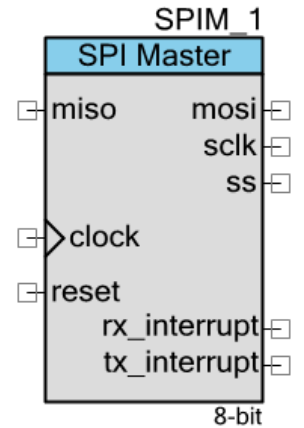


串行外设接口（SPI）主设备

2.40

特性

- 3~16 位数据宽度
- 4 种 SPI 工作模式
- 比特率可高达 18 Mbps^[1]



概述

SPI 主设备组件提供了行业标准的 4 线主设备 SPI 接口。此外，它还提供 3 线（双向）SPI 接口。这两种接口都支持四种 SPI 工作模式，可与任何 SPI 从设备进行通信。除了标准 8 位字长之外，SPI 主设备还支持可配置的 3 ~ 16 位字长，用于与非标准 SPI 字长进行通信。

SPI 信号包括标准串行时钟（SCLK）、主入从出（MISO）、主出从入（MOSI）、双向串行数据（SDAT）和从设备选择（SS）。

何时使用 SPI 主设备

用户可使用 PSoC SPI 主设备连接一个或多个 SPI 从设备进行通信。除了标有“SPI 从设备”的器件外，与实现移位寄存器类型接口的器件进行通信时，也可以使用 SPI 主设备。

在 PSoC 器件和 SPI 主设备器件进行通信时，用户可使用 SPI 从设备组件。在利用移位寄存器组件的底层灵活性来提供 SPI 主设备组件中不可用的硬件功能的场合，用户应当使用移位寄存器组件。

¹ 只有在设置了 High Speed Mode Enable（高速模式使能）选项时，该值才有效（有关详情，请参见[直流和交流电气特性](#)）。否则，最大比特率可达 9 Mbps。

输入/输出接口

本节介绍 **SPI** 组件的各种输入和输出接口。I/O 列表中的星号 (*) 表示, 在 I/O 说明部分中所列出的特定条件下, 该 I/O 可能不可见。

miso — 输入 *

miso 输入传输从设备中的主入从出 (MISO) 信号。当 **Data Lines** 参数被设置为 **MOSI + MISO** 时, 此输入可见。可见时, 则必须连接此输入。

sdat — 输入输出*

sdat 输入输出传输串行数据（SDAT）信号。**Data Lines** 参数被设置为 **Bi-directional**（双向）时，可以使用该输入。

注意：双向模式提供了内部回送功能，因此在两种模式中任一模式下，另一方向仍处于活动状态（填充或清空其缓冲区）。

图 1. SPI Bidirectional (双向) 模式 (数据由主设备传输到从设备)

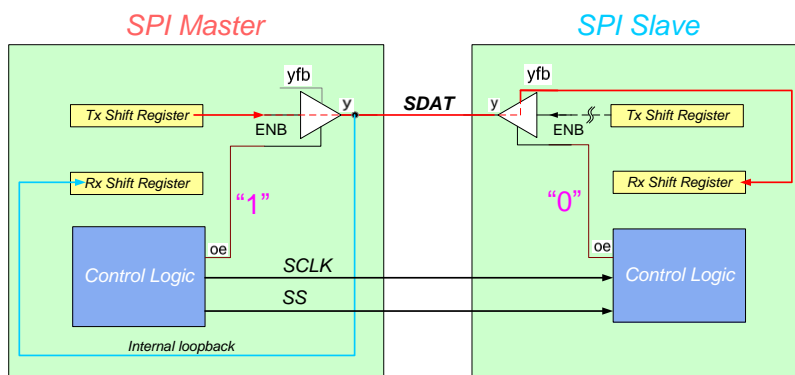
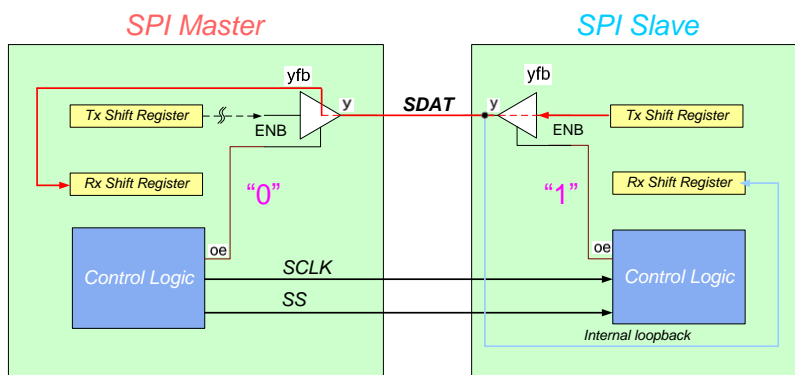


图 2. SPI 双向模式（数据由从设备传输到主设备）



双向模式下器件的初始状态为 Rx 模式或数据由从设备传输到主设备，如图 2 所示。用户可使用 SPIM_TxEnable() 和 SPIM_Tx_Disable() API 函数来切换 Rx 模式和 Tx 模式。

clock — 输入 *

时钟输入决定串行通信的比特率。比特率是输入时钟频率的 1/2。

当将 **Clock Selection** (时钟选择) 参数设置为 **External Clock** (外部时钟) 时，时钟输入可见。如果可见，则必须连接此输入。如果选择 **Internal Clock** (内部时钟)，则必须确定需要的数据比特率；所需时钟可通过配置 PSoC Creator 获得。

reset — 输入

将 SPI 状态机复位为闲置状态。这将丢掉当前正在传输或接收的任何数据，但不清除 FIFO 中已经收到或准备发送的数据。

mosi — 输出 *

mosi 输出传输主出从入 (MOSI) 数据信号，由主设备输出，输入到从设备。如果 **Data Lines** 参数被设为 **MOSI + MISO**，该输出会可见。

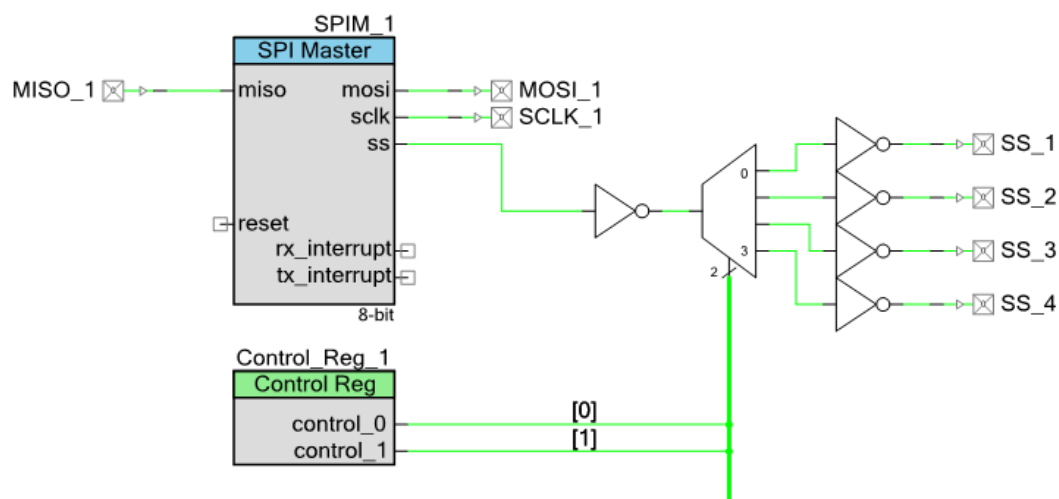
sclk — 输出

sclk 输出传输串行时钟 (SCLK) 信号。它通过总线将主设备上的同步时钟输出提供给从设备。

SS — 输出

ss 输出受硬件控制。它通过总线将从设备选择（SS）信号传输到从设备。用户可以连接一个数字多路输出选择器来处理多个从设备器件，或者完全由软件控制 SS。请参见图 3 和图 4。

图 3. 从设备选择输出至多路输出选择器



除非将反相器置于多路选择器的输出端，否则逻辑 ‘0’ 不能通过它传输，因为其他未选中通道也会有逻辑 ‘0’。为对此进行补偿，在多路选择器输入端放置一个反相器。

图 4. 固件控制的从设备选择

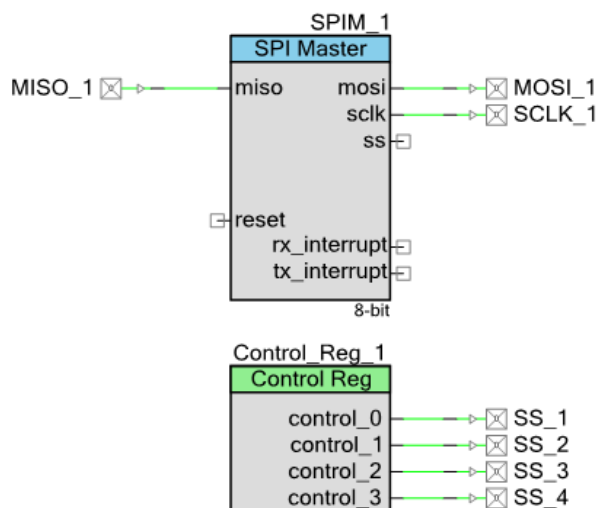
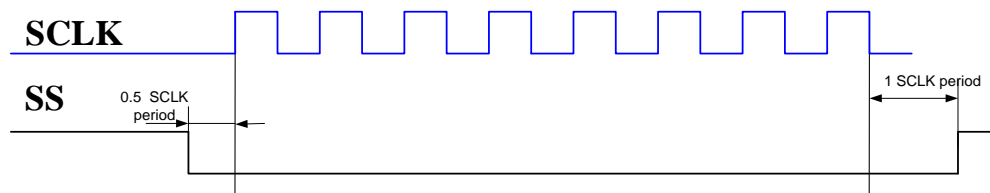


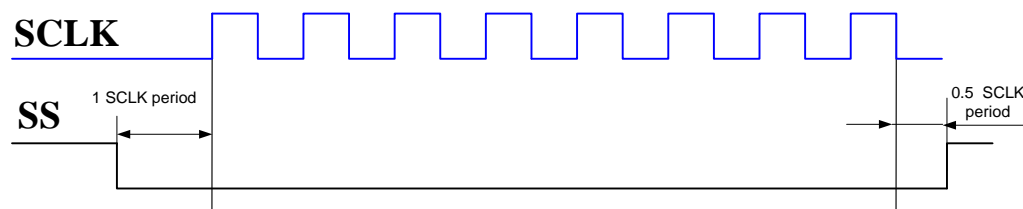
图 5 显示的是 SS 和 SCLK 之间的时序关系。

图 5. SS 和 SCLK 的时序关系

CPHA = 0:



CPHA = 1:



注意：如果未产生 SPI Done (SPI 完成) 条件，那么在传输多个字节/字的过程中不会将 SS 设置为高电平。

rx_interrupt — 输出

该中断输出是 Rx 中断源组的逻辑或运算结果。当任何使能的 Rx 中断源产生中断时，该信号将变为高电平。

tx_interrupt — 输出

该中断输出是 Tx 中断源组的逻辑或运算结果。当任何使能的 Tx 中断源产生中断时，该信号变为高电平。

原理图宏信息

默认情况下，PSoC Creator 组件目录包含 SPI 主设备组件的原理图宏实现。这些宏包含已连接并调整的输入引脚、输出引脚以及时钟源。原理图宏可用于 4 线（全双工）和 3 线（双向）SPI 连接。

图 6.4 线（全双工）连接原理图宏

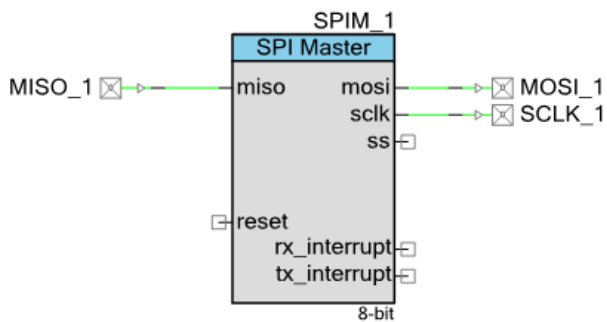
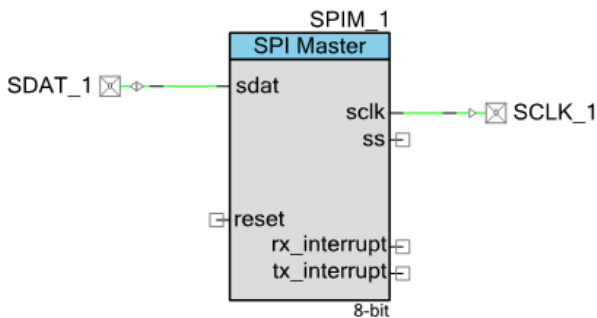


图 7.3 线（双向）连接原理图宏



注意： 如果不使用原理图宏，请配置引脚组件以取消选择每个已分配输入引脚（MISO 或 SDAT 输入输出）的同步输入参数。该参数位于相应‘引脚配置’对话框的 **Pins > Input** 选项卡中。

组件参数

将 SPI Master（SPI 主设备）组件拖放到设计工程中。双击组件符号，以打开 **Configure**（配置）对话框。

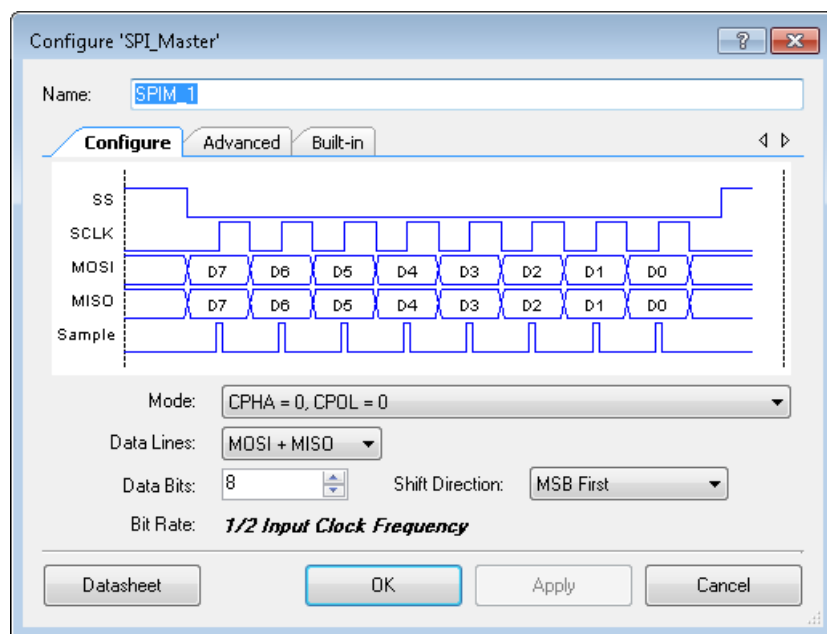
下面的章节介绍 SPI 主设备参数以及如何使用 **Configure** 对话框对它们进行配置。这些章节还指明了这些选项是在硬件中还是在软件中实现的。

硬件和软件配置选项

硬件配置选项用于更改在硬件中合成及放置项目的方式。如果您对这些项中的任何选项进行了更改，则必须重新编译硬件。软件配置选项并不影响项目的合成或放置。在编译之前设置这些参数，为其设置初始值，这些初始值随时可通过提供的 API 进行修改。仅硬件参数标有星号（*）。

Configure（配置）选项卡

Configure 选项卡包含每个 SPI 组件所需的基本参数。当您打开 **Configure** 对话框时，首先会看到这些参数。



注意：波形中的 **Sample**（采样）信号不是系统的输入或输出；它仅表示何时对主设备或从设备器件进行数据采样。

Mode（模式）*

Mode 参数定义了通信中使用的时钟相位和时钟极性模式。下表中定义了这些模式。有关详细信息，请参见[功能说明](#)。

CPHA	CPOL
0	0
0	1
1	0
1	1

Data Lines（数据线）

Data Lines 参数定义了使用 4 线（MOSI+MISO）还是 3 线（双向）SPI 通信接口。

Data Bits（数据位）*

数据位值定义了使用 `SPIM_ReadRxData()` 和 `SPIM_WriteTxData()` 函数传输时一次传输的数据位宽。默认位数为单字节（8 位）。3 ~ 16 之间的任何整数都是有效设置。

Shift Direction（移位方向）*

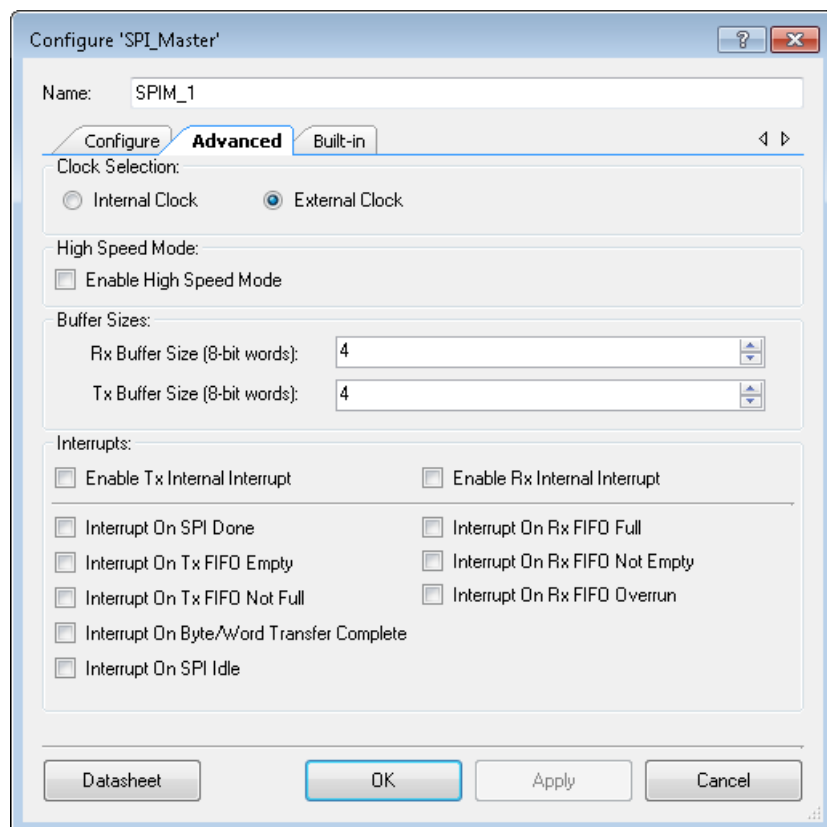
Shift Direction 参数用于定义串行数据传输的方向。当设置为 **MSB First**（MSB 优先）时，表示首先传输最高有效位。通过将数据向左移位实现此操作。当设置为 **LSB First**（LSB 优先）时，表示首先传输最低有效位。通过将数据向右移位来实现该操作。

Bit Rate（比特率）*

如果位于 **Advanced** 选项卡中的 **Clock Selection**（时钟选择）参数被设为 **Internal Clock**，那么 **Bit Rate** 参数将定义 SCLK 速率（单位为赫兹）。内部时钟频率将为 SCLK 速率的 2 倍。如果 **Clock Selection** 参数设置为**外部时钟**，则该参数不起作用。

Advanced（高级）选项卡

Advanced 选项卡包含提供附加功能的参数。



Clock Selection（时钟选择） *

通过 **Clock Selection** 参数，用户为数据速率和 SCLK 发生器选择内部配置的时钟或外部配置的时钟。设置为 **Internal Clock** 时，PSoC Creator 根据 **Bit Rate** 参数计算并配置所需的时钟频率。设置为 **External Clock** 时，组件不控制数据速率，但将根据用户连接的时钟源显示预期的比特率。如果该参数设置为 **Internal Clock**，则时钟输入在宏模块上不可见。

注意：当设置比特率或外部时钟频率值时，请确保 PSoC Creator 可以使用当前系统时钟频率提供该值。否则，编译项目时会生成时钟准确度范围警告。警告中包含 PSoC Creator 所设置的实际时钟值。此时可选择通过更改系统时钟或组件时钟来满足时钟设置系统的要求，以达到最佳值。

RX Buffer Size（RX 缓冲区大小） *

Rx Buffer Size 参数定义了分配给循环数据缓冲区的存储器大小（单位为字节/字）。如果将该参数设置为 1 到 4，则在硬件中实现 FIFO 的第 4 个字节/字。1-3 之间的值仅适用于与先前版本兼容；使用它们时会出现一个错误图标，该图标表示该值不正确。所有其他不超过 255 字节/字的值将使用 4 字节/字 FIFO 和提供的 API 所控制的存储器阵列。



TX Buffer Size（TX 缓冲区大小）*

TX Buffer Size 参数定义了分配给循环数据缓冲区分配的存储器大小（以字节/字为单位）。如果将该参数设置为 1 到 4，则在硬件中实现 FIFO 的第 4 个字节/字。1-3 之间的值仅适用于与先前版本兼容；使用它们时会出现一个错误图标，该图标表示该值不正确。其他不超过 255 字节/字的所有值都将采用 4 字节/字 FIFO 和提供的 API 所控制的存储器阵列。

使用软件缓冲区

选择大于 4 的 Rx/Tx 缓冲区大小的值，即可使用 Rx/Tx 软件循环缓冲区。当选择了 Tx/Rx 软件缓冲区时，将使用内部中断处理程序。它的主要目的是提供软硬 Tx/Rx 缓冲区之间的交互。在初始状态中，**BufferRead** 和 **BufferWrite** 指针指向软件缓冲区的零元素。写好第一个数据后，**BufferWrite** 指针将移至软件缓冲区中第一个元素，并指向写数据；**BufferRead** 指针仍留在零元素的位置。缓冲区工作时，指针会移动到下一个缓冲区元素。**BufferWrite** 指针指向最后写入的数据。**BufferRead** 指针指向尚未被读取的最早数据。即使没有任何溢出指示，软件缓冲区溢出仍可发生。用户必须处理所有软件缓冲区溢出情况。

另外，还应考虑到：使用软件缓冲区将导致被传输各字之间的时序间隔增大，因为中断处理程序的执行需要额外的时间（取决于所选总线时钟的值）。在设定各个被传输的字之间的时序间隔时，请使用 DMA 及硬件缓冲区。

Enable High Speed Mode（使能高速模式）

通过使用 **Enable High Speed Mode** 参数，最高比特率的值可增大至 18 mbps（有关详细信息，请参见“直流与交流电气特性”一节）

Enable TX / RX Internal Interrupt（使能 TX/RX 内部中断）

Enable TX / RX Internal Interrupt 选项允许用户使用 SPI 主设备预先定义的 Tx 和 Rx 中断服务程序，或者提供自己自定义的中断服务程序。如果使能内部中断，您可以将自己的代码添加到这些预定义的中断服务程序中（如果需要少量更改）。如果取消选择内部中断，用户可以提供外部中断组件和自定义代码，以便将其连接至 SPI 主设备中断输出。

如果 Rx 或 Tx 缓冲区的大小大于 4，则组件自动设置相应参数，因为需要内部中断服务程序 ISR 来处理数据从硬件 FIFO 到 Rx 缓冲区或 Tx 缓冲区或两者间的传输。SPI 主设备的中断输出引脚始终可见且可用，输出的信号与传输到内部中断的信号是相同的。该输出可以作为 DMA 请求源使用，也可以当成可编程数字系统所需的数字信号。

注意

- 当 Rx 缓冲区空间大于 4 字节/字时，“RX FIFO NOT EMPTY”中断始终被使能状态，不能禁用它，否则会引起缓冲区功能错误。
- 当 Tx 缓冲区空间大于 4 字节/字时，“Tx FIFO NOT FULL”中断始终被使能状态，不能禁用它，否则会引起缓冲区功能错误。
- 对于大于 4 字节/字的缓冲区大小，SPI 从设备和全局中断必须处于使能状态，这样才能进行正确的缓冲区处理。

中断

通过**中断**选项参数，用户可配置已使能的、可引起中断的内部事件。中断是由所有处于使能的 Tx 和 Rx 状态寄存器位的逻辑或运算结果产生的，并且可屏蔽。选择这些参数的位来初始化组件配置，是通过其掩码实现的。

时钟选择

当选择内部时钟配置时，PSoC Creator 计算所需的频率和时钟源，并生成所需的时钟资源。否则，用户必须提供时钟组件并计算所需的时钟频率。该频率是所需比特率和 SCLK 频率的 2 倍。

注意：当设置比特率或外部时钟频率值时，请确保 PSoC Creator 可以通过使用当前系统时钟频率提供该值。否则，编译项目时会生成时钟准确度范围警告。该警告将包含 PSoC Creator 设置的真实时钟值。此时可选择更改系统时钟或组件时钟以满足时钟设置系统要求，达到最佳值。

应用编程接口（API）

应用编程接口（API）函数允许用户在运行时使用软件配置组件。下表列出每个函数的接口，并进行了说明。以下各节将对每个函数加以说明。

默认情况下，PSoC Creator 将实例名称“SPIM_1”分配给设计中使用的第一个组件实例。您可以将该实例重新命名为符合标识符语法规则的任意唯一值。该实例名称会成为每个全局函数名称、变量和常量符号的前缀。为便于阅读，下表中使用了实例名称“SPIM”。

函数

函数	说明
SPIM_Start()	调用SPIM_Init()和SPIM_Enable()两个函数。应当在第一次启动组件时调用。
SPIM_Stop()	禁用SPI主设备操作。
SPIM_EnableTxInt()	使能了内部Tx中断请求。
SPIM_EnableRxInt()	使能了内部Rx中断请求。
SPIM_DisableTxInt()	禁用内部Tx中断请求。
SPIM_DisableRxInt()	禁用内部Rx中断请求。
SPIM_SetTxInterruptMode()	使能Tx中断源。
SPIM_SetRxInterruptMode()	使能Rx中断源。
SPIM_ReadTxStatus()	返回Tx状态寄存器的当前状态。
SPIM_ReadRxStatus()	返回Rx状态寄存器的当前状态。
SPIM_WriteTxData()	向传输缓冲区放置要在下一个可用总线时间发送的字节/字。
SPIM_ReadRxData()	返回接收缓冲区中有效接收数据的下一个字节/字。
SPIM_GetRxBufferSize()	返回Rx存储器缓冲区中所收到的数据大小（单位为字节/字）。
SPIM_GetTxBufferSize()	返回Tx存储器缓冲区中等待传输的数据大小（单位为字节/字）。
SPIM_ClearRxBuffer()	清除Rx缓冲区存储器阵列和Rx FIFO中所收到的数据。
SPIM_ClearTxBuffer()	清除Tx缓冲区存储器阵列或Tx FIFO中所发送的数据。 注意： 只有未使用软件缓冲区时，才能清除Tx FIFO。
SPIM_TxEnable()	如果配置为双向模式，则该函数将设置SDAT输入输出引脚，以进行传输数据。
SPIM_TxDisable()	如果配置为双向模式，则该函数将设置SDAT输入输出引脚，以进行接收数据。
SPIM_PutArray()	将数据阵列放在传输缓冲区内。
SPIM_ClearFIFO()	清除Rx硬件FIFO所接收的数据。

函数	说明
SPIM_Sleep()	通过调用SPIM_SaveConfig()和SPIM_Stop()函数，准备将SPI主设备置于低功耗模式。
SPIM_Wakeup()	从低功耗模式唤醒后恢复并重新使能 SPI主设备。
SPIM_Init()	初始化并恢复SPI主设备的默认配置。
SPIM_Enable()	使能SPI主设备以开始工作。
SPIM_SaveConfig()	保存SPI主设备硬件配置。
SPIM_RestoreConfig()	恢复SPI主设备硬件配置。

void SPIM_Start(void)

说明： 该函数调用了SPIM_Init()和SPIM_Enable()。应当在第一次启动组件时调用该函数。

参数： 无

返回值： 无

其他影响： 无

void SPIM_Stop(void)

说明： 如果SPI主设备是以该方式配置的，那么可以通过禁用内部时钟和内部中断来禁用SPI主设备操作。

参数： 无

返回值： 无

其他影响： 无

void SPIM_EnableTxInt(void)

说明： 使能内部Tx中断请求。

参数： 无

返回值： 无

其他影响： 无

void SPIM_EnableRxInt(void)

说明：使能了内部Rx中断请求。

参数：无

返回值：无

其他影响：无

void SPIM_DisableTxInt(void)

说明：禁用内部Tx中断请求。

参数：无

返回值：无

其他影响：无

void SPIM_DisableRxInt(void)

说明：禁用内部Rx中断请求。

参数：无

返回值：无

其他影响：无

void SPIM_SetTxInterruptMode(uint8 intSrc)

说明：配置哪些状态位可触发中断事件。

参数：uint8 intSrc: 包含需要使能的中断的位字段。

位	说明
SPIM_INT_ON_SPI_DONE	由于SPI完成而使能中断
SPIM_INT_ON_TX_EMPTY	由于Tx FIFO为空而使能中断
SPIM_INT_ON_TX_NOT_FULL	由于Tx FIFO未满而使能中断
SPIM_INT_ON_BYTE_COMP	由于字节/字传输操作已完成，使能中断
SPIM_INT_ON_SPI_IDLE	由于SPI IDLE而使能中断

基于Tx状态寄存器的位字段排列。该值必须是头文件中定义的Tx状态寄存器位掩码的组合。
更多信息，请参见[定义](#)。

返回值：无

其他影响：无

void SPIM_SetRxInterruptMode(uint8 intSrc)

说明：配置哪些状态位可触发中断事件。

参数：uint8 intSrc: 包含需要使能的中断的位字段。

位	说明
SPIM_INT_ON_RX_FULL	由于Rx FIFO已满而使能中断
SPIM_INT_ON_RX_NOT_EMPTY	由于Rx FIFO为非空而使能中断
SPIM_INT_ON_RX_OVER	由于Rx缓冲区溢出而使能中断

基于Rx状态寄存器的位字段排列。该值必须是头文件中定义的Rx状态寄存器位掩码的组合。更多信息，请参见[定义](#)。

返回值：无

其他影响：无

uint8 SPIM_ReadTxStatus(void)

说明：返回Tx状态寄存器的当前状态。更多信息，请参见[状态寄存器位](#)。

参数：无

返回值：uint8: 当前的Tx状态寄存器值

位	说明
SPIM_STS_SPI_DONE	SPI已完成
SPIM_STS_TX_FIFO_EMPTY	Tx FIFO为空
SPIM_STS_TX_FIFO_NOT_FULL	Tx FIFO未满
SPIM_STS_BYTE_COMPLETE	字节/字的传输已完成
SPIM_STS_SPI_IDLE	SPI IDLE

其他影响：Tx状态寄存器位在读取时被清除。

uint8 SPIM_ReadRxStatus(void)

说明： 返回Rx状态寄存器的当前状态。更多信息，请参见[状态寄存器位](#)。

参数： 无

返回值： uint8: 当前的Rx状态寄存器值

位	说明
SPIM_STS_RX_FIFO_FULL	Rx FIFO已满
SPIM_STS_RX_FIFO_NOT_EMPTY	Rx FIFO非空
SPIM_STS_RX_FIFO_OVERRUN	Rx缓冲区溢出

其他影响： Rx状态寄存器位在读取时清除。

void SPIM_WriteTxData(uint8/uint16 txData)

说明： 向传输缓冲区放置要在下一个可用SPI总线时间发送的字节/字。

参数： uint8/uint16 txData: 要从SPI传输的数据值。

返回值： 无

其他影响： 数据可以放置在存储器缓冲区中，直到前面的所有其他数据传输完毕后才传输。该函数一直处于阻塞状态，直到输出存储器缓冲区中有空间为止。
清除组件的Tx状态寄存器。

uint8/uint16 SPIM_ReadRxData(void)

说明： 返回接收缓冲区中有效接收数据的下一个字节/字。

参数： 无

返回值： uint8/uint16: 从FIFO中读取的下一个数据字节/字。

其他影响： 如果FIFO为空，返回无效数据。调用SPIM_GetRxBufferSize()，如果它返回非零值，则调用SPIM_ReadRxData()函数是安全的。

uint8 SPIM_GetRxBufferSize(void)

- 说明：** 返回Rx缓冲区中当前保存的所收到的数据字节数/字数。
- 如果禁用了Rx软件缓冲区，此函数会返回0（表示FIFO为空）或1（表示FIFO非空）。
 - 如果使能了Rx软件缓冲区，则该函数返回Rx软件缓冲区中的数据大小。该计数中不包括FIFO数据。
- 参数：** 无
- 返回值：** uint8: Rx缓冲区中字节数/字数的整型计数值。
- 其他影响：** 清除组件的Rx状态寄存器。

uint8 SPIM_GetTxBufferSize(void)

- 说明：** 返回Tx缓冲区中当前保存的将要传输的数据字节数/字数。
- 如果禁用了Tx软件缓冲区，该函数会返回0（表示FIFO为空）、1（表示FIFO未满）或4（表示FIFO已满）。
 - 如果使能了Tx软件缓冲区，则该函数将返回Tx软件缓冲区中的数据大小。该计数中不包括FIFO数据。
- 参数：** 无
- 返回值：** uint8: Tx缓冲区中字节数/字数的整型计数值。
- 其他影响：** 清除组件的Tx状态寄存器。

void SPIM_ClearRxBuffer(void)

- 说明：** 清除Rx缓冲区存储器阵列和Rx硬件FIFO中所接收的数据。通过将读取指针和写入指针都设置为零，可以清除Rx RAM缓冲区。将指针设置为零表示没有需要读取的数据。因此，在地址0重新写入，会覆盖RAM中剩下的所有数据。
- 参数：** 无
- 返回值：** 无
- 其他影响：** 当新数据覆盖时，保存在RAM缓冲区和FIFO中任何未读取的接收数据将会丢失。

void SPIM_ClearTxBuffer(void)

- 说明：**清除等待传输的Tx缓冲区存储器数据阵列。通过将读取指针和写入指针都设置为零，清除Tx RAM缓冲区。将指针设置为零表示没有需要传输的数据。因此，在地址0重新写入，会覆盖RAM中的所有数据。
- 参数：**无
- 返回值：**无
- 其他影响：**如果使用软件缓冲区，它不会清除已放入Tx FIFO中的数据。当新数据覆盖时，尚未从RAM缓冲区中传输的所有数据均会丢失。

void SPIM_TxEnable(void)

- 说明：**如果配置SPI主设备使用一个双向引脚，则它将引脚设置为用于传输的双向引脚。
- 参数：**无
- 返回值：**无
- 其他影响：**无

void SPIM_TxDisable(void)

- 说明：**如果配置SPI主设备使用单一双向引脚，则它将引脚设置为用于接收的双向引脚。
- 参数：**无
- 返回值：**无
- 其他影响：**无

void SPIM_PutArray(const uint8/uint16 buffer[], uint8 byteCount)

- 说明：**将数据阵列放入传输缓冲区
- 参数：**
const uint8 buffer[]: 指向RAM中包含需要发送数据的位置的指针
uint8 byteCount: 移至传输缓冲区的字节/字数
- 返回值：**无
- 其他影响：**系统将阻塞在该函数位置，直到所有数据已传输到缓冲区内为止。如果Tx缓冲区中没有足够的空间，该函数将阻塞系统。如果主设备未传输数据并且Tx缓冲区已满，则该函数在该循环中保持锁定。

void SPIM_ClearFIFO(void)

说明：清除Tx FIFO和Rx FIFO中的所有数据。

参数：无

返回值：无

其他影响：清除组件的状态寄存器。

void SPIM_Sleep(void)

说明：准备使SPI主设备进入低功耗模式。调用SPIM_SaveConfig()和SPIM_Stop()函数。

参数：无

返回值：无

其他影响：无

void SPIM_Wakeup (void)

说明：退出低功耗模式后恢复SPI主设备配置。调用SPIM_RestoreConfig()和SPIM_Enable()函数。清除保存在Rx缓冲区、Tx缓冲区和硬件FIFO中的所有数据。

参数：无

返回值：无

其他影响：无

void SPIM_Init(void)

说明：根据自定义 **Configure** 对话框设置初始化或恢复组件。不必调用 SPIM_Init()，因为 SPIM_Start() 函数将调用此函数，这是开始组件操作的首选方法。

参数：无

返回值：无

其他影响：当调用该函数时，它将初始化所需执行的参数。这些操作包括设置初始中断掩码、配置中断服务子程序、配置位计数器参数以及清除FIFO和状态寄存器。

void SPIM_Enable(void)

- 说明：

使能SPI主设备以开始工作。如果SPI主设备选择了内部时钟，则开启内部时钟。如果选择使用外部时钟，则在调用该函数之前必须启动外部时钟。应当在SPI主设备中断使能之前调用SPIM_Enable()函数。因为该函数配置中断源并清除器件配置中的任何挂起中断，然后使能内部中断（若有）。以前必须已调用SPIM_Init()函数。
- 参数：

无
- 返回值：

无
- 其他影响：

无

void SPIM_SaveConfig(void)

- 说明：

进入低功耗模式之前，保存SPI主设备硬件配置。
- 参数：

无
- 返回值：

无
- 其他影响：

无

void SPIM_RestoreConfig(void)

- 说明：

从低功耗模式唤醒后恢复由SPIM_SaveConfig()函数保存的SPI主设备硬件配置。
- 参数：

无
- 返回值：

无
- 其他影响：

如果在未调用SPIM_SaveConfig()的情况下调用该函数，则在下列寄存器值将恢复为Configure（配置）对话框中的默认值：

SPIM_STATUS_MASK_REG
SPIM_COUNTER_PERIOD_REG

全局变量

变量	说明
SPIM_initVar	指示是否已初始化SPI主设备。变量初始化为0，在第一次调用SPIM_Start()时设置为1。因此，第一次调用SPIM_Start()函数后，组件无需重新初始化便可重新启动。如果需要重新初始化组件，则在调用SPIM_Start()或SPIM_Enable()函数之前可调用SPIM_Init()函数。
SPIM_txBufferWrite	表示通过API将最后数据写入到传输缓冲区中的位置。
SPIM_txBufferRead	表示从缓冲区读取并由SPI主设备硬件发送的最后数据的传输缓冲区位置。
SPIM_rxBufferWrite	表示由SPI主设备硬件接收后写入到缓冲区内的最后数据的接收缓冲区位置。



变量	说明
SPIM_rxBufferRead	表示通过API从缓冲区读取的最终数据的接收缓冲区位置。
SPIM_rxBufferFull	指示软件缓冲区已发生溢出。
SPIM_rxBuffer[]	用于存储接收到的数据。
SPIM_txBuffer[]	用于存储需要发送的数据。

定义

- **SPIM_TX_INIT_INTERRUPTS_MASK** — 定义 **Configure** 对话框中选择的中断源的初始配置。这是 Tx 状态寄存器中在配置时被使能作为中断源的位掩码。有关位域的详细信息，请参见[状态寄存器位](#)。
- **SPIM_RX_INIT_INTERRUPTS_MASK** — 定义 **Configure** 对话框中选择的中断源的初始配置。这是 Rx 状态寄存器中在配置时被使能作为中断源的位掩码。有关位域的详细信息，请参见[状态寄存器位](#)。

状态寄存器位

SPIM_TXSTATUS

位	7	6	5	4	3	2	1	0
值	中断	未使用	未使用	SPI IDLE	字节/字的传输已完成	Tx FIFO 未 满	Tx FIFO 为空	SPI已完成

SPIM_RXSTATUS

位	7	6	5	4	3	2	1	0
值	中断	Rx缓冲区 溢出	Rx FIFO 非空	Rx FIFO 已满	未使用	未使用	未使用	未使用

- 字节/字传输已完成：当字节/字的数据传输操作已完成时设置该位。
- Rx FIFO 溢出：当 Rx 数据已溢出 4 字节/字 FIFO 而未被移到 Rx 缓冲区存储器阵列（如果该阵列存在）时，将设置该位。
- Rx FIFO 非空：当 Rx 数据 FIFO 非空时，将设置该位。即，至少有一个字节/字位于 Rx FIFO 中（不表示 Rx 缓冲区 RAM 阵列条件）。
- Rx FIFO 已满：当 Rx 数据 FIFO 已满时，将设置该位（不表示 Rx 缓冲区 RAM 阵列条件）。

- **Tx FIFO 未滿：**当 Tx 数据 FIFO 未滿时，将设置该位（不表示 Tx 缓冲区 RAM 阵列条件）。
- **Tx FIFO 为空：**当 Tx 数据 FIFO 为空时，将设置该位（不表示 Tx 缓冲区 RAM 阵列条件）。
- **SPI 已完成：**当已发送 Tx FIFO 中所有数据时，将设置该位。无需使用字节/字完成状态，可以使用该位表示传输操作已完成。（当已设置“Byte/Word Complete”（字节/字传输已完成）并且 Tx 数据 FIFO 为空时，设置它。）
- **SPI IDLE：**当 SPI 主设备状态机处于 IDLE 状态时进行设置。这是组件启动后的默认状态。它也是 SPI 完成后的下一个状态。一直处于 IDLE 状态，直到检测到 Tx FIFO Not Empty（Tx FIFO 非空）为止。

SPIM_TX_BUFFER_SIZE

定义了需要分配给 Tx 存储器阵列缓冲区的存储器大小。它不包括 FIFO 中的 4 个字节/字。如果该值大于 4，将会生成中断，并自动将数据从循环存储器缓冲区移动到 FIFO。

SPIM_RX_BUFFER_SIZE

定义了需要分配给 Rx 存储器阵列缓冲区的存储器大小。它不包括 FIFO 中的 4 个字节/字。如果该值大于 4，将会生成中断，并自动将数据从 FIFO 移动到循环存储器缓冲区内。

SPIM_DATA_WIDTH

定义了 **Configure** 对话框中所选每个数据传输的位数。

示例固件源代码

在 Find Example Project 对话框中，PSoC Creator 提供了大量的示例项目，包括原理图和示例代码。要获取组件特定的示例，请右击组件并打开 Find Example Project 对话框来查看组件实例。要查看通用示例，请打开‘Start Page’（起始页）或 **File**（文件）菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options**（滤波器选项）项来限定可选的项目列表。

更多信息，请参考《PSoC Creator 帮助》中主题为“查找示例项目”一节的内容。

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性和本器件的偏差情况。有两种偏差类型，如下定义：

- **项目偏差** — 适用于所有 PSoC Creator 组件的偏差
- **特定偏差** — 仅适用于该组件的偏差

本节介绍了有关组件特定偏差的信息。系统参考指南的“MISRA 合规性”章节中介绍了项目偏差以及有关 MISRA 合规性验证环境的信息。

SPI 主设备具有以下特定偏差：

MISRA-C:2004规则	规则类别 (必须 (R) / 建议 (A))	规则说明	偏差说明
19.7	A	函数应该优先使用类似函数的宏。	由于使用了函数宏以实现更高效的代码，所以出现了偏差。 组件使用带输入参数的宏： SPIM_GET_STATUS_TX() SPIM_GET_STATUS_RX()

该组件具有以下嵌入式组件：时钟。MISRA 合规性与特定偏差的相关信息，请参见相应组件数据手册。

API 存储器的使用情况

根据编译器、器件、所使用的 API 数量以及组件的配置不同，组件存储器的使用情况也不一样。下表提供了给定组件配置中的所有 API 所使用存储空间。

下表中的存储器大小是在将相应编译器设置为 Release 模式并且优化选项为 Size 的情况下测得的。对于特定的设计，分析编译器生成的映射文件后可以确定存储器的使用情况。

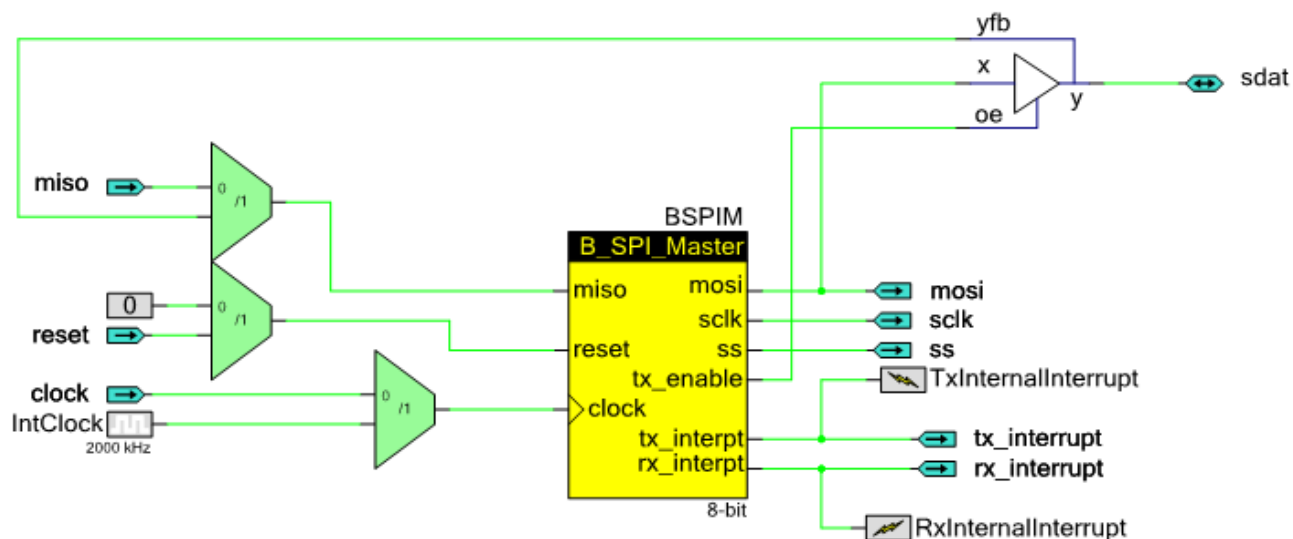
配置	PSoC 3 (Keil_PK51)		PSoC 5LP (GCC)	
	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节
8位 (MOSI+MISO)	372	5	580	5
8位 (双向)	366	5	580	5
16位 (MOSI+MISO)	431	5	620	5
16位 (双向)	439	5	620	5
8位高速 (MOSI+MISO)	372	5	580	5
8位高速 (双向)	370	5	580	5
16位高速 (MOSI+MISO)	439	5	620	5
16位高速 (双向)	437	5	620	5



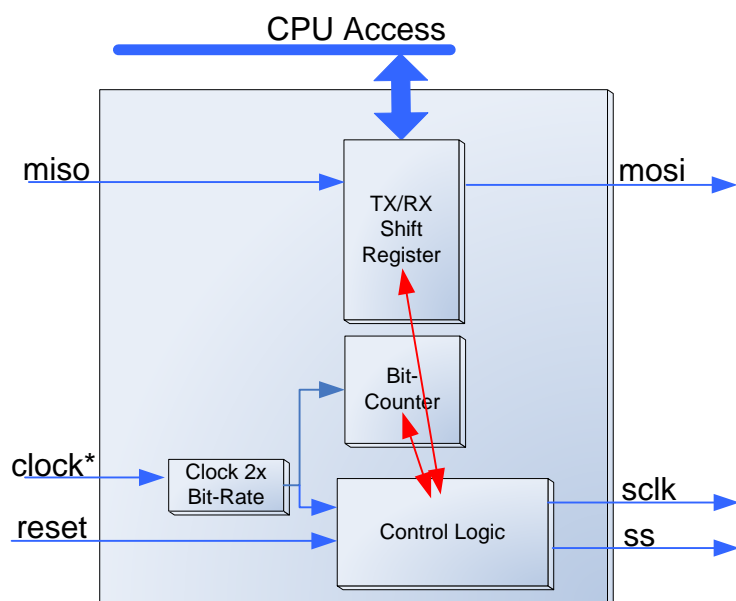
功能说明

框图和配置

SPI 主设备仅作为 UDB 配置提供。此处描述的寄存器用于定义 SPI 主设备的硬件实现。



下面的框图中描述了该实现。



默认配置

SPI 主设备的默认配置是 8 位，配置为 (CPHA = 0, CPOL = 0)。默认情况下，选择内部时钟，使用 1 Mbps 比特率。

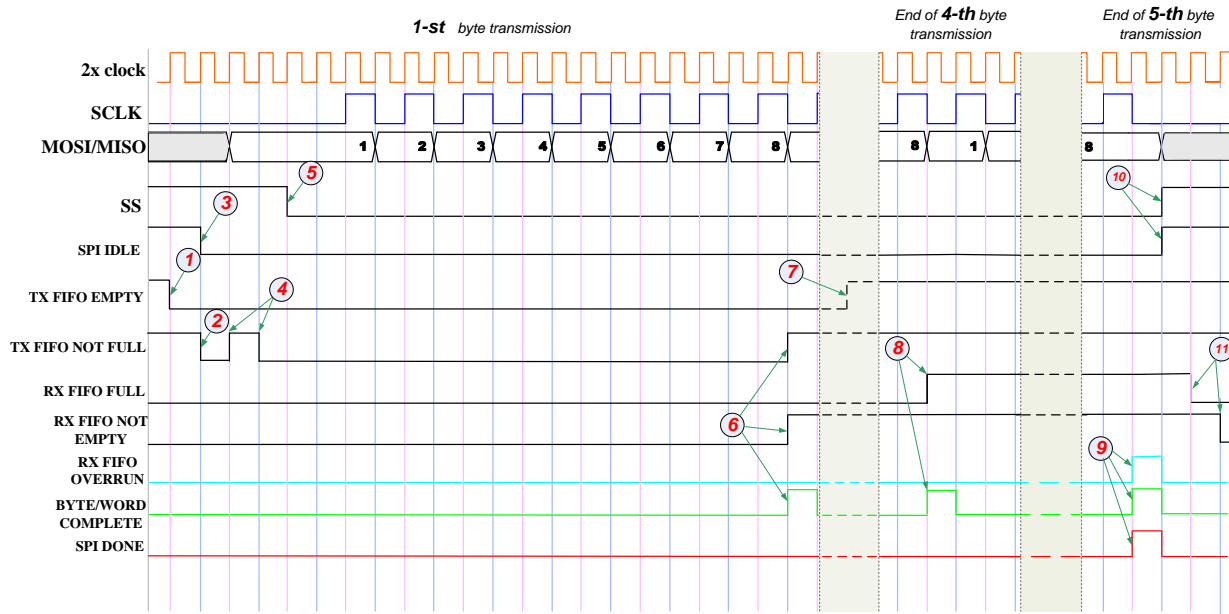
模式

以下四个波形显示组件的状态位、以及在数据传输中所取的信号值。假设传输五个数据字节（四个字节在传输开始时写入 SPI 主设备的 Tx 缓冲区，并在第一个字节加载到 A0 寄存器之后会将第五个字节写入 SPI 主设备的 Tx 缓冲区中）。波形上圆圈中的数字表示以下事件：

1. 当将 4 个字节写入到 Tx 缓冲区时，清除 Tx FIFO Empty (Tx FIFO 为空)。
2. 由于写入 4 个字节后 Tx FIFO 已满，所以清除 Tx FIFO Not Full (Tx FIFO 未滿)。
3. 由于在 Tx 缓冲区中检测到字节，所以清除 SPI IDLE 状态位。
4. 当第一个字节已载入 A0 寄存器中时，设置 Tx FIFO Not Full (Tx FIFO 未滿) 状态，并且在第五个字节已写入 Tx 缓冲区中的空位置时，清除该状态。
5. “Slave Select (从设备选择)” 线设置为低电平，表示传输开始。
6. 当第二个位载入 A0 时，设置 Tx FIFO Not Full (Tx FIFO 未滿) 状态。当收到的第 1 个字节加载到 Rx 缓冲区中时，设置 Rx Not Empty (Rx 非空) 状态。另外，也设置 Byte/Word Complete (字节/字完成)。
7. 在要发送的最后一个字节被加载到 A0 寄存器中的那一刻，设置 Tx FIFO Empty (Tx FIFO 为空) 状态 (为简化起见，不详细显示)。
8. 当接收了第四个字节时，同时设置 “Rx FIFO 已满” 和 “字节/字传输已完成”。
9. 由于所有字节已被传输，并尝试将数据载入到已满的 Rx 缓冲区中，所以设置了 “字节/字传输已完成”、“SPI 已完成” 以及 “Rx 溢出”。
10. 将 SS 线设置为高电平时表示传输完成。另外，也设置 SPI IDLE 状态。
11. 当从 Rx 缓冲区读取了第一个字节时，将清除 “Rx FIFO 已满” 状态；如果读取了所有字节，会设置 “Rx FIFO 为空” 状态。

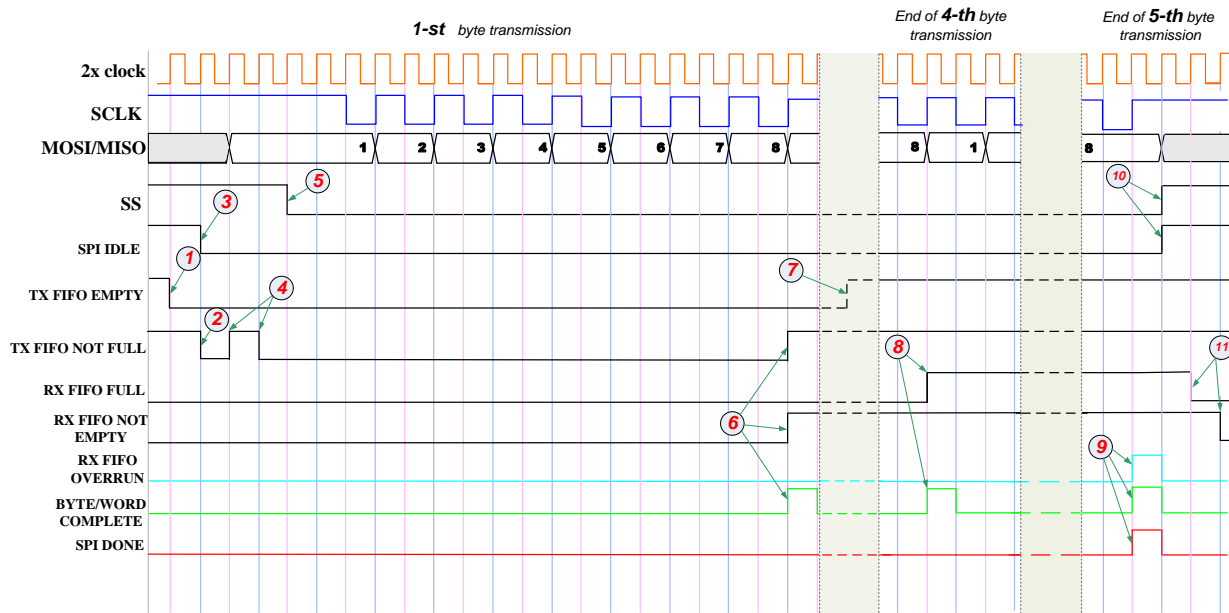
SPI 主设备（CPHA = 0, CPOL = 0）

该模式具有下列特性：



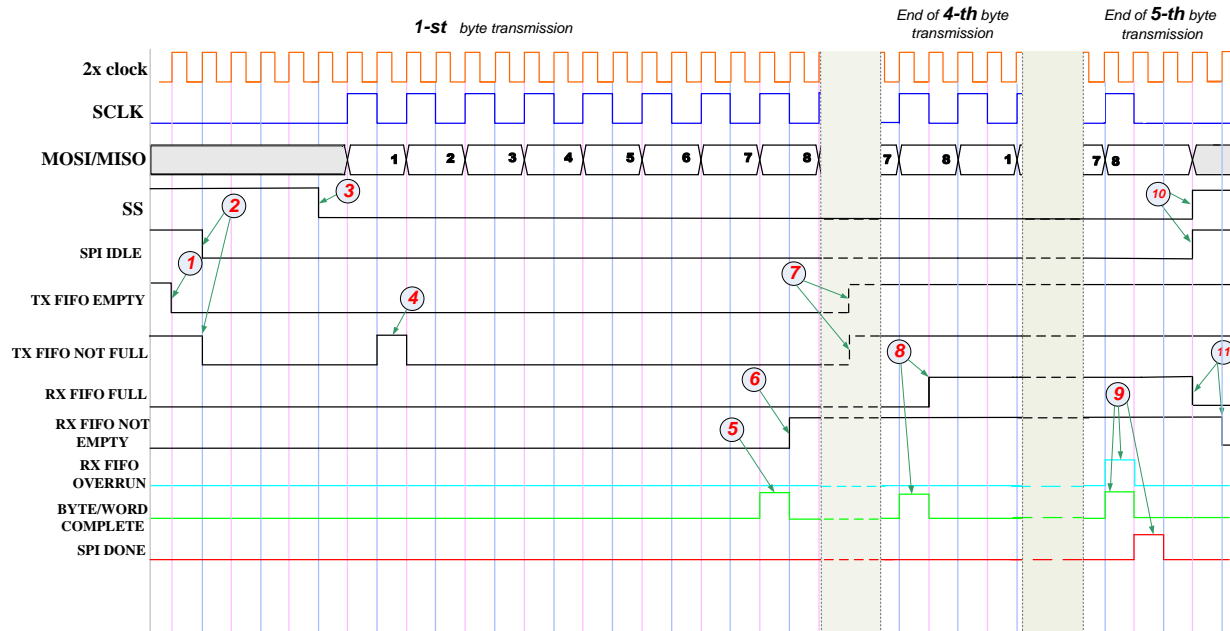
SPI 主设备模式：（CPHA = 0, CPOL = 1）

该模式具有下列特性：

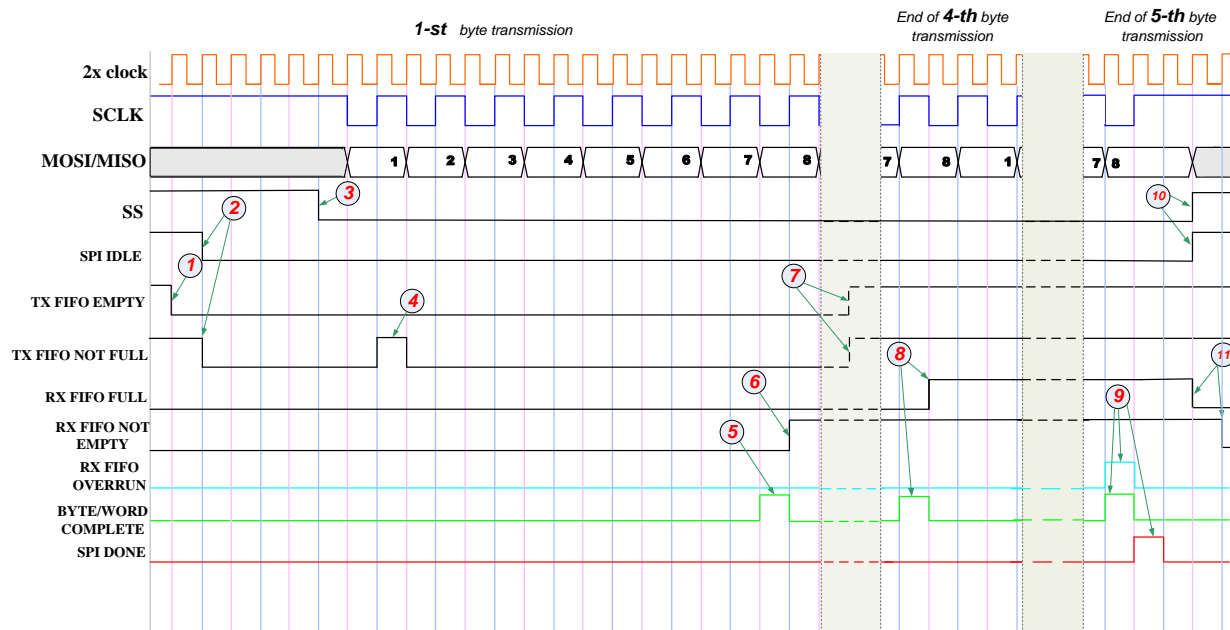


SPI 主设备模式：（CPHA = 1，CPOL = 0）

该模式具有下列特性：

**SPI 主设备模式：（CPHA = 1，CPOL = 1）**

该模式具有下列特性：



寄存器

Tx 状态寄存器

Tx 状态寄存器是只读寄存器，它包含为 SPI 主设备的给定实例定义的各种传输状态位。假设 SPI 主设备的实例被命名为“SPIM”，用户可以使用 `SPIM_ReadTxStatus()` 函数得到该寄存器的值。

通过对 Tx 状态寄存器中的掩码位字段进行 OR 运算可以生成中断输出。通过使用 `SPIM_SetTxInterruptMode()` 函数，您可设置掩码。接收中断后，可以通过使用 `SPIM_ReadTxStatus()` 函数读取 Tx 状态寄存器来检索中断源。

读取时清除 Tx 状态寄存器中的粘滞位，因此中断源一直保留，直到调用 `SPIM_ReadTxStatus()` 函数为止。Tx 状态寄存器中的所有操作必须使用位字段的下列定义，这是因为构建时这些位字段可以在 Tx 状态寄存器内移动。必须使用 CPU 或 DMA 读取来清除用于生成中断或 DMA 数据操作的粘滞位，这样能够避免连续生成中断或 DMA。

有一些为 Tx 状态寄存器定义的位字段。这些位字段的任意组合都可以作为中断源。下表中标有星号(*)的位字段配置为 Tx 状态寄存器中的粘滞位。所有其他位配置为状态的实时指示符。粘滞位对存储器状态进行锁存，以便稍后可以读取它们，然后在读取时清除。生成的头文件（例如 *SPIM.h*）中提供下列 `#define`：

- `SPIM_STS_SPI_DONE *` — 在 SCLK 的数据锁存沿（沿取决于模式）为输出时，设置为高电平。这在单个 SPI 字中配置的几位中的最后一位被输出至 MOSI 线之后、且传输 FIFO 为空时发生。当 SPI 主设备正在传输数据或传输 FIFO 具有挂起数据时，会清除它。并告诉用户 SPI 何时完成多字数据操作。
- `SPIM_STS_TX_FIFO_EMPTY` — 当传输 FIFO 不包含挂起的数据时，则读取它会输出高值。如果数据正在等待传输，则读取它会输出低值。
- `SPIM_STS_TX_FIFO_NOT_FULL` — 当传输 FIFO 未满足且有写入更多数据的空间时，读取它会输出高值。如果 FIFO 填满等待传输的数据且此时没有更多写入空间，则读取它会输出低值。告诉用户何时可以安全地将更多数据挂起到传输 FIFO 中。
- `SPIM_STS_BYTE_COMPLETE *` — 当单一 SPI 字中配置的位数的最后一位输出到 MOSI 线中时，将其设置为高电平。当输出 SCLK 的数据锁存边沿（边沿取决于模式）时，它自动清除*。
- `SPIM_STS_SPI_IDLE *` — 只要组件状态机处于 SPI IDLE 状态（组件在等待 Tx 数据，并不传输任何数据），该位就设置为高电平。

Rx 状态寄存器

Rx 状态寄存器是只读的寄存器，它包含了为 SPI 主设备定义的各种接收状态位。可以使用 `SPIM_ReadRxStatus()` 函数得到该寄存器的值。

对 Rx 状态寄存器中的掩码位字段进行“或”运算可以生成中断输出信号。通过使用 `SPIM_SetRxInterruptMode()` 函数，可设置掩码。接收中断时，可以通过使用 `SPIM_ReadRxStatus()` 函数读取 Rx 状态寄存器来检索中断源。读取时清除 Rx 状态寄存器中的粘滞位，因此中断源一直保留，直到调用 `SPIM_ReadRxStatus()` 函数为止。Rx 状态寄存器中的所有操作必须使用位字段的下列定义，这是因为编译时这些位字段可以在 Rx 状态寄存器内移动。必须使用 CPU 或 DMA 读取来清除用于生成中断或 DMA 数据操作的粘滞位，这样能够避免连续生成中断或 DMA。

有一些为 Rx 状态寄存器定义的位字段。这些位字段的任意组合都可以作为中断源。下表中标有星号(*)的位字段配置为 Rx 状态寄存器中的粘滞位。所有其他位配置为状态的实时指示符。粘滞位对存储器状态进行锁存，以便稍后可以读取它们，然后在读取时清除。生成的头文件（例如 `SPIM.h`）中提供下列 `#define`：

- `SPIM_STS_RX_FIFO_FULL` — 当接收 FIFO 已满且没有更多空间存储收到的数据时，读取它会输出高值。如果 FIFO 未满足且有空间容纳额外接收的数据，则读取它会输出低值。用于了解是否有空间来存储新接收的数据。
- `SPIM_STS_RX_FIFO_NOT_EMPTY` — 当接收 FIFO 不为空时，那么读取它会输出高值。如果 FIFO 为空且有空间容纳额外接收的数据，读取它会输出低值。
- `SPIM_STS_RX_FIFO_OVERRUN *` — 当接收 FIFO 已满且有附加数据写入它时，那么读取它会输出高值。告诉用户数据是否由于对 FIFO 操作缓慢而导致 FIFO 中数据丢失。

Tx 数据寄存器

Tx 数据寄存器包含需要发送的传输数据值。它可用作 SPI 主设备中的 FIFO。提供了一个可选的更高级别软件状态机，以控制从传输存储器缓冲区取得的数据。它处理要发送的、超出 FIFO 容量的大量数据。所有涉及传输数据的 API 都必须通过该寄存器才能将数据放到总线上。如果该寄存器中有数据且控制状态机指示可以发送该数据，则该数据将传输到总线上。该寄存器（FIFO）一旦为空，总线上就不再发送数据，直到向 FIFO 添加新的数据为止。可使用头文件中定义的 `TXDATA_REG` 地址，设置 DMA 来填充该 FIFO。

Rx 数据寄存器

Rx 数据寄存器包含收到的数据。它可用作 SPI 主设备中的 FIFO。提供了一个可选的更高级别软件状态机，以控制数据从此接收 FIFO 到存储器缓冲区的移动。通常，Rx 中断表示接收了数据。此时，数据有几种路径可进入固件。可以设置 DMA，以将该寄存器上的数据传输到此存储器阵列；也可由固件简单地调用 `SPIM_ReadRxData()` 函数。DMA 必须使用头文件中定义的 `RXDATA_REG` 地址。

有条件编译信息

SPI 主设备只需一个有条件编译定义，即可处理实现配置的 **NumberOfDataBits** 所需的 8 或 16 位数据路径配置。API 需要有条件地编译定义的数据带宽。API 永远不应当直接使用这些参数，但应当使用下列定义：

- **SPIM_DATAWIDTH** — 它定义有多少数据位将组成单一“字节”传输。有效范围为 3 到 16 位。

资源

SPI 主设备被放置在整個 UDB 阵列中。该组件使用了以下资源。

配置	资源类型					
	Datapath单元	宏单元	状态单元	控制单元	DMA通道	中断
8位（MOSI+MISO）	1	12	2	1	—	2
8位（双向）	1	12	2	2	—	2
16位（MOSI+MISO）	2	12	2	1	—	2
16位（双向）	2	12	2	2	—	2
8位高速（MOSI+MISO）	1	18	2	1	—	2
8位高速（双向）	1	18	2	2	—	2
16位高速（MOSI+MISO）	2	18	2	1	—	2
16位高速（双向）	2	18	2	2	—	2

直流和交流电气特性

除非另有说明，否则这些规范的适用条件是： $-40\text{ }^{\circ}\text{C} \leq T_A \leq 85\text{ }^{\circ}\text{C}$ 、 $T_J \leq 100\text{ }^{\circ}\text{C}$ 且 $1.71\text{ V} \sim 5.5\text{ V}$ 。

直流特性

参数	说明	最小值	典型值 ^[2]	最大值	单位 ^[3]
I _{DD} (8-bit)	组件电流消耗（8位；MOSI+MISO）				
	闲空电流 ^[4]	—	20	—	μA/MHz
	工作电流 ^[5]	—	28	—	μA/MHz
	组件电流消耗（8位；双向）				
	空闲电流 ^[3]	—	21	—	μA/MHz
	工作电流 ^[4]	—	30	—	μA/MHz
	组件电流消耗（8位；高速；MOSI+MISO）				
	空闲电流 ^[3]	—	23	—	μA/MHz
	工作电流 ^[4]	—	34	—	μA/MHz
	组件电流消耗（8位；高速；双向）				
	空闲电流 ^[3]	—	27	—	μA/MHz
	工作电流 ^[4]	—	36	—	μA/MHz
I _{DD} (16-bit)	组件电流消耗（16位；MOSI+MISO）				
	空闲电流 ^[3]	—	23	—	μA/MHz
	工作电流 ^[4]	—	30	—	μA/MHz
	组件电流消耗（16位；双向）				
	空闲电流 ^[3]	—	25	—	μA/MHz
	工作电流 ^[4]	—	32	—	μA/MHz
	组件电流消耗（16位；高速；MOSI+MISO）				
	空闲电流 ^[3]	—	27	—	μA/MHz

2. 未包括器件 IO 和时钟分配的电流。这些都是在温度为 25 °C 时的值。

3. 根据新的组件时钟指定电流消耗。

4. 组件使能但不传输/接收数据时消耗电流。

5. 组件使能且传输/接收数据时消耗电流。



参数	说明	最小值	典型值 ^[2]	最大值	单位 ^[3]
	工作电流 ^[4]	—	38	—	μA/MHz
	组件电流消耗（16位；高速；双向）				
	空闲电流 ^[3]	—	30	—	μA/MHz
	工作电流 ^[4]	—	40	—	μA/MHz

交流特性

参数	说明	最小值	典型值	最大值 ^[6]	单位
f _{SCLK}	SCLK频率				
	8位（MOSI+MISO）	—	—	9	MHz
	8位（双向）	—	—	9	MHz
	16位（MOSI+MISO）	—	—	8	MHz
	16位（双向）	—	—	8	MHz
	8位高速（MOSI+MISO）	—	—	18	MHz
	8位高速（双向）	—	—	18	MHz
	16位高速（MOSI+MISO）	—	—	16	MHz
	16位高速（双向）	—	—	16	MHz
f _{CLOCK}	组件时钟频率	—	2 × f _{SCLK}	—	MHz
t _{CKH}	SCLK为高电平的时间	—	0.5	—	1/f _{SCLK}
t _{CKL}	SCLK为低电平的时间	—	0.5	—	1/f _{SCLK}
t _{s_MISO}	MISO输入建立时间	25	—	—	ns
t _{H_MISO} ^[7]	MISO输入的保留时间	—	0	—	ns
t _{SS_SCLK}	SS有效到SCLK有效的时间	-20		20	ns
t _{SCLK_SS}	SCLK无效到SS无效的时间	-20		20	ns

⁶ 最大组件时钟频率派生自 t_{SCLK_MISO} 与 SCLK 输入和 MISO 输出的路由路径延迟的组合（本文中稍后将描述）。这些“Nominal”（额定值）数字提供了额定路由条件下组件的最大安全工作频率。可以在更高的时钟频率下运行组件，在更高的频率下运行时需要使用 STA 结果验证时序要求。

⁷ V_{DDIO} 供电电压的范围为 3.0 V 到 5.5 V。电压值可针对 PSoC 5 变化，因此必须与 STA 结果进行验证。

图 8. 在 **CPHA = 0** 模式下的时序图

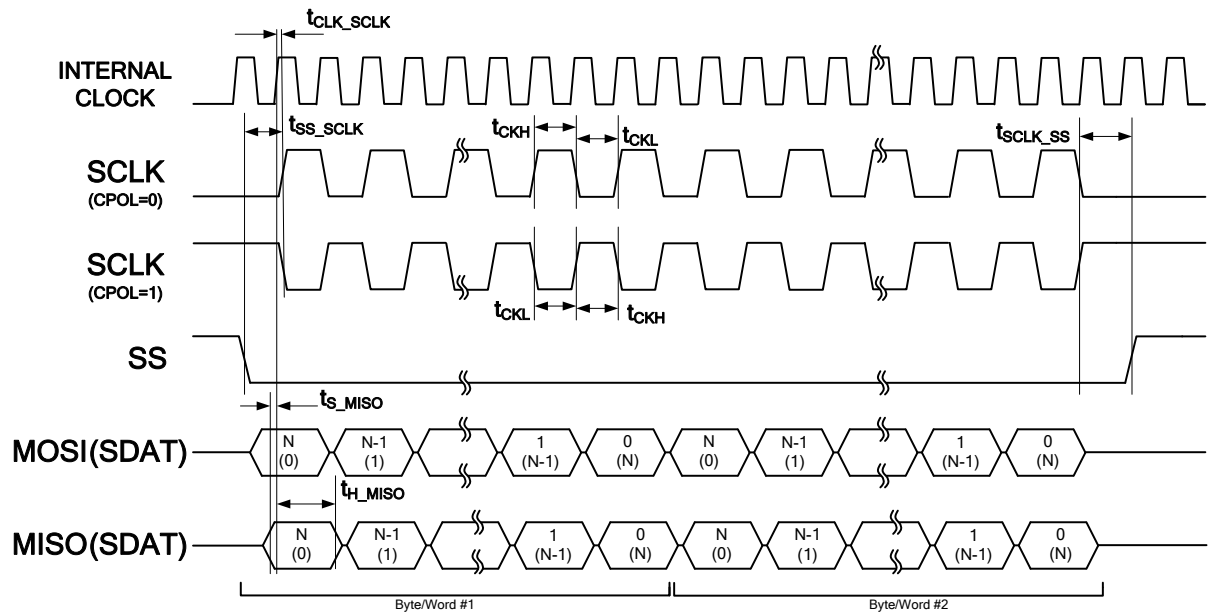
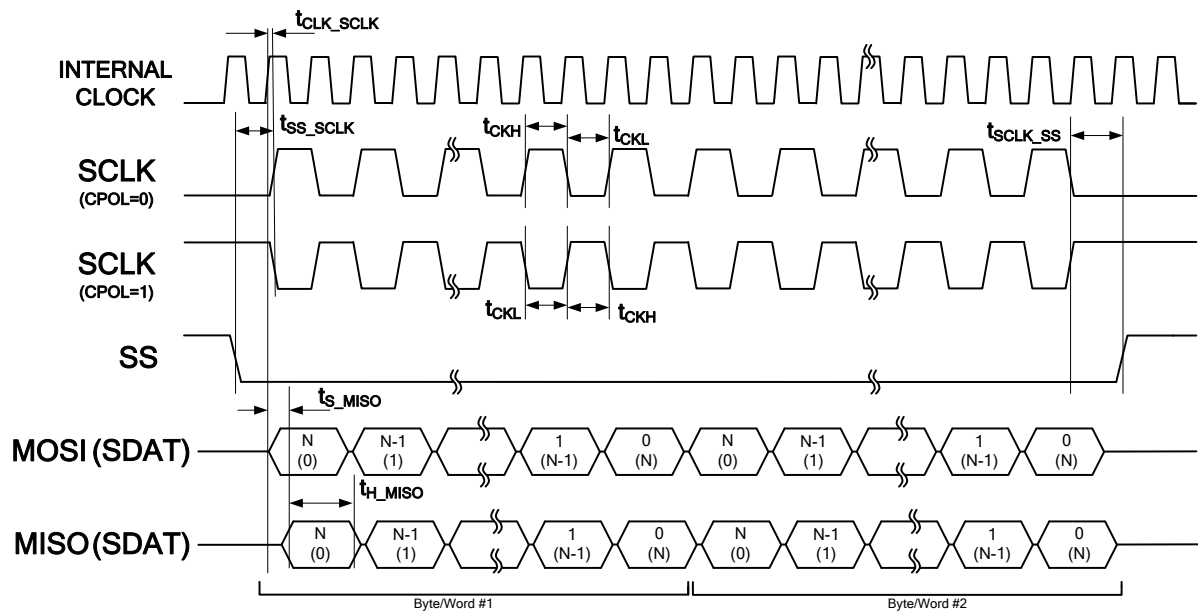


图 9. 模式的 **CPHA = 1** 时序图

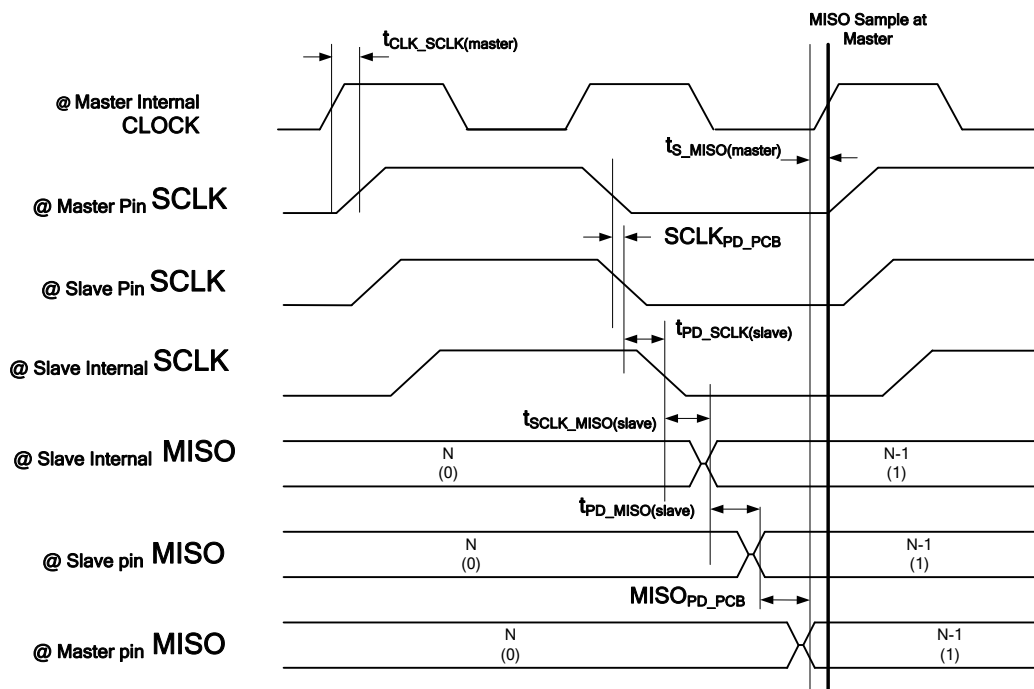


如何将 STA 结果用于特性数据

通过使用静态时序分析（STA）进行多次测试，从而收集额定最大值。您可以通过下列方法，使用 STA 结果计算设计的最大值：

f_{SCLK} STA 中不直接提供 SCLK 的最大频率（或最大比特率）。不过，STA 结果中所提供的数据指示了某些内部逻辑时序限制。要计算最大比特率，您必须考虑若干因素。需要电路板布局和从设备通信器件规范，才能充分理解最大值。此参数的主要限制因素是主设备引脚的 SCLK 下降沿到从设备的往复路径延迟，以及从设备的 MISO 输出返回到主设备的路径延迟。

图 10. 计算最大 f_{SCLK} 频率



在此情况下，组件必须使用下列公式满足主设备的 MISO 建立时间要求：

$$t_{RT_PD} < 1 / \{ [\frac{1}{2} \times f_{SCLK}] - t_{CLK_SCLK(master)} - t_{S_MISO(master)} \}$$

-- 或者 --

$$f_{SCLK} < 1 / \{ 2 * [T_{RT_PD} + t_{CLK_SCLK(master)} + t_{S_MISO(master)}] \}$$

其中：

$t_{CLK_SCLK(master)}$ 是输入 CLK 到 SCLK 输出的路径延迟。这是在 STA 结果时钟输出部分中提供的，如下所示：

- Clock To Output Section

- SPIM_1_IntClock

Source	Destination	Delay (ns)
Net 25/q	SCLK 1(0) PAD	24.800

tS_MISO(Master)是从 MISO 输入引脚到主设备内部逻辑的路径延迟。这是在 STA 结果输入时钟部分中提供的，如下所示：

- Input To Clock Section

- SPIM_1_IntClock

Source	Destination	Delay (ns)
MISO 1(0) PAD	\SPIM 1:BSPIM:sR8:Dp:u0\route si	20.906

对于高速模式，建立时间公式如下：

$$t_{RT_PD} < 1 / \{ [f_{SCLK}] - t_{CLK_SCLK(master)} - t_{S_MISO(master)} \}$$

-- 或者 --

$$f_{SCLK} < 1 / \{ [T_{RT_PD} + t_{CLK_SCLK(master)} + t_{S_MISO(master)}] \}$$

注意：报告中有两个值：寄存器 A0 和寄存器 A1 处的组件时钟建立时间。应选择较大值进行计算。

tRT_PD 的定义为：

$$t_{RT_PD} = [SCLK_{PD_PCB} + t_{PD_SCLK(slave)} + t_{SCLK_MISO(slave)} + t_{PD_MISO(slave)} + MISO_{PD_PCB}]$$

并且：

SCLKPD_PCB 是从主设备引脚到从设备引脚的 SCLK 的 PCB 路径延迟。

tPD_SCLK(slave) + tSCLK_MISO(slave) + tPD_MISO(slave)必须来自从设备数据手册。MISOPD_PCB 是由从设备引脚到主设备引脚的 MISO 的 PCB 路径延迟。

提供 SCLK 的最大频率、从而得到最大比特率的最终公式如下：

$$f_{SCLK} (Max.) = 1 / \{ 2 \times [t_{CLK_SCLK(master)} + SCLK_{PD_PCB} + t_{PD_SCLK(slave)} + t_{SCLK_MISO(slave)} + t_{PD_MISO(slave)} + MISO_{PD_PCB} + t_{S_MISO(master)}] \}$$

对于高速模式：

$$f_{SCLK} (Max.) = 1 / \{ [t_{CLK_SCLK(master)} + SCLK_{PD_PCB} + t_{PD_SCLK(slave)} + t_{SCLK_MISO(slave)} + t_{PD_MISO(slave)} + MISO_{PD_PCB} + t_{S_MISO(master)}] \}$$

fcLock

器件的最大时钟频率作为内部时钟（如果选择了内部时钟）或命名的外部时钟显示在时钟汇总中的时序结果中。下面是 STA 报告文件中的内部时钟限制的示例：



- Clock Summary Section

Clock	Type	Nominal Frequency (MHz)	Required Frequency (MHz)	Maximum Frequency (MHz)	Violation
BUS_CLK	Sync	60.000	60.000	N/A	
ClockBlock/clk bus	Async	60.000	60.000	N/A	
ClockBlock/dclk 0	Async	15.000	15.000	N/A	
ILO	Async	0.001	0.001	N/A	
IMO	Async	3.000	3.000	N/A	
MASTER_CLK	Sync	60.000	60.000	N/A	
PLL_OUT	Async	60.000	60.000	N/A	
SPIM 1 IntClock	Sync	15.000	15.000	61.870	

它的值被限制为 f_{CLOCK} 的 2 倍或者 STA 报告中的数字，但实际中，限值为 f_{CLOCK} 的 2 倍。

t_{CKH} SPI 主设备产生 50% 占空比的 SCLK。

t_{CKL} SPI 主设备产生 50% 占空比的 SCLK。

t_{CLK_SCLK} 内部时钟到 SCLK 输出的时间。从内部时钟上升沿到主设备引脚上可用 SCLK 的时间。

t_{S_MISO} 为了满足内部逻辑的建立时间，在引脚上的内部时钟进入有效状态前，引脚上的 MISO 必须有效。

t_{H_MISO} 为了满足内部逻辑的保留时间，在引脚上的内部时钟有效后，引脚上的 MISO 必须有效。

t_{SS_SCLK} 为了满足该模块的内部功能，在 SCLK 通过此参数在引脚处有效前，Slave Select（从设备选择）（SS）必须对该引脚有效。

t_{SCLK_SS} 最大值 — 为了满足模块的内部功能。该参数在引脚上的 SCLK 最后一个下降沿后，引脚上的 Slave Select（从设备选择）（SS）必须有效。

组件勘误表

本节列出了组件的已知问题。

赛普拉斯ID	组件版本	问题	解决方案
191257	v2.40	在没有修正PSoC Creator 3.0 SP1中的版本编号时进行更改这组件。有关更多信息，请参见基础知识文章KBA94159（网页地址： www.cypress.com/go/kba94159 ）。	解决方案并非必要的。设计不受任何影响。

组件更改

本节列出了组件与先前版本相比的主要更改的内容。

版本	更改说明	更改原因/影响
2.40.c	编辑了数据手册，以便添加组件勘误章节。	请注意，虽然组件被更改，但设计不受任何影响。
2.40.b	编辑了数据手册。	更新了“Configure选项卡”章节中的框图。 更新了章节的内容，使之符合模板。
2.40.a	编辑了数据手册。	删除了过时的PSoC 5器件的参考内容。
2.40	添加了MISRA合规性章节。	该组件具有所描述的特定偏差。
	向所有API增加了SPIM_rxBufferFull设置为零，可清除RX缓冲区。	在发生溢出后，软件RX缓冲区不能很好清除。
2.30	向“.cyre”文件中包括的所有组件API添加了CYREENTRANT关键词。	并非所有API都是真正可重入的函数。组件API源文件中的注释指出了适用的函数。 对于采用了安全方式并且是不可重入的函数，则需要该项变更，这样可以消除编译器警告：通过标志或关键节防止同时调用。
	添加了PSoC 5LP支持	
	向数据手册中添加了直流电特性一节	
2.21	添加了高速模式功能。修正了SPI模式图	更新了Verilog实现，这样最高比特率可达18 Mbps。添加了双向模式的详细说明。修改了SPI模式图，以便隐藏了Rx数据作，从而可以详细介绍内部实现。将高速模式的说明添加到“Advanced（高级）选项卡”部分中。在“资源”、“交流与直流电气特性”和“如何将STA结果用于特性数据”等部分中，更新了与高速模式相关的数据。
2.20	已经用cy_clock_v1_60更新了内部时钟组件。修复引起STA警告的Verilog缺陷。	时钟v1_60是组件的最新版本。修复了Verilog缺陷，以便删除使用更新的STA工具发现的STA警告。使用新STA工具生成的报告来更新“交流与直流电气特性”一节的截图。
2.10.a	数据手册纠正（已经纠正了模式的图表（其中CPHA = 1））	修正数据手册缺陷
2.10	将数据位范围从2到16位更改为3到16位。	更改了与状态同步相关的问题，在当前版本中修复了这些问题。
	向组件配置对话框中添加了“SPI空闲时的中断”复选框。	修复了组件自定义程序缺陷

版本	更改说明	更改原因/影响
	将“Byte transfer complete”（字节传输完成）复选框的名称更改为“Byte/Word transfer complete”（字节/字传输完成）	目的是符合真实含义
	向数据手册添加了特性数据	
	对数据手册进行了少量编辑和更新	
2.0.a	将组件移动到组件目录的子文件夹中。	
2.0	添加了 SPIM_Sleep()/SPIM_Wakeup() 和 SPIM_Init()/SPIM_Enable API。	用于支持低功耗模式，并提供常用接口，以单独控制大多数组件的初始化和使能。
	更改了组件I/O的数量和位置： <ul style="list-style-type: none"> 在默认放置中现在可以看到时钟输入（外部时钟源现在是默认设置） 复位输入具有另一个位置 删除了中断输出。添加了 rx_interrupt、tx_interrupt输出。 	<p>添加了时钟输入，以便与SPI从设备保持一致。</p> <p>由于添加了时钟输入，因此更改了复位输入位置。现在提供了两个状态中断寄存器（Tx和Rx），而不是提供一个共享寄存器。</p> <p>当从以前SPI的版本迁移时需要考虑这些更改以避免绑定错误</p>
	删除了SPIM_EnableInt()、SPIM_DisableInt()、SPIM_SetInterruptMode()和SPIM_ReadStatus()等API函数。 添加了SPIM_EnableTxInt()、SPIM_EnableRxInt()、SPIM_DisableTxInt()、SPIM_DisableRxInt()、SPIM_SetTxInterruptMode()、SPIM_SetRxInterruptMode()、SPIM_ReadTxStatus()和SPIM_ReadRxStatus() API。	被删除的API已过时，因为现在组件包含了Rx和Tx中断，而不是包含一个共享中断。另外，还更新了Tx和Rx缓冲区中的断处理程序实现。
	分别将SPIM_ReadByte()和SPIM_WriteByte()API重命名为SPIM_ReadRxData()和SPIM_WriteTxData()。	阐明了各API以及使用它们的方式。
	在使用Verilog实现的基本SPI主设备 B_SPI_Master_v2_0中，进行了下列更改：	
	spim_ctrl内部模块替换为新状态机。	它使用较少的硬件资源，并且不含任何异步逻辑。

版本	更改说明	更改原因/影响
	<p>现在提供了两个状态寄存器（分别为Tx和Rx提供了状态），而不是将一个共用状态寄存器用于二者。</p> <pre>/*SPI_Master_v1_20 status bits*/ SPIM_STS_SPI_DONE_BIT = 3'd0; SPIM_STS_TX_FIFO_EMPTY_BIT = 3'd1; SPIM_STS_TX_FIFO_NOT_FULL_BIT = 3'd2; SPIM_STS_RX_FIFO_FULL_BIT = 3'd3; SPIM_STS_RX_FIFO_NOT_EMPTY_BIT = 3'd4; SPIM_STS_RX_FIFO_OVERRUN_BIT = 3'd5; SPIM_STS_BYTE_COMPLETE_BIT = 3'd6; /*SPI_Master_v2_0 status bits*/ localparam SPIM_STS_SPI_DONE_BIT = 3'd0; localparam SPIM_STS_TX_FIFO_EMPTY_BIT = 3'd1; localparam SPIM_STS_TX_FIFO_NOT_FULL_BIT = 3'd2; localparam SPIM_STS_BYTE_COMPLETE_BIT = 3'd3; localparam SPIM_STS_SPI_IDLE_BIT = 3'd4; localparam SPIM_STS_RX_FIFO_FULL_BIT = 3'd4; localparam SPIM_STS_RX_FIFO_NOT_EMPTY_BIT = 3'd5; localparam SPIM_STS_RX_FIFO_OVERRUN_BIT = 3'd6;</pre>	<p>修复了以前版本中发现的软件缓冲区未按预期工作的缺陷。</p>



版本	更改说明	更改原因/影响
	基本组件中添加了“ BidirectMode ”布尔参数。 现在已选择含“时钟”输入和 SYNC 模式位的 Control Register （控制寄存器）来驱动 PSoC 3 Production 芯片的“ tx_enable ”输出。当选择 ES2 芯片时， Control Register （控制寄存器） w/o 时钟输入驱动“ tx_enable ”。 组件原理图中使用了 Bufoe 组件，以支持双向模式。基本组件的 MOSI 输出连接到 bufoe ‘x’ 输入。 ‘yfb’ 连接至 ‘miso’ 输入。则 Bufoe ‘y’ 输出连接到“ sdat ”输出终端。 将路由的复位连接到数据路径、计数器 7 和状态机。	添加了针对组件的双向模式支持
	将带有同步 = “ TRUE ”参数的 udb_clock_enable 组件添加到 Verilog 实现中。	新增了对用于指示功能的 Verilog 中使用的所有时钟的要求，使该工具可以支持同步和静态时序分析。
	在 Counter7 （计数器 7）周期值中，“ *2 ”被替换为“ << 1 ”。	改进了 Verilog 。
	将最大比特率值更改为 10 Mbps	不支持大于 10 Mbps 的比特率值（在组件特性化期间已验证）
	添加了双向模式的说明	修复了数据手册缺陷
	目前，复位输入说明包含有关 ES2 芯片不兼容性的注释	修复了数据手册缺陷
	更改了 SS 与 SCLK 信号之间的时序关联图	修复了数据手册缺陷
	移除了示例固件源代码	添加了对组件示例项目的参考
	更改了 SPI 模式图（添加了 Tx 和 Rx FIFO 状态值）	修复了数据手册缺陷

赛普拉斯半导体公司，2012-2016 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可）（1）在赛普拉斯软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担任何全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。

