



本ドキュメントはCypress (サイプレス) 製品に関する情報が記載されております。本ドキュメントには、仕様の開発元企業として「スパンション」, 「Spansion」, 「富士通」または「Fujitsu」の名が記載されておりますが、これらの製品は Cypress が新規および既存のお客様に引き続き提供してまいります。

商品仕様の継続性について

Cypress 製品として提供することに伴う商品仕様としての変更はなく、ドキュメントとしての変更もありません。また本ページのお知らせは、変更情報として追記いたしません。本ドキュメントに変更情報が記載されている場合、それは本お知らせを除いた前版からの変更点です。なお、今後改訂は必要に応じて行われますが、その際の変更内容は改訂後のドキュメントに記載いたします。

オーダ型格および品名について

Cypress は既存のオーダ型格および品名を引き続きサポートいたします。これらの製品をご注文の際は、このドキュメントに記載されているオーダ型格および品名をご使用ください。

詳しいお問い合わせ先

Cypress 製品およびそのソリューションの詳細につきましては、お近くの営業所へお問い合わせください。

サイプレスについて

サイプレス (銘柄コード: CY) は、車載や産業機器、ネットワーキング プラットフォームから高機能民生機器およびモバイル機器まで、今日の最先端組み込みシステム向けに高性能で高品質のソリューションを提供します。NOR フラッシュ メモリや F-RAMTM、SRAM、TraveoTM マイクロコントローラー、業界唯一の PSoC[®] プログラマブル システムオンチップ ソリューション、アナログおよび PMIC Power Management IC、CapSense[®] 静電容量タッチセンシング コントローラー、Wireless BLE Bluetooth[®] Low-Energy、USB コネクティビティ ソリューションなど、幅広い差別化製品ポートフォリオを、一貫した革新性と業界最高クラスの技術サポート、比類のないシステム バリューとともにグローバルに提供します。

FR Family
32-BIT MICROCONTROLLER
MB91F353A/355A

ユーザプログラムによるフラッシュメモリへの データ書き込み/消去方法

注意事項

- 本資料の記載内容は、予告なしに変更することがありますので、ご用命の際は営業部門にご確認ください。
- 本資料に記載された動作概要や応用回路例は、半導体デバイスの標準的な動作や使い方を示したもので、実際に使用する機器での動作を保証するものではありません。従いまして、これらを使用するにあたってはお客様の責任において機器の設計を行ってください。これらの使用に起因する損害などについては、当社はその責任を負いません。
- 本資料に記載された動作概要・回路図を含む技術情報は、当社もしくは第三者の特許権、著作権等の知的財産権やその他の権利の使用権または実施権の許諾を意味するものではありません。また、これらの使用について、第三者の知的財産権やその他の権利の実施ができることの保証を行うものではありません。したがって、これらの使用に起因する第三者の知的財産権やその他の権利の侵害について、当社はその責任を負いません。
- 本資料に記載された製品は、通常の産業用、一般事務用、パーソナル用、家庭用などの一般的用途に使用されることを意図して設計・製造されています。極めて高度な安全性が要求され、仮に当該安全性が確保されない場合、社会的に重大な影響を与えかつ直接生命・身体に対する重大な危険性を伴う用途（原子力施設における核反応制御、航空機自動飛行制御、航空交通管制、大量輸送システムにおける運行制御、生命維持のための医療機器、兵器システムにおけるミサイル発射制御をいう）、ならびに極めて高い信頼性が要求される用途（海底中継器、宇宙衛星をいう）に使用されるよう設計・製造されたものではありません。したがって、これらの用途にご使用をお考えのお客様は、必ず事前に営業部門までご相談ください。ご相談なく使用されたことにより発生した損害などについては、責任を負いかねますのでご了承ください。
- 半導体デバイスはある確率で故障が発生します。当社半導体デバイスが故障しても、結果的に人身事故、火災事故、社会的な損害を生じさせないように、お客様は、装置の冗長設計、延焼対策設計、過電流防止対策設計、誤動作防止設計などの安全設計をお願いします。
- 本資料に記載された製品を輸出または提供する場合は、外国為替及び外国貿易法および米国輸出管理関連法規等の規制をご確認の上、必要な手続きをおとりください。
- 本書に記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

Copyright© 2008 FUJITSU MICROELECTRONICS LIMITED all rights reserved

改版履歴

版数	日付	内容
1.0 版		新規作成

目次

注意事項	1
改版履歴	2
目次	3
0 はじめに	4
1 データ書き込み手順	5
1.1 概要	5
1.2 プログラム動作手順	5
2 フラッシュメモリ	7
3.1 フラッシュメモリの概要	7
3.2 レジスタ	8
3.2.1 FLCRレジスタ	8
3.2.2 FLCRレジスタ	8
3.3 自動アルゴリズム	9
3.4 ハードウェアシーケンスフラグ	10
3.5 自動アルゴリズム動作の基本シーケンス	11
4 プログラミング例	12
4.1 概要	12
4.2 main.c	14
4.3 FLASH_WRITE.c	15
4.4 FLASH_WRITE.cの個別設定	16
4.5 リンカの設定	17
4.6 STARTUP.asm	18
4.7 RAM実行プログラム内でのライブラリ呼び出しの方法について	19

0 はじめに

このアプリケーションノートは、MB91F353A/355A における、ユーザプログラムによるフラッシュメモリへのデータ書き込み/消去方法について記述します。

1 データ書き込み手順

1.1 概要

MB91F353A/355Aに搭載されているフラッシュメモリは、フラッシュメモリ上でのプログラム実行中に、フラッシュメモリの消去、書き込みはできません。¹これをシングルオペレーション型フラッシュメモリと呼びます。フラッシュメモリへのデータ書き込みや自己プログラム書き換えを行う場合は、書込み/書き換えプログラムをRAMへ配置し、RAM上でそのプログラムを実行する必要があります。

本書はフラッシュメモリ上のデータ(以下ユーザデータとします)書き換えを行う場合のプログラム記述、および Softune Workbench での各種設定について説明します。

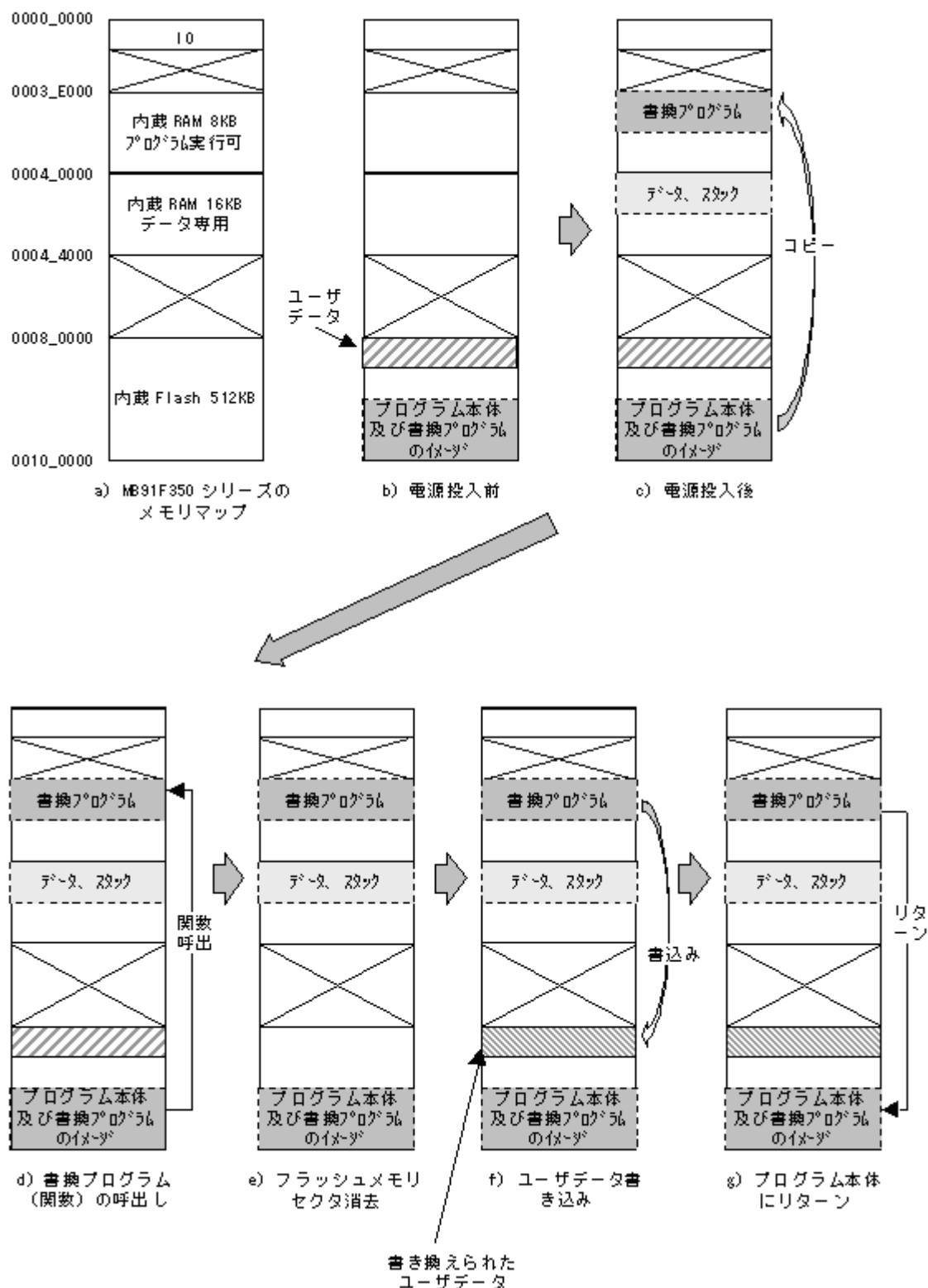
なお、エパチップ(MB91V350)とICEを使った評価環境では、評価プログラム用メモリとして SRAM を使用します。このためユーザデータ書き換えの評価は行えません。MB91F353A/355A を用いた評価環境でご確認ください。

1.2 プログラム動作手順

次ページの図に沿って説明します。

- MB91F353A/355A のシングルチップモード時のメモリマップです。2 種類の RAM が搭載されていますが、プログラムを展開して動作が可能なのは 0x3E000-0x3FFFF の RAM 領域です。
- あらかじめプログラムをフラッシュメモリに書き込んでおきます。このとき、RAM で実行されるフラッシュメモリ書き換え用プログラムもフラッシュメモリに書き込みます。
- マイコンの電源をオンすると、ユーザプログラムにより RAM 領域中にデータやスタック領域を割当て、書き換えプログラムコードを RAM 領域にコピーします。
- フラッシュメモリ内にある本体プログラムから、書き換えプログラムの関数を呼び出します。
- 書き換えプログラムを実行することでフラッシュメモリへコマンドを発行し、ユーザデータを書き込むセクタに対しセクタ消去を行います。
- 新たなユーザデータをフラッシュメモリに書き込みます。
- フラッシュメモリへデータが書き込まれたら、関数リターンでフラッシュメモリ上のプログラム本体に戻ります。

¹ これに対し、同一フラッシュメモリ上にプログラム用バンクとデータ用バンクを持ち、プログラム実行中にデータの書込みと消去を行えるものを、デュアルオペレーション型フラッシュと呼びます。



2 フラッシュメモリ

3.1 フラッシュメモリの概要

MB91F353A/355A には 256kByte,32bit 幅のフラッシュメモリが 2 個(ROM1、ROM2)搭載されています。下図にフラッシュメモリのメモリマップを示します。1 つのボックスが 1 セクタで、SAA 番号がセクタ番号になります。

連続したアドレスでは ROM1 と ROM2 が 4 バイト毎に交互に使用されます。フラッシュメモリではデータ書き込み前にセクタ単位でのデータ消去を行いますので、ユーザデータが書き込まれるセクタと、プログラムが入るセクタは別になるように配置する必要があります。

FLCR レジスタの WE ビットを WE = 1 に設定することにより、フラッシュメモリへの書き換え・消去が有効になります。このとき、ハードウェアの仕様上 FLASH メモリからのデータ読み出しは 16/8 ビットアクセスのみ可能で、32 ビットアクセスはできません。

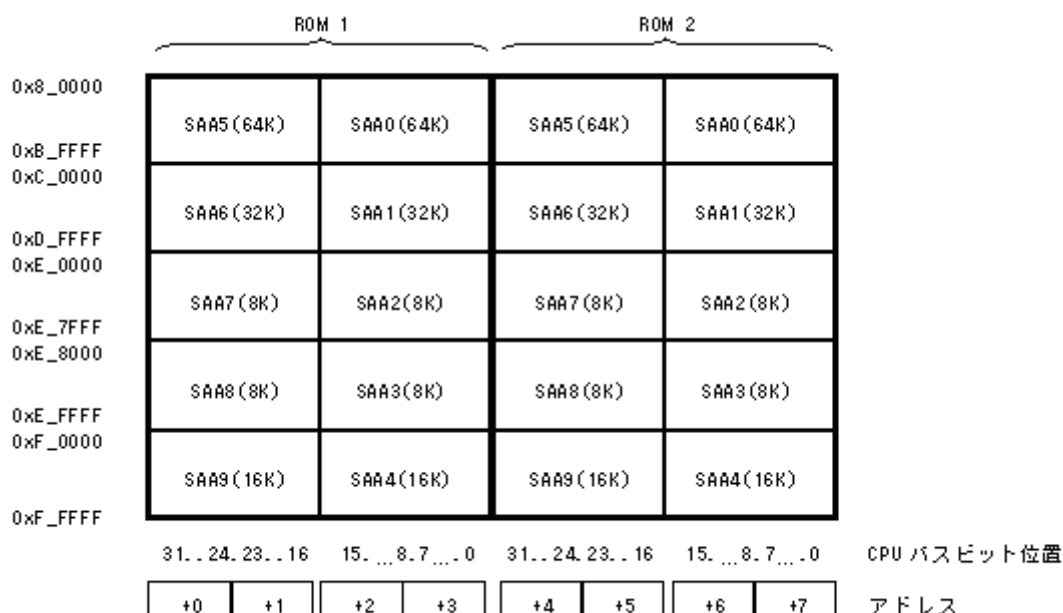


図1 FLASH メモリのセクタマップ(CPU モード*)

*CPU モードとは、CPU のプログラム/データ格納用メモリとして機能するモードです。

FLASH モードとは、CPU 内の FLASH メモリが単体メモリとして見えるモードで、パラレル書き込み時に機能するモードです。

3.2 レジスタ

フラッシュメモリ操作のレジスタとしては FLCR と FLWC があります。これらのレジスタはフラッシュメモリへの制御信号線の操作や、インタフェース回路の設定を行います。詳細に関しては MB91350A シリーズハードウェアマニュアルをご参照ください。

3.2.1 FLCR レジスタ

フラッシュメモリの制御および、ステータスを示すレジスタです。本レジスタはリードモディファイライト系命令ではアクセスしないでください。

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x7_0000 FLCR	-	-	-	RDYEG	RDY	-	WE	-
	R/W	R/W	R/W	R	R	R/W	R/W	R/W
	(0)	(1)	(1)	(0)	(X)	(0)	(0)	(0)

[bit1] WE (Write Enable)

本ビットが"0"の間は、フラッシュメモリへのデータおよびコマンドの書込みはすべて無効になります。また、フラッシュメモリからのデータ読み出しが倍速化します(32/16/8 ビットアクセスが可能)。

本ビットが"1"の間は、フラッシュメモリへのデータおよびコマンドの書込みが有効となり、自動アルゴリズムの起動が可能となります。ただしフラッシュメモリからのデータ読み出しは低速になります(16/8 ビットアクセスのみ可能)。

本ビットの書き換えは、必ず RDY ビットにより自動アルゴリズム(書込み/消去)が停止している事を確認してから行ってください。RDY ビットが"0"の間は、本ビットの値を書き換えることができません。

3.2.2 FLWC レジスタ

フラッシュメモリアクセスのウェイトサイクルを制御するレジスタです。

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x7_0004 FLWC	-	-	FAC1	FAC0	-	WTC2	WTC1	WTC0
	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	(0)	(0)	(0)	(1)	(0)	(0)	(1)	(1)

[bit5-4] FAC1,0

フラッシュ制御の内部パルス生成を制御するための設定ビットです。MB91F353/355A では FAC1 ビットに"1"、FAC0 ビットに"0"を設定してください。

[bit2-0] WTC2,1,0 : Wait Cycle ビット

フラッシュメモリのウエイトサイクルを制御します。書込み/消去時はウエイト数の値を 4 以上にする必要があります。

WTC2	WTC1	WTC0	ウエイト数	WE ビット "0" 時	WE ビット "1" 時	
0	0	0	-	設定禁止	設定禁止	
0	0	1	1	設定禁止	設定禁止	
0	1	0	2	動作可	設定禁止	
0	1	1	3	動作可	設定禁止	初期値
1	0	0	4	動作可	動作可	
1	0	1	5	動作可	動作可	
1	1	0	6	動作可	動作可	
1	1	1	7	動作可	動作可	

3.3 自動アルゴリズム

WE=1 で書き込み/消去が可能な状態にした後、実際に書き込み/消去を行う時にはコマンドシーケンスでコマンドを発行し、自動アルゴリズムを動作させます。

コマンドシーケンスは下記の表のように、特定アドレスに特定データを順に書き込んでいく動作になります。

コマンド シーケンス	ROM	1 st ライト		2 nd ライト		3 rd ライト		4 th ライト		5 th ライト		6 th ライト	
		アド レス	デー タ	アド レス	デー タ	アド レス	デー タ	アド レス	デー タ	アド レス	デー タ	アド レス	デー タ
書き込み	ROM1	AAAAA	00AA	D5552	0055	AAAAA	00A0	(PA)	(PD)				
	ROM2	AAAAE	00AA	D5556	0055	AAAAE	00A0	(PA)	(PD)				
チップ消去	ROM1	AAAAA	00AA	D5552	0055	AAAAA	0080	AAAAA	00AA	D5552	0055	AAAAA	0010
	ROM2	AAAAE	00AA	D5556	0055	AAAAE	0080	AAAAE	00AA	D5556	0055	AAAAE	0010
セクタ消去	ROM1	AAAAA	00AA	D5552	0055	AAAAA	0080	AAAAA	00AA	D5552	0055	(SA)	0030
	ROM2	AAAAE	00AA	D5556	0055	AAAAE	0080	AAAAE	00AA	D5556	0055	(SA)	0030

(PA) : 書き込みアドレス

(PD) : 書き込みデータ

(SA) : セクタアドレス (セクタ内の任意のアドレス)

3.4 ハードウェアシーケンスフラグ

自動アルゴリズムが終了するまで、このフラッシュメモリに対する次のコマンド発行やリードは行えません。終了を判定するにはハードウェアシーケンスフラグを用います。ハードウェアシーケンスフラグは自動アルゴリズム実行中のフラッシュメモリ任意アドレスをリードする事により読み出せます。自動アルゴリズムが正常終了すると、リードしているアドレスのデータが読み出されます。

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
自動アルゴリズム 実行中の、フラッ シュメモリ内 任意アドレス	DPOLL	TOGGLE	TLOVER	(不定)	SETIMR	TOGGL2	(不定)	(不定)

[bit7] DPOLL : データポーリングフラグ

書込み時

自動アルゴリズム実行中は、最後に書き込まれたデータの bit7 の反転データを出力します。自動アルゴリズムが終了するとリードアドレスのデータが読み出されます。

チップ消去/セクタ消去時

自動アルゴリズム実行中は 0 を出力します。終了時は 1 を出力します。

[bit6] TOGGLE : トグルビットフラグ

書込み時、チップ消去/セクタ消去時共に、自動アルゴリズム実行中は、リード毎に"1"と"0"が交互に読み出されます。書込み時は自動アルゴリズム終了によりリードアドレスのデータが、消去時は"1"が読み出されます。終了判定ループ内で複数回同一アドレスを読み出し、データが同一になる事で終了が確認できません。

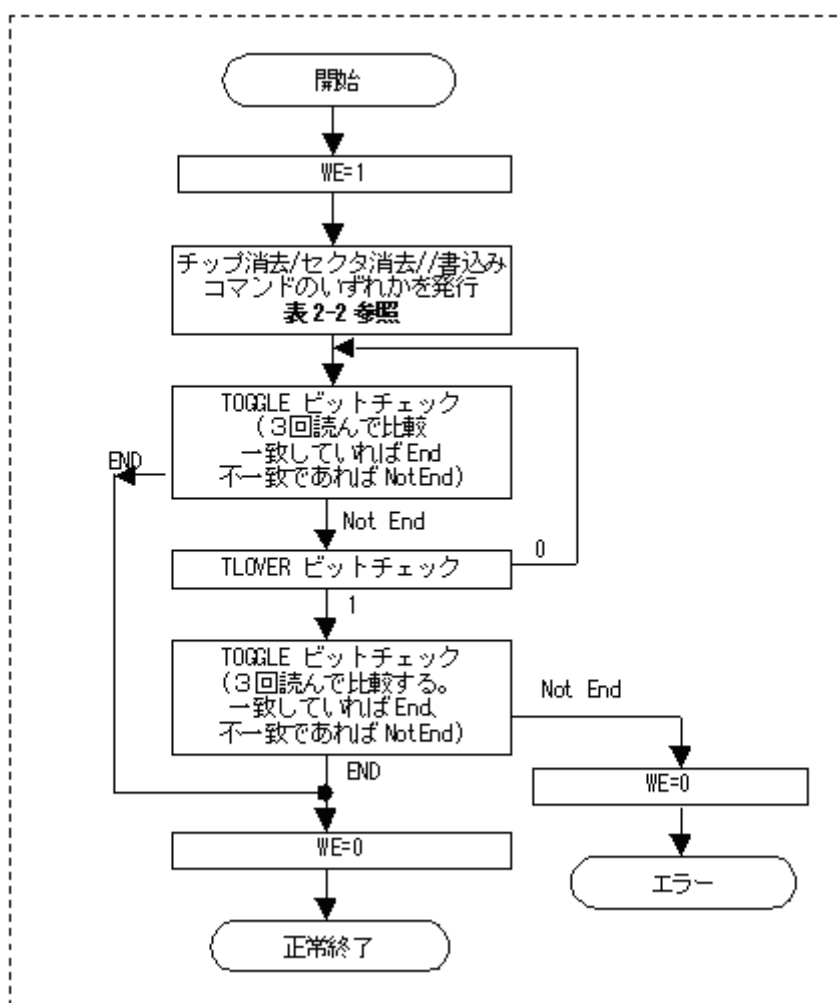
[bit5] TLOVER : タイミングリミット超過フラグ

本フラグは自動アルゴリズムの実行が規定された時間を超えてしまった事を知らせるフラグです。規定時間内であれば"0"、超えている場合は"1"が読み出されます。

3.5 自動アルゴリズム動作の基本シーケンス

書込み/チップ消去/セクタ消去の動作は下記図のようなシーケンスで行います。終了判定には FLWC レジスタの RDY ビットや RDYEG ビットではなく、ハードウェアシーケンスフラグの DPOLL ビットや TOGGLE ビットを用いる事を推奨します。これは、何らかの要因でフラッシュメモリがエラー状態(タイミングリミット超過状態)に陥った場合、RDY ビットや RDYEG ビットでは、本状態を検出する事ができないからです。フラッシュメモリのエラー状態(タイミングリミット超過)検出は、フラッシュメモリのハードウェアシーケンスのみで可能となります。

なお、下記シーケンス図の様に、TOGGLE ビットのチェックを用いる事で、書込みと消去のシーケンスを共通にする事ができます。



大容量のセクタ消去やチップ消去を行う場合、終了までに時間がかかりますが、TOGGLE ビットチェックでの終了判定時に他方の ROM(ROM1 側を行っている場合、ROM2 側)へのコマンド発行を行う事で並行に処理が行われ、処理時間短縮になります。

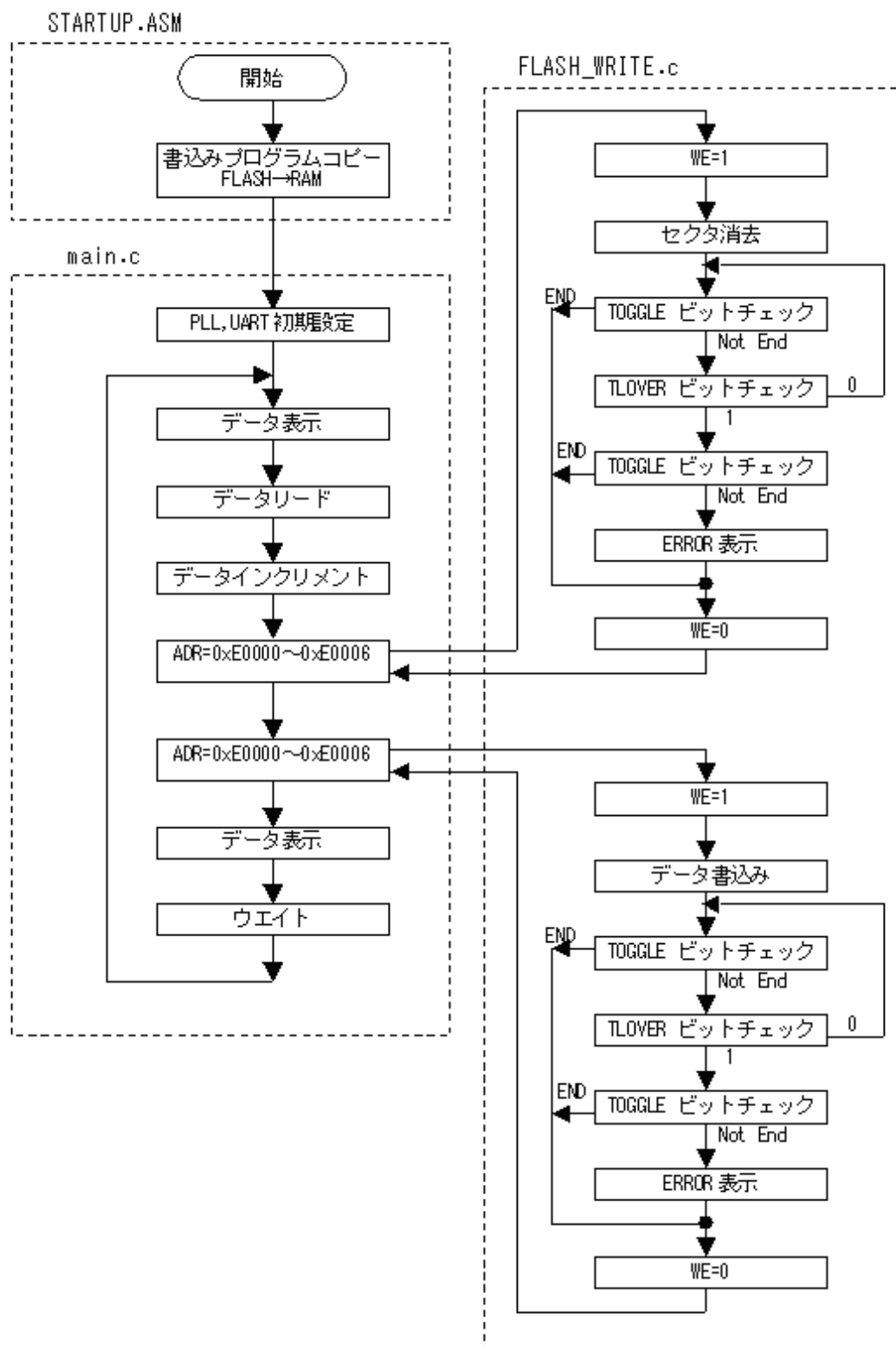
4 プログラミング例

4.1 概要

サンプルプログラムを例にプログラムコードの作成方法を解説します。本サンプルプログラムはフラッシュメモリ上 0xE0000-0xE0007 番地のユーザデータを 0x00 0x11 0x22 という様書き換えて行くものです。プログラムの進捗は UART3 から ASCII データで出力されるので、RS-232C で PC に接続することでモニタ可能です。ボーレートは 38400bps、8bit データ、パリティ無し、1bit ストップビットです。
サンプルプログラムのソースファイル一覧を下記に示します。

ソースファイル名	説明
STARTUP.ASM	スタートアップルーチン。マイコンの初期設定、および RAM 実行プログラムの転送を行います。
main.c	main 関数およびベクタテーブル。
FLASH_WRITE.c	RAM 上に展開され、フラッシュメモリの書き換えを行う関数群。
UART_TX.C	UART の初期設定。
Interrupt_routine.c	割り込み処理関数。

また、サンプルプログラムのフローチャートを次ページに示します。



4.2 main.c

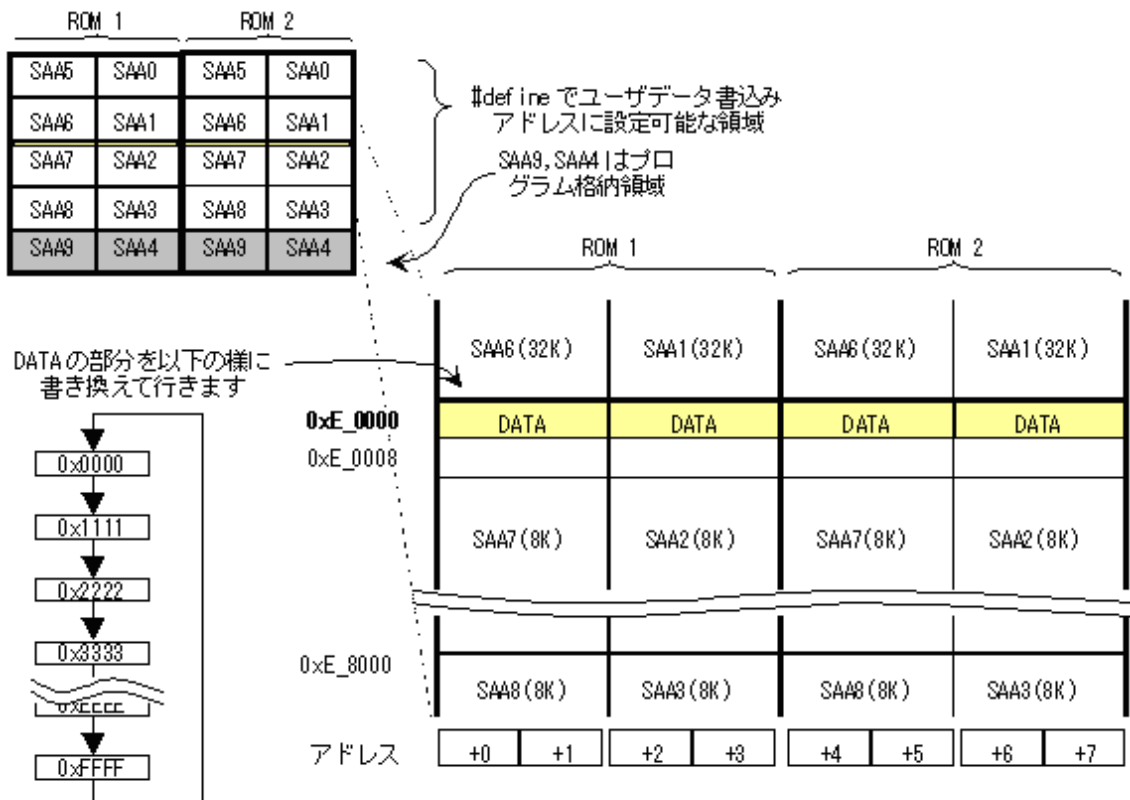
main 関数では、はじめに動作クロックの設定と UART 初期化関数の呼び出しを行った後、無限ループに入ります。ループ内では、

1. フラッシュメモリ内アドレス 0xE0000-0xE0007 のデータを UART に出力
2. アドレス 0xE0000、0xE0002、0xE0004、0xE0006 のデータ (unsigned short) のリード
3. リードした各データに 0x1111 を加算 (0xFFFF であれば 0x0000 にする)
4. 加算したデータを各アドレスに書き戻し (FLASH_WRITE.c 内のフラッシュ消去関数およびフラッシュ書き込み関数を呼び出します)
5. アドレス 0xE0000-0xE0007 のデータを UART に出力
6. ウェイト

を繰り返します。

```
#define DATA_ADR 0xE0000
```

は書き換えを行うユーザデータのスタートアドレスを定義しています。下記図の様に、スタートアドレスを起点に ROM1、ROM2 を 8 バイト分書き換えて行きます。DATA_ADR の値は 0x80000 から 0xEFFF8 の範囲で、8 の倍数で書き換える事が可能です。



4.3 FLASH_WRITE.c

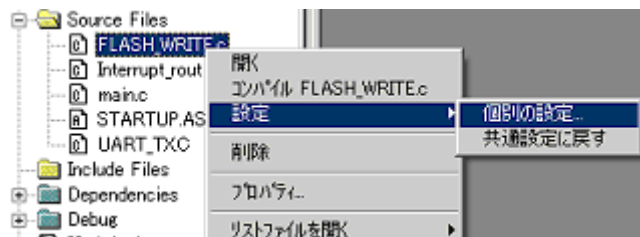
本ソースファイルは RAM 上で実行されるプログラムで、実際にフラッシュメモリに対して行うセクタ消去、データ書き込み動作が記述されています。主な関数の説明を下記表に示します。関数名最後の"1"は ROM1 を操作する関数である事を示します。ROM2 を使用する場合は、関数名最後の文字が"2"になっている関数を使用します。本サンプルプログラムでは進捗を UART でモニタするために"put_char"および"put_hex"関数を用意していますが、FLASH 書き換え・消去中に使用される事があるこれらの関数は、RAM 上で実行される本ソースファイル内にまとめておく必要があります。また、割込みベクタもフラッシュメモリ領域にあるため、割込み機能は使用できません。

関数名	説明
Flash_Sector_Erase1	セクタ消去関数。
Flash_WriteData1	書き換え関数。
Flash_WE_ON	WE レジスタを ON に設定。セクタ消去/書き込み関数の子関数。
Flash_WE_OFF	WE レジスタを OFF に設定。セクタ消去/書き込み関数の子関数。
Seq_Flag_Check	自動シーケンス終了フラグのチェック。セクタ消去/書き込み関数の子関数。

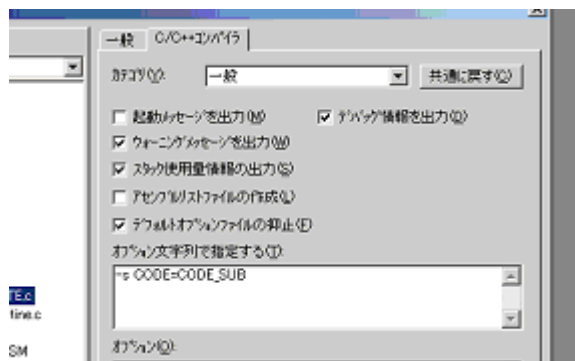
4.4 FLASH_WRITE.c の個別設定

4.3前節で解説したFLASH_WRITE.cに対しては、RAMに配置するためコンパイラの個別設定をする必要があります。なお本設定を行う前にプロジェクト全体に対して行ったコンパイラ設定(インクルードパスの指定など)は本ファイルに対しても有効になりますが、この個別設定後にプロジェクト全体に対してコンパイラ設定を行った場合は、本設定には反映されませんのでご注意ください。

Softune 左側にあるプロジェクトビュー画面の"FLASH_WRITE.c"上で右クリックし、『個別の設定』メニューを表示させます。

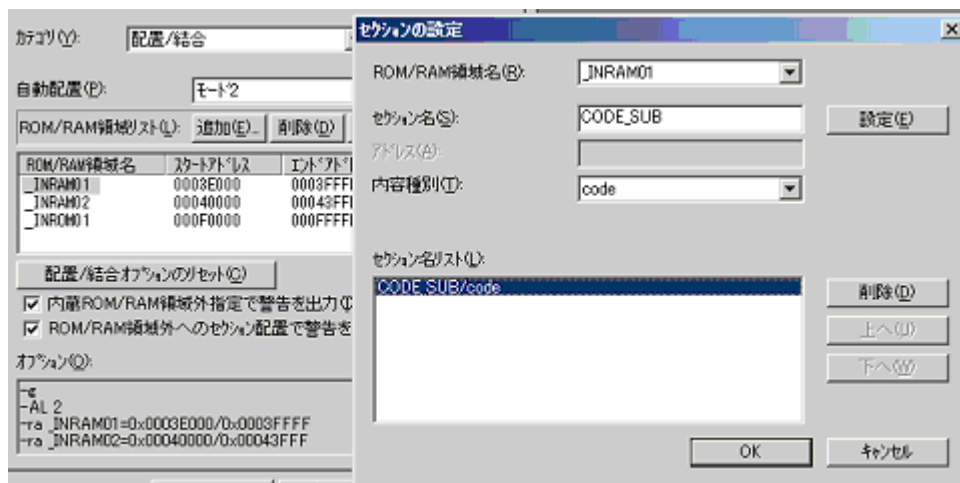


『個別の設定』の『C/C++コンパイラ』タブ内で、『-s CODE=CODE_SUB』という様に、オプションを設定します。これはFLASH_WRITE.cのソースファイルのセクションを任意なセクション名(ここでは CODE_SUB)に変更する設定になります。以降のリンカ設定やSTARTUP.asmの説明では、この任意名のセクションに対する処理に関して記述しています。



4.5 リンカの設定

『プロジェクト設定』の『リンカの設定』『配置/結合』で FLASH_WRITE.c の配置位置を設定します。このコードは 0x3E000-0x3FFFF にある、プログラム実行可能な RAM に配置する必要がありますので、下記図の様に RAM 領域に CODE_SUB のセクションを設定します。この領域は 1.2 章 c) のコピー先の RAM に相当します。



次にコピー元になる ROM 側の設定をします。書き換えプログラムセクション名の前に"@"をつけた名前で設定します。また本アプリケーションマニュアルの命題であるフラッシュメモリ内へのユーザーデータ書き込みの領域を確保するため、ROM のスタートアドレスを 0xF0000 からに変更しています。



4.6 STARTUP.asm

通常の記述以外に、フラッシュメモリ書き換え関数群を RAM にコピーする記述を加えます。


まずコピー元である ROM 側、コピー先である RAM 側のアドレスを示すラベルを使用する事を宣言します。本サンプルプログラムではリンカ設定に於いてセクション名を"CODE_SUB"としていますが、これに"_RAM_"および"_ROM_"と付いたラベルを Softune のリンカが自動的に生成します。リンカで決められた各アドレス値がそれらに入ります。

```

;-----
; external declaration of symbols
;-----
.export  __start
.export  __exit
.import  _main
.import  _exit
.import  __stream_init

.global  _RAM_CODE_SUB
.global  _ROM_CODE_SUB

```




次の記述により"CODE_SUB"セクションのアラインを定義します。

```

;-----
; definition to start address of data, const and code section
;-----
.section DATA,      data, align=4
.section INIT,       data, align=4
.section CONST,      const, align=4
.section CODE_SUB,   code, align=2
.section CODE,       code, align=2

```



IO_FRLR へのレジスタ設定で RAM の使用領域を確保します。このレジスタは命令実行可能な RAM の領域を制限するレジスタで、初期値は 1 (4K バイト)となっています。本サンプルプログラムでは書き換えプログラムを配置する領域となりますので、メモリを大きくするため 2 (8K バイト)を設定しています。

IO_FRLR	説明
0	(設定禁止)
1	アドレス 0x3F000 ~ 0x3FFFF の 4K バイトが使用できます(初期値)
2	アドレス 0x3E000 ~ 0x3FFFF の 8K バイトが使用できます
3	アドレス 0x3C000 ~ 0x3FFFF の 16K バイトが使用できます ただしこの設定は評価用チップ (MB91V350A) に限ります

次の記述でフラッシュメモリ書き換え関数を ROM から RAM へコピーする動作を組み込みます。

```

;-----
;CODE DATA copy
;-----
        ldi        #_RAM_CODE_SUB, r0
        ldi        #_ROM_CODE_SUB, r1
        ldi        #sizeof CODE_SUB, r13
        cmp        #0, r13
        beq        data_cpy_end1
data_cpy1:
        add2       #-1, r13
        ldub       @(r13, r1), r12
        bne:d      data_cpy1
        stb        r12, @(r13, r0)
data_cpy_end1:

```

4.7 RAM 実行プログラム内でのライブラリ呼び出しの方法について

FR コンパイラ fcc911s が準備しているライブラリファイルは次の四つがあります。

lib911.lib : 標準 C ライブラリ

lib911if.lib: シミュレータデバッグ用低水準関数ライブラリ

lib911e.lib : EC++ライブラリ

lib911p.lib : C++ライブラリ(lib911e.lib と内容は同じです)

これらのライブラリファイルを使用する場合、あらかじめライブラリのセクション名が固定で定義されており、ユーザは任意にセクション名を変えることができません。

セクション種別	セクション名
コードセクション	CODE
データセクション	DATA
イニシャライズドセクション	INIT
コンスタントセクション	CONST

リンカの配置/結合を自動配置に設定していると、自動的に ROM 領域に CODE セクションが配置されます。この場合、RAM 実行プログラムからライブラリ関数を呼び出す動作があると、Flash にアクセスできないためプログラム暴走の原因となることが懸念されます。

これらを防ぐために、3.4～3.6 章で紹介したように、使用するライブラリ関数についてもあらかじめ RAM 領域へ配置する必要があります。

サンプルプログラムでは、ライブラリ関数の CODE セクション名を「CODE」と定義し、他の CODE セクション名については「CODE2」と定義し、「CODE」を CODE_SUB と同様に RAM へ配置するように設定をしています。これによってライブラリ関数はプログラム RAM 実行中の呼び出しも可能となります。

(FLASH_WRITE.c プログラム内の FLASH_Sector_Erase1 関数内で sbrk 関数を呼び出しています。これはライブラリ呼び出しの例として記述しています。実際の FLASH 書き込み消去の機能とは関係がありません。)

Main.c や Interrupt_routine.c や UART_TX.c では、3.4 章での設定と同様に、『-s CODE=CODE2』とオプションを設定します。

あるいは、下記のようにプログラムの先頭付近で下記セクション定義を行っても同様の設定となります。

```
#pragma section CODE=CODE2,attr=CODE
```

また、startup.asm では、下記のようにプログラムのスタート位置の直前に下記セクション定義を行っています。

```
.section CODE2, code, align=2
```

「CODE」を ROM 領域から RAM 領域へコピーするための設定は、3.5～3.6 章と同じ方法により実現可能です。

ライブラリ関数が「CODE」だけでなく、ROM に配置される「CONST」領域も使用する場合は、同様の設定でユーザ側のセクション名側を変更してリンクに登録する必要があります。

- 以上 -