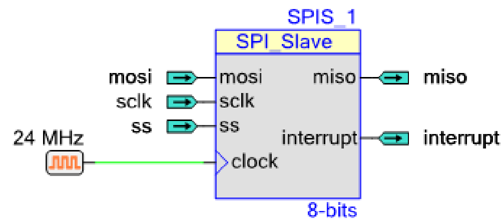
Serial Peripheral Interface (SPI) Slave

1.0

Features

- 2 to 16-bit Data Width
- 4 SPI Modes
- Data Rates to 33Mb/s



General Description

The SPI Slave provides an industry-standard 4-wire Slave SPI interface. The interface supports all 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 and MOSI pins and 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's states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

clock – Input

The clock input defines the sampling rate of the status register. All data clocking happens on sclk so the clock input DOES NOT handle the bit-rate of the SPI Slave. This input is always visible and must be connected.

Cypress Semiconductor Corporation • 198 Champion Court • San Jose, CA 95134-1709 • 408-943-2600
Document Number: 001-49273 Rev. ** Revised March 31, 2009

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.

mosi – Input

The mosi input carries the master output – slave input serial data from the master device on the bus. This input is always visible and must be connected.

sclk– Input

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

ss – Input

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

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.

Parameters and Setup

Drag an SPI Slave component onto your design and double-click it to open the Configure dialog.

Figure 1 Configure SPI Slave Basic Dialog

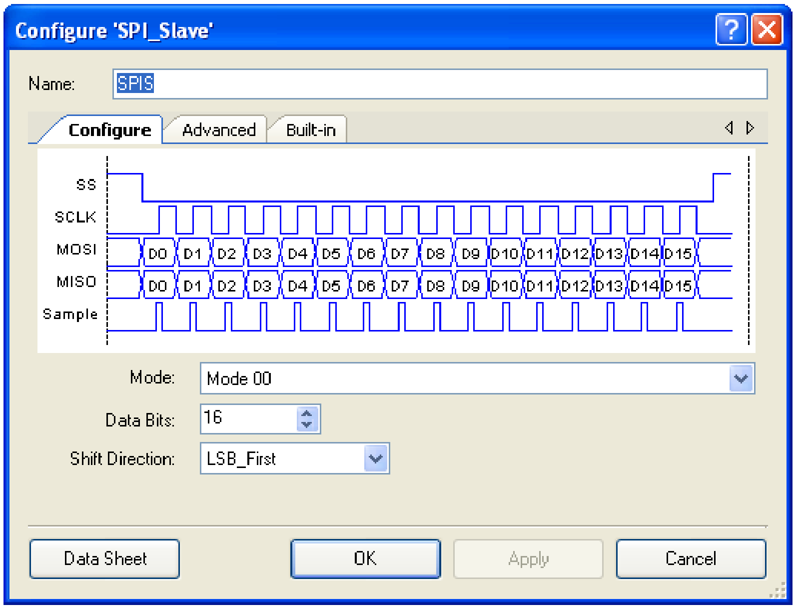
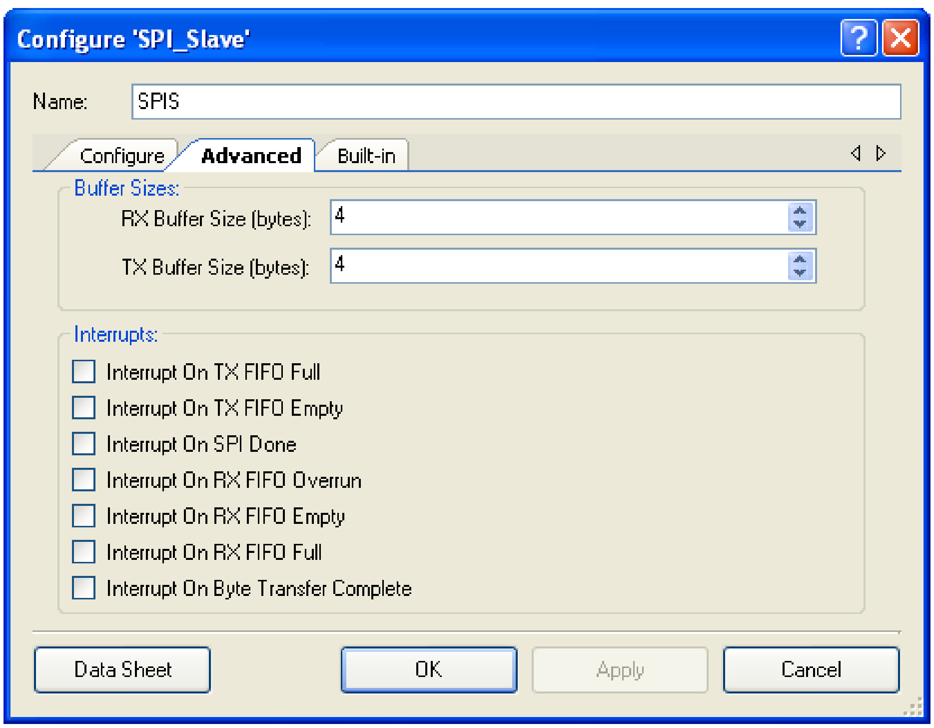


Figure 2 Configure SPI Slave Advanced Dialog



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.

Basic Tab

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

Mode (enum) *

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.

Data Bits (uint8) *

The number of data bits defines the bit-width of a single transfer as transferred with the WriteByte() and ReadByte() API. The default number of bits is a single byte (8-bits). Any integer from 2 to 16 may be selected

Shift Direction (enum) *

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

RxBufferSize (uint8) *

The RX Buffer Size parameter defines the size (in bytes) of memory allocated for a circular data buffer. If this parameter is set to 1 a single byte FIFO is implemented in the hardware. If the parameter is set to 2-4 then the 4-byte FIFO is implemented in hardware. All other values up to 255 (8-bit Processor) or 64535 (32-bit Processor) will use the 4-byte FIFO and a memory array controlled by the supplied API. The default value is 8.

TxBufferSize (uint8) *

The TX Buffer Size parameter defines the size (in bytes) of memory allocated for a circular data buffer. If this parameter is set to 1 a single byte FIFO is implemented in the hardware. If the parameter is set to 2-4 then the 4-byte FIFO is implemented in hardware. All other values up to 255 (8-bit Processor) or 64535 (32-bit Processor) will use the 4-byte FIFO and a memory array controlled by the supplied API. The default value is 8.

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 status register bits. The bit's chosen with these parameters defines the mask implemented at the initial configuration of this component.

Clock Selection

The external clock input to the SPI Slave is only fed to the status register. The Bit-Rate is defined from the sclk input from the master device.

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

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
SPI Slave 8-bit	2	*	1	0	1			*
SPI Slave 16-bit	4	*	1	0	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
void SPIS_Start(void)	Enable the SPIS operation.
void SPIS_Stop(void)	Disable the SPIS operation.
void SPIS_EnableInt (void)	Enables the internal interrupt irq.
void SPIS_DisableInt (void)	Disables the internal interrupt irq
void SPIS_SetInterruptMode (uint8)	Configures the interrupt sources enabled
uint8 SPIS_ReadStatus (void)	Returns the current state of the status register
void SPIS_WriteByte (uint8/16)	Places a byte in the transmit buffer which will be sent at the next available bus time
void SPIS_WriteByteZero(uint8/16)	Places a byte in the shift register directly. This is required for SPI Modes 00 and 01.
uint8/16 SPIS_ReadByte (void)	Returns the next byte of received data
uint8/uint16 SPIS_GetRxBufferSize (void)	Returns the size (in bytes) of the RX memory buffer
uint8/uint16 SPIS_GetTxBufferSize (void)	Returns the size (in bytes) of the TX memory buffer
void SPIS_ClearRxBuffer (void)	Clears the memory array of all received data
void SPIS_ClearTxBuffer (void)	Clears the memory array of all transmit data
void SPIS_TxEnable (void)	Enables the TX portion of the SPI Master (MOSI)
void SPIS_TxDisable (void)	Disables the TX portion of the SPI Master (MOSI)
void SPIS_PutArray (uint16* RamString, uint8 ByteCount)	Places an array of data into the transmit buffer
void SPIS_ClearFIFO(void)	Clears any received data from the RX FIFO

void SPIS_Start(void)

Description: Only necessary for initial configuration.

Parameters: void

Return Value: void

Side Effects: The first time this function is called it initializes all of the necessary parameters for execution. i.e. setting the initial interrupt mask, configuring the interrupt service routine, configuring the bit-counter parameters and clearing the RX FIFO

void SPIS_Stop(void)

Description: Has no affect on the SPIS operation

Parameters: void

Return Value: void

Side Effects: None

void SPIS_EnableInt (void)

Description: Enables the internal interrupt irq

Parameters: void

Return Value: void

Side Effects: None

void SPIS_DisableInt (void)

Description: Disables the internal interrupt irq

Parameters: void

Return Value: void

Side Effects: None

void SPIS_SetInterruptMode (uint8)

Description: Configures the interrupt sources enabled

Parameters: uint8: Bit-Field containing the interrupts to enable. Based on the bit-field arrangement of the status register. This value must be a combination of status register bit-masks defined in the header file.

Return Value: void

Side Effects: None

uint8 SPIS_ReadStatus (void)

Description: Returns the current state of the status register

Parameters: void

Return Value: uint8: Current status register value

Side Effects: Status register bits are clear on read.

void SPIS_WriteByte (uint8/16)

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

Parameters: uint8/16: data byte

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.

void SPIS_WriteByteZero (uint8/16)

Description: Places a byte directly into the shift register for transmit which will be sent during the next clock phase from the master device

Parameters: uint8/16: data byte

Return Value: void

Side Effects: Required for Modes 00 and 01 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.

uint8/16 SPIS_ReadByte (void)

Description: Returns the next byte of received data

Parameters: void

Return Value: uint8/16: data byte

Side Effects: This function blocks until there is data in the input memory buffer.

uint8/uint16 SPIS_GetRxBufferSize (void)

Description: Returns the size (in bytes) of the available space in the RX memory buffer

Parameters: void

Return Value: uint8/uint16: Buffers size (in bytes)

Side Effects: None

uint8/uint16 SPIS_GetTxBufferSize (void)

Description: Returns the size (in bytes) of the available space in the TX memory buffer

Parameters: void

Return Value: uint8/uint16: Buffers size (in bytes)

Side Effects: None

void SPIS_ClearRxBuffer (void)

Description: Clears the memory array of all received data

Parameters: void

Return Value: void

Side Effects: None

void SPIS_ClearTxBuffer (void)

Description: Clears the memory array of all transmit data

Parameters: void

Return Value: void

Side Effects: Will not clear data already placed in the TX FIFO.

void SPIS_TxEnable (void)

Description: Enables the TX portion of the SPI Slave (MISO)

Parameters: void

Return Value: void

Side Effects:

void SPIS_TxDisable (void)

Description: Disables the TX portion of the SPI Slave (MISO)

Parameters: void

Return Value: void

Side Effects: None

void SPIS_PutArray (uint16* RamString, uint8 ByteCount)

Description: Places an array of data into the transmit buffer

Parameters: uint16*: RamString – Location of the first byte of the data to move to the transmit buffer
uint8: Byte Count – Number of bytes in the array.

Return Value: void

Side Effects: None

Defines

SPIS_INIT _INTERRUPTS_MASK

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

Status Register Bits

Table 1 SPIS_STATUS

Bits	7	6	5	4	3	2	1	0
Value	Unused	Byte Complete	RX Buf. Overrun	RX FIFO Empty	RX FIFO Not Empty	TX FIFO Full	TX FIFO Not Full	SPI Done

- Byte Complete: Set when a Byte has been transmitted.
- RX Buffer Overrun: Set when RX Data has overrun the 4 byte FIFO or 1 Byte FIFO without being moved to the Memory array (if one exists)
- 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 full (Does not indicate the RAM array conditions)
- TX FIFO Full: Set when the TX Data FIFO is full (Does not indicate the RAM array conditions):
- TX FIFO Not Full: 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.

SPIS_TXBUFFERSIZE

Defines the amount of memory to allocate for the TX memory array buffer. This does not include the 4 bytes 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 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.

Mode 00 or Mode 01

```
#include <device.h>
#include "SPIS_1.h"

void main()
{
    uint8 i = 0;
    uint8 val[4];

    SPIS_1_Start();
    /* Preload the FIFO with TX data up to 4 bytes */
    SPIS_1_WriteByteZero(0xA3);
    SPIS_1_WriteByte(0xE7);
    SPIS_1_WriteByte(0x96);
    SPIS_1_WriteByte(0x28);

    /* Read the four bytes transmitted from the master */
    for(i=0;i<4;i++)
         val[i] = SPIS_1_ReadByte();
}
```

Mode 10 or Mode 11

```
#include <device.h>
#include "SPIS_1.h"

void main()
{
    uint8 i = 0;
    uint8 val[4];

    SPIS_1_Start();
    /* Preload the FIFO with TX data up to 4 bytes */
    SPIS_1_WriteByte(0xA3);
    SPIS_1_WriteByte(0xE7);
    SPIS_1_WriteByte(0x96);
    SPIS_1_WriteByte(0x28);

    /* Read the four bytes transmitted from the master */
    for(i=0;i<4;i++)
         val[i] = SPIS_1_ReadByte();
}
```

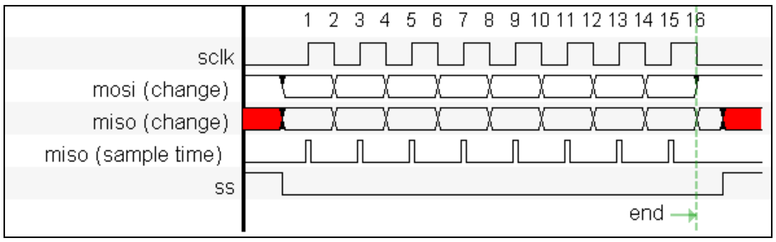
Functional Description

Default Configuration

The default configuration for the SPIS is as an 8-bit SPIS with Mode 00 configuration.

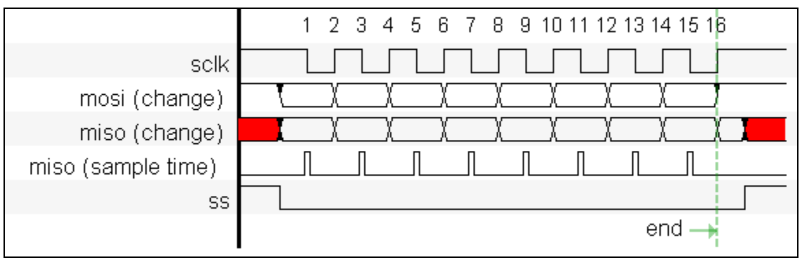
SPIS Mode: 00

Mode 00 defines the Clock Phase of 0 and the Clock Polarity of 0 which has the following characteristics:



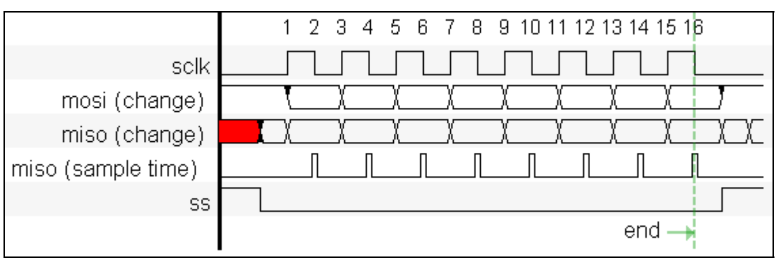
SPIS Mode: 01

Mode 01 defines the Clock Phase of 0 and the Clock Polarity of 1 which has the following characteristics:



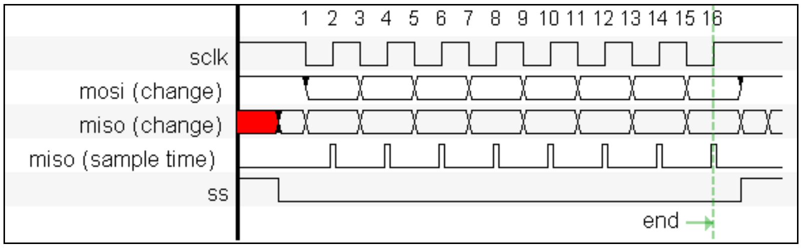
SPIS Mode: 10

Mode 10 defines the Clock Phase of 1 and a Clock Polarity of 0 which has the following characteristics:



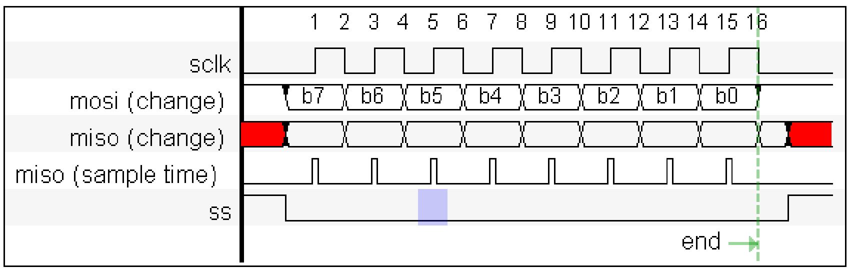
SPIS Mode: 11

Mode 11 defines the Clock Phase of 1 and a Clock Polarity of 1 which has the following characteristics:



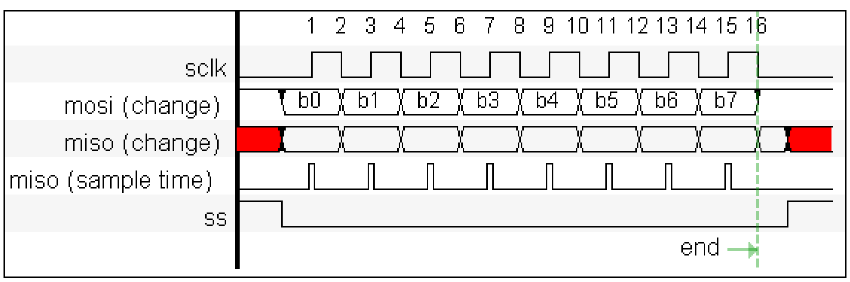
SPIS ShiftDir: MSB_First

When setting the Shift Direction parameter to MSB_First the data is shifted out Most Significant bit first. For an 8-bit Transfer with Mode 00 the transfer looks like this:



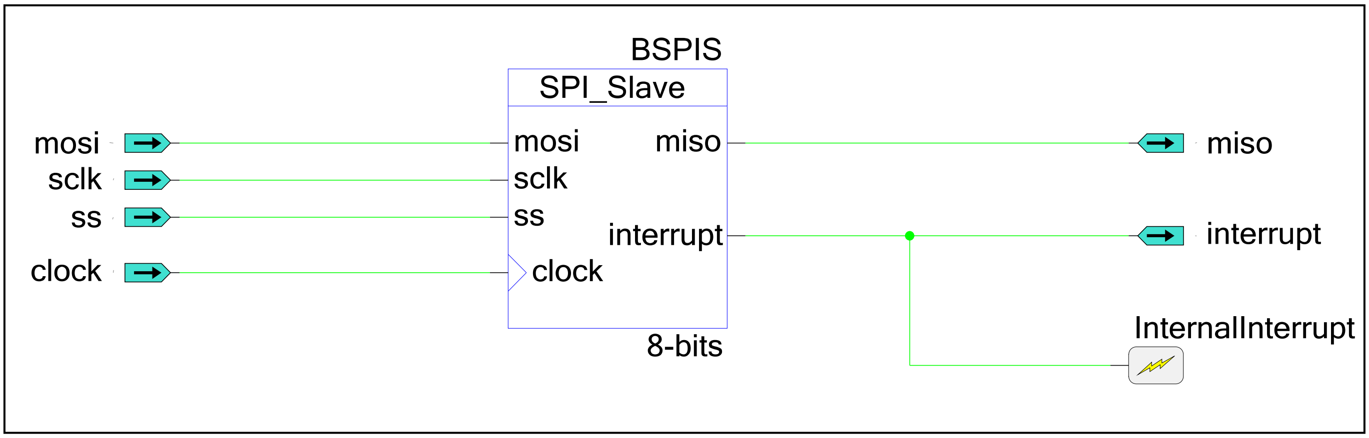
SPIS ShiftDir: LSB_First

When setting the Shift Direction parameter to LSB_First the data is shifted out Least Significant bit first. For an 8-bit Transfer with Mode 00 the transfer looks like this:



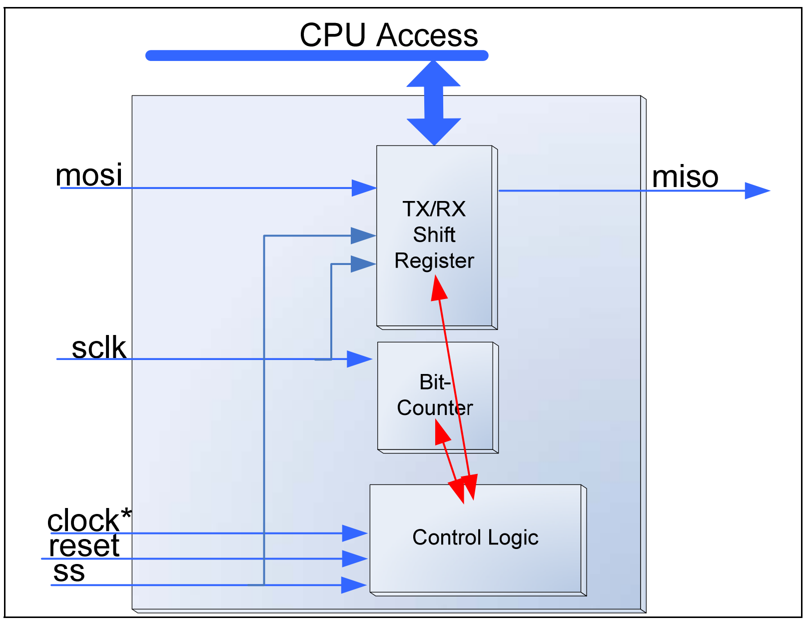
Block Diagram and Configuration

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 3 UDB Implementation



Registers

Status

The status register is a read only register which contains the various status bits defined for the SPIS. The value of this registers is available with the SPIS_ReadStatus() and function call. The interrupt output signal is generated from an ORing of the masked bit-fields within the status register. You can set the mask using the SPIS_SetInterruptMode() function call and upon receiving an interrupt you can retrieve the interrupt source by reading the Status register with the

SPIS_GetInterruptSource() function call. The Status register is clear on read so the interrupt source is held until either of the SPIS_ReadStatus() or SPIS_GetInterruptSource() function is called. All operations on the status register must use the following defines for the bit-fields as these bit-fields may be moved around within the status register at build time.

There are several bit-fields masks defined for the 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 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_BUF_NOT_FULL

Defined as the bit-mask of the Status register bit “Transmit Buffer Empty”.

SPIS_STS_TX_BUF_FULL

Defined as the bit-mask of the Status register bit “Transmit Buffer Full”.

SPIS_STS_RX_BUF_NOT_EMPTY

Defined as the bit-mask of the Status register bit “Receive Buffer Full”.

SPIS_STS_RX_BUF_EMPTY

Defined as the bit-mask of the Status register bit “Receive Buffer Empty”.

SPIS_STS_RX_BUF_OVERRUN *

Defined as the bit-mask of the Status register bit “Receive Buffer Overrun”.

SPIS_STS_BYTE_COMPLETE *

Defined as the bit-mask of the Status register bit “Byte Complete”.

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 API dealing with the transmitting of 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 TX_DATA_ADDR 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 RX_DATA_ADDR 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	