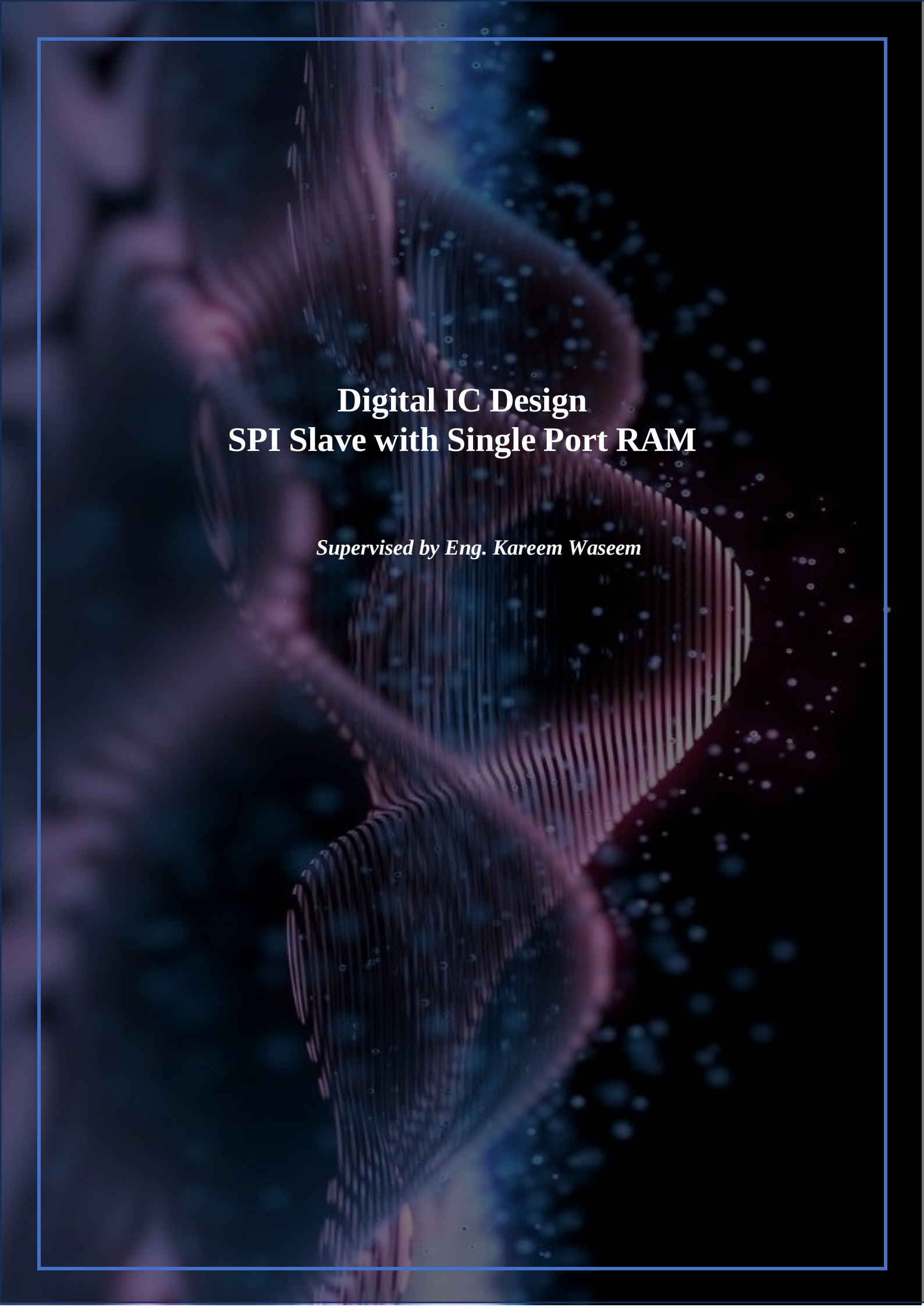
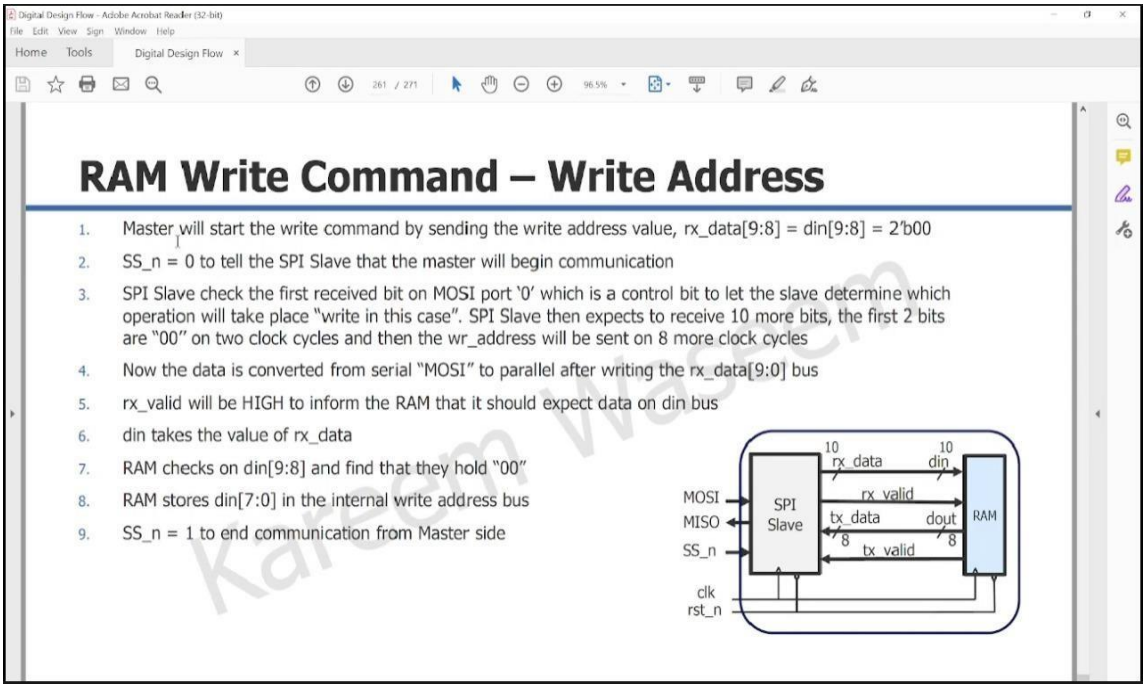
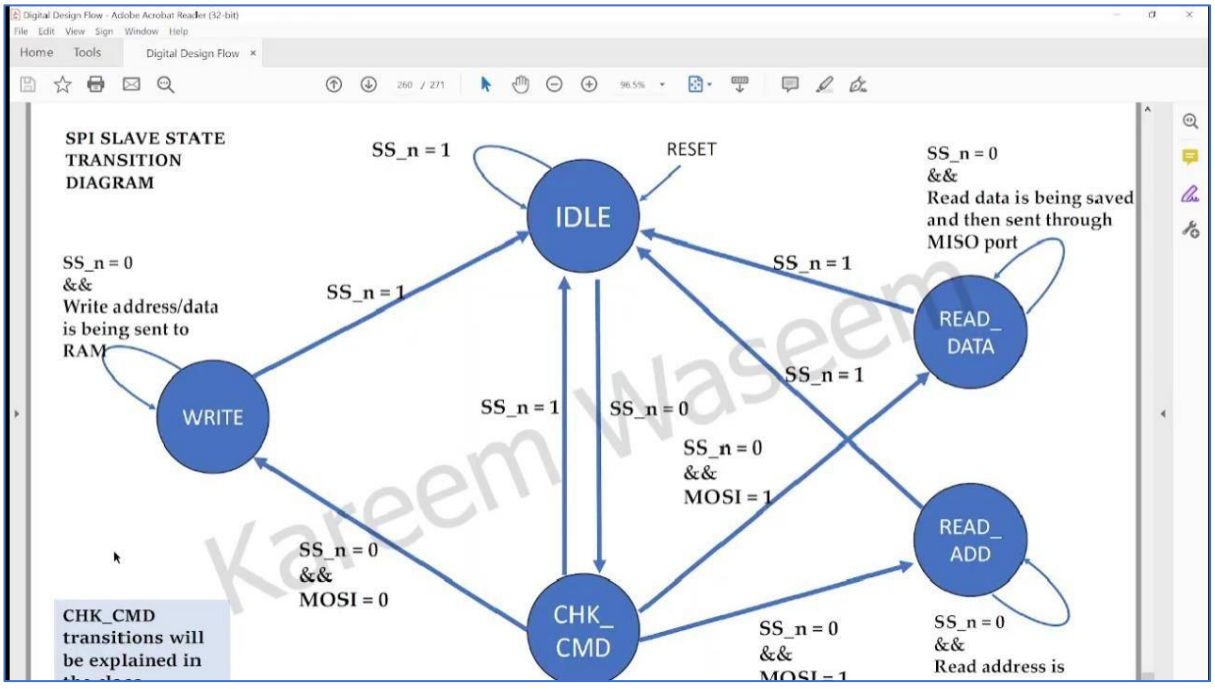


Digital IC Design
SPI Slave with Single Port RAM

Supervised by Eng. Kareem Waseem

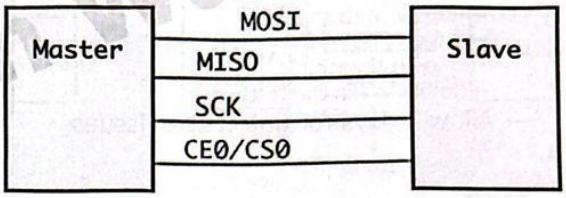
1.0 QUESTION


Digital Design Flow - Adobe Acrobat Reader (32-bit)
Home
Tools
Digital Design Flow
RAM Write Command – Write Address
1. Master will start the write command by sending the write address value, rx_data[9:8] = din[9:8] = 2'b00
2. SS_n = 0 to tell the SPI Slave that the master will begin communication
3. SPI Slave check the first received bit on MOSI port '0' which is a control bit to let the slave determine which operation will take place "write in this case". SPI Slave then expects to receive 10 more bits, the first 2 bits are "00" on two clock cycles and then the wr_address will be sent on 8 more clock cycles
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus
5. rx_valid will be HIGH to inform the RAM that it should expect data on din bus
6. din takes the value of rx_data
7. RAM checks on din[9:8] and find that they hold "00"
8. RAM stores din[7:0] in the internal write address bus
9. SS_n = 1 to end communication from Master side
MOSI
MISO
SS_n
clk
rst_n
SPI Slave
10
rx_data
din
rx_valid
tx_data
dout
8
tx_valid
RAM


Digital Design Flow - Adobe Acrobat Reader (32-bit)
Home
Tools
Digital Design Flow
SPI SLAVE STATE TRANSITION DIAGRAM
SS_n = 1
RESET
IDLE
SS_n = 0 && Read data is being saved and then sent through MISO port
SS_n = 0 && Write address/data is being sent to RAM
SS_n = 1
SS_n = 1
READ_ DATA
WRITE
SS_n = 1
SS_n = 0
SS_n = 0 && MOSI = 1
SS_n = 0 && MOSI = 0
CHK_ CMD
READ_ ADD
CHK_CMD transitions will be explained in
SS_n = 0 && MOSI = 1
SS_n = 0 && Read address is

SPI Interface

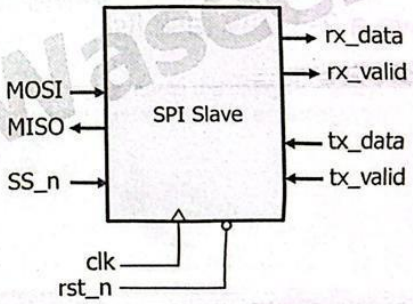
- One of the most popular Interfaces nowadays
- Stands for Serial-Peripheral Interface
- Four Wires
  - MOSI: Master-Out-Slave-In
  - MISO: Master-In-Slave-Out
  - SCK: Clock
  - SS_n: Slave Select
- High Data Rates



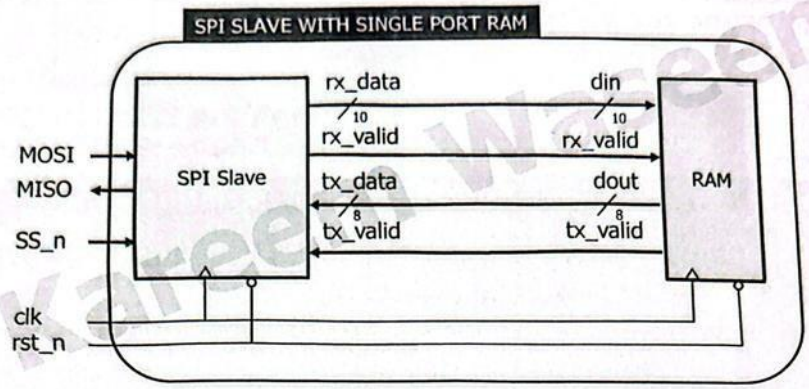
299

Project: 1- SPI Slave Interface

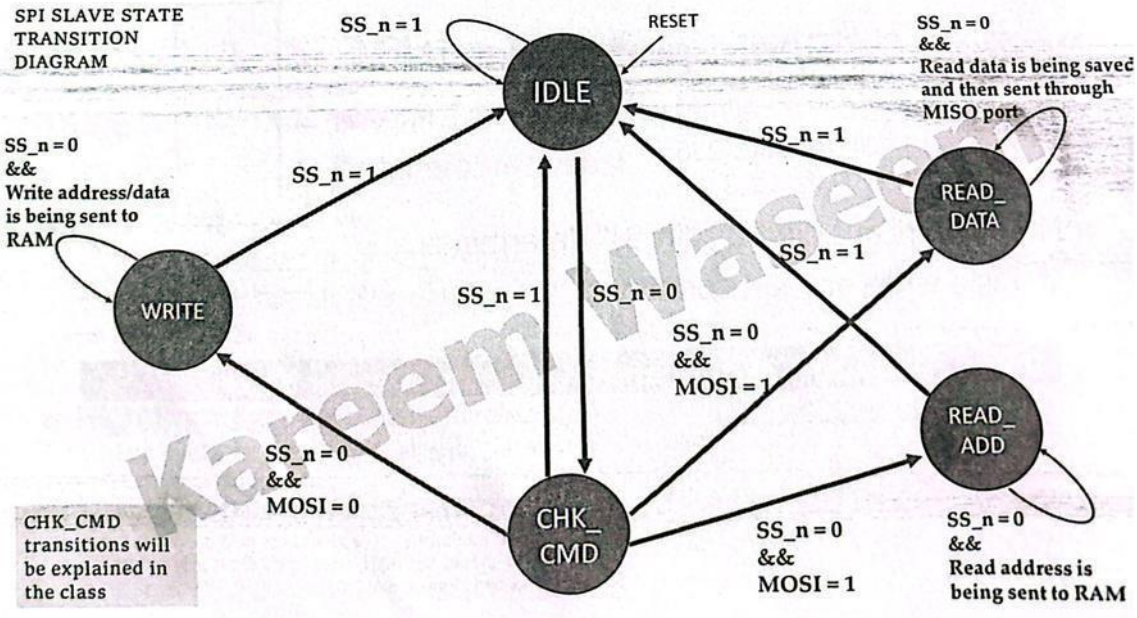
- One of the most popular Interfaces nowadays
- Stands for Serial-Peripheral Interface
- Four Wires
  - MOSI: Master-Out-Slave-In
  - MISO: Master-In-Slave-Out
  - SCK: Clock
  - SS_n: Slave Select
- High Data Rates



Project: 3- SPI Wrapper



303



2.0 CODE

2.1 Ram

```
module RAM_MODULE (clk,rst_n,rx_valid,din,dout,tx_valid);

 parameter MEM_DEPTH = 256;
 parameter ADDR_SIZE = 8;

 input clk;
 input rst_n;
 input rx_valid;
 input [9:0] din;

 output reg [7:0] dout;
 output reg tx_valid;

reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
reg [7:0] addr_wr;
reg [7:0] addr_re;


always @(posedge clk) begin
    if (!rst_n) begin
        dout <= 8'b0;
        tx_valid <= 1'b0;
        addr_wr <= 8'b0;
        addr_re <= 8'b0;
    end else begin
        if (rx_valid) begin
            case (din[9:8])
                2'b00: begin
                    addr_wr <= din[7:0];
                    tx_valid <= 1'b0;
                end
                2'b01: begin
                    mem[addr_wr] <= din[7:0];
                    tx_valid <= 1'b0;
                end
                2'b10: begin
                    addr_re <= din[7:0];
                    tx_valid <= 1'b0;
                end
                2'b11: begin
                    dout <= mem[addr_re];
                    tx_valid <= 1'b1;
                end
            endcase
        end
    end
end
endmodule
```

2.2 SPI Module

2.2.1 Definition

```
module SPI_MODULE(clk,rst_n,SS_n,MOSI,tx_valid,tx_data,MISO,rx_valid,rx_data);
    input MOSI , tx_valid , clk , rst_n , SS_n ;
    input [7:0] tx_data ;
    output reg  MISO , rx_valid ;
    output reg [9:0] rx_data;
    ////////////////////////////////////////////
parameter IDLE = 3'b000;
parameter  CHK_CMD = 3'b001;
parameter  WRITE = 3'b010;
parameter  READ_ADD = 3'b011;
parameter  READ_DATA = 3'b100;

reg [2:0] cs , ns ;
reg ADD_DATA_checker ;
reg [3:0] counter1 ;
reg [2:0] counter2 ;
reg [9:0] bus ;
```

2.2.2 State Memory

```
always @(posedge clk)
begin
    if(~rst_n)
        cs <= IDLE;
    else
        cs <= ns ;
end
```

2.2.3 Output Logic

```
always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
        counter1 <= 9;
        counter2 <= 7;
        ADD_DATA_checker <= 1;
        bus <= 0;
        rx_data <= 0;
        rx_valid <= 0;
        MISO  <= 0;
    end

    else begin
        if(cs == IDLE) begin
            rx_valid <= 0;
            counter1 <= 9 ;
            counter2 <= 7 ;
        end

        else if(cs == WRITE) begin
            if (counter1 >= 0)begin
                bus[counter1] <= MOSI;
                counter1 <= counter1 - 1;
            end
            if(counter1 == 4'b1111) begin
                rx_valid = 1;
                rx_data <= bus ;
            end
        end

        else if (cs == READ_ADD) begin
            if (counter1 >= 0)begin
                bus[counter1] <= MOSI;
                counter1 <= counter1 - 1;
            end
            if(counter1 == 4'b1111) begin
                rx_valid <= 1;
                rx_data <= bus ;
                ADD_DATA_checker <= 0;
            end
        end

        else if (cs == READ_DATA) begin
            if (counter1 >= 0)begin
                bus[counter1] <= MOSI;
                counter1 <= counter1 - 1;
            end
            if(counter1 == 4'b1111) begin
                rx_valid <= 1;
                rx_data <= bus ;
                counter1 <= 9 ;
            end
            if(rx_valid  == 1) rx_valid <= 0;
            if(tx_valid==1 && counter2 >=0)begin
                MISO <= tx_data[counter2] ;
                counter2 <= counter2 - 1 ;
            end
            if(counter2 == 3'b111)begin
                ADD_DATA_checker <= 1;
            end
        end
    end
end
```

2.2.4 Next State Logic

```
lways @(*) begin
    ns = cs ;
    case(cs)
        IDLE : begin
            if(SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
            if(SS_n)
                ns = IDLE;
            else begin
                if((~SS_n) && (MOSI == 0))
                    ns = WRITE;
                else if ((~SS_n) && (MOSI == 1) && (ADD_DATA_checker == 1))
                    ns = READ_ADD;
                else if ((~SS_n) && (MOSI == 1) && (ADD_DATA_checker == 0))
                    ns = READ_DATA;
            end
        end
        WRITE : begin
            if(SS_n || counter1 == 4'b1111)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if(SS_n || counter1 == 4'b1111)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA : begin
            if(SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
    endcase
end

endmodule
```

2.3 SPI Wrapper

```
module SPI_Wrapper (clk,ss_n,MOSI,MISO,rst_n);
parameter MEM_DEPTH = 256 ;
parameter ADDR_SIZE = 8;
input clk , ss_n , MOSI , rst_n;
output MISO;
wire [9:0] rxdata ;
wire [7:0] txdata ;
wire rx_valid , tx_valid ;

SPI_MODULE SPI(.MOSI(MOSI),.MISO(MISO),.clk(clk),.SS_n(ss_n),.rst_n(rst_n)
,.rx_data(rxdata),.tx_data(txdata),.rx_valid(rx_valid),.tx_valid(tx_valid));

RAM_MODULE #(.MEM_DEPTH(MEM_DEPTH),.ADDR_SIZE(ADDR_SIZE)) RAM (.din(rxdata),.dout(txdata),.clk(clk)
,.rx_valid(rx_valid),.tx_valid(tx_valid),.rst_n(rst_n));
endmodule

```

2.4 Testbench

2.4.1 Definition

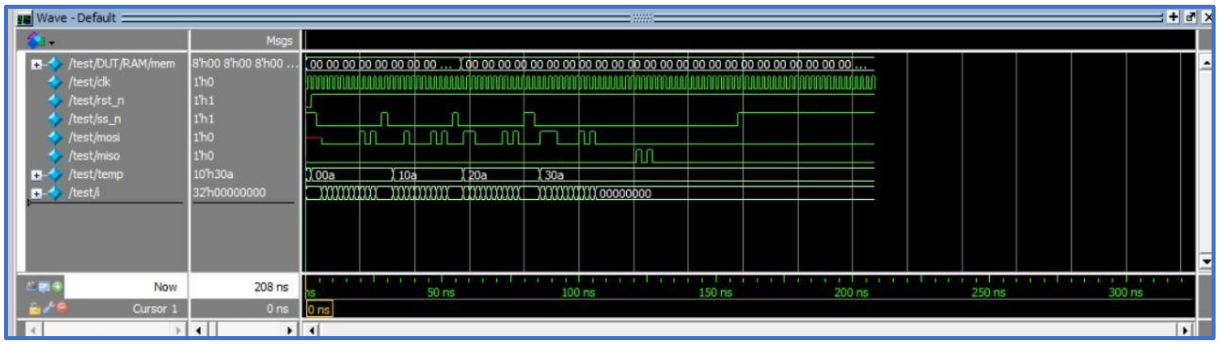
```
module test();
reg clk,rst_n,ss_n,mosi;
wire miso;
SPI_Wrapper  DUT(.clk(clk),.rst_n(rst_n),.MOSI(mosi),.MISO(miso),.ss_n(ss_n));
reg [9:0]temp;
integer i=0;

initial begin
clk=0;
forever
#1 clk=~clk;
end

```

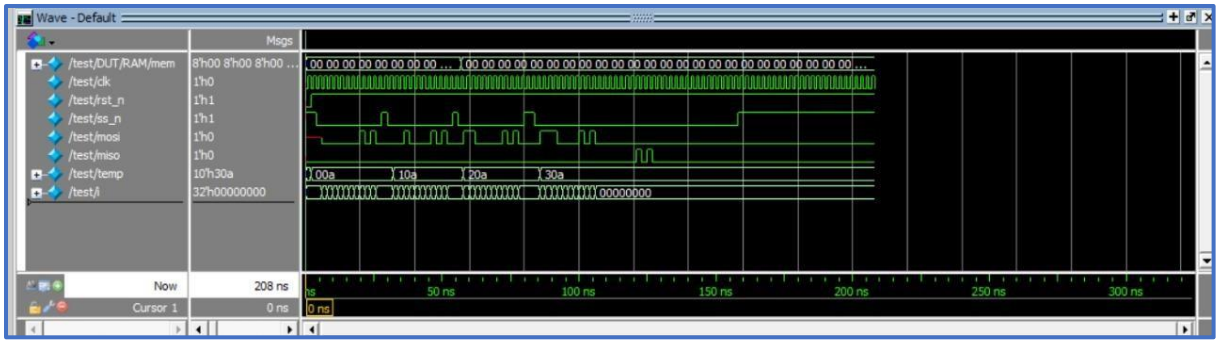
2.4.2 Main stimulus process

```
initial begin
$readmemh("mem.dat",DUT.RAM.mem);
rst_n=0;
ss_n=1;
temp=0;
@(negedge clk);
temp=10'b0000_01010;
rst_n=1;


@(negedge clk)begin
ss_n=0;
end
@(negedge clk)begin
mosi=0;
end
for(i=10;i>0;i=i-1)begin
@(negedge clk)
    mosi=temp[i-1];
end

@(negedge clk)
ss_n=1;

@(negedge clk)
ss_n=0;
@(negedge clk)begin
mosi=0;

temp=10'b01000_01010;
end
```

```
for(i=10;i>0;i=i-1)begin
@(negedge clk)
    mosi=temp[i-1];
end
@(negedge clk)
ss_n=1;

@(negedge clk)
ss_n=0;
@(negedge clk)begin
mosi=1;

temp=10'b10000_01010;
end
for(i=10;i>0;i=i-1)begin
@(negedge clk)
    mosi=temp[i-1];
end
@(negedge clk)
ss_n=1;
@(negedge clk)
ss_n=1;

@(negedge clk)
ss_n=0;
@(negedge clk)begin
mosi=1;

temp=10'b11000_01010;
end
for(i=10;i>0;i=i-1)begin
@(negedge clk)
    mosi=temp[i-1];
end

repeat(25)@(negedge clk);
@(negedge clk)
ss_n=1;
repeat(25)@(negedge clk);
$stop;
end

initial begin
    $monitor("mosi=%b,miso=%b,SS_n=%b",mosi,miso,ss_n);
end
endmodule
```

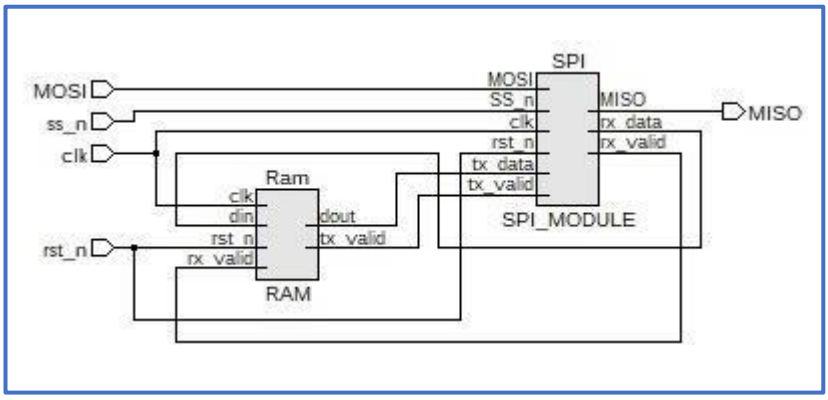
3.0 WAVEFORM





4.0 LINTING

4.1 Schematic



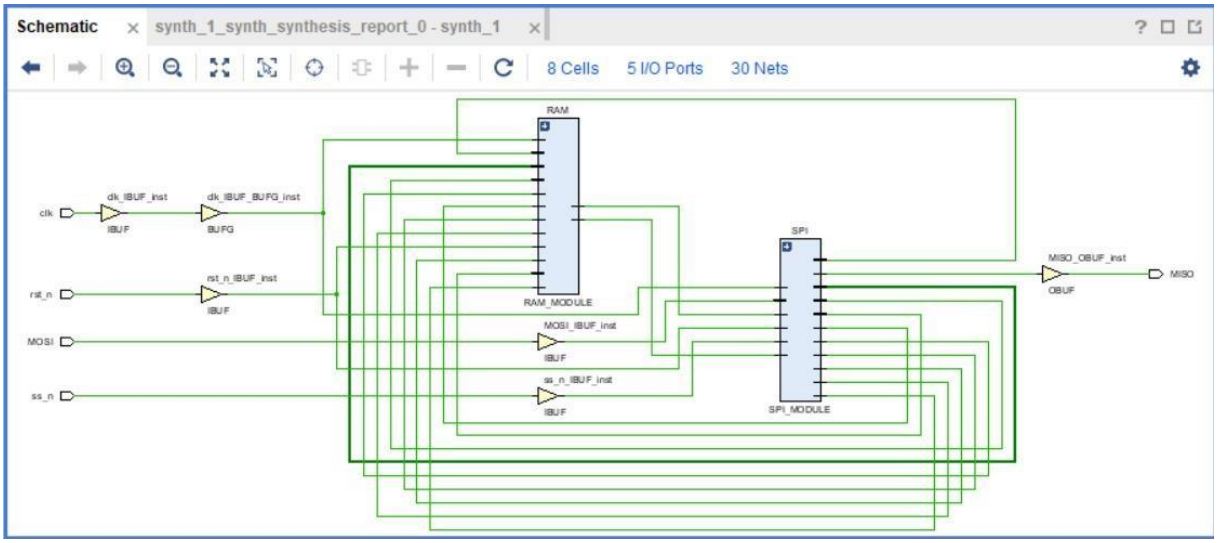
4.2 Errors

```
# End time: 22:26:24 on Jul 31,2025, Elapsed time: 0:00:00
#
# Errors: 0, Warnings: 0
#
```

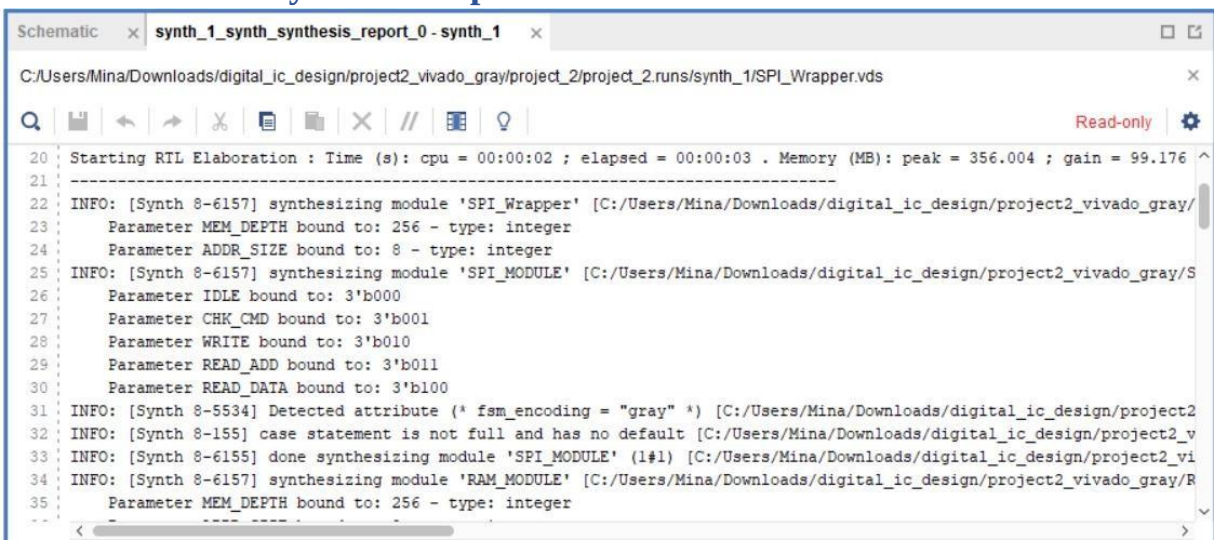
5.0 SYNTHESIS

5.1 Grey Coding

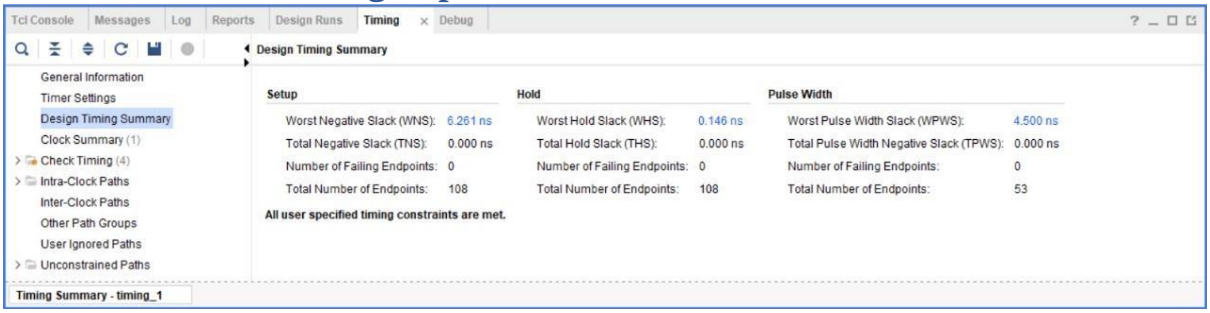
5.1.1 Schematic



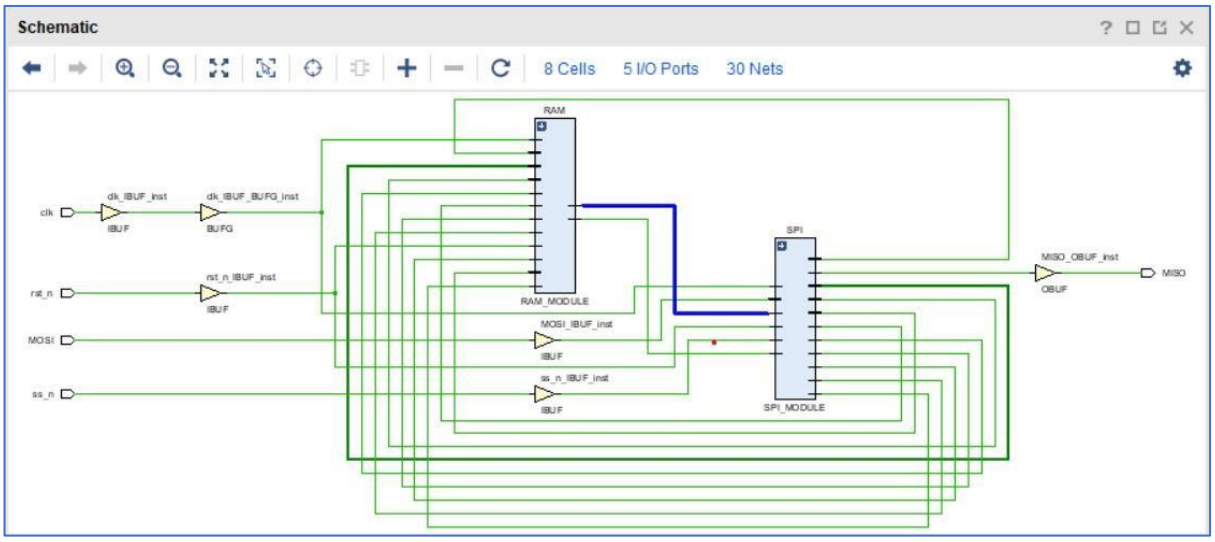
5.1.2 Synthesis Report



5.1.3 Timing Report



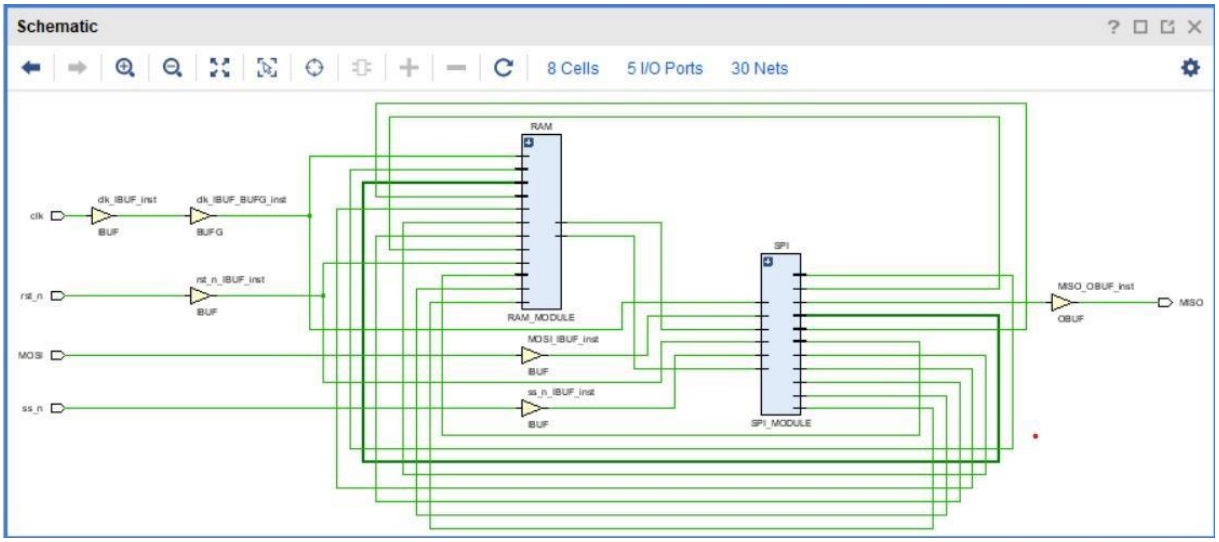
5.1.4 Critical Path



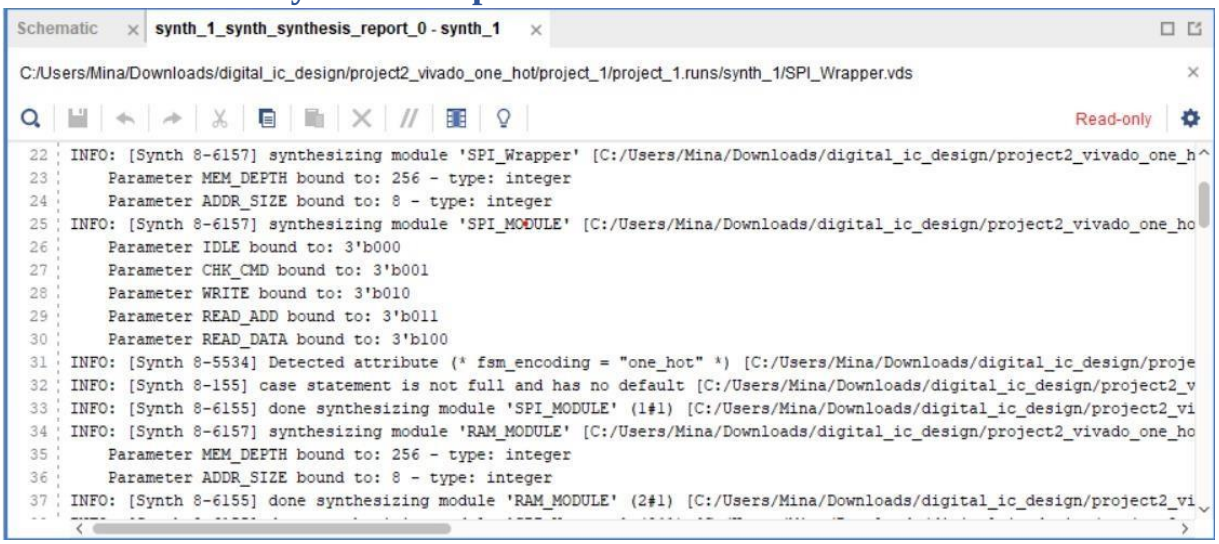
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source ...
Path 1	6.261	2	3	1	RAM/mem_reg/CLKBWRCLK	SPI/MISO_reg/D	3.623	2.823	0.800	10.0	sys_clk_pin

5.2 One Hot

5.2.1 Schematic

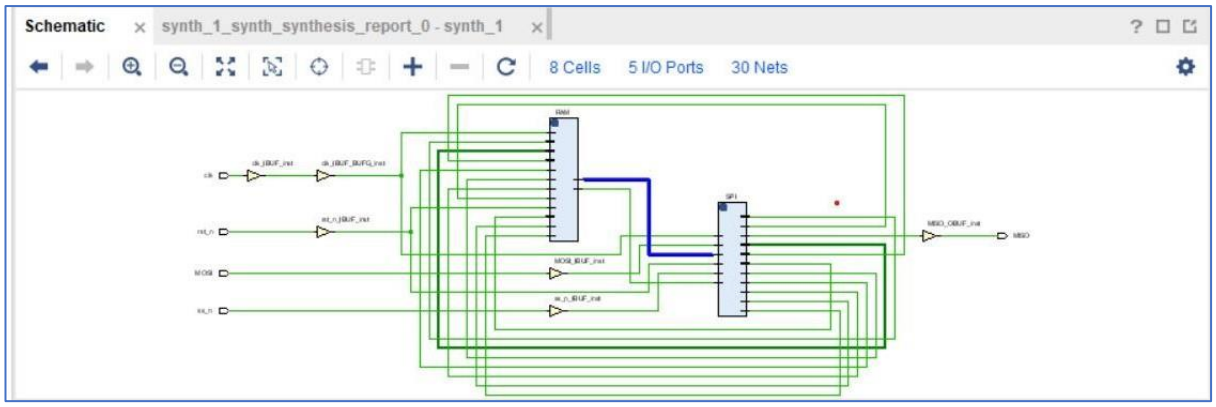


5.2.2 Synthesis Report



5.2.3 Timing Report

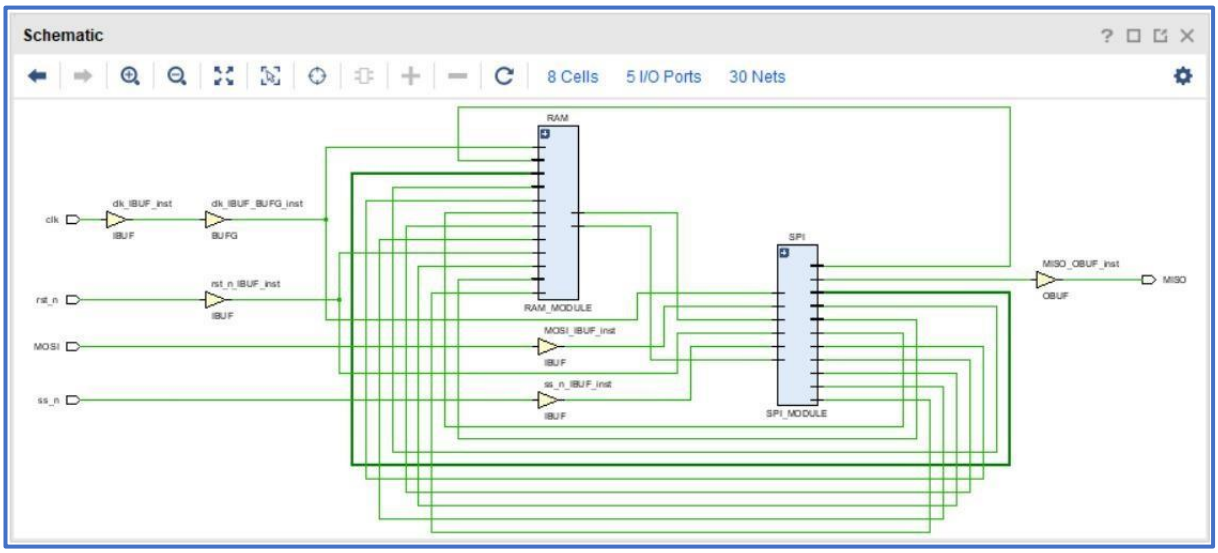
5.2.4 Critical Path



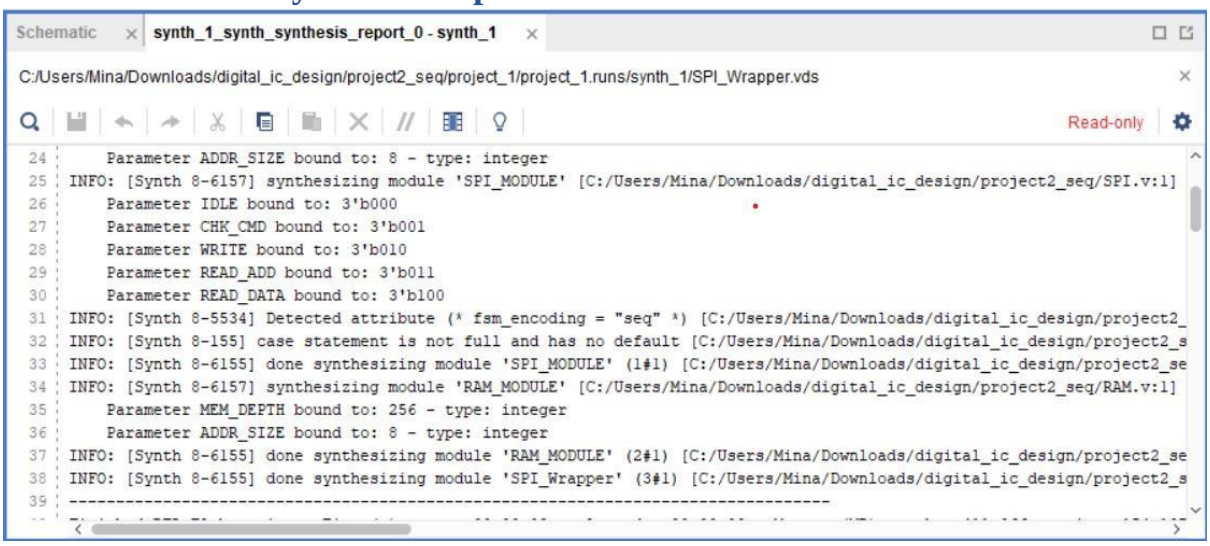
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	6.261	2	3	1	RAM/mem_reg/CLKBWRCLK	SPI/MISO_reg/D	3.623	2.823	0.800	10.0

5.3 Seq

5.3.1 Schematic

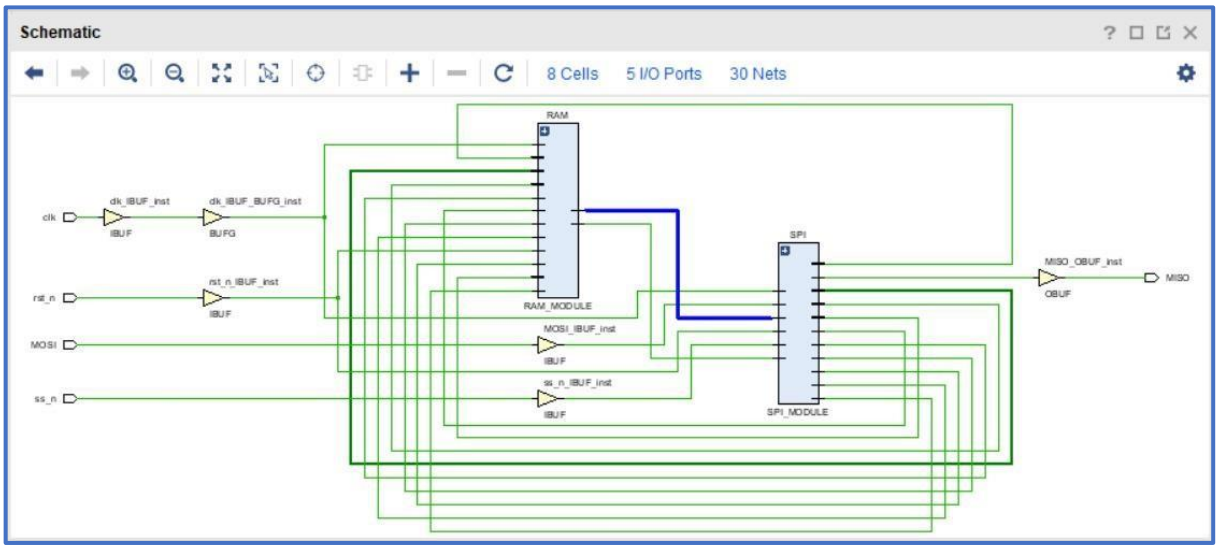


5.3.2 Synthesis Report



5.3.3 Timing Report

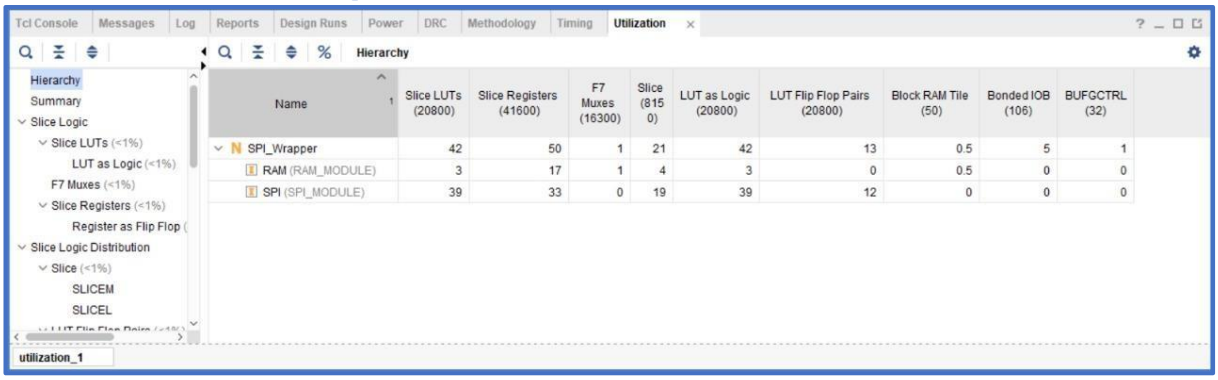
5.3.4 Critical Path



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	6.261	2	3	1	RAM/mem_reg/CLKBWRCLK	SPI/MISO_reg/D	3.623	2.823	0.800	10.0	sys_clk_pin	sys_clk_pin

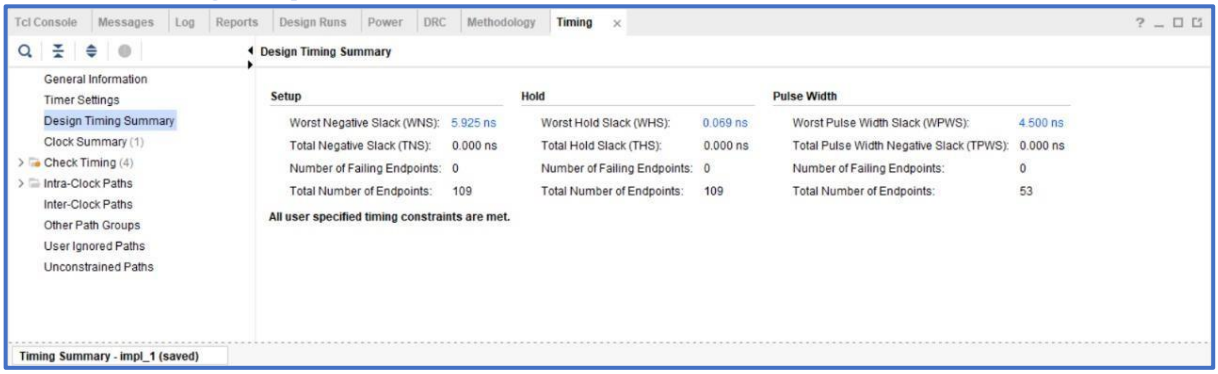
6.0 IMPLEMENTATION

6.1 Utilization Report

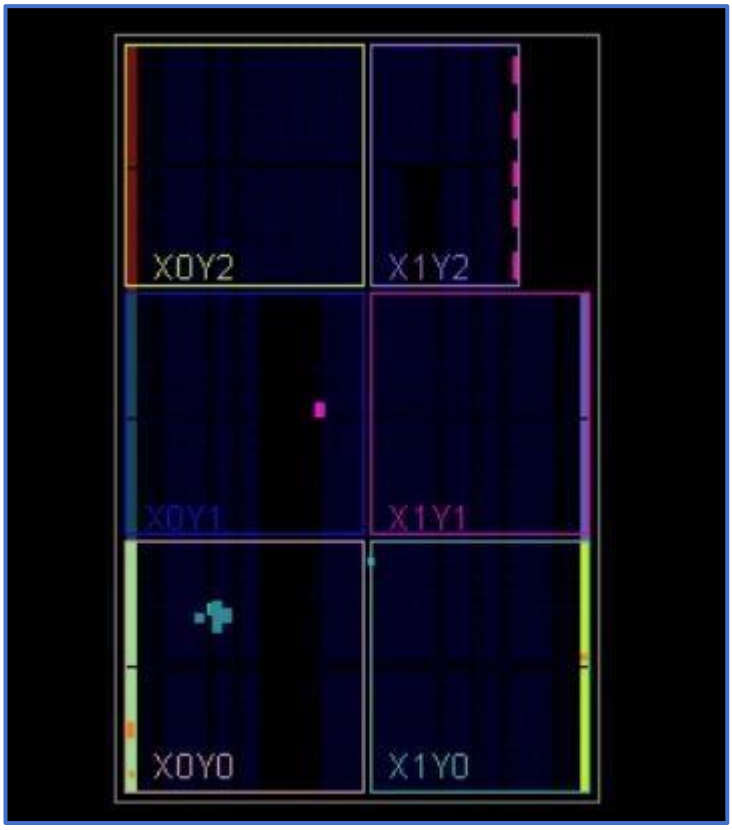


Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_Wrapper	42	50	1	21	42	13	0.5	5	1
RAM (RAM_MODULE)	3	17	1	4	3	0	0.5	0	0
SPI (SPI_MODULE)	39	33	0	19	39	12	0	0	0

6.2 Timing Report

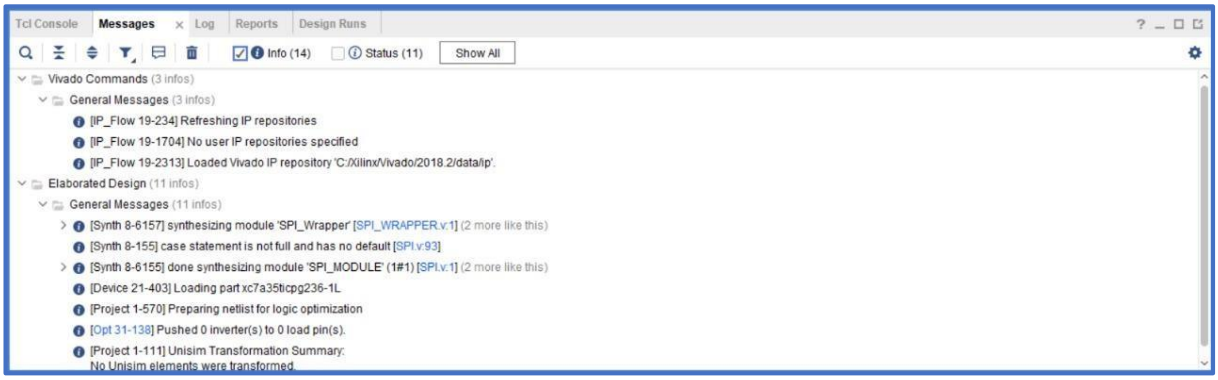


6.3 FPGA device snippet



7.0 CRITICAL WARNINGS

7.1 Elaboration



7.2 Synthesis

7.3 Implementation