Easy I^2C 从器件数据手册 EzI2Cs V 1.2

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

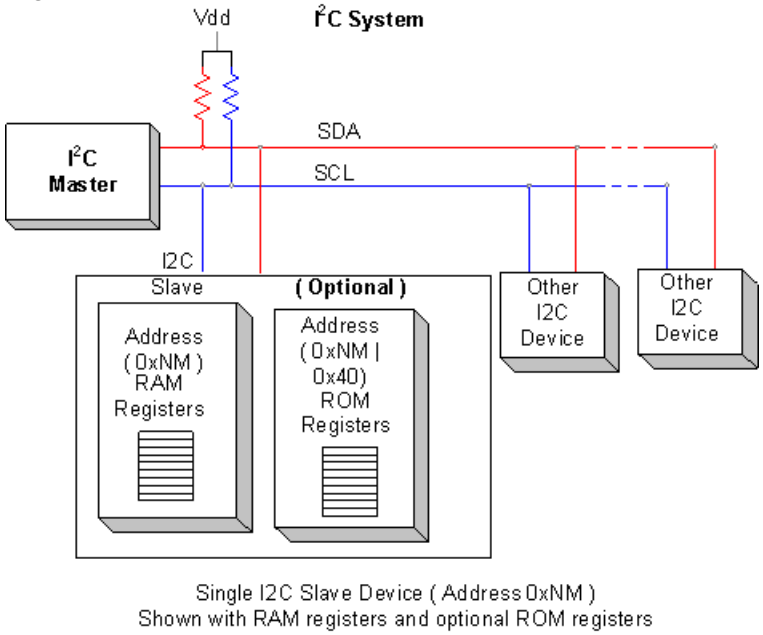
资源	PSoC® 模块			API 内存 （字节）		引脚 （根据外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx、CY8C23x33、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28xxx、 CY8C21x12						
RAM 读 / 写缓冲区	0	0	0	315	6	2
增加了 ROM 缓冲区支持	0	0	0	439	6	2

功能和概述

- 行业标准 Philips I^2C 总线兼容接口
- 仿真通用 I^2C EEPROM 接口
- 只有两个引脚 （SDA 和 SCL）需要与 I^2C 总线连接
- 100/400 kbps 标准数据速率
- 高级 API 只需少量用户编程

EzI2Cs 用户模块实现基于 I^2C 寄存器的从器件。 I^2C 总线是由 Philips® 开发的基于行业标准的两线硬件接口。 主控在 I^2C 总线上启动所有通信，并为所有从器件提供时钟。 EzI2Cs 用户模块支持最大速度达 400 kbps 的标准模式。 此模块不使用数字或模拟 PSoC 模块。 EzI2Cs 用户模块与同一总线上的多个器件兼容。

Figure 1. I²C 框图



功能说明

EzI2Cs 用户模块采用不同的方法，使用随 PSoC Designer 提供的 I2CHW 用户模块。 此用户模块仅支持具有一个或两个 I²C 地址的 I²C 从器件配置。 第一个地址始终为 RAM 分配的区域，可选的第二个地址将访问 ROM 区域。 第二个地址为用 0x40 进行或运算的 RAM 区域。 例如，如果选择地址 0x05，则 RAM 区域地址为 0x05，可选的第二个地址为 0x45。 这两个地址都是右对齐。 RAM 和 ROM 区域都可以具有 1 到 255 字节的数据结构 （字节、整数、数组和结构）。

由于 I²C 硬件是中断驱动的，此用户模块要求您启用全局中断。 即使此用户模块要求中断，您也不需要向 ISR （中断服务子程序）添加任何代码。 该模块为所有中断 （数据传输）提供服务，与您的代码无关。 为此接口分配的存储器缓冲区看上去类似于您的应用程序与 I²C 主控之间的简单双端口存储器。

如果需要，可以通过在数据结构中定义信号和命令位置，在主控与此从器件之间创建更高级别接口。

RAM 区域接口

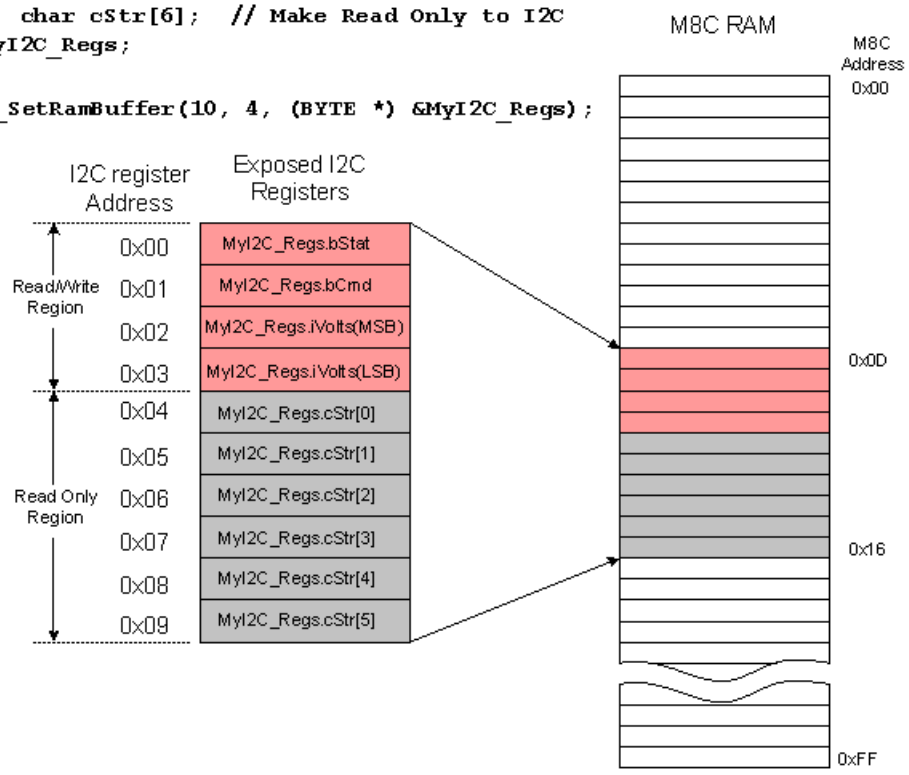
对于 I²C 主控，该接口看上去类似于通用 256 字节 I²C EEPROM。 可以将 PSoC API 视为可配置为简单变量、数组或结构的 RAM。 在某种意义上，它作为您的程序与 I²C 总线上的 I²C 主控之间的共享 RAM 接口。 API 允许用户将任何数据结构公开给 I²C 主控。 用户模块只允许 I²C 主控访问 RAM 的指定区域，阻止该区域外的任何读写。 公开给 I²C 接口的数据可以是单一变量、值的数组或结构。 初始化时所需的只是指向变量或数据结构起点的指针。 请参见下图。

[+] Feedback

Figure 2. RAM 区域接口

```
struct I2C_Regs {
    BYTE bStat;
    BYTE bCmd;
    int  iVolts;
    char cStr[6];  // Make Read Only to I2C
}MyI2C_Regs;

I2C_SetRamBuffer(10, 4, (BYTE *) &MyI2C_Regs);
```



例如，您可以创建下列结构：

```
struct I2C_Regs {    // Example I2C interface structure
    BYTE bStat;
    BYTE bCmd;
    int  iVolts;
    char cStr[6];  // Read only string
} MyI2C_Regs;
```

此结构可以包含具有任何名称的任何变量组，前提是该变量组在存储器中是连续的且被指针引用。 该接口（I^2C 主控）仅将其视为字节数组，不能访问已定义区域外的任何存储器。 使用上述结构示例，提供的 API 用于将数据结构公开给 I^2C 接口。 第一个参数设置公开给 I^2C 接口的 RAM 的大小，在此例中，它是整个结构。 第二个参数通过设置读 / 写区域中的字节数，设置读 / 写区域和只读区域之间的边界。 读 / 写区域在最前面，后面跟随只读区域。 在此例中，只能写入前 4 个字节，但是 I^2C 主控可以读取所有字节。 第三个参数是指向数据的指针。

```
EzI2Cs_SetRamBuffer(sizeof(MyI2C_Regs), 4,(BYTE *) &MyI2C_Regs);
```

在下例中，创建一个 15 字节数组，该数组公开给 I^2C 接口。 该数组的前 8 字节为读 / 写型，其余 7 字节为只读型。

```
char theArray[15];
EzI2Cs_SetRamBuffer(15, 8, (BYTE *) theArray);
```


[+] Feedback

下面的示例是仅公开一个整数 （2 字节）的非常简单的示例。 这两个字节都可由 I^2C 主控读写。

```
int iRegister;
EzI2Cs_SetRamBuffer(2, 2, (BYTE *) (&iRegister);
```

ROM 区域接口 （可选）

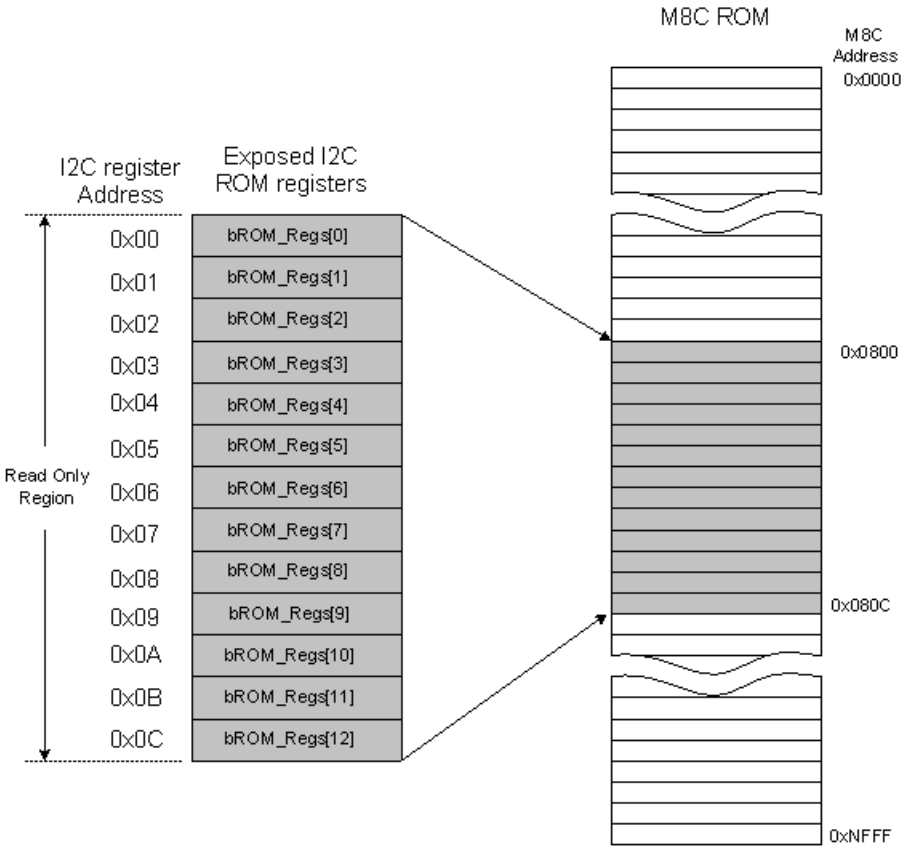
ROM 接口几乎等同于 RAM 接口，只不过它是只读的。 如下例所示，SetRomBuffer 函数类似于 SetRamBuffer 函数。 在此例中，数据区域是简单的常量字符串，长度为 13 字节 （包括结束的 NULL (0x00) 字符）。

```
Const BYTE bROM_Regs[13] = "I2C ROM AREA";

EZI2Cs_SetRomBuffer(13, (const BYTE *)bROM_Regs);
```

如上所示，ROM 区域有其自己的 I^2C 地址。 此地址只是用 0x40 进行或运算的 RAM 区域地址。 如果主 RAM 区域地址为 0x05，则用于访问 ROM 区域的 I^2C 地址将为 0x45。 下图显示此例在存储器中的可能显示方式。

Figure 3. ROM 区域接口



[+] Feedback

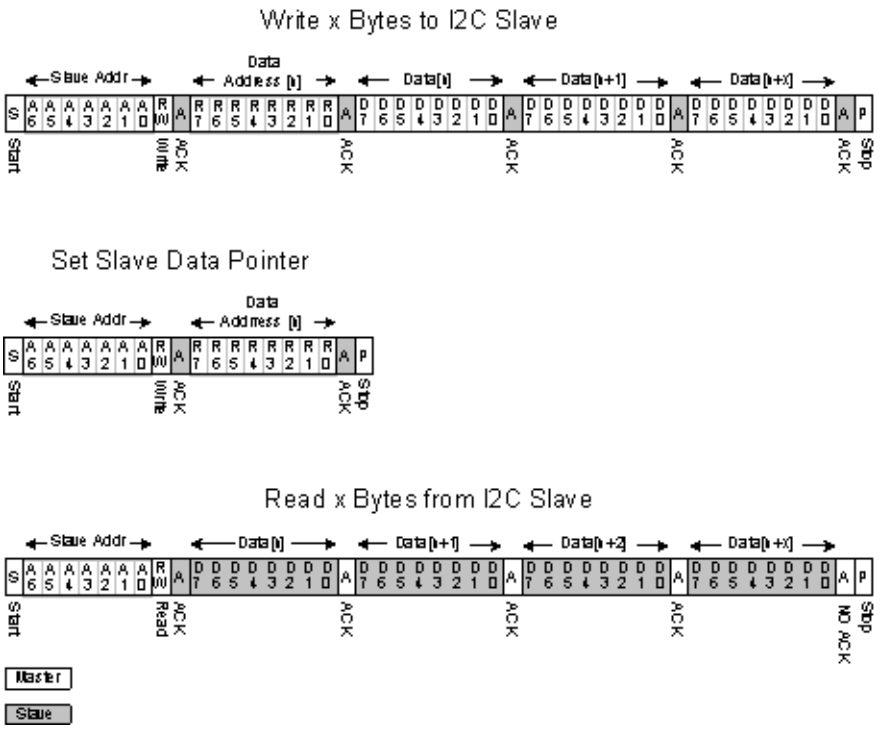
外部主控可视的接口

EzI2Cs 用户模块支持 RAM 区域的基本读写操作和 ROM 区域的只读操作。 RAM 和 ROM 区域接口包含使用写入操作的第一个数据字节设置的单独数据指针。 甚至 ROM 区域也可以接受写入单一字节来设置数据指针。 当写入一个或多个 RAM 字节时，第一个数据字节始终是数据指针。 数据指针后的字节写入到数据指针字节指向的位置。 第三个字节 （第二个数据字节）写入数据指针加 1 的位置，依此类推。 此数据指针按每个读取或写入的字节递增，但是在每个新读取操作开始时复位为第一个写入的值。 新读取操作开始在数据指针指向的位置读取数据。

例如，如果数据指针设置为 4，则读取操作开始在位置 4 读取数据，并按顺序继续读取，直到达到数据末尾或主机完成读取操作为止。 例如，如果数据指针设置为 4，则每个读取操作将数据指针复位为 4，并从该位置按顺序读取。 无论执行单一读取操作还是执行多个读取操作，都是如此。 数据指针在启动新写入操作之前不会更改。

如果 I^2C 主控尝试跨过 SetRamBuffer() 函数指定的区域写入数据，则该数据被丢弃，不会影响指定 RAM 区域内外的任何 RAM。 不能在允许的范围外读取数据。 主控在允许范围外的任何读取请求都会导致返回无效结果。

下图说明了数据写入、数据指针写入和数据读取操作的总线通信。 请记住，数据写入操作始终重新编写数据指针。



在复位或加电时，将配置 EzI2Cs 用户模块并提供 API，但是必须使用 EzI2Cs_Start() 函数明确启用资源。

I^2C 资源支持逐字节级别的数据传输。 在每个地址或数据发送 / 接收末尾，将报告状态，或者可能触发专门的中断。 状态报告和中断生成取决于硬件检测到的数据传输方向和 I^2C 总线条件。 中断可以配置为在字节完成或总线错误检测时发生。

每个 I^2C 数据操作由起点、地址、读 / 写方向、数据和终点组成。 对于发往的 I^2C 从器件，中断在传入数据的第 8 个字节后发生。 如果发往总线上的另一个从器件，则该地址放置在总线上后只有一个中断。 在此位置点，接收器必须决定是确认 （ack） 还是不确认 （nak） 传入字节 （无论它是地址还是数据）。 接

[+] Feedback

收器然后将适当的控制位写入 I2C_SCR 寄存器，将 ack/nak 状态通知给 I^2C 资源。 通过安装总线、在总线上提供 ack/nak 状态和将下一个数据字节移入，写入 I2C_SCR 寄存器的操作对总线上的数据流进行步进操作。 对于发送器的第二种情况，在外部接收器提供 ack 或 nak 状态后发生中断。 可以读取 I2C_SCR 以确定此位的状态。 对于发送器，数据加载到 I2C_DR 寄存器中，再次写入 I2C_SCR 寄存器以触发下一部分的传输。

在 Philips 网站上提供的完整 I^2C 规范中，以及通过参考随 PSoC Designer 提供的器件数据手册，可以获得 I^2C 总线的详细说明和此处的实现。

设计考虑事项

将主振荡器设置为任意时钟速度。 数据吞吐量可能受处理器速度的限制，但逐字节数据传输以指定的 I^2C 速度运行。 如果使用其他生成经常性中断的用户模块 （例如极高速模拟 / 数字转换器），则可能需要提高主振荡器时钟速度 （或降低其他用户模块的速度）来维护通过 I^2C 的通信。I^2C 中断的优先级比大多数其他中断低，因此，如果另一个用户模块生成中断的速度过快，则可能出现永不或很少调用 I^2C 中断子程序的情况。

动态重新配置

不要将 EzI2Cs 资源合并到动态加载或卸载的叠层。 仅将 EzI2Cs 资源作为基本配置的一部分放置。 可以根据操作需要所述修改 EzI2Cs 模块的操作，但是尝试删除作为动态重新配置一部分的资源会对外部 I^2C 器件产生负面影响。 还有可能意外地重新配置所选引脚以禁用 I^2C 函数。 当在配置之间切换时，一定要注意避免这种可能性。

外部电气连接

如框图所示，I^2C 总线需要外部上拉电阻。 上拉电阻 (R_P) 由供电电压、时钟速度和总线电容确定。 将输出阶段的任何器件 （主控或从器件）的最小灌电流设置为不超过 3 mA （在 AV_{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 1.5 kΩ。 R_P 的最大值取决于总线电容和时钟速度。 对于总线电容为 150 pF 的 5V 系统，上拉电阻不大于 6 kΩ。 有关 “I^2C 总线规范 ” 的更多信息，请参见 Philips 网站 www.philips.com。

Note 从赛普拉斯或其获得分许可的其中一个联营公司处购买 I^2C 组件，即可根据 Philips I^2C 专利权获得一份许可，以便在 I^2C 系统中使用这些组件，但前提是该系统符合 Philips 定义的 I^2C 标准规范。

直流和交流电气特性

有关 I^2C 接口的电气特征，请参见您的 PSoC 器件的器件数据手册。

放置

EzI2Cs 用户模块不需要任何数字或模拟 PSoC 模块。 不存在放置限制。 放置多个 I^2C 模块是不可能的，因为 I^2C 模块使用专用 PSoC 资源模块和中断。

[+] Feedback

参数和资源

Slave_Addr

选择 I^2C 主控使用的 7-bit 从器件地址可对从器件寻址。 如果仅使用 RAM 寄存器，则有效选择为 0x00 到 0x7F。 如果启用 ROM 寄存器，则只有 0x00 和 0x3F 之间的地址有效，这是因为 ROM 寄存器空间为 RAM 地址加上 0x40 （0x40 到 0x7F）。

Auto_Addr_Check

选择是禁用还是启用硬件地址功能。 如果启用，则硬件地址比较功能将不可用。 将此选项设置为启用意味着 I2C 模块不支持特殊系统地址定义。

此参数是用户仅可针对 CY8C28045 器件配置的参数。

Address_Type

选择地址是静态地址还是动态地址。 如果设置为静态，则不能更改 I^2C 地址，但是可以保存一个字节的 RAM。 将此选项设置为动态则允许固件随时更改地址。 此选项通常适合使用外部电阻或双列直插开关设置 I^2C 地址的 LSB 的应用场合。

ROM_Registers

始终启用 RAM 区域，但是您可以启用另一个地址以允许您访问 ROM 区域中的常量。 如果需要 ROM 地址，请选择此参数的启用选项；如果不需要，则选择禁用。第二个地址是用 0x40 进行或运算的基地址。

CPU_CLK_SPEED_CY8C27xxxA

此参数修复了早期 CY8C27xxxA 系列部件中的错误。 对于 CY8C27xxxB 系列部件和支持硬件 I^2C 的所有其他系列部件，已修复了这种情况。如果 CPU 在 6 MHz 或之下运行，请选择 **6MHz 或更少**选项。如果 CPU 时钟在 6 MHz 以上运行，请选择**高于 6MHz（仅限 CY8C27xxxA）**选项。 对于除 CY8C27xxxA 部件之外的所有部件，请选择**非 CY8C27xxxA** 选项。

对于解决了此错误的产品系列，不显示此参数。

I2C_Clock

此参数选择用于此从器件的时钟速度。 选项有 50 kHz、100 kHz 或 400 kHz。 如果 IMO (SysClk) 为 24 MHz，则只能选择 400 kHz I^2C 速度。

I2C_Pin

从端口 1 选择用于 I^2C 信号的引脚。 无需为这些引脚选择适当的驱动模式，PSoC Designer 会自动完成这项选择。 这使您可以将 I^2C 时钟和数据信号放在 P1[5] - P1[7] 或 P1[1] - P1[0] 上。

中断生成控制

当在 PSoC Designer 中选中 **启用中断生成控制** 复选框时，会有一个附加参数变为可用。 可以通过依次单击以下菜单获得此参数： **项目 > 设置 > 芯片编辑器**. 当多个外覆层用于由多个用户模块跨外覆层共享的中断时，中断生成控制非常重要：

IntDispatchMode

IntDispatchMode 参数用于指定如何为位于同一模块但在不同外覆层中的多个用户模块共享的中断来处理中断请求。 选择 “ActiveStatus” 会导致固件在处理共享的中断请求之前测试哪一个外覆层正处于活动状态。 每当请求共享中断时，都会发生此测试。 这会增加延迟，还会生成为共享中断请求提供服务的不确定过程，但是不需要任何 RAM。 选择 “OffsetPreCalc” 会导致固件仅当初始加载外覆层时计算共享中断请求的源。 此计算减少了中断延迟，并生成为共享中断请求提供服务的确定过程，但是耗费了一字节 RAM。

[+] Feedback

应用程序编程接口

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 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 ” 策略是为了提高效率，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，但是无法保证它们将来也会如此。

API 变量

要保证长度为 2 或更多字节的变量的数据完整性，请在代码中更改 EzI2Cs_bBusy_Flag 变量之前对这些变量进行检查，例如：

```
if(!(EzI2Cs_bBusy_Flag == EzI2Cs_I2C_BUSY_RAM_READ))
{
data.wData1++; //safely increment some 2 byte variable
}
```

EzI2Cs_bBusy_Flag 变量在 ISR 中自动更新，并具有下列预定义值：

Table 1. EzI2Cs_bBusy_Flag 变量的预定义值

符号名称	值	说明
I2C_FREE	0x00	当前没有数据操作
I2C_BUSY_RAM_READ	0x01	正在进行 RAM 读取数据操作
I2C_BUSY_RAM_WRITE	0x03	正在进行 RAM 写入数据操作
I2C_BUSY_ROM_READ	0x01	正在进行 ROM 读取数据操作
I2C_BUSY_ROM_WRITE	0x03	正在进行 ROM 写入数据操作

API 函数

EzI2Cs_Start

说明：

启用 I^2C 从器件并启用中断。 在调用此函数之前调用所需的 EzI2Cs_SetRamBuffer() 函数。

C 原型：

```
void  EzI2Cs_Start(void);
```

汇编程序：

```
lcall EzI2Cs_Start
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

[+] Feedback

EzI2Cs_SetAddr

说明：

此函数设置 EzI2Cs 用户模块识别的地址。 仅当 Address_Type 在参数中设置为 “ 动态 ” 时，此命令才有效。 仅在 EzI2Cs_Start() 函数后调用此函数。 这是因为启动函数设置了默认地址，此函数将会覆盖该地址。 此函数是可选的，只有当您更改默认地址，使其不再是默认地址时，才需要此函数。

C 原型：

```
void  EzI2Cs_SetAddr(BYTE bAddr);
```

汇编程序：

```
mov    A,0x05             ; Set I2C slave address to 0x05
lcall  EzI2Cs_SetAddr
```

参数：

BYTE char： 介于 1 和 127 之间的从器件地址。

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_GetActivity

说明：

如果自上次调用此函数后发生了 I^2C 读取或写入循环，则此函数返回非零值。 在此函数调用结束时，活动标志将复位为零。

C 原型：

```
BYTE EzI2Cs_GetActivity(void);
```

汇编程序：

```
lcall  EzI2Cs_GetActivity     ; Return value in A
```

参数：

无。

返回值：

如果执行了 I^2C 读取或写入，则返回非零值。 如果自上次调用该函数以来未执行任何活动，则返回零。

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_GetAddr

说明：

此函数返回该 I^2C 从器件的当前地址。 此函数是可选的，通常仅当应用程序必须确定当前或默认地址时才需要此函数。

[+] Feedback

C 原型：

```
BYTE EzI2Cs_GetAddr(void);
```

汇编程序：

```
lcall  EzI2Cs_GetAddr     ; Return value in A
```

参数：

无。

返回值：

此 I^2C 从器件的当前 I^2C 地址

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_Stop

说明：

通过禁用 I^2C 中断来禁用 EzI2Cs。

C 原型：

```
void   EzI2Cs_Stop(void);
```

汇编程序：

```
lcall  EzI2Cs_Stop
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_DisableSlave

说明：

禁用 EzI2Cs，而不禁用 I^2C 中断。

C 原型：

```
void   EzI2Cs_DisableSlave(void);
```

汇编程序：

```
lcall  EzI2Cs_DisableSlave
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_EnableInt

说明：

启用 I^2C 中断以便实现启动条件检测。 请牢记要通过以下宏来调用这个全局中断启用函数：M8C_EnableGInt. 如果调用 EzI2Cs_Start 函数，则不需要此函数。 默认情况下，此函数会清除待处理中断。 若要避免清除待处理中断，请参见 ResumeInt 函数。

C 原型：

```
void   EzI2Cs_EnableInt(void);
```

汇编程序：

```
lcall  EzI2Cs_EnableInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_ResumeInt

说明：

启用 I^2C 中断以便实现启动条件检测。 此函数在未启用中断的情况下不会清除待处理中断。 请牢记要通过以下宏来调用这个全局中断启用函数： M8C_EnableGInt. 如果调用 EzI2Cs_Start 函数，则不需要此函数。

C 原型：

```
void   EzI2Cs_ResumeInt(void);
```

汇编程序：

```
lcall  EzI2Cs_ResumeInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_DisableInt

说明：

通过禁用 SDA 中断来禁用 I^2C 的从器件。 执行的操作与 EzI2Cs_Stop. 相同

[+] Feedback

C 原型：

```
void   EzI2Cs_DisableInt(void);
```

汇编程序：

```
lcall EzI2Cs_DisableInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

EzI2Cs_SetRamBuffer

说明：

此函数设置向 I^2C 主控公开的 RAM 缓冲区 （最多包含 255 个字节）的位置和大小。 此函数是必需的，应在调用 EzI2Cs_Start(). 之前调用

C 原型：

```
void EzI2Cs_SetRamBuffer(BYTE bSize, BYTE bRWboundry, (BYTE *)pAddr);
```

汇编程序：

```
lcall  EzI2Cs_SetRamBuffer
```

参数：

BYTE bSize： 向 I^2C 主控公开的数据结构的大小。 bSize 的最小值为 1，最大值为 255。 BYTE bRWboundry： 从第一个位置开始算起的可写位置的大小。 该值必须始终小于或等于 bSize 才能正常工作。 BYTE * pAddr： 一个指向数据结构的指针。

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。 如果 bRWboundry 设置为大于 bSize 的值，则整个 bRWboundry 大小将可写。 如果将 bSize 设置为无效值 0，则可允许 I^2C 主机写入未定义的存储器位置。

EzI2Cs_SetRomBuffer

说明：

此函数设置向 I^2C 主控公开的 ROM 缓冲区（最多包含 255 个字节）的位置和大小。 如果需要使用 ROM，则此函数是必需的，应在启用 EzI2Cs_Start 函数之前调用。

C 原型：

```
void EzI2Cs_SetRomBuffer(BYTE bSize,(const BYTE *)pAddr);
```

汇编程序：

```
lcall  EzI2Cs_SetRomBuffer
```


[+] Feedback

参数：

BYTE bSize： 向 I^2C 主控公开的数据结构的大小。常量 BYTE * pAddr： 一个指向闪存中的数据结构的指针。

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

固件源代码示例

C 语言代码示例

```
//************************************************************
// Example code to demonstrate the use of the EzI2Cs UM
//
// This code sets up the EzI2Cs slave to expose both
// a RAM and a ROM data structure.  The RAM structure is
// addressed at I2C address 0x05, and the ROM structure
// is at I2C address 0x45.
//
// * The instance name of the EzI2Cs User Module is "EzI2Cs".
// * The "Address_Type" parameter is set to "Dynamic"
// * The "ROM_Registers" parameter is set to "Enable"
//
// NOTE: to guarantee data integrity of variables with length 2 or more
// bytes check the EzI2Cs_bBusy_Flag variable before changing values of such
// variables.
//************************************************************

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

struct I2C_Regs {    // Example I2C interface structure
    BYTE bStat;
    BYTE bCmd;
    int  iVolts;
    char cStr[6];    // Read only string
} MyI2C_Regs;

const BYTE DESC[] = "Hello I2C Master";

void main(void)
{

    // Set up RAM buffer
   EzI2Cs_SetRamBuffer(sizeof(MyI2C_Regs), 4, (BYTE *) &MyI2C_Regs);


   EzI2Cs_SetRomBuffer(sizeof(DESC), DESC);   // Set up ROM buffer

   M8C_EnableGInt ;                           // Turn on interrupts

   EzI2Cs_Start();                            // Turn on I2C
```


[+] Feedback

```
   EzI2Cs_SetAddr(5);                          // Change address to 5


   while(1) {
      // Place user code here to update and read structure data.
   }
}
```

ASM 代码示例

```
;-------------------------------------------------------------------------
; Example code to demonstrate the use of the ExI2Cs UM
;
; This code sets up the EzI2Cs slave to expose both
; a RAM and a ROM data structure.  The RAM structure is
; addressed at I2C address 0x05, and the ROM structure
; is at I2C address 0x45.
;
; * The instance name of the EzI2Cs User Module is "EzI2Cs".
; * The "Address_Type" parameter is set to "Dynamic".
; * The "ROM_Registers" parameter is set to "Enable".
;
; NOTE: to guarantee data integrity of variables with length 2 or more
; bytes check the EzI2Cs_bBusy_Flag variable before changing values of such
; variables.
;-------------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

area bss (RAM, REL, CON)        ; Allocate Variables & define constants
RAM_BUF_SIZE:     equ  10       ; Top of ramp value
BUF_RW_SIZE:      equ   5       ; Only first five bytes writable from I2C Master

I2C_RAM_Buf:  blk  RAM_BUF_SIZE ; Reserve RAM for I2C buffer

area text (ROM, REL, CON)

.LITERAL
I2C_ROM_BUF:                    ; I2C ROM Buffer
   DB  11h, 22h, 33h, 44h, 55h, 66h, 77h, 88h
.ENDLITERAL
ROM_BUF_SIZE:     equ   8

export _main

_main:
    ; Set RAM Buffer
    mov    A,>I2C_RAM_Buf       ; Save MSB of RAM buffer address
    push   A
    mov    A,<I2C_RAM_Buf       ; Save LSB of RAM buffer address
    push   A
    mov    A,BUF_RW_SIZE        ; Save RW size param
    push   A
```


[+] Feedback

```
    mov     A,RAM_BUF_SIZE        ; Save I2C buffer size
    push    A
    call    EzI2Cs_SetRamBuffer
    ADD     SP,-4                 ; Reset Stack

    ; Set ROM Buffer
    mov     A,>I2C_ROM_BUF        ; Save MSB of ROM buffer address
    push    A
    mov     A,<I2C_ROM_BUF        ; Save LSB of ROM buffer address
    push    A
    mov     A,ROM_BUF_SIZE        ; Save ROM buffer size
    push    A
    call    EzI2Cs_SetRomBuffer
    add     SP,-3                 ; Reset Stack

    M8C_EnableGInt                ; Enable Global Interrupts

    call    EzI2Cs_Start          ; Start and enable IRQ
    mov     A,5                   ; Set address to 5
    call    EzI2Cs_SetAddr

.Loop:

    ;; User Code goes here

    jmp     .Loop
```

配置寄存器

本节介绍 EzI2Cs 用户模块所使用或修改的 PSoC 资源寄存器。 当使用 EzI2Cs 用户模块时无需使用这些寄存器，但是这些寄存器可作为参考提供。

Table 2. 资源 I2C_CFG：组 0 reg[D6] 配置寄存器

位	7	6	5	4	3	2	1	0
值	Reserved	PinSelect	Bus Error IE	Stop IE	时钟 Rate[1]	时钟 Rate[0]	0	Enable Slave

引脚选择：选择 SCL 和 SDA 作为 P1[5] – P1[7] 或 P1[1] – P1[0]。

总线错误中断启用：在发生总线错误时 I^2C 中断生成。

停止错误中断启用：在 I^2C 停止条件下启用 I^2C 中断。

时钟频率 [1,0]：在三种有效时钟频率 50、100、400 kbps 之间选择。

启用从器件：以总线从器件的形式启用 I^2C HW 模块。

[+] Feedback

Table 3. 资源 I2C_SCR：组 0 reg[D7] 状态控制寄存器

位	7	6	5	4	3	2	1	0
值	总线错误	NA	停止状态	输出 ACK	地址	发送	最后接收位 (LRB)	字节完整

总线错误：表示检测到总线错误条件。

停止状态：表示检测到 I^2C 停止条件。

输出 ACK：指示 I^2C 模块对所收到的字节进行确认 (1) 或否认 (0)。

地址：所接收或所发送的字节是一个地址。

最后接收位 (LRB)：传输序列中最后收到的位 （第 9 位）的值，表示目标设备中的 Ack/Nak 状态。

字节完整： 接收到了 8 个数据位。 对于接收模式，总线在等待确认 / 否认应答时停顿。 在发送模式下，已经接收了确认 / 否认应答 （请参见 LRB），并且总线停顿以等待下一项操作。

Table 4. 资源 I2C_DR：组 0 reg[D8] 数据寄存器

位	7	6	5	4	3	2	1	0
值	Data							

所接收或所发送的数据。 要发送数据，必须在向 I2C_SCR 寄存器写入前加载该寄存器。 从此寄存器中读取所接收的数据，其中可能包含地址或数据。

[+] Feedback

版本历史记录

版本	创作者	说明
1.2	DHA	已进行了更新以防止 SCL 停滞在低电平上。 对 Start 函数进行了如下更改： 1. 已将用户模块引脚的初始漏极开路低电平驱动模式更改为 HI-Z 模拟。 2. 已启用 I^2C 模块。 3. 给定延时 5 秒的 NOP 指令。 4. 已恢复初始 I^2C 引脚驱动模式。 已将 EzI2C_StopSlave API 设为公开。 已将偏移指针初始化添加到 Start API。

Note PSoC Designer 5.1 在所有的用户模块数据手册中提供版本历史记录。 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 © 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.