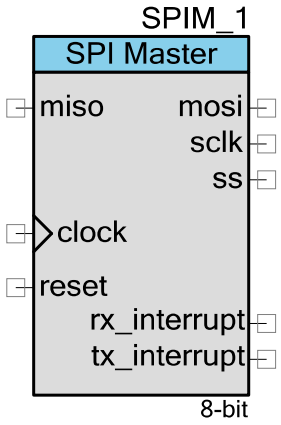
串行外设接口 (SPI) 主控

2.10



特性

- 3 到 16 位数据宽度
- 4 SPI 操作模式
- 位速率最高达 9 Mbps *

概述

SPI 主控组件提供了行业标准的 4 线主控 SPI 接口。它还提供 3 线（双向）SPI 接口。这两种接口都支持全部四个 SPI 操作模式，允许与任何 SPI 从器件设备通信。除了标准 8 位字长，SPI 主控还支持可配置 3 到 16 位字长，用于与非标准 SPI 字长通信。

SPI 信号包括标准串行时钟 (SCLK)、主控入从器件出 (MISO)、主控出从器件入 (MOSI)、双向串行数据 (SDAT) 和从器件选择 (SS)。

何时使用 SPI 主控

只要 PSoC 器件需要与一个或多个 SPI 从器件设备连接，就应当使用 SPI 主控组件。除了用于标有“SPI 从器件”的器件，SPI 主控还可用于许多实现移位寄存器类型串行接口的器件。

在需要 PSoC 器件与 SPI 主控器件通信的实例中，应当使用 SPI 从器件组件。在利用移位寄存器组件的底层灵活性来提供 SPI 主控组件中不可用的硬件功能的场合，应当使用移位寄存器组件。

---

* 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（由于内部双向引脚限制）。

输入/输出连接

本节介绍 SPI 组件的各种输入和输出连接。I/O 列表中的星号 (*) 表示该 I/O 可能在 I/O 说明中列出的情况下隐藏在符号中。

miso – 输入 *

“miso 输入”传输来自从器件设备的“主控入从器件出 (MISO)”信号。当**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**数据线**参数设置为“双向”时使用此输入。

图 1：SPI 双向模式（数据由主控传输到从器件）

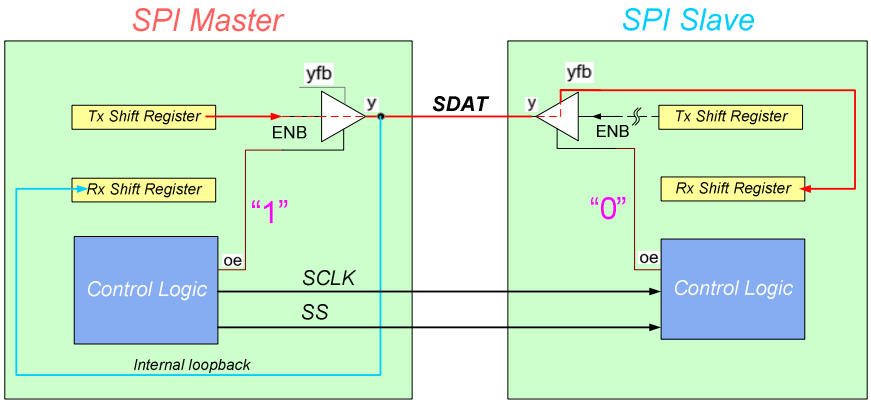
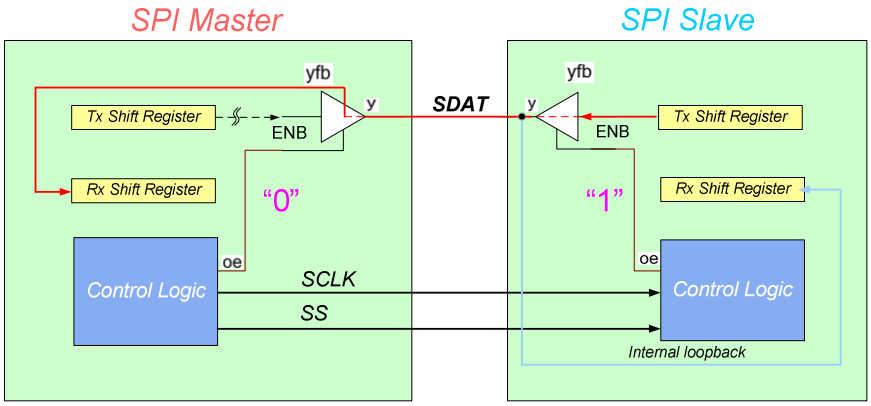


图 2：SPI 双向模式（数据由从器件传输到主控）



时钟 – 输入*

时钟输入定义串行通信的位速率。位速率是输入时钟频率的 1/2。

当**时钟选择**参数设置为“外部”时，时钟输入可见。如果可见，则必须连接此输入。如果使用“内部时钟”，则您定义所需的数据位速率，所需的时钟由 PSoC Creator 解析并提供。

复位 – 输入

将 SPI 状态机复位到空闲状态。这将丢掉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。请注意，ES2 芯片不支持路由复位功能，因而当在 ES2 芯片项目中使用组件时，应当断开此输入。

mosi – 输出 *

mosi 输出传输来自总线上的主控器件的“主控出从器件入 (MOSI)”信号。当**数据线**参数设置为“MOSI + MISO”时，此输出可见。

sclk – 输出

sclk 输出传输串行时钟 (SCLK) 信号。它将主控同步时钟输出提供给总线上的从器件设备。

ss – 输出

ss 输出受硬件控制。它将“从器件选择 (SS)”信号传输到总线上的从器件设备。可以连接数字解复用器以处理多个从器件设备，或者具有完全由固件控制的 SS。请参见下图的示例。

图 3：从器件选择输出至解复用器

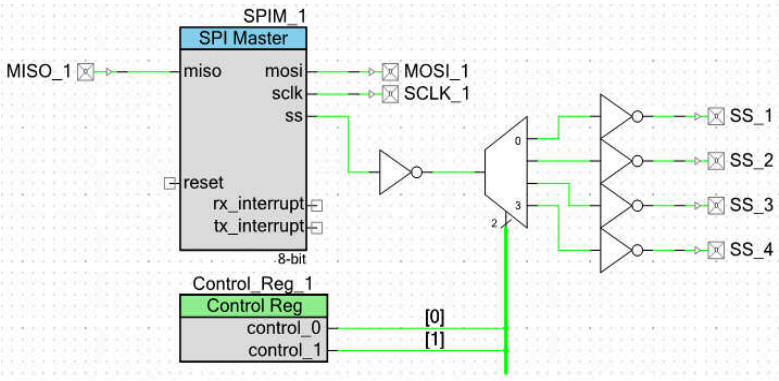
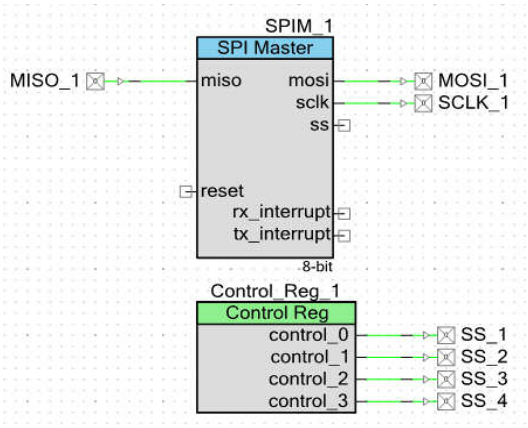
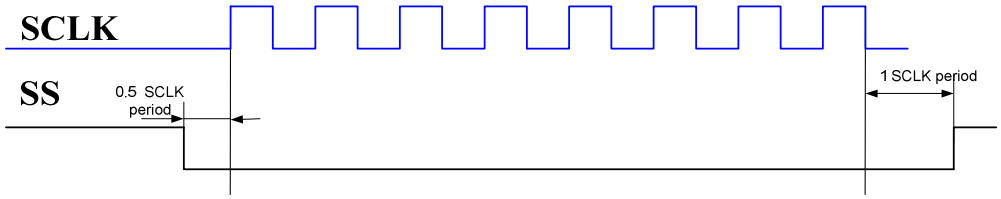


图 4：固件控制的从器件选择

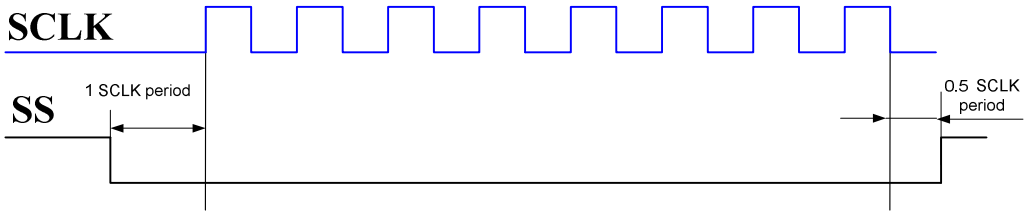


下图显示了 SS 和 SCLK 之间的时序关联：

CPHA =0：



CPHA =1：



注意：如果未生成“SPI 完成”条件，则 SS 在多字节/字传输期间不设置为“高电平”。

rx_interrupt – 输出

该中断输出是可能的 RX 中断源组的逻辑 OR 运算结果。当任何使能的 RX 中断源为 true 时，此信号将变为高电平。

tx_interrupt – 输出

该中断输出是可能的 TX 中断源组的逻辑 OR 运算结果。当任何使能的 TX 中断源为 true 时，此信号变为高电平。

原理图宏信息

默认情况下，PSoC Creator 组件目录包含 SPI 主控组件的示意宏实现。这些宏包含已连接并调整的输入和输出引脚和时钟源。示意宏可同时用于 4 线（全双工）和 3 线（双向）SPI 连接。

图 5：4 线（全双工）连接示意宏

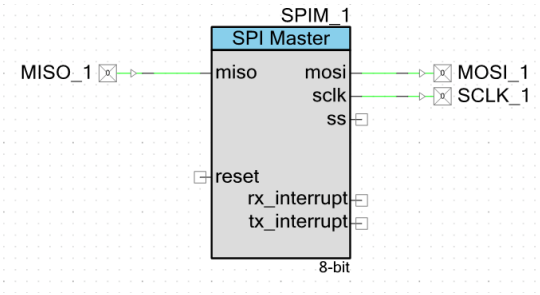
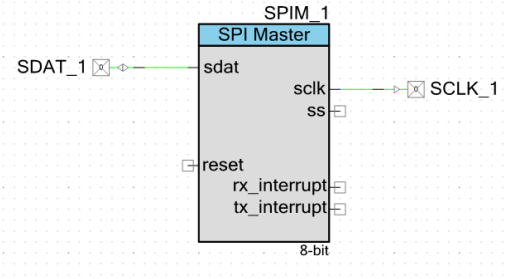


图 6：3 线（双向）连接示意宏



注意：如果不使用示意宏，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**已同步输入**参数。该参数位于相应“引脚配置”对话框的**引脚 > 输入**选项卡中。

参数和设置

将 SPI 主控组件拖动到设计中。双击组件符号以打开“配置”对话框。

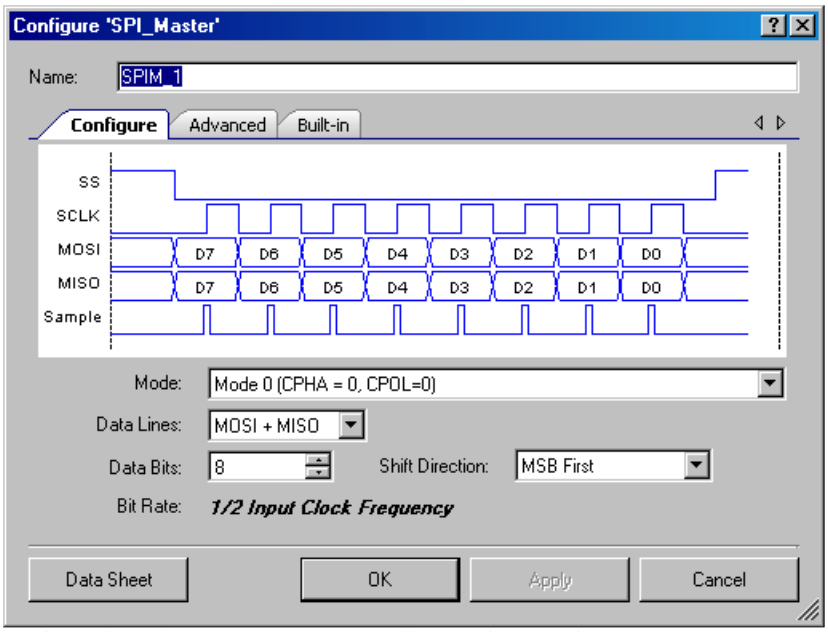
下面的章节介绍 SPI 主控参数、以及如何使用“配置”对话框对它们进行配置。这些章节还指明了是在硬件还是在软件中实现选项。

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。当您在构建之前设置这些参数时，需要设置它们的初始值，可随时使用提供的 API 修改这些初始值。仅硬件参数标有星号 (*)。

“配置”选项卡

配置选项卡包含每个 SPI 组件所需的基本参数。因此，这些参数是首先可见的用于配置的参数。



注意：波形中的示例信号不是系统的输入或输出；它只表示何时在主控或从器件针对选择的模式设置进行数据采样。

模式 *

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。选项有“模式 0”、“模式 1”、“模式 2”和“模式 3”。下表中定义了以下模式。另请参阅此数据表的“功能说明”部分。

模式	CPHA	CPOL
0	0	0
1	0	1
2	1	0
3	1	1

数据线

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 *

数据位数定义使用 SPIM_ReadRxData() 和 SPIM_WriteTxData() 函数传输时一次传输的位宽度。默认位数为单字节（8 位）。可以选择从 3 到 16 的任意整数。

移位方向 *

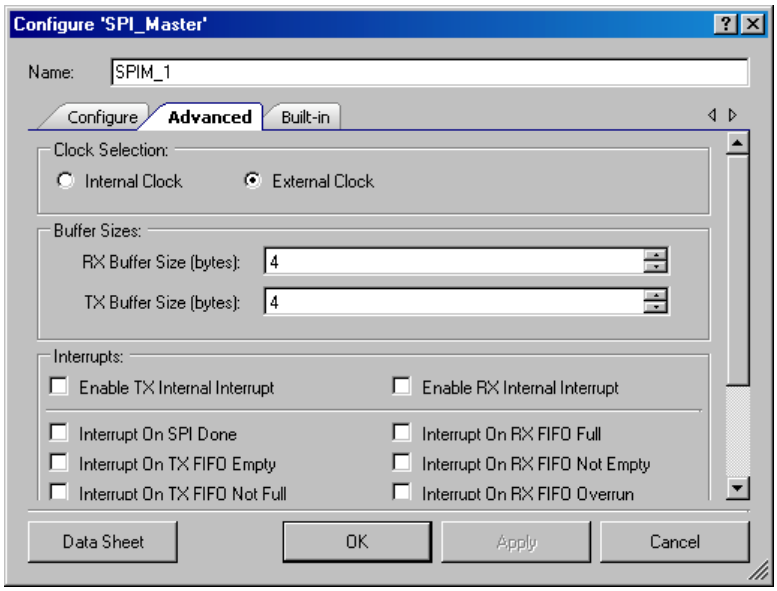
移位方向参数用于定义串行数据的传输方向。当设置为 MSB_First 时，首先传输最高有效位。这是通过将数据向左移位实现的。当设置为 LSB_First 时，首先传输最低有效位。这是通过将数据向右移位实现的。

位速率 *

如果**时钟选择**参数（位于**高级**选项卡上）设置为“内部时钟”，**位速率**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“时钟选择”参数设置为“外部时钟”，则此参数没有作用。

“高级”选项卡

高级选项卡包含提供附加功能的参数。



时钟选择 *

时钟选择参数允许用户为数据速率和 SCLK 生成选择内部配置的时钟或外部配置的时钟。当设置为“内部时钟”时，PSoC Creator 根据**位速率**参数计算并配置所需的时钟频率。当设置为“外部时钟”时，组件不控制数据速率，但是将根据用户连接的时钟源显示预计的位速率。如果此参数设置为“内部时钟”，则时钟输入在符号上不可见。

注意：当设置位速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 设置的实际时钟值。 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和达到最佳值。

RX 缓冲区大小 *

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 1-4，则在硬件中实现 FIFO 的第 4 字节/字。值 1　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图标。所有其他最大长度为 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 缓冲区大小 *

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 1-4，则在硬件中实现 FIFO 的第 4 字节/字。值 1　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图标。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。

使能 TX / RX 内部中断

使能 TX / RX 内部中断选项允许您使用 SPI 主控组件的预先定义 TX 和 RX ISR，或者提供您自己的自定义 ISR。如果启用，您可以将自己的代码添加到这些预定义 ISR 中（如果需要很少更改）。如果不选择内部中断，则可以使用连接到 SPI 主控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 主控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，SPI 从器件和全局中断必须处于使能状态，才能进行正确的缓冲区处理。

中断

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。

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。该频率是所需位速率和 SCLK 频率的 2 倍。

注意：

当设置位速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设置系统要求和达到最佳值。

放置

SPI 主控组件放置到 UDB 阵列中，所有放置信息通过 *cyfitter.h* 文件提供给 API。

资源

	数字模块					API 存储器（字节）		
分辨率	数据路径	宏单元	状态寄存器	控制寄存器	计数器 7	闪存	RAM	引脚（每个外部 I/O）
SPI 主控 8 位	1	12	2	1	1	849	19	4
SPI 主控 16 位	2	12	2	1	1	926	21	4

应用程序编程接口

应用程序编程接口 (API) 子程序允许您在运行时使用软件配置组件。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“SPIM_1”分配给提供的设计中的第一个组件实例。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。该实例名称成为每个全局函数名称、变量和常量符号的前缀。为增加可读性，下表中使用了实例名称“SPIM”。

函数	说明
SPIM_Start	调用 SPIM_Init() 和 SPIM_Start()。应当在第一次启动组件时调用。

函数	说明
SPIM_Stop	禁止 SPIM 操作。
SPIM_EnableTxInt	使能内部 TX 中断请求。
SPIM_EnableRxInt	使能内部 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	将 TX 中断源配置为使能。
SPIM_SetRxInterruptMode	将 RX 中断源配置为使能。
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	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。
SPIM_ReadRxData	返回接收缓冲区中可用接收数据的下一个字节/字。
SPIM_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）。
SPIM_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）。
SPIM_ClearRxBuffer	清除所有收到的数据的 RX 缓冲区存储器阵列和 RX FIFO。
SPIM_ClearTxBuffer	清除所有传输数据的 TX 缓冲区存储器阵列和 TX FIFO。
SPIM_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。
SPIM_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。
SPIM_PutArray	将数据数组放入传输缓冲区。
SPIM_ClearFIFO	清除从 RX 硬件 FIFO 收到的任何数据。
SPIM_Sleep	通过调用 SPIM_SaveConfig() 和 SPIM_Stop() 函数，准备将 SPIM 组件置于低功耗模式。
SPIM_Wakeup	在从低功耗模式唤醒后恢复并重新使能 SPIM 组件。
SPIM_Init	初始化并恢复默认 SPIM 配置。
SPIM_Enable	使 SPIM 能够启动操作。
SPIM_SaveConfig	保存 SPIM 硬件配置。
SPIM_RestoreConfig	恢复 SPIM 硬件配置。

全局变量

变量	说明
SPIM_initVar	指示是否已初始化 SPI 主控。变量初始化为 0，稍后在第一次调用 SPIM_Start() 时设置为 1。这允许第一次调用 SPIM_Start() 子程序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，则可以在 SPIM_Start() 或 SPIM_Enable() 函数之前调用 SPIM_Init() 函数。
SPIM_txBufferWrite	API 上次将数据写入缓冲区的传输缓冲区位置。
SPIM_txBufferRead	SPIM 硬件上次从将缓冲区读取并传输数据的传输缓冲区位置。
SPIM_rxBufferWrite	SPIM 硬件上次收到数据后将数据写入缓冲区的接收缓冲区位置。
SPIM_rxBufferRead	API 上次从缓冲区读取数据的接收缓冲区位置。
SPIM_rxBufferFull	指示软件缓冲区已溢出。
SPIM_RXBUFFER[]	用于存储收到的数据。
SPIM_TXBUFFER[]	用于存储要发送的数据。

void SPIM_Start(void)

说明： 此函数同时调用 SPIM_Init() 和 SPIM_Start()。应当在第一次启动组件时调用此函数。

参数： 无

返回值： 无

副作用： 无

void SPIM_Stop(void)

说明： 通过禁用内部时钟和内部中断（如果这样配置），禁用 SPIM 操作。

参数： 无

返回值： 无

副作用： 无

void SPIM_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 SPIM_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 SPIM_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 SPIM_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 SPIM_SetTxInterruptMode (uint8 intSrc)

说明： 配置哪些状态位触发中断事件。

参数： uint8 intSrc：包含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/字完成而使能中断
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。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，请参考此数据表中的“定义”一节。

返回值： 无

副作用： 无

void SPIM_SetRxInterruptMode (uint8 intSrc)

说明： 配置哪些状态位触发中断事件。

参数： uint8 intSrc：包含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 为空而使能中断
SPIM_INT_ON_RX_OVER	由于 RX Buf 过速而使能中断

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，请参考此数据表中的“定义”一节。

返回值： 无

副作用： 无

uint8 SPIM_ReadTxStatus (void)

说明： 返回 TX 状态寄存器的当前状态。有关更多信息，请参见此数据表中的“状态寄存器位”一节。

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 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/字完成
SPIM_STS_SPI_IDLE	SPI IDLE

副作用： TX 状态寄存器位在读取时清除。

uint8 SPIM_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“状态寄存器位”一节。

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 不为空
SPIM_STS_RX_FIFO_OVERRUN	RX Buf 过速

副作用： RX 状态寄存器位在读取时清除。

void SPIM_WriteTxData (uint8/uint16 txData)

说明： 在下一个可用 SPI 总线时间放置传输缓冲区中要发送的字节/字。

参数： uint8/uint16 txData：要从 SPI 传输的数据值。

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。

uint8/uint16 SPIM_ReadRxData (void)

说明： 返回接收缓冲区中可用接收数据的下一个字节/字。

参数： 无

返回值： uint8/uint16：从 FIFO 读取的下一字节/字的数据。

副作用： 如果 FIFO 为空，将返回无效数据。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： uint8：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 SPIM_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： uint8：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 SPIM_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。因此，写入将在地址 0 处继续，覆盖 RAM 中所有可能已存在的数据。

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 SPIM_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。因此，写入将在地址 0 处继续，覆盖 RAM 中所有可能已存在的数据。

参数： 无

返回值： 无

副作用： 将不清除 TX FIFO 中已放置的数据。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 SPIM_TxEnable (void)

说明： 如果 SPI 主控配置为使用单一双向引脚，则此函数将设置要用于传输的双向引脚。

参数： 无

返回值： 无

副作用： 无

void SPIM_TxDisable (void)

说明： 如果 SPI 主控配置为使用单一双向引脚，则此函数将设置要用于接收的双向引脚。

参数： 无

返回值： 无

副作用： 无

void SPIM_PutArray (uint8/uint16 * buffer, uint8/uint16 byteCount)

说明： 将数据数组放入传输缓冲区

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

uint8/uint16 byteCount：要移动到传输缓冲区的字节/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。如果主控未传输数据而 TX 缓冲区已满，则此函数在该循环中保持锁定。

void SPIM_ClearFIFO (void)

说明： 清除从 TX 和 RX FIFO 接收的任何数据。

参数： 无

返回值： 无

副作用： 清除组件的状态寄存器。

void SPIM_Sleep (void)

说明： 通过调用 SPIM_SaveConfig() 和 SPIM_Stop() 函数，准备将 SPIM 组件置于低功耗模式。

参数： 无

返回值： 无

副作用： 无

void SPIM_Wakeup (void)

说明： 准备 SPIM 组件以从低功耗模式中唤醒。调用 SPIM_RestoreConfig() 和 SPIM_Enable() 函数。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： 无

void SPIM_Init(void)

说明： 根据定制器“配置”对话框设置来初始化或恢复组件。不必调用 SPIM_Init()，因为 SPIM_Start() 子程序将调用此函数，这是开始组件操作的首选方法。

参数： 无

返回值： 无

副作用： 当调用此函数时，它初始化所有需要执行的参数。这些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 SPIM 能够启动操作。启动内部时钟（如果这样配置）。如果配置了外部时钟，则必须在调用此函数之前单独启动它。应当在 SPIM 中断使能之前调用 SPIM_Enable() 函数。这是因为此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（如果这样配置）。以前必须已调用 SPIM_Init() 函数。

参数： 无

返回值： 无

副作用： 无

void SPIM_SaveConfig (void)

说明： 在进入低功耗模式之前保存 SPIM 硬件配置。

参数： 无

返回值： 无

副作用： 无

void SPIM_RestoreConfig (void)

说明： 从低功耗模式唤醒后恢复 SPIM_SaveConfig() 函数保存的 SPIM 硬件配置。

参数： 无

返回值： 无

副作用： 如果在未首先调用 SPIM_SaveConfig() 的情况下调用此函数，则在下列寄存器中，将恢复为“配置”对话框中的默认值：

```
SPIM_STATUS_MASK_REG
SPIM_COUNTER_PERIOD_REG
```

定义

- **SPIM_TX_INIT_INTERRUPTS_MASK** – 定义“配置”对话框中选择的中断源的初始配置。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。
- **SPIM_RX_INIT_INTERRUPTS_MASK** – 定义“配置”对话框中选择的中断源的初始配置。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”一节。

状态寄存器位

表 1 SPIM_TXSTATUS

位	7	6	5	4	3	2	1	0
值	中断	未使用	未使用	SPI IDLE	字节/字完成	TX FIFO 未满	TX FIFO 为空	SPI 完成

表 2 SPIM_RXSTATUS

位	7	6	5	4	3	2	1	0
值	中断	RX 缓冲区过速	RX FIFO 不为空	RX FIFO 已满	未使用	未使用	未使用	未使用

- 字节/字完成：当完成数据字节/字传输时设置。
- RX FIFO 过速：当 RX 数据在未移到 RX 缓冲区存储器阵列（如果此阵列存在）的情况下已过速 4 字节/字 FIFO 时设置
- 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）。
- 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）。
- 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）。
- 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）。
- 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。（当设置了“字节/字完成”且 TX 数据 FIFO 为空时设置）。
- SPI IDLE：当 SPIM 状态机处于 IDLE 状态时设置。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。仍一直设置 IDLE，直到检测到“TX FIFO 非空状态”。

SPIM_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，则实现将数据自动从循环存储器缓冲区移动到 FIFO 的中断。

SPIM_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，则实现将数据自动从 FIFO 移动到循环存储器缓冲区的中断。

SPIM_DATAWIDTH

定义“配置”对话框中选择的每个数据传输的位数。

固件源代码示例

PSoC Creator 在“查找示例项目”对话框中提供了大量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，请打开开始页或**文件**菜单中的对话框。根据需要，使用对话框中的**滤波器选项**可缩小可选项目的列表。

有关更多信息，请参考 PSoC Creator 帮助中的“查找示例项目”主题。

功能说明

默认配置

SPIM 的默认配置是具有模式 0 配置的 8 位 SPIM。默认情况下，使用 1 Mbps 位速率选择内部时钟。

模式

为了显示数据传输期间预计的组件状态位和组件信号值，显示了 4 个波形。假设传输了 5 个数据字节（在传输开始时 4 个字节写入 SPI 主控的 TX 缓冲区，第 1 个字节加载到 A0 寄存器中后丢弃第 5 个字节）。取整的数字表示下列事件：

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 – 由于检测到进入 Tx 缓冲区的字节，已清除 SPI IDLE 状态位；

4 – 在第 1 个字节加载到 A0 寄存器中时设置“Tx FIFO 未满”状态，在第 5 个字节写入 Tx 缓冲区中的空闲位置时清除“Tx FIFO 未满”状态。

5 – “从器件选择”线路设置为低电平状态，表示传输开始。

6 – 当第 2 位加载到 A0 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”。

7 – 在最后一个要发送的字节加载到 A0 寄存器中时设置“Tx FIFO 为空”状态（为了简单起见，不显示详细信息）。

8 – 当收到第 4 个字节时，“Rx FIFO 已满”与“字节/字完成”一起设置。

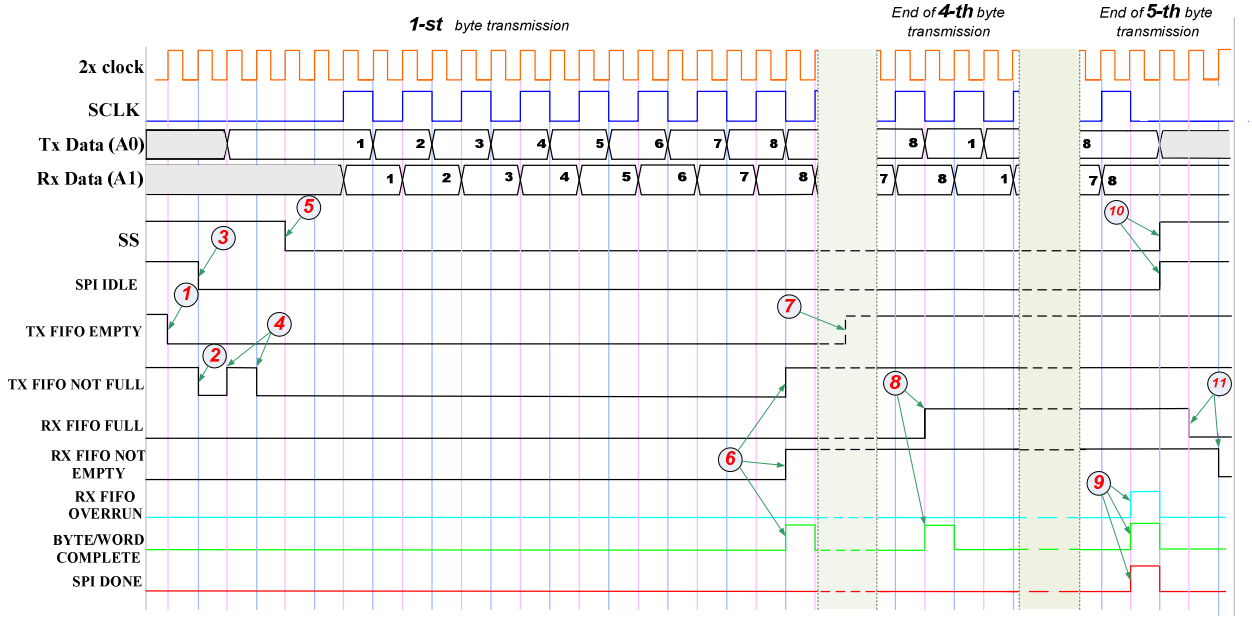
9 – 设置了“字节/字完成”、“SPI 完成”和“Rx 过速”，这是因为所有字节都已传输，且检测到尝试将数据加载到整个 Rx 缓冲区。

10 – SS 线路设置为高电平，以指示传输已完成。还设置了“SPI IDLE”状态。

11 – 当从 Rx 缓冲区读取第 1 个字节时清除“Rx FIFO 已满”，当读取完所有字节时设置“Rx FIFO 为空”。

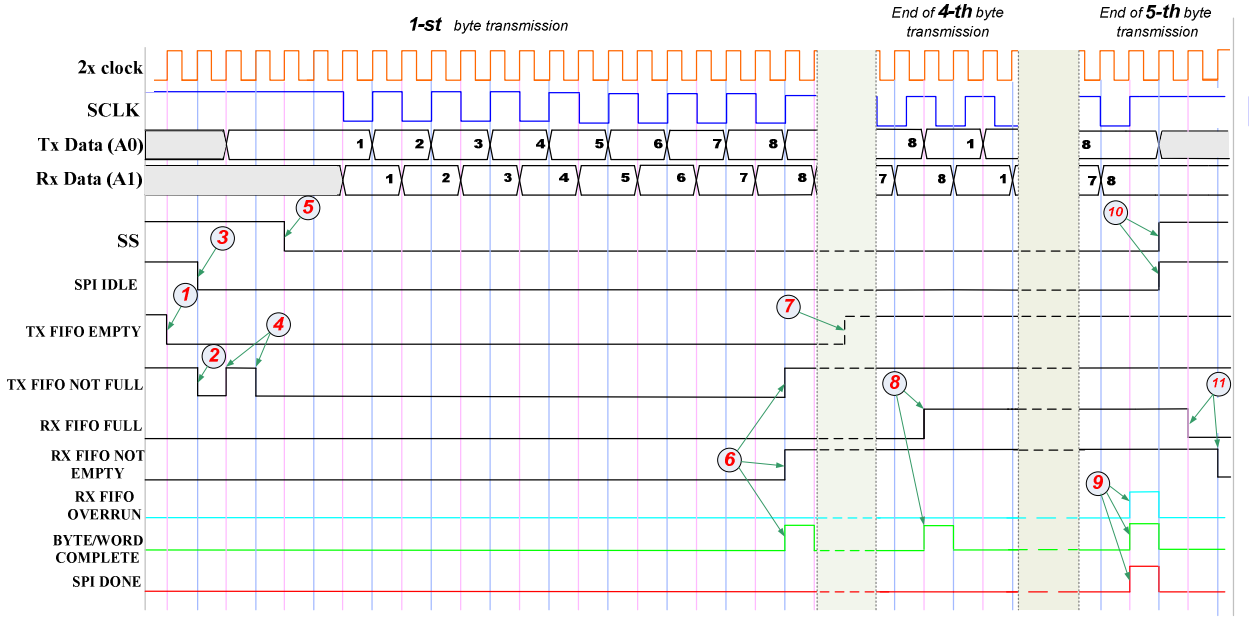
SPIM 模式：0（CPHA == 0，CPOL == 0）

模式 0 具有下列特性：



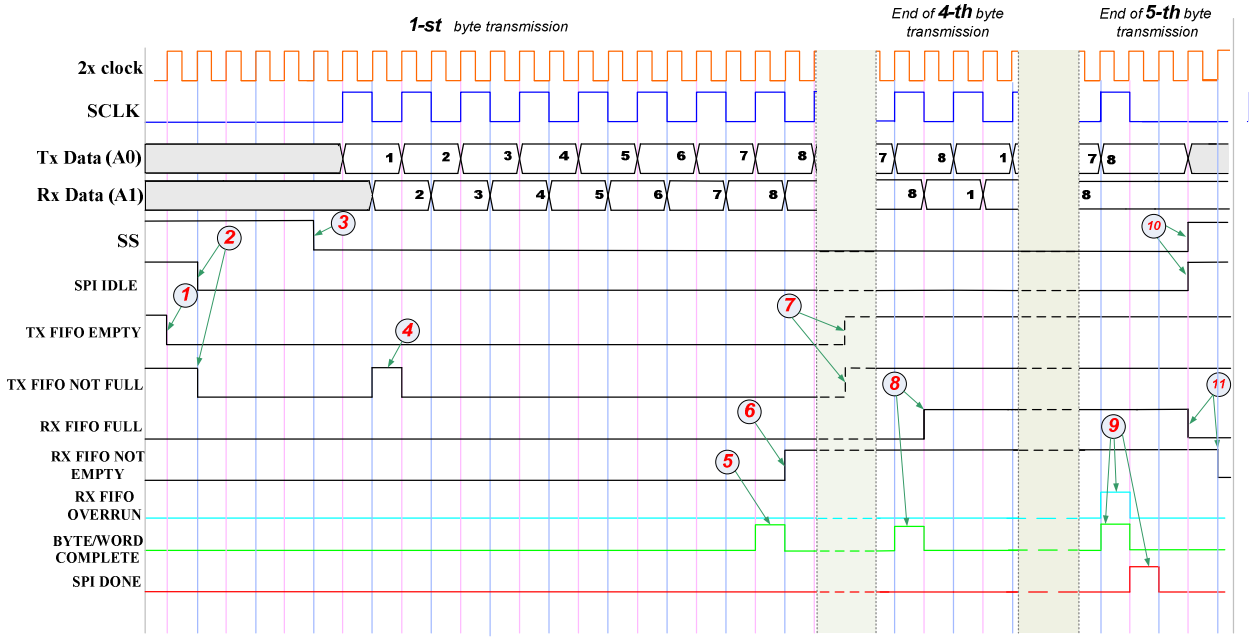
SPIM 模式：1（CPHA == 0，CPOL == 0）

模式 1 具有下列特性：



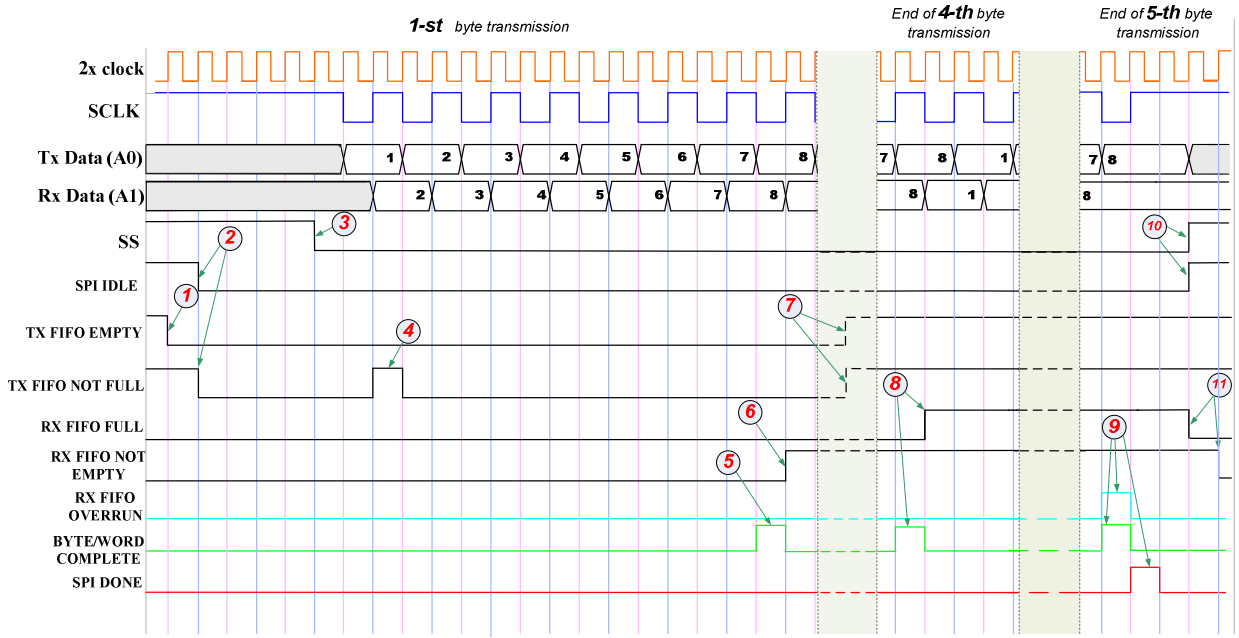
SPIM 模式: 2 (CPHA == 1, CPOL == 0)

模式 2 具有下列特性：



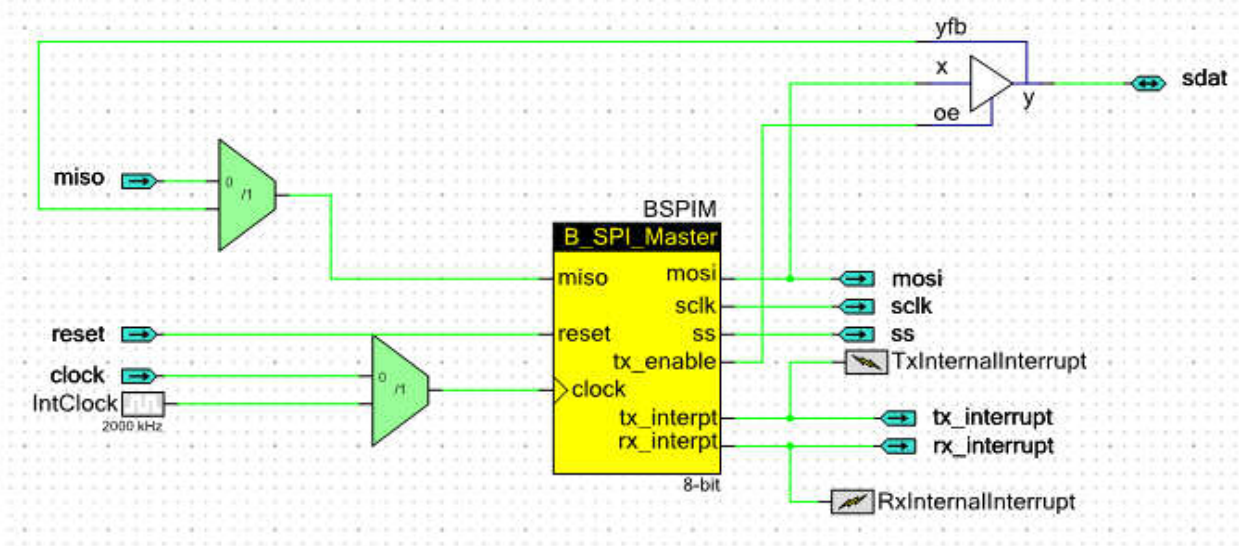
SPIM 模式: 3 (CPHA == 1, CPOL == 1)

模式 3 具有下列特性：

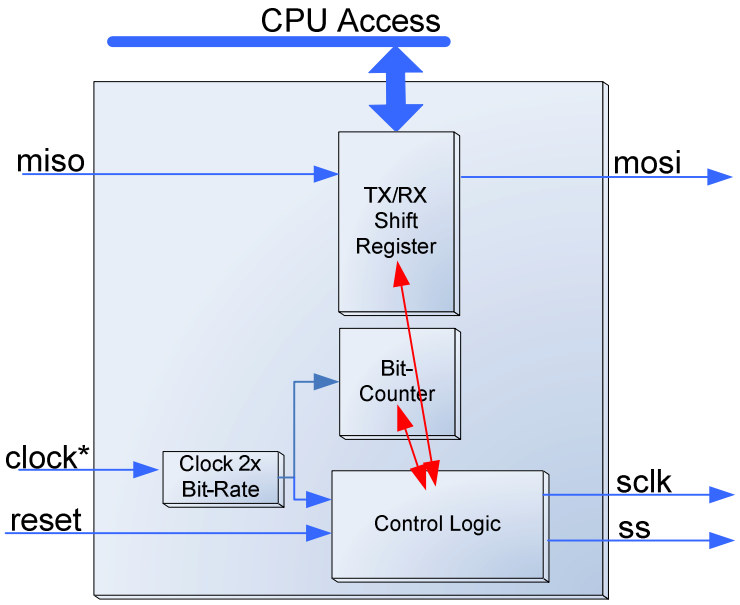


框图和配置

SPIM 仅作为 UDB 配置提供。此处描述的寄存器用于定义 SPIM 的硬件实现。



下面的框图中描述了实现。



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 SPIM 组件的给定实例定义的各种传输状态位。假设 SPIM 实例命名为“SPIM”，在 SPIM_ReadTxStatus() 函数中提供此寄存器的值。

中断输出信号是 TX 状态寄存器中的掩码位字段的 OR 运算结果。可以使用 SPIM_SetTxInterruptMode() 函数设置掩码。接收中断时，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到调用 SPIM_ReadTxStatus() 函数为止。TX 状态寄存器上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 状态寄存器中移动。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，以清除这些位和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 #defines：

- SPIM_STS_SPI_DONE * – 在单一 SPI 字中的配置位数的最后一位输出到 MOSI 线路且传输 FIFO 为空后输出 SCLK 的数据锁存边沿（边沿取决于模式）时，设置为高电平。当 SPIM 正在传输数据或传输 FIFO 具有挂起数据时清除。用于了解何时使用多字数据操作完成 SPIM。

- SPIM_STS_TX_FIFO_EMPTY – 当传输 FIFO 不包含数据挂起传输时，设置为高电平。如果有等待传输的数据，则设置为低电平。
- SPIM_STS_TX_FIFO_NOT_FULL – 当传输 FIFO 未满且有写入更多数据的空间时，设置为高电平。如果 FIFO 充满等待传输的数据且此时没有更多写入空间，则设置为低电平。用于了解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 字中的配置位数的最后一位输出到 MOSI 线路中时，设置为高电平。当输入 SCLK 的数据锁存边沿（边沿取决于模式）时清除*。
- SPIM_STS_SPI_IDLE * – 只要组件状态机处于 SPI IDLE 状态，此位就会设置为高电平。（组件正在等待 Tx 数据，且未传输任何数据）

RX 状态寄存器

RX 状态寄存器是只读寄存器，它包含为 SPIM 定义的各种接收状态位。通过 SPIM_ReadRxStatus() 函数提供此寄存器的值。

中断输出信号是 RX 状态寄存器中的掩码位字段的 OR 运算结果。可以使用 SPIM_SetRxInterruptMode() 函数设置掩码。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到调用 SPIM_ReadRxStatus() 函数为止。RX 状态寄存器上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 RX 状态寄存器中移动。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，以清除这些位和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 #defines：

- 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，设置为高电平。如果 FIFO 未满且此时有空间存储其他接收的数据，则设置为低电平。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，设置为高电平。如果 FIFO 为空且此时有空间存储其他接收的数据，则设置为低电平。
- SPIM_STS_RX_FIFO_OVERRUN * – 当接收 FIFO 已满且有附加数据写入它时，设置为高电平。用于了解 FIFO 中的数据是否由于 FIFO 服务速度慢而丢失。

TX 数据寄存器

TX 数据寄存器包含要发送的传输数据值。它实现为 SPIM 中的 FIFO。提供了一个可选的更高级别软件状态机来控制传输存储器缓冲区中的数据，以处理大量要发送的超过 FIFO 容量的数据。所

有处理传输数据的 API 必须通过此寄存器才能将数据放到总线上。如果此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。只要此寄存器 (FIFO) 为空，则在数据添加到 FIFO 中之前将没有更多数据传输到总线上。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 数据寄存器

RX 数据寄存器包含收到的数据。它实现为 SPIM 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 中断将指示数据具有固件的若干路由时已收到数据。可以设置从寄存器到存储器阵列进行 DMA，或者固件只需调用 SPIM_ReadRxData() 函数。DAM 必须使用头文件中定义的 RXDATA_REG 地址。

有条件编译信息

SPIM 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宽度。API 永远不应当直接使用这些参数，但是应当使用下列定义：

- 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 3-16 位。

参考

不适用

直流和交流电气特性

下面的值表示了预计性能，它们基于初始特性数据。

时序特性“额定路由的最大值”

参数	说明	配置	最小值	典型值	最大值 [1]	单位
f_{SCLK}	SCLK 频率	配置 1 [2]			9	MHz
		配置 2 [3]			9	MHz

[1]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。

[2] 配置 1 选项：
数据线： MOSI+MISO
数据位： 8

		配置 3 [4]			8	MHz
		配置 4 [5]			8	MHz
f_{clock}	组件时钟频率 [6]	配置 1 [2]	2 * f_{SCLK}		18	MHz
		配置 2 [3]	2 * f_{SCLK}		18	MHz
		配置 3 [4]	2 * f_{SCLK}		16	MHz
		配置 4 [5]	2 * f_{SCLK}		16	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_{S_MISO}	MISO 输入设置时间		25			ns
t_{H_MISO}	MISO 输入保留时间			0		ns
t_{SS_SCLK}	SS 有效到 SCLK 有效		-20		20	ns
t_{SCLK_SS}	SCLK 无效到 SS 无效		-20		20	ns

[3] 配置 2 选项：
数据线： MOSI+MISO
数据位： 16

[4] 配置 3 选项：
数据线： 双向
数据位： 8

[5] 配置 4 选项：
数据线： 双向
数据位： 16

[6]组件时钟仅用于状态寄存器；它不影响基本功能或位速率。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。

时序特性“所有路由的最大值”

参数	说明	配置	最小值	典型值	最大值 [1]	单位
f_{SCLK}	SCLK 频率	配置 1 [2]			4.5	MHz
		配置 2 [3]			4.5	MHz

[1] “所有路由”的最大值的计算方法是：〈额定值〉/2，然后取整到最近的整数。此值提供了一个基础，因此用户不必担心在以该组件频率或低于该组件频率运行时遇到时序问题。

[2] 配置 1 选项：
数据线： MOSI+MISO
数据位： 8

		配置 3 [4]			4	MHz
		配置 4 [5]			4	MHz
f_{clock}	组件时钟频率[6]	配置 1 [2]	2 * f_{SCLK}		9	MHz
		配置 2 [3]	2 * f_{SCLK}		9	MHz
		配置 3 [4]	2 * f_{SCLK}		8	MHz
		配置 4 [5]	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_{S_MISO}	MISO 输入设置时间		25			ns
t_{H_MISO}	MISO 输入保留时间			0		ns
t_{SS_SCLK}	SS 有效到 SCLK 有效		-20		20	ns
t_{SCLK_SS}	SCLK 无效到 SS 无效		-20		20	ns

[3] 配置 2 选项：
数据线： MOSI+MISO
数据位： 16

[4] 配置 3 选项：
数据线： 双向
数据位： 8

[5] 配置 4 选项：
数据线： 双向
数据位： 16

[6]组件时钟仅用于状态寄存器；它不影响基本功能或位速率。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。

图 7. 模式 CPHA=0 时序图

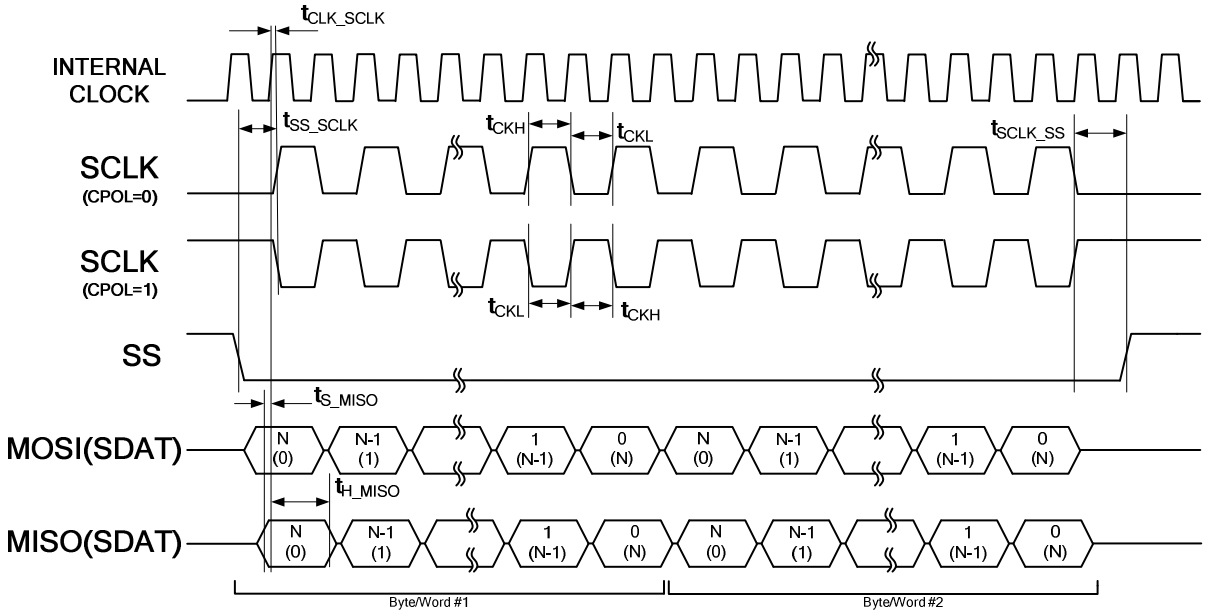
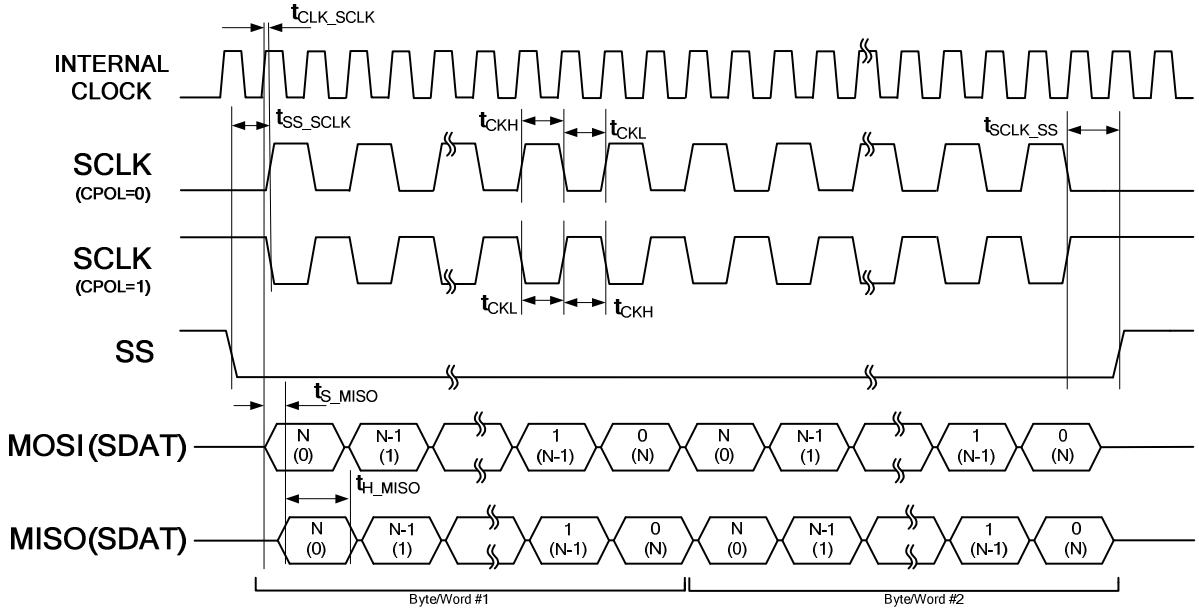


图 8. 模式 CPHA=1 时序图



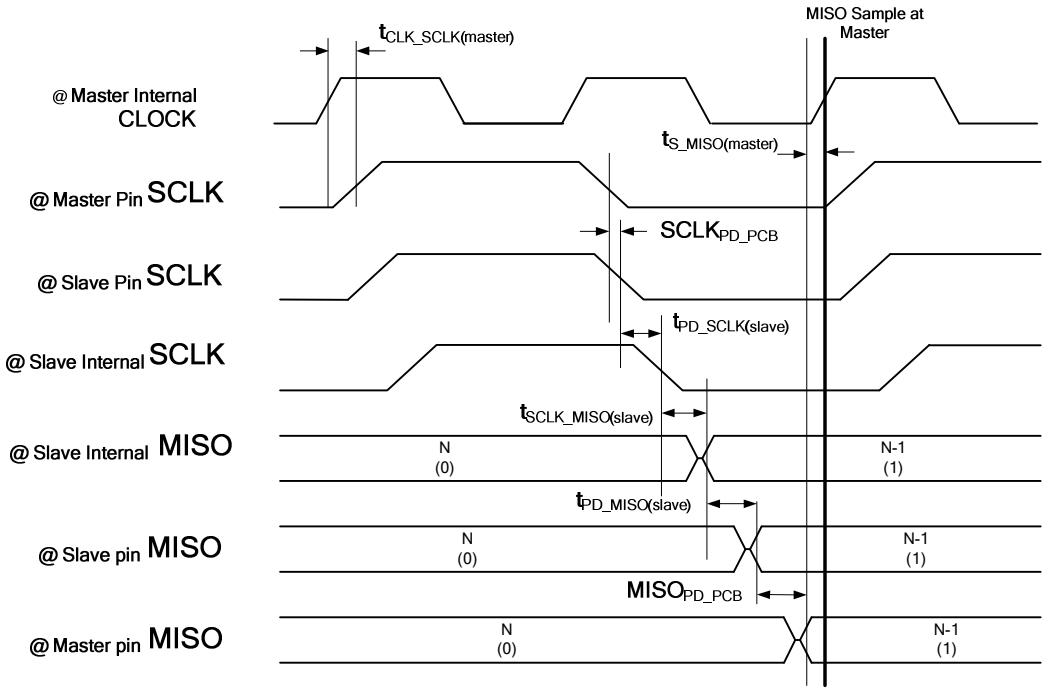
如何将 STA 结果用于特性数据

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（或最大位速率）。不过，STA 结果中提供的数据指示了某些内部逻辑时序限制。若要计算最大位速率，必须考虑一些因素。需要板卡布局

和从器件通信器件规范，才能完全了解最大值。此参数的主要限制因素是主控引脚的 SCLK 下降沿到从器件的往复路径延迟，以及从器件的 MISO 输出返回到主控的路径延迟。

图 9 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控的 MISO 设置时间：

$$t_{RT_PD} < 1 / \{ [½ * f_{SCLK}] - t_{CLK_SCLK(master)} - t_{S_MISO(master)} \}$$

-- 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 结果时钟输出时间中提供的，如下所示：

SPIM_1_IntClock	Net_25:macrocell.mc_q	SCLK_1(0):iocell.pad_out	23.241

$t_{S_MISO(Master)}$ 是从 MISO 输入引脚到主控组件内部逻辑的路径延迟；这是在 STA 结果输入设置时间中提供的，如下所示：

-Setup times to clock SPIM_1_IntClock

Start	Register	Clock	Delay (ns)
MISO_1(0):iocell.pad_in	\SPIM_1:BSPIM:sR8:Dp:u0\:datapathcell.regd_a1	SPIM_1_IntClock	19.358
MISO_1(0):iocell.pad_in	\SPIM_1:BSPIM:sR8:Dp:u0\:datapathcell.regd_a0	SPIM_1_IntClock	19.348

注意：报告中有两个值：寄存器 A0 和寄存器 A1 处的组件时钟设置时间。应当选择较大的值来进行计算。

t_{RT_PD} 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：

$SCLK_{PD_PCB}$ 是从主控组件引脚到从器件组件引脚的 SCLK 的 PCB 路径延迟。

$t_{PD_SCLK(slave)} + t_{SCLK_MISO(slave)} + t_{PD_MISO(slave)}$ 必须来自从器件设备数据手册。$MISO_{PD_PCB}$ 是由从器件组件引脚到主控组件引脚的 MISO 的 PCB 路径延迟。

最后的等式将提供 SCLK 的最大频率，因此最大位速率为：

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

f_{clk} 最大组件时钟频率是在时钟汇总的时序结果中以 IntClock（如果选择了内部时钟）或命名外部时钟的形式提供的。下面是 _timing.html 文件中的内部时钟限制示例：

-Clock Summary

Clock	Actual Freq	Max Freq	Violation
BUS_CLK	66.000 MHz	UNKNOWN	
SPIM_1_IntClock	2.000 MHz	45.108 MHz	

t_{CKH} SPI 主控组件需要 50% 占空比 SCLK

t_{CKL} SPI 主控组件需要 50% 占空比 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机引脚上可用 SCLK 的时间。

t_{S_MISO} 为了满足内部逻辑的设置时间，在引脚上的 FCLK 有效之前的这段时间，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 FCLK 有效之后的这段时间，引脚上的 MISO 必须有效。

t_{SS_SCLK} 为了满足模块的内部功能。此参数表示在引脚上的 SCLK 有效之前，引脚上的从器件选择 (SS) 必须有效。

t_{SCLK_SS} 最大值 - 为了满足模块的内部功能。此参数表示在引脚上的 SCLK 最后一个下降沿后，引脚上的从器件选择 (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
2.10	数据位范围从 2-16 位更改为 3-16	当前版本中修复了与状态同步相关的更改问题
	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
	“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中的子文件夹	
2.0	添加了 SPIM_Sleep()/SPIM_Wakeup() 和 SPIM_Init()/SPIM_Enable API。	为了支持低功耗模式，以及提供通用接口以分别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： • 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。	添加了时钟输入，以便与 SPI 从器件保持一致。 由于添加了时钟输入，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。 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
	删除了 SPIM_EnableInt()、SPIM_DisableInt()、_SetInterruptMode 和 _ReadStatus API。 添加了 SPIM_EnableTxInt()、SPIM_EnableRxInt()、SPIM_DisableTxInt()、SPIM_DisableRxInt()、SPIM_SetTxInterruptMode()、SPIM_SetRxInterruptMode()、SPIM_ReadTxStatus() 和 SPIM_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。
	将 SPIM_ReadByte() 和 SPIM_WriteByte()APIs 重命名为 SPIM_ReadRxData() 和 SPIM_WriteTxData()。	阐明了 API 以及使用它们的方式。
在使用仿真模型实现的基本 SPI 主控组件 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，且不包含任何异步逻辑。

版本	更改说明	更改/影响原因

	现在提供了 statusi 寄存器（分别为 Tx 和 Rx 提供了状态，而不是将一个通用状态寄存器用于二者）。
	/*SPI_Master_v1_20 状态位*/
	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 状态位*/
	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;
		修复了以前组件版本中发现的软件缓冲区未按预期工作的缺陷。

	基本组件中添加了“BidirectMode”布尔参数。 现在选择了具有“clock”输入和 SYNC 模式位的控制寄存器，以驱动 ES3 芯片的“tx_enable”输出。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”。 组件图示中使用了 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”连接到“miso”输入。Bufoe“y”输出连接到“sdat”输出终端。 路由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
	udb_clock_enable 组件添加到仿真模型实现中，并添加了同步 = “TRUE”参数。	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。
	在计数器 7 周期值中，“*2”替换为“<< 1”	改进了仿真模型。
	最大位速率值更改为 10 Mbps	不支持大于 10 Mbps 的位速率值（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
	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，2011。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC® 是赛普拉斯半导体公司的注册商标，PSoC Creator™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码（软件和/或固件）均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和/或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定的用途之外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对此材料提供任何类型的明示或暗示保证，包括（但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不做出通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用的赛普拉斯软件许可协议限制。