MINI-PROJECT FOR THE COURSE OF FPGA LAB

IMPLEMENTATION OF SPI PROTOCOL ON FPGA BOARD

Submitted To:

T.V.K HANUMANTHA RAO

Professor – Department of Electronics and Communication

ATUL KUMAR NISHAD

Assistant Professor – Department of Electronics and Communication Engineering

Submitted by:

D. SAI VISWAS (21ECB0A14)

G. SUSHMA (21ECEB0A15)

K. BHAVYA (21ECB0A28)

N. AKHILA (21ECB0A38)

TABLE OF CONTENTS

ABSTRACT

Today, at the low end of the Communication Protocols there are mainly Two Protocols: Inter- Integrated circuit (I2C) and the Serial Peripheral Interface (SPI) Protocols. Both the protocols are well suited for communications between Integrated Circuits for communication with ON-Board Peripherals. SPI protocol has the advantage of being a full-duplex communication interface and can transfer data at very high rates (around tens of MHzs).

SPI is one of the most commonly used serial protocols for both inter-chip and intra-chip low/medium speed data-stream transfer. In conformity with design-reuse methodology, this project introduces SPI IP with one Master One Slave configuration with that of 8-bit data transfer which incorporates all necessary features required by modern ASIC/SoC applications.

The Designed SPI is used for communication between different peripherals with that of a processor in a SoC application. The Designed SPI is Implemented and also Verified using a Verilog in order to show its code coverage and functional correctness. The whole RTL design code is written in Verilog for synthesis and its Verification.

INTRODUCTION

There are many communication protocols for both short and long distance communication purpose such as ETHERNET, USB,SATA ,PCI-EXPRESS are used for long distance and I2C and SPI are used for short distance communications. SPI is a serial interface protocol, compared to other protocols, it has high transmission speed, simple to use and little pins advantages. The four interfaces are required by standard SPI protocol at least.

Usually, the devices which based on SPI protocol are divided into master device and slave-device for transmitting the data. The chip select signal and clock signal have be generated by the master-device when the data exchange has been processed. SPI is often considered as the "little" communication protocol which is used for On-Board communication.

ABOUT THE SPI PROTOCOL

Standard SPI is a high-speed, full-duplex, synchronous communication bus. For saving the chip ports and space on PCB layout, the ports of the SPI only take four lines. It is working in the Master-Slave full duplex mode which has one master device and one slave device and requires four lines whose components are SDI (data in), SDO (data out), SCK (clock), SS (Slave select) .When the SPI master wants to send data to a slave, it will pull the SS line low for selecting slave, and activates the clock signal which usable between the master and the slave at same time. The master transmits the data to the MOSI (master's SDO and slave's SDI) line and receives the data from the MISO (master's SDI and slave's SDO) line at the time.

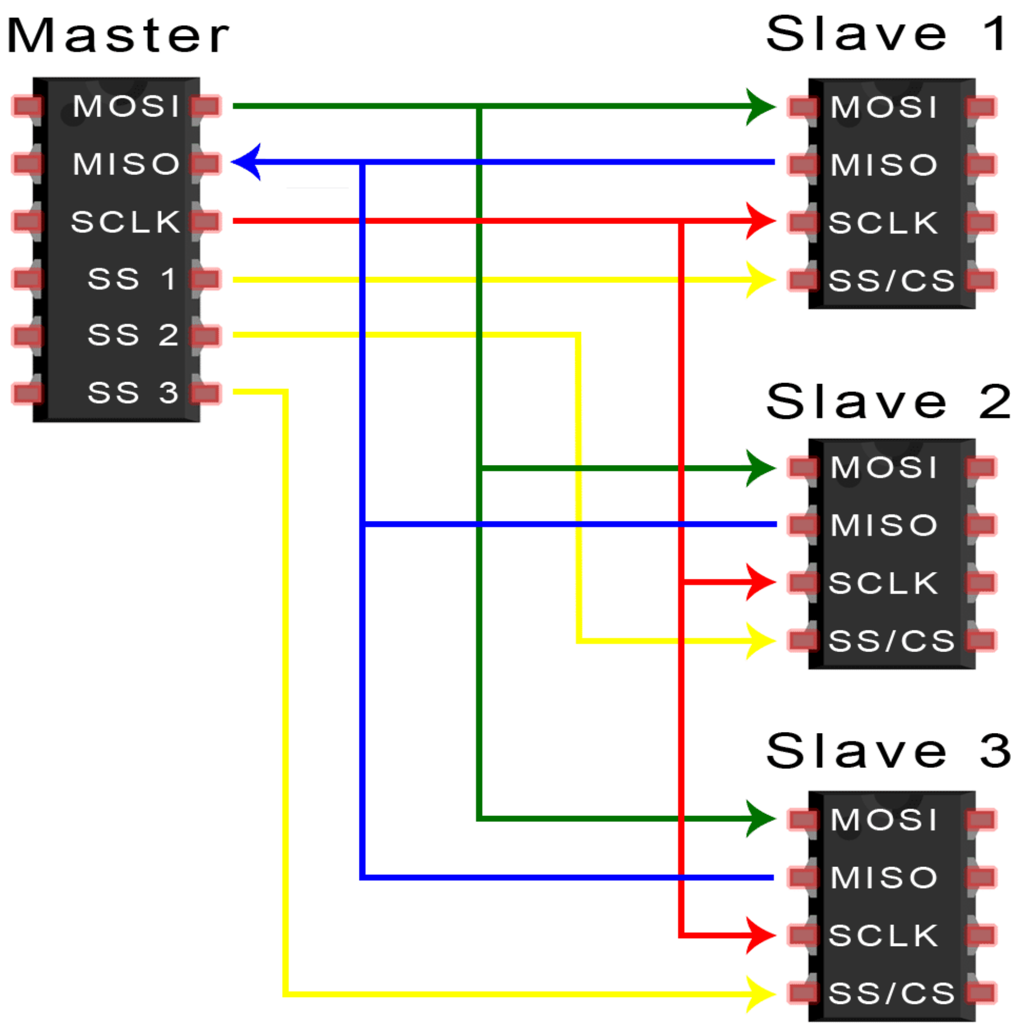
SPI is a serial communication protocol, that data is transmitted bit by bit. The clock pulse is provided by SCK and SDI, SDO is based on this pulse to making the data transmission. Data output through the master's SDO line at the rising or falling edge of the clock, and be read by slave in the falling or rising edge followed. So 8-bit data transfer need at least 8 times the clock signal changes.

This module appears to be an implementation of a Serial Peripheral Interface (SPI) controller. It takes in several input signals including a clock, a reset signal, a start signal, and an 8-bit data input signal. It also has several output signals including a MOSI (Master Out Slave In) signal, an SCK (Serial Clock) signal, an 8-bit data output signal, and several control signals including a busy signal and a new_data signal.

The module has several internal registers to keep track of its state and data. It uses a finite state machine to manage its operation. The states are defined as IDLE, WAIT_HALF, and TRANSFER. In the IDLE state, the module waits for a start signal to be asserted. Once the start signal is detected, the module transitions to the WAIT_HALF state where it waits for the SCK signal to reach its half cycle before transitioning to the TRANSFER state where it sends and receives data in a synchronous manner with the SCK signal.

During the TRANSFER state, the module sends the most significant bit (MSB) of the 8-bit data output signal over the MOSI signal on the rising edge of the SCK signal. On the falling edge of the SCK signal, the module receives the data input signal on the MISO (Master In Slave Out) signal and shifts it into the 8-bit data register. Once all 8 bits have been sent and received, the module transitions back to the IDLE state and outputs the received data on the data_out signal.

Overall, this module is a basic implementation of an SPI controller that can be used to communicate with other SPI devices.



Block diagram of SPI Protocol

MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.

MISO (Master Input/Slave Output) – Line for the slave to send data to the master.

SCLK (Clock) – Line for the clock signal.

SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

Wires used	4
Maximum Speed	Up to 10 Mbps
Synchronous or Asynchronous	Synchronous
Serial or Parallel?	Serial
Max #of Masters	1
Max # of slaves	Theoretically Unlimited

WORKING OF SPI

THE CLOCK

The clock signal synchronizes the output of data bits from the master to the sampling of bits by the slave. One bit of data is transferred in each clock cycle, so the speed of data transfer is determined by the frequency of the clock signal. SPI communication is always initiated by the master since the master configures and generates the clock signal.

Any communication protocol where devices share a clock signal is known as synchronous. SPI is a synchronous communication protocol.

There are also asynchronous methods that don't use a clock signal. For example, in UART communication, both sides are set to a pre-configured baud rate that dictates the speed and timing of data transmission.

The clock signal in SPI can be modified using the properties of clock polarity and clock phase. These two properties work together to define when the bits are output and when they are sampled. Clock polarity can be set by the master to allow for bits to be output and sampled on either the rising or falling edge of the clock cycle. Clock phase can be set for output and sampling to occur on either the first edge or second edge of the clock cycle, regardless of whether it is rising or falling.

SLAVE SELECT

The master can choose which slave it wants to talk to by setting the slave's CS/SS line to a low voltage level. In the idle, non-transmitting state, the slave select line is kept at a high voltage level. Multiple CS/SS pins may be available on the master, which allows for multiple slaves to be wired in parallel. If only one CS/SS pin is present, multiple slaves can be wired to the master by daisy-chaining.

MULTIPLE SLAVES

SPI can be set up to operate with a single master and a single slave, and it can be set up with multiple slaves controlled by a single master. There are two ways to connect multiple slaves to the master. If the master has multiple slave select pins, the slaves can be wired in parallel. If only one slave select pin is available, the slaves can be daisy-chained.

MOSI AND MISO

The master sends data to the slave bit by bit, in serial through the MOSI line. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with the most significant bit first.

The slave can also send data back to the master through the MISO line in serial. The data sent from the slave back to the master is usually sent with the least significant bit first.

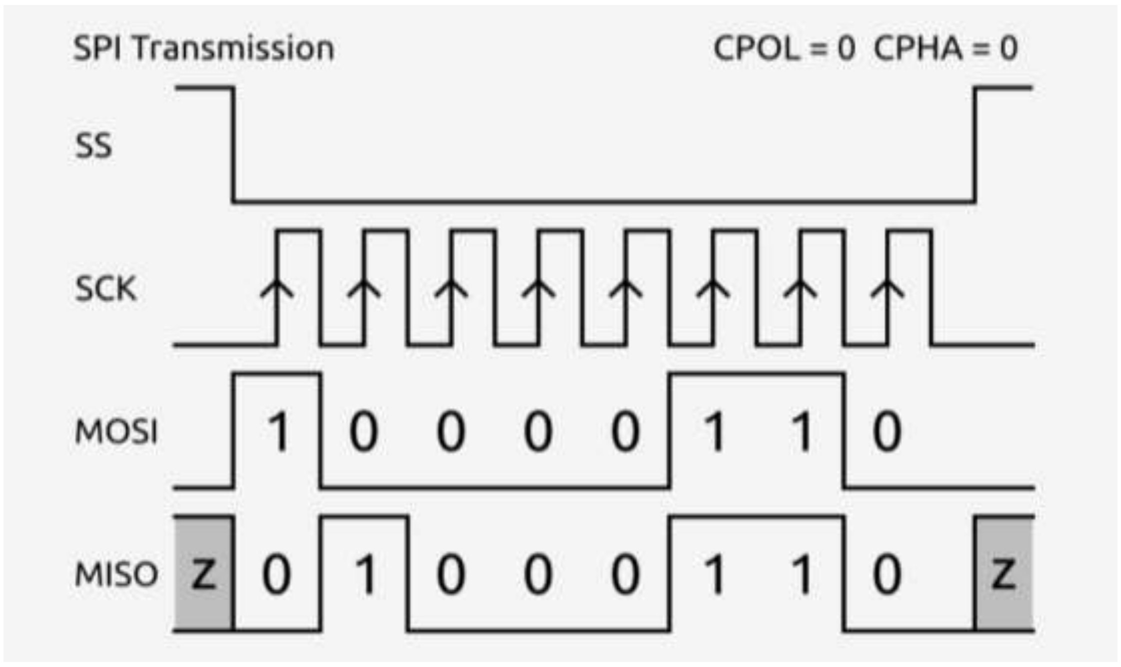
Modes of SPI Protocol:
There are 4 different SPI bus standards that all have to do with the SCK signal. The 4 modes are broken down into two parameters, CPOL and CPHA. CPOL stands for Clock Polarity and designates the default value (high/low) of the SCK signal when the bus is idle. CPHA stands for Clock Phase and determines which edge of the clock data is sampled (rising/falling). The data sheet for any device will specify these parameters so you can adjust accordingly. The most common settings are CPOL=0 (idle low) and CPHA=0 (sample rising edge).

Mode 0: In this mode, the clock polarity (CPOL) is set to 0 and the clock phase (CPHA) is set to 0. This means that the data is sampled on the leading edge of the clock signal (when the clock transitions from low to high) and the data is output on the falling edge of the clock (when the clock transitions from high to low).

Mode 1: In this mode, the CPOL is set to 0 and the CPHA is set to 1. This means that the data is sampled on the trailing edge of the clock (when the clock transitions from high to low) and the data is output on the rising edge of the clock (when the clock transitions from low to high).

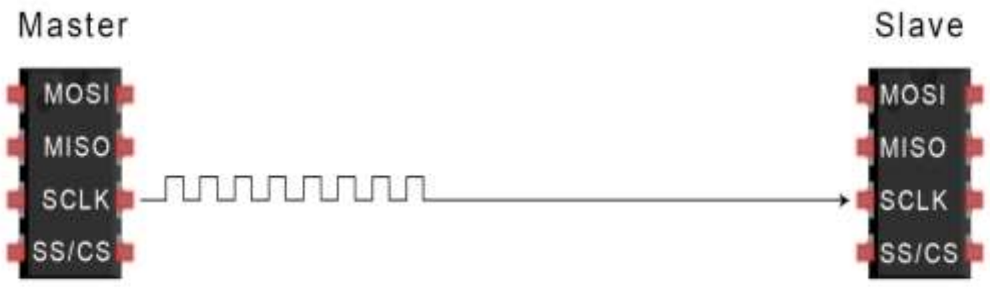
Mode 2: In this mode, the CPOL is set to 1 and the CPHA is set to 0. This means that the data is sampled on the leading edge of the clock and the data is output on the rising edge of the clock.

Mode 3: In this mode, the CPOL is set to 1 and the CPHA is set to 1. This means that the data is sampled on the trailing edge of the clock and the data is output on the falling edge of the clock.
The specific mode that is used depends on the requirements of the specific application and the devices that are communicating with each other.
Here is an example transfer with CPOL=0 and CPHA=0.

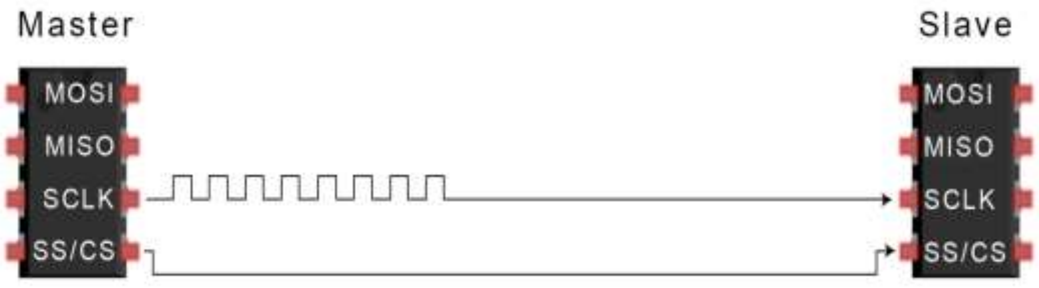


STEPS OF DATA TRANSMISSION:

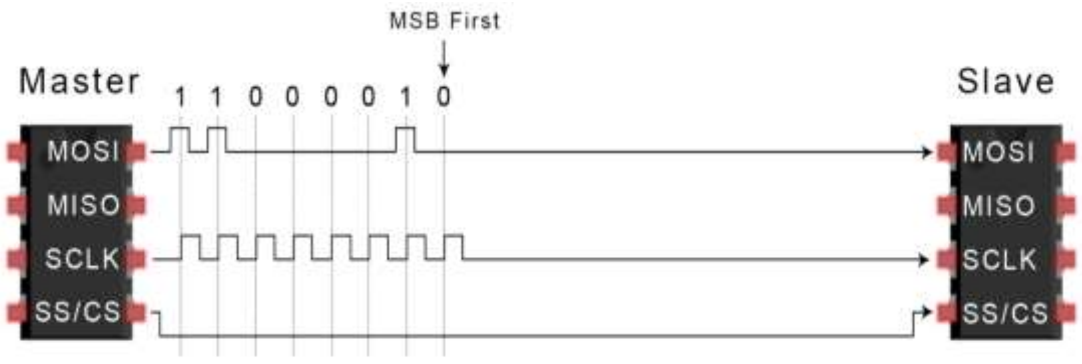
1. The SPI protocol gets initialised when the master firsts starts generating the synchronising clock pulse, this is the basis on which the whole communication occurs.



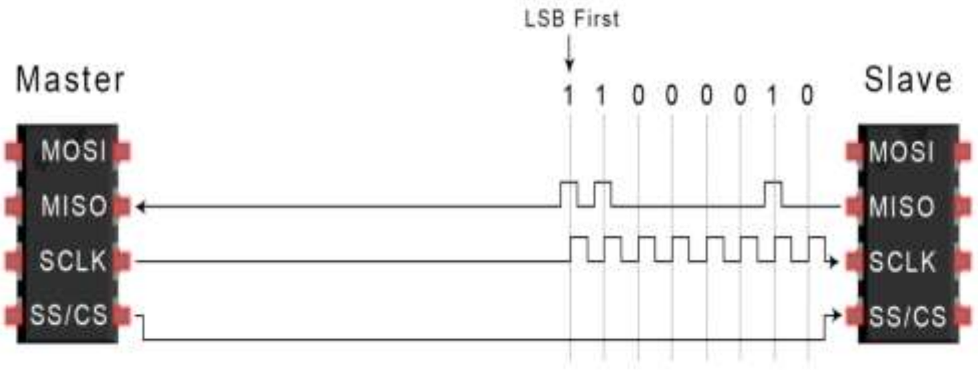
2. After initialisation, the master chooses the slave it needs to communicate with. This is done using the slave select line. The CS line to each of the slaves is HIGH by default, and once the master wants to activate any particular slave, it pulls the CS pin of that slave to LOW until the communication is finished.



3. This is followed by the master or the slave sending the encrypted data bit by bit. The slave transmits through the MISO line and the master transmits through the MOSI line.
   Also, the transmission happens in the MSB first format.



4. The received data is then stored in a register.
5.Once the master feels that the communication is completed, it released the CS pin back to HIGH.



VERILOG CODE:

//module master

```
module spi_master(
    input wire clk,
    input wire reset,
   input wire [15:0] datain,
   output wire spi_cs_l,
   output wire spi_sclk,
   output wire spi_data,
   output [4:0] counter
);
//reg dclk;
   reg[15:0]MOSI;
  reg[4:0]count;
  reg cs_l;
  reg sclk;
  reg [2:0]state;

  always@(posedge clk or posedge reset)
      if(reset)begin
         MOSI<=16'b0;
        count<=5'd16;
        cs_l<=1'b1;
        sclk<=1'b0;
end
  else  begin
case(state)
     0:begin
     sclk<=1'b0;
      cs_l<=1'b1;
      state<=1;
end
1:begin
       sclk<=1'b0;
       cs_l<=1'b0;
      MOSI<=datain[count-1];
      count<=count-1;
     state<=2;
end
2:begin
     sclk<=1'b1;
     if(count>0)
    state<=1;
else begin
     count<=16;
     state<=0;
end
end
        default:state<=0;
endcase
end
        assign spi_cs_l=cs_l;
        assign spi_sclk=sclk;
       assign spi_data=MOSI;
       assign counter=count;
endmodule

//slave module
module spi_slave(
    input clk,
    input rst,
    input ss,
    input mosi,
    output miso,
    input sck,
    output done,
    input [7:0] din,
    output [7:0] dout
  );

  reg mosi_d, mosi_q;
  reg ss_d, ss_q;
  reg sck_d, sck_q;
  reg sck_old_d, sck_old_q;
  reg [7:0] data_d, data_q;
  reg done_d, done_q;
  reg [2:0] bit_ct_d, bit_ct_q;
  reg [7:0] dout_d, dout_q;
  reg miso_d, miso_q;

  assign miso = miso_q;
  assign done = done_q;
  assign dout = dout_q;

  always @(*) begin
    ss_d = ss;
    mosi_d = mosi;
    miso_d = miso_q;
    sck_d = sck;
    sck_old_d = sck_q;
```

```
    data_d = data_q;
    done_d = 1'b0;
    bit_ct_d = bit_ct_q;
    dout_d = dout_q;

    if (ss_q) begin                    // if slave select is high deselcted)
      bit_ct_d = 3'b0;                  // reset bit counter
      data_d = din;                    // read in data
      miso_d = data_q[7];                   // output MSB
    end else begin                      // else slave select is low
(selected)
      if (!sck_old_q && sck_q) begin        // rising edge
        data_d = {data_q[6:0], mosi_q};      // read data in and shift
        bit_ct_d = bit_ct_q + 1'b1;         // increment the bit
counter
        if (bit_ct_q == 3'b111) begin        // if we are on the last bit
          dout_d = {data_q[6:0], mosi_q};     // output the byte
          done_d = 1'b1;                  // set transfer done flag
          data_d = din;                  // read in new byte
        end
      end else if (sck_old_q && !sck_q) begin // falling edge
        miso_d = data_q[7];                // output MSB
      end
    end
  end

  always @(posedge clk) begin
    if (rst) begin
      done_q <= 1'b0;
      bit_ct_q <= 3'b0;
      dout_q <= 8'b0;
      miso_q <= 1'b1;
    end else begin
      done_q <= done_d;
      bit_ct_q <= bit_ct_d;
      dout_q <= dout_d;
      miso_q <= miso_d;
    end

    sck_q <= sck_d;
    mosi_q <= mosi_d;
    ss_q <= ss_d;
    data_q <= data_d;
    sck_old_q <= sck_old_d;

  end
endmodule
//top module
module top (clk, rst,start,data_in,ss,busy,new_data,data);
       input clk, rst, start,ss;
       input [7:0] data_in;
       output busy, new_data;
       inout [7:0] data;

         wire sckw;
        wire busy,mosiw, misow, done;
        wire [7:0] data_outw;

   spi sm1(clk,rst,misow,data_in,start,sckw,data,busy,
new_data, mosiw);
```

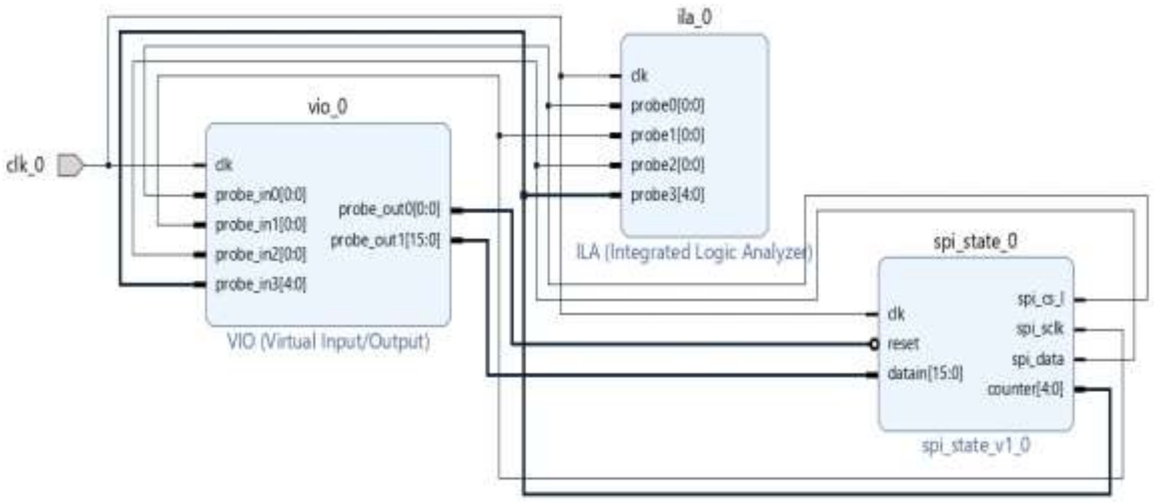
```
  spi_slave
sm2(clk,rst,ss,mosiw,sckw,data,data_outw,done,misow);
endmodule
```

TEST BENCH

```
//master module
module spi_tb();
     reg clk;
     reg reset;
     reg [15:0]datain;
    wire spi_cs_l;
    wire spi_sclk;
    wire spi_data;
    wire [4:0]counter;

spi_state dut(
.clk(clk),
    .reset(reset),
   .counter(counter),
  .datain(datain),
   .spi_cs_l(spi_cs_l),
  .spi_sclk(spi_sclk),
  .spi_data(spi_data)
);
initial begin
    clk=0;
   reset=1;
  datain=0;
end
```

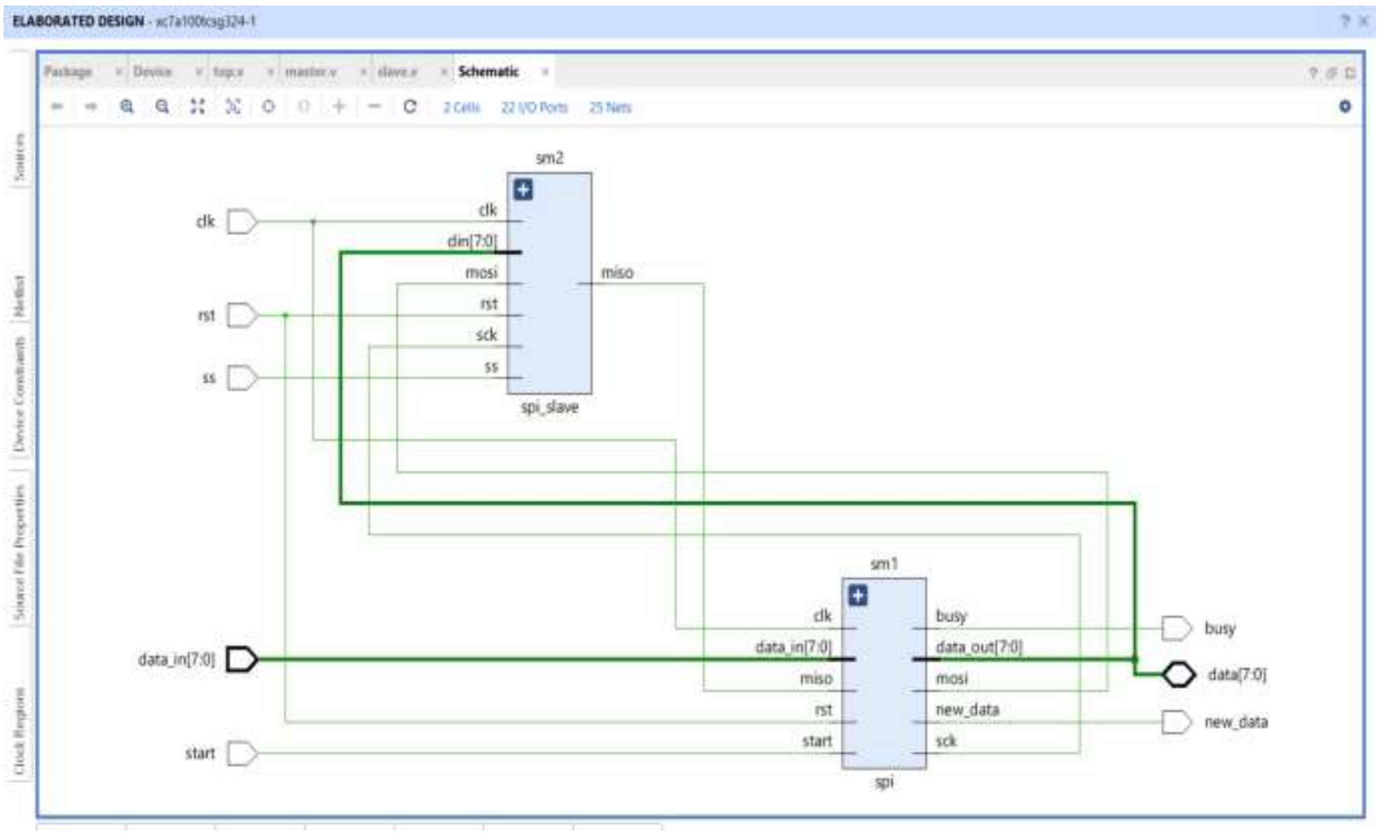
```
    always #5 clk=~clk;
initial begin
     #10 reset=1'b0;
     #10 datain=16'hA569;
     #335 datain=16'h2563;
     #335 datain=16'h9B63;
    #335 datain=16'h6A61;
    #335 datain=16'hA265;
    #335 datain=16'h7564;
end

endmodule
```

Block Design using VIO and ILA for Master:

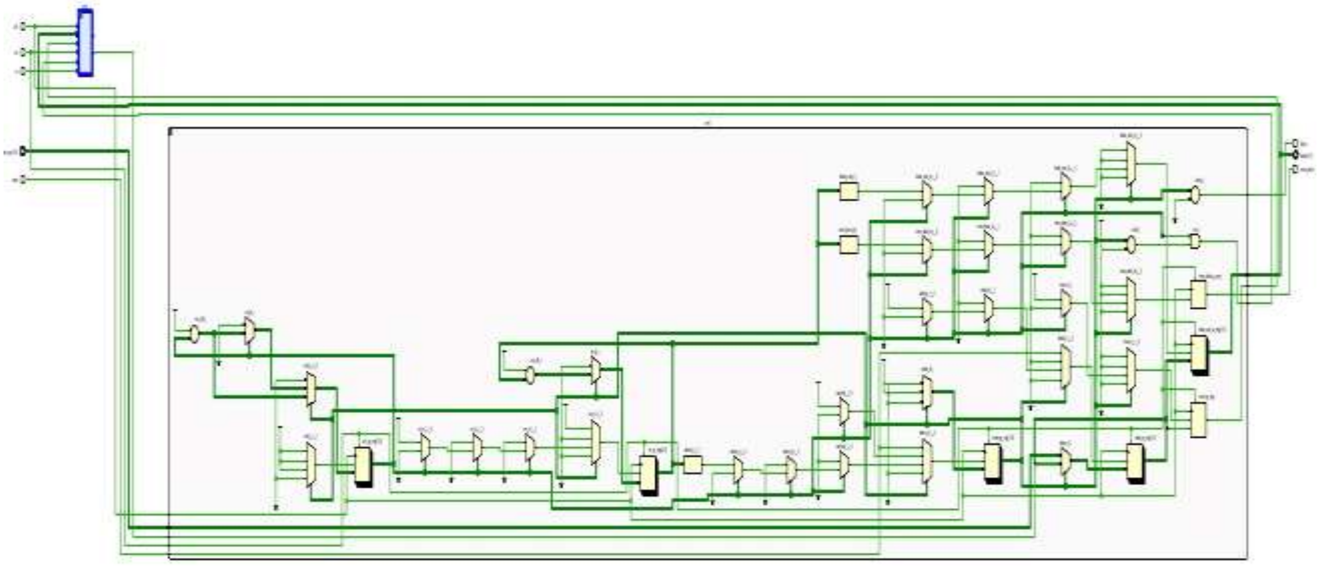


SCHEMATIC

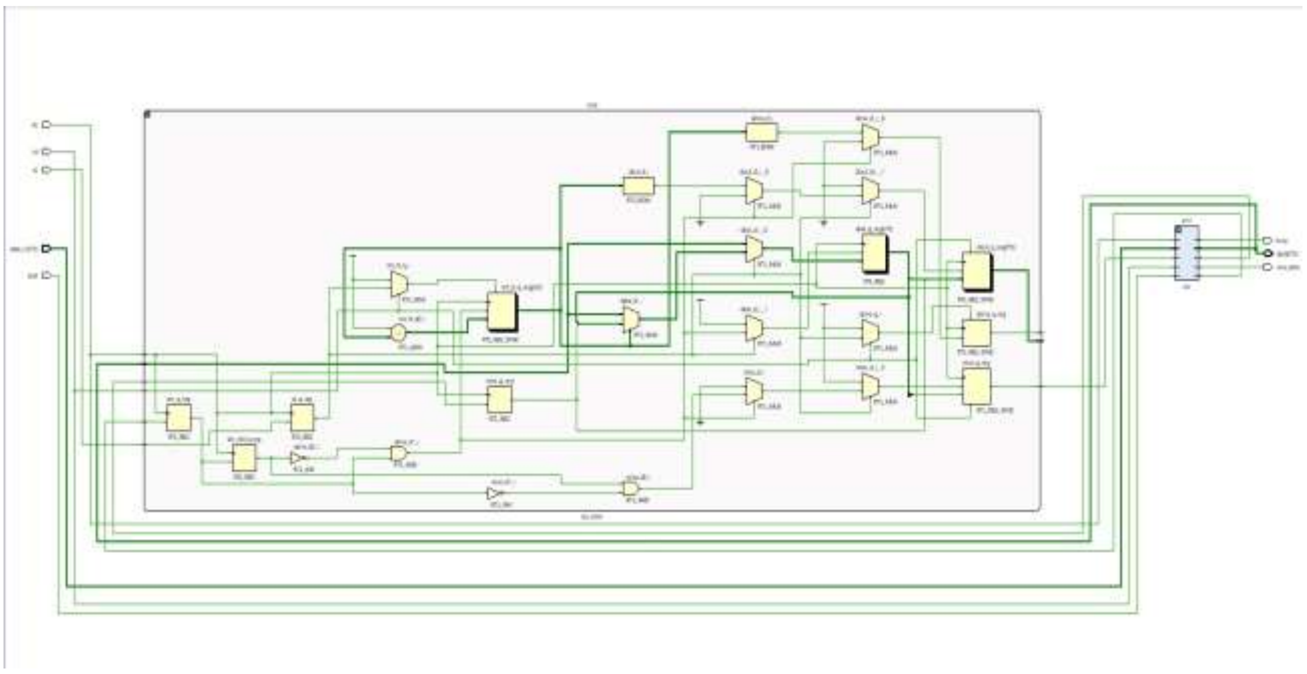
TOP MODULE:



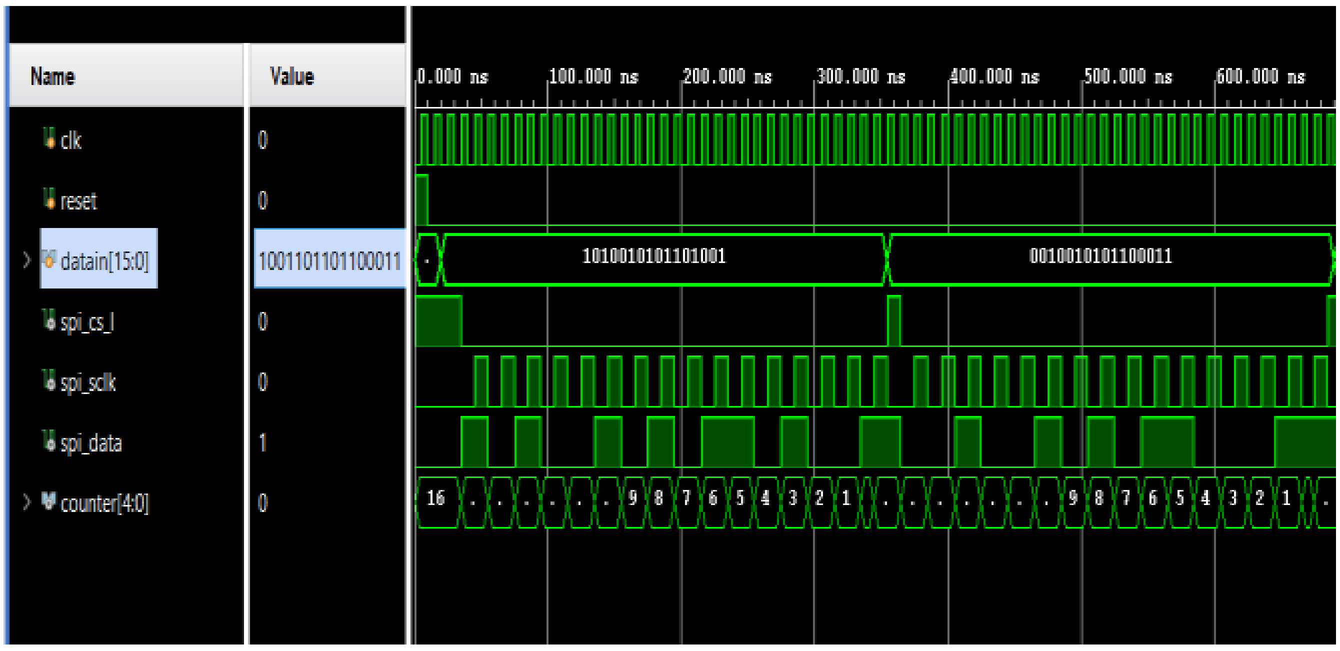
MASTER MODULE



SLAVE MODULE



SIMULATION



Name	Value
clk	0
reset	0
datain[15:0]	1001101101100011
spi_cs_l	0
spi_sclk	0
spi_data	1
counter[4:0]	0

APPLICATIONS

The SPI protocol with encrypted data transmission can be used in areas where sensitive signals have to be transmitted with high accuracy, minimal hardware and high speed. Some of the application include:

Interfacing with Sensors: SPI is commonly used for interfacing with sensors such as accelerometers, gyroscopes, and temperature sensors. These sensors often require high-speed communication and low latency, making SPI an ideal choice.

Memory Devices: SPI is used for interfacing with memory devices such as EEPROM, Flash memory, and SRAM. The protocol's high-speed data transfer capabilities make it ideal for reading and writing large amounts of data.

Audio and Video Applications: SPI is used for audio and video applications, such as controlling audio codecs, LCD displays, and graphics controllers. The protocol's high-speed data transfer capabilities make it suitable for transferring large amounts of data in real-time.

Control Systems: SPI is used for control systems, such as motor control, where high-speed communication is required between the microcontroller and motor driver.

Networking and Communication: SPI is used in networking and communication applications, such as Ethernet controllers and wireless transceivers. The protocol's high-speed data transfer capabilities make it ideal for transferring large amounts of data quickly and efficiently.

FUTURE SCOPE

The data transferred can be encrypted with secure encryption algorithms like the AES.

Differentiating the slave and master modules and implementing one on each FPGA to establish practical connection between two digital devices through the SPI communication protocol.

Expanding the architecture of the master to cater for multiple slaves and multiple modes of transmission.

CONCLUSION

The project ‘Implementation of SPI protocol data transmission’ aims at designing the digital system possessed by a master and a slave present in a network connected with the SPI protocol. With the externals controls like the system clock, data for transmission and trigger to indicate the master to start the transmission being given by the user through the pins on the FPGA board.