

I²C 主器件 (I2C Master) 基本介绍 I2Cm V 1.4

Copyright © 2012 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC [®] 模块			API 存储器 (字节)		引脚 (每个外部 I/O)
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx, CY7C603xx, CY7C64215, CYWUSB6953, CY8C23x33, CY8CLED0xD, CY8CLED0xG, CY8CLED02/04/08/16, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12	0	0	0	791	4	2

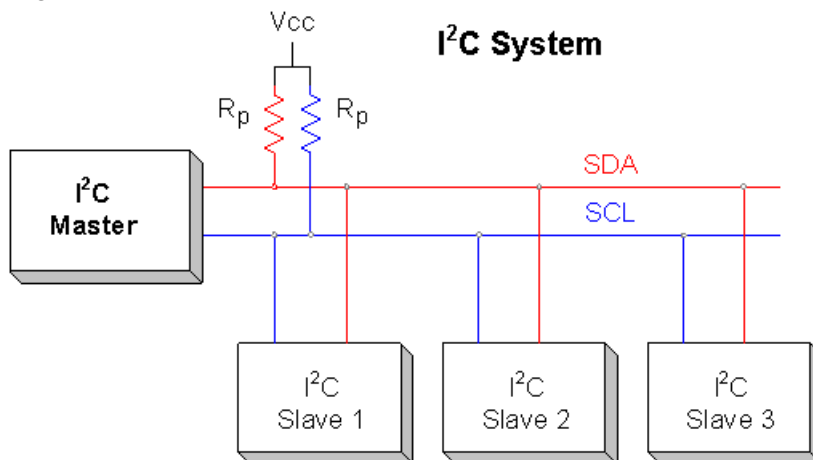
功能概述

- 与行业标准 Philips I²C 总线相兼容的接口
- 只需两个引脚 (SDA 和 SCL) 就可以与多个 I²C 从器件连接
- 标准模式数据支持的速率为 100 kbps
- 高级 API 只需少量用户编程
- 低级 API 提供灵活性

I²Cm 用户模块在固件中实现 I²C 主控器件。I²C 总线是 Philips[®] 开发的两线串行接口的行业标准。I²C 主控器件可以仅使用两条线路与数个从器件通信。在 I²C 总线上，主控启动所有通信，并为所有从器件提供时钟。I²Cm 用户模块支持的速率高达 100 kbps。此模块不需使用任何数字或模拟用户模块。

应用程序编程接口 (API) 固件通过单一的函数调用，可提供支持发送和接收多字节的高级命令。支持重复启动，但不支持多主控仲裁和 10 位寻址。

Figure 1. I²C 框图



功能描述

此用户模块在固件中实现 I²C 主控。当 CPU 时钟频率配置为 24 MHz 时，它能够支持高达 100 kbps 的数据传输速率。可以使用较慢的 CPU 时钟，但数据传输速率也会相应降低。I²C 规范允许主控以从 100 kHz 到低至 DC (直流电平) 的时钟频率运行。可在单个端口中选择任意两个引脚驱动 SDA 和 SCL 信号。

此模块无需任何模拟或数字 PSoC 模块，因此无需使用中断。当传输数据时，CPU 的利用率是 100%。由于 I²C 总线规范允许在标准模式下总线时钟可在直流 DC 至 100 kHz 之间运行，因此在传输过程中无需禁用后台中断。仅支持 7 位地址寻址模式。

上拉电阻 (R_p) 由供电电压、时钟频率和总线电容确定。输出阶段的任何器件 (主控或从器件) 的最小灌电流应不小于 3 mA (在 $V_{OLmax} = 0.4V$ 的条件下)。这会将 5V 系统的最小上拉电阻值限制为大约 1.5K ohms。上拉电阻 R_p 的最大值取决于总线电容和时钟频率。对于总线电容为 150 pF 的 5V 系统，上拉电阻不应大于 6K ohms。有关 “I²C 总线规范” 的更多信息，请参见 Philips 网站 www.philips.com。

I²C 地址包含在地址字节的前 7 位中。有效的选择范围为 0-127(dec)。该字节的最低有效位 LSB 标识 R/^W 位。如果该位为 0，那么向被寻址的从器件写入数据；如果 LSB 是 1，那么将从被寻址的从器件中读取数据。

在内部，用户模块将获取输入地址，移位并将其与读 / 写位组合成为完整的地址字节。

例如：地址 0x48 作为参数传递。包含读 / 写信息的参数将单独传递。I²C 主控将发送一个 0x90 字节 (8 位) 以向从器件写入数据，发送字节 0x91 以从从器件中读取数据。

向 I2Cm 用户模块使用的端口 PORT 直接写操作，将会干扰 I²C 操作。I2Cm 用户模块使用的端口将自动添加屏蔽寄存器。例如：如果此模块使用端口 1，那么名为 Port_1_DriveMode_0_SHADE 的屏蔽寄存器将添加至该工程。所有写入端口 1 的操作将使用此屏蔽寄存器完成。例如：如果 I2Cm 用户模块使用端口 1 的第 5 和 7 位，那么可对端口引脚 6 作如下修改：

C 代码：

```
Port_1_DriveMode_0_SHADE ^= 0x40;
PRT1DR = Port_1_DriveMode_0_SHADE;
```

汇编代码：

```
xor [Port_1_DriveMode_0_SHADE], 0x40
mov A, [Port_1_DriveMode_0_SHADE]
mov reg[PRT1DR], A
```

Note 从赛普拉斯或获得其分许可的一家联营公司处购买 I²C 组件，即可根据 Philips I²C 专利权获得一份许可，以便在 I²C 系统中使用这些组件，但前提是该系统符合 Philips 定义的 I²C 标准规范。

放置

I²C 主控用户模块使用一个端口的两个 I/O 引脚，无需数字或模拟 PSoC 模块。不存在放置限制。尽管单一 I2C 总线不支持多个主控模式，但单一工程中可以放置多个 I²C 主控模块。

参数和资源

I2C_Port

选择使用某个 PSoC I/O 端口接收 SDA 和 SCL 信号。

SDA_Pin

选择使用 I2C_Port 的某个引脚，作为 SDA 数据信号。无需为此引脚选择适当的驱动模式；PSoC Designer 将会自动完成这项选择。

SCL_Pin

选择使用 I2C_Port 的某个引脚作为 SCL 时钟信号。无需为此引脚选择适当的驱动模式；PSoC designer 将会自动完成这项选择。

应用程序编程接口

应用程序编程接口 (API) 程序作为用户模块的一部分提供，从而使设计人员能够在上层处理模块。本节详细规定每个函数的接口，以及由“引用(include)”文件所提供的相关常量。

Note

在这里，如同所有用户模块 API 中的一样，通过调用 API 函数，A 和 X 寄存器的值可能发生更改。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(registers are volatile)”策略可以提高效率，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。虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们一直不变。

提供的 API 函数支持用于传递 API 参数的两种不同方式。随着 C 编译器后续版本的出现，早期版本的参数传递方式已经过时。使用汇编代码和附带小型存储器模块设备的旧 API 调用结构的客户，只有在当他们应用更新为大型存储器模块设备时，才会感受到该变化带来的影响。新版本的应用可通过在汇编调用语句中添加一个下划线，来使用下列能够执行汇编语言的调用结构（新型参数传递）。C 语言编写的应用不会受到影响。适用于此描述的子程序为：I2Cm_fReadBytes、I2Cm_bWriteBytes 及 I2Cm_bWriteCBytes。

对于大型存储器模块设备，保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是由调用的程序负责。尽管部分寄存器现在可能不可修改，但是无法保证在将来的版本中也会如此。

I2Cm_Start

说明：

初始化 I/O 模式以及选定引脚的初始电平，实现 SDA 及 SCL 的功能。

C 语言原型：

```
void I2Cm_Start(void);
```

汇编程序：

```
lcall I2Cm_Start
```

参数：

None

返回值：

None

副作用：

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。当前，仅修改 CUR_PP 页面指针寄存器。

I2Cm_Stop

说明:

此函数不执行任何功能，为将来的兼容性而实施。

C 语言原型:

```
void I2Cm_Stop(void);
```

汇编程序:

```
lcall I2Cm_Stop
```

参数:

None

返回值:

None

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责通过调用 fastcall16 函数保持寄存器的值。

I2Cm_fReadBytes

说明:

从 I²C 从器件读取一个或更多的字节 (bCnt)，并将数据写入 pbXferData 指定的阵列。

C 语言原型:

```
Bool I2Cm_fReadBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

汇编程序:

```
mov    A,I2Cm_CompleteXfer    ; Pass complete transfer flag
push  A
mov    A,0x09                 ; Pass the byte count
push  A
mov    A,>sData                ; Load the MSB of the sData pointer
push  A
mov    A,<sData                ; Load the LSB of the sData pointer
push  A
mov    X,SP                   ; Get the stack location +1
dec    X                      ; X now points to the last byte pushed
mov    A,0x68                 ; Pass slave address 0x68
lcall  _I2Cm_fReadBytes       ; Call function to read data from slave
; Reg A contains return value.
add    sp,-4                  ; Restore the stack
```

参数:

bSlaveAddr: 7 位从器件地址。

pbXferData: 指向 RAM 中的数据阵列的指针。

bCnt: 需要读取的数据量。

bMode: 操作模式。如果设置成 I2Cm_CompleteXfer 模式，将执行完整传输。如果设置成 I2Cm_RepStart 模式，将会生成重复启动条件，而非单个启动条件。如果设置成 I2Cm_NoStop 模式，

将不会生成停止条件。这使 I²C 总线能够将需要发送的数据整合传输至从器件中。请参见本章节末的表格。

返回值:

如果传输未出现错误，将返回非零值。如果传输失败，将返回 0。

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。目前修改了 CUR_PP 和 IDX_PP 页面指针寄存器。

I2Cm_bWriteBytes

说明:

将 (pbXferData) 指定的 RAM 阵列中的一个或多个字节 (bCnt) 写入从器件的从地址 (bSlaveAddr) 中

C 语言原型:

```
BYTE I2Cm_bWriteBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

汇编程序:

```
mov    A,I2Cm_CompleteXfer    ; Pass complete transfer flag
push  A
mov    A,0x09                 ; Pass the byte count
push  A
mov    A,>sData                ; Load the MSB of the sData pointer
push  A
mov    A,<sData                ; Load the LSB of the sData pointer
push  A
mov    X,SP                   ; Get the stack location +1
dec    X                      ; X now points to the last byte pushed
mov    A,0x68                 ; Pass slave address 0x68
lcall  _I2Cm_bWriteBytes      ; Call function to write data to slave
                                           ; Reg A contains return value.
add    sp,-4                  ; Restore the stack
```

参数:

bSlaveAddr: 7 位从器件地址。

pbXferData: 指向 RAM 中数据阵列的指针。

bCnt: 要写入的数据量。

bMode: 操作模式。如果设置成 I2Cm_CompleteXfer 模式，将执行完整传输。如果设置成 I2Cm_RepStart 模式，将发送重复启动，而非单个启动条件。如果设置成 I2Cm_NoStop 模式，将不会发送停止条件。这使 I²C 总线能够将需要发送的数据整合传输至从器件中。请参见本章节末的表格。

返回值:

返回向从器件写入并确认的字节数。如果从器件确认没有字节写入成功，将返回 0。

副作用:

A 和 X 寄存器可能因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx), 所有 RAM 页面指针寄存器也会出现这种状况。如果需要, 调用函数负责通过调用 fastcall16 函数保存寄存器的值。目前修改了 CUR_PP 和 IDX_PP 页面指针寄存器。

I2Cm_bWriteCBytes**说明:**

从常量闪存阵列 (pbXferData) 向从器件的地址中 (bSlaveAddr) 写入一个或多个字节 (bCnt)。

C 语言原型:

```
BYTE I2Cm_bWriteCBytes(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

汇编程序:

```
mov    A,I2Cm_CompleteXfer    ; Pass complete transfer flag
push  A
mov    A,0x09                 ; Pass the byte count
push  A
mov    A,>sData                ; Load the MSB of the sData pointer
push  A
mov    A,<sData                ; Load the LSB of the sData pointer
push  A
mov    X,SP                   ; Get the stack location +1
dec   X                       ; X now points to the last byte pushed
mov    A,0x68                 ; Pass slave address 0x68
lcall  _I2Cm_bWriteCBytes     ; Call function to write data to slave
                                ; A contains return value.
add   sp,-4                   ; Restore the stack
```

参数:

bSlaveAddr: 7 位从器件地址。

pbXferData: 指向闪存中的“const”数据阵列的指针。

bCnt: 要写入的数据量。

bMode: 操作模式。如果设置成 I2Cm_CompleteXfer 模式, 将执行完整传输。如果设置成 I2Cm_RepStart 模式, 将生成重复启动条件, 而非单个启动条件。如果设置成 I2Cm_NoStop 模式, 将不会生成停止条件。这使 I²C 总线能够将需要发送的数据整合传输至从器件中。请参见本小节末的表格。

Note bMode 参数可用来执行 I²C 总线整合格式传输。要执行整合传输, 首先将 bMode 参数设置成 I2Cm_NoStop (0x02), 执行 I2Cm_bWriteBytes 或 I2Cm_bWriteCBytes 命令。这将执行写入, 且不会停止。然后, 将 bMode 参数设置成 I2Cm_RepStart (0x01), 执行 I2Cm_fReadBytes 命令。

返回值:

返回向从器件写入和确认的字节数。如果从器件确认没有字节成功写入, 将返回 0。返回值应与 bCnt 进行比较, 以确定是否已传输所有的字节。如果传输一个或多个字节时, 传输的字节数比 bCnt 指定的字节数少, 则返回非零值, 表示出现错误。

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx), 所有 RAM 页面指针寄存器也会出现这种状况。如果需要, 调用函数负责通过调用 fastcall16 函数保存寄存器的值。当前, 仅修改 CUR_PP 页面指针寄存器。

Table 1. 针对 I2Cm_bWriteBytes、I2Cm_bWriteCBytes 及 I2Cm_fReadBytes 的 bMode 常量

常量	值	说明
I2Cm_CompleteXfer	0x00	执行从“启动”到“停止”的完整传输
I2Cm_RepStart	0x01	发送重复启动而不是单个启动
I2Cm_NoStop	0x02	执行传输，而不产生停止条件。

API 低级功能

多数应用不需要低级 API 函数功能。低级 API 函数功能可以为特定的应用提供更高的灵活性。

Table 2. 低层 API 函数功能常量

常量	值	说明
I2Cm_WRITE	0x00	启动 I ² C 写入队列
I2Cm_READ	0x01	启动 I ² C 读取队列
I2Cm_ACKslave	0x01	读取一个字节时应答从器件
I2Cm_NAKslave	0x00	读取字节时非应答从器件

I2Cm_fSendStart

说明:

生成 I²C 总线启动条件，发送地址和读写 (R/W) 位，然后返回 应答 (ACK) 的结果。读写 R/W 位的值取决于 fRW 参数。

C 语言原型:

```
BYTE I2Cm_fSendStart( BYTE bSlaveAddr, BYTE fRW );
```

汇编程序:

```
mov    A,0x68                ; Load slave address
mov    X,I2Cm_WRITE          ; Prepare for a write sequence
lcall  I2Cm_fSendStart       ; Return value in A
```

参数:

bSlaveAddr: 7 位从器件地址。

fRW: 如果设置成 I2Cm_READ, 将启动读取队列。如果设置成 I2Cm_WRITE, 将启动写入队列。

返回值:

如果返回值非零, 则表示从器件已确认地址。如果返回值是零, 则表示从器件未应答地址。

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx), 所有 RAM 页面指针寄存器也会出现这种状况。如果需要, 调用函数负责通过调用 fastcall16 函数来保存寄存器的值。当前, 仅修改 CUR_PP 页面指针寄存器。

I2Cm_fSendRepeatStart

说明:

生成 I²C 总线重复启动条件, 发送地址和读写 (R/W) 位, 然后返回 应答 (ACK) 的结果。读写 R/W 位的值取决于 fRW 参数。

C 语言原型:

```
BYTE I2Cm_fSendRepeatStart( BYTE bSlaveAddr, BYTE fRW );
```

汇编程序:

```
mov    A, 0x68                ; Load address
mov    X, I2Cm_READ           ; Prepare for a read sequence
lcall  I2Cm_fSendRepeatStart   ; Return value in A
```

参数:

bSlaveAddr: 7 位从器件地址。

fRW: 如果设置成 I2Cm_READ, 将启动读取队列。如果设置成 I2Cm_WRITE, 将启动写入队列。

返回值:

如果返回值非零, 则表示从器件已确认地址。如果返回值是零, 则表示从器件未响应地址。

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx), 所有 RAM 页面指针寄存器也会出现这种状况。如果需要, 调用函数负责通过调用 fastcall16 函数来保存寄存器的值。当前, 仅修改 CUR_PP 页面指针寄存器。

I2Cm_SendStop

说明:

生成 I²C 总线停止条件。

C 语言原型:

```
void I2Cm_SendStop( void );
```

汇编程序:

```
lcall  I2Cm_SendStop          ; Generate I2C stop condition
```

参数:

None

返回值:

None

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx), 所有 RAM 页面指针寄存器也会出现这种状况。如果需要, 调用函数负责通过调用 fastcall16 函数来保存寄存器的值。当前, 仅修改 CUR_PP 页面指针寄存器。

I2Cm_fWrite

说明:

I²C 总线向从器件写入单一字节并产生应答 (ACK) 信号。此函数不会生成启动或停止条件。

C 语言原型:

```
BYTE I2Cm_fWrite( BYTE bData );
```

汇编程序:

```
mov    A, [bRamData]          ; Load data to send to slave
lcall  I2Cm_fWrite            ; Initiate I2C write
```

参数:

bData: 将要发送至从器件的字节。

返回值:

如果从器件已应答主控，则返回非零值。如果从器件未应答主控，则返回 0。

副作用:

A 和 X 寄存器可能会因为此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责通过调用 fastcall16 函数来保存寄存器的值。当前，仅修改 CUR_PP 页面指针寄存器。

*I2Cm_bRead***说明:**

启动 I²C 总线读取从设备的一个字节和产生应答信号 (ACK) 阶段。此函数不会生成启动或停止条件。应答标志 (fACK) 决定从器件在收到数据后是否应答。

C 语言原型:

```
BYTE I2Cm_bRead( BYTE fACK );
```

汇编程序:

```
mov    A, I2Cm_ACKslave      ; Set flag to ACK slave
lcall  I2Cm_bRead            ; Read single byte from slave
; Return data is in reg A
```

参数:

fACK: 如果主控在收到数据后应向从器件发送应答信号 ACK，请设置成 I2Cm_ACKslave；否则，将标志设置成 I2Cm_NAKslave。

返回值:

从从器件接收的字节。

副作用:

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责通过调用 fastcall16 函数来保存寄存器值。当前，仅修改 CUR_PP 页面指针寄存器。

固件源代码示例

以下是汇编语言和 C 语言的 I²C 器件通信的示例。

```

;-----
; Sample assembly code to communicate with the Dallas DS1307 clock/EEROM
; via the I2C interface.
;
; This code sets the time, then reads the 7-byte time string back
; over and over in a loop.
; The address of the DS1307 is 0x68.
;-----
include "m8c.inc"          ; part specific constants and macros
include "PSoCAPI.inc"     ; PSoC API definitions for all User Modules

export _main

area bss (RAM)
sRxData:   blk 16
area text (ROM,REL)

.LITERAL
sTxData:
  db    0x00                ; Slave internal address 0
      db    0x12,0x34,0x08    ; Seconds and minutes in BCD 8:34:12am
      db    0x01                ; Day of Week, Monday
  db    0x15,0x03,0x02      ; Day-Month-year 15-Mar-02
      db    0x93                ; Enable clock output
.ENDLITERAL

_main:

  call  I2Cm_Start          ; Initialize I2C master

  mov   A,I2Cm_CompleteXfer ; Pass normal transfer mode
  push  A
  mov   A,0x09              ; Pass all 9 bytes of sTxData
  push  A
  mov   A,>sTxData           ; Load the MSB of the sTxData pointer
  push  A
  mov   A,<sTxData           ; Load the LSB of the sTxData pointer
  push  A
  mov   X,SP                 ; Get the stack location +1
  dec   X                   ; That points to the last byte pushed
  mov   A,0x68              ; Pass slave address 0x68
  call  _I2Cm_bWriteCBytes  ; Call function to write data to slave
                          ; A contains return value.
  add   SP,-4               ; Restore the stack

RxLoop:                      ; Keep reading the time from the
                          ; Dallas DS1307

  ; Do a combined transfer, write then read
; The write sets the sub-address value to 0x00. The read then
; starts reading the values starting at the sub-address 0x00.
;

```

```

mov    A,I2Cm_NoStop           ; Don't generate a stop sequence
push  A
mov    A,01h                   ; Write only the first byte of the string
push  A                         ; which is the internal sub-address.
mov    A,>sTxData               ; Load the MSB of the sTxData pointer
push  A
mov    A,<sTxData               ; Load the LSB of the sTxData pointer
push  A
mov    X,SP                     ; Get the stack location +1
dec    X                       ; Dec the pointer to point to the last byte
mov    A,0x68                   ; Pass slave address 0x68
call   _I2Cm_bWriteCBytes      ; Call function to write data to slave
                                           ; Reg A contains return value.
add    SP,-4                   ; Restore the stack

mov    A,I2Cm_RepStart         ; Start with a Repeat start
push  A
mov    A,0x07                   ; Read just the 7 time bytes back
push  A
mov    A,>sRxData               ; Load the MSB of the sRxData pointer
push  A
mov    A,<sRxData               ; Load the LSB of the sRsData pointer
push  A
mov    X,SP                     ; Get the stack location +1
dec    X                       ; That points to the last byte pushed
mov    A,0x68                   ; Pass slave address 0x68
call   _I2Cm_fReadBytes       ; Call function to read data from slave
                                           ; A contains return value.
add    SP,-4                   ; Restore the stack

jmp    RxLoop                   ; Setting a breakpoint here, you should
                                           ; be able to see the returned time data
                                           ; in the sRxData RAM locations.

ret

```

C 语言示例代码如下。

```

//-----
// Sample C code to communicate with the Dallas DS1307 clock/EEROM
// via the I2C interface.
//
// This code sets the time, then reads the 7-byte time string back
// over and over in a loop.
// The address of the DS1307 is 0x68.
//-----

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

BYTE rxBuf[8];

const BYTE txCBuf[] = { 0x00,           // Slave internal sub-address 0
                        0x12,0x34,0x08, // Seconds and minutes in BCD
                                           // 8:34:12am
                        0x01,           // Day of Week, Monday
                        0x15,0x03,0x02, // Day-Month-year 15-Mar-02

```

```

        0x93 };          // Enable clock output

void main(void)
{
    BYTE status;          // I2C communication status
    BYTE i;              // Temp counter variable

    I2Cm_Start();        // Initialize I2C Master interface

    // Set the time
    status = I2Cm_bWriteCBytes(0x68,txCBuf,9,I2Cm_CompleteXfer);

    // In a endless loop, keep reading the time from the DS1307
    do {

        // Write sub-address to DS1307
        // Perform a combined transfer by leaving off the Stop on the Write
        // command and beginning the Read with a Repeat Start.

        I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop );

        status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart );

        if(status == 0) {
            // Flag an error condition
            }

        // This next section of code performs exactly the same sequence
        // as the above code, but with low level commands.

        I2Cm_fSendStart(0x68,I2Cm_WRITE);          // Do a write
        I2Cm_fWrite(0x00);                          // Set sub address
                                                    // to zero
        I2Cm_fSendRepeatStart(0x68,I2Cm_READ);     // Do a read

        for(i = 0; i < 6; i++) {
            rxBuf[i] = I2Cm_bRead(I2Cm_ACKslave); // Read first 6 bytes,
                                                    // and ACK the slave
        }

        rxBuf[7] = I2Cm_bRead(I2Cm_NAKslave);      // Read data byte and
                                                    // NAK the slave to signify
                                                    // end of read.

        I2Cm_SendStop();

    } while(1);
}

```

版本历史记录

版本	创作者	说明
1. 4	DHA	添加了版本历史

Note PSoC Designer 5.1 在所有用户模块数据手册中都引入了“版本历史”。本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2012 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.