ESE 382 Digital System Design Using VHDL and PLDs

Spring 20, Ken Short
April 27, 2020 7:03 pm

PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY

Laboratory 09: SPI Transmitter and Receiver

This laboratory is to be performed the week starting April 26th.

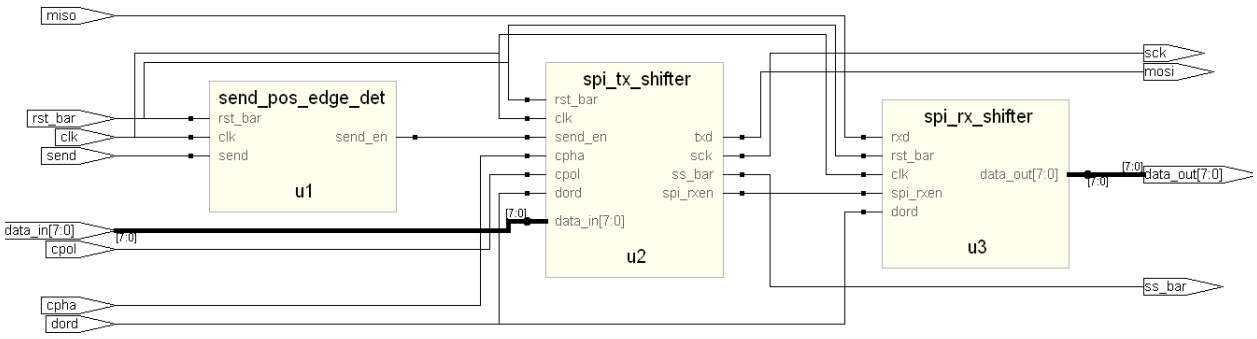
Prerequisite Reading

1. Chapter 10 of the text.

Purpose

In this laboratory the functionality of the SPI master, designed in Laboratory 8, is extended. A `DORD` input is added. This input selects whether the serial byte is transmitted most significant bit first or least significant bit first. A `MISO` input is also added. This input receives serial data sent from the slave to the master. The byte received is output at `data_out`.

A block diagram of the system for this laboratory is:



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

Design Tasks

Task 1: Data Order Selection and Receiver Enable

When the data order input `dord` is 0 at the start of a transmission, the `data_in` byte is transmitted most significant bit first. When the data order input is 1 at the start of a transmission, the `data_in` byte is transmitted least significant bit first. This feature is implemented by modification of entity `spi_tx_shifter`.

The entity declaration for `spi_tx_shifter` is modified to:

```
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
            dord : in std_logic;        -- data order
            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
            spi_rxen : out std_logic    -- enable receiver to shift data
            );
end spi_tx_shifter;
```

This entity declaration adds the `dord` input. It also adds output `spi_rxen` that is used to enable the receiver shift register of Task 2 to shift each bit of data in from the slave.

Modify the code for `spi_tx_shifter` to add these features. Write a non self-checking testbench. Simulate your design using your testbench. Your testbench must check for all combinations of `cpol`, `cpha`, and `dord`.

Submit your properly commented design description source file and non-self-checking testbench, list of circuit features to be verified, and waveform editor output.

Task 2: SPI Receive Shifter

The receiver shifter `spi_rx_shifter` converts the serial data at its `rxd` input to parallel and and provides this parallel result as output `data_out`. The entity declaration for `spi_rx_shifter` is:

```
entity spi_rx_shifter is
      port(
            rxd : in std_logic;         -- data received from slave
            rst_bar : in std_logic;     -- asynchronous reset
            clk : in std_logic;         -- system clock
            spi_rxen : in std_logic;    -- signal to enable shift
            dord : in std_logic;        -- data order bit
            data_out : out std_logic_vector(7 downto 0) -- received data
            );
end spi_rx_shifter;
```

The serial data is received at the `rxd` input. This data may arrive most significant bit first or least significant bit first, as determined by the `dord` input. The `spi_rxen` input must be asserted by `spi_tx_shifter` to enable the `spi_rx_shifter` to shift in a bit of data at the next rising edge of `clk`. The final parallel value at `data_out` must always have its most significant bit in the leftmost position even if the serial data was received least significant bit first.

Write the code for `spi_rx_shifter`. Write a non self-checking testbench. 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, and waveform editor output.

Task 3: SPI Test System II Top Level
The top-level structure simply combines the two previous entities and the positive edge detector from Laboratory 8. The top-level entity declaration is:

```
entity SPI_test_system_II 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
       dord : in std_logic;   -- transmission data order 0 => msb first
       miso : in std_logic;   -- master in slave out
       data_in : in std_logic_vector(7 downto 0);     -- parallel input data
       data_out : out std_logic_vector(7 downto 0);   -- parallel output 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_II;
```

Write the code for `SPI_test_system_II` and a non-self-checking top-level testbench. This testbench must verify all SPI modes (combinations of CPOL and CPHA) and both data orders.

Submit your properly commented design description source file and non-self-checking testbench, list of circuit features to be verified, and waveform editor output.

Laboratory Tasks
Task: `SPI_test_system_II`
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 FPGA. In Lattice Diamond, view and print the chip report.

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