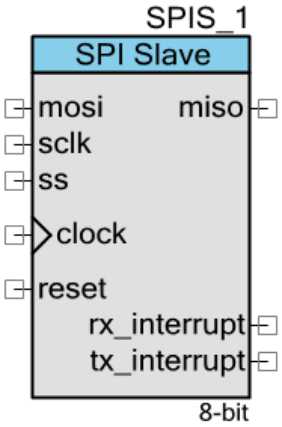
# 串行外设接口 (SPI) 从设备

2.50

## 特性

- 3 至 16 位数据宽度
- 四种 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。它还提供了一个 3 线（双向）SPI 接口。这两种接口都支持全部四种 SPI 操作模式，允许与任何 SPI 主设备通信。除了标准的 8 位字长之外，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 PSoC 组件需要与 SPI 主设备连接，便可以使用 SPI 从设备组件。除了标有“SPI 主设备”的组件外，可以将 SPI 从设备与许多组件配合使用，从而实现移位寄存器类型接口。

在需要 PSoC 组件与 SPI 从设备通信时，应使用 SPI 主设备组件。移位寄存器组件应在其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才使用。

## 输入/输出连接

本节介绍 SPI 的各种输入和输出连接。I/O 列表中的星号 (*) 表示，在 I/O 说明中列出的情况下，该 I/O 可能不可见。

[1] 此值仅对 MOSI+MISO（全双工）接口模式（请参见直流和交流电气特性一节以获取详细信息）有效，在双向模式下最高限制为 1 Mbps（由于内部双向管脚限制）。

## mosi — 输入 *

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，此输入可见。如果可见，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**（数据线）参数设置为 **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，会在组件时钟与 SCLK 信号之间报告异步时钟交叉警告。下面是这类消息的示例：“Path(s) exist between clocks IntClock and SCLK(0)_PAD,but the clocks are not synchronous to each other.（时钟 IntClock 与 SCLK(0)_PAD 之间存在路径，但时钟彼此不同步。）” 该消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，SCLK 不应运行。只要遵循此规则，便不会有问题，并且您可以忽略此消息。

**注意**双向模式提供内部环回功能，因此在两种模式的任一模式下，另一方向均仍活动（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）**

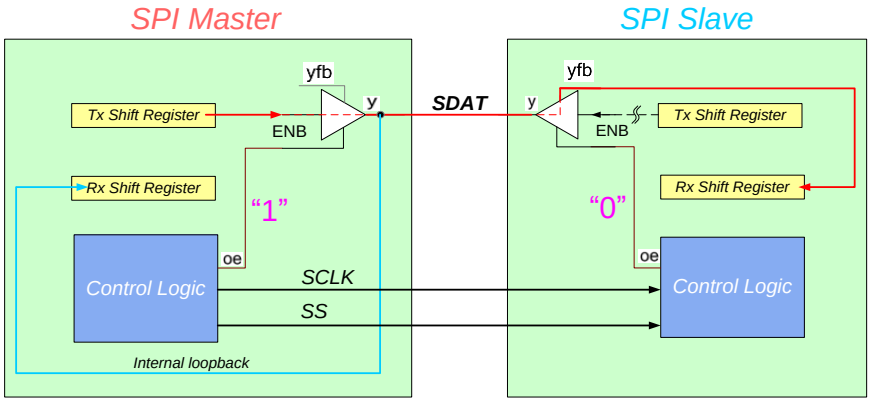
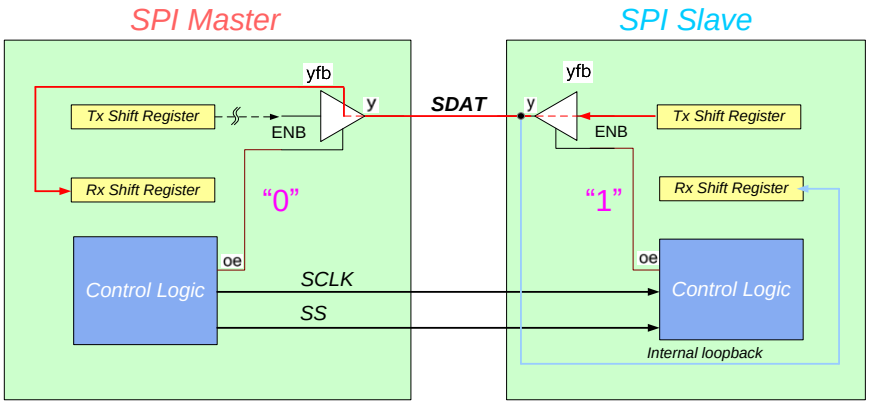



**图 2. SPI 双向模式（数据由从设备传输到主设备）**



双向模式下组件初始状态为 Rx 模式、或从主设备到从设备进行数据传输，如 图 2 所示。应使用 API 函数 SPIS_TxEnable() 和 SPIS_Tx_Disable() 在 Rx 和 Tx 模式间切换。

## sclk — 输入

sclk 输入承载串行时钟 (SCLK) 信号。它向该组件提供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。

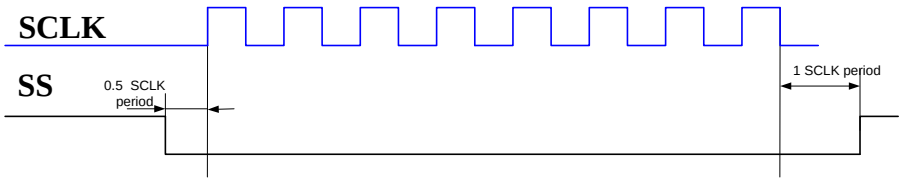
**注：**某些 SPI 主设备（例如，TotalPhase Aardvark I2C/SPI 主机适配器）以特定方式驱动 sclk 输出。为使 SPI 从设备组件与此类组件在 CPOL = 1 的模式下正常运行，sclk 管脚应设置为电阻上拉驱动模式。否则，它将输出损坏的数据。有关模式的更多信息，请参考功能描述一节。

## ss — 输入

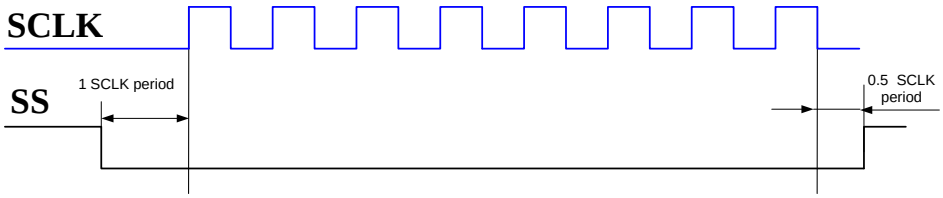
ss 输入承载传输至此组件的从设备选择 (SS) 信号。该输入始终可见并必须处于连接状态。

下图显示了 SCLK 和 SS 信号之间的时序相关性

CPHA = 0：




CPHA = 1：




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

## 复位 — 输入

复位输入会复位 SPI 从设备。它会删除当前正在传输或接收的任何数据，但是不会清除 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，因此当用于这些组件时，会忽略此输入。使用复位输入会导致在执行时序分析时，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。下面是这类消息的示例：“Path(s) exist between clocks BUS_CLKand SCLK(0)_PAD, but the clocks are not synchronous to each other.（时钟 BUS_CLK 与

SCLK(0)_PAD 之间存在路径，但时钟彼此不同步。）” 该消息适用于从 Reset 信号到由 SCLK 计时的 SPI 组件操作之间的路径。在更改复位信号时，SCLK 不应运行。只要遵循此规则，便不会有问题，并且您可以忽略此消息。

复位输入可以不连接外部信号，置于悬空状态。如果复位线上无任何连接，组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。所有的数据时钟都出现在 sclk 输入上，因此，时钟输入**不会**处理 SPI 从设备的比特率。

当 **Clock Selection**（时钟选择）参数设置为 **External**（外部）时，时钟输入可见。如果可见，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。当 **Data Lines**（数据线）参数设置为 **MOSI + MISO** 时，此输出可见。

### 中断 — 输出

中断输出是所有可能的中断源的逻辑或（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和输出管脚和时钟源。原理图宏可用于 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**

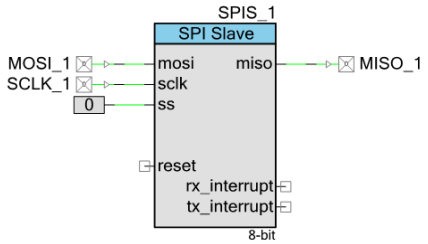


**图 4. 3 线（全双工）连接原理图宏**

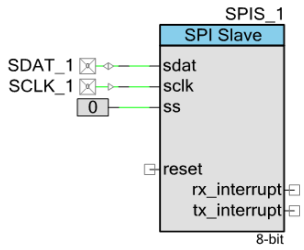
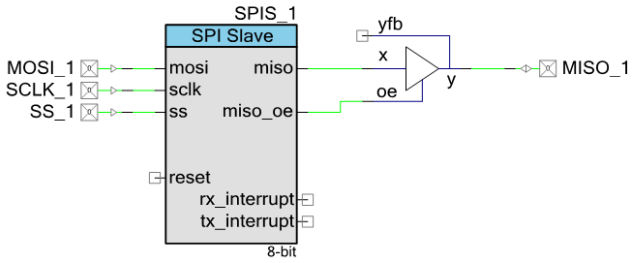



**图 5. 多从模式原理图宏**




**注：**如果不使用原理图宏，则需要配置管脚组件，取消选中每个分配的输入管脚（MOSI、SCLK 和 SS）的 **Input Synchronized**（输入同步）参数。该参数位于相应“Pins Configure”（管脚配置）（对话框的 **Pins（管脚） > Input（输入）**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**（配置）对话框。

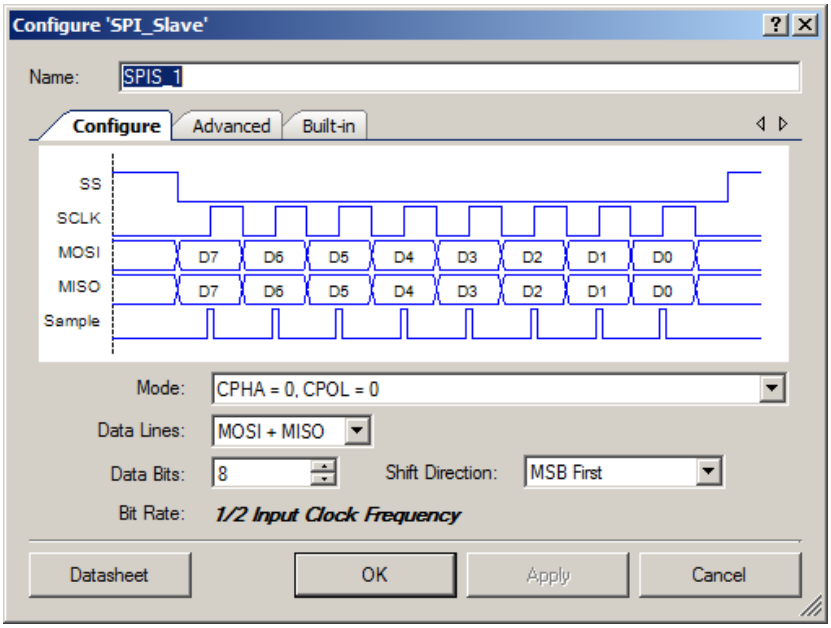
以下各节将介绍 SPI 从设备参数，以及如何使用“Configure（配置）”对话框对其进行配置。它们还指明了选项是硬件还是软件。

## 硬件和软件选项

硬件配置选项用于更改项目合成和放置在硬件中的方式。如果您对任何这些选项进行了更改，则必须重新构建硬件。软件配置选项不影响合成或放置。如果在构建之前设置这些参数，则需要设置其初始值，这些初始值随时可能随提供的 API 修改。仅硬件参数标有星号 (*)。

## “配置”选项卡

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




**注意**：波形中的示例信号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选择的模式设置进行数据采样。

### 模式 *

**Mode**（模式）参数用于定义通信中需要使用的时钟相位和时钟极性模式。下表中定义了以下模式。另请参阅此数据手册的 功能描述（功能描述）一节。

| CPHA | CPOL |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

### 数据线

**Data Lines（数据线）**参数定义哪个接口用于 SPI 通信 — 4 线 (**MOSI + MISO**) 或 3 线（**双向**）。

### 数据位 *

**数据位**的数量定义使用 SPIS_ReadRxData() 和 SPIS_WriteTxData() API 传输时，单个传输的位宽度。默认位数为单字节（8 位）。3 到 16 之间的任何整数都是有效值。

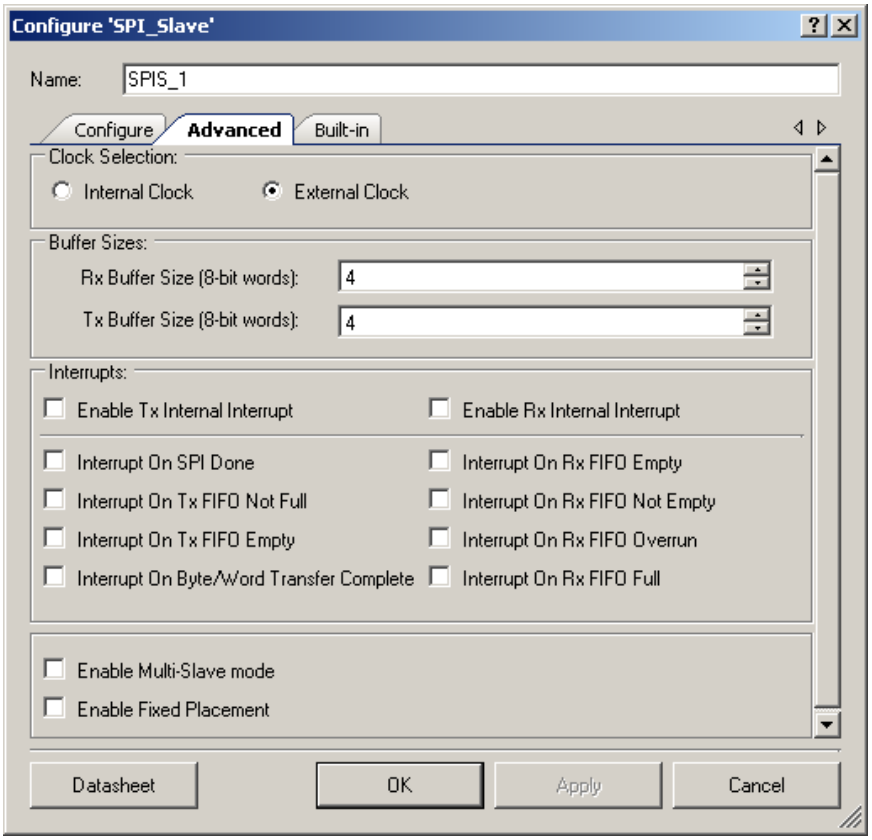
### 移位方向 *

**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，首先传输最高有效位。这是通过将数据左移实现。设置为 **LSB First**（SB 优先）时，首先传输最低有效位。这是通过将数据右移实现。

### 比特率 *

如果 **Clock Selection**（时钟选择）参数（位于 **Advanced**（高级）选项卡上）设置为 **内部时钟**，**比特率**参数定义 SCLK 速度（以赫兹为单位）。内部时钟的时钟频率将为 SCLK 速率的 2 倍。如果 **Clock Selection**（时钟选择）参数设置为 **External Clock**（外部时钟），则此参数没有作用。

## “Advanced（高级）”选项卡




### 时钟选择

**Clock Selection**（时钟选择）参数指定是使用内部时钟还是外部时钟。有关更多信息，请参考本数据手册中后面的时钟选择一节。

### Rx Buffer Size（Rx 缓冲区大小）*

**Rx Buffer Size** （Rx 缓冲区大小）参数定义为循环数据缓冲区分配的存储器大小（以字节/字为单位）。如果此参数设置为 1 到 4，则在硬件中实现 FIFO 的第 4 字节/字。值 1 到 3 仅用于与以前版本兼容；使用它们会导致出现一个指示该值不正确的错误消息。所有其他最大长度为 255 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（Tx 缓冲区大小）**参数定义为循环数据缓冲区分配的存储器大小（以字节/字为单位）。如果此参数设置为 1 到 4，则在硬件中实现 FIFO 的第 4 字节/字。值 1 到 3 仅用于与以前版本兼容；设置它们会导致出现一个指示该值不正确的错误消息。所有其他最大长度为 255 的值将使用 4 字节/字 FIFO 和提供的 API 所控制的存储器阵列。

### 使用 Software Buffer

选大于 4 的 Rx/Tx Buffer 大小的值，即可使用 Rx/Tx 软件循环缓冲区。当您选择 Tx/Rx 软件缓冲区选项时，使用的是内部中断处理程序。其主要目的是提供软硬件 Tx/Rx 缓冲区之间的交互。在初始状态，BufferRead 和 BufferWrite 指针指向软件缓冲区的零元素。在写第一个数据后，BufferWrite 指针移至软件缓冲区的第一个元素，并指向写数据；BufferRead 指针留在零元素。当缓冲区工作时，指针移动到下一个缓冲区元素。BufferWrite 指针指向最后写入的数据。BufferRead 指针指向还尚未被读取的最早的数据。软件缓冲区溢出可在无任何溢出指示的情况下发生。用户必须处理所有软件缓冲区溢出情形。

用户还应考虑到：使用软件缓冲区导致被传输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（依据所选总线时钟的值而定）。在设定被传输字间时序间隔时，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）**选项允许您使用 SPI 从设备组件的预先定义 Tx 和 Rx ISR，或者提供您自己的自定义 ISR。如果使能这些选项，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组件。

如果 Rx 或 Tx 缓冲区大小大于 4，则组件自动设置相应参数，因为需要内部 ISR 来处理从硬件 FIFO 到 Rx 和/或 Tx 缓冲区的传输。SPI 从设备的中断输出管脚始终可见且可用，输出的信号与传输到内部中断的信号相同。然后此输出可以用作 DMA 请求源，或者用作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 字节/字时，“Rx FIFO NOT EMPTY”中断始终处于使能状态，不能禁用，否则会导致错误的缓冲区功能。

- 当 Tx 缓冲区大小大于 4 字节/字时，“Tx FIFO NOT FULL”中断始终处于使能状态，不能禁用，否则会导致错误的缓冲区功能。
- 对于大于 4 字节/字的缓冲区大小，必须使能全局中断和组件中断，才能进行正确的缓冲区处理。

### 中断

**Interrupts**（中断）选择参数允许您配置已使能的导致中断的内部事件。中断生成是所有处于使能的 Tx 和 Rx 状态寄存器位的掩码 OR 运算结果。使用这些参数选择的位用于定义使用初始组件配置实现的掩码。

### 使能多从模式

该设置在当前 SPI 从设备组件通过其他 SPI 从设备连接至共享总线时使用。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 输出转变为高阻抗状态。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

与不受约束的放置相比，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。连接到组件的管脚的放置不受控制，但是最好使用 P[0] 的管脚来实现最佳性能。

组件的固定放置方面消除了对于“所有路由的最大值”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）。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

当选择内部时钟配置时，PSoC Creator 计算所需的频率和时钟源，并生成实现所需的时钟设置资源。否则，您必须提供时钟组件并计算所需的时钟频率。该频率至少是最大比特率和 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 可以使用当前系统时钟频率提供此值。否则，将在生成项目时显示一条有关时钟精度范围的警告。此警告将包含 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系统要求和达到最佳值。

## 放置

SPI 从设备组件放置于 UDB 阵列中，并且所有放置信息通过 *cyfitter.h* 文件提供给 API。

# 应用程序编程接口

应用程序编程接口 (API) 子程序允许您使用软件配置组件。下表列出了每个函数的接口，并进行了说明。以下各节将更详细地介绍每个函数。

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。实例名称会成为每个全局函数名称、变量和常量符号的前缀。为增加可读性，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() | 将 Tx 中断源配置为使能。 |
| SPIS_SetRxInterruptMode() | 将 Rx 中断源配置为使能。 |
| 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 的 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 存储器缓冲区中收到的数据的大小（以字节/字为单位）。 |
| SPIS_GetTxBufferSize() | 返回 Tx 存储器缓冲区中等待传输的数据的大小（以字节/字为单位）。 |
| SPIS_ClearRxBuffer() | 清除所有收到的数据的 Rx 缓冲区存储器阵列和 Rx FIFO。 |
| SPIS_ClearTxBuffer() | 清除所有传输数据的 Tx 缓冲区存储器阵列或 Tx FIFO。**注意**只有未使用软件缓冲区时，才会清除 Tx FIFO。 |
| SPIS_TxEnable() | 如果针对双向模式进行配置，则设置用于传输的 SDAT 输入输出。 |
| SPIS_TxDisable() | 如果针对双向模式进行配置，则设置用于接收的 SDAT 输入输出。 |
| SPIS_PutArray() | 将数据数组放入传输缓冲区。 |
| SPIS_ClearFIFO() | 清除所有数据的 Rx 和 Tx FIFO 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() | 在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 从设备组件是否已初始化。该变量初始化为 0，并在第一次调用 SPIS_Start() 时设置为 1。这样，第一次调用 SPIS_Start() 子程序后，组件不用重新初始化即可重启。<br>如需重新初始化组件，可在 SPIS_Start() 或 SPIS_Enable() 函数前调用 SPIS_Init() 函数。 |
| SPIS_txBufferWrite | API 上次将数据写入缓冲区的传输缓冲区位置。 |
| SPIS_txBufferRead | 发送从缓冲区读取并由 SPIS 硬件发送的最后数据的缓冲位置。 |
| SPIS_rxBufferWrite | 接收由 SPIS 硬件接收后写入缓冲区的最后数据的缓冲位置。 |
| SPIS_rxBufferRead | API 上次从缓冲区读取数据的接收缓冲区位置。 |
| SPIS_rxBufferFull | 指示软件缓冲区已溢出。 |
| SPIS_RxBUFFER[] | 用于存储收到的数据。 |
| SPIS_TxBUFFER[] | 用于存储要发送的数据。 |

## void SPIS_Start(void)

**说明：** 这是开始执行组件操作的首选方法。SPIS_Start() 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

**副作用：** 无

## void SPIS_Stop(void)

**说明：** 禁用 SPI 从设备组件中断。对 SPIS 操作没有影响。

**参数：** 无

**返回值：** 无

**副作用：** 无

## void SPIS_EnableTxInt(void)

**说明：** 使能内部 Tx 中断请求。

**参数：** 无

**返回值：** 无

**副作用：** 无

## void SPIS_EnableRxInt(void)

**说明：** 使能内部 Rx 中断请求。

**参数：** 无

**返回值：** 无

**副作用：** 无

## void SPIS_DisableTxInt(void)

**说明：** 禁用内部 Tx 中断请求

**参数：** 无

**返回值：** 无

**副作用：** 无

## void SPIS_DisableRxInt(void)

**说明：** 禁用内部 Rx 中断请求

**参数：** 无

**返回值：** 无

**副作用：** 无

## void SPIS_SetTxInterruptMode(uint8 intSrc)

**说明：** 配置已使能的 Tx 中断源。

**参数：** uint8 intSrc：包含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/字完成而使能中断 |

基于 Tx 状态寄存器的位字段排列。此值必须是头文件中定义的 Tx 状态寄存器位掩码的组合。

有关更多信息，请参考此数据手册中的定义一节。

**返回值：** 无

**副作用：** 无

## void SPIS_SetRxInterruptMode(uint8 intSrc)

**说明：** 配置已使能的 Rx 中断源。

**参数：** uint8 intSrc：包含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 非空而使能中断 |
| SPIS_STS_Rx_FIFO_OVERRUN | 由于 Rx 缓冲区溢出而使能中断 |
| 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。此值必须是头文件中定义的 Rx 状态寄存器位掩码的组合。

有关更多信息，请参考此数据手册中的定义一节。

**返回值：** 无

**副作用：** 无

## uint8 SPIS_ReadTxStatus(void)

**说明：** 返回 Tx 状态寄存器的当前状态。有关更多信息，请参见此数据手册中的状态寄存器位一节。

**参数：** 无

**返回值：** uint 8：当前的 Tx 状态寄存器值

| 位 | 说明 |
|---|---|
| SPIS_STS_SPI_DONE | SPI 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/字完成 |

**副作用：** Tx 状态寄存器位在读取后清除。

## uint8 SPIS_ReadRxStatus(void)

**说明：** 返回 Rx 状态寄存器的当前状态。有关更多信息，请参见此数据手册中的状态寄存器位一节。

**参数：** 无

**返回值：** uint 8：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 非空 |
| SPIS_STS_Rx_FIFO_OVERRUN | Rx 缓冲区溢出 |
| SPIS_STS_Rx_FIFO_FULL | Rx FIFO 已满 |

**副作用：** Rx 状态寄存器位在读取后清除。

## void SPIS_WriteTxData(uint8/uint16 txData)

**说明：** 在发送缓冲区中放置一个字节，该字节将会在下一个可用总线时间发送。

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

**返回值：** 无

**副作用：** 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。此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/字直接放置到移位寄存器中以进行传输。此字节/字将在下一个时钟相位过程中从主设备发送。

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

**返回值：** 无

**副作用：** 这是 CPHA == 0 的模式的必要条件，这些模式下，数据必须位于第一个时钟沿之前的移位寄存器中。如果已有数据移出，并且如果 FIFO 中有更多数据，则固件必须控制该子程序。该子程序不应用于 CPHA == 1 的模式。

## uint8/uint16 SPIS_ReadRxData(void)

**说明：** 读取通过 SPI 接收的数据的下一字节。

**参数：** 无

**返回值：** uint8/uint16：从 FIFO 中读取的数据的下一字节/字

**副作用：** 如果 FIFO 为空，将返回无效数据。调用 SPIS_GetRxBufferSize()，并且如果其返回一个非零值，则调用 SPIS_ReadRxData() 函数是安全的。

## uint8 SPIS_GetRxBufferSize(void)

**说明：** 返回 Rx 缓冲区中当前保存的收到的数据的字节/字数。

- 如果禁用 Rx 软件缓冲区，则此函数返回 0（FIFO 为空）或 1（FIFO 不为空）。
- 如果使能 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：** uint 8：Rx 缓冲区中字节/字数的整数计数。

**副作用：** 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要传输的数据字节/字数。

- 如果禁用 Tx 软件缓冲区，则此函数返回 0（FIFO 为空）、1（FIFO 未满）或 4（FIFO 已满）。
- 如果使能 Tx 软件缓冲区，则此函数返回 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：** uint 8：Tx 缓冲区中字节/字数的整数计数

**副作用：** 清除组件的 Tx 状态寄存器。

## void SPIS_ClearRxBuffer(void)

**说明：** 清除所有收到的数据的 Rx 缓冲区存储器阵列和 Rx 硬件 FIFO。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要读取的数据。这样，在地址 0 恢复写入，可能覆盖留在 RAM 中的所有数据。

**参数：** 无

**返回值：** 无

**副作用：** 当新数据覆盖时，任何未从 RAM 缓冲区和 FIFO 中读取的已收到数据将丢失。

## void SPIS_ClearTxBuffer(void)

**说明：** 清除等待传输的 Tx 缓冲区存储器数据数组。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要传输的数据。这样，在地址 0 恢复写入，可能覆盖留在 RAM 中的所有数据。

**参数：** 无

**返回值：** 无

**副作用：** 如果使用软件缓冲区，它不清除已放入 Tx FIFO 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丢失。

## void SPIS_TxEnable(void)

**说明：** 如果 SPI 从设备配置为使用单个双向管脚，那么该函数会将双向管脚设置为发送。

**参数：** 无

**返回值：** 无

**副作用：** 无

## void SPIS_TxDisable(void)

**说明：** 如果 SPI 从设备配置为使用单个双向管脚，那么该函数会将双向管脚设置为接收。

**参数：** 无

**返回值：** 无

**副作用：** 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。坚持尝试，直至所有数据传递至 Tx 缓冲区。如果使用 CPHA = 0 的模式，则在调用 SPIS_PutArray() 函数之前调用 SPIS_WriteTxDataZero() 函数。

**参数：** uint8/uint16 *buffer：指向 RAM 中包含要发送的数据的位置的指针

uint8 byteCount：移至发送缓冲区的字节数/字数

**返回值：** 无

**副作用：** 系统将保留在此函数位置，直到所有数据已传输到缓冲区。如果 Tx 缓冲区中没有足够空间，则此函数保持封锁。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。

## void SPIS_ClearFIFO(void)

**说明：** 清除所有数据的 RX 和 TX FIFO 重新开始。

**参数：** 无

**返回值：** 无

**副作用：** 清除组件的状态寄存器。

## void SPIS_Sleep(void)

**说明：** 这是准备组件进入低功耗模式的首选子程序。SPIS_Sleep() 子程序保存当前组件的状态。然后它调用 SPIS_Stop() 函数并调用 SPIS_SaveConfig() 以保存硬件配置。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 SPIS_Sleep() 函数。有关电源管理函数的更多信息，请参考 PSoC Creator *System Reference Guide*（《系统参考指南》）。

**参数：** 无

**返回值：** 无

**副作用：** 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。如果组件在调用 SPIS_Sleep() 函数前已使能，则 SPIS_Wakeup() 函数也将重新使能组件。清除 Rx 缓冲区、Tx 缓冲区和硬件 FIFO 的所有数据。

**参数：** 无

**返回值：** 无

**副作用：** 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（配置）”对话框设置来初始化或恢复组件。不需要调用 SPIS_Init()，因为 SPIS_Start() 子程式会调用此函数，这是开始组件操作的首选方法。

**参数：** 无

**返回值：** 无

**副作用：** 当调用此函数时，它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，则必须在调用此 API 之前单独启动外部时钟。应在使能 SPIS 中断之前调用 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

**副作用：** 无

## void SPIS_SaveConfig(void)

**说明：** 此函数会保存组件配置和非保留寄存器。它还保存 Configure（配置）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

**副作用：** 无

## void SPIS_RestoreConfig(void)

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

**参数：** 无

**返回值：** 无

**副作用：** 如果在调用此函数前未调用 SPIS_SaveConfig()，则以下寄存器将使用“配置”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“Configure（配置）”对话框中所选中断源的初始配置。这是 Tx 状态寄存器中的在配置时作为中断源使能的位掩码。有关位字段的详细信息，请参考状态寄存器位一节。

### SPIS_RX_INIT_INTERRUPTS_MASK

定义“配置”对话框中所选中断源的初始配置。这是 Rx 状态寄存器中的在配置时作为中断源使能的位掩码。有关位字段的详细信息，请参考 状态寄存器位（状态寄存器位）一节。

# 状态寄存器位

### SPIS_TXSTATUS

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 中断 | 字节/字完成 | 未使用 | 未使用 | 未使用 | Tx FIFO 为空 | Tx FIFO 未满 | SPI 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/字完成：当字节/字发送已完成时设置。
- Rx FIFO 溢出：当 Rx 数据在未移到 Rx 缓冲区存储器阵列（如果此阵列存在）的情况下已溢出 4 字节/字 FIFO 时设置
- Rx FIFO 已满：当 Rx 数据 FIFO 已满时设置（不表示 Rx 缓冲区 RAM 阵列条件）。

- Rx FIFO 为空：当 Rx 数据 FIFO 为空时设置（不表示 Rx 缓冲区 RAM 阵列条件）。
- Rx FIFO 不为空：当 Rx 数据 FIFO 不为空时设置。即，至少一个字节/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 为空时设置（不表示 Tx 缓冲区 RAM 阵列条件）。
- Tx FIFO 未满：当 Tx 数据 FIFO 未满时设置（不表示 Tx 缓冲区 RAM 阵列条件）。
- SPI 完成：当已发送传输 FIFO 中的所有数据时设置。无需使用字节/字完成状态，可以使用该参数表示传输完成。（当已设置“字节/字完成”并且 Tx 数据 FIFO 为空时设置。）

### SPIS_TXBUFFERSIZE

定义要为 Tx 存储器阵列缓冲区分配的存储器数。这不包括 FIFO 中包含的 4 字节/字。如果此值大于 4，数据自动从循环存储器缓冲区移动到 FIFO 的中断将会出现。

### SPIS_RXBUFFERSIZE

定义要为 Rx 存储器阵列缓冲区分配的存储器数。这不包括 FIFO 中包含的 4 字节/字。如果此值大于 4，数据自动从 FIFO 移动到循环存储器缓冲区的中断将会出现。

### SPIS_DATAWIDTH

定义“Configure（配置）”对话框中选择的每个数据传输的位数。

## 引导加载程序支持

SPI 从设备组件可用作引导加载程序的通信组件。使用以下配置可为外部系统到引导加载程序的通信协议提供支持：

- **模式**：必须与主机（引导设备）数据速率一致。
- **数据线**：MOSI + MISO
- **数据位**：8
- **移位方向**：必须与主机（引导设备）数据速率一致。
- **比特率**：必须与主机（引导设备）数据速率一致。
- **RX 缓冲区大小**：64
- **TX 缓冲区大小**：64

有关引导加载程序的更多信息，请参考 *System Reference Guide*（《系统参考指南》）中的 “Bootloader System”（引导加载程序系统）一节。

SPI 从设备组件为使用引导加载程序提供了一组 API 函数。

| 函数 | 说明 |
|---|---|
| SPIS_CyBtldrCommStart | 使能 SPIS 组件，并使能其中断。 |
| SPIS_CyBtldrCommStop | 禁用 SPIS 组件并禁用其中断。 |
| SPIS_CyBtldrCommReset | 复位接收和传输通信缓冲区。 |
| SPIS_CyBtldrCommRead | 允许调用程序读取引导加载程序主机中的数据。该函数将处理轮询以便从主机组件完全接收到数据块。 |
| SPIS_CyBtldrCommWrite | 允许调用程序将数据写入引导加载程序主机中。此函数使用阻塞写入功能写入使用 SPIS 通信组件的数据。 |

### void SPIS_CyBtldrCommStart(void)

**说明：** 启动 SPIS 通信组件。

**参数：** 无

**返回值：** 无

**副作用：** 此组件自动使能全局中断。

### void SPIS_CyBtldrCommStop(void)

**说明：** 此函数禁用 SPIS 组件并禁用其中断。

**参数：** 无

**返回值：** 无

**副作用：** 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

**副作用：** 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引导加载程序主机中的数据。该函数将处理轮询以便从引导加载程序主机完全接收到数据块。

**参数：** uint8 pData[]: 指向要从引导加载程序主机读取的数据模块的存储的指针

uint16 size: 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将返回 CYRET_SUCCESS，或返回可详尽描述该问题的值。有关更多信息，请参考 *System Reference Guide*（《系统参考指南》）中的“Return Codes”（返回代码）一节。

**副作用：** 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引导加载程序主机中。此函数使用阻塞写入功能写入使用 SPIS 通信组件的数据。

**参数：** const uint8 pData[]: 指向要写入引导加载程序主机的数据模块的指针

uint16 size: 要写入的字节数

uint16 *count: 用于写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将返回 CYRET_SUCCESS，或返回可详尽描述该问题的值。有关更多信息，请参考 *System Reference Guide*（《系统参考指南》）中的“Return Codes”（返回代码）一节。

**副作用：** 无

# MISRA 合规性

本节介绍了本组件与 MISRA-C:2004 的合规和偏差情况。定义了两种类型的偏差：

- 项目偏差 - 适用于所有 PSoC Creator 组件的偏差
- 特定偏差 - 仅适用于此组件的偏差

本节提供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。

还未证明 SPI 从设备组件源代码符合 MISRA-C:2004 编码准则。

# 固件源代码示例

PSoC Creator 在“Find Example Project（查找示例项目）”对话框中提供了很多包括原理图和代码示例的示例项目。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开 Start Page（开始页）或 **File（文件）**菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（筛选选项）可缩小可选项目的列表。

有关更多信息，请参见 PSoC Creator 帮助中的“Find Example Project（查找示例项目）”主题。

# 功能描述

## 默认配置

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

## 模式

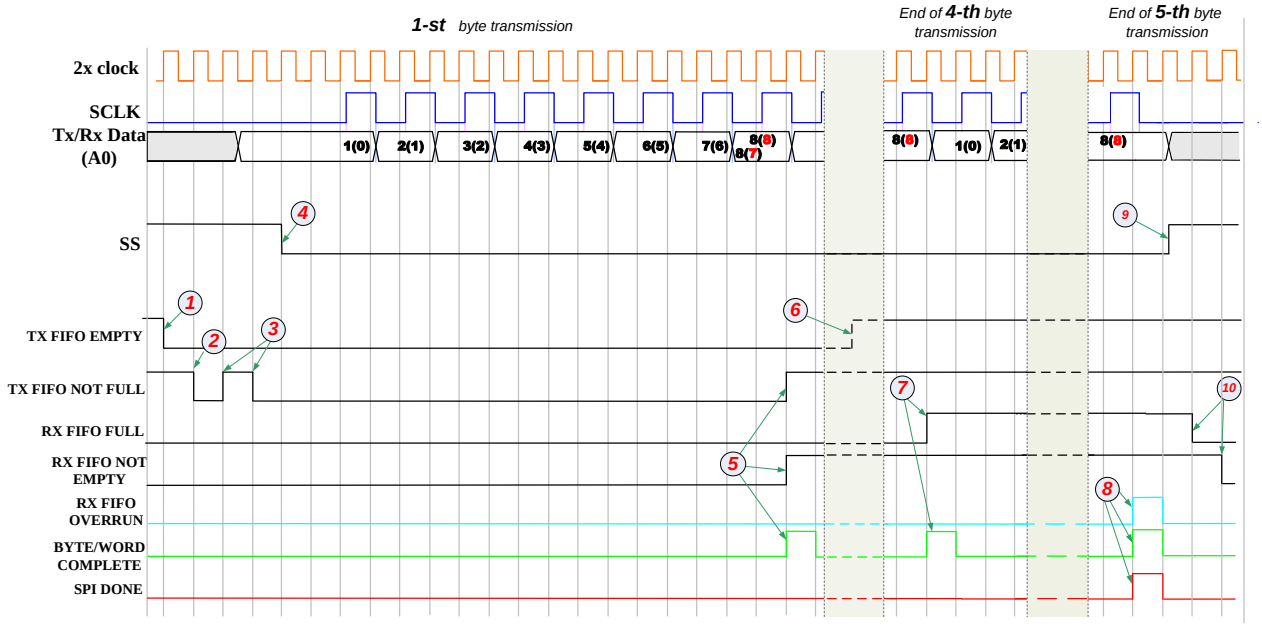
模式控制组件的状态位以及在数据传输过程中呈现的组件信号值。显示了四个波形。假设传输五个数据字节（四个字节在传输开始时写入 SPI 从设备的 Tx 缓冲区，第五个字节在第一个字节加载到 A0 寄存器之后抛出）。圆圈中的数字表示以下事件（请参见以下波形）：

1. 当 4 字节写入 Tx 缓冲区时已清除“Tx FIFO 为空”；
2. 由于写入 4 字节后 Tx FIFO 已满，已清除“Tx FIFO 未满”；
3. 当第一个字节已载入 A0 寄存器时，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，清除该状态。
4. “从设备选择”线设置为指示传输开始的低状态。
5. 当第二个位加载到 A0 寄存器时，设置“Tx FIFO 未满”状态。当收到的第 1 个字节加载到 Rx 缓冲区中时，设置“Rx 不为空”状态。还将设置“字节/字完成”。
6. 当要发送的最后字节已载入 A0 寄存器时，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. 当已接收第四个字节时，同时设置“Rx FIFO 已满”和“字节/字完成”。
8. 已设置“字节/字完成”、“SPI 完成”和“Rx 溢出”，因为所有字节已传输并且已检测到将数据载入满 Rx 缓冲区的尝试。
9. SS 线设置为高，以指示传输完成。
10. 当已从 Rx 缓冲区读取第一个字节时，清除“Rx FIFO 已满”；当已读取所有字节时，设置“Rx FIFO 为空”。

**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）。”

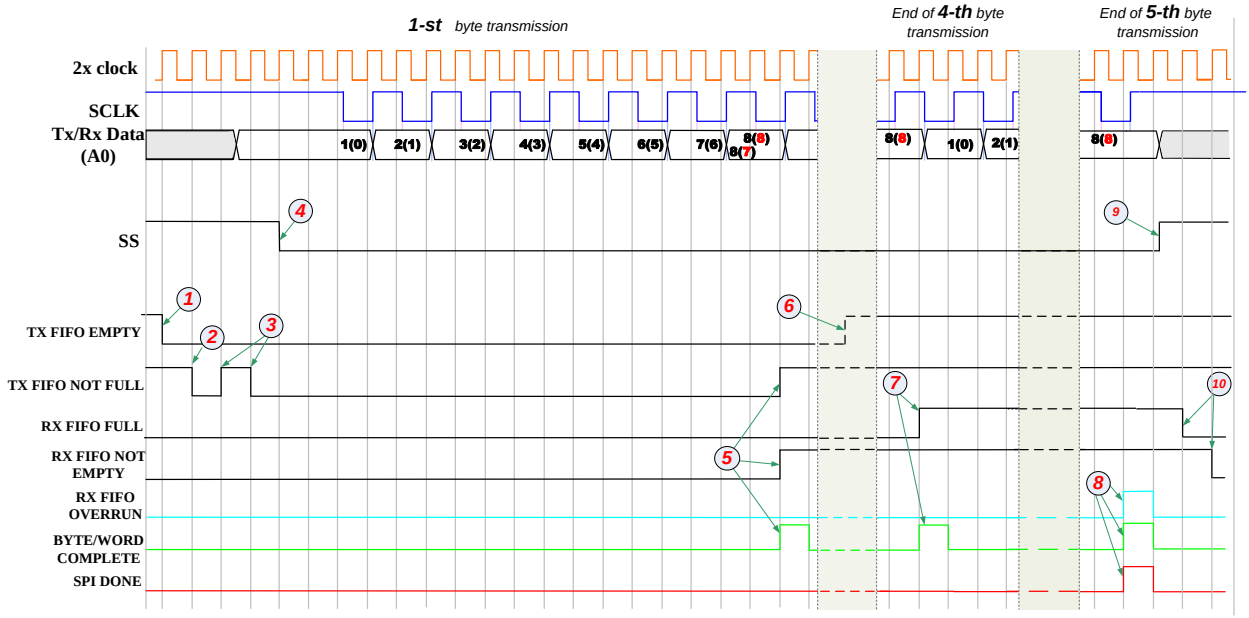
### SPIS 模式：(CPHA == 0, CPOL == 0)

模式 0 具有下列特性：



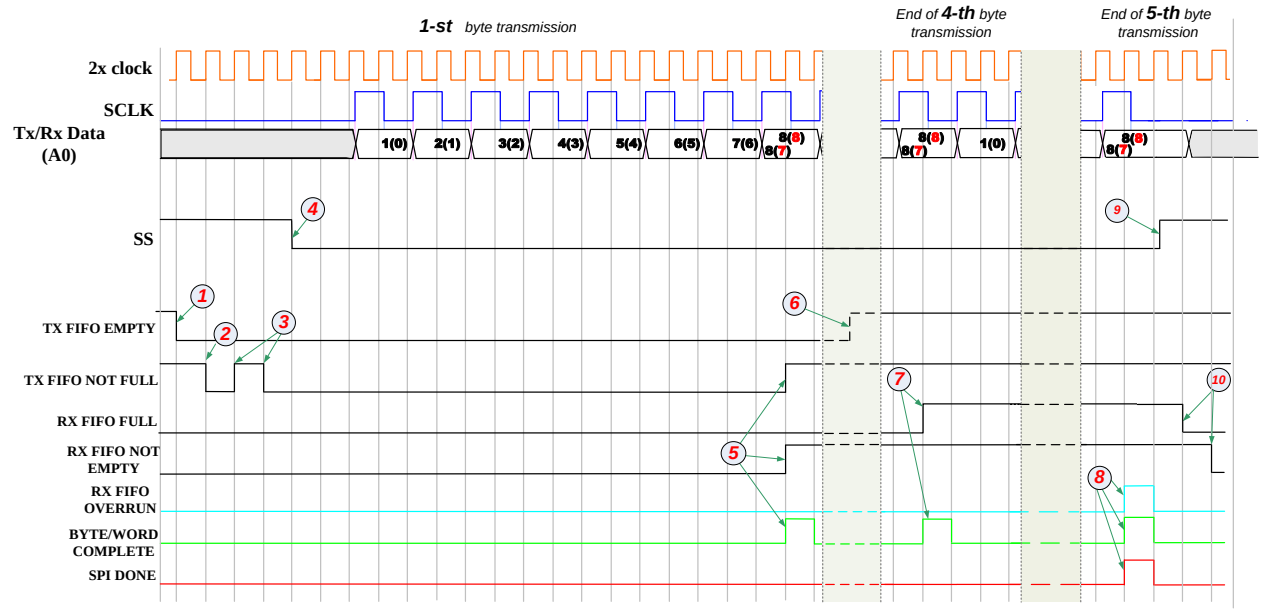
## SPIS 模式：(CPHA == 0, CPOL == 1)

模式 1 具有下列特性：



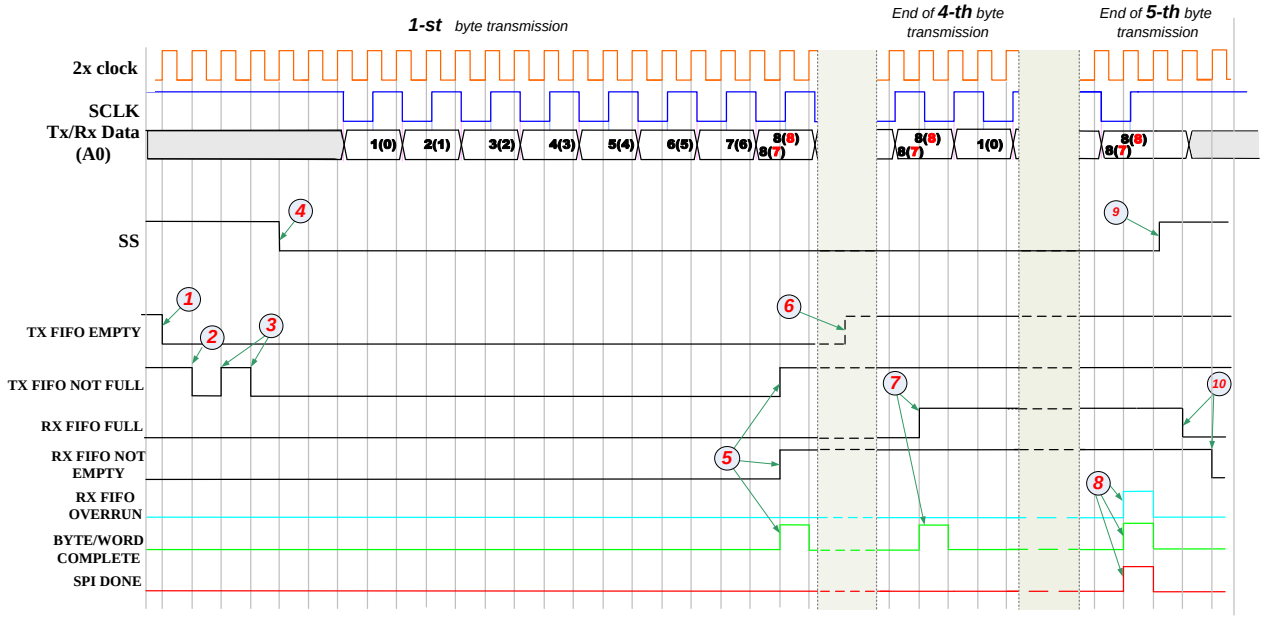

## SPIS 模式：(CPHA == 1, CPOL == 0)

模式 2 具有下列特性：



### SPIS 模式：(CPHA == 1, CPOL == 1)

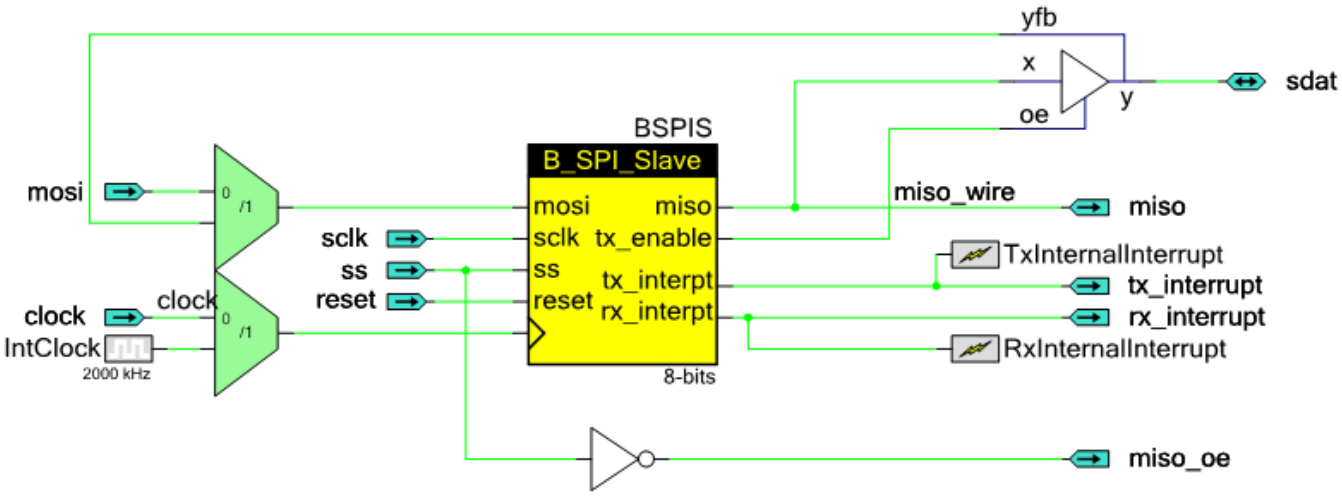
模式 3 具有下列特性：



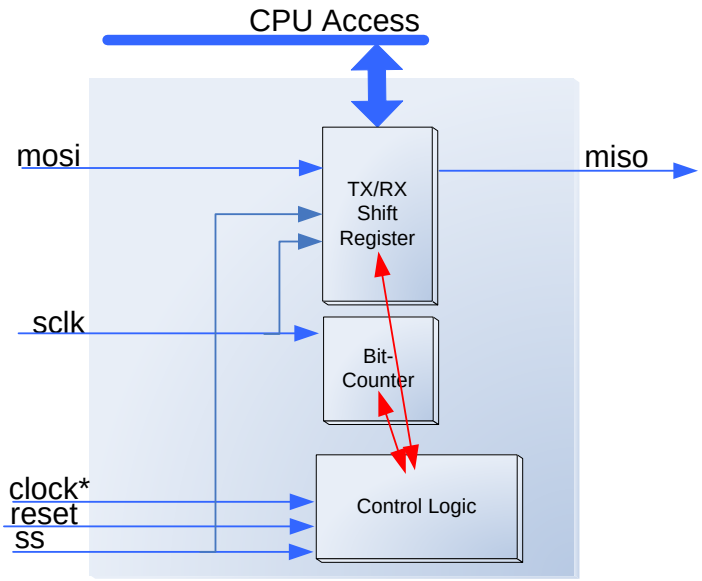

**注：**某些 SPI 主设备（例如，TotalPhase Aardvark I2C/SPI 主机适配器）以特定方式驱动 sclk 输出。为使 SPI 从设备组件与此类组件在 CPOL = 1 的模式下正常运行，sclk 管脚应设置为电阻上拉驱动模式。否则，它将输出损坏的数据。

# 框图和配置

SPIS 仅可用作模块的 UDB 配置。上面所描述的 API 和此处所描述的寄存器用于定义 SPIS 的整体实现。



下面的框图中描述了实现。




# 寄存器

## Tx 状态

Tx 状态寄存器为只读寄存器，其包含为 SPIS 组件给定实例定义的多种状态位。假设 SPI 从设备的实例名为“SPIS”，这些寄存器的值可从 SPIS_ReadTxStatus() 函数调用。中断输出信号由 Tx 状态寄存器内已屏蔽位字段的 ORing 生成。可以使用 SPIS_SetTxInterruptMode() 函数调用设置掩码。接收中断时，可以通过使用 SPIS_ReadTxStatus() 函数调用读取 Tx 状态寄存器来检索中断源。Tx 状态寄存器在读取后清除，所以会保留中断源，直至调用 SPIS_ReadTxStatus() 函数。由于在构建时位字段可能在 Tx 状态寄存器内移动，所以 Tx 状态寄存器上的所有操作必须为位字段使用以下定义。

Tx 状态寄存器中定义了一些位字段掩码。这些位字段中的任何一个可能作为中断源包括在内。用星号 (*) 指示的位字段配置为 TX 状态寄存器中的粘连位，所有其他位配置为状态的实时指示符。

#定义在如下已生成的头文件 (.h) 中可用：

- SPIS_STS_SPI_DONE * — 定义为状态寄存器位“SPI 完成”的位掩码。
- SPIS_STS_TX_FIFO_NOT_FULL — 定义为状态寄存器位“发送 FIFO 为空”的位掩码。
- SPIS_STS_TX_FIFO_EMPTY — 定义为状态寄存器位“发送 FIFO 为空”的位掩码。
- SPIS_STS_BYTE_COMPLETE * — 定义为状态寄存器位“字节完成”的位掩码。

## Rx 状态

Rx 状态寄存器为只读寄存器，其包含为 SPIS 定义的多种状态位。这些寄存器的值可通过 SPIS_ReadRxStatus() 函数调用。中断输出信号由 Rx 状态寄存器内已屏蔽位字段的 ORing 生成。可以使用 SPIS_SetRxInterruptMode() 函数调用设置掩码。接收中断时，可以通过使用 SPIS_ReadRxStatus() 函数调用读取 Rx 状态寄存器来检索中断源。Rx 状态寄存器在读取后清除，所以会保留中断源，直至调用 SPIS_ReadRxStatus() 函数。由于在构建时位字段可能在 Rx 状态寄存器内移动，所以 Rx 状态寄存器上的所有操作必须为位字段使用以下定义。

Rx 状态寄存器中定义了一些位字段掩码。这些位字段中的任何一个可能作为中断源包括在内。用星号 (*) 指示的位字段配置为 Rx 状态寄存器中的粘连位，所有其他位配置为状态的实时指示符。

#定义在如下已生成的头文件 (.h) 中可用：

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

## Tx 数据

Tx 数据寄存器包含要发送的传输数据值。这在 SPIS 中作为 FIFO 实现。软件状态机可以控制来自发送存储器缓冲区的数据以处理较大量要发送的数据。为了将数据放置在总线上，所有处理数据发送的 API 必须通过该寄存器。如果该寄存器中有数据并且流量控制指示数据可发送，那么数据将会在总线上发送。该寄存器 (FIFO) 一旦为空，总线上就不再发送数据，直至将其添加到 FIFO。DMA 可以设置为在此 FIFO 为空时使用头文件中定义的 TXDATA_REG 地址填充此 FIFO。

## Rx 数据

Rx 数据寄存器包含收到的数据。这在 SPIS 中作为 FIFO 实现。软件状态机器控制数据从接收 FIFO 移至存储器缓冲区。通常，Rx 中断将指示数据具有固件的若干路由时已收到数据。DMA 可能从该寄存器建立至存储器阵列，或者固件可能仅随意轮询数据。这会使用头文件中定义的 RXDATA_REG 地址。

## 有条件编译信息

SPIS 仅需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，但是应使用下面所列的定义。

- 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。该组件利用以下资源。

| 配置 | 资源类型 | | | | | |
|---|---|---|---|---|---|---|
| | 数据路径单元 | 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（双向的） | 1 | 12 | 3 | 2 | – | 2 |
| 16 位 (MOSI+MISO) | 2 | 12 | 3 | 1 | – | 2 |
| 16 位（双向的） | 2 | 12 | 3 | 2 | – | 2 |

# API 存储器使用

根据编译器、组件、所用 API 数量和组件配置的不同，组件内存使用会出现较大变化。下表提供指定组件配置中可用的 API 的存储器使用。

测量是使用 Release 模式（带有针对大小的优化设置）下配置的相关编译器进行的。对于特定设计，可以分析编译器生成的映射文件以确定存储器使用。

| 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) | 423 | 3 | 518 | 5 | 528 | 5 |
| 8 位（双向的） | 429 | 3 | 518 | 5 | 528 | 5 |
| 16 位 MOSI + MISO | 503 | 3 | 554 | 5 | 568 | 5 |
| 16 位（双向的） | 503 | 3 | 554 | 5 | 568 | 5 |

# 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。除非另有说明，否则这些规范的适用范围为 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 | | | | |
| | 空闲状态电流[3] | – | 18 | – | µA/MHz |
| | 工作电流[4] | – | 27 | – | µA/MHz |
| $I_{DD}$ (8 位 （双向的） | 组件电流消耗 | | | | |
| | 空闲状态电流[3] | – | 20 | – | µA/MHz |
| | 工作电流[4] | – | 32 | – | µA/MHz |
| $I_{DD}$(16 位 (MOSI+MISO) | 组件电流消耗 | | | | |
| | 空闲状态电流[3] | – | 32 | – | µA/MHz |
| | 工作电流[4] | – | 38 | – | µA/MHz |
| $I_{DD}$(16 位 （双向的） | 组件电流消耗 | | | | |
| | 空闲状态电流[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。这些值是在 25 °C 时的值。

3. 电流消耗根据新的组件时钟而定。

## 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 1 [5] | 2 * $f_{SCLK}$ | – | 10 | MHz |
| | | 配置 2 [6] | 2 * $f_{SCLK}$ | – | 10 | MHz |
| | | 配置 3 [7] | 2 * $f_{SCLK}$ | – | 8 | MHz |
| | | 配置 4 [8] | 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“额定”数字提供了额定路由条件下组件的最大安全运行频率。可以在更高的时钟频率运行组件，在该频率将需要使用 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；它不影响基本功能或比特率。路由可能限制此参数的最大频率；因此使用额定路由结果列出最大值。

| 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 | | 20 | – | – | ns |
| $t_{SCLK_SS}$ | SCLK 无效到 SS 无效 | | –20 | – | 20 | ns |

**图 6. 模式 CPHA = 0 时序图**

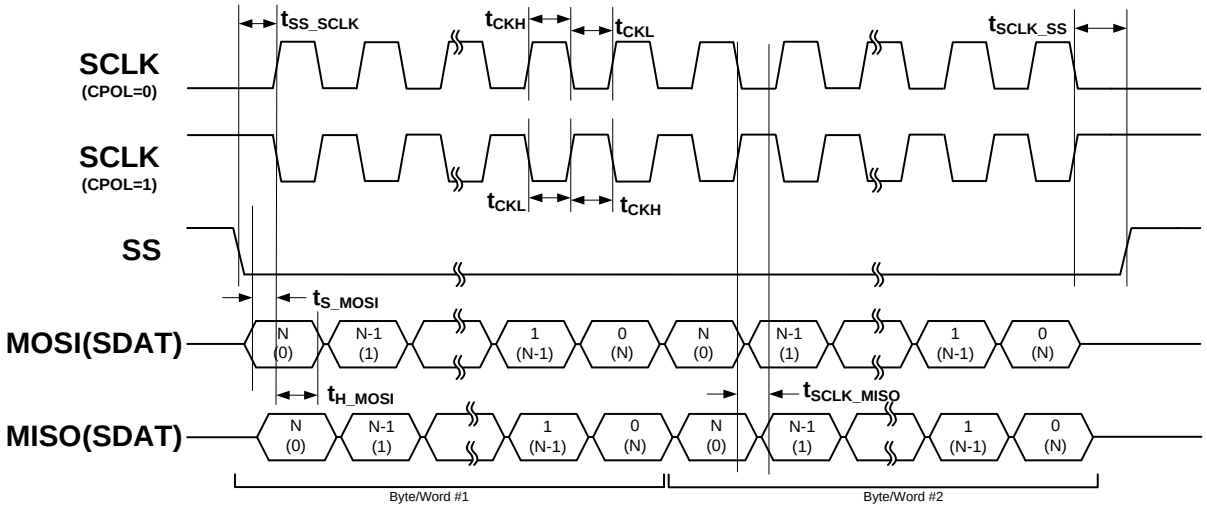
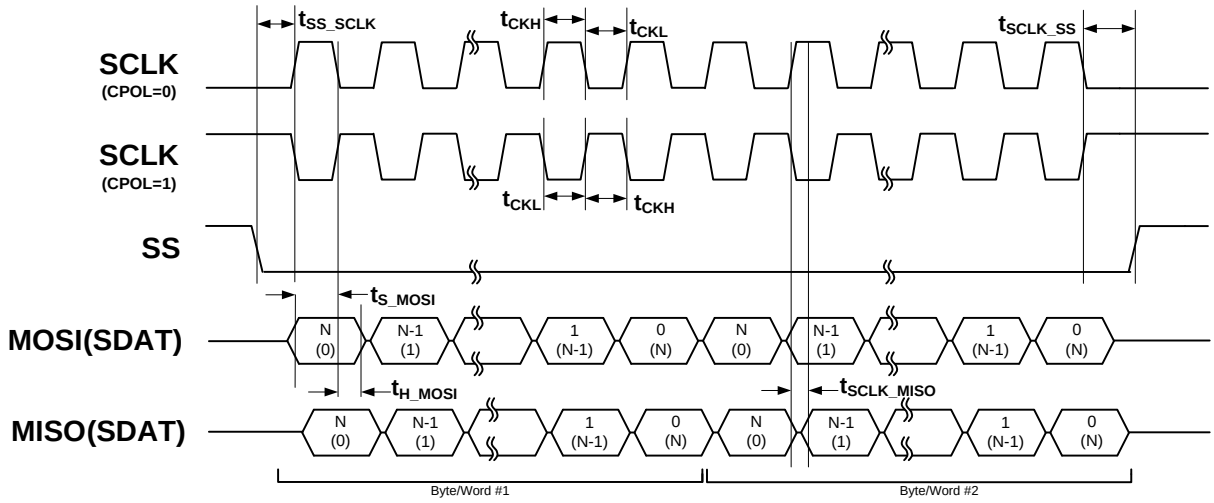



**图 7. 模式 CPHA = 1 时序图**

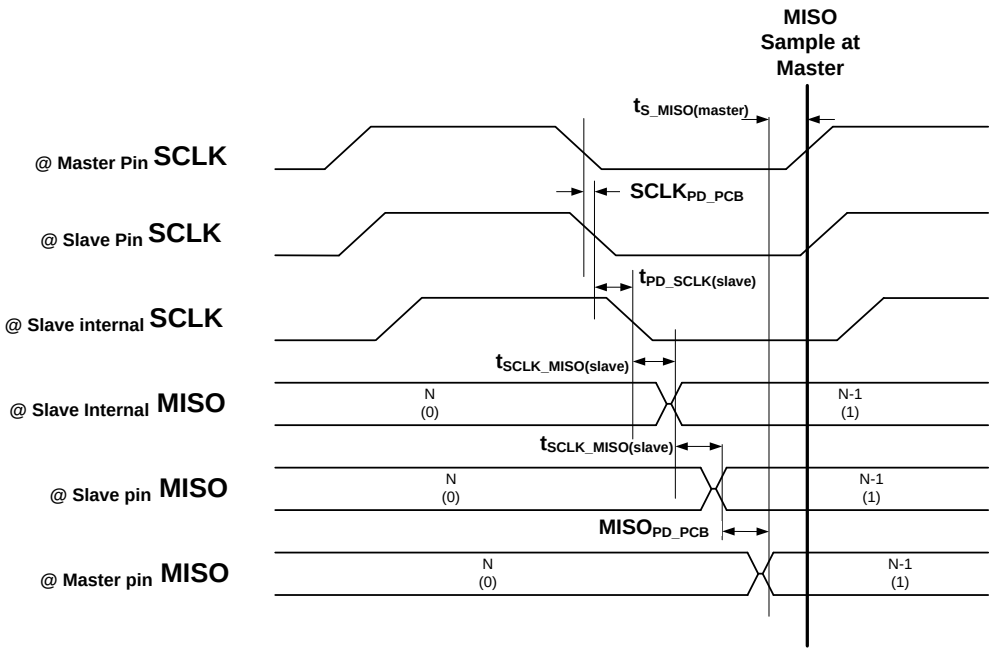



## 如何将 STA 结果用于特性数据

额定路由最大值是通过使用静态时序分析 (STA) 进行多次测试而收集的。可以通过将 STA 结果与下列机制一起使用来计算设计的最大值

$f_{SCLK}$ STA 中不直接提供 SCLK 的最大频率（或最大比特率）。不过，STA 结果中提供的数据指示了某些内部逻辑时序限制。若要计算最大比特率，必须考虑一些因素。需要板卡布局和从设备通信组件规范，才能完全了解最大值。此参数的主要限制因素是主设备管脚的 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 8. 计算最大 $f_{SCLK}$ 频率**




在此情况下，组件必须使用下列等式满足主设备的 MISO 设置时间：

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

OR

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

此处 $t_{PD_SCLK(master)} + t_{S_MISO(Master)}$ 必须源自主设备数据手册。$t_{RT_PD}$ 定义为：

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

并且：

$SCLK_{PD_PCB}$ 是从主设备管脚到从设备组件管脚的 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；$t_{SCLK_MISO(slave)}$ 为到从设备的内部逻辑路径延迟的 SCLK 管脚；且 $t_{PD_MISO(slave)}$ 为内部 MISO 到管脚的路径延迟。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 $t_{PD_SCLK(Slave)}$ 为前两个数字，$t_{SCLK_MISO(slave)}$ 为中间两个数字，$t_{PD_MISO(slave)}$ 为最后两个数字。三个参数的完整路径为 45.889 ns。

$MISO_{PD_PCB}$ 为 MISO 的 PCB 路径延迟，即由从设备组件的管脚到主设备的管脚的路径延迟。

最后的等式将提供 SCLK 的最大频率，因此最大比特率为：

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

$f_{CLOCK}$ 最大组件时钟频率作为内部时钟（如果已选择内部时钟）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，SCLK 在管脚处有效之前，MOSI 必须于此时间在管脚处有效。

$t_{H_MOSI}$ 为了满足内部逻辑的保持时间，SCLK 在管脚处有效之后，MOSI 必须于此时间在管脚处有效。

$t_{SS_SCLK}$ 为了满足模块的内部功能，在 SCLK 通过此参数在管脚处有效之前，从设备选择 (SS) 必须在管脚处有效。

$t_{SCLK_SS}$ 满足模块的内部功能的最大值。此参数表示在管脚上的 SCLK 最后一个下降沿后，管脚上的从设备选择 (SS) 必须有效。

# 组件更改

本节介绍组件与以前版本相比的主要更改。

| 版本 | 更改说明 | 更改/影响原因 |
|---|---|---|
| 2.50 | 禁用PSoC 4 的禁用固定放置选项。根据 PSoC 4 所用存储器更新数据手册。 | |
| 2.40 | 添加 MISRA 合规性章节。 | 此组件未进行 MISRA 合规性验证。 |
| | 集成了特定 API 以支持引导加载程序：CyBtldrCommStart、CyBtldrCommStop、CyBtldrCommReset、CyBtldrCommWrite、CyBtldrCommRead。 | SPI 从设备可以用作具有此功能的引导加载程序的通信组件。 |
| | 改变了 tSS_SCLK 时序参数值。最小值设置为 20 ns。最大值被删除，因为不适用。 | 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。组件 API 源文件中的注释指出了适用的函数。<br>此改变是为了消除编译器警告，其是因为不能安全地编译不可重入函数引起的：可防止同时调用标志或关键节。 |
| | 添加了 PSoC 5LP 支持 | |
| | 向数据手册中添加了 DC 特性部分 | |
| 2.20.a | 向数据手册添加了固定放置说明 | |
| 2.20 | 如果 SS 管脚在传输过程中变为高电平，则 SPI 从设备现在会丢弃已收到的任何部分字和已传输的任何部分字。只会在 ES3 芯片上出现缺陷。 | 修复了仿真模型缺陷。 |
| | 向自定义程序添加了使能固定放置选项 | |
| 2.10 | 数据位范围从 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 |
| | “字节传输完成”复选框名称更改为“字节/字传输完成” | 目的是符合真实含义 |
| | 向数据手册中添加了特性数据 | |
| | 对数据表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。 | |
| | 对数据表进行了少量编辑和更新 | |
| 2.0 | 已添加 SPIS_Sleep()/SPIS_Wakeup() 和 SPIS_Init()/SPIS_Enable() API。 | 为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。 |

| 版本 | 更改说明 | 更改/影响原因 |
|---|---|---|
| | 组件输出数和位置已改变：<br>■ 已添加复位输入；<br>■ 已删除中断输出；而添加了 rx_interrupt、tx_interrupt 输出。 | 已添加生产 PSoC 3 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。必须考虑这些变化，以避免从先前版本迁移时出现绑定错误 |
| | 已删除 SPIS_EnableInt()、SPIS_DisableInt()、SPIS_SetInterruptMode() 和 SPIS_ReadStatus() API。<br>已添加 SPIS_EnableTxInt()、SPIS_EnableRxInt()、SPIS_DisableTxInt()、SPIS_DisableRxInt()、SPIS_SetTxInterruptMode()、SPIS_SetRxInterruptMode()、SPIS_ReadTxStatus()、SPIS_ReadRxStatus() API。 | 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。还为 TX 和 Rx 缓冲区更新了中断处理程序实现。 |
| | 已将 SPIS_ReadByte()、SPIS_WriteByte() 和 SPIS_WriteByteZero() API 重命名为 SPIS_ReadRxData()、SPIS_WriteTxData()、SPIS_WriteTxDataZero()。 | 阐明了 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 芯片的两个单独的实现。<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 功能更新，而这有助于在 ES3 中优化资源使用。 |
| | ES2 支持实现中的更改：<br>现在提供两个状态寄存器（Tx 和 Rx 的状态是单独的），而非两者使用一个共同的状态寄存器。 | 这提供正确的软件缓冲区功能。 |
| | “双向模式”布尔参数已添加到基本组件（仿真模型实现）。<br>现在已选择含“时钟”输入和 SYNC 模式位的控制寄存器来驱动 ES3 芯片的“tx_enable”输出。<br>当选择 ES2 芯片时，由不带时钟输入的控制寄存器来驱动“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>“yfb”连接到“miso”输入。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，以便该工具可以支持同步和静态时序分析。 |

| 版本 | 更改说明 | 更改/影响原因 |
|---|---|---|
| | 更改了 Rx 数据路径配置。“FIFO 快速”选项设置为“DP”而非“BUS” | 修复了组件先前版本中的缺陷，即存在影响正确组件数据捕获的时序窗口。 |
| | 改变“ SPI 完成”和“字节完成”状态生成时间的仿真模型实现的附加逻辑。 | 修复了部件先前版本中的缺陷，即有时甚至在通信完成之后也不生成“SPI 完成”。 |
| | 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 |
| | 复位输入说明现在包含有关 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 |
| | 删除了示例固件源代码 | 添加了对组件示例项目的引用 |
| | SPI 模式图已改变（已添加 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，**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。