EEPROM データシート E2PROMx128 V 1.1

Copyright © 2008-2011 Cypress Semiconductor Corporation. All Rights Reserved.

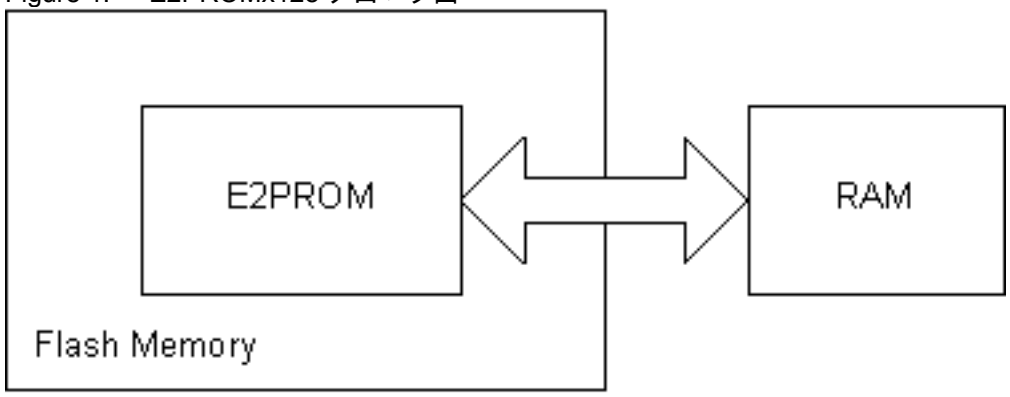
リソース	PSoC® ブロック				API メモリ（バイト数）	
	CapSense®	I2C/SPI	タイマ	コンパレータ	フラッシュ	RAM
CY8C20x66, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20x46, CY8C20x96, CY7C645xx, CY7C643/4/5xx, CY7C60424, CY7C6053x, CYONS2000, CYONS2001, CYONS2100, CYONS2101, CYONS2110, CYONSFN2xxx, CYONSTB2xxx						
	1	-	1	-	1540	35

特性および概要

- フルバイト ベースのエミュレーション
- 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

E2PROMx128 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. E2PROMx128 ブロック図



機能説明

E2PROMx128 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、32K デバイスでは、128 バイトの 256 ブロックとして構成されます。PSoC デバイス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 128 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、E2PROMx128_E2Read() および E2PROMx128_bE2Write() 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。

E2PROMx128_E2Read() API アルゴリズムは、ROMXM8C 命令を使用して、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 を使用する必要があります。

E2PROMx128_bE2Write() 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、E2PROMx128_bE2Write() 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 を使用する必要があります。

128 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 128-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、ブロック内で未修正データを保存するために必要な動作です。ブロック全体が 128 バイトにわたるセグメントの場合、一時バッファは作成されません。

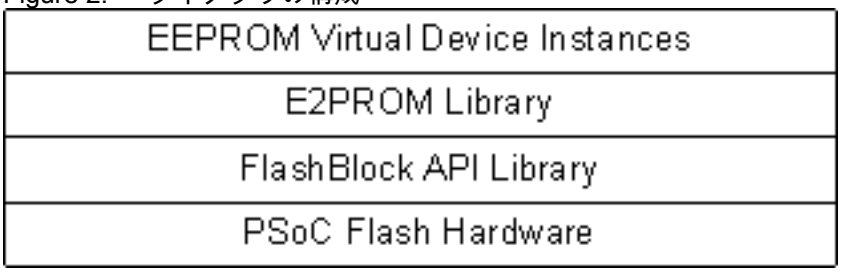
パラメータを最適化してシステムオーバーヘッドを低減する方法については、後述の*パラメータとリソース*セクションを参照してください。

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 E2PROMx128 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 E2PROMx128 API 長がリソースメータで計算されます。1 つの FlashBlock API ライブラリと E2PROMx128 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。

E2PROMx128 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、E2PROMx128 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。

E2PROMx128 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 デバイスを構成するブロックへの Flash 書き込みを可能にするために、*flashsecurity.txt* ファイルを編集しなければなりません。

タイミング

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。

Table 1. 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。

パラメータ	標準値	制限	単位	条件および注記
0°C ~ 100°C 書き込みの時間	45	120[3,4]	ms/Block	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]
-40°C ~ 0°C 書き込みの時間	70	240[3,4]	ms/Block	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]
読み取り時間	81 + 46N	--	CPU クロック周期	N は読み込みバイト数です。

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 ブロックを取り消して書き込む 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工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 電気的特性と AC 電気的特性

EEPROM の電気的特性については、PSoC デバイスのデバイス データシートを参照してください。

配置

EEPROMx128 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 255、16K デバイスでは 127、8K デバイスでは 3、4K デバイスでは 31、2K デバイスでは 15 です。

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 * 128 に位置し、最後の物理バイトは、(FirstBlock + 長さ - 1) * 128 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
E2PROMx128_E2Read()	スタック領域 : 8 バイト RAM High メモリ : 0xF8 – 0xFF
E2PROMx128_bE2Write()。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
E2PROMx128_bE2Write() 。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

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、128-byte 長の倍数で EEPROM デバイスを構成します。

- 常に、128-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 E2PROMx128 ユーザ モジュールを構成します。長さのパラメータを 128 に設定します。これにより、Flash メモリに 128-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 128 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。

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より高度なレベルでモジュールを処理できるようにユーザ モジュールの一部として提供されます。このセクションでは、各機能に対するインタフェースを「include」ファイルによって提供される関連定数とともに示します。

ユーザ モジュールを配置するたびに、インスタンス名が割り当てられます。デフォルトでは、PSoC Designer プロジェクトで、このユーザ モジュールの最初のインスタンスに E2PROMx128_1 を割り当てます。これは識別子の構文ルールに従った一意の値に変更できます。割り当てたインスタンス名が、全てのグローバル関数名、変数、および定数記号の接頭語になります。次の説明では、簡単にするために、インスタンス名は省略されて単に「E2PROMx128」となっています。

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。大容量メモリ モデル（c）のすべての RAM ページ ポインタ レジスタにも同じことが言えます。関数を呼び出す場合、呼出し後に A と X の値が必要になるならば、必ず呼び出し前に A と X の値を保存してください。PSoC Designer のバージョン 1.0 以降では、効率を高めるために、この「registers are volatile (レジスタの揮発性)」ポリシーが選択され、実施されています。C コンパイラは自動的にこの条件を処理します。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しなければなりません。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。

EEPROMx128 ユーザ モジュール API は、このセクションに記述されています。

E2PROMx128_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  E2PROMx128_Start(void)
```

アセンブラ

```
lcall  E2PROMx128_Start
```

パラメータ

なし

戻り値

なし

[+] Feedback

副作用

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。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 関数の呼び出しでこれらの値を保存してください。

E2PROMx128_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  E2PROMx128_Stop(void)
```

アセンブラ

```
lcall  E2PROMx128_Stop
```

パラメータ

なし

戻り値

なし

副作用

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。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 関数の呼び出しでこれらの値を保存してください。

E2PROMx128_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  E2PROMx128_bE2Write(WORD wAddr, BYTE *pbData, WORD wByteCount,
                    CHAR cTemperature);
```

アセンブラ

```
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  E2PROMx128_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命令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	説明	値
E2PROMx128_NOERROR	操作完了。	0
E2PROMx128_FAILURE	操作失敗。Flash 保護ビットエラーの可能性大。	-1
E2PROMx128_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

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。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 関数の呼び出しでこれらの値を保存してください。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。

E2PROMx128_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 E2PROMx128_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   E2PROMx128_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命令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

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。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 関数の呼び出しでこれらの値を保存してください。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。このサンプルコードは、CY7C64215 デバイスファミリ用に開発されたものです。このユーザ モジュールがサポートする別のデバイスファミリの中には、使用可能ポート、ピン、ドライブモードが異なっている場合があります。このコードをユーザのデバイスで使用する際、カスタマイズが必要な場合があります。

```
//*****************************************************************************
//*    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.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 = E2PROMx128_bE2Write(ADDRESS_OFFSET, abInitialSerialNumber,
NUMBER_OF_BYTES, TEMPERATURE);


    if ( bError == E2PROMx128_NOERROR )
    {
        /* Read the Serial Number back into a RAM buffer */
        E2PROMx128_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   E2PROMx128_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   E2PROMx128_E2Read
    add    SP, -E2_RD_ARG_STACK_FRAME_SIZE
    pop    X

.ExampleDone:
    jmp .ExampleDone
```


Copyright © 2008-2011 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.