Atmel AVR 8ﾋﾞｯﾄ ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ

AVR106 : ﾌﾗｯｼｭ ﾒﾓﾘ読み書き用C関数

応用記述

序説

Atmel® AVR®ﾃﾞﾊﾞｲｽは自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘと呼ばれる機能を持ちます。この機能はﾌﾟﾛｸﾞﾗﾑを実行している間にﾌﾗｯｼｭ ﾒﾓﾘをﾌﾟﾛｸﾞﾗﾐﾝｸﾞすることをAVRﾃﾞﾊﾞｲｽに許します。このような機能はﾌｧｰﾑｳｪｱを自己更新しなければならない、またはﾌﾗｯｼｭ内へﾊﾟﾗﾒｰﾀを保存する応用に対して役立ちます。

この応用記述はﾌﾗｯｼｭ ﾒﾓﾘの入出力用のC関数についての詳細を提供します。

特徴

- ﾌﾗｯｼｭ ﾒﾓﾘ入出力用C関数
  - ﾊﾞｲﾄ読み込み
  - ﾍﾟｰｼﾞ読み込み
  - ﾊﾞｲﾄ書き込み
  - ﾍﾟｰｼﾞ書き込み
- 電力不足での回復任意選択
- 自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘを持つどのﾃﾞﾊﾞｲｽでも使うことができる関数
- ﾊﾟﾗﾒｰﾀ記憶用に応用ﾌﾗｯｼｭ領域をｱｸｾｽするためのﾌﾟﾛｼﾞｪｸﾄ例

Rev. 2575C-03/2016, 2575CJ1-03/2021

目次

1. 動作の理屈

本項はAVRの自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ機能の使用を取り巻くいくつかの基本的な理屈を含みます。自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞに関係する全ての機能のより良い理解については、ﾃﾞﾊﾞｲｽのﾃﾞｰﾀｼｰﾄまたは「AVR109:自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ」を参照してください。

1.1. SPM命令の使用

ﾌﾗｯｼｭ ﾒﾓﾘはSPM(Store Program Memory)命令を使ってﾌﾟﾛｸﾞﾗﾐﾝｸﾞすることができます。自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ機能を含むﾃﾞﾊﾞｲｽではﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘが応用ﾌﾗｯｼｭ領域とﾌﾞｰﾄ ﾌﾗｯｼｭ領域の2つの主な領域に分けられます。

ﾌﾞｰﾄ領域を持つﾃﾞﾊﾞｲｽではSPM命令がﾌﾗｯｼｭ ﾒﾓﾘ全体に書く能力を持ちますが、ﾌﾞｰﾄ領域だけから実行できます。応用領域からのSPM実行は無効です。より小さなﾃﾞﾊﾞｲｽではﾌﾞｰﾄ領域を持たず、SPM命令はﾌﾗｯｼｭ ﾒﾓﾘ全体から実行できます。

ﾌﾞｰﾄ領域へのﾌﾗｯｼｭ書き込みの間はCPUが常に停止されます。けれども、殆どのﾃﾞﾊﾞｲｽは応用領域への書き込み中にﾌﾞｰﾄ領域からｺｰﾄﾞを(読んで)実行できます。応用領域書き込み中に実行されるｺｰﾄﾞが応用領域から読むのを試みないことが重要です。これが起きると、ﾌﾟﾛｸﾞﾗﾑ実行全体が不正になり得ます。

これら2つのﾒﾓﾘ領域の位置と容量はﾃﾞﾊﾞｲｽとﾋｭｰｽﾞ ﾋﾞｯﾄに依存します。いくつかのﾃﾞﾊﾞｲｽはﾌﾗｯｼｭ ﾒﾓﾘ空間全体からSPM命令を実行する能力を持ちます。

1.2. 書き込み手順

ﾌﾗｯｼｭ ﾒﾓﾘはﾍﾟｰｼﾞ単位形式で書かれなければなりません。書き込み関数ははﾌﾗｯｼｭ書き込みに先立って、ﾍﾟｰｼﾞ全体に対するﾃﾞｰﾀをﾍﾟｰｼﾞ一時緩衝部内へ格納することによって実行されます。どのﾌﾗｯｼｭ ｱﾄﾞﾚｽに書くかはZﾚｼﾞｽﾀとRAMPZﾚｼﾞｽﾀの内容によって決定されます。ﾌﾗｯｼｭ ﾍﾟｰｼﾞは一時緩衝部内に格納されたﾃﾞｰﾀでﾌﾟﾛｸﾞﾗﾐﾝｸﾞされ得る前に消去されなければなりません。本応用記述に含まれる関数はﾌﾗｯｼｭ ﾍﾟｰｼﾞ書き込み時に次の手順を使います。

1. ﾍﾟｰｼﾞ一時緩衝部を満たします。
2. ﾌﾗｯｼｭ ﾍﾟｰｼﾞを消去します。
3. ﾌﾗｯｼｭ ﾍﾟｰｼﾞを書きます。

この手順で明らかなように、ﾍﾟｰｼﾞ消去直後にﾘｾｯﾄまたは電力不足が起きる場合にﾃﾞｰﾀ損失の可能性があります。ﾃﾞｰﾀ損失は不揮発性ﾒﾓﾘでの緩衝部を伴う、ｿﾌﾄｳｪｱでの必要な予防処置を講ずることによって避けられます。本応用記述に含まれる書き込み関数は書き込み時に任意選択の緩衝を提供します。これらの関数はﾌｧｰﾑｳｪｱ項で更に説明されます。

書き中の読み(RWW:Read-While-Write)機能を支援するﾃﾞﾊﾞｲｽはﾌﾞｰﾄ ﾛｰﾀﾞ ｺｰﾄﾞに書き込み中の実行を許します。書き込み関数は書き込みが完了されるまで戻りません。

1.3. ｱﾄﾞﾚｽ指定

AVRのﾌﾗｯｼｭ ﾒﾓﾘは16ﾋﾞｯﾄ語に分割されています。これは各々のﾌﾗｯｼｭ ｱﾄﾞﾚｽ位置が2ﾊﾞｲﾄのﾃﾞｰﾀを格納できることを意味します。ATmega128に関しては64K語または128Kﾊﾞｲﾄまでのｱﾄﾞﾚｽのﾌﾗｯｼｭ ﾃﾞｰﾀが可能です。いくつかの場合でﾌﾗｯｼｭ ﾒﾓﾘは語ｱﾄﾞﾚｽ指定を使うことによって、また他の場合ではﾊﾞｲﾄ ｱﾄﾞﾚｽ指定を使うことによって参照され、そしてそれは混同し得ます。本応用記述に含まれる全ての関数はﾊﾞｲﾄ ｱﾄﾞﾚｽ指定を使います。ﾊﾞｲﾄ ｱﾄﾞﾚｽと語ｱﾄﾞﾚｽ間の関連は次のとおりです。

ﾊﾞｲﾄ ｱﾄﾞﾚｽ = 語ｱﾄﾞﾚｽ×2

ﾌﾗｯｼｭ ﾍﾟｰｼﾞはﾍﾟｰｼﾞ内の先頭ﾊﾞｲﾄに対してﾊﾞｲﾄ ｱﾄﾞﾚｽを使うことによってｱﾄﾞﾚｽ指定されます。ﾍﾟｰｼﾞに対するﾍﾟｰｼﾞ番号(順位0,1,2,～)とﾊﾞｲﾄ ｱﾄﾞﾚｽ間の関連は次のとおりです。

ﾊﾞｲﾄ ｱﾄﾞﾚｽ = ﾍﾟｰｼﾞ番号×(ﾊﾞｲﾄでの)ﾍﾟｰｼﾞ容量

ﾊﾞｲﾄ ｱﾄﾞﾚｽ指定の例:

ATmega128のﾌﾗｯｼｭ ﾍﾟｰｼﾞは256ﾊﾞｲﾄ長です。ﾊﾞｲﾄ ｱﾄﾞﾚｽ$0200(512)は次の位置を示します。

- ﾌﾗｯｼｭ ﾊﾞｲﾄ$0200(512)は2ﾍﾟｰｼﾞのﾊﾞｲﾄ0(先頭ﾊﾞｲﾄ)に等しい
- ﾌﾗｯｼｭ ﾍﾟｰｼﾞ2

ATmega128でﾍﾟｰｼﾞをｱﾄﾞﾚｽ指定する時にｱﾄﾞﾚｽの下位ﾊﾞｲﾄは常に0です。語をｱﾄﾞﾚｽ指定する時にｱﾄﾞﾚｽの最下位ﾋﾞｯﾄ(LSB)は常に0です。

1.4. ﾍﾟｰｼﾞ容量

定数PAGESIZEは使うﾃﾞﾊﾞｲｽの(ﾊﾞｲﾄでの)ﾌﾗｯｼｭ ﾍﾟｰｼﾞ容量に等しく定義されなければなりません。

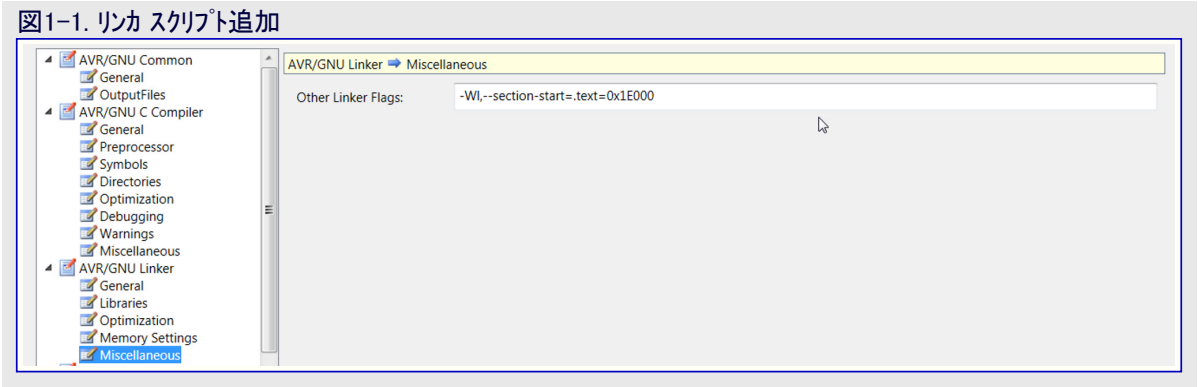
1.5. 書き込み用ﾌﾗｯｼｭ ﾒﾓﾘの定義

関数が書くのを許されるﾒﾓﾘ範囲は定数ADR_LIMIT_LOWとADR_LIMIT_HIGHによって定義されます。書き込み関数はADR_LIMIT_LOW以上で且つADR_LIMIT_HIGH未満のｱﾄﾞﾚｽに書けます。

1.6. ブート領域内にコード全体を配置

フラッシュのブート領域内に応用コード全体を配置するために、次図で示されるようにリンカ任意選択を含めることが必要です。ブート領域の位置と容量は使うデバイスとヒューズ設定で変化します。BOOTRSTヒューズのプログラム(0)はブート領域の先頭にリセット ベクタを移動します。ブート領域に全ての割り込みベクタを移動することも可能です。より多くの情報についてはデバイスのデータシートで「**割り込み**」章を参照してください。

図1-1. リンカ スクリプト追加



リンカその他任意選択(Linker Miscellaneous option)はブート ローダの開始アドレスに対応して定義されるべきです。ATmegaq128のブート ローダ開始語アドレスは4Kブート ローダが0xF000です。リンカ任意選択はバイト アドレス、故に等価なバイト アドレス=0x1E000に更新されるべきです。上記の図で示されるようにリンカ スクリプトを含めることによって、応用コード全体がブート領域内に配置されます。

1.7. ブート領域内に選んだ関数を配置

代わりにフラッシュ メモリの定義済みセグメント内に選択した関数だけを配置することが可能です。実際にブート領域内に配置されなければならないのは書き込みに関する関数だけです。これは以下の例で示されるように、ブート メモリ空間に対応する新しいフラッシュ セグメントを定義し、このセグメント内に望む関数を配置するためにBOOTLOADER SECTION属性を使うことによって行えます。

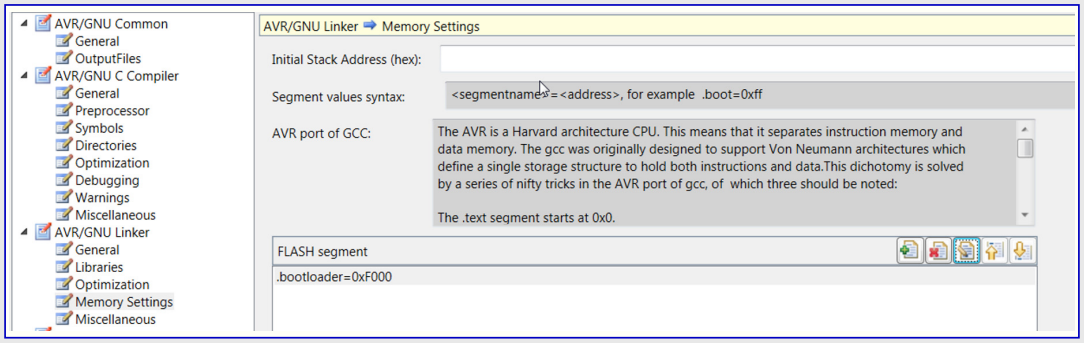
ブート セグメントの定義:

ブート ローダ セグメント定義は下図で示されるように、プロジェクト プロパティ ウィンドウのAVR/GNU Linker option内のMemory Settingsタブで行われるべきです。

“.bootloader=0xF000”

0xF000はATmega128での4Kブート ローダ セグメントの開始アドレスです。

図1-2. ブート セグメント定義



定義したセグメント内へのC関数を配置:

```
#define BOOTLOADER_SECTION __attribute__ ((section(".bootloader")))
void Example_Function ()BOOTLOADER_SECTION;
void Example_Function (){
-------
}
```

上図で示されるようにプロジェクト プロパティ ウィンドウでブート ローダ セグメントを定義して上のCコードを構築することで、それは定義した".bootloader"メモリ セグメント内にExample_function()を配置します。

2. ソフトウェア実装と使用法

ファームウェアはAtmel Studio 7.0.582でのAVR-GCCコンパイラ用に作られています。この関数は他のコンパイラへ移転できるでしょうが、これはavr-gccツールチェーンから様々な関数が使われるためにいくつかの作業が必要かもしれません。自己プログラミング使用時、書き込み用の関数がフラッシュ メモリのブート領域内に配置されることが非常に重要です。ブート領域内にフラッシュ書き込み関数を配置する手順は「ブート領域内に選んだ関数を配置」項下で説明されます。残りの関数はフラッシュの応用領域に配置することができます。ファームウェアに関する他に必要な全ての構成設定はSelf_programming.hファイル内で行われます。

この応用記述で利用可能なzipファイルはATmega128に対してAtmel Studio 7.0を使って作成された例プロジェクトから成ります。この例プロジェクトでは、フラッシュ書き込み関数がフラッシュのブート領域に配置され、残りのコードはフラッシュの応用領域に配置されます。

2.1. フラッシュ回復

_FLASH_RECOVER定数の定義は電力不足の場合のデータ損失を避けるためのフラッシュ回復任意選択を許可します。フラッシュ回復が許可されると、1つのフラッシュ ページが回復緩衝部として扱われます。_FLASH_RECOVERの値はこの目的に使われるフラッシュ ページのアドレスを決めます。このアドレスは応用領域のフラッシュ ページ内のアドレスを指示するバイト アドレスでなければならず、書き込み関数はこのページに書くことができません。フラッシュ回復はプログラム始動でRecoverFlash()関数を呼ぶことによって実行されます。

フラッシュ回復任意選択が許可されると、ページ書き込みは与えられたフラッシュ ページへの書き込みを行う前に、フラッシュ内の回復専用ページ内へデータの予備保存を伴います。フラッシュ回復ページがデータを含むことを示す状態バイトと共に、書かれるべきページに対するアドレスがEEPROMに保存されます。この状態バイトは与えられたフラッシュ ページへの実際の書き込みが成功裏に完了される時に解除されます。EEPROM内の変数とフラッシュ回復緩衝部は、必要な時にデータを回復するRecoverFlash()フラッシュ回復関数によって使われます。EEPROMへの1バイト書き込みはフラッシュへのページ全体書き込みと同じくらいの時間がかかります。従って、フラッシュ回復任意選択許可時、総書き込み時間がかなり増加するでしょう。フラッシュでの少量バイト予約がそれらのバイトを含むフラッシュ ページ全体の柔軟な使用を阻害するため、フラッシュの代わりにEEPROMが使われます。

2.2. C関数説明

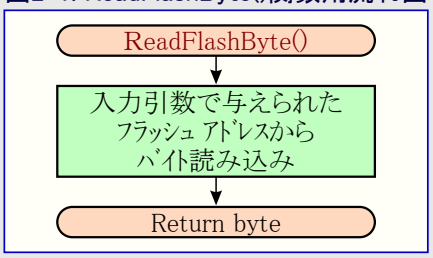
表2-1. フラッシュ メモリ入出力用C関数

関数	引数	戻り値
ReadFlashByte()	MyAddressType flashAdr	unsigned char
ReadFlashPage()	MyAddressType flashStartAdr, unsigned char *dataPage	unsigned char
WriteFlashByte()	MyAddressType flashAddr, unsigned char data	unsigned char
WriteFlashPage()	MyAddressType flashStartAdr, unsigned char *dataPage	unsigned char
RecoverFlash()	Void	unsigned char

データ型MyAddressTypeはSelf_programming.hで定義されます。このデータ型の大きさは使われるデバイスに依存します。それは64Kバイトを越えるフラッシュ メモリを持つデバイスの使用時にlong intとして、64Kバイト以下のフラッシュ メモリを持つデバイスの使用時にint(16ビット)として定義され得ます。このデータ型は_flashまたは_farflashポインタ(結果的に16と24ビット)として実際に使われます。何故新しいデータ型が定義されるかの理由は、整数型がポインタ型よりもずっと多くの柔軟な使用を許すからです。

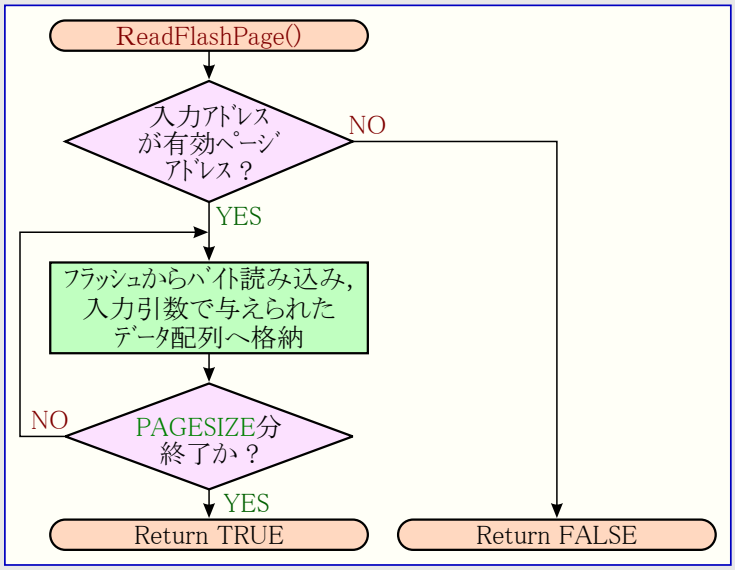
ReadFlashByte()はFlashAdr入力引数によって与えられたフラッシュ アドレスに配置された1バイトを返します。

図2-1. ReadFlashByte()関数用流れ図



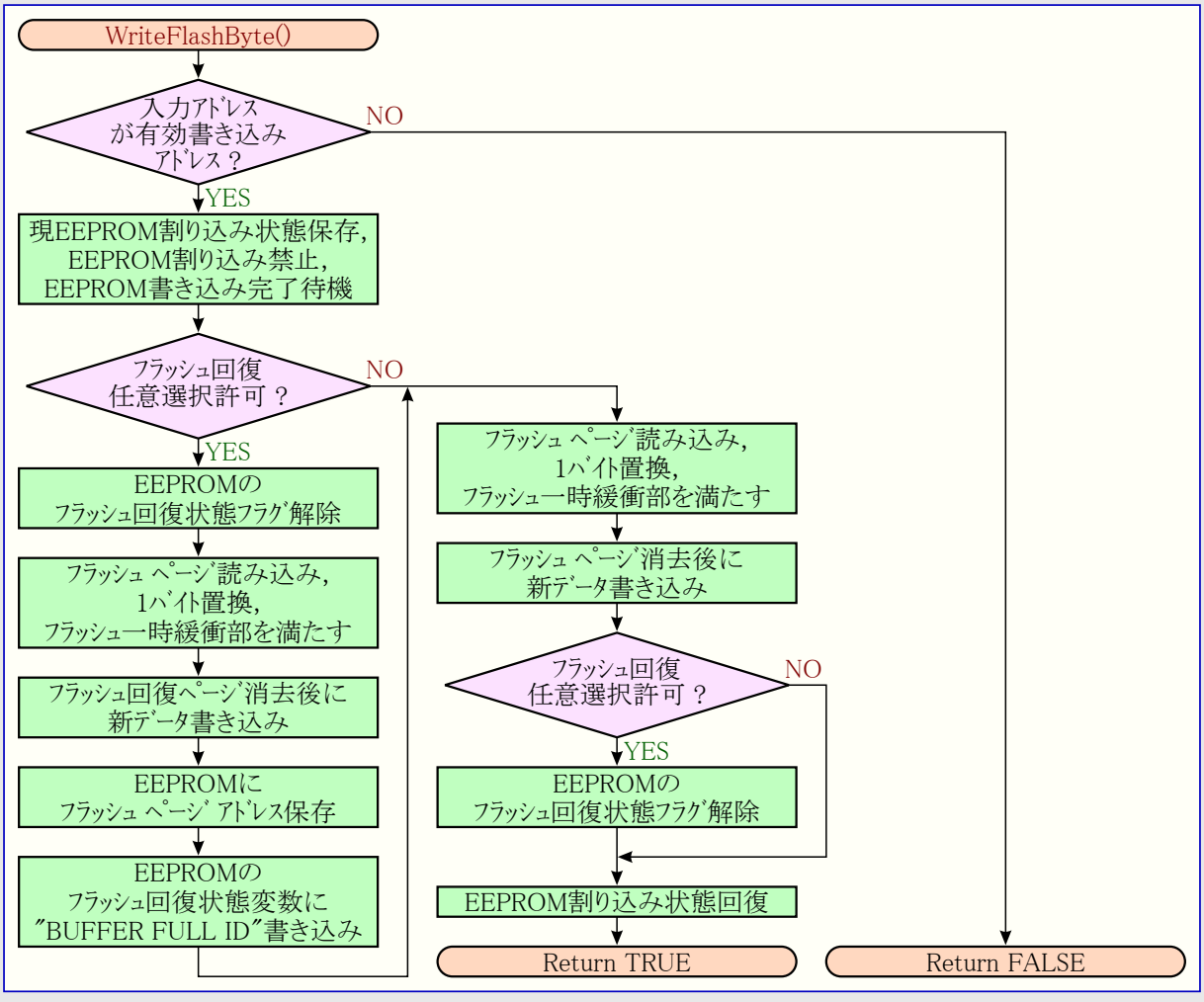
ReadFlashPage()はFlashStartAdr入力引数によって与えられたｱﾄﾞﾚｽからの1ﾌﾗｯｼｭ ﾍﾟｰｼﾞを読み、pucDataPage[]入力引数によって与えられた配列にﾃﾞｰﾀを格納します。格納されるﾊﾞｲﾄ数はﾌﾗｯｼｭ ﾍﾟｰｼﾞの大きさに依存します。関数は入力ｱﾄﾞﾚｽがﾌﾗｯｼｭ ﾍﾟｰｼﾞ ｱﾄﾞﾚｽでなければ偽(FALSE)を、そうでなければ真(TRUE)を返します。

図2-2. ReadFlashPage()関数用流れ図



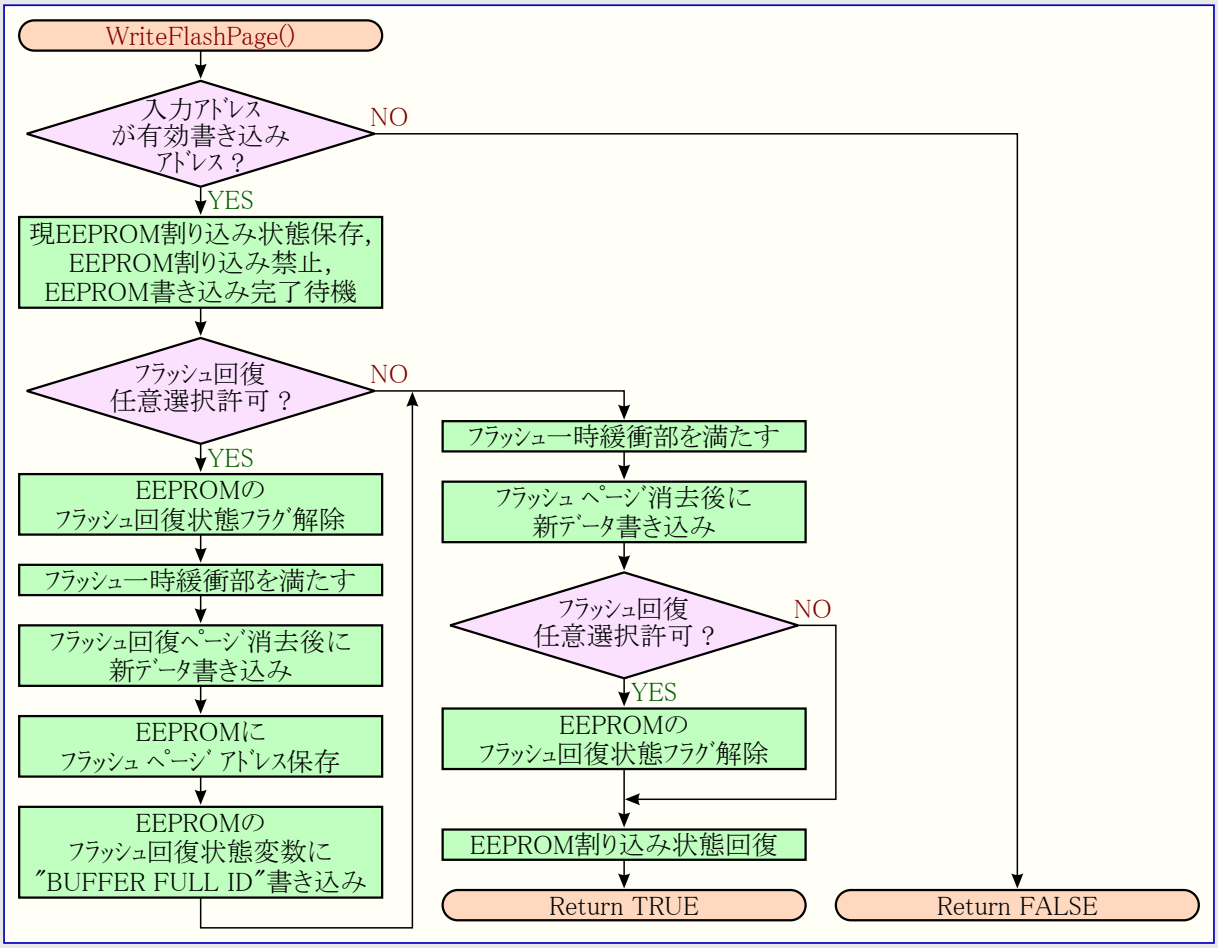
WriteFlashByte()はData入力引数によって与えられたﾊﾞｲﾄをFlashAddr入力引数によって与えられたｱﾄﾞﾚｽに書きます。関数は入力ｱﾄﾞﾚｽが書き込みに関して有効なﾌﾗｯｼｭ ﾊﾞｲﾄ ｱﾄﾞﾚｽでない場合に偽(FALSE)を、そうでなければ真(TRUE)を返します。

図2-3. WriteFlashByte()関数用流れ図



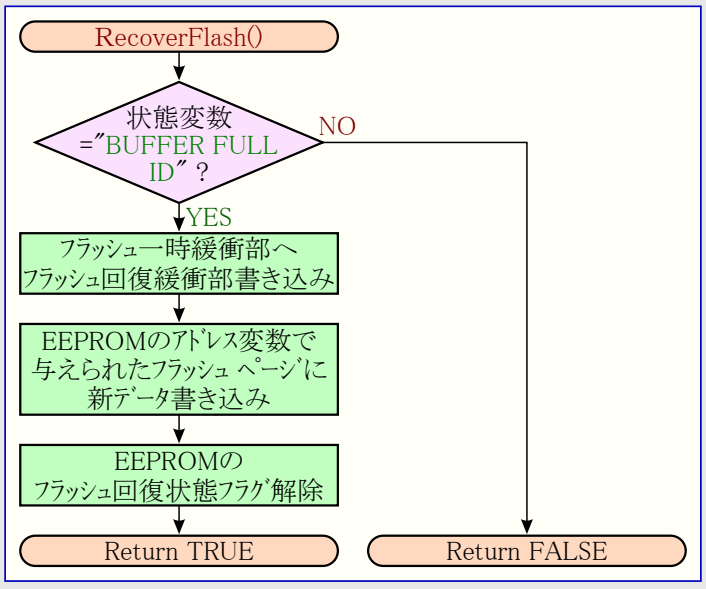
WriteFlashPage()はDataPage[]入力引数によって与えられた配列からのﾃﾞｰﾀをFlashStartAdr入力引数によって与えられたﾌﾗｯｼｭﾍﾟｰｼﾞｱﾄﾞﾚｽに書きます。書かれるﾊﾞｲﾄ数はﾌﾗｯｼｭ ﾍﾟｰｼﾞの大きさに依存します。関数は入力ｱﾄﾞﾚｽが書き込みに関して有効なﾌﾗｯｼｭ ﾍﾟｰｼﾞ ｱﾄﾞﾚｽでない場合に偽(FALSE)を、そうでなければ真(TRUE)を返します。

図2-4. WriteFlashPage()関数用流れ図



RecoverFlash()はEEPROM内の状態変数を読み、必要な場合にﾌﾗｯｼｭ ﾍﾟｰｼﾞを回復します。関数はﾌﾗｯｼｭ回復任意選択が許可された場合にﾌﾟﾛｸﾞﾗﾑの始動で呼ばれなければなりません。関数はﾌﾗｯｼｭ回復が行われた場合に真(TRUE)を、さもなくば偽(FALSE)を返します。

図2-5. RecoverFlash()関数用流れ図



2.3. 他のﾃﾞﾊﾞｲｽに実装する手順

ﾌｧｰﾑｳｪｱは特定のﾌﾞｰﾄ ﾛｰﾀﾞ領域を持つ他のAVRﾃﾞﾊﾞｲｽで再利用することができます。

ｺｰﾄﾞを再利用するための手順は次のとおりです。

1. 使うﾃﾞﾊﾞｲｽで利用可能なﾌﾞｰﾄ ｾｸﾞﾒﾝﾄの大きさでself_programming.hﾌｧｲﾙ内で利用可能な次のﾏｸﾛを変更してください。

```
#define BOOTSECTORSIZE 0x2000    // 4096語
```

2. 利用可能なﾌﾗｯｼｭの大きさに基づいてﾌﾗｯｼｭ回復緩衝部に対する位置でself_programming.hﾌｧｲﾙ内で利用可能な次のﾏｸﾛを変更してください。

```
#define ADR_FLASH_BUFFER 0xEF00
```

3. 「**図1-2. ﾌﾞｰﾄ ｾｸﾞﾒﾝﾄ定義**」で示されるように、このｾｸﾞﾒﾝﾄ内に望む関数を配置するのにBOOTLOADER SECTION属性を使い、ﾌﾞｰﾄ ﾒﾓﾘ空間に対応するﾌﾟﾛｼﾞｪｸﾄ ﾌﾟﾛﾊﾟﾃｨ ｳｨﾝﾄﾞｳ下で利用可能なﾌﾗｯｼｭ ｾｸﾞﾒﾝﾄ定義を変更してください。この図はATmega128用のﾒﾓﾘ ｾｸﾞﾒﾝﾄ定義を示します。ATmega128の4Kﾌﾞｰﾄ ﾛｰﾀﾞのﾌﾞｰﾄ ﾛｰﾀﾞ開始語ｱﾄﾞﾚｽは0xF000です。同様に使うﾃﾞﾊﾞｲｽ用にﾒﾓﾘ ｾｸﾞﾒﾝﾄ定義を変更してください。

4. 望むBOOTSZﾋｭｰｽﾞ設定がﾌﾟﾛｸﾞﾗﾐﾝｸﾞされているのを確実にしてください。ATmega129については4096W_F000であるべきです。

専用ﾌﾞｰﾄ領域なしﾃﾞﾊﾞｲｽでの実装

専用のﾌﾞｰﾄ領域を持たないﾃﾞﾊﾞｲｽに対してｺｰﾄﾞを実装すると同時に、以下の条件が満足されることを確実にしてください。

1. 専用ﾌﾞｰﾄ領域を持たないﾃﾞﾊﾞｲｽに対しては、ﾌﾗｯｼｭ書き込み関数を配置するためのﾌﾞｰﾄ領域としてﾌﾗｯｼｭ ﾒﾓﾘの最後で少量のﾍﾟｰｼﾞを利用してください。ﾌﾗｯｼｭ書き込みﾙｰﾁﾝは概ね550ﾊﾞｲﾄのﾒﾓﾘを消費します。故に、模倣したﾌﾞｰﾄ領域が550ﾊﾞｲﾄ未満でないことを確実にしてください。
2. 実際の応用ｺｰﾄﾞが上書きされないような方法でﾌﾗｯｼｭ書き込み関数のｱﾄﾞﾚｽと回復ﾌﾗｯｼｭ緩衝部のｱﾄﾞﾚｽを提供してください。

3. 要約

この応用記述はﾌﾗｯｼｭ ﾒﾓﾘをｱｸｾｽするための4つの異なるC関数を提供します。

この応用記述では以下の機能が網羅されます。

1. ﾌﾗｯｼｭ ﾍﾟｰｼﾞ書き込み
2. ﾌﾗｯｼｭ ﾍﾟｰｼﾞ読み込み
3. ﾌﾗｯｼｭ ﾊﾞｲﾄ書き込み
4. ﾌﾗｯｼｭ ﾊﾞｲﾄ読み込み

ﾌｧｰﾑｳｪｱはﾌﾗｯｼｭ書き込みﾙｰﾁﾝをﾌﾞｰﾄ領域内に、残りのｺｰﾄﾞをﾌﾗｯｼｭ ﾒﾓﾘの応用領域に置きます。この応用記述で利用可能なﾌｧｰﾑｳｪｱは上で言及した動作をATmega128で実行します。他のﾃﾞﾊﾞｲｽに対して応用ｺｰﾄﾞを再利用するための手順も説明します。

4. 更なる読み物

- ATmega128ﾃﾞｰﾀｼｰﾄ
- AVR109：自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ
- AVR105：ﾌﾗｯｼｭ ﾒﾓﾘでの電力効率的な高耐久性ﾊﾟﾗﾒｰﾀ記憶
- AVR108：LPM命令の初期設定と使用
- AVR116：DataFlashでの摩耗平滑化
- AVR947：自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ能力を持つ任意MCU用単線ﾌﾞｰﾄ ﾛｰﾀﾞ

5. 改訂履歴

文書改訂	日付	注釈
2575B	2006年8月	初版文書公開
2575C	2016年3月	ﾌｧｰﾑｳｪｱがAtmel Studio 7.0に移転され、他のﾃﾞﾊﾞｲｽでの実装の手順を追加。加えて選んだ関数をﾌﾞｰﾄ領域内に配置するようにｺｰﾄﾞを変更。

本応用記述はAtmelのAVR106応用記述(Rev.2575C-03/2016)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。