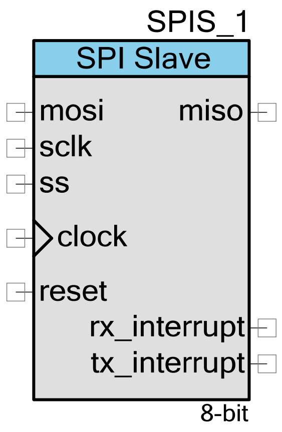
Serial Peripheral Interface (SPI) Slave

2.10

Features

- 3 to 16-bit data width
- 4 SPI modes
- Bit Rate up to 5 Mbps*



General Description

The SPI Slave provides an industry-standard, 4-wire slave SPI interface and 3-wire. It can also provide a 3-wire (bidirectional) SPI interface. Both interfaces support all four SPI operating modes, allowing communication with any SPI master device. In addition to the standard 8-bit word length, the SPI Slave supports a configurable 3- to 16-bit word length for communicating with nonstandard SPI word lengths.

SPI signals include the standard Serial Clock (SCLK), Master In Slave Out (MISO), Master Out Slave In (MOSI), bidirectional Serial Data (SDAT), and Slave Select (SS).

When to Use the SPI Slave

The SPI Slave component should be used any time the PSoC device is required to interface with an SPI Master device. In addition to "SPI Master" labeled devices, the SPI Slave can be used with many devices implementing a shift register type interface.

The SPI Master component should be used in instances requiring the PSoC device to communicate with an SPI Slave device. The Shift Register component should be used in situations where its low level flexibility provides hardware capabilities not available in the SPI Slave component.

* This value is valid only for MOSI+MISO (Full Duplex) interfacing mode (see “DC and AC electrical characteristics” section for details) and is restricted up to 1 Mbps in Bidirectional mode because of internal bidirectional pin constraints.

Input/Output Connections

This section describes the various input and output connections for the SPI. An asterisk (*) in the list of I/O indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

mosi – Input *

The miso input carries the Master In Slave Out (MISO) signal from a slave device. This input is visible when the **Data Lines** parameter is set to "MOSI + MISO." If visible, this input must be connected.

sdat – Inout *

The sdat inout carries the Serial Data (SDAT) signal. This input is used when the **Data Lines** parameter is set to "Bidirectional."

Figure 1. SPI Bidirectional Mode (data transmission from Master to Slave)

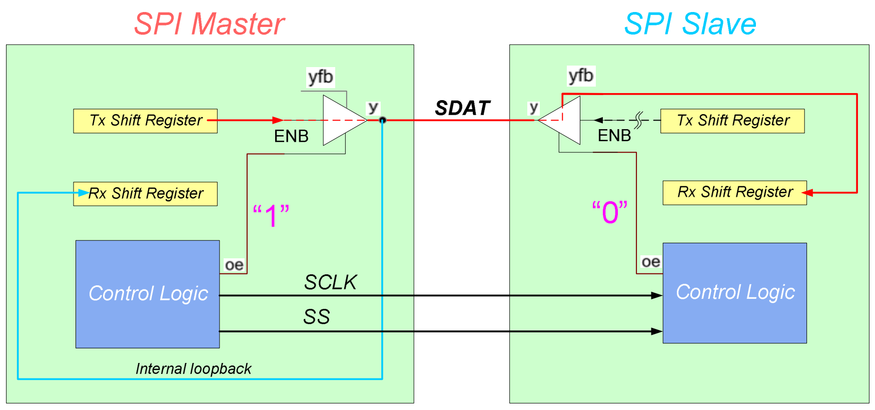
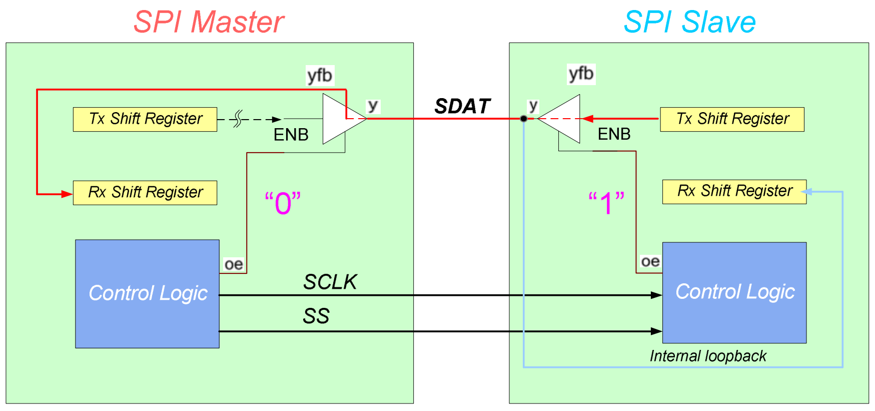


Figure 2. SPI Bidirectional Mode (data transmission from Slave to Master)



sclk– Input

The sclk input carries the Serial Clock (SCLK) signal. It provides the slave synchronization clock input to this device. This input is always visible and must be connected.

Note Some SPI Master devices (such as the TotalPhase Aardvark I2C/SPI host adapter) drive the sclk output in a specific way. For the SPI Slave component to function properly with such devices in modes 1 and 3 (when CPOL =1), the sclk pin should be set to resistive pull up drive mode. Otherwise, it gives out corrupted data. See the Functional Description section for more information about modes.

ss – Input

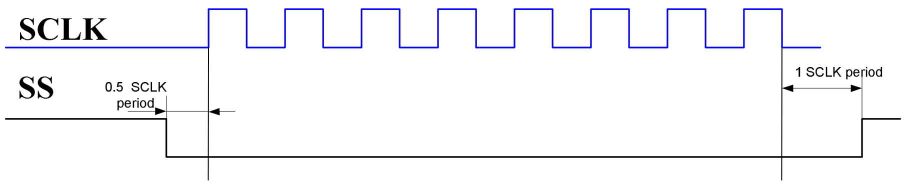
The ss input carries the Slave Select (SS) signal to this device. This input is always visible and must be connected.

The following diagrams show the timing correlation between SCLK and SS signals

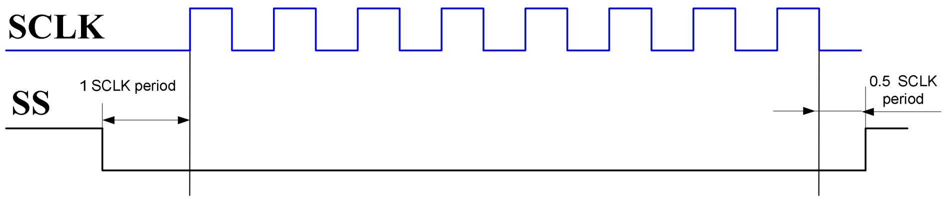
reset – Input

Resets the SPI Slave. This will throw out any data that was currently being transmitted or received but will not clear data from the FIFO that has already been received or is ready to be transmitted. Note that ES2 silicon does not support routed reset functionality and this input should be unconnected when component is used in ES2 silicon projects.

CPHA = 0:



CPHA = 1:



Note The SS timing shown in this diagram is valid for the PSoC Creator SPI Master. Generally 0.5 of the SCLK period is enough delay between the SS negative edge and the first SCLK edge for the SPI Slave to work correctly in all supported bit rate ranges.

clock – Input *

The clock input defines the sampling rate of the status register. All data clocking happens on the sclk input, so the clock input does **not** handle the bit-rate of the SPI Slave.

The clock input is visible when the **Clock Selection** parameter is set to "External." If visible, this input must be connected.

miso – Output *

The miso output carries the Master In Slave Out (MISO) signal to the master device on the bus. This output is visible when the **Data Lines** parameter is set to "MOSI + MISO."

interrupt – Output

The interrupt output is the logical OR of the group of possible interrupt sources. This signal will go high while any of the enabled interrupt sources are true.

Schematic Macro Information

By default, the PSoC Creator Component Catalog contains Schematic Macro implementations for the SPI Slave component. These macros contain already connected and adjusted input and output pins and clock source. Schematic Macros are available for 4–wire (Full Duplex), 3-wire (Bidirectional), and Full Duplex Multislave SPI interfacing.

Figure 3. 4-wire (Full Duplex) Interfacing Schematic Macro

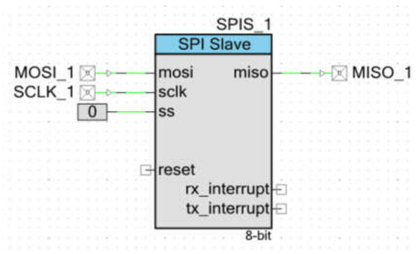


Figure 4. 3-wire (Bidirectional) Interfacing Schematic Macro

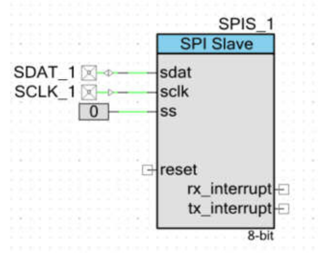
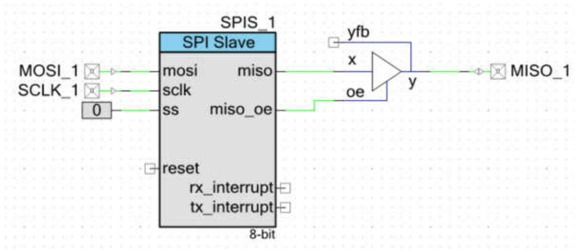


Figure 5. Multislave Mode Schematic Macro



Note If you do not use a Schematic Macro, configure the Pins component to deselect the **Input Synchronized** parameter for each of your assigned input pins (MOSI, SCLK and SS). The parameter is located under the **Pins > Input** tab of the applicable Pins Configure dialog.

Parameters and Setup

Drag an SPI Slave component onto the design. Double-click component symbol to open the Configure dialog.

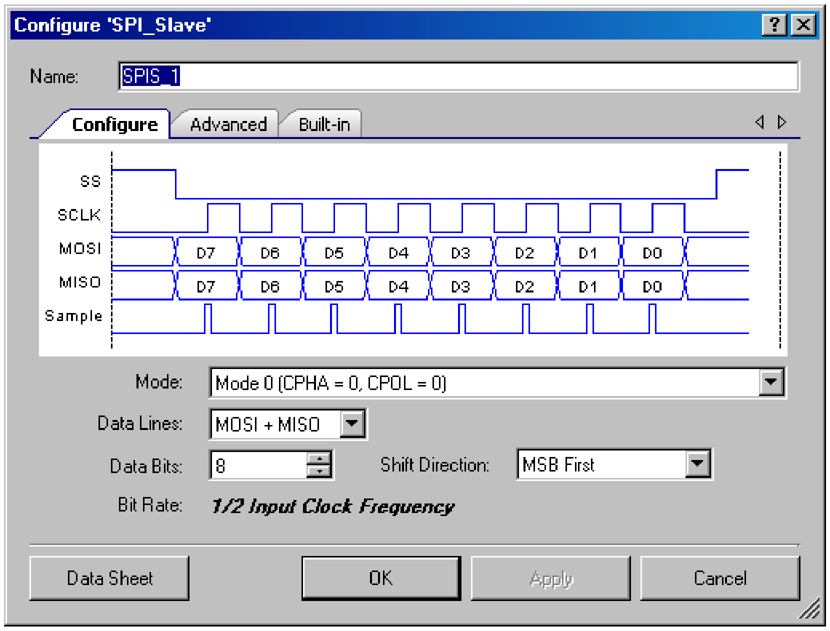
The following sections describe the SPI Slave parameters, and how they are configured using the Configure dialog. They also indicate whether the options are hardware or software.

Hardware vs. Software Options

Hardware configuration options change the way the project is synthesized and placed in the hardware. You must rebuild the hardware if you make changes to any of these options. Software configuration options do not affect synthesis or placement. When setting these parameters before build time you are setting their initial value which may be modified at any time with provided APIs. Hardware only parameters are marked with an asterisk (*).

Configure Tab

The **Configure** tab contains basic parameters required for every SPI component. As such, these parameters are the first ones visible to configure.



Name: SPIS_1

Mode: Mode 0 (CPHA = 0, CPOL = 0)

Data Lines: MOSI + MISO

Data Bits: 8

Shift Direction: MSB First

Bit Rate: *1/2 Input Clock Frequency*

Note The sample signal in the waveform is not an input or output of the system; it simply indicates when the data is sampled at the master and slave for the mode settings selected.

Mode *

The **Mode** parameter defines the desired clock phase and clock polarity mode used in the communication. The options are "Mode 0" (default), "Mode 1," "Mode 2," and "Mode 3." These modes are defined in the following table. Refer also to the Functional Description section of this data sheet.

Mode	CPHA	CPOL
0	0	0
1	0	1
2	1	0
3	1	1

Data Lines

The **Data Lines** parameter defines which interfacing is used for SPI communication – 4-wire (MOSI+MISO) or 3-wire (Bidirectional).

Data Bits *

The number of **Data Bits** defines the bit-width of a single transfer as transferred with the SPIS_ReadRxData() and SPIS_WriteTxData() APIs. The default number of bits is a single byte (8 bits). Any integer from 3 to 16 may be selected

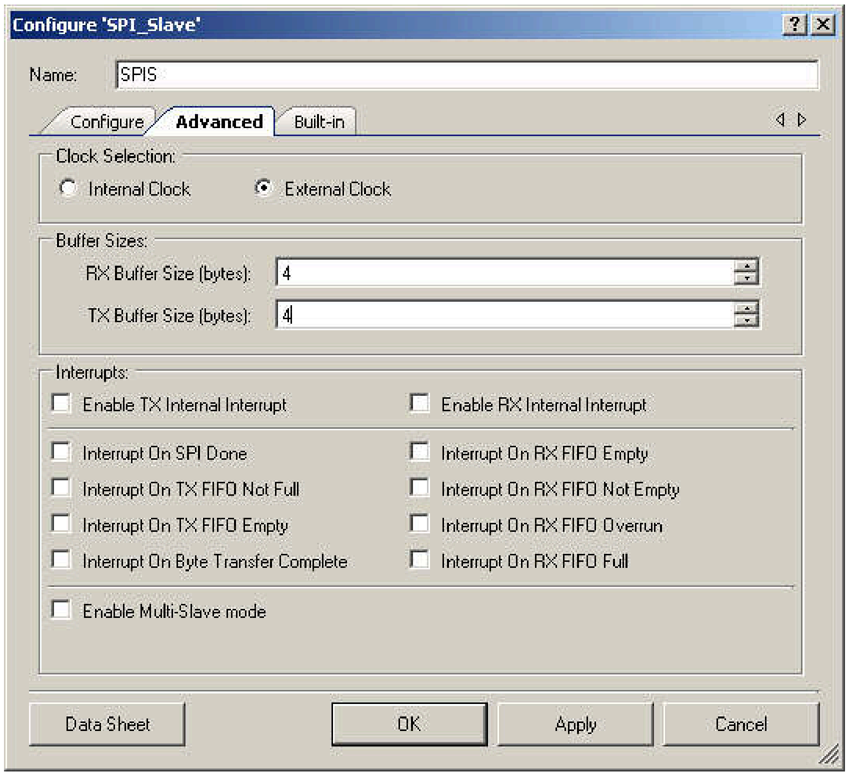
Shift Direction *

The **Shift Direction** parameter defines the direction the serial data is transmitted. When set to MSB_First, the most significant bit is transmitted first. This is implemented by shifting the data left. When set to LSB_First, the least significant bit is transmitted first. This is implemented by shifting the data right.

Bit Rate *

If the **Clock Selection** parameter (on the **Advanced** tab) is set to "Internal Clock," the **Bit Rate** parameter defines the SCLK speed in Hertz. The clock frequency of the internal clock will be 2x the SCLK rate. This parameter has no affect if the Clock Selection parameter is set to "External Clock."

Advanced Tab



Name: SPIS

Clock Selection:
- Internal Clock
- [✓ External Clock]

Buffer Sizes:
- RX Buffer Size (bytes): 4
- TX Buffer Size (bytes): 4

Interrupts:
- ☐ Enable TX Internal Interrupt
- ☐ Enable RX Internal Interrupt
- ☐ Interrupt On SPI Done
- ☐ Interrupt On RX FIFO Empty
- ☐ Interrupt On TX FIFO Not Full
- ☐ Interrupt On RX FIFO Not Empty
- ☐ Interrupt On TX FIFO Empty
- ☐ Interrupt On RX FIFO Overrun
- ☐ Interrupt On Byte Transfer Complete
- ☐ Interrupt On RX FIFO Full
- ☐ Enable Multi-Slave mode

Clock Selection

Specifies whether to use an Internal Clock or External Clock. Refer to the Clock Selection section later in this data sheet for more information.

RX Buffer Size *

The **RX Buffer Size** parameter defines the size (in bytes/words) of memory allocated for a The **TX Buffer Size** parameter defines the size (in bytes/words) of memory allocated for a circular data buffer. If this parameter is set to 1-4, the 4th byte/word of FIFO is implemented in the hardware. Values 1-3 are available only for compatibility with the previous versions; using them will cause an error icon to display that value is incorrect. All other values up to 255 will use the 4-byte/word FIFO and a memory array controlled by the supplied API.

TX Buffer Size *

The **TX Buffer Size** parameter defines the size (in bytes/words) of memory allocated for a circular data buffer. If this parameter is set to 1-4, the 4th byte/word of FIFO is implemented in the hardware. Values 1-3 are available only for compatibility with the previous versions; using them will cause an error icon to display that value is incorrect. All other values up to 255 will use the 4-byte/word FIFO and a memory array controlled by the supplied API.

Enable TX / RX Internal Interrupt

The **Enable TX / RX Internal Interrupt** options allow you to use the predefined TX and RX ISRs of the SPI Master component, or supply your own custom ISRs. If enabled, you may add your own code to these predefined ISRs if small changes are required. If the internal interrupt is deselected, you may supply an external interrupt component with custom code connected to the interrupt outputs of the SPI Master.

If the RX or TX buffer size is greater than 4, the component automatically sets the appropriate parameters, as the internal ISR is needed to handle transferring data from the hardware FIFO to the RX and/or TX buffer. At all times the interrupt output pins of the SPI master are visible and usable, outputting the same signal that goes to the internal interrupt. This output can then be used as a DMA request source or as a digital signal to be used as required in the programmable digital system.

Notes:

- When RX buffer size is greater than 4 bytes/words, the 'RX FIFO NOT EMPTY' interrupt is always enabled and cannot be disabled, because it causes incorrect buffer functionality.
- When TX buffer size is greater than 4 bytes/words, the 'TX FIFO NOT FULL' interrupt is always enabled and cannot be disabled, because it causes incorrect buffer functionality.
- For buffer sizes greater than 4 bytes/words, the SPI slave and global interrupt must be enabled for proper buffer handling.

Interrupts

The **Interrupts** selection parameters allow you to configure the internal events that are enabled to cause an interrupt. Interrupt generation is a masked OR of all of the enabled TX and RX status register bits. The bits chosen with these parameters define the mask implemented with the initial component configuration.

Enable Multi-Slave mode

This setting is used when current SPI Slave component is connected to the shared bus with other SPI Slave devices. MISO_OE output becomes visible on the component symbol. External BUF_OE component should be connected to MISO output in this mode. This mode allows you to turn MISO output to high impedance state when SS line is high. Multislave mode macro can be used to provide all necessary connections quickly.

Clock Selection

When the internal clock configuration is selected, PSoC Creator calculates the needed frequency and clock source, and generates the clocking resource needed for implementation. Otherwise, you must supply the clock component and calculate the required clock frequency. That frequency is at a minimum 2x the maximum bit-rate and SCLK frequency.

Note When setting the bitrate or external clock frequency value, make sure that this value can be provided by PSoC Creator using the current system clock frequency. Otherwise, a warning about the clock accuracy range will be generated while building the project. This warning will contain the real clock value set by PSoC Creator. Choose whether the system clock or component clock should be changed to fit the clocking system requirements and achieve an optimal value.

Placement

The SPI Slave component is placed into the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

Resources

	Digital Blocks					API Memory (Bytes)		
Resolution	**Datapaths**	**Macro cells**	**Status Registers**	**Control Registers**	**Counter7**	**Flash**	**RAM**	**Pins (per External I/O)**
SPI Slave 8-bit	1	16	2	1	1	1436	30	4
SPI Slave 16-bit	2	16	2	1	1	1571	38	4

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "SPIS_1" to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "SPIS."

Function	Description
SPIS_Start	Calls both SPIS_Init() and SPIS_Start(). Should be called the first time the component is started.
SPIS_Stop	Disable SPIS operation.
SPIS_EnableTxInt	Enables the internal TX interrupt irq.
SPIS_EnableRxInt	Enables the internal RX interrupt irq.
SPIS_DisableTxInt	Disables the internal TX interrupt irq.
SPIS_DisableRxInt	Disables the internal RX interrupt irq.
SPIS_SetTxInterruptMode	Configures the TX interrupt sources enabled.
SPIS_SetRxInterruptMode	Configures the RX interrupt sources enabled.
SPIS_ReadTxStatus	Returns the current state of the TX status register.
SPIS_ReadRxStatus	Returns the current state of the RX status register.
SPIS_WriteTxData	Places a byte/word in the transmit buffer which will be sent at the next available bus time.
SPIS_WriteTxDataZero	Places a byte/word in the shift register directly. This is required for SPI Modes 00 and 01.
SPIS_ReadRxData	Returns the next byte/word of received data available in the receive buffer.
SPIS_GetRxBufferSize	Returns the size (in bytes/words) of received data in the RX memory buffer.
SPIS_GetTxBufferSize	Returns the size (in bytes/words) of data waiting to transmit in the TX memory buffer.
SPIS_ClearRxBuffer	Clears the RX buffer memory array and RX FIFO of all received data.
SPIS_ClearTxBuffer	Clears the TX buffer memory array and TX FIFO of all transmit data.
SPIS_TxEnable	If configured for bidirectional mode, sets the SDAT inout to transmit.
SPIS_TxDisable	If configured for bidirectional mode, sets the SDAT inout to receive.
SPIS_PutArray	Places an array of data into the transmit buffer.
SPIS_ClearFIFO	Clears any received data from the RX hardware FIFO.

Function	Description
SPIS_Sleep	Prepare SPIS component for low power modes by calling SPIS_SaveConfig() and SPIS_Stop() functions.
SPIS_Wakeup	Restore and re-enable the SPIS component after waking from low power mode.
SPIS_Init	Initializes and restores the default SPIS configuration.
SPIS_Enable	Enables the SPIS to start operation.
SPIS_SaveConfig	Saves SPIS hardware configuration.
SPIS_RestoreConfig	Restores SPIS hardware configuration.

Global Variables

Function	Description
SPIS_initVar	Indicates whether the SPI Slave has been initialized. The variable is initialized to 0 and later set to 1 the first time SPIS_Start() is called. This allows the component to restart without reinitialization after the first call to the SPIS_Start() routine. If reinitialization of the component is required, then the SPIS_Init() function can be called before the SPIS_Start() or SPIS_Enable() function.
SPIS_txBufferWrite	Transmit buffer location of the last data written into the buffer by the API.
SPIS_txBufferRead	Transmit buffer location of the last data read from the buffer and transmitted by SPIS hardware.
SPIS_rxBufferWrite	Receive buffer location of the last data written into the buffer after received by SPIS hardware.
SPIS_rxBufferRead	Receive buffer location of the last data read from the buffer by the API.
SPIS_rxBufferFull	Indicates the software buffer has overflowed.
SPIS_RXBUFFER[]	Used to store received data.
SPIS_TXBUFFER[]	Used to store data for sending.

void SPIS_Start(void)

Description: This function calls both SPIS_Init() and SPIS_Start(). This should be called the first time the component is started.

Parameters: None

Return Value: None

Side Effects: None

void SPIS_Stop(void)

Description:	Disables the SPI Slave component. Has no affect on the SPIS operation.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_EnableTxInt (void)

Description:	Enables the internal TX interrupt irq.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_EnableRxInt (void)

Description:	Enables the internal RX interrupt irq.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_DisableTxInt (void)

Description:	Disables the internal TX interrupt irq
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_DisableRxInt (void)

Description:	Disables the internal RX interrupt irq
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_SetTxInterruptMode (uint8 intSrc)

Description: Configures the TX interrupt sources enabled

Parameters: uint8 intSrc: Bit-field containing the interrupts to enable.

Bit	Description
SPIS_INT_ON_SPI_DONE	Enable interrupt due to SPI done
SPIS_INT_ON_TX_EMPTY	Enable interrupt due to TX FIFO empty
SPIS_INT_ON_TX_NOT_FULL	Enable interrupt due to TX FIFO not full
SPIS_INT_ON_BYTE_COMP	Enable interrupt due to Byte/Word complete

Based on the bit-field arrangement of the TX status register. This value must be a combination of TX status register bit-masks defined in the header file.
For more information, refer also to the Defines section in this data sheet.

Return Value: None

Side Effects: None

void SPIS_SetRxInterruptMode (uint8 intSrc)

Description: Configures the RX interrupt sources enabled

Parameters: uint8 intSrc: Bit-field containing the interrupts to enable.

Bit	Description
SPIS_INT_ON_RX_EMPTY	Enable interrupt due to RX FIFO empty
SPIS_INT_ON_RX_NOT_EMPTY	Enable interrupt due to RX FIFO not empty
SPIS_INT_ON_RX_OVER	Enable interrupt due to RX Buf overrun
SPIS _INT_ON_RX_FULL	Enable interrupt due to RX FIFO full

Based on the bit-field arrangement of the RX status register. This value must be a combination of RX status register bit-masks defined in the header file.
For more information, refer to the Defines section in this data sheet.

Return Value: None

Side Effects: None

uint8 SPIS_ReadTxStatus (void)

Description: Returns the current state of the TX status register. For more information see the Status Register Bits section.

Parameters: None

Return Value: uint8: Current TX status register value

Bit	Description
SPIS_INT_ON_SPI_DONE	SPI done
SPIS_INT_ON_TX_EMPTY	TX FIFO empty
SPIS_INT_ON_TX_NOT_FULL	TX FIFO not full
SPIS_INT_ON_BYTE_COMP	Byte/Word complete

Side Effects: TX status register bits are clear on read.

uint8 SPIS_ReadRxStatus (void)

Description: Returns the current state of the RX status register. For more information see the Status Register Bits section.

Parameters: None

Return Value: uint8: Current RX status register value

Bit	Description
SPIS_INT_ON_RX_EMPTY	RX FIFO empty
SPIS_INT_ON_RX_NOT_EMPTY	RX FIFO not empty
SPIS_INT_ON_RX_OVER	RX Buf overrun
SPIS _INT_ON_RX_FULL	RX FIFO full

Side Effects: RX status register bits are clear on read.

void SPIS_WriteTxData (uint8/uint16 txData)

Description: Places a byte in the transmit buffer which will be sent at the next available bus time

Parameters: uint8/uint16: txData: The data value to send across the SPI

Return Value: None

Side Effects: Data may be placed in the memory buffer and will not be transmitted until all other previous data has been transmitted. This function blocks until there is space in the output memory buffer.

Clears the TX status register of the component.

void SPIS_WriteTxDataZero (uint8/uint16 txData)

Description:	Places a byte/word directly into the shift register for transmit which will be sent during the next clock phase from the master device
Parameters:	uint8/uint16: txData: The data value to send across the SPI
Return Value:	None
Side Effects:	Required for Modes 0 and 1 (CPHA == 0) where data must be in the shift register before the first clock edge. Firmware must control this if there is already data being shifted out and if there is more data in the FIFO. This routine should not to be used for Mode 2 or Mode 3 where CPHA == 1.

uint8/uint16 SPIS_ReadRxData (void)

Description:	Read the next byte of data received across the SPI.
Parameters:	None
Return Value:	uint8/uint16: The next byte/word of data read from the FIFO
Side Effects:	Will return invalid data if the FIFO is empty. Call SPIS_GetRxBufferSize() and if it returns a non-zero value then it is safe to call the SPIS_ReadRxData() function.

uint8 SPIS_GetRxBufferSize (void)

Description:	Returns the number of bytes/words of received data currently held in the RX buffer. • If the RX software buffer is disabled, this function returns 0 = FIFO empty or 1 = FIFO not empty. • If the RX software buffer is enabled, this function returns the size of data in the RX software buffer. FIFO data not included in this count.
Parameters:	None
Return Value:	uint8: Integer count of the number of bytes/words in the RX buffer.
Side Effects:	Clears the RX status register of the component.

uint8 SPIS_GetTxBufferSize (void)

Description:	Returns the number of bytes/words of data ready to transmit currently held in the TX buffer. • If TX software buffer is disabled, this function returns 0 = FIFO empty, 1 = FIFO not full, or 4 = FIFO full. • If the TX software buffer is enabled, this function returns the size of data in the TX software buffer. FIFO data not included in this count.
Parameters:	None
Return Value:	uint8: Integer count of the number of bytes/words in the TX buffer
Side Effects:	Clears the TX status register of the component.

void SPIS_ClearRxBuffer (void)

Description:	Clears the TX buffer memory array of data waiting to transmit. Clears the TX RAM buffer by setting both the read and write pointers to zero. Setting the pointers to zero indicates that there is no data to transmit. Thus, writing will resume at address 0, overwriting any data that may have remained in the RAM.
Parameters:	None
Return Value:	None
Side Effects:	Any received data not read from the RAM buffer and FIFO will be lost when overwritten by new data.

void SPIS_ClearTxBuffer (void)

Description:	Clears the memory array of all transmit data. Clear the TX RAM buffer by setting the read and write pointers both to zero. Setting the pointers to zero makes the system believe there is no data to read and writing will resume at address 0 overwriting any data that may have remained in the RAM.
Parameters:	None
Return Value:	None
Side Effects:	Will not clear data already placed in the TX FIFO. Any data not yet transmitted from the RAM buffer will be lost when overwritten by new data.

void SPIS_TxEnable (void)

Description:	If the SPI Slave is configured to use a single bidirectional pin then this will set the bidirectional pin to transmit.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_TxDisable (void)

Description:	If the SPI Slave is configured to use a single bidirectional pin then this will set the bidirectional pin to receive.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_PutArray (uint8 *buffer, uint8 byteCount)

Description:	Write available data from RAM/ROM to the TX buffer while space is available. Keep trying until all data is passed to the TX buffer. If using Mode 00 or 01, call the SPIS_WriteTxDataZero() function before calling the SPIS_PutArray() function.
Parameters:	uint8 *buffer: Pointer to the location in RAM containing the data to send
	uint8 byteCount: The number of bytes/words to move to the transmit buffer
Return Value:	None
Side Effects:	The system will stay in this function until all data has been transmitted to the buffer. This function is blocking if there is not enough room in the TX buffer. It may get locked in this loop if data is not being transmitted by the master and the TX buffer is full.

void SPIS_ClearFIFO (void)

Description:	Clears any received data from the RX FIFO
Parameters:	None
Return Value:	None
Side Effects:	Clear status register of the component.

void SPIS_Sleep (void)

Description:	Prepare SPIS component for low power modes by calling SPIS_SaveConfig() and SPIS_Stop() functions.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_Wakeup (void)

Description:	Prepare SPIS component to wake up from a low power mode. Calls SPIS_RestoreConfig() and SPIS_Enable() functions. Clears all data from RX buffer, TX buffer, and hardware FIFOs.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call SPIS_Init() because the SPIS_Start() routine calls this function and is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: When this function is called, it initializes all of the necessary parameters for execution. These include setting the initial interrupt mask, configuring the interrupt service routine, configuring the bit-counter parameters and clearing the FIFO and Status Register.

void SPIS_Enable(void)

Description: Enables SPIS to start operation. Starts the internal clock if so configured. If an external clock is configured it must be started separately prior to calling this API. The SPIS_Enable() function should be called before SPIS interrupts are enabled. This is because this function configures the interrupt sources and clears any pending interrupts from device configuration, and then enables the internal interrupts if so configured. A SPIS_Init() function must have been previously called.

Parameters: None

Return Value: None

Side Effects: None

void SPIS_SaveConfig (void)

Description: Saves SPIS hardware configuration prior to entering a low power mode.

Parameters: None

Return Value: None

Side Effects: None

void SPIS_RestoreConfig (void)

Description: Restores SPIS hardware configuration saved by the SPIS_SaveConfig() function after waking from a lower power mode.

Parameters: None

Return Value: None

Side Effects: If this function is called without first calling SPIS_SaveConfig() then in the following registers will be default values from the Configure dialog:

```
SPIS_STATUS_MASK_REG
SPIS_COUNTER_PERIOD_REG
```

Defines

SPIS_TX_INIT_INTERRUPTS_MASK

Defines the initial configuration of the interrupt sources chosen in the Configure dialog. This is a mask of the bits in the TX status register that have been enabled at configuration as sources for the interrupt. Refer to Status Register Bits section for bit-field details.

SPIS_RX_INIT_INTERRUPTS_MASK

Defines the initial configuration of the interrupt sources chosen in the Configure dialog. This is a mask of the bits in the RX status register that have been enabled at configuration as sources for the interrupt. Refer to Status Register Bits section for bit-field details.

Status Register Bits

Table 1 SPIS_TXSTATUS

Bits	7	6	5	4	3	2	1	0
Value	Interrupt	Byte/Word Complete	Unused	Unused	Unused	TX FIFO Empty	TX FIFO. Not Full	SPI Done

Table 2 SPIS_RXSTATUS

Bits	7	6	5	4	3	2	1	0
Value	Interrupt	RX FIFO Full	RX Buf. Overrun	RX FIFO Empty	RX FIFO Not Empty	Unused	Unused	Unused

- Byte/Word Complete: Set when a byte/word transmit has completed.
- RX FIFO Overrun: Set when RX Data has overrun the 4 byte/word FIFO without being moved to the RX buffer memory array (if one exists)
- RX FIFO Full: Set when the RX Data FIFO is full (does not indicate the RX buffer RAM array conditions).
- RX FIFO Empty: Set when the RX Data FIFO is empty (does not indicate the RX buffer RAM array conditions).
- RX FIFO Not Empty: Set when the RX Data FIFO is not empty. That is, at least one byte/word is in the RX FIFO (does not indicate the RX buffer RAM array conditions).
- TX FIFO Empty: Set when the TX Data FIFO is empty (does not indicate the TX buffer RAM array conditions).
- TX FIFO Not Full: Set when the TX Data FIFO is not full (does not indicate the TX buffer RAM array conditions).

- SPI Done: Set when all of the data in the transmit FIFO has been sent. This may be used to signal a transfer complete instead of using the byte/word complete status. (Set when Byte/Word Complete has been set and TX Data FIFO is empty.)

SPIS_TXBUFFERSIZE

Defines the amount of memory to allocate for the TX memory array buffer. This does not include the 4 bytes/words included in the FIFO. If this value is greater than 4, interrupts are implemented which move data to the FIFO from the circular memory buffer automatically.

SPIS_RXBUFFERSIZE

Defines the amount of memory to allocate for the RX memory array buffer. This does not include the 4 bytes/words included in the FIFO. If this value is greater than 4, interrupts are implemented which move data from the FIFO to the circular memory buffer automatically.

SPIS_DATAWIDTH

Defines the number of bits per data transfer chosen in the Configure dialog.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

Functional Description

Default Configuration

The default configuration for the SPIS is as an 8-bit SPIS with Mode 0 configuration.

Modes

To show the component's status bits and component signals values which they assume during data transmission 4 waveforms are shown. It is supposed that 5 data bytes are transmitted (4 bytes are written to the SPI Slave's TX buffer at the beginning of transmission and 5th – is thrown after 1st byte has been loaded into the A0 register). Rounded numbers represent the following events:

1 – Tx FIFO Empty has being cleared when 4 bytes are written to the Tx buffer;

2 – Tx FIFO Not Full has been cleared because Tx FIFO is full after 4 bytes written;

3 – Tx FIFO Not Full status is set when 1st byte has been loaded into the A0 register and cleared after 5th byte has been written to the free place into the Tx buffer.

4 – Slave Select line is set to Low state indicating beginning of the transmission.

5 – Tx FIFO Not Full status is set when 2nd bit is loaded to the A0. Rx Not Empty status is set when 1st received byte has been loaded into the Rx buffer. Byte/Word Complete is set as well.

6 – Tx FIFO Empty status is set at the moment when last byte to be sent has been loaded into the A0 register.(is not shown in details for simplification).

7 – the moment when 4th byte has been received so Rx FIFO Full is set along with Byte/Word Complete.

8 – Byte/Word Complete, SPI Done and Rx Overrun are set because all bytes have been transmitted and an attempt to load data into the full Rx buffer has been detected.

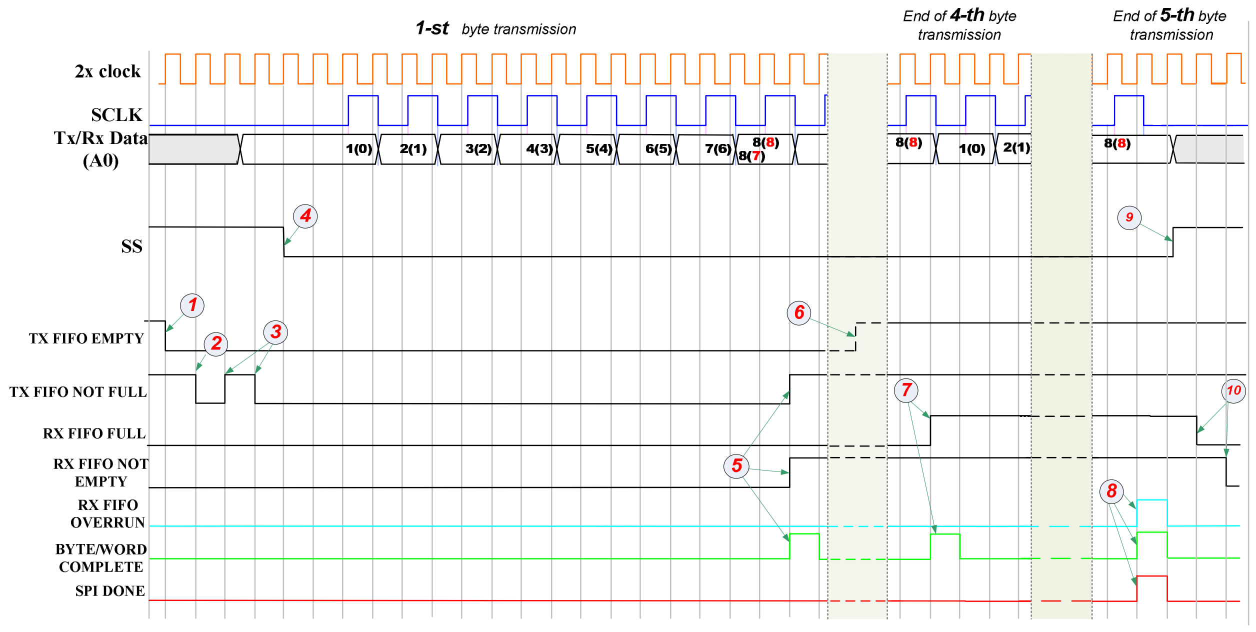
9 – SS line is set to High to indicate that transmission is complete.

10 – Rx FIFO Full is cleared when 1st byte has been read from the Rx buffer and Rx FIFO Empty is set when all of them have been read.

Note Because the same register is used to transmit and receive data the diagram section "Tx/Rx Data (A0)" contains two bit numbers in the following format : "Tx bit number (Rx bit number)".

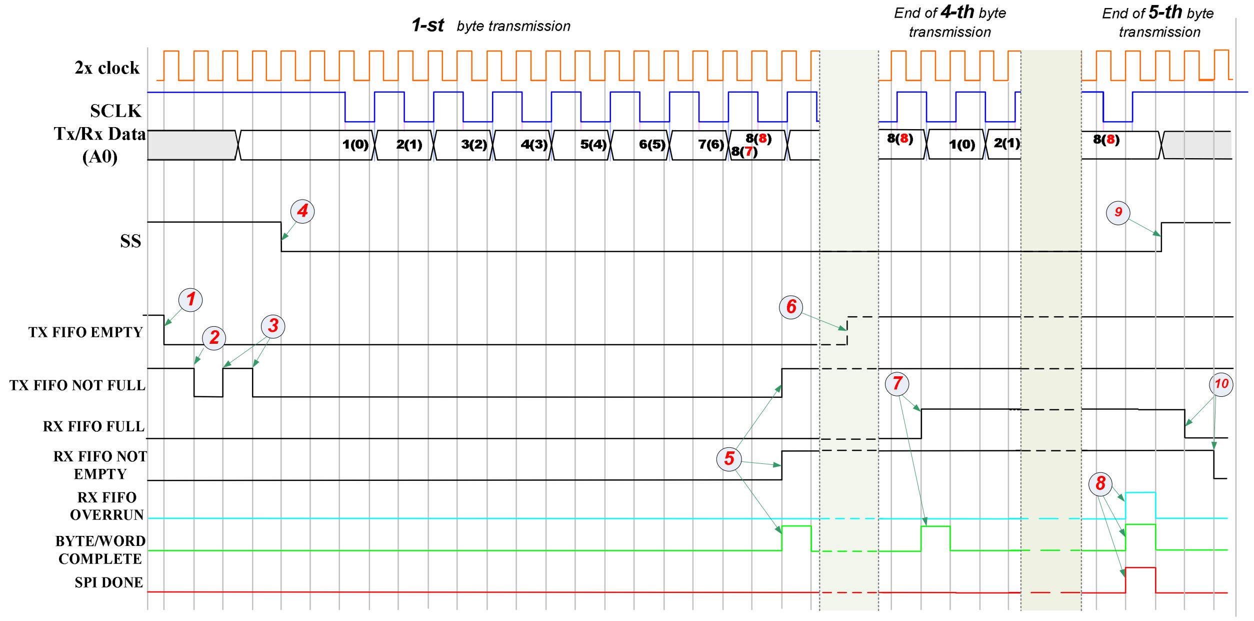
SPIS Mode: 0 (CPHA == 0, CPOL == 0)

Mode 0 has the following characteristics:



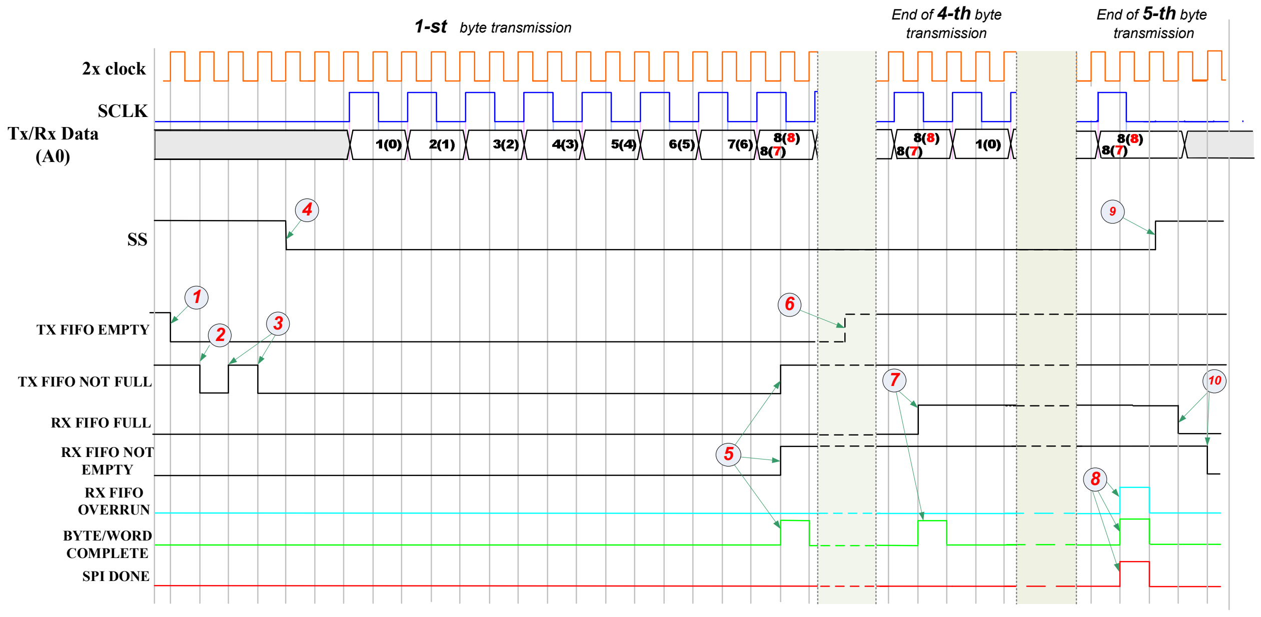
SPIS Mode: 1 (CPHA == 0, CPOL == 1)

Mode 1 has the following characteristics:



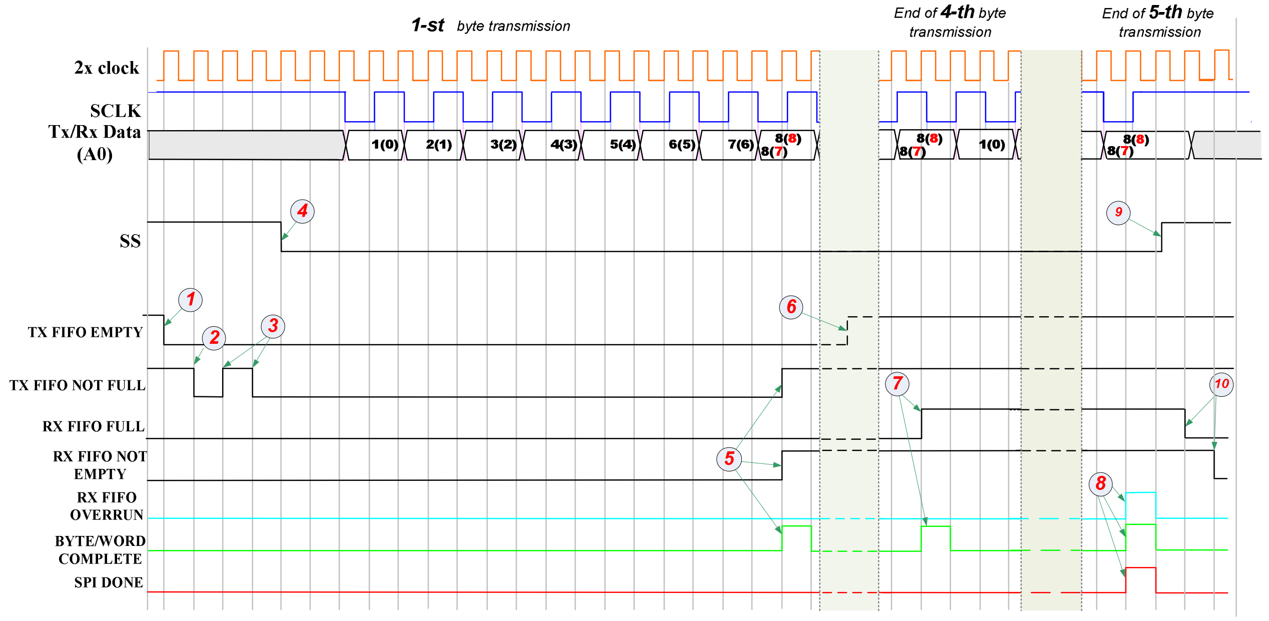
SPIS Mode: 2 (CPHA == 1, CPOL == 0)

Mode 2 has the following characteristics:



SPIS Mode: 3 (CPHA == 1, CPOL == 1)

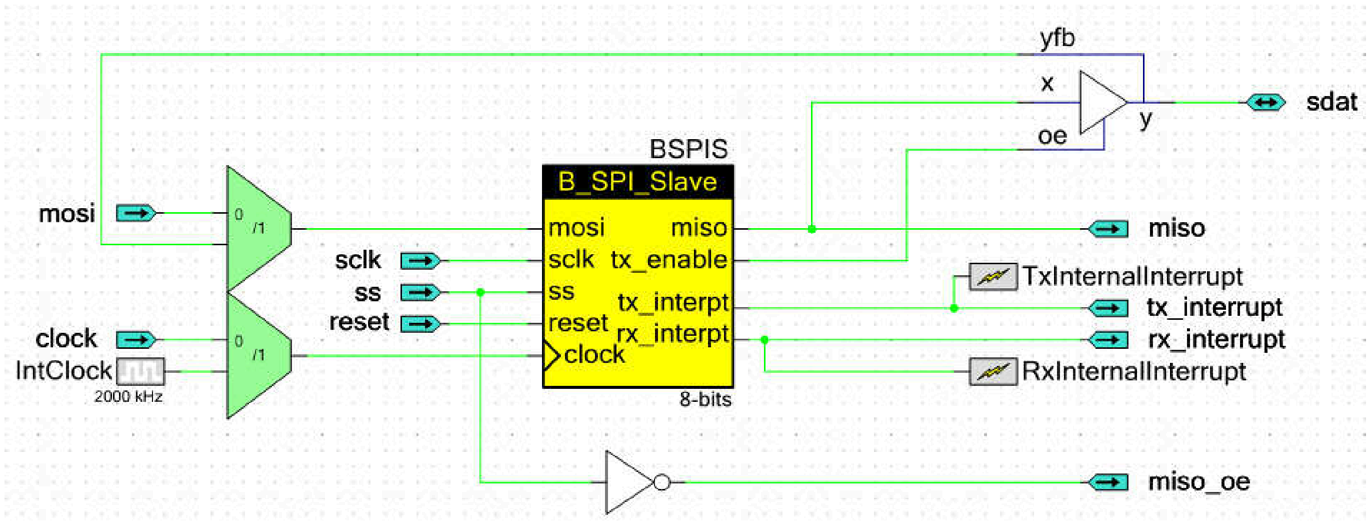
Mode 3 has the following characteristics:



Note Some SPI Master devices (such as the TotalPhase Aardvark I2C/SPI host adapter) drive the sclk output in a specific way. For the SPI Slave component to function properly with such devices in modes 1 and 3 (when CPOL =1), the sclk pin should be set to resistive pull up drive mode. Otherwise, it gives out corrupted data.

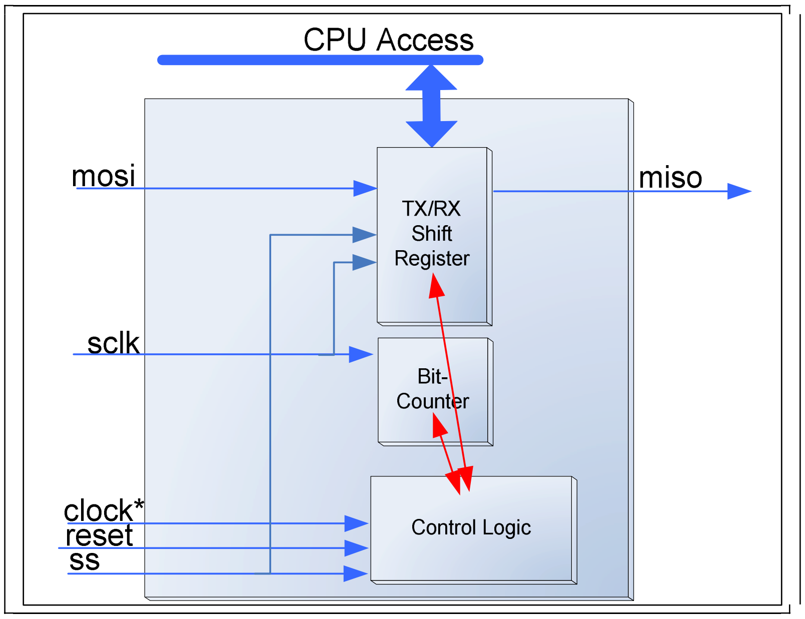
Block Diagram and Configuration

The SPIS is only available as a UDB configuration of blocks. The API is described above and the registers are described here to define the overall implementation of the SPIS.



The implementation is described in the following block diagram.

Figure 6. UDB Implementation



Registers

Status TX

The TX status register is a read only register which contains the various status bits defined for a given instance of the SPIS Component. Assuming that an instance of the SPI Slave is named "SPIS," the value of these registers is available with the SPIS_ReadTxStatus() function call. The interrupt output signal is generated from an ORing of the masked bit-fields within the TX status register. You can set the mask using the SPIS_SetTxInterruptMode() function call and upon receiving an interrupt you can retrieve the interrupt source by reading the TX Status register with the SPIS_ReadTxStatus () function call. The TX Status register is cleared on reading so the interrupt source is held until the SPIS_ReadTxStatus() function is called. All operations on the TX status register must use the following defines for the bit-fields as these bit-fields may be moved around within the TX status register at build time.

There are several bit-fields masks defined for the TX status registers. Any of these bit-fields may be included as an interrupt source. The bit-fields indicated with an * are configured as sticky bits in the TX status register, all other bits are configured as real-time indicators of status. The #defines are available in the generated header file (.h) as follows:

- SPIS_STS_SPI_DONE * – Defined as the bit-mask of the Status register bit "SPI Done."
- SPIS_STS_TX_FIFO_NOT_FULL – Defined as the bit-mask of the Status register bit "Transmit FIFO Empty."

- SPIS_STS_TX_FIFO_EMPTY – Defined as the bit-mask of the Status register bit "Transmit FIFO Empty."
- SPIS_STS_BYTE_COMPLETE * – Defined as the bit-mask of the Status register bit "Byte Complete."

Status RX

The RX status register is a read only register which contains the various status bits defined for the SPIS. The value of these registers is available with the SPIS_ReadRxStatus() and function call. The interrupt output signal is generated from an ORing of the masked bit-fields within the RX status register. You can set the mask using the SPIS_SetRxInterruptMode() function call and upon receiving an interrupt you can retrieve the interrupt source by reading the RX Status register with the SPIS_ReadRxStatus () function call. The RX Status register is clear on read so the interrupt source is held until the SPIS_ReadRxStatus() function is called. All operations on the RX status register must use the following defines for the bit-fields as these bit-fields may be moved around within the RX status register at build time.

There are several bit-fields masks defined for the RX status registers. Any of these bit-fields may be included as an interrupt source. The bit-fields indicated with an * are configured as sticky bits in the RX status register, all other bits are configured as real-time indicators of status.

The #defines are available in the generated header file (.h) as follows:

- SPIS_STS_RX_FIFO_FULL – Defined as the bit-mask of the Status register bit "Receive FIFO Full."
- SPIS_STS_RX_FIFO_NOT_EMPTY – Defined as the bit-mask of the Status register bit "Receive FIFO Not Empty."
- SPIS_STS_RX_FIFO_OVERRUN * – Defined as the bit-mask of the Status register bit "Receive FIFO Overrun."

TX Data

The TX data register contains the transmit data value to send. This is implemented as a FIFO in the SPIS. There is a software state machine to control data from the transmit memory buffer to handle much larger portions of data to be sent. All APIs dealing with transmitting the data must go through this register to place the data onto the bus. If there is data in this register and flow control indicates that data can be sent, then the data will be transmitted on the bus. As soon as this register (FIFO) is empty no more data will be transmitted on the bus until it is added to the FIFO. DMA may be setup to fill this FIFO when empty using the TXDATA_REG address defined in the header file.

RX Data

The RX data register contains the received data. This is implemented as a FIFO in the SPIS. There is a software state machine to control data movement from this receive FIFO into the memory buffer. Typically the RX interrupt will indicate that data has been received at which time that data has several routes to the firmware. DMA may be setup from this register to the memory array or the firmware may simply poll for the data at will. This will use the RXDATA_REG address defined in the header file.

Conditional Compilation Information

The SPIS requires only one conditional compile definition to handle the 8 or 16 bit Datapath configuration necessary to implement the expected NumberOfDataBits configuration it must support. It is required that the API conditionally compile Data Width defined in the parameter chosen. The API should never use these parameters directly but should use the define listed below.

- SPIS_DATAWIDTH – This defines how many data bits will make up a single "byte" transfer.

References

Not applicable

DC and AC Electrical Characteristics

The following values indicate expected performance and are based on initial characterization data.

Timing Characteristics “Maximum with Nominal Routing”

Parameter	Description	Config	Min	Typ	Max [1]	Units
f_{SCLK}	SCLK Frequency	Config 1 [2]			5	MHz
		Config 2 [3]			5	MHz
		Config 3 [4]			4	MHz
		Config 4 [5]			4	MHz
f_{clock}	Component clock frequency [6]	Config 1 [2]	2 * f_{SCLK}		10	MHz
		Config 2 [3]	2 * f_{SCLK}		10	MHz
		Config 3 [4]	2 * f_{SCLK}		8	MHz
		Config 4 [5]	2 * f_{SCLK}		8	MHz
t_{CKH}	SCLK High time			0.5		$1/f_{SCLK}$
t_{CKL}	SCLK Low time			0.5		$1/f_{SCLK}$
t_{SCLK_MISO}	SCLK to MISO Output Time				52	ns
t_{SCLK_SDAT} (Bidirect Mode only)	SCLK to SDAT Output Time				54	ns
t_{S_MOSI}	MOSI Input Setup Time		25			ns
t_{H_MOSI}	MOSI Input Hold Time			0		ns
t_{SS_SCLK}	SS Active to SCLK Active		-20		20	ns
t_{SCLK_SS}	SCLK Inactive to SS Inactive		-20		20	ns

[1] The component maximum component clock frequency is derived from t_{SCLK_MISO} in combination with the routing path delays of the SCLK input and the MISO output (Described later in this document). These “Nominal” numbers provide a maximum safe operating frequency of the component under nominal routing conditions. It is possible to run the component at higher clock frequencies, at which point you will need to validate the timing requirements with STA results.

[2] Config 1 options:

Data Lines:	MOSI+MISO
Data Bits:	8

[3] Config 2 options:

Data Lines:	MOSI+MISO
Data Bits:	16

[4] Config 3 options:

Data Lines:	Bidirectional
Data Bits:	8

[5] Config 4 options:

Data Lines:	Bidirectional
Data Bits:	16

[6] Component Clock is for status register only; it does not affect base functionality or bit-rate. Routing may limit the maximum frequency of this parameter; therefore the maximum is listed with nominal routing results.

Timing Characteristics “Maximum with All Routing”

Parameter	Description	Config	Min	Typ	Max [1]	Units
f_{SCLK}	SCLK Frequency	Config 1 [2]			4	MHz
		Config 2 [3]			4	MHz
		Config 3 [4]			2	MHz
		Config 4 [5]			2	MHz
f_{clock}	Component clock frequency [6]	Config 1 [2]	2 * f_{SCLK}		8	MHz
		Config 2 [3]	2 * f_{SCLK}		8	MHz
		Config 3 [4]	2 * f_{SCLK}		4	MHz
		Config 4 [5]	2 * f_{SCLK}		4	MHz
t_{CKH}	SCLK High time			0.5		$1/f_{SCLK}$
t_{CKL}	SCLK Low time			0.5		$1/f_{SCLK}$
t_{SCLK_MISO}	SCLK to MISO Output Time				52	ns
t_{SCLK_SDAT} (Bidirect Mode only)	SCLK to SDAT Output Time				54	ns
t_{S_MOSI}	MOSI Input Setup Time (to SCLK)		25			ns
t_{H_MOSI}	MOSI Input Hold Time (from SCLK)			0		ns
t_{SS_SCLK}	SS Active to SCLK Active		-20		20	ns
t_{SCLK_SS}	SCLK Inactive to SS Inactive		-20		20	ns

[1] Maximum for “All Routing” is calculated by <nominal>/2 rounded to the nearest integer. This value provides a basis for the user to not have to worry about meeting timing if they are running at or below this component frequency.

[2] Config 1 options:

Data Lines:	MOSI+MISO
Data Bits:	8

[3] Config 2 options:

Data Lines:	MOSI+MISO
Data Bits:	16

[4] Config 3 options:

Data Lines:	Bidirectional
Data Bits:	8

[5] Config 4 options:

Data Lines:	Bidirectional
Data Bits:	16

[6] Component Clock is for status register only; it does not affect base functionality or bit-rate. Routing may limit the maximum frequency of this parameter; therefore the maximum is listed with nominal routing results.

Figure 7. Mode CPHA=0 Timing Diagram

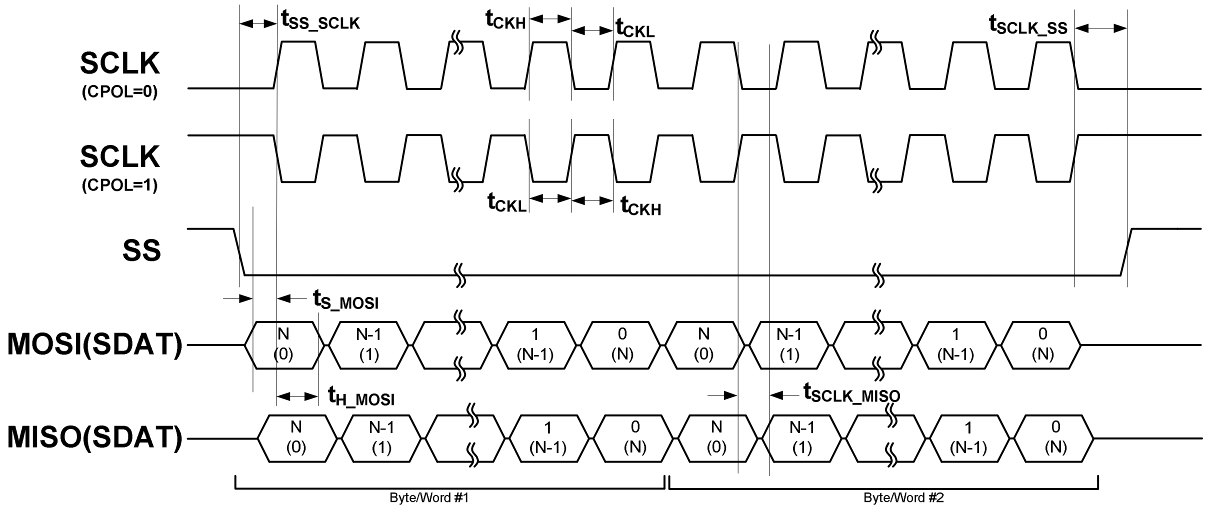
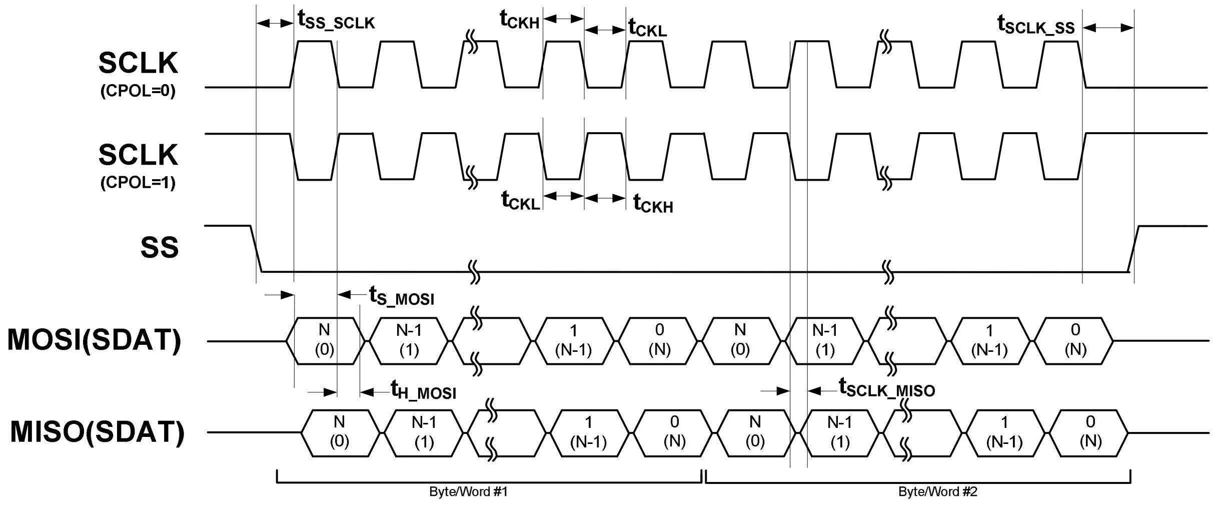


Figure 8. Mode CPHA=1 Timing Diagram

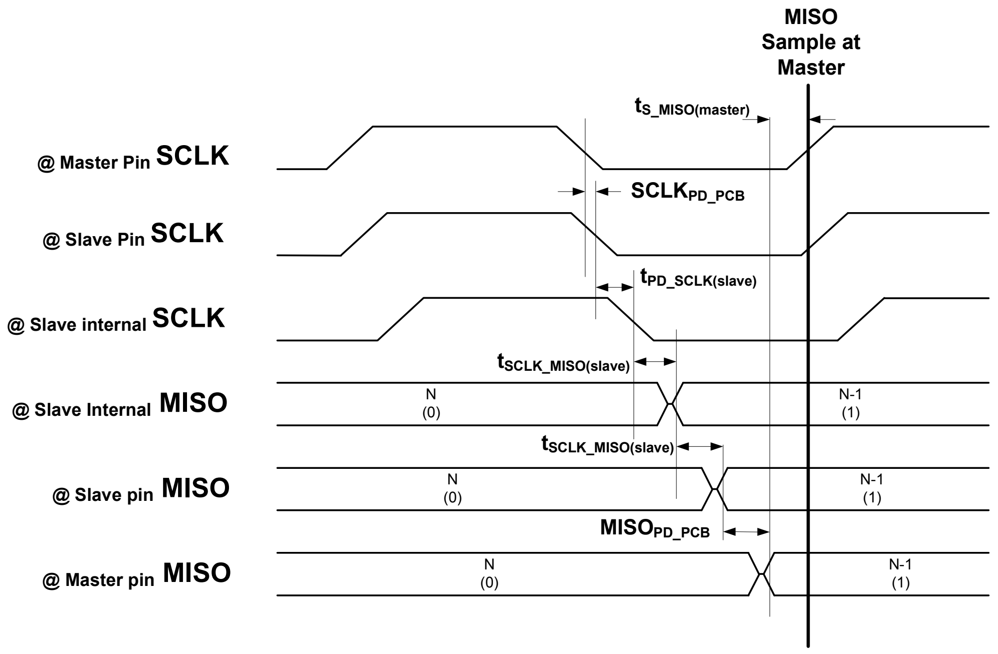


How to Use STA Results for Characteristics Data

Nominal route maximums are gathered through multiple test passes with Static Timing Analysis (STA). The maximums may be calculated for your designs using the STA results with the following mechanisms

f_{SCLK} The Maximum frequency of SCLK (or the Maximum Bit-Rate) is not provided directly in the STA. However the data provided in the STA results indicate some of the internal logic timing constraints. To calculate the maximum bit-rate several factors must be taken into account. Board Layout and slave communication device specs are needed to fully understand the maximum. The main limiting factor in this parameter is the round trip path delay from the falling edge of SCLK at the pin of the master, to the slave and the path delay of the MISO output of the slave back to the master.

Figure 9. Calculating Maximum f_{SCLK} Frequency



In this case the component must meet the setup time of MISO at the Master using the equation below:

$$t_{RT\_PD} < 1 / \{ [½ * f_{SCLK}] – t_{PD\_SCLK(master)} - t_{S\_MISO(master)} \}$$

-- OR --

$$f_{SCLK} < 1 / \{2 * [T_{RT\_PD} + t_{PD\_SCLK(master)} + t_{S\_MISO(master)}] \}$$

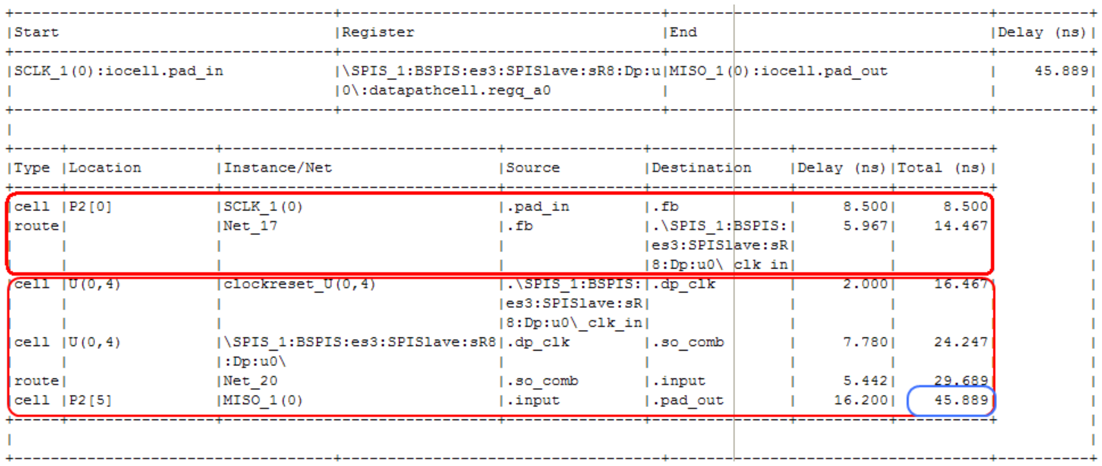
Where $t_{PD\_SCLK(master)} + t_{S\_MISO(Master)}$ must come from the Master Device Datasheet. And t_{RT_PD} is defined as:

$$t_{RT\_PD} = [SCLK_{PD\_PCB} + t_{PD\_SCLK(slave)} + t_{SCLK\_MISO(slave)} + MISO_{PD\_PCB}]$$

and:

$SCLK_{PD\_PCB}$ is the PCB Path delay of SCLK from the pin of the master component to the pin of the slave component.

$T_{PD\_SCLK(Slave)}$ is the path delay of the input SCLK to the internal logic; $T_{SCLK\_MISO(slave)}$ is the SCLK pin to internal logic path delay of the slave component; And $t_{PD\_MISO(slave)}$ is the path delay of the internal MISO to the pin. The easiest way to find the values for these three parameters is to take the combined path as directly listed in the STA results, shown in the figure below:



```
+-----------------------------------+-----------------------------------+-----------------------------------+----------+
|Start                              |Register                           |End                                |Delay (ns)|
+-----------------------------------+-----------------------------------+-----------------------------------+----------+
|SCLK_1(0):iocell.pad_in            |\SPIS_1:BSPIS:es3:SPISlave:sR8:Dp:u|MISO_1(0):iocell.pad_out           |    45.889|
|                                   |0\:datapathcell.regq_a0            |                                   |          |
+-----------------------------------+-----------------------------------+-----------------------------------+----------+
|                                                                                                                      |
+-----+----------------+------------------------------+---------------+---------------+----------+----------+          |
|Type |Location        |Instance/Net                  |Source         |Destination    |Delay (ns)|Total (ns)|          |
+-----+----------------+------------------------------+---------------+---------------+----------+----------+          |
|cell |P2[0]           |SCLK_1(0)                     |.pad_in        |.fb            |     8.500|     8.500|          |
|route|                |Net_17                        |.fb            |.\SPIS_1:BSPIS:|     5.967|    14.467|          |
|     |                |                              |               |es3:SPISlave:sR|          |          |          |
|     |                |                              |               |8:Dp:u0\_clk_in|          |          |          |
|cell |U(0,4)          |clockreset_U(0,4)             |.\SPIS_1:BSPIS:|.dp_clk        |     2.000|    16.467|          |
|     |                |                              |es3:SPISlave:sR|               |          |          |          |
|     |                |                              |8:Dp:u0\_clk_in|               |          |          |          |
|cell |U(0,4)          |\SPIS_1:BSPIS:es3:SPISlave:sR8|.dp_clk        |.so_comb       |     7.780|    24.247|          |
|     |                |:Dp:u0\                       |               |               |          |          |          |
|route|                |Net_20                        |.so_comb       |.input         |     5.442|    29.689|          |
|cell |P2[5]           |MISO_1(0)                     |.input         |.pad_out       |    16.200|    45.889|          |
+-----+----------------+------------------------------+---------------+---------------+----------+----------+          |
|                                                                                                                      |
+-----------------------------------+-----------------------------------+-----------------------------------+----------+
```

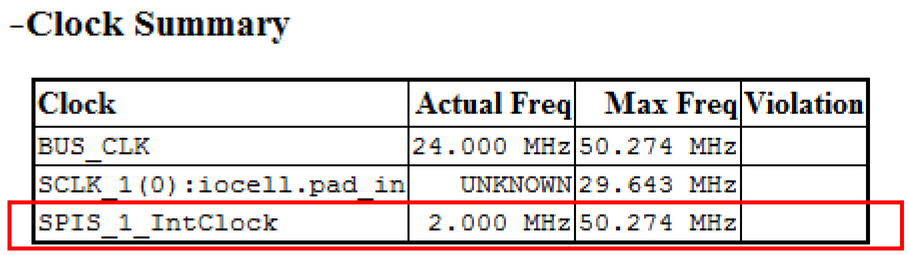
Where $T_{PD\_SCLK(Slave)}$ is the first two numbers, $T_{SCLK\_MISO(slave)}$ is the second two numbers, and $t_{PD\_MISO(slave)}$ is the last two numbers. The full path of the three parameters is 45.889ns.

$MISO_{PD\_PCB}$ is the PCB path delay of the MISO from the pin of the slave component to the pin of the master component

The final equation that will provide the maximum frequency of SCLK and therefore the maximum bit-rate is:

$$f_{SCLK}\text{ (Max.)} = 1 / \{ 2^{*} [SCLK_{PD\_PCB} + t_{PD\_SCLK(slave)} + t_{SCLK\_MISO(slave)} + MISO_{PD\_PCB} + t_{PD\_SCLK(master)} + t_{S\_MISO(master)}] \}$$

f_{clock} Maximum Component Clock Frequency is provided in Timing results in the clock summary as the IntClock (if internal clock is selected) or the named external clock. An example of the internal clock limitations from the _timing.html file is below:



–Clock Summary

Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	50.274 MHz	
SCLK_1(0):iocell.pad_in	UNKNOWN	29.643 MHz	
SPIS_1_IntClock	2.000 MHz	50.274 MHz	

t_{CKH} The SPI Slave component requires a 50% duty cycle SCLK

t_{CKL} The SPI Slave component requires a 50% duty cycle SCLK

t_{S_MOSI} To meet the setup time of the internal logic MOSI must be valid at the pin, before SCLK is valid at the pin, by this amount of time

t_{H_MOSI} To meet the hold time of the internal logic MOSI must be valid at the pin, after SCLK is valid at the pin, by this amount of time.

t_{SS_SCLK} To meet the internal functionality of the block. Slave Select (SS) must be valid at the pin before SCLK is valid at the pin by this parameter.

t_{SCLK_SS} Maximum - To meet the internal functionality of the block. Slave Select (SS) must be valid at the pin after the last falling edge of SCLK at the pin by this parameter.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
2.10	Data Bits range is changed from 2-16 bits to 3-16	Changes related to status synchronization issues fixed in current version
	"Byte transfer complete" checkbox name is changed to the "Byte/Word transfer complete"	To fit the real meaning
	Added characterization data to datasheet	
	Minor datasheet edits and updates	
2.0.a	Moved component into subfolders of the component catalog.	
	Minor datasheet edits and updates	
2.0	Added SPIS_Sleep()/SPIS_Wakeup() and SPIS_Init()/SPIS_Enable() APIs.	To support low power modes, as well as to provide common interfaces to separate control of initialization and enabling of most components.
	Number and positions of component outputs have been changed: • The reset input was added; • The interrupt output was removed; rx_interrupt, tx_interrupt outputs were added instead.	PSoC 3 ES3 reset functionality was added. Two status interrupt registers (Tx and Rx) are now presented instead of one shared. The changes must be taken into account to prevent binding errors when migrating from previous SPI versions
	Removed SPIS_EnableInt(), SPIS_DisableInt(), SPIS_SetInterruptMode(), and SPIS_ReadStatus() APIs. Added SPIS_EnableTxInt(), SPIS_EnableRxInt(), SPIS_DisableTxInt(), SPIS_DisableRxInt(), SPIS_SetTxInterruptMode(), SPIS_SetRxInterruptMode(), SPIS_ReadTxStatus(), SPIS_ReadRxStatus() APIs.	The removed APIs are obsolete because the component now contains RX and TX interrupts instead of one shared interrupt. Also updated the interrupt handler implementation for TX and RX Buffer.
	Renamed SPIS_ReadByte(), SPIS_WriteByte(), and SPIS_WriteByteZero() APIs to SPIS_ReadRxData(), SPIS_WriteTxData(), SPIS_WriteTxDataZero().	Clarifies the APIs and how they should be used.

Version	Description of Changes	Reason for Changes / Impact
The following changes were made to the base SPI Slave component B_SPI_Slave_v2_0, which is implemented using Verilog:		
	B_SPI_Slave_v2_0 now contains two separate implementations for ES2 and ES3 silicon. One datapath is used for 8-bit SPI for Tx and Rx in ES3 silicon instead of two in ES2.	Requirement that all components support ES2 and ES3 silicon. Use of ES3 feature updates is a requirement and this helps optimize resource usage in ES3. .
	Changes in ES2 support implementation: Two statusi registers are now presented (status are separate on Tx and Rx) instead of using one common status register for both.	This provides correct software buffer functionality.
	'BidirectMode' boolean parameter is added to base component (Verilog implementation). Control Register with 'clock' input and SYNC mode bit is now selected to drive 'tx_enable' output for ES3 silicon. Control Register w/o clock input drives 'tx_enable' when ES2 silicon is selected. Bufoe component is used on component schematic to support Bidirectional Mode. MOSI output of base component is connected to bufoe 'x' input. 'yfb' is connected to 'miso' input. Bufoe 'y' output is connected to 'sdat' output terminal.	Added Bidirectional Mode support to the component
	Four udb_clock_enable components are added to Verilog implementation with sync = \`TRUE\` parameter. One of them has sync = \`TRUE\` (status register clock), other three have sync = \`FALSE\`	New requirements for all clocks used in Verilog to indicate functionality so the tool can support synchronization and Static Timing Analysis.
	Rx datapath configuration changed 'FIFO FAST' option is set to 'DP' instead of 'BUS'	Fixes a defect in previous versions of the component where there was a timing window affecting correct component capture of data.
	Additional logic to Verilog implementation to change the moments when SPI DONE and BYTE COMPLETE status are generated.	Fixes a defect in previous versions of the component where SPI DONE is sometimes not generated even after communication is complete.
	Maximum Bit Rate value is changed to 10 Mbps	Bit Rate value more than 10 Mbps is not supported (verified during component characterization)
	Description of the Bidirectional Mode is added	Data sheet defect fixed
	Reset input description now contains the note about ES2 silicon incompatibility	Data sheet defect fixed
	Timing correlation diagram between SS and SCLk signals is changed	Data sheet defect fixed
	Sample firmware source code is removed	Reference to the component example project is added instead

Version	Description of Changes	Reason for Changes / Impact
	SPI Modes diagrams were changed (Tx and Rx FIFO status values were added)	Data sheet defect fixed