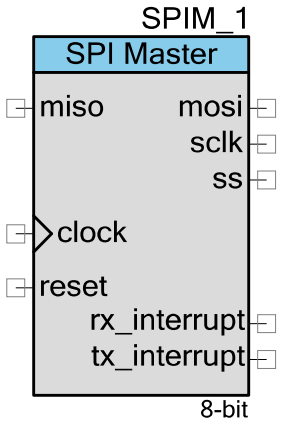
Serial Peripheral Interface (SPI) Master

2.10



Features

- 3- to 16-bit data width
- 4 SPI operating modes
- Bit Rate up to 9 Mbps *

General Description

The SPI Master component provides an industry-standard, 4-wire master SPI interface. It can also provide a 3-wire (bidirectional) SPI interface. Both interfaces support all four SPI operating modes, allowing communication with any SPI slave device. In addition to the standard 8-bit word length, the SPI Master 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 Master

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

The SPI Slave component should be used in instances requiring the PSoC device to communicate with an SPI Master device. The Shift Register component should be used in situations where its low level flexibility provides hardware capabilities not available in the SPI Master 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 component. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

miso – 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)

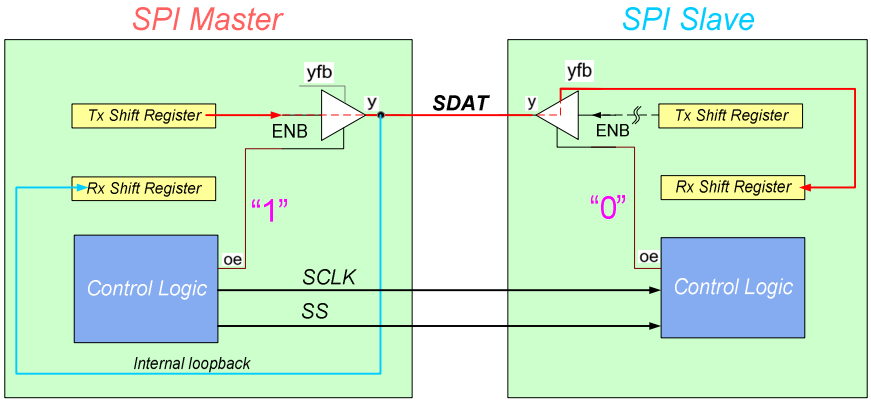
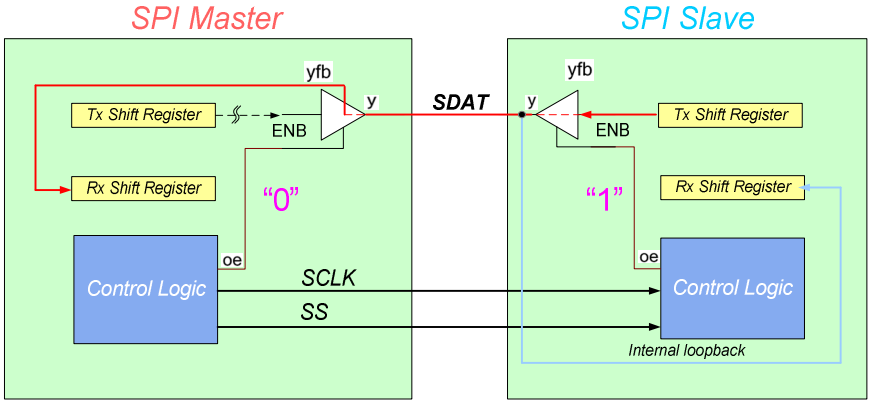


Figure 2: SPI Bidirectional Mode (data transmission from Slave to Master)



clock – Input *

The clock input defines the bit-rate of the serial communication. The bit-rate is 1/2 the input clock frequency.

The clock input is visible when the **Clock Selection** parameter is set to "External." If visible, this input must be connected. If "Internal Clock" is used, then you define the desired data bit-rate and the clock needed is solved and provided by PSoC Creator.

reset – Input

Resets the SPI state machine to the idle state. 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.

mosi – Output *

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

sclk– Output

The sclk output carries the Serial Clock (SCLK) signal. It provides the master synchronization clock output to the slave device(s) on the bus.

ss – Output

The ss output is hardware controlled. It carries the Slave Select (SS) signal to the slave device(s) on the bus. It is possible to connect a digital De-Multiplexer to handle multiple slave devices, or to have a completely firmware controlled SS. See the following figures for examples.

Figure 3: Slave Select Output to De-Multiplexer

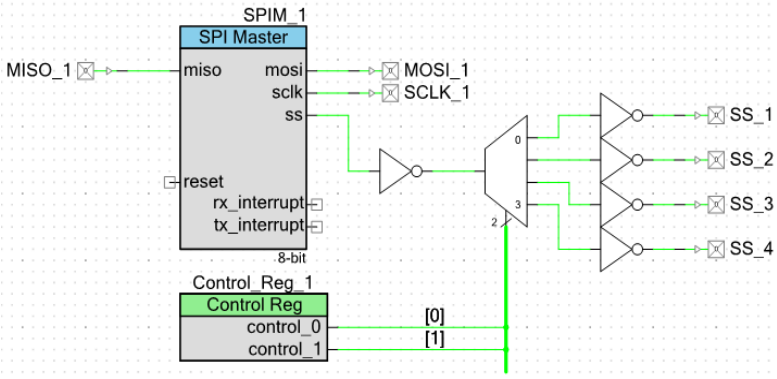
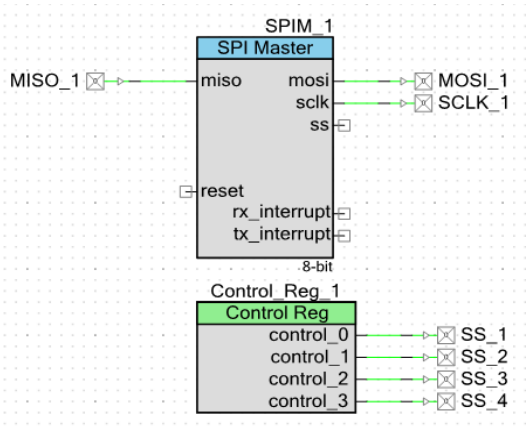
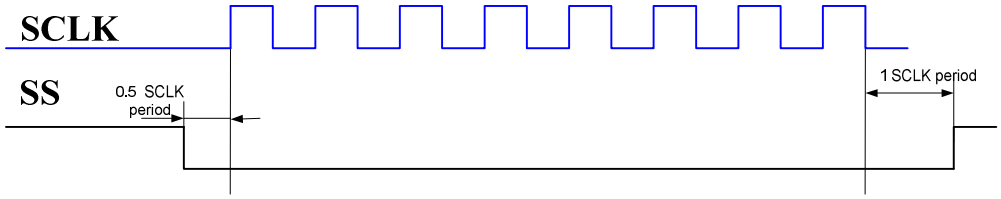


Figure 4: Firmware Controlled Slave Select(s)

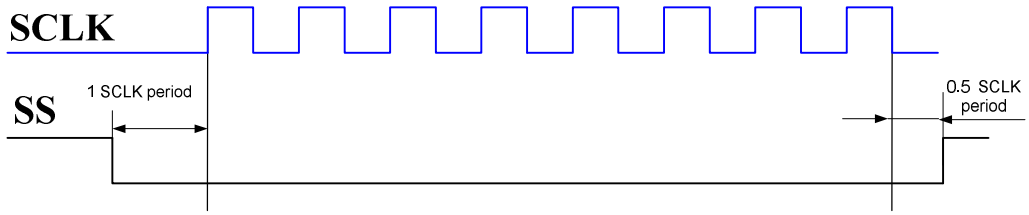


The following diagram shows the timing correlation between SS and SCLK:

CPHA =0:



CPHA =1:



Note SS is not set to "high" during a multi-byte/word transmission if the "SPI Done" condition was not generated.

rx_interrupt – Output

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

tx_interrupt – Output

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

Schematic Macro Information

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

Figure 5: 4-Wire (Full Duplex) Interfacing Schematic Macro

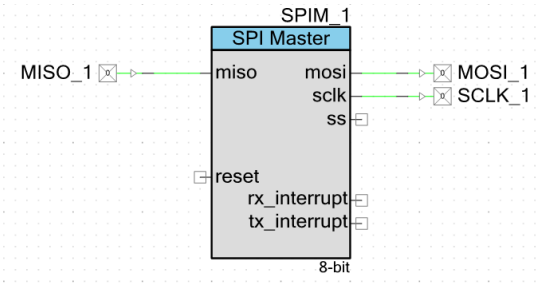
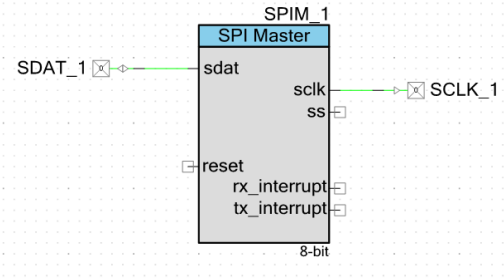


Figure 6: 3-Wire (Bidirectional) Interfacing Schematic Macro



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

Parameters and Setup

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

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

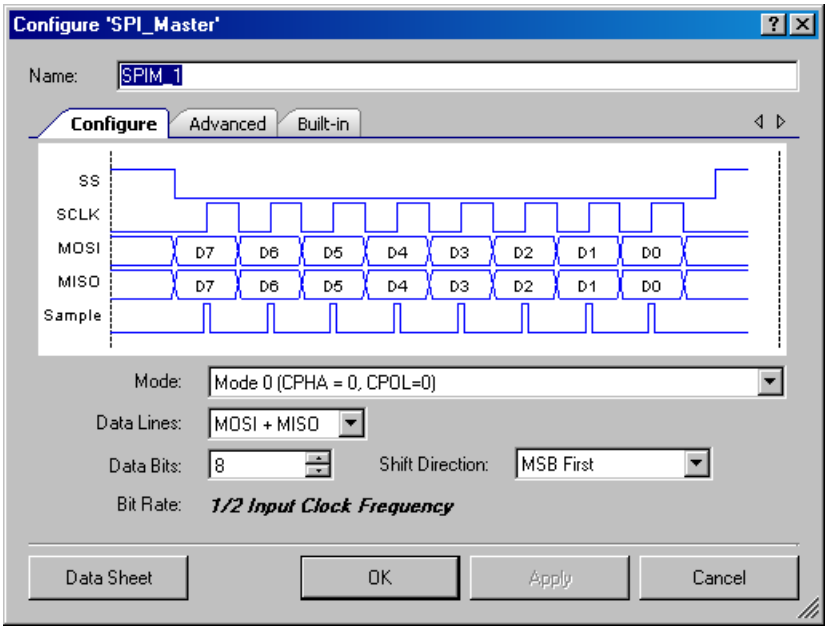
Hardware vs. Software Options

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.



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," "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 interface 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 SPIM_ReadRxData() and SPIM_WriteTxData() functions. The default number of bits is a single byte (8 bits). Any integer from 3 to 16 may be selected.

Shift Direction *

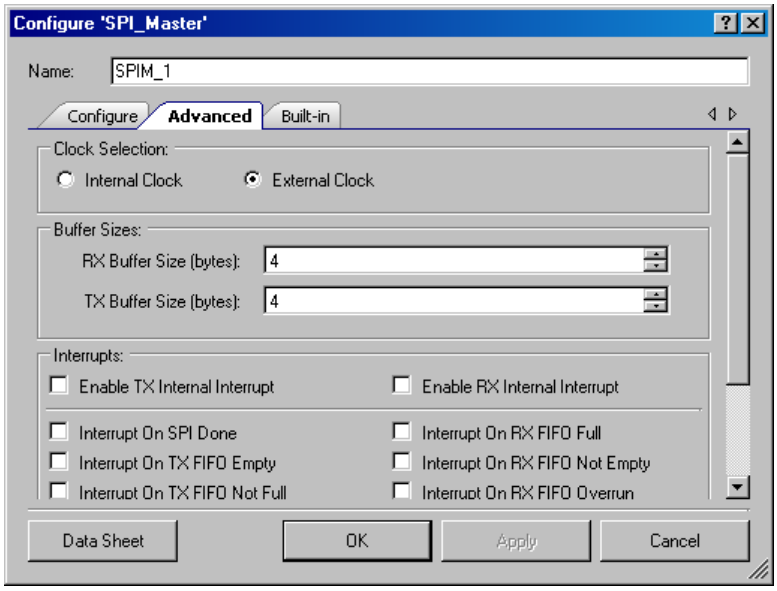
The **Shift Direction** parameter defines the direction the serial data is transmitted. When set to MSB_First, the Most Significant bit is transmitted first. This is implemented by shifting the data left. When set to LSB_First, the Least Significant bit is transmitted first. This is implemented by shifting 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

The **Advanced** tab contains parameters that provide additional functionality.



Clock Selection *

The **Clock Selection** parameter allows the user to choose between an internally configured clock or an externally configured clock for data rate and SCLK generation. When set to "Internal Clock," the required clock frequency is calculated and configured by PSoC Creator based on the **Bit Rate** parameter. When set to "External Clock," the component does not control the data rate

but will display the expected bit rate based on the user-connected clock source. If this parameter is set to "Internal Clock" then the clock input is not visible on the symbol.

Note When setting the bit rate 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 actual 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.

RX Buffer Size *

The **RX 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 bytes/words 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.

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 2x the desired bit-rate and SCLK frequency.

Note When setting the bit rate 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 Master component is placed into the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

Resources

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
SPI Master 8-bit	1	12	2	1	1	849	19	4
SPI Master 16-bit	2	12	2	1	1	926	21	4

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software at runtime. 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 "SPIM_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 the instance. For readability, the instance name used in the following table is "SPIM".

Function	Description
SPIM_Start	Calls both SPIM_Init() and SPIM_Start(). Should be called the first time the component is started.
SPIM_Stop	Disables SPIM operation.
SPIM_EnableTxInt	Enables the internal TX interrupt irq.
SPIM_EnableRxInt	Enables the internal RX interrupt irq.
SPIM_DisableTxInt	Disables the internal TX interrupt irq.
SPIM_DisableRxInt	Disables the internal RX interrupt irq.
SPIM_SetTxInterruptMode	Configures the TX interrupt sources enabled.
SPIM_SetRxInterruptMode	Configures the RX interrupt sources enabled.
SPIM_ReadTxStatus	Returns the current state of the TX status register.
SPIM_ReadRxStatus	Returns the current state of the RX status register.
SPIM_WriteTxData	Places a byte/word in the transmit buffer which will be sent at the next available bus time.
SPIM_ReadRxData	Returns the next byte/word of received data available in the receive buffer.
SPIM_GetRxBufferSize	Returns the size (in bytes/words) of received data in the RX memory buffer.
SPIM_GetTxBufferSize	Returns the size (in bytes/words) of data waiting to transmit in the TX memory buffer.
SPIM_ClearRxBuffer	Clears the RX buffer memory array and RX FIFO of all received data.
SPIM_ClearTxBuffer	Clears the TX buffer memory array and TX FIFO of all transmit data.
SPIM_TxEnable	If configured for bidirectional mode, sets the SDAT inout to transmit.
SPIM_TxDisable	If configured for bidirectional mode, sets the SDAT inout to receive.
SPIM_PutArray	Places an array of data into the transmit buffer.
SPIM_ClearFIFO	Clears any received data from the RX hardware FIFO.

Function	Description
SPIM_Sleep	Prepare SPIM component for low power modes by calling SPIM_SaveConfig() and SPIM_Stop() functions.
SPIM_Wakeup	Restores and re-enables the SPIM component after waking from low power mode.
SPIM_Init	Initializes and restores the default SPIM configuration.
SPIM_Enable	Enables the SPIM to start operation.
SPIM_SaveConfig	Saves SPIM hardware configuration.
SPIM_RestoreConfig	Restores SPIM hardware configuration.

Global Variables

Variable	Description
SPIM_initVar	Indicates whether the SPI Master has been initialized. The variable is initialized to 0 and later set to 1 the first time SPIM_Start() is called. This allows the component to restart without reinitialization after the first call to the SPIM_Start() routine. If reinitialization of the component is required, then the SPIM_Init() function can be called before the SPIM_Start() or SPIM_Enable() function.
SPIM_txBufferWrite	Transmit buffer location of the last data written into the buffer by the API.
SPIM_txBufferRead	Transmit buffer location of the last data read from the buffer and transmitted by SPIM hardware.
SPIM_rxBufferWrite	Receive buffer location of the last data written into the buffer after received by SPIM hardware.
SPIM_rxBufferRead	Receive buffer location of the last data read from the buffer by the API.
SPIM_rxBufferFull	Indicates the software buffer has overflowed.
SPIM_RXBUFFER[]	Used to store received data.
SPIM_TXBUFFER[]	Used to store data for sending.

void SPIM_Start(void)

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

Parameters: None

Return Value: None

Side Effects: None

void SPIM_Stop(void)

Description: Disables SPIM operation by disabling the internal clock and internal interrupts, if so configured.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_EnableTxInt (void)

Description: Enables the internal TX interrupt irq.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_EnableRxInt (void)

Description: Enables the internal RX interrupt irq.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_DisableTxInt (void)

Description: Disables the internal TX interrupt irq.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_DisableRxInt (void)

Description: Disables the internal RX interrupt irq.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_SetTxInterruptMode (uint8 intSrc)

Description: Configure which status bits trigger an interrupt event.

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

Bit	Description
SPIM_INT_ON_SPI_DONE	Enable interrupt due to SPI done
SPIM_INT_ON_TX_EMPTY	Enable interrupt due to TX FIFO empty
SPIM_INT_ON_TX_NOT_FULL	Enable interrupt due to TX FIFO not full
SPIM_INT_ON_BYTE_COMP	Enable interrupt due to Byte/Word complete
SPIM_INT_ON_SPI_IDLE	Enable interrupt due to SPI IDLE

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 SPIM_SetRxInterruptMode (uint8 intSrc)

Description: Configure which status bits trigger an interrupt event.

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

Bit	Description
SPIM_INT_ON_RX_FULL	Enable interrupt due to RX FIFO Full
SPIM_INT_ON_RX_NOT_EMPTY	Enable interrupt due to RX FIFO Not Empty
SPIM_INT_ON_RX_OVER	Enable interrupt due to RX Buf Overrun

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 SPIM_ReadTxStatus (void)

Description: Returns the current state of the TX status register. For more information, see the Status Register Bits section in this data sheet.

Parameters: None

Return Value: uint8: Current TX status register value

Bit	Description
SPIM_STS_SPI_DONE	SPI done
SPIM_STS_TX_FIFO_EMPTY	TX FIFO empty
SPIM_STS_TX_FIFO_NOT_FULL	TX FIFO not full
SPIM_STS_BYTE_COMPLETE	Byte/Word complete
SPIM_STS_SPI_IDLE	SPI IDLE

Side Effects: TX Status register bits are cleared on read.

uint8 SPIM_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
SPIM_STS_RX_FIFO_FULL	RX FIFO Full
SPIM_STS_RX_FIFO_NOT_EMPTY	RX FIFO Not Empty
SPIM_STS_RX_FIFO_OVERRUN	RX Buf Overrun

Side Effects: RX Status register bits are cleared on read.

void SPIM_WriteTxData (uint8/uint16 txData)

Description: Places a byte/word in the transmit buffer to be sent at the next available SPI bus time.

Parameters: uint8/uint16 txData: The data value to transmit from 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.

uint8/uint16 SPIM_ReadRxData (void)

Description: Returns the next byte/word of received data available in the receive buffer.

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 SPIM_GetRxBufferSize() and if it returns a non-zero value then it is safe to call the SPIM_ReadRxData() function.

uint8 SPIM_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 SPIM_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 SPIM_ClearRxBuffer (void)

Description: Clears the RX buffer memory array and RX hardware FIFO of all received data. Clears the RX RAM buffer by setting both the read and write pointers to zero. Setting the pointers to zero indicates that there is no data to read. 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 SPIM_ClearTxBuffer (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: 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 SPIM_TxEnable (void)

Description: If the SPI Master is configured to use a single bi-directional pin then this will set the bidirectional pin to transmit.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_TxDisable (void)

Description: If the SPI master 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 SPIM_PutArray (uint8/uint16 * buffer, uint8/uint16 byteCount)

Description: Places an array of data into the transmit buffer

Parameters: uint8 * buffer: Pointer to the location in RAM containing the data to send

uint8/uint16 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 SPIM_ClearFIFO (void)

Description: Clears any received data from the TX and RX FIFOs.

Parameters: None

Return Value: None

Side Effects: Clear status register of the component.

void SPIM_Sleep (void)

Description: Prepare SPIM component for low power modes by calling SPIM_SaveConfig() and SPIM_Stop() functions.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_Wakeup (void)

Description: Prepare SPIM Component to wake up from a low power mode. Calls SPIM_RestoreConfig() and SPIM_Enable() functions. Clears all data from RX buffer, TX buffer, and hardware FIFOs.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call SPIM_Init() because the SPIM _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 SPIM_Enable(void)

Description: Enables SPIM 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 function. The SPIM_Enable() function should be called before SPIM 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 SPIM_Init() function must have been previously called.

Parameters: None

Return Value: None

Side Effects: None

void SPIM_SaveConfig (void)

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

Parameters: None

Return Value: None

Side Effects: None

void SPIM_RestoreConfig (void)

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

Parameters: None

Return Value: None

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

```
SPIM_STATUS_MASK_REG
SPIM_COUNTER_PERIOD_REG
```

Defines

- **SPIM_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.
- **SPIM_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 SPIM_TXSTATUS

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

Table 2 SPIM_RXSTATUS

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

- Byte/Word Complete: Set when a byte/word of data 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 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).
- RX FIFO Full: Set when the RX Data FIFO is full (does not indicate the RX 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).
- TX FIFO Empty: Set when the TX Data FIFO is empty (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).
- SPI IDLE: Set when the SPIM state machine is in the IDLE State. This is the default state after the component starts. It is also the next state after SPI Done. IDLE is still set until TX FIFO Not Empty status has been detected.

SPIM_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.

SPIM_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.

SPIM_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 SPIM is as an 8-bit SPIM with Mode 0 configuration. By Default the Internal clock is selected with a bit-rate of 1 Mbps.

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 Master'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 – SPI IDLE state bit has been cleared because of bytes detected into the Tx buffer;

4 – 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.

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

6 – 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.

7 – 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).

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

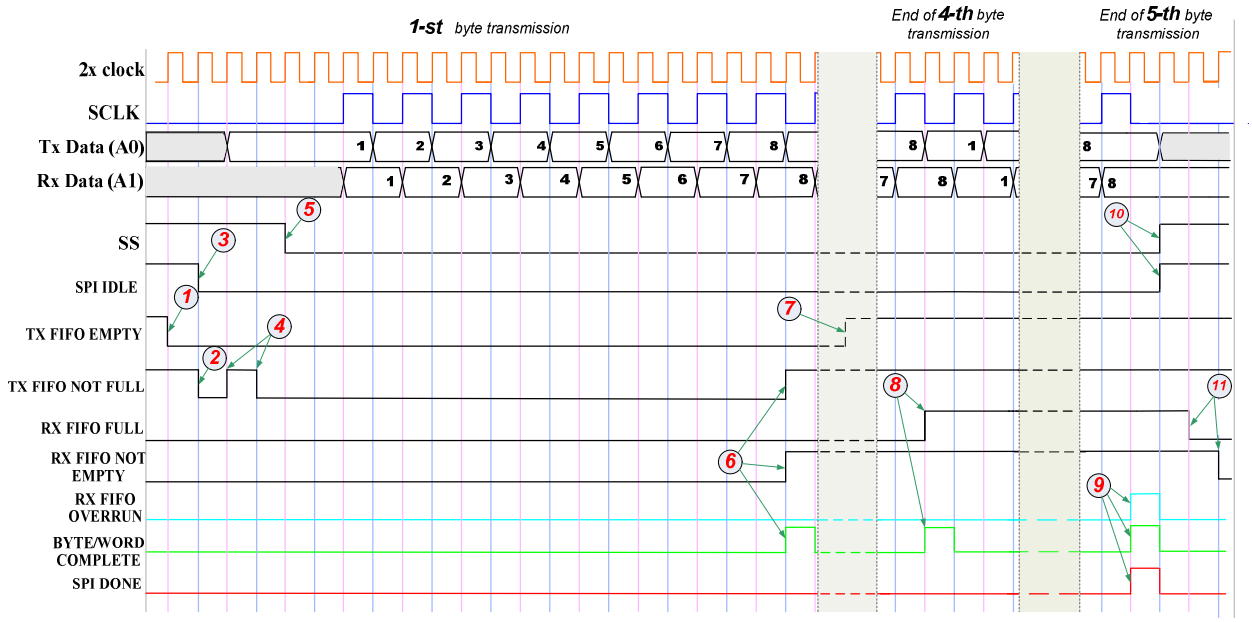
9 – Byte/Word Complete, SPI Done and Rx Overrun are set because all bytes have been transmitted and an attempt to load data into the full Rx buffer has been detected.

10 – SS line is set to High to indicate that transmission is complete. SPI IDLE state is set as well.

11 – 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.

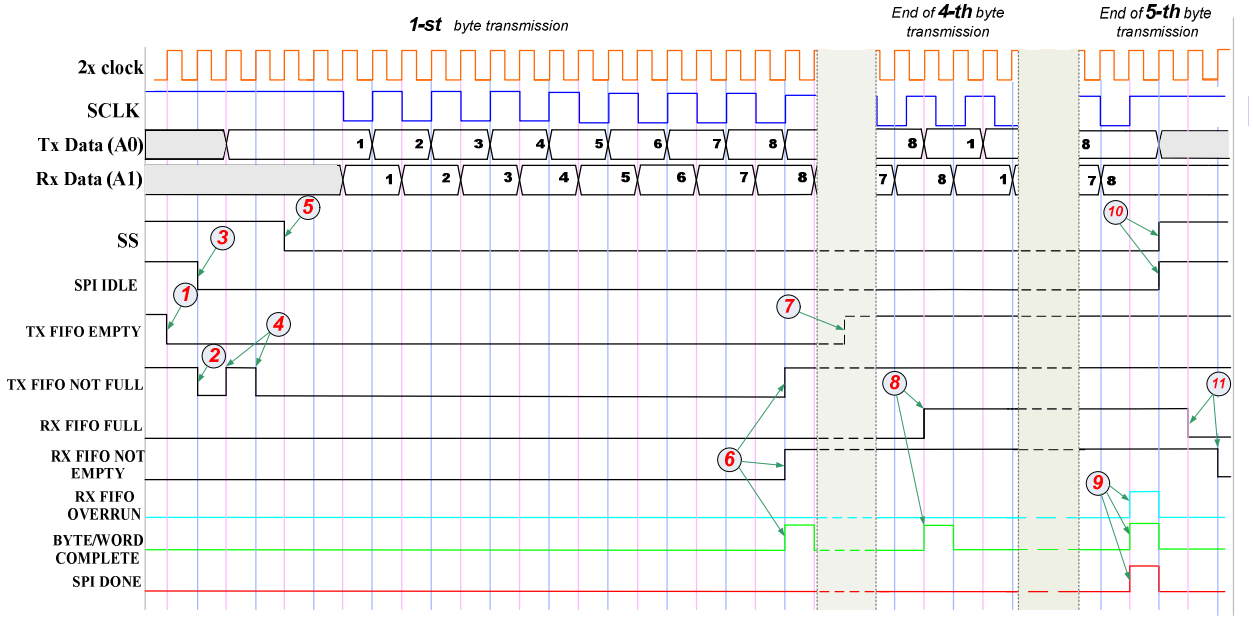
SPIM Mode: 0 (CPHA == 0, CPOL == 0)

Mode 0 has the following characteristics:



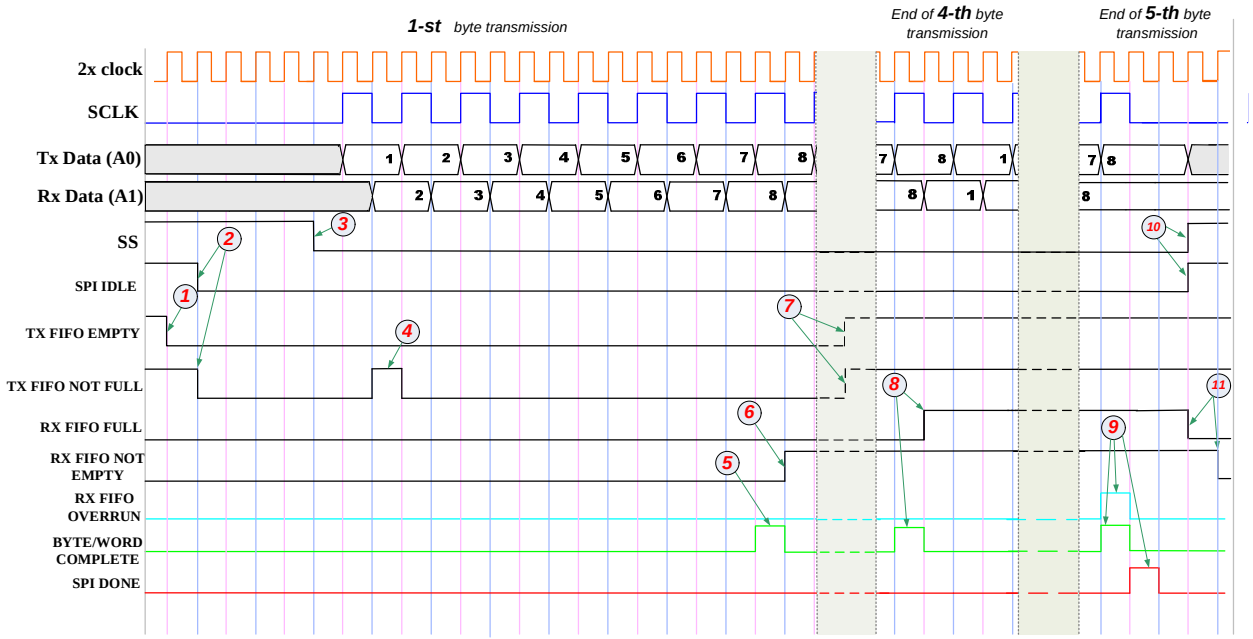
SPIM Mode: 1 (CPHA == 0, CPOL == 1)

Mode 1 has the following characteristics:



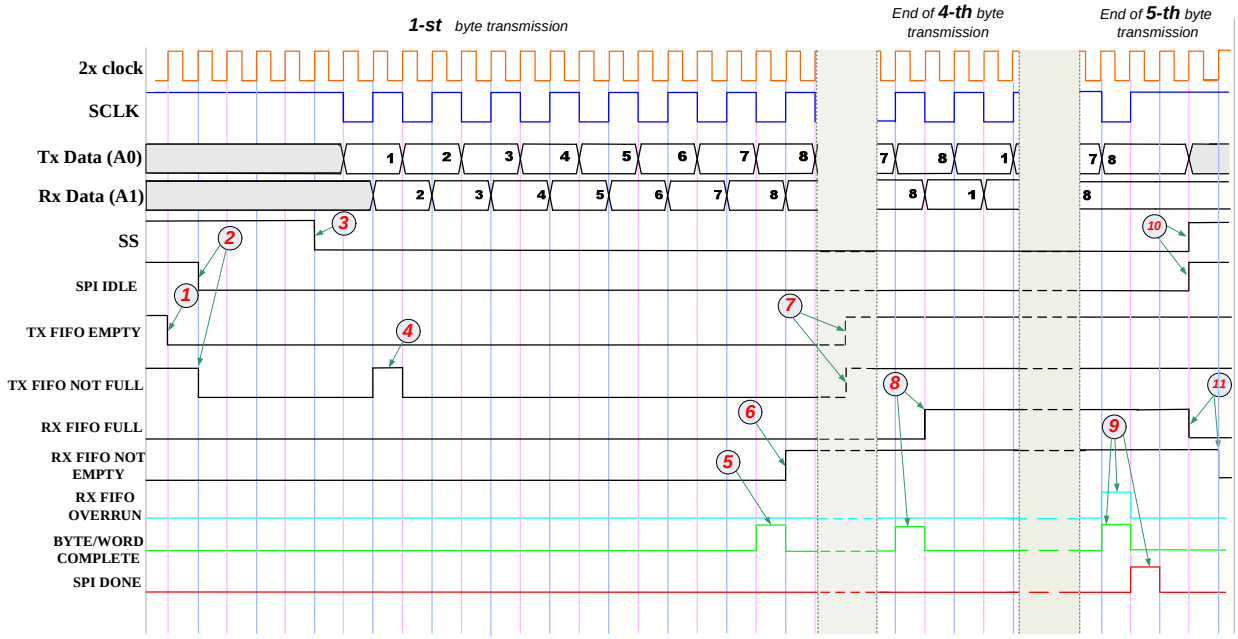
SPIM Mode: 2 (CPHA == 1, CPOL == 0)

Mode 2 has the following characteristics:



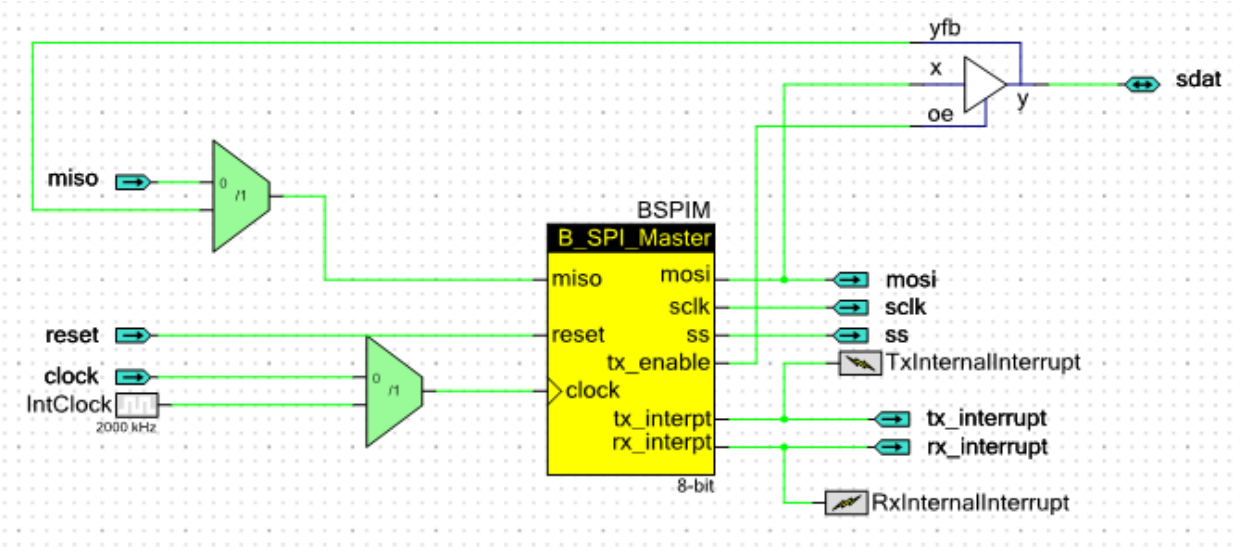
SPIM Mode: 3 (CPHA == 1, CPOL == 1)

Mode 3 has the following characteristics:

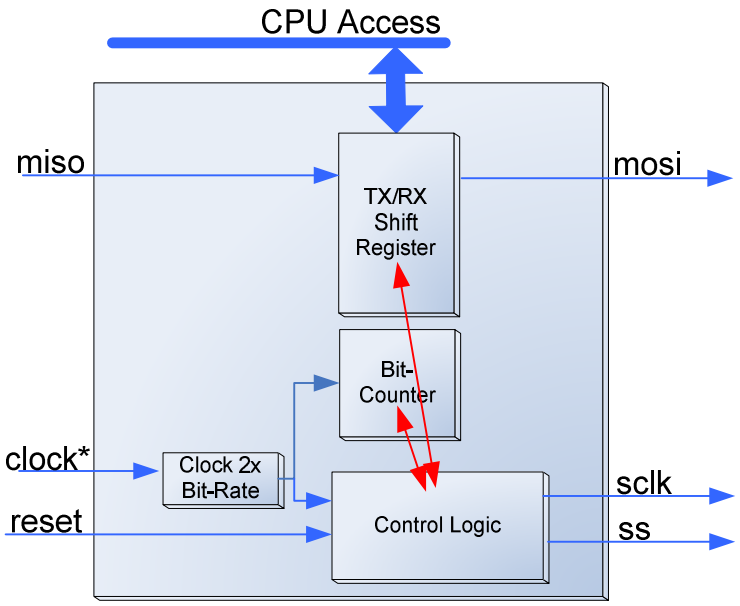


Block Diagram and Configuration

The SPIM is only available as a UDB configuration. The registers are described here to define the hardware implementation of the SPIM.



The implementation is described in the following block diagram.



Registers

TX Status Register

The TX status register is a read only register which contains the various transmit status bits defined for a given instance of the SPIM component. Assuming that an instance of the SPIM is named "SPIM," the value of this register is available with the SPIM_ReadTxStatus() function.

The interrupt output signal is generated from ORing the masked bit-fields within the TX status register. You can set the mask using the SPIM_SetTxInterruptMode() function. Upon receiving an interrupt, you can retrieve the interrupt source by reading the TX status register with the SPIM_ReadTxStatus () function.

Sticky bits in the TX status register are cleared on reading, so the interrupt source is held until the SPIM_ReadTxStatus() function is called. All operations on the TX status register must use the following defines for the bit-fields, because these bit-fields may be moved around within the TX status register at build time. Sticky bits used to generate an interrupt or DMA transaction must be cleared with either a CPU or DMA read to clear the bits and avoid continuously generating the interrupt or DMA.

There are several bit-fields defined for the TX status registers. Any combination of these bit-fields may be included as an interrupt source. The bit-fields indicated with an asterisk (*) in the following list are configured as sticky bits in the TX status register. All other bits are configured as real-time indicators of status. Sticky bits latch a momentary state so that they may be read at a later time and cleared on read. The following #defines are available in the generated header file (for example, *SPIM.h*):

- SPIM_STS_SPI_DONE * – Set high as the data latching edge of SCLK (edge is mode dependant) is output after the last bit of the configured number of bits in a single SPI word is output onto the MOSI line and the transmit FIFO is empty. Cleared when the SPIM is transmitting data or the transmit FIFO has pending data. Useful for knowing when the SPIM is complete with a multi-word transaction.
- SPIM_STS_TX_FIFO_EMPTY – Reads high while the transmit FIFO contains no data pending transmission. Reads low if data waiting for transmission is present.
- SPIM_STS_TX_FIFO_NOT_FULL – Reads high while the transmit FIFO is not full and has room to write more data. Reads low if the FIFO is full of data pending transmit and there is no room for more writes at this time. Useful for knowing when it is safe to pend more data into the transmit FIFO.
- SPIM_STS_BYTE_COMPLETE * – Set high as the last bit of the configured number of bits in a single SPI word is output onto the MOSI line. Cleared* as the data latching edge of SCLK (Edge is mode dependent) is output.
- SPIM_STS_SPI_IDLE * – This bit is set high as long as the component State Machine is in the SPI IDLE state. (Component is waiting for Tx data and is not transmitting any data)

RX Status Register

The RX status register is a read only register that contains the various receive status bits defined for the SPIM. The value of this register is available with the SPIM_ReadRxStatus() function.

The interrupt output signal is generated from ORing the masked bit-fields within the RX status register. You can set the mask using the SPIM_SetRxInterruptMode() function. Upon receiving an interrupt, you can retrieve the interrupt source by reading the RX status register with the SPIM_ReadRxStatus () function.

Sticky bits in the RX status register are cleared on reading, so the interrupt source is held until the SPIM_ReadRxStatus() function is called. All operations on the RX status register must use the following defines for the bit-fields, because these bit-fields may be moved around within the RX status register at build time. Sticky bits used to generate an interrupt or DMA transaction must be cleared with either a CPU or DMA read to clear the bits and avoid continuously generating the interrupt or DMA.

There are several bit-fields defined for the RX status register. Any combination of these bit-fields may be included as an interrupt source. The bit-fields indicated with an asterisk (*) in the following list are configured as sticky bits in the RX status register. All other bits are configured as real-time indicators of status. Sticky bits latch a momentary state so that they may be read at a later time and cleared when read. The following #defines are available in the generated header file (for example, *SPIM.h*):

- SPIM_STS_RX_FIFO_FULL – Reads high when the receive FIFO is full and has no more room to store received data. Reads low if the FIFO is not full and has room for additional received data at this time. Useful to know if there is room for new received data to be stored.

- SPIM_STS_RX_FIFO_NOT_EMPTY – Reads high when the receive FIFO is not empty. Reads low if the FIFO is empty and has room for additional received data at this time.
- SPIM_STS_RX_FIFO_OVERRUN * – Reads high when the receive FIFO is already full and additional data was written to it. Useful to know if data has been lost from the FIFO due to slow FIFO servicing.

TX Data Register

The TX data register contains the transmit data value to send. This is implemented as a FIFO in the SPIM. There is an optional higher level software state machine to control data from the transmit memory buffer to handle large amounts of data to be sent that exceed the FIFO's capacity. All APIs dealing with transmitting data must go through this register to place the data onto the bus. If there is data in this register and the control state machine 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 set up to fill this FIFO when empty using the TXDATA_REG address defined in the header file.

RX Data Register

The RX data register contains the received data. This is implemented as a FIFO in the SPIM. There is an optional higher level 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 set up from this register to the memory array, or the firmware may simply call the SPIM_ReadRxData() function. DAM must use the RXDATA_REG address defined in the header file.

Conditional Compilation Information

The SPIM requires only one conditional compile definition to handle the 8 or 16 bit datapath configuration necessary to implement the configured NumberOfDataBits. It is required that the API conditionally compile for the data width defined. APIs should never use these parameters directly but should use the following define:

- SPIM_DATAWIDTH – This defines how many data bits will make up a single "byte" transfer. Valid range is 3 to 16 bits.

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]			9	MHz
		Config 2 [3]			9	MHz
		Config 3 [4]			8	MHz
		Config 4 [5]			8	MHz
f_{clock}	Component clock frequency [6]	Config 1 [2]	$2 * f_{SCLK}$		18	MHz
		Config 2 [3]	$2 * f_{SCLK}$		18	MHz
		Config 3 [4]	$2 * f_{SCLK}$		16	MHz
		Config 4 [5]	$2 * f_{SCLK}$		16	MHz
t_{CKH}	SCLK High time			0.5		$1/f_{SCLK}$
t_{CKL}	SCLK Low time			0.5		$1/f_{SCLK}$
t_{S_MISO}	MISO Input Setup Time		25			ns
t_{H_MISO}	MISO 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]			4.5	MHz
		Config 2 [3]			4.5	MHz
		Config 3 [4]			4	MHz
		Config 4 [5]			4	MHz
f_{clock}	Component clock frequency[6]	Config 1 [2]	$2 * f_{SCLK}$		9	MHz
		Config 2 [3]	$2 * f_{SCLK}$		9	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_{S_MISO}	MISO Input Setup Time		25			ns
t_{H_MISO}	MISO 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] 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

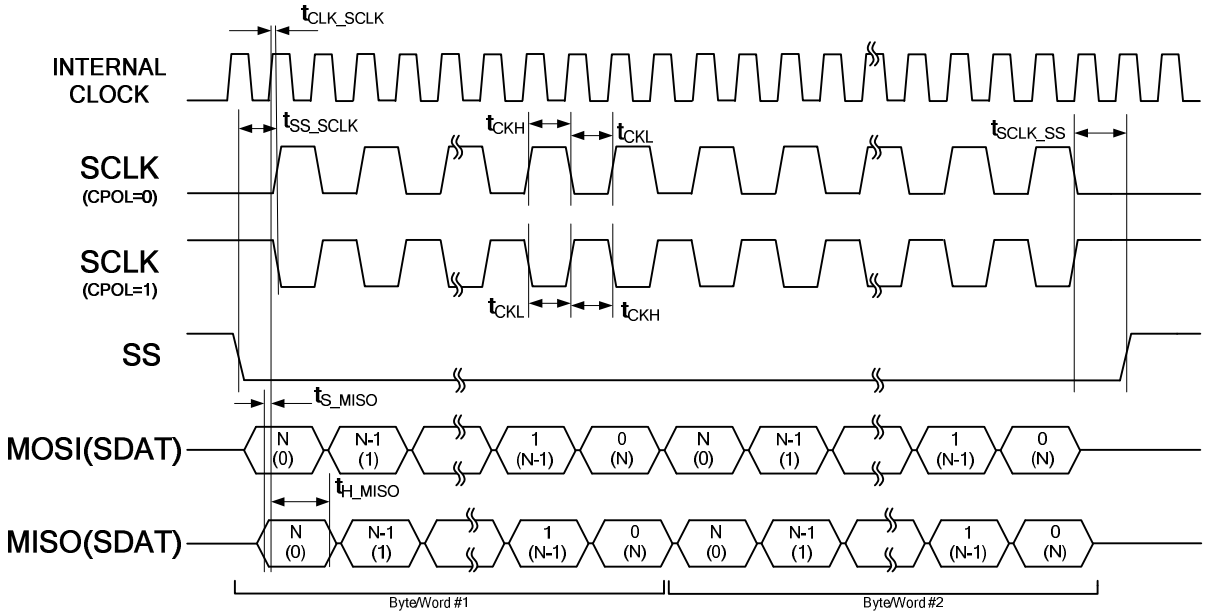
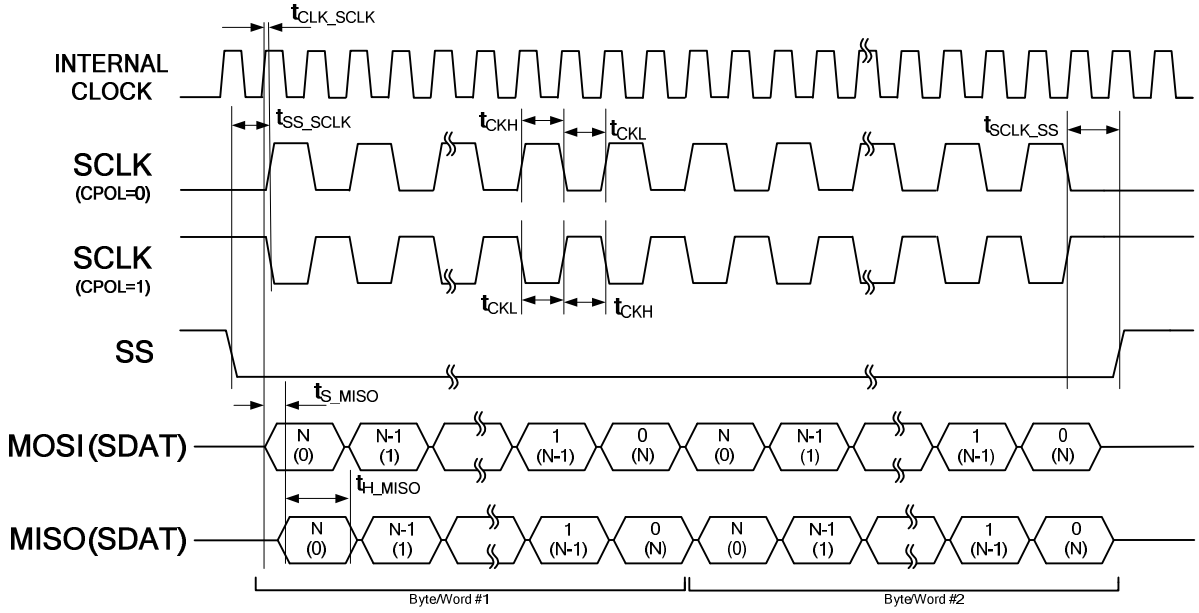


Figure 8. Mode CPHA=1 Timing Diagram



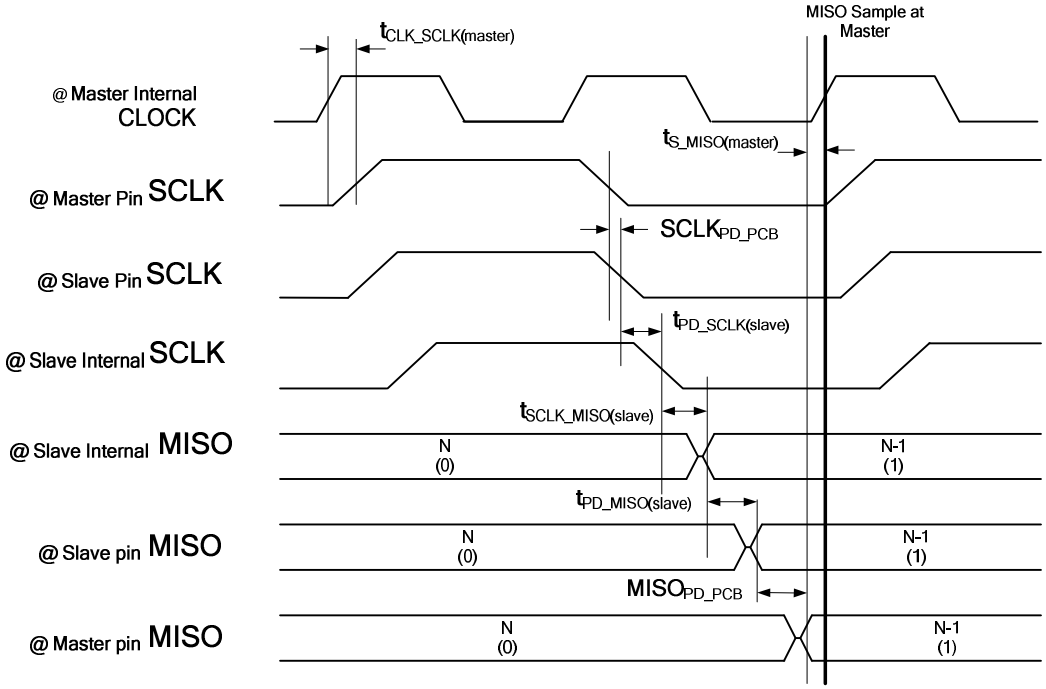
How to Use STA Results for Characteristics Data

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:

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

-- OR --

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

Where:

$t_{CLK_SCLK(master)}$ is the path delay of the input CLK to the SCLK output. This is provided in the STA results clock to output times as shown below:

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

$t_{S_MISO(Master)}$ is the path delay from MISO input pin to internal logic of the master component; This is provided in the STA results input setup times as shown below:

-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

Note: We have two values in the report: setup times to the component clock at register A0 and register A1. The bigger one should be selected for calculation.

And t_{RT_PD} is defined as:

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

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.

$t_{PD_SCLK(slave)} + t_{SCLK_MISO(slave)} + t_{PD_MISO(slave)}$ must come from the Slave Device Datasheet. $MISO_{PD_PCB}$ is the PCB Path delay of 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:

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

f_{clk} 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	66.000 MHz	UNKNOWN	
SPIM_1_IntClock	2.000 MHz	45.108 MHz	

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

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

t_{CLK_SCLK} Internal clock to SCLK output time. Time from posedge of Internal Clock to SCLK available on master pin.

t_{S_MISO} To meet the setup time of the internal logic MISO must be valid at the pin, before FCLK is valid at the pin, by this amount of time

t_{H_MISO} To meet the hold time of the internal logic MISO must be valid at the pin, after FCLK 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
	"Interrupt on SPI Idle" checkbox is added to the component configure dialog.	Component customizer defect fixed
	"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.	
2.0	Added SPIM_Sleep()/SPIM_Wakeup() and SPIM_Init()/SPIM_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 I/Os have been changed: • The clock input is now visible in default placement (external clock source is the default setting now) • The reset input has a different position • The interrupt output was removed. rx_interrupt, tx_interrupt outputs are added instead.	The clock input was added for consistency with SPI Slave. The reset input place changed because the clock input wasadded. Two status interrupt registers (Tx and Rx) are now presented instead of one shared. These changes be taken into account to prevent binding errors when migrating from previous SPI versions
	Removed SPIM_EnableInt(), SPIM_DisableInt(), _SetInterruptMode, and _ReadStatus APIs. Added SPIM_EnableTxInt(), SPIM_EnableRxInt(), SPIM_DisableTxInt(), SPIM_DisableRxInt(), SPIM_SetTxInterruptMode(), SPIM_SetRxInterruptMode(), SPIM_ReadTxStatus(), SPIM_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 SPIM_ReadByte(), SPIM_WriteByte()APIs to SPIM_ReadRxData(), SPIM_WriteTxData().	Clarifies the APIs and how they should be used.
The following changes were made to the base SPI Master component B_SPI_Master_v2_0, which is implemented using Verilog:		
	spim_ctrl internal module was replaced by a new state machine.	It uses less hardware resources and does not contain any asynchronous logic.

Version	Description of Changes	Reason for Changes / Impact

	Two statusi registers are now presented (status are separate for Tx and Rx instead of using one common status register for both.
	`/*SPI_Master_v1_20 status bits*/`
	`SPIM_STS_SPI_DONE_BIT = 3'd0;`
	`SPIM_STS_TX_FIFO_EMPTY_BIT = 3'd1;`
	`SPIM_STS_TX_FIFO_NOT_FULL_BIT = 3'd2;`
	`SPIM_STS_RX_FIFO_FULL_BIT = 3'd3;`
	`SPIM_STS_RX_FIFO_NOT_EMPTY_BIT = 3'd4;`
	`SPIM_STS_RX_FIFO_OVERRUN_BIT = 3'd5;`
	`SPIM_STS_BYTE_COMPLETE_BIT = 3'd6;`
	
	`/*SPI_Master_v2_0 status bits*/`
	`localparam SPIM_STS_SPI_DONE_BIT = 3'd0;`
	`localparam SPIM_STS_TX_FIFO_EMPTY_BIT = 3'd1;`
	`localparam SPIM_STS_TX_FIFO_NOT_FULL_BIT = 3'd2;`
	`localparam SPIM_STS_BYTE_COMPLETE_BIT = 3'd3;`
	`localparam SPIM_STS_SPI_IDLE_BIT = 3'd4;`
	`localparam SPIM_STS_RX_FIFO_FULL_BIT = 3'd4;`
	`localparam SPIM_STS_RX_FIFO_NOT_EMPTY_BIT = 3'd5;`
	`localparam SPIM_STS_RX_FIFO_OVERRUN_BIT = 3'd6;`
		Fixed a defect found in previous versions of the component where software buffers were not working as expected.

	'BidirectMode' boolean parameter is added to base component. 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. Routed reset is connected to datapaths, Counter7 and State Machine.	Added Bidirectional Mode support to the component
	udb_clock_enable component is added to Verilog implementation with sync = `TRUE` parameter.	New requirements for all clocks used in Verilog to indicate functionality so the tool can support synchronization and Static Timing Analysis.
	'*2' is replaced by '<< 1' in Counter7 period value.	Verilog improvements.
	Maximum Bit Rate value is changed to 10 Mbps	Bit Rate value more than 10 Mbps is not supported (verified during the 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

Version	Description of Changes	Reason for Changes / Impact
	Sample firmware source code is removed	Reference to the component example project is added instead
	SPI Modes diagrams are changed (Tx and Rx FIFO status values are added)	Data sheet defect fixed

© Cypress Semiconductor Corporation, 2009-2010. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.