EE 214: Digital Circuits Lab
Dev Arora (23B1271)
Arjun Singh (23B1272)

Implementation of SPI Protocol

Task 1

October 23, 2024

Work Distribution:

Dev:

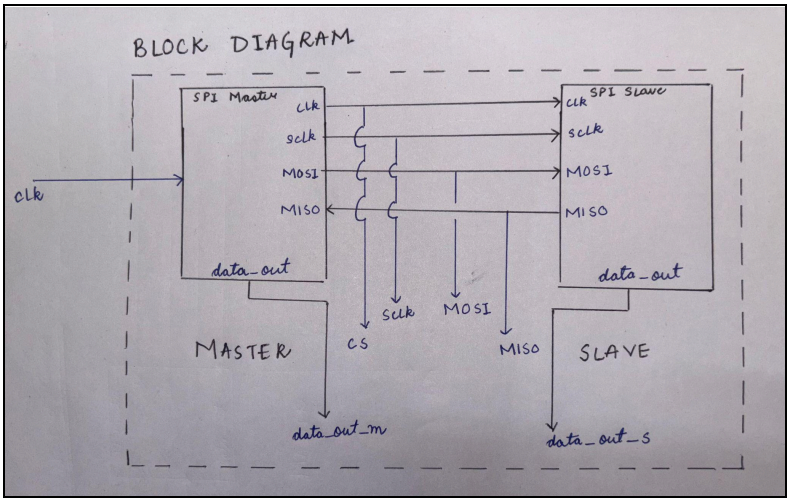
- Read about SPI from given resources and youtube.
- Wrote the code for Master module
- Wrote the testbench for the simulation
- Debugged the compiled code together
- Completed SPI protocol, Master and Slave module sections in the report

Arjun:

- Read about SPI from given resources and youtube
- Wrote the code for slave module
- Wrote the code for top level entity
- Debugged the compiled code together
- Completed Toplevel and testbench, results and observations sections in the report

Understanding SPI Communication Protocol:

Block Diagram:



Mode0:

In mode 0, the clock polarity is low in idle state i.e. when the chip select bit is high and is transitioning to low. In this mode, the data is sampled on the rising edge of the clock and is shifted out on the falling edge of the clock. (sampling is basically receiving data from the other module and shifting is updating the mosi/miso logic for next clock event)

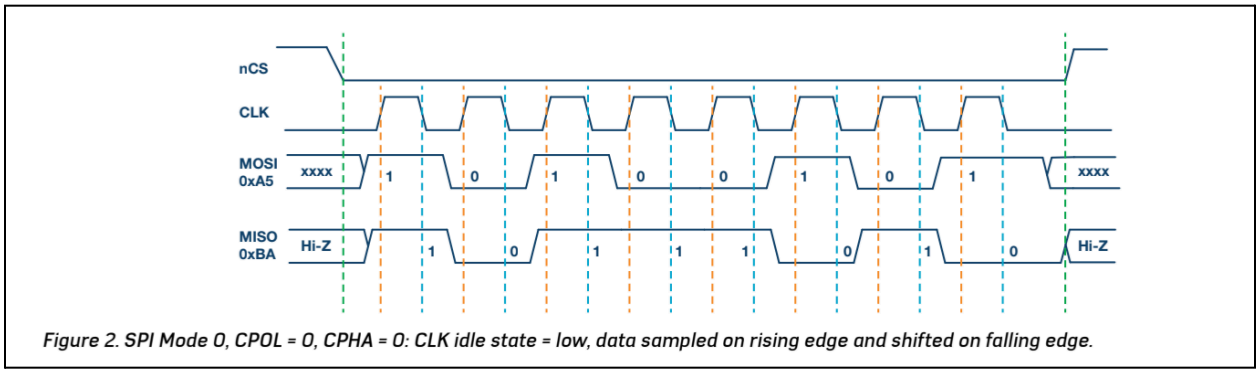


Figure 2. SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.

Ports Used:

MASTER		SLAVE	
PORT	TYPE	PORT	TYPE
initial_comm	in		
clk	in		
chip_select	out	chip_select	in
sclk	out	sclk	in
mosi	out	mosi	in
miso	in	miso	out
master_out	out		
		slave_out	out

- **initial_comm:** Master puts this bit to start communication
- **clk:** Clock given as input to the master
- **chip_select:** This bit is low for the slave which is selected for communication, hence this is output for master and input for slave.
- **sclk:** Clock frequency at which serial communication takes place. This is decided by the master, hence it is output for master and input for slave.
- **mosi**: bit which stores the value to be shifted from master to slave.
- **miso:** bit which stores the value to be shifted from master to slave.
- **master_out:** Data vector which stores the the binary data received from slave
- **Slave_out:** Data vector which stores the the binary data received from master

Master and Slave Module:

Working and Code Explained:

- In our code, we start transmission from the MSB and go down to LSB.
- We have broken down the code into three processes, first is the receiving or sampling process, second is the shifting or sending process, and the third process is used for printing waveform purposes.
- We have used the logic of a finite state machine (fsm) to define the states during the process of transmission.
- The codes for master and slave are attached below in pdf format

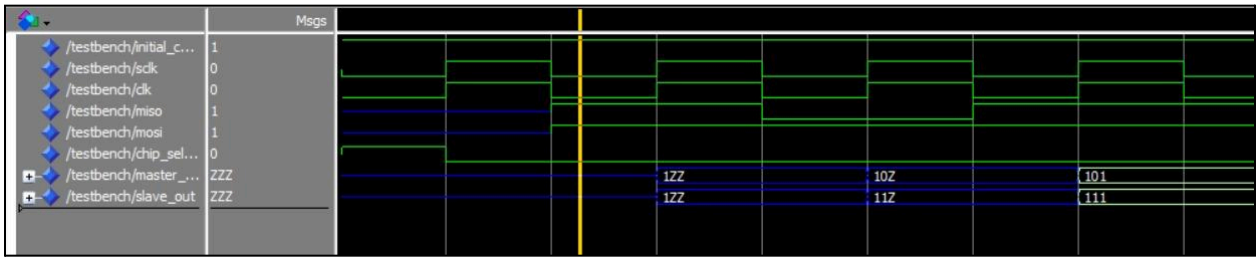
Top Level Entity and Testbench:

Working and Code Explained:

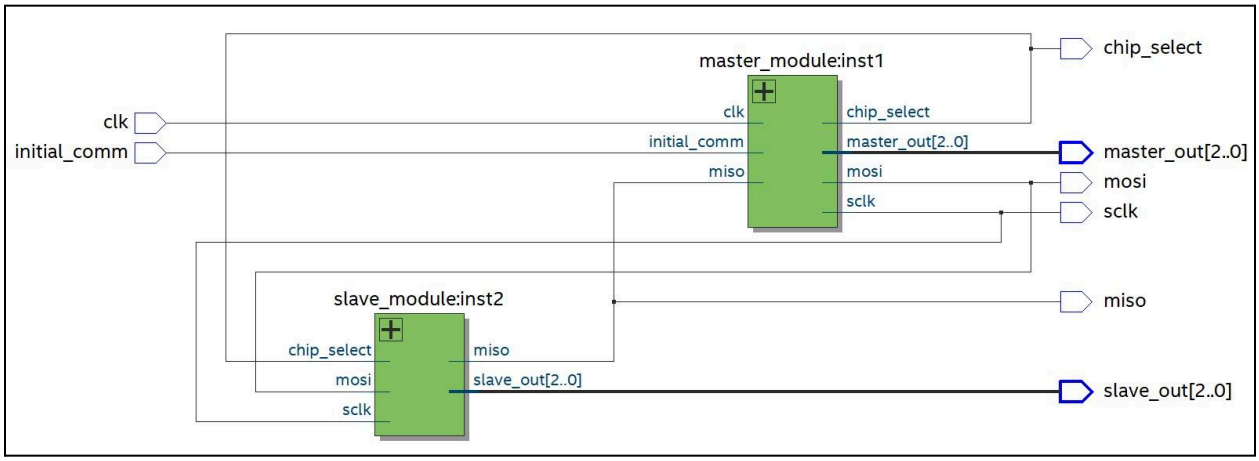
- In this code, the top level entity is basically needed to instantiate both slave and master modules at the same time using the common ports and defining other ports in terms of these common ports.
- The testbench is defined for the top level entity only and the simulation is thus observed.
- The codes for top level entity and testbench are attached below in pdf format.

Results and Observation:

Waveform:



Netlist Viewer:



Key Observations:

- mosi/miso update at falling edge of the clock
- master_out/slave_out update at rising edge of the clock confirming mode 0 configuration.
- When chip_select is high, no transmission can happen between the slave and master
- Initial_comm (initialize communication) is kept high all the time to activate the master at all times.

Challenges faced:

1. **Understanding SPI and Mode0:** SPI came without any background and a lot of time went in understanding how SPI works? Why is it needed? and what are the functions of various ports?

2. **Different modes and correct clock events**: SPI has 4 modes and we have to use mode 0 in this task. We faced difficulty in figuring out when to do sampling/shifting and initial results were wrong because registers were updated at different times.
3. **Defining miso_error:** We had to define a miso_error which stores the value of miso to be used one cycle before. Without this, we were getting “110” as master_out which was basically displaced by one bit.

Master Module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity master_module is
    port(
        initial_comm : in std_logic;
        clk : in std_logic;
        chip_select : out std_logic := '1';
        sclk : out std_logic;
        mosi : out std_logic := 'Z';
        miso : in std_logic;
        master_out : out std_logic_vector(2 downto 0)
    );
end entity;

architecture archi_master_module of master_module is
    signal buffer_data : std_logic_vector(2 downto 0) := "ZZZ";
    signal required_outp : std_logic_vector(2 downto 0) := "111";

    type state is (initial, s1, s2, s3, s4);
    signal s_present, s_next : state := initial;

begin
    sclk <= clk;

    receiving : process(initial_comm, clk)
    begin
        if rising_edge(clk) then
            case s_present is
                when initial =>
                    if initial_comm = '1' then
                        s_next <= s1;
                        chip_select <= '0';
                    else
                        s_next <= initial;
                        chip_select <= '1';
                    end if;
                when s1 =>
                    buffer_data(2) <= miso;
                    s_next <= s2;
                when s2 =>
                    buffer_data(1) <= miso;
                    s_next <= s3;
                when s3 =>
                    buffer_data(0) <= miso;
                    s_next <= s4;
                when s4 =>
                    chip_select <= '1';
            end case;
        end if;
```

```
53      end process;
54
55      sending : process(clk)
56      begin
57          if falling_edge(clk) then
58              mosi <= required_outp(2);
59              case s_present is
60                  when s1 =>
61                      mosi <= required_outp(1);
62                  when s2 =>
63                      mosi <= required_outp(0);
64                  when others => null;
65              end case;
66          end if;
67      end process;
68
69      print : process(s_next)
70      begin
71          s_present <= s_next;
72          master_out <= buffer_data;
73      end process;
74
75  end architecture;
```

Slave Module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity slave_module is
    port(
        chip_select : in std_logic;
        sclk : in std_logic;
        mosi : in std_logic;
        miso : out std_logic := 'Z';
        slave_out : out std_logic_vector(2 downto 0)
    );
end entity;

architecture archi_slave_module of slave_module is
    signal buffer_data : std_logic_vector(2 downto 0) := "ZZZ";
    signal required_outp : std_logic_vector(2 downto 0) := "101";

    signal miso_error : std_logic := required_outp(2);
    type state is (initial, s1, s2, s3, s4);
    signal s_present, s_next : state := initial;

begin
    receiving : process(chip_select, sclk)
    begin
        if rising_edge(sclk) then
            case s_present is
                when initial =>
                    if chip_select <= '0' then
                        s_next <= s1;
                    else
                        s_next <= initial;
                    end if;
                when s1 =>
                    buffer_data(2) <= mosi;
                    s_next <= s2;
                when s2 =>
                    buffer_data(1) <= mosi;
                    s_next <= s3;
                when s3 =>
                    buffer_data(0) <= mosi;
                    s_next <= s4;
                when s4 => s_next <= s4;
            end case;
        end if;
    end process;

    sending : process(sclk)
    begin
        if falling_edge(sclk) then
            miso <= miso_error;
```

```
53              case s_present is
54                  when s1 =>
55                      miso_error <= required_outp(1);
56                  when s2 =>
57                      miso_error <= required_outp(0);
58                  when others => null;
59              end case;
60          end if;
61      end process;
62
63      print : process(s_next)
64      begin
65          s_present <= s_next;
66          slave_out <= buffer_data;
67      end process;
68
69  end architecture;
```

Top-Level Module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity toplevel is
    port (
        initial_comm : in std_logic;
        clk : in std_logic;
        sclk : out std_logic;
        mosi, miso : out std_logic;
        chip_select : out std_logic;
        master_out : out std_logic_vector(2 downto 0);
        slave_out : out std_logic_vector(2 downto 0)
    );
end entity;

architecture archi_toplevel of toplevel is

    component master_module is
        port (
            initial_comm : in std_logic;
            clk : in std_logic;
            chip_select : out std_logic;
            sclk : out std_logic;
            mosi : out std_logic;
            miso : in std_logic;
            master_out : out std_logic_vector(2 downto 0)
        );
    end component;

    component slave_module is
        port (
            chip_select : in std_logic;
            sclk : in std_logic;
            mosi : in std_logic;
            miso : out std_logic;
            slave_out : out std_logic_vector(2 downto 0)
        );
    end component;

    signal miso_temp, mosi_temp, chip_select_temp, sclk_temp :
       std_logic;

begin
    inst1: master_module port map(initial_comm, clk, chip_select_temp,
       sclk_temp, mosi_temp, miso_temp, master_out);
    inst2: slave_module port map(chip_select_temp, sclk_temp, mosi_temp
       , miso_temp, slave_out);

    sclk <= sclk_temp;
    miso <= miso_temp;
```

```
50      mosi <= mosi_temp;
51      chip_select <= chip_select_temp;
52
53  end architecture;
```

Testbench

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity testbench is
5  end entity;
6
7  architecture archi_testbench of testbench is
8
9      component toplevel is
10         port (
11             initial_comm : in std_logic;
12             clk : in std_logic;
13             sclk : out std_logic;
14             mosi, miso : out std_logic;
15             chip_select : out std_logic;
16             master_out : out std_logic_vector(2 downto 0);
17             slave_out : out std_logic_vector(2 downto 0)
18         );
19     end component;
20
21     signal initial_comm, sclk : std_logic := '1';
22     signal clk : std_logic := '0';
23     signal miso, mosi, chip_select : std_logic;
24     signal master_out, slave_out : std_logic_vector(2 downto 0);
25     constant clk_proc : time := 100 ns;
26
27 begin
28     inst1: toplevel port map(
29         initial_comm, clk, sclk, mosi, miso, chip_select, master_out,
               slave_out
30     );
31
32     clock_process : process
33     begin
34         clk <= not clk after clk_proc / 2;
35         wait for clk_proc / 2;
36     end process;
37
38 end architecture;
```