SPI 主控数据表 SPIM V 2.5

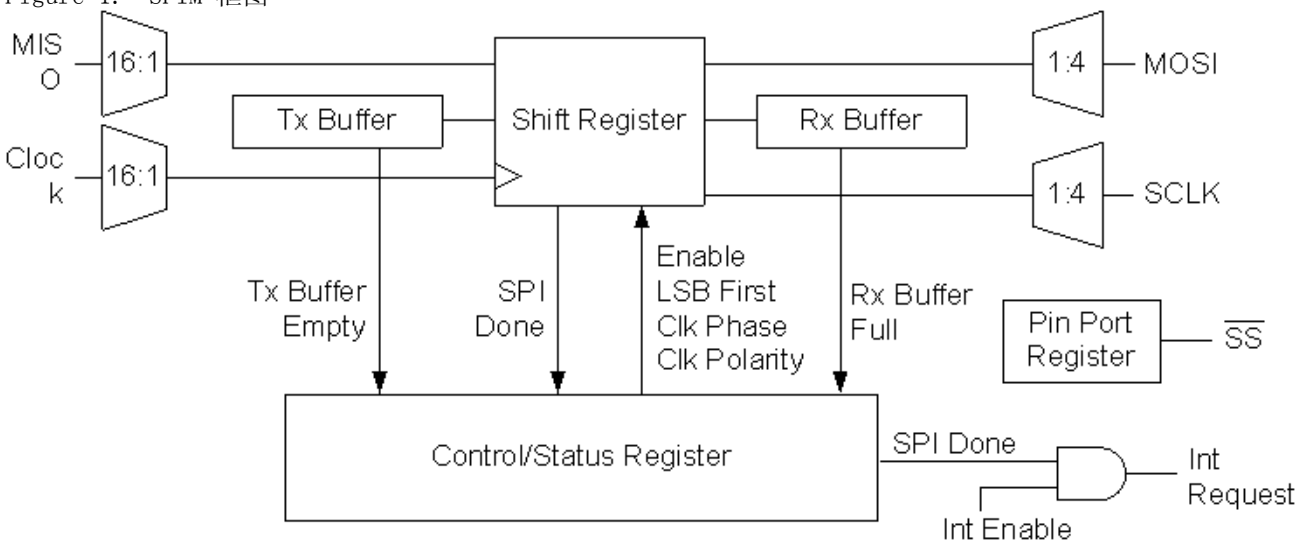
资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	CapSense®	I2C/SPI	定时器	闪存	RAM	
CY8C20x34、CY8C20x24、CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C604xx、CY7C643xx、CYONS2010、CYONS2011、CYONSFN2051、CYONSFN2053、CYONSFN2061、CYONSFN2151、CYONSFN2161、CYONSFN2162、CY8CTST200、CY8CTMG2xx						
		X		27	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）。
- 基于 SPI 完成条件的可编程中断。
- 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。 通过用户提供的软件进行控制，从器件选择信号能够控制一个或多个 SPI 从器件。 SPIM PSoC 模块的输入和输出信号具有可选路由以及可编程中断驱动控制。

Figure 1. SPIM 框图



功能说明

SPIM 是一种实现串行外设互连主控的用户模块。 它使用 I2C/SPI PSoC 模块的 Tx 缓冲区、 Rx 缓冲区、控制和配置寄存器、用于数据移位的数据寄存器以及一个或多个引脚端口寄存器。

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) 子程序初始化并配置控制寄存器。 初始化包括设置最低有效位优先配置和 SPI 传输 / 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 模式定义如下。

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	同相	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	同相	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有效低电平从器件选择信号 (~SS)，以控制所选的引脚端口寄存器位，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，但是每次只有一个从器件选择信号有效。

为适应所有 SPI 时钟模式，对于从 SPI 主控传输至所选 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 信号是 SPI 传送和接收时钟。 它是输入时钟信号的时钟速率的一半。 有效传送和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 “ 主出从入 ” 数据信号，即将数据从主控传送至符合 SPI 协议的从器件。 MISO 信号是 “ 主入从出 ” 数据信号，即将数据从 SPI 从器件传送至此用户模块。

SPIM 硬件在 MOSI 信号中传送来自主 SPI 器件的数据，并同时在 MISO 信号中接收来自所选从 SPI 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 协议是仅主控启动响应协议。 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。

已传送的数据字节写入至 Tx 缓冲区寄存器。 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 缓冲区寄存器之前将检查 “ 缓冲区空 ” 状态位。 此时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，并且来自从器件 SPI 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。

传送并同时接收完所有位之后，已接收数据从移位寄存器传输至 Rx 缓冲区寄存器，并且将 Tx 缓冲区寄存器传输至移位寄存器。 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。

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，则轮询控制寄存器来监控 “Rx 缓冲区满 ”(Rx Buffer Full) 状态位。 在下一个数据字节完全接收或者设置过速错误状态位之前，读取来自 Rx 缓冲区寄存器的已接收数据。

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，可通过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，监控 “SPI 完成 ” 位以确定何时禁用 SPIM 用户模块。 这样可以确保 SPI 主控和从器件之间的所有时钟信号都是完整的。

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	极限值	单位
F_{max}	最大位速率	--	12	MHz

放置

SPIM 可以映射到单个 PSoC 模块，并可置于 I2C/SPI 模块内。

保留引脚端口寄存器位，以供从器件选择信号用于控制 SPI 从器件。 将这些端口位配置为标准 CPU 端口引脚。

时序

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很重要。

参数和资源

中断模式

此选项决定何时对 TX 模块产生中断。 “TxRegEmpty” 选项会在数据从数据寄存器传输至移位寄存器后立即产生中断。 选择第二个选项 “TxComplete” 时会延迟中断，直到最终位移出移位寄存器。 第二个选项在需要了解数据字节何时完全发送的情况下很有用。 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。

时钟选择

此参数用于设置 SPI 主控的工作频率。 它基于 “ 全局资源 ”(Global Resources) 中的 SysClk 设置。 例如，如果通过在 “ 全局资源 ”(Global Resources) 中选择 “3.3V/12MHz” 或 “5.0V/12MHz” 的电源设置将 SysClk 频率设置为 12 MHz，并在用户模块参数中选择 “SysClk/4” 的时钟选择，则 SPI 主控的位速率为 3MBits/ 秒。 “ 全局资源 ”(Global Resources) 和 “ 用户模块参数 ”(User Module Parameters) 位于器件编辑器的 “ 互连视图 ”(Interconnect View) 中。

“ 时钟选择 ”(Clock Select) 选项包括：

SysClk/2

SysClk/4

SysClk/8

SysClk/16

SysClk/32

SysClk/64

SysClk/128

SysClk/256

数据顺序

此选项决定串行数据移出的方式，即最低有效位 (LSB) 优先还是最高有效位 (MSB) 优先。

中断生成控制

下列参数仅在选中了 PSoC Designer 中 “ 启用中断生成控制 ”(Enable interrupt generation control) 复选框时可用。该复选框位于 “ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。 **“ 启用中断生成控制 ”** 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。.**

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) 子程式作为用户模块的一部分提供，使您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。

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）。 为实现高效，选择了这项 “ 寄存器为易失性 ” 的策略，该策略从 PSoC Designer 1.0 版本开始实施。 C 语言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。 虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

以下列出了 SPIM 提供的 API 函数：

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，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。 在用户提供的子程式中执行此操作。

C 语言原型：

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

汇编语言：

```
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 ”(LSB First) 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称及其相关数值。 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.

符号名称	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：

此函数可能会修改 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，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。 在用户提供的子程式中执行此操作。

C 语言原型：

```
void  SPIM_Stop(void)
```

汇编语言：

```
lcall  SPIM_Stop
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 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 语言原型：

```
void  SPIM_EnableInt(void)
```

汇编语言：

```
lcall  SPIM_EnableInt
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIM_DisableInt

说明：

在 “SPI 完成 ” 条件下禁用 SPIM 中断。

C 语言原型：

```
void  SPIM_DisableInt(void)
```

汇编语言：

```
lcall  SPIM_DisableInt
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIM_SendTxData

说明：

对 SPI 从器件启动 SPI 发送操作。

在进行此调用之前，将指定的 SPI 从器件的信号置为低电平。 在用户提供的子程式中执行此操作。

C 语言原型：

```
BOOL  SPIM_SendTxData(BYTE bSPIMData)
```

汇编语言：

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

参数：

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

返回值：

无

副作用：

此函数可能会修改 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。 在调用该子程式之前检查 “Rx 缓冲区满 ” 标志，以验证数据字节是否已接收到。

C 语言原型：

```
BYTE  SPIM_bReadRxData(void)
```

汇编语言：

```
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：

此函数可能会修改 A 和 X 寄存器。

SPIM_bReadStatus

说明：

读取并返回当前的 SPIM 控制 / 状态寄存器。

C 语言原型：

```
BYTE  SPIM_bReadStatus(void)
```

汇编语言：

```
lcall  SPIM_bReadStatus
and    A, SPIM_SPI_COMPLETE | 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。

使用定义的掩码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。 此函数可能会修改 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。 I2C/SPI 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。 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 Message[] = "Hello World.";
CHAR *pbStrPtr = Message;

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：

```
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 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!
    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

用来配置该用户模块的 I2C/SPI PSoC 模块寄存器在此处进行描述。 只对经过参数化的符号进行了解释。

Table 3. SPIM 模块，配置寄存器

位	7	6	5	4	3	2	1	0
值	Clock Sel			旁路	SS_	SS_EN_	Int Sel	从器件

Clock Sel - 时钟选择。 这些位决定 SPI 主控的工作频率。

000b - SysClk / 2

001b - SysClk / 4

010b - SysClk / 8

011b - SysClk / 16

100b - SysClk / 32

101b - SysClk / 64

110b - SysClk / 128

111b - SysClk / 256

旁路 - 旁路同步。 此位决定输入是否同步到 SYSCLK。

SS_ - 从器件选择。 此位决定当 SS_EN_ 信号激活 (SS_EN_ = 0) 时，SS_ 信号的逻辑值。

SS_EN_ - 从器件选择使能。 此有效低位决定从器件选择 (SS_) 信号是否为内部驱动。 如果是内部驱动，则其逻辑电平由 SS_ 位决定。 如果是外部驱动，则其逻辑电平由外部引脚决定。

Int Sel - 中断选择。 此位选择产生中断的条件，无论它是基于 “TX 寄存器空 ” 条件还是 “SPI 完成 ” 条件。

从器件 - 此位决定模块用作主控还是从器件。

Table 4. SPIM 模块，TX 数据缓冲区寄存器

位	7	6	5	4	3	2	1	0
值	TX 缓冲区寄存							

Tx 缓冲区寄存器： 写入此缓冲区的数据在 PSoC 模块启用时传输到移位寄存器。

Table 5. SPIM 模块，RX 数据缓冲区寄存器

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 6. SPIM 模块，控制寄存器

位	7	6	5	4	3	2	1	0
值	最低有效位优先	RX 过速错误	SPI_ 完成	TX 缓冲区空	RX 缓冲区满	时钟相位	时钟极性	使能

最低有效位优先 – 指定最低有效位应优先发送。

Rx 过速错误 – 指示在接收下一个字节之前未读取先前接收到的数据字节的标志。

SPI 完成 – 指示完整 SPI 发送 / 接收周期结束的标志。

Tx 缓冲区空 – 指示 Tx 缓冲区为空的标志。

Rx 缓冲区满 – 指示从移位寄存器接收到一个数据字节的标志。

时钟相位 – 指示 SCLK 信号的相位。 它是定义 SPI 模式的参数之一。

时钟极性 – 指示 SCLK 信号的极性。 它是定义 SPI 模式的参数之一。

使能 – 设置时使能 SPI PSoC 模块。

版本历史记录

版本	创作者	说明
2.5	DHA	CY7C64315 和 CY7C64316 PSoC 设备的 P1[0] 设置为 SPI CLK 引脚。

Note PSoC Designer 5.1 在所有用户模块数据表中提供版本历史记录。 本部分记录了当前和先前用户模块版本之间区别的高级描述。