78K0R マイクロコントローラ

78K0R マイクロコントローラ
フラッシュ・セルフ・プログラミング・ライブラリ　Type03

R01AN0005JJ0100
Rev.1.00
2010.06.04

要旨

このユーザーズ・マニュアルは，78K0R マイクロコントローラのフラッシュ・セルフ・プログラミング機能を理解し，それを用いたアプリケーション・システムを設計するユーザを対象としています。

このユーザーズ・マニュアルは，78K0R マイクロコントローラのフラッシュ・メモリの書き換えを行うために使用するフラッシュ・セルフ・プログラミング・ライブラリの使用方法を理解していただくことを目的としています。

動作確認デバイス

78K0R/Fx3, 78K0R/Hx3

目 次

第1章　概　説

1.1　概　要

フラッシュ・セルフ・プログラミング・ライブラリは，78K0Rマイクロコントローラに搭載されたファームウエアを使用し，フラッシュ・メモリ内のデータを書き換えるためのソフトウエアです。

フラッシュ・セルフ・プログラミング・ライブラリをユーザ・プログラムから呼び出すことにより，フラッシュ・メモリの内容を書き換えることができるため，ソフトウエアの開発期間を大幅に短縮することが可能となります。

なお，本フラッシュ・セルフ・プログラミング・ライブラリ・ユーザーズ・マニュアルは，対象デバイスのマニュアルと合わせてご使用ください。

用　　語　このマニュアルで使用する用語について，その意味を次に示します。

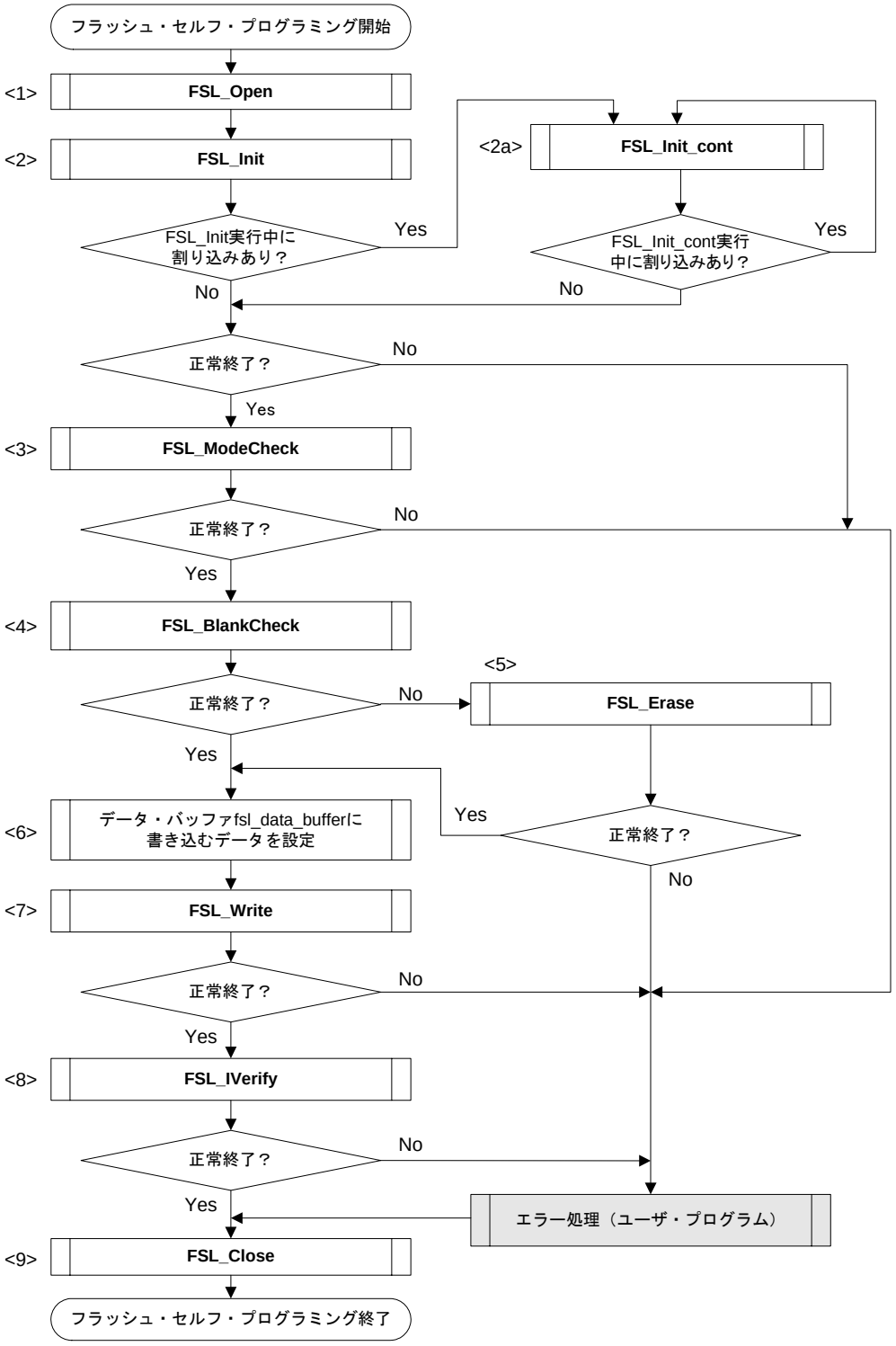
- **フラッシュ・セルフ・プログラミング**
  ユーザ・プログラム自身によるフラッシュ・メモリへの書き込み動作です。
- **フラッシュ・セルフ・プログラミング・ライブラリ**
  78K0Rマイクロコントローラが提供する機能によるフラッシュ・メモリ操作のためのライブラリです。
- **フラッシュ環境**
  フラッシュ・メモリの操作が可能である状態です。通常のプログラムの実行とは異なる制限事項があります。
- **ブロック番号**
  フラッシュ・メモリのブロックを示す番号です。消去，ブランク・チェック，ベリファイ（内部ベリファイ）の操作単位です。
- **ブート・クラスタ**
  ブート・スワップに使用するブート領域です。ブート・クラスタ0とブート・クラスタ1があり，ブートするクラスタを選択できます。
- **内部ベリファイ**
  フラッシュ・メモリへの書き込み後，フラッシュ・メモリのセルの信号レベルが適正であるかを確認することです。内部ベリファイでエラーとなった場合，そのデバイスは故障していると判断されます。ただし，内部ベリファイ・エラー後に，再度，データの消去→書き込み→内部ベリファイを実行し，正常終了した場合には，そのデバイスは正常であると判断します。
- FSL
  フラッシュ・セルフ・プログラミング・ライブラリの略称です。
- FSW
  フラッシュ・シールド・ウインドウの略称です。

1. 2　フラッシュ・セルフ・プログラミング・ライブラリの呼び出し方法

フラッシュ・セルフ・プログラミング・ライブラリは，C言語，およびアセンブリ言語どちらのユーザ・プログラムからも呼び出し可能です。

図1－1にフラッシュ・セルフ・プログラミング・ライブラリを利用したフラッシュ・メモリ書き換えフローを示します。

図 1－1　フラッシュ・セルフ・プログラミング（フラッシュ・メモリの書き換え）のフロー



<1> **前処理**

ユーザがFSL_Open**関数を呼び出し，割り込みの保存と設定を行います（任意）。**

FLMD0**端子はハイ・レベル状態になります。**

<2> **フラッシュ・セルフ・プログラミングで使用する**RAM**の初期化**

FSL_Init**関数を呼び出し，フラッシュ・セルフ・プログラミングで使用する**RAM**を初期化します。**

<2a> FSL_Init_cont**関数を呼び出し，割り込み後のフラッシュ・セルフ・プログラミングで使用する**RAM**の初期化プロセスを継続します。**

<3> **電圧レベルの確認**

FSL_ModeCheck**関数を呼び出し，**FLMD0**端子の電圧レベルを確認します。**

<4> **指定ブロック（**1 K**バイト）のブランク・チェック**

FSL_BlankCheck**関数を呼び出し，指定ブロック（**1 K**バイト）のブランク・チェックを行います。**

<5> **指定ブロック（**1 K**バイト）の消去**

FSL_Erase**関数を呼び出し，指定ブロック（**1 K**バイト）の消去を行います。**

<6> **フラッシュ・メモリに書き込むデータをデータ・バッファに入れます。**

<7> **指定アドレスに対する**1-64**ワードのデータ書き込み**

FSL_Write**関数を呼び出し，指定アドレスに対する**1-64**ワード・データの書き込みを行います。**

<8> **指定ブロック（**1 K**バイト）のベリファイ（内部ベリファイ）**

FSL_IVerify**関数を呼び出し，指定ブロック（**1 K**バイト）をベリファイ（内部ベリファイ）します。**

<9> **後処理**

FSL_Close**関数を呼び出し，保存しておいた割り込みマスク状態に戻します。（任意）。**

FLMD0**をプルダウンします。**

備考　1**ワード** = 4**バイト**

1. 3　ブロック番号

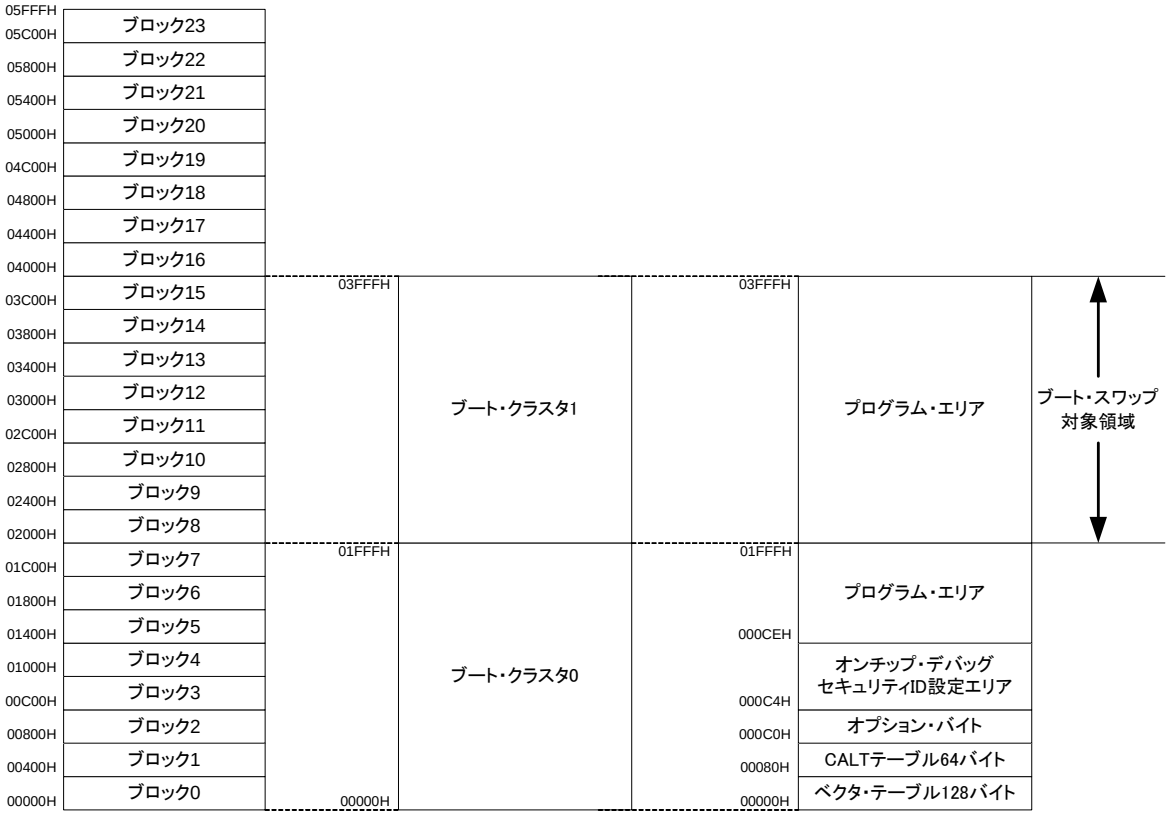
78K0Rマイクロコントローラは，フラッシュ・メモリが1 Kバイト単位でブロック分割されています。フラッシュ・セルフ・プログラミングでは，このブロックを単位として消去処理，ブランク・チェック処理，ベリファイ（内部ベリファイ）処理を行います。これらのフラッシュ・セルフ・プログラミング・ライブラリを呼び出す際には，ブロック番号を指定します。

また，78K0Rマイクロコントローラの00000H-01FFFH，02000H-03FFFH はブート・クラスタに割り付けられています。

なお，ブート・クラスタは，ベクタ領域を含むエリアを書き換える際，電源の瞬断，書き換え中のリセットなどにより，ベクタ・テーブル・データ，プログラムの基本機能などが破壊され，ユーザ・プログラムが立ち上がらなくなることを防止するための領域です。詳細については，**第5章　ブート・スワップ機能**を参照してください。

図1－2に，ブロック番号とブート・クラスタを示します。

図 1−2　ブロック番号とブート・クラスタ（フラッシュ・メモリ・サイズが 24 K バイトの場合）



第2章　プログラミング環境

この章では，ユーザがフラッシュ・セルフ・プログラミング・ライブラリを使用してフラッシュ・メモリの書き換えを行ううえで，必要なハードウエア環境とソフトウエア環境について説明します。

2. 1　ハードウエア環境

FLMD0端子の電圧は，通常のユーザ・プログラム実行中はロウ・レベル（通常動作モード）に，フラッシュ・セルフ・プログラミング実行中はハイ・レベル（フラッシュ書き換えモード）にする必要があります。

このFLMD0端子がロウ・レベルの際は，書き換え用のファームウエア，およびソフトウエアは動作しますが，フラッシュ・メモリの書き換え回路は動作しません。このため，実際の書き換え動作は行われません。

なお，FLMD0端子についての詳細は，対象デバイスのマニュアルを参照してください。

78K0Rマイクロコントローラは，FLMD0端子のプルアップ機能とプルダウン機能を内蔵しており，フラッシュ・セルフ・プログラミングの関数をコールすることによってハイ・レベルやロウ・レベルにすることが可能です。

・FLMD0端子の電圧を切り替えるには

FSL_Open関数を呼び出すと，FLMD0端子がプルアップされます。

FSL_Close関数を呼び出すと，FLMD0端子がプルダウンされます。

注意 1. FLMD0 端子を 100 kΩ未満でプルダウンしている場合，FSL_Open 関数をコールしても FLMD0 端子はハイ・レベルになりません。この場合は，チップ外部で FLMD0 端子にハイ・レベルを入力してください。

2. FSL_Open 関数，FSL_Close 関数内（ファイル名：fsl_user.c, fsl_user.h）の FLMD0 端子制御を変更しないでください。ファイル fsl_user.c と fsl_user.h 内の FLMD0 端子制御の内容は次のとおりです。

fsl_user.c

```
void FSL_Open(void)
{
   /* ..... */
   /* switch FLMD0 to HIGH      */
   FSL_FLMD0_HIGH;
   /* ..... */
}
・・・・
void FSL_Close(void)
{
   /* ..... */
   /* switch FLMD0 to LOW       */
   FSL_FLMD0_LOW;
   /* ..... */
}
```

fsl_user.h

```
/* FLMD0 control bit                    */
#define  FSL_FLMD0_HIGH   {BECTL.7 = 1;}
#define  FSL_FLMD0_LOW    {BECTL.7 = 0;}
```

備考　FSL_Open 関数，FSL_Close 関数は，FLMDPUP フラグを制御することにより，FLMD0 端子の処理を切り替えています。

図 2－1　バック・グラウンド・イベント・コントロール・レジスタ（BECTL）のフォーマット

アドレス：FFFBEH　リセット時：00H　R/W

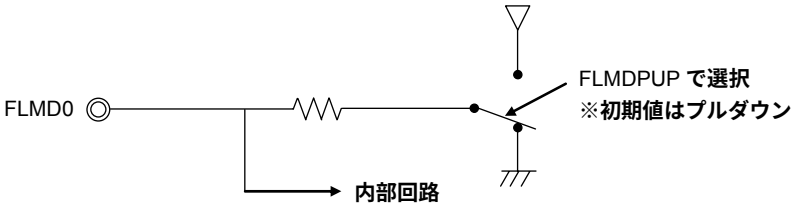
略号	7	6	5	4	3	2	1	0
BECTL	FLMDPUP	0	0	0	0	0	0	0

FLMDPUP	FLMD0端子のソフトウエア制御
0	プルダウン選択
1	プルアップ選択

表 2－1　FLMD0 端子の端子処理と FLMD0 端子の電圧レベルの切り替え

FLMD0 端子処理	ソフト引き込み可否
オープン	可
100 kΩ 以上の抵抗でプルダウン	可
100 kΩ 未満の抵抗でプルダウン	保証しない
V_{DD} に直接接続	不可

図 2－2　FLMD0 電圧の内部回路生成回路



2.2 ソフトウエア環境

フラッシュ・セルフ・プログラミング・ライブラリでは，該当プログラムをユーザ領域に配置するため，使用するライブラリの容量のプログラム領域を消費します。また，フラッシュ・セルフ・プログラミング・ライブラリ自身は，CPU（レジスタ・バンク3），スタック，データ・バッファ，ライブラリ用データ退避エリアを使用します。

必要となるソフトウエア・リソースについては，インストーラに添付されている文書「使用上の留意点」を参照してください。

2.2.1 レジスタ・バンク

フラッシュ・セルフ・プログラミング・ライブラリが使用するレジスタ・バンク。レジスタ・バンク3を使用します。

フラッシュ・セルフ・プログラミング関数内でレジスタ・バンクが自動的に切り替わるため，ユーザによるバンク3切り替え設定は必要ありませんが，フラッシュ・セルフ・プログラミング実行中に割り込みを使用する場合はバンク切り替え設定が必要です。バンク切り替え設定を行わない場合，割り込み処理内でレジスタ・バンク3が使用され，フラッシュ・セルフ・プログラミングで使用していたレジスタ値が破壊されてしまいます。

2.2.2 スタック，データ・バッファ，ライブラリ用データ退避エリア

フラッシュ・セルフ・プログラミング・ライブラリは，CPUを使用してフラッシュ・メモリへの書き込みを行います。このため，フラッシュ・セルフ・プログラミング動作は，ユーザ・プログラムで指定されているスタックを使用して行います。

備考 ユーザが指定するアドレスに，スタック，データ・バッファ，ライブラリ用データ退避エリアを配置するためには，リンク・ディレクティブを使用します。

・スタック

スタックは，フラッシュ・セルフ・プログラミング動作におけるスタック処理で，ユーザ使用RAMが破壊されないように配置する必要があります。スタックの指定可能範囲は，FFE20H-FFEFFH以外の内部RAMとなります。

・データ・バッファ

データ・バッファは，フラッシュ・セルフ・プログラミング・ライブラリ内部処理で使用するワーク領域，またはFSL_Write関数では設定するデータを配置する領域として使用します。なお，このデータ・バッファはFSL_user.cファイル内にバッファ名：fsl_data_bufferですでに定義されていますが，ユーザによって定義されたfsl_data_buffer(データ・バッファ)以外のバッファを使用していただくことも可能です。

データ・バッファの先頭アドレスの指定可能範囲は，スタックと同様，FFE20H-FFEFFH以外の内部RAMとなります。

・ライブラリ用データ退避エリア

ライブラリ用データ退避エリアは，フラッシュ・セルフ・プログラミング・ライブラリがデータ退避用に使用するエリアです。ライブラリ用データ退避エリアの指定可能範囲は，スタックと同様，FFE20H-FFEFFH以外の内部RAMとなります。

なお，フラッシュ・メモリに書き込むデータについては，FSL_Write関数を呼び出す前に設定処理を完了してください。

2. 2. 3　フラッシュ・セルフ・プログラミング・ライブラリ

すべてのライブラリ関数がリンクされるのではなく，使用するライブラリ関数に限定してリンクします。

・フラッシュ・セルフ・プログラミング・ライブラリのメモリ配置について

フラッシュ・セルフ・プログラミング・ライブラリでは使用する関数や変数にセグメントが割り当てられており，フラッシュ・セルフ・プログラミング・ライブラリで使用される領域を特定の場所に指定することができます。

詳しくは，インストーラに添付されている文書「使用上の留意点」を参照してください。

2. 2. 4　プログラム領域

フラッシュ・セルフ・プログラミング関数をコールするユーザ・プログラムです。ユーザ・プログラムのため，容量はユーザによる作成方法に依存します。

2. 3　プログラミング環境に関する注意事項

(1) CPUがサブシステム・クロック動作時の場合，フラッシュ・セルフ・プログラミング機能は使用できません。

(2) ユーザがフラッシュ・セルフ・プログラミング実行中，ソフトウエア・リソースを操作した場合，フラッシュ・セルフ・プログラミングの動作は保証されません。フラッシュ・セルフ・プログラミングの動作中は，これらのリソースを操作しないでください。

(3) ユーザは，フラッシュ・セルフ・プログラミング・ライブラリを呼び出す前に，フラッシュ・セルフ・プログラミング・ライブラリが使用するソフトウエア・リソースを解放および確保する必要があります。

(4) FSL_CNST（セグメント名）はFSL_Init関数，FSL_Init_cont関数で使用する定数を配置するためのセグメントです。そのため，FSL_CNST（セグメント名）を消去対象ブロックに配置し，消去したあとに，FSL_Init関数，FSL_Init_cont関数を実行すると正常に実行できません。

(5) EEPROMエミュレーション・ライブラリとの同時使用はできません。フラッシュ・セルフ・プログラミング機能を使用する場合は，必ずEEPROMエミュレーション・ライブラリを終了させてください。

第3章　フラッシュ・セルフ・プログラミング中の割り込み処理

3.1　概　要

78K0Rマイクロコントローラでは，フラッシュ・セルフ・プログラミングを実行している状態でも，一部のフラッシュ・セルフ・プログラミング・ライブラリ関数については割り込みを発生させることが可能です。

ただし，通常の割り込みとは異なり，割り込み処理後に，中断されていた処理を再び行うか否かは，フラッシュ・セルフ・プログラミング・ライブラリ関数からの戻り値を見てユーザが判断することになります。割り込み後にただちにユーザ・プログラムに制御を戻すか，または割り込み後もフラッシュ・セルフ・プログラミング・ライブラリを続行するかを指定することが可能です。

表3－1に，フラッシュ・セルフ・プログラミング・ライブラリの割り込みの受け付け可否について示します。

表 3－1　割り込みの受け付け

関数名	割り込み受け付け
FSL_Open	可
FSL_Close	
FSL_Init	
FSL_Init_cont	
FSL_ModeCheck	
FSL_BlankCheck	
FSL_Erase	
FSL_IVerify	
FSL_Write	
FSL_EEPROMWrite	
FSL_GetSecurityFlags	不可
FSL_GetActiveBootCluster	
FSL_GetBlockEndAddr	
FSL_GetFlashShieldWindow	
FSL_InvertBootFlag	可
FSL_SwapBootCluster	不可
FSL_ForceReset	可
FSL_SwapActiveBootCluster	
FSL_SetChipEraseProtectFlag	
FSL_SetBlockEraseProtectFlag	
FSL_SetWriteProtectFlag	
FSL_SetBootClusterProtectFlag	
FSL_SetFlashShieldWindow	
FSL_SetInterruptMode	不可

割り込みの受け付けが可のフラッシュ・セルフ・プログラミング・ライブラリの関数には，フラッシュ・セルフ・プログラミング実行中に割り込みの発生により中断したか判断する機能と，中断していた場合の後処理を行う機能があります。

3. 2　フラッシュ・セルフ・プログラミング・ライブラリ実行中の割り込み

フラッシュ・セルフ・プログラミング・ライブラリ実行中に割り込みを受け付ける場合，割り込み後の動作を指定することが可能です。

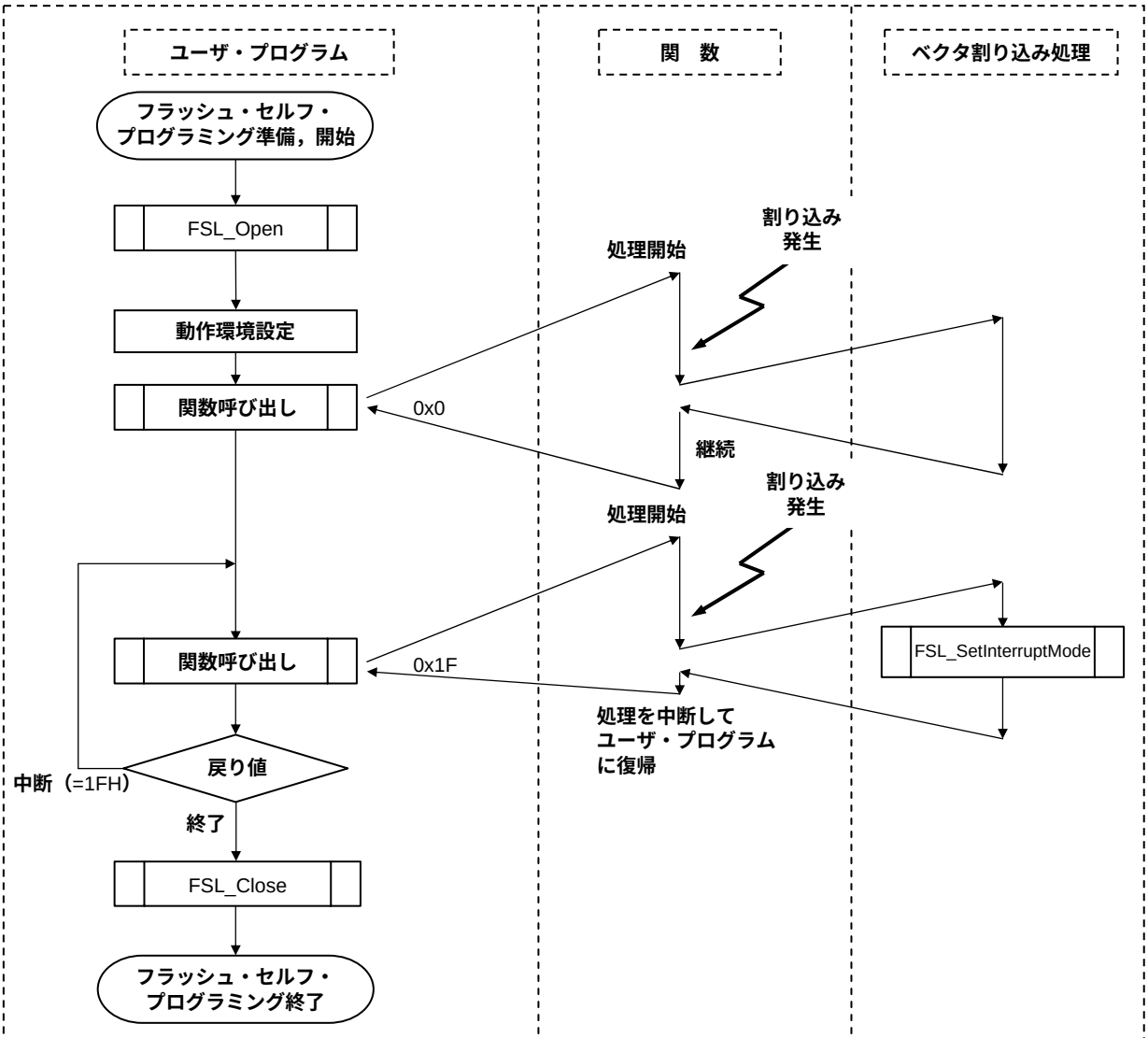
デフォルトでは，割り込み終了後も関数の実行を継続しますが，割り込み処理内でFSL_SetInterruptMode関数を呼び出すと，割り込み処理終了後に関数の実行をただちに中断し，ユーザ・プログラムに復帰します。

表 3－2　割り込み処理後の動作

割り込み処理内での FSL_SetInterruptMode 関数の呼び出し	割り込み処理後の動作
なし	関数の実行を継続
あり	関数の実行を中断してユーザ・プログラムへ復帰

図3－1に，フラッシュ・セルフ・プログラミング・ライブラリ関数の処理を実行中に割り込みが発生し，割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合の処理フローを示します。

図 3−1　割り込み発生時の処理のフロー



関数実行中に割り込みが発生すると，割り込みベクタ処理が終了後，再び関数の実行を継続します。この場合は戻り値として0x0を返します。
ただし，割り込み処理内でFSL_SetInterruptMode関数を実行すると，ベクタ割り込み処理後，関数の実行を中断してただちにユーザ・プログラムに復帰します。この場合は戻り値として0x1Fを返します。

表3－3に割り込みの発生により中断されていたフラッシュ・セルフ・プログラミング・ライブラリの処理の再開方法を示します。

表3－3以外のフラッシュ・セルフ・プログラミング・ライブラリについては，割り込み禁止状態で該当処理を実行するため，すべての処理が完了するまで割り込みが受け付けられることはありません。

表3－3　割り込みの発生により中断されていた処理の再開方法

関数名	再開方法
FSL_Init関数 FSL_Init_cont関数	割り込みの発生により中断されていたFSL_Init関数，FSL_Init_cont関数を再開する場合，FSL_Init_cont関数を呼び出します。
FSL_BlankCheck関数	割り込みの発生により中断されていたFSL_BlankCheck関数を再開する場合，FSL_BlankCheck関数を再度呼び出します。
FSL_Erase関数	割り込みの発生により中断されていたFSL_Erase関数を再開する場合，FSL_Erase関数を再度呼び出します。
FSL_Write関数	割り込みの発生により中断されていたFSL_Write関数を再開する場合，FSL_Write関数を再度呼び出します。
FSL_IVerify関数	割り込みの発生により中断されていたFSL_IVerify関数を再開する場合，FSL_IVerify関数を再度呼び出します。
FSL_SetXXX関数	割り込みの発生により中断されていたFSL_SetXXX関数を再開する場合，FSL_SetXXX関数を再度呼び出します。

注意1. **各フラッシュ・セルフ・プログラミング・ライブラリ関数を再開する場合は同じ引数を設定してください。**

2. **上記の関数から呼び出されたフラッシュ・セルフ・プログラミング・ライブラリ関数で，一部割り込みを受け付けない区間があります。**

3. 3　割り込みに関する注意事項

- フラッシュ・セルフ・プログラミング中に割り込みを禁止するためには，通常動作モード時と同様に，DI 命令により IE フラグがクリア（0）されている状態でフラッシュ・セルフ・プログラミング・ライブラリを実行してください。割り込みを許可する場合は，EI 命令により IE フラグがセット（1）されている状態で，受け付ける割り込みの割り込みマスク・フラグをクリア（0）して，フラッシュ・セルフ・プログラミング・ライブラリを実行してください。

- フラッシュ・セルフ・プログラミングではレジスタ・バンク 3 を使用するため，割り込み処理でレジスタ・バンク 3 を使用しないでください。

- フラッシュ・セルフ・プログラミング・ライブラリ内では，ライブラリ関数呼び出し前のレジスタ・バンクを使用している区間と，レジスタ・バンク 3 を使用している区間が混在します。このため，特定のレジスタ・バンクを割り込み処理で使用する場合は，使用するバンクに切り替え，使用後は元のレジスタ・バンクに戻してください。

- 割り込み処理の中で使用するレジスタは，割り込み処理の中で退避，および復帰させてください。

- フラッシュ・セルフ・プログラミング実行中もウォッチドッグ・タイマは停止しません。

- 割り込みの発生により中断されたフラッシュ・セルフ・プログラミング・ライブラリ関数の処理を再開せず，別のフラッシュ・セルフ・プログラミング・ライブラリ関数を実行したり，別ブロックに対してフラッシュ・セルフ・プログラミング・ライブラリ関数を呼び出したりしないでください。また，割り込みの発生により中断された処理は，必ず終了するまでリトライして，処理を終了してください。

- スタック，およびデータ・バッファは，一連の処理が終了するまで破壊しないでください。

- フラッシュ・セルフ・プログラミングを実行中に発生した割り込み処理の中で，割り込みマスク・フラグ・レジスタの設定変更を行っても，その設定は割り込み処理終了後に無効化される場合があります。フラッシュ・セルフ・プログラミングを実行中に発生する割り込み処理の中では，割り込みマスク・フラグ・レジスタの設定変更を行わないでください。

- OS 上でフラッシュ・セルフ・プログラミングを実行する場合は，RX78K0R-S01　バージョン V4.20，RX78K0R-xxx　バージョン V4.20 以上をご使用ください。

第4章　セキュリティ設定

フラッシュ・メモリに書かれたユーザ・プログラムの書き換えを禁止するセキュリティ機能をサポートしており，第三者によるプログラムの改ざん防止などに対応可能となっています。
なお，セキュリティ設定についての詳細は，対象デバイスのマニュアルを参照してください。

4. 1　セキュリティ・フラグ

フラッシュ・セルフ・プログラミング・ライブラリはセキュリティ・フラグを設定する関数を実装しています（この関数のAPIの詳細は**第6章　フラッシュ・セルフ・プログラミング・ライブラリ**を参照してください）。

セキュリティ・フラグを設定する関数	機　　能
FSL_SetChipEraseProtectFlag	チップ消去許可フラグの設定
FSL_SetBlockEraseProtectFlag	ブロック消去許可フラグの設定
FSL_SetWriteProtectFlag	書き込み許可フラグの設定
FSL_SetBootClusterProtectFlag	ブート領域書き換え許可フラグの設定

4. 2　フラッシュ・シールド・ウインドウ機能

フラッシュ・セルフ・プログラミング時のセキュリティ機能の一つとして，フラッシュ・シールド・ウインドウ機能があります。フラッシュ・シールド・ウインドウ機能は，指定したウインドウ範囲以外の書き込みおよび消去を，フラッシュ・セルフ・プログラミング時のみ禁止にするセキュリティ機能です。ウインドウ範囲は，スタート・ブロックとエンド・ブロックを指定することで設定できます。ウインドウ範囲以外の領域は，フラッシュ・セルフ・プログラミング時には書き込み／消去禁止となります。

フラッシュ・シールド・ウインドウを設定する関数	機　　能
FSL_SetFlashShieldWindow	フラッシュ・シールド・ウインドウの設定

第5章　ブート・スワップ機能

5.1　概　要

ベクタ・テーブル・データ，プログラムの基本機能，およびフラッシュ・セルフ・プログラミング領域の書き換え中に，電源の瞬断，外乱要因によるリセットの発生などにより書き換えが失敗した場合，書き換え中のデータが破壊され，その後のリセットによるユーザ・プログラムの再スタートや，再書き込みができなくなります。この問題を回避するための機能がブート・スワップ機能です。

5.2　ブート・スワップ機能

ブート・スワップ機能では，ブート・プログラム領域であるブート・クラスタ0[注]とブート・スワップ対象領域であるブート・クラスタ1[注]を置換します。

書き換え処理を行う前に，あらかじめ新しいブート・プログラムをブート・クラスタ1に書き込んでおきます。このブート・クラスタ1とブート・クラスタ0をスワップし，ブート・クラスタ1をブート・プログラム領域にします。

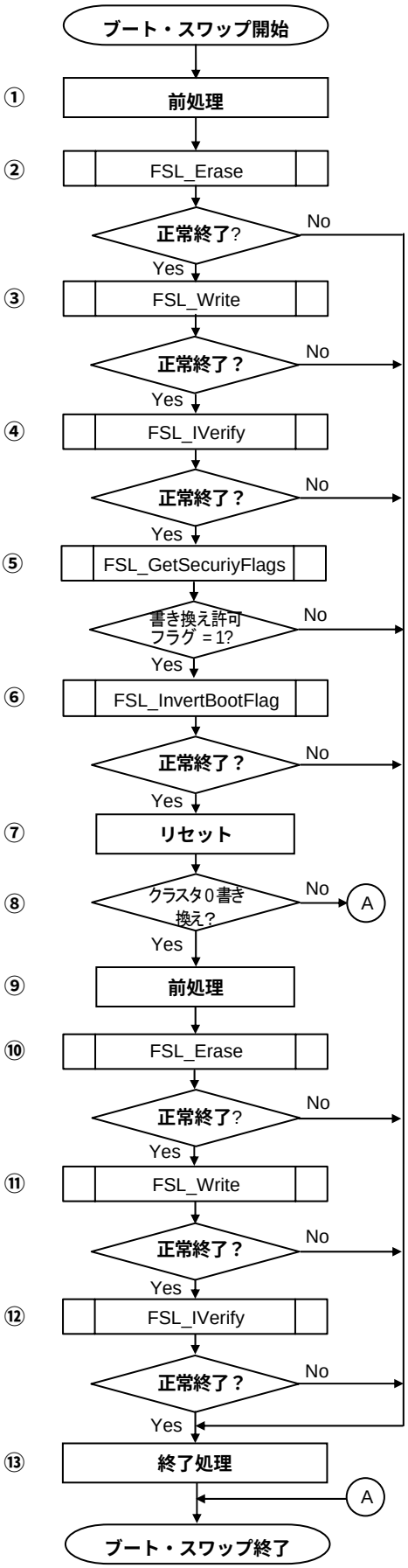
これによって，ブート・プログラム領域の書き換え中に電源の瞬断が発生しても，次のリセット・スタートはブート・クラスタ1からブートを行うため，正常にプログラムが動作します。このあと，必要であれば，ブート・クラスタ0への消去や書き込み処理を行うことも可能です。

注 ブート・クラスタ0（00000H-01FFFH）：本来のブート・プログラム領域
　ブート・クラスタ1（02000H-03FFFH）：ブート・スワップ対象領域

5.3　ブート・スワップ手順

図5－1に，フラッシュ・セルフ・プログラミング・ライブラリを利用してブート・スワップを行うフローを示します。

図 5−1　ブート・スワップのフロー



① 前処理

ブート・スワップの前処理として，

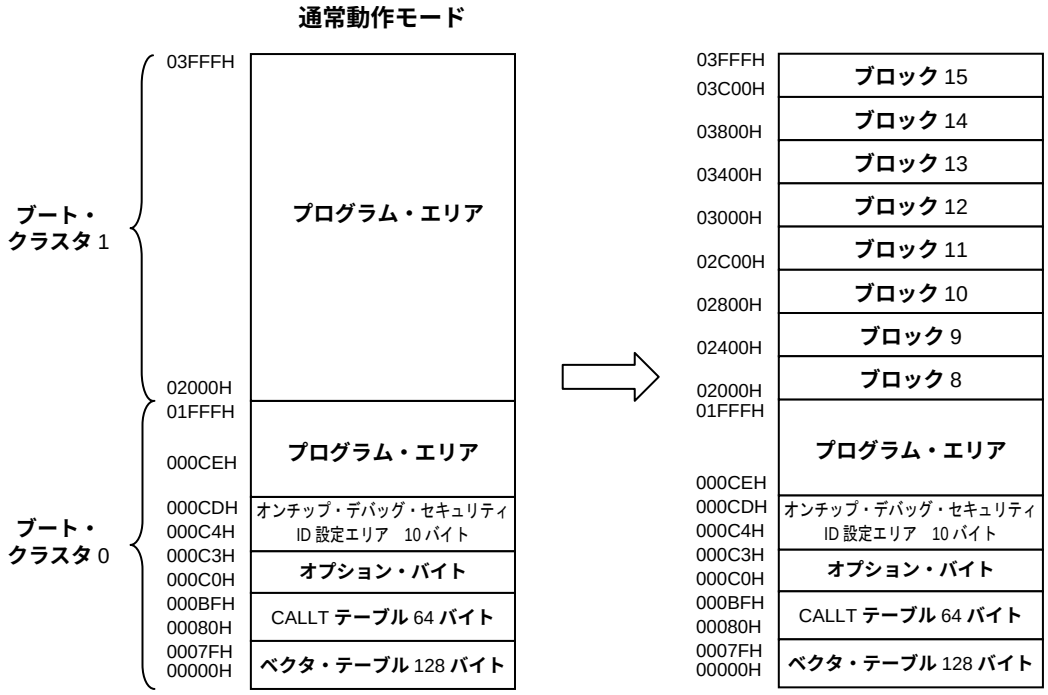
- ハードウエア環境の設定(FLMD0 端子をハイレベルに設定)
- フラッシュ・セルフ・プログラミングの開始宣言(FSL_Open 関数)
- ソフトウエア環境の設定(データバッファの確保など)
- RAM の初期化(FSL_Init 関数)
- 電圧レベルの確認(FSL_ModeCheck 関数)

を行います。

② ブート・クラスタ 1 の消去

FSL_Erase 関数の呼び出しにより，ブロック 8-ブロック 15 を消去します。

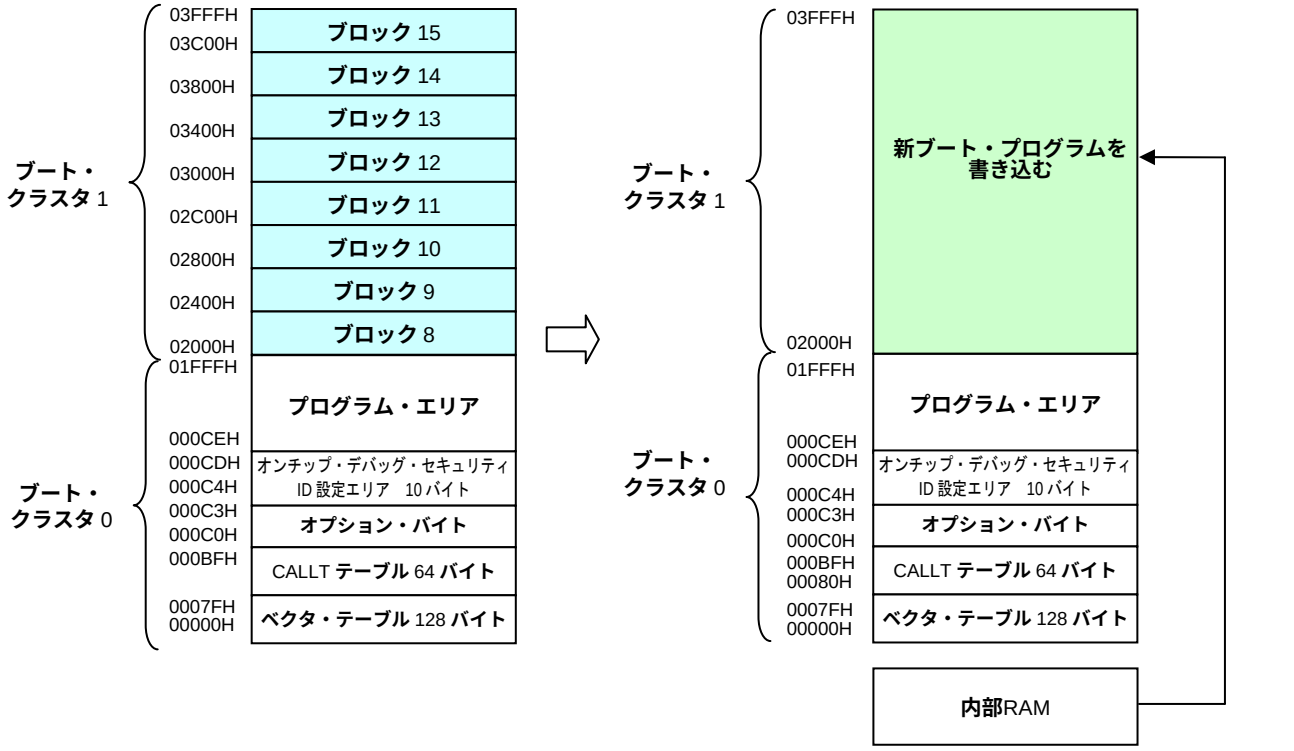
備考　FSL_Erase 関数はブロック単位で消去を行います。



③ ブート・クラスタ１へ新しいブート・プログラムをコピー

FSL_Write 関数の呼び出しにより，新ブート・プログラム（ブート・スワップ処理後に 00000H-01FFFH に配置したいプログラム）を 02000F-03FFFH に書き込みます。

備考 FSL_Write 関数はワード単位（1 ワード = 4 バイト，最大 64 ワード（256 バイト））で書き込みを行います。



新ブート・プログラムを，外部I/F（3線式SIO，UARTなど）経由で内部RAMにダウン・ロードし，順次書き込みます。

④ ブート・クラスタ 1 のベリファイ

FSL_IVerify 関数の呼び出しにより，ブロック 8-ブロック 15 をベリファイします。

備考 FSL_IVerify 関数はブロック単位でベリファイを行います。

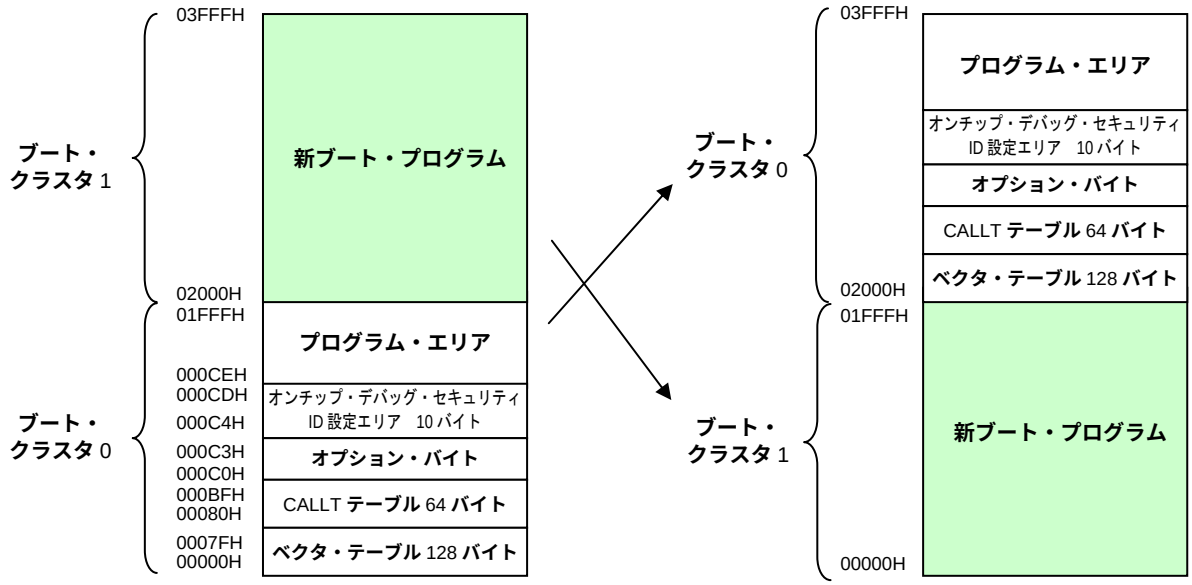
⑤ ブート・スワップ・ビットの確認（推奨）

FSL_GetSecurityFlags 関数の呼び出しにより，セキュリティ・フラグ情報を取得し，ブート領域書き換え許可フラグが 1（許可）になっていることを確認します。

備考 ブート領域書き換え許可フラグが 0（禁止）になっている場合，⑥の FSL_InvertBootFlag 関数の呼び出しにより，エラーとなります。

⑥　ブート・スワップ・ビットの設定
FSL_InvertBootFlag 関数を実行することにより,ブート・フラグの切り替えを行います。

⑦　イベントの発生
リセットを発生させることにより，ブート・クラスタ 1 がブート・プログラム領域になります。



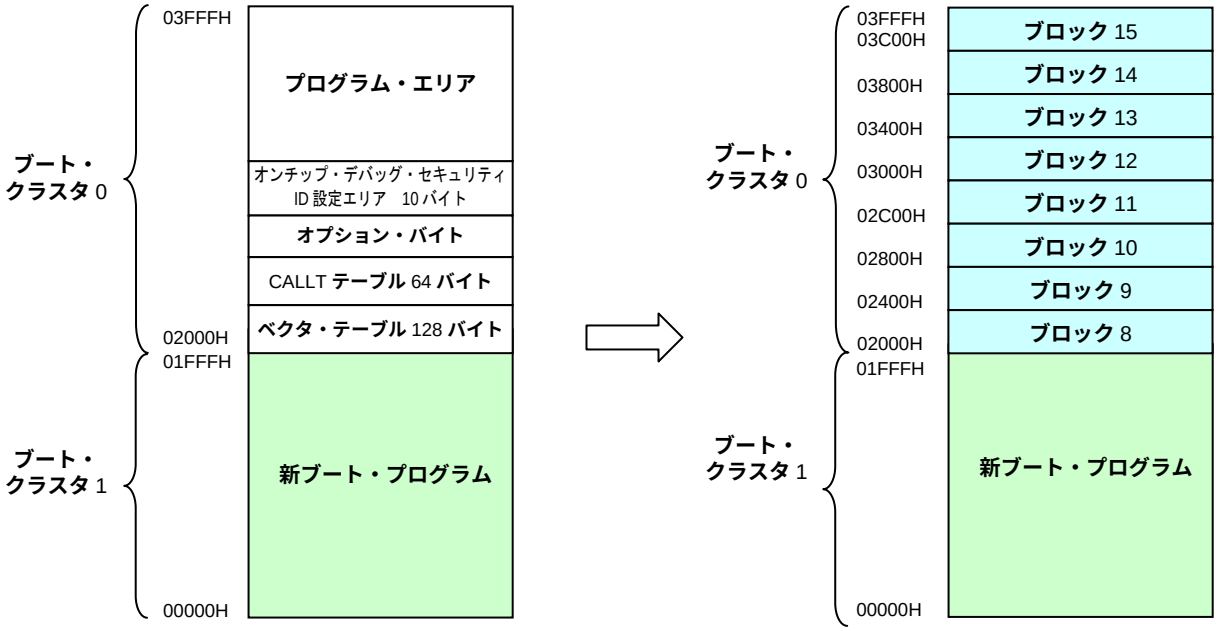
⑧　スワップ処理（ブート・クラスタ 1）の終了
②-⑦の操作により，ブート・クラスタ 1 に対するスワップ処理が終了となります。
ブート・クラスタ 0 を書き換える必要がない場合は，そのまま終了処理してください。
ブート・クラスタ 0 を書き換える必要がある場合は，⑨以降の処理を行ってください。

⑨　前処理
①と同様の処理を行います。

⑩ **ブート・クラスタ 0 の消去**

FSL_Erase **関数の呼び出しにより，ブート・クラスタ 0（ブロック 8-ブロック 15）を消去します。**

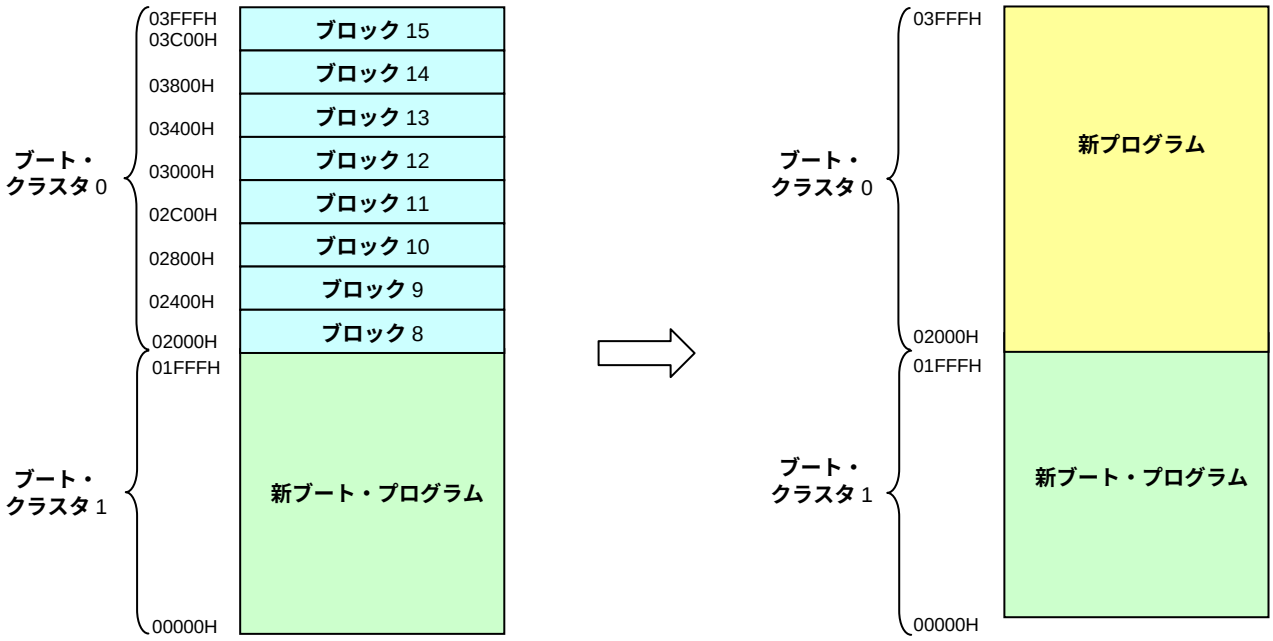
備考 FSL_Erase **関数はブロック単位で消去を行います。**



⑪ **ブート・クラスタ 0 への新しいプログラムの書き込み**

FSL_Write **関数の呼び出しにより，新しいプログラムの内容を** 02000H-03FFFH **に書き込みます。**

備考 FSL_Write **関数はワード単位（1 ワード** = 4 **バイト，最大** 64 **ワード（**256 **バイト））で書き込みを行います。**



⑫ ブート・クラスタ 0 のベリファイ
FSL_IVerify 関数の呼び出しにより，ブロック 8-ブロック 15 をベリファイします。

備考 FSL_IVerify 関数はブロック単位でベリファイを行います。

⑬ 終了処理
ブート・スワップの終了処理として，FSL_Close 関数を呼び出します。

5. 4 ブート・スワップに関する注意事項

ブート領域書き換え許可フラグが0（禁止）になっている場合，ブート・スワップは実行できません。

第6章　フラッシュ・セルフ・プログラミング・ライブラリ

この章では，フラッシュ・セルフ・プログラミング・ライブラリの詳細について説明しています。

6. 1　フラッシュ・セルフ・プログラミング・ライブラリの種類

フラッシュ・セルフ・プログラミング・ライブラリは，次に示すライブラリで構成されています。

表6－1　フラッシュ・セルフ・プログラミング・ライブラリ一覧

関数名	説　明
FSL_Open	フラッシュ・セルフ・プログラミングの開始宣言
FSL_Close	フラッシュ・セルフ・プログラミングの終了宣言
FSL_Init	フラッシュ・セルフ・プログラミング環境の初期化
FSL_Init_cont	フラッシュ・セルフ・プログラミング環境の初期化（割り込み後，初期化を継続）
FSL_ModeCheck	FLMD0端子の電圧レベルの確認
FSL_BlankCheck	指定ブロックのブランク・チェック
FSL_Erase	指定ブロックの消去
FSL_IVerify	指定ブロックのベリファイ（内部ベリファイ）
FSL_Write	指定アドレスに対する1-64ワード・データの書き込み（1ワード = 4バイト）
FSL_EEPROMWrite	指定アドレスに対する1-64ワード・データの書き込み（1ワード = 4バイト）注
FSL_GetSecurityFlags	セキュリティ情報の取得
FSL_GetActiveBootCluster	ブート・フラグ情報の取得
FSL_GetBlockEndAddr	指定したブロックの最終アドレスの取得
FSL_GetFlashShieldWindow	フラッシュ・シールド・ウインドウの開始ブロック番号と終了ブロック番号の取得
FSL_InvertBootFlag	ブート・フラグの現在値を反転
FSL_SwapBootCluster	ブート・スワップを実行し，スワップ後の領域のリセット・ベクタに登録されているアドレスから処理を実行
FSL_ForceReset	使用中のマイクロコントローラをリセット
FSL_SwapActiveBootCluster	ブート・フラグの現在値を反転し，ブート・スワップの実行
FSL_SetChipEraseProtectFlag	各セキュリティ・フラグの設定
FSL_SetBlockEraseProtectFlag	
FSL_SetWriteProtectFlag	
FSL_SetBootClusterProtectFlag	
FSL_SetFlashShieldWindow	フラッシュ・シールド・ウインドウの開始ブロックと終了ブロックの設定
FSL_SetInterruptMode	割り込み後の動作の中断

注　EEPROMエミュレーション・ライブラリ専用のデータ書き込み処理です。

備考　アセンブリ言語で記述されたユーザ・プログラムから呼び出す場合，フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は "!"，それ以外の場合は "!!" で呼び出してください。

6. 2 戻り値，戻り型一覧

各戻り値の意味は次のとおりです。

表 6－2 戻り値一覧

定　　義	戻り値	説　　明
FSL_OK	0x00	正常終了
FSL_ERR_FLMD0	0x01	FLMD0がロウ・レベル
FSL_ERR_PARAMETER	0x05	パラメータ・エラー
FSL_ERR_PROTECTION	0x10	プロテクト・エラー
FSL_ERR_ERASE	0x1A	消去エラー
FSL_ERR_BLANKCHECK	0x1B	ブランク・チェック・エラー
FSL_ERR_IVERIFY	0x1B	内部ベリファイ・エラー
FSL_ERR_WRITE	0x1C	書き込みエラー
FSL_ERR_INTERRUPTION	0x1F	フラッシュ・セルフ・プログラミング実行中に割り込みあり

戻り型は次のとおりです。

表 6－3 戻り型一覧

	戻り値	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u08	C
NEC Large model		

6. 3　フラッシュ・セルフ・プログラミング・ライブラリ関数の説明

フラッシュ・セルフ・プログラミング・ライブラリ関数について，次の形式で説明します。

フラッシュ・セルフ・プログラミング・ライブラリ関数名

【概　要】
この関数の機能概要を示します。

【書　式】
＜C 言語＞
この関数を C 言語で記述されたユーザ・プログラムから呼び出す際の書式を示します。
＜アセンブラ＞
この関数をアセンブリ言語で記述されたユーザ・プログラムから呼び出す際の書式を示します。

【事前設定】
この関数の事前設定を示します。

【機　能】
この関数の機能詳細と注意事項を示します。

【呼び出し後のレジスタ状態】
この関数を呼び出した後のレジスタの状態を示します。

【引　数】
この関数の引数を示します。

【戻り値】
この関数からの戻り値を示します。
データ型定義

fsl_u08 - unsigned char

fsl_u16 - unsigned int

fsl_u32 - unsigned long int

【フロー】（ユーザ関数（FSL_Open, FSL_Close**）のみ）**
この関数の動作フローを示します。

FSL_Open

【概　要】

フラッシュ・セルフ・プログラミングの開始宣言

【書　式】

＜C言語＞

```
void FSL_Open(void)
```

＜アセンブラ＞

```
CALL !_FSL_Open または CALL !!_FSL_Open
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

なし

【機　能】

- フラッシュ・セルフ・プログラミング**の開始宣言を行います。**フラッシュ・セルフ・プログラミング**操作の最初に，この関数を呼び出してください。この関数終了後，**FLMD0 **端子はハイ・レベル状態になります。**
- **割り込みフラグの設定を一時的に退避（任意）**
- **この関数はユーザによって変更することが可能です（上記フローのユーザ・プログラム部分）。**

備考　この関数は fsl_user.c **ファイル内に配置されています。**

【呼び出し後のレジスタ状態】

レジスタは破壊されません。

注意　本関数にユーザ・コードを追加せず，ご使用された場合に限ります。

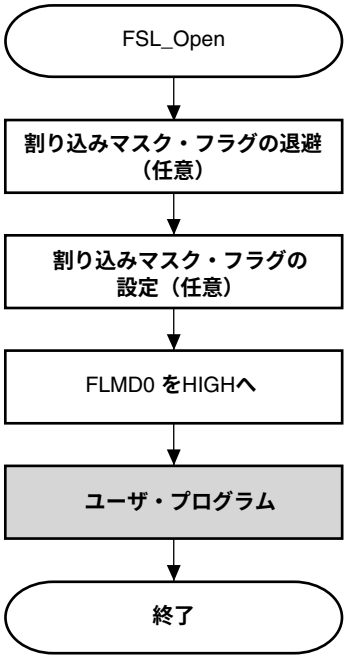
【引　数】

なし

【戻り値】

なし

【フロー】


FSL_Open
割り込みマスク・フラグの退避（任意）
割り込みマスク・フラグの設定（任意）
FLMD0 をHIGHへ
ユーザ・プログラム
終了

FSL_Close

【概　要】

フラッシュ・セルフ・プログラミングの終了宣言

【書　式】

＜C言語＞

```
void FSL_Close(void)
```

＜アセンブラ＞

```
CALL !_FSL_Close または CALL !!_FSL_Close
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

なし

【機　能】

・フラッシュ・セルフ・プログラミングの終了宣言を行います。フラッシュ・メモリへの書き込み操作を終了し，通常動作モード**に戻します。**フラッシュ・セルフ・プログラミング**操作の最後に，この関数を呼び出してください。この関数終了後，**FLMD0**端子はロウ・レベル状態になります。**

・割り込み設定を復帰（任意）。

・割り込みフラグの設定を復帰（任意）。

・この関数はユーザによって変更することが可能です（上記フローのユーザ・プログラム部分）。

備考　この関数は fsl_user.c **ファイル内に配置されています。**

【呼び出し後のレジスタ状態】

レジスタは破壊されません。

注意　本関数にユーザ・コードを追加せず，ご使用された場合に限ります。

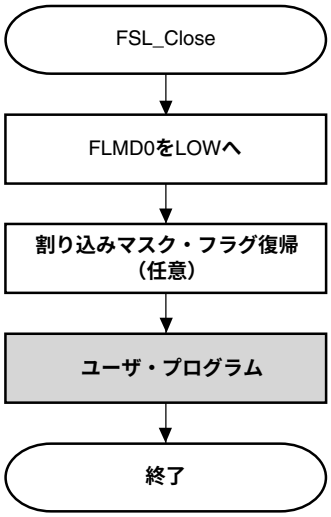
【引　数】

なし

【戻り値】

なし

【フロー】



FSL_Init

【概　要】

フラッシュ・セルフ・プログラミング環境の初期化

【書　式】

＜C言語＞

```
fsl_u08 FSL_Init(fsl_u08* fsl_data_buffer_pu08)
```

＜アセンブラ＞

```
CALL !_FSL_Init または CALL !!_FSL_Init
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

- **定数**fsl_fx_MHz_u08**が設定されている必要があります。この定数は使用する周波数を示します（例**: 20 MHz**の場合，**fsl_fx_MHz_u08 = 0x14)。
- **定数**fsl_low_voltage_u08**が設定されている必要があります。この定数で次のように電圧モードを定義します。**

00H：　**フルスピード・モード**（2.7V - 5.5V）

上記以外：　設定禁止

【機　能】

- **フラッシュ・セルフ・プログラミングで使用する**RAM**の確保，および初期化を行います。**
- **定数**fsl_fx_MHz_u08**によって**フラッシュ・セルフ・プログラミング**・ライブラリ内で使用するすべてのタイミング・データが計算されます。**

注意　関数の実行中に割り込みがあり，ベクタ割り込み処理内でFSL_SetInterruptMode**関数が呼び出された場合は，戻り値が**0x1F**となり，初期化プロセスを継続するには**FSL_Init_cont(…)**関数を呼び出す必要があります。**

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：AX（**引数**），ES，RB3

備考　アセンブリ言語の戻り値は C **レジスタに格納されます。**

【引　数】

引　数	説　明
Fsl_data_buffer_pu08	データ・バッファ（ライブラリ内で使用するワーク領域）。 FSL_user.cファイルに定義(fsl_data_buffer)されています。

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u08* fsl_data_buffer_pu08	AX
NEC Large model	fsl_u08* fsl_data_buffer_pu08	AX(0-15), C(16-23)

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了 - 初期設定が完了した状態
0x05(FSL_ ERR_PARAMETER)	パラメータ・エラー - 周波数の値が設定可能範囲外
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_Init_cont

【概　要】

フラッシュ・セルフ・プログラミング環境の初期化（割り込み後，初期化を継続）

【書　式】

＜C言語＞

```
fsl_u08 FSL_Init_cont(fsl_u08* fsl_data_buffer_pu08)
```

＜アセンブラ＞

```
CALL !_FSL_Init_cont または CALL !!_FSL_Init_cont
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

・FSL_Init関数で使用した引数および定数を変更せずに設定してください。

【機　能】

・フラッシュ・セルフ・プログラミングで使用するRAMの確保，および初期化を行います。

・定数fsl_fx_MHz_u08によってフラッシュ・セルフ・プログラミング・ライブラリ内で使用するすべてのタイミング・データが計算されます。

注意　関数の実行中に割り込みがあり，ベクタ割り込み処理内でFSL_SetInterruptMode関数が呼び出された場合は，戻り値が0x1Fとなり，初期化プロセスを継続するにはFSL_Init_cont(…)関数を呼び出す必要があります。

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：AX（引数），ES，RB3

【引　数】

引　　数	説　　明
fsl_data_buffer_pu08	データ・バッファ（ライブラリ内で使用するワーク領域）。 FSL_user.cファイルに定義(fsl_data_buffer)されています。

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u08* fsl_data_buffer_pu08	AX
NEC Large model	fsl_u08* fsl_data_buffer_pu08	AX(0-15), C(16-23)

【戻り値】

状　　態	説　　明
0x00(FSL_OK)	正常終了 - 初期設定が完了した状態
0x05(FSL_ ERR_PARAMETER)	パラメータ・エラー - 周波数の値が設定可能範囲外
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_ModeCheck

【概　要】

FLMD0 端子の電圧レベルの確認

【書　式】

＜C言語＞

```
fsl_u08 FSL_ModeCheck(void);
```

＜アセンブラ＞

```
CALL !_FSL_ModeCheck または CALL !!_FSL_ModeCheck
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

なし

【機　能】

FLMD0 端子の電圧レベル（ハイ・レベル／ロウ・レベル）を確認します。
FSL_Init 関数，FSL_Init_cont 関数の呼び出し後に，この関数を呼び出し，FLMD0 端子の電圧状態を確認してください。

注意　FLMD0端子がロウ・レベルの場合，フラッシュ・メモリの消去，書き込みなどの操作ができません。したがって，フラッシュ・セルフ・プログラミングでフラッシュ・メモリの操作を行う場合は，この関数を呼び出し，FLMD0端子の状態がハイ・レベルであることを確認する必要があります。また，書き換えの一連の処理が終了（FSL_Close実行）するまでハイ・レベルを一定に保つ必要があります。

【呼び出し後のレジスタ状態】

戻り値：C

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了 - FLMD0端子がハイ・レベル
0x01(FSL_ERR_FLMD0)	異常終了 - FLMD0端子がロウ・レベル

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_BlankCheck

【概　要】

指定ブロックのブランク・チェック

【書　式】

＜C言語＞

```
fsl_u08 FSL_BlankCheck(fsl_u16 block_u16)
```

＜アセンブラ＞

```
CALL !_FSL_BlankCheck または CALL !!_FSL_BlankCheck
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

指定したブロックのフラッシュ・メモリが消去レベルにあることを確認します。
エラーの場合は FSL_Erase **関数を実行してください。**
なお，FSL_Erase **関数の実行が正常終了した場合には，ブランク・チェックは不要です。**
指定したブロック番号がフラッシュ・メモリの最終ブロック番号より大きい場合，パラメータ・エラー（05H**）を返します。**

備考1. FSL_BlankCheck**関数は，フラッシュ・メモリのセルが十分なマージンを持って消去レベルを満たしているかを確認するものです。ブランク・チェック・エラーは，フラッシュ・メモリに問題があることを示すものではありませんが，ブランク・チェック・エラー後は，消去処理を行ってから書き込みをしてください。**

2. **ブランク・チェックは**1**ブロックのみ行うので，複数のブロックをブランク・チェックする場合は，この関数を複数回呼び出してください。**

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：AX**（引数），**ES，RB3

【引　数】

引　数	説　明
block_u16	ブランク・チェックするブロックのブロック番号

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u16 block_u16	AX
NEC Large model	fsl_u16 block_u16	AX

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了 - 指定したブロックがブランク状態
0x05(FSL_ERR_PARAMETER)	パラメータ・エラー - ブロック番号の指定が設定可能範囲外
0x1B(FSL_ERR_BLANKCHECK)	ブランク・チェック・エラー - 指定したブロックがブランク状態でない
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_Erase

【概　要】

指定ブロックの消去

【書　式】

＜C言語＞

```
fsl_u08 FSL_Erase(fsl_u16 block_u16)
```

＜アセンブラ＞

```
CALL !_FSL_Erase または CALL !!_FSL_Erase
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

指定したブロックのフラッシュ・メモリの内容を消去します。

指定したブロック番号が，フラッシュ・メモリ最終ブロック番号より大きい場合は，パラメータ・エラー（05H**）を返します。**

備考1. **消去は**1**ブロックのみ行うので，複数のブロックを消去する場合は，この関数を複数回呼び出してください。**

2. FSL_Erase**関数の実行が正常終了した場合，消去後のフラッシュ・メモリのセルの値を読み出すと**FFH**になります。ただし，フラッシュ・メモリのセルの値が**FFH**の場合に，ブロックがブランク状態であるということは保証できません。**

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：AX**（引数），**ES，RB3

【引　数】

引　数	説　明
block_u16	消去するブロックのブロック番号

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u16 block_u16	AX
NEC Large model	fsl_u16 block_u16	AX

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x05(FSL_ERR_PARAMETER)	パラメータ・エラー - ブロック番号の指定が設定可能範囲外
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - 指定ブロックがブート領域に含まれており，ブート領域書き換え許可フラグが禁止に設定されている - 指定したブロックがFSW設定領域外
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_IVerify

【概　要】

指定ブロックのベリファイ（内部ベリファイ）

【書　式】

＜C言語＞

```
fsl_u08 FSL_IVerify(fsl_u16 block_u16)
```

＜アセンブラ＞

```
CALL !_FSL_IVerify または CALL !!_FSL_IVerify
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

指定されたブロックへの書き込みレベルを確認するためのベリファイを行います。
このベリファイは，指定されたブロックのフラッシュ・メモリに書き込まれたデータが消去レベル（データ“1**”）／書き込みレベル（データ“**0**”）にあることを確認するものです。**
エラーの場合は FSL_Erase **を実行後，再度** FSL_Write **により書き込みを実行してください。**
指定したブロック番号がフラッシュ・メモリの最終ブロック番号より大きい場合，パラメータ・エラー（05H**）を返します。**

注意1. **データ書き込み後に，書き込みを行った範囲を含むブロックのベリファイ（内部ベリファイ）を実行しない場合，書き込んだデータは保証されません。**

2. **内部ベリファイ・エラー後に，再度，データの消去→書き込み→内部ベリファイを実行し，正常終了した場合には，そのデバイスは正常であると判断します。**

備考　ベリファイは1**ブロックのみ行うので，複数のブロックをベリファイする場合は，この関数を複数回呼び出してください。**

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：AX**（引数），**ES，RB3

【引　数】

引　数	説　明
block_u16	ベリファイするブロック番号

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u16 block_u16	AX
NEC Large model	fsl_u16 block_u16	AX

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x05(FSL_ERR_PARAMETER)	パラメータ・エラー - ブロック番号の指定が設定可能範囲外
0x1B(FSL_ERR_IVERIFY)	ベリファイ（内部ベリファイ）エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_Write

【概　要】

指定アドレスに対する 1-64 **ワード・データの書き込み**

備考　1 **ワード** = 4 **バイト**

【書　式】

＜C言語＞

```
fsl_u08 FSL_Write (fsl_u32 s_address_u32, fsl_u08 word_count_u08)
```

＜アセンブラ＞

```
CALL !_FSL_Write または CALL !!_FSL_Write
```

備考　**フラッシュ・セルフ・プログラミング・ライブラリを**00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

- **この関数の実行前に必ず**FSL_Open**関数と，**FSL_Init**関数（または**FSL_Init_cont**関数），**FSL_ModeCheck**関数を実行してください。**
- **この関数を呼び出す前に，フラッシュ・メモリに書き込むデータをデータ・バッファに格納してください。**

【機　能】

指定されたアドレスのフラッシュ・メモリへの書き込みを行います。
書き込みを実施したブロックに対しては，書き込み後，必ず FSL_IVerify **を実行してください。**
FSL_Write **関数は，消去されたブロックに対してのみ実行してください。**
一度に最大256**バイト（**4**バイト単位）のデータの書き込みを行うことができます。**
次の場合（指定されたワード数，アドレスが設定可能範囲外の場合），パラメータ・エラー（05H**）を返します。**

ワード数チェック

- **0ワードの場合**
- **65ワード以上の場合**

アドレス・チェック

- **先頭アドレスから4バイト単位の設定になっていない場合**
- **書き込み終了アドレスがフラッシュ・メモリの最終アドレスを越えた場合**

注意　**データの書き込み後，書き込みを行った範囲を含むブロックのベリファイ（内部ベリファイ）を実行してください。実行しない場合，書き込んだデータは保証されません。**

備考　256 **バイトを越えるデータを書き込む場合は，この関数を複数回呼び出してください。**

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：AX**（引数），**BC**（引数），**ES**，**RB3

【引　数】

引　　数	説　　明
s_address_u32	書き込むデータの開始アドレス
word_count_u08	書き込むデータ数（1-64：ワード単位）

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u32 s_address_u32 fsl_u08 word_count_u08	s_address_u32: AX(0-15), BC(16-31) word_count_u08: スタック
NEC Large model	fsl_u32 s_address_u32 fsl_u08 word_count_u08	s_address_u32: AX(0-15), BC(16-31) word_count_u08: スタック

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x05(FSL_ERR_PARAMETER)	パラメータ・エラー - 開始アドレスが1ワード（4バイト）の倍数でない - 書き込みデータ数が0 - 書き込みデータ数が64ワードを越えている - 書き込み終了アドレス（開始アドレス＋（書き込みデータ数×4バイト））がフラッシュ・メモリ領域を越えている
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - 指定範囲にブート領域が含まれており，ブート領域書き換え許可フラグが禁止に設定されている - 指定したブロックがFSW設定領域外
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，C レジスタに格納されます。

FSL_EEPROMWrite

【概　要】

指定アドレスに対する 1-64 **ワード・データの書き込み**[注]

注　この関数は EEPROM **エミュレーション・ライブラリ専用のデータ書き込み処理となります。**
セルフ・プログラミング・ライブラリを使用してのデータの書き込みについては，FSL_Write **関数をご使用ください。**

備考　1 **ワード** = 4 **バイト**

【書　式】

＜C言語＞

```
fsl_u08 FSL_EEPROMWrite (fsl_u32 s_address_u32, fsl_u08 word_count_u08)
```

＜アセンブラ＞

```
CALL !_FSL_EEPROMWrite または CALL !!_FSL_EEPROMWrite
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

- **この関数の実行前に必ず**FSL_Open**関数と，**FSL_Init**関数（または**FSL_Init_cont**関数），**FSL_ModeCheck**関数を実行してください。**
- **この関数を呼び出す前に，フラッシュ・メモリに書き込むデータをデータ・バッファに格納してください。**

【機　能】

指定されたアドレスのフラッシュ・メモリへの書き込みを行います。

一度に最大256**バイト（**4**バイト単位）のデータの書き込みを行うことができます。**

次の場合（指定されたワード数，アドレスが設定可能範囲外の場合），パラメータ・エラー（05H**）を返します。**

ワード数チェック

- **・**0**ワードの場合**
- **・**65**ワード以上の場合**

アドレス・チェック

- **・先頭アドレスから**4**バイト単位の設定になっていない場合**
- **・書き込み終了アドレスがフラッシュ・メモリの最終アドレスを越えた場合**

備考　256 **バイトを越えるデータを書き込む場合は，この関数を複数回呼び出してください。**

【呼び出し後のレジスタ状態】

戻り値　：C

破壊レジスタ：AX**（引数）**，BC**（引数）**，ES，RB3

【引　数】

引　数	説　明
s_address_u32	書き込むデータの開始アドレス
word_count_u08	書き込むデータ数（1-64：ワード単位）

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u32 s_address_u32 fsl_u08 word_count_u08	s_address_u32: AX(0-15), BC(16-31) word_count_u08: スタック
NEC Large model	fsl_u32 s_address_u32 fsl_u08 word_count_u08	s_address_u32: AX(0-15), BC(16-31) word_count_u08: スタック

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x05(FSL_ERR_PARAMETER)	パラメータ・エラー - 開始アドレスが1ワード（4バイト）の倍数でない - 書き込みデータ数が0 - 書き込みデータ数が64ワードを越えている - 書き込み終了アドレス（開始アドレス＋（書き込みデータ数×4バイト））がフラッシュ・メモリ領域を越えている
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - 指定範囲にブート領域が含まれており，ブート領域書き換え許可フラグが禁止に設定されている - 指定したブロックがFSW設定領域外
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1D(FSL_ERR_EEP_IVERIFY)	ベリファイ（内部ベリファイ）エラー - 書き込みデータのベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1E(FSL_ERR_EEP_BLANKCHECK)	ブランク・チェック・エラー - 指定した領域がブランク状態でない
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，C レジスタに格納されます。

FSL_GetSecurityFlags

【概　要】

セキュリティ情報の取得

【書　式】

＜C言語＞

```
fsl_u08 FSL_GetSecurityFlags(fsl_u16 *destination_pu16)
```

＜アセンブラ＞

```
CALL !_FSL_GetSecurityFlags または CALL !!_FSL_GetSecurityFlags
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

セキュリティ･フラグがデータ・バッファに書き込まれます。

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：AX**（引数），**BC，RB3

【引　数】

引　数	説　明
destination_pu16	**データ・バッファ** - **専用のデータ格納バッファを確保してください**

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u16 *destination_pu16	AX
NEC Large model	fsl_u16 *destination_pu16	AX(0-15), C(16-23)

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

セキュリティ・ビットはデータ・バッファに書き込まれます。各ビットの意味は次のとおりです。

ビット0：　チップ消去許可フラグ（0：禁止，1：許可）
ビット1：　ブロック消去許可フラグ（0：禁止，1：許可）
ビット2：　書き込み許可フラグ（0：禁止，1：許可）
ビット4：　ブート領域書き換え許可フラグ（0：禁止，1：許可）
ビット8-15：　ブート領域終了ブロック（ブート・クラスタ 0）
その他のビット = 1

FSL_GetActiveBootCluster

【概　要】
ブート・フラグ情報の取得

【書　式】
＜C言語＞

```
fsl_u08 FSL_GetActiveBootCluster(fsl_u08 *destination_pu08)
```

＜アセンブラ＞

```
CALL !_FSL_GetActiveBootCluster または CALL !!_FSL_GetActiveBootCluster
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】
この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】
ブート･クラスタ･フラグはデータ・バッファに格納されます。

【呼び出し後のレジスタ状態】
戻り値　　：C
破壊レジスタ：RB3

【引　数】

引　数	説　明
destination_pu08	**データ・バッファ** - **専用のデータ格納バッファを確保してください**

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u08 *destination_pu08	AX
NEC Large model	fsl_u08 *destination_pu08	AX(0-15), C(16-23)

【戻り値】

状　態	説　明
0x00(FSL_OK)	**正常終了**

ブート・フラグ情報

destination_pu08	説　明
0x00	ブート領域の入れ替えを行っていない
0x01	ブート領域の入れ替えを行っている

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_GetBlockEndAddr

【概　要】

指定したブロックの最終アドレスの取得

【書　式】

＜C言語＞

```
fsl_u08 FSL_GetBlockEndAddr(fsl_u32* destination_pu32, fsl_u16 block_u16)
```

＜アセンブラ＞

```
CALL !_FSL_GetBlockEndAddr または CALL !!_FSL_GetBlockEndAddr
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数数を実行してください。**

【機　能】

指定したブロックの最終アドレスがデータ・バッファに格納されます。

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：AX**（引数），**ES，RB3

【引　数】

引　数	説　明
destination_pu32	**データ・バッファ** **- 専用のデータ格納バッファを確保してください**
block_u16	**ブロック番号**

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u32 *destination_pu32 fsl_u16 block_u16	&destination_pu32: AX block_u16: **スタック**
NEC Large model	fsl_u32 *destination_pu32 fsl_u16 block_u16	&destination_pu32: AX(0-15), C(16-23) block_u16: **スタック**

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x05(FSL_ ERR_PARAMETER)	パラメータ・エラー - ブロック番号の指定が指定可能範囲外

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_GetFlashShieldWindow

【概　要】

フラッシュ・シールド・ウインドウの開始ブロック番号と終了ブロック番号の取得

【書　式】

＜C言語＞

```
fsl_u08 FSL_GetFlashShieldWindow(fsl_u16* start_block_pu16,
                                 fsl_u16* end_block_pu16);
```

＜アセンブラ＞

```
CALL !_FSL_GetFlashShieldWindow または CALL !!_FSL_GetFlashShieldWindow
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

フラッシュ･シールド・ウインドウの開始ブロックは start_block_pu16 **に，終了ブロックは** end_block_pu16 **に格納されます。**

【呼び出し後のレジスタ状態】

戻り値　：C

破壊レジスタ：RB3

【引　数】

引　数	説　明
start_block_pu16	FSW**の開始ブロック格納バッファ** - **専用のデータ格納バッファを確保してください**
end_block_pu16	FSW**の終了ブロック格納バッファ** - **専用のデータ格納バッファを確保してください**

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u16* start_block_pu16 fsl_u16* end_block_pu16	&start_block_pu16: AX &end_block_pu16: **スタック**
NEC Large model	fsl_u16* start_block_pu16 fsl_u16* end_block_pu16	&start_block_pu16: AX(0-15), C(16-23) &end_block_pu16: **スタック**

【戻り値】

状　態	説　明
0x00(FSL_OK)	**正常終了**

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_SetXXX

セキュリティ・フラグの設定はビット単位で行います。したがって，1度に変更できるセキュリティ・ビットは1つのみです。

なお，セキュリティ設定についての詳細は，対象デバイスのマニュアルを参照してください。

注意　フラッシュ・セルフ・プログラミングではすでに処理が禁止に設定されているフラグを許可に変更することはできません。

FSL_InvertBootFlag

【概　要】

ブート・フラグの現在値を反転。

【書　式】

＜C言語＞

```
fsl_u08 FSL_InvertBootFlag(void)
```

＜アセンブラ＞

```
CALL !_FSL_InvertBootFlag または CALL !!_FSL_InvertBootFlag
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

ブート・フラグの現在値を反転させます。リセット後，ブート・クラスタはブート・フラグの設定に沿った状態となります。

注意　ブート・クラスタの入れ替えは行いません。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - すでに禁止が設定されているフラグを許可しようとした - ブート領域書き換え禁止状態で，ブート領域入れ替えフラグを変更しようとした
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した（FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した（FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_SwapBootCluster

【概　要】

ブート・スワップを実行し，スワップ後の領域のリセット・ベクタに登録されているアドレスへ移動

【書　式】

＜C言語＞

```
fsl_u08 FSL_SwapBootCluster(void)
```

＜アセンブラ＞

```
CALL !_FSL_SwapBootCluster または CALL !!_FSL_SwapBootCluster
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

この関数の実行前に必ず FSL_Open 関数と，FSL_Init 関数（または FSL_Init_cont 関数），FSL_ModeCheck 関数を実行してください。

【機　能】

関数実行直後にブート・クラスタの入れ替えを実行し，FLMD0 をロウ・レベルに設定したあと，入れ替えられた領域のリセット・ベクタに登録されているアドレスへ移動します。

注意 1. この関数が正常に実行された場合は，入れ替えられたブート・クラスタのリセット・ベクタに登録されているアドレスに移動するため，この関数以降の処理は実行されません。
2. この関数ではブート・フラグの反転は行いません。
リセットを実行した場合，ブート・クラスタはブート・フラグの設定に沿った状態となります。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：ES, CS, RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - ブート領域書き換え禁止状態で，ブート・スワップを実行しようとした

備考1. 正常終了の場合は，戻り値は確認できません。
2. アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_ForceReset

【概　要】

使用中のマイクロコントローラをリセット

【書　式】

＜C言語＞

```
void FSL_ForceReset(void)
```

＜アセンブラ＞

```
CALL !_FSL_ForceReset または CALL !!_FSL_ForceReset
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

この関数の実行前に必ず FSL_Open 関数と，FSL_Init 関数（または FSL_Init_cont 関数），FSL_ModeCheck 関数を実行してください。

【機　能】

関数実行直後に使用中のマイクロコントローラをリセットします。

注意 1. 使用中のマイクロコントローラをリセットしますので，この関数以降の処理は実行されません。
2. MINICUBE®または IECUBE®を使用中に本関数を実行した場合，Break が発生し，処理がとまります。Break 発生以降は正常に動作できませんので，手動でリセットを実行してください。

【呼び出し後のレジスタ状態】

レジスタは破壊されません。

【引　数】

なし

【戻り値】

なし

FSL_SwapActiveBootCluster

【概　要】

ブート・フラグの現在値を反転し，ブート・スワップの実行

【書　式】

＜C言語＞

```
fsl_u08 FSL_SwapActiveBootCluster(void)
```

＜アセンブラ＞

```
CALL !_FSL_SwapActiveBootCluster または CALL !!_FSL_SwapActiveBootCluster
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

この関数を実行すると，ブート・フラグの現在値が反転し，ブート・クラスタの入れ替えも実行されます。

注意　スワップ実行前に，必ずオプション・バイトなどのスワップ後の動作に必要な設定情報を，スワップ先の領域に書き込んでおいてください。必要な情報がない場合，本関数を実行後に正常に動作できなくなることがあります。
また，本関数を実行する場合は，フラッシュ・セルフ・プログラミング・ライブラリ，および呼び出し元のユーザ・プログラムを，ブート・クラスタ内に配置しないでください。関数実行時にブート・クラスタの入れ替えも行われるため，プログラムが正常に動作できなくなる恐れがあります。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - すでに禁止が設定されているフラグを許可しようとした - ブート領域書き換え禁止状態で，ブート領域入れ替えフラグを変更しようとした
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，C レジスタに格納されます。

FSL_SetChipEraseProtectFlag

【概　要】

チップ消去許可フラグの設定

【書　式】

＜C言語＞

```
fsl_u08 FSL_SetChipEraseProtectFlag(void)
```

＜アセンブラ＞

```
CALL !_FSL_SetChipEraseProtectFlag または CALL !!_FSL_SetChipEraseProtectFlag
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

この関数の実行前に必ず FSL_Open 関数と，FSL_Init 関数（または FSL_Init_cont 関数），FSL_ModeCheck 関数を実行してください。

【機　能】

チップ消去許可フラグを設定します。禁止に設定した場合，プログラマによるデバイスのチップ消去が実行できなくなるため，すべての禁止設定は解除できなくなります。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - すでに禁止が設定されているフラグを許可しようとした
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，C レジスタに格納されます。

FSL_SetBlockEraseProtectFlag

【概　要】

ブロック消去許可フラグを設定

【書　式】

＜C言語＞

```
fsl_u08 FSL_SetBlockEraseProtectFlag(void)
```

＜アセンブラ＞

```
CALL !_FSL_SetBlockEraseProtectFlag または CALL !!_FSL_SetBlockEraseProtectFlag
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

この関数の実行前に必ず FSL_Open 関数と，FSL_Init 関数（または FSL_Init_cont 関数），FSL_ModeCheck 関数を実行してください。

【機　能】

ブロック消去許可フラグを設定します。禁止に設定した場合，プログラマによるデバイスへのブロック消去ができなくなります。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - すでに禁止が設定されているフラグを許可しようとした
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_SetWriteProtectFlag

【概　要】

書き込み許可フラグを設定

【書　式】

＜C言語＞

```
fsl_u08 FSL_SetWriteProtectFlag(void)
```

＜アセンブラ＞

```
CALL !_FSL_SetWriteProtectFlag または CALL !!_FSL_SetWriteProtectFlag
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFHに配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。

【事前設定】

この関数の実行前に必ず FSL_Open 関数と，FSL_Init 関数（または FSL_Init_cont 関数），FSL_ModeCheck 関数を実行してください。

【機　能】

書き込み許可フラグを設定します。禁止に設定した場合，プログラマによるデバイスへの書き込みができません。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - すでに禁止が設定されているフラグを許可しようとした
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0 端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_SetBootClusterProtectFlag

【概　要】

ブート領域書き換え許可フラグの設定

【書　式】

＜C言語＞

```
fsl_u08 FSL_SetBootClusterProtectFlag(void)
```

＜アセンブラ＞

```
CALL !_FSL_SetBootClusterProtectFlag または CALL !!_FSL_SetBootClusterProtectFlag
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

ブート領域書き換え許可フラグを禁止に設定します。禁止に設定した場合，ブート・クラスタのスワップ，消去，書き込みはできません。

また，プログラマによるチップ消去も実行することができません。

【呼び出し後のレジスタ状態】

戻り値　　：C
破壊レジスタ：RB3

【引　数】

なし

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x10(FSL_ERR_PROTECTION)	プロテクト・エラー - すでに禁止が設定されているフラグを許可しようとした - ブート領域書き換え禁止状態で，ブート領域入れ替えフラグを変更しようとした
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_SetFlashShieldWindow

【概　要】

フラッシュ・シールド・ウインドウの設定

【書　式】

＜C言語＞

```
fsl_u08 FSL_SetFlashShieldWindow (fsl_u16 start_block_u16,
                                  fsl_u16 end_block_u16)
```

＜アセンブラ＞

```
CALL !_FSL_SetFlashShieldWindow または CALL !!_FSL_SetFlashShieldWindow
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

この関数の実行前に必ず FSL_Open **関数と，**FSL_Init **関数（または** FSL_Init_cont **関数），**FSL_ModeCheck **関数を実行してください。**

【機　能】

フラッシュ・シールド・ウインドウを設定します。

備考　フラッシュ・セルフ・プログラミング時のセキュリティ機能として，フラッシュ・シールド・ウインドウ機能を搭載しています。

フラッシュ・セルフ・プログラミング**時では，ウインドウとして指定した範囲内のフラッシュ・メモリは，書き込みおよび消去が可能となり，指定範囲以外のフラッシュ・メモリは，書き込みおよび消去が禁止となります。ただし，オンボード／オフボード・プログラミング時では，ウインドウとして指定した範囲以外のフラッシュ・メモリも，書き込みおよび消去が可能となります。また，ウインドウとして指定した範囲とブート・クラスタ**0**の書き換え禁止領域が重なる場合は，ブート・クラスタ**0**の書き換え禁止が優先されます。**

【呼び出し後のレジスタ状態】

戻り値　　：C

破壊レジスタ：ES，RB3

【引　数】

引　数	説　明
start_block_u16	フラッシュ・シールド・ウインドウの開始ブロック
end_block_u16	フラッシュ・シールド・ウインドウの終了ブロック

	引数型／レジスタ	
	C言語	アセンブリ言語
NEC Small and medium model	fsl_u16 start_block_u16 fsl_u16 end_block_u16	start_block_u16: AX end_block_u16: スタック
NEC Large model	fsl_u16 start_block_u16 fsl_u16 end_block_u16	start_block_u16: AX end_block_u16: スタック

【戻り値】

状　態	説　明
0x00(FSL_OK)	正常終了
0x05(FSL_ERR_PARAMETER)	パラメータ･エラー - ブロック番号の指定が設定可能範囲外
0x1A(FSL_ERR_ERASE)	消去エラー - 消去処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1B(FSL_ERR_IVERIFY)	内部ベリファイ・エラー - ベリファイ（内部ベリファイ）処理中にエラーが発生した
0x1C(FSL_ERR_WRITE)	書き込みエラー - 書き込み処理中にエラーが発生した （FLMD0端子がハイ・レベルか確認してください）
0x1F(FSL_ERR_INTERRUPTION)	割り込み発生に伴う処理の中断 （ベクタ割り込み処理内でFSL_SetInterruptMode関数を呼び出した場合のみ）

備考　アセンブリ言語の戻り値は，Cレジスタに格納されます。

FSL_SetInterruptMode

【概　要】

割り込み後の動作の中断

【書　式】

＜C言語＞

```
void FSL_SetInterruptMode( void );
```

＜アセンブラ＞

```
CALL !_FSL_SetInterruptMode または CALL !!_FSL_SetInterruptMode
```

備考　フラッシュ・セルフ・プログラミング・ライブラリを00000H-0FFFFH**に配置する場合は“！”，それ以外の場合は“！！”で呼び出してください。**

【事前設定】

なし

【機　能】

フラッシュ・セルフ・プログラミング中にベクタ割り込みが発生した場合，ベクタ割り込み処理中に本関数を実行すると，ベクタ割り込み終了後，ユーザ・プログラムに復帰します。実行しない場合は，割り込みで中断していた関数を継続実行します。

【呼び出し後のレジスタ状態】

レジスタは破壊されません。

【引　数】

なし

【戻り値】

なし

6.4 フラッシュ・セルフ・プログラミング・ライブラリに関する注意事項

(1) フラッシュ・セルフ・プログラミング・ライブラリは，78K0RマイクロコントローラのCPU，レジスタ，RAMを使用してフラッシュ・メモリの書き換えを行うため，フラッシュ・セルフ・プログラミング・ライブラリの処理の実行中は，ユーザ・プログラムが実行されません。

(2) FLMD0端子がロウ・レベルの場合，フラッシュ・メモリの消去，書き込みなどの操作ができません。したがって，フラッシュ・セルフ・プログラミングでフラッシュ・メモリの操作を行う場合は，FSL_ModeCheck関数を呼び出し，FLMD0端子の状態がハイ・レベルであることを確認する必要があります。また，書き換えの一連の処理が終了(FSL_Close実行)するまでハイ・レベルを一定に保つ必要があります。

付録A　総合索引

〔メ　　モ〕

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/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2010.06.04	—	初版発行

CMOS**デバイスの一般的注意事項**

(1) **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**デバイス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。入力レベルが固定な場合はもちろん，**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等が入らないようご使用ください。**

(2) **未使用入力の処理：**CMOS**デバイスの未使用端子の入力レベルは固定してください。未使用端子入力については，**CMOS**デバイスの入力に何も接続しない状態で動作させるのではなく，プルアップかプルダウンによって入力レベルを固定してください。また，未使用の入出力端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介して**V_{DD}**または**GND**に接続することが有効です。資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。**

(3) **静電気対策：**MOS**デバイス取り扱いの際は静電気防止を心がけてください。**MOS**デバイス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，または導電性の緩衝材，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，**MOS**デバイス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。**

(4) **初期化以前の状態　電源投入時，**MOS**デバイスの初期状態は不定です。電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。**

(5) **電源投入切断順序　内部動作および外部インタフェースで異なる電源を使用するデバイスの場合，原則として内部電源を投入した後に外部電源を投入してください。切断の際には，原則として外部電源を切断した後に内部電源を切断してください。逆の電源投入切断順により，内部素子に過電圧が印加され，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「電源投入切断シーケンス」についての記載のある製品については，その内容を守ってください。**

(6) **電源**OFF**時における入力信号　当該デバイスの電源が**OFF**状態の時に，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**OFF**時における入力信号」についての記載のある製品については，その内容を守ってください。**

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、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に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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おりますが、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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。お客様が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. 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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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その総株主の議決権の過半数を直接または間接に保有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。

RENESAS

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・電話番号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 エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■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：http://japan.renesas.com/inquiry

Colophon 1.0