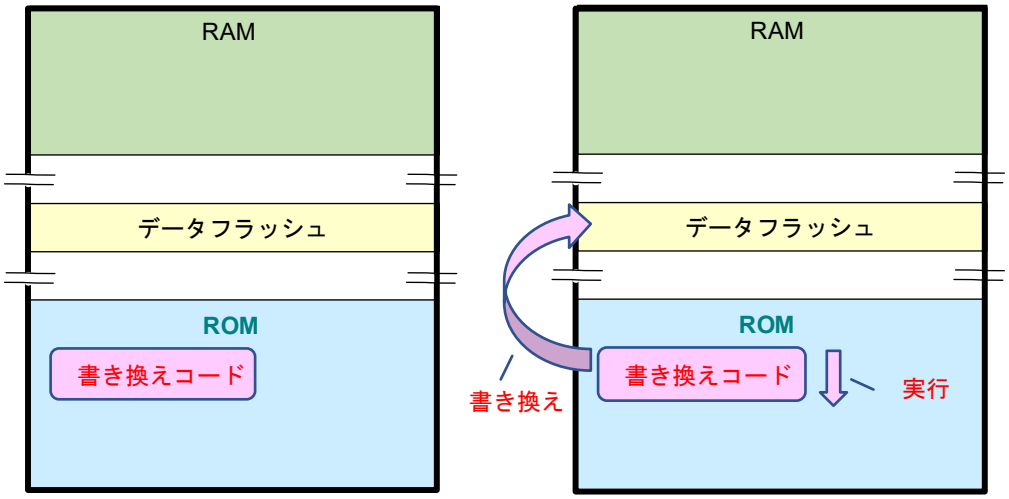


図 1.1 フラッシュメモリの書き換えに必要なコードの配置と書き換え動作

1.3　API の動作モード（ブロッキングモードとノンブロッキングモード）

本 API はブロッキングモードとノンブロッキングモードの 2 つのモードで動作します。

ブロッキングモードは API を呼び出し後、フラッシュメモリへの書き換え処理が完了してから戻るモードです。ノンブロッキングモードは API を呼び出し後、フラッシュメモリへの書き換え処理が完了する前に戻るモードです。API の動作モードの詳細は 2.6 ブロッキングモードとノンブロッキングモードを参照してください。

1.4　API 実行中のアクセス制限

FCU には、プログラムを実行したりデータを読み出したりするリードモード、フラッシュメモリを書き換えるためのプログラム/イレーズモード（P/E モード）があります。いくつかの API 関数を除き、API 関数を実行すると FCU は P/E モードに遷移してフラッシュメモリの書き換えを実行します。FCU が P/E モードである間、下記表のようにフラッシュメモリへのリードアクセスに制限があります。

もし P/E モード中に書き換え対象の領域をリードすると、読み出される値は不定です。

表 1.1　リードアクセスの制限

書き換え対象（P/E モード中）の領域	リードアクセス領域			
	ROM	データフラッシュ	RAM	外部メモリ
ROM	×	○	○	○
データフラッシュ	○	×	○	○

○：アクセス可能
×：アクセス禁止

そのため、図 1.2 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。

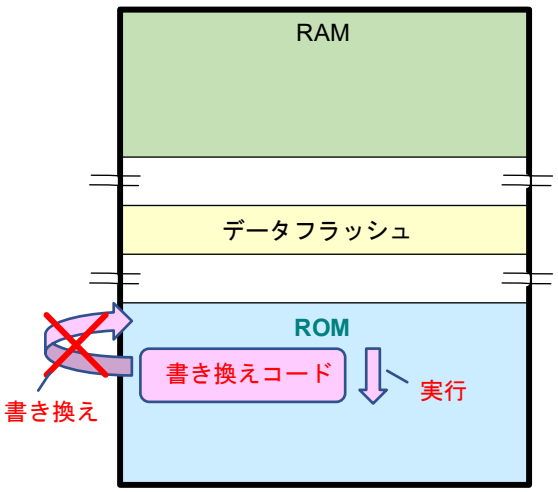


図 1.2 フラッシュメモリの書き換えに必要なコードが配置されている領域と同じ領域の書き換え

ROM を書き換えるためには、ROM 以外の領域、つまり RAM 領域（外部含む）でプログラムを実行する必要があります。また、書き込むデータも同様です。

1.5 API の実行領域と必要な作業について

フラッシュメモリを書き換えるためには、書き換え対象の領域とは異なる領域で API を実行する必要があります。

本 API のデフォルト設定では ROM 領域（P セクション、C セクション）に配置されますが、API のモードや書き換え対象領域によって API を実行する領域を変更する必要があります。また、それらに紐づき、割り込みベクタの配置の変更やコールバック関数の準備が必要です。

表 1.2 API モードによる対応

書き換え対象領域	モード	API の実行および割り込みベクタテーブルの配置可能な領域
データフラッシュ	ブロッキングモード	・RAM（外部含む） ・ROM
	ノンブロッキングモード	
ROM	ブロッキングモード	・RAM（外部含む）
	ノンブロッキングモード	

RAM からのプログラム実行については 2.8 API のコードを RAM から実行する方法、
割り込みベクタテーブルの移動については 2.7 割り込みのベクタテーブルと割り込み処理について
をそれぞれ参照してください。
また、API のモードがノンブロッキングモードの場合はコールバック関数を準備する必要があります。
コールバック関数の準備については 2.6.2 ノンブロッキングモード
を参照してください。

2. API 情報

2.1 ツールチェイン

本 API は以下のツールチェインで動作確認しています。

RX ファミリ用 C/C++コンパイラパッケージ(CC-RX) V3.04.00

2.2 ヘッダファイル

本 API をユーザプログラムで使用するには、r_flash_api_rx_if.h をインクルードしてください。また、本 API を使用するには r_flash_api_rx_config.h をユーザプログラムに合わせて設定してください。

2.3　コンフィグレーション

本 API の設定は r_flash_api_rx_config.h で行います。表 2.1 に r_flash_api_rx_config.h のコンフィグレーションオプションを示します。

表 2.1 コンフィグレーション項目（r_flash_api_rx_config.h）

コンフィグレーションオプション	
FLASH_MCU_xxxx	使用するデバイスに合わせて定義します。 例) RX63N の場合 #define FLASH_MCU_RX63N
FLASH_API_RX_CFG_ICLK_HZ	システムクロック（ICLK）の周波数を Hz で指定します。 例) 100MHz の場合 #define FLASH_API_RX_CFG_ICLK_HZ (100000000)
FLASH_API_RX_CFG_FCLK_HZ	RX610、RX62x グループ： 周辺モジュールクロック（PCLK）の周波数を Hz で指定します。 RX63x、RX2x グループ： FlashIF クロック（FCLK）の周波数を Hz で指定します。 例) 50MHz の場合 #define FLASH_API_RX_CFG_FCLK_HZ (50000000)
FLASH_API_RX_CFG_ROM_SIZE_BYTES	ROM サイズをバイト数で指定します。 サイズ指定用のマクロ定義（SIZE_xB）も使用できます。（本ヘッダファイル内で定義） 例) 2MB の場合 #define FLASH_API_RX_CFG_ROM_SIZE_BYTES (SIZE_2MB)
FLASH_API_RX_CFG_DATA_FLASH_SIZE_BYTES	データフラッシュサイズをバイト数で指定します。 サイズ指定用のマクロ定義（SIZE_xB）も使用できます。（本ヘッダファイル内で定義） 例) 32KB の場合 #define FLASH_API_RX_CFG_ROM_SIZE_BYTES (SIZE_32KB)
FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING	定義された場合、ROM の書き換えが有効になります。ROM の書き換えを有効にする場合は、2.8　API のコードを RAM から実行する方法に従って設定してください。 未定義の場合、データフラッシュの書き換えのみが有効です。

コンフィグレーションオプション	
FLASH_API_RX_CFG_FLASH_TO_FLASH	定義された場合、ROM から ROM への書き換え操作およびデータフラッシュからデータフラッシュへの書き換え操作が有効になります。有効の場合、書き込むデータを保持するための RAM バッファが必要です。RAM バッファのサイズは、各デバイスの ROM の最大書き込みサイズと同じです。
FLASH_API_RX_CFG_DATA_FLASH_BGO	定義された場合、ノンブロッキングモードによるデータフラッシュの書き換えを有効にします。データフラッシュの書き換えはバックグラウンドで実行され、API は書き換えが完了する前に戻ります。 書き換えが完了するとフラッシュレディ割り込み（FRDYI）が発生し、コールバック関数が実行されます。 未定義の場合、API はデータフラッシュの書き換えが完了してから戻ります。
FLASH_API_RX_CFG_ROM_BGO	定義された場合、ノンブロッキングモードによる ROM の書き換えを有効にします。ROM の書き換えはバックグラウンドで実行され、API は書き換えが完了する前に戻ります。 書き換えが完了するとフラッシュレディ割り込み（FRDYI）が発生し、コールバック関数が実行されます。 未定義の場合、API は ROM の書き換えが完了してから戻ります。
FLASH_API_RX_CFG_FLASH_READY_IPL	フラッシュレディ割り込みの割り込み優先レベルを指定します。 ノンブロッキングモードによる書き換えが有効な場合に有効な設定です。
FLASH_API_RX_CFG_IGNORE_LOCK_BITS	定義された場合、ロックビットプロテクト機能は無効になります。 未定義の場合、ロックビットプロテクトが有効になり、ロックビットがセットされたブロックに対しての書き込み／消去は失敗します。
FLASH_API_RX_CFG_COPY_CODE_BY_API	定義された場合、R_FlashCodeCopy 関数を有効にします。 本 API が配置されている ROM セクション（PFRAM）を RAM セクション（RPFRAM）にコピーする際、R_FlashCodeCopy 関数を使用せず、dbsct.c を編集する方法を用いる場合は、本定義を無効にしてください。

2.4 戻り値

API 関数の戻り値を以下に示します。これらは r_flash_api_rx_if.h で定義しています。戻り値の中には同じ値を持つものがありますが、いずれの API 関数も戻り値として同じ値を持つ定義を同時に使用することはありません。

```
/**** Function Return Values ****/
/* Operation was successful */
#define FLASH_SUCCESS          (0x00)
/* Flash area checked was blank, making this 0x00 as well to keep existing
   code checking compatibility */
#define FLASH_BLANK            (0x00)

/* The address that was supplied was not on aligned correctly for ROM or DF */
#define FLASH_ERROR_ALIGNED     (0x01)
/* Flash area checked was not blank, making this 0x01 as well to keep existing
   code checking compatibility */
#define FLASH_NOT_BLANK        (0x01)

/* The number of bytes supplied to write was incorrect */
#define FLASH_ERROR_BYTES       (0x02)
/* The address provided is not a valid ROM or DF address */
#define FLASH_ERROR_ADDRESS     (0x03)
/* Writes cannot cross the 1MB boundary on some parts */
#define FLASH_ERROR_BOUNDARY    (0x04)
/* Flash is busy with another operation */
#define FLASH_BUSY             (0x05)
/* Operation failed */
#define FLASH_FAILURE          (0x06)
/* Lock bit was set for the block in question */
#define FLASH_LOCK_BIT_SET      (0x07)
/* Lock bit was not set for the block in question */
#define FLASH_LOCK_BIT_NOT_SET  (0x08)
/* 'Address + number of bytes' for this operation went past the end of this
 * memory area. */
#define FLASH_ERROR_OVERFLOW (0x09)
```

2.5 ブロッキングモードとノンブロッキングモード

2.5.1 ブロッキングモード

ブロッキングモードは API を呼び出し後、フラッシュメモリへの書き換え処理が完了してから戻るモードです。ブロッキングモードで実行するには、コンフィグレーションオプションで以下のマクロ定義を無効にしてください。

- FLASH_API_RX_CFG_DATA_FLASH_BGO
- FLASH_API_RX_CFG_ROM_BGO

2.5.2 ノンブロッキングモード

ノンブロッキングモードは API を呼び出し後、フラッシュメモリへの書き換え処理が完了する前に戻るモードです。フラッシュメモリの書き換えが完了するとフラッシュレディ割り込み（FRDYI）が発生し、割り込み処理ルーチンからコールバック関数が実行されます。

ノンブロッキングモードで実行するには、コンフィグレーションオプションで以下のマクロ定義を有効にしてください。有効にした処理でのみノンブロッキングモードで動作します。

- FLASH_API_RX_CFG_DATA_FLASH_BGO
- FLASH_API_RX_CFG_ROM_BGO

ノンブロッキングモードで動作する API 関数は以下の関数のみです。それ以外の API 関数ではフラッシュレディ割り込み（FRDYI）、およびコールバック関数は実行されません。（ブロッキングモードで動作します。）

R_FlashErase 関数

R_FlashEraseRange 関数

R_FlashWrite 関数

R_FlashDataAreaBlankCheck 関数

API 関数によって以下のコールバック関数が実行されます。これらコールバック関数の本体はユーザが用意する必要があり、省略することはできません。

- void FlashEraseDone(void)
  - R_FlashErase 関数または R_FlashEraseRange 関数による ROM またはデータフラッシュの消去が完了したときに実行されます。

- void FlashWriteDone(void)
  - R_FlashWrite 関数による ROM またはデータフラッシュへの書き込みが完了したときに実行されます。

- void FlashBlankCheckDone(uint8_t result)
  - R_FlashDataAreaBlankCheck 関数によるデータフラッシュのブランクチェックが完了したときに実行されます。引数 result にはブランクチェックの結果が格納されます。結果がブランクなら FLASH_BLANK が格納され、ブランクでないなら FLASH_NOT_BLANK が格納されます。

- void FlashError(void)
  — 消去、書き込み、ブランクチェックが失敗したときに実行されます。本コールバック関数が実行されるまでにコマンドロックや各種エラーステータスは解除されていますので、ユーザが FCU に対して実行する処理はありません。ユーザシステムに応じて処理を実施してください。

2.6　割り込みのベクタテーブルと割り込み処理について

1.4 API 実行中のアクセス制限で説明しているリードアクセスの制限により、FCU がフラッシュメモリを書き換えるために P/E モードへ移行すると ROM から正しいデータを読み出すことができません。これは可変ベクタテーブル（割り込みベクタテーブル）や固定ベクタテーブル、割り込み処理のコードなどすべてに該当します。

P/E モード中に割り込み処理を実行するには割り込みベクタテーブルと割り込み処理のコードを本 API と同様に RAM 領域（外部含む）にコピーし、割り込みテーブルレジスタ（INTB）の値を変更します。
固定ベクタテーブルは配置を変更することができないため、P/E モード中に例外処理を発生させないよう注意が必要です。

以下に、割り込みベクタテーブルを RAM にコピーし、割り込みテーブルレジスタ（INTB）を変更する方法の例を示します。
一例ですので、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。

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

/* 割り込みベクタテーブルへアクセスするためのポインタ */
uint32_t *pvect_table;

/* ループ用変数 */
uint16_t i;


/* == 割り込みベクタテーブルを RAM に再配置 == */
pvect_table = (uint32_t *)__sectop("C$VECT");

for( i=0 ; i < 256 ; i++)
{
   ram_vect_table[i] = pvect_table[i]; /* FRDYI 割り込み関数のアドレスをコピー */
}

set_intb((void *)ram_vect_table);
```

2.7　API のコードを RAM から実行する方法

API のコードを RAM から実行するには、API を格納するセクションを RAM（RPFRAM）と ROM（PFRAM）の両方に用意し、リセット解除後に ROM セクション（PFRAM）から RAM セクション（RPFRAM）に API を含むコードをコピーします。

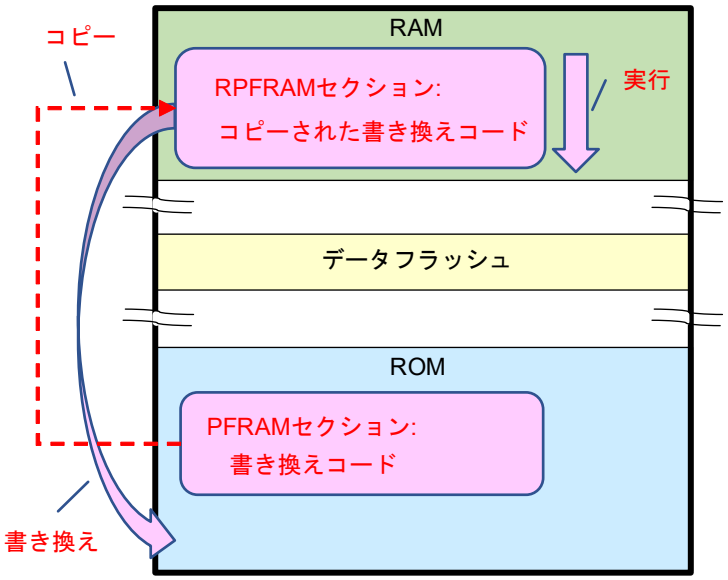


図 2.1 RAM からコードを実行して ROM を書き換える

以下の手順に従って設定してください。

(1) コンフィグレーションオプションの設定

#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING

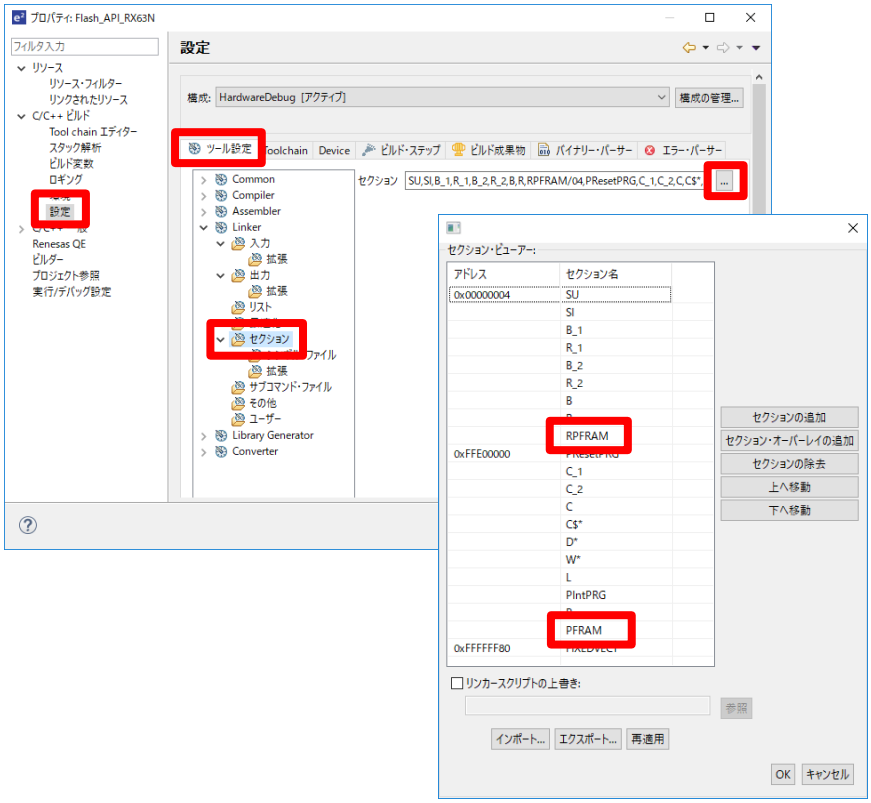
の定義を有効にしてください。

(2) セクションの追加

RAM 領域に‘**RPFRAM**’セクションを、ROM 領域に‘**PFRAM**’セクションを追加します。

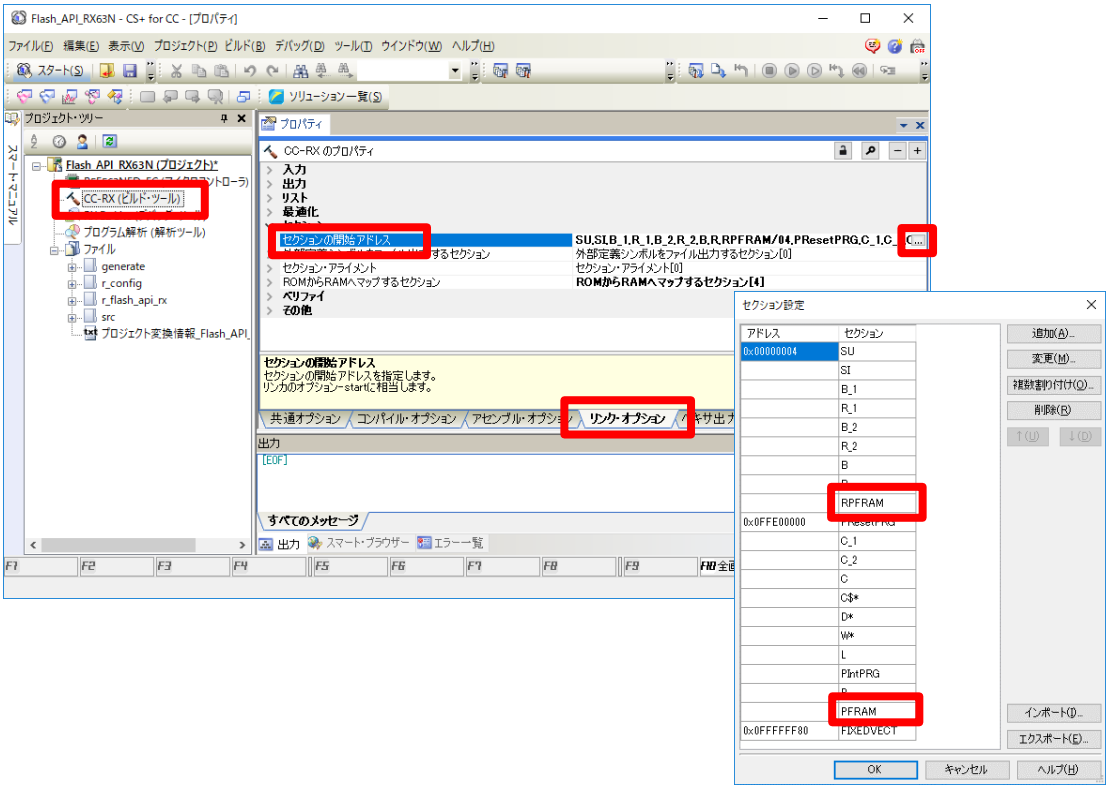
e^2 studio の場合

Linker >> セクションから「…」クリックすると「セクション・ビューアー」ウィンドウが開きます。下記図のように PFRAM セクションと RPFRAM セクションを追加します。



CS+の場合

CC-RX（ビルド・ツール）>> リンク・オプション >> セクション >> セクションの開始アドレス から「...」クリックすると「セクション設定」ウィンドウが開きます。下記図のように PFRAM セクションと RPFRAM セクションを追加します。

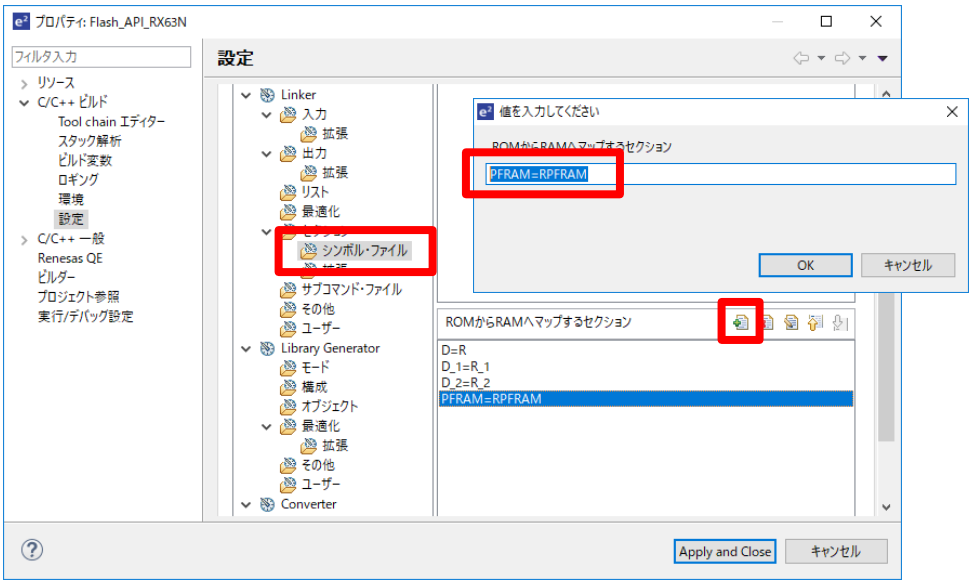


(3) ROM から RAM へのマッピング

ROM セクション（PFRAM）から RAM セクション（RPFRAM）へのリンカマップを作成するには、“ROM から RAM へマップするセクション”に以下の項目を追加します。

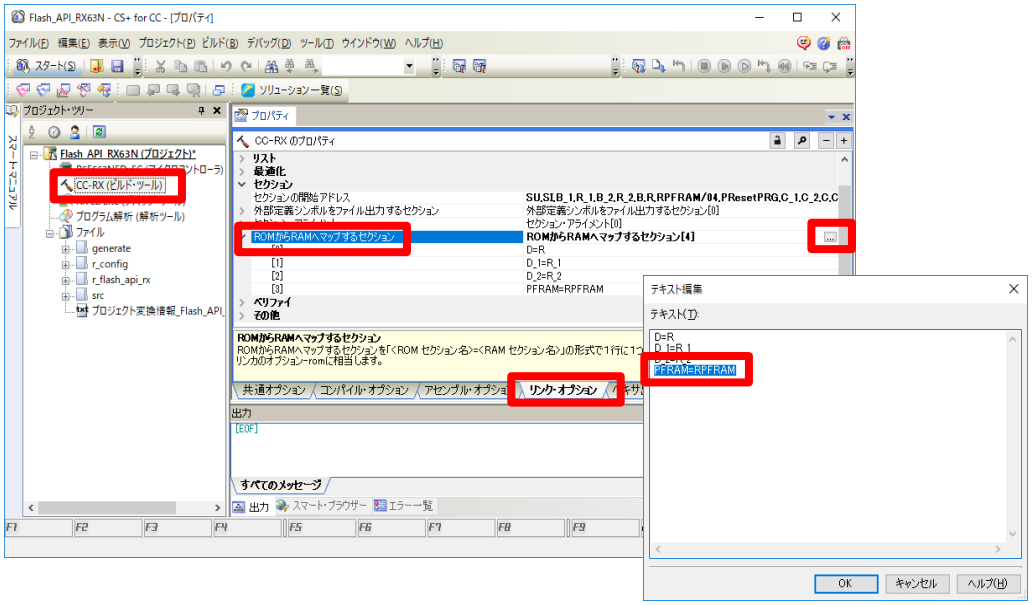
e² studio の場合

Linker >> セクション >> シンボル・ファイルから“ROM から RAM へマップするセクション”の‘追加’ボタンをクリックするとウィンドウが開きます。下記図のように PFRAM=RPFRAM を追加します。



CS+の場合

CC-RX（ビルド・ツール）>> リンク・オプション >> セクション >> ROM から RAM へマップするセクション から「…」クリックするとウィンドウが開きます。下記図のように PFRAM=RPFRAM を追加します。



(4) リセット解除後の ROM から RAM へのプログラム転送

ROM セクション（PFRAM）から RAM セクション（RPFRAM）にプログラムを転送します。

プログラムを RAM セクション（RPFRAM）にコピーするには、以下の 2 通りの方法があります。

1. dbsct.c ファイルの編集

dbsct.c ファイルは、リセット解除後に初期化する領域が指定されています。以下の赤文字で示すように、PFRAM セクションから RPFRAM セクションにプログラムを転送するためのコードを追加します。

```
-- FILE [dbsct.c] --
#pragma section $DSEC
static const struct {
    _UBYTE *rom_s;  /* Initial address on ROM of initialization data section */
    _UBYTE *rom_e;  /* Final address on ROM of initialization data section */
    _UBYTE *ram_s;  /* Initial address on RAM of initialization data section */
}   DTBL[] = {
    { __sectop("D"), __secend("D"), __sectop("R") } ,
    { __sectop("PFRAM"), __secend("PFRAM"), __sectop("RPFRAM") }
};
```

2. R_FlashCodeCopy 関数の実行

R_FlashCodeCopy 関数はプログラムを RAM セクション（RPFRAM）に転送する関数です。ユーザプログラムにおいて、他の API 関数を呼び出す前にこの関数を呼び出します。R_FlashCodeCopy 関数を使用するためには、r_flash_api_rx_config.h で FLASH_API_RX_CFG_COPY_CODE_BY_API を定義しておく必要があります。

2.8　ROM から ROM、データフラッシュからデータフラッシュへの書き換え

本機能はソフトウェアによる実装です。FCU に搭載されている機能ではありません。

デフォルトのコンフィグレーションオプションの設定では FLASH_API_RX_CFG_FLASH_TO_FLASH は未定義になっており、R_FlashWrite 関数の第 2 引数に指定できる書き込み元アドレス（buffer_addr）には書き込み先の領域のアドレスを指定できません。

しかし、コンフィグレーションオプションで FLASH_API_RX_CFG_FLASH_TO_FLASH の定義を有効にすると、書き込み元アドレス（buffer_addr）に書き込み先の領域のアドレスを指定することができます。例えば、書き込み先の領域が ROM の場合は ROM のアドレスを、データフラッシュの場合はデータフラッシュのアドレスを指定できます。R_FlashWrite 関数の詳細は、3.4 R_FlashWrite を参照してください。

本機能では R_FlashWrite 関数を実行すると、P/E モードに移行する前に書き込みデータを RAM に退避し、P/E モードに移行後は RAM に退避したデータを使用して書き込みます。そのため、FLASH_API_RX_CFG_FLASH_TO_FLASH の定義を有効にすると、書き込みデータの退避用に RAM バッファを確保します。RAM バッファのサイズは、各デバイスの ROM の最大書き込みサイズと同じです。

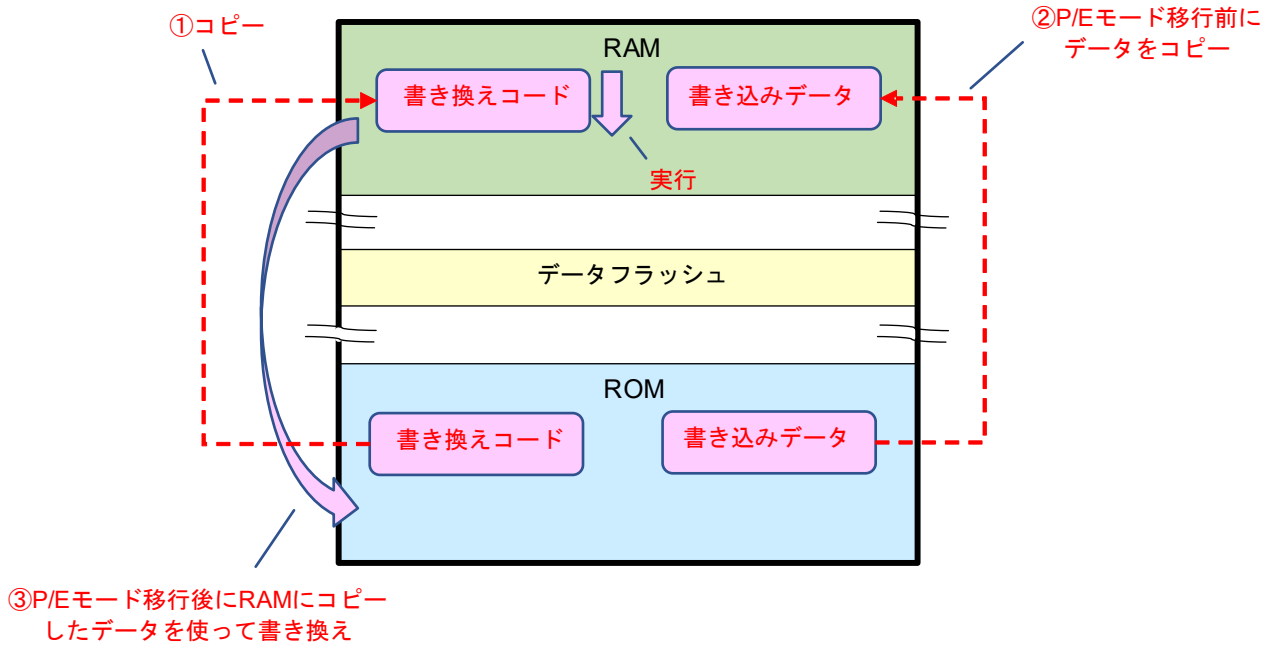


図 2.2 ROM から ROM の書き換え例

2.9 使用上の注意事項

2.9.1 API 実行時の動作周波数

API を実行するときには表 2.2 の範囲に設定してください。コンフィグレーションオプションの FLASH_API_RX_CFG_FCLK_HZ には動作周波数と同じ値を設定してください。

表 2.2 動作周波数

デバイス	FCU のクロックソース	周波数範囲
RX610 グループ RX62x グループ	周辺モジュールクロック（PCLK）	8MHz ～ PCLK の最大周波数
RX63x グループ RX2x グループ	FlashIF クロック（FCLK）	4MHz ～ FCLK の最大周波数

2.9.2 リセット解除後のデータフラッシュへのアクセス

リセット解除後の初期状態ではデータフラッシュへのアクセス（読み出しや書き込み、消去）は禁止です。データフラッシュへアクセスするには R_FlashDataAreaAccess 関数を使ってアクセスを許可する必要があります。詳細は 3.5 R_FlashDataAreaAccess を参照してください。

2.9.3 イレーズ後のフラッシュメモリの値

イレーズ後の値は ROM とデータフラッシュで異なります。イレーズ後、ROM は FFh が読めますが、データフラッシュからは不定値が読めます。データフラッシュがブランクかどうかを確認するには R_FlashDataAreaBlankCheck 関数を使用してください。

2.9.4 ブロックアドレスの定数

本 API では const 型の定数配列 g_flash_BlockAddresses[]を使用しています。本配列はフラッシュメモリの各ブロックの先頭アドレスを定義しています。ROM のアドレスは P/E モード用のアドレスで定義しており、読み出し用のアドレスではないことにご注意ください。また、本配列はデフォルトで ROM 領域（C セクション）に格納されます。ROM を消去する際に本配列を消去しないよう注意してください。本配列は r_flash_api_rx¥src¥targets 以下に格納されている各デバイス用のヘッダファイル（r_flash_api_rxXXX.h）に定義しています。

```
/* Data Structure */
const uint32_t g_flash_BlockAddresses[86] = {
   0x00FFF000,  /* EB00 */
   0x00FFE000,  /* EB01 */
   0x00FFD000,  /* EB02 */
   0x00FFC000,  /* EB03 */
   ...
};
```

2.9.5　ROM 領域の境界を跨ぐ書き込み

RX ファミリには複数の ROM 領域を持つ製品があります。例えば、ROM 容量が 2MB の RX63N グループには、4 つの ROM 領域（領域 0、1、2、および 3）があります。一度の API 関数の実行で書き込みできるのは同じ領域内に限られており、ROM 領域の境界を跨いだ書き込みはできません。ROM 領域の境界を跨ぐ書き込みを行うためには書き込みデータを分割して書き込みしてください。

各製品の ROM 領域については表 2.3 を参照してください。

表 2.3 ROM 領域の境界

ROM 容量	アドレス	RX610	RX62N RX621	RX62T RX62G	RX630 RX63N RX631	RX63T	RX210	RX220	RX21A
2MB	FFE0 0000h	領域 1			領域 3				
1.5MB	FFE8 0000h				領域 2				
1MB	FFF0 0000h	領域 0			領域 1		領域 1		
512KB	FFF8 0000h		領域 0		領域 0	領域 0	領域 0		領域 0
256KB	FFFC 0000h			領域 0				領域 0	

2.10 メモリ使用量

ROM および RAM の使用量は、有効にしたコンフィギュレーションオプションによって異なります。たとえば、ROM の書き換え（FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING）を有効にした場合、RAM 上から API を実行して ROM を書き換えるため、RAM と ROM の両方にプログラムを格納する領域が必要になります。

表 2.4 にいくつかのコンフィギュレーションパターン、表 2.5 にコンフィグレーションパターンにおけるメモリ使用量を示します。

表 2.4　コンフィグレーションパターン

	コンフィグレーションオプション	オプション 有効/無効
1	データフラッシュのみの書き換え	
	FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING	無効
	FLASH_API_RX_CFG_FLASH_TO_FLASH	無効
	FLASH_API_RX_CFG_DATA_FLASH_BGO	有効
	FLASH_API_RX_CFG_ROM_BGO	無効
	FLASH_API_RX_CFG_IGNORE_LOCK_BITS	有効
	FLASH_API_RX_CFG_COPY_CODE_BY_API	無効
2	ROM の書き換え	
	FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING	有効
	FLASH_API_RX_CFG_FLASH_TO_FLASH	無効
	FLASH_API_RX_CFG_DATA_FLASH_BGO	無効
	FLASH_API_RX_CFG_ROM_BGO	無効
	FLASH_API_RX_CFG_IGNORE_LOCK_BITS	無効
	FLASH_API_RX_CFG_COPY_CODE_BY_API	有効
3	ROM/RAM 使用量最大	
	FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING	有効
	FLASH_API_RX_CFG_FLASH_TO_FLASH	有効
	FLASH_API_RX_CFG_DATA_FLASH_BGO	有効
	FLASH_API_RX_CFG_ROM_BGO	有効
	FLASH_API_RX_CFG_IGNORE_LOCK_BITS	無効
	FLASH_API_RX_CFG_COPY_CODE_BY_API	有効

コンパイラオプションの設定（共通）

- 言語指定：C89
- 最適化レベル：レベル 2

表 2.5　メモリ使用量

デバイス	コンフィギュレーションパターン	ROM（バイト）	RAM（バイト）
RX610	1	1850	19
	2	2039	1300
	3	2554	2409
RX62N	1	1926	19
	2	2094	1247
	3	2609	2376
RX62G, RX62T	1	1862	19
	2	2030	1247
	3	2545	2376
RX630, RX63N	1	2302	19
	2	2496	1526
	3	3051	2566
RX63T	1	2054	19
	2	2189	1387
	3	2744	2427
RX210	1	3993	19
	2	4146	1469
	3	4702	2494
RX220	1	2378	19
	2	2508	1411
	3	3064	2432
RX21A	1	2896	19
	2	3026	1413
	3	3582	2438

2.11 ユーザプロジェクトへの組み込み方法

本 API をユーザプロジェクトに組み込むには以下の手順を実施してください。

1. Source ディレクトリ内の r_flash_api_rx ディレクトリをすべてユーザプロジェクトにコピーします。
2. r_flash_api_rx¥src¥r_flash_api_rx.c ファイルをビルド対象に追加します。
3. r_flash_api_rx ディレクトリをインクルードパスに追加します。
4. r_flash_api_rx¥src ディレクトリをインクルードパスに追加します。
5. r_flash_api_rx¥src¥targets ディレクトリからユーザシステムで使用しない製品名のディレクトリを削除します。
6. ref ディレクトリから r_flash_api_rx_config_reference.h をユーザプロジェクトのディレクトリにコピーし、ファイル名を r_flash_api_rx_config.h にリネームします。
7. r_flash_api_rx_config.h ファイルが格納されているディレクトリをインクルードパスに追加します。
8. r_flash_api_rx_config.h でユーザシステムに応じたコンフィグレーションを行います。
9. API を使用するソースファイルで r_flash_api_rx_if.h ファイルのインクルードを追加します。

※r_flash_api_rx¥ref ディレクトリはユーザプロジェクトから削除してもビルドに影響はありません。

ROM の書き換えを行う場合は、続けて 2.8 API のコードを RAM から実行する方法 の手順に従って設定してください。

3. API 関数

3.1 概要

API 関数の概要を表 3.1 に示します。

表 3.1 API 関数一覧

関数	内容
R_FlashErase	フラッシュメモリの対象ブロックを消去します。
R_FlashEraseRange	データフラッシュの対象範囲のブロックを消去します。 （RX610、RX62x グループは使用不可）
R_FlashWrite	フラッシュメモリにデータを書き込みます。
R_FlashDataAreaAccess	データフラッシュへのアクセス（読み出し、書き込み、消去）を許可/禁止します。
R_FlashDataAreaBlankCheck	データフラッシュをブランクチェックします。
R_FlashProgramLockBit	ROM の対象ブロックにロックビットを設定し、消去や書き込みを禁止します。
R_FlashReadLockBit	ROM の対象ブロックのロックビットの状態を読み出します。
R_FLashSetLockBitProtection	ロックビットプロテクト機能の有効/無効を切り替えます。
R_FlashGetStatus	API の処理状態を返します。
R_FlashCodeCopy	API のコードを ROM セクション（PFRAM）から RAM セクション（RPFRAM）にコピーします。

3.2 R_FlashErase

フラッシュメモリの対象ブロックを消去します。

フォーマット

```
uint8_t R_FlashErase(uint32_t block);
```

パラメータ

block

消去するブロック番号を指定します。ブロック番号は r_flash_api_rx/src/targets 配下のデバイスごとに用意されている r_flash_api_rxXXX.h で定義されています。たとえば RX610 のアドレス 0xFFFFE000 にあるブロックは、ユーザーズマニュアルではブロック 0 となっており、このパラメータには“BLOCK_0”と指定します。

戻り値

FLASH_SUCCESS:	正常終了（ノンブロッキングモード有効時は、フラッシュメモリへの処理が正常に開始されたことを示す）
FLASH_FAILURE:	異常終了（ロックビットが設定されている ROM 領域またはアクセス禁止に設定されているデータフラッシュ領域への処理、または処理のタイムアウト）
FLASH_BUSY:	フラッシュメモリに対して別の処理を実行中
FLASH_ERROR_ADDRESS:	無効なブロック番号

説明

引数で指定したブロックを消去します。ブロックサイズは、デバイスグループやデバイスのフラッシュメモリ領域によって異なります。また、一部のデバイスのデータフラッシュでは、ブロックサイズが細かく定義されているため、複数ブロックをまとめて消去します。ブロック構成の詳細は、各デバイスのユーザーズマニュアルを参照してください。

各デバイスの消去サイズを表 3.2 に示します。

表 3.2 消去サイズ

デバイス	ROM	データフラッシュ
RX610 グループ	128KB、64KB、8KB	8KB
RX62x グループ	16KB、4KB	2KB
RX630、RX63N、RX631 グループ	64KB、32KB、16KB、4KB	2KB（32B×64 ブロック）
RX63T グループ	16KB、4KB	2KB（32B×64 ブロック）
RX2x グループ	2KB	2KB（128B×16 ブロック）

リエントラント

リエントラントではありません。

使用例

```
uint32_t loop;
uint8_t ret;

/* Specify the erase block */
ret = R_FlashErase(BLOCK_0);

/* Check for errors. */
if (FLASH_SUCCESS != ret)
{
    . . .
}

/* Erase multiple blocks (erase block 0 to block "NUM_BLOCKS_TO_ERASE")*/
for (loop = 0; loop < NUM_BLOCKS_TO_ERASE; loop++)
{
    /* Erase block */
    ret = R_FlashErase(loop);

    /* Check for errors. */
    if (FLASH_SUCCESS != ret)
    {
        . . .
    }
}
```

注意事項

データフラッシュのブロックを消去するときには、R_FlashDataAreaAccess 関数を使って事前にデータフラッシュへのアクセスを許可してください。

3.3　R_FlashEraseRange（RX610、RX62x グループは使用不可）

データフラッシュの対象範囲のブロックを消去します。

フォーマット

```
uint8_t R_FlashEraseRange(uint32_t start_addr, uint32_t bytes);
```

パラメータ

start_addr

消去する対象範囲のブロックの先頭アドレスを指定します。アドレスはブロックサイズのアライメントである必要があります。各デバイスのブロックサイズとアドレスの計算方法は表 3.3 を参照してください。

bytes

消去するバイト数を指定します。この値はデータフラッシュのブロックサイズの倍数である必要があります。たとえば RX630 の場合、データフラッシュのブロックサイズは 32 バイトのため、設定する値は 32、64、96... などになります。

戻り値

FLASH_SUCCESS:	正常終了（ノンブロッキングモード有効時は、フラッシュメモリへの処理が正常に開始されたことを示す）
FLASH_FAILURE:	異常終了（アクセス禁止に設定されているデータフラッシュ領域への処理、または処理のタイムアウト）
FLASH_BUSY:	フラッシュメモリに対して別の処理を実行中
FLASH_ERROR_BYTES:	バイト数がブロックサイズの倍数ではない
FLASH_ERROR_ADDRESS:	無効なアドレス
FLASH_ERROR_ALIGNED:	ブロックの先頭アドレスが指定されていない
FLASH_ERROR_OVERFLOW:	消去の範囲がデータフラッシュ領域を超えている

説明

指定した範囲（start_addr ～ （start_addr + bytes））のブロックを消去します。

表 3.3 データフラッシュのブロックサイズ

デバイス	ブロックサイズ	ブロック N のアドレス計算方法
RX63x グループ	32B （32B × 1024 ブロック= 32KB）	N × 32 + データフラッシュ領域の先頭アドレス（0010 0000h）
RX2x グループ	128B （128B × 64 ブロック= 8KB）	N × 128 + データフラッシュ領域の先頭アドレス（0010 0000h）

リエントラント

リエントラントではありません。

使用例

```
uint8_t ret;

/* Erase 64 bytes. */
ret = R_FlashEraseRange(address, 64);

/* Check for errors. */
if (FLASH_SUCCESS != ret)
{
    . . .
}
```

注意事項

- この関数は RX610 および RX62x グループでは使用できません。R_FlashErase 関数を使って消去してください。
- この関数はデータフラッシュに対してのみ有効です。
- この関数を使用するときには R_FlashDataAreaAccess 関数を使って事前にデータフラッシュへのアクセスを許可してください。

3.4 R_FlashWrite

フラッシュメモリにデータを書き込みます。

フォーマット

```
uint8_t R_FlashWrite(uint32_t   flash_addr,
                     uint32_t   buffer_addr,
                     uint16_t   bytes);
```

パラメータ

flash_addr

書き込み先のアドレスを指定します。アドレスは最小書き込みサイズのアライメントである必要があります。

buffer_addr

書き込み元のアドレスを指定します。

bytes

書き込むデータのバイト数を指定します。この値は最小書き込みサイズの倍数である必要があります。各デバイスの最小書き込みサイズは表 3.4 を参照してください。

戻り値

FLASH_SUCCESS:	正常終了（ノンブロッキングモード有効時は、フラッシュメモリへの処理が正常に開始されたことを示す）
FLASH_FAILURE:	異常終了（ブランクではない領域、またはロックビットが設定されている ROM 領域やアクセス禁止に設定されているデータフラッシュ領域への処理、または処理のタイムアウト）
FLASH_BUSY:	フラッシュメモリに対して別の処理を実行中
FLASH_ERROR_ALIGNED:	アドレスが最小書き込みサイズのアライメントではない
FLASH_ERROR_BYTES:	バイト数が最小書き込みサイズの倍数ではない
FLASH_ERROR_ADDRESS:	無効なアドレス
FLASH_ERROR_BOUNDARY:	ROM 領域の境界を跨ぐ書き込みは不可
FLASH_ERROR_OVERFLOW:	書き込みの範囲が ROM またはデータフラッシュの領域を超えている

説明

フラッシュメモリにデータを書き込みます。書き込みアドレスは、最小書き込みサイズのアライメントである必要があります。また、書き込みバイト数は最小書き込みサイズの倍数である必要があります。

最小書き込みサイズは下記表のとおりデバイスによって異なります。

表 3.4 最小書き込みサイズ

デバイス	ROM	データフラッシュ
RX610 および RX62x グループ	256B	8B、128B
RX63x グループ	128B	2B
RX210 グループ	2B、8B、128B	2B、8B

また、RX ファミリには複数の ROM 領域を持つ製品があります。一度の R_FlashWrite 関数の実行で、ROM 境界を跨いだ書き込みはできません。ROM 領域の境界を跨ぐ書き込みを行うためには書き込みデータを分割して R_FlashWrite 関数を実行してください。ROM 領域の詳細は 2.10.5 ROM 領域の境界を跨ぐ書き込みを参照してください。

さらに、r_flash_api_rx_config.h で FLASH_API_RX_CFG_FLASH_TO_FLASH の定義が有効の場合、書き込み先と同じ領域のアドレスを指定することができます。たとえば、書き込み先の領域が ROM の場合は ROM のアドレスを、データフラッシュの場合はデータフラッシュのアドレスを指定できます。この機能の詳細は 2.9 ROM から ROM、データフラッシュからデータフラッシュへの書き換えを参照してください。

リエントラント
リエントラントではありません。

使用例
```
uint8_t ret;
uint8_t write_buffer[PROGRAM_SIZE] = "Hello World...";

/* Write data to internal memory. */
ret = R_FlashWrite(address, (uint32_t)write_buffer, PROGRAM_SIZE);

/* Check for errors. */
if (FLASH_SUCCESS != ret)
{
    . . .
}
```

注意事項
データフラッシュに書き込みするときには、R_FlashDataAreaAccess 関数を使って事前にデータフラッシュへのアクセスを許可してください。

3.5　R_FlashDataAreaAccess

データフラッシュへのアクセス（読み出し、書き込み、消去）を許可/禁止します。リセット解除後は R_FlashDataAreaAccess 関数を実行してデータフラッシュへのアクセス（読み出し、書き込み、消去）を許可してください。

フォーマット

```
void R_FlashDataAreaAccess(uint16_t read_en_mask,
                           uint16_t write_en_mask);
```

パラメータ

read_en_mask

各ビットに対応するブロックの読み出しを許可/禁止します。'1'に設定すると許可され、'0'に設定すると禁止されます。各ビットに対応するブロックは表 3.5 を参照してください。

write_en_mask

各ビットに対応するブロックの書き込みまたは消去を許可/禁止します。'1'に設定すると許可され、'0'に設定すると禁止されます。各ビットに対応するブロックは表 3.5 を参照してください。

戻り値

なし

説明

データフラッシュへのアクセスを許可/禁止します。データフラッシュにアクセスする前に R_FlashDataAreaAccess 関数を実行してアクセスを許可してください。

表 3.5 に各ビットに対応するブロックを示します。”—”のビットへの設定は無効です。

表 3.5 ビットごとの対応ブロック

デバイス	サイズ	ブロック構成	read_en_mask、write_en_mask							
			b15	b14	b13	b12	b11	b10	b9	b8
			b7	b6	b5	b4	b3	b2	b1	b0
RX610	32KB	8KB×4 ブロック	—	—	—	—	—	—	—	—
			—	—	—	—	DB3	DB2	DB1	DB0
RX621 RX62N	32KB	2KB×16 ブロック	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
RX62T RX62G	8KB	2KB×4 ブロック	—	—	—	—	—	—	—	—
			—	—	—	—	DB3	DB2	DB1	DB0
	32KB	2KB×16 ブロック	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
RX630 RX631 RX63N	32KB	32B×1024 ブロック	960～1023	896～959	832～895	768～831	704～767	640～703	576～639	512～575
			448～511	384～447	320～383	256～319	192～255	128～191	64～127	0～63
RX63T	8KB	32B×256 ブロック	—	—	—	—	—	—	—	—
			—	—	—	—	192～255	128～191	64～127	0～63
	32KB	32B×1024 ブロック	960～1023	896～959	832～895	768～831	704～767	640～703	576～639	512～575
			448～511	384～447	320～383	256～319	192～255	128～191	64～127	0～63
RX210 RX220 RX21A	8KB	128B×64 ブロック	—	—	—	—	—	—	—	—
			—	—	—	—	DB48～DB63	DB32～DB47	DB16～DB31	DB00～DB15

リエントラント
リエントラントではありません。

使用例
```
/* Enable reading, writing, and erasing of all data flash blocks. */
R_FlashDataAreaAccess(0xFFFF, 0xFFFF);
```

注意事項
なし

3.6　R_FlashDataAreaBlankCheck

データフラッシュをブランクチェックします。

フォーマット

```
uint8_t R_FlashDataAreaBlankCheck(uint32_t address,
                                  uint8_t size);
```

パラメータ

address

ブランクチェックする領域のアドレスまたはブロック番号を指定します。アドレスの場合は size で指定したブランクチェックサイズのアライメントで指定します。ブロック番号の場合は r_flash_api_rx/src/targets 配下のデバイスごとに用意されている r_flash_api_rxXXX.h の定義を使用します。

size

ブランクチェックのサイズを指定します。指定できる値は以下の 2 通りです。

BLANK_CHECK_SMALLEST　：最小ブランクチェックサイズ
BLANK_CHECK_ENTIRE_BLOCK　：複数ブロックサイズ

戻り値

FLASH_BLANK:　（ブロッキングモードの場合）
　結果はブランク
（ノンブロッキングモードで size が BLANK_CHECK_SMALLEST の場合）
　結果はブランク
（ノンブロッキングモードで size が BLANK_CHECK_ENTIRE_BLOCK の場合）
　ブランクチェックの処理が正常に開始
FLASH_NOT_BLANK:　結果はブランクではない
FLASH_FAILURE:　異常終了（通常では戻りません）
FLASH_BUSY:　フラッシュメモリに対して別の処理を実行中
FLASH_ERROR_ADDRESS:　無効なアドレス
FLASH_ERROR_BYTES:　無効なサイズ
FLASH_ERROR_ALIGNED:　アドレスがブランクチェックサイズのアライメントではない
（size が BLANK_CHECK_SMALLEST のときのみ）

説明

データフラッシュをブランクチェックします。R_FlashDataAreaBlankCheck 関数はデータフラッシュ専用です。ブランクチェックのサイズは BLANK_CHECK_SMALLEST（最小ブランクチェックサイズ）と BLANK_CHECK_ENTIRE_BLOCK（複数ブロックサイズ）が指定できます。ブランクチェックサイズは表 3.6 のとおりデバイスによって異なります。

表 3.6 ブランクチェックサイズ

デバイス	ブランクチェックサイズ
RX610	BLANK_CHECK_SMALLEST：8 バイト
	BLANK_CHECK_ENTIRE_BLOCK：1 ブロック（8KB）
RX62x	BLANK_CHECK_SMALLEST：8 バイト
	BLANK_CHECK_ENTIRE_BLOCK：1 ブロック（2KB）
RX63x	BLANK_CHECK_SMALLEST：2 バイト
	BLANK_CHECK_ENTIRE_BLOCK：64 ブロック（2KB）
RX210	BLANK_CHECK_SMALLEST：2 バイト
	BLANK_CHECK_ENTIRE_BLOCK：16 ブロック（2KB）

リエントラント

リエントラントではありません。

使用例

```
uint8_t ret;

/* Blank check a small data flash address. Blocking mode operation Only */
ret = R_FlashDataAreaBlankCheck(address, BLANK_CHECK_SMALLEST);

/* Check result. */
if (FLASH_NOT_BLANK == ret)
{
    /* Block is not blank. */
    . . .
}
else if (FLASH_BLANK == ret)
{
    /* Block is blank. */
    . . .
}

/* Blank check an entire data flash block. */
ret = R_FlashDataAreaBlankCheck(BLOCK_DB0, BLANK_CHECK_ENTIRE_BLOCK);

/* Check result. */
if (FLASH_NOT_BLANK == ret)
{
    /* Block is not blank. */
    . . .
}
else if (FLASH_BLANK == ret)
{
    /* Block is blank. */
    . . .
}
```

注意事項

ノンブロッキングモードに設定している場合でも、第二引数（size）に BLANK_CHECK_SMALLEST（最小ブランクチェックサイズ）を指定すると、書き込みや消去に比べて処理が早く完了するためブロッキングモードとして動作します。BLANK_CHECK_ENTIRE_BLOCK（複数ブロックサイズ）は、設定した API の動作モードで動作します。

3.7 R_FlashProgramLockBit

ROM の対象ブロックにロックビットを設定し、消去や書き込みを禁止します。

フォーマット

```
uint8_t R_FlashProgramLockBit(uint32_t block);
```

パラメータ

block

ロックビットを設定するブロック番号を指定します。ブロック番号は r_flash_api_rx/src/targets 配下のデバイスごとに用意されている r_flash_api_rxXXX.h で定義されています。たとえば RX610 のアドレス 0xFFFFE000 にあるブロックは、ユーザーズマニュアルではブロック 0 となっており、このパラメータには“BLOCK_0”と指定します。

戻り値

FLASH_SUCCESS:	正常終了
FLASH_FAILURE:	異常終了（既にロックビットが設定されている領域への処理）
FLASH_BUSY:	フラッシュメモリに対して別の処理を実行中
FLASH_ERROR_ADDRESS:	無効なブロック番号

説明

指定したブロックにロックビットを設定します。

リエントラント

リエントラントではありません。

使用例

```
uint8_t ret;

/* Enable lock bit protection (this is default out of reset) */
ret = R_FlashSetLockBitProtection(true);

/* Check for errors. */
if (FLASH_SUCCESS != ret)
{
    . . .
}

/* Program lock bits */
ret = R_FlashProgramLockBit(flash_block);

/* Check for errors. */
if (FLASH_SUCCESS != ret)
{
    . . .
}
```

注意事項

- ロックビットプロテクト機能が無効の状態でブロックが消去されると、そのブロックのロックビットはクリアされます。
- R_FlashProgramLockBit 関数を使用するには r_flash_api_rx_config.h で FLASH_API_RX_CFG_IGNORE_LOCK_BITS の定義を無効にしてください。

3.8 R_FlashReadLockBit

ROM の対象ブロックのロックビットの状態を読み出します。

フォーマット

```
uint8_t R_FlashReadLockBit(uint32_t block);
```

パラメータ

block

ロックビットの状態を読み出すブロック番号を指定します。

戻り値

FLASH_LOCK_BIT_SET:	ロックビットが設定されている
FLASH_LOCK_BIT_NOT_SET:	ロックビットはクリアされている
FLASH_FAILURE:	異常終了（通常では戻りません）
FLASH_BUSY:	フラッシュメモリに対して別の処理を実行中
FLASH_ERROR_ADDRESS:	無効なブロック番号

説明

指定したブロックのロックビットの状態を読み出します。読み出した結果は戻り値で判断します。戻り値が FLASH_LOCK_BIT_SET（0x07）の場合、対象ブロックにロックビットが設定されています。戻り値が FLASH_LOCK_BIT_NOT_SET（0x08）の場合、対象ブロックのロックビットはクリアされています。

リエントラント

リエントラントではありません。

使用例

```
uint8_t ret;

/* Program lock bits */
ret = R_FlashReadLockBit(flash_block);

/* Check result. */
if (FLASH_LOCK_BIT_SET == ret)
{
    /* Lock bit is set for this block. */
    . . .
}
else if (FLASH_LOCK_BIT_NOT_SET == ret)
{
    /* Lock bit was not set for this block. */
    . . .
}
```

注意事項

- ロックビットプロテクト機能が無効の状態でブロックが消去されると、そのブロックのロックビットはクリアされます。
- R_FlashReadLockBit 関数を使用するには r_flash_api_rx_config.h で FLASH_API_RX_CFG_IGNORE_LOCK_BITS の定義を無効にしてください。

3.9 R_FlashSetLockBitProtection

ロックビットプロテクト機能の有効/無効を切り替えます。

フォーマット

```
uint8_t R_FlashSetLockBitProtection(uint32_t lock_bit);
```

パラメータ

lock_bit

ロックビットプロテクト機能を有効/無効を指定します。true または 1 以上を指定すると有効に、false または 0 を指定すると無効になります。

戻り値

FLASH_SUCCESS: 正常終了
FLASH_BUSY: フラッシュメモリに対して別の処理を実行中

説明

ロックビットプロテクト機能の有効/無効を切り替えます。有効の場合、ロックビットが設定されたブロックに対して書き込みまたは消去することができません。無効にした場合、ロックビットが設定されているかどうかにかかわらず、すべてのブロックに対して書き込みおよび消去することができます。

リエントラント

リエントラントではありません。

使用例

```
uint8_t ret;

/* Enable lock bit protection (this is default out of reset) */
ret = R_FlashSetLockBitProtection(true);

/* Check for errors. */
if (FLASH_SUCCESS != ret)
{
    . . .
}
```

注意事項

- ロックビットプロテクト機能が無効の状態でブロックが消去されると、そのブロックのロックビットはクリアされます。
- R_FlashSetLockBitProtection 関数を使用するには r_flash_api_rx_config.h で FLASH_API_RX_CFG_IGNORE_LOCK_BITS の定義を無効にしてください。

3.10 R_FlashGetStatus

API の処理状態を返します。

フォーマット

```
uint8_t R_FlashGetStatus(void);
```

パラメータ

なし

戻り値

FLASH_SUCCESS: API 関数を実行可能
FLASH_BUSY: フラッシュメモリに対して別の処理を実行中

説明

ノンブロッキングモードのときに API の処理状態を確認することができます。

リエントラント

リエントラントです。

使用例

```
uint8_t ret;

/* Blank check an entire data flash block. */
ret = R_FlashDataAreaBlankCheck(address, BLANK_CHECK_ENTIRE_BLOCK);

while( R_FlashGetStatus() == FLASH_BUSY )
{
    /* Wait for previous operation to finish. You could also stall this task
       and do some real work. */
}
```

注意事項

なし

3.11 R_FlashCodeCopy

API のコードを ROM セクション（PFRAM）から RAM セクション（RPFRAM）にコピーします。

フォーマット

```
void R_FlashCodeCopy(void);
```

パラメータ

なし

戻り値

なし

説明

R_FlashCodeCopy 関数を実行することで ROM セクション（PFRAM）から RAM セクション（RPFRAM）に API のコードをコピーします。

リエントラント

リエントラントです。

使用例

```
/* Transfer Flash API code to RAM so that we can program/erase ROM. */
R_FlashCodeCopy();

/* Flash API can now program/erase ROM. */
```

注意事項

- dbsct.c を編集する方法を使用している場合には、R_FlashCodeCopy 関数を実行する必要はありません。
- R_FlashCodeCopy 関数を使用するには r_flash_api_rx_config.h で FLASH_API_RX_CFG_COPY_CODE_BY_API の定義を有効にしてください。

3.12 R_FlashGetVersion

本 API のバージョンを返します。

フォーマット

```
uint32_t R_FlashGetVersion(void);
```

パラメータ

なし

戻り値

API のバージョン。

説明

API のバージョンを戻り値として返します。戻り値の上位 2 バイトがメジャーバージョン番号を、下位 2 バイトがマイナーバージョンを示します。たとえばバージョン 4.25 では 0x00040019 です。

リエントラント

リエントラントです。

使用例

```
uint32_t cur_version;

/* Get version of installed Flash API. */
cur_version = R_FlashGetVersion();

/* Check to make sure version is new enough for this application's use. */
if (MIN_VERSION > cur_version)
{
    /* This Flash API version is not new enough and does not have XXX feature
       that is needed by this application. Alert user. */
    ....
}
```

注意事項

R_FlashGetVersion 関数は r_flash_api_rx.c ファイル内でインライン関数として定義されています。

4. 参考情報

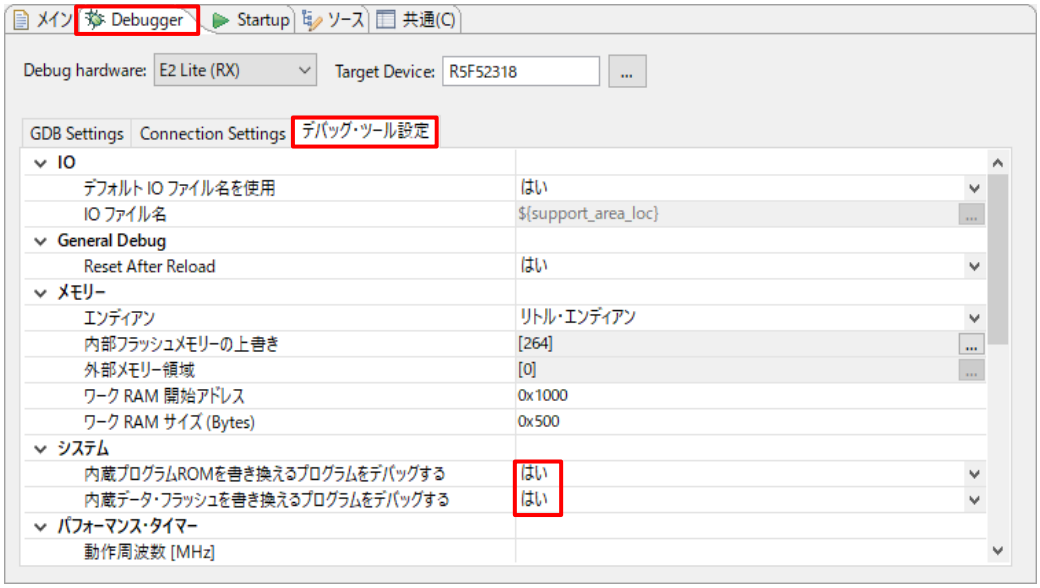
4.1 エミュレータのデバッグ設定

統合開発環境（e² studio や CS+）のデフォルトのデバッグ設定では、本 API で書き換えたフラッシュメモリの内容をメモリモニタ機能で確認することはできません。書き換えはできていますが、ダウンロードした値からモニタ値が更新されません。書き換え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。

e² studio の場合

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。

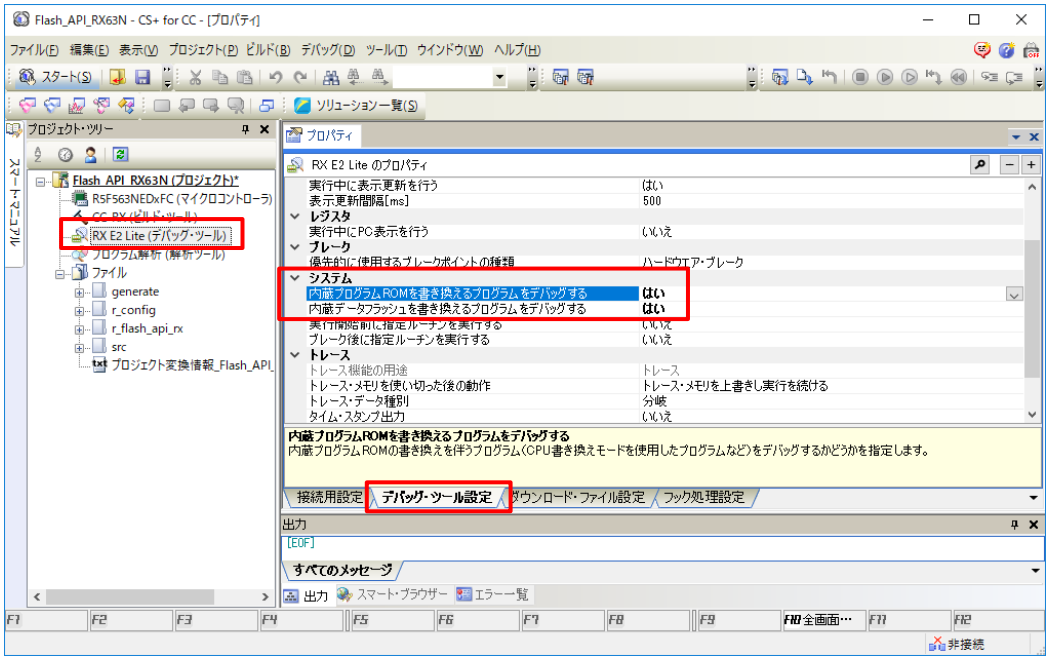
- システム
  - 内蔵プログラム ROM を書き換えるプログラムをデバッグする = “はい”
  - 内蔵データ・フラッシュを書き換えるプログラムをデバッグする = “はい”



CS+の場合

RX E2 Lite（デバッグ・ツール）>> デバッグ・ツール設定 >> システム から以下のように設定します。

- 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”



4.2 フラッシュプログラマによる書き換えデータの読み出し

ユーザプログラムで書き換えたフラッシュメモリの内容を Renesas Flash Programmer（RFP）で読み出したい場合、ユーザプログラムをダウンロードするときに ID コードを設定しておく必要があります。ID コードは RX ファミリのデバイスが持つセキュリティ機能です。ID コードが設定されていない状態で接続すると、RFP によりフラッシュメモリの領域がすべて削除されますのでご注意ください。ID コードの詳細については各デバイスのユーザーズマニュアルを参照してください。

5. サンプルプロジェクト

5.1 概要

本アプリケーションノートでは、対象デバイスのサンプルプロジェクトを用意しています。サンプルプロジェクトは e² studio プロジェクトの形式で提供しており、e² studio または CS+でインポートして動作確認できます。

5.2 動作確認環境

サンプルプロジェクトは以下の環境で動作を確認しています。

表 5.1 動作確認環境

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 2022-01
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 V3.04.00 コンパイルオプションは統合開発環境のデフォルト設定を使用しています。
エンディアン	ビッグエンディアン/リトルエンディアン
動作モード	シングルチップモード
プロセッサモード	スーパバイザモード
使用ボード	Renesas Starter Kit for RX610（型名：R0K55610xxxxxxx） Renesas Starter Kit for RX62G（型名：R0K50562Gxxxxxx） Renesas Starter Kit+ for RX62N（型名：R0K5562Nxxxxxxx） Renesas Starter Kit for RX62T（型名：R0K5562Txxxxxxx） Renesas Starter Kit for RX630（型名：R0K505630xxxxxx） Renesas Starter Kit+ for RX63N（型名：R0K50563Nxxxxxx） Renesas Starter Kit for RX63T (144-pin)（型名：R0K5563THxxxxxx） Renesas Starter Kit for RX210(B 版)（型名：R0K505210xxxxxx） Renesas Starter Kit for RX220（型名：R0K505220xxxxxx） 株式会社北斗電子製 HSB シリーズ マイコンボード (製品型名: HSBRX21AP-B)

5.3　サンプルプログラムの基本動作

サンプルプロジェクトで動作するプログラムは、コンフィグレーションオプションの設定に応じて、オプションの内容に関連する処理を実行します。表 5.2 にサンプルプログラムの動作について示します。詳細はサンプルプログラム（flash_api_rx_demo_main.c）を参照してください。

表 5.2　サンプルプログラムの動作

関数および処理概要	関連コンフィグレーションオプション
flash_api_demo_df_tests 関数 データフラッシュのすべてのブロックに対して、API 関数を使用したイレーズ、ブランクチェック、書き込み等の処理を実行します。	FLASH_API_RX_CFG_DATA_FLASH_BGO FLASH_API_RX_CFG_FLASH_TO_FLASH
flash_api_demo_rom_tests 関数 プログラムが格納されているブロック(16KB 分)以外のすべてのブロックに対して、API 関数を使用したイレーズ、書き込み、ロックビットの設定等の処理を実行します。	FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING FLASH_API_RX_CFG_ROM_BGO FLASH_API_RX_CFG_FLASH_TO_FLASH FLASH_API_RX_CFG_IGNORE_LOCK_BITS
flash_api_demo_rom_bgo_init 関数 ノンブロッキングモードの場合に、割り込みベクタテーブルを RAM に再配置する処理を実行します。	FLASH_API_RX_CFG_ROM_BGO
flash_api_demo_lock_bit_tests 関数 ロックビットの許可/禁止や設定、読み出し等の処理を実行します。	FLASH_API_RX_CFG_IGNORE_LOCK_BITS

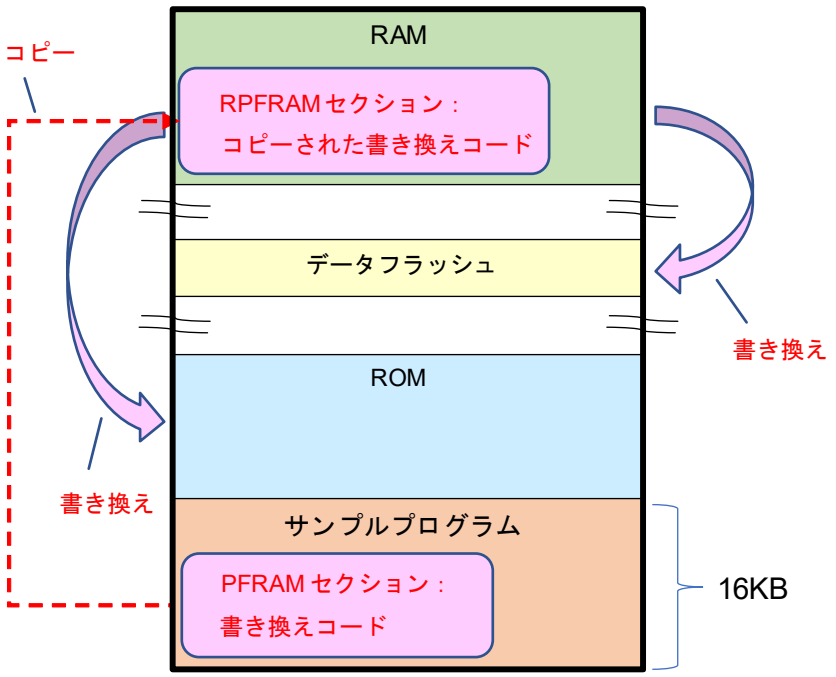


図 5.1 サンプルプログラムの処理概要（ROM の書き換えが有効の場合）

5.4 サンプルプロジェクトの動作クロックの設定

いくつかのデバイスのサンプルプロジェクトにおいて、動作クロックの設定に初期設定例のアプリケーションノートを使用しています。動作クロックは初期設定例のアプリケーションノートのデフォルト設定です。初期設定例のアプリケーションノートについては 6 参考ドキュメントを参照してください。

各サンプルプロジェクトにおける動作クロックの設定は、generate/hwsetup.c の HardwareSetup 関数で実施しています。

5.5　プロジェクトをインポートする方法

サンプルコードは e² studio のプロジェクト形式で提供しています。本章では、 e² studio および CS+へプロジェクトをインポートする方法を示します。インポート完了後、ビルドおよびデバッグの設定を確認してください。

5.5.1　e² studio での手順

e² studio でご使用になる際は、下記の手順で e² studio にインポートしてください。

なお、e² studio で管理するプロジェクトのフォルダ名、およびそのフォルダに至るファイルパスには、空白文字の他、半角カナ文字、全角文字、半角記号(特に'$','#','%') が混じらないようにしてください。

(使用する e² studio のバージョンによっては画面が異なる場合があります。)

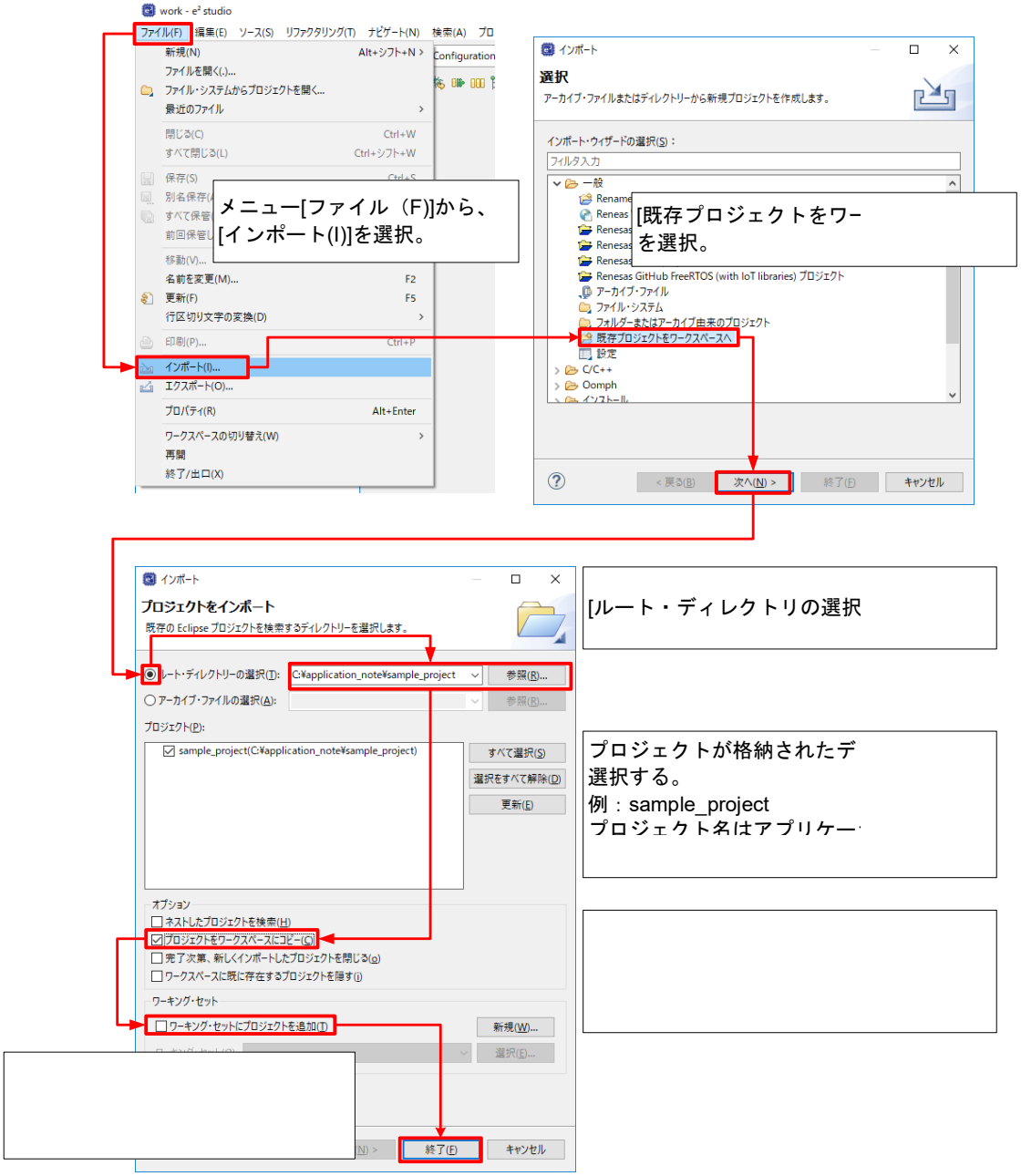


図 5.2　プロジェクトを e² studio にインポートする方法

5.5.2　CS+での手順

CS+でご使用になる際は、下記の手順で CS+にインポートしてください。

なお、CS+で管理するプロジェクトのフォルダ名、およびそのフォルダに至るファイルパスには、 空白文字の他、半角カナ文字、全角文字、半角記号(特に'$','#','%') が混じらないようにしてください。

(使用する CS+のバージョンによっては画面が異なる場合があります。)

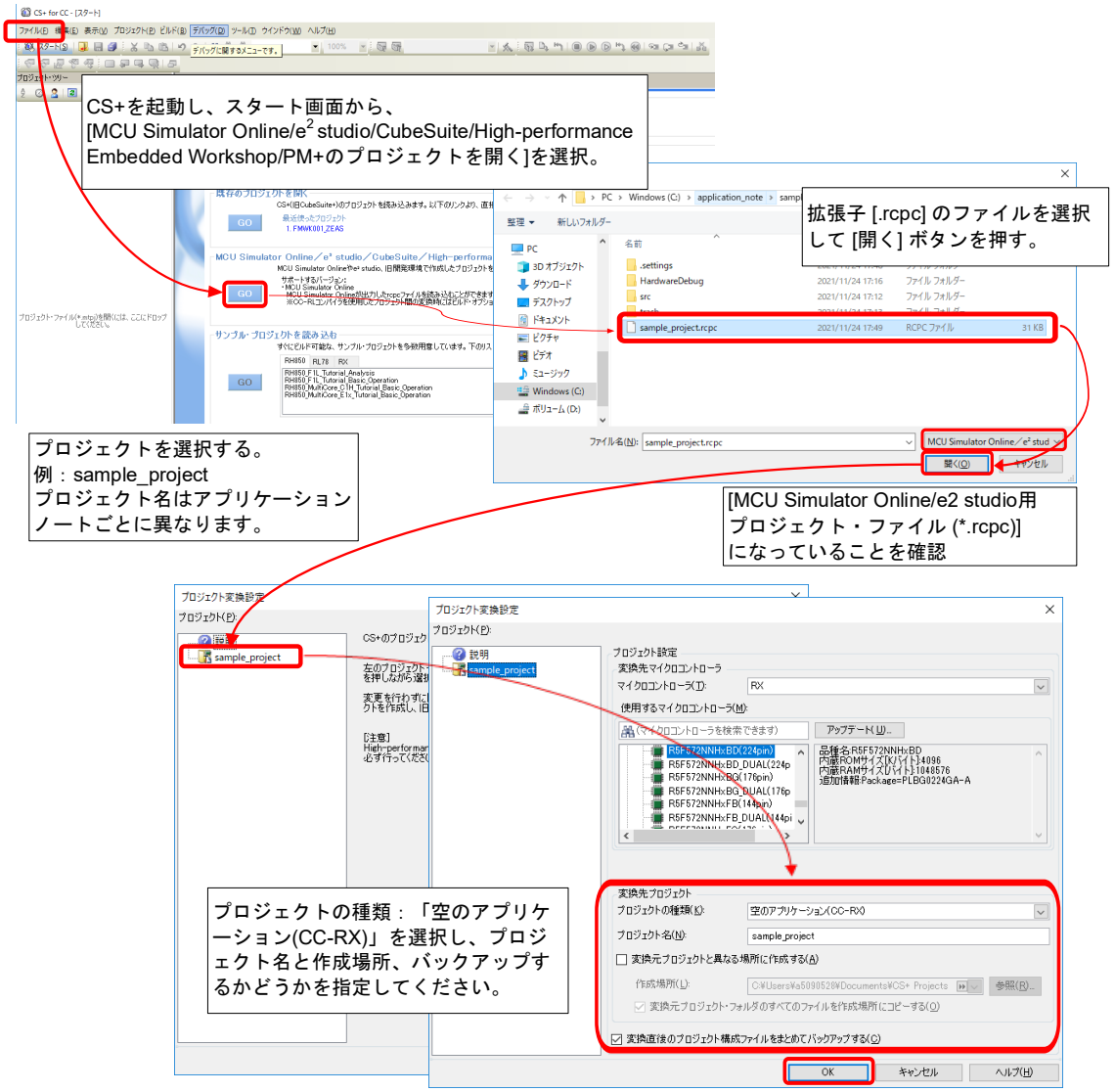


図 5.3　プロジェクトを CS+にインポートする方法

6. 参考ドキュメント

ユーザーズマニュアル：

タイトル	Rev.	ドキュメント番号
RX610 グループ ユーザーズマニュアル ハードウェア編	1.20	R01UH0032JJ0120
RX62N グループ。RX621 グループ ユーザーズマニュアル ハードウェア編	1.40	R01UH0033JJ0140
RX62T グループ、RX62G グループ ユーザーズマニュアル ハードウェア編	2.00	R01UH0034JJ0200
RX630 グループ ユーザーズマニュアル ハードウェア編	1.60	R01UH0040JJ0160
RX63N グループ、RX631 グループ ユーザーズマニュアル ハードウェア編	1.80	R01UH0041JJ0180
RX63T グループ ユーザーズマニュアル ハードウェア編	2.20	R01UH0238JJ0220
RX210 グループ ユーザーズマニュアル ハードウェア編	1.50	R01UH0037JJ0150
RX220 グループ ユーザーズマニュアル ハードウェア編	1.10	R01UH0292JJ0110
RX21A グループ ユーザーズマニュアル ハードウェア編	1.10	R01UH0251JJ0110

アプリケーションノート：

タイトル	Rev.	ドキュメント番号
RX630 グループ 初期設定例	1.00	R01AN1004JJ0100
RX63N グループ、RX631 グループ 初期設定例	1.10	R01AN1245JJ0110
RX63T グループ 初期設定例	1.01	R01AN1252JJ0101
RX210 グループ 初期設定例	2.21	R01AN1002JJ0221
RX220 グループ 初期設定例	1.10	R01AN1494JJ0110
RX21A グループ 初期設定例	1.10	R01AN1486JJ0110

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
1.00	2010.01.27	—	初版発行
1.20	2010.02.11	—	文章細部の変更、および割り込みの禁止に関する記述セクションの追加
1.30	2010.03.05	—	RTE からの指摘に基づく修正
1.40	2010.05.26	—	RX62x グループのサポートを追加するための変更
1.41	2010.06.11	—	誤植などの訂正
1.43	2011.02.18	—	消去確認（blank check）関数のパラメータ記述の更新
2.00	2011.04.27	—	バックグランド操作（BGO）、フラッシュからフラッシュへの転送、ロックビット保護の機能の追加
2.10	2011.07.11	—	RX630、RX631 および RX63N デバイスのサポートの追加。DATA_FLASH_OPERATION_PIPL および ROM_OPERATION_PIPL 定義の削除と、その理由の記述セクションを追加。API に R_FlashEraseRange() 関数を追加。RX610 と RS63x デバイスに対応するための ROM 領域境界に関する記述の（セクション 3.4 としての）書き換え。
2.20	2012.03.27	—	新しい文書形式への移行。r_bsp パッケージの利用に関する既存の情報の再構成と新たな情報の追加。API への R_FlashCodeCopy() 関数の追加。
2.30	2012.09.12	—	F_FlashGetVersion()関数を API に追加。 コード内で r_bsp を自動的に認識するよう変更されたため r_bsp を利用しない際のマクロの構成を削除。 「データフラッシュのみを操作対象とする構成」、「ユーザアプリケーション領域（ROM）全体の消去」、「リセット直後のデータフラッシュの読み込み」、「データフラッシュの特定個所がブランクか（消去されているか）の確認」、「フラッシュ API をユーザブート領域に配置する場合」セクションの追加。 ブランクチェックのサイズの表を R_FlashDataAreaBlankCheck セクションに追加。
2.40	2013.07.01	—	RX210、RX62G および RX63T デバイスのサポートの追加。サポート対象に RX200 シリーズのデバイスが加わったため、アプリケーションノートの表題を「RX600 用シンプルフラッシュ API」から「RX 用シンプルフラッシュ API」に変更。 「データフラッシュの特定個所がブランクか（消去されているか）の確認」セクションの追加と、データフラッシュの消去されたデータ読み出しが 0xFF とならない理由の記述個所に関するコメントを先頭ページに付記。 API 関数の一覧表を API 関数セクションの先頭に追加。プロジェクトの実例のセクションを追加。
2.50	2015.03.06	—	RX21A および RX220 のサポートの追加。HEW を参照したすべての説明について、e^2 studio 用の等価のステップを提供。 「データフラッシュから実行」サブセクションを追加。「アクセスルール」サブセクションを追加。関連ドキュメントを表紙ページに追加。「メモリ要件」サブセクションを追加。 「ブートローダの実装」セクションを削除。新しい BLANK_CHECK_SMALLEST マクロを使用して「R_FlashDataAreaBlankCheck」サブセクションを変更。 BLANK_CHECK_2_BYTE および BLANK_CHECK_8_BYTE は廃止予定。「デモ」セクションを削除。

3.00	2022.02.14	全体	BSP に依存する構成を依存しない構成に変更 RX631 グループの ROM サイズ 256KB 製品に対応
		ドキュメント	ドキュメント全体の構成および内容変更 ＜要点＞ ・2.4 コンフィグレーションオプションの設定 以下の新規オプションを追加 - FLASH_MCU_xxxx - FLASH_API_RX_CFG_ICLK_HZ - FLASH_API_RX_CFG_FCLK_HZ - FLASH_API_RX_CFG_ROM_SIZE_BYTES - FLASH_API_RX_CFG_DATA_FLASH_SIZE_BYTES ・5. サンプルプロジェクト Rev2.50 で削除した章を再追加
		プログラム	r_bsp に依存する構成を依存しない構成に変更 ・r_flash_api_rx_config.h に以下の新規オプションを追加 - FLASH_MCU_xxxx - FLASH_API_RX_CFG_ICLK_HZ - FLASH_API_RX_CFG_FCLK_HZ - FLASH_API_RX_CFG_ROM_SIZE_BYTES - FLASH_API_RX_CFG_DATA_FLASH_SIZE_BYTES ・すべてのソースファイル、ヘッダファイルにおいて r_bsp に関する定義（”BSP_XXX”）を新規オプション定義に変更 ・r_bsp のハードウェアロック機構を使用しない仕様に変更 - flash_grab_state 関数を変更 - flash_release_state 関数を変更 ・r_flash_api_rx_if.h ファイルから以下の定義を削除 - BLANK_CHECK_8_BYTE - BLANK_CHECK_2_BYTE RX631 グループの ROM サイズ 256KB に対応 ・r_flash_api_rx63n.h に 256KB 製品の情報を追加 ツールニュース（R20TS0805JJ0100）の対策を実施

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。