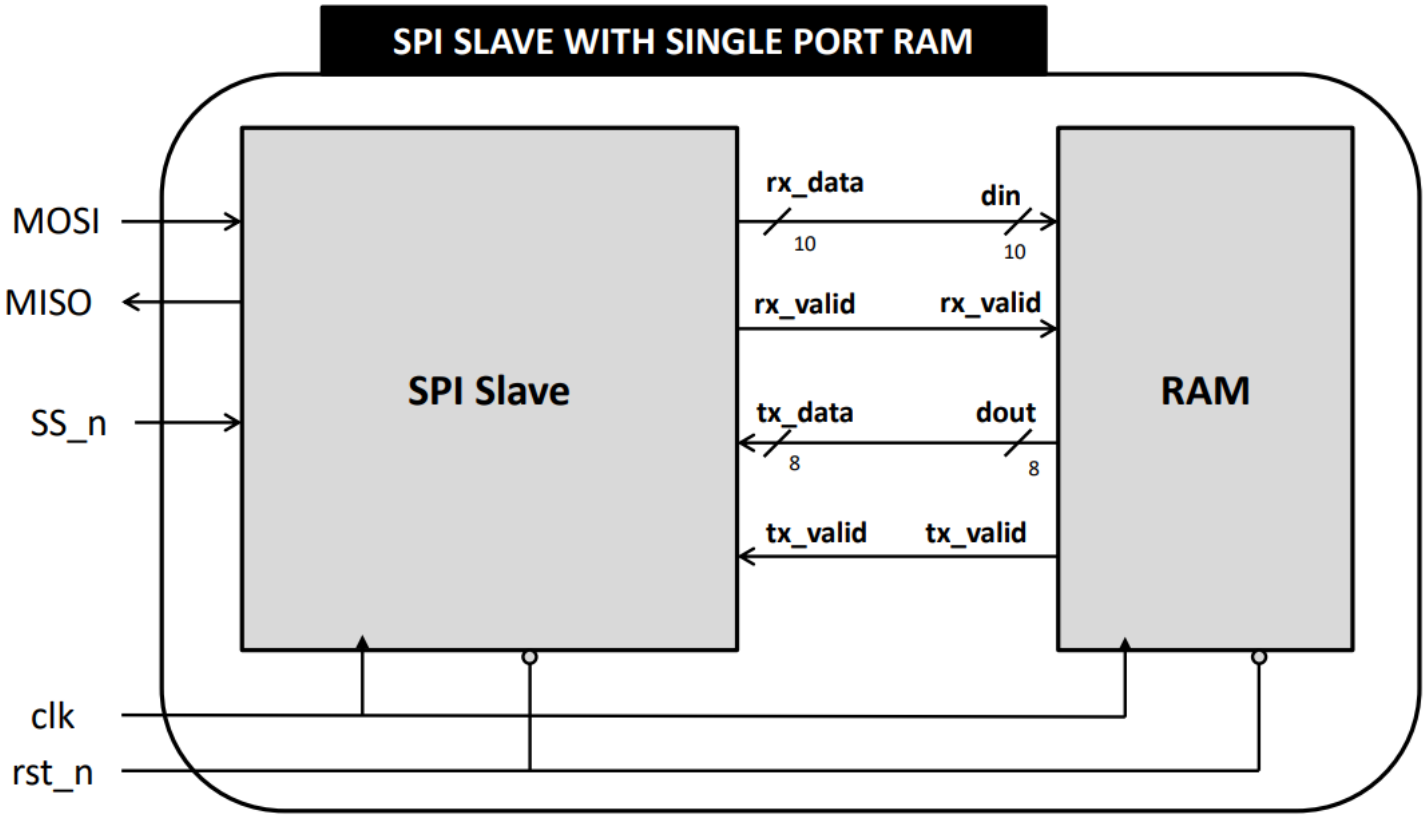
FINAL PROJECT

SPI SLAVE Interface

By: Khaled Ahmed Hamed

Under Supervision of Eng. Kareem Waseem



Introduction

The "SPI Slave with Single Port RAM" project implements an SPI slave module that communicates with a single-port RAM. The SPI slave receives data from a master device, storing and retrieving data to and from the RAM.

For more details, visit the GitHub Repo

Single Port Async. RAM

The single port asynchronous RAM module implements a memory block with a single data port.

It has the following parameters:

- MEM_DEPTH (default: 256): Depth of the memory.
- ADDR_SIZE (default: 8): Size of the memory address.

Ports:

Name	Type	Size	Description
clk	Input	1 bit	Clock signal
rst_n	Input	1 bit	Active low reset signal
din	Input	10 bit	Data input
rx_valid	Input	1 bit	If HIGH, accepts din[7:0] to save the write/read address internally or writes a memory word depending on the most significant 2 bits din[9:8]
dout	Output	8 bit	Data output
tx_valid	Output	1 bit	Whenever the command is a memory read, tx_valid should be HIGH

The most significant bits of din (din[9:8]) determine the operation to be performed:

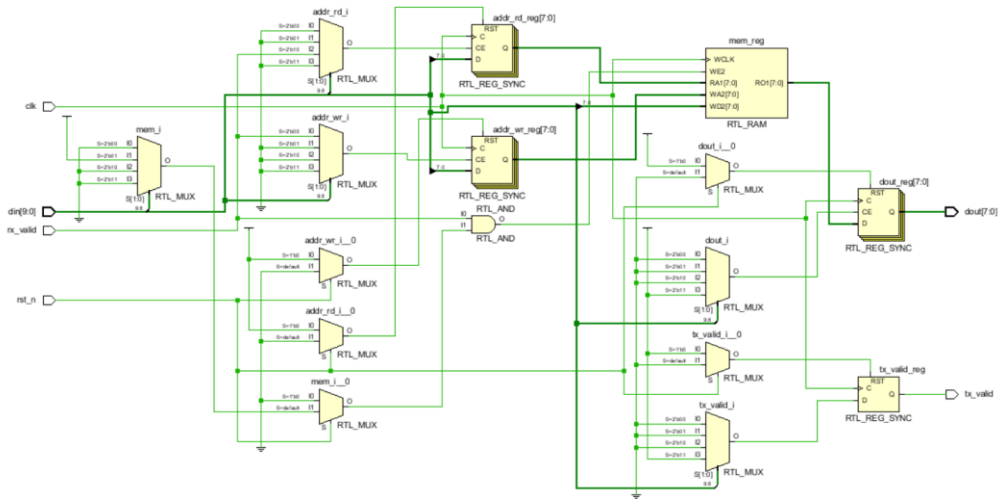
Port	din[9:8]	Command	Description
din	00	Write	Holds din[7:0] internally as a write address
din	01	Write	Writes din[7:0] to the memory with the write address held previously
din	10	Read	Holds din[7:0] internally as a read address
din	11	Read	Reads the memory with the read address held previously. tx_valid should be HIGH, and dout holds the word read from the memory. din[7:0] is ignored

RTL Code Snippet:

```
module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
//Ports
input [9:0] din ;
input clk,rst_n,rx_valid ;
output reg [7:0] dout ;
output reg tx_valid ;
//Parameters
parameter MEM_DEPTH = 256 ;
parameter ADDR_SIZE = 8 ;
//Addresses
reg [ADDR_SIZE-1:0] addr_rd , addr_wr ;
//Memory , Memory Width >> 8 Bits as ADDR_SIZE & din 10 bits & 1st 2 bits are for protocol only
reg [7:0] mem [MEM_DEPTH-1:0] ;
//RAM Functionality , Active Low Async. rst , but in vivado async. signals make problems in synthesis so its sync. here
always @(posedge clk) begin
    if (~rst_n) begin
        dout<=0;
        tx_valid<=0;
        addr_rd<=0;
        addr_wr<=0;
    end
    else begin
        case(din[9:8])
            2'b00: begin
                //Hold Write Address
                tx_valid<=0;
                if (rx_valid) begin
                    addr_wr<=din[7:0];
                end
            end
            2'b01: begin
                //Write in the memory with address held previously
                tx_valid<=0;
                if (rx_valid) begin
                    mem[addr_wr]<=din[7:0];
                end
            end
            2'b10: begin
                //Hold Read Address
                tx_valid<=0;
                if (rx_valid) begin
                    addr_rd<=din[7:0];
                end
            end
            2'b11: begin
                //Readt the memory with address held previously , din[7:0] Is dummy & Raise tx_valid high
                dout<=mem[addr_rd];
                tx_valid<=1;
            end
        endcase
    end
end
//tx_valid could be also calculated as : assign tx_valid = din[9] & din[8] , Out of the always block
endmodule
```

Elaboration:

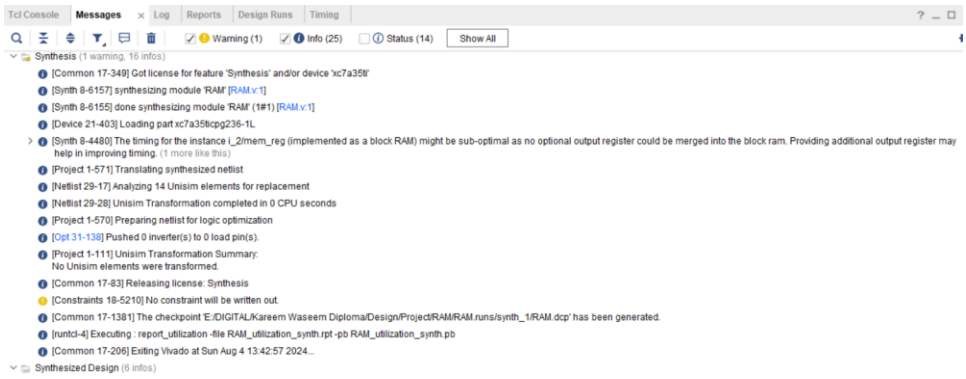
Schematic:



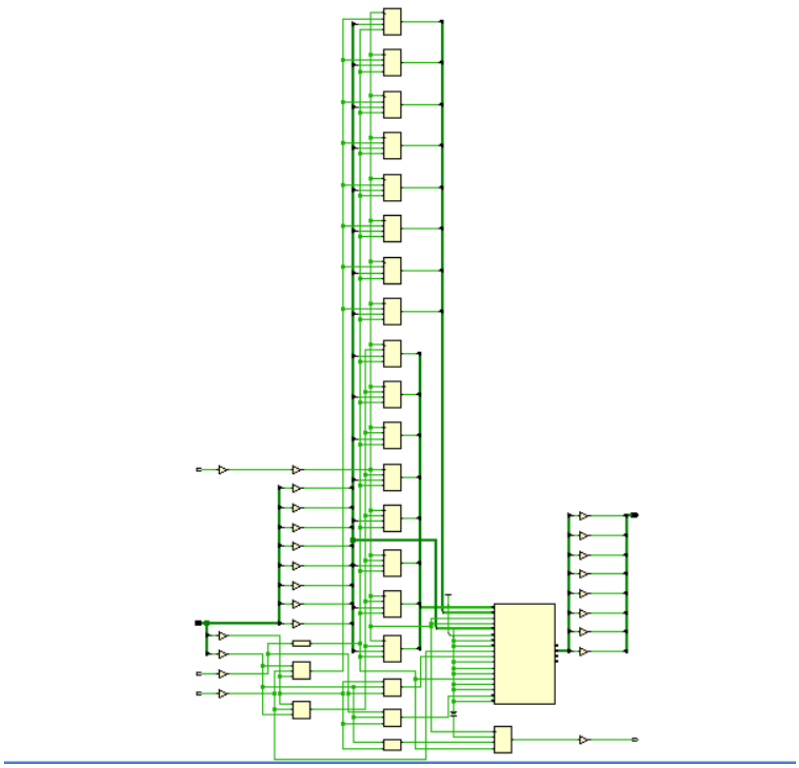
Messages Tab:

Synthesis:

Messages Tab:



Schematic:



Constraints File:

```
# Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst_n]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

Timing Report Summary:

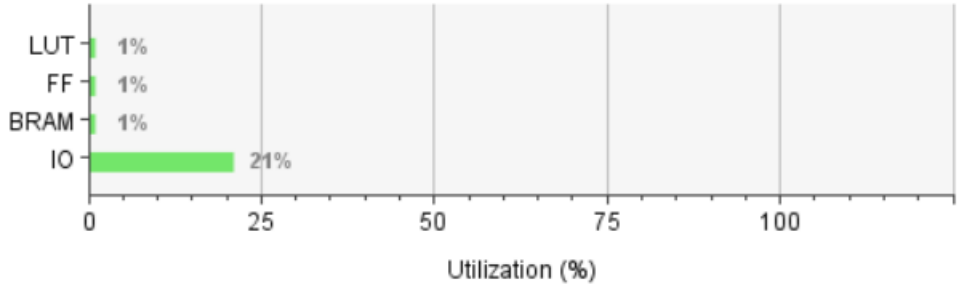
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.823 ns	Worst Hold Slack (WHS):	0.204 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	16	Total Number of Endpoints:	16	Total Number of Endpoints:	20

All user specified timing constraints are met.

Utilization Report:

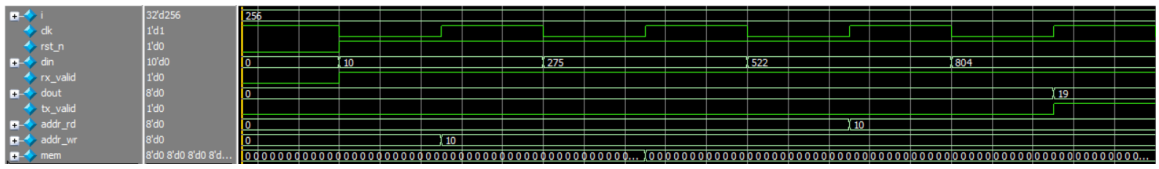
Resource	Utilization	Available	Utilization %
LUT	6	20800	0.03
FF	17	41600	0.04
BRAM	0.50	50	1.00
IO	22	106	20.75



Testbench Code Snippet:

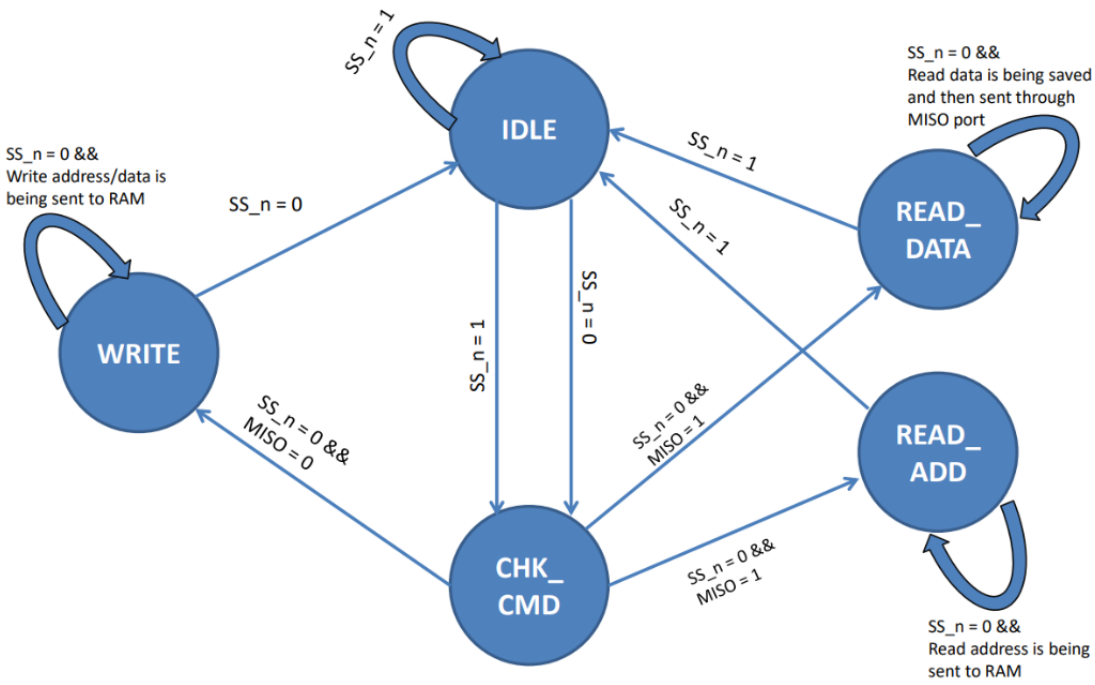
```
`timescale 1ns/1ps
module RAM_tb ();
//Signals Declaration
reg [9:0] din ;
reg clk,rst_n,rx_valid ;
wire [7:0] dout ;
wire tx_valid ;
integer i = 0;
//Clock Generation
initial begin
    clk=0;
    forever #5 clk=~clk;//Period=10ns
end
//DUT Instantiation
RAM #(.MEM_DEPTH(256),.ADDR_SIZE(8)) DUT (.din(din),.clk(clk),.rst_n(rst_n),.rx_valid(rx_valid),.dout(dout),.tx_valid(tx_valid));
//Test Stimiulus Generator
initial begin
    //Initializing Memory
    for(i=0;i<256;i=i+1)
        DUT.mem[i]=0;
    //Activate rst_n & Initialize all signals
    rst_n = 0;
    din=0;
    rx_valid=0;
    @(negedge clk);
    //De-Activate rst_n & Start Testing
    rst_n=1;
    //Test Holding din[7:0] as a Write Address
    rx_valid=1;
    din[9:8]=2'b00;
    din[7:0]=8'd10;//Memory Location Where data goes to
    @(negedge clk);
    //Test Writing in the memory with address held previously
    rx_valid=1;
    din[9:8]=2'b01;
    din[7:0]=8'd19;//Data which will bw stored in memory
    @(negedge clk);
    //Test Holding din[7:0] as a Read Address
    rx_valid=1;
    din[9:8]=2'b10;
    din[7:0]=8'd10;//Read data from this memory location
    @(negedge clk);
    //Test Writing in the memory with address held previously & Raising tx_valid high
                /*rx_valid Will not affect neither is 0 nor 1*/
                   /*Expected Output is 19 in location 10*/
    din[9:8]=2'b11;
    din[7:0]=$random;//Dummy Data
    @(negedge clk);
    $stop;
end
//Test Monitor & Results
initial begin
    $monitor("clk=%d,rst_n=%d,rx_valid=%d,din=%d,dout=%d,tx_valid=%d",clk,rst_n,rx_valid,din,dout,tx_valid);
end
endmodule
```

Simulation Waveform Snippet:



SPI SLAVE

SPI SLAVE State Transition Diagram



Introduction to SPI SLAVE Module:

The SPI slave module effectively manages data transfer between a master and a slave device. It uses state machines to control the flow of operations, ensuring proper read and write transactions based on the SPI protocol. The counters and flags aid in converting data formats and ensuring synchronization between the master and slave devices.

RTL Code Snippets:

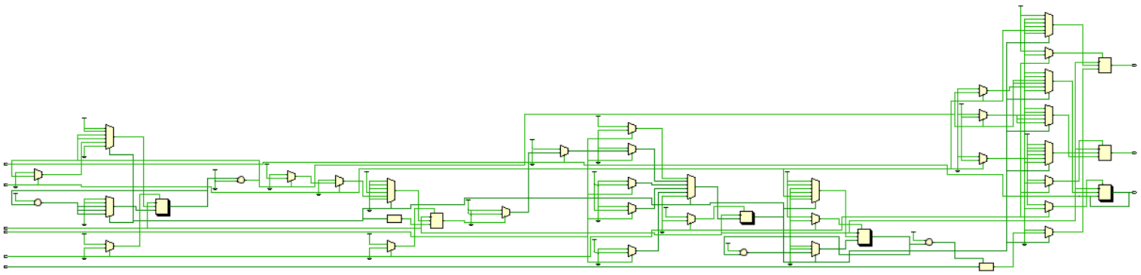
```
module SPI_Slave (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
//Ports
input MOSI,SS_n,clk,rst_n,tx_valid;
input [7:0] tx_data;
output reg [9:0] rx_data;
output reg rx_valid,MISO;
//Internal Signals
reg [3:0] serial_to_parallel_counter;//Data Converted from serial to parallel in 10 clock cycles
reg [2:0] parallel_to_serial_counter;//Data Converted from parallel to serial in 8 clock cycles
reg addr_read_rec;//Address Read Received >> Ensures that reading address is done before reading data
//CS & NS
(*fsm_encoding="sequential"*)
reg [2:0] cs ,ns;
//States
parameter IDLE = 3'b000;
parameter CHK_CMD = 3'b001;
parameter WRITE = 3'b010;
parameter READ_ADD = 3'b011;
parameter READ_DATA = 3'b100;
//Next State Logic (Comb)
always @(*) begin//always@(cs,MOSI,tx_data,tx_valid)
    case(cs)
        IDLE:begin
            if (~SS_n) begin
                ns=CHK_CMD;
            end
            else begin
                ns=IDLE;
            end
        end
        CHK_CMD:begin
            if (SS_n) begin
                ns=IDLE;
            end
            else begin
                if (~MOSI) begin
                    ns=WRITE;
                end
                else begin
                    if (~addr_read_rec) begin
                        ns=READ_ADD;
                    end
                    else begin
                        ns=READ_DATA;
                    end
                end
            end
        end
```

```
        WRITE:begin
            if (SS_n) begin
                ns=IDLE;
            end
            else begin
                ns=WRITE;
            end
        end
        READ_ADD:begin
            if (SS_n) begin
                ns=IDLE;
            end
            else begin
                ns=READ_ADD;
            end
        end
        READ_DATA:begin
            if (SS_n) begin
                ns=IDLE;
            end
            else begin
                ns=READ_DATA;
            end
        end
        default : ns = IDLE;
    endcase
end
//State Memory (Seq)
always @(posedge clk) begin
    if (~rst_n) begin
        cs<=IDLE;
    end
    else begin
        cs<=ns;
    end
end
//Output Logic (Seq)
always @(posedge clk) begin
    if (~rst_n) begin
        rx_data<=0;
        rx_valid<=0;
        MISO<=0;
        addr_read_rec<=0;
        serial_to_parallel_counter<=0;
        parallel_to_serial_counter<=0;
    end
    else begin
        case(cs)
            IDLE:begin
                rx_valid<=0;
                MISO<=0;
                serial_to_parallel_counter<=0;
                parallel_to_serial_counter<=0;
            end
```

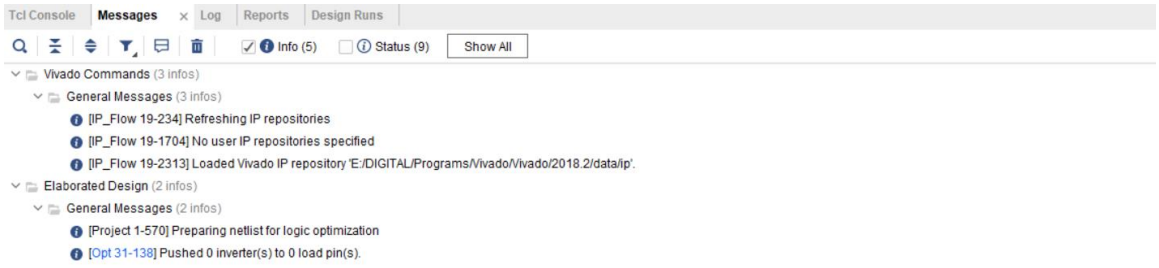
```
            CHK_CMD:
            begin
                rx_valid<=0;
                serial_to_parallel_counter<=0;
                parallel_to_serial_counter<=0;
            end
            WRITE:begin
                if (serial_to_parallel_counter<10) begin
                    rx_data <= {rx_data[8:0],MOSI};//Could be calculated as shift register or Using Up Counter : rx_data[9-serial_to_parallel_counter]<=MOSI;
                    serial_to_parallel_counter<=serial_to_parallel_counter+1;
                    rx_valid<=0;
                end
                else begin
                    rx_valid<=1;//Conversion is done
                end
            end
            READ_ADD:begin
                addr_read_rec<=1;
                if (serial_to_parallel_counter<10) begin
                    rx_data <= {rx_data[8:0],MOSI};//Could be calculated as shift register or Using Up Counter : rx_data[9-serial_to_parallel_counter]<=MOSI;
                    serial_to_parallel_counter<=serial_to_parallel_counter+1;
                    rx_valid<=0;
                end
                else begin
                    rx_valid<=1;//Conversion is done
                end
            end
            READ_DATA:begin
                if (tx_valid && parallel_to_serial_counter<8) begin
                    MISO <= tx_data[7 - parallel_to_serial_counter];
                    parallel_to_serial_counter <= parallel_to_serial_counter + 1;
                    /*Could be calculated as shift register or Using Up Counter :MISO <= tx_data[7];
                                                                                  tx_data <= tx_data << 1; */
                end
                else begin
                    if (serial_to_parallel_counter<10) begin
                        rx_data <= {rx_data[8:0],MOSI};//Could be calculated as shift register or Using Up Counter : rx_data[9-serial_to_parallel_counter]<=MOSI;
                        serial_to_parallel_counter<=serial_to_parallel_counter+1;
                        rx_valid<=0;
                    end
                    else begin
                        rx_valid<=1;//Conversion is done
                        addr_read_rec<=0;
                    end
                end
            end
        endcase
    end
end
endmodule
```

Elaboration:

Schematic:



Messages Tab:

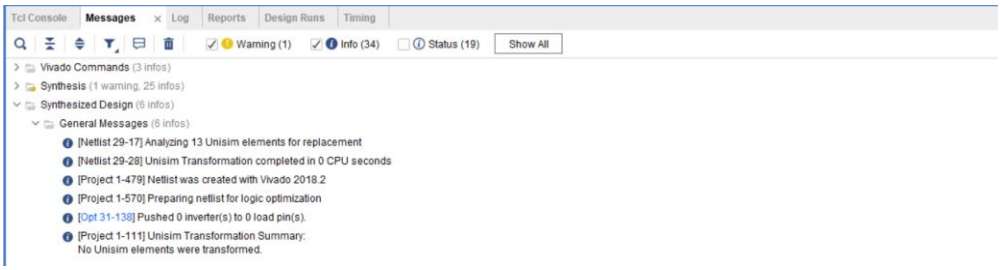


Synthesis:

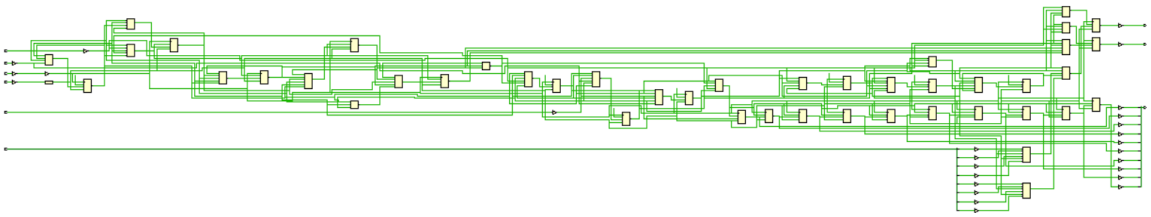
Encoding:

State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	001	001
WRITE	011	010
READ_ADD	010	011
READ_DATA	111	100

Messages Tab:



Schematic:



Timing Report Summary:

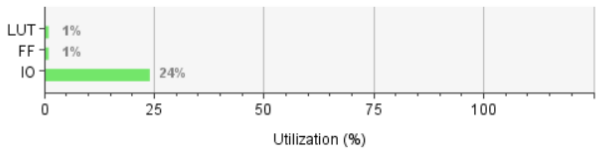
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.975 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	38	Total Number of Endpoints:	38	Total Number of Endpoints:	24

All user specified timing constraints are met.

Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	22	20800	0.11
FF	23	41600	0.06
IO	25	106	23.58

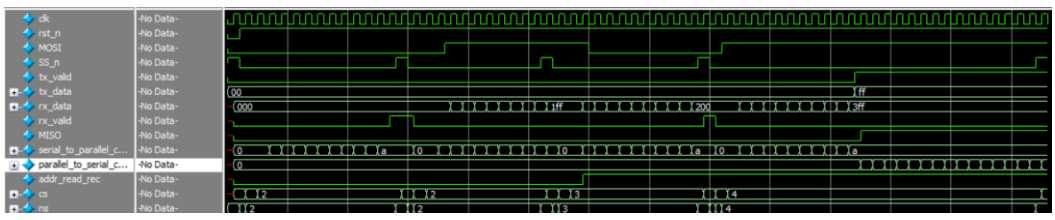


Testbench Code Snippets:

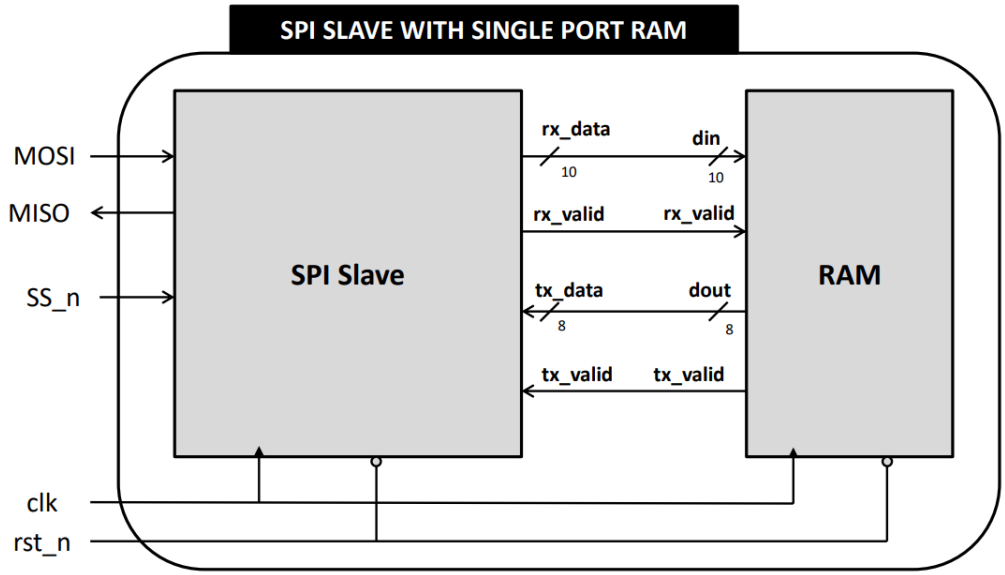
```
module SPI_Slave_tb();
//Signals Declaration
reg MOSI,SS_n,clk,rst_n,tx_valid;
reg [7:0] tx_data;
wire [9:0] rx_data;
wire rx_valid,MISO;
//Clock Generation
initial begin
    clk=0;
    forever
    #5 clk=~clk;//Period=10ns
end
//DUT Instantiation
SPI_Slave DUT (.MOSI(MOSI),.MISO(MISO),.SS_n(SS_n),.clk(clk),.rst_n(rst_n),.rx_data(rx_data),.rx_valid(rx_valid),.tx_data(tx_data),.tx_valid(tx_valid));
//Test Stimiulus Generator
initial begin
    //Activate Reset ,Un-Enable SS_n& Initialize all signals
    rst_n=0;
    MOSI=0;
    SS_n=1;
    tx_data=0;
    tx_valid=0;
    @(negedge clk);
    //De-Activate Reset ,Enable SS_n & Start testing
    rst_n=1;
    SS_n=0;
    @(negedge clk);
    //Test Writing Address
    SS_n=0;
    @(negedge clk);
    MOSI=0;//Write
    @(negedge clk);
    repeat(2) begin//2'b00 >> To write address
        MOSI=0;
        @(negedge clk);
    end
    repeat(8) begin//8'b00000000 >> Address
        MOSI=0;
        @(negedge clk);
    end
    SS_n=1;
    @(negedge clk);
```

```
    //Test Writing Data
    SS_n=0;
    @(negedge clk);
    MOSI=0;//Write
    @(negedge clk);
    MOSI=0;
    @(negedge clk);
    MOSI=1;//2'b01 >> To write Data
    repeat(8) begin//8'b11111111 >> Data
        MOSI=1;
        @(negedge clk);
    end
    SS_n=1;
    @(negedge clk);
    //Test Reading Address
    SS_n=0;
    @(negedge clk);
    MOSI=1;//Read
    @(negedge clk);
    MOSI=1;
    @(negedge clk);
    MOSI=0;//2'b10 >> To Read address
    @(negedge clk);
    repeat(8) begin//8'b00000000 >> Address
        MOSI=0;
        @(negedge clk);
    end
    SS_n=1;
    @(negedge clk);
    //Test Reading Data
    SS_n=0;
    @(negedge clk);
    MOSI=1;//Read
    @(negedge clk);
    repeat(2) begin//2'b11 >> To Read Data
        MOSI=1;
        @(negedge clk);
    end
    repeat(8) begin//8'b00000000 >> Dummy Data
        MOSI=1;
        @(negedge clk);
    end
    tx_valid=1;
    tx_data=8'b11111111;//Data to be transmitted to MISO
    repeat(15) @(negedge clk);//Wait To Read Data
    SS_n=1;
    @(negedge clk);
$stop;
end
//Test Monitor & Results
initial begin
    $monitor("MOSI = %b, MISO = %b, SS_n = %b, rx_data = %b, rx_valid = %b",MOSI, MISO, SS_n, rx_data, rx_valid);
end
endmodule
```

Simulation Waveform Snippet:



SPI Wrapper



Introduction to Wrapper:

The Wrapper module integrates the Slave and RAM modules to enable SPI communication with a memory block. The Slave manages the SPI protocol and data transfer with the master device, while the RAM module handles data storage and retrieval. This setup allows the master device to read from and write to the memory block via the SPI interface, with the Slave ensuring proper data transfer and the RAM maintaining the data integrity.

Main Wires between RAM & Slave

1. rx_data in the SPI slave module is connected to the din port in the RAM module.
2. rx_valid in the SPI slave module is connected to rx_valid in the RAM module.
3. dout in the RAM module is connected to tx_data in the SPI slave module.
4. tx_valid in the RAM module is connected to tx_valid in the SPI slave module.

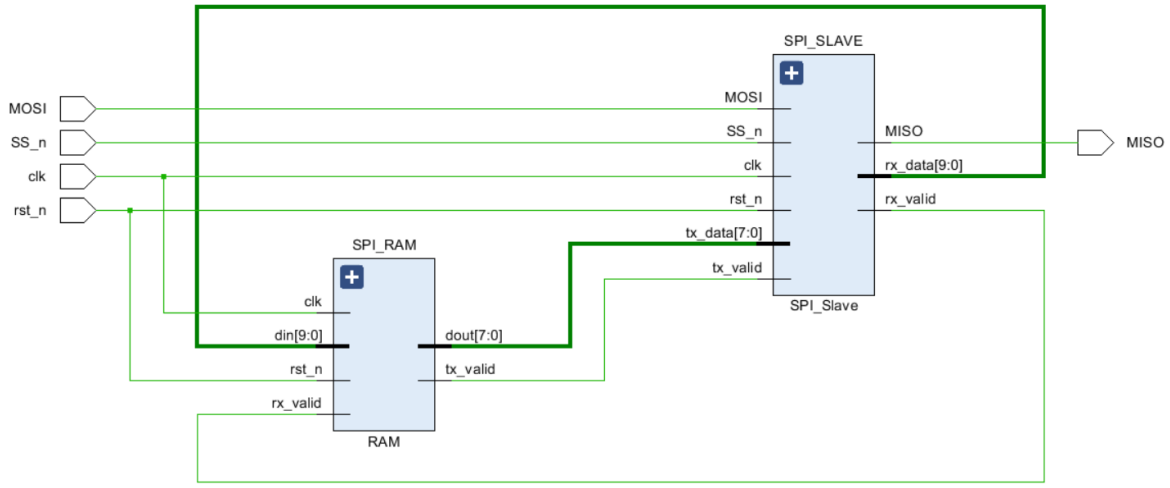
RTL Code Snippet:

```
module SPI_Wrapper(MOSI, MISO, SS_n, clk, rst_n);
// Ports
input MOSI, clk, rst_n, SS_n;
output MISO;

// Internal Signals
wire [9:0] rx_data;
wire rx_valid;
wire [7:0] tx_data;
wire tx_valid;

// Instantiations
SPI_Slave SPI_SLAVE (.MOSI(MOSI), .MISO(MISO), .SS_n(SS_n), .clk(clk), .rst_n(rst_n),.rx_data(rx_data),.rx_valid(rx_valid), .tx_valid(tx_valid), .tx_data(tx_data));
RAM #(.ADDR_SIZE(8), .MEM_DEPTH(256)) SPI_RAM (.din(rx_data), .rx_valid(rx_valid), .clk(clk),.rst_n(rst_n), .dout(tx_data),.tx_valid(tx_valid));
endmodule
```

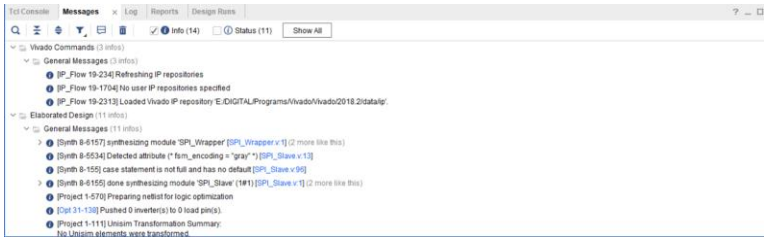
Gray Encoding

Elaboration:

Schematic:



Messages Tab:

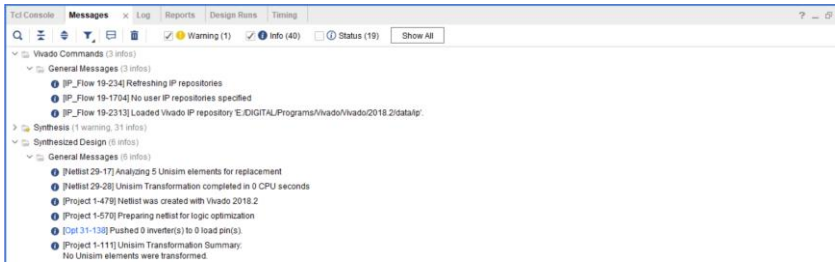


Synthesis:

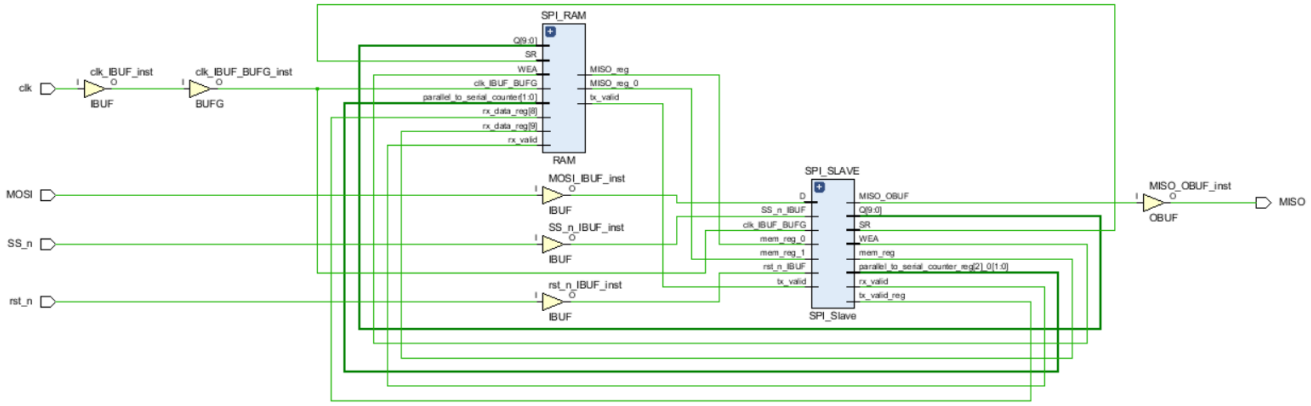
Encoding:

State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	001	001
WRITE	011	010
READ_ADD	010	011
READ_DATA	111	100

Messages Tab:



Schematic:



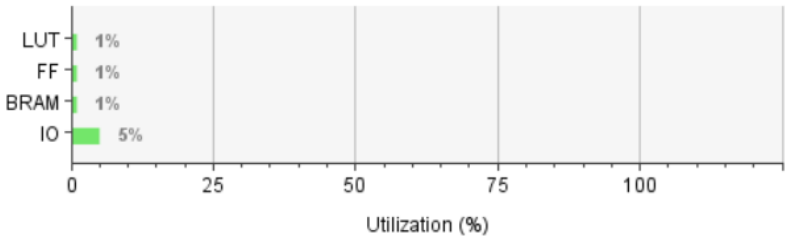
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.898 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	98	Total Number of Endpoints:	98	Total Number of Endpoints:	43

All user specified timing constraints are met.

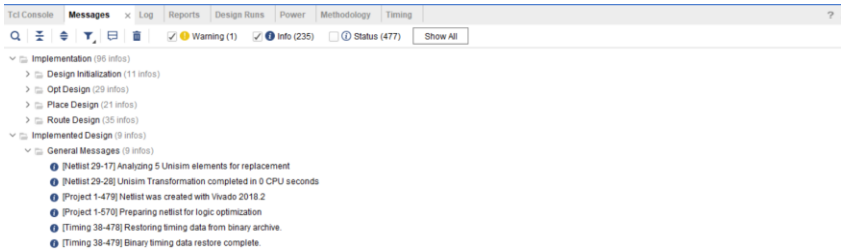
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	26	20800	0.13
FF	40	41600	0.10
BRAM	0.50	50	1.00
IO	5	106	4.72

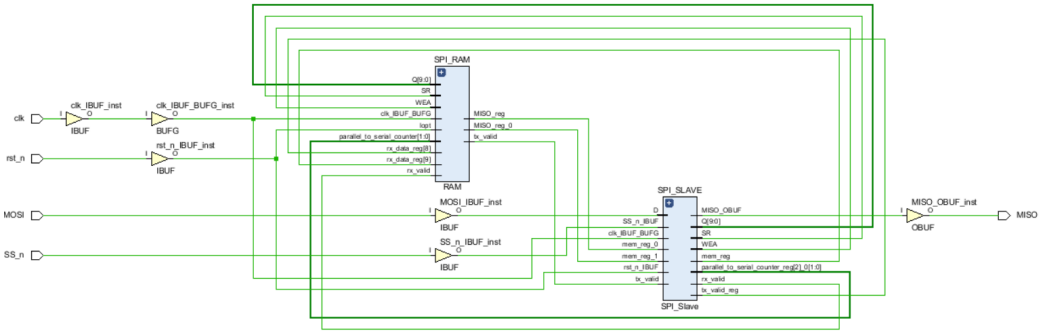


Implementation:

Messages Tab:



Schematic:



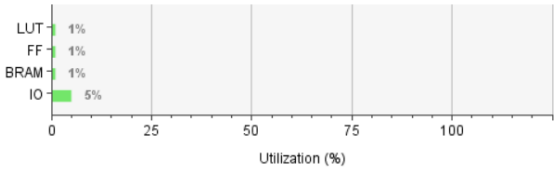
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.375 ns	Worst Hold Slack (WHS):	0.101 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	98	Total Number of Endpoints:	98	Total Number of Endpoints:	43

All user specified timing constraints are met.

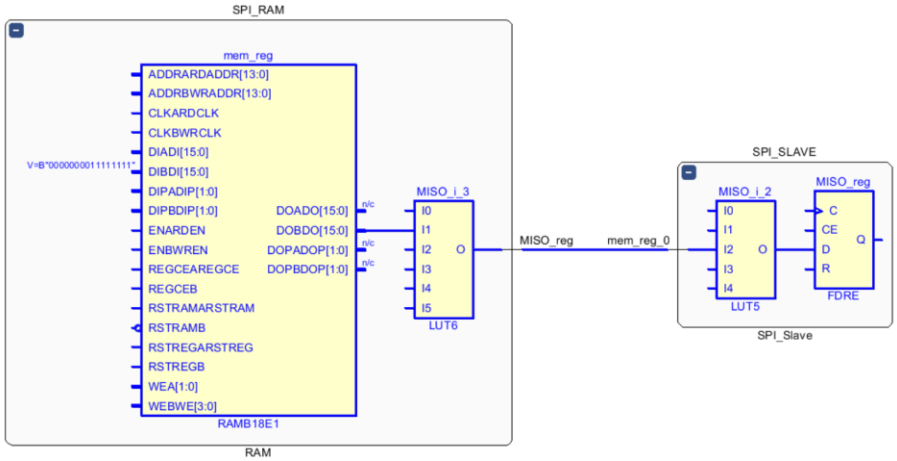
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	26	20800	0.13
FF	40	41600	0.10
BRAM	0.50	50	1.00
IO	5	106	4.72

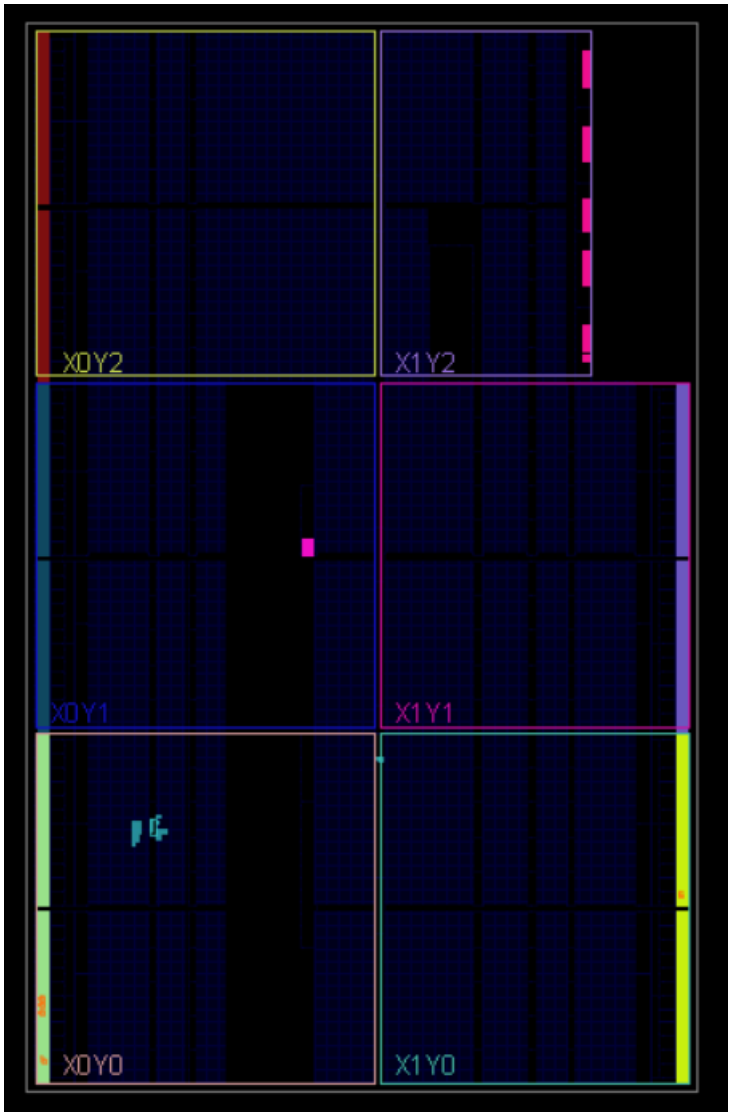


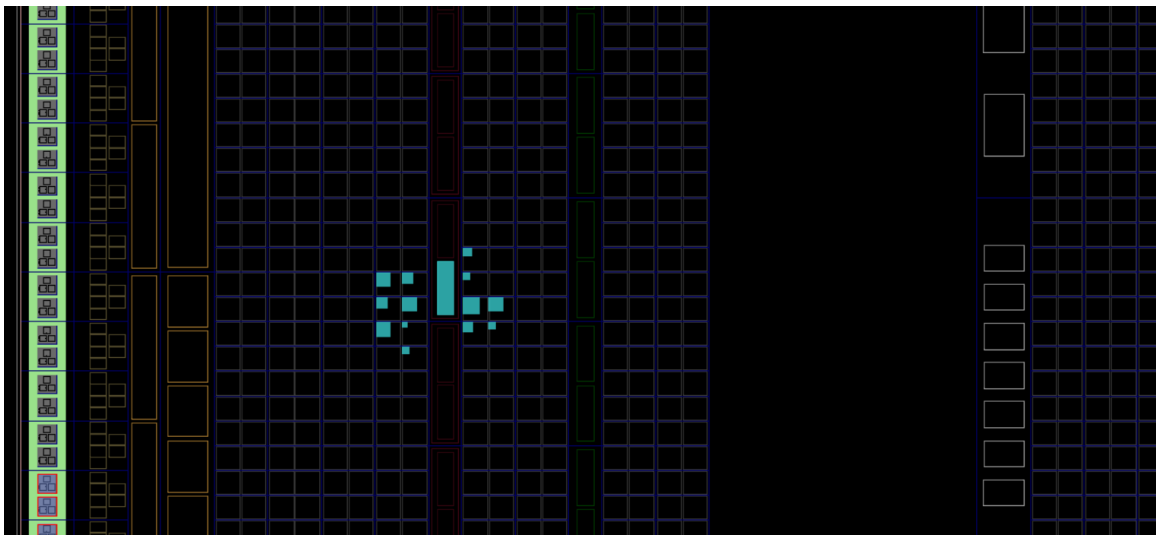
Critical Path:

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	5.375	2	1	SPI_RAM/mem.../CLKBWRCLK	SPI_SLAVE/MISO_reg/D	4.652	2.702	1.950



Device:

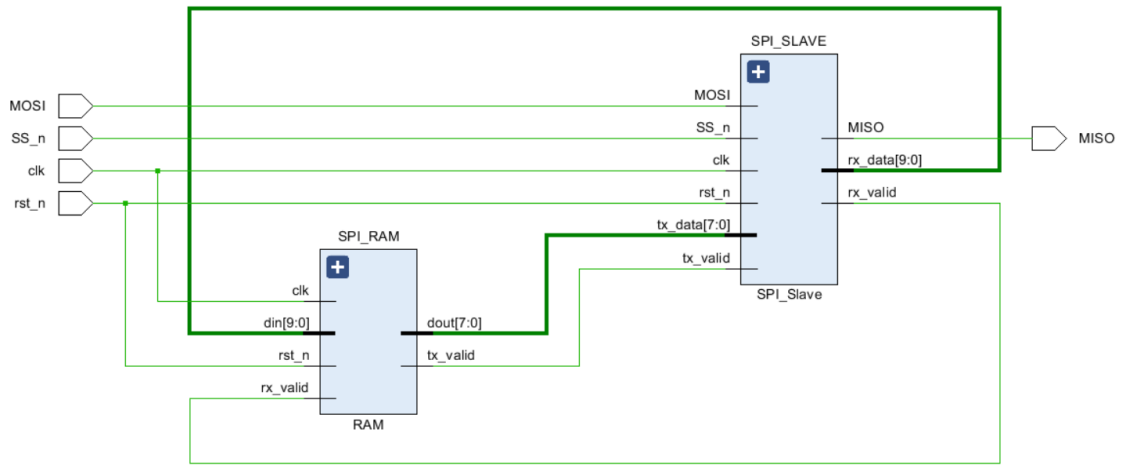




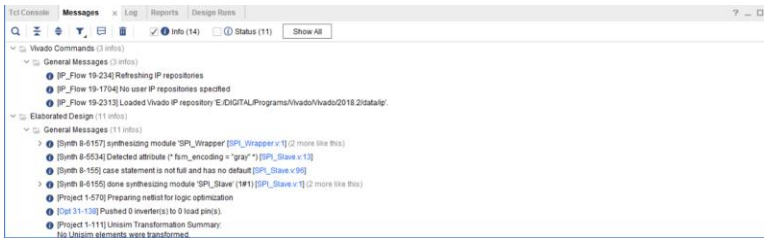
Sequential Encoding

Elaboration:

Schematic:



Messages Tab:



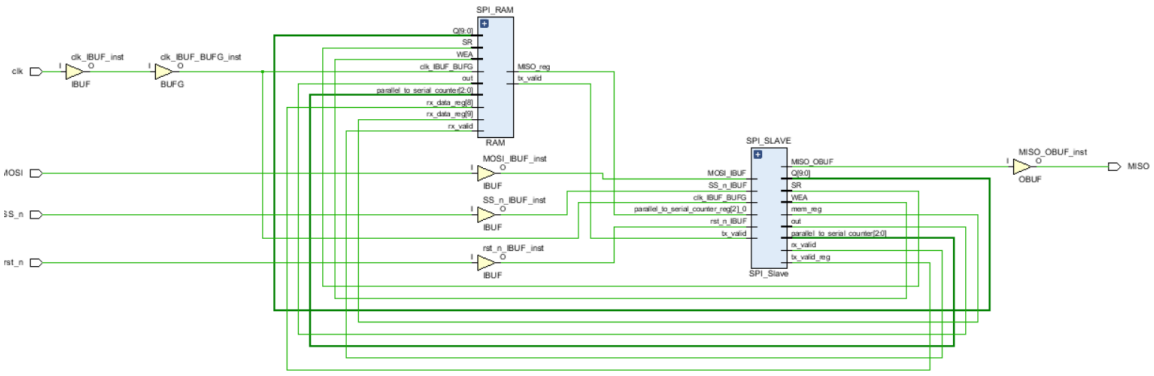
Synthesis:

Encoding:

State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	001	001
WRITE	010	010
READ_ADD	011	011
READ_DATA	100	100

Messages Tab:

Schematic:



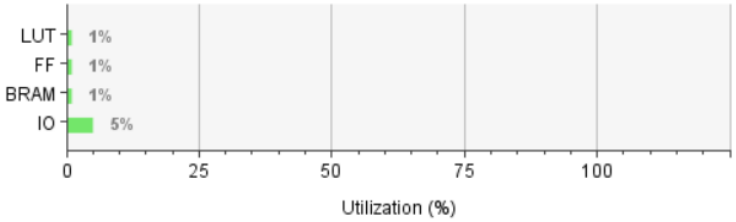
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.898 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	98	Total Number of Endpoints:	98	Total Number of Endpoints:	43

All user specified timing constraints are met.

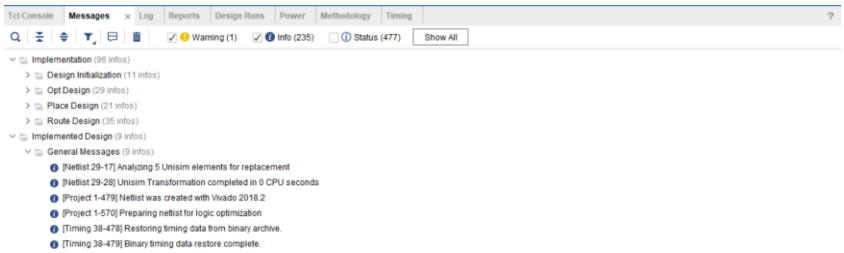
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	27	20800	0.13
FF	40	41600	0.10
BRAM	0.50	50	1.00
IO	5	106	4.72

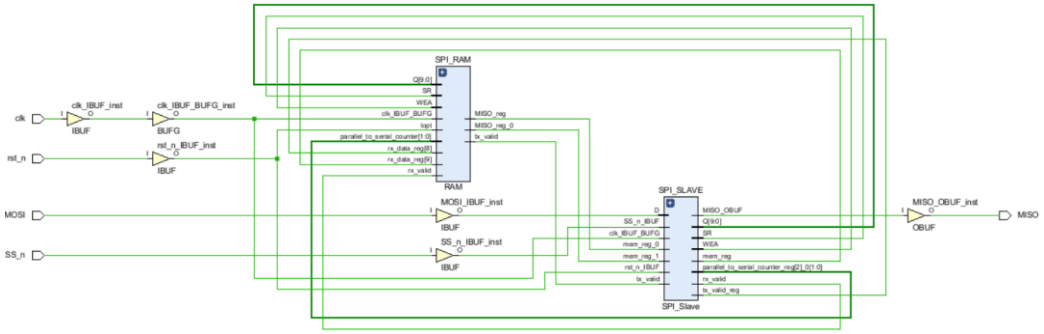


Implementation:

Messages Tab:



Schematic:



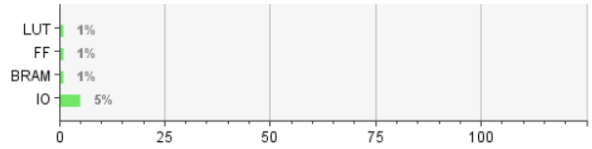
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.639 ns	Worst Hold Slack (WHS):	0.068 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	98	Total Number of Endpoints:	98	Total Number of Endpoints:	43

All user specified timing constraints are met.

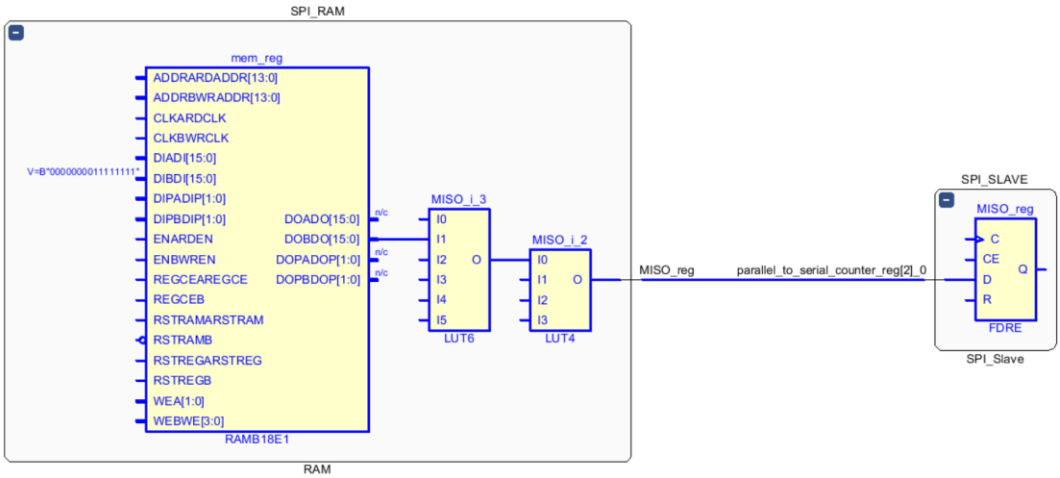
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	27	20800	0.13
FF	40	41600	0.10
BRAM	0.50	50	1.00
IO	5	106	4.72

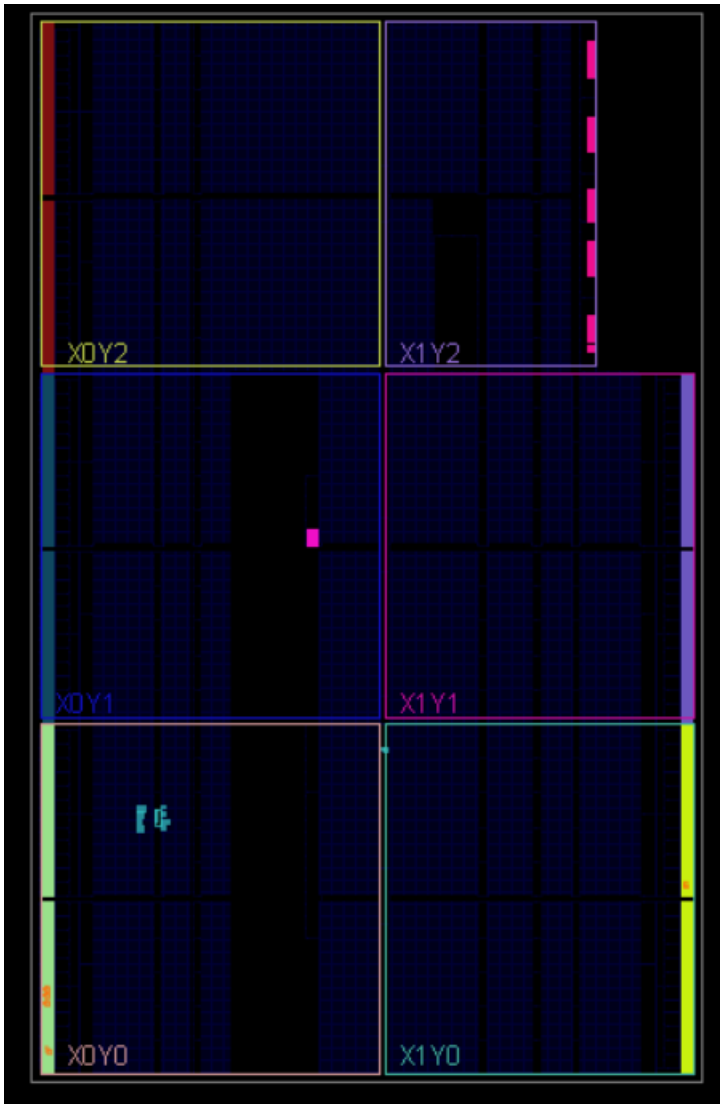


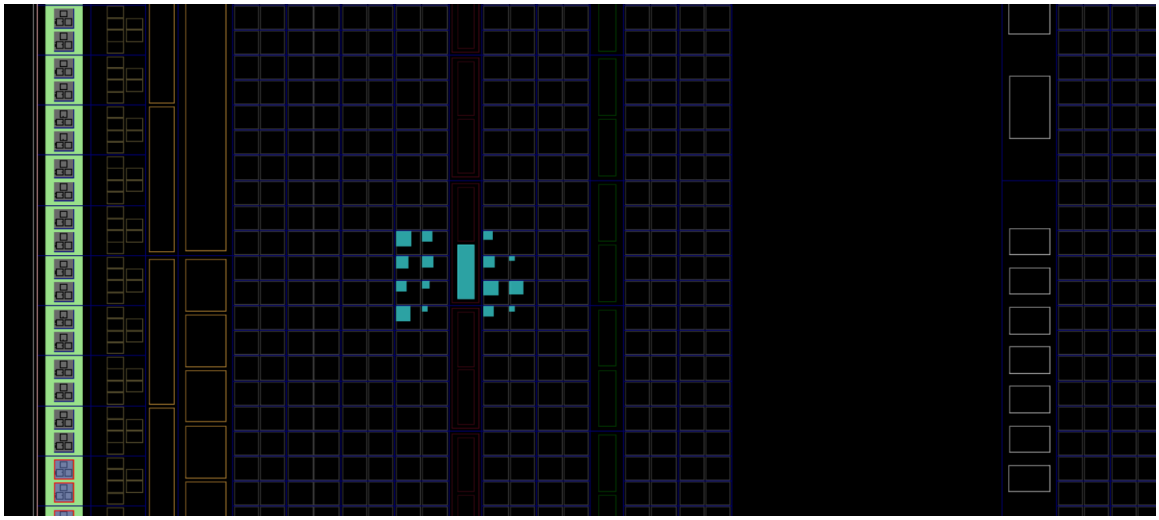
Critical Path:

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	5.639	2	1	SPI_RAM/mem.../CLKBWRCLK	SPI_SLAVE/MISO_reg/D	4.388	2.702	1.686



Device:

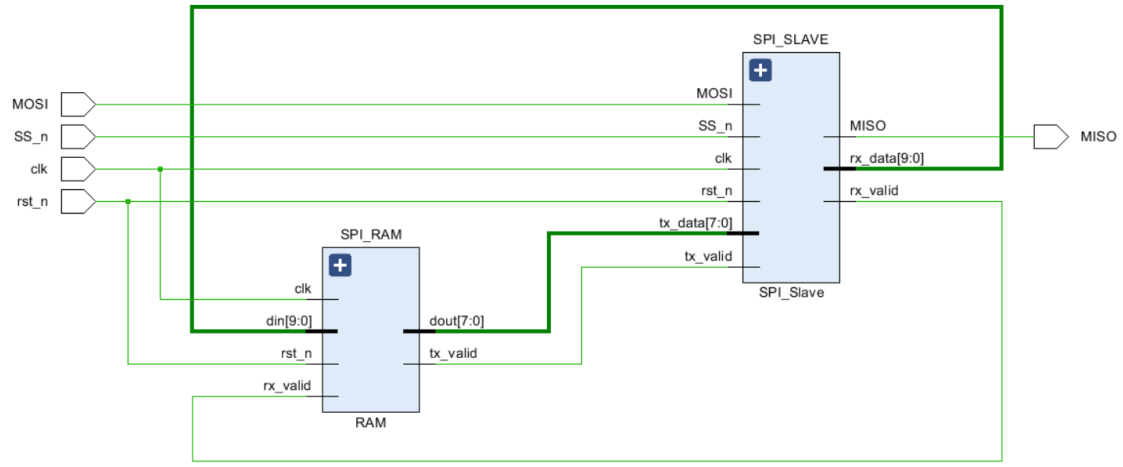




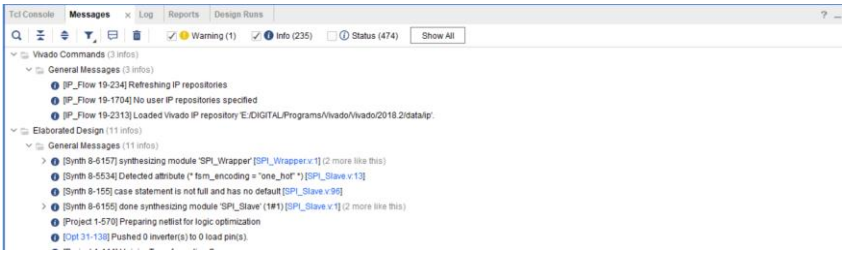
One Hot Encoding

Elaboration:

Schematic:



Messages Tab:

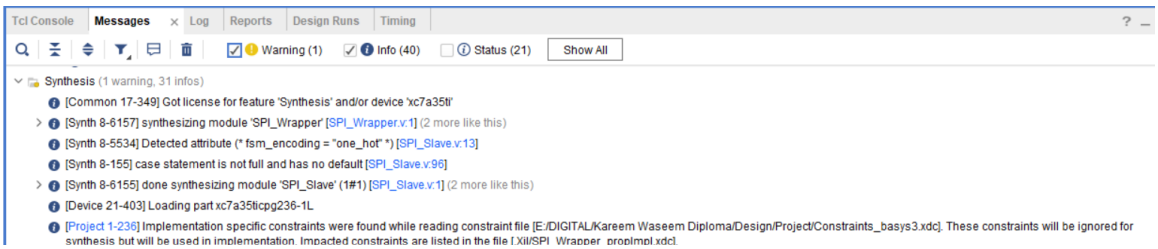


Synthesis:

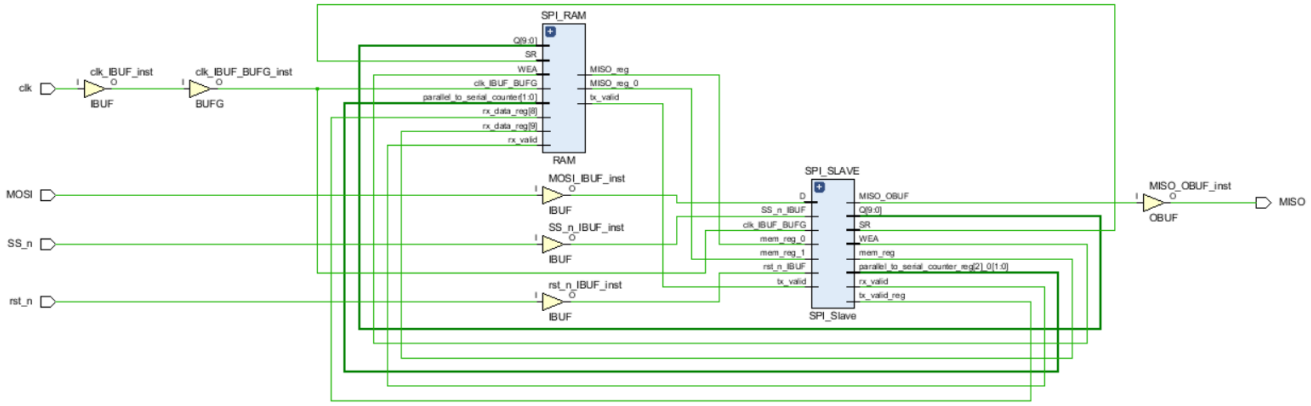
Encoding:

State	New Encoding	Previous Encoding
IDLE	00001	000
CHK_CMD	00010	001
WRITE	00100	010
READ_ADD	01000	011
READ_DATA	10000	100

Messages Tab:



Schematic:



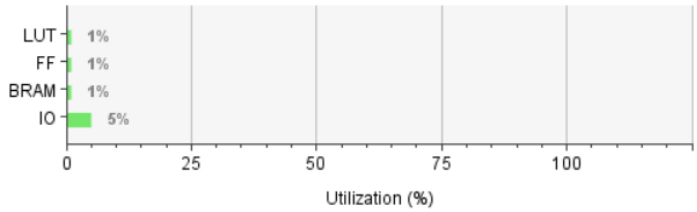
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.898 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	100	Total Number of Endpoints:	100	Total Number of Endpoints:	45

All user specified timing constraints are met.

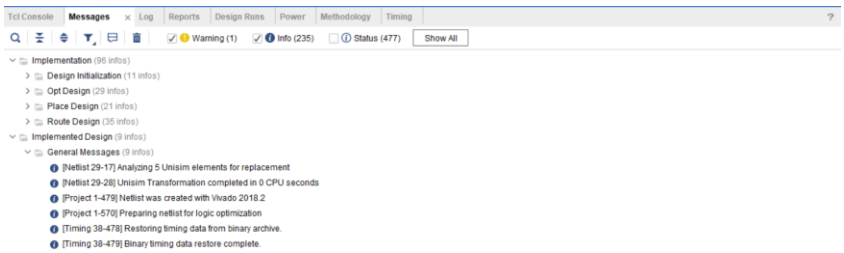
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	29	20800	0.14
FF	42	41600	0.10
BRAM	0.50	50	1.00
IO	5	106	4.72

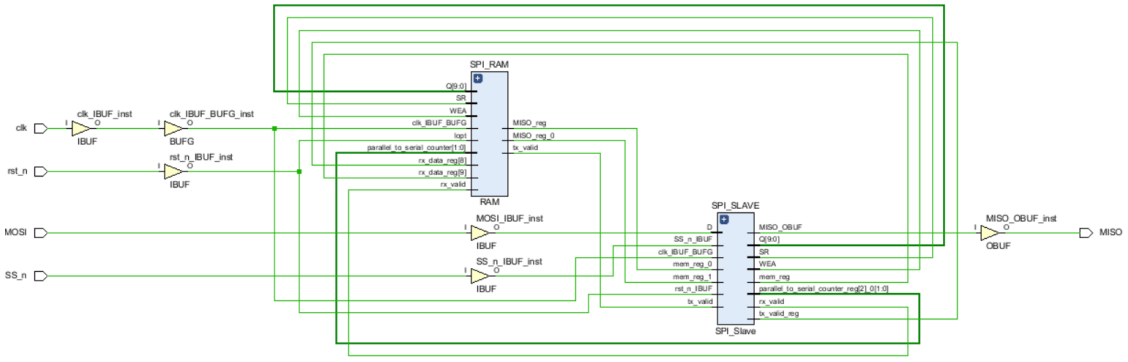


Implementation:

Messages Tab:



Schematic:



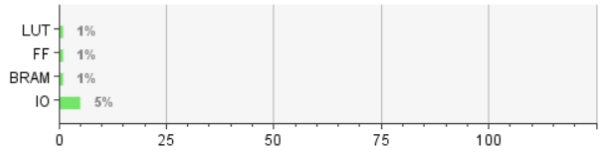
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.806 ns	Worst Hold Slack (WHS):	0.075 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	100	Total Number of Endpoints:	100	Total Number of Endpoints:	45

All user specified timing constraints are met.

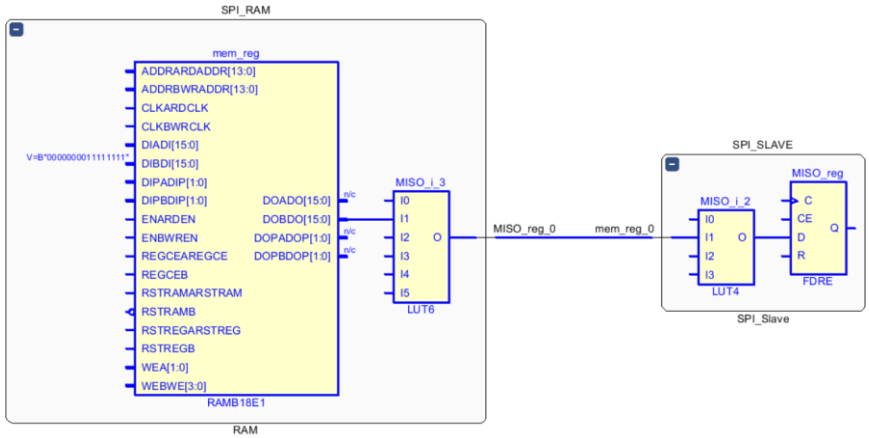
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	29	20800	0.14
FF	42	41600	0.10
BRAM	0.50	50	1.00
IO	5	106	4.72

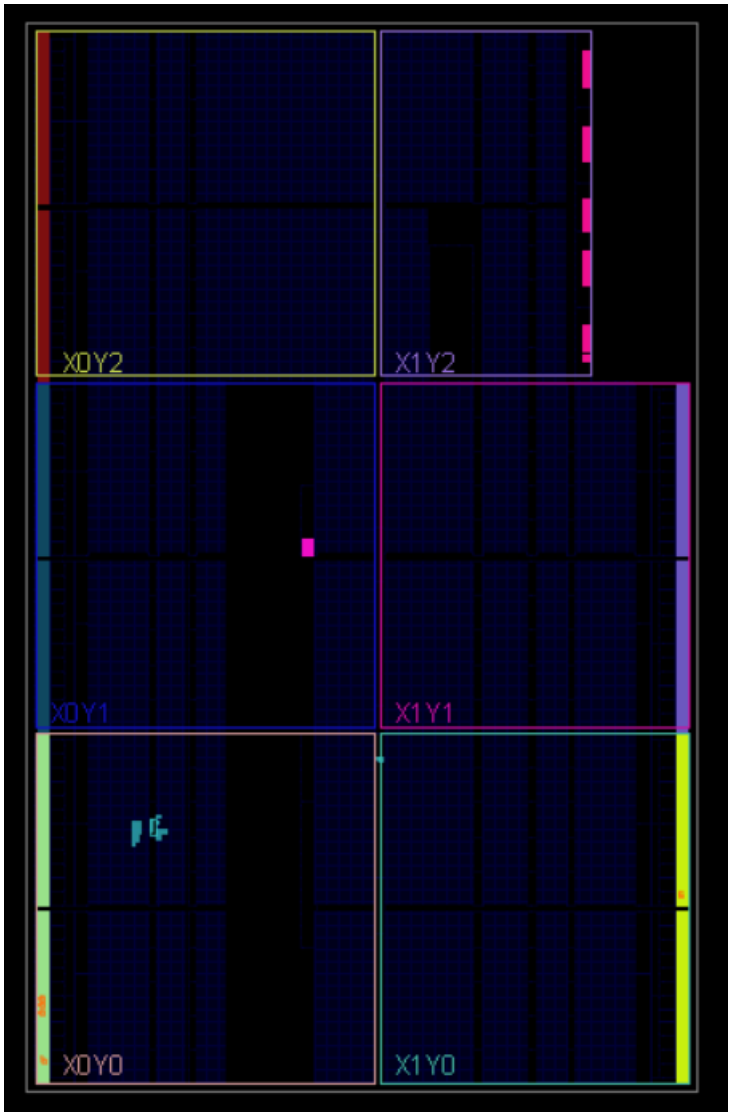


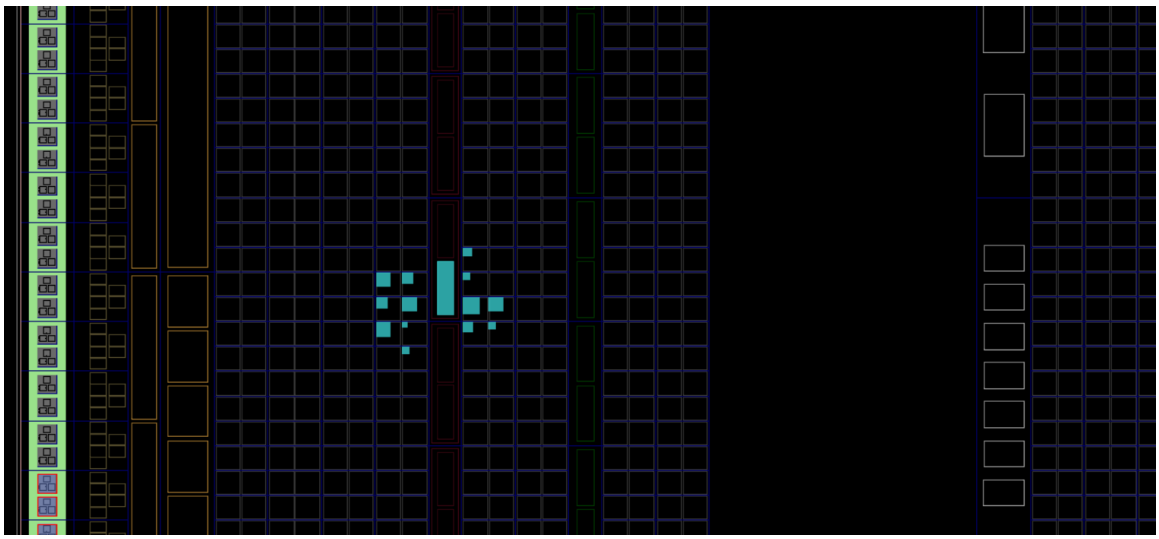
Critical Path:

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	5.806	2	1	SPI_RAM/mem.../CLKBWRCLK	SPI_SLAVE/MISO_reg/D	4.168	2.702	1.466



Device:





The Best Encoding

based on the best timing report that gives the high setup/hold slack after implementation.

Gray Encoding:

- **Worst Negative Setup Slack (WNSS):** 5.376 ns
- **Worst Hold Slack (WHS):** 0.101 ns
- **Worst Pulse Width Slack (WPWS):** 4.500 ns

Sequential Encoding:

- **Worst Negative Setup Slack (WNSS):** 5.693 ns
- **Worst Hold Slack (WHS):** 0.068 ns
- **Worst Pulse Width Slack (WPWS):** 4.500 ns

One-Hot Encoding:

- **Worst Negative Setup Slack (WNSS):** 5.806 ns
- **Worst Hold Slack (WHS):** 0.144 ns
- **Worst Pulse Width Slack (WPWS):** 4.500 ns

Analysis

- **Setup Slack:** One-Hot encoding has the highest worst negative setup slack (5.806 ns), indicating it has the most timing margin for setup constraints.
- **Hold Slack:** One-Hot encoding has the best worst hold slack (0.144 ns), providing the highest timing margin for hold constraints.
- **Pulse Width Slack:** All three encodings have the same worst pulse width slack (4.500 ns).

Comparison:

- **Setup Slack (Higher is better):**
  - Gray: 5.376 ns
  - Sequential: 5.693 ns
  - One-Hot: 5.806 ns
- **Hold Slack (Higher is better):**
  - Gray: 0.101 ns
  - Sequential: 0.068 ns
  - One-Hot: 0.144 ns

Conclusion

Given that One-Hot encoding has the highest setup and hold slack values, it provides the best timing margin overall. Therefore, to operate at the highest frequency possible, you should choose **One-Hot encoding**.

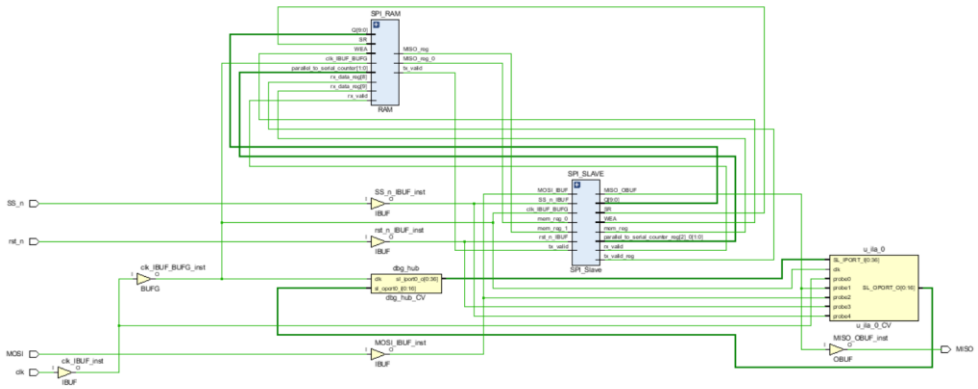
Constraints File:

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {rst_n}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {SS_n}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {MOSI}]

## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {MISO}]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

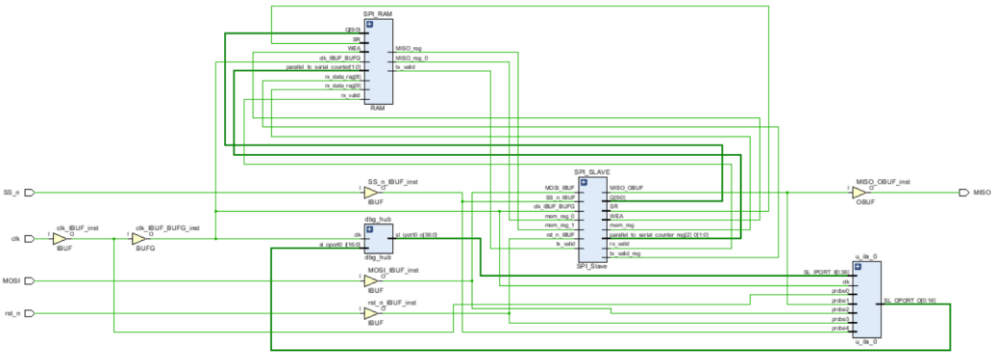
Debug Core:

Debug Core is added to One Hot Encoded Design

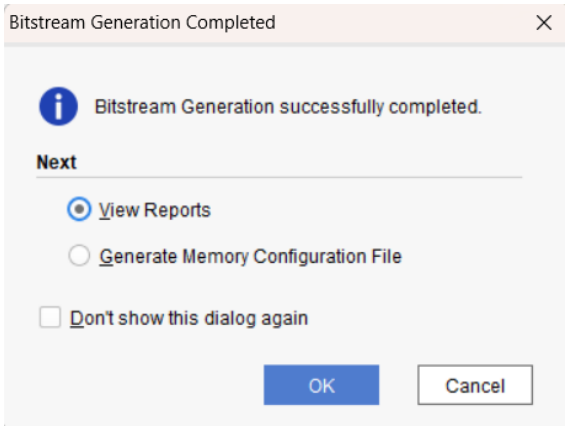
Synthesis Schematic:



Implementation Schematic:



Bitstream Generation Success:




```
0009 0ff0 0ff0 0ff0 0ff0 0000 0161 003b
5350 495f 5772 6170 7065 723b 434f 4d50
5245 5353 3d54 5255 453b 5573 6572 4944
3d30 5846 4646 4646 4646 463b 5665 7273
696f 6e3d 3230 3138 2e32 0062 000d 3761
3335 7469 6370 6732 3336 0063 000b 3230
3234 2f30 382f 3034 0064 0009 3137 3a33
343a 3437 0065 0007 93c4 ffff ffff ffff
ffff ffff ffff ffff ffff ffff ffff ffff
ffff ffff ffff ffff ffff 0000 00bb 1122
0044 ffff ffff ffff ffff aa99 5566 2000
0000 3003 e001 0000 026b 3000 8001 0000
0012 2000 0000 3002 2001 0000 0000 3002
0001 0000 0000 3000 8001 0000 0000 2000
0000 3000 8001 0000 0007 2000 0000 2000
```

Constraints file after adding debug core:

```
## Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]

## Switches
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports rst_n]
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports SS_n]
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports MOSI]
## LEDs
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports MISO]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

create_debug_core u_ila_0 ila
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk_IBUF_BUFG]]
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list clk_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
set_property port_width 1 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list MISO_OBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe2]
set_property port_width 1 [get_debug_ports u_ila_0/probe2]
connect_debug_port u_ila_0/probe2 [get_nets [list MOSI_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe3]
set_property port_width 1 [get_debug_ports u_ila_0/probe3]
connect_debug_port u_ila_0/probe3 [get_nets [list rst_n_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe4]
set_property port_width 1 [get_debug_ports u_ila_0/probe4]
connect_debug_port u_ila_0/probe4 [get_nets [list SS_n_IBUF]]
set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
connect_debug_port dbg_hub/clk [get_nets clk_IBUF_BUFG]

```

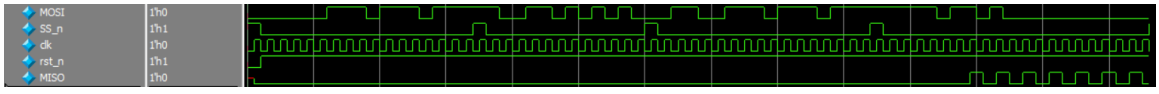
Master (SPI SLAVE Interface TB) RTL Code Snippet:

```
module MASTER();
//Signals Declaration
reg MOSI;
reg SS_n;
reg clk;
reg rst_n;
wire MISO;
//DUT Instantiation
SPI_Wrapper DUT(.MOSI(MOSI),.MISO(MISO),.SS_n(SS_n),.clk(clk),.rst_n(rst_n));
//Clock Generation
initial begin
    clk = 0;
    forever
        #5 clk = ~clk;
end
//Test Stimiulus Generator
initial begin
    //Activate Reset ,Un-Enable SS_n& Initialize all signals
    rst_n = 0;
    SS_n  = 1;
    MOSI  = 0;
    @(negedge clk);
    //De-Activate Reset& Start testing
    rst_n = 1;

    //Test Writing Address
    SS_n = 0;
    @(negedge clk);

    MOSI = 0;//Write
    @(negedge clk);

    MOSI = 0;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    //2'b00 >> To write address
    repeat(3) begin
        MOSI = 0;
        @(negedge clk);
        MOSI = 1;
        @(negedge clk);
        MOSI = 1;
        @(negedge clk);
        MOSI = 1;
        @(negedge clk);
    end
    //Address : 8'b01110111
    SS_n = 1;
    @(negedge clk);

```

```
    //Test Writing Data
    SS_n = 0;
    @(negedge clk);
    MOSI = 0;//Wtite
    @(negedge clk);

    MOSI = 0;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    //2'b01 >> To write Data
    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    //Data:8'b10101010
    SS_n = 1;
    @(negedge clk);

    //Test Reading Address
    SS_n = 0;
    @(negedge clk);

    MOSI = 1;//Read
    @(negedge clk);

    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
```

```
93      @(negedge clk);
94      //2'b10 >> To Read address
95      repeat(3) begin
96          MOSI = 0;
97          @(negedge clk);
98          MOSI = 1;
99          @(negedge clk);
100         MOSI = 1;
101         @(negedge clk);
102         MOSI = 1;
103         @(negedge clk);
104     end
105     //Address : 8'b01110111
106     SS_n = 1;
107     @(negedge clk);
108     //Test Reading Data
109     SS_n = 0;
110     @(negedge clk);
111
112     MOSI = 1;//Read
113     @(negedge clk);
114
115     MOSI = 1;
116     @(negedge clk);
117     MOSI = 1;
118     @(negedge clk);
119     //2'b11 >> To Read Data
120     MOSI = 0;
121     @(negedge clk);
122     MOSI = 1;
123     @(negedge clk);
124     MOSI = 1;
125     @(negedge clk);
126     MOSI = 0;
127     @(negedge clk);
128     MOSI = 1;
129     @(negedge clk);
130     MOSI = 0;
131     @(negedge clk);
132     MOSI = 0;
133     @(negedge clk);
134     MOSI = 0;
135     @(negedge clk);
136     //Dummy Data
137     repeat(8) @(negedge clk);
138     SS_n = 1;
139     $stop;
140 end
141 //Test Monitor & Results
142 initial begin
143     $monitor("MOSI = %b, MISO = %b, SS_n = %b",MOSI, MISO, SS_n);
144 end
145 endmodule
```

Simulation Waveform Snippet:



Do File:

```
vlib work
vlog RAM.v SPI_Slave.v SPI_Wrapper.v MASTER.v
vsim -voptargs=+acc work.MASTER
add wave *
run -all
#quit -sim
```

تم بحمد الله

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

وَأَن لَّيۡسَ لِلۡإِنسَٰنِ إِلَّا مَا سَعَىٰ ﴿٣٩﴾

وَقُلِ ٱعۡمَلُواْ فَسَيَرَى ٱللَّهُ عَمَلَكُمۡ وَرَسُولُهُۥ وَٱلۡمُؤۡمِنُونَۖ وَسَتُرَدُّونَ إِلَىٰ عَٰلِمِ ٱلۡغَيۡبِ وَٱلشَّهَٰدَةِ فَيُنَبِّئُكُم بِمَا كُنتُمۡ تَعۡمَلُونَ ﴿١٠٥﴾