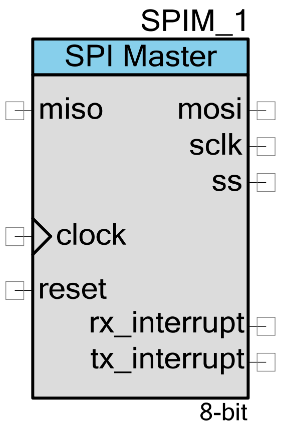
Serial Peripheral Interface (SPI) Master
2.0

Features

- 2- to 16-bit data width
- 4 SPI operating modes
- Data rates to 33 Mb/s



General Description

The SPI Master component provides an industry-standard 4-wire master SPI interface, as well as a 3-wire (or bidirectional) SPI mode. The interface supports 4 SPI operating modes, allowing interface with any SPI slave device. In addition to the standard 8-bit interface, the SPI Master supports a configurable 2- to 16-bit interface for interfacing to nonstandard SPI word lengths. SPI signals include the standard SCLK, MISO + MOSI (or SDAT) pins, and Slave Select (SS) signal generation.

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 interface.

The SPI Slave component should be used in instances requiring the PSoC device to interface with a 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.

Input/Output Connections

This section describes the various input and output connections for the SPI. 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 input – slave output serial data from a slave device on the bus. This input is visible when the **Data Lines** parameter is set to MOSI + MISO. If visible, it must be connected.

sdat – Inout *

The sdat inout is used in Bidirectional mode instead of mosi+miso. This input is visible when the **Data Lines** parameter is set to Bidirectional.

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 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.

mosi – Output *

The mosi output carries the master output – slave input serial data 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 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 signal to a 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 Slave select. See the following figures.

Figure 1: Slave Select Output to De-Multiplexer

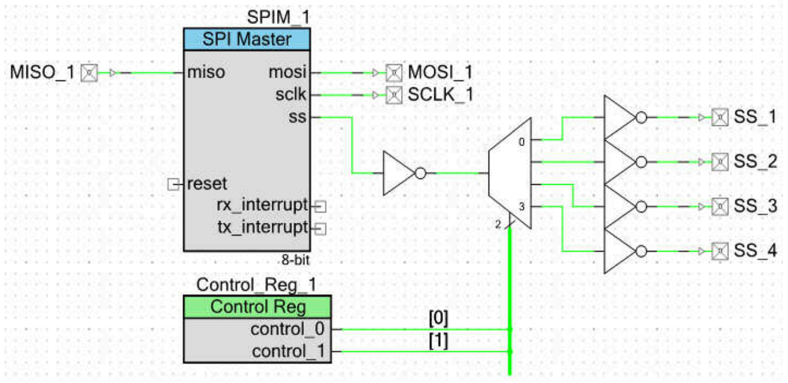
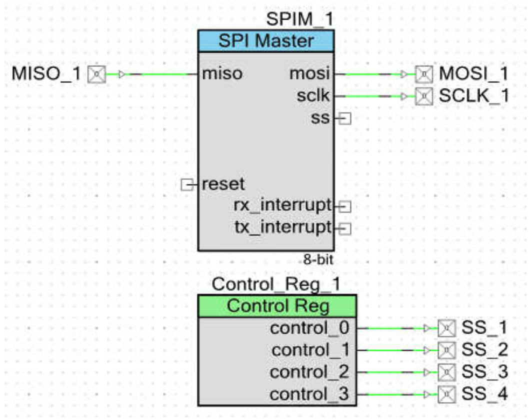
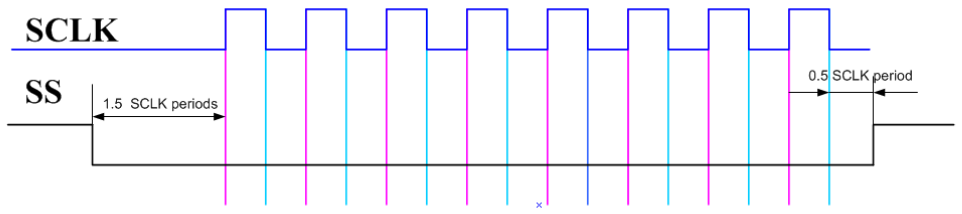


Figure 2: Firmware Controlled Slave Select(s)



The following diagram shows the timing correlation between SS and SCLK (valid for all SPI modes):



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 a clock source. Schematic Macros are available both for 4–wire (Full Duplex) and 3-wire (Bidirectional) SPI interfacing.

Figure 3: 4-Wire (Full Duplex) Interfacing Schematic Macro

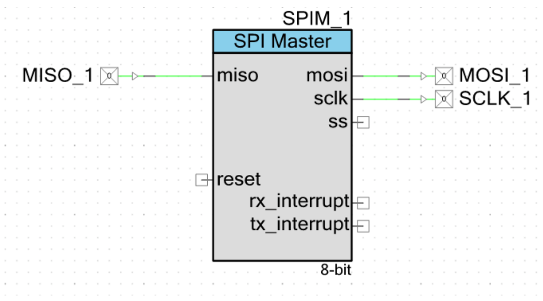
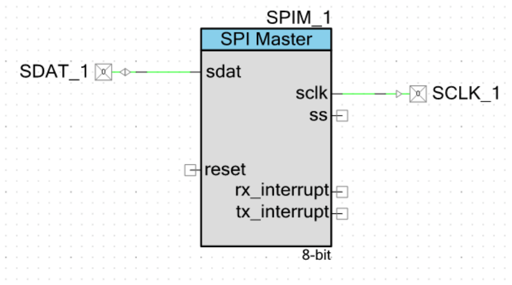


Figure 4: 3-Wire (Bidirectional) Interfacing Schematic Macro



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

PRELIMINARY

Parameters and Setup

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

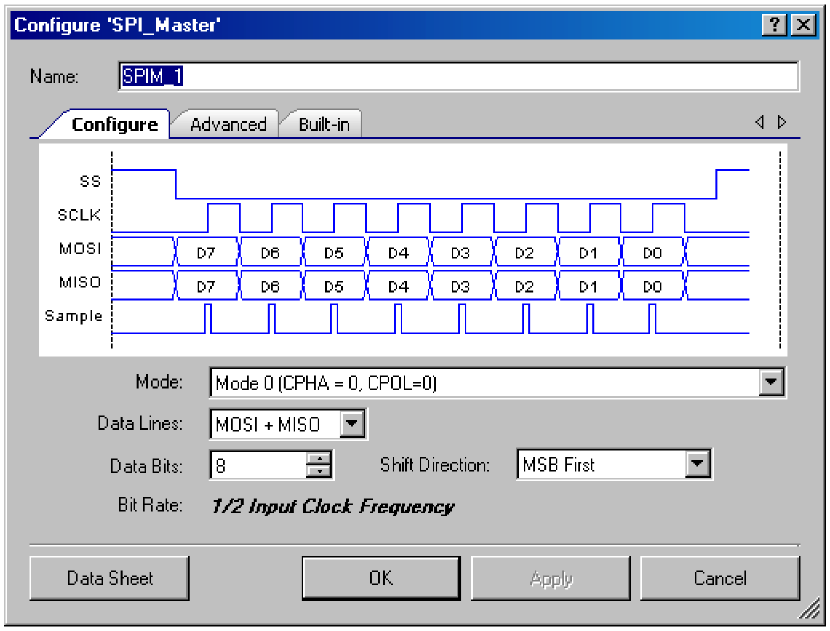
If the component will be used for communication with one or more external SPI Slave devices, then connect the appropriate digital input and output pins.

The following sections describe the SPI Master parameters, and how they are configured using the 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 the API provided. Hardware only parameters are marked with an asterisk (*).

Configure Tab



Name: SPIM_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*

These are basic parameters expected for every SPI component and are therefore the first parameters visible to configure.

Mode *

The **Mode** parameter defines the desired clock phase and clock polarity mode used in the communication. The options are "Mode 00", "Mode 01", "Mode 10" and "Mode 11" which are defined in the implementation details below.

PRELIMINARY

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 ReadRxData() and WriteTxData() APIs. The default number of bits is a single byte (8-bits). Any integer from 2 to 16 may be selected.

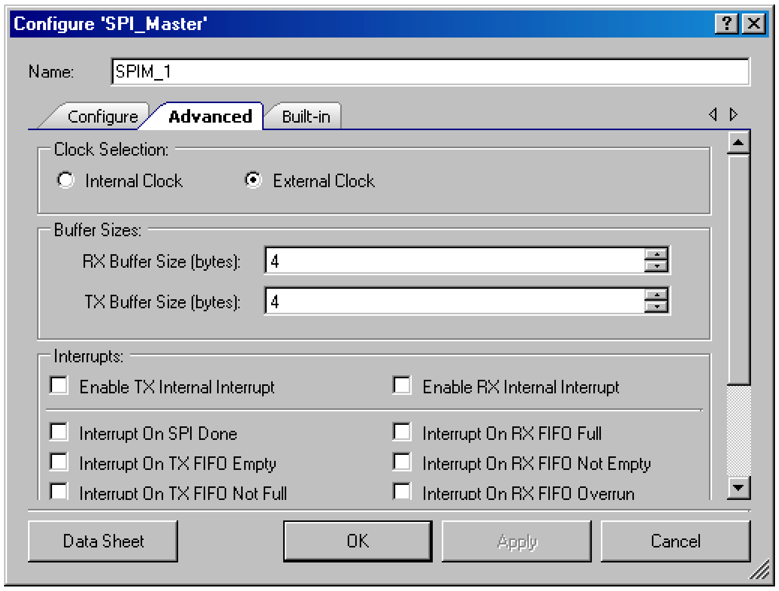
Shift Direction *

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

Bit Rate *

The **Bit Rate** parameter defines the communication speed in Hertz. If the internal clock is selected this parameter will define the clock frequency of the internal clock as 2x the bit rate. This parameter has no affect if the external clock option is set.

Advanced Tab



Name: SPIM_1

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 Full
☐ Interrupt On TX FIFO Empty ☐ Interrupt On RX FIFO Not Empty
☐ Interrupt On TX FIFO Not Full ☐ Interrupt On RX FIFO Overrun

Clock Selection *

The **Clock Selection** parameter allows the user to choose between an internally configured clock or an externally configured clock or I/O for the data-rate generation. When set to "Internal Clock" the required clock frequency is calculated and configured by PSoC Creator based on the

PRELIMINARY

Bit Rate parameter. When set to "External Clock," the component does not control the data rate but can calculate the expected bit rate. If this parameter is "Internal Clock" then the clock input is not visible on the symbol.

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 (8-bit processor) or 64535 (32-bit processor) 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 (8-bit processor) or 64535 (32-bit processor) 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, RX ISR of the SPI Master component. You may add to these ISR if selected or deselect the internal interrupt and handle the ISR with an external interrupt component connected to the interrupt outputs of the SPI Master.

If you select an RX or TX buffer size greater than 4, the appropriate parameters are set automatically as the internal ISR is needed to handle transferring data from the 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 based on the selected status interrupts. This output may then be used as a DMA request source to DMA from the RX or TX buffer independent of the interrupt or as another interrupt dependant upon the desired functionality.

Note When RX buffer size is greater than 4 bytes/words interrupt from ‘RX FIFO NOT EMPTY’ event is always enabled and can’t be disabled by user because it causes incorrect handler functionality.

When TX buffer size is greater than 4 bytes/words interrupt from ‘TX FIFO NOT FULL’ is always enabled and can’t be disabled by user because it causes incorrect handler functionality.

Interrupts

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

Clock Selection

When the internal clock configuration is selected PSoC Creator will calculate the needed frequency and clock source and will generate the resource needed for implementation. Otherwise, you must supply the clock and calculate the bit-rate at 1/2 the input clock 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 Master component is placed throughout 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 Master 8-bit	1	*	2	1	1			*
SPI Master 16-bit	2	*	2	1	1			*

* Unknown

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 "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

PRELIMINARY

name, variable, and constant symbol. For readability, the instance name used in the following table is "SPIM".

Function	Description
SPIM_Init	Initializes and restores default SPIM configuration provided with the customizer
SPIM_Enable	Enable the SPIM operation
SPIM_Start	Enable the SPIM operation
SPIM_Stop	Disable the 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 in the transmit buffer which will be sent at the next available bus time
SPIM_ReadRxData	Returns the next byte/word of received data
SPIM_GetRxBufferSize	Returns the size (in bytes/words) of the RX memory buffer
SPIM_GetTxBufferSize	Returns the size (in bytes/words) of the TX memory buffer
SPIM_ClearRxBuffer	Clears the memory array of all received data
SPIM_ClearTxBuffer	Clears the memory array of all transmit data
SPIM_TxEnable	Enables the TX portion of the SPI Master (MOSI)
SPIM_TxDisable	Disables the TX portion of the SPI Master (MOSI)
SPIM_PutArray	Places an array of data into the transmit buffer
SPIM_ClearFIFO	Clears any received data from the RX FIFO
SPIM_SaveConfig	Saves SPIM configuration
SPIM_RestoreConfig	Restores SPIM configuration
SPIM_Sleep	Prepare SPIM Component goes to sleep
SPIM_Wakeup	Prepare SPIM Component to wake up

Global Variables

Variable	Description
SPIM_initVar	Indicates whether the SPI Master has been initialized. The variable is initialized to 0 and 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	Amount of bytes/words written to TX Software Buffer is stored in this variable.
SPIM_txBufferRead	Amount of bytes/words reading from TX Software Buffer is stored in this variable.
SPIM_rxBufferWrite	Amount of bytes/words written to RX Software Buffer is stored in this variable.
SPIM_rxBufferRead	Amount of bytes/words reading from RX Software Buffer is stored in this variable.
SPIM_rxBufferFull	Indicate the Software Buffer overflow.
SPIM_RXBUFFER[]	Used to store data to sending.
SPIM_TXBUFFER[]	used to store received data

void SPIM_Init(void)

Description: Initializes and restores default SPIM configuration provided with the customizer.

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: Enable SPIM component.

Parameters: None

Return Value: None

Side Effects: None

PRELIMINARY

void SPIM_Start(void)

Description:	Enable the SPIM operation by enabling the internal clock. If external clock is selected then this function is only necessary for initial configuration. Start() function should be called before the interrupt is enabled as it configures the interrupt sources and clears any pending interrupts from device configuration.
Parameters:	Void
Return Value:	Void
Side Effects:	None

void SPIM_Stop(void)

Description:	Disable the SPIM operation by disabling the internal clock. If external clock is selected then this function has no affect on the SPIM operation
Parameters:	Void
Return Value:	Void
Side Effects:	None

void SPIM_EnableTxInt (void)

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

void SPIM_EnableRxInt (void)

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

void SPIM_DisableTxInt (void)

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

void SPIM_DisableRxInt (void)

Description:	Disables the internal RX interrupt irq.
Parameters:	Void
Return Value:	Void
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. 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 to the Defines section.
Return Value:	Void
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. 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.
Return Value:	Void
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.
Parameters:	Void
Return Value:	uint8: Current TX status register value
Side Effects:	TX Status register bits are clear 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:	Void
Return Value:	uint8: Current RX status register value
Side Effects:	RX Status register bits are clear on read.

void SPIM_WriteTxData (uint8/uint16 txData)

Description:	Places a byte/word in the transmit buffer which will be sent at the next available SPI bus time
Parameters:	uint8/uint16 txData: The data value to send across the SPI.
Return Value:	Void
Side Effects:	Data may be placed in the memory buffer and will not be transmitted until all other data has been transmitted. This function blocks until there is space in the output memory buffer. If this function is called again before the previous byte is finished then the next byte will be appended to the transfer with no time between the byte transfers. Clear status register of the component.

uint8/uint16 SPIM_ReadRxData (void)

Description:	Returns the next byte/word of received data
Parameters:	Void
Return Value:	uint8/uint16: The next byte of data read from the FIFO.
Side Effects:	Will return invalid data if the FIFO is empty. Call GetRxBufferSize() and if it returns a non-zero value then it is safe to call the ReadRxData() function.

uint8 SPIM_GetRxBufferSize (void)

Description: Returns the number of bytes/words of data currently held in the RX buffer. If RX Software Buffer not used then function return 0 if FIFO empty or 1 if FIFO not empty. In another case function return size of RX Software Buffer.

Parameters: void

Return Value: uint8: Integer count of the number of bytes/words in the RX buffer.

Side Effects: Clear status register of the component.

uint8 SPIM_GetTxBufferSize (void)

Description: Returns the number of bytes/words of data currently held in the TX buffer. If TX Software Buffer not used then function return 0 - if FIFO empty, 1 - if FIFO not full, 4 - if FIFO full. In another case function return size of TX Software Buffer.

Parameters: void

Return Value: uint8: Integer count of the number of bytes/words in the TX buffer

Side Effects: Clear status register of the component.

void SPIM_ClearRxBuffer (void)

Description: Clears the memory array and RX FIFO of all received data. Clear the RX 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: void

Return Value: void

Side Effects: Any received data not read from the RAM buffer will be lost when overwritten.

void SPIM_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: void

Return Value: void

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.

void SPIM_TxEnable (void)

Description:	If the SPI Master is configured to use a single bi-directional pin then this will set the bi-directional pin to transmit.
Parameters:	void
Return Value:	void
Side Effects:	None

void SPIM_TxDisable (void)

Description:	If the SPI master is configured to use a single bi-directional pin then this will set the bi-directional pin to receive.
Parameters:	void
Return Value:	void
Side Effects:	None

void SPIM_PutArray (uint8 * buffer, uint8 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 byteCount: The number of bytes/words to move to the transmit buffer.
Return Value:	Void
Side Effects:	Will stay in this routine until all data has been sent. May get locked in this loop if data is not being initiated by the master if there is not enough room in the TX FIFO.

void SPIM_ClearFIFO (void)

Description:	Clears any received data from the TX and RX FIFO.
Parameters:	Void
Return Value:	void
Side Effects:	Clear status register of the component.

void SPIM_SaveConfig (void)

Description:	Saves SPIM configuration.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIM_RestoreConfig (void)

Description:	Restores SPIM configuration.
Parameters:	None
Return Value:	None
Side Effects:	If this API is called without first calling SaveConfig then in the following registers will be default values from Customizer: `SPIM_STATUS_MASK_REG` `SPIM_COUNTER_PERIOD_REG`

void SPIM_Sleep (void)

Description:	Prepare SPIM Component goes to sleep.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIM_Wakeup (void)

Description:	Prepare SPIM Component to wake up.
Parameters:	None
Return Value:	None
Side Effects:	None

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.

PRELIMINARY

- **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.

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 has been transmitted.
- RX FIFO Overrun: Set when RX Data has overrun the 4 byte/word FIFO or 1 Byte/word FIFO without being moved to the Memory array (if one exists)
- RX FIFO Not Empty: Set when the RX Data FIFO is not empty i.e. at least one byte/word is in the RX FIFO (Does not indicate the RAM array conditions)
- RX FIFO Full: Set when the RX Data FIFO is full (Does not indicate the RAM array conditions)
- TX FIFO Not Full: Set when the TX Data FIFO is not full (Does not indicate the RAM array conditions):
- TX FIFO Empty: Set when the TX Data FIFO is empty (Does not indicate the 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 complete status.
- SPI IDLE: Set when the SPIM state machine being in IDLE State.

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

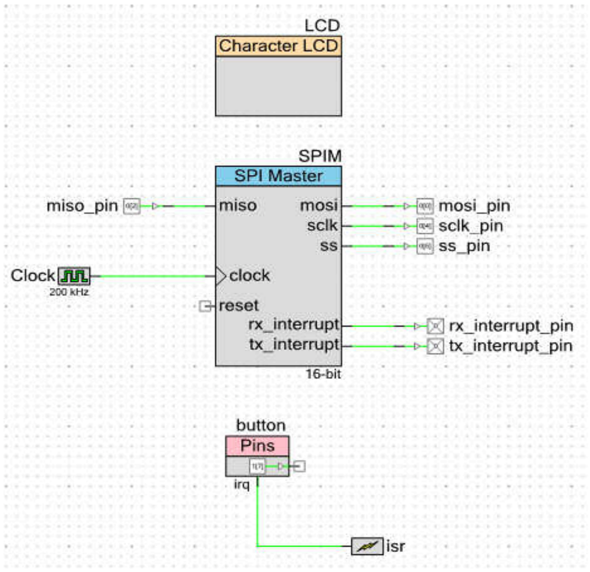
Defines the number of bits per data transfer chosen in the Configure dialog.

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the SPI Master component. This example assumes the component has been placed in a design with the name "SPIM."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

The following diagram shows the schematic to be used with the sample code.



SPI Master configuration:

- Mode - Mode 0(CPHA==0, CPOL ==0)
- Data lines - MOSI+MISO
- Data bits - 16
- Bit Rate - 100 Kbit/s
- Shift Direction - MSB First

PRELIMINARY

The SPI Master transfers eight 16-bit words. Eight 16-bit words received from the SPI Slave are displayed on the LCD.

A button with an Interrupt component are also placed in the design and connected through the appropriate pin to prevent impact of the startup glitch on the connected SPI Slave(s) when an SPI Slave can turn into unexpected active state. The same or another similar method is recommended for the SPI Slave(s) side. It is not necessary only if SPI Master and SPI Slave components are placed on the same chip. The `startVar` variable is set into the *main.c* file and `while(startVar)` cycle prevents code execution until `startVar` is cleared inside the button interrupt handler.

It is expected that the SPI Slave component will have the same settings, and be connected to the appropriate pins for correct SPI communication.

```
#include <device.h>

uint8 volatile startVar = 1;

void main()
{

    CYGlobalIntEnable;

    isr_Start();
    LCD_Start();

    while (startVar);

    button_ClearInterrupt();
    isr_ClearPending();

    SPIM_Start();
    LCD_Position(3,0);
    LCD_PrintString("start");

    SPIM_WriteTxData(0x1111);
    SPIM_WriteTxData(0x1112);
    SPIM_WriteTxData(0x1113);
    SPIM_WriteTxData(0x1114);
    SPIM_WriteTxData(0x1115);
    SPIM_WriteTxData(0x1116);
    SPIM_WriteTxData(0x1117);
    SPIM_WriteTxData(0x1118);

    while((SPIM_ReadTxStatus() & SPIM_STS_SPI_DONE) != SPIM_STS_SPI_DONE);

    LCD_Position(0,0);
    LCD_PrintHexUint16(SPIM_ReadRxData());
    LCD_PrintHexUint16(SPIM_ReadRxData());
    LCD_PrintHexUint16(SPIM_ReadRxData());
    LCD_PrintHexUint16(SPIM_ReadRxData());
    LCD_PrintHexUint16(SPIM_ReadRxData());
    LCD_PrintHexUint16(SPIM_ReadRxData());
    LCD_PrintHexUint16(SPIM_ReadRxData());
```

```
        LCD_PrintHexUint16(SPIM_ReadRxData());

        while(1){;}
    }

    isr.c code (button interrupt handler):

    External variable definition:
    /* `#START isr_intc` */
    extern uint8 volatile startVar;
      /* `#END` */

    CY_ISR(isr_Interrupt) user section:

        /* `#START isr_Interrupt` */
            startVar = 0;
        /* `#END` */
```

Functional Description

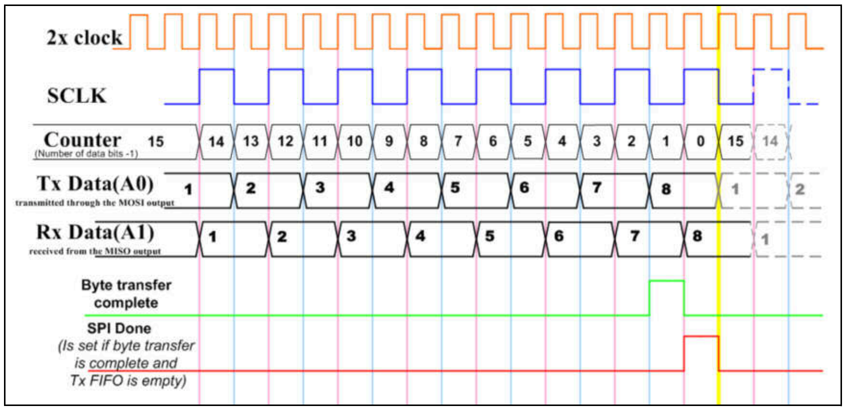
Default Configuration

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 Mb/s.

Modes

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

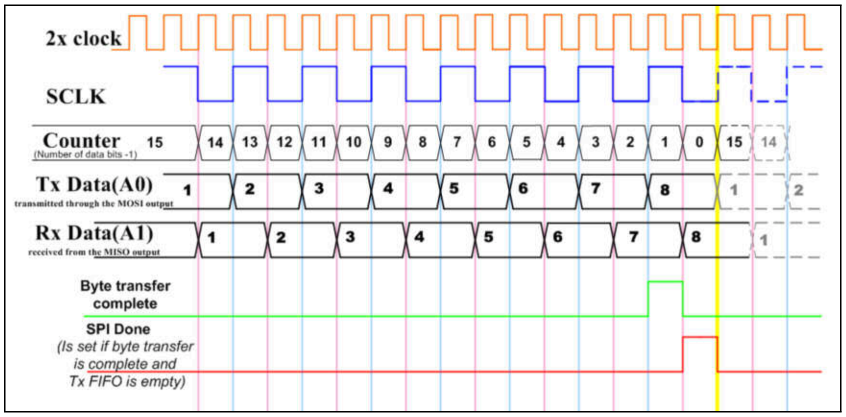
Mode 0 has the following characteristics:



PRELIMINARY

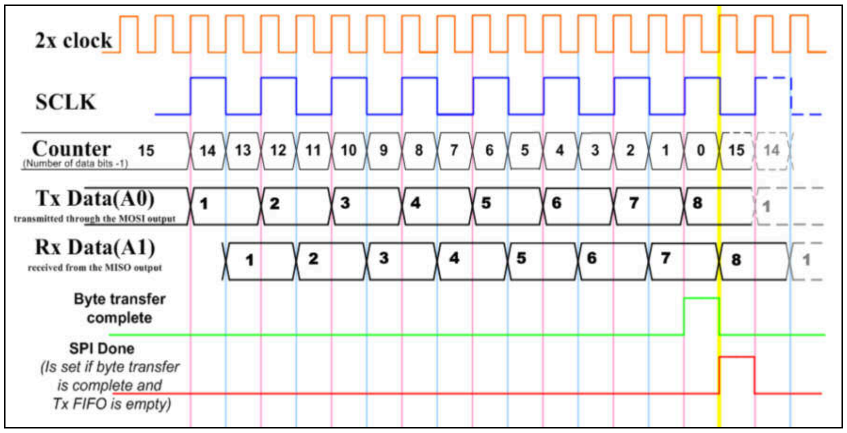
SPIM Mode: 1 (CPHA == 0, CPOL == 1)

Mode 1 has the following characteristics:


2x clock
SCLK
Counter (Number of data bits -1)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 15 14
Tx Data(A0) transmitted through the MOSI output
1 2 3 4 5 6 7 8 1 2
Rx Data(A1) received from the MISO output
1 2 3 4 5 6 7 8 1
Byte transfer complete
SPI Done (Is set if byte transfer is complete and Tx FIFO is empty)

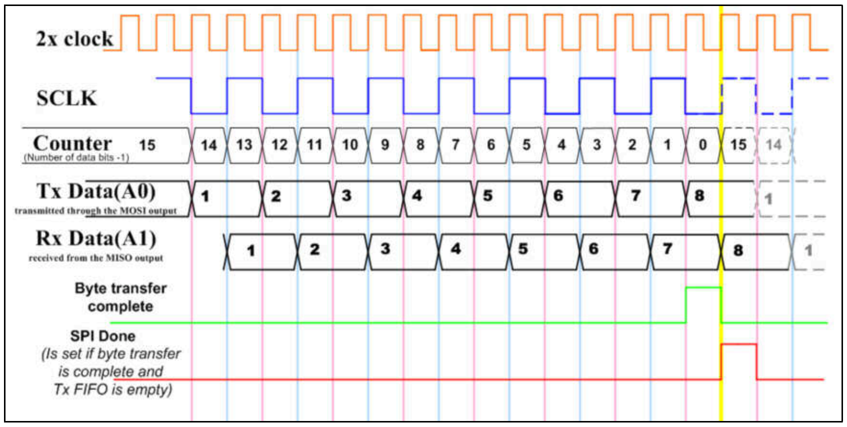
SPIM Mode: 2 (CPHA == 1, CPOL == 0)

Mode 2 has the following characteristics:


2x clock
SCLK
Counter (Number of data bits -1)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 15 14
Tx Data(A0) transmitted through the MOSI output
1 2 3 4 5 6 7 8 1
Rx Data(A1) received from the MISO output
1 2 3 4 5 6 7 8 1
Byte transfer complete
SPI Done (Is set if byte transfer is complete and Tx FIFO is empty)

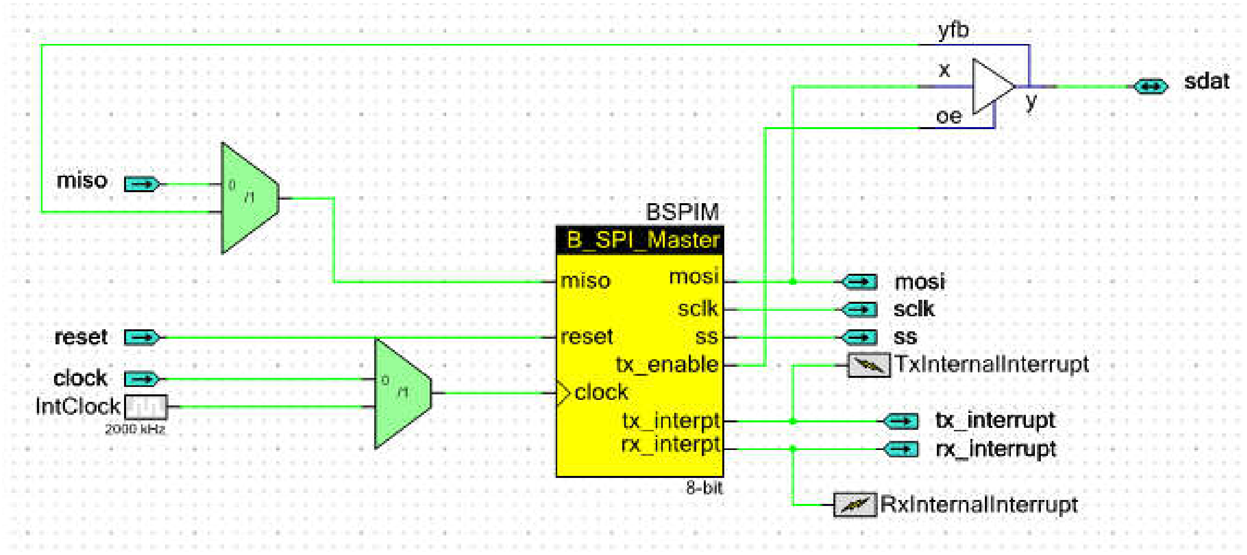
SPIM Mode: 3 (CPHA == 1, CPOL == 1)

Mode 3 defines has the following characteristics:

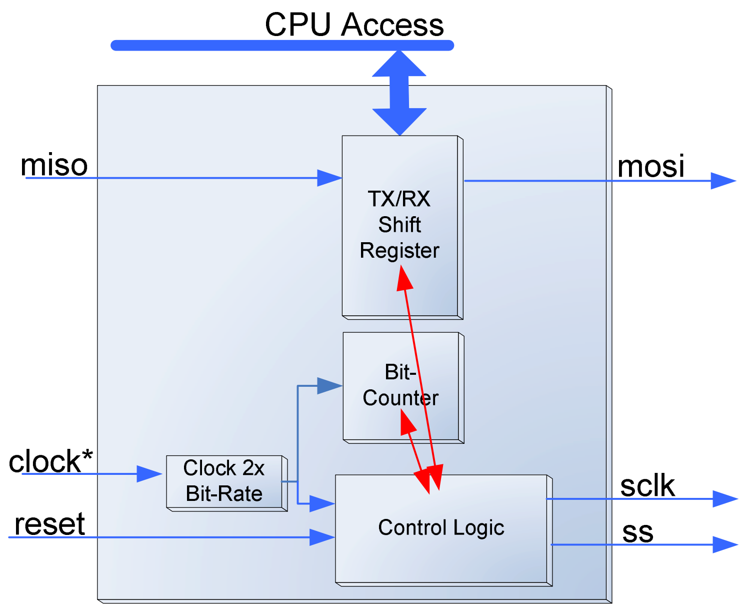

2x clock
SCLK
Counter (Number of data bits -1)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 15 14
Tx Data(A0) transmitted through the MOSI output
1 2 3 4 5 6 7 8 1
Rx Data(A1) received from the MISO output
1 2 3 4 5 6 7 8 1
Byte transfer complete
SPI Done (Is set if byte transfer is complete and Tx FIFO is empty)

Block Diagram and Configuration

The SPIM 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 SPIM.



The implementation is described in the following block diagram.



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 SPIM Component. Assuming that an instance of the SPIM is named 'SPIM' The value of this registers is available with the SPIM_ReadTxStatus() function call. The interrupt output signal is generated from an OR'ing of the masked bit-fields within the TX status register. You can set the mask using the SPIM_SetTxInterruptMode() function call and upon receiving an interrupt you can retrieve the interrupt source by reading the TX Status register with the SPIM_ReadTxStatus () function call. The TX Status register is 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 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:

- SPIM_STS_SPI_DONE * – Defined as the bit-mask of the Status register bit "SPI Done".
- SPIM_STS_TX_FIFO_EMPTY – Defined as the bit-mask of the Status register bit "Transmit FIFO Empty".
- SPIM_STS_TX_FIFO_NOT_FULL –
    - Defined as the bit-mask of the Status register bit "Transmit FIFO Not Full".
    - Defined as the bit-mask of the Status register bit "Receive FIFO Overrun".
- SPIM_STS_BYTE_COMPLETE * – Defined as the bit-mask of the Status register bit "Byte Complete".
- SPIM_STS_SPI_IDLE * – Defined as the bit-mask of the Status register bit "SPI IDLE".

Status RX

The RX status register is a read only register which contains the various status bits defined for the SPIM. The value of this registers is available with the SPIM_ReadRxStatus() and function call. The interrupt output signal is generated from an OR'ing of the masked bit-fields within the RX status register. You can set the mask using the SPIM_SetRxInterruptMode() function call and upon receiving an interrupt you can retrieve the interrupt source by reading the RX Status register with the SPIM_ReadRxStatus () function call. The RX Status register is clear on read 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 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:

- SPIM_STS_RX_FIFO_FULL – Defined as the bit-mask of the Status register bit "Receive FIFO Full".
- SPIM_STS_RX_FIFO_NOT_EMPTY – Defined as the bit-mask of the Status register bit "Receive FIFO Not Empty".
- SPIM_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 SPIM. 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 SPIM. 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 SPIM 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 compiles Data Width defined in the parameter chosen. The API 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.

References

Not applicable

DC and AC Electrical Characteristics

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

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.0.a	Moved component into subfolders of the component catalog.	
2.0	Added Sleep/Wakeup and Init/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 _EnableInt, _DisableInt, _SetInterruptMode, and _ReadStatus APIs. Added _EnableTxInt(), _EnableRxInt(), _DisableTxInt(), _DisableRxInt(), _SetTxInterruptMode(), _SetRxInterruptMode(), _ReadTxStatus(), _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.

Version	Description of Changes	Reason for Changes / Impact
	Renamed ReadByte(), WriteByte(), and WriteByteZero()APIs to ReadRxData(), WriteTxData(), WriteTxDataZero().	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.

	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 silcon. 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.