EzI2C 从数据表 EzI2Cs V 1.2

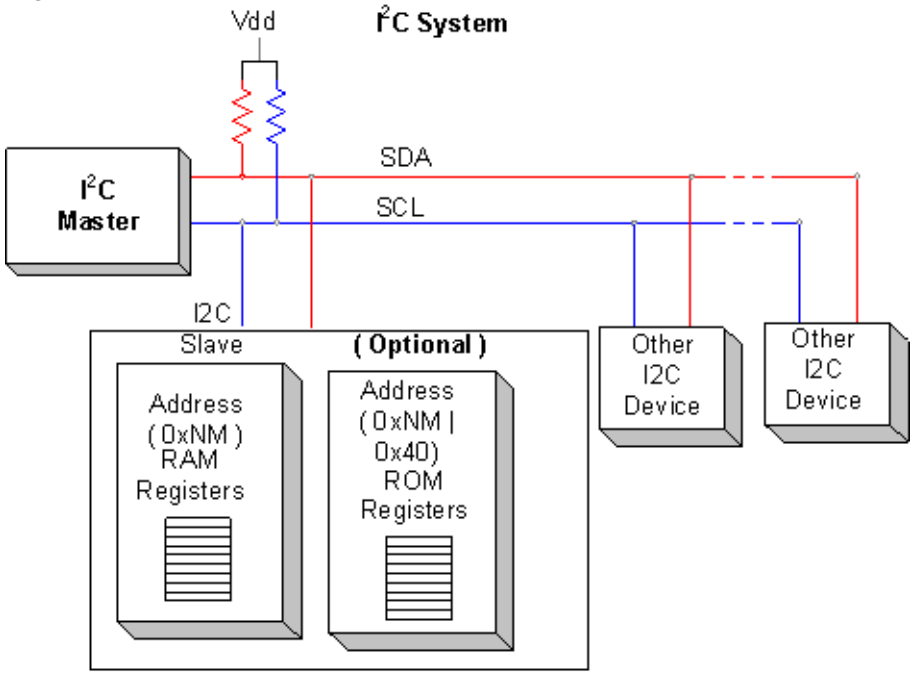
资源	PSoC® 模块				API 存储器（字节）		每个传感器所使用的引脚数
	CapSense®	I2C/SPI	定时器	比较器	闪存	RAM	
CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C643/4/5xx、CY7C60413、CY7C60424、CY7C6053x、CYONS2010、CYONS2011、CYONSFN2051、CYONSFN2053、CYONSFN2061、CYONSFN2151、CYONSFN2161、CYONSFN2162、CY8CTST200、CY8CTMG2xx、CY8CTMA30xx							
RAM 读取 / 写入缓冲区	-	1	-	-	264	6	2
增加了 ROM 缓冲区支持	-	1	-	-	379	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
- 仅有两个引脚（SDA 和 SCL）需要与 I^2C 总线连接
- 标准数据速率为 100/400 kbps
- 高级 API 只需少量用户编程

EzI2Cs 用户模块实现基于 I^2C 寄存器的从设备。I^2C 总线是由 Philips® （现在称为 NXP）开发的行业标准两线硬件接口。主控在 I^2C 总线上启动所有通信，并为所有从设备提供时钟。EzI2Cs 用户模块支持最大速度为 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设备兼容。

Figure 1. I²C 框图



Single I2C Slave Device (Address 0xNM)
Shown with RAM registers and optional ROM registers

功能说明

EzI2Cs 用户模块采用的方法与随 PSoC Designer 提供的 I2CHW 用户模块的方法不同。 此用户模块仅支持带有一个或两个 I²C 地址的 I²C 从配置。 第一个 I²C 地址访问 RAM 分配的区域，可选第二个 I²C 地址访问 ROM 区域。 第二个 I²C 地址为 RAM 区域的 I²C 地址和 0x40 作罗辑 “ 或 ” 的结果。 例如，如果选择 I²C 地址 0x15，则 RAM 区域 I²C 地址为 0x15，可选 ROM I²C 地址为 0x55。

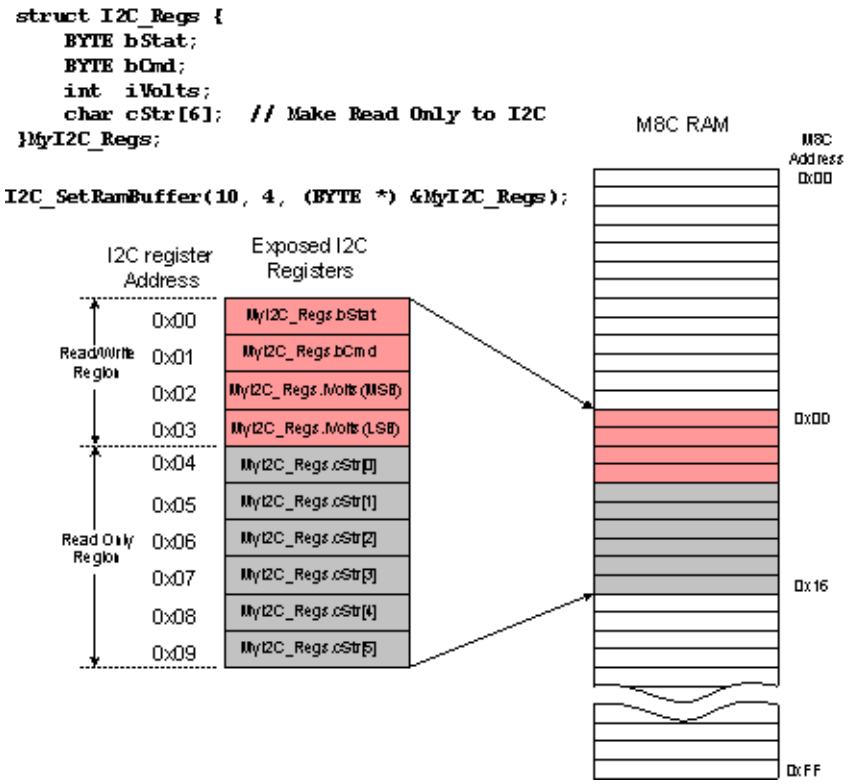
由于 I²C 硬件是由中断驱动的，此用户模块要求启用全局中断。 即使此用户模块需要中断，用户代码也不需要向 ISR （中断服务例程）添加任何代码。 此模块为所有中断 （数据传输）提供服务，与用户代码无关。 相反，为此接口分配的存储器缓冲区看上去类似于用户应用程序与 I²C 主控之间的简单双端口存储器。

如果需要，用户可以通过在数据结构中定义信号和命令位置，在主控与该从器件之间创建较高级别的接口。

RAM 区域接口

对于 I²C 主控而言，接口看上去非常类似于通用 256 字节 I²C EEPROM。 PSoC API 被当作可配置为简单变量、数组或结构的 RAM。 在某种程度上，它充当用户程序与 I²C 总线上 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，避免在该数据范围外进行任何读写。 公开给 I²C 接口的数据可以是简单变量、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。

Figure 2. RAM 区域接口



例如，用户可以创建此结构。

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

只要此结构在存储器中是连续的且被指针引用，就可以包含具有任何名称的任何变量组。 接口 （I²C 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。 借助上述示例结构，可以使用提供的 API 将数据结构公开给 I²C 接口。 第一个参数将公开的 RAM 的大小设置为 I²C 接口，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 / 写区域与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，只有前 4 个字节可以写入，但是所有字节都可以由 I²C 主控读取。 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，创建了 15 字节数组，并将其公开给 I^2C 接口。 数组的前 8 个字节为读 / 写型，其余 7 个字节为只读型。

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

下例是一个很简单的示例，其中仅公开了单一整数 （2 个字节）。 这两个字节都可以由 I^2C 主控读 / 写。

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

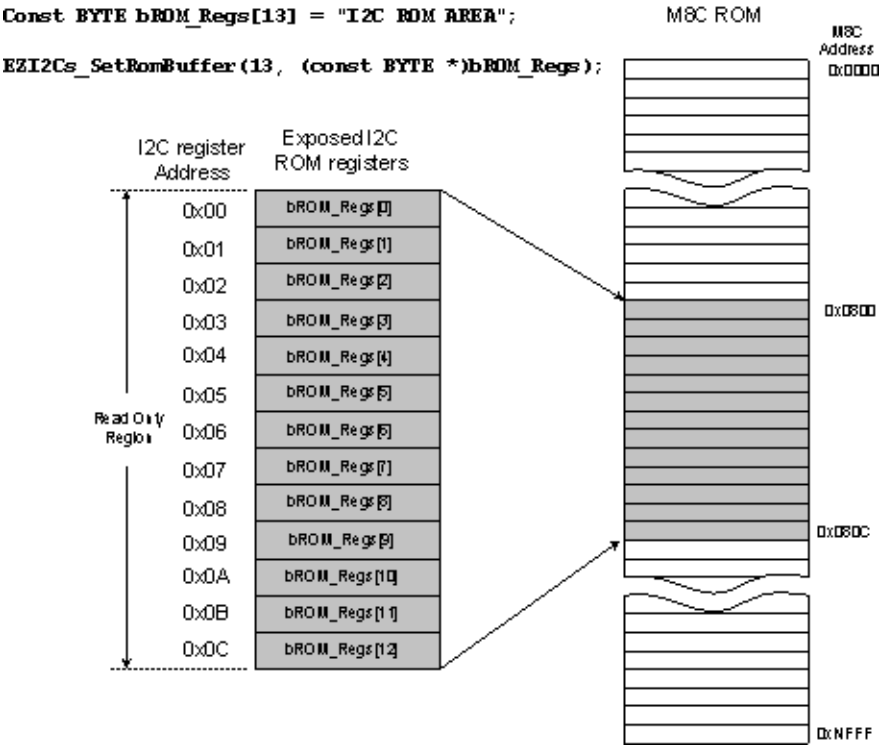
ROM 区域接口 （可选）

ROM 接口几乎与 RAM 接口相同，只不过它是只读的。 如上例所示，SetRomBuffer 函数类似于 SetRamBuffer 函数。 在此示例中，数据区域是简单的常量字符串，其长度为 13 字节（包括结尾的 NULL (0x00) 字符）。

```
Const BYTE bROM_Regs[13] = "I2C ROM AREA";
EZI2Cs_SetRomBuffer(13, (const BYTE *)bROM_Regs);
```

如上所述，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 地址为 0x15，则用于访问 ROM 区域的 I^2C 地址将为 0x55。 下图显示了此示例在存储器中的显示形式：

Figure 3. ROM 区域接口



外部主控可以监控的接口

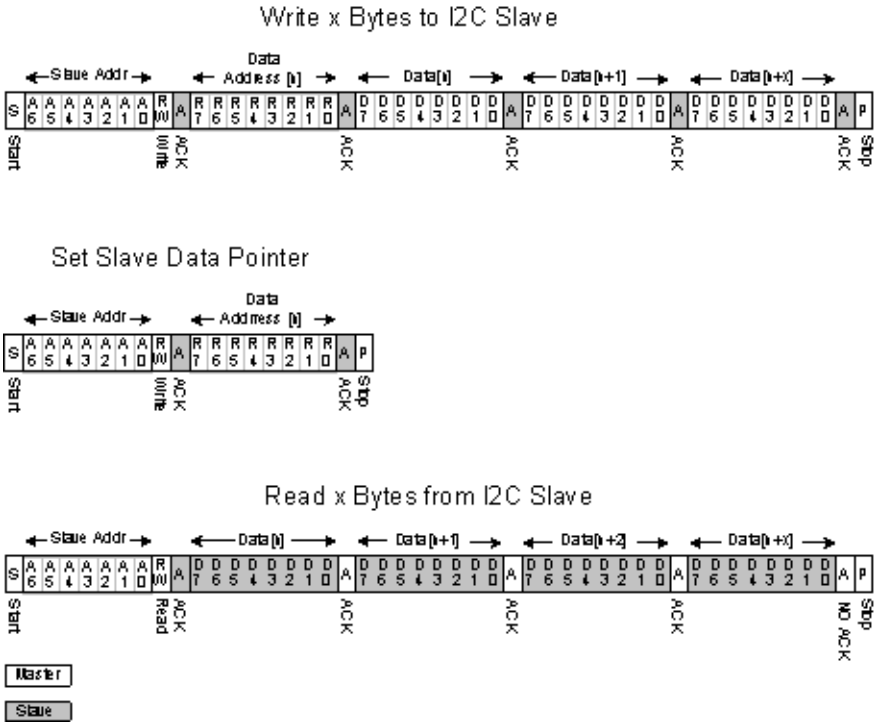
EzI2Cs 用户模块支持针对 RAM 区域的基本读写操作和针对 ROM 区域的只读操作。 RAM 和 ROM 区域接口包含使用写入操作的第一个数据字节设置的单独数据指针。 甚至 ROM 区域也可以接受写入单一字节来设置数据指针。 当写入一个或多个 RAM 字节时，第一个数据字节始终是数据指针。 数据指针后的字节写入到数据指针字节指向的位置。 第三个字节 （第二个数据字节）写入数据指针加 1 的位置，依此类推。 此数据指针按每个读取或写入的字节递增，但是在每个新读取操作开始时复位为第一个写入的值。 新的读取操作开始在数据指针指向的位置读取数据。

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，并连续读取，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。

Figure 4. 将 x 字节读写到 I2C 从器件



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传输完成或检测到总线错误时发生。

每次的 I^2C 数据操作由启动、寻址、R/W 方向、数据和终止构成。 对于 I^2C 从器件，将在传入数据的第 8 位后发生中断。 在此位置点，接收设备必须决定是确认 (ack) 还是不确认 (nak) 传入字节 （无论它是地址还是数据）。 接收设备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，在外部接收设备提供 ack 或 nak 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 NXP 网站上提供的完整 I^2C 规范中，以及通过参考随 PSoC Designer 提供的设备数据表，可以获得 I^2C 总线的详细说明和应用部署。

设计注意事项

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，但是在指定的 I^2C 速度下受逐字节数据传输函数的限制。

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。 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 设备产生负面影响。 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，一定要注意避免此种可能性。

直流和交流电气特性

请参考 PSoC 设备的数据表，了解 I^2C 接口的电气特性。

根据框图说明，I^2C 总线要求外部上拉电阻。 上拉电阻 (R_P) 由供电电压、时钟速率和总线电容决定。 将输出阶段的任何设备 （主控或从器件）的最小吸收电流设置为不超过 3 mA (t V_{OLmax} = 0.4V)。 对于 5V 的系统，这样可以将最小上拉电阻值限制到大约为 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，上拉电阻不得大于 6 kΩ。 有关 “I^2C 总线规范 ” 的信息，请参见 NXP 网站 www.nxp.com。

Note 从赛普拉斯或它的一个许可联营公司处购买 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 NXP 定义的 I^2C 标准规范。

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。

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，则有效选择为从 0 到 127（ 十进制格式）。 如果启用 ROM 寄存器，则只有 0 到 63 之间的地址有效，这是因为 ROM 寄存器空间为 RAM 地址加上 64 （十进制格式的 64 到 127）。

Address_Type

选择地址是静态地址还是动态地址。 如果设置为静态地址，则不能更改 I^2C 地址，但是可以保存 RAM 的一个字节。 设置此选项可以动态允许固件随时更改地址。 通常，此选项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

此 RAM 区域一定会被启用，但是可以启用另一个地址以允许访问 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。此第二个地址是用 0x40 进行或运算的基地址。

HW Addr Rec

此参数启用 / 禁用硬件 I2C 从地址识别功能。 它还允许减少 ROM/RAM 的使用和 I2C ISR 的执行时间。

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。 只有当 IMO (SysClk) 为 12 MHz 时，才能使用 400 kHz 速度。

I2C_Pin

选择用于 I^2C 信号的端口 1 的引脚。 不需要为引脚选择正确的驱动模式，PSoC Designer 会自动执行此操作。 这允许将 I2C 时钟和数据信号放在 P1[5] - P1[7] 或 P1[1] - P1[0] 上。

应用程序编程接口

若要确保长度为 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 变量的值之前检查此类变量，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	正在进行的 RAM 读取数据操作
I2C_BUSY_RAM_WRITE	0x03	正在进行的 RAM 写入数据操作
I2C_BUSY_ROM_READ	0x01	正在进行的 ROM 读取数据操作
I2C_BUSY_ROM_WRITE	0x03	正在进行的 ROM 写入数据操作

EzI2Cs_Start

说明：

启用 I^2C 从器件并启用中断。 在此函数调用之前调用所需的 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：

可以用此函数更改 A 和 X 寄存器。

EzI2Cs_Stop

说明：

在禁用 I^2C 中断的情况下，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：

可以用此函数更改 A 和 X 寄存器。

EzI2Cs_DisableSlave

说明：

在不禁用 I^2C 中断的情况下，禁用 EzI2Cs。

C 原型：

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

汇编程序：

```
lcall  EzI2Cs_DisableSlave
```

参数：

无

返回值：

无

副作用：

此函数会更改 A 和 X 寄存器。

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，此函数覆盖该地址。 此函数可选，只有根据默认值更改地址时，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： 1 和 127 之间的从地址。

返回值：

无

副作用：

可以用此函数更改 A 和 X 寄存器。

EzI2Cs_GetActivity

说明：

如果自从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。 活动标志在此函数调用的末尾复位为零。C 和汇编中提供了符号常量。 它们的值定义如下：

符号	值	含义
EzI2Cs_ANY_ACTIVITY	0x80	发生写入或读取循环
EzI2Cs_READ_ACTIVITY	0x20	发生读取循环
EzI2Cs_WRITE_ACTIVITY	0x10	发生写入循环

C 原型：

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

汇编程序：

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

参数：

无。

返回值：

如果发生 I^2C 读取或写入，则返回非零值。 如果自从上次调用此函数以来未发生任何活动，则为零。

副作用：

可以用此函数更改 A 和 X 寄存器。

EzI2Cs_GetAddr

说明：

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

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：

可以用此函数更改 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。请牢记要通过使用以下这个宏来调用这个全局中断启用函数: M8C_EnableGInt. 如果调用了 EzI2Cs_Start 函数，则不需要此函数。 默认情况下，此函数清除待处理中断。 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：

可以用此函数更改 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。 此函数在启用中断之前不清除待处理中断。 请牢记要通过使用以下这个宏来调用这个全局中断启用函数: M8C_EnableGInt. 如果调用了 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：

可以用此函数更改 A 和 X 寄存器。

EzI2Cs_DisableInt

说明：

通过禁用 SDA 中断禁用 I^2C 从器件。 执行与 EzI2Cs_Stop. 相同的操作

C 原型：

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

汇编程序：

```
lcall EzI2Cs_DisableInt
```

参数：

无

返回值：

无

副作用：

可以用此函数更改 A 和 X 寄存器。

副作用：

可以用此函数更改 A 和 X 寄存器。

EzI2Cs_SetRamBuffer

说明：

此函数用于设置为 I^2C 主控公开的 RAM 缓冲区的位置和大小（最多 255 字节）。 此函数是必需的，在调用 EzI2Cs_Start(). 之前调用此函数

C 原型：

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

汇编程序：

```
lcall  EzI2Cs_SetRamBuffer
```

参数：

BYTE bSize： 公开给 I^2C 主控的数据结构的大小。 bSize 的最小值为 1，最大值为 255。 BYTE bRWboundry： 从第一个位置开始的可写入位置的大小。 此值应始终小于或等于相应操作的 bSize。 BYTE * pAddr： 指向数据结构的指针。

返回值：

无

副作用：

可以用此函数更改 A 和 X 寄存器。 如果 bRWboundry 设置为大于 bSize 的值，则整个 bRWboundry 大小是可写入的。 将 bSize 设置为无效值 0 会允许 I2C 主机写入未定义的存储器位置。

EzI2Cs_SetRomBuffer

说明：

此函数设置公开给 I^2C 主控的 ROM 缓冲区的位置和大小（最多 255 字节）。 如果需要使用 ROM，则在启用 EzI2Cs_Start 函数之前需要此函数。

C 原型：

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

汇编程序：

```
lcall  EzI2Cs_SetRomBuffer
```

参数：

BYTE bSize： 公开给 I^2C 主控的数据结构的大小。const BYTE * pAddr： 指向闪存 ROM 中的数据结构的指针。

返回值：

无

副作用：

可以用此函数更改 A 和 X 寄存器。

固件源代码示例

C 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
   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
    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 用户模块使用或修改的 PSoC 资源寄存器。 当使用 EzI2Cs 用户模块时不需要使用这些寄存器，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	Clock Rate[1]	Clock Rate[0]	0	Enable Slave

Pin Select：以 P1[5] - P1[7] 或 P1[1] - P1[0] 的形式选择 SCL 和 SDA。

Bus Error Interrupt Enable：在发生总线错误时启用 I^2C 中断生成。

Stop Error Interrupt Enable：在 I^2C 停止条件下启用 I^2C 中断。

Clock Rate[1,0]：在三个有效时钟速率 50、100、400 kbps 中选择。

Enable Slave：以总线从器件的形式启用 I^2C HW 模块。

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	Bus Error	NA	Stop Status	ACK out	Address	Transmit	Last Recd Bit (LRB)	Byte Complete

Bus Error：指示总线错误条件的检测。

Stop Status：指示 I^2C 停止条件的检测。

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

Address：已接收或发送的字节为地址。

Last Received Bit (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。

Byte Complete： 已收到 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。 对于传输模式，也会收到 Ack Nak （请参见 LRB），总线停止工作，等待下次操作。

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。 从此寄存器读取接收的数据，这些数据可以包含地址或数据。

版本历史记录

版本	创作者	说明
1.2	DHA	“I2C 引脚” 参数的默认引脚值已被删除，因为该值会破坏其他用户模块最初设置的相应引脚的驱动模式。 已更新以防止 SCL 停滞在低电平上。 对启动函数进行以下更改： 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. 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。 已添加 *.h 和 *.inc 文件的 “ 导出 EzI2C_StopSlave API”。 增加通过影子寄存器更新输出引脚的功能。 更新的 RAM 分页管理。 已添加 RAM 和 ROM 偏移指针初始化以启动 API。 已添加硬件地址识别功能。

Note PSoC Designer 5.1 在所有的用户模块数据表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。