E2PROM データシート E2PROM V 1.7

Copyright © 2004-2011 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック			API メモリ（バイト）		ピン（外部 I/O あたり）
	デジタル	アナログ CT	アナログ SC	フラッシュ	RAM	
CY8C29x66, CY8CLED16, CY8CPLC20, CY8CLED16P01	0	0	0	26+1013 + サイズ	下記参照	0
CY8C27x43, CY8C24x94, CY8C22x13, CY7C64215, CY8CLED04/08, CY8CLED0xD, CY8CLED0xG, CY8C22x45, CY8C28x45, CY8C28xxx	0	0	0	25+750 + サイズ	下記参照	0
CY8C24x23A, CY8C23x33, CY8C21x23, CY8CLED02	0	0	0	25+750 + サイズ	下記参照	0
CY8C21x34, CY7C603xx, CYWUSB6953, CY8C20x24, CY8C20x34, CY8C21x45, CY8C21x12	0	0	0	25+1044 + サイズ	下記参照	0
CY8C26/25xxx	0	0	0	19+777 + サイズ	下記参照	0

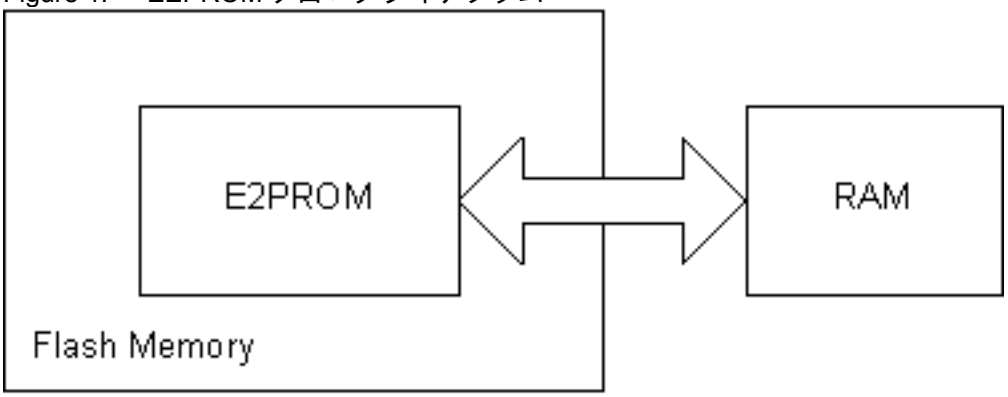
このユーザ モジュールを使用する機能例として完全に構成されたプロジェクトについて を参照してください。www.cypress.com/psocexampleprojects

特徴と概要

- フルバイト ベースの EEPROM エミュレーション
- ブロック指向の Flash アーキテクチャの要約
- メモリの効率的な使用

EEPROM ユーザ モジュールは、PSoC データの Flash デバイス内で EEPROM デバイスをエミュレートします。EEPROM デバイスは、1 から Flash メモリ空間の残りまでのバイト長を用いて、任意の Flash ブロック境界で開始するよう定義できます。API は、一回に 1 ~ N バイトまでの読み取り・ 書き込みを可能にします。

Figure 1. EEPROM ブロック ダイアグラム



機能説明

EEPROM ユーザ モジュールは、PSoC デバイスのハードウェアリソースを使用しないソフトウェアアルゴリズムです。これらの EEPROM 仮想デバイスのインスタンスは 1 つまたは複数作成できます。

Flash はデバイスごとに、32K デバイスでは 64 バイト・ 511 ブロック、16K デバイスでは 64 バイト・ 256 ブロック、8K デバイスでは 64 バイト・ 128 ブロック、4K デバイスでは 64 バイト・ 64 ブロックに構成されます。PSoC のアーキテクチャでは、Flash データのバイトごとの読み取りが可能ですが、データはブロックごとに、一回に 64 バイトに書き込む必要があります。このユーザ モジュールの目的は、EEPROM デバイス（バイト読み取り、バイト書き込みベースのデバイス）を Flash ベースのメモリデバイス（バイト読み取り、ブロック書き込みベースのデバイス）にエミュレートすることです。

EEPROM 記憶装置エリアは、Flash メモリブロック境界で開始し、1 バイト以上で構成されます。この仮想デバイスへは、E2Read() および E2Write() API ルーチンを使用してアクセスできます。仮想アドレス空間は 0 ~ N-1 で、ここでの N は EEPROM デバイスのバイト長 / サイズです。記憶用に予約されている領域のサイズは、Flash メモリの使用に含まれますが、その大きさはリソースメータでは計算できません。このサイズ要因は、上のメモリ使用表で 「Size（サイズ）」という項目にあたります。

E2Read() API アルゴリズムは、ROMX M8C 指示を使用して、Flash メモリをバイトごとに効率的に読み取ります。このアルゴリズムは、RAM の最後の 8 バイト 0xF8 ~ 0xFF

E2Write() API アルゴリズムは、データを Flash メモリにブロックごとに書き込みます。EEPROM メモリ空間への開始アドレスオフセットに基づき、E2Write() ルーチンはブロック境界に配置されているセグメントに書き込まれるデータを解析します。このアルゴリズムも、RAM の最後の 8 バイト 0xF8 ~ 0xFF

1 つのブロック全体（64 バイト）にまたがるセグメントでは、FlashBlockWrite() ルーチンが呼び出され、ブロックに書き込みが行われます。

64 バイト未満のセグメントでは、未書き込み領域と修正領域の両方で一時的な 64-byte スタックバッファが構成されます。データの書き込み後、バッファはスタックから解放されます。これは、ブロック内でデータを保存するために必要な動作です。

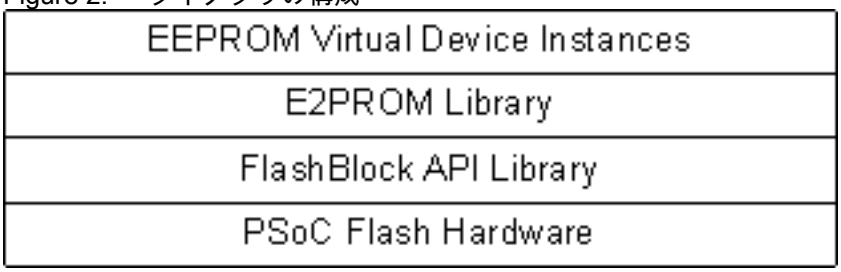
パラメータを最適化してシステムオーバーヘッドを低減する方法については、後述の*パラメータとリソース*セクションを参照してください。

ソフトウェアの構成

次の図は、2 つのライブラリとこの N EEPROM 仮想デバイスインスタンスを含むいくつかのレイヤから成るソフトウェアの構成を示しています。

上のリソース使用には、Flash の API ライブラリのサイズと、FlashBlock API ライブラリおよび E2PROM ライブラリに含まれるコードが挙げられています。このデバイスのインスタンスが複数使用される場合は、UserModule E2PROM API 長がリソースメータで計算されます。1 つの FlashBlock API と E2PROM ライブラリだけが、任意の数のユーザ モジュールのインスタンスに使用されます。

Figure 2.　ライブラリの構成



FlashBlock API ライブラリは、基本的な Flash ブロックの読み取りと書き込みルーチンを可能にする API を提供します。これらのルーチンはハードウェアと直接インターフェース接続しており、デバイスをシステム監視モードに移行させます。この間、すべての割り込みがマスクされます。すべての EEPROM 仮想デバイスのインスタンスについて、このライブラリは一度だけリンクされます。

E2PROM ライブラリは、ブロック配列とバッファ使用により、ブロック動作をバイトベースの動作に変換します。次に、Flash に書き込むために FlashBlock API ライブラリを呼び出します。EEPROM 仮想デバイス数に関わらず、このライブラリは一度だけリンクされます。

EEPROM 仮想デバイスレイヤは、E2PROM ライブラリと FlashBlock API ライブラリの両方にインスタンスわずか 1 つ分のオーバーヘッドをかけながら、EEPROM デバイスの複数のインスタンスを許可するシンプルな手段を提供します。EEPROM 仮想デバイスの各インスタンスには、最低が 16 バイトのソースコードのカスタマイズコピーが含まれます。

E2PROM と FlashBlock API ライブラリの両方がライブラリファイルに含まれています。このファイルは、その関数が呼び出された場合、自動的にプロジェクトにリンクされます。

Note　これらのライブラリは再入可能ではありません。

Flash 書き込みサイクルの寿命

Flash メモリは書き込み回数の寿命があります。デバイス別の合計数については、該当するデバイスファミリのデバイスシートを参照してください。

フラッシュの保護

EEPROM を構成するブロックに Flash 書き込みを行うためには、*flashsecurity.txt* EEPROM を構成するブロックに Flash 書き込みを行うためには、flashsecurity.txt ファイルを編集しなければなりません。

デバイスの各 Flash ブロックに対応する flashsecurity.txt には、ASCII 文字が 1 つあります。この ASCII 文字の有効なオプションは、「W」、「R」、「U」、「F」です。このファイル中の ASCII 文字は、各 Flash ブロックに適用される Flash セキュリティ設定を定義します。「W」はデフォルト設定で、Flash ブロックをすべての書き込みから保護します。E2PROM ユーザ モジュールでブロックの読み取りと書き込みを可能にするには、保護オプションを「R」、「F」、「U」のいずれかに変更します。オプションは、E2PROM が占有している Flash ブロック別に変更します。でセキュリティオプションを適切に変更するには　*flashsecurity.txt*、影響を受ける Flash ブロックに対応する文字を削除します。任意のセキュ

[+] Feedback

リティ設定に対応する文字を新規に入力します。変更設定が有効になったことを保証するために、プロジェクトを保存、再構築します。

モード	設定	意味	PSoC デザイナ
00b	SR ER EW IW	未保護	U = 未保護
01b	SR ER EW IW	読み取り保護	F = 工場アップグレード
10b	SR ER EW IW	外部書き込みの無効化	R = 現場アップグレード
11b	SR ER EW IW	内部書き込みの無効化	W = 完全保護

メモリの効率的な使用

次のガイドラインは、RAM と Flash リソースの両方の効率を最大化するために使用します。

Flash メモリが十分ある場合（RAM の効率的使用）

- 保存されるデータが少ない場合でも、64-byte 長の倍数で EEPROM デバイスを構成します。
- 常に、64-byte 長の倍数でデータを書き込みます。
- 常に、正しいバイト カウントのデータを読み取ります。

例えば、シリアル番号データの長さは 10 バイトです。名前のシリアル番号を用いて EEPROM ユーザモジュールを構成します。長さのパラメータを 64 に設定します。これにより、Flash メモリに 64-byte ブロックが割り当てられます。RAM では、10-byte データアレイを設定して、シリアル番号データを保持します。E2Write() を介してシリアル番号を書くときは、wByteCount パラメータを 64 に設定します。ただし、シリアル番号データを読み取るときは、wByteCount パラメータを 10 に設定します。

Flash メモリがプレミアムの場合（Flash の効率的な使用）

- 必要バイト数のみを使用して EEPROM デバイスを構成します。
- 常に、正しいバイト カウントのデータを書き込みます。
- 常に、正しいバイト カウントのデータを読み取ります。

タイミング

次の表には、EEPROM アルゴリズムのタイミングがまとめられています。

Table 1.　別途但し書きがない限り、すべての制限は、T_A = 25°C、V_{dd} = 5.0V を条件としています。

パラメータ	条件および注意	標準値	制限	単位
0°C ~ 100°C 書き込みの時間	書き込みの時間は、更新を必要とする Flash ブロック数によって異なります。[1,2]	45	120[3,4]	ms/Block
-40°C ~ 0°C 書き込みの時間	書き込みの時間は、更新を必要とする Flash ブロック数によって異なります。[1,2]	70	240[3,4]	ms/Block

[+] Feedback

パラメータ	条件および注意	標準値	制限	単位
読み取り時間	81 + 46N クロック周期	--	--	Clks/Bytes
	最悪の場合は 1 バイト :	--	127	
	16 バイトの場合 : (81+46(16))/10	51	--	
	64 バイトの場合 : (81+46(64))/64	48	--	

タイミング表に関する注意

1. M8C がシステム監視モードのとき、Flash 読み取り· 書き込み操作の間、割り込みはマスクされます。
2. 書き込み時間は、各 Flash ブロックを消去して書き込む FlashBlockWrite() ルーチンによって決定します。
3. Flash 書き込み総数には制限があります。ブロックごとの書き込み数の仕様については、パーツ別のデータシートを参照してください。
4. 標準的プログラム操作には制限が設けられています。デバッガを使用したエミュレーションモードでは、書き込みパルスが 1 ~ 2 秒かかる場合もあります。

DC および AC の電気的特徴

Flash の特徴と仕様については、PSoC デバイスのデータシートを参照してください。

配置

EEPROM ユーザ モジュールはソフトウェアに実装されており、配置は不要です。

パラメータおよびリソース

FirstBlock

FirstBlock は、EEPROM デバイスが存在する開始ブロックです。値はデバイス別に 0 ~最大 Flash ブロック数の範囲にあり、32K デバイスでは 511、16K デバイスでは 255、8K デバイスでは 127、4K デバイスでは 63、2K デバイスでは 31 です。

E2PROM ユーザ モジュールの Flash 記憶場所が、Flash にあるコードやその他の重要データと重複しないよう、十分注意してください。E2PROM 記憶場所は Flash メモリ内でできるだけ高い位置に置きます。例えば、512 ブロックの Flash を持つデバイスの場合、E2PROM ユーザ モジュールの配置場所としては Flash ブロック 511 が適切です。これを Flash ブロック 0 や 1 に配置すると、E2PROM 場所が割り込みベクタと重複するため、システム障害が起きる可能性が高くなります。ブロック 0 や 1 における E2PROM に書き込みをすると、割り込みベクタ テーブルが上書きされます。Flash メモリの中央近くに E2PROM を配置することも適切ではありません。ImageCraft コンパイラは、E2PROM 用に別途のメモリ領域を予約しません。したがって、コードが大きくなると、Flash メモリの中央近くの E2PROM 位置と重複してしまいます。

長さ

長さは、EEPROM デバイスのサイズをバイト数で示します。有効な範囲は 1 ~ N で、ここでの N は EEPROM 仮想デバイスのサイズをバイト数で示したものです。

警告 : 仮想 EEPROM デバイスの物理的位置が、デバイスの Flash サイズを超えないように注意してください。EEPROM デバイスの第 1 物理バイトは、FirstBlock * 64 に位置し、最後の物理バイトは、FirstBlock * 64 + 長さ - 1 に位置します。エラーチェックは行われません。サイズが大きすぎ

ないように制限する措置を怠ると、意図しないブロックに予期しない Flash 書き込みが起こる場合があります。

RAM メモリのオーバーヘッド

次の RAM メモリリソースを使用します。

API ルーチン	メモリ使用
E2Read()	スタック領域： 8 バイト RAM High メモリ : 0xF8 – 0xFF
E2Write() 部分ブロック書き込みなし	スタック領域： 32 バイト RAM High メモリ : 9 Bytes 0xF8 – 0xFF
E2Write()。部分ブロック書き込みあり	スタック領域： 103 バイト RAM High メモリ : 9 Bytes 0xF8 – 0xFF

Application Programming Interface（アプリケーション プログラミング インタフェース）

アプリケーション プログラミング インタフェース (API) ルーチンは、設計者が高級言語でモジュールを操作できるようにユーザ モジュールをコンポーネントとして提供します。このセクションは、“include” ファイルによって記載される関連する定数と共に各機能に対するインターフェースを指定しています。

Note すべてのユーザ モジュール API の場合と同じように、API 関数を呼び出すことで A と X レジスタの値が変更されることがあります。関数の呼び出し後に A と X の値が必要になる場合は、必ず呼び出し前に A と X の値を保存してください。“registers are volatile”(レジスタは揮発性である) ポリシーは、効率的な理由から選択されて、PSoC Designer のバージョン 1.0 より有効となっています。C コンパイラは、自動的にこの条件で処理されています。アセンブラ言語のプログラマは、コードがこのポリシーを遵守していることも確認する必要があります。ユーザ モジュール API 関数の中には、A と X を変更しないものもありますが、今後もそのまま残されていない可能性があります。

下記は、EEPROM ユーザ モジュール API の説明です。

E2PROM_Start

説明：

ヌル関数。ユーザ モジュール API 整合性を保つことが目的です。

C プロトタイプ：

```
void  E2PROM_Start(void)
```

アセンブラ：

```
lcall  E2PROM_Start
```

パラメータ：

なし

戻り値：

なし

[+] Feedback

副作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは、CY8C29xxx の大容量メモリ モデルにおける RAM ページポインタの全レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

E2PROM_Stop

説明：

ヌル関数。ユーザ モジュール API 整合性を保つことが目的です。

C プロトタイプ：

```
void  E2PROM_Stop(void)
```

アセンブラ：

```
lcall  E2PROM_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは、CY8C29xxx の大容量メモリ モデルにおける RAM ページポインタの全レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

E2PROM_bE2Write

説明：

RAM バッファから指定 EEPROM に指定データを書き込みます。注：Flash 書き込みを許可するために、*flashsecurity.txt* ファイルは適正に設定しなければなりません。

C プロトタイプ：

```
CHAR  E2PROM_bE2Write(WORD wAddr, BYTE *pbData, WORD wByteCount,
                      CHAR cTemperature);
```

アセンブリ：

```
push  X
mov   X, SP
mov   A, <cTemperature>              ; cTemperature argument
push  A
mov   A, <wByteCount>                ; wByteCount – MSB
push  A
mov   A, <wByteCount+1>              ; wByteCount - LSB
push  A
mov   A, <pbData>                    ; pbData – MSB
push  A
mov   A, <pbData+1>                  ; pbData – LSB
push  A
mov   A, <wAddr>                     ; wAddr - MSB
```

```
push  A
mov   A, <wAddr+1>                    ; wAddr - LSB
push  A
lcall  E2PROM_bE2Write
add   SP, -E2_WR_ARG_STACK_FRAME_SIZE ; restore call stack
pop   X
```

上記の「<..>」は、累算器に参照データを配置する指示数やアドレスモードを意味します。

パラメータ：

wAddr: RAM データの書き込み元である EEPROM デバイス領域のアドレスオフセット。この範囲は 0 ~ N-1 で、ここでの N は EEPROM デバイス長です。

pbData: 書き込みデータを含んでいる RAM バッファを示すポインタ。

wByteCounter: Flash へ書き込むバイト数。

cTemperature: PSoC ダイの温度（摂氏）。この値は、次のうちいずれかを使用して指定します。

- FlashTemp などのユーザ モジュール。
- 外部デバイスやセンサ。
- PSoC デバイスが経験するすべての環境条件に適用される公称値。

例：室温 = 25°C

戻り値：

次の値が戻されます。

戻りフラグ	意味	値
NOERROR	操作完了。	0
FAILURE	操作失敗。Flash 保護ビットエラーの可能性大。	-1
STACKOVERFLOW	スタック領域が不十分で、アルゴリズム要件に適合できませんでした。	-2

副作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは、CY8C29xxx の大容量メモリ モデルにおける RAM ページポインタの全レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現在変更されているのは、CUR_PP、IDX_PP、MVW_PP のページポインタレジスタだけです。

E2PROM_E2Read

説明：

指定 EEPROM デバイスデータを、Flash から指定の RAM バッファへ読み取ります。

C プロトタイプ：

```
void E2PROM_E2Read(WORD wAddr, BYTE *pbData, WORD wByteCount)
```

アセンブラ：

```
push  X
mov   X, SP
mov   A, <wByteCount>                  ; wByteCount - MSB
```

```
push  A
mov   A, <wByteCount+1>                ; wByteCount - LSB
push  A
mov   A, <pbDataDest>                  ; pbDataDest - MSB
push  A
mov   A, <pbDataDest+1>                ; pbDataDest - LSB
push  A
mov   A, <wAddr>                       ; wAddr - MSB
push  A
mov   A, <wAddr+1>                     ; wAddr - LSB
push  A
lcall   E2PROM_E2Read
add   SP, -E2_RD_ARG_STACK_FRAME_SIZE  ; restore call stack
pop   X
```

上記の 「<..>」は、累算器に参照データを配置する指示数やアドレスモードを意味します。

パラメータ：

wAddr: Flash データの読み取り元である EEPROM デバイス領域のアドレスオフセット。この範囲は 0 ~ N-1 で、ここでの N は EEPROM デバイス長です。

pbData: データ読み込み先の RAM バッファを示すポインタ。

wByteCounter: Flash から読み取りバイト数。

戻り値：

なし

副作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは、CY8C29xxx の大容量メモリ モデルにおける RAM ページポインタの全レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現在変更されているのは、CUR_PP、MVW_PP のページポインタレジスタだけです。

ファームウェア ソースコードの例

以下に、C 言語のソースコードを示します。

```
//*****************************************************************************
//*    EEPROM User Module Example Code:
//*
//*    A SerialNumber EEPROM was created to start at block 250 with a length
//*    of 10 bytes. Remember to edit the Flashsecurity.txt file to allow
//*    writes to flash  (Set the 250th block to U - Unprotected).
//*
//*    This example:
//*
//*        a) Writes the initial data to the EEPROM block area
//* Note that this will invoke the SavePartial algorithm which
//* allocates a temporary 64-byte buffer on the stack. If
//* Flash memory is plentiful and the extra 54 bytes can be
//* wasted, set the SerialNumber device to a length of 64 and when
//* writing, specify a bytecount of 64. This will write an entire
//* block without using a temporary buffer. The extra 54 bytes
//* beyond the SerialNumber will be bogus data.
//*
//*        b) Reads the data back into a RAM buffer
//*
//*****************************************************************************/

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#define  ADDRESS_OFFSET    0
#define  NUMBER_OF_BYTES   10
#define  TEMPERATURE       25

/* Initialize a RAM buffer with default Serial Number */
BYTE  abInitialSerialNumber[] = {'0','1','2','3','4','5','6','7','8','9' };
BYTE  abSerialNumberBuffer[NUMBER_OF_BYTES];

void main(void)
{
    BYTE bError;


    /* Write the Serial Number - assume temp of 25C */
    bError = E2PROM_bE2Write(ADDRESS_OFFSET, abInitialSerialNumber, NUMBER_OF_BYTES,
TEMPERATURE);


    if ( bError == E2PROM_NOERROR )
    {
        /* Read the Serial Number back into a RAM buffer */
        E2PROM_E2Read( ADDRESS_OFFSET, abSerialNumberBuffer, NUMBER_OF_BYTES );
    }

    while(1)
    {
    }
```

```
}
```

アセンブリでの同じコードは以下のようになります。

```
;***************************************************************************
;    EEPROM User Module Example Code:
;
;    A SerialNumber EEPROM was created to start at block 250 with a length
;    of 10 bytes. Remember to edit the Flashsecurity.txt file to allow
;    writes to Flash (Set the 250th block to U - Unprotected).
;
;    This example:
;
;        a) Writes the initial data to the EEPROM block area
;        Note that this will invoke the SavePartial algorithm which
;        allocates a temporary 64-byte buffer on the stack. If
;        Flash memory is plentiful and the extra 54 bytes can be
;        wasted, set the SerialNumber device to a length of 64 and when
;        writing, specify a bytecount of 64. This will write an entire
;        block without using a temporary buffer. The extra 54 bytes
;        beyond the SerialNumber will be bogus data.
;
;        b) Reads the data back into a RAM buffer
;
;***************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants and macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

ADDRESS_OFFSET:             equ     0
NUMBER_OF_BYTES:            equ     10
TEMPERATURE:                equ     25

export _main
export  _SerialNumberExample            ; C export
export   SerialNumberExample            ; assembler export

export  abSerialNumberString
export  abInitialSerialNumber
export  abSerialNumberBuffer
export  bCounter
export  pPtr

    AREA    bss     (RAM,REL)

abInitialSerialNumber:    blk    NUMBER_OF_BYTES   ; string holds initial serial data
abSerialNumberBuffer:     blk    NUMBER_OF_BYTES   ; buffer to read back serial data
bCounter:                 blk     1                ; counter to load initial string
pPtr:                     blk     1                ; pointer to initial string

    AREA    text (ROM,REL)

;Table to hold initial serial number string
.LITERAL
```

```
abSerialNumberString:    db     '0','1','2','3','4','5','6','7','8','9'
.ENDLITERAL

_main:

; load the serialnumber into RAM
    mov    [bCounter], NUMBER_OF_BYTES       ; load 10 bytes from ROM into RAM

    RAM_SETPAGE_MVR >abInitialSerialNumber
    RAM_SETPAGE_MVW >abInitialSerialNumber
    mov    [pPtr], <abInitialSerialNumber ; ptr RAM data to put Flash data
    mov    X, <abSerialNumberString       ; LSB of abSerialNumberString
    mov    A, >abSerialNumberString          ; MSB of abSerialNumberString

; Use ROMX and MVI to copy the data from Flash to RAM
.loop:
    push   A                              ; Save MSB of abSerialNumberString to Stack
    romx
    mvi    [pPtr], A                      ; Save the value to RAM
    pop    A                              ; Get MSB of abSerialNumberString from Stack
    inc    X                              ; Increment LSB of abSerialNumberString
    dec    [bCounter]
    jnz    .loop


; Write the Serial Number - assume temp of 25C
    mov    A, TEMPERATURE                 ; temperature = 25C
    push   A
    mov    A, >NUMBER_OF_BYTES            ; MSB of wByteCount = 0
    push   A
    mov    A, <NUMBER_OF_BYTES        ; LSB of wByteCount = 10
    push   A
    mov    A, >abInitialSerialNumber      ; MSB of pbDest= >abInitialSerialNumber
    push   A
    mov    A, <abInitialSerialNumber  ; LSB of pbDest=abInitialSerialNumber
    push   A
    mov    A, >ADDRESS_OFFSET             ; MSB of wAddr=0
    push   A
    mov    A, <ADDRESS_OFFSET         ; LSB of wAddr=0
    push   A
    call   E2PROM_bE2Write                ; Write the data
    add    SP, -E2_WR_ARG_STACK_FRAME_SIZE
    pop    X


    ;if ( bError == NOERROR )
    cmp    A, 0
    jnz    .ExampleDone


    ; Read the Serial Number back into a RAM buffer
    mov    A, >NUMBER_OF_BYTES             ; MSB of wByteCount = 0
    push   A
    mov    A, <NUMBER_OF_BYTES         ; LSB of wByteCount = 10
    push   A
```

```
    mov     A, >abSerialNumberBuffer        ;MSB of pbDest= >abSerialNumberBuffer
    push    A
    mov     A, <abSerialNumberBuffer    ; LSB of pbDest=abInitialSerialNumber
    push    A
    mov     A, >ADDRESS_OFFSET              ; MSB of wAddr=0
    push    A
    mov     A, <ADDRESS_OFFSET          ; LSB of wAddr=0
    push    A
    call    E2PROM_E2Read
    add     SP, -E2_RD_ARG_STACK_FRAME_SIZE
    pop     X

.ExampleDone:
    jmp .ExampleDone
```


Copyright © 2004-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer ™ and Programmable System-on-Chip ™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.