ESE 382 Digital System Design Using VHDL and PLDs

Spring 20, Ken Short
April 15, 2020 8:56 pm

Laboratory 8: SPI Transmitter

This laboratory is to be performed the week starting April 12th. Your report is due April 24th by 5:00 pm.

Prerequisite Reading

1. Chapter 10 of the text.
2. M68HC11E Family Data Sheet, Chapter 8 SPI.
3. Atmel ATmega128 Data Sheet, pages 165 through 173, Chapter 19 Serial Peripheral Interface.

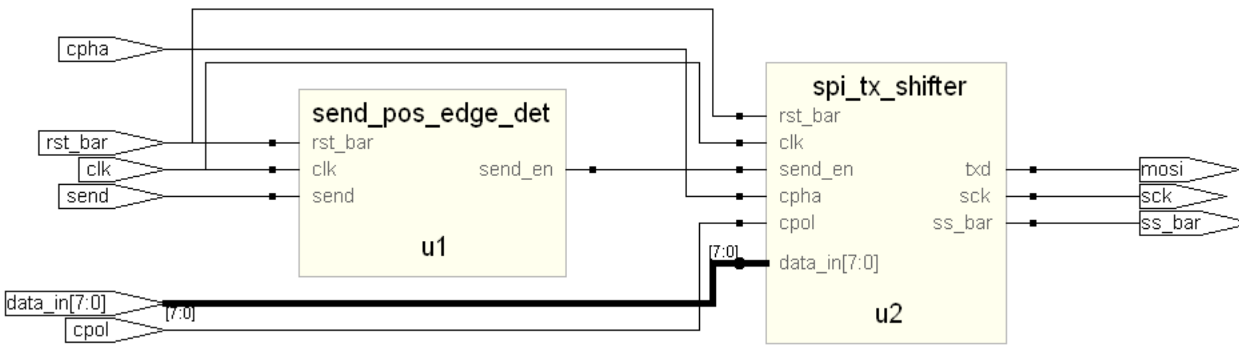
Purpose

The Serial Peripheral Interface (SPI) is a synchronous serial bus interface commonly used to transfer information between integrated circuits. A system implemented as a FPGA often has a SPI interface so that it can communicate with other ICs in a larger system. The project that you will implement over the next few laboratories involves the design of a SPI system module, implemented using VHDL and a FPGA. This module could be used stand alone to test other systems having SPI interfaces or it could be used as an entity in a much larger system that requires an SPI interface.

SPI is a de facto standard that was originally developed by Motorola. As a de facto standard, there is no official standard document. A practical reference to define SPI is a chapter in a Motorola microcontroller user's manual, such as reference 2 above. Another reference that we will use is the Atmel ATmega128 microcontroller data sheet, reference 3 above. We will most closely follow the signal and bit names used in the Atmel data sheet.

This laboratory focuses on implementing part of the transmission functionality of an SPI master. The next few laboratories add components until we have a complete system.

A block diagram of the system for this laboratory is:



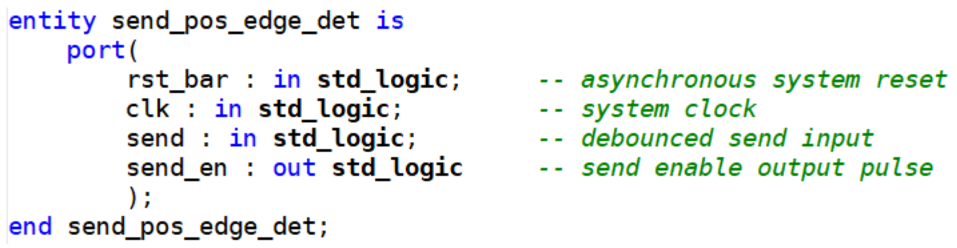
This system consists of two components and a top-level structural architecture.

Design Tasks

Design Task 1: Send Positive Edge Detector

A debounced pushbutton is to be used to provide a positive pulse `send` signal to indicate when the system is to serially transmit a byte of data. The debounced pushbutton generates this pulse for as long as it is pressed. Thus, the pulse may be very long in comparison to the time it takes to transmit the entire byte of data. If the system were to use the level of the `send` signal to determine when to transmit the data byte, it could end up sending the same byte multiple times for one press of the pushbutton. To eliminate this possibility, a positive edge detector is used to detect the positive edge of the `send` and to generate a narrow pulse that will be used by the `spi_tx_shifter` to determine when it should start to send a byte.

The entity declaration for the edge detector is:



```
entity send_pos_edge_det is
    port(
        rst_bar : in std_logic;     -- asynchronous system reset
        clk : in std_logic;         -- system clock
        send : in std_logic;        -- debounced send input
        send_en : out std_logic     -- send enable output pulse
        );
end send_pos_edge_det;
```

Use the Moore FSM description of a positive edge detector in Listing 10.4.1 of the text as a basis for your design. Write a non self-checking testbench to provide a single long `send` pulse to the `send_pos_edge_det` entity to verify its functionality. Make the duration of the `send` pulse about 200 ns. Make the clock period 40 ns. See the testbench provided for Laboratory 7 Task 3 for the code for a system clock and a reset. Simulate your design using your testbench.

Submit your properly commented design description source file and non-self-checking testbench, list of circuit features to be verified, description of required stimulus waveforms, and an annotated waveform editor output showing what is being verified during each portion of the waveforms.

Design Task 2: SPI Transmit Shifter

The transmitter shifter `spi_tx_shifter` converts the parallel data byte `data_in` to serial and transmits its data bits along with a synchronous clock `sck`. An SPI slave uses `sck` to determine when to sample each serial data bit. For now, we will transmit the data most significant bit first. In a later laboratory, the data order (whether the least or most significant bit is transferred first) will be configurable.

Inputs `cpol` and `cpha` determine the clock polarity and clock phase of the transmitted data, respectively. `cpol` determines whether the clock idles at 0 or 1. `cpha` determines whether the data must be sampled on the leading or trailing edge of the shift clock. The values of `cpol` and `cpha` are set to be compatible with the SPI slave that receives the data. An important assumption that is made in this design is that `data_in`, `cpol`, and `cpha` are set on switches before the send pushbutton is pressed and that these switches are not changed in the middle of a transmission.

The figure that follows is from the Atmel ATmega128 reference and illustrates the interpretation of `cpol` and `cpha`.

Figure 1. SPI Transfer Format with CPHA = 0

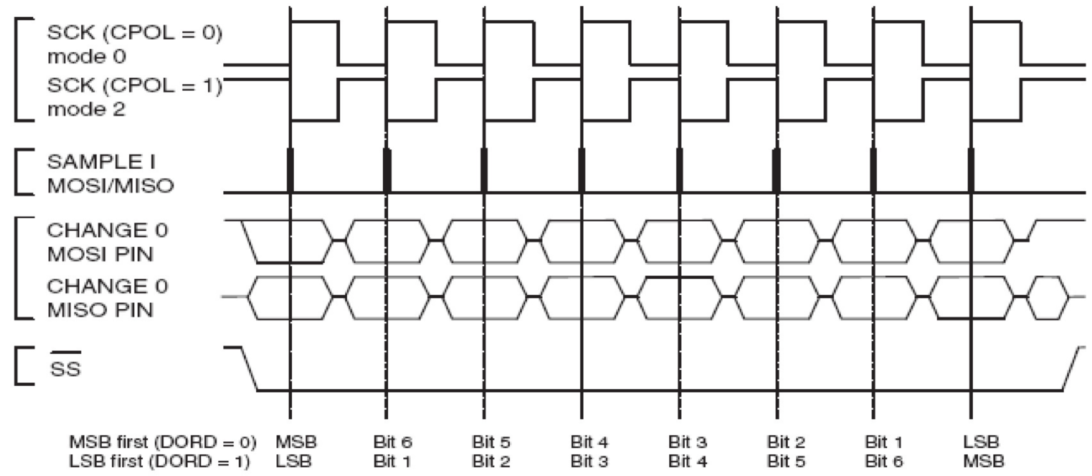
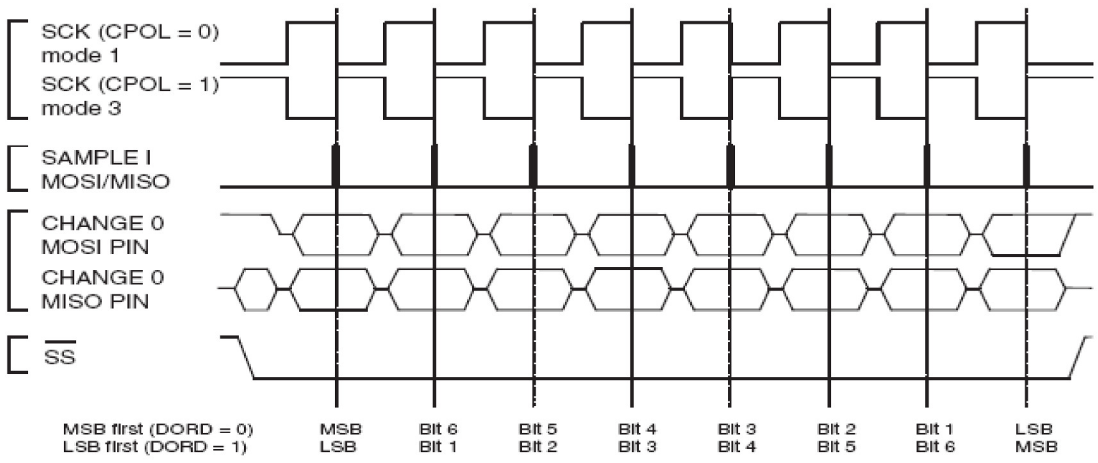


Figure 2. SPI Transfer Format with CPHA = 1



For design purposes, each clock period for `sck` can be divided into two phases, `ph1` and `ph2`. Data is sampled by the receiver at the end of `ph1` and is shifted by the transmitter at the end of `ph2`.

The entity declaration for `spi_tx_shifter` is:

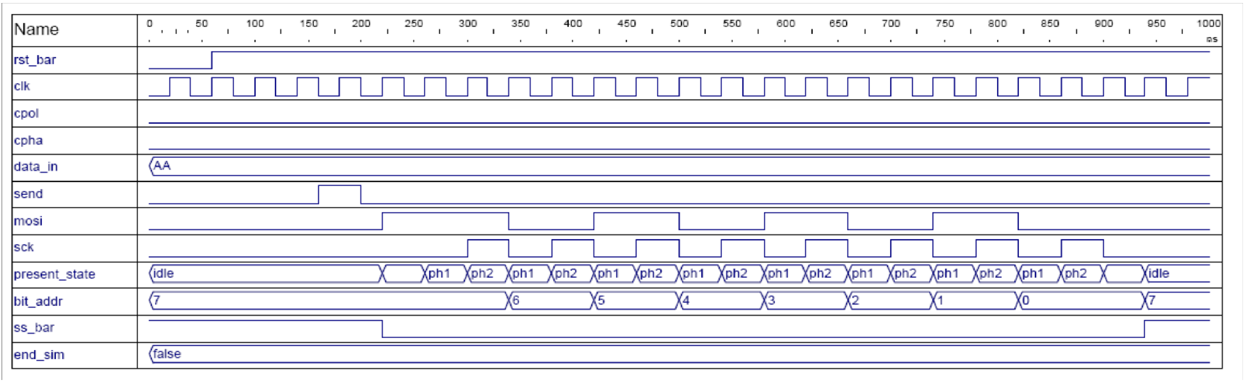
```
entity spi_tx_shifter is
    port(
        rst_bar : in std_logic;      -- asynchronous system reset
        clk : in std_logic;          -- system clock
        send_en : in std_logic;      -- enable data transmission
        cpha : in std_logic;         -- clock phase
        cpol : in std_logic;         -- clock polarity
        data_in : in std_logic_vector(7 downto 0);  -- data to send
        txd : out std_logic;         -- serial output data
        sck : out std_logic;         -- synchronous shift clock
        ss_bar : out std_logic       -- slave select
        );
end spi_tx_shifter;
```

Since output sck is the sample clock used by the slave, it must be a registered output to eliminate any possibility of glitches. See the discussion on pages 402 and 403 of the text.

Write a design description of an FSM that implements spi_tx_shifter. The description must be a three-process FSM, as described in the text. Draw a state diagram for the system. Two states ph1 and ph2 can be used to generate each period of sck. A separate bit counter can be used by the FSM to transition between these two states once for each bit transmitted. This will greatly reduce the size of the state diagram. Using this approach a state diagram with as few as five states can be created.

The bit counter can also be used to implement shifting by simply using the count to select which bit from the parallel input data is output. Accordingly, this counter is referred to as the bit_addr counter.

A timing diagram of the expected system output waveforms for cpol = 0 and cpha = 0 follows.



Your design must be able to provide the correct output for all four combinations of cpol and cpha.

Submit your state diagram and properly commented design description source file and non-self-checking testbench, list of circuit features to be verified, description of required stimu-

lus waveforms, and an annotated waveform editor output showing what is being verified during each portion of the waveforms.

Design Task 3: Top Level Structure
The top-level structure simply combines the two previous entities. The entity declaration is:

```
entity SPI_test_system_I is
     port(
          rst_bar : in std_logic;     -- asynchronous system reset
          clk : in std_logic;         -- system clock
          send : in std_logic;        -- positive pulse to start transmission
          cpol : in std_logic;        -- clock polarity setting
          cpha : in std_logic;        -- clock phase setting
          data_in : in std_logic_vector(7 downto 0); -- parallel input data
          mosi : out std_logic;       -- master out slave in SPI serial data
          sck : out std_logic;        -- SPI shift clock to slave
          ss_bar : out std_logic      -- slave select signal
          );
end SPI_test_system_I;
```

Submit your state diagram and properly commented design description source file and non-self-checking testbench, list of circuit features to be verified, description of required stimulus waveforms, and an annotated waveform editor output showing what is being verified during each portion of the waveforms.

Laboratory Tasks

Laboratory Task 3: Send Positive Edge Detector

If Code2Graphics is available on your system, use it create a block diagram of your system from Task 3 and state diagrams for your FSMs from your source files for Task 3.

Use Lattice Diamond to fit the synthesized logic into a Lattice LFXP3C-4T100C-3I FPGA. In Lattice Diamond, view and print the chip report.

Submit your block and state diagrams from Code2Graphics and Chip Report.

Questions

1. How long is the `send_en` pulse generated by `send_pos_edge_det`? State this time duration as both number of clock pulses and as ns.

2. What is the frequency relationship between the SPI clock `sck` and the system clock `clk`?

3. From the log file determine the default state assignment encoding used by the synthesizer for your FSMs.