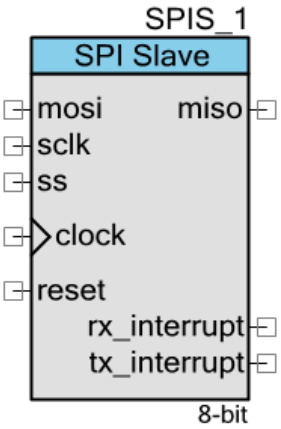
# 串行外设接口（SPI）从设备

2.60

## 特性

- 3~16 位数据宽度
- 四个 SPI 模式
- 比特率高达 5 Mbps[1]




## 概述

SPI 从设备提供了一个符合行业标准的 4 线从设备 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个 SPI 操作模式，允许与任何 SPI 主设备进行通信。除了标准的 8 位字长外，SPI 从设备还支持可配置的 3 至 16 位字长，用于与非标准的 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 从设备

只要将 PSoC 器件连接至 SPI 主设备时，便可以使用 SPI 从设备组件。除了标有“SPI 主设备”的器件外，可以将 SPI 从设备与许多器件配合使用，从而实现移位寄存器类型接口。

在需要将 PSoC 器件与 SPI 从设备进行通信时，应使用 SPI 主设备组件。应在移位寄存器组件的低灵活性提供的硬件功能在 SPI 从设备组件中不可用的情况下使用它。

[1] 此值仅对 MOSI 和 MISO（全双工）接口模式（请参见直流和交流电气特性一节，了解更详细的信息）才有效；但在双向模式下，最高限制为 1 Mbps（由于内部双向引脚限制）。

# 输入/输出接口

本节介绍了 SPI 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情况下，该 I/O 可能不可见。

**注意**：如果不使用原理图宏，则需要配置引脚组件，以为每个已分配的输入引脚（MOSI、SCLK 和 SS）将“Sync Mode”参数设为“Transparent”。在相应“Pins Configure”对话框‘mosi – 输入*’中，依次选择 Pins（引脚）> Input（输入）选项卡，可找到该参数。

mosi 输入承载来自主设备的主出从入（MOSI）信号。当将 **Data Lines**（数据线）参数设置为“MOSI + MISO”时，可见到此输入。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。当 **Data Lines** 参数被设为 **Bidirectional**（双向）时，使用此输入。对于 PSoC 3 和 PSoC 5 芯片，当执行时序分析时，会在器件时钟与 SCLK 信号之间提示异步时钟交叉警告。下面是此类信息的示例：“Path(s) exist between clocks IntClock and SCLK(0)_PAD, but the clocks are not synchronous to each other”。该信息适用于从控制方向的寄存器到 SCLK 进行的数据采样之间的路径。在更改方向时，不应运行 SCLK。只要遵循此规则，便不会有问题，并且您可以忽略此信息。

**注意**：双向模式提供了内部回送功能，因此在两种模式的任一模式下，另一方向仍活动（填充或清空其缓冲区）。

**图 1. SPI 双向模式（数据由主设备传输到从设备）**

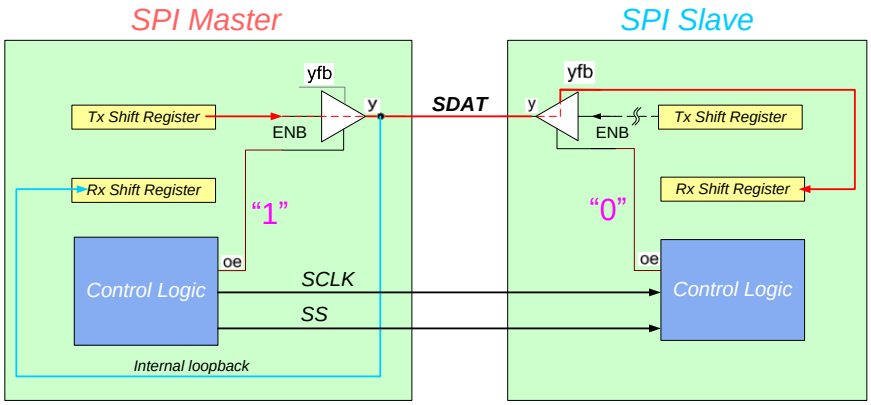
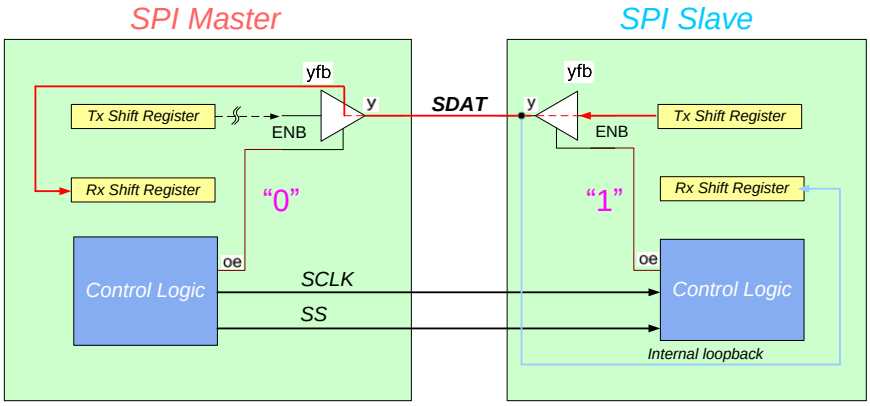


**图 2. SPI 双向模式（数据由从设备传输到主设备）**




在双向模式下，器件的初始状态为 Rx 模式或从主设备到从设备数据传输，如图 2 所示。应当使用 SPIS_TxEnable()和 SPIS_Tx_Disable()等 IPI 函数切换 Rx 和 Tx 模式。

## sclk — 输入

sclk 输入承载串行时钟（SCLK）信号。它向该器件提供了从设备同步时钟输入。该输入始终可见，并必须处于连接状态。

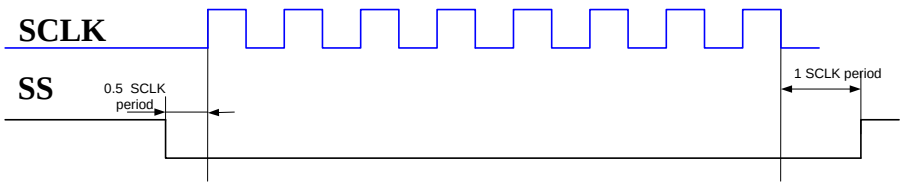
**注意**：某些 SPI 主设备（例如，TotalPhase Aardvark I2C/SPI 主机适配器）以特定方式驱动 sclk 输出。为了使 SPI 从设备组件与此类器件在 CPOL = 1 的模式下正常运行， 该将 sclk 引脚设为电阻上拉驱动模式。否则，它将输出损坏的数据。欲了解更多有关模式的信息，请参考功能说明一节中所介绍的内容。

## ss — 输入

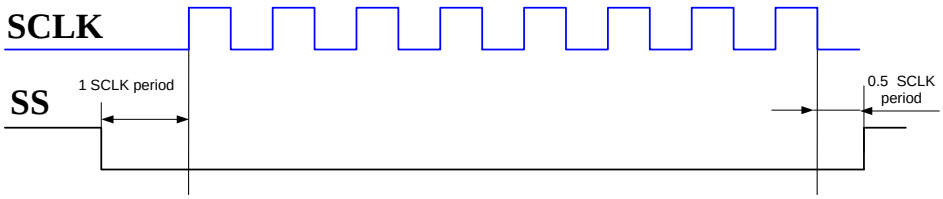
ss 输入承载传输至器件的从设备选择（SS）信号。该输入始终可见，并必须处于连接状态。

下图显示的是 SCLK 和 SS 信号之间的时序相关性

CPHA = 0 时：




CPHA = 1 时：



**注意**：这些图中所显示的 SS 时序对于 PSoC Creator SPI 主设备有效。一般来说，SS 下降沿和第一个 SCLK 边沿之间只需要半个 SCLK 周期延迟，就可以使 SPI 从设备在所有受支持的比特率范围内正常工作。

## 复位 — 输入

复位输入将复位 SPI 从设备。它会删除当前正在传输或接收的任何数据，但不会清除 FIFO 中已经收到或准备传输的数据。PSoC 5 芯片不支持此复位功能，因此当用于这些器件时，会忽略此输入。执行时序分析时，使用复位输入会导致在生成复位输入的时钟与 SCLK 信号之间报告异步时钟交叉警告。下面是此类信息的示例：“Path(s) exist between clocks BUS_CLK and SCLK(0)_PAD, but the clocks are not synchronous to each other”。该信息适用于从“Reset”信号到由 SCLK 提供脉冲的 SPI 组件操作之间的路径。更改“Reset”信号时，不应运行 SCLK。只要遵循此规则，便不会有问题，并且您可以忽略此信息。

复位输入可以保持悬空，不与外部连接。如果复位线上没有任何连接，器件会给它分配一个常量逻辑 0。

## 时钟 — 输入*

时钟输入定义了状态寄存器的采样速率。所有的给数据提供时钟脉冲都在 sclk 输入上发生，因此，时钟输入将**不会**处理 SPI 从设备的比特率。

将 **Clock Selection** 参数设置为 **External** 时，该时钟输入会可见。可见时，则必须连接此输入。

## miso — 输出 *

miso 输出将主入从出（MISO）信号发送至总线上的主设备。如果 **Data Lines** 参数被设为 **MOSI + MISO**，该输出会可见。

## 中断 — 输出

中断输出是可能中断源系列的逻辑“或”（OR）。任何已使能中断源为“true”时，此信号将变为高电平。

# 原理图宏信息

默认情况下，PSoC Creator 组件目录包括 SPI 从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。原理图宏适用于 4 线（全双工）、3 线（双向）和全双工多从 SPI 连接。

**图 3. 4 线（全双工）连接原理图宏**

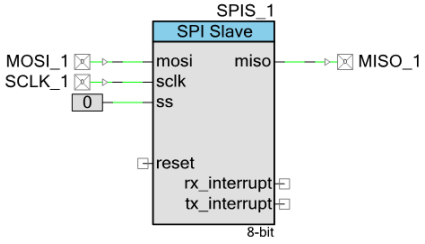



**图 4. 3 线（双向）连接原理图宏**

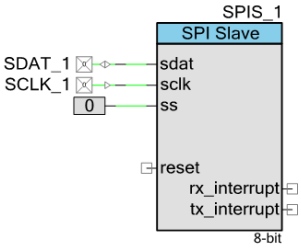
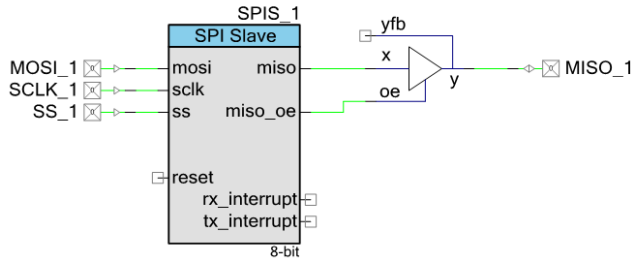



**图 5. 多从模式原理图宏**




**注意：**如果不使用原理图宏，则需要配置引脚组件，以为每个已分配的输入引脚（MOSI、SCLK 和 SS）将 **Sync Mode** 参数设为 **Transparent**。该参数位于相应“Pins Configure”对话框的 **Pins > Input** 选项卡中。

# 组件参数

将一个 SPI 从设备组件拖放到设计上。双击组件符号，以打开 **Configure** 对话框。

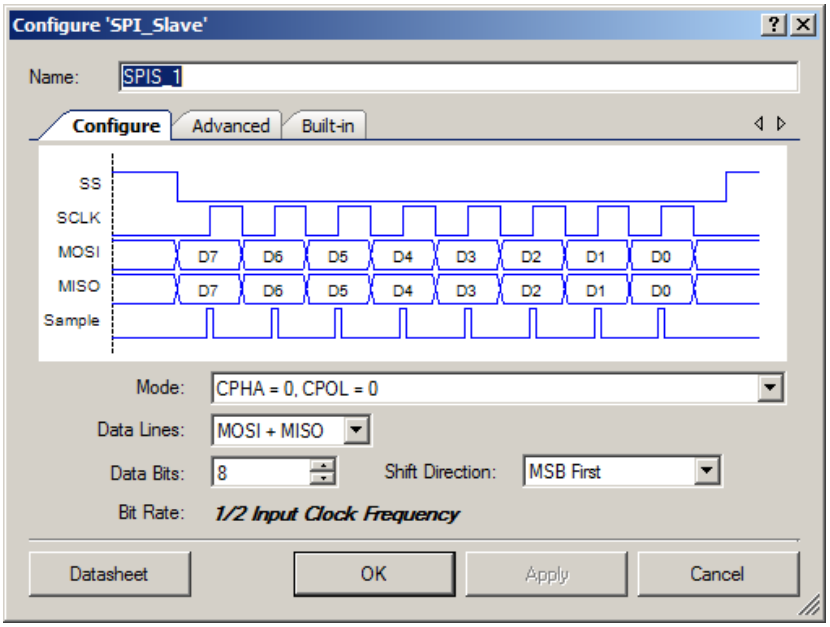
以下各节将介绍 SPI 从设备参数，以及如何使用“Configure”（配置）对话框对其进行配置。它们还指示已经选择了硬件还是软件。

## 硬件与软件配置选项

硬件配置选项用于更改项目合成，并放置在硬件中。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。如果在构建之前设置这些参数，即正在设置其初始值，那么这些初始值随时可能由所提供的 API 修改。仅有硬件参数才标有星号（*）。

## Configure 选项卡

**Configure**（配置）选项卡包含每个 SPI 组件所需要的基本参数。




**注意**：波形中的示例信号并不是系统的输入或输出；它只表示何时在主设备或从设备针对所选的模式设置进行数据采样。

### Mode（模式）*

**Mode**（模式）参数定义了通信时需要使用的时钟相位以及时钟极性模式。下表中定义了这些模式。另请参阅此数据手册的功能说明一节中所介绍的内容。

| 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（**Bidirectional**）SPI 通信接口。

### Data Bits（数据位）*

**Data Bits**（数据位）的数量定义了在 SPIS_ReadRxData()和 SPIS_WriteTxData() API 进行传输时所使用的单个传输的位宽度。默认位数为单字节（8 位）。有效的取值范围为 3 到 16 之间的任何整数。

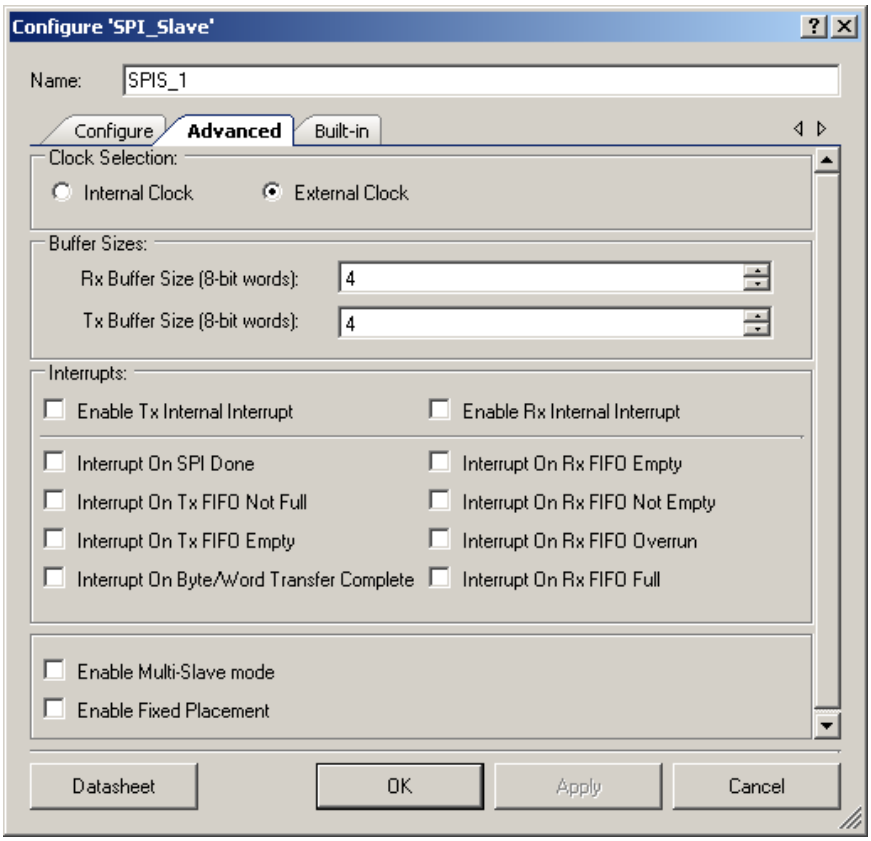
### Shift Direction（移位方向）*

**Shift Direction**（移位方向）参数用于定义串行数据的传输方向。设置为 **MSB First**（MSB 优先）时，会先传输最高有效位。通过将数据向左移位实现此操作。设置为 **LSB First**（LSB 优先）时，则先传输最低有效位。这是通过数据右移实现的。

### Bit Rate（比特率）*

如果位于 **Advanced** 选项卡上的 **Clock Selection** 参数被设为 **Internal Clock**，那么 **Bit Rate** 参数将定义 SCLK 速率（单位为赫兹）。内部时钟的时钟频率将为 SCLK 速率的 2 倍。如果 **Clock Selection** 参数被设为 **External Clock** 时，则该参数不起任何作用。

## Advanced（高级）选项卡




### Clock Selection（时钟选择）

**Clock Selection**（时钟选择）参数指定了使用内部时钟还是外部时钟。请参考本数据手册中后面的时钟选择一节的内容，了解更多有关的信息。

### Rx Buffer Size（Rx 缓冲区大小）*

**Rx Buffer Size**（Rx 缓冲区大小）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）。如果将此参数设置为 1 到 4，则在硬件中实现 FIFO 的第 4 个字节/字。数值为 1 到 3 仅用于与以前版本的兼容性；使用它们会导致出现一个指示该值不正确的错误信息。所有其他长度不超过 255 的值将使用 4 字节/字 FIFO 和已给的 API 所控制的存储器阵列。

### Tx Buffer Size（Tx 缓冲区大小）*

**Tx Buffer Size**（Tx 缓冲区大小）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）。如果将此参数设置为 1 到 4，则在硬件中实现 FIFO 的第 4 个字节/字。数值为 1 到 3 仅用于与以前版本的兼容性；使用它们会导致出现一个指示该值不正确的错误信息。所有其他长度不超过 255 的值将使用 4 字节/字 FIFO 和已给的 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。编写第一个数据后，BufferWrite 指针将移至软件缓冲区的第一个元素，并指向写数据；BufferRead 指针仍留在零元素。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 Tx/Rx Internal Interrupt（使能 Tx/Rx 内部中断）

**Enable Tx/Rx Internal Interrupt**（使能 Tx/Rx 内部中断）选项允许您可使用 SPI 从设备组件的预先定义 Tx 和 Rx ISR，或提供您自己的自定义 ISR。使能这些选项时，您可以将自己的代码添加到这些预定义 ISR 中（如果只需少量更改）。如果不选择内部中断，则可以使用连接到 SPI 从设备中断输出的自定义代码来提供内部中断器件。

当 Rx 或 Tx 缓冲区大小大于 4，器件会自动设置相应的参数，因为需要使用内部 ISR 处理从硬件 FIFO 到 Rx 和/或 Tx 缓冲区的传输。SPI 从设备的中断输出引脚始终可见且可用，它所输出的信号与传输到内部中断的信号相同。此输出可以作为 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 缓冲区大小大于 4 个字节/字时，“Rx FIFO NOT EMPTY”中断始终处于使能状态，不能禁用它，否则会导致错误的缓冲区功能。
- 如果 Tx 缓冲区大小大于 4 个字节/字，“Tx FIFO NOT FULL”中断始终处于使能状态，不能禁用它，否则会导致错误的缓冲区功能。
- 对于大于 4 个字节/字的缓冲区大小，必须使能全局中断和组件中断，这样才能正确地处理缓冲区。

### Interrupts（中断）

**Interrupts**（中断）选项参数允许用户可配置已使能的导致中断的内部事件。所有使能的 Tx 和 Rx 状态寄存器屏蔽位的逻辑“或”运算都可以生成中断。通过这些参数所选的位定义了使用初始组件配置实现的屏蔽。

### 使能多从模式

在当前 SPI 从设备组件通过其他 SPI 从设备连接至共享总线时，会使用该设置。MISO_OE 输出在组件符号上变为可见。在该模式下，外部 BUF_OE 组件应连接至 MISO 输出。SS 线处于高电平时，该模式允许将 MISO 输出调试为高阻抗状态。多从模式宏可用于迅速提供所有必要的连接。

### 使能固定放置

与不受限制的放置相比，此设置可用于提高 SPI 从设备组件性能。固定放置为组件提供单个放置。这意味着在单个设计中只能使用组件的一个实例。连接到组件的引脚放置不受控制，但最好应该使用 P[0]的引脚实现最佳性能。

组件的固定放置方面移除了对于“所有走线的最大值”情况需要考虑的差异性（有关详细信息，请参见直流和交流电气特性一节中的内容）。此外，还可以在一个完全空白的设计中按照与非固定放置设计相同的方式继续运行固定放置。

## 时钟选择

当选中内部时钟配置时，PSoC Creator 计算所需的频率和时钟源的频率，并生成实现所需的时钟资源。否则，用户必须提供时钟组件并计算所需的时钟频率。该频率至少为最大比特率和 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 能够使用当前系统时钟频率提供该数值。否则，在构建项目时会显示一条有关时钟准确度范围的警告。该警告将包含 PSoC Creator 设置的真实时钟值。选择是应当更改系统时钟还是组件时钟以满足时钟设置系统要求，达到最佳值。

## 放置

SPI 从设备组件放置于 UDB 阵列中，并且 *cyfitter.h* 文件为 API 提供了所有放置信息。

# 应用编程接口

通过应用编程接口（API），您可以使用软件对组件进行配置。下表列出并说明了每个函数的接口。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“SPIS_1”分配给指定设计中组件的第一个实例。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。实例名称会成为与该组件相关的每个全局函数名称、变量和常量符号的前缀。为了增加可读性，下表中使用了实例名称“SPIS”。

| 函数 | 说明 |
|---|---|
| SPIS_Start() | 调用SPIS_Init()和SPIS_Enable()。应当在第一次启动组件时调用。 |
| SPIS_Stop() | 禁用SPIS操作。 |
| SPIS_EnableTxInt() | 使能内部Tx中断请求。 |
| SPIS_EnableRxInt() | 使能内部 Rx中断请求。 |
| SPIS_DisableTxInt() | 禁用内部Tx中断请求。 |
| SPIS_DisableRxInt() | 禁用内部Rx中断请求。 |
| SPIS_SetTxInterruptMode() | 使能Tx中断源。 |
| SPIS_SetRxInterruptMode() | 使能Rx中断源。 |
| SPIS_ReadTxStatus() | 返回Tx状态寄存器的当前状态。 |
| SPIS_ReadRxStatus() | 返回Rx状态寄存器的当前状态。 |
| SPIS_WriteTxData() | 在传输缓冲区中放置将于下一个可用总线时间发送的字节/字。 |
| SPIS_WriteTxDataZero() | 直接在移位寄存器中放置一个字节/字。这是CPHA = 0时SPI模式所需的条件。 |
| SPIS_ReadRxData() | 返回接收缓冲区中可用接收数据的下一个字节/字。 |
| SPIS_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）。 |
| SPIS_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）。 |
| SPIS_ClearRxBuffer() | 清除了Rx缓冲区存储器阵列和Rx FIFO中所有收到的数据。 |
| SPIS_ClearTxBuffer() | 清除 Tx 缓冲区存储器阵列或Tx FIFO中所有要发送的数据。 |
| SPIS_TxEnable() | 如果SPI组件被配置为双向模式，则该函数将设置SDAT输入输出引脚，以进行传输数据。 |
| SPIS_TxDisable() | 如果配置为双向模式，则该函数将设置SDAT输入输出引脚，已进行接收数据。 |
| SPIS_PutArray() | 将数据阵列放入传输缓冲区内。 |
| SPIS_ClearFIFO() | 清除了RX和TX FIFO的所有数据，以重新开始。 |
| SPIS_Sleep() | 通过调用SPIS_SaveConfig()和SPIS_Stop()函数，使SPIS组件准备进入低功耗模式。 |

| 函数 | 说明 |
|---|---|
| SPIS_Wakeup() | 从低功耗模式唤醒后，恢复并重新使能SPIS组件。 |
| SPIS_Init() | 初始化并恢复默认的SPIS配置。 |
| SPIS_Enable() | 使能SPIS以开始操作。 |
| SPIS_SaveConfig() | 保存SPIS硬件配置。 |
| SPIS_RestoreConfig() | 恢复SPIS硬件配置。 |

## 全局变量

| 函数 | 说明 |
|---|---|
| SPIS_initVar | SPIS_initVar指示SPI从设备组件是否已被初始化。将该变量初始化为0，并在第一次调用SPIS_Start()时将它设置为1。这样，第一次调用SPIS_Start()子程序后，组件不用重新初始化仍可重启。<br>如果需重新初始化器件，可在调用SPIS_Start()或SPIS_Enable()函数前先调用SPIS_Init()函数。 |
| SPIS_txBufferWrite | 表示由API将最终数据写入缓冲区内的传输缓冲区位置。 |
| SPIS_txBufferRead | 表示从缓冲区读取并由SPIS硬件发送的最后数据的传输缓冲区位置。 |
| SPIS_rxBufferWrite | 表示由SPIS硬件接收到后写入缓冲区内的最后数据的接收缓冲区位置。 |
| SPIS_rxBufferRead | 表示API从缓冲区读取的最终数据的接收缓冲区位置。 |
| SPIS_rxBufferFull | 指示软件缓冲区已发生溢出。 |
| SPIS_RxBUFFER[] | 用于存储接收到的数据。 |
| SPIS_TxBUFFER[] | 用于存储需要发送的数据。 |

## void SPIS_Start(void)

**说明：** 这是开始执行组件操作的首选方法。通过SPIS_Start()可设置initVar变量，调用SPIS_Init()函数，然后调用SPIS_Enable()函数。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_Stop(void)

**说明：** 禁用了SPI从设备组件中断。不影响SPIS操作。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_EnableTxInt(void)

**说明：** 使能了内部Tx中断请求。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_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 SPIS_DisableTxInt(void)

**说明：** 禁用了内部Tx中断请求

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_DisableRxInt(void)

**说明：** 禁用了内部Rx中断请求

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_SetTxInterruptMode(uint8 intSrc)

**说明：** 配置了已使能的Tx中断源。

**参数：** uint8 intSrc：包含需要使能的中断的位字段。

| 位 | 说明 |
|---|---|
| SPIS_STS_SPI_DONE | 由于SPI完成而使能中断 |
| SPIS_STS_Tx_FIFO_EMPTY | 由于Tx FIFO为空而使能中断 |
| SPIS_STS_Tx_FIFO_NOT_FULL | 由于Tx FIFO未满而使能中断 |
| SPIS_STS_BYTE_COMPLETE | 由于字节/字传输操作完成而使能中断 |

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。更多有关的信息，请参考本数据手册中的定义一节的内容。

**返回值：** 无

**其他影响：** 无

## void SPIS_SetRxInterruptMode(uint8 intSrc)

**说明：** 配置已使能的Rx中断源。

**参数：** uint8 intSrc：包含需要使能的中断的位字段。

| 位 | 说明 |
|---|---|
| SPIS_STS_Rx_FIFO_EMPTY | 由于Rx FIFO为空而使能中断 |
| SPIS_STS_Rx_FIFO_NOT_EMPTY | 由于Rx FIFO非为空而使能中断 |
| SPIS_STS_Rx_FIFO_OVERRUN | 由于Rx缓冲区发生溢出而使能中断 |
| SPIS_STS_Rx_FIFO_FULL | 由于Rx FIFO已满而使能中断 |

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。更多有关的信息，请参考本数据手册中的定义一节的内容。

**返回值：** 无

**其他影响：** 无

## uint8 SPIS_ReadTxStatus(void)

**说明：** 返回Tx状态寄存器的当前状态。更多有关的信息，请参见本数据手册中状态寄存器位一节的内容。

**参数：** 无

**返回值：** uint8：当前的Tx状态寄存器值

| 位 | 说明 |
|---|---|
| SPIS_STS_SPI_DONE | SPI已完成 |
| SPIS_STS_Tx_FIFO_EMPTY | Tx FIFO为空 |
| SPIS_STS_Tx_FIFO_NOT_FULL | Tx FIFO未满 |
| SPIS_STS_BYTE_COMPLETE | 字节/字的传输已完成 |

**其他影响：** Tx状态寄存器位在读取后被清除。

## uint8 SPIS_ReadRxStatus(void)

**说明：** 返回Rx状态寄存器的当前状态。更多有关的信息，请参见本数据手册中状态寄存器位一节的内容。

**参数：** 无

**返回值：** uint8：当前的Rx状态寄存器值

| 位 | 说明 |
|---|---|
| SPIS_STS_Rx_FIFO_EMPTY | Rx FIFO为空 |
| SPIS_STS_Rx_FIFO_NOT_EMPTY | Rx FIFO非为空 |
| SPIS_STS_Rx_FIFO_OVERRUN | Rx缓冲区发生溢出现象 |
| SPIS_STS_Rx_FIFO_FULL | Rx FIFO已满 |

**其他影响：** Rx状态寄存器位在读取后被清除。

## void SPIS_WriteTxData(uint8/uint16 txData)

**说明：** 在传输缓冲区中放置一个字节，该字节将在下一个可用总线时间发送。

**参数：** uint8/uint16: txData：需要通过SPI发送的数据值

**返回值：** 无

**其他影响：** 数据可能被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能传输。该函数一直处于封锁状态，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。

## void SPIS_WriteTxDataZero(uint8/uint16 txData)

**说明：** 将一个字节/字直接放入移位寄存器中，以进行传输。在下一个时钟相位过程中，将此字节/字发送到主设备。

**参数：** uint8/uint16: txData：需要通过SPI发送的数据值

**返回值：** 无

**其他影响：** 这是各种模式在CPHA = 0时的必要条件。在这些模式下，数据必须位于第一个时钟边沿前的移位寄存器中。如果已有被的数据，并且FIFO中有更多数据，那么固件必须控制该子程序。该子程序不应用于CPHA = 1时的模式。

## uint8/uint16 SPIS_ReadRxData(void)

**说明：** 读取通过SPI接收到的下一数据字节。

**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，将返回无效数据。调用SPIS_GetRxBufferSize()，并且如果它返回一个非零值，则可安全地调用SPIS_ReadRxData()函数。

## uint8 SPIS_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 非为空）。
- 如果使能了 Rx 软件缓冲区，则此函数返回 Rx 软件缓冲区中的数据大小。此计数中不包括 FIFO 数据。

**参数：** 无

**返回值：** uint8：Rx缓冲区中字节数/字数的取整计数。

**其他影响：** 清除组件的Rx状态寄存器。

## uint8 SPIS_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，此函数会返回 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，则此函数将返回 Tx 软件缓冲区中的数据大小。此计数中不包括 FIFO 数据。

**参数：** 无

**返回值：** uint8：Tx缓冲区中字节数/字数的取整计数。

**其他影响：** 清除组件的Tx状态寄存器。

## void SPIS_ClearRxBuffer(void)

**说明：** 清除Rx缓冲区存储器阵列和Rx硬件FIFO中所有接收到的数据。通过将读取指针和写入指针都设置为零，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中的所有数据。

**参数：** 无

**返回值：** 无

**其他影响：** 当新数据覆盖时，任何尚未从RAM缓冲区和FIFO中读取的已收到数据将被丢失。

## void SPIS_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，任何尚未从RAM缓冲区传输的数据将被丢失。

## void SPIS_TxEnable(void)

**说明：** 如果配置SPI从设备使用单个双向引脚，那么该函数会将双向引脚设置为发送。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_TxDisable(void)

**说明：** 如果配置SPI从设备使用单个双向引脚，那么该函数会将双向引脚设置为接收。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_PutArray(uint8/uint16 *buffer、uint8 byteCount)

**说明：** 当空间可用时，将来自RAM/ROM的可用数据写入Tx缓冲区内。坚持尝试，直至所有数据传递至Tx缓冲区内为止。如果使用CPHA = 0的模式，则在调用SPIS_PutArray()函数前先调用SPIS_WriteTxDataZero()函数。

**参数：** uint8/uint16 *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数/字数

**返回值：** 无

**其他影响：** 系统将阻塞在此函数位置，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，此函数将阻塞系统。如果从设备未传输数据而Tx缓冲区已满，则此函数在该循环中保持锁定状态。

## void SPIS_ClearFIFO(void)

**说明：** 清除了RX和TX FIFO的所有数据，以重新开始。

**参数：** 无

**返回值：** 无

**其他影响：** 清除组件的状态寄存器。

## void SPIS_Sleep(void)

**说明：** 这是使器件准备进入低功耗模式的首选子程序。SPIS_Sleep()子程序先保存当前组件的状态。然后，它依次调用SPIS_Stop()和SPIS_SaveConfig()函数，以保存硬件配置。

在调用CyPmSleep()或CyPmHibernate()函数前，先调用 SPIS_Sleep()函数。有关功耗管理函数的详细信息，请参考PSoC Creator *《系统参考指南》*。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_Wakeup(void)

**说明：** 该函数是将组件恢复到调用SPIS_Sleep()时状态的首选子程序。SPIS_Wakeup()函数调用SPIS_RestoreConfig()函数，以恢复配置。如果调用SPIS_Sleep()函数前使能了组件，则SPIS_Wakeup()函数也会重新使能组件。清除Rx缓冲区、Tx缓冲区以及硬件FIFO中的所有数据。

**参数：** 无

**返回值：** 无

**其他影响：** 如果调用SPIS_Wakeup()函数前未调用SPIS_Sleep()或SPIS_SaveConfig()函数，可能会产生意外行为。

## void SPIS_Init(void)

**说明：** 根据自定义程序“Configure”对话框设置，初始化或恢复组件。不需要调用SPIS_Init()，因为SPIS_Start()子程序会调用此函数，并且这是开始器件操作的首选方法。

**参数：** 无

**返回值：** 无

**其他影响：** 当调用此函数时，它将初始化所有需要执行的参数。包括设置初始中断屏蔽、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 SPIS_Enable(void)

**说明：** 使能SPIS以开始操作。启动内部时钟（如果这样配置）。如果配置了外部时钟，必须在调用此API前单独启动外部时钟。应在使能SPIS中断前调用SPIS_Enable()函数。这是因为此函数配置中断源，然后使能内部中断（如果这样配置）。之前必须已调用SPIS_Init()函数。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_SaveConfig(void)

**说明：** 此函数会保存组件配置以及非保留寄存器。它还保存“Configure”对话框中定义的或通过相应API修改的当前组件参数值。此函数由SPIS_Sleep()函数调用。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void SPIS_RestoreConfig(void)

**说明：** 从较低功耗模式唤醒后，恢复SPIS_SaveConfig()函数所保存的SPIS硬件配置。

**参数：** 无

**返回值：** 无

**其他影响：** 如果在调用此函数前未调用SPIS_SaveConfig()，那么以下寄存器将使用Configure对话框中的默认值：

SPIS_STATUS_MASK_REG
SPIS_COUNTER_PERIOD_REG

# 定义

### SPIS_TX_INIT_INTERRUPTS_MASK

定义了 Configure 对话框中所选中断源的初始配置。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。有关位字段的详细信息，请参考状态寄存器位一节中的内容。

### SPIS_RX_INIT_INTERRUPTS_MASK

定义了 Configure 对话框中所选中断源的初始配置。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。有关位字段的详细信息，请参考状态寄存器位一节中的内容。

# 状态寄存器位

### SPIS_TXSTATUS

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

### SPIS_RXSTATUS

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

- 字节/字传输已完成：当字节/字的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（如果此阵列存在）时，将设置该位。
- 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）。

- Rx FIFO 为空：当 Rx 数据 FIFO 为空时，将设置该位（不表示 Rx 缓冲区 RAM 阵列条件）。
- Rx FIFO 非空：当 Rx 数据 FIFO 非为空时，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）。
- 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）。
- Tx FIFO 未满：当 Tx 数据 FIFO 未满时，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。）

### SPIS_TXBUFFERSIZE

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。如果此值大于 4，会生成中断，并数据将自动从循环存储器缓冲区移动到 FIFO 中。

### SPIS_RXBUFFERSIZE

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。如果此值大于 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。

### SPIS_DATAWIDTH

定义了 Configure 对话框中所选的每个数据传输的位数。

## Bootloader 支持

SPI 从设备组件可以作为 Bootloader 的通信器件。使用以下配置可支持从外部系统到 Bootloader 的通信协议：

- **Mode**（模式）：必须与主机（引导器件）数据速率保持一致。
- **Data Lines**（数据线）：MOSI + MISO
- **Data bits**（数据位数）：8
- **Shift Direction**（移位方向）：必须与主机（引导器件）数据速率一致。
- **Bit Rate**（比特率）：必须与主机（引导器件）数据速率一致。
- **RX Buffer Size**（Rx 缓冲区大小）：64
- **TX Buffer Size**（Tx 缓冲区大小）：64

更多有关 Bootloader 的信息，请查阅*《系统参考指南》*中“Bootloader 系统”一节的内容。

SPI 从设备组件为 Bootloader 提供了一组 API 函数。

| 函数 | 说明 |
|---|---|
| SPIS_CyBtldrCommStart | 使能SPIS组件以及它的中断。 |
| SPIS_CyBtldrCommStop | 禁用SPIS组件及它的中断。 |
| SPIS_CyBtldrCommReset | 复位接收和传输通信缓冲区。 |
| SPIS_CyBtldrCommRead | 允许调用程序读取Bootloader主机中的数据。该函数将处理轮询，以便从主机器件完整接收到数据块。 |
| SPIS_CyBtldrCommWrite | 允许调用程序将数据写入Bootloader主机中。通过SPIS通信组件，该函数可以使用一个阻塞写入函数编写数据。 |

### void SPIS_CyBtldrCommStart(void)

**说明：** 启动SPIS通信组件。

**参数：** 无

**返回值：** 无

**其他影响：** 此组件会自动使能全局中断。

### void SPIS_CyBtldrCommStop(void)

**说明：** 此函数将禁用SPIS组件以及它的中断。

**参数：** 无

**返回值：** 无

**其他影响：** 无

### void SPIS_CyBtldrCommReset(void)

**说明：** 复位接收和传输通信缓冲区。

**参数：** 无

**返回值：** 无

**其他影响：** 无

### cystatus SPIS_CyBtldrCommRead(uint8 pData[]、uint16 size、uint16 * count、uint8 timeOut)

**说明：** 此函数允许调用程序读取Bootloader主机中的数据。该函数将处理轮询，以便接收来自Bootloader主机的完整数据块。

**参数：** uint8 pData[]：指向要从Bootloader主机读取数据块的存储区的指针

uint16 size：需要读取的字节数

uint16 *count：指向用于写实际读取字节数的变量的指针

uint8 timeOut：等待的单位数（时间为10毫秒），之后会因超时而返回

**返回值：** cystatus：如果未遇到任何问题则返回CYRET_SUCCESS，或是返回对该问题描述最准确的值。更多有关信息，请参考*《系统参考指南》*中“返回代码”一节中的内容。

**其他影响：** 无

### cystatus SPIS_CyBtldrCommWrite(const uint8 pData[]、uint16 size、uint16 * count、uint8 timeOut)

**说明：** 允许调用程序将数据写入Bootloader主机中。通过SPIS通信组件，该函数可以使用一个阻塞写入函数编写数据。

**参数：** const uint8 pData[]：指向要写入Bootloader主机的数据块的指针

uint16 size：需要写入的字节数

uint16 *count：指向实际写入字节数的变量指针

uint8 timeOut：等待的单位数（时间为10毫秒），之后会因超时而返回

**返回值：** cystatus：如果未遇到任何问题则返回CYRET_SUCCESS，或是返回对该问题描述最准确的值。更多有关信息，请参考*《系统参考指南》*中“返回代码”一节。

**其他影响：** 无

# 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”章节中介绍项目偏差以及有关 MISRA 合规性验证环境的信息。

SPI 从设备组件具有以下特定偏差：

| 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 | 建议 | 函数应该优先于类似于函数的宏。 | 由于使用了函数宏以实现更高效的代码，所以出现了偏差。<br>组件使用带输入参数的宏：<br>SPIS_GET_STATUS_TX()<br>SPIS_GET_STATUS_RX() |

该组件具有以下的嵌入式组件：时钟和中断。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。

# 示例固件源代码

PSoC Creator 在“Find Example Project”（查找示例项目）对话框中提供了多种包括原理图和代码示例的示例工程。要查看特定组件实例，请打开“Component Catalog”中的对话框或原理图中的组件示例。要查看通用示例，请打开‘Start Page’或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参考《PSoC Creator 帮助》部分中主题为“查找示例项目”的内容。

# 功能说明

## 默认配置

SPIS 的默认配置是 8 位 SPIS，配置为（CPHA = 0, CPOL = 0）。

## 模式

模式将控制组件的状态位以及在数据传输过程中假定的组件信号值。显示了四个波形。假设传输五个数据字节（其中：四个字节在传输开始时写入 SPI 从设备的 Tx 缓冲区内，第五个字节在第一个字节加载到 A0 寄存器后丢弃）。圆圈中的数字表示以下事件（请参见以下各波形）：

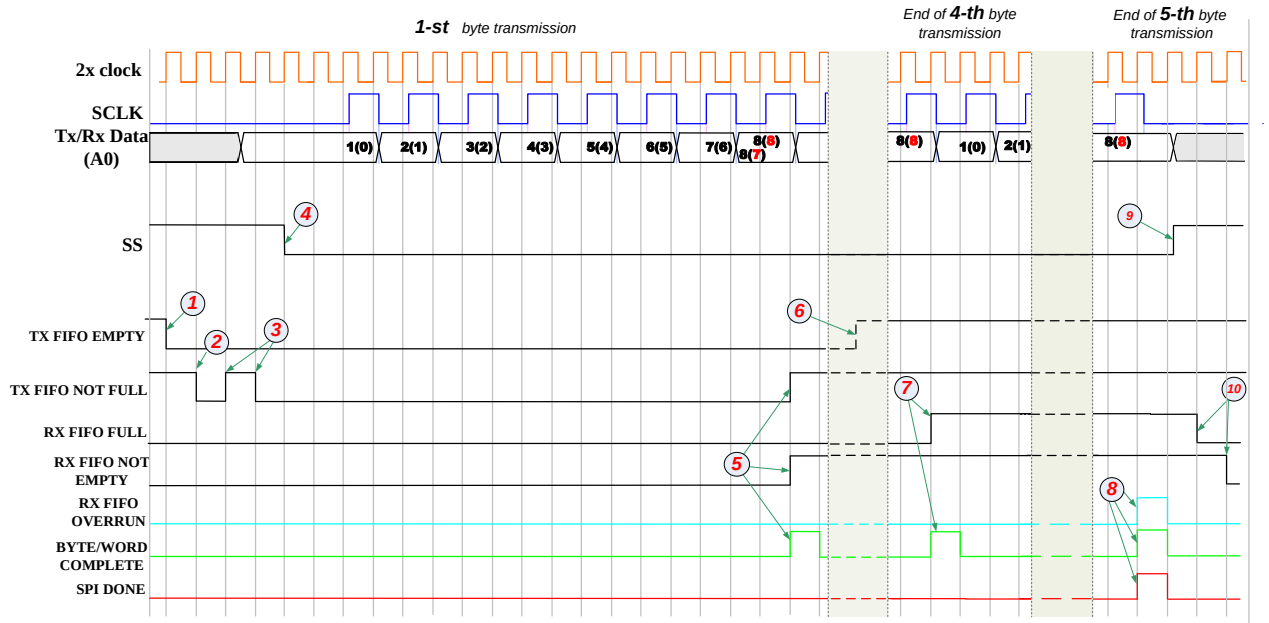
1. 当 4 个字节写入到 Tx 缓冲区内时，已清除了“Tx FIFO 为空”；
2. 由于写入 4 个字节后 Tx FIFO 已满，因此已清除了“Tx FIFO 未满”；
3. 当第一个字节已载入 A0 寄存器时，将设置“Tx FIFO 未满”状态，并且在第五个字节已写入 Tx 缓冲区中的可用空间时，会清除该状态。
4. “Slave Select”（从设备选择）线被设为低电平，以指示传输刚开始的状态。

5. 当第二个位载入 A0 寄存器时，将设置“Tx FIFO 未满”状态。所收到的第一个字节载入 Rx 缓冲区中时，则设置“Rx 非为空”状态。此外，还设置“字节/字传输已完成”状态。
6. 如果需要发送的最后字节已载入 A0 寄存器时，将设置“Tx FIFO 为空”状态。为了简单起见，这未在波形详细信息中显示。
7. 当已接收到第四个字节时，将同时设置“Rx FIFO 已满”和“字节/字传输已完成”状态。
8. 由于所有字节已被传输，并已检测到将数据载入已满的 Rx 缓冲区中的尝试，因此设置了“字节/字传输已完成”、“SPI 已完成”和“Rx 溢出”。
9. 将 SS 线设置为高电平，以指示传输已完成。
10. 当已从 Rx 缓冲区读取第一个字节时，将清除“Rx FIFO 已满”状态；如果已读了取所有字节，会设置“Rx FIFO 为空”状态。

**注意**：因为相同的寄存器用于发送和接收数据，“Tx/Rx 数据（A0）”图部分包含以下格式的两位数：“Tx 位数（Rx 位数）”。

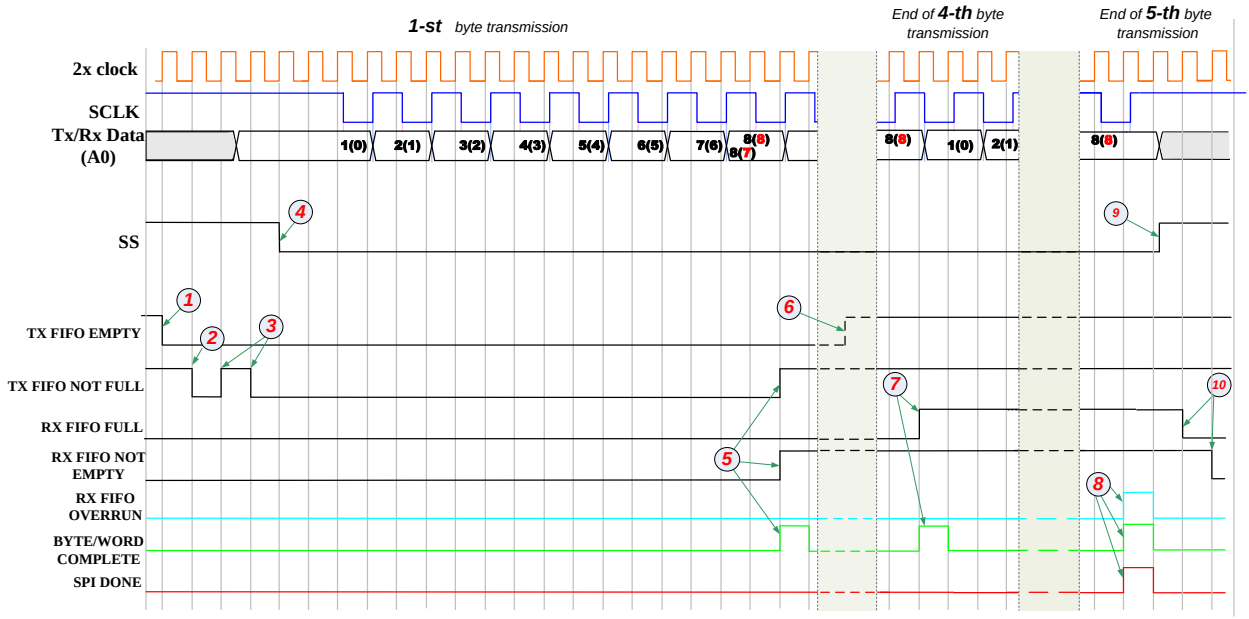
### SPIS 模式：（CPHA = 0，CPOL = 0）

该模式具备下列特性：



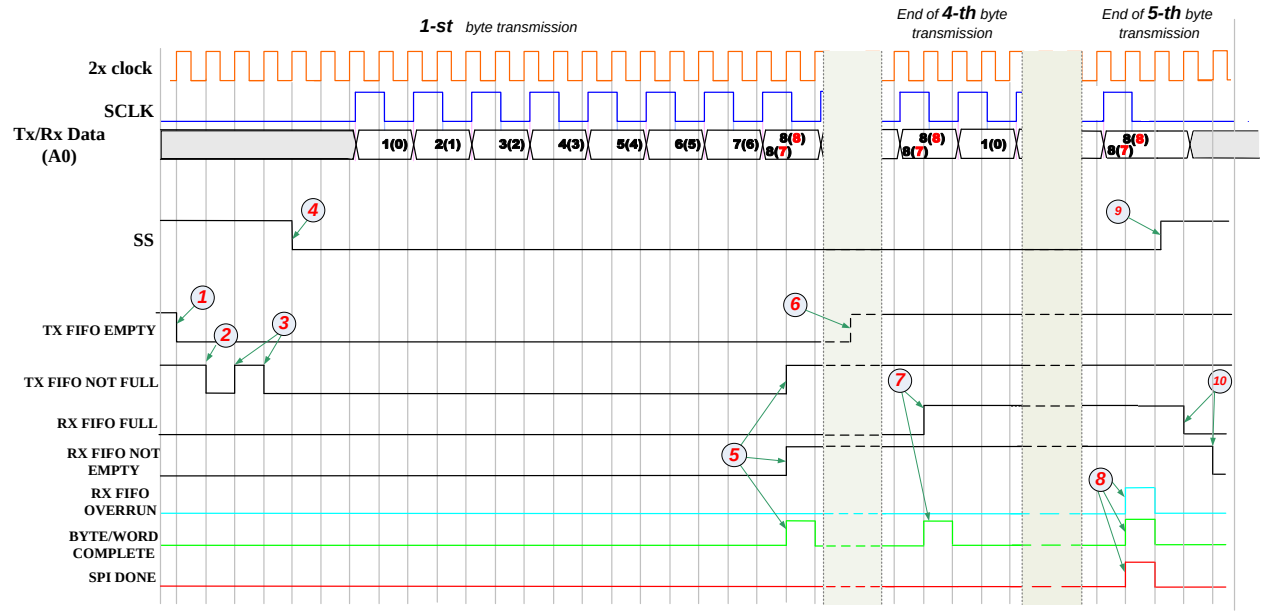
## SPIS 模式：（CPHA = 0，CPOL = 1）

该模式具有下列特性：



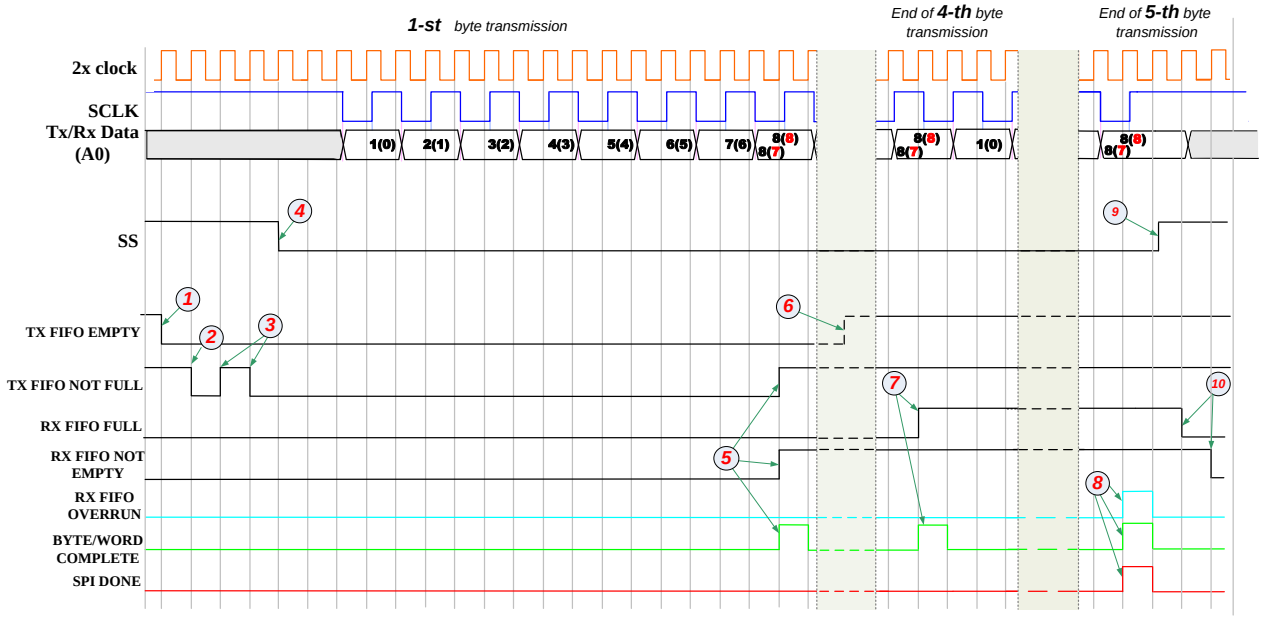

## SPIS 模式：（CPHA = 1，CPOL = 0）

该模式具有下列特性：



### SPIS 模式：（CPHA = 1，CPOL = 1）

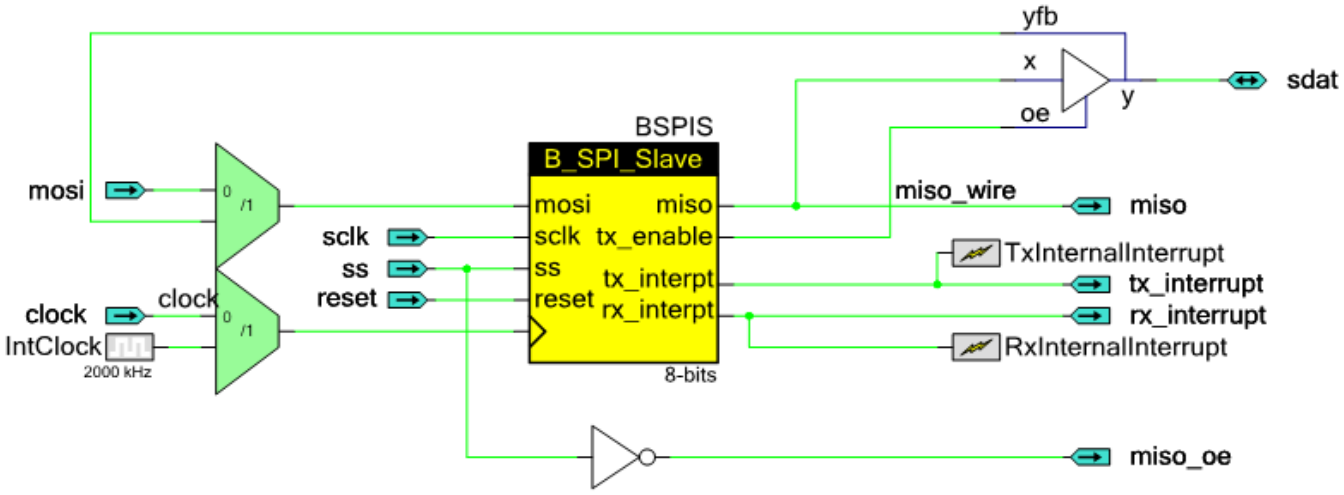
该模式具有下列特性：



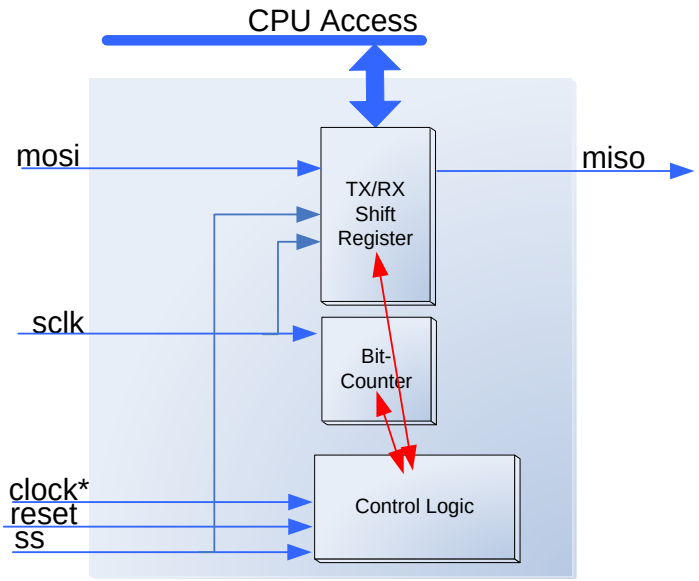

**注意**：某些 SPI 主设备（例如，TotalPhase Aardvark I2C/SPI 主机适配器）以特定方式驱动 sclk 输出。为了使 SPI 从设备组件与此类器件在 CPOL = 1 的模式下正常运行，应将 sclk 引脚设置为电阻上拉驱动模式。否则，它将输出损坏的数据。

# 框图和配置

SPIS 仅可作为模块的 UDB 配置使用。使用上述的 API 和此处所描述的寄存器定义 SPIS 的整体实现。



下面的框图中描述了该实现。




# 寄存器

## Tx 状态

Tx 状态寄存器是一个只读寄存器，它包含为 SPIS 组件给定实例定义的多种状态位。假设 SPI 从设备的实例名称为“SPIS”，这些寄存器的值可通过 SPIS_ReadTxStatus()函数调用。在 Tx 状态寄存器中，通过对各屏蔽位字段进行“或”运算，将生成中断输出信号。可以调用 SPIS_SetTxInterruptMode()函数设置掩码。接收中断时，可以通过调用 SPIS_ReadTxStatus()函数读取 Tx 状态寄存器来检索中断源。Tx 状态寄存器在读取后会被清除，所以会保留中断源，直至调用 SPIS_ReadTxStatus()函数为止。由于构建时位字段可能在 Tx 状态寄存器内移动，所以该寄存器上的所有操作必须使用以下的位字段定义。

Tx 状态寄存器有一些位字段掩码定义。这些位字段中的任何一个都可以作为中断源使用。标有星号（*）的位字段被设为 TX 状态寄存器中的粘滞位，则所有其他位被配为状态的实时指示符。

#定义位于已生成的头文件（.h）中，具体如下：

- SPIS_STS_SPI_DONE * — 定义为状态寄存器上位“SPI 已完成”的位掩码。
- SPIS_STS_TX_FIFO_NOT_FULL — 定义为状态寄存器上位“发送 FIFO 为空”的位掩码。
- SPIS_STS_TX_FIFO_EMPTY — 定义为状态寄存器上位“发送 FIFO 为空”的位掩码。
- SPIS_STS_BYTE_COMPLETE * — 定义为状态寄存器上位“字节传输已完成”的位掩码。

## Rx 状态

Rx 状态寄存器是一个只读寄存器，它包含为 SPIS 定义的多种状态位。通过调用 SPIS_ReadRxStatus()函数，可读取这些寄存器的值。Rx 状态寄存器内已屏蔽位字段的“或”运算将生成中断输出信号。通过调用 SPIS_SetRxInterruptMode()函数，可设置掩码。接收中断时，可以通过调用 SPIS_ReadRxStatus()函数读取 Rx 状态寄存器来检索中断源。Rx 状态寄存器在读取后将被清除，所以会保留中断源，直至调用 SPIS_ReadRxStatus()函数为止。由于构建时位字段可能在 Rx 状态寄存器内移动，所以该寄存器上的所有操作必须使用以下位字段定义。

Rx 状态寄存器有一些位字段掩码定义。这些位字段中的任何一个都可以作为中断源使用。标有星号（*）的位字段被设为 Rx 状态寄存器中的粘滞位，所有其他位则被配置为状态的实时指示符。

#定义位于已生成的头文件（.h）中，具体如下：

- SPIS_STS_RX_FIFO_FULL — 定义为状态寄存器上位“接收 FIFO 已满”的位掩码。
- SPIS_STS_RX_FIFO_NOT_EMPTY — 定义为状态寄存器上位“接收 FIFO 非空”的位掩码。
- SPIS_STS_RX_FIFO_OVERRUN * — 定义为状态寄存器上位“接收 FIFO 溢出”的位掩码。

## Tx 数据

Tx 数据寄存器包含需要发送的传输数据值。该寄存器在 SPIS 中作为 FIFO 实现。使用软件状态机可以控制来自传输存储器缓冲区的数据，以处理较大量需要发送的数据。所有的 API 处理发送数据必须通过该寄存器将数据放置在总线上。如果该寄存器中有数据，并且流量控制指示数据可发送，那么数据将在总线上发送。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。该 FIFO 为空时，可通过使用头文件中所定义的 TXDATA_REG 地址设置 DMA，以填充该 FIFO。

## Rx 数据

Rx 数据寄存器包含收到的数据。该寄存器在 SPIS 中作为 FIFO 实现。使用软件状态机控制从接收 FIFO 移至存储器缓冲区内的数据。通常情况下，Rx 中断将指示已收到数据，并在此时该数据具有连至固件的一些走线。可能设置 DMA，以将该寄存器上的数据传输到此存储器阵列；或固件可能只是随意轮询该寄存器的数据。这会使用头文件中所定义的 RXDATA_REG 地址。

## 有条件编译信息

SPIS 仅需要一个有条件编译定义，以便处理实现其必须支持的预期“数据位数”配置所必需的 8 或 16 位“数据路径”配置。API 必须有条件地编译在所选参数中定义的数据宽度。API 不应直接使用这些参数，但应使用下面所列出的定义。

- SPIS_DATAWIDTH — 定义了组成单个“字节”传输的数据位数。

# 资源

SPI 从设备组件被放置在整个 UDB 阵列中。该器件利用以下资源。

| 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 | 3 | 1 | – | 2 |
| 8位（双向） | 1 | 12 | 3 | 2 | – | 2 |
| 16位（MOSI+MISO） | 2 | 12 | 3 | 1 | – | 2 |
| 16位（双向） | 2 | 12 | 3 | 2 | – | 2 |

# 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情况的不同，组件所用的存储器使用情况也不一样。下表提供了给定组件配置中的所有 API 的存储器使用情况。

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 4 （GCC） | | PSoC 5LP（GCC） | |
|---|---|---|---|---|---|---|
| | 闪存 字节 | SRAM 字节 | 闪存 字节 | SRAM 字节 | 闪存 字节 | SRAM 字节 |
| 8位（MOSI+MISO） | 332 | 5 | 526 | 3 | 542 | 3 |
| 8位（双向） | 348 | 5 | 558 | 3 | 570 | 3 |
| 16位（MOSI+MISO） | 407 | 5 | 558 | 3 | 568 | 3 |
| 16位（双向） | 423 | 5 | 590 | 3 | 600 | 3 |

# 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 °C ≤ $T_A$ ≤ 85 °C且$T_J$ ≤ 100 °C。电压范围为 1.71 V到5.5 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位（MOSI+MISO） | 组件电流消耗 | | | | |
| | 闲置电流[3] | – | 18 | – | µA/MHz |
| | 工作电流[4] | – | 27 | – | µA/MHz |
| $I_{DD}$（8位（双向） | 组件电流消耗 | | | | |
| | 闲置电流[3] | – | 20 | – | µA/MHz |
| | 工作电流[4] | – | 32 | – | µA/MHz |
| $I_{DD}$（16位（MOSI+MISO） | 组件电流消耗 | | | | |
| | 闲置电流[3] | – | 32 | – | µA/MHz |
| | 工作电流[4] | – | 38 | – | µA/MHz |
| $I_{DD}$（16位（双向） | 组件电流消耗 | | | | |
| | 闲置电流[3] | – | 33 | – | µA/MHz |
| | 工作电流[4] | – | 40 | – | µA/MHz |

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

[3]。电流消耗与组件的输入时钟相关。

## 交流特性

| 参数 | 说明 | 配置 | 最小值 | 典型值 | 最大值 [4] | 单位 |
|---|---|---|---|---|---|---|
| $f_{SCLK}$ | SCLK频率 | 配置1 [5] | – | – | 5 | MHz |
| | | 配置2 [6] | – | – | 5 | MHz |
| | | 配置3 [7] | – | – | 4 | MHz |
| | | 配置4 [8] | – | – | 4 | MHz |
| $f_{CLOCK}$ | 组件时钟频率 [9] | 配置15 | 2 * $f_{SCLK}$ | – | 10 | MHz |
| | | 配置26 | 2 * $f_{SCLK}$ | – | 10 | MHz |
| | | 配置37 | 2 * $f_{SCLK}$ | – | 8 | MHz |
| | | 配置48 | 2 * $f_{SCLK}$ | – | 8 | MHz |
| $t_{CKH}$ | SCLK高电平时间 | | – | 0.5 | – | 1/$f_{SCLK}$ |
| $t_{CKL}$ | SCLK低电平时间 | | – | 0.5 | – | 1/$f_{SCLK}$ |
| $t_{SCLK_MISO}$ | SCLK到MISO输出时间 | | – | – | 52 | ns |
| $t_{SCLK_SDAT}$ （仅限于双向模式） | SCLK到SDAT输出时间 | | – | – | 54 | ns |
| $t_{S_MOSI}$ | MOSI输入设置时间 | | 25 | – | – | ns |
| $t_{H_MOSI}$ | MOSI输入保持时间 | | – | 0 | | ns |

[4]组件的最大器件时钟频率派生自 $t_{SCLK_MISO}$ 与 SCLK 输入和 MISO 输出的布线路径延迟的组合（本文中稍后将描述）。这些“Nominal”数字提供了额定走线条件下组件的最大安全工作频率。可以在更高的时钟频率运行组件，该频率将需要使用 STA 结果验证时序要求。

[5]配置 1 选项：
数据线： MOSI+MISO
数据位： 8

[6]配置 2 选项：
数据线： MOSI+MISO
数据位： 16

[7]配置 3 选项：
数据线： 双向
数据位： 8

[8]配置 4 选项：
数据线： 双向
数据位： 16

[9]组件时钟仅用于状态寄存器；它不会影响到基本功能或比特率。走线可能限制此参数的最大频率；因此使用额定走线结果会列出最大值。

| 参数 | 说明 | 配置 | 最小值 | 典型值 | 最大值 [4] | 单位 |
|---|---|---|---|---|---|---|
| $t_{SS_SCLK}$ | SS有效到SCLK有效的时间 | | 20 | – | – | ns |
| $t_{SCLK_SS}$ | SCLK无效到SS无效的时间 | | –20 | – | 20 | ns |

**图 6. 在 CPHA = 0 模式下的时序图**

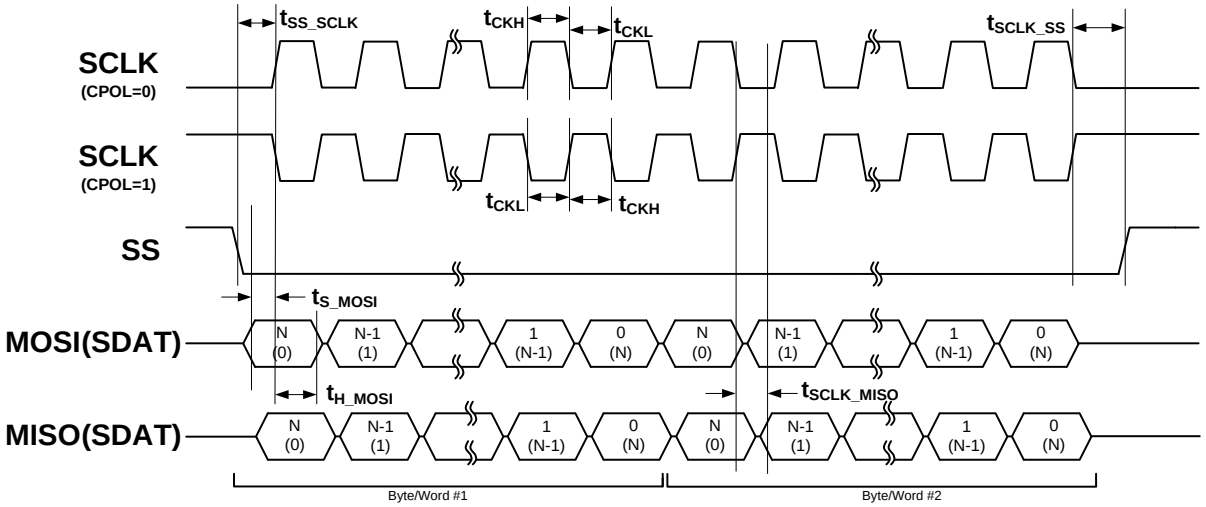
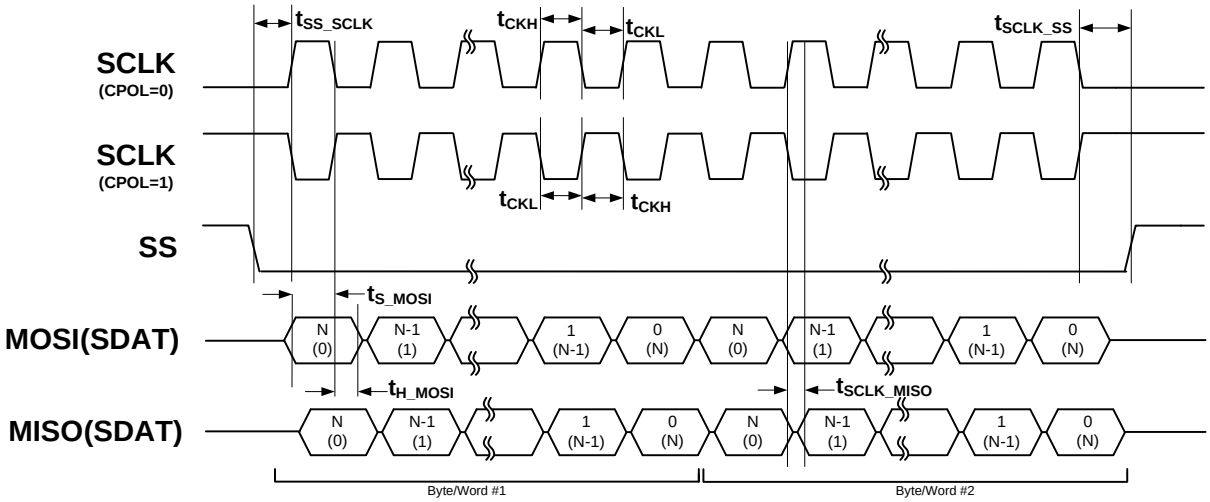



**图 7. 在 CPHA = 1 模式下的时序图**

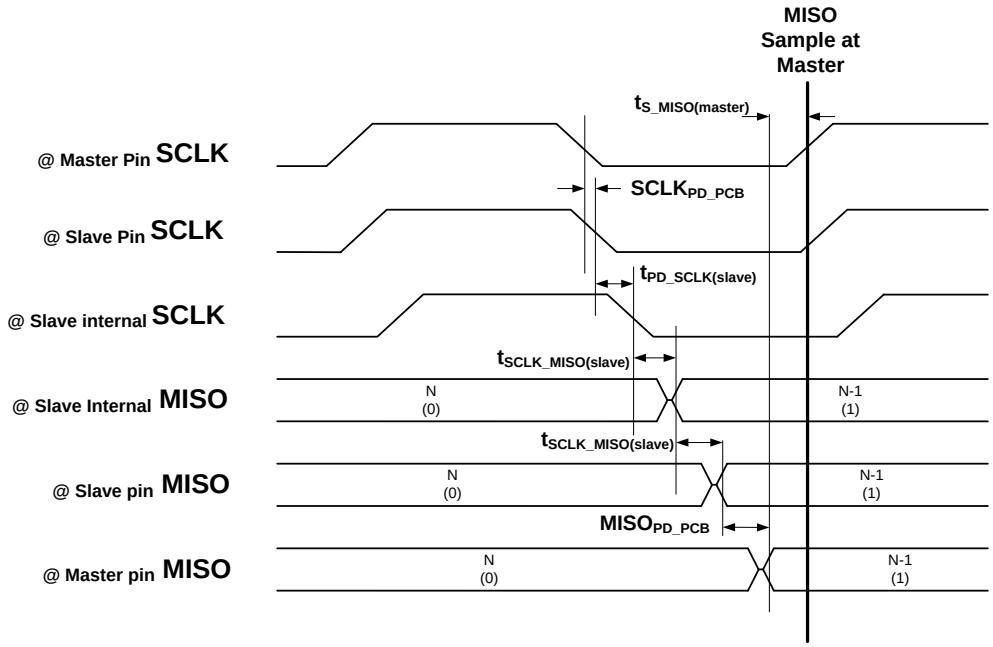



## 如何将 STA 结果用于特性数据

通过使用静态时序分析（STA）进行多次测试，从而收集各个额定走线最大值。可以同时使用 STA 结果与下列机制计算设计的最大值

**$f_{SCLK}$** 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。要想计算最大比特率，必须考虑一些因素。需要电板布局和从设备通信器件规范，才能彻底地了解最大值。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返路径延迟，以及从设备的 MISO 输出返回到主设备的路径延迟。

**图 8. 计算 $f_{SCLK}$ 最大频率**




在此情况下，组件必须使用下列公式满足主设备的 MISO 设置时间要求：

$$t_{RT_PD} < 1 \div \{ [½ \times f_{SCLK}] - t_{PD_SCLK(master)} - t_{S_MISO(master)} \}$$

或

$$f_{SCLK} < 1 \div \{2 \times [T_{RT_PD} + t_{PD_SCLK(master)} + t_{S_MISO(master)}] \}$$

其中：$t_{PD_SCLK(master)} + t_{S_MISO(Master)}$必须源于主设备数据手册。$t_{RT_PD}$ 如下定义：

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

并且：

$SCLK_{PD_PCB}$ 是从主设备组件引脚到从设备组件引脚的 SCLK 的 PCB 路径延迟。

$t_{PD_SCLK(Slave)}$为输入 SCLK 到内部逻辑的路径延迟；$t_{SCLK_MISO(slave)}$为从设备组件的 SCLK 引脚到内部逻辑的路径延迟；且 $t_{PD_MISO(slave)}$为内部 MISO 到引脚的路径延迟。寻找这三个参数的值最简单方法是从 STA 所列出的结果中直接获取组合路径，如下图所示：

- **Clock To Output Section**

  - **SCLK_1(0)_PAD**

| Source | Destination | Delay (ns) |
|---|---|---|
| \SPIS_1:BSPIS:es3:SPISlave:sR8:Dp:u0\/so_comb | MISO_1(0)_PAD | 47.895 |

其中：$t_{PD_SCLK(Slave)}$为前两个数值，$t_{SCLK_MISO(slave)}$为中间两个数值，且 $t_{PD_MISO(slave)}$为最后两个数值。三个参数的完整路径为 45.889 ns。

$MISO_{PD_PCB}$ 是从设备组件引脚到主设备组件引脚的 MISO 的 PCB 路径延迟 。

最后公式将提供 SCLK 的最大频率，因此最大比特率为：

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

**$f_{CLOCK}$** 组件的最大时钟频率作为内部时钟（如果选择了内部时钟）或外部时钟显示在时钟汇总的时序结果内。下面是 STA 记录文件中的内部时钟限制示例：

- **Clock Summary Section**

| Clock | Type | Nominal Frequency (MHz) | Required Frequency (MHz) | Maximum Frequency (MHz) | Violation |
|---|---|---|---|---|---|
| BUS_CLK | Sync | 24.000 | 24.000 | N/A | |
| ClockBlock/clk_bus | Async | 24.000 | 24.000 | N/A | |
| ClockBlock/dclk_0 | Async | 2.000 | 2.000 | N/A | |
| ILO | Async | 0.001 | 0.001 | N/A | |
| IMO | Async | 3.000 | 3.000 | N/A | |
| MASTER_CLK | Sync | 24.000 | 24.000 | N/A | |
| PLL_OUT | Async | 24.000 | 24.000 | N/A | |
| SCLK_1(0)_PAD | Async | UNKNOWN | UNKNOWN | 33.636 | |
| SPIS_1_IntClock | Sync | 2.000 | 2.000 | 75.746 | |

**$t_{CKH}$** SPI 从设备组件需要 50%占空比的 SCLK。

**$t_{CKL}$** SPI 从设备组件需要 50%占空比的 SCLK。

**$t_{S_MOSI}$** 为了满足内部逻辑的设置时间要求，SCLK 在引脚上有效前，MOSI 必须于此时间在引脚上有效。

**$t_{H_MOSI}$** 为了满足内部逻辑的保持时间要求，SCLK 在引脚上有效后，MOSI 必须于此时间在引脚上有效。

**$t_{SS_SCLK}$** 为了满足该模块的内部功能，在 SCLK 通过此参数在引脚上有效前，Slave Select（从设备选择）（SS）必须在该引脚上有效。

**$t_{SCLK_SS}$** 满足模块的内部功能的最大值。此参数表示在引脚上 SCLK 的最后一个下降沿到来后，引脚上的从设备选择（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 | v2.60 | 该组件被修改，但没有更改PSoC Creator 3.0 SP1中的版本编号。更多有关信息，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

本节列出了该组件各版本中的主要更改内容。

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| 2.60.a | 编辑了数据手册，以添加组件勘误章节。 | 请注意，虽然组件被更改，但设计不受任何影响。 |
| 2.60 | 向组件添加了MISRA-C 2004规范支持。更新了数据手册中的相应章节内容。 | |
| 2.50 | 禁止了PSoC 4的固定放置选项。更新了PSoC 4的数据手册以及存储器占用情况。 | |
| 2.40 | 已添加了MISRA合规性章节。 | 此组件未进行MISRA合规性验证。 |
| | 集成了各个特定API，以支持Bootloader：CyBtldrCommStart、CyBtldrCommStop、CyBtldrCommReset、CyBtldrCommWrite、CyBtldrCommRead。 | SPI从设备可以作为具有此功能的Bootloader的通信组件使用。 |
| | 更改了tSS_SCLK时序参数值。最小值被设为20 ns。最大值因不适用而被移除。 | 先前的时序值不正确。 |
| 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。<br>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 | |
| 2.20.a | 向数据手册添加了固定放置说明 | |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| 2.20 | 如果在传输过程中SS引脚变为高电平，则SPI从设备现在会丢弃已收到的以及已传输的任何部分字。只在ES3芯片上才会出现缺陷。 | 修复了仿真模型缺陷。 |
| | 向定制器添加了使能固定放置选项 | |
| 2.10 | 已将数据位数范围从2到16位更改为3到16位 | 当前版本中修复了与状态同步相关的更改问题 |
| | 已将“Byte transfer complete”（字节传输完成）复选框名称更改为“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中的子文件夹内。 | |
| | 对数据手册进行了少量编辑和更新 | |
| 2.0 | 已添加了SPIS_Sleep()/SPIS_Wakeup()以及SPIS_Init()/SPIS_Enable() API。 | 为了支持低功耗模式并提供常用接口，以单独控制大多数器件的初始化和使能。 |
| | 器件输出数和位置已改变：<br>已添加了复位输入；<br>已删除中断输出；而添加了rx_interrupt、tx_interrupt输出。 | 已添加了Production PSoC 3的复位功能。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。必须考虑这些更改，以避免从先前版本迁移时出现绑定错误 |
| | 已移除了SPIS_EnableInt()、SPIS_DisableInt()、SPIS_SetInterruptMode()以及SPIS_ReadStatus()等API。<br>已添加了SPIS_EnableTxInt()、SPIS_EnableRxInt()、SPIS_DisableTxInt()、SPIS_DisableRxInt()、SPIS_SetTxInterruptMode()、SPIS_SetRxInterruptMode()、SPIS_ReadTxStatus()、SPIS_ReadRxStatus()等API。 | 被移除的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。此外，还为TX和Rx缓冲区更新了中断处理程序实现。 |
| | 已分别将SPIS_ReadByte()、SPIS_WriteByte()和SPIS_WriteByteZero() API重命名为SPIS_ReadRxData()、SPIS_WriteTxData()、SPIS_WriteTxDataZero()。 | 阐明了各API以及使用它们的方式。 |
| 对通过使用仿真模型实现的从设备组件B_SPI_Slave_v2_0做出了以下更改： | | |
| | B_SPI_Slave_v2_0现在包含对ES2和ES3芯片的两种单独的实现。<br>在ES3芯片中，Tx和Rx的8位SPI使用一个数据路径，而 ES2芯片中则使用两个数据路径。 | 要求所有组件支持ES2和ES3芯片。要求使用ES3特性更新，这样有助于优化ES3中资源的使用情况。 |
| | ES2支持实现中的更改： | 这样会提供正确的软件缓冲区功能。 |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| | 现在ES2中提供了两个状态寄存器（即单独的Tx和Rx状态寄存器），而非两者使用一个共同的状态寄存器。 | |
| | 已将‘BidirectMode’ 布尔参数添加到基本组件（仿真模型实现）。<br>目前已选择了含‘时钟’输入和SYNC模式位的控制寄存器，以驱动ES3芯片的‘tx_enable’输出。<br>选择ES2芯片时，不带时钟输入的控制寄存器将驱动‘tx_enable’。<br>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。<br>将“yfb”连接到“miso”输入。则Bufoe“y”输出连接到“sdat”输出终端。 | 添加了针对组件的双向模式支持 |
| | 四个udb_clock_enable组件已添加到含sync =“TRUE”参数的仿真模型实现。其中一个含sync =“TRUE”（状态寄存器时钟），其余三个含sync =“FALSE” | 新增了对用于指示功能的仿真模型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。 |
| | 更改了Rx数据路径配置。将‘FIFO FAST’选项设为“DP”而不是“BUS” | 修复了组件先前版本中的缺陷，即存在影响正确器件数据捕获的时序窗口。 |
| | 将逻辑添加到Verilog实现，以改变了生成“SPI DONE”和“BYTE COMPLETE”状态的时间。 | 修复了组件先前版本中的缺陷，即有时甚至在通信完成后也不生成“SPI DONE”。 |
| | 已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引用 |
| | 已改变了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，2013-2016 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（无再许可权）（1）在赛普拉斯特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，WICED，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。