SPI 主控数据表 SPIM V 2.6

Copyright © 2002-2011 Cypress Semiconductor Corporation. All Rights Reserved.

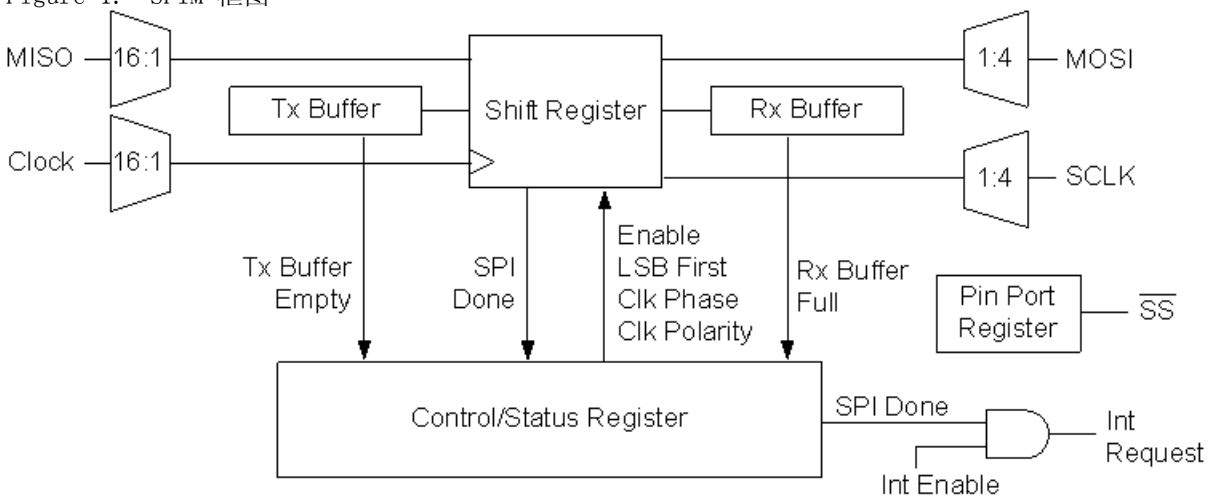
资源	PSoC® 模块			API 内存 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx、CY7C603xx、CY7C64215、CYWUSB6953、CY8C23x33、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、CY8CLED16P01、CYRF69xx3、 CY8C28xxx						
	1	0	0	27	0	3 - 4
CY8C26/25xxx	1	0	0	37	0	3 - 4

特性与概述

- 支持串行外设互连 (SPI) 主控协议
- 支持 SPI 时钟模式 0、1、2 和 3
- 可选输入源 （时钟和 MISO）
- 可选输出路由 （MOSI 和 SCLK）
- SPI 完成条件下的可编程中断
- 可独立选择 SPI 从器件

SPIM 用户模块是一种串行外设互连主控。 它执行全双工同步 8-bit 数据传输。 可指定 SCLK 相位、SCLK 极性和最低有效位优先，以适应大多数 SPI 时钟模式。 用户提供的软件可对从器件选择信号进行控制，从而将其配置为控制一个或多个 SPI 从器件。 SPIM PSoC 模块可为输入与输出信号提供可选路由，并拥有可编程中断驱动控制。

Figure 1. SPIM 框图



功能说明

SPIM 是一个实施串行外设互连主控的用户模块。 该模块使用数字通信型 PSoC 模块的 Tx 缓冲区寄存器、Rx 缓冲区寄存器、控制寄存器和移位寄存器以及一个或多个引脚端口寄存器。

可使用器件编辑器和 / 或 SPIM 用户模块固件应用程序编程接口 (API) 子程序对控制寄存器进行初始化与配置。 初始化包括设置低有效位优先配置和 SPI 发送 / 接收时钟模式。 支持 SPI 模式 0、1、2 和 3。 为了正常通信，必须为 SPI 主控和从器件设置相同的时钟模式和位配置。 SPI 模式按表 1 中所列内容定义：

Table 1. SPI 模式

模式	执行数据栓锁的 SCLK 边沿	时钟极性	注
0	上升沿	非反相	上升沿栓锁数据。 数据在时钟下降沿上发生更改。
1	上升沿	反相	
2	下降沿	非反相	下降沿栓锁数据。 数据在上升沿上发生更改。
3	下降沿	反相	

必须使用用户提供的软件子程序设置低电平有效从器件选择信号 ~SS，以控制所选引脚端口寄存器位，从而正确启用 SPI 从器件。 可配置一个或多个从器件选择信号，但一次只能有一个从器件选择信号处于活动状态。

为适应所有 SPI 时钟模式，对于从 SPI 主控发送至所选 SPI 从器件的数据，应当为每个字节置位并取消置位从器件选择信号。 然而，这并不是一项严格的要求，因为用于 SPI 主控和从器件之间 SPI 通信的特定字节传输协议有所不同。

SCLK 信号即为 SPI 发送 / 接收时钟。 其时钟频率为输入时钟信号频率的一半。 有效发送 / 接收比特率是输入时钟速率的一半。 使用器件编辑器指定输入时钟。

MOSI 信号是将数据从主控发送到符合 SPI 协议的从器件的主出从入 (Master-Out-Slave-In) 数据信号。 MISO 信号是将数据从 SPI 从器件发送到此用户模块的主入从出 (Master-In-Slave-Out) 数据信号。

SPIM 硬件在 MOSI 信号中发送来自 SPI 主控的数据，并同时在 MISO 信号中接收来自所选 SPI 从器件的数据。 使用同一个 SCLK 信号发送和接收主控和从器件的数据。

SPI 协议是仅主控起始响应协议。 主控负责确定所选从器件是否准备好接收或发送数据。

当使用 API 子程序在控制寄存器中设置 SPI 使能位时，SPIM 用户模块将可以运行。 此时，应将所有从器件选择信号置为高电平。

在向所选 SPI 从器件传送字节之前，应将指定的从器件选择信号置为低电平。

要发送的数据字节将会写入到 Tx 缓冲区寄存器中。 这将清除控制寄存器内的 “Tx 缓冲区空 ” 状态位。 在下一个时钟中，Tx 缓冲区中的数据将传输至移位寄存器，并将 “TX 缓冲区空 ” 状态位置位。 为防止发送过速情况出现，在将字节写入 Tx 缓冲区寄存器之前应检查 “ 缓冲区空 ” 状态位。 此时可将另一个要发送的数据字节写入 （预加载）至 Tx 缓冲区寄存器。 当前字节传送结束后，此数据将立即在下一个 SCLK 信号中传送。

将为移位寄存器中的每个数据字节位生成 SCLK 输出信号，移位寄存器中的数据将移至 MOSI 输出，并且来自 SPI 从器件的输入数据将从 MISO 输入移入移位寄存器。 SCLK、MOSI 和 MISO 信号的特定时序基于 SPI 时钟模式配置。

当所有位已完成传输并同时完成接收后，接收的数据将从移位寄存器传输到 Rx 缓冲区寄存器，Tx 缓冲区寄存器中的数据将传输到移位寄存器。 设置 “Rx 缓冲区满 ”(Rx Buffer Full) 和 “SPI 完成 ”(SPI Done) 状态位。 如果启用了中断，“SPI 完成 ”(SPI Done) 状态位将触发中断。

[+] Feedback

如果不使用 “SPI 完成 ”(SPI Done) 中断条件检索来自 Rx 缓冲区寄存器的数据字节，则应轮询控制寄存器来监控 “Rx 缓冲区满 ”(Rx Buffer Full) 状态位。 在完全接收下一个数据字节或设定 “ 过速错误 ” 状态位前，必须从 Rx 缓冲区寄存器中读取已接收的数据。

如果数据的待处理字节已预加载至 Tx 缓冲区寄存器，则此字节将重启 SPI 状态机并且立即开始传输。

在每个数据字节传输后，可部署两种方式来控制从器件选择信号：无中断级或中断级。 方式的选择取决于 SPI 时钟模式、字节传输协议以及已传输数据所需的比特率速度。 SPI 时钟模式 0 和 1 要求在传输下一个数据字节前将从器件选择信号关闭，然后再次打开。 对于向同一从器件进行多字节传输的情况，字节传输协议可能要求或者不要求用户切换从器件选择开关。 切换从器件选择开关可能还会产生从器件衍生，这种情况下则需要一段设置时间，以便使从器件能够在 SCLK 信号启动前在 MISO 信号中置位其数据。

在无中断级控制从器件选择信号，要求在传送 SPI 数据时使用专用微处理器来监控或者采样 “SPI 完成 ” 状态位。 如果数据比特率非常高 （约为 1 MHz 或者更快），监控状态位的系统开销将很有限。

对于较慢的比特率，监控 “SPI 完成 ” 状态位的开销可能阻碍微处理器进行任何其他所需操作，并在中断级执行从器件选择控制。 加上执行管理从器件选择指令所用的时间，最小中断延迟为 833 ns （时钟频率为 24 MHz 时）。

多字节数据传输 （其中 Tx 缓冲区预加载要发送的数据）可能是一个值得关注的问题。 然而，如果管理所选的从器件信号需要进行极少的跨字节处理或无需进行跨字节处理，并且对于同一个 SPI 从器件执行多字节传送时，则有可能实现 1 MHz 或更高的位传送速率。

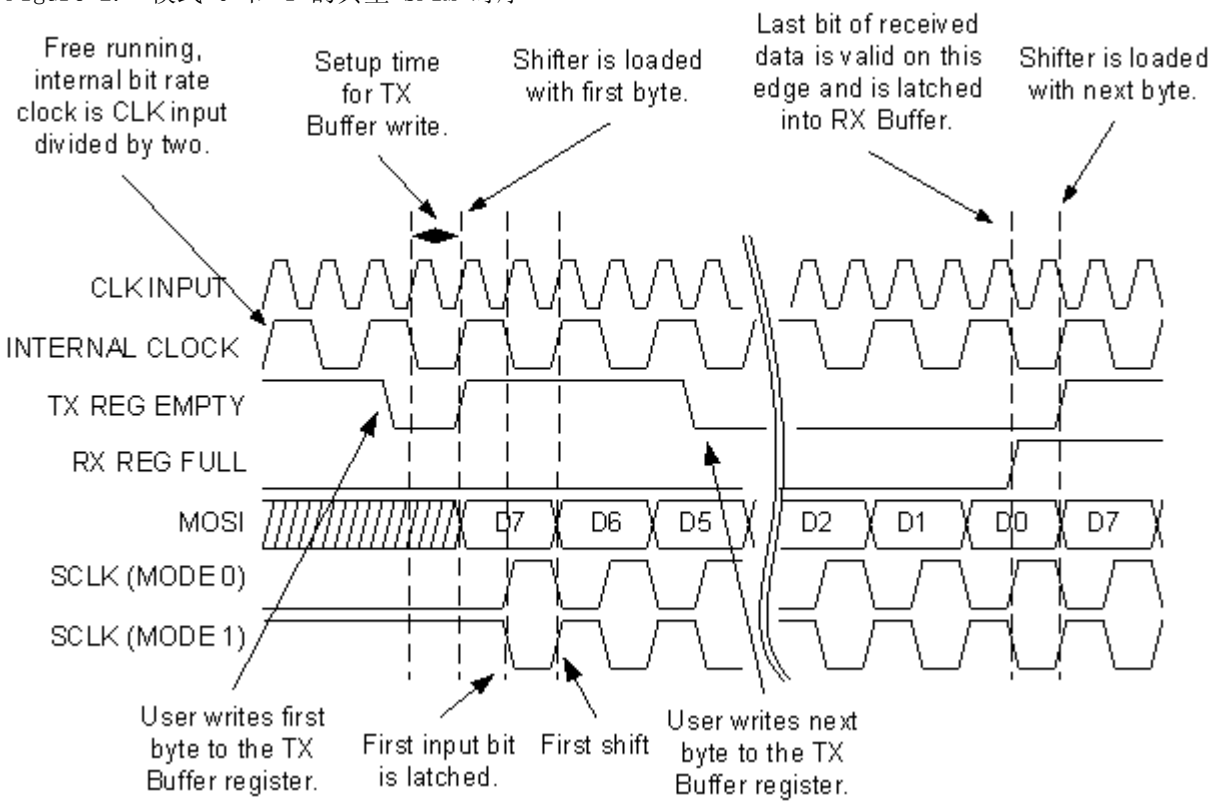
在最终数据传送完成后，应监控 “SPI 完成 ” 位以确定何时禁用 SPIM 用户模块。 这可以保证所有时钟信号已经在 SPI 主控与从器件之间完成。

[+] Feedback

时序

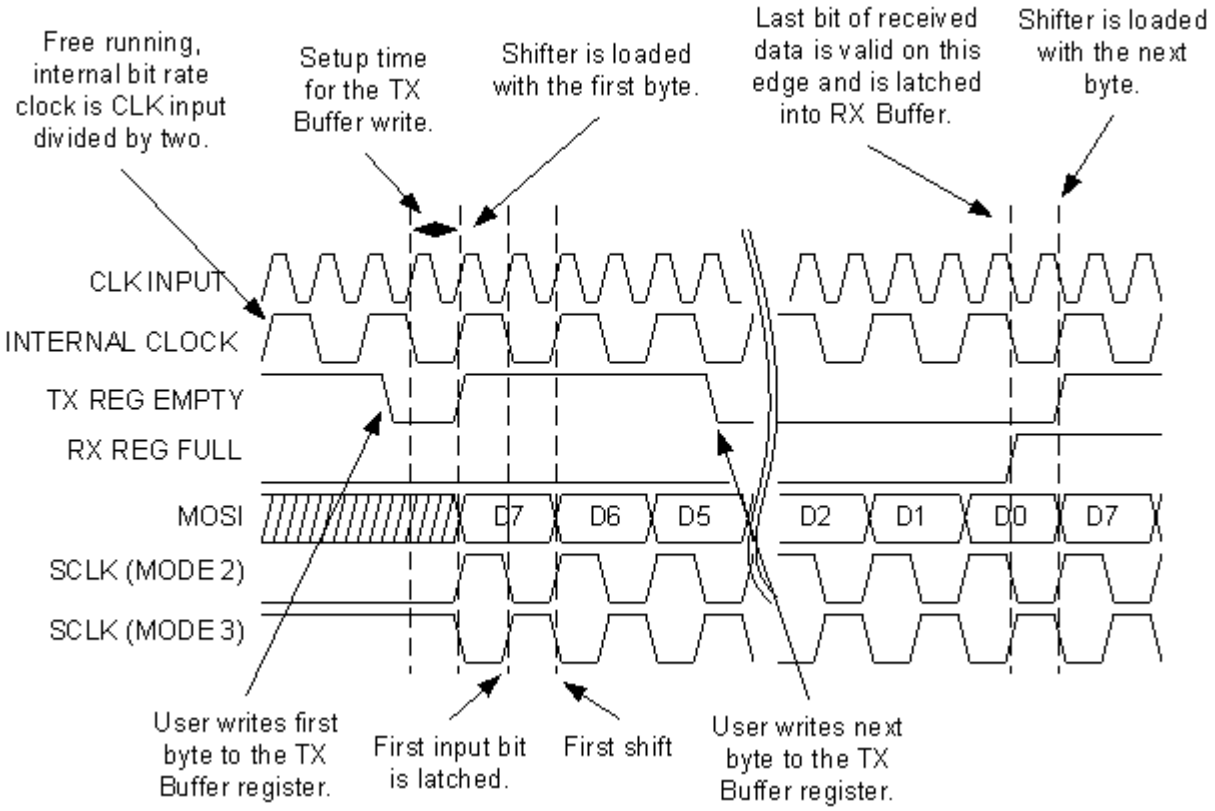
SPIM 传输的典型时序如下图所示。 更多 SPIM 时序信息请参见 《技术参考手册》。

Figure 2. 模式 0 和 1 的典型 SPIM 时序



[+] Feedback

Figure 3. 模式 2 和 3 的典型 SPIM 时序



直流和交流电气特性

Table 2. SPIM 直流和交流电气特性

参数	条件和注释	典型值	限值	单位
F_{max}	最大位速率	4.1[a]	--	Mbps

a. 典型速率适用于许多 PSoC 器件，但最大位速率因器件而异。 最大数据速率是最大时钟值的一半。 该值也可能会由于器件中断延迟而受到限制。 有关器件最大值的信息，请参见器件数据表。

放置

SPIM 可以映射到单个 PSoC 模块，并可置于任意数字通信模块内。

需要保留引脚端口寄存器位，以便从器件选择信号将其用于控制 SPI 从器件。 应将这些端口位作为标准 CPU 端口引脚进行配置。

[+] Feedback

参数和资源

时钟

SPIM 的时钟来自于 16 个可能来源之一。 全局 I/O 总线可用于将时钟输入连接至外部引脚或其他 PSoC 模块生成的时钟功能。 在模块中使用外部数字时钟时，为达到最高精度并使用睡眠操作应关闭行输入同步。 48 MHz 时钟、CPU_32 kHz 时钟、分频后时钟中的任一个（24V1 或 24V2）或者另一个 PSoC 模块输出均可以指定为 SPIM 时钟输入。

必须将时钟频率设置为所需位速率的两倍。 每两个输入时钟期间发送和 / 或接收一个数据位。

MISO

可将主入从出输入信号路由到 16 个可能来源之一。 允许的 MISO 源包括高电平、低电平、全局 I/O 总线、模拟比较器总线或者可指定为提供 MISO 输入的另一个 PSoC 模块。

MOSI

可将 SPIM 的主出从入输出路由到全局输出总线之一。 然后，该全局输出总线可连接至外部引脚或另一个 PSoC 模块，以进行下一步处理。

SCLK

此输出时钟由 SPI 主控生成。 通常情况下，它通过某条全局输出线路由到端口引脚，再连接到 SPI 从器件。 此时钟定义了有效位传送速率。

中断模式

此选项决定了何时为 TX 模块产生中断。 “TxRegEmpty” 选项可在数据从数据寄存器传输到移位寄存器之后立刻生成中断。 选择第二个选项 “TxComplete” 将会延迟中断，直到最后位移出移位寄存器。 如果需要知道字符在何时全部发送，第二个选项会很有帮助。 第一个选项 “TxRegEmpty” 最适合用于最大化发送器的输出。 此选项允许在前 1 个字节正在发送时加载 1 个字节。

ClockSync

在 PSoC 器件中，数字模块可以在系统时钟以外提供时钟源。 数字时钟源甚至可以用连锁方式串联起来。 这样就会引起与系统时钟相关的时滞。 这些时滞对于 CY8C29/27/24/22/21xxx 和 CY8CLED04/08/16 PSoC 器件系列具有更为关键的意义，因为其中存在各种数据路径优化，特别是那些应用于系统总线的部分。 此参数可用于控制时钟时滞，确保读取和写入 PSoC 模块寄存器值时进行正确操作。 此参数的正确数值应当由下表确定。

时钟同步值	使用说明
同步到 SysClk	此设置值用于任何由 24 MHz (SysClk) 经过 2 分频或更多分频所衍生出来的时钟源。 这样的时钟源包括 VC1、VC2、VC3 （在 VC3 由 SysClk 驱动时）、32KHz 和以 SysClk 作为时钟源的数字 PSoC 模块。 外部生成的时钟源也应使用此值来确保执行正确的同步操作。
Sync 到 SysClk*2	此设置可以适用于任何基于 48 MHz (SysClk*2) 的时钟，除非产生的频率为 48 MHz （换句话说，所有分频器的乘积为 1 时）。
使用 SysClk Direct	需要 24 MHz (SysClk/1) 的时钟时使用。 此项选择并不真正执行同步，但提供了对系统时钟本身的低时滞访问方式。 如果选择此项，则此选项将覆盖之前 “ 时钟 ”(Clock) 参数的设置。 在所有分频器组合起来最终生成了 24 MHz 的输出时，一定要使用此项，而不使用 VC1、VC2、VC3 或数字模块。
Unsynchronized	选中 48 MHz (SysClk*2) 输入时使用。 在需要未同步输入时使用。 一般来说，只有在中断生成是计数器的唯一应用时才推荐使用此选项。

[+] Feedback

InvertMISO

用户可选择此参数反转 MISO 输入。

时序

必须将时钟频率设置为所需位速率的两倍。

SPI 数据传送的时序也会影响由于处理从器件选择信号而造成的延迟。 考虑 SPI 从器件的选择信号建立时间十分重要。

中断产生控制 (Interrupt Generation Control)

当选中 PSoC Designer 中的 **“ 启用中断产生控制 ”** 复选框时，有两个附加参数将变为可用。 此复选框位于 **“ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。**. 当拥有多个程序层而且多个用户模块在不同的程序层共享中断时，中断生成控制是非常重要的：

- 中断 API (Interrupt API)
- IntDispatchMode

InterruptAPI

InterruptAPI 参数允许有条件地生成一个用户模块的中断处理程序和中断矢量表入口。 选择 “ 启用 ”(Enable) 可生成中断处理程序和中断矢量表条目。 选择 “ 禁用 ”(Disable) 可取消生成中断处理程序和中断矢量表条目。 在那些拥有多个程序层而且有多个程序层使用同一模块资源的项目中，特别推荐要正确地选择是否要生成中断 API。 仅在必要时选择生成中断 API，这样可以避免生成中断调度代码，从而减少开销。

IntDispatchMode

IntDispatchMode 参数用于指定如何处理中断请求（当处于同一模块的多个用户模块在不同的程序层共享该中断时）。 选择 “ActiveStatus” 会导致固件在为共享的中断请求提供服务之前测试哪一个外覆层正处于活动状态。 这项测试发生在每次请求共享中断的时候。 这样会导致延迟增加并且会产生一个服务于共享中断请求的非确定性的程序，但并不要求任何 RAM 资源。 选择 “OffsetPreCalc” 会导致固件在最初只有一个外覆层载入时计算共享中断请求的来源。 这项计算减少了中断延迟会产生一个服务于共享中断请求的确定性程序，但其代价是 1 个字节的 RAM 资源。

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，使设计人员能够在较高的层级处理模块。本部分具体说明了每个函数对应的接口以及 “include” 文件所提供的相关常量。

每次布置用户模块时，都会为其分配一个实例名称。 默认情况下，PSoC Designer 将会为给定项目中此用户模块的第一个实例分配 SPIM_1。 可将该值更改为符合标识符语法规则的任意唯一值。 分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。 为简便起见，在以下说明中将实例名称缩写为 SPIM。

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能因调用 API 函数而更改。 发起调用 API 的函数有责任在调用之前保留 A 和 X 的值 （如果调用后需要再次用到它们）。 选择这种 “ 寄存器易变 ” 策略是为了提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。 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。

本节列出了 SPIM 提供的 API 函数。

[+] Feedback

SPIM_Start

说明：

通过在控制寄存器中设置正确的位来设置 SPI 接口的模块配置，并启用 SPIM 模块。 在调用此函数之前，应将所有的从器件选择信号置为高电平，以取消选中连接的 SPI 从器件。 此操作应在用户提供的子程序中完成。

C 原型：

```
void  SPIM_Start(BYTE bConfiguration)
```

汇编：

```
mov   A,SPIM_MODE_2 | SPIM_LSB_FIRST
lcall  SPIM_Start
```

参数：

bConfiguration: 用于指定 SPI 模式和最低有效位优先配置的一个字节。 它在累加器内进行传递。 下表给出了在 C 语言和汇编语言中提供的符号名及其相关值。 请注意，符号名称可以通过 “ 或 ” 运算结合起来形成 SPI 接口的配置。 而且还要注意，用户模块的实例名称加在下表列出的符号名称前面。 例如，如果在放置用户模块时将其命名为 SPIM1，那么第一个模式的符号名称为 SPIM1_SPIM_MODE_0.

符号名	值
SPIM_MODE_0	0x00
SPIM_MODE_1	0x02
SPIM_MODE_2	0x04
SPIM_MODE_3	0x06
SPIM_LSB_FIRST	0x80
SPIM_MSB_FIRST	0x00

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

SPIM_Stop

说明：

可通过在控制寄存器中清除启用位来禁用 SPIM 模块。 调用此函数后，必须将所有的从器件选择信号置为高电平，以禁用所有连接的 SPI 从器件。 此操作应在用户提供的子程序中完成。

C 原型：

```
void  SPIM_Stop(void)
```

汇编：

```
lcall  SPIM_Stop
```


[+] Feedback

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

SPIM_EnableInt

说明：

在 “SPI 完成 ” 条件下启用 SPIM 中断。 SPIM 的放置位置决定了特定中断矢量和优先级。

C 原型：

```
void  SPIM_EnableInt(void)
```

汇编：

```
lcall  SPIM_EnableInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

SPIM_DisableInt

说明：

在 “SPI 完成 ” 条件下禁用 SPIM 中断。

C 原型：

```
void  SPIM_DisableInt(void)
```

汇编：

```
lcall  SPIM_DisableInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

SPIM_SendTxData

说明：

启动到 SPI 从器件的 SPI 传输。 在进行此调用之前，必须将指定的 SPI 从器件的信号置为低电平。 此操作应在用户提供的子程序中完成。

C 原型：

```
void  SPIM_SendTxData(BYTE bSPIMData)
```

汇编：

```
mov   A, bSPIMData
lcall  SPIM_SendTxData
```

参数：

BYTE bSPIMData： 要发送到 SPI 从器件的数据。 它在累加器内进行传递。

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

SPIM_bReadRxData

说明：

返回在从器件中接收到的数据字节。 调用此子程序之前应检查 “Rx 缓冲区已满 ” 标志，确认已收到数据字节。

C 原型：

```
BYTE  SPIM_bReadRxData(void)
```

汇编：

```
lcall  SPIM_bReadRxData
mov   bRxData, A
```

参数：

无

返回值：

在从器件 SPI 中接收并在累加器中返回的数据字节。

副作用：

此函数可能更改 A 和 X 寄存器。

[+] Feedback

SPIM_bReadStatus

说明：

读取并返回当前的 SPIM 控制 / 状态寄存器。

C 原型：

```
BYTE  SPIM_bReadStatus(void)
```

汇编：

```
lcall  SPIM_bReadStatus
and   A, SPIM_DONE | SPIM_RX_BUFFER_FULL
jnz   SpimCompleteGetRxData
```

参数：

无

返回值：

返回状态字节读取内容并在累加器中返回。 利用定义的掩码来测试特定的状态条件。 请注意，这些掩码可以通过 “ 或 ” 运算结合起来测试多种条件。 而且还要注意，用户模块的实例名称加在下表列出的符号名称前面。 例如，如果在放置用户模块时将其命名为 SPIM1，那么第一个掩码的符号名称为 SPIM1_SPIM_SPI_COMPLETE.

SPIM 状态掩码	值
SPIM_SPI_COMPLETE	0x20
SPIM_RX_OVERRUN_ERROR	0x40
SPIM_TX_BUFFER_EMPTY	0x10
SPIM_RX_BUFFER_FULL	0x08

副作用：

调用此函数之后将会清除状态位。 此函数可能更改 A 和 X 寄存器。

固件源代码示例

在下列 C 语言和汇编语言示例代码中，SPI 主控发送存储在 RAM 中以零为结尾的字符串。 此函数使用 SPIM API 将字节逐一加载到数字 PSoC 模块 TX 寄存器中。 数字模块通过将其 TX 寄存器传输到其移位寄存器开始进行发送。 重复 API 调用，此函数立即使用下一个字节重新加载发送寄存器。 这样一来，发送操作将以最大持续速率进行。 在这个简单版本中，从该从器件中接收的任何数据都会被丢弃。 此函数在最后一个字节已在 TX 寄存器中加载，而倒数第二个字节仍在 MOSI 线路上移出时返回。 对此函数的后续调用不会影响正在进行的发送操作，因为在将值加载到 TX 寄存器之前始终对状态进行检查。

```
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

CHAR Message1[] = "Hello World.";
CHAR *pbStrPtr = Message1;

void main(void)
{
    SPIM_Start(SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST);

    while( *pbStrPtr != 0 )  /* While data remains to be sent */
    {
        /* Ensure the transmit buffer is free */
        while( ! (SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY ) );

        SPIM_SendTxData( *pbStrPtr );  /* load the next byte */
        pbStrPtr++;
    }
}
```

写入汇编语言的等效代码为：

```
AREA bss (RAM, REL)
Message1: blk 13
AREA text (ROM, REL)

_main:

    ; [...]                    ; ( String is copied to RAM )

    mov   A, SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST
    call  SPIM_Start           ; Initialize the digital PSoC Block
    mov   X, Message1          ; Set index to beginning of string.
    ; [...]                    ; Set SlaveSelect signal low to enable slave)
.TxNextByteLoop:
    mov   A, [X+0]             ;    Data remaining to be sent? ...
    jz    .Finish              ;    No, bail out of the loop.
.WaitForTxEmpty:
    call  SPIM_bReadStatus     ;   Transmit buffer free? ...
    and   A, SPIM_SPIM_TX_BUFFER_EMPTY
    jz    .WaitForTxEmpty      ;   No,  keep checking the status
    mov   A, [X+0]             ;   Reload next value into A
    call  SPIM_SendTxData      ;   Yes, load TX with the next byte, ...
    inc   X                    ;        Advance the pointer, ...
    jmp   .TxNextByteLoop      ;        and repeat until done.
.Finish:                       ;   Transmit complete!
```


[+] Feedback

```
    call  SPIM_bReadStatus      ;   Buffer empty for last time? ...
    and   A, SPIM_SPIM_TX_BUFFER_EMPTY
    jz    .Finish               ;   No,  keep checking the status
.WaitForTxComplete:
    call  SPIM_bReadStatus      ;   Last byte transmission complete? ...
    and   A, SPIM_SPIM_SPI_COMPLETE
    jz    .WaitForTxComplete    ;   No,  keep checking the status
    ; [...]                     ;   (Reset SlaveSelect signal back high)
.AllDone:
    ; Endless loop
    jmp   .AllDone
```

配置寄存器

用于配置此用户模块的 A 型数字通信 PSoC 模块寄存器描述如下。 仅解释参数化符号。

Table 3. SPIM 模块，寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	1	1	0

此寄存器定义将此 “A” 型数字通信模块配置为 SPIM 用户模块。

Table 4. SPIM 模块，寄存器： 输入

位	7	6	5	4	3	2	1	0
值	MISO				时钟			

MISO 是主入从出输入信号。 时钟是用于驱动 SPIM 时序所选择的时钟。 二者均在参数选择过程中使用器件编辑器进行设置。

Table 5. SPIM 模块，寄存器： 输出

位	7	6	5	4	3	2	1	0
值	0	0	SCLK			MOSI		

MOSI 是主出从入输出信号。 SCLK 是 SPI 时钟信号。 二者均在参数选择过程中使用器件编辑器进行设置。

Table 6. SPIM 模块，移位寄存器： DR0

位	7	6	5	4	3	2	1	0
值	移位寄存器							

此移位寄存器是 SPI 移位寄存器。

Table 7. SPIM 模块，TX 数据缓冲区寄存器： DR1

位	7	6	5	4	3	2	1	0
值	TX 缓冲区寄存器							

[+] Feedback

Tx 缓冲区寄存器： 启用 PSoC 模块后，写入此缓冲区的数据将传输到移位寄存器。

Table 8. SPIM 模块，RX 数据缓冲区寄存器： DR2

位	7	6	5	4	3	2	1	0
值	RX 缓冲区寄存器							

RX 缓冲区寄存器： 在此移位寄存器中接收的数据在 SPI 发送周期结束后传输到此寄存器。

Table 9. SPIM 模块，控制寄存器： CR0

位	7	6	5	4	3	2	1	0
值	最低有效位优先	RX 过速错误	SPI_COMPLETE	TX 缓冲区为空	Rx 缓冲区已满	时钟相位	时钟极性	SPIM 启用

最低有效位优先指定最低有效位应优先发送。

“Rx 过速错误 ” 标志表示接收到新字节时尚未读取之前接收到的数据字节。

“SPI 完成 ” 标志表示完整的 SPI 传输 / 接收循环已完成。

“Tx 缓冲区为空 ” 标志表示 Tx 缓冲区为空。

“Rx 缓冲区已满 ” 标志表示已从移位寄存器接收到数据字节。

“ 时钟相位 ” 表示 SCLK 信号的相位。 它是定义 SPI 模式的参数之一。

“ 时钟极性 ” 表示 SCLK 信号的极性。 它是定义 SPI 模式的参数之一。

“SPIM 启用 ” 经设置后可启用 SPIM PSoC 模块。

[+] Feedback

版本历史记录

版本	创作者	说明
2.6	TDU	更新了时钟说明，内容包括： 在模块中使用外部数字时钟时，为达到最高精度并使用睡眠操作应关闭行输入同步。

Note PSoC Designer 5.1 在所有用户模块数据表中引入了版本历史。 此部分记录了当前用户模块版本和以前用户模块版本之间区别的高级描述。

Copyright © 2002-2011 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.