MSc. Embedded System Design

VHDL LAB REPORT

SPI MASTER COMMUNICATION PROTOCOL

Under the Guidance of

Prof. Dr.-Ing. Kai Müller

Submitted by,

Name: Joe Joseph Nelson
Matriculation No. : 38750

Table of Contents

1. Aim

To design and simulate SPI (Serial Peripheral Interface) communication protocol for Master as state machines in VHDL with parameters CPOL=0 and CPH=1.

2. Introduction

Serial Peripheral Interface or SPI is a synchronous serial communication protocol that provides full – duplex communication at very high speeds. It uses a dedicated clock signal to synchronize the transmitter and receiver or Master and Slave. An SPI Master Transmission protocol is designed in programmable logic as VHDL program. A state diagram is designed using fizzim which is illustrated in fig. 1 below.

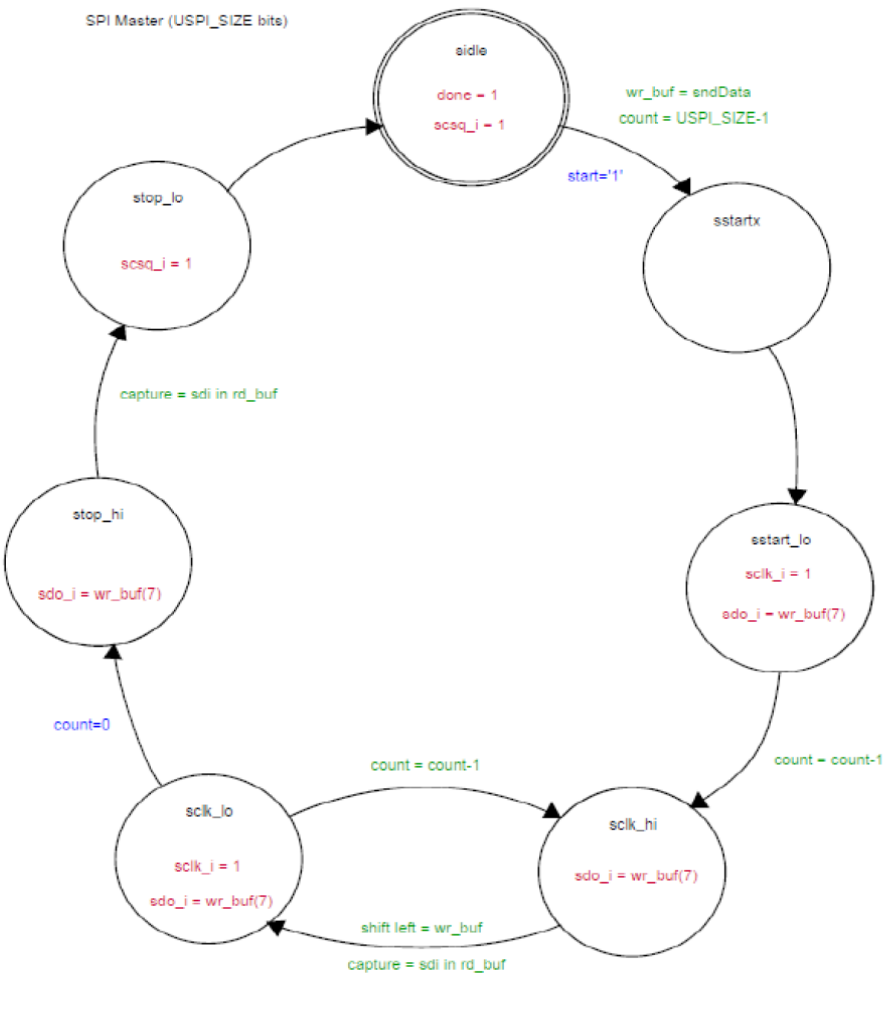

Figure 1 Timed State Machine

• The default state for Master is sidle with rising edge of bclk(clock) and resetn = ‘0’. In state sidle, done and scsq_i are set. When start = ‘1’, the next_state (next state) will be set to sstartx. But transition to next state will done only on the rising edge of bclk.

• During transition, the signals count and wr_buf are initialized.

• In state sstartx, only the next state is set to sstart_lo (start low).

• At state sstart_lo, sclk_i = ‘1’ so that at the next rising edge of clock sclk should be ‘1’. It is same with sdo_i and sdo. But sdo_i is set with the MSB of wr_buf.

• The count value will decrement when transition occurs from state sstart_lo to state sclk_hi.

• At state sclk_hi, sdo_i is set with the value of MSB of wr_buf. And next state is set to sclk_lo. During the transition, value of rd_buf is shifted left and appended with sdi to LSB. Value of wr_buf is also shifted left so that next bit to be transferred to sdo_i is at MSB.

• Now the state machine will toggle between states sclk_hi and sclk_lo until the count value becomes ‘0’. The value of sclk_i is set while toggling between states so that the clock should be high when it is in state sclk_hi and it should be low in state sclk_lo.

• Each bit transferred is available at the output both when sclk is high and low. The slave reads the value at the falling edge of the clock only.

• Last bit to be transferred is sent to sdo_i in state sstop_hi.

• After the end of transmission, the chip select is disabled by setting value to scsq_i in state sstop_lo. Subsequently, at the rising edge of clock it returns to state sidle.

• The size of bits to be transferred is generalized as USPI_SIZE bits. It can be any size. We have considered USPI_SIZE=’8’ bits.

3. VHDL CODE

3.1 Source Code:

```
----------------------------------------------------------------------------
-------
-- Company:          Univ. Bremerhaven
-- Engineer:         Joe Joseph Nelson
-- Create Date:      27/06/2021
-- Description:      SPI Transmitter for (CPOL=0, CPHA=1)
----------------------------------------------------------------------------
-------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity spims is
    GENERIC ( USPI_SIZE : integer := 16);
    PORT ( resetn : in std_logic;
             bclk : in std_logic;
             start : in std_logic;
             done : out std_logic;
             scsq : out std_logic;
             sclk : out std_logic;
             sdi : in std_logic;
             sdo : out std_logic;
             sndData : in std_logic_vector (USPI_SIZE- 1 downto 0);
             rcvData : out std_logic_vector (USPI_SIZE - 1 downto 0)
             );
end spims;

architecture Behavioral of spims is
TYPE State_type IS (sidle, sstartx, sstart_lo, sclk_hi, sclk_lo, sstop_hi,
sstop_lo);
SIGNAL state, next_state : State_type;
SIGNAL sclk_i, scsq_i, sdo_i : std_logic;
SIGNAL wr_buf : std_logic_vector (USPI_SIZE - 1 downto 0);
SIGNAL rd_buf : std_logic_vector (USPI_SIZE - 1 downto 0);

SIGNAL count : integer RANGE 0 to USPI_SIZE - 1;
CONSTANT CLK_DIV : integer := 4;
SUBTYPE  ClkDiv_type IS integer RANGE 0 to CLK_DIV-1;
SIGNAL spi_clkp : std_logic;
begin

    rcvData <= rd_buf;

    --clock division
    clk_d : PROCESS(bclk)
    VARIABLE clkd_cnt : ClkDiv_type;
    BEGIN
        IF rising_edge(bclk) THEN
            spi_clkp <= '0';
            IF resetn = '0' THEN
                clkd_cnt := CLK_DIV-1;
            ELSIF clkd_cnt =0 THEN
                spi_clkp <= '1';
                clkd_cnt := CLK_DIV-1;
            ELSE
                clkd_cnt :=clkd_cnt -1;
            END IF;
        END IF;
    END PROCESS clk_d;

    -- spi sequential logic
    sseq_p : PROCESS(bclk)
    BEGIN
        IF rising_edge (bclk) THEN
            IF resetn = '0' THEN
                state <= sidle;
                count <= USPI_SIZE-1;
            ELSIF spi_clkp <= '1' THEN
                IF next_state = sstartx THEN
                    wr_buf <= sndData;
                    count <= USPI_SIZE-1;
                ELSIF next_state = sclk_hi THEN
                    count <= count-1;
                ELSIF next_state = sclk_lo THEN
                    wr_buf <= wr_buf (USPI_SIZE-2 downto 0) & '-';
                    rd_buf <= rd_buf (USPI_SIZE-2 downto 0) & sdi;
                ELSIF next_state = sstop_lo THEN
```

```
                    rd_buf <= rd_buf (USPI_SIZE-2 downto 0) & sdi;
                END IF;
                state <= next_state;
                scsq <= scsq_i;
                sclk <= sclk_i;
                sdo <= sdo_i;
            END IF;
        END IF;
    END PROCESS sseq_p ;

    -- spi combinational logic
    scmb_p : PROCESS(state, start)
    BEGIN
        next_state <= state;
        scsq_i <= '0';
        done <= '0';
        scsq_i <= '0';
        sclk_i <= '0';
        sdo_i <= '0';
        CASE state IS
            WHEN sidle =>
                done <= '1';
                scsq_i <= '1';
                IF start = '1' THEN
                    next_state <= sstartx;
                END IF;
            WHEN sstartx =>
                next_state <= sstart_lo;
            WHEN sstart_lo =>
                sclk_i <= '1';
                sdo_i <=  wr_buf( USPI_SIZE-1);
                next_state <= sclk_hi;
            WHEN sclk_hi =>
                sdo_i <=  wr_buf( USPI_SIZE-1);
                next_state <= sclk_lo;
            WHEN sclk_lo =>
                sdo_i <=  wr_buf( USPI_SIZE-1);
                sclk_i <= '1';
                IF count=0 THEN
                    next_state <= sstop_hi;
                ELSE
                    next_state <= sclk_hi;
                END IF;
            WHEN sstop_hi =>
                sdo_i <=  wr_buf( USPI_SIZE-1);
                next_state <= sstop_lo;
            WHEN sstop_lo =>
                scsq_i <= '1';
                next_state <= sidle;
        END CASE;
    END PROCESS scmb_p ;

end Behavioral;
```

3.2 Test bench file

```
---------------------------------------------------------------------------
-------
-- Company:          Univ. Bremerhaven
-- Engineer:         Joe Joseph Nelson
-- Create Date:      27/06/2021
-- Description:      SPI Transmitter for (CPOL=0, CPHA=1)
---------------------------------------------------------------------------
-------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity spims_tb is
end spims_tb;

architecture Behavioral of spims_tb is
COMPONENT spims is
    GENERIC ( USPI_SIZE : integer := 16);
    PORT ( resetn : in std_logic;
             bclk : in std_logic;
             start : in std_logic;
             done : out std_logic;
             scsq : out std_logic;
             sclk : out std_logic;
             sdi : in std_logic;
             sdo : out std_logic;
             sndData : in std_logic_vector (USPI_SIZE- 1 downto 0);
             rcvData : out std_logic_vector (USPI_SIZE - 1 downto 0)
             );
end COMPONENT spims;
CONSTANT SPI_NBITS : integer := 8;

SIGNAL  resetn : std_logic := '1';
SIGNAL   bclk : std_logic := '0';
SIGNAL   start : std_logic := '0';
SIGNAL   done : std_logic := '0';
SIGNAL   scsq : std_logic := '0';
SIGNAL   sclk : std_logic := '0';
SIGNAL   sdi : std_logic := '0';
SIGNAL   sdo : std_logic := '0';
SIGNAL   sndData : std_logic_vector(SPI_NBITS-1 downto 0) := x"5A";
SIGNAL   rcvData : std_logic_vector(SPI_NBITS-1 downto 0) := x"00";

CONSTANT clk_period : time := 10 ns;

begin

    uut : spims
    GENERIC MAP (USPI_SIZE => SPI_NBITS)
    PORT MAP (resetn => resetn,
                 bclk => bclk,
                 start => start,
                 done => done,
                 scsq => scsq,
                 sclk => sclk,
                sdi => sdi,
                sdo => sdo,
                sndData => sndData,
                rcvData => rcvData
    );

    clk_p : PROCESS
    BEGIN
        bclk <='0';
        wait for clk_period / 2;
        bclk <= '1';
        wait for clk_period / 2;
    END PROCESS clk_p;

    sdi <= NOT sdo;

    stim_p : PROCESS
    BEGIN
        wait for clk_period;
        resetn <='0';
        wait for clk_period;
        resetn <='1';
        wait for clk_period;
        start <='1';
        wait for clk_period * 5;
        start <='0';
        wait for clk_period * 100;
        report " spims test bench finished";
        wait ;
    END PROCESS stim_p;

end Behavioral;
```

4. VHDL Code explanation

4.1. Libraries:

To use all the components of the package STD_LOGIC_64, part of library IEEE "use" statement is used. And "Library" statement is used to declare IEEE is a library. So that complier will not get confused

4.2. Entity Declaration:

Entity named **spims** is declared which represents all the external interfacings and ports. The input and output ports declared in the entity **spims** are:

- bclk : A bus clock signal
- resetn : reset signal (active low)
- start : signal to start transmission
- sdi : serial data in
- sndData : A vector containing data to be sent
- done : Indicates transmission is done
- scsq : SPI chip select signal
- sclk : serial clock
- sdo : serial data out
- rcvData : A vector containing data to be received

4.3. Architecture

Architecture is a description of the inner design operation. All the required signals are declared and initialized in the architecture block. In addition it contains the processes that contains the operational logic.

4.3.1. Types and Signals:

A user defined type state_type is declared which represents the states of the system namely, sidle, sstartx, sstart_lo, sclk_hi, sclk_lo, sstop_hi and sstop_hi.

The architecture block contains the following signal:

- Signals **state**, and **next_state** are of state_type.
- **Count** of type count type**.**
- **wr_buf** and **rd_buf** of type standard logic vector**.**
- **sclk_i, scsq_i,** and **sdo_i** of type standard logic

4.3.2. Constants and Subtypes:

This program uses a constant called MAX_COUNT, which is assigned a value of 10 and a subtype ranging from 0 to MAX_COUNT – 1.

4.3.3. Processes:

SPI Master module consists of 3 process :

1. **clk_d**: This process performs clock division by 3. It provides a clock signal in spi_clkp for every 3 cycles of bclk.

2. **sseq_p**:

• It consists of sequential logic process.

• Any changes for resetn will occur only on the rising edge of the bclk(clock).

• When the value of resetn='0', then the present state will be set to default state sidle.

• As the bclk(clock) is divided by 3 in the process clk_d, the change in any scsq, sclk, sdo, and state will occur only when spi_clkp is high. The clock division is done to make the data transfer visible to naked eye when configured with hardware.

• Initialization of wr_buf and count is done during transition to state sstartx, is initialized in this sequential process.

• The changes in wr_buf, count and rd_buf which occur during transition as stated in state diagram is also done in the sequential process.

• Since the process is reading the values of bclk, resetn, count, next_state, sndData, scsq_i, sclk_i, sdo_i and spi_clkp, it is necessary to mention it in the sensitivity list.

3. **scmb_p**:

• It consists of combinational logic process with state transition conditions.

• Default values for outputs done, next_state, scsq_i, sclk_i and sdo_i are defined so that unwanted latches are not synthesized in the hardware.

• State transitions are implemented with CASE statements, where scsq_i, sclk_i, sdo_i and next_state is set with appropriate values at different states.

• Transition to state sstartx will only take place when start = '1'. If start = '0' then state machine will stop. This is to prevent endless state transitions. The value of start will become 0 when the all the data bits are transferred.

• Since the process is reading the values of state, start, count and wr_buf, it is necessary to define it in the sensitivity list of process.

5. Simulation Result

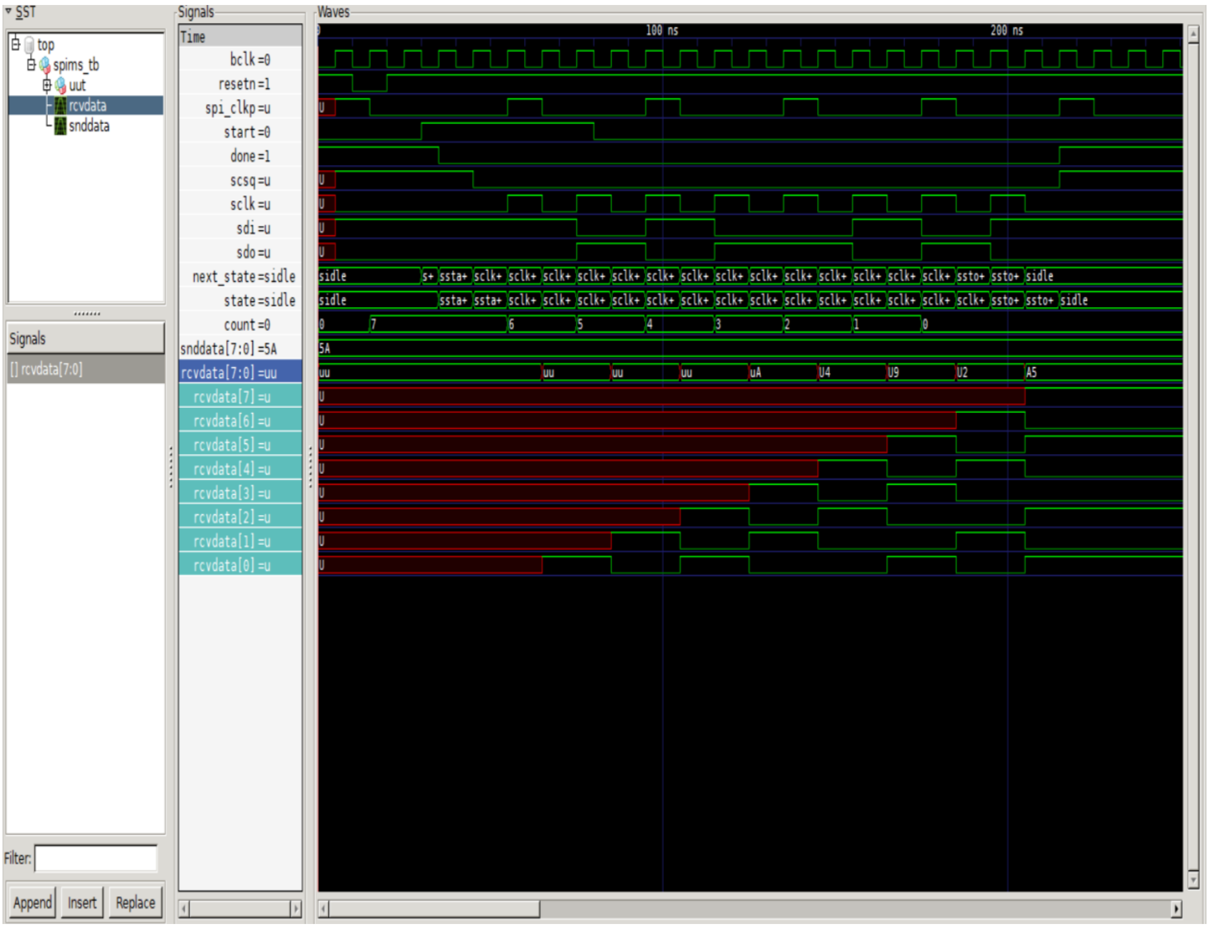

Figure 2 Output waveforms

6. Conclusion

The designing and simulation of VHDL program to implement serial peripheral interface communication protocol master transmission in programmable logic is done.