Spring 2021

California State University, Northridge Department of Electrical & Computer Engineering

# Lab 4
# Design of a SPI Slave Core

February 28, 2021

ECE 526L

Written By: Jose Luis Martinez

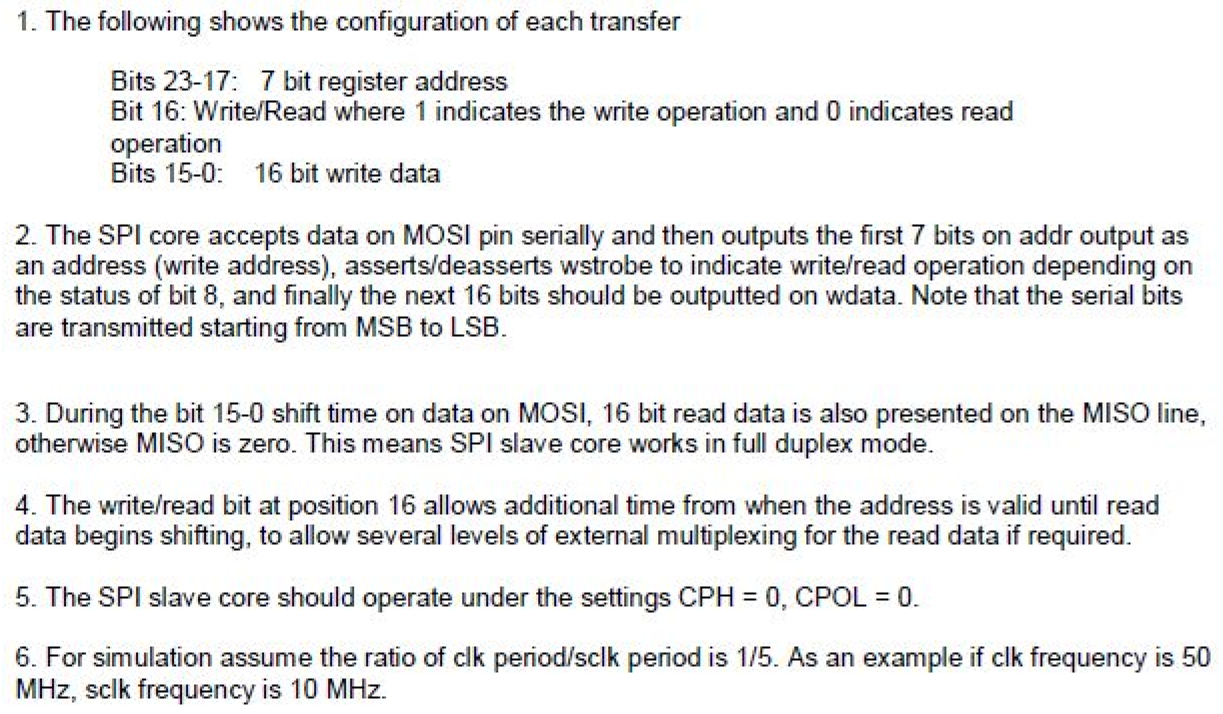
## Introduction

SPI is a communication protocol that uses a clock signal to synchronize the output data from the master to the slave by sampling bits. The SPI transmits one bit per clock cycle and the speed is determined by the synchronous clock(SCLOCK) frequency. The way a master selects the slave is by setting the CS line for the slave core that the master wants to communicate with. Whenever the master is not communicating with the slave the CS line is held high. The master sends data out serially from its master out slave in (MOSI) pin and reads data serially from its master in slave out (MISO) pin. The slave sends data out serially from its master in slave out (MISO) pin and reads data serially from its master out slave in (MOSI) pin.

## Objective

For this lab we are to design a SPI slave core and read/write from it.

The SPI slave core must follow the requirements below:



1. The following shows the configuration of each transfer

   Bits 23-17: 7 bit register address
   Bit 16: Write/Read where 1 indicates the write operation and 0 indicates read operation
   Bits 15-0: 16 bit write data

2. The SPI core accepts data on MOSI pin serially and then outputs the first 7 bits on addr output as an address (write address), asserts/deasserts wstrobe to indicate write/read operation depending on the status of bit 8, and finally the next 16 bits should be outputted on wdata. Note that the serial bits are transmitted starting from MSB to LSB.

3. During the bit 15-0 shift time on data on MOSI, 16 bit read data is also presented on the MISO line, otherwise MISO is zero. This means SPI slave core works in full duplex mode.

4. The write/read bit at position 16 allows additional time from when the address is valid until read data begins shifting, to allow several levels of external multiplexing for the read data if required.

5. The SPI slave core should operate under the settings CPH = 0, CPOL = 0.

6. For simulation assume the ratio of clk period/sclk period is 1/5. As an example if clk frequency is 50 MHz, sclk frequency is 10 MHz.

## Methodology

For this lab I chose to design the hardware in verilog because I am currently taking a course in that topic so I thought this would be a great way to practice. I designed the SPI slave core using behavioral language. I coded two verilog files with one being the actual SPI slave core and the other being the test bench file to drive and test the core. The following is the code I used and its results.

## Results

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: CSUN
// Engineer: Jose Martinez
//
// Create Date: 02/23/2021 06:17:00 PM
// Module Name: spi_slave_core
// Project Name: Lab 4 - Design of a SPI Slave Core
//////////////////////////////////////////////////////////////////////////////////


module spi_slave_core(sclk, mosi, ssn, rdata, clk, wdata, wstrobe, miso, addr);
    input sclk, ssn, clk, mosi;
    input [15:0]rdata;
    output wstrobe, miso;
    output [15:0]wdata;
    output [6:0]addr;

    integer count = 0;
    reg misoR = 0;
    reg wsvalue = 0;
    reg writing = 0;
    reg [6:0]addrReg = 0;
    reg [15:0]writeReg = 0;


    always@(posedge sclk) begin
    if (ssn == 0) begin
        if (count < 7) addrReg = {addrReg[5:0], mosi};
        else if (count == 7) writing = mosi;
        else if (count < 24) writeReg = {writeReg[14:0], mosi};
        else if (writing == 1) begin
            wsvalue = 1;
            #50 wsvalue = 0;
            writing = 0;
        end
        count = count + 1;
        if (count >= 25) count = 0;
    end

    end

    always@(posedge clk) begin
    if ((count > 7) & (writing == 0) & (count < 24)) begin
```

```
        #20 misoR = rdata[count-8];
    end else if( count > 23) #20 misoR = 0;
    end

    assign miso = misoR;
    assign wstrobe = wsvalue;
    assign wdata = writeReg;
    assign addr = addrReg;

endmodule
```

**Fig. 1** (spi_slave_core.v)

```
////////////////////////////////////////////////////////////////////////////////////
// Company: CSUN
// Engineer: Jose Martinez
//
// Create Date: 02/23/2021 06:17:00 PM
// Module Name: spi_slave_core_tb
// Project Name: Lab 4 - Design of a SPI Slave Core
////////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps
`define sclk_per 100
`define clk_per 20

module spi_slave_core_tb();
    reg CLK, SCLK, MOSI, SSN;
    reg [15:0]RDATA;
    wire MISO, WSTROBE;
    wire [15:0]WDATA;
    wire [6:0]ADDR;
    integer i = 0;

    spi_slave_core spi1(.sclk(SCLK), .mosi(MOSI), .ssn(SSN), .rdata(RDATA),
           .clk(CLK), .wdata(WDATA), .wstrobe(WSTROBE), .miso(MISO), .addr(ADDR));

    initial begin
        CLK = 0;
        forever begin
            #(`clk_per/2) CLK = ~CLK;
        end
    end

    initial begin
        SCLK = 0;
        #(`sclk_per)
        for (i=0; i<50; i = i+1) begin
            #(`sclk_per/2) SCLK = ~SCLK;
        end
    end

    initial begin
        RDATA = 16'h8f0f;
        SSN = 1;
        MOSI = 0;
        #(`sclk_per) SSN = 0;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 1;

        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 0;

        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 1;

        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 1;

        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 0;
        #(`sclk_per) MOSI = 1;

        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 1;
        #(`sclk_per) MOSI = 1;

        #(`sclk_per) MOSI = 0;
        #(`sclk_per) SSN = 1;
        #(`sclk_per*4) $finish;
    end

endmodule
```

**Fig. 2** (spi_slave_core_tb.v)

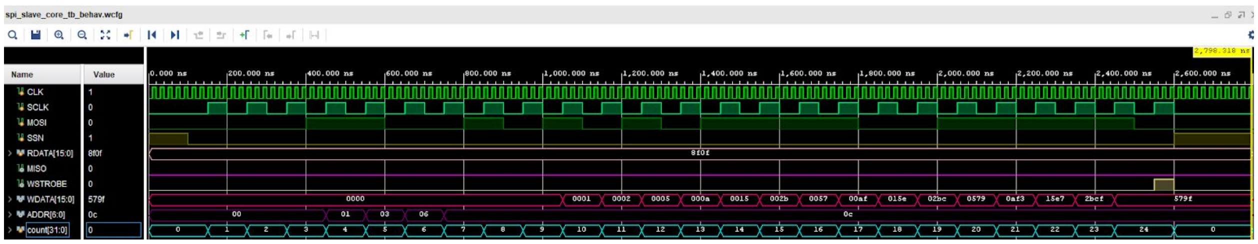


**Fig. 3** Waveform (MOSI Transmits Serially(MSB) -> 0001 1001 0101 0111 1001 1111)

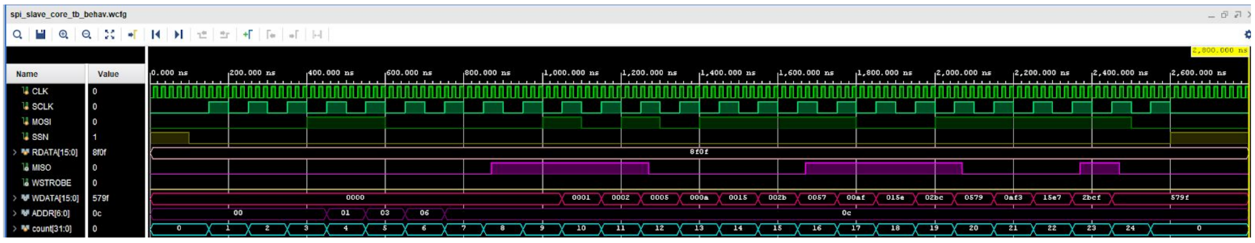


**Fig. 4** Waveform (MOSI Transmits Serially(MSB) -> 0001 1000 0101 0111 1001 1111)

## Analysis

In the spi_slave_core.v file we hold our SPI slave core module. In this module two always blocks control the logic for the core. One is made to read data from the MOSI line and the other to transmit to the MISO line. The first always block is also in charge of keeping track of what position the core is reading from the MOSI line. The second always block then uses that information to know if it needs to transmit and what bit to transmit.

The spi_slave_core_tb.v sends clock, sclock, ssn, and MOSI signals to the core to check if the core is working properly. This verilog file has three initial blocks one being the clock signal, second one being the sclock, and the last one being in charge of transmitting the rest of the signals. In this case we are running the clock at a period of 20ns and the sclock is being run at 100ns.

As we can see from **Fig. 3 & 4** the ssn is on for the first 100ns and the sclock does not start until that 100ns period is over. This is to comply with the SPI settings of CHPA=0 and CPOL=0. As we can see the data transmitted from the MOSI is transmitted from MSB to LSB and each bit lasts one complete sclock cycle. The address is done reading by the 7th sclock cycle and then the w/r bit is read. In **Fig. 3** the 8th bit is ‘1’ so therefore the data will continue to be read and when the last bit is transmitted the slave core will turn the wstrobe on for 1/2 slcock cycle to signify that it is ready to write. In **Fig. 4** the 8th bit is ‘0’ meaning that the slave should read the data from that address and transmit it. As we can see the MISO line begins transmitting data read from the provided address as soon as it reaches the 9th count.

## Conclusion

In conclusion we learned how the SPI communication protocol works by implementing it in vivado using either verilog or vhdl. We had to make sure that the timing is correct and that we follow all the rules that came with SPI. The results from this lab show that my verilog code is working as a SPI slave core should. I have never designed a communication interface and now that I have, I feel more comfortable designing a different communication protocol.