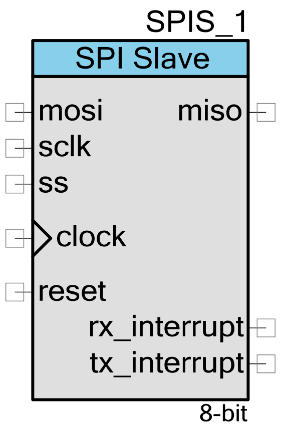
Serial Peripheral Interface (SPI) Slave

2.0

Features



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

General Description

The SPI Slave provides an industry-standard 4-wire slave SPI interface and 3-wire (or bidirectional) SPI mode. The interface supports 4 SPI operating modes, allowing interface with any SPI master device. In addition to the standard 8-bit interface, the SPI Slave 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.

When to use the SPI Slave

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

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

Input/Output Connections

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

mosi – Input *

The mosi input carries the master output – slave input serial data from the master 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.

sclk– Input

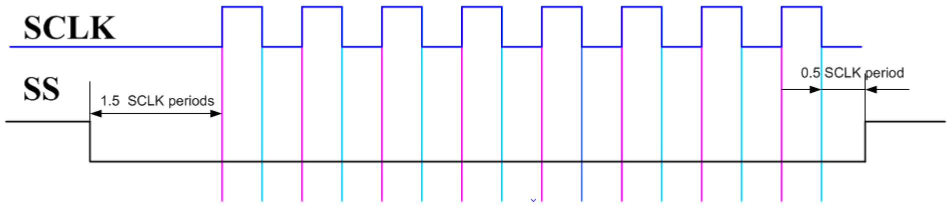
The sclk input provides the slave synchronization clock input to this device. This input is always visible and must be connected.

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

ss – Input

The ss input carries the slave select signal to this device. This input is always visible and must be connected.

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



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

clock – Input *

The clock input defines the sampling rate of the status register. All data clocking happens on the sclk input so the clock input DOES NOT handle the bit-rate of the SPI Slave.

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

miso – Output *

The miso output carries the slave output – master input serial data to the master device on the bus. This output is always visible and must be connected for TX operations.

This output is visible when the **Data Lines** parameter is set to MOSI + MISO.

interrupt – Output

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

Schematic Macro Information

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

Figure 1 4-wire (Full Duplex) Interfacing Schematic Macro

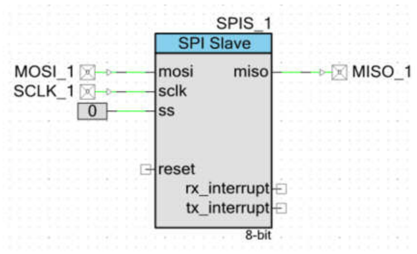


Figure 2 3-wire (Bidirectional) Interfacing Schematic Macro

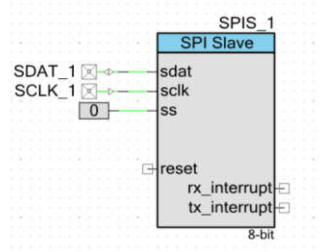
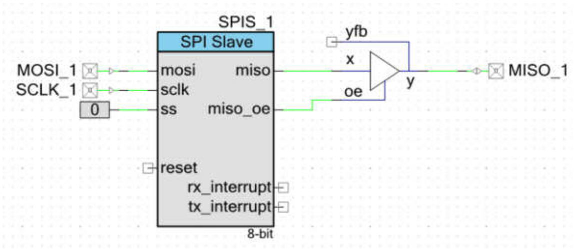


Figure 3 Multislave Mode Schematic Macro



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

PRELIMINARY

Parameters and Setup

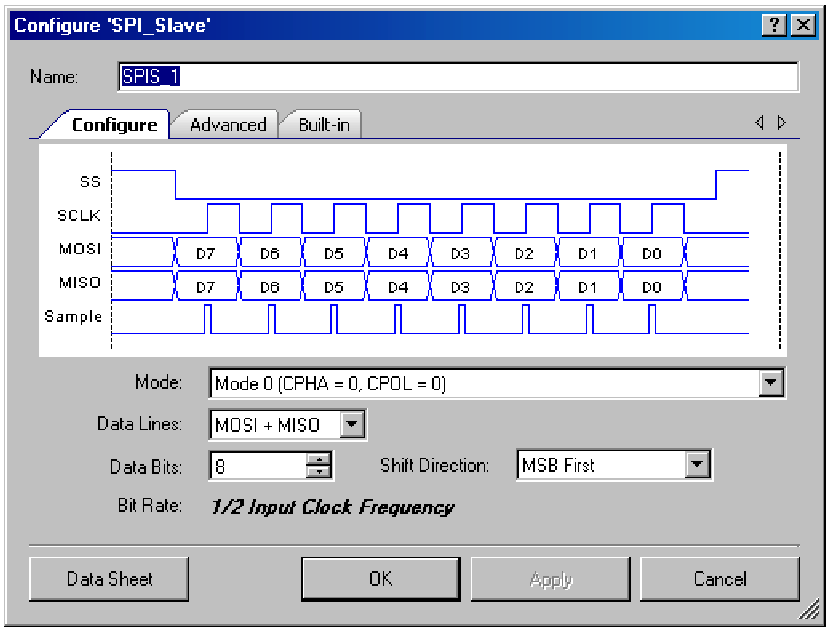
Drag an SPI Slave component onto your design. If component is used to communicate with an external SPI Master then connect appropriate digital input and output pins. Double-click component symbol to open the Configure dialog.

The following sections describe the SPI Slave 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: SPIS_1

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

Data Lines: MOSI + MISO

Data Bits: 8

Shift Direction: MSB First

Bit Rate: *1/2 Input Clock Frequency*

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" (default), "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).

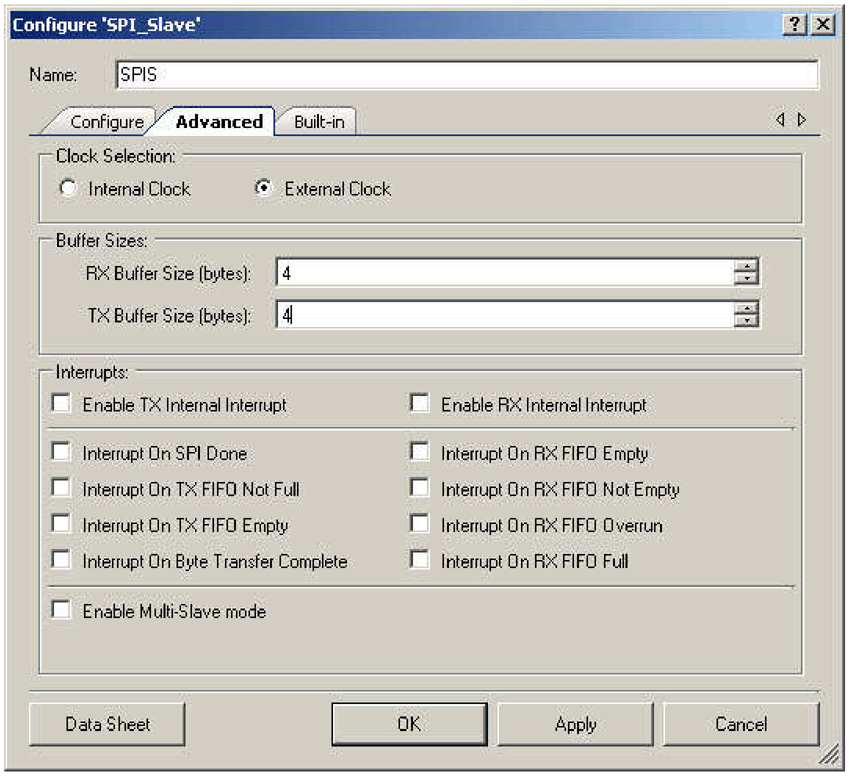
Data Bits *

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 (default) 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.

Advanced Tab


Configure 'SPI_Slave'

Name: SPIS

Configure | Advanced | Built-in

Clock Selection:
Internal Clock [✓ External Clock]

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

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

Clock Selection

Specifies whether to use an Internal or External Clock. See Clock Selection section for more information.

PRELIMINARY

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 allows the user to use the predefined TX, RX ISR of the SPI Master component. The user 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 the user selects a RX or TX buffer size greater than 4 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 useable, 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 the user 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 bit's chosen with these parameters defines the mask implemented at the initial configuration of this component.

Enable Multi-Slave mode

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

Clock Selection

The external clock input to the SPI Slave is only fed to the status register. It is recommended that this clock frequency will be set to 2x of the bit rate (for example it should be set to 4 MHz for 2 Mbit/s bit rate). The bit rate is defined from the sclk input from the master device.

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

Placement

The SPI Slave component is placed 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 Slave 8-bit	2	*	2	1	1			*
SPI Slave 16-bit	4	*	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 "SPIS_1" to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "SPIS."

Function	Description
SPIS_Init	Initializes/restores default SPIS configuration provided with the customizer.
SPIS_Enable	Enable the SPIS operation.
SPIS_Start	Enable the SPIS operation.
SPIS_Stop	Disable the SPIS operation.
SPIS_EnableTxInt	Enables the internal TX interrupt irq.
SPIS_EnableRxInt	Enables the internal RX interrupt irq.
SPIS_DisableTxInt	Disables the internal TX interrupt irq
SPIS_DisableRxInt	Disables the internal RX interrupt irq
SPIS_SetTxInterruptMode	Configures the TX interrupt sources enabled
SPIS_SetRxInterruptMode	Configures the RX interrupt sources enabled
SPIS_ReadTxStatus	Returns the current state of the TX status register
SPIS_ReadRxStatus	Returns the current state of the RX status register
SPIS_WriteTxData	Places a byte/word in the transmit buffer which will be sent at the next available bus time
SPIS_WriteTxDataZero	Places a byte/word in the shift register directly. This is required for SPI Modes 00 and 01.
SPIS_ReadRxData	Returns the next byte/word of received data
SPIS_GetRxBufferSize	Returns the size (in bytes/words) of the RX memory buffer
SPIS_GetTxBufferSize	Returns the size (in bytes/words) of the TX memory buffer
SPIS_ClearRxBuffer	Clears the memory array of all received data
SPIS_ClearTxBuffer	Clears the memory array of all transmit data
SPIS_TxEnable	Enables the TX portion of the SPI Slave (MOSI)
SPIS_TxDisable	Disables the TX portion of the SPI Slave (MOSI)
SPIS_PutArray	Places an array of data into the transmit buffer

PRELIMINARY

Function	Description
SPIS_ClearFIFO	Clears any received data from the RX FIFO
SPIS_SaveConfig	Saves SPIS configuration.
SPIS_RestoreConfig	Restores SPIS configuration.
SPIS_Sleep	Prepare SPIS Component goes to sleep.
SPIS_Wakeup	Prepare SPIS Component to wake up.

Global Variables

Function	Description
SPIS_initVar	Indicates whether the SPI Slave has been initialized. The variable is initialized to 0 and set to 1 the first time SPIS_Start() is called. This allows the component to restart without reinitialization after the first call to the SPIS_Start() routine. If reinitialization of the component is required, then the SPIS_Init() function can be called before the SPIS_Start() or SPIS_Enable() function.
SPIS_txBufferWrite	Amount of bytes/words written to TX Software Buffer is stored in this variable.
SPIS_txBufferRead	Amount of bytes/words reading from TX Software Buffer is stored in this variable.
SPIS_rxBufferWrite	Amount of bytes/words written to RX Software Buffer is stored in this variable.
SPIS_rxBufferRead	Amount of bytes/words reading from RX Software Buffer is stored in this variable.
SPIS_rxBufferFull	Indicate the Software Buffer overflow
SPIS_RXBUFFER[]	Used to store data to sending.
SPIS_TXBUFFER[]	used to store received data

void SPIS_Init(void)

Description: Initializes/Restores default SPIS 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 SPIS_Enable(void)

Description:	Enable SPIS Component.
Parameters:	None
Return Value:	None
Side Effects:	None

void SPIS_Start(void)

Description:	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 SPIS_Stop(void)

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

void SPIS_EnableTxInt (void)

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

void SPIS_EnableRxInt (void)

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

void SPIS_DisableTxInt (void)

Description: Disables the internal TX interrupt irq

Parameters: Void

Return Value: Void

Side Effects: None

void SPIS_DisableRxInt (void)

Description: Disables the internal RX interrupt irq

Parameters: Void

Return Value: Void

Side Effects: None

void SPIS_SetTxInterruptMode (uint8 intSrc)

Description: Configures the TX interrupt sources enabled

Parameters: uint8 intSrc: Bit-Field containing the interrupts to enable. 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 SPIS_SetRxInterruptMode (uint8 intSrc)

Description: Configures the RX interrupt sources enabled

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 SPIS_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: Contents of the TX status register.
Side Effects:	TX status register bits are clear on read.

uint8 SPIS_ReadRxStatus (void)

Description:	Returns the current state of the RX status register. For more information see the Status Register Bits section.
Parameters:	Void
Return Value:	uint8: Current RX status register value
Side Effects:	RX status register bits are clear on read.

void SPIS_WriteTxData (uint8/uint16 txData)

Description:	Places a byte in the transmit buffer which will be sent at the next available bus time
Parameters:	uint8/uint16: txData: The data value to send across the SPI
Return Value:	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/word is finished, then the next byte/word will be appended to the transfer with no time between the byte transfers. Clear status register of the component.

void SPIS_WriteTxDataZero (uint8/uint16 txData)

Description:	Places a byte/word directly into the shift register for transmit which will be sent during the next clock phase from the master device
Parameters:	uint8/uint16: txData: The data value to send across the SPI
Return Value:	Void
Side Effects:	Required for Modes 0 and 1 (CPHA == 0) where data must be in the shift register before the first clock edge. Firmware must control this if there is already data being shifted out and if there is more data in the FIFO. This routine should not to be used for Mode 2 or Mode 3 where CPHA == 1.

PRELIMINARY

uint8/uint16 SPIS_ReadRxData (void)

Description: Read the next byte of data received across the SPI.

Parameters: Void

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

uint8 SPIS_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: None

uint8 SPIS_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 SPIS_ClearRxBuffer (void)

Description: Clears the memory array 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 SPIS_ClearTxBuffer (void)

Description:	Clears the memory array of all transmit data. Clear the TX RAM buffer by setting the read and write pointers both to zero. Setting the pointers to zero makes the system believe there is no data to read and writing will resume at address 0 overwriting any data that may have remained in the RAM.
Parameters:	void
Return Value:	void
Side Effects:	Any data not yet transmitted from the RAM buffer will be lost when overwritten.

void SPIS_TxEnable (void)

Description:	If the SPI Slave 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 SPIS_TxDisable (void)

Description:	If the SPI Slave 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 SPIS_PutArray (uint8 *buffer, uint8 byteCount)

Description:	Write available data from RAM/ROM to the TX buffer while space is available. Keep trying until all data is passed to the TX buffer. If using Mode 00 or 01, call the WriteTxDataZero() function before calling the PutArray() function.
Parameters:	uint8 *buffer: Pointer to the location in RAM containing the data to send
	uint8 byteCount: The number of bytes/words to move to the transmit buffer
Return Value:	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 SPIS_ClearFIFO (void)

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

void SPIS_SaveConfig (void)

Description:	Saves SPIS configuration.
Parameters:	void
Return Value:	void
Side Effects:	None

void SPIS_RestoreConfig (void)

Description:	Restores SPIS configuration.
Parameters:	void
Return Value:	void
Side Effects:	If this API is called without first calling SaveConfig then in the following registers will be default values from Customizer: SPIS_STATUS_MASK_REG and SPIS_COUNTER_PERIOD_REG.

void SPIS_Sleep (void)

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

void SPIS_Wakeup (void)

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

Defines

SPIS_TX_INIT_INTERRUPTS_MASK

Defines the initial configuration of the interrupt sources chosen in the Configure dialog. This is a mask of the bits in the TX status register that have been enabled at configuration as sources for the interrupt.

SPIS_RX_INIT_INTERRUPTS_MASK

Defines the initial configuration of the interrupt sources chosen in the Configure dialog. This is a mask of the bits in the RX status register that have been enabled at configuration as sources for the interrupt.

Status Register Bits

Table 1 SPIS_TXSTATUS

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

Table 2 SPIS_RXSTATUS

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

- Byte/Word Complete: Set when a byte/word 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 Full: Set when the RX Data FIFO is full (Does not indicate the RAM array conditions)
- RX FIFO Empty: Set when the RX Data FIFO is empty (Does not indicate the RAM array conditions)
- RX FIFO Not Empty: Set when the RX Data FIFO is not empty [that is, at least one byte/word is in the RX FIFO (does not indicate the RAM array conditions)].
- TX FIFO Empty: Set when the TX Data FIFO is empty (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):
- 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.

SPIS_TXBUFFERSIZE

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

SPIS_RXBUFFERSIZE

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

SPIS_DATAWIDTH

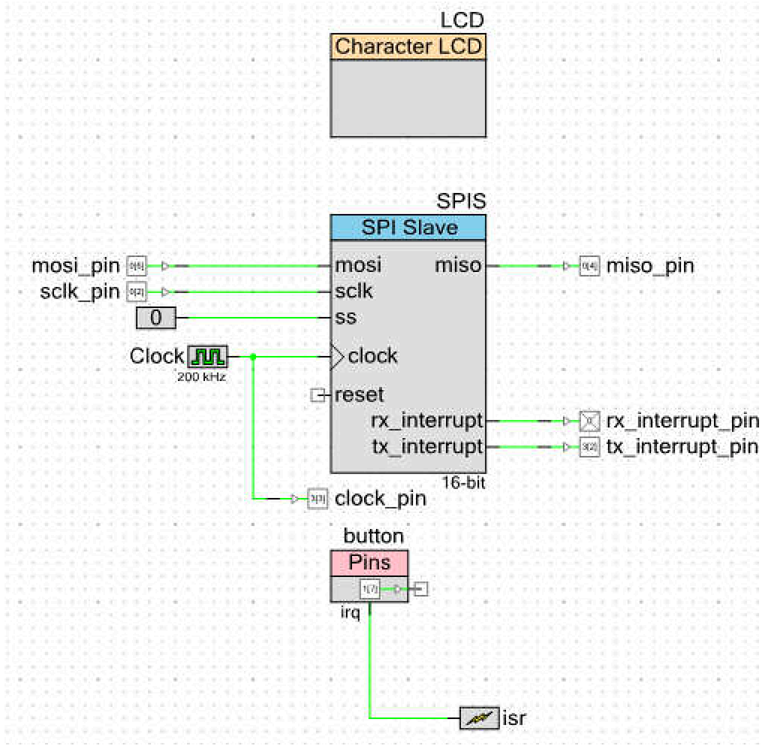
Defines the number of bits per data transfer chosen by the user.

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the SPI Slave component. This example assumes the component has been placed in a design with the default name "SPIS_1."

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 Slave configuration:

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

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

PRELIMINARY

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

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 Master when an SPI Slave can turn into unexpected active state. 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.

```
#include <device.h>

uint8 volatile startVar = 1;

void main()
{
    CYGlobalIntEnable;

    isr_Start();

    LCD_Start();
    LCD_ClearDisplay();

    while (startVar);

    button_ClearInterrupt();
    isr_ClearPending();

    SPIS_Start();

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

    SPIS_WriteTxData(0x2221);
    SPIS_WriteTxData(0x2222);
    SPIS_WriteTxData(0x2223);
    SPIS_WriteTxData(0x2224);
    SPIS_WriteTxData(0x2225);
    SPIS_WriteTxData(0x2226);
    SPIS_WriteTxData(0x2227);
    SPIS_WriteTxData(0x2228);

    while((SPIS_ReadTxStatus() & SPIS_STS_SPI_DONE) != SPIS_STS_SPI_DONE);

    LCD_Position(0,0);
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_ReadRxData());
    LCD_PrintHexUint16(SPIS_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

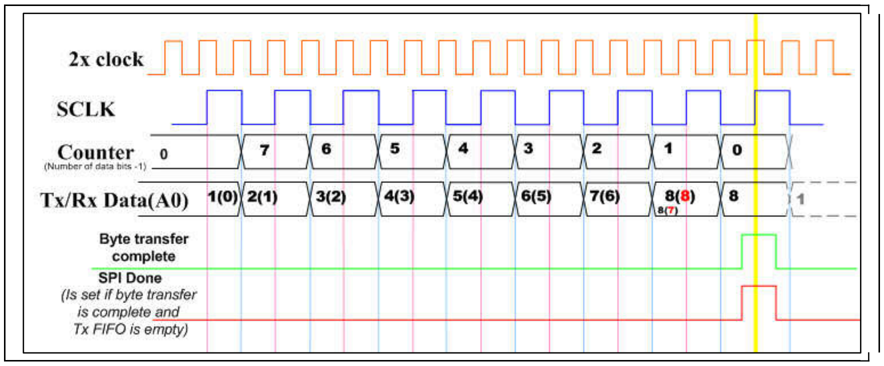
Default Configuration

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

Modes

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

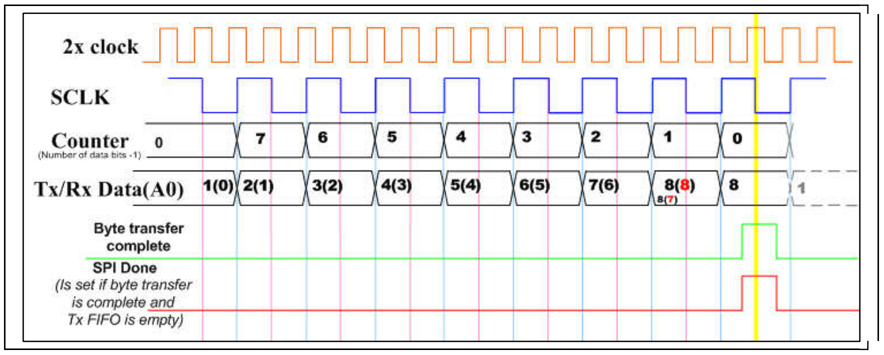
Mode 0 has the following characteristics:



PRELIMINARY

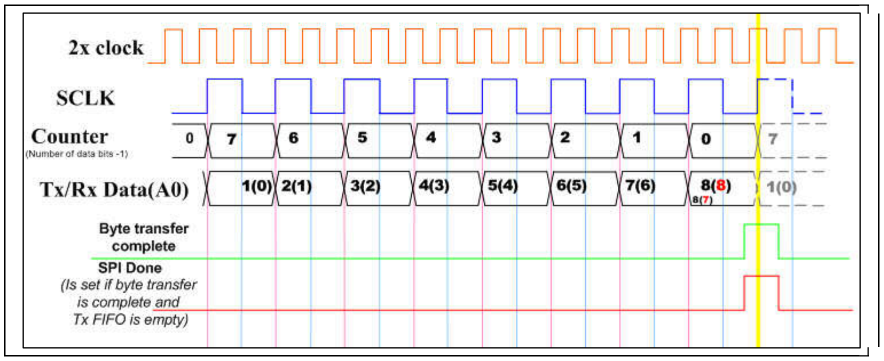
SPIS Mode: 1 (CPHA == 0, CPOL == 1)

Mode 1 has the following characteristics:



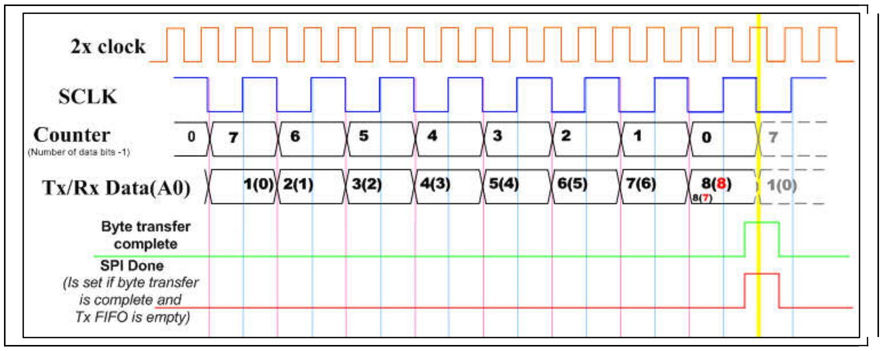
SPIS Mode: 2 (CPHA == 1, CPOL == 0)

Mode 2 has the following characteristics:



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

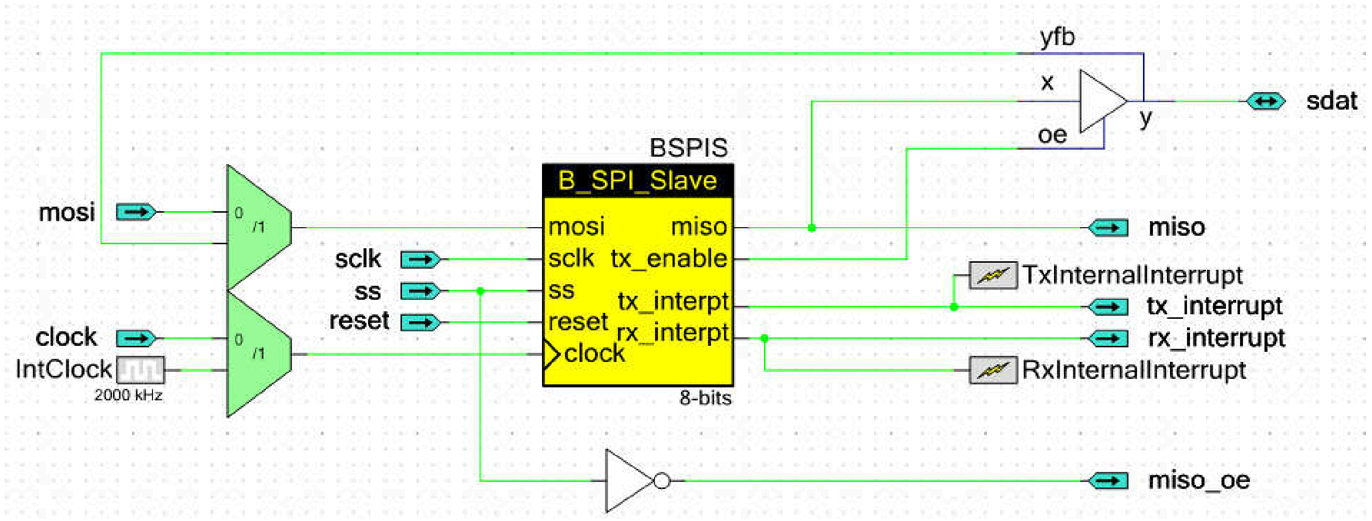
Mode 3 has the following characteristics:



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

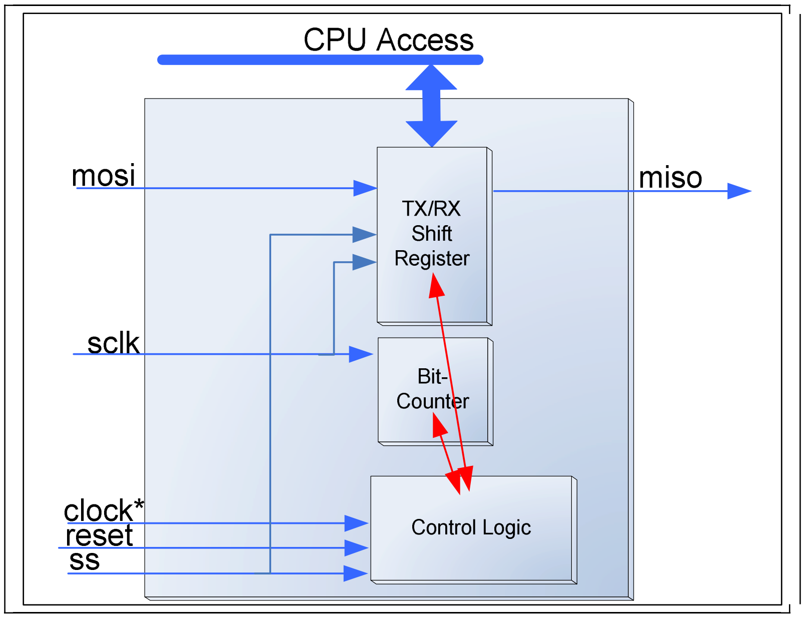
Block Diagram and Configuration

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



The implementation is described in the following block diagram.

Figure 4 UDB Implementation



PRELIMINARY

Registers

Status TX

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

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

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

Status RX

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

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

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

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

TX Data

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

RX Data

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

Conditional Compilation Information

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

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

References

Not applicable

DC and AC Electrical Characteristics

The following values 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 outputs have been changed: • The reset input was added; • The interrupt output was removed; rx_interrupt, tx_interrupt outputs were added instead.	PSoC 3 ES3 reset functionality was added. Two status interrupt registers (Tx and Rx) are now presented instead of one shared. The changes must be taken into account to prevent binding errors when migrating from previous SPI versions
	Removed _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.
	Renamed ReadByte(), WriteByte(), and WriteByteZero()APIs to ReadRxData(), WriteTxData(), WriteTxDataZero().	Clarifies the APIs and how they should be used.

Version	Description of Changes	Reason for Changes / Impact
The following changes were made to the base SPI Slave component B_SPI_Slave_v2_0, which is implemented using Verilog:		
	B_SPI_Slave_v2_0 now contains two separate implementations for ES2 and ES3 silicon. One datapath is used for 8-bit SPI for Tx and Rx in ES3 silicon instead of two in ES2.	Requirement that all components support ES2 and ES3 silicon. Use of ES3 feature updates is a requirement and this helps optimize resource usage in ES3. .
	Changes in ES2 support implementation: Two statusi registers are now presented (status are separate on Tx and Rx) instead of using one common status register for both.	This provides correct software buffer functionality.
	'BidirectMode' boolean parameter is added to base component (Verilog implementation). Control Register with 'clock' input and SYNC mode bit is now selected to drive 'tx_enable' output for ES3 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.	Added Bidirectional Mode support to the component
	Four udb_clock_enable components are added to Verilog implementation with sync = \`TRUE\` parameter. One of them has sync = \`TRUE\` (status register clock), other three have sync = \`FALSE\`	New requirements for all clocks used in Verilog to indicate functionality so the tool can support synchronization and Static Timing Analysis.
	Rx datapath configuration changed 'FIFO FAST' option is set to 'DP' instead of 'BUS'	Fixes a defect in previous versions of the component where there was a timing window affecting correct component capture of data.
	Additional logic to Verilog implementation to change the moments when SPI DONE and BYTE COMPLETE status are generated.	Fixes a defect in previous versions of the component where SPI DONE is sometimes not generated even after communication is complete.