I²C 硬件模块数据手册 I2CSBUF V 1.00

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

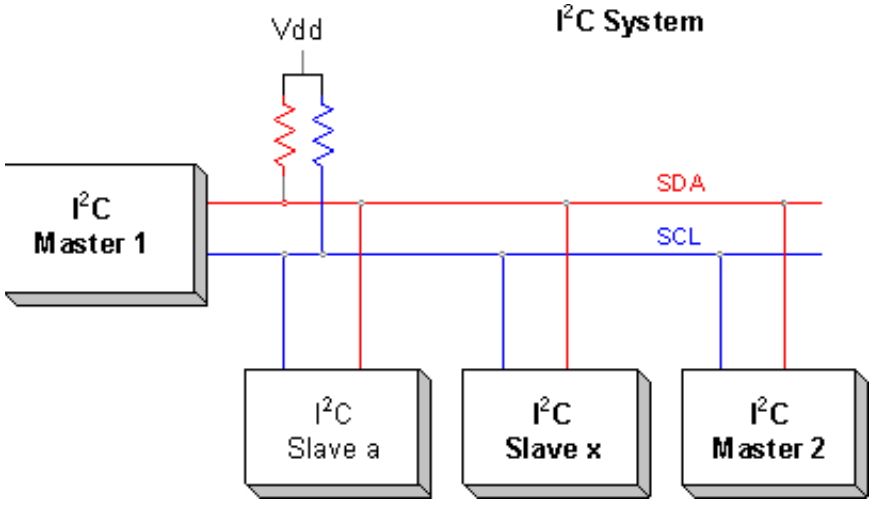
使用资源	PSoC® 模块				API 存储器 （字节）		每个传感器所使用的引脚数量
	CapSense®	I2C/SPI	定时器	电压比较器	闪存	RAM	
CY8C20xx7/S、 CY8C20055、 CY8C24093、 CY8C24393、 CY8C24293、 CY8C24693							
仅限从设备	–	1	–	–	213	1	2

特性与概述

- 工业标准 Philips I²C 总线兼容接口
- 仅适用于从设备的操作
- 只有两个引脚 （SDA 和 SCL）需要与 I²C 总线连接
- 标准数据速率为 50、 100、和 400 kbps
- 无需时钟延展
- 32 字节硬件数据缓冲区

I2CSBUF 用户模块实现了基于 I²C 寄存器的从设备。 I²C 总线是 Philips® （现更名为 NXP）开发的满足行业标准的两线硬件接口。主设备在 I²C 总线上启动所有通信，并为所有从设备提供时钟。 I2CSBUF 用户模块支持最高速度可达 400 kbps 的标准模式。 I2CSBUF 用户模块与同一总线上的多个器件兼容。

图 1. I²C 框图



功能说明

I2CSBUF 用户模块采用了一个简单易用的数据缓冲区，该模块同 I2C 从设备一样允许对缓冲区进行外部访问。 I2C 主设备可以对该缓冲器进行简单的读和写访问。由于缓冲区是在硬件中实现的，因此 I2C 从设备不需要时钟延展。

在每个地址或数据传输 / 接收结束时， PSoC 寄存器将报告传输 / 接收状态并且会触发专用中断。状态报告和中断生成取决于数据传输的方向和硬件检测到的 I2C 总线的条件。可以将中断配置为在完成字节、地址匹配和检测到停止条件时发生。

每个 I2C 数据操作由 “ 开始 ” （Start）、地址、读 / 写方向、数据和停止位组成。仅可以将该模块所使用的 I2C 资源作为 I2C 从设备进行操作。通信通过前台函数调用启动。

I2C 接口允许主设备按以下方式执行读和写操作。主设备通过设置基本数据指针启动操作。它通过传输从设备地址、写位以及所需的数据指针来执行该操作。这样会设置用户模块的数据指针，并确保硬件缓冲区内进行的读写操作开始于该位置。

为了进行读操作，主设备需要传输从设备地址，然后传输读取位。接下来，它会应答每个从设备的数据操作，直到完成读取所需字节为止，这时会取消应答并启动停止条件。下面显示的是一个读传输示例 （‘W’ 表示写入、‘R’ 表示读取、 05 为从设备地址、‘DP’ 表示数据指针、‘X’ 表示读取一个字节，另外 ‘P’ 表示停止条件）：

W 05 DP P

R 05 x x x P

为了写入到缓冲区，需要启动写通信并设置基本的数据指针 （I2C_BP）。然后，主机将会传输需要写入到 I2C 缓冲区内的数据字节，并在数据传输完成后启动停止条件。

下面显示的是一个写传输示例 （‘W’ 表示写入、‘R’ 表示读取、 05 为从设备地址、‘DP’ 表示数据指针、‘X’ 表示读取一个字节以及 ‘P’ 表示停止条件）

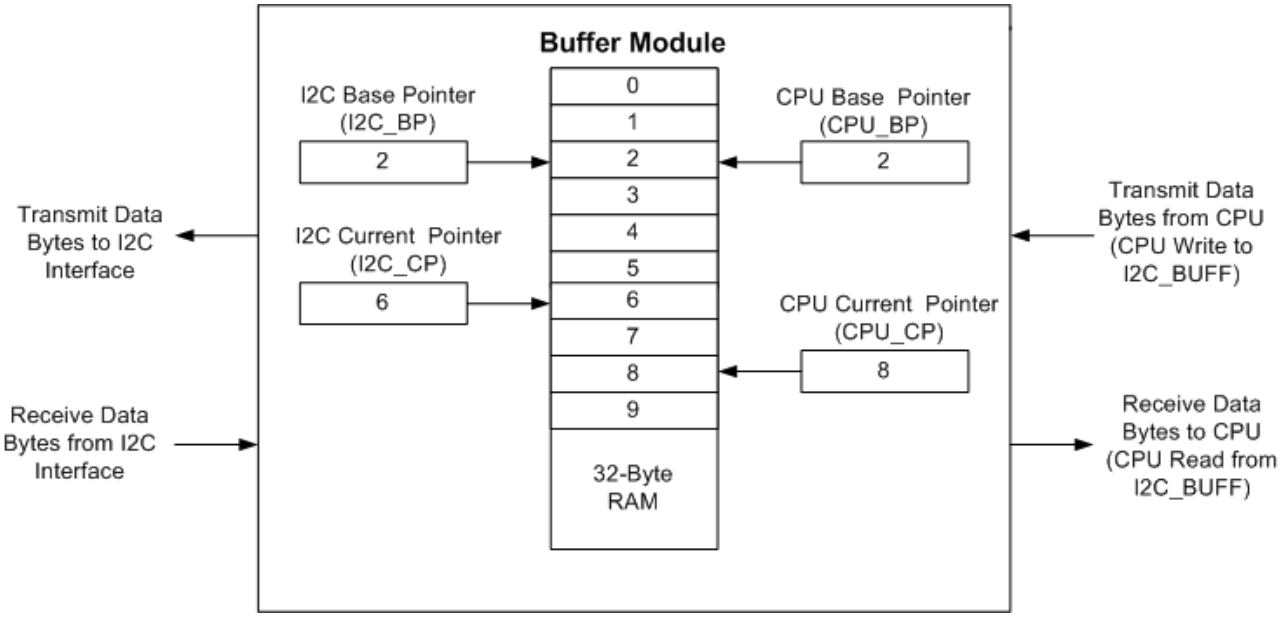
W 05 DP <Write Byte 1> <Write Byte 2> ... <Write Byte N> P

不管写入的是一个 RAM 字节还是多个 RAM 字节，第一个数据字节始终为基本的数据指针。有另外一个名称为当前数据指针的数据指针，它按每个读取或写入的字节递增 （I2C_CP），但每次发生地址匹配事件或写入基本数据指针时，该指针都被复位到基本数据指针。基本数据指针后的字节被写入到当前数据指针所指向的位置。第三个字节 （第二个数据字节）被写入到当前数据指针加 1 的位置，依此类推。新的读取操作开始将数据存储在基本数据指针所指向的位置 （地址匹配时，当前基本指针会复位到基本指针）。

例如，如果数据指针被设置为 4，则每个读取操作 （带有从设备地址的新读取操作）会将数据指针复位为 4，并从该位置连续进行读取，直至到达数据缓冲区的终端或主机完成读取操作为止。执行单一的读取操作和多个读取操作情况相同。在启动新写入操作前不会更改基本数据指针。

下图显示的是如何配置 I2CSBUF 用户模块中的地址指针。在该示例中，外部主设备发送启动条件、从设备地址和 2 字节数据，用以启动基本的地址指针 （I2C_BP 寄存器）和当前的地址指针 （I2C_CP 寄存器）。然后，写入 4 字节的数据，或同时发出启动 / 重启条件和设备地址，并读取 4 字节的数据。在 CPU 端，两个字节被写入到 CPU 基本地址寄存器 （CPU_BP 寄存器）内，并且 CPU 读取 I2C_BUFF 寄存器内 6 个连续字节或向该寄存器内写入 6 个连续字节。

图 2. EZI2C 模式中的地址指针



动态重配置

赛普拉斯强烈建议不要将 I2CSBUF 资源与动态加载 / 卸载层整合在一起。仅将 I2CSBUF 资源作为基本配置的组成部分。根据运行要求所述对 I2CSBUF 模块操作进行修改，但试图移除作为动态重新配置一部分的资源可能对外部 I^2C 器件造成不利影响。

I^2C 寻址

I^2C 地址包含在读取或写入数据操作的第一个字节的前 7 位内。I^2C 主设备使用该字节对从设备进行寻址。有效的地址范围为 0 到 127 （十进制）。字节的最低有效位包含 R/~W 位。如果该位为 0，将对该地址进行写操作；如果该位为 1，则已寻址的从设备将读取该位的数据。

在内部，用户模块会获取输入地址、移位，并将该地址同读 / 写位组合成为完整的地址字节。

示例

0x48 地址作为参数传递或被定义为从设备地址。传送一个包含读 / 写信息的单独参数。I^2C 主设备发送 0x90 字节 （8 位）以将数据写入到从设备，并发送 0x91 字节从从设备中读取数据。

因为从设备模块的地址参数可以是基于十进制的数值输入，所以采用了十进制键入 7 位地址 （十进制数 72）。

CPU 和 I2C 主设备会同时访问 I2C 缓冲区

不建议同时从外部主设备和 CPU 访问 I2C 硬件缓冲区，这是因为该操作会损坏数据。提供了信号量机制，用以避免 CPU 和 I2C 主设备同时访问 I2C 缓冲区。当 CPU 需要访问缓冲区 （读取或写入）时，它要检查主机是否正在访问该缓冲区。通过使用 ‘I2CSBUF_bCheckStatus’ API 可以在 ‘API.I2CSBUF_BUS_BUSY’ 标志中进行检查。总线繁忙标志在地址匹配时被置位，在停止条件下复位，或在启动条件下重复。在访问缓冲区前，如果总线处于空闲状态，则 CPU 将设置自动 NACK 模式，

用以防止主机访问缓冲区。如果主机尝试同时访问该缓冲区，则 I2C 模块将发出 NACK 信号。 CPU 访问该缓冲区后，自动 NACK 将被清除，而且缓冲区被释放以便主机能都对其进行访问。“ 示例固件源代码 ” 一节提供了一个代码段，通过该代码段可以有效地使用信号量机制。

睡眠模式和 I2C 操作

CY8C20xx7/7S 器件系列支持 I2C 从设备地址匹配事件的唤醒操作。当主机尝试访问 CY8C20xx7/7S 器件时，该特性可使器件从睡眠模式唤醒。为启动该特性，需要通过设置 SLP_CFG2 寄存器中的 I2C_ON 位来打开 I2C 模块的电源，并在进入睡眠模式前设置自动 NACK 特性。地址得到匹配时，器件将取消应答地址并从睡眠模式唤醒 （只有地址匹配时才会发生）。一旦将器件从睡眠模式唤醒，自动 NACK 功能被自动禁用。“ 示例 C 代码：使用 I2C 地址匹配事件从睡眠模式中唤醒 ” 一节提供了示例代码。如果在进入睡眠模式前未给 I2C 模块设置自动取消应答，那么 I2C 总线上的通信会被破坏。

直流和交流电气特性

欲了解 I^2C 接口的电气特性，请参考您 PSoC 器件的器件数据手册。

如该框图所示， 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决定。在输出阶段，当 V_{OLmax} = 0.4 V 时，所有器件 （主设备或从设备）的最小灌电流不小于 3 mA。这样可将 5 V 系统的最小上拉电阻值限制为 1.5 kΩ 左右。 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 5 V 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 NXP 网站：www.nxp.com。

注意： 从赛普拉斯或某个获得分许可的联营公司处购买 I^2C 组件，可根据赛普拉斯 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赛普拉斯所定义的 I^2C 标准规范。

放置

I2CSBUF 用户模块为 SCL 和 SDA 提供了两个选项。一个选项将 SCL 和 SDA 分别置于 P1[7] 和 P1[5] 上。另一个将 SCL 和 SDA 分别置于 P1[1] 和 P1[0] 上。如果允许，建议使用 P1[5]/P1[7] 引脚，因为 P1[0]/P1[1] 引脚被 ISSP 和 ECO 共享。不允许 I^2C 模块具有多种放置，这是因为 I^2C 模块使用了专用的 PSoC 资源模块和中断。

参数与资源

所有缓冲区均由 I^2C 主设备根据其用途进行命名。例如，名称或被描述为 “ 读取 ” 的缓冲区是由 I^2C 主设备读取的。

Slave_Addr

Slave_Addr 选择 I^2C 主设备所使用的 7 位从设备地址，以寻址从设备。有效选项为 0 至 127 （十进制）。

I2C 时钟

指定 I^2C 接口运行所需的时钟频率。存在三种时钟速率：

- 50 K 标准
- 100 K 标准
- 400 K 快速

I2C 引脚

从端口 1 选择用于 I²C 信号的引脚。 PSoC Designer 会自动选择驱动模式。因此，您可以将 I²C 时钟和数据信号路由到 P1[5] — P1[7] 或 P1[1] — P1[0]。

应用编程接口 （API）

API 函数是作为用户模块的一部分来提供的，通过该函数您能够以更高级别处理该模块。本节指定了每个函数的接口，以及引用文件所提供的相关常量。

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 会将 I2CSBUF_1 分配到已给项目中此用户模块的第一个实例。可将该名称更改为符合标识符语法规则的任意唯一值。所分配的实例名称作为每个全局函数名称、变量和常量符号的前缀。在下面说明中，为表达简单，将实例名称缩写为 I2CSBUF。

注意： ** 在这里，同所有用户模块 API 中的一样， A 和 X 寄存器的值可通过调用 API 函数来更改。如果调用后需要使用 A 和 X 的值，则调用函数将保留调用前 A 和 X 的值。选择这种 “ 寄存器易失 ” 规则是为了提高效率，自 PSoC Designer 1.0 版本起强制使用这种规则。 C 编译器自动满足该要求。汇编语言编程员也要保证他们的代码遵守该规则。虽然某些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来是否也被保留。

对于大型存储器模型器件，调用程序需要保留 CUR_PP、 IDX_PP、 MVR_PP 以及 MVW_PP 等寄存器中的所有值。尽管目前某些寄存器未被修改，但无法保证在将来的版本中也会如此。

I2CSBUF_Start

说明：

启动 I2C 硬件缓冲区从设备的操作。

C 语言原型：

```
void  I2CSBUF_Start(void)
```

汇编程序：

```
lcall I2CSBUF_Start
```

参数：

无

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_Stop

说明：

停止 I²C 硬件缓冲区从设备的操作。

C 语言原型：

```
void  I2CSBUF_Stop(void)
```

汇编程序：

```
lcall I2CSBUF_Stop
```

参数：

无

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_EnableInt

说明：

根据传递给 API 函数的参数使能 I^2C 中断。

C 语言原型：

```
void  I2CSBUF_EnableInt(BYTE bIntMask)
```

汇编程序：

```
mov   A, bIntMask
lcall I2CSBUF_EnableInt
```

参数：

bIntMask — 所需中断使能的标志组。下表列出了 C 语言和汇编语言中所提供的符号名称及其相关值。

符号名称	数值
I2CSBUF_INT_ADDR	0x01
I2CSBUF_INT_STOP	0x02
I2CSBUF_INT_ADDR_STOP	0x03
I2CSBUF_INT_BC	0x04
I2CSBUF_INT_ADDR_BC	0x05
I2CSBUF_INT_STOP_BC	0x06
I2CSBUF_INT_ADDR_STOP_BC	0x07

返回值：

无

其他影响：

请参见 API 章节开始部分的注释。

I2CSBUF_DisableInt

说明：

根据传递给 API 函数的参数禁用 I²C 中断。

C 语言原型：

```
void  I2CSBUF_DisableInt(BYTE bIntMask)
```

汇编程序：

```
mov   A, bIntMask
lcall I2CSBUF_DisableInt
```

参数：

bIntMask — 所需中断使能的标志组。下表列出了 C 语言和汇编语言中所提供的符号名称及其相关值。

符号名称	数值
I2CSBUF_INT_ADDR	0x01
I2CSBUF_INT_STOP	0x02
I2CSBUF_INT_ADDR_STOP	0x03
I2CSBUF_INT_BC	0x04
I2CSBUF_INT_ADDR_BC	0x05
I2CSBUF_INT_STOP_BC	0x06
I2CSBUF_INT_ADDR_STOP_BC	0x07

返回值：

无

其他影响：

请参见 API 章节开始部分的注释。

I2CSBUF_bGetAddress

说明：

返回用户模块的 7 位 I²C 地址。

C 语言原型：

```
BYTE  I2CSBUF_bGetAddress(void)
```

汇编程序：

```
lcall I2CSBUF_bGetAddress
```

参数：

无

返回值：

7 位 I²C 从设备地址值。

其他影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_SetAddress

说明：

将用户模块的 7 位 I²C 地址设置为用户定义值。

C 语言原型：

```
void  I2CSBUF_SetAddress(BYTE bAddress)
```

汇编程序：

```
mov   A, bAddress
lcall I2CSBUF_SetAddress
```

参数：

bAddress — 7 位 I²C 从设备地址值

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_SetAutoNACK

说明：

通过设置相应位可以选用 I²C 从设备模块的自动 NACK （也称为强制 NACK）特性。自动 NACK 可能不会立即被激活，只有达到下一个字节边界后才能激活它。

C 语言原型：

```
void  I2CSBUF_SetAutoNACK(void)
```

汇编程序：

```
lcall I2CSBUF_SetAutoNACK
```

参数：

无

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_ClearAutoNACK

说明：

通过清除相应位可以禁用 I²C 从设备模块的自动 NACK 特性。

C 语言原型：

```
void  I2CSBUF_ClearAutoNACK(void)
```

汇编程序：

```
lcall I2CSBUF_ClearAutoNACK
```

参数：

无

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_WriteBuffer

说明：

用户能够将已提供的各个数值写入到 I²C HW 缓冲区中特定的范围内。需要输入的设置内容为：缓冲区中需要写入的起始索引、需要写入的数据的长度以及指向存储器中某一位置的指针 （从该位置进行写操作）。

C 语言原型：

```
void  I2CSBUF_WriteBuffer(BYTE bBufIndex, BYTE * pWriteData, BYTE bDataLen)
```

汇编程序：

```
mov   A, bDataLen
push  A
mov   A, >pWriteData
push  A
mov   A, <pWriteData
push  A
mov   A, bBufIndex
push  A
lcall I2CSBUF_WriteBuffer
add   SP, -4
```

参数：

bBufIndex — 缓冲区中需要写入的起始索引。

pWriteData — 指向存储器中某个位置的指针 （从该位置开始进行写操作）。

bDataLen — 需要写入的数据的长度。

返回值：

成功写入到硬件缓冲区中的字节数量。

其他影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_bReadBuffer

说明：

您能够读取 I²C HW 缓冲区范围外某些空间内的值。需要输入的设置内容为：缓冲区中需要读取的起始索引、需要读取的数据的长度以及指向存储器中某个位置的指针 （从该位置进行读操作）。

C 语言原型：

```
BYTE  I2CSBUF_bReadBuffer(BYTE bBufIndex, BYTE *pReadData, BYTE bDataLen)
```

汇编程序：

```
mov   A, bDataLen
push  A
mov   A, >pReadData
push  A
mov   A, <pReadData
push  A
mov   A, bBufIndex
push  A
lcall I2CSBUF_bReadBuffer
add   SP, -4
```

参数：

bBufIndex — 缓冲区中需要读取的起始索引。

pWriteData — 指向存储器中某个位置的指针 （从该位置开始进行读取）。

bDataLen — 需要读取的数据的长度。

返回值：

从应用存储器的硬件缓冲区中成功读取的字节数量。

其他影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_WriteBufferByte

说明：

您能够将单字节写入到硬件缓冲区内。应该将输入设置为缓冲区中需要写入的索引和写入该位置的单个字节。

C 语言原型：

```
void  I2CSBUF_WriteBufferByte(BYTE bBufIndex, BYTE bData)
```

汇编程序：

```
mov   A, bBufIndex
mov   X, bData
lcall I2CSBUF_WriteBufferByte
```

参数：

bBufIndex — 缓冲区中需要写入的索引。

bData — 需要写入的数据字节。

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_bReadBufferByte

说明：

您能够从硬件缓冲区读取单个字节。输入为：缓冲区中需要读取的索引。将返回存储在该位置的字节。

C 语言原型：

```
BYTE  I2CSBUF_bReadBufferByte(BYTE BufIndex)
```

汇编程序：

```
mov   A, bBufIndex
lcall I2CSBUF_bReadBufferByte
```

参数：

bBufIndex — 缓冲区中需要读取的索引。

返回值：

需要读取的数据字节。

其他影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_bCheckStatus

说明：

返回指出 I2C 数据处理状态的值。

C 语言原型：

```
BYTE  I2CSBUF_bCheckStatus(void)
```

汇编程序：

```
lcall I2CSBUF_bCheckStatus
```

参数：

无

返回值：

I2C 数据处理的状态：当使用 I2C_XSTAT 寄存器时，一个字节的四位返回值显示的是以下信息的值。下面显示的是用于这些位的符号名称。

符号名称	值	说明
I2CSBUF_BUS_BUSY	0x80	只有在总线上运行 I2C 通信时才能设置该位。
I2CSBUF_AUTO_NACK_ON	0x04	仅在 I2C 模块使能自动 NACK 特性后，才能设置该位。
I2CSBUF_LAST_TX_RD	0x20	如果主机进行的最后 I2C 数据操作为读操作，将设置该位。
I2CSBUF_LAST_TX_WR	0x40	如果主机进行的最后 I2C 数据操作为写操作，将设置该位。

注意：如果 I2CSBUF_AUTO_NACK_ON 位被置位，那么所有其他状态位会被自动清除。

其他影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_ClearWriteStatus

说明：

清除 I2C 模块中的写状态位。如果 I2CSBUF_AUTO_NACK_ON 位被置位 （已调用 I2CSBUF_SetAutoNACK()），将自动清除写状态位。

C 语言原型：

```
void  I2CSBUF_ClearWriteStatus(void)
```

汇编程序：

```
lcall I2CSBUF_ClearWriteStatus
```

参数：

无

返回值：

无

其它影响：

请参见 API 章节开始部分的注释 **。

I2CSBUF_ClearReadStatus

说明：

清除 I2C 模块中的读状态位。如果 I2CSBUF_AUTO_NACK_ON 位被置位 （已调用 I2CSBUF_SetAutoNACK()），读状态位将被自动清除。

C 语言原型：

```
void  I2CSBUF_ClearReadStatus(void)
```

汇编程序：

```
lcall I2CSBUF_ClearReadStatus
```

参数：

无

返回值：

无

其它影响：

请参见 API 一节开始部分的注释 **。

示例固件源代码

下面显示的是 I2CSBUF 从设备用户模块的一个实现示例：

```
//-------------------------------------------------------
// Example code to demonstrate the use of the I2CSBUF UM
//
```

```
// This code sets up the I2CSBUF slave to expose 32 byte RAM buffer.
//
// * The instance name of the I2CSBUF User Module is "I2CSBUF".
//
//*************************************************************
#include <m8c.h>     // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

BYTE baI2CSBUFReadBuffer[8], baI2CSBUFWriteBuffer[] = {1,2,3,4,5,6,7,8};

void main(void)
{
    I2CSBUF_Start();
    I2CSBUF_DisableInt(I2CSBUF_INT_ADDR_STOP_BC);

    while(1)
    {
        // Check if last transaction was written from host
        if ((I2CSBUF_bCheckStatus() & I2CSBUF_LAST_TX_WR) && !(I2CSBUF_bCheckStatus()
        & I2CSBUF_BUS_BUSY))
        {
            // Engage the auto NACK from block to prevent the host access to buffer
            I2CSBUF_SetAutoNACK();

            // Check if the auto NACK is active, before accessing the buffer
            if (I2CSBUF_bCheckStatus() & I2CSBUF_AUTO_NACK_ON)
            {
                // Read out data from hardware buffer to RAM
                I2CSBUF_bReadBuffer(0, baI2CSBUFReadBuffer, 8);

                // Add user code here that acts upon data written by master,
                //for example:
                baI2CSBUFWriteBuffer[0] = baI2CSBUFReadBuffer[0] + 1;
                baI2CSBUFWriteBuffer[1] = baI2CSBUFReadBuffer[1] + 1;
                baI2CSBUFWriteBuffer[2] = baI2CSBUFReadBuffer[2] + 1;

                // Write data back to buffer to be read by master
                I2CSBUF_WriteBuffer(0, baI2CSBUFWriteBuffer, 8);
            }

            // Clear auto NACK to allow the host access to buffer
            I2CSBUF_ClearAutoNACK();
        }
    }
}
```

下面显示的是通过 I2C 地址匹配事件从睡眠模式唤醒的一个实现示例。

```
//*************************************************************
// Example code to demonstrate the use of the I2CSBUF UM with sleep mode
//
// This code sets up the I2CSBUF slave to expose 32 byte RAM buffer
// and demonstrate waking device up from I2C address match event
//
// * The instance name of the I2CSBUF User Module is "I2CSBUF"
```

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

void main (void)
{
    I2CSBUF_Start(); // Start I2C block
    I2CSBUF_EnableInt(0); //I2CSBUF_INT_ADDR_STOP_BC);
    // Enable I2C block interrupt for wake up from sleep

    M8C_EnableGInt; // Enable global interrupt

    SLP_CFG2 |= SLP_CFG2_I2C_ON; // Set bit to power i2C block during sleep.

    I2CSBUF_WriteBufferByte(0, 0); //Clear buffer[0]
    I2CSBUF_WriteBufferByte(1, 0); //Clear buffer[1]

    while(1) // infinite loop
    {
        // Check if last transaction was written from host
        if ((I2CSBUF_bCheckStatus() & I2CSBUF_LAST_TX_WR) && !(I2CSBUF_bCheckStatus()
        & I2CSBUF_BUS_BUSY))
        {
            if(I2CSBUF_bReadBufferByte(0)) //Check buffer[0]
            {
                I2CSBUF_WriteBufferByte(0, 0); //Clear buffer[0];

                I2CSBUF_SetAutoNACK(); // Set auto NACK and check if auto NACK is
                //engaged

                if (I2CSBUF_bCheckStatus() & I2CSBUF_AUTO_NACK_ON) M8C_Sleep;
                // Check if auto NACK is engaged

                I2CSBUF_WriteBufferByte(1, (I2CSBUF_bReadBufferByte(1) + 1));
                //Increment buffer[1]
            }
        }
    }
}
```

下面显示的是使用汇编代码编写的 I2CSBUF 从设备用户模块的一个实现示例：

```
;-----------------------------------------------------------------------
; Example code to demonstrate the use of the I2CSBUF UM
;
; This code sets up the I2CSBUF slave to expose 32 byte RAM buffer.
;
; * The instance name of the I2CSBUF User Module is "I2CSBUF".
;-----------------------------------------------------------------------

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

AREA I2CSBUF_RAM (RAM, REL, CON)

baI2CSBUFReadBuffer: blk 8
baI2CSBUFWriteBuffer: blk 8

AREA text (ROM, REL, CON)

_main:

    ; M8C_EnableGInt ; Uncomment this line to enable Global Interrupts
    lcall I2CSBUF_Start
    mov   A, I2CSBUF_INT_ADDR_STOP_BC
    lcall I2CSBUF_DisableInt

.mainloop:
    lcall I2CSBUF_bCheckStatus
    and   A, I2CSBUF_LAST_TX_WR
    jz    .mainloop
    lcall I2CSBUF_bCheckStatus
    and   A, I2CSBUF_BUS_BUSY
    jnz   .mainloop
    lcall I2CSBUF_SetAutoNACK
    lcall I2CSBUF_bCheckStatus
    and   A, I2CSBUF_AUTO_NACK_ON
    jz    .auto_nack_off

    mov   A, 8
    push  A
    mov   A, >baI2CSBUFReadBuffer
    push  A
    mov   A, <baI2CSBUFReadBuffer
    push  A
    mov   A, 0
    push A
    lcall I2CSBUF_bReadBuffer
    add   SP, -4

    RAM_SETPAGE_CUR >baI2CSBUFReadBuffer

    mov   A, [<baI2CSBUFReadBuffer]
    inc   A
    mov   [<baI2CSBUFWriteBuffer], A
    mov   A, [<baI2CSBUFReadBuffer+1]
    inc   A
    mov   [<baI2CSBUFWriteBuffer+1], A
    mov   A, [<baI2CSBUFReadBuffer+2]
    inc   A
    mov   [<baI2CSBUFWriteBuffer+2], A

    mov   A, 8
    push  A
```

```
    mov   A, >baI2CSBUFWriteBuffer
    push  A
    mov   A, <baI2CSBUFWriteBuffer
    push  A
    mov   A, 0
    push  A
    lcall I2CSBUF_WriteBuffer
    add   SP, -4

.auto_nack_off:
    lcall I2CSBUF_ClearAutoNACK
    jmp   .mainloop
.terminate:
    jmp .terminate
```

配置寄存器

本节介绍的是由 I2CSBUF 用户模块使用或修改的 PSoC 资源寄存器。

表 1. I2C_CFG 资源：组 0 reg[D6] 配置寄存器

位	7	6	5	4	3	2	1	0
值	保留	引脚选择	保留	停止 IE	时钟频率 [1:0]		保留	使能

引脚选择：该位被清除时， SDA 和 SCL 分别位于 P1[5] 和 P1[7] 上。该位被置位时， SDA 和 SCL 则分别位于 P1[0] 和 P1[1] 上。

停止错误中断使能：在 I^2C 停止条件下使能 I^2C 中断。

时钟频率 [1,0]：可选的三种有效时钟频率分别为 50、 100 和 400 kbps。

00b = 100 kHz

01b = 400 kHz

10b = 50k

11b = 保留

使能：使能 I^2C 硬件模块。

表 2. I2C_SCR 资源：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	保留	停止状态	ACK 输出	地址	发送	最后接收位（LRB）	字节完成

总线错误：表示检测到一个总线错误。

停止状态：表示检测到一个 I^2C 停止条件。

ACK 输出：指示 I^2C 模块应答 （1）或取消应答 （0）所收到的字节。

地址：所接收或传送的字节是一个地址。

传输：使用该位可设置后续字节传输的移位器方向。移位器始终移入 I^2C 总线上的数据，但对其写入 ‘1’ 会使能移位器的输出来驱动 SDA 输出。由于写入该寄存器会启动下一个传输，因此在写入该位前必须将数据写入到该数据寄存器内。在接收模式中 （0），执行本次写入操作前，必须从数据寄存器中读取之前接收的数据。固件从已接收从设备地址中的 RW 位中衍生出该方向。这个方向控制仅对数据传输有效。地址字节的方向由硬件确定。

最后接收位 （LRB）：传输序列中最后接收位 （位 9）的值，表示目标器件中 ACK/NAK 的状态。

字节完成：表示已经接收到 8 个数据位。在接收模式下，总线将暂停以等待某个 ACK/NAK 信号。在发送模式下，会收到 ACK NAK 信号 （参见 LRB），并且总线会处于停顿状态并等待下一个操作。

表 3. I2C_ADDR 资源：组 0 reg[CA] I2C 地址寄存器

位	7	6	5	4	3	2	1	0
值	保留	从设备地址						

这七位保留的是从设备的设备地址。

表 4. I2C_BP 资源：组 0 reg[CB] I2C 基本地址指针寄存器

位	7	6	5	4	3	2	1	0
值	EZ_RD_IE	EZ_WR_IE	CLK_STRETCH_EN	I^2C 基本指针				

I^2C 基本地址指针寄存器包含 RAM 数据缓冲区的基本地址值和其他控制位。

EZ_RD_IE：EZ_RD_STATUS 的中断使能位。

EZ_WR_IE：EZ_WR_STATUS 的中断使能位。

CLK_STRETCH_EN：在从睡眠模式转换到唤醒模式期间，通过该位配置时钟延展模式。

表 5. I2C_CP 资源：组 0 reg[CC] I2C 当前地址指针寄存器

位	7	6	5	4	3	2	1	0
值	保留	保留	保留	I^2C 当前指针				

设置该寄存器时，也会设置 I2C_BP 寄存器，并且两者的设置值相同。每次完成当前 I2C 数据操作的数据字节后，该寄存器的值将增加 1。该值始终用于确定数据的读取位置或写入位置。

表 6. CPU_BP 资源：组 0 reg[CD] CPU 基本地址指针寄存器

位	7	6	5	4	3	2	1	0
值	保留	保留	保留	CPU 基本指针				

该寄存器值完全由 CPU 所控制的 IO 写入控制。写入该寄存器时，也使用了相同的值来更新当前地址指针（即 CPU_CP）。对 I2C_ BUF 寄存器进行的第一个读或写操作都是从这个地址开始的。固件可确保从设备始终具有有效的数据，或者在被覆盖前读取数据。

表 7. CPU_CP 资源：组 0 reg[CE] CPU 当前地址指针寄存器

位	7	6	5	4	3	2	1	0
值	保留	保留	保留	CPU 当前指针				

设置该寄存器时，也会设置 CPU_BP 寄存器，并且两者的设置值相同。每次对 I2C_BUF 寄存器进行写或读操作， CPU_CP 都会自动递增。

表 8. I2C_BUF 资源：组 0 reg[CF] I2C 数据缓冲区寄存器

位	7	6	5	4	3	2	1	0
数值	数据缓冲区							

I^2C 数据缓冲区寄存器 （I2C_BUF）是 CPU 到数据缓冲区的读取和写入接口。每次读取该寄存器，都会返回 CPU 当前指针 （CPU_CP）所指位置的数据。同样，每次对该寄存器进行写操作，数据将被传输到缓冲区内，并被写入到 CPU 当前指针 （CPU_CP）所指的位置。每当读取该寄存器而没有通过 I^2C 或 CPU 接口来初始化 RAM 内容时，都不会返回任何有效值。

表 9. I2C_XCFG 资源：组 0 I2C 扩展控制寄存器

位	7	6	5	4	3	2	1	0
值	CSR_CLK_EN	保留	FORCE_NACK_MODE	FORCE_NACK	No BC Int	保留	缓冲区模式	HW Addr EN

I^2C 扩展控制寄存器 （I2C_XCFG）对增强型特性进行配置。当所有位被设置为默认复位状态 ‘0’ 时，模块将在兼容模式下工作。位 0 到位 7 （位 2 除外）都是可读写位。

表 10. I2C_XSTAT 资源：组 0 I2C 扩展状态寄存器

位	7	6	5	4	3	2	1	0
值	CSR_CLK_EN	保留	FORCE_NACK_MODE	FORCE_NACK	No BC Int	保留	缓冲区模式	HW Addr EN

I^2 扩展状态寄存器 （I2C_XSTAT）会读取增强型特性状态。所有位都是只读位。

版本历史记录

版本	创作者	说明
1.00	DHA	初始版本。
1.00.b	DHA	本用户模块数据手册中更新了 RAM 和 ROM 的用途。

注意： PSoC Designer 5.1 在所有用户模块数据手册中提供了版本历史记录，详细介绍了当前用户模块和之前用户模块之间的区别。

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。