SMBus 从设备数据手册 SMBusSlave V 2.00

Copyright © 2012-2014 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		引脚
	数字	模拟 CT	模拟 SC	闪存	RAM	
支持的器件包括： CY8C29x66、 CY8C27x43、CY8C28xxx、 CY8C24x23、 CY8C24x33、 CY8C21x23、 CY8C21x34、 CY8C21x45、 CY8C22x45、 CY8C24x94						
配置 1	0	0	0	934	31	2–5

功能和概述

- 行业标准 SMBus 可兼容从设备接口
- 通过使用指令和专用的通知引脚，支持 SMBus 主机通知协议 （HNP）
- 可选的数据包错误检查 （PEC）
- 50/100 kbps 的标准数据速率
- 用户只需要基于高级应用编程接口 （API）进行少量的编程

SMBus 从设备用户模块提供了 SMBus 从设备接口，该接口与 SMBus 规范版本 2.0 中介绍的物理和数字链接层完全兼容。可以将该用户模块和连接到单个 SMBus 段的其他主设备和从设备结合使用。对于物理层，该用户模块采用了一个 I2C 控制器及其中断。

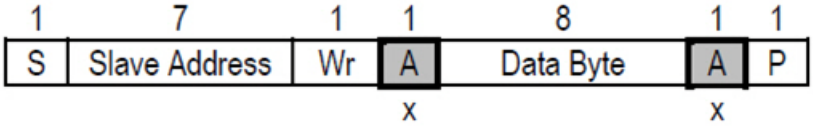
功能说明

通过使用 SMBus 规范版本 2.0 的数据链接层中所规定的协议， SMBus 从设备用户模块与总线主设备进行通信。它通过 SMBDAT 和 SMBCLK 线进行通信，并具有以下各项功能：

- 检测启动 （START）、重复启动 （Repeat START）、停止 （STOP）和总线闲置等条件
- 当该用户模块被总线主设备寻址时，它会使用 ACK 数据包做出响应
- 实现时钟延长和同步操作
- 按照总线协议的所有规范进行通信
- 使用 ACK/NACK 数据包进行响应
- PEC 操作可以由用户指定
- 使用 SMBus HNP 通知给主机

SMBus 从设备用户模块支持 SMBus 规范版本 2.0 第 5.5 节中所规定的所有总线协议。如果使能了数据包错误检查参数，那么所有上述协议都可进行 PEC 操作。图 1 显示的是通用协议框图。

图 1. 通用协议框图



S	Start Condition
Sr	Repeated Start Condition
Rd	Read (bit value of 1)
Wr	Write (bit value of 0)
x	Shown under a field indicates that that field is required to have the value of 'x'
A	Acknowledge (this bit position may be '0' for an ACK or '1' for a NACK)
P	Stop Condition
PEC	Packet Error Code
	Master-to-Slave
	Slave-to-Master
...	Continuation of protocol

注意： SMBus 从设备用户模块不支持地址解析协议 （ARP）。

寻址机制

从设备地址参数可以在参数向导中配置从设备地址。也可以使用 SMBusSlave_SetAddr() 函数在固件中动态修改该地址。

通过使能硬连线从设备地址参数可以使用两个 GPIO 引脚设置从设备地址的低两位从设备地址引脚 A0 和 A1 参数用于选择两个 GPIO 引脚。如果硬连线从设备地址参数被使能，那么在启动期间会监控这两个引脚，并且可以使用其逻辑电平进行设置两个从设备地址的最低有效位 （引脚 A1 = 从设备地址位 1，引脚 A0 = 从设备地址位 0）。从设备地址的高五位等于从设备地址参数的位 6–2。

当 SMBusSlave_SetAddr() 用于动态修改固件的从设备地址时，即使硬连线从设备地址已经被使能，从设备地址引脚 A0 和从设备地址引脚 A1 也会被忽略。新的从设备地址将被设置为 SMBusSlave_SetAddr() 函数的参数。仅在启动期间，从设备地址引脚 A0 和从设备地址引脚 A1 才被监控。

SMBus 指令的 RAM 接口

通过 SMBus 从设备用户模块，主设备可以使用每个 SMBus 指令访问指定的 RAM 缓冲区。为主设备提供的数据可能是单一变量、数组或结构。通过不同的 SMBus 指令，可以访问不同的 RAM 缓冲区。根据 SMBus 从设备用户模块收到的来自主设备的指令，它会对相应的 RAM 缓冲区进行读 / 写操作。

进行任何传输操作前，用户必须通过使用 SMBusSlave_SetCommand() 和 SMBusSlave_SetBuffer() 函数为每个 SMBus 指令分配指令代码并初始化 RAM 缓冲区。启动用户模块前，在初始化期间必须执行这两个函数。

分配指令代码

SMBusSlave_SetCommand （BYTE Command_Name, BYTE Command_Code）

Command Name （指令名称）：是指用户模块用以将指令名称映射到其指令代码的 SMBus 指令名称。该参数的可选值为：WRITE_BYTE、WRITE_WORD、READ_BYTE、READ_WORD、PROCESS_CALL、BLOCK_WRITE、 BLOCK_READ 和 BLOCK_WRITE_BLOCK_READ_PROCESS_CALL。

Command Code （指令代码）：是给定指令名称的指令代码。该参数的可能值为 0x00 - 0xFF。用户必须确保避免发生两个指令具有同一个指令代码的情况。

例外：不用为 Quick Command （快速指令）、 Send Byte （发送字节）和 Receive Byte （接收字节）等指令分配指令代码。

初始化 RAM 缓冲区

SMBusSlave_ SetBuffe （BYTE Command_Name, (BYTE *) pAddr）

Command_Name：是指用户模块用于将指令名称映射到其 RAM 缓冲区的 SMBus 指令名称。此参数的可选值为：WRITE_BYTE、 WRITE_WORD、READ_BYTE、 READ_WORD、 BLOCK_WRITE 和 BLOCK_READ。

pAddr：指向给定指令名称的 RAM 缓冲区的指针。

例外：

- 快速指令、发送字节指令和接受字节指令无需分配 RAM 缓冲区。这些指令采用了默认的单字节 RAM 变量。
- 进程调用指令采用了写 / 读字 （Write Word/Read Word) 指令缓冲区。
- 模块写 / 读 / 进程调用 （Block Write/ Block Read/Process Call）指令采用了模块写 / 读指令缓冲区。

缓冲区的长度分别为：

- 读 / 写字节 （Read/Write Byte）指令缓冲区的长度为一个字节
- 读 / 写字指令缓冲区的长度为两个字节
- 模块写指令缓冲区的长度 = 模块写缓冲区长度参数

  模块读指令缓冲区的长度 = 模块读缓冲区长度参数

  注意：这两个缓冲区的总长度不能超过 32 个字节

有关初始化代码的信息，请参考固件源代码示例部分。

指令说明

快速指令

SMBus 从设备设置 / 清除预定义标志 （变量），该标志反映的是通过该指令接收到的读 / 写位。

发送字节指令

SMBus 从设备将接收到的字节写给某个预定义的用户模块变量。

接收字节指令

SMBus 从设备将发送预定义用户模块变量的字节变量。

写字节指令

当 SMBus 从设备通过该协议以及由用户定义的有效指令代码接收到一个字节时，它会将该字节写入由用户初始化的 RAM 缓冲区内。

写字指令

当 SMBus 从设备通过该协议以及由用户定义的有效指令代码接收到一个字时，它会将该字写入由用户初始化的 RAM 缓冲区内。

读字节指令

当 SMBus 从设备接收到该指令以及由用户定义的有效指令代码时，它会从用户初始化的 RAM 缓冲区的发送一个字节。

读字指令

当 SMBus 从设备接收到此指令以及由用户定义的有效指令代码时，它会从用户初始化的 RAM 缓冲区中发送一个字。

进程调用指令

当 SMBus 从设备收到此指令以及由用户定义的有效指令代码时，它会接收由主设备发送的字，并将其写入到分配给写字指令的 RAM 缓冲区内，然后再将分配给读字指令的 RAM 缓冲区中的字发送出去。

模块写指令

当 SMBus 从设备接收到此指令以及一个由用户定义的有效指令代码时，它将检查主设备需要写的字节数量 （N）。如果 N 不大于模块写缓冲区长度参数，则从设备会读取主设备的 N 字节，然后将其写入到分配给模块写指令的 RAM 缓冲区内。如果 N 大于模块写缓冲区长度参数，则从设备会否认（NACK）主设备所发送的字节数量 （Byte Count）字节。

模块读指令

当 SMBus 从设备接收到此指令以及一个由用户定义的有效指令代码时，它会将字节数量 （N）发送给主设备，其中 N 等于模块读缓冲区长度参数。然后，它将 N 字节从分配给模块读指令的 RAM 缓冲区发送给主设备。

模块写 - 读进程调用指令

当 SMBus 从设备接收到该指令以及一个由用户定义的有效指令代码时，它将检查主设备需要写的字节数量 （N）。如果 N 不大于模块写缓冲区长度参数，则它会读取主设备的 N 字节，然后将它写到分配给模块写指令的 RAM 缓冲区内。总线执行重复开始 （Repeat Start）后， SMBus 从设备会将字节数量 （M）发送给主设备，其中 M 等于模块读缓冲区长度参数。然后，它将 M 字节从分配给模块读指令的 RAM 发送给主设备。在执行该指令期间，所传输的字节总数必须小于 32 个字节。

SMBus 主机通知协议

该协议不会由主设备发起。一旦需要通知给主机，用户可使用 SMBusSlave_NotifyHost() 进行此操作。

SMBusSlave_NotifyHost （BYTE Type、 BYTE Databyte1、 BYTE Databyte2）

Type （类型）：该参数决定通知给主机的方式：

0 = SMBALERT 引脚方式

1 = 主机通知指令方式

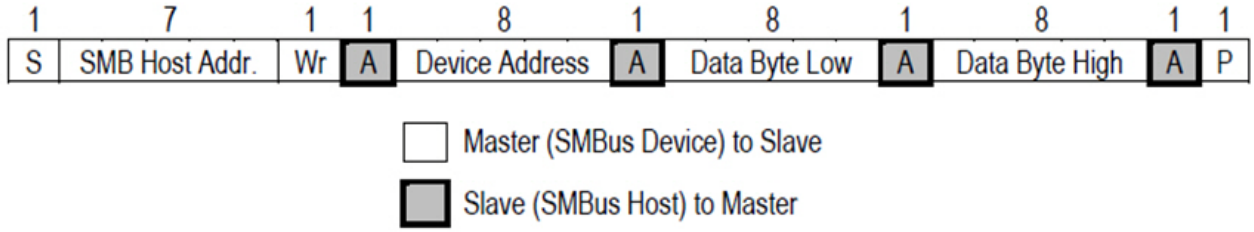
Databyte1：它是 SMBus 主机通知协议的数据低字节。

Databyte2：它是 SMBus 主机通知协议的数据高字节。

注意： Databyte1 和 Databyte2 参数仅适用于 HNP 方式。对于 SMBALERT 引脚方式，会忽视这些参数。

用户模块会暂时作为主设备，并同时驱动 SMBDAT 和 SMBCLK。 SMBus 会使用图 2 中显示的字节序列通知主机。

图 2. 主机通知协议



直流和交流电气特性

下列各值表示预计的性能，它们是根据初始特性数据得到的。除非另有指定，否则温度范围为 –40 °C ~ 125 °C，供电电压范围为 2.95 V ~ 5.3 V。

表 1. 直流电气特性

符号	参数	范围		单位
		最小值	最大值	
VIL	数据、时钟输入低电压	–	0.8	V
VIH	数据、时钟输入高电压	2.1	VDD	V
VOL	数据、时钟输出低电压	–	0.4	V
ILEAK	输入漏电流	–	±5	µA
IPULLUP	经过上拉电阻电流或拉电流	100	350	µA
VDD	额定总线电压	2.7	5.5	V

由于受硬件限制，此用户模块不满足 SMBus 规范版本 2.0 中所规定的 $T_{TIMEOUT}$、 T_{SEXT} 和 T_{MEXT} 规范

表 2. 交流电气特性

符号	参数	范围		单位
		最小值	最大值	
FSMB	SMBus 工作频率	10	100	kHz
TBUF	停止和启动条件之间的总线空闲时间	4.7	–	µs
THD:STA	（重复）启动条件后的保持时间。经过时间段后，会生成第一个时钟	4.0	–	µs
TSU:STA	重复启动条件的建立时间	4.7	–	µs
TSU:STO	停止条件的建立时间	4.0	–	µs

符号	参数	范围		单位
		最小值	最大值	
THD:DAT	数据保持时间	300	–	ns
TSU:DAT	数据建立时间	250	–	ns
TLOW	时钟为低电平的周期	4.7	–	µs
THIGH	时钟为高电平的周期	4.0	–	µs
TF	时钟 / 数据的下降时间	–	300	ns
TR	时钟 / 数据的上升时间	–	1000	ns
TPOR	上电复位后器件必须处于工作状态的时间	–	500	ms

放置

SMBus 从设备用户模块不需要任何数字或模拟 PSoC 模块。它使用了一个 I^2C 控制器模块及其专用的中断。具有多个 I^2C 控制器模块的器件支持放置多个 SMBus 从设备用户模块。其它器件不支持多个放置。

参数和资源

使用器件编辑器选择和放置 SMBus 从设备用户模块后，便能够选择和更改下列参数的值。

从设备地址

此参数为 SMBus_Slave 用户模块选择了 7 位从设备地址。

类型	特性
范围	0 – 127
默认值	0

其它参数的依赖性：

- 如果硬连线从设备地址被使能，则从设备地址仅限于五个最高有效位。其余两位会被设置为从设备地址引脚 A0 和从设备地址引脚 A1。
- 如果硬连线从设备地址被禁用，则从设备地址为该参数的全部七位。

硬连线从设备地址

可使用该参数选择是否对从设备地址的两个最低有效位进行硬连线连接。

类型	Boolean 逻辑
范围	使能 / 禁用
默认值	禁用

数据包错误检查

使用该参数可检查在通信期间器件是否使用了 PEC。

类型	Boolean 逻辑
范围	使能 / 禁用
默认值	使能

地址自动检查

通过该参数可选择是否使能硬件地址识别功能。如果禁用了该功能，则硬件地址比较功能不可用。该参数仅在 CY8C28xxx 系列 PSoC 器件中可用。

类型	Boolean 逻辑
范围	使能 / 禁用
默认值	使能

SMBus 时钟

该参数用于选择 SMBus 从设备的时钟速度。

类型	整数
范围	50 kHz – 100 kHz
默认值	100 kHz

SMBus 引脚

此参数为 SMBus 从设备信号 （SMBDAT 和 SMBCLK）选择端口 1 引脚。

类型	Boolean 逻辑
范围	P1[0] - P1[1] 或 P1[5] - P1[7]
默认值	P1[0] - P1[1]

SMBALERT 引脚

通过该参数为 SMBALERT 信号选择引脚。

类型	枚举
范围	任意 GPIO
默认值	无

从设备地址引脚 A0

通过该参数为从设备地址的 A0 位选择引脚。

类型	枚举
范围	任意 GPIO
默认值	无

其它参数的依赖性：

- 如果硬连线从设备地址被使能，则该参数有效
- 否则，该参数进入无效状态

从设备地址引脚 A1

通过该参数为从设备地址的 A1 位选择引脚。

类型	枚举
范围	任意 GPIO
默认值	无

其它参数的依赖性：

- 如果硬连线的从设备地址被使能，则该参数有效
- 否则，该参数进入无效状态

模块写缓冲区长度

使用该参数可为模块写指令设置 RAM 缓冲区长度。

类型	整数
范围	1–32
默认值	1

模块读缓冲区长度

使用该参数可为模块读指令设置 RAM 缓冲区长度。

类型	整数
范围	1–32
默认值	1

应用编程接口 （API）

所提供的应用编程接口 （API）程序作为用户模块的一部分，通过它设计人员能够采用更高级的方式处理模块。本部分具体说明了每个函数所对应的接口以及 “include” 文件所提供的相关常量。

注意

在这里，同所有用户模块 API 中的一样，调用 API 函数可能会修改 A 和 X 寄存器的值。如果在调用后需要 A 和 X 的值，则调用函数要保留调用前 A 和 X 的值。选择该 “ 寄存器易失 ” 策略的目的是为了提高效率，从 PSoC Designer 版本 1.0 起开始强制使用此策略。 C 编译器会自动遵循该要求。汇编语言编程员也要确保其代码遵循该策略。虽然一些用户模块 API 函数可以保持 A 和 X 不变，但是无法保证它们将来也会如此。

从设备函数

下面的函数只适用于 SMBusSlave 用户模块的从设备版本。

SMBusSlave_Start

说明：

使能 SMBusSlave。

C 原型：

```
void   SMBusSlave_Start (void);
```

汇编：

```
lcall  SMBusSlave_Start
```

参数：

无

返回值：

无

SMBusSlave_SetAddr

说明：

使用该函数可设置由 SMBus 主设备识别的从设备地址。应根据硬连线从设备地址参数来限制此地址的范围。

C 原型：

```
void   SMBusSlave_SetAddr (BYTE bAddr);
```

汇编：

```
movA, 0x05
lcall  SMBusSlave_SetAddr
```

参数：

BYTE bAddr：取决于硬连线从设备地址的从设备地址范围

返回值：

无

SMBusSlave_Stop

说明：

禁用 SMBusSlave。

C 原型：

```
void SMBusSlave_Stop (void);
```

汇编：

```
lcall SMBusSlave_Stop
```

参数：

无

返回值：

无

SMBusSlave_SetBuffer

说明：

为主设备的读 / 写操作配置 RAM 缓冲区，以便执行某个特殊指令。

C 原型：

```
void SMBusSlave_SetBuffer (BYTE bCommand_Name, BYTE * pAddr);
```

汇编：

```
lcall SMBusSlave_SetBuffer
```

参数：

BYTE Command_Name：SMBus 被初始化的指令。

BYTE *pAddr: 指定指令对应的缓冲区。

返回值：

无

SMBusSlave_SetCommand

说明：

将指令代码分配给某个指定的 SMBus 指令。

C 原型：

```
void SMBusSlave_SetCommand (BYTE Command_Name, BYTE Command_Code);
```

汇编：

```
lcall SMBusSlave_SetCommand
```

参数：

BYTE Command_Name：SMBus 被初始化的指令。

BYTE Command_Code：对应相应指令的指令代码。

返回值：

无

SMBusSlave_NotifyHost

说明：

根据 “Type” 参数决定通过 SMBALERT 引脚或主机通知指令来通知主机。

C 原型：

```
void SMBusSlave_NotifyHost (BYTE Type, BYTE Databyte1, BYTE Databyte2);
```

汇编：

```
lcall SMBusSlave_NotifyHost
```

参数：

BYTE Type：该参数决定了通知主机的方式。

0 = SMBALERT 引脚方式

1 = 主机通知指令方式

BYTE Databyte1：它是 SMBus 主机通知协议的数据字节 1。只有 Type = 1 时，该参数才有效。

BYTE Databyte2：它是 SMBus 主机通知协议的数据字节 2。只有 Type = 1 时，该参数才有效。

返回值：

无

固件源代码示例

下面的 C 语言代码说明了如何使用 API：

```
//-------------------------------------------------------------------------
// SMBusSlave sample code
// UM should be configured as follows:
// - Name: SMBusSlave
// - Slave Address: 4
// - Block Write Buffer Length: 6
// - Block Read Buffer Length: 6
// All other UM parameters should be left by default
//
//Here is almost SMBus Protocols to use with Bridge Control Panel tool:
// w 4 bb 80 p ;send byte NOTIFYHOST (PEC 80)
// r 4 x x p ;receive byte (PEC e2)
// w 4 40 b6 01 p ;write byte SMBALERT (PEC 01)
// w 4 50 ab cd 76 p ;write word (PEC 76)
// w 4 60 r 4 x x p ;read byte (PEC 82)
// w 4 70 r 4 x x x p ;read word (PEC b6)
// w 4 80 ab cd r 4 x x x p ;process call (PEC 83)
// w 4 20 4 1 2 3 4 bd p ;block write (PEC bd)
// w 4 30 r 4 x x x x x x x x p ;block read (PEC a6)
// w 4 10 5 2 3 4 5 6 r 4 x x x x x x x x p ;block process call (PEC 04)
//-------------------------------------------------------------------------

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

```
BYTE baBlockWriteBuffer[SMBusSlave_BLOCK_WRITE_BUFFER_LENGTH];
BYTE baBlockReadBuffer[] = {10, 11, 12, 13, 14, 15};
BYTE WriteByteBuffer;
WORD WriteWordBuffer;
BYTE ReadByteBuffer = 0xad;
WORD ReadWordBuffer = 0xbcde;

void main(void)
{
    M8C_EnableGInt;
    SMBusSlave_Start();

    SMBusSlave_SetCommand(SMBusSlave_BLOCK_WRITE, 0x20);
    SMBusSlave_SetCommand(SMBusSlave_BLOCK_READ, 0x30);
    SMBusSlave_SetCommand(SMBusSlave_WRITE_BYTE, 0x40);
    SMBusSlave_SetCommand(SMBusSlave_WRITE_WORD, 0x50);
    SMBusSlave_SetCommand(SMBusSlave_READ_BYTE, 0x60);
    SMBusSlave_SetCommand(SMBusSlave_READ_WORD, 0x70);
    SMBusSlave_SetCommand(SMBusSlave_PROCESS_CALL, 0x80);
    SMBusSlave_SetCommand(SMBusSlave_BLOCK_WRITE_BLOCK_READ_PROCESS_CALL, 0x10);

    SMBusSlave_SetBuffer(SMBusSlave_BLOCK_WRITE, baBlockWriteBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_BLOCK_READ, baBlockReadBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_WRITE_BYTE, &WriteByteBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_WRITE_WORD, (BYTE *) &WriteWordBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_READ_BYTE, &ReadByteBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_READ_WORD, (BYTE *) &ReadWordBuffer);

    SMBusSlave_bReceiveByte = 0xaa;

    while(1)
    {
        if(SMBusSlave_bSendByte == 0xbb)
        {
            SMBusSlave_bSendByte = 0;
            SMBusSlave_NotifyHost(SMBusSlave_NOTIFYHOST, 5, 6);
        }
        if(WriteByteBuffer == 0xb6)
        {
            WriteByteBuffer = 0;

            SMBusSlave_NotifyHost(SMBusSlave_SMBALERT, 5, 6);
        }
    }
}
```

下面显示的是汇编语言示例代码：

```
;-------------------------------------------------------------------
; SMBusSlave sample code
;
; UM should be configured as follows:
; - Name: SMBusSlave
; - Slave Address: 4
; - Block Write Buffer Length: 6
```

```
; - Block Read Buffer Length: 6
; All other UM parameters should be left by default
;
; Here are some SMBus Protocols to use with Bridge Control Panel tool:
; w 4 bb 80 p ;send byte NOTIFYHOST (PEC 80)
; w 4 40 b6 01 p ;write byte SMBALERT (PEC 01)
; w 4 20 4 1 2 3 4 bd p ;block write (PEC bd)
; w 4 30 r 4 x x x x x x x x p ;block read (PEC a6)
; w 4 10 5 2 3 4 5 6 r 4 x x x x x x x x p ;block process call (PEC 04)
;-----------------------------------------------------------------

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

export _main
export baBlockWriteBuffer
export baBlockReadBuffer
export bWriteByteBuffer

area data (rel,con,ram)

baBlockWriteBuffer: BLK SMBusSlave_BLOCK_WRITE_BUFFER_LENGTH
baBlockReadBuffer: BLK SMBusSlave_BLOCK_READ_BUFFER_LENGTH
bWriteByteBuffer: BLK 1

area text (rel,con,rom,code)

_main:
    mov [baBlockReadBuffer + 0], 10
    mov [baBlockReadBuffer + 1], 11
    mov [baBlockReadBuffer + 2], 12
    mov [baBlockReadBuffer + 3], 13
    mov [baBlockReadBuffer + 4], 14
    mov [baBlockReadBuffer + 5], 15
    mov [SMBusSlave_bReceiveByte], aah

    M8C_EnableGInt
    lcall SMBusSlave_Start

    mov X, 40h
    mov A, SMBusSlave_WRITE_BYTE
    lcall SMBusSlave_SetCommand
    mov X, 30h
    mov A, SMBusSlave_BLOCK_READ
    lcall SMBusSlave_SetCommand
    mov X, 20h
    mov A, SMBusSlave_BLOCK_WRITE
    lcall SMBusSlave_SetCommand
    mov X, 10h
    mov A, SMBusSlave_BLOCK_WRITE_BLOCK_READ_PROCESS_CALL
    lcall SMBusSlave_SetCommand

    mov A, >baBlockReadBuffer
    push A
```

```
    mov A, <baBlockReadBuffer
    push A
    mov A, SMBusSlave_BLOCK_READ
    push A
    lcall SMBusSlave_SetBuffer
    add SP, -3

    mov A, >baBlockWriteBuffer
    push A
    mov A, <baBlockWriteBuffer
    push A
    mov A, SMBusSlave_BLOCK_WRITE
    push A
    lcall SMBusSlave_SetBuffer
    add SP, -3

    mov A, >bWriteByteBuffer
    push A
    mov A, <bWriteByteBuffer
    push A
    mov A, SMBusSlave_WRITE_BYTE
    push A
    lcall SMBusSlave_SetBuffer
    add SP, -3

.mainloop:
    cmp [SMBusSlave_bSendByte], bbh
    jnz .SmbAlertCheck
    mov [SMBusSlave_bSendByte], 0
    mov A, 6 ;Data Byte High
    push A
    mov A, 5 ;Data Byte Low
    push A
    mov A, SMBusSlave_NOTIFYHOST
    push A
    lcall SMBusSlave_NotifyHost
    add SP, -3

.SmbAlertCheck:
    cmp [bWriteByteBuffer], b6h
    jnz .mainloop
    mov [bWriteByteBuffer], 0
    mov A, SMBusSlave_SMBALERT
    push A
    lcall SMBusSlave_NotifyHost
    add SP, -1
    jmp .mainloop
```

配置寄存器

本节介绍了由 SMBusSlave 用户模块使用或修改的 PSoC 资源寄存器。使用 SMBusSlave 用户模块时无需使用这些寄存器，但是可参考这些寄存器。

表 3. I2C_CFG 资源：组 0 reg[D6] 配置寄存器

位	7	6	5	4	3	2	1	0
数值	预留	PinSelect	Bus Error IE	Stop IE	Clock Rate[1]	Clock Rate[0]	0	使能从设备

Pin Select （引脚选择）：分别将 SCL 和 SDA 选为 P1[5]/P1[7] 或 P1[0]/P1[1]。

Bus Error Interrupt Enable （总线错误中断使能）：在发生总线错误时生成 I^2C 中断。

Stop Error Interrupt Enable （停止错误中断使能）：在 I^2C 停止条件下使能 I^2C 中断。

Clock Rate[1,0]：在 50、 100 和 400 kbps 三种有效时钟速率间选择。

Enable Slave （使能从设备）：允许将 I^2C 硬件模块作为总线从设备使用。

表 4. I2C_SCR 资源：组 0 reg[D7] 状态控制寄存器

位	7	6	5	4	3	2	1	0
数值	Bus Error	NA	Stop Status	ACK out	Address	Transmit	Last Recd Bit (LRB)	Byte Complete

Bus Error （总线错误）：表示检测到总线错误条件。

Stop Status （停止状态）：表示检测到 I^2C 停止条件。

ACK out：指示 I^2C 模块对所收到的字节进行 ACK （确认 -1）或 NACK （不确认 -0）。

Address （地址）：所接收或传送的字节是一个地址。

Last Received Bit （LRB）（最后接收位）：传输序列中最后接收到的位 （位 9）的值，该值表示目标器件中的 ACK/NACK 状态。

Byte Complete （字节完成）：接收到 8 个数据位。在接收模式下，总线将停顿以等待 ACK/NACK 数据包。在发送模式下，会接收 ACK/NACK 数据包 （请参见 LRB），并且总线被停止以等待执行下一个操作。

表 5. I2C_DR 资源：组 0 reg[D8] 数据寄存器

位	7	6	5	4	3	2	1	0
数值	数据							

所接收或发送的数据。要发送数据，必须在写入 I2C_SCR 寄存器前加载此寄存器。从该寄存器中读取所接收的数据，其中可能包含地址或数据。

表 6. I2C_ADDR 资源：组 1 reg[AD] 数据寄存器

位	7	6	5	4	3	2	1	0
数值	数据							

I^2C 地址寄存器用于配置硬件地址自动比较功能，从而使微控制器避免了不必要的从设备请求干扰。仅在 CY8C21x45、 CY8C22x45 和 CY8C28xxx 系列 PSoC 器件的从设备模式下，硬件地址自动比较功能才可用。

版本历史记录

版本	创作者	说明
1.00	DHA	初始版本。
2.00	MYKZ	1. 更正了清除发布中断的方法。 2. 修复了在用户模块参数中保存的 I2C 地址，并更正了 I2C 地址常量的生成。

注意： PSoC Designer 5.1 在所有用户模块数据手册中都引入了 “ 版本历史 ”。该章节高度概括了当前和先前用户模块版本间的区别。

Copyright © 2012-2014 赛普拉斯半导体公司。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不会根据专利权或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC Designer™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标， PSoC® 是赛普拉斯半导体公司的注册商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码 （软件和 / 或固件）均归赛普拉斯半导体公司 （赛普拉斯）所有，并受全球专利法规 （美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和 / 或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定用途外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括 （但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理预计可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而导致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用于赛普拉斯软件许可协议的限制。