

Introduction

The Serial Peripheral Interface (SPI) is used primarily for synchronous serial communication between a host processor and its peripherals. The SPI bus is often selected because of its low pin count and full-duplex mode that can achieve data throughput in the tens of Mbps range. The SPI bus uses a 4-wire interface with two unidirectional data lines to communicate between the master and the selected slave. It supports one master with multiple slaves on one bus and allows protocol flexibility for the bit transferred.

This reference design implements a SPI slave device interface that provides full-duplex, synchronous, serial communication with the SPI master. A simple back-end parallel interface provides the flexibility to interface with any system. This reference design instantly adds SPI bus capability to a device in an embedded system. It is available in both Verilog and VHDL languages.

Features

Although the SPI bus is not governed by any standard organization, most applications adhere to the features listed below.

- SPI slave with full-duplex capability.
- Supports 4-clock polarity and clock phase modes.
- Shares the SPI bus with other SPI slave devices.
- Receives and transmits registers configurable from 1 to 32 bits wide. Longer transfers can be done with software support.
- Double-buffered transmission allows new data to be written at the same time that previous data is being shifted out.
- Option for least-significant bit or most-significant bit first.

Functional Description

This reference design implements all the features commonly supported for SPI bus applications. When the SPI master initiates data transfer by pulling down the slave select signal and generating the clock, data can be transferred in either or both directions simultaneously. The SPI master and slave determine whether the received data is meaningful. Figure 1 provides an overview of the reference design's architecture, including the SPI and back-end interfaces. Table 1 lists the I/O port names of the SPI slave design.

Figure 1. Serial Peripheral Interface Block Diagram

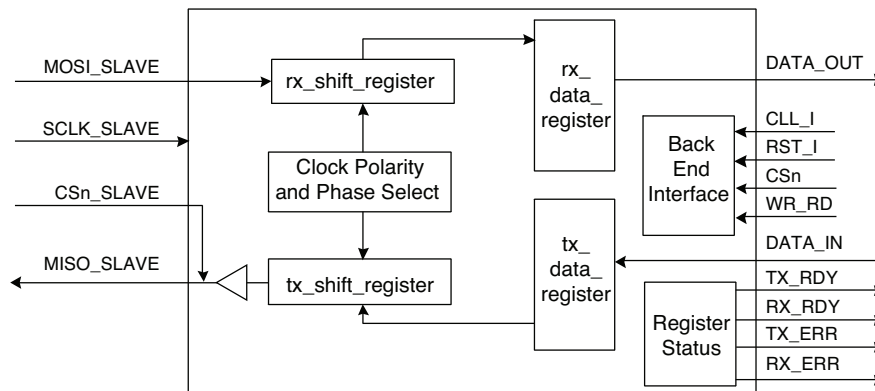


Table 1. SPI-GPIOr I/O Port Names

Signal Name	Signal Direction	Active State	Definition
SPI Master Interface			
SCLK_SLAVE	Input	N/A	SPI clock input from the master.
CSn_SLAVE	Input	Low	SPI slave select input.
MOSI_SLAVE	Input	N/A	Serial data input from the master.
MISO_SLAVE	Output	N/A	Serial data output to the master.
Back End Interface			
CLK_I	Input	N/A	Synchronous clock from back end device.
RST_I	Input	High	Asynchronous reset signal from back end device.
CSn	Input	Low	Chip select signal from the back end device.
WR_RD	Input	High:Low	Read/write signal from the back end device; '0' indicates the back end device writes data and '1' indicates the back end device reads data.
DATA_IN	Input	N/A	Parallel data from the back end device which is sent to the SPI master through the SPI interface.
DATA_OUT	Output	N/A	Parallel data to the back end device which is received from the SPI master through the SPI interface.
TX_RDY	Output	High	Transmit ready status: '1' = empty. Indicates that the transmit register is empty and can accept new data from the back end device.
RX_RDY	Output	High	Receive ready status: '1' = data available. Indicates that the receive register has data and is ready to be read by the back end device.
TX_ERR	Output	High	Transmit overrun error: '1' = error. Error indicates that the transmit register received new data before the previous data was moved to the shift register.
RX_ERR	Output	High	Receive overrun error: '1' = error. Error indicates that the receive register received new data before the previous data was read.

SPI Interface

This reference design implements a SPI slave function with full-duplex capability. Within the 4-wire interface, three wires are generated by the SPI master, thus becoming inputs to the SPI slave. The only SPI output signal from the SPI slave is the MISO_SLAVE pin. It is a tri-state output which becomes high impedance when the CSn_SLAVE pin is inactive. This function enables the SPI slave to share the SPI bus and SPI segments with other SPI slave devices.

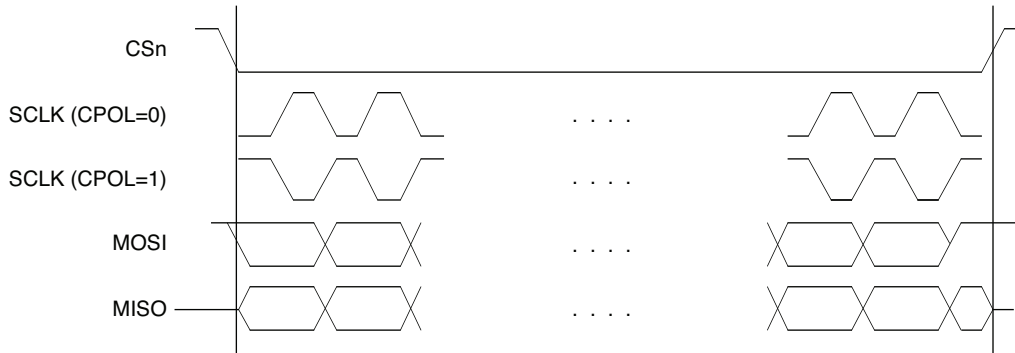
The SPI protocol has the flexibility of the length of bit transferred. This is achieved by configuring the rx_shift_register and tx_shift_register between 1 and 32 bits. Once the length of the bits transferred is specified, the SPI slave ignores any SPI communications in which the number of clock pulses is greater than the length of the bits transferred. The parameter DATA_LENGTH is used to control the number of bit transferred. This parameter also defines the bit width of the rx_data_register and the tx_data_register.

The rx_shift_register and tx_shift_register can also be configured either as the least-significant bit first or the most-significant bit first. The parameter SHIFT_DIRECTION can be used for this purpose.

The clock polarity and clock phase parameters of the SPI bus interface determine which edge of the clock signal will be used to drive or receive data. A master and slave must agree to use the clock polarity and clock phase mode for communication. This design supports all four SPI clock polarity and clock phase modes. The mode is represented as (x,y), where x is the clock polarity and y is the clock phase.

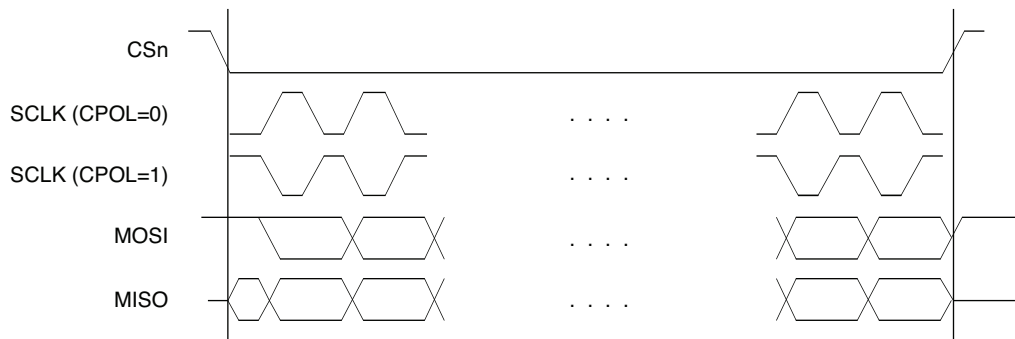
In mode (0,0), the data are sampled on the rising edge of clock and are updated on the falling edge. In mode (1,0), the data are sampled on the falling edge of clock and are updated on the rising edge, as shown in Figure 2.

Figure 2. Mode (0,0), Mode (1,0) Transfer Format



In mode (0,1), the data are sampled on the falling edge of clock and are updated on the rising edge. In mode (1,1), the data are sampled on the rising edge of clock and are updated on the falling edge, as shown in Figure 3.

Figure 3. Mode (0,1), Mode (1,1) Transfer Format



The parameters CLOCK_POLARITY and CLCOK_PHASE are used to specify the SPI mode. The parameter descriptions are summarized in Table 2.

Table 2. Parameters

Parameter	Description	Active Value	Default Value
DATA_LENGTH	Specifies the number of serial data bits.	1 to 32	8
SHIFT_DIRECTION	Specifies whether the most significant bit or least significant bit is first. 0 = MSB first. 1 = LSB first.	0, 1	0
CLOCK_POLARITY	Specifies the polarity of SCLK. If 0, the idle state for SCLK is low. If 1, the idle state for SCLK is high.	0, 1	0
CLOCK_PHASE	Specifies the SCLK phase. If 0, the data is latched on the leading edge of SCLK and data changes on the trailing edge of SCLK. If 1, the data is latched on the trailing edge of SCLK and data changes on the leading edge of SCLK.	0, 1	0

Back-end Interface

This reference design implements a simple back-end parallel interface. After the SPI master finishes sending serial data, the back-end interface converts the serial data to parallel data and sends it to the back-end device through the rx_data_register. Conversely, the back-end interface receives the parallel data from the back-end device through the tx_data_register and converts the parallel data to serial data to be transmitted through the SPI interface.

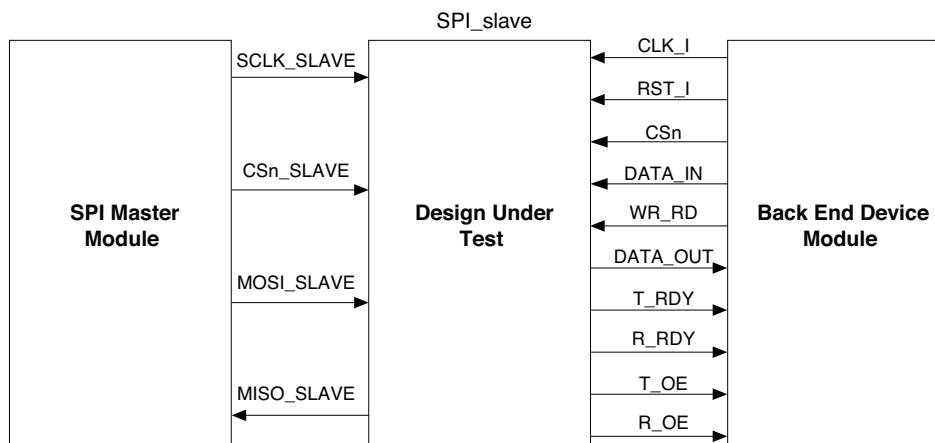
- **CLK_I** is the clock from the back-end device. Other signals in the back-end interface are synchronous to this clock. It is possible that this clock is asynchronous to the SCLK and the frequency of this clock must be greater than the frequency of SCLK.
- **RST_I** is the asynchronous reset signal from the back-end device.
- **CSn** is the chip select signal from the back-end device. Before the back-end device writes or reads, this signal must be activated.
- **WR_RD** is the read/write signal from the back-end device. Low indicates that the back-end device will implement a write operation and high indicates a read operation.
- **DATA_IN** is the parallel data from the back-end device. The bit width of this signal is defined by the parameter `DATA_LENGTH`.
- **DATA_OUT** is the parallel data to the back-end device. The bit width of this signal is defined by the parameter `DATA_LENGTH`.
- **TX_RDY** is the transmit ready status flag, output to the back-end device. When this signal is active, it indicates that the `tx_shift_register` is empty and can accept new data from the back-end device.
- **RX_RDY** is the receive ready status flag, output to the back-end device. When this signal is active, it indicates that the `rx_shift_register` has data and is ready to be read by the back-end device.
- **TX_ERR** is the transmit overrun error status flag, output to the back-end device. When this signal is active, it indicates that the `tx_shift_register` will receive new data before the previous data is moved to the SPI master. The new data from the back-end device is discarded if this occurs.
- **RX_ERR** is the receive overrun error status flag, output to the back-end device. When this signal is active, it indicates that the `rx_shift_register` will receive new data from the SPI master before the previous data is read by the back-end device. The previous data is lost if this occurs.

Test Bench Description

The test bench for this design consists the following functional blocks, as shown in Figure 4:

- SPI master module
- Back-end device module
- Design under test (SPI slave)

Figure 4. Test Bench Architecture



The parameters `DATA_LENGTH`, `SHIFT_DIRECTION`, `CLOCK_POLARITY` and `CLOCK_PHASE` are all set to their default values for the simulation. The SPI master initiates two SPI communications. In the first communication,

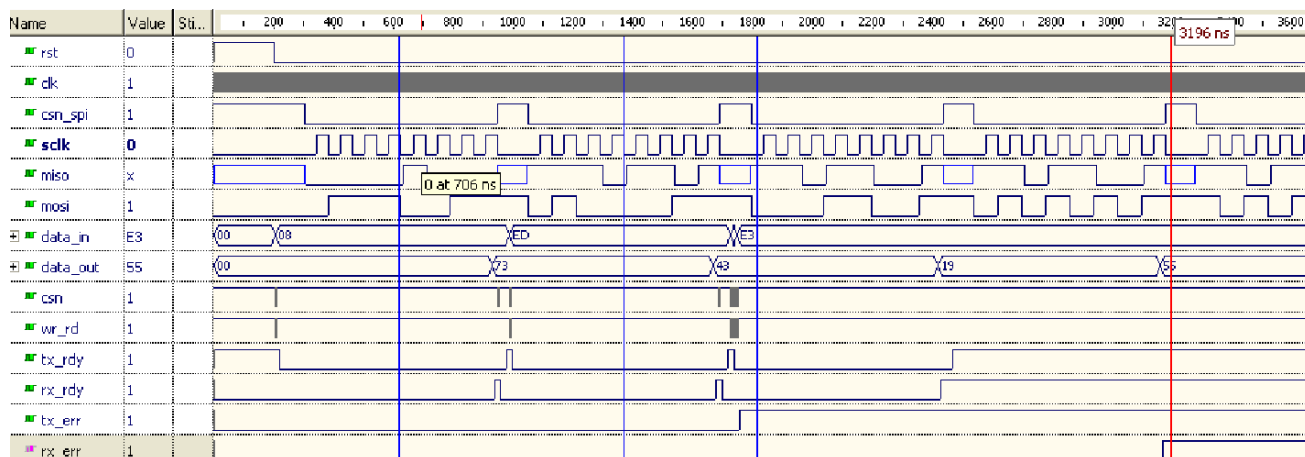
the SPI master sends 8-bit data 0x73 and the back-end device sends 8-bit data 0x08. When this data transfer is complete, the SPI master receives data 0x08 and the back-end device receives data 0x73.

In the second communication, the SPI master sends 8-bit data 0x43 and the back-end device sends 8-bit data 0xed. When the data transfer is complete, the SPI master receives data 0xed and the back-end device receives data 0x43.

After these two communications, the back-end device sends data before the tx_shift_register is empty, so the signal TX_ERR becomes active.

The SPI master sends data before the previous data is read by the back-end device, so the signal RX_ERR becomes active. Figure 5 shows the simulation results.

Figure 5. Simulation Results



Implementation

Table 3. Performance and Resource Utilization

Device Family	Language	Speed Grade	Utilization	f _{MAX} (MHz)	I/Os	Architecture Resources
MachXO™ 1	Verilog	-3	37 LUTs	>80	28	N/A
	VHDL	-3	37 LUTs	>80	28	N/A
ispMACH® 4000ZE ²	Verilog	-4 (ns)	60 Macrocells	>80	28	N/A
	VHDL	-4 (ns)	60 Macrocells	>80	28	N/A
Platform Manager™ 3	Verilog	-3	37 LUTs	>80	28	N/A
	VHDL	-3	37 LUTs	>80	28	N/A

1. Performance and utilization characteristics are generated using LCMXO640C-3T100C with Lattice Diamond™ 1.1 or ispLEVER® 8.1 SP1 software. When using this design in a different device, density, speed or grade, performance and utilization may vary.
2. Performance and utilization characteristics are generated using LC4064ZE-4TN100C with Lattice ispLEVER Classic 1.4 software. When using this design in a different device, density, speed or grade, performance and utilization may vary.
3. Performance and utilization characteristics are generated using LPTM10-12107-3FTG208CES, with ispLEVER 8.1 SP1 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
March 2010	01.0	Initial release.
December 2010	01.1	Added support for Platform Manager device family.
		Added support for Lattice Diamond 1.1 and ispLEVER 8.1 SP1 design software.