INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING

KURNOOL

Design and Implementation of SPI Protocol

Using Two ZedBoards (Master and Slave)

Submitted by:

V. Abhinav

Roll No: 523EC0013

Department of Electronics and Communication Engineering

Under the Guidance of

DR P.RANGA BABU SIR

Contents

Abstract

The Serial Peripheral Interface (SPI) protocol is one of the most efficient synchronous communication systems used for high-speed data transfer between digital devices. This project presents the design and implementation of SPI protocol using two Digilent Zed-Boards, where one acts as a **Master** and the other as a **Slave**. The Master board generates and transmits an 8-bit data pattern, which is received and displayed on LEDs connected to the Slave board. The entire design is realized using **Verilog HDL** on the Xilinx Vivado platform. The project highlights synchronous serial communication principles, FPGA hardware interfacing, and timing control for reliable data transmission.

1 Introduction

Serial Peripheral Interface (SPI) is a full-duplex synchronous communication protocol that allows data exchange between one Master and multiple Slave devices. It is widely used in embedded systems due to its simplicity and speed advantages over other serial interfaces such as UART and I^{2}C.

In this project, SPI communication is implemented using two ZedBoards. The Master board generates clock and control signals and transmits 8-bit data, while the Slave board receives the data and displays it through LEDs. The Verilog HDL implementation ensures complete control over timing, synchronization, and bit-level communication.

2 Objectives

- To design SPI Master and Slave modules using Verilog HDL.
- To establish synchronous communication between two ZedBoards.
- To transfer an 8-bit pattern from Master to Slave.
- To visualize received data using LEDs on the Slave board.
- To verify proper synchronization through simulation and hardware testing.

3 Tools Used

- **Software:** Xilinx Vivado Design Suite
- **Language:** Verilog HDL
- **Hardware:** Two Digilent ZedBoards (Zynq-7000 SoC)
- **Additional Tools:** Jumper wires, breadboard, connecting cables

4 Description of SPI Protocol

SPI is a four-wire communication protocol comprising:

- **MOSI (Master Out Slave In):** Transmits data from Master to Slave.
- **MISO (Master In Slave Out):** Transmits data from Slave to Master.
- **SCLK (Serial Clock):** Clock signal generated by Master for synchronization.
- **SS (Slave Select):** Active-low signal used to enable a particular Slave.

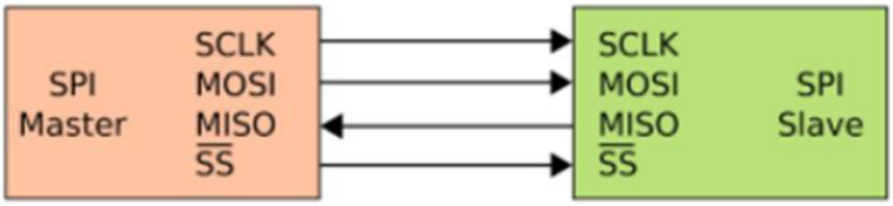

Figure 1: SPI Communication Between Master and Slave ZedBoards

4.1 Working Principle of SPI Communication

The SPI communication in this design works on the concept of clock-driven data exchange, where each bit is shifted in and out synchronously with clock edges.

4.1.1 1. Clock Synchronization

The master ZedBoard generates a serial clock (SCLK) that governs the timing of all data transfers. The slave board uses this clock to sample data on the MOSI line. For this design, SPI Mode 0 (CPOL=0, CPHA=0) is used:

- Clock is idle LOW.
- Data is sampled at the rising edge of SCLK.

4.1.2 2. Data Transmission

When communication begins, the master sets the **Slave Select (SS)** line low to enable the slave. The 8-bit data from the master is loaded into a shift register. With each clock pulse:

- The master shifts out one bit on MOSI.

- The slave reads that bit on the rising edge of SCLK.

After 8 clock cycles, all 8 bits have been transferred. The SS line is then pulled high, marking the end of the frame.

4.1.3 3. Data Reception at the Slave

The slave has an 8-bit shift register synchronized with the master's clock. It continuously shifts in bits from the MOSI line and latches them once the full byte is received. The final 8-bit data is then mapped to LEDs, visually confirming successful data transfer.

4.1.4 4. Master-Slave Synchronization Logic

Both boards contain Finite State Machines (FSMs) that control their operation. **Master FSM:**

- IDLE → LOAD → TRANSFER → COMPLETE

Slave FSM:

- IDLE → RECEIVE → DISPLAY

This FSM-based design ensures that communication remains deterministic and error-free.

4.1.5 5. Timing Considerations

SPI timing requires that data be stable before the sampling edge. Therefore, MOSI is updated on the falling edge and read on the rising edge of the clock. This guarantees no setup/hold violations.

4.2 Connection Diagram

Signal	**Master (JA Header)**	**Slave (JA Header)**	**Direction**
MOSI	JA1	JA1	Master → Slave
MISO	JA2	JA2	Slave → Master
SCLK	JA3	JA3	Master → Slave
SS	JA4	JA4	Master → Slave
GND	GND	GND	Common Ground

Table 1: Pin Connections Between Master and Slave ZedBoards

5 Verilog HDL Implementation

5.1 Master Module Code

Listing 1: SPI Master Module

```verilog
`timescale 1ns / 1ps
module spi_master(
    input  wire clk,              // System clock (100 MHz)
    input  wire reset_n,          // Active-low reset
    input  wire start,            // Start signal to initiate SPI
        transfer
    input  wire [7:0] mosi_data,  // Data to transmit
    output reg  [7:0] miso_data,  // Data received from slave
    output reg  done,             // Indicates transfer complete
    output reg  SCLK,             // SPI Clock
    output reg  MOSI,             // Master Out Slave In
    input  wire MISO,             // Master In Slave Out
    output reg  SS                // Slave Select (Active Low)
);
    reg [3:0] bit_cnt;
    reg [7:0] shift_reg;
    reg [2:0] state;
    parameter IDLE = 3'b000,
              LOAD = 3'b001,
              TRANSFER = 3'b010,
              DONE = 3'b011;

    always @(posedge clk or negedge reset_n) begin
        if(!reset_n) begin
            state     <= IDLE;
            SCLK      <= 0;
            SS        <= 1;
            MOSI      <= 0;
            done      <= 0;
            bit_cnt   <= 0;
            miso_data <= 8'd0;
            shift_reg <= 8'd0;
        end else begin
            case(state)
            IDLE: begin
                done <= 0;
                SCLK <= 0;
                SS   <= 1;
                if(start)
                    state <= LOAD;
            end

            LOAD: begin
                SS        <= 0;             // Activate slave
                shift_reg <= mosi_data;     // Load data
                bit_cnt   <= 7;             // 8 bits total
                state     <= TRANSFER;
            end

            TRANSFER: begin

                SCLK <= ~SCLK;

                if(SCLK == 0) begin

                    MOSI <= shift_reg[bit_cnt];
                end else begin

                    shift_reg[bit_cnt] <= MISO;
                    if(bit_cnt == 0)
                        state <= DONE;
                    else
                        bit_cnt <= bit_cnt - 1;
                end
            end

            DONE: begin
                SCLK      <= 0;
                SS        <= 1;             // Deactivate slave
                miso_data <= shift_reg;
                done      <= 1;
                state     <= IDLE;
```

```
                    end
                endcase
            end
        end
endmodule
```

5.2 Master Test bench code

Listing 2: SPI Master Test bench Code

```
`timescale 1ns/1ps
module tb_spi_gate;
    reg clk, rst, start;
    reg miso;
    wire mosi, sclk, cs;
    wire [7:0] miso_data;

    spi_master_gate uut_master (
        .clk(clk),
        .rst(rst),
        .start(start),
        .mosi_data(8'b11001100),
        .miso_data(miso_data),
        .miso(miso),
        .mosi(mosi),
        .sclk(sclk),
        .cs(cs)
    );

    spi_slave_gate uut_slave (
        .sclk(sclk),
        .cs(cs),
        .mosi(mosi),
        .miso(miso)
    );

    initial clk = 0;
    always #5 clk = ~clk; // 100 MHz clock

    initial begin
```

```
        rst = 1; start = 0;
        #20 rst = 0;
        #50 start = 1;
        #10 start = 0;
        #2000;
        $display("Received␣data:␣%b", miso_data);
        $finish;
    end

endmodule
```

5.3 Master board top level module code

Listing 3: Master board top module code

```
`timescale 1ns / 1ps
module top_spi_master_zedboard(
    input  wire clk,
    input  wire reset_n,
    input  wire start,        // From BTNU
    input  wire [7:0] SW,     // 8 DIP switches
    output wire MOSI,
    input  wire MISO,
    output wire SCLK,
    output wire SS
);

    wire [7:0] tx_data;
    wire [7:0] rx_data;
    wire done;

    // Send DIP switch values
    assign tx_data = SW;

    spi_master spi_inst (
        .clk(clk),
        .reset_n(reset_n),
        .start(start),
        .mosi_data(tx_data),
        .miso_data(rx_data),
        .done(done),
        .SCLK(SCLK),
        .MOSI(MOSI),
        .MISO(MISO),
        .SS(SS)
    );

endmodule
```

5.4 Master board Constraints (XDC) File

Listing 4: Master board Constraints file

```

## -------------------------------
## Clock (100 MHz) - Bank 13 (3.3V)
## -------------------------------
set_property PACKAGE_PIN Y9 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 10.0 -name sys_clk -waveform {0 5} [
   get_ports clk]

## -------------------------------
## Reset Button (BTNC) - Bank 34 (1.8V)
## -------------------------------
set_property PACKAGE_PIN P16 [get_ports reset_n]
set_property IOSTANDARD LVCMOS18 [get_ports reset_n]

## -------------------------------
## Start Button (BTNU) - Bank 34 (1.8V)
## -------------------------------
set_property PACKAGE_PIN R18 [get_ports start]
set_property IOSTANDARD LVCMOS18 [get_ports start]

## -------------------------------
## DIP Switches SW[7:0] - Bank 35 (1.8V)
## -------------------------------
set_property PACKAGE_PIN F22 [get_ports {SW[0]}]
set_property PACKAGE_PIN G22 [get_ports {SW[1]}]
set_property PACKAGE_PIN H22 [get_ports {SW[2]}]
set_property PACKAGE_PIN F21 [get_ports {SW[3]}]
set_property PACKAGE_PIN H19 [get_ports {SW[4]}]
set_property PACKAGE_PIN H18 [get_ports {SW[5]}]
set_property PACKAGE_PIN H17 [get_ports {SW[6]}]
set_property PACKAGE_PIN M15 [get_ports {SW[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[*]}]

## ------------------------------
## SPI Interface (PMOD JA) - Bank 13 (3.3V)
## ------------------------------
# JA1 = SS, JA2 = MOSI, JA3 = MISO, JA4 = SCLK
set_property PACKAGE_PIN Y11  [get_ports SS]
set_property PACKAGE_PIN AA11 [get_ports MOSI]
set_property PACKAGE_PIN Y10  [get_ports MISO]
set_property PACKAGE_PIN AA9  [get_ports SCLK]

set_property IOSTANDARD LVCMOS33 [get_ports SS]
set_property IOSTANDARD LVCMOS33 [get_ports MOSI]
set_property IOSTANDARD LVCMOS33 [get_ports MISO]
set_property IOSTANDARD LVCMOS33 [get_ports SCLK]

## ------------------------------
## Bank-wide I/O Voltage Declaration
## ------------------------------
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [
   get_iobanks 13]]
set_property IOSTANDARD LVCMOS18 [get_ports -of_objects [
   get_iobanks 34]]
set_property IOSTANDARD LVCMOS18 [get_ports -of_objects [
   get_iobanks 35]]
```

5.5 Slave Module Code

Listing 5: SPI Slave Module

```
`timescale 1ns / 1ps
module spi_slave(
    input  wire        clk,        // System clock (not SPI
       clock)
    input  wire        reset_n,    // Active-low reset
    input  wire        SCLK,        // SPI clock from master
    input  wire        SS,          // Slave select (active low)
    input  wire        MOSI,        // Master Out Slave In
    output reg         MISO,        // Master In Slave Out
    output reg   [7:0] miso_data,   // Data received from master
    input  wire  [7:0] mosi_data,   // Data to send back
    output reg         done         // Indicates data reception complete
);
    reg [7:0] shift_in;
    reg [7:0] shift_out;
    reg [2:0] bit_cnt;
    reg       sclk_prev;
    always @(posedge clk or negedge reset_n) begin
        if(!reset_n) begin
            shift_in   <= 8'd0;
            shift_out  <= 8'd0;
            miso_data  <= 8'd0;
            bit_cnt    <= 3'd0;
            MISO       <= 1'b0;
            done       <= 1'b0;
            sclk_prev  <= 1'b0;
        end else begin
            sclk_prev <= SCLK;
            if(!SS) begin
                done <= 1'b0;
                if(sclk_prev == 0 && SCLK == 1) begin
                    // Rising edge: sample MOSI
                    shift_in <= {shift_in[6:0], MOSI};
                    bit_cnt  <= bit_cnt + 1;
                end
                else if(sclk_prev == 1 && SCLK == 0) begin
                    // Falling edge: drive MISO
                    MISO <= shift_out[7];
                    shift_out <= {shift_out[6:0], 1'b0};
                end
                if(bit_cnt == 3'd7 && sclk_prev == 0 && SCLK == 1) begin
                    miso_data <= {shift_in[6:0], MOSI};
                    done <= 1'b1;
                    bit_cnt <= 0;
                    shift_out <= mosi_data; // Prepare next byte
                       to send
                end
            end else begin
                // When slave not selected
                bit_cnt <= 0;
                done <= 0;
                MISO <= 0;
                shift_out <= mosi_data;
            end
        end
    end
endmodule
```

5.6 Slave test bench Code

Listing 6: SPI Slave Test bench code

```
`timescale 1ns/1ps
module tb_spi_slave_gate;
    reg sys_clk;
    reg reset_n;
    reg sclk;
    reg ss;
    reg mosi;
    wire miso;
    wire [7:0] miso_data;
    reg  [7:0] master_tx_data;
    reg  [7:0] slave_tx_data;
    integer i;
    spi_slave uut_slave (
        .clk(sys_clk),
        .reset_n(reset_n),
        .SCLK(sclk),
        .SS(ss),
        .MOSI(mosi),
        .MISO(miso),
        .miso_data(miso_data),
        .mosi_data(slave_tx_data),
        .done()
    );
    initial sys_clk = 0;
    always #5 sys_clk = ~sys_clk;  // 100 MHz
    initial begin
        sclk = 0;
        ss   = 1;
        mosi = 0;
        master_tx_data = 8'b1010_1100;
        slave_tx_data  = 8'b0101_1010;
        reset_n = 0;
        #50;
        reset_n = 1;
        #100;
        ss = 0;
        $display("Starting␣SPI␣transfer␣at␣time␣%t", $time);
        for (i = 7; i >= 0; i = i - 1) begin
            mosi = master_tx_data[i];
            #50 sclk = 1;
            #50 sclk = 0;
            $display("Bit␣%0d␣sent␣=␣%b,␣MISO␣=␣%b", i, mosi,
               miso);
        end
        s = 1;
        #100;
        $display("Transfer␣complete␣at␣%t", $time);
        $display("Slave␣received␣(miso_data)␣=␣%b", miso_data);
        #100;
        $finish;
    end
endmodule
```

5.7 Slave board top level module code

Listing 7: SPI Slave top level Module code

```
`timescale 1ns / 1ps
module top_spi_slave (
    input  wire        clk,        // 100 MHz system clock from
       board
    input  wire        reset_n,    // Active-low reset (
       pushbutton)
    input  wire        SCLK,       // SPI clock from master
    input  wire        SS,         // Slave Select (active low)
    input  wire        MOSI,       // Master Out Slave In
    output wire        MISO,       // Master In Slave Out
    output wire [7:0]  LD          // LEDs to show received data
);
    wire [7:0] miso_data;
    wire       done;
    reg  [7:0] mosi_data_reg = 8'h5A;
    spi_slave u_spi_slave (
        .clk(clk),
        .reset_n(reset_n),
        .SCLK(SCLK),
        .SS(SS),
        .MOSI(MOSI),
        .MISO(MISO),
        .miso_data(miso_data),
        .mosi_data(mosi_data_reg),
        .done(done)
    );
    assign LD = miso_data;
endmodule
```

5.8 Slave board Contraints(XDC) File

Listing 8: SPI Slave Board Constraints File

```
# ==========================================
# System Clock - 3.3V, Bank 13
# ==========================================
set_property PACKAGE_PIN Y9 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
create_clock -period 10.0 -name sys_clk -waveform {0 5} [
   get_ports clk]

# ==========================================
# Reset Button - 1.8V, Bank 34
# ==========================================

set_property PACKAGE_PIN P16 [get_ports reset_n]
set_property IOSTANDARD LVCMOS18 [get_ports reset_n]

# ==========================================
# SPI Interface on PMOD JA - Bank 35, 1.8V
# ==========================================

set_property PACKAGE_PIN AA9 [get_ports SCLK]
set_property IOSTANDARD LVCMOS18 [get_ports SCLK]

set_property PACKAGE_PIN AA11 [get_ports MOSI]
set_property IOSTANDARD LVCMOS18 [get_ports MOSI]

set_property PACKAGE_PIN Y10 [get_ports MISO]
set_property IOSTANDARD LVCMOS18 [get_ports MISO]

set_property PACKAGE_PIN Y11 [get_ports SS]
set_property IOSTANDARD LVCMOS18 [get_ports SS]

# ==========================================
# User LEDs - Bank 33
# ==========================================

set_property PACKAGE_PIN T22 [get_ports {LD[0]}];  # "LD0"
set_property IOSTANDARD LVCMOS33 [get_ports LD[0]]

set_property PACKAGE_PIN T21 [get_ports {LD[1]}];  # "LD1"
set_property IOSTANDARD LVCMOS33 [get_ports LD[1]]

set_property PACKAGE_PIN U22 [get_ports {LD[2]}];  # "LD2"
set_property IOSTANDARD LVCMOS33 [get_ports LD[2]]

set_property PACKAGE_PIN U21 [get_ports {LD[3]}];  # "LD3"
set_property IOSTANDARD LVCMOS33 [get_ports LD[3]]

set_property PACKAGE_PIN V22 [get_ports {LD[4]}];  # "LD4"
set_property IOSTANDARD LVCMOS33 [get_ports LD[4]]

set_property PACKAGE_PIN W22 [get_ports {LD[5]}];  # "LD5"
set_property IOSTANDARD LVCMOS33 [get_ports LD[5]]

set_property PACKAGE_PIN U19 [get_ports {LD[6]}];  # "LD6"
set_property IOSTANDARD LVCMOS33 [get_ports LD[6]]

set_property PACKAGE_PIN U14 [get_ports {LD[7]}];  # "LD7"
set_property IOSTANDARD LVCMOS33 [get_ports LD[7]]
```

5.9 Schematic diagram of Master and Slave Communications

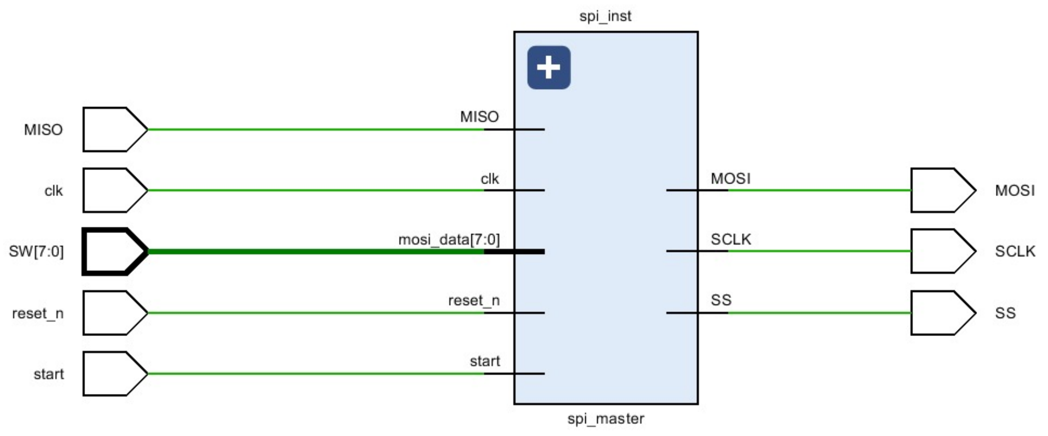

Figure 2: Schematic of SPI Master Communication

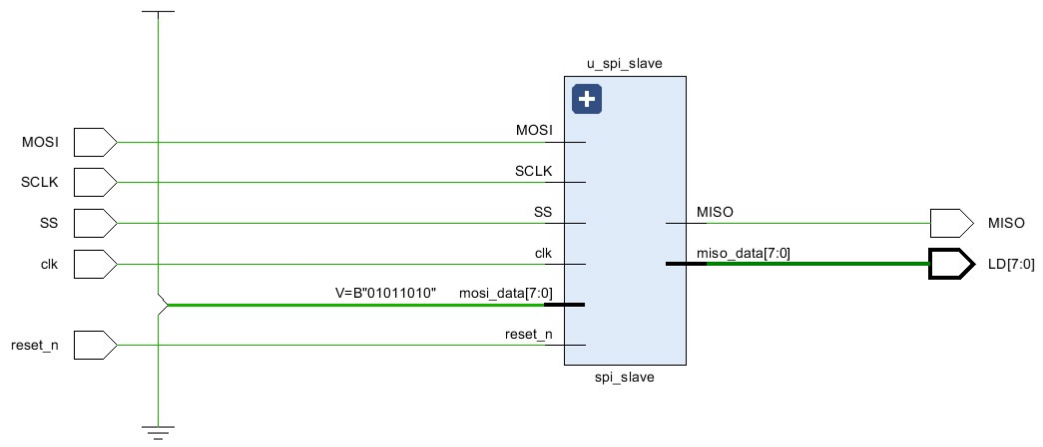

Figure 3: Schematic of SPI Slave Communication

6 ZedBoard Implementation

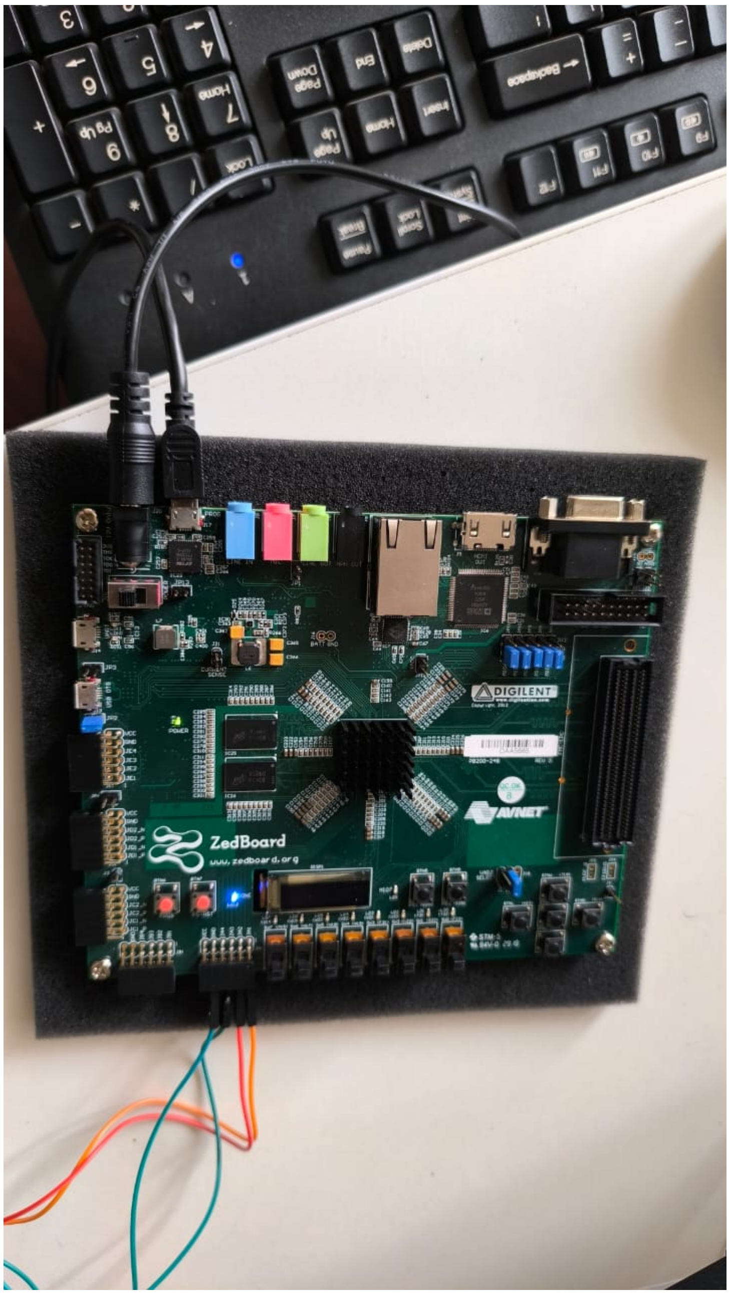

Figure 4: SPI Master Implementation on ZedBoard

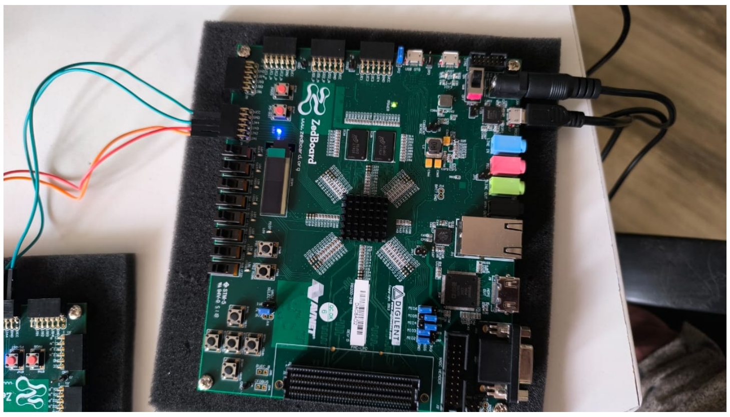

Figure 5: SPI Slave Implementation on ZedBoard

7 Results and Discussion

- The SPI protocol was successfully established between two ZedBoards.
- The Master transmitted 8-bit data patterns accurately.
- The Slave displayed corresponding LED patterns with no data loss.
- Timing waveforms verified synchronization between MOSI, SCLK, and SS.

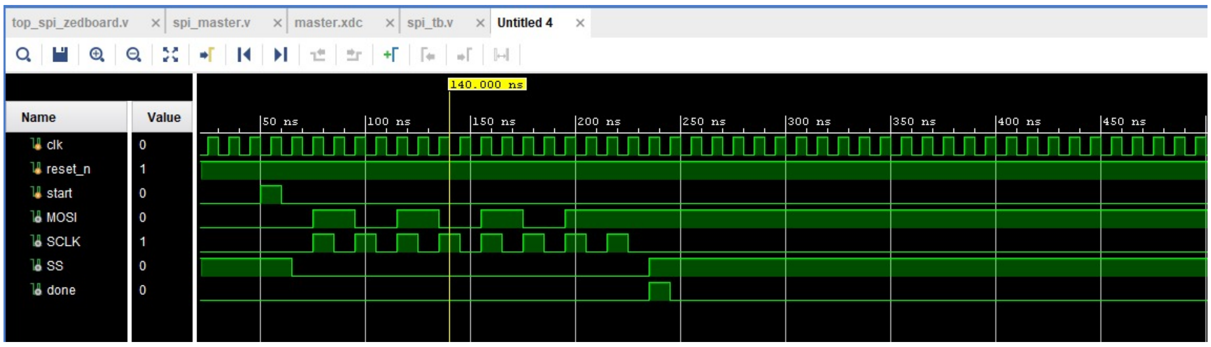

Figure 6: Simulation Waveform of SPI Master Communication

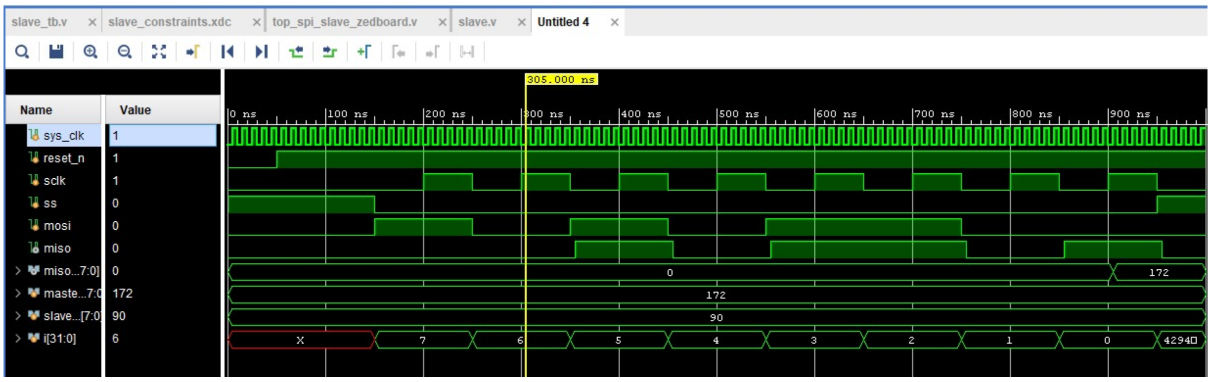

Figure 7: Simulation Waveform of SPI Slave Communication

8 Advantages

- High-speed synchronous data transfer.
- Simple and efficient design using minimal FPGA resources.
- Supports full-duplex communication.
- Easily scalable for multi-slave systems.

9 Conclusion

The SPI protocol was successfully designed and implemented using two ZedBoards in a Master-Slave configuration. The system efficiently transferred an 8-bit pattern and displayed it on the Slave board's LEDs, confirming reliable data exchange. The implementation verified the fundamental working of SPI communication and demonstrated FPGA-based protocol realization.

References

1. Digilent ZedBoard Reference Manual.
2. Xilinx Vivado Design Suite User Guide.
3. Motorola SPI Bus Specification.
4. P. P. Chu, "FPGA Prototyping by Verilog Examples," Wiley Publications.