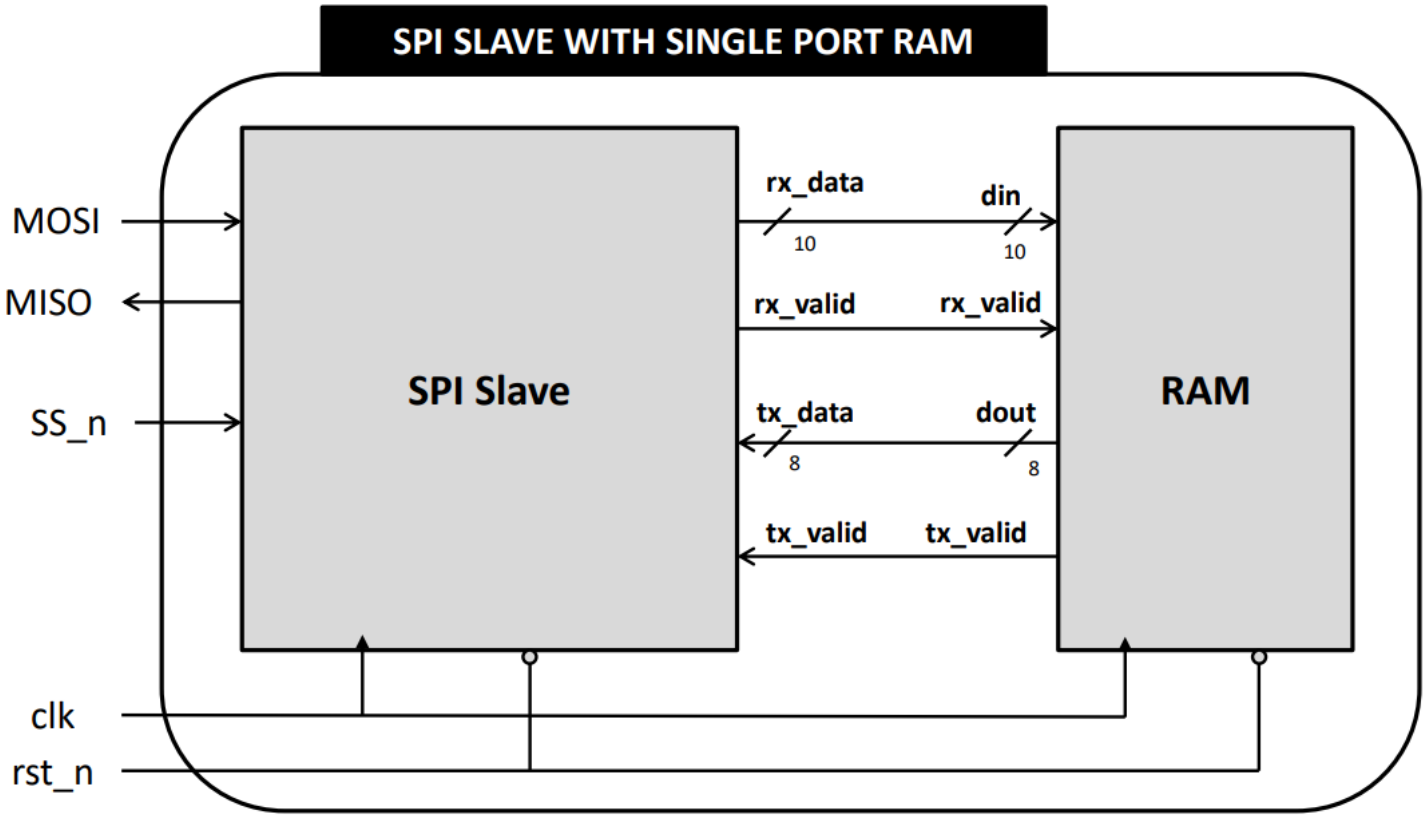
FINAL PROJECT

SPI SLAVE Interface

By: Sarah Abdelatty Ibrahem

Under Supervision of Eng. Kareem Waseem



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:

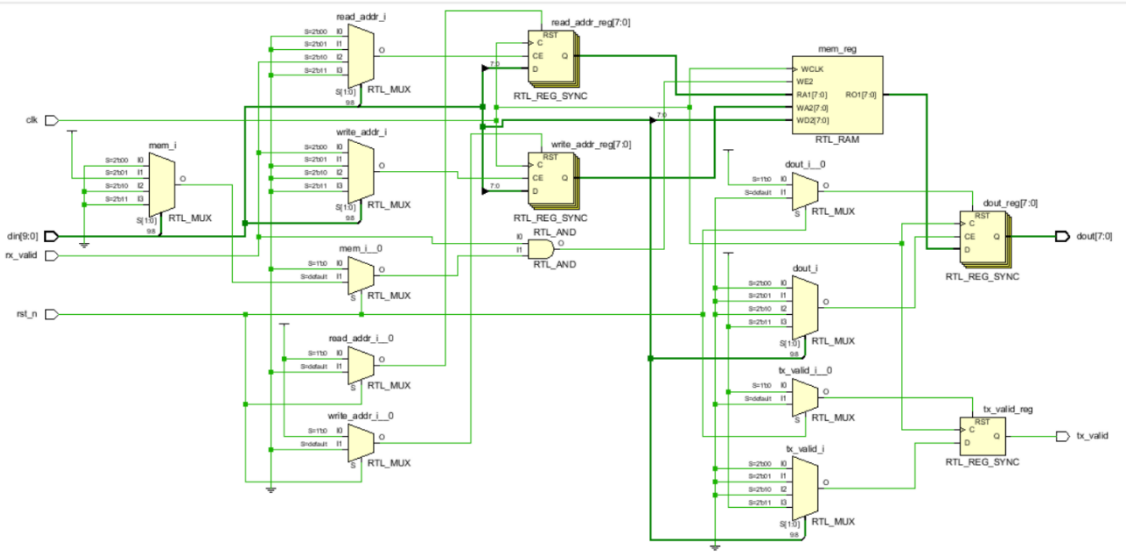
Port	din[9:8]	Command	Description
din	00	Write	Holds din[7:0] internally as a write address
din	01	Write	Writes din[7:0] to the memory with the write address held previously
din	10	Read	Holds din[7:0] internally as a read address
din	11	Read	Reads the memory with the read address held previously. tx_valid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

RTL Code Snippet:

```
module SYN_RAM (din, clk, rst_n, rx_valid, dout, tx_valid);
// Define Parameter
parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;

// Define input ports
input [9:0] din;
input clk, rst_n, rx_valid;

// Define output ports
output reg [7:0] dout;
output reg tx_valid;

// Define Memory with width size = 8
reg [7:0] mem [MEM_DEPTH-1:0];

// Define adresses
reg [7:0] write_addr, read_addr;

// Define Data input
reg [7:0] data;

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

Elaboration:

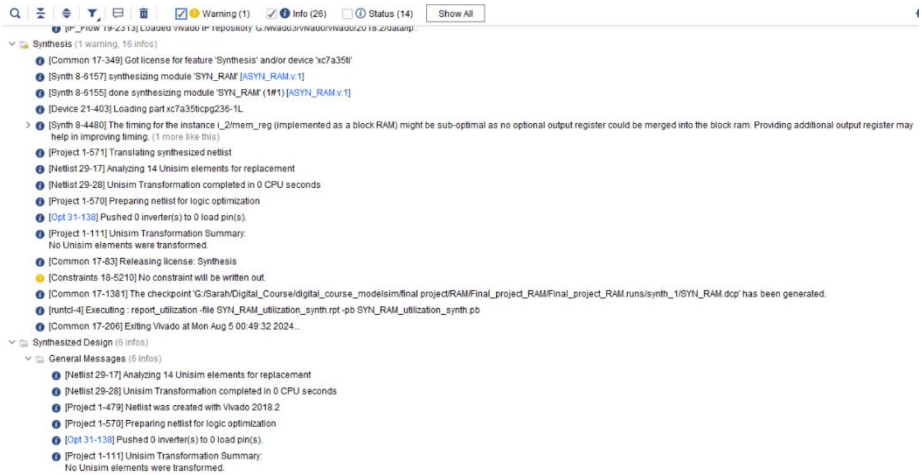
Schematic:



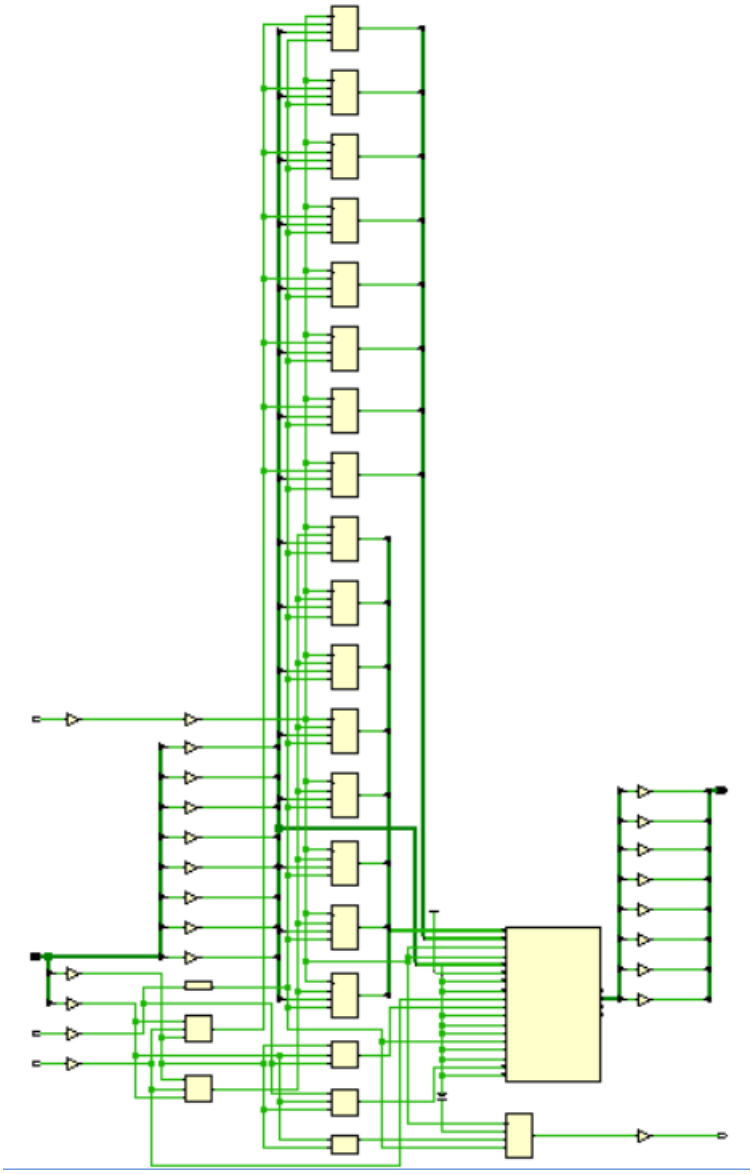
Messages Tab:

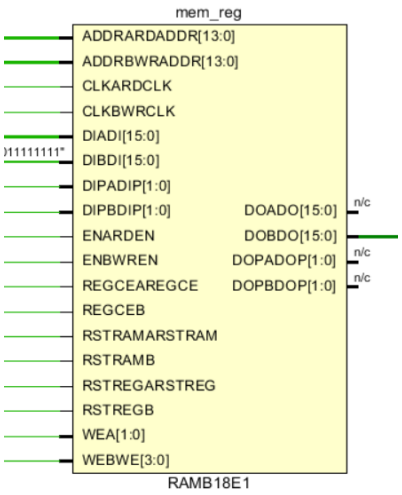
Synthesis:

Messages Tab:



Schematic:





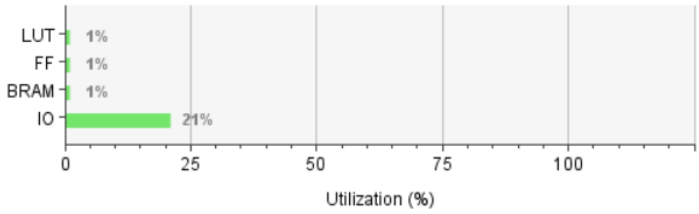
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	88	Total Number of Endpoints:	88	Total Number of Endpoints:	NA

There are no user specified timing constraints.

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

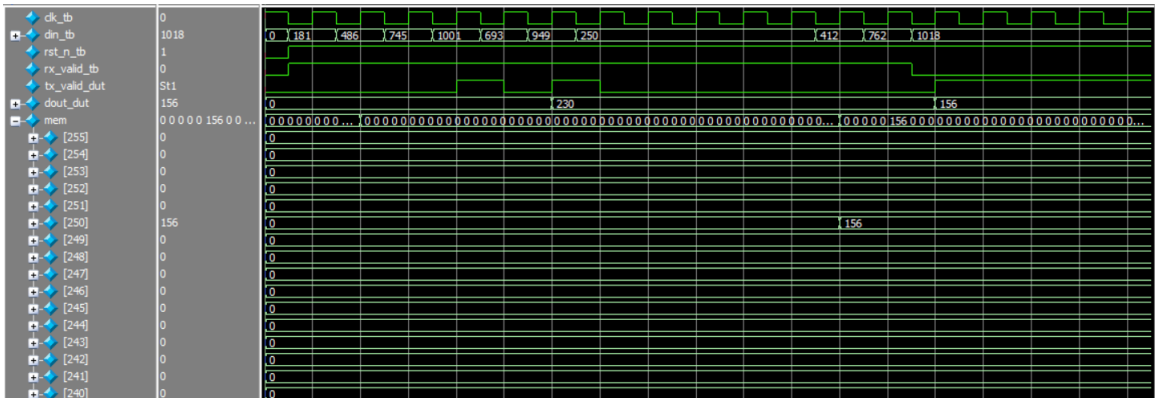


Testbench Code Snippet:

```
module SYN_RAM_tb ();
parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;
reg [9:0] din_tb;
reg clk_tb, rst_n_tb, rx_valid_tb;
wire [7:0] dout_dut;
wire tx_valid_dut;
integer i;

SYN_RAM DUT(
.din(din_tb), .clk(clk_tb),
.rst_n(rst_n_tb), .rx_valid(rx_valid_tb),
.dout(dout_dut), .tx_valid(tx_valid_dut)
);

initial begin
    clk_tb = 1;
    forever
        #1 clk_tb = ~clk_tb;
end

initial begin
    // Initialize all comntrol signals
    din_tb = 0;
    rst_n_tb = 1;
    rx_valid_tb = 0;

    // Initialize the memory
    for (i=0; i<256; i=i+1)
        DUT.mem[i] = 0;

    // Test reset
    rst_n_tb = 0;
    @(negedge clk_tb);

    // Release reset and activate write (first write address)
    rst_n_tb = 1;
    rx_valid_tb = 1;
    din_tb[9:8] = 2'b00;
    din_tb[7:0] = 8'd181; // addres of write = 181
    @(negedge clk_tb);
    // Activate write (then write data)
    din_tb[9:8] = 2'b01;
    din_tb[7:0] = 8'd230; // data input = 230
    @(negedge clk_tb);

    // Activate read (first read address)
    din_tb[9:8] = 2'b10;
    din_tb[7:0] = 8'd233; // addrees that i want to read = 233
    @(negedge clk_tb);
    // Activate read (first read address)
    din_tb[9:8] = 2'b11; // read zero
    @(negedge clk_tb);
```

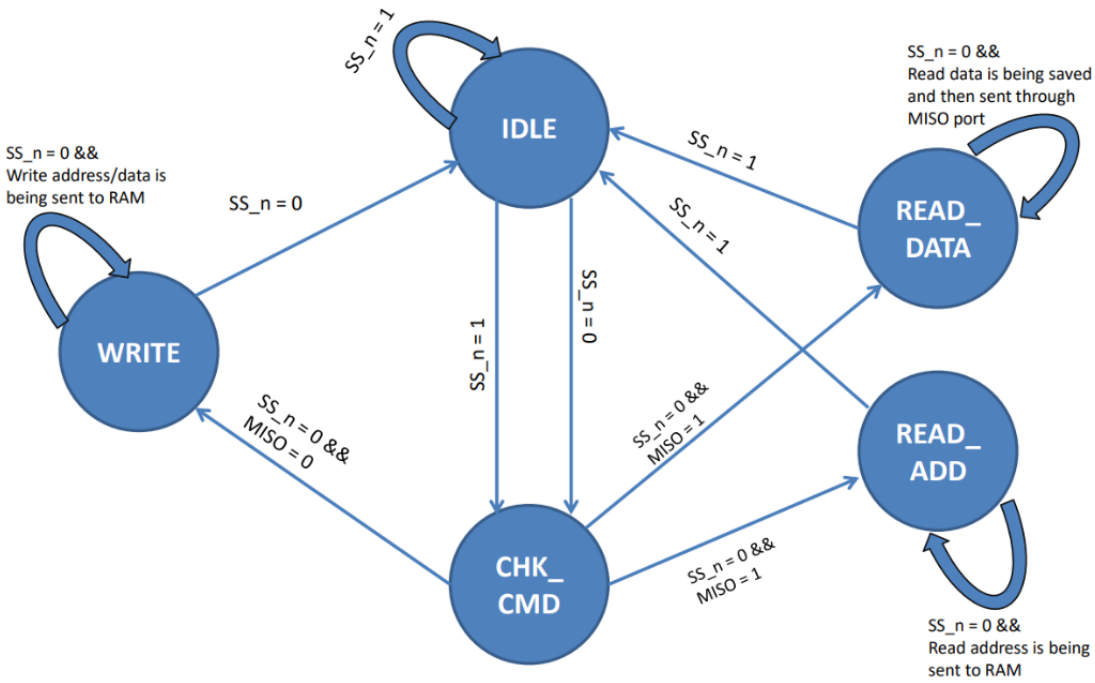
```
    // Activate read (read the address that have been writen 181)
    din_tb[9:8] = 2'b10;
    din_tb[7:0] = 8'd181; // addrees that i want to read = 181
    @(negedge clk_tb);
    din_tb[9:8] = 2'b11; // read 230
    @(negedge clk_tb);

    // Activate write
    din_tb[9:8] = 2'b00;
    din_tb[7:0] = 8'd250; // addres of write = 250
    repeat (5) @(negedge clk_tb);
    din_tb[9:8] = 2'b01;
    din_tb[7:0] = 8'd156; // data input = 156
    @(negedge clk_tb);

    // Activate read (read the address that have been writen 250)
    din_tb[9:8] = 2'b10;
    din_tb[7:0] = 8'd250; // addrees that i want to read = 250
    @(negedge clk_tb);
    rx_valid_tb = 0; // Test that the fourth condition doesn't depend on rx_valid
    din_tb[9:8] = 2'b11; // read 156
    repeat (5) @(negedge clk_tb);
    $stop;
end

initial begin
    $monitor("din = %d, rx_valid = %b, dout = %d, tx_valid = %b",
             din_tb, rx_valid_tb, dout_dut, tx_valid_dut);
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.

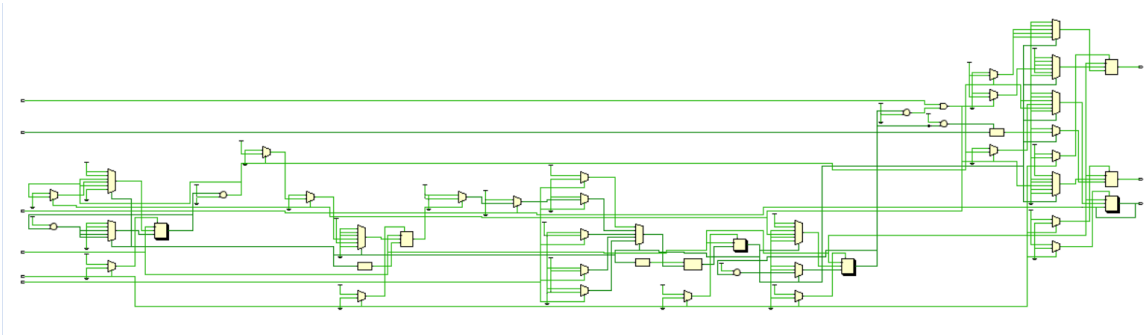
RTL Code Snippets:

```
module FSM_SPI (MOSI, MISO, ss_n, clk, rst_n, rx_data, rx_valid, tx_data, tx_valid);
// Parameters for states
parameter IDEL = 3'b000;
parameter CHK_CMD = 3'b001;
parameter WRITE = 3'b010;
parameter READ_DATA = 3'b011;
parameter READ_ADD = 3'b100;

// Input ports
input MOSI, ss_n, tx_valid, clk, rst_n;
input [7:0] tx_data;

// Output ports
output reg MISO, rx_valid;
output reg [9:0] rx_data;

// Define current and next states
(*fsm_encoding="sequential"*)
reg [2:0] cs, ns;
reg rd_sig; // Define internal signal to select read(address or data)
reg [3:0] counter_par, counter_ser; // Counters to convert from series to parallel or vice versa
// state memory block
always @(posedge clk) begin
    if (~rst_n)
        cs <= IDEL;
    else
        cs <= ns;
end

// next state logic block
always @(*) begin
    case(cs)
        IDEL:
            if (~ss_n)
                ns = CHK_CMD;
            else
                ns = IDEL;
        CHK_CMD:
            if (ss_n)
                ns = IDEL;
            else if (~ss_n) begin
                if (~MOSI)
                    ns = WRITE;
                else if (MOSI) begin
                    if (rd_sig)
                        ns = READ_DATA;
                    else if (~rd_sig)
                        ns = READ_ADD;
                end
            end
        WRITE:
            if (ss_n)
                ns = IDEL;
            else
                ns = WRITE;
        READ_DATA:
            if (ss_n)
                ns = IDEL;
            else
                ns = READ_DATA;
        READ_ADD:
            if (ss_n)
                ns = IDEL;
            else
                ns = READ_ADD;
    endcase
end

/// output logic block
always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        MISO <= 0;
        rd_sig <= 0;
        counter_ser <= 3'b000;
        counter_par <= 3'b000;
    end
    else begin
        case(cs)
            IDEL: begin
                rx_valid <= 0;
                MISO <= 0;
                counter_ser <= 3'b000;
                counter_par <= 3'b000;
            end
            CHK_CMD: begin
                rx_valid <= 0;
                counter_ser <= 3'b000;
                counter_par <= 3'b000;
            end
            WRITE: begin
                // Convert Data series to parallel(write data or address)
                if (counter_ser < 10) begin
                    rx_valid <= 0;
                    rx_data <= {rx_data[8:0], MOSI};
                    counter_ser <= counter_ser + 1;
                end
                else
                    rx_valid <= 1;
            end
            READ_DATA: begin
                // Convert Data parallel to series(read data)
                if (tx_valid && counter_par < 8) begin
                    MISO <= tx_data[7-counter_par];
                    counter_par <= counter_par + 1;
                end
```

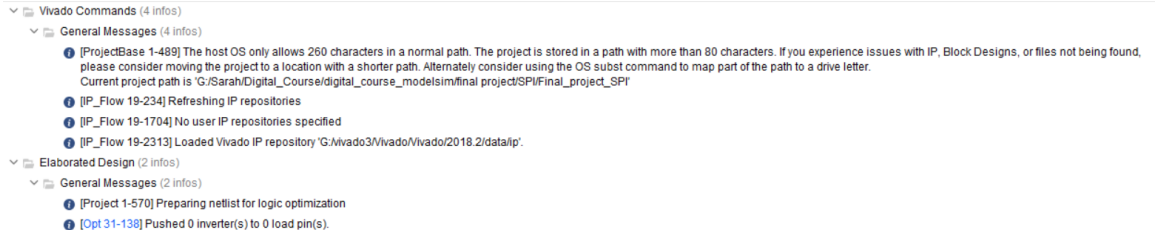
```
                else begin
                    // Convert Data series to parallel(read address that I want its data)
                    if (counter_ser<10) begin
                        rx_valid <= 0;
                        rx_data <= {rx_data[8:0], MOSI};
                        counter_ser <= counter_ser + 1;
                    end
                    else begin
                        rx_valid <= 1;
                        rd_sig <= 0;
                    end
                end
            end
            READ_ADD: begin
                rd_sig <= 1;
                // Convert Data series to parallel(read address)
                if (counter_ser < 10) begin
                    rx_valid <= 0;
                    rx_data <= {rx_data[8:0], MOSI};
                    counter_ser <= counter_ser + 1;
                end
                else
                    rx_valid <= 1;
            end
        endcase
    end
end
endmodule
```

Elaboration:

Schematic:



Messages Tab:

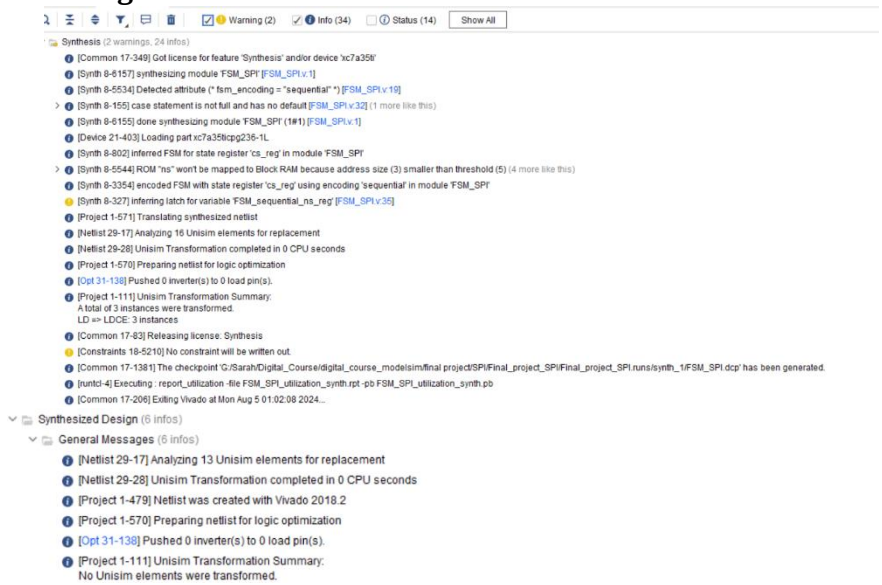


Synthesis:

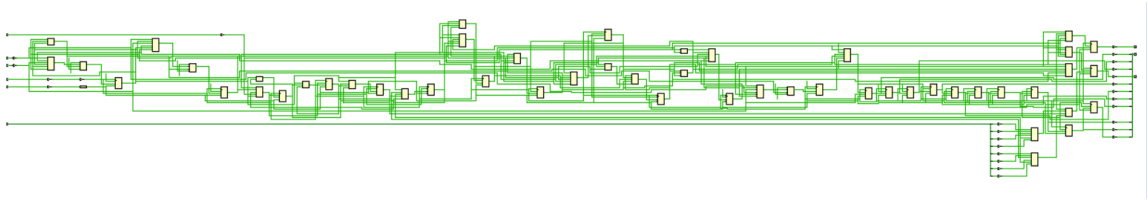
Encoding:

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:



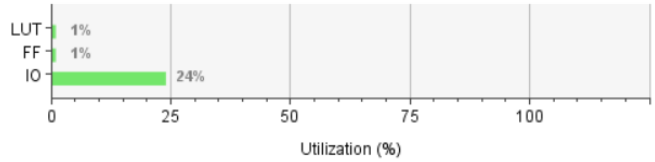
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	83	Total Number of Endpoints:	83	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	25	20800	0.12
FF	27	41600	0.06
IO	25	106	23.58



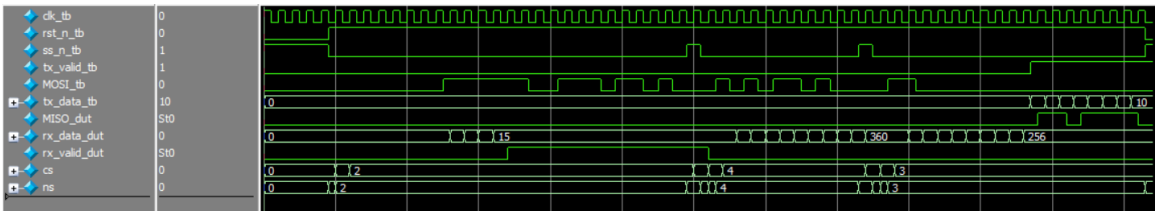
Testbench Code Snippets:

```
module FSM_SPI_tb ();
reg MOSI_tb, ss_n_tb, tx_valid_tb, clk_tb, rst_n_tb;
reg [7:0] tx_data_tb;
wire MISO_dut, rx_valid_dut;
wire [9:0] rx_data_dut;

FSM_SPI DUT(
.MOSI(MOSI_tb), .MISO(MISO_dut), .ss_n(ss_n_tb),
.clk(clk_tb), .rst_n(rst_n_tb), .rx_data(rx_data_dut),
.rx_valid(rx_valid_dut), .tx_data(tx_data_tb), .tx_valid(tx_valid_tb)
);

initial begin
    clk_tb = 1;
    forever
        #1 clk_tb = ~clk_tb;
end

initial begin
    // Initialize all control signals
    MOSI_tb = 0;
    ss_n_tb = 1;
    tx_data_tb = 0;
    rst_n_tb = 1;
    tx_valid_tb = 0;

    // Test Reset
    rst_n_tb = 0; // In the IDEL state
    repeat(5) @(negedge clk_tb);

    // Release reset and move to CHK_CMD state
    rst_n_tb = 1;
    ss_n_tb = 0;
    repeat(5) @(negedge clk_tb);

    // Check the WRITE state (Write address)
    MOSI_tb = 0;
    repeat(2) @(negedge clk_tb);
    repeat (8) begin
        MOSI_tb = $random;
        @(negedge clk_tb);
    end

    // Check the WRITE state (Write data)
    MOSI_tb = 0;
    @(negedge clk_tb);
    MOSI_tb = 1;
    @(negedge clk_tb);
    repeat (8) begin
        MOSI_tb = $random;
        @(negedge clk_tb);
    end
```

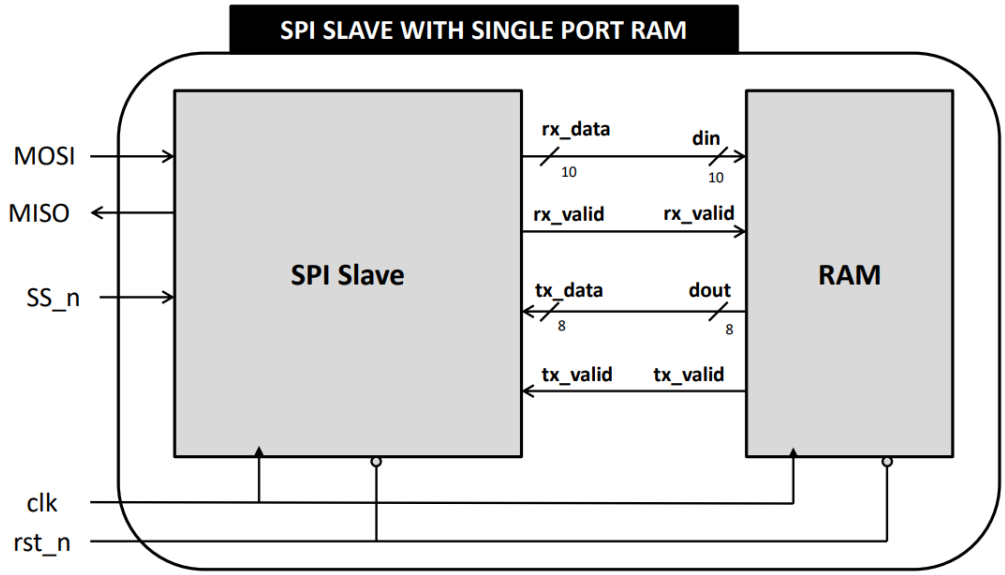
•

```
    // Back to IDEL state
    ss_n_tb = 1;
    @(negedge clk_tb);

    // Move to CHK_CMD state
    ss_n_tb = 0;
    @(negedge clk_tb);

    // Check the READ_ADD state (Read address)
    MOSI_tb = 1;
    @(negedge clk_tb);
    MOSI_tb = 0;
    @(negedge clk_tb);
    repeat (8) begin
        MOSI_tb = $random;
        @(negedge clk_tb);
    end

    // Back to IDEL state
    ss_n_tb = 1;
    @(negedge clk_tb);

    // Move to CHK_CMD state
    ss_n_tb = 0;
    @(negedge clk_tb);

    // Check the READ_DATA state (Read data)
    MOSI_tb = 1;
    repeat(2) @(negedge clk_tb);
    repeat (8) begin
        MOSI_tb = 0;
        @(negedge clk_tb);
    end
    tx_valid_tb = 1;
    repeat (8) begin
        tx_data_tb = $random;
        @(negedge clk_tb);
    end

    // Reset all signals
    ss_n_tb = 1;
    rst_n_tb = 0;
    @(negedge clk_tb);
    $stop;
end

initial begin
    $monitor("MOSI = %b, MISO = %b, ss_n = %b, rx_data = %b, rx_valid = %b, tx_data = %b, tx_valid = %b",
    MOSI_tb, MISO_dut, ss_n_tb, rx_data_dut, rx_valid_dut, tx_data_tb, tx_valid_tb);
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.

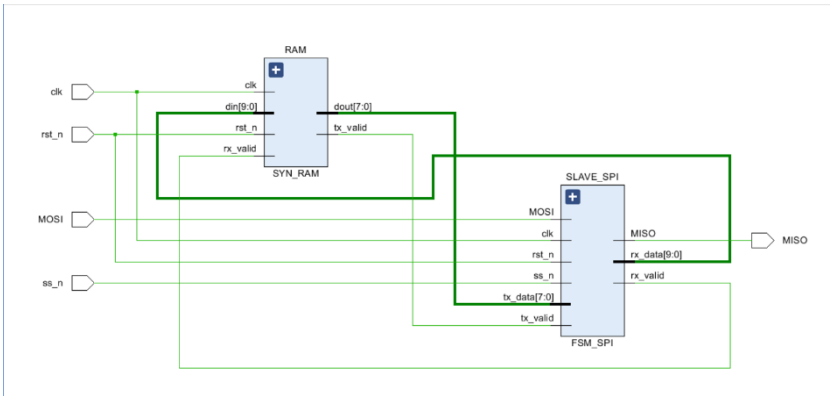
RTL Code Snippet:

```
module WRAPPER (MOSI, MISO, ss_n, clk, rst_n);
// Input ports
input MOSI, clk, rst_n, ss_n;

// Output ports
output MISO;

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

// Instintiate Master Slave SPI
FSM_SPI SLAVE_SPI(
.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)
);

// Instintiate Memory "RAM"
SYN_RAM #(
.MEM_DEPTH(256), .ADDR_SIZE(8)
) RAM(
.din(rx_data), .clk(clk),
.rst_n(rst_n), .rx_valid(rx_valid),
.dout(tx_data), .tx_valid(tx_valid)
);

endmodule
```

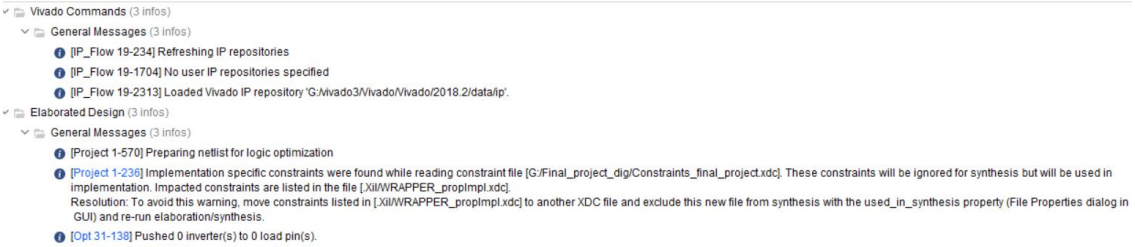
Gray Encoding

1. **Elaboration:**

Schematic:



Messages Tab:

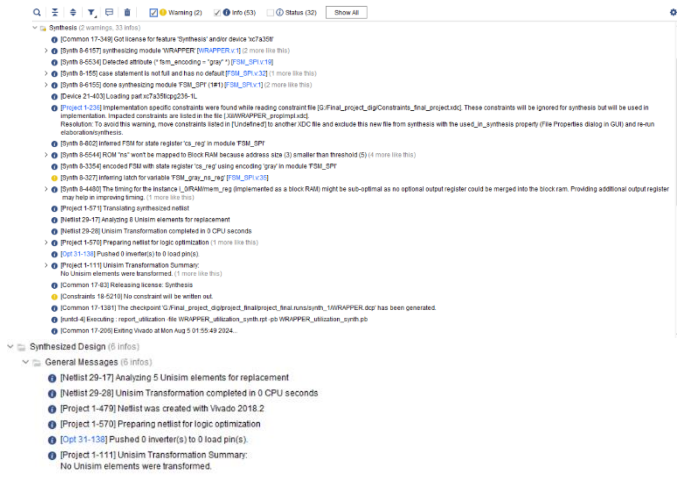


2. **Synthesis:**

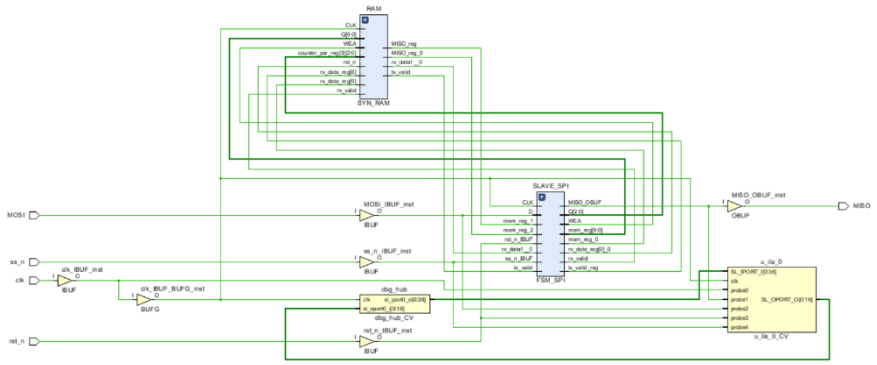
Encoding:

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:



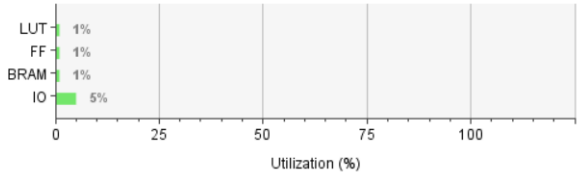
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.898 ns	Worst Hold Slack (WHS):	0.142 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:	100	Total Number of Endpoints:	100	Total Number of Endpoints:	44

All user specified timing constraints are met.

Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	29	20800	0.14
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

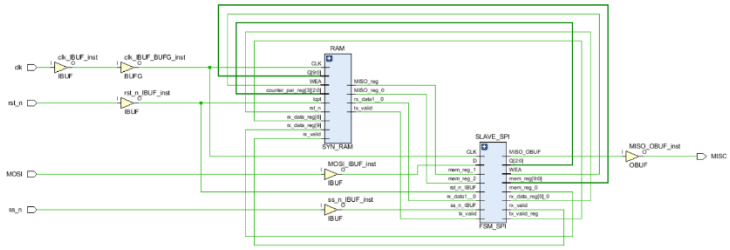


3. Implementation:

Messages Tab:

- Implemented Design (9 infos)
  - General Messages (9 infos)
    - [Netlist 29-17] Analyzing 5 Unisim elements for replacement
    - [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
    - [Project 1-479] Netlist was created with Vivado 2018.2
    - [Project 1-570] Preparing netlist for logic optimization
    - [Timing 38-478] Restoring timing data from binary archive.
    - [Timing 38-479] Binary timing data restore complete.
    - [Project 1-856] Restoring constraints from binary archive.
    - [Project 1-853] Binary constraint restore complete.
    - [Project 1-111] Unisim Transformation Summary:
      No Unisim elements were transformed.

Schematic:



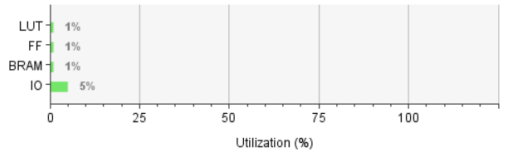
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.446 ns	Worst Hold Slack (WHS):	0.042 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:	44

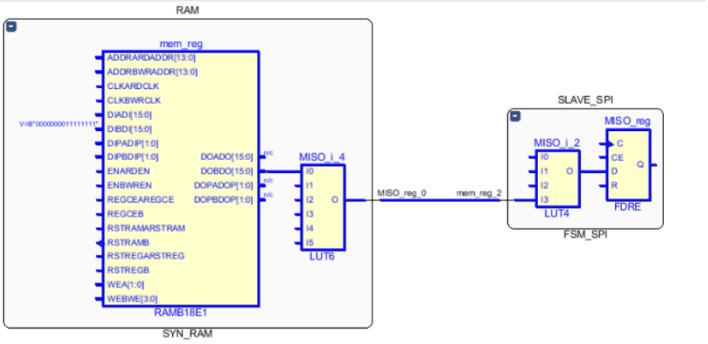
All user specified timing constraints are met.

Utilization Report:

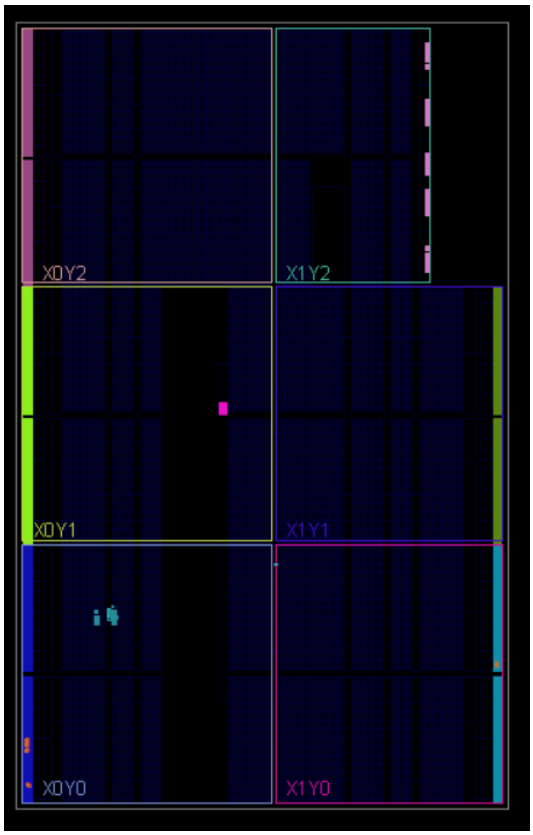
Resource	Utilization	Available	Utilization %
LUT	29	20800	0.14
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72



Critical Path:



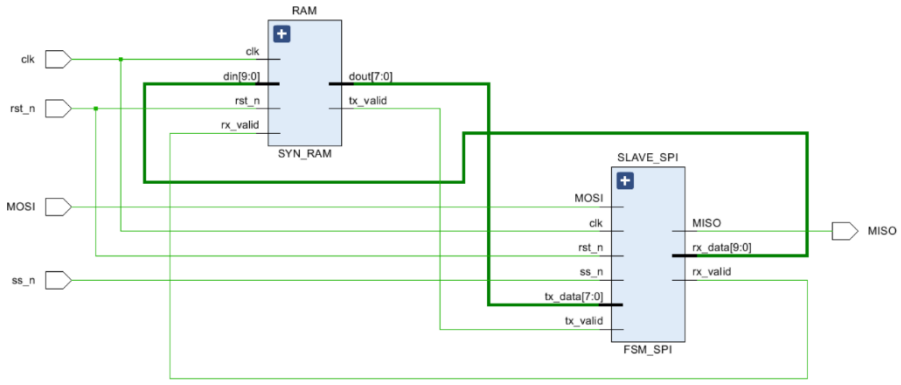
Device:



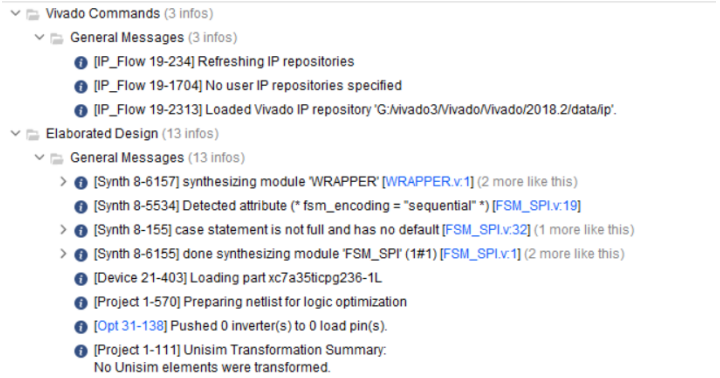
Sequential Encoding

4. Elaboration:

Schematic:



Messages Tab:

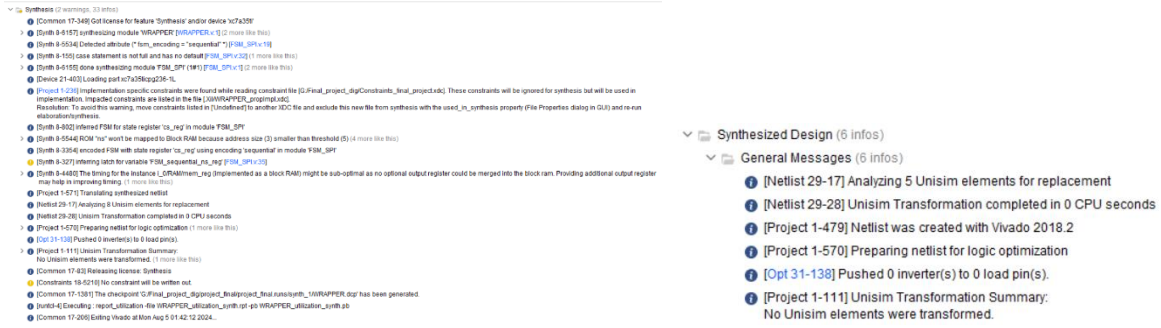


5. Synthesis:

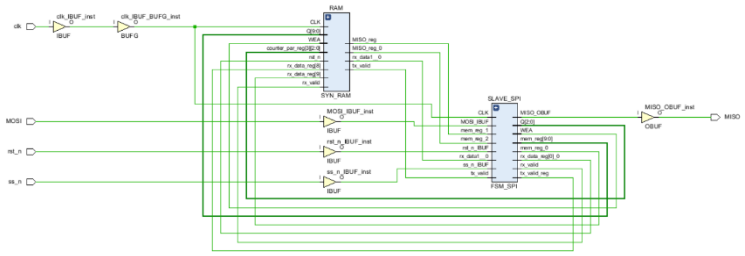
Encoding:

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:



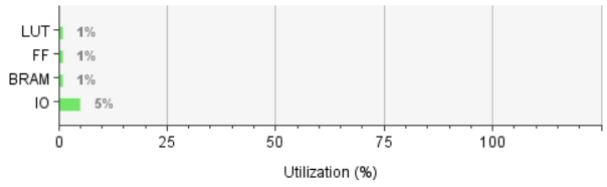
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.898 ns	Worst Hold Slack (WHS):	0.142 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:	100	Total Number of Endpoints:	100	Total Number of Endpoints:	44

All user specified timing constraints are met.

Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	29	20800	0.14
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

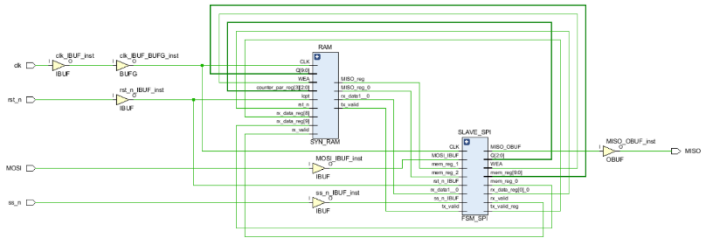


6. Implementation:

Messages Tab:

- Implemented Design (9 infos)
  - General Messages (9 infos)
    - [Netlist 29-17] Analyzing 5 Unisim elements for replacement
    - [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
    - [Project 1-479] Netlist was created with Vivado 2018.2
    - [Project 1-570] Preparing netlist for logic optimization
    - [Timing 38-478] Restoring timing data from binary archive.
    - [Timing 38-479] Binary timing data restore complete.
    - [Project 1-856] Restoring constraints from binary archive.
    - [Project 1-853] Binary constraint restore complete.
    - [Project 1-111] Unisim Transformation Summary:
      No Unisim elements were transformed.

Schematic:



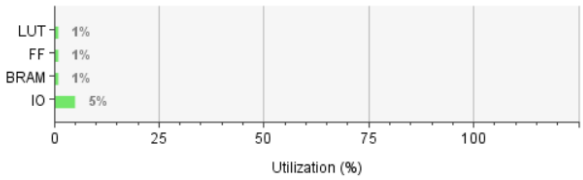
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.465 ns	Worst Hold Slack (WHS):	0.071 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:	44

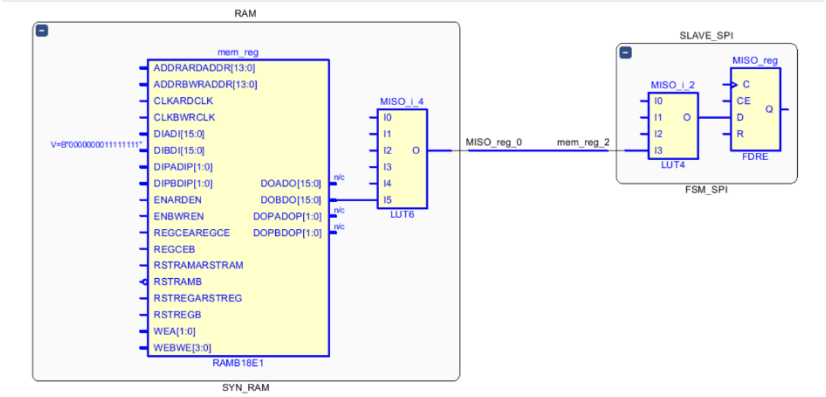
All user specified timing constraints are met.

Utilization Report:

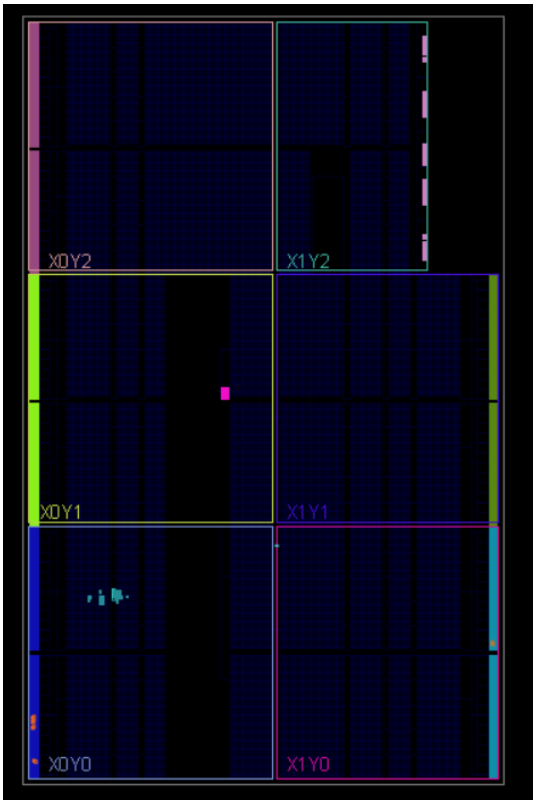
Resource	Utilization	Available	Utilization %
LUT	29	20800	0.14
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72



Critical Path:



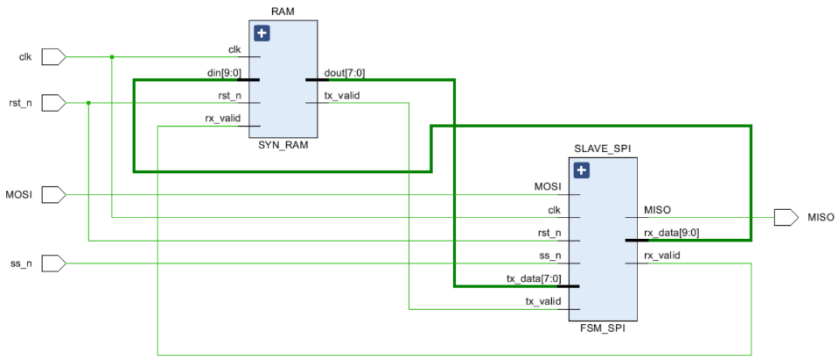
Device:



One Hot Encoding

7. Elaboration:

Schematic:



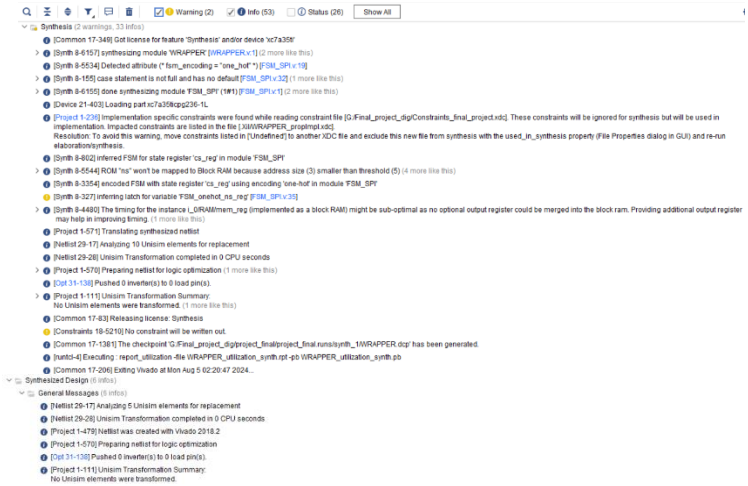
Messages Tab:

8. Synthesis:

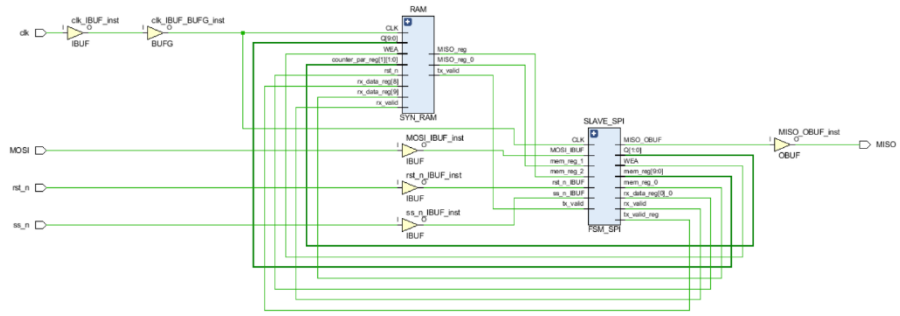
Encoding:

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:



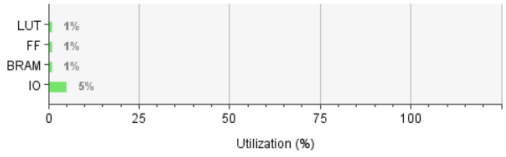
Timing Report Summary:

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:	46

All user specified timing constraints are met.

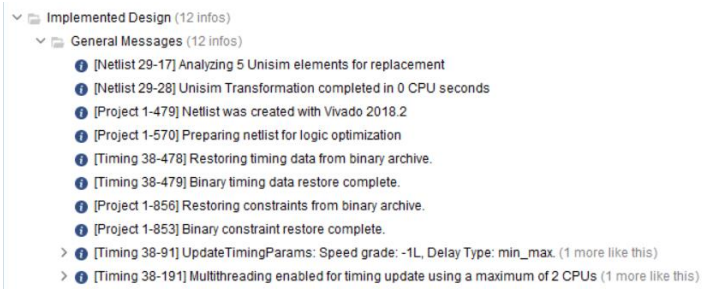
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	31	20800	0.15
FF	48	41600	0.12
BRAM	0.50	50	1.00
IO	5	106	4.72

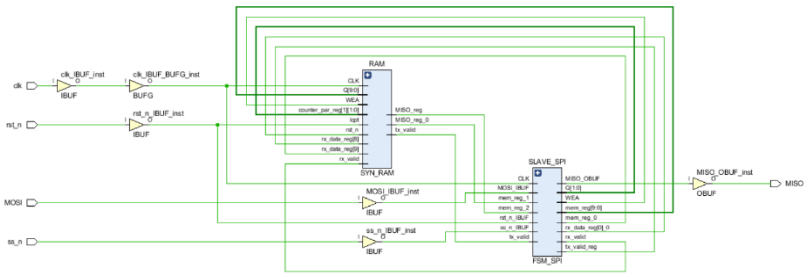


9. Implementation:

Messages Tab:



Schematic:



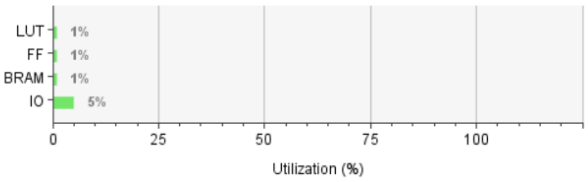
Timing Report Summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.415 ns	Worst Hold Slack (WHS):	0.046 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:	46

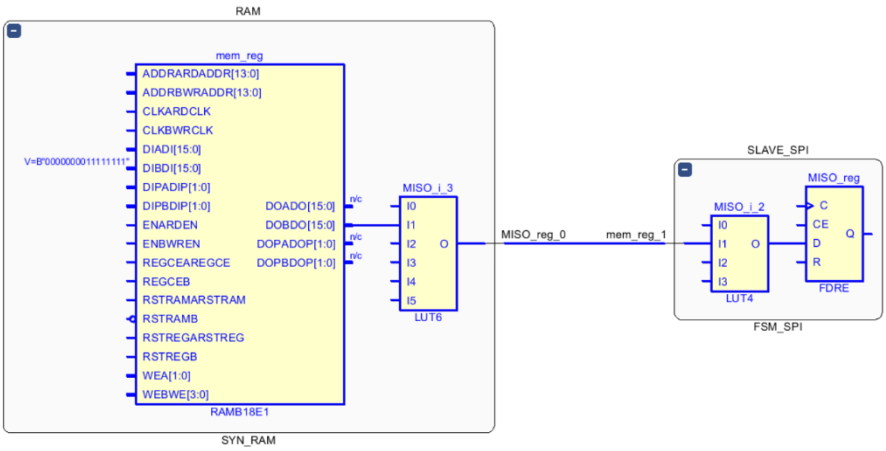
All user specified timing constraints are met.

Utilization Report:

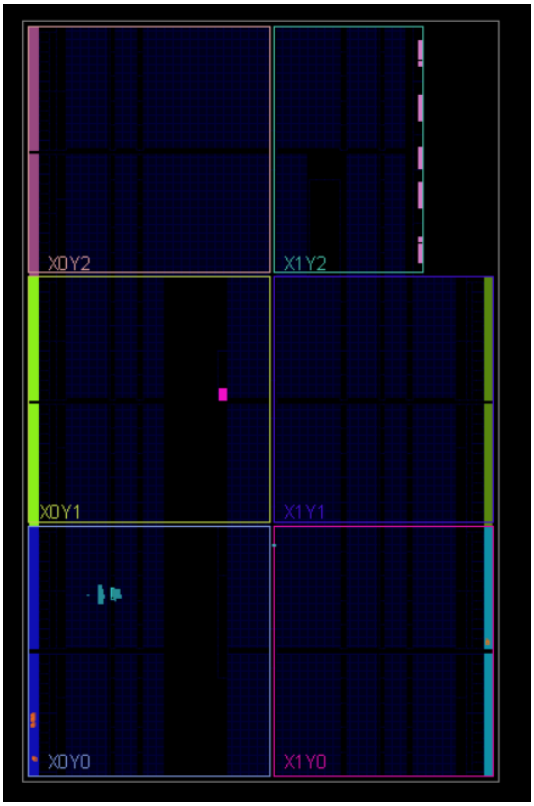
Resource	Utilization	Available	Utilization %
LUT	31	20800	0.15
FF	48	41600	0.12
BRAM	0.50	50	1.00
IO	5	106	4.72



Critical Path:



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):** 5.446 ns
- **Worst Hold Slack (WHS):** 0.042 ns
- **Worst Pulse Width Slack (WPWS):** 4.500 ns

Sequential Encoding:

- **Worst Negative Setup Slack (WNSS):** 5.465 ns
- **Worst Hold Slack (WHS):** 0.071 ns
- **Worst Pulse Width Slack (WPWS):** 4.500 ns

One-Hot Encoding:

- **Worst Negative Setup Slack (WNSS):** 5.415 ns
- **Worst Hold Slack (WHS):** 0.046 ns
- **Worst Pulse Width Slack (WPWS):** 4.500 ns

Analysis

- **Setup Slack:** Sequential encoding has the highest worst negative setup slack (5. 465 ns), indicating it has the most timing margin for setup constraints.
- **Hold Slack:** Sequential encoding has the best worst hold slack (0.071 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. 446 ns
  - Sequential: 5. 465 ns
  - One-Hot: 5.415 ns
- **Hold Slack (Higher is better):**
  - Gray: 0.042 ns
  - Sequential: 0.071 ns
  - One-Hot: 0.046 ns

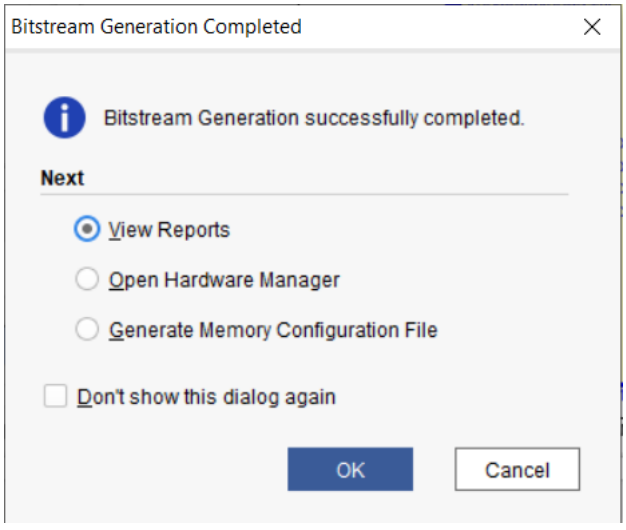
Conclusion

Given that Sequential 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 **Sequential encoding**.

Constraints File:

```
## 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]
```

Bitstream Generation Success:



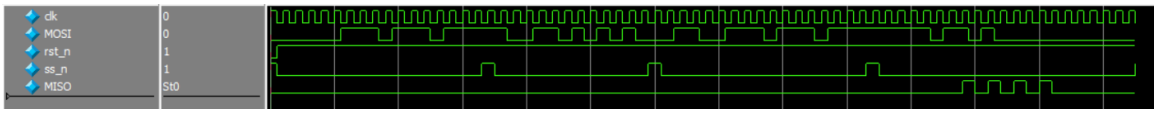
Master (SPI SLAVE Interface TB) RTL Code Snippet:

```
module MASTER_tb();
//Signals Declaration
reg MOSI;
reg ss_n;
reg clk;
reg rst_n;
wire MISO;
//DUT Instantiation
WRAPPER DUT(.MOSI(MOSI),.MISO(MISO),.ss_n(ss_n),.clk(clk),.rst_n(rst_n));
//Clock Generation
initial begin
    clk = 1;
    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);
    //2'b10 >> To Read 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;
```

```
    //Address : 8'b01110111
    ss_n = 1;
    @(negedge clk);

    //Test Reading Data
    ss_n = 0;
    @(negedge clk);

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

    MOSI = 1;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    //2'b11 >> To Read Data
    MOSI = 0;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    MOSI = 1;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    MOSI = 0;
    @(negedge clk);
    //Dummy Data
    repeat(8) @(negedge clk);
    ss_n = 1;
    $stop;
end
//Test Monitor & Results
initial begin
    $monitor("MOSI = %b, MISO = %b, ss_n = %b",MOSI, MISO, ss_n);
end
endmodule
```

Simulation Waveform Snippet:



Do File:

```
vlib work
vlog SYN_RAM.v FSM_SPI.v WRAPPER.v MASTER_tb.v
vsim -voptargs=+acc work.MASTER_tb
add wave *
run -all
#quit -sim
```