EEPROM 数据表 EEPROMV 1.1

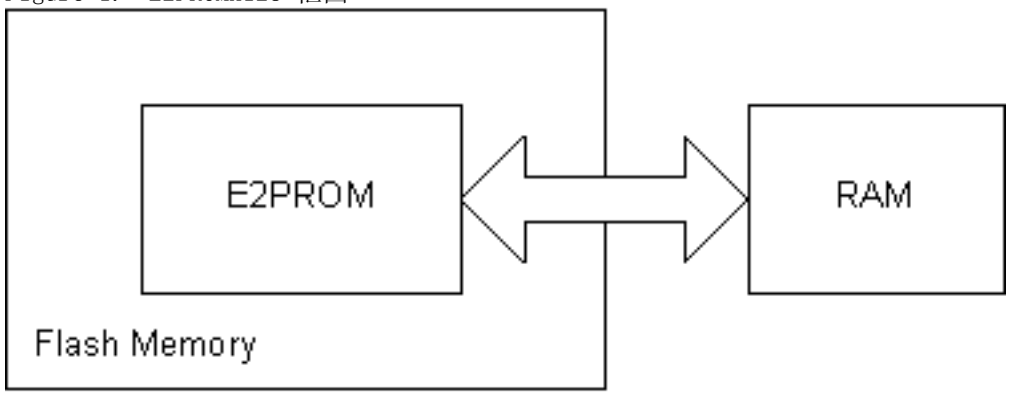
资源	PSoC® 模块				API 存储器 （字节）	
	CapSense®	I2C/SPI	定时器	比较器	闪存	RAM
CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C645xx、CY7C643/4/5xx、CY7C60424、CY7C6053x、CYONS2000、CYONS2001、CYONS2100、CYONS2101、CYONS2110、CYONSFN2xxx、CYONSTB2xxx						
	1	–	1	–	1540	35

功能和概述

- 完全面向字节的 EEPROM 仿真
- 抽象的面向模块的闪存体系架构
- 高效使用存储器

E2PROMx128 用户模块可以在 PSoC 器件的闪存内存中对 EEPROM 器件进行仿真。 可以将 EEPROM 器件定义为起始于任意的闪存模块边界，字节长度范围从 1 到剩余的闪存空间。 API 使用户能每次读取和写入 1 到 N 个字节。

Figure 1. E2PROMx128 框图



功能说明

E2PROMx128 用户模块是一种软件算法，不使用 PSoC 器件的任何硬件资源。 可以创建相应 EEPROM 虚拟器件的一个或多个实例。

对于 32K 的器件，每个器件的闪存分为 256 个模块，每个模块 128 个字节。 PSoC 器件的体系架构允许逐个字节读取闪存数据，但是要求逐个模块 （每次 128 个字节）写入数据。 这种用户模块的用途是在基于闪存的存储器 （面向字节读取、模块写入的器件）上对 EEPROM 器件 （面向字节读写的器件）进行仿真。

EEPROM 存储区域起始于闪存模块的边界，包含 1 个或多个字节。 可使用 E2PROMx128_E2Read() 和 E2PROMx128_bE2Write() API 例程访问此虚拟设备。 虚拟地址空间为 0 到 N-1，其中 N 是以字节为单位

的 EEPROM 设备长度 （大小）。 在闪存使用中还必须考虑为存储保留的区域大小，但是此数量不能通过资源仪来计算。 在上述存储器使用表中，此大小元素标识为 “ 大小 ”。

E2PROMx128_E2Read() API 算法使用 ROMX M8C 指令逐字节读取闪存。 此算法需要使用 RAM 的最后 8 个字节： 0xF8 到 0xFF。

E2PROMx128_bE2Write() API 算法将数据逐块写入闪存。 根据 EEPROM 存储器空间的起始地址偏移，E2PROMx128_bE2Write() 例程对要写入到与块边界对齐的段中的数据进行解析。 此算法也需要使用 RAM 的最后 8 个字节： 0xF8 到 0xFF。

对于小于 128 字节的段，临时 128-byte 堆栈缓冲区由未写入的数据和经过修改的数据组成。 写入数据后，缓冲区从堆栈释放。 这是将未经修改的数据保留在块中所必需的操作。 对于跨整个块的 128 字节的段，不创建临时缓冲区。

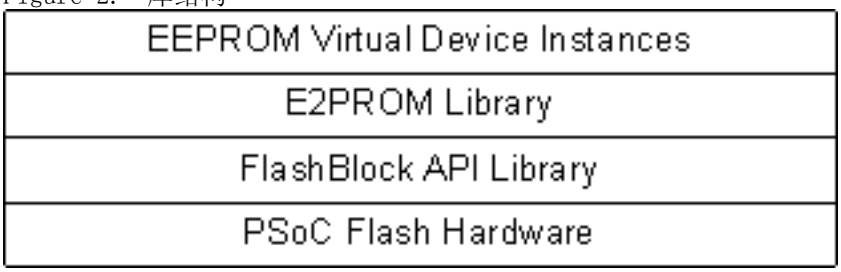
要通过优化参数降低系统开销，请参见上面的 *参数和资源* 一节。

软件组织

下图显示了如何将软件组织到由两个库和 N EEPROM 虚拟设备实例组成的一些层中。

上面的资源用法列出了闪存中的 API 库大小，以及 FlashBlock API 库和 E2PROMx128 库中包含的附加代码。 如果使用了此设备的多个实例，则资源计中仅计算 UserModule E2PROMx128 API 长度。 只有一个 FlashBlock API 库和 E2PROMx128 库用于任意数量的用户模块实例。

Figure 2. 库结构



FlashBlock API 库提供的 API 启用了基本闪存块的读写例程。 这些例程直接与硬件连接，使设备进入系统监控模式。 在此过程中，所有中断都被屏蔽。 对于实例化的 EEPROM 虚拟设备的所有实例，此库仅链接一次。

E2PROMx128 库通过块对齐和缓冲，将块操作转换为按位操作。 然后，它调用 FlashBlock API 库函数以写入闪存。 此库仅链接一次，与 EEPROM 虚拟设备数无关。

EEPROM 虚拟设备层提供简单方法以允许 EEPROM 设备的多个实例，而开销只是 E2PROMx128 库和 FlashBlock API 库的一个实例。 EEPROM 虚拟设备的每个实例包含此源代码的自定义副本，其最小值仅为 16 字节。

E2PROMx128 和 FlashBlock API 库都包含在库文件中。 如果对这些函数进行调用，则此文件自动链接到项目中。

Note 这些库不是重新参与者。

闪存写入周期寿命

闪存对生命周期写入总数有限制。 在相应的设备系列数据表中，可以找到此设备特定总数。

闪存保护

必须对 *flashsecurity.txt* 文件进行编辑，以允许闪存写入组成 EEPROM 设备的块。

定时

除非下表中指定，否则保证的所有限制为 T_A = 25° C，V_{dd} = 5.0V。

Table 1. 下表中汇总了 EEPROM 算法的定时：

参数	典型值	限制	单位	条件和注释
写入时间 (0° C-100° C)	45	120[3,4]	毫秒 / 块	写入时间取决于需要更新的闪存块的数量。[1,2]
写入时间 (-40° C-0° C)	70	240[3,4]	毫秒 / 块	写入时间取决于需要更新的闪存块的数量。[1,2]
读取时间	81 + 46N	--	CPU 时钟周期	N 是读取的字节数。

定时表说明

1. 当 M8C 处于系统监控模式下时，闪存读取操作期间会屏蔽中断。
2. 写入时间由用于擦除和写入每个闪存块的 FlashBlockWrite() 例程确定。
3. 闪存写入总数受限制。 有关每个块的写入周期数的说明，请参见部件特定数据表。
4. 为正常编程操作指定了限制。 在使用调试器的仿真模式下，写入脉冲可能花费 1 到 2 秒。

直流和交流电气特性

有关 EEPROM 的电子特征，请参见 PSoC 设备的设备数据表。

放置

EEPROMx128 用户模块是在软件中实现的，不需要放置。

参数和资源

FirstBlock

FirstBlock 是 EEPROM 设备驻留的起始块。 值的范围为从 0 到特定设备的最大闪存块数: 对于 32K 设备，为 255；对于 16K 设备，为 127；对于 8K 设备，为 63；对于 4K 设备，为 31；对于 2K 设备，为 15。

长度

长度定义 EEPROM 设备的大小（以字节为单位）。 有效值范围为从 1 到 N，其中 N 是 EEPROM 虚拟设备的大小 （以字节为单位）。

警告: 确保虚拟 EEPROM 设备的物理位置不超过设备闪存的大小。 EEPROM 设备的第一个物理字节位于第一个块 * 128。 虚拟设备的最后一个字节物理上位于（第一个块 + 长度 - 1）* 128。 不执行错误检查。 如果无法确保不超过该大小，可能导致闪存不可预见地写入非指定块中。

RAM 存储器开销

将使用下面的 RAM 存储器资源：

API 例程	存储器使用
E2PROMx128_E2Read()	堆栈空间： 8 字节 RAM 高存储器： 0xF8 - 0xFF
不包含部分块写入的 E2PROMx128_bE2Write()	堆栈空间： 32 字节 RAM 高存储器： 9 字节 0xF8 - 0xFF
包含部分块写入的 E2PROMx128_bE2Write()	堆栈空间： 103 字节 RAM 高存储器： 9 字节 0xF8 - 0xFF

存储器的高效使用

可以使用下面的指南最大程度地提高 RAM 和闪存资源的效率。

- 如果闪存非常 “ 充足 ” （高效使用 RAM）
  - 即使要存储的数据较少，也以 128-byte 长度的倍数创建 EEPROM 设备。
  - 始终以 128-byte 长度的倍数写入数据。
  - 始终以正确的字节数读取数据。

  例如，序列号数据的长度为 10 字节。 创建具有名称序列号的 E2PROMx128 用户模块。 将长度参数设置为 128。 这将在闪存中分配 128-byte 块。 在 RAM 中，设置 10-byte 数据阵列以保存序列号数据。 当通过 E2Write() 写入序列号时，将 wByteCount 参数指定为 128。 但是，当读取序列号数据时，应将 wByteCount 参数指定为 10。
- 如果闪存空间是主要考虑 （高效使用闪存）

  仅使用所需的字节数创建 EEPROM 设备。

  始终以正确的字节数写入数据。

  始终以正确的字节数读取数据。

应用程序编程接口

应用程序编程接口 (API) 例程作为用户模块的一部分提供，使设计者能够以较高级别处理模块。 本节指定每个函数的接口以及 “ 引用 ” 文件提供的相关常量。

每次放置用户模块时，都会为其分配一个实例名称。 默认情况下，PSoC Designer 向给定项目中此用户模块的第一个实例分配 E2PROMx128_1。 可将该值更改为符合标识符语法规则的任意唯一值。 分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。 在下面的说明中，为了简单起见，实例名称已缩写为 E2PROMx128。

Note ** 在此，像在所有用户模块 API 中一样，可以通过调用 API 函数更改 A 和 X 寄存器的值。 对于大型存储器模型中的所有 RAM 页指针寄存器，也是如此。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略旨在提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然某些用户模块 API 函数可能保持 A 和 X 不变，但是不保证它们将来也这样做。

本节描述了 EEPROMx128 用户模块 API。

E2PROMx128_Start

说明：

为了使用户模块 API 一致而维护的 null 函数。

C 原型：

```
void  E2PROMx128_Start(void)
```

汇编程序：

```
lcall  E2PROMx128_Start
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 对于 CY8C29xxx 中的大型存储器模型中的所有 RAM 页指针寄存器，也是如此。 如果需要，由调用函数负责在对 fastcall16 函数的调用中保留该值。

E2PROMx128_Stop

说明：

为了使用户模块 API 一致而维护的 null 函数。

C 原型：

```
void  E2PROMx128_Stop(void)
```

汇编程序：

```
lcall  E2PROMx128_Stop
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 对于 CY8C29xxx 中的大型存储器模型中的所有 RAM 页指针寄存器，也是如此。 如果需要，由调用函数负责在对 fastcall16 函数的调用中保留该值。

E2PROMx128_bE2Write

说明：

将指定的数据从 RAM 缓冲区写入定义的 EEPROM。 请注意，必须正确设置 *flashsecurity.txt* 文件才能写入闪存。

C 原型：

```
CHAR  E2PROMx128_bE2Write(WORD wAddr, BYTE *pbData, WORD wByteCount,
```

```
                    CHAR cTemperature);
```

汇编：

```
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
```

其中 <..> 指的是将参考数据放入累加器中所需的任何寻址模式或指令数。

参数：

wAddr： 从其写入 RAM 数据的 EEPROM 设备地址空间的地址偏移。 它可以为 0 到 N-1，其中 N 是 EEPROM 设备的长度。

pbData： 指向包含要写入的数据的 RAM 缓冲区的指针。

wByteCounter： 要写入闪存的字节数。

cTemperature： PSoC die 温度，以摄氏度表示。 可以使用下列选项之一指定此值：

- 用户模块，例如 FlashTemp。
- 外部设备或传感器。
- 适用于 PSoC 设备的所有环境条件的额定值。

  例如，室温 = 25° C。

返回值：

可以返回下列值：

返回标志	说明	值
E2PROMx128_NOERROR	成功完成操作。	0
E2PROMx128_FAILURE	未成功完成操作。 最可能是闪存保护位错误的结果。	-1
E2PROMx128_STACKOVERFLOW	堆栈空间不足以满足算法要求。	-2

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 对于 CY8C29xxx 中的大型存储器模型中的所有 RAM 页指针寄存器，也是如此。 如果需要，由调用函数负责在对 fastcall16 函数的调用中保留该值。 当前，仅修改 CUR_PP、IDX_PP 和 MVW_PP 页指针寄存器。

E2PROMx128_E2Read

说明：

将指定的 EEPROM 设备数据从闪存读取到指定的 RAM 缓冲区中。

C 原型：

```
void E2PROMx128_E2Read(WORD wAddr, BYTE *pbData, WORD wByteCount)
```

汇编程序：

```
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
```

其中 <..> 指的是将参考数据放入累加器中所需的任何寻址模式或指令数。

参数：

wAddr： 从其读取闪存数据的 EEPROM 设备地址空间的地址偏移。 它可以为 0 到 N-1，其中 N 是 EEPROM 设备的长度。

pbData： 指向数据读取到的 RAM 缓冲区。

wByteCounter： 要从闪存读取的字节数。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 对于 CY8C29xxx 中的大型存储器模型中的所有 RAM 页指针寄存器，也是如此。 如果需要，由调用函数负责在对 fastcall16 函数的调用中保留该值。 当前，仅修改 CUR_PP 和 MVW_PP 页指针寄存器。

固件源代码示例

下面是 C 语言源代码。 此示例代码是针对 CY7C64215 设备系列开发的。 此用户模块支持的其他设备系列在可用端口、引脚和可用驱动模式方面可能有所不同。 可能需要对此代码进行一些自定义，此代码才能在相应设备上运行。

```
//*****************************************************************************
//*    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)
```

```
    {
    }
}
```

汇编中的相同代码如下。

```
;***************************************************************************
;    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
```