Engineering Sector
Digital Electronics Design Diploma
Eng. Kareem Waseem

Project 2

SPI Slave with Single Port RAM

Using FPGA Design Flow

BY: Youssef Ekramy

Under the supervision of:

Eng. Kareem Waseem

Table of Contents

Table of Figures

1 Project Specifications

1.1 Input-Output Ports

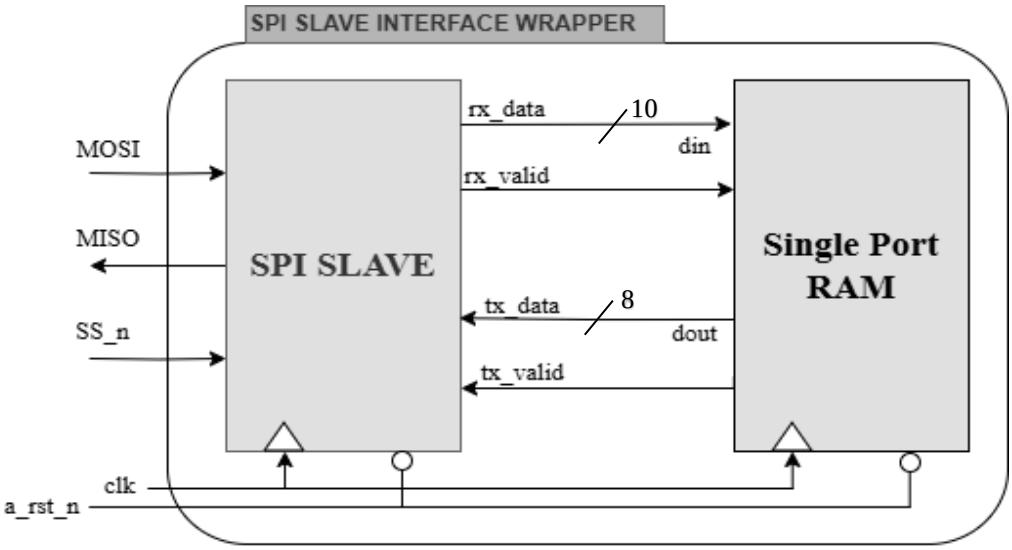

Figure 1: SPI Slave Interface with Single Port RAM Wrapper

1.2 SPI Slave Interface

1.2.1 Block Diagram

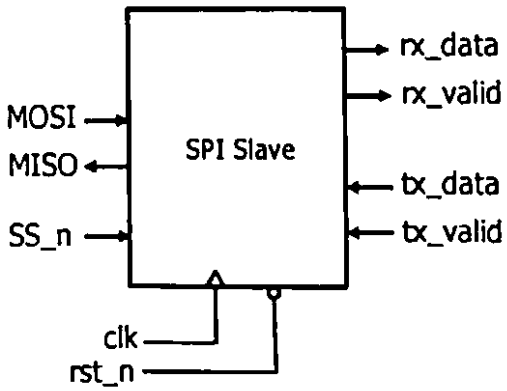

Figure 2: SPI Slave Interface Block Diagram

1.2.2 Ports

Name	Type	Size	Description
MOSI	Input	1 bit	Master output slave input signal
SS_n		1 bit	Active low Slave select signal
clk		1 bit	Clock Signal
a_rst_n		1 bit	Active low asynchronous reset signal
tx_data		8 bits	Transmitted data required from the RAM to the Master
tx_valid		1 bit	HIGH only when the data is ready to be received from RAM
rx_data	Output	10 bits	Received data from the Master converted from serial into parallel to the RAM
rx_valid		1 bit	HIGH only when the data is ready to be sent to RAM
MISO		1 bit	Master input slave output signal

1.3 Single Port Synchronous RAM

1.3.1 Block Diagram

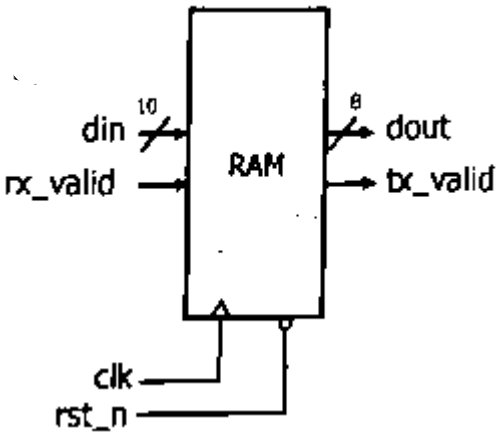

Figure 3: Single Port Synchronous RAM Block Diagram

1.3.2 Parameters

- MEM_DEPTH, Default: 256
- ADDR_SIZE, Default: 8

1.3.3 Ports

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

Din[9:8] selects the mode for Read/ Write on the single port asynchronous RAM

din[9:8]	Command	Description
00	Write	Hold din[7:0] internally as write address
01		Write din[7:0] in the memory with write address held previously
10	Read	Hold din[7:0] internally as read address
11		Read the memory with read address held previously, tx_valid should be HIGH, dout holds the word read from the memory, ignore din[7:0].

1.4 SPI Slave FSM Transitions

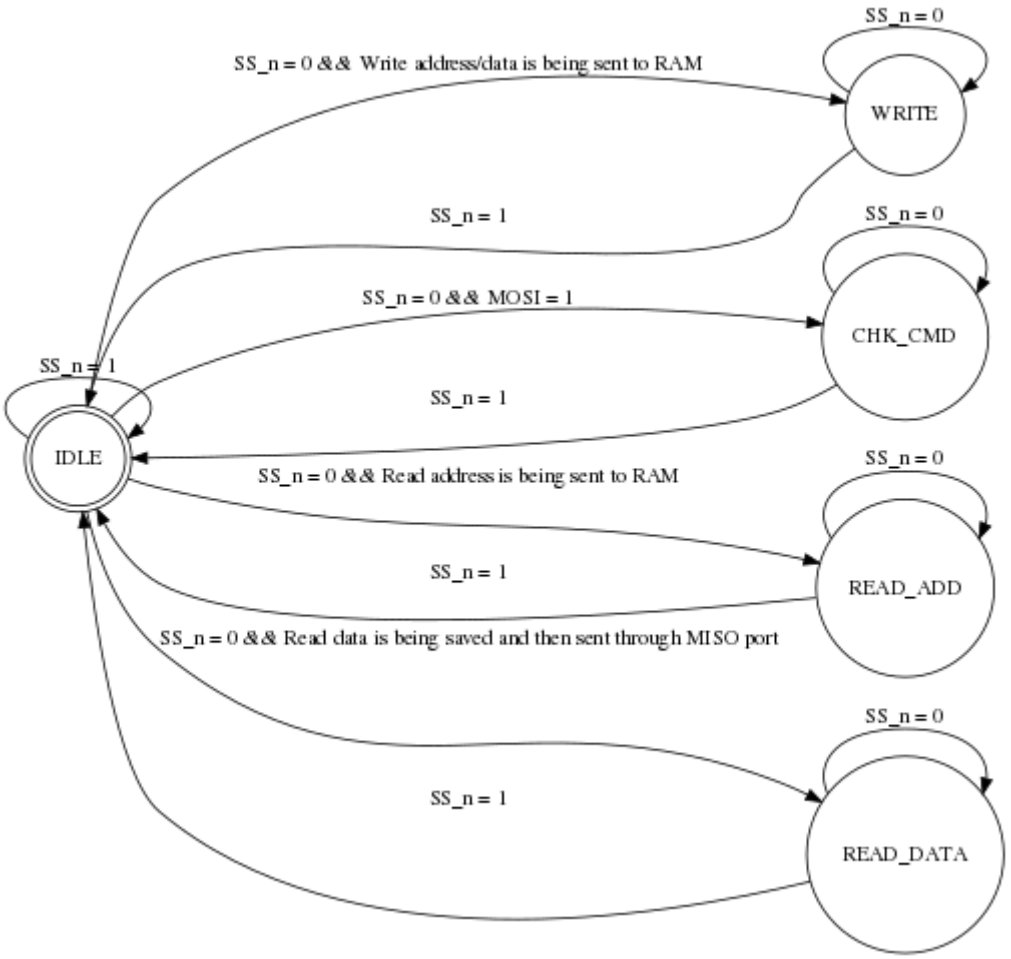

Figure 4: SPI Slave State Transitions - Coded using Graphviz's DOT language

2 Design Flow Code

2.1 Single Port Synchronous RAM Code

```
module Single_Port_Synchronous_RAM #(
    /* ---------------------- Design Parameters ---------------------- */
    /* Width of the word in memory */
    parameter MEM_WIDTH = 8,

    /* Depth of the memory (No of words in memory) */
    parameter MEM_DEPTH = 256,

    /* Address of the location in memory
       (calculated by HDL compiler using $clog2(..)) */
    parameter MEM_ADDR_WIDTH = $clog2(MEM_DEPTH)

) (
    /* ---------------------- Input Ports ---------------------- */
    /*  [9:8] MODE SELECTION
        ------------------------------------------------------------
        00   ==>       ==> Hold din[7:0] internally as write address
                 Write
        01   ==>       ==> Write din[7:0] in the memory with write
                           address held previously
        ------------------------------------------------------------
        10   ==>       ==> Hold din[7:0] internally as read address
                 Read
        11   ==>       ==> Read the memory with read address held
                           previously, tx_valid should be HIGH,
                           dout holds the word read from the memory
                           ignore din[7:0]
        ------------------------------------------------------------ */
    input [MEM_WIDTH+1:0] din,

    /* Clock Signal */
    input clk,

    /* Active Low asynchronous reset */
    input a_rst_n,

    /* HIGH ==> Accept din[7:0] to save the R/W address internally
                OR
                Write a memory word depending on the 2 MSBs din[9:8] */
    input rx_valid,

    /*  ---------------------- Output Ports  ---------------------- */
    /* Data Output */
    output reg [MEM_WIDTH-1:0] dout,

    /* Whenever the command is memory read the tx_valid is HIGH */
    output reg tx_valid
);
```

```
    /* Create the RAM block */
    reg [MEM_WIDTH-1:0] RAM [MEM_DEPTH-1:0];

    /* Register holder for R/W addresses */
    reg [MEM_ADDR_WIDTH-1:0] Addr_rd, Addr_wr;

    /* Controller for reset RAM for loop */
    integer i;

    /* Single port Synchronous RAM logic */
    always @(posedge clk or negedge a_rst_n) begin
        if(~a_rst_n) begin
            /* Initialize all RAM by zeroes */
            for (i = 0; i<MEM_DEPTH; i = i + 1) begin
                RAM[i] <= {MEM_WIDTH{1'b0}};
            end

            /* Give the outputs zero value */
            dout <= 0;
            tx_valid <= 0;
        end
        else begin
            case(din[9:8])
                /* Write Functionality */
                2'b00: begin
                    if(rx_valid)
                        Addr_wr <= din[7:0];

                end
                2'b01: begin
                    if(rx_valid)
                        RAM[Addr_wr] <= din[7:0];

                end

                /* Read Functionality */
                2'b10: begin
                    Addr_rd <= din[7:0];
                end
                2'b11: begin
                    dout <= RAM[Addr_rd];
                end
            endcase

            /* assign tx_valid value according to Opcode in din */
            if(din[9:8] == 2'b11)
                tx_valid <= 1;
            else
                tx_valid <= 0;
        end
    end
endmodule
```

2.2 SPI Slave Interface Code

```
module SPI_Slave_Interface (
    /* ---------------------- Input Ports ---------------------- */
    /* Input from Master (Master-out-Slave-in) */
    input MOSI,

    /* Activating the slave communication
        "0" ==> Start the communication
        "1" ==> End Communication */
    input SS_n,

    /* Clock Signal */
    input clk,

    /* Active Low asynchronous reset */
    input a_rst_n,

    /* Transmitted Data from RAM */
    input [7:0] tx_data,

    /* Signal for validity of tx */
    input tx_valid,

    /*  ---------------------- Output Ports  ---------------------- */
    /* Input to Master (Master-in-Slave-out) */
    output reg MISO,

    /* Output to the data to be written in RAM */
    output reg [9:0] rx_data,

    /* Signal for validity of rx */
    output reg rx_valid
);
    /*  --------------------- Internal Signals  ------------------- */
    /* Flag if read address done before read data */
    reg Check_READ_ADD_flag;

    /* Counter for Converting Serial to Parallel to tx_data port
                  Converting Parallel to Series from rx_data port */
    integer Counter = 0;

    /* Storage register for the data used in conversion of Serial to
       Parallel or vice versa */
    reg [9:0] mid_data;

    /* Signal for knowing if it is first time to enter READ_DATA */
    reg READ_DATA_First_time;

    /*  ---------------------- FSM States  ----------------------- */
    /* Defining FSM States Parameters */
    localparam IDLE = 3'b000;
```

```
    localparam CHK_CMD = 3'b001;
    localparam WRITE = 3'b010;
    localparam READ_ADD = 3'b011;
    localparam READ_DATA = 3'b100;

    /* Register for Next and Current States */
    reg [2:0] NS,CS;

    /*  -------------------- Next State Logic  -------------------- */
    always @(CS,MOSI,SS_n) begin
        case (CS)
            IDLE:
                begin
                    if(~a_rst_n)
                        NS = IDLE;
                    else if(SS_n)
                        NS = IDLE;
                    else if(~SS_n)
                        NS = CHK_CMD;
                    else
                        NS = IDLE;
                end
            CHK_CMD:
                begin
                    if(SS_n)
                        NS = IDLE;
                    else if((SS_n == 0) && (MOSI == 0))
                        NS = WRITE;
                    else if((SS_n == 0) && (MOSI == 1)) begin
                        /* Check if Read Address is done first or not */
                        if(Check_READ_ADD_flag) begin
                            NS = READ_DATA;
                        end
                        else begin
                            NS = READ_ADD;
                        end
                    end
                    else
                        NS = CHK_CMD;
                end
            WRITE:
                begin
                    if(SS_n)
                        NS = IDLE;
                    else
                        NS = WRITE;
                end
            READ_ADD:
                begin
                    if(SS_n)
                        NS = IDLE;
                    else
```

```
                    NS = READ_ADD;
            end
        READ_DATA:
            begin
                if(SS_n)
                    NS = IDLE;
                else
                    NS = READ_DATA;
            end
        default:
            begin
                NS = IDLE;
            end
    endcase
end

/*  ---------------------- State Memory  ---------------------- */
always @(posedge clk or negedge a_rst_n) begin
    if(~a_rst_n) begin
        /* Reset Current State */
        CS <= IDLE;

        /* Reset Internal Signals (State Controllers) */
        Check_READ_ADD_flag <= 0;
        Counter <= 0;
        mid_data <= 0;
        READ_DATA_First_time <= 1;
    end
    else begin
        /* Assign the next state in the Current state */
        CS <= NS;
    end
end

/*  ---------------------- Output Logic  ---------------------- */
always @(posedge clk or negedge a_rst_n) begin
    if(~a_rst_n) begin
        /* Reset Outputs */
        MISO <= 0;
        rx_data <= 0;
        rx_valid <= 0;
        Counter <= 0;
        READ_DATA_First_time <= 1;
    end
    else begin
        /* Assign Outputs */
        case(CS)
        IDLE:
            begin
                /* Reset Outputs */
                MISO <= 0;
                rx_data <= 0;
```

```
                rx_valid <= 0;
                Counter <= 0;
                READ_DATA_First_time <= 1;
            end
        CHK_CMD:

            begin
                /* Reset Outputs */
                MISO <= 0;
                rx_data <= 0;
                rx_valid <= 0;
            end

        WRITE:

            begin
                if(Counter < 9) begin
                    /* Receives Serial Data in Mid-data register */
                    mid_data <= (mid_data << 1) + MOSI;
                    Counter <= Counter + 1;
                end
                else begin
                    /* Completes receiving and send the mid-data
                        to RAM to be stored (either address or
                        data as the RAM will detect the behavior
                        according to rx_data 2 MSBs )  */
                    rx_data <= (mid_data << 1) + MOSI;
                    rx_valid <= 1;
                    Counter <= 0;
                    mid_data <= 0;
                end
            end

        READ_ADD:

            begin
                if(Counter < 9) begin
                    /* Receives Serial Address in Mid-data register */
                    mid_data <= (mid_data << 1) + MOSI;
                    Counter <= Counter + 1;
                end
                else begin
                    /* Completes receiving and send the mid-data
                        to RAM to detect which Address will be
                        read */
                    rx_data <= (mid_data << 1) + MOSI;
                    rx_valid <= 1;
                    Check_READ_ADD_flag <= 1;
                    Counter <= 0;
                    mid_data <= 0;
                end
            end
```

```
            READ_DATA:
                begin
                    /* Read instruction completely */
                    if((Counter < 9) && READ_DATA_First_time) begin
                        mid_data <= (mid_data << 1) + MOSI;
                        Counter <= Counter + 1;
                    end
                    else
                        /* Completes receiving and send the mid-data
                            to RAM */
                        if((Counter == 9) && READ_DATA_First_time) begin
                            rx_data <= (mid_data << 1) + MOSI;
                            rx_valid <= 1;
                            Counter <= 0;
                            READ_DATA_First_time <= 0;
                        end
                    else
                        if(tx_valid && Check_READ_ADD_flag && (~READ_DATA_First_time))
begin
                            /* Check if address is sent or not to the RAM
                            for successful operation */

                            if(Counter < 8) begin
                                /* Converts Parallel data to Serial to be
                                    sent to Master */
                                MISO <= tx_data [Counter];
                                Counter <= Counter + 1;
                            end
                            else begin
                                /* Completes sending successfully and resets
                                the address flag and counter */
                                Check_READ_ADD_flag <= 0;
                                READ_DATA_First_time <= 1;
                                Counter <= 0;
                                mid_data <= 0;
                            end
                        end
                end
            default:
                begin
                    /* To avoid any other invalid CS will reset outputs
                       without changing any related internal signals to
                       be able to continue functionality again */
                    MISO <= 0;
                    rx_data <= 0;
                    rx_valid <= 0;
                end
            endcase
        end
    end

endmodule
```

2.3 Top Module Code

```
module SPI_Top_module(
    /* ---------------------- Input Ports ---------------------- */
    /* Input from Master (Master-out-Slave-in) */
    input MOSI,

    /* Activating the slave communication */
    input SS_n,

    /* Clock Signal */
    input clk,

    /* Active Low asynchronous reset */
    input a_rst_n,

    /*  ---------------------- Output Ports  ---------------------- */
    /* Input to Master (Master-in-Slave-out) */
    output MISO
);
    /*  --------------------- Internal Signals  ------------------- */
    /* Receiving Data to RAM */
    wire [9:0] rx_data;
    wire rx_valid;

    /* Transmitting Data to RAM */
    wire [7:0] tx_data;
    wire tx_valid;

    /*  -------------------- Modules Instantiation  ---------------- */
    /* SPI Slave Module */
    SPI_Slave_Interface SPI (
        .MISO(MISO),
        .MOSI(MOSI),
        .SS_n(SS_n),
        .clk(clk),
        .a_rst_n(a_rst_n),
        .rx_data(rx_data),
        .rx_valid(rx_valid),
        .tx_data(tx_data),
        .tx_valid(tx_valid)
    );
    /* RAM Module */
    Single_Port_Synchronous_RAM RAM(
        .clk(clk),
        .a_rst_n(a_rst_n),
        .din(rx_data),
        .rx_valid(rx_valid),
        .dout(tx_data),
        .tx_valid(tx_valid)
    );
endmodule
```

3 Top Module Testbench Code

```
module SPI_Top_module_tb();
    /* ---------------------- Input Ports ---------------------- */
    /* Input from Master (Master-out-Slave-in) */
    reg MOSI;

    /* Activating the slave communication */
    reg SS_n;

    /* Clock Signal */
    reg clk;

    /* Active Low asynchronous reset */
    reg a_rst_n;

    /*  ---------------------- Output Ports  ---------------------- */
    /* Input to Master (Master-in-Slave-out) */
    wire MISO;

    /*  --------------------- Internal Signal  --------------------- */
    /* Register to hold data input to module */
    reg [9:0] Input_Data_Address;

    /* Register to hold data output from module */
    reg [7:0] Data_module;

    /* Controller for the for loops */
    integer i;

    /*  ------------------- Module Instantiation  -------------------- */
    SPI_Top_module DUT (
        .MOSI(MOSI),
        .SS_n(SS_n),
        .clk(clk),
        .a_rst_n(a_rst_n),
        .MISO(MISO)
        );

    /* ---------------------- Clock Generation ----------------------*/
    initial begin
        clk = 0;
        forever begin
            #20;                    // 20 ns period => 50 MHz frequency
            clk = ~clk;
        end
    end

    /* ------------------- Testbench Test Cases --------------------*/
    initial begin
        $display("START THE SIMULATION");
```

```
        /* Test Case 1: Check Reset Functionality */
        $display("Test Case 1: Check Reset Functionality");
        a_rst_n = 0;            // Active Low Reset
        repeat(3) @(negedge clk);
        self_checking_task(MISO, 0);
        a_rst_n = 1;            // Release Reset

        /* Test Case 2: Slave is not selected */
        $display("TEST CASE 2: Slave is not selected");
        SS_n = 1;               // Slave not selected
        repeat(3) @(posedge clk);
        self_checking_task(MISO, 0);

        /* Test Case 3: Send Write address and Data in this address */
        $display("TEST CASE 3: Send Write address and Data in this address ");
        SS_n = 1;               // Slave not selected
        MOSI = 0;
        @(negedge clk);

        SS_n = 0;               // Slave selected
        @(negedge clk);

        /*     "00"     ==> Write Address Command
            "1010_1100" ==>   Address Selected  (AC)  */
        Input_Data_Address = 10'b00_1010_1100;

        for(i=0; i<10; i=i+1) begin
            @(negedge clk);
            MOSI = Input_Data_Address[9-i];
        end

        @(negedge clk);         // Ensure data is stable
        MOSI = 0;               // Clear MOSI
        @(negedge clk);         // Hold SS_n low for one more clock cycle
        SS_n = 1;               // Stop communication
        repeat(3) @(negedge clk);

        SS_n = 0;               // Slave selected
        @(negedge clk);

        /*     "01"     ==> Write Data Command
            "1110_1110" ==>  Data Added     (EE)    */
        Input_Data_Address = 10'b01_1110_1110;

        for(i=0; i<10; i=i+1) begin
            @(negedge clk);
            MOSI = Input_Data_Address[9-i];
        end

        @(negedge clk);         // Ensure data is stable
        MOSI = 0;               // Clear MOSI
        @(negedge clk);         // Hold SS_n low for one more clock cycle
```

```
        SS_n = 1;              // Stop communication
        repeat(3) @(negedge clk);

        $display("Check address 'hAC and data '1110_1110' written in it in RAM");
        // $stop;

        /* TEST CASE 4: Send Read address and Read Data in this address */
        $display("TEST CASE 4: Send Read address and Read Data in this address ");

        /*     "10"     ==> Read Address Command
           "1010_1100"  ==>   Address Selected    */
        Input_Data_Address = 10'b10_1010_1100;

        SS_n = 0;
        @(negedge clk);
        MOSI = Input_Data_Address[9];
        @(negedge clk); // More delay for processing

        for(i=1; i<10; i=i+1) begin
            @(negedge clk);
            MOSI = Input_Data_Address[9-i];
        end

        @(negedge clk);        // Ensure data is stable
        MOSI = 0;               // Clear MOSI
        @(negedge clk);         // Hold SS_n low for one more clock cycle
        SS_n = 1;               //Stop communication
        repeat(3) @(negedge clk);


        /*     "11"     ==> Read Data Command
            "1011_1100" ==>   Redundant bits      */
        Input_Data_Address = 10'b11_1011_1100;

        SS_n = 0;               // Slave selected
        @(negedge clk);

        MOSI = Input_Data_Address[9];
        repeat(2) @(negedge clk); // More delay for processing

        for(i=1; i<10; i=i+1) begin
            @(negedge clk);
            MOSI = Input_Data_Address[9-i];
        end

        @(negedge clk);        // Ensure data is stable

        for(i=0; i<8; i=i+1) begin
            @(negedge clk);
            Data_module[i] = MISO;
        end
```

```
        self_checking_8_bit_task(Data_module, 'b1110_1110);

        @(negedge clk);         // Ensure data is stable
        MOSI = 0;               // Clear MOSI
        SS_n = 1;               //Stop communication
        repeat(3) @(negedge clk);

        $display("END THE SIMULATION");
        $stop;
    end

    /* ------------------ Self-Checking 1-bit Task -------------------*/
    task self_checking_task;
        input module_out;
        input tb_required;
        begin
            // Check if the output is correct
            if (module_out == tb_required) begin
                $display("Self-checking task: Output is correct");
                end
            else begin
                $display("Self-checking task: Output is incorrect \n");
                $display("module_out = %b, tb_required = %b", module_out,tb_required);
                $stop;
            end
        end
    endtask

    /* ------------------- Self-Checking 8-bit Task -------------------*/
    task self_checking_8_bit_task;
        input [7:0] module_out;
        input [7:0] tb_required;
        begin
            // Check if the output is correct
            if (module_out == tb_required) begin
                $display("Self-checking task: Output is correct");
                end
            else begin
                $display("Self-checking task: Output is incorrect \n");
                $display("module_out = %b, tb_required = %b", module_out,tb_required);
                $stop;
            end
        end
    endtask
endmodule
```

4 Automation Codes

4.1 Do Questa sim Simulation Code

```
vlib work

# compile the design modules and the top module
vlog Single_Port_Synchronous_RAM.v SPI_Slave_Interface.v SPI_Top_Module.v

# compile the testbench module
vlog SPI_Top_Module_tb.v

# simulate the testbench module
vsim -voptargs="+acc" work.SPI_Top_module_tb

# for the DUT signals
# add wave /DUT/*
# for the internal signals each on its own
# add wave -position insertpoint sim:/SPI_Top_module_tb/DUT/SPI/*
# add wave -position insertpoint sim:/SPI_Top_module_tb/DUT/RAM/*

add wave -position insertpoint \
    sim:/SPI_Top_module_tb/DUT/clk \
    sim:/SPI_Top_module_tb/Input_Data_Address \
    sim:/SPI_Top_module_tb/DUT/MOSI \
    sim:/SPI_Top_module_tb/DUT/SS_n \
    sim:/SPI_Top_module_tb/DUT/a_rst_n \
    sim:/SPI_Top_module_tb/DUT/SPI/NS \
    sim:/SPI_Top_module_tb/DUT/SPI/CS \
    sim:/SPI_Top_module_tb/DUT/MISO \
    sim:/SPI_Top_module_tb/DUT/SPI/mid_data \
    sim:/SPI_Top_module_tb/DUT/SPI/Check_READ_ADD_flag \
    sim:/SPI_Top_module_tb/DUT/rx_data \
    sim:/SPI_Top_module_tb/DUT/rx_valid \
    sim:/SPI_Top_module_tb/DUT/RAM/din \
    sim:/SPI_Top_module_tb/DUT/RAM/Addr_wr \
    sim:/SPI_Top_module_tb/DUT/RAM/Addr_rd \
    sim:/SPI_Top_module_tb/DUT/tx_data \
    sim:/SPI_Top_module_tb/DUT/RAM/dout \
    sim:/SPI_Top_module_tb/DUT/tx_valid \
    sim:/SPI_Top_module_tb/DUT/SPI/Counter \
    sim:/SPI_Top_module_tb/DUT/RAM/i

run -all
# Save the data in the RAM
mem save -o RAM.mem -f mti -data symbolic -addr hex /SPI_Top_module_tb/DUT/RAM/RAM
```

4.2 TCL Vivado Design Flow Automation Code

```
create_project project_6 F:/Electronics/Digital Electronics - KW/SPI
Project/Youssef_Ekramy_Project2 -part xc7a35ticpg236-1L -force

## Add source files & XDC files
add_files Single_Port_Synchronous_RAM.v SPI_Slave_Interface.v SPI_Top_Module.v
Constraint_SPI_Slave_Interface.xdc

## Elaborate Design (Will open the schematic)
synth_design -rtl -top SPI_Top_Module > elab.log

## Save Schematic
write_schematic elaborated_schematic.pdf -format pdf -force

## Synthesize Design
launch_runs synth_1 > synth.log

## open gui (Schematic)
wait_on_run synth_1
open_run synth_1

## Save Schematic
write_schematic synthesized_schematic.pdf -format pdf -force

## Generate netlist
write_verilog -force switch_LEDs_netlist.v

## Implementation
launch_runs impl_1 -to_step write_bitstream

## open gui (Schematic & Device view)
wait_on_run impl_1
open_run impl_1

## Open Hardware Manager
open_hw

## load bitstream to FPGA
connect_hw_server
```

5 Questa Sim Simulation

5.1 Simulation Waveform

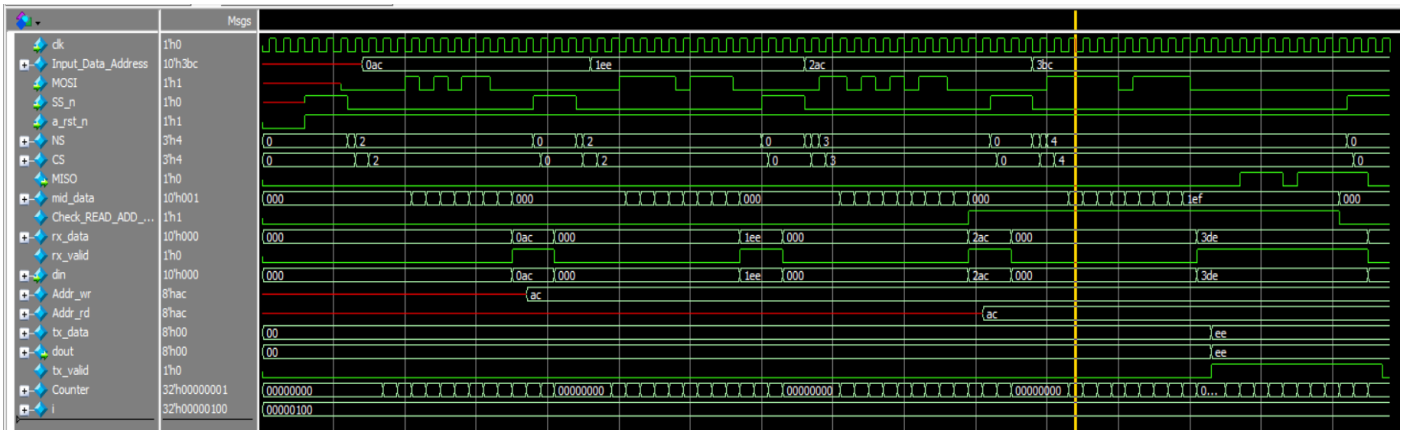

Figure 5: Complete testbench waveform

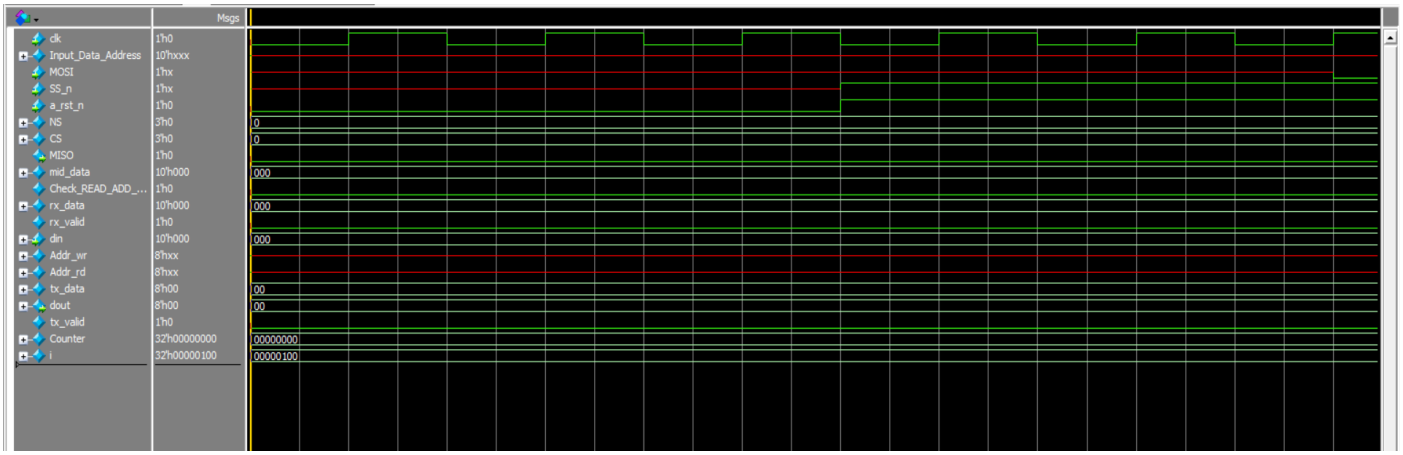

Figure 6: Reset Functionality Test Waveform

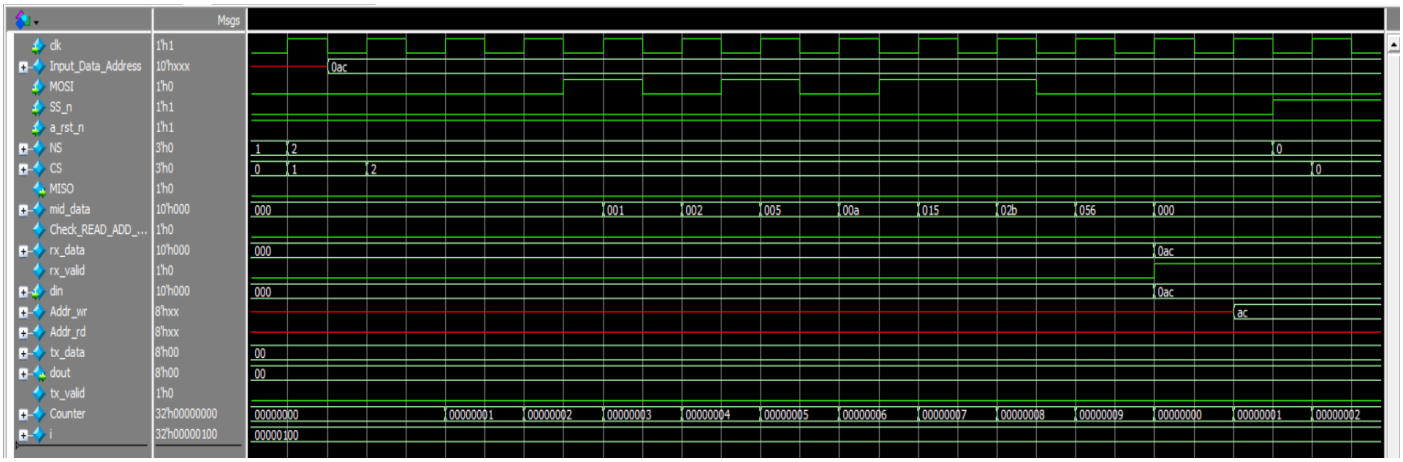

Figure 7: Write Address Functionality Test Waveform

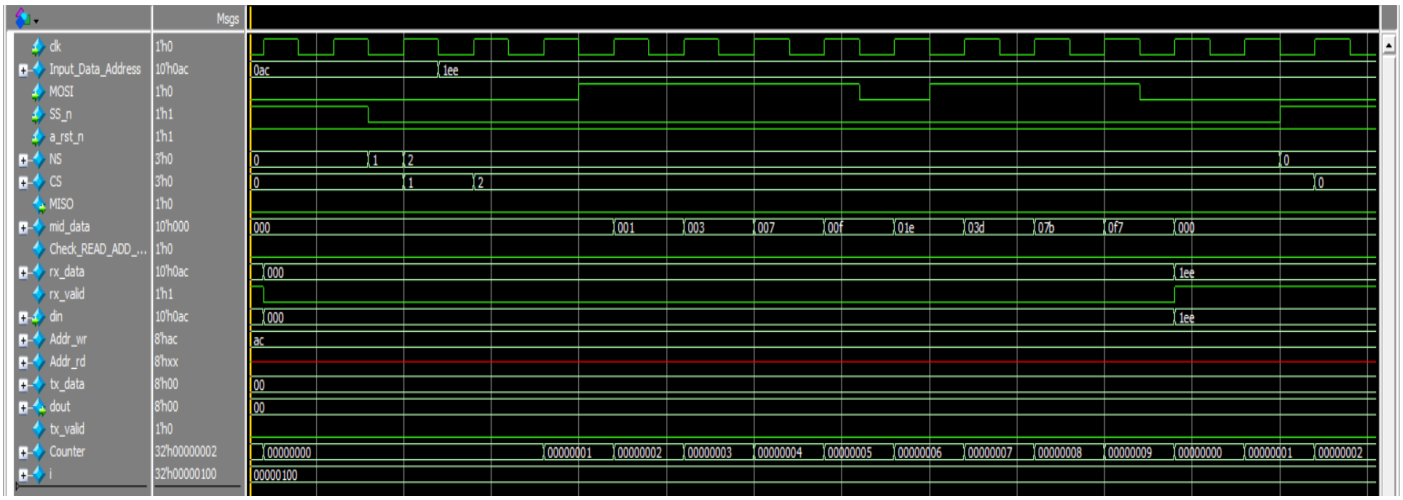

Figure 8: Write Data Functionality Test Waveform

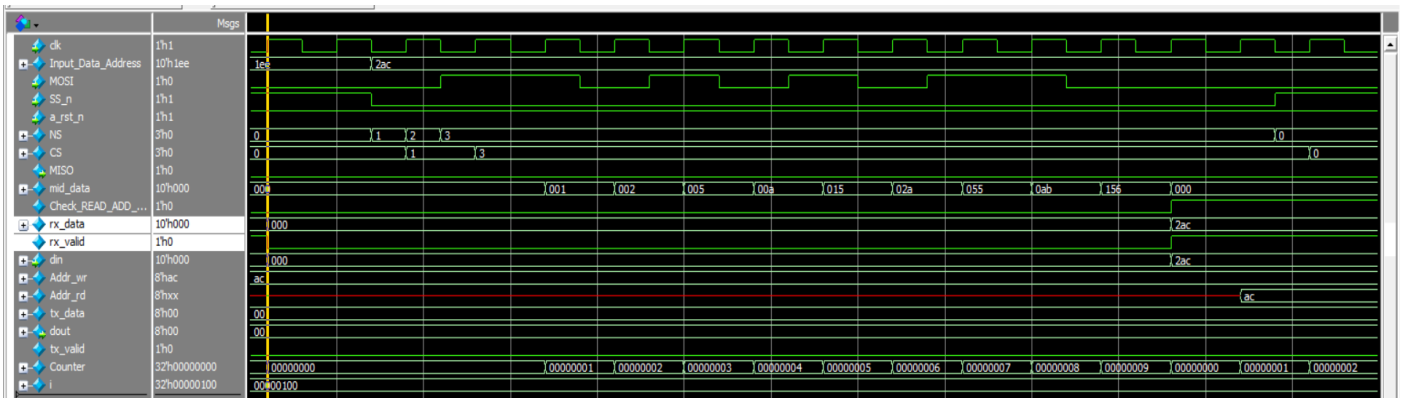

Figure 9: Read Address Functionality Test Waveform

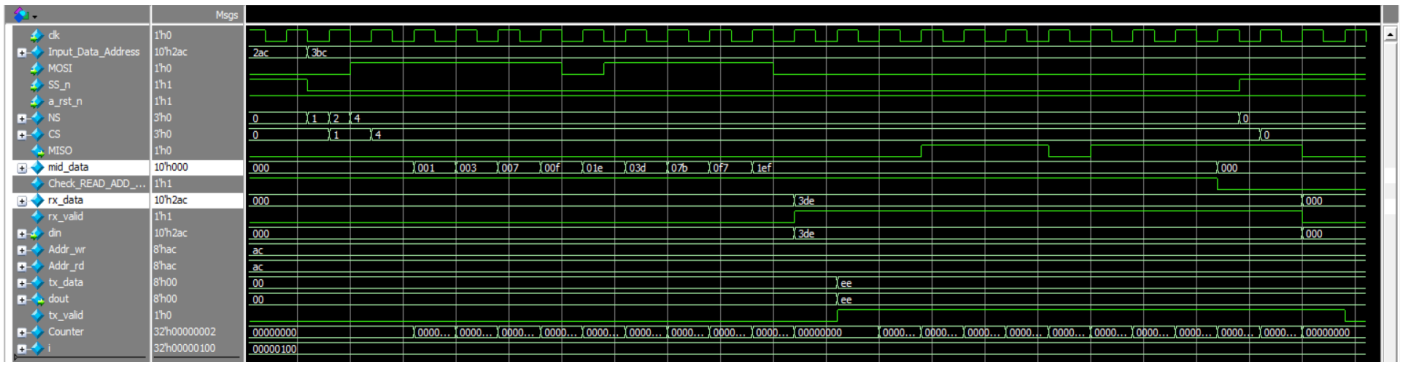

Figure 10: Read Data Functionality Test Waveform

5.2 Simulation RAM file

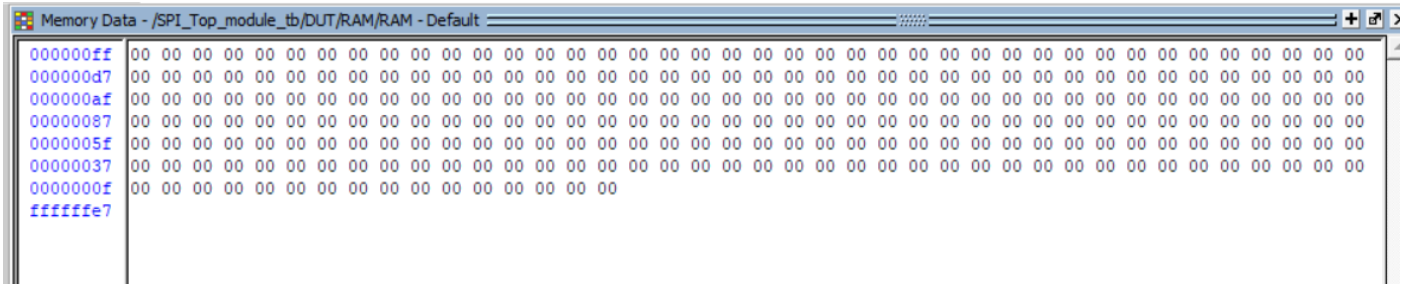

Figure 11: Reset Functionality Test Memory Data

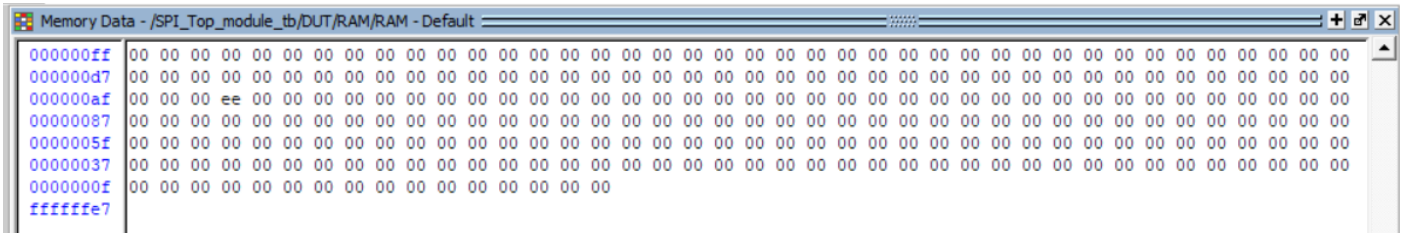

Figure 12: Write Functionality Test Memory Data

6 FPGA Constraint File

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level
signal names in the project

## Clock signal

# w5 PIN CONNECTED TO CLOCK  33 IS THE DEFNITION OF 3.3v PASSED TO PINS
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk]
#add rhe name of clock in design after -name
create_clock -period 10.000 -name clk -waveform {0.000 5.000} -add [get_ports clk]

## Switches
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {a_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]
```

7 Sequential FSM-encoded Design

7.1 Elaboration Schematic

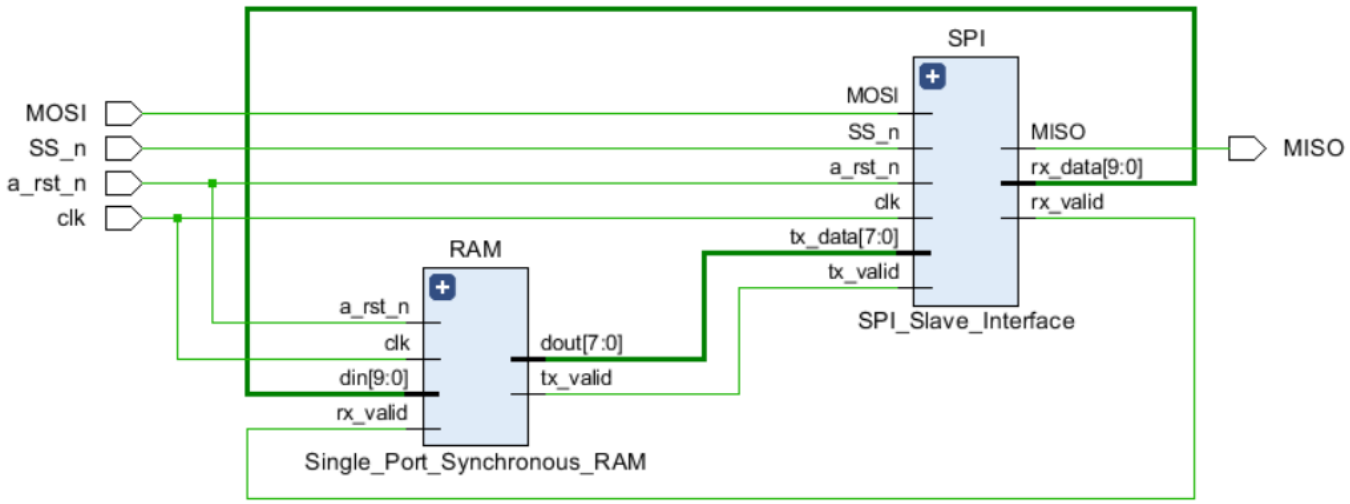

Figure 13: Sequential FSM-encoded Elaboration Schematic

7.2 Synthesis Schematic

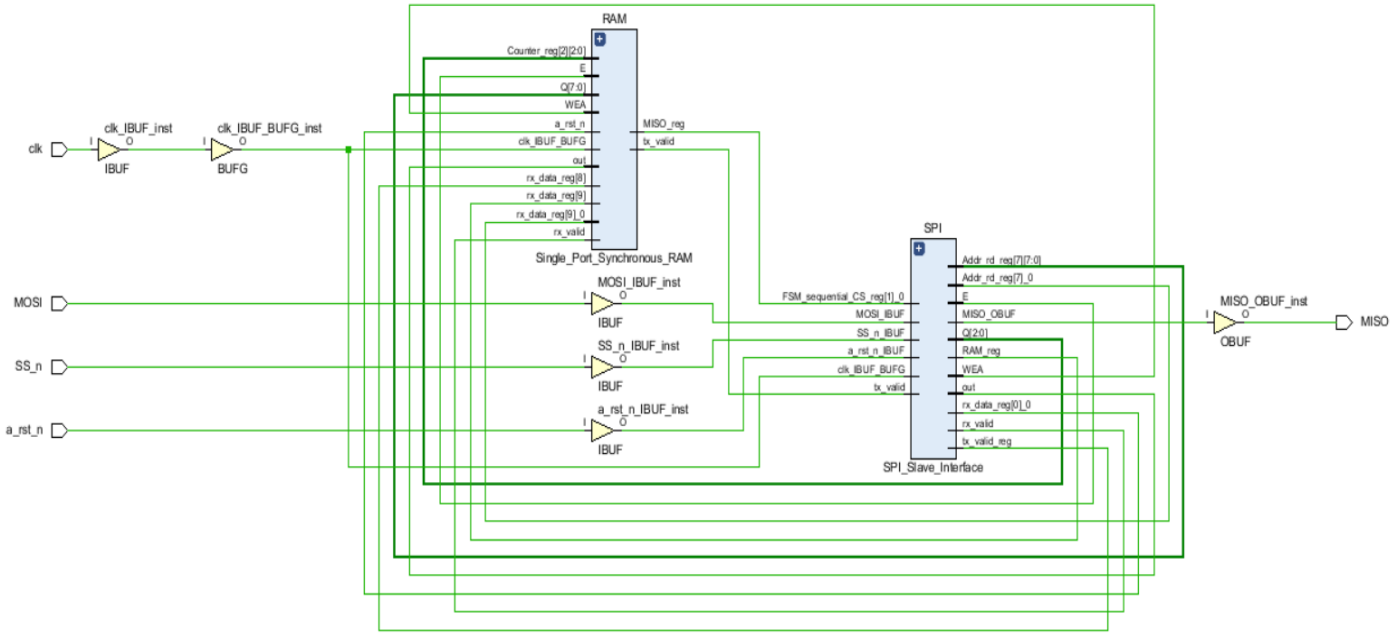

Figure 14: Sequential FSM-encoded Synthesis Schematic

7.3 Synthesis Report

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

```
INFO: [Synth 8-3354] encoded FSM with state register 'CS_reg' using encoding 'sequential' in module 'SPI_Slave_Interface'
```

Figure 15: Sequential FSM-encoded Synthesis Report

7.4 Synthesis Timing Report

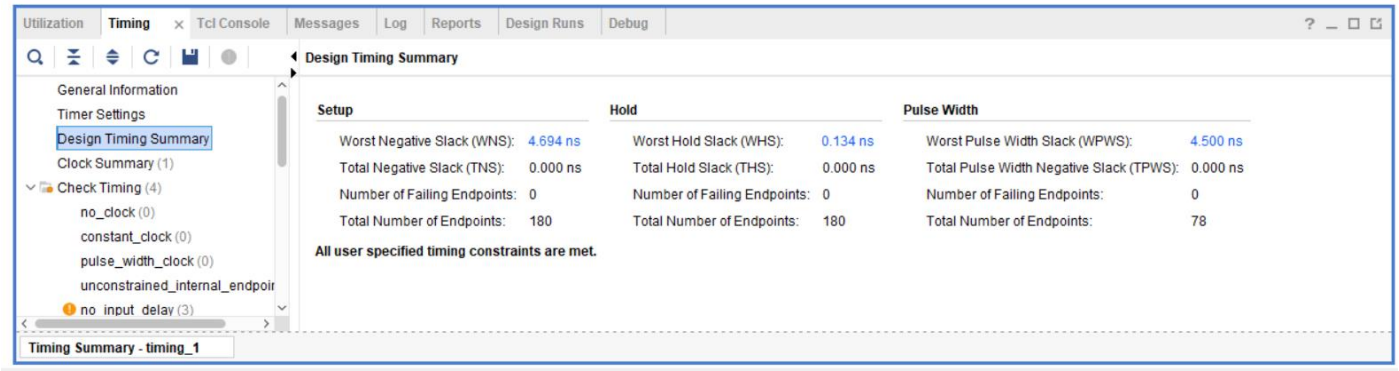

Figure 17: Sequential FSM-encoded Timing Report

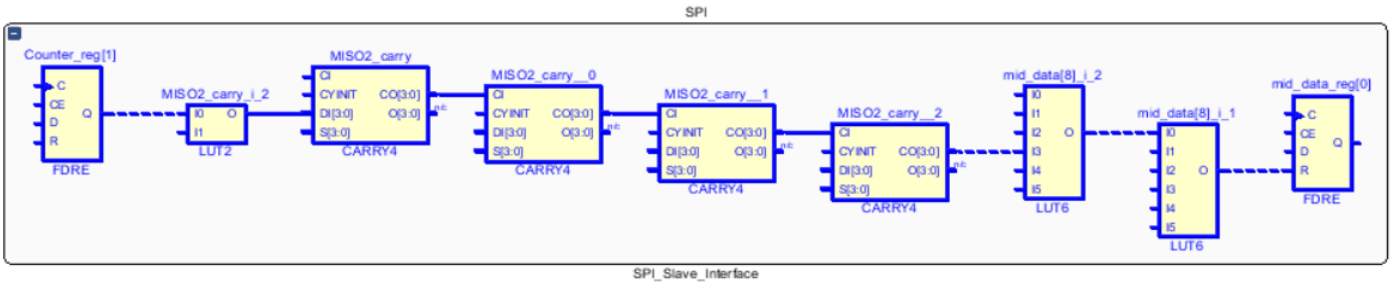

Figure 18: Sequential FSM-encoded Critical Path

7.5 Implementation Device

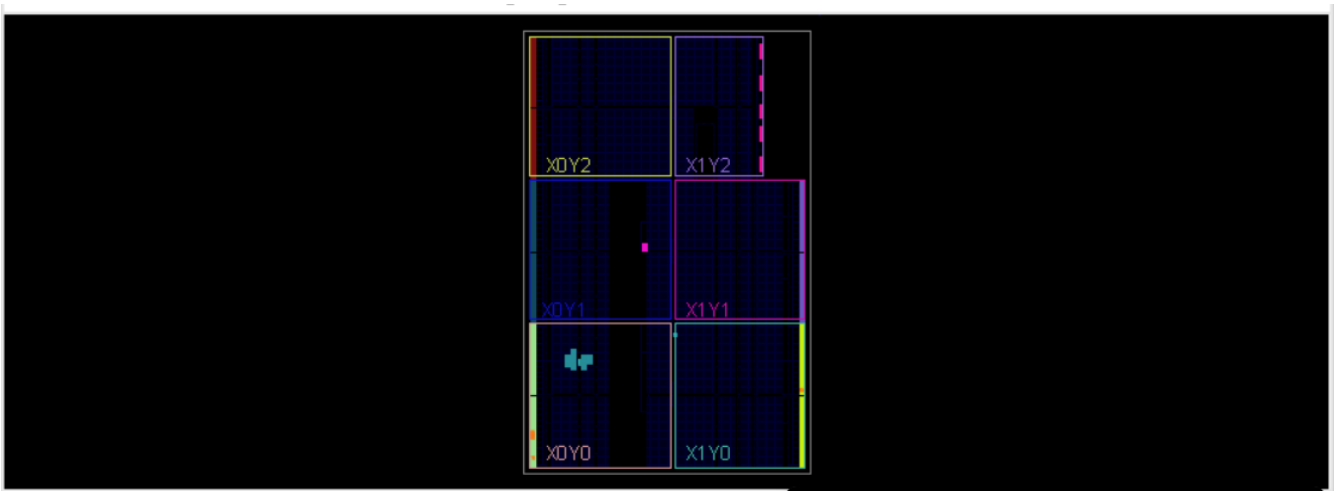

Figure 16: Sequential FSM-encoded Implementation Device

7.6 Implementation Utilization Report

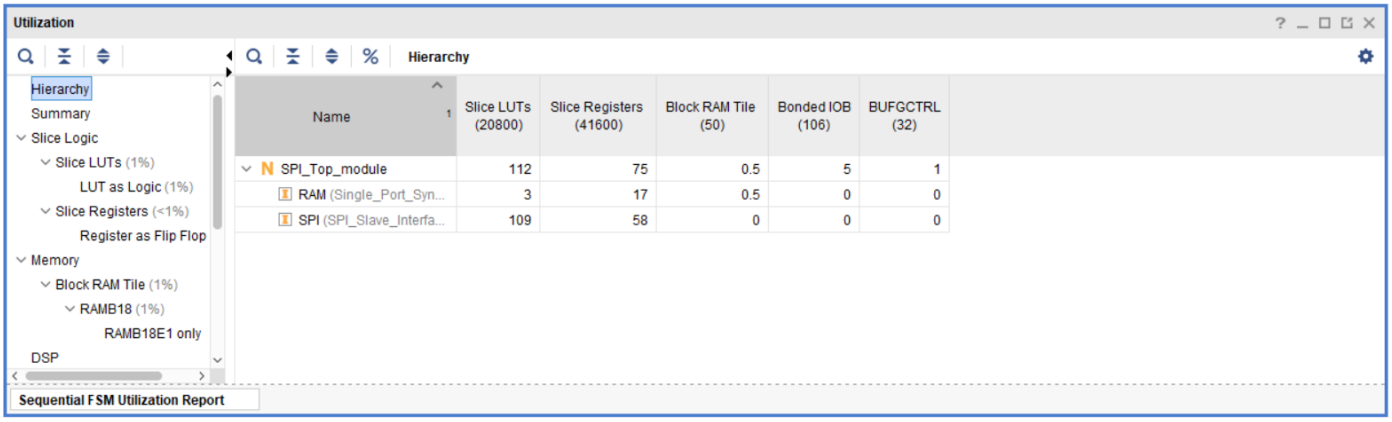

Figure 19: Sequential FSM-encoded Hierarchy Utilization Report

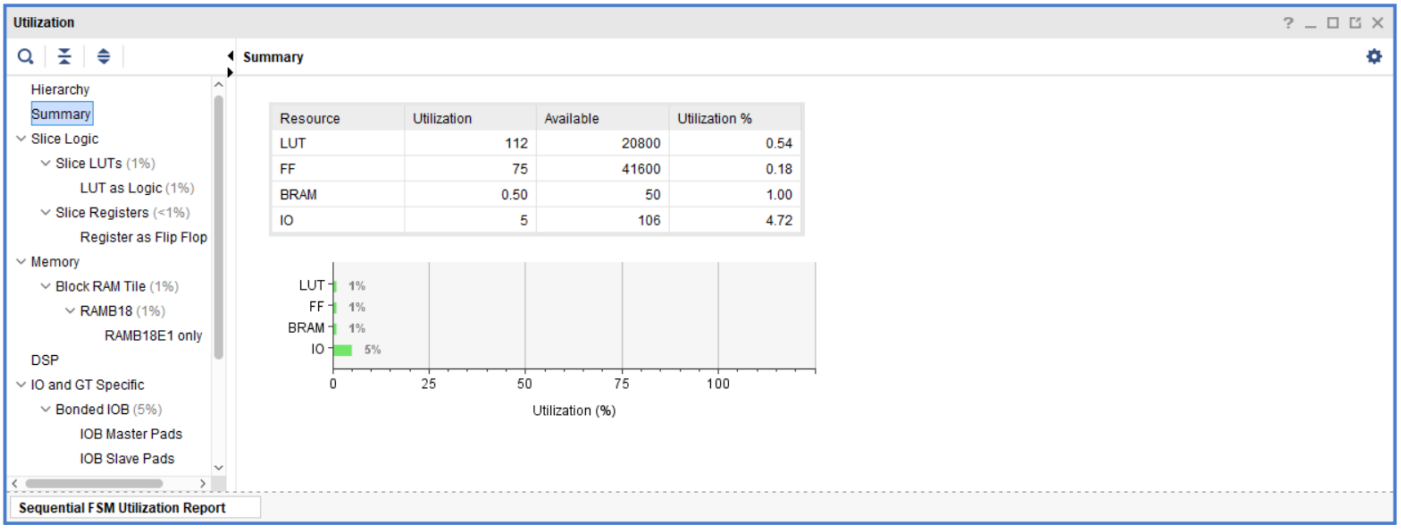

Figure 20: Sequential FSM-encoded Summary Utilization Report

7.7 Implementation Timing Report

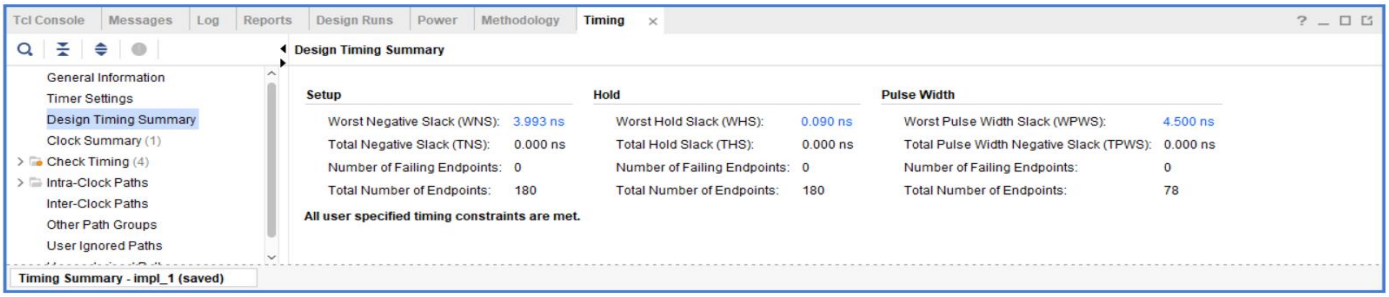

Figure 21: Sequential FSM-encoded Implementation Timing Report

7.8 Messages Tab

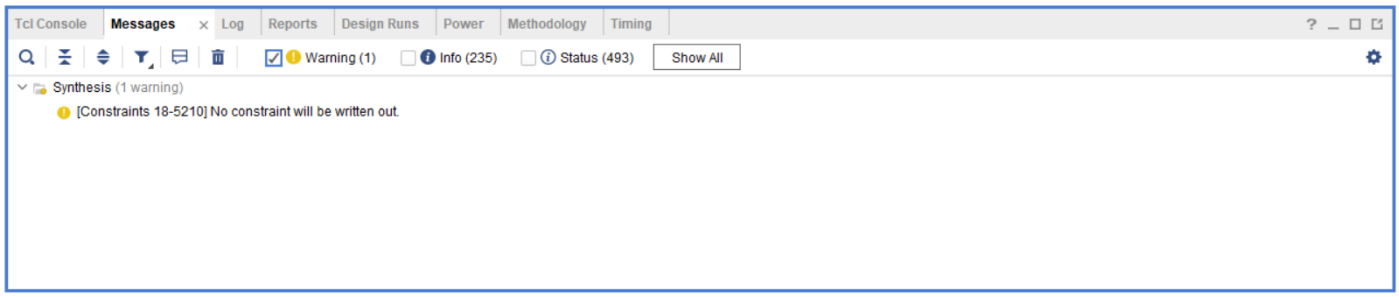

Figure 22: Sequential FSM-encoded Messages Tab

8 One-Hot FSM-encoded Design (BEST Timing)

8.1 Elaboration Schematic

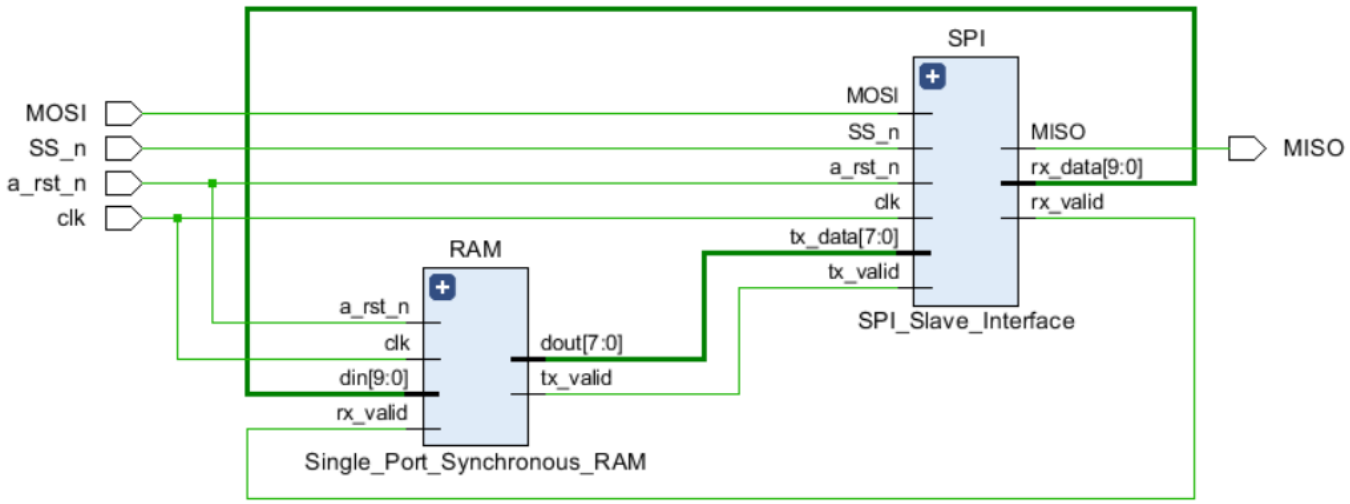

Figure 23: One-Hot FSM-encoded Elaboration Schematic

8.2 Synthesis Schematic

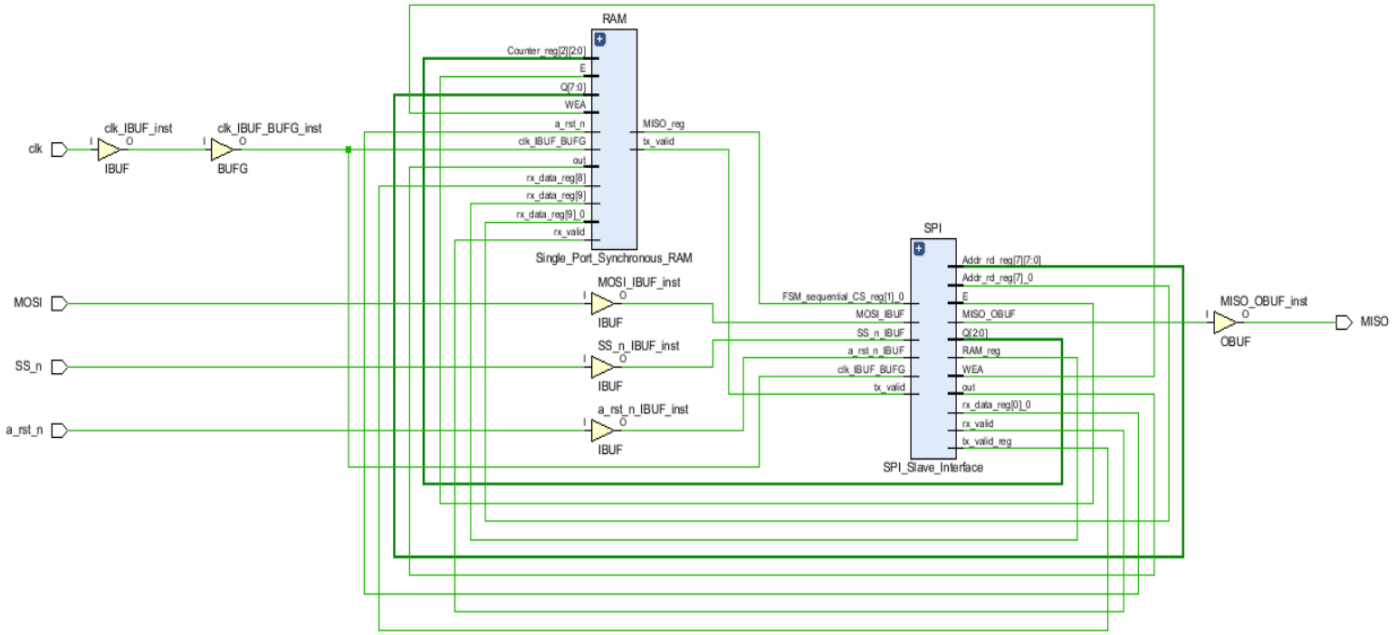

Figure 24: One-Hot FSM-encoded Synthesis Schematic

8.3 Synthesis Report

```
INFO: [Synth 8-6157] synthesizing module 'SPI_Top_module' [F:/Electronics/Digital Electronics - KW/SPI Project/SPI_Top_Module.v:1]
INFO: [Synth 8-6157] synthesizing module 'SPI_Slave_Interface' [F:/Electronics/Digital Electronics - KW/SPI Project/SPI_Slave_Interface.v:1]
    Parameter IDLE bound to: 1 - type: integer
    Parameter CHK_CMD bound to: 2 - type: integer
    Parameter WRITE bound to: 4 - type: integer
    Parameter READ_ADD bound to: 8 - type: integer
    Parameter READ_DATA bound to: 16 - type: integer
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "one-hot" *) [F:/Electronics/Digital Electronics - KW/SPI Project/SPI_Slave_Interface.v:59]
INFO: [Synth 8-6155] done synthesizing module 'SPI_Slave_Interface' (1#1) [F:/Electronics/Digital Electronics - KW/SPI Project/SPI_Slave_Interface.v:1]
INFO: [Synth 8-6157] synthesizing module 'Single_Port_Synchronous_RAM' [F:/Electronics/Digital Electronics - KW/SPI Project/Single_Port_Synchronous_RAM.v:1]
    Parameter MEM_WIDTH bound to: 8 - type: integer
    Parameter MEM_DEPTH bound to: 256 - type: integer
    Parameter MEM_ADDR_WIDTH bound to: 8 - type: integer
INFO: [Synth 8-5534] Detected attribute (* ram_style = "block" *) [F:/Electronics/Digital Electronics - KW/SPI Project/Single_Port_Synchronous_RAM.v:50]
INFO: [Synth 8-6155] done synthesizing module 'Single_Port_Synchronous_RAM' (2#1) [F:/Electronics/Digital Electronics - KW/SPI Project/Single_Port_Synchronous_RAM.v:1]
INFO: [Synth 8-6155] done synthesizing module 'SPI_Top_module' (3#1) [F:/Electronics/Digital Electronics - KW/SPI Project/SPI_Top_Module.v:1]
```

Figure 25: Hot-One FSM-encoded Synthesis Report

```
INFO: [Synth 8-3898] No Re-encoding of one hot register 'CS_reg' in module 'fsm17EC30361100'
```

8.4 Synthesis Timing Report

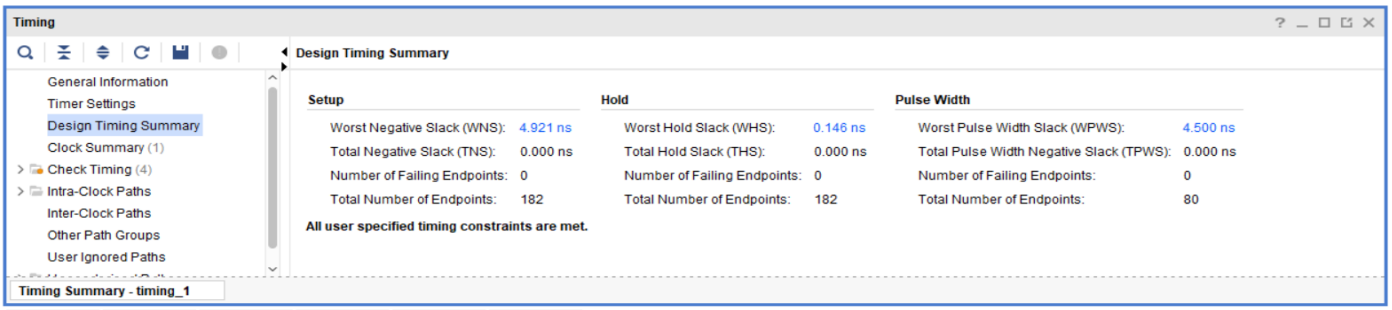

Figure 26: One-Hot FSM-encoded Timing Report

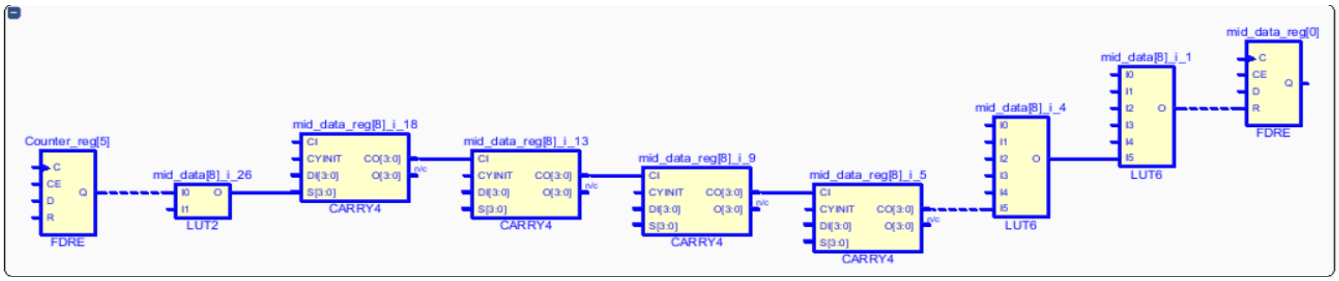

Figure 27: One-Hot FSM-encoded Critical Path

8.5 Implementation Device

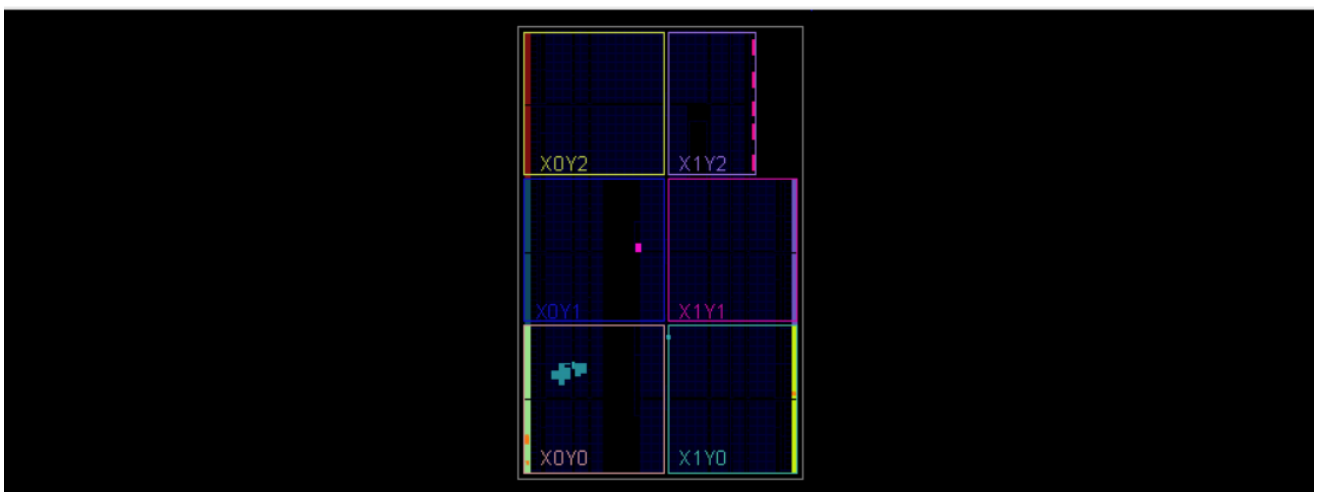

Figure 28: One-Hot FSM-encoded Implementation Device

8.6 Implementation Utilization Report

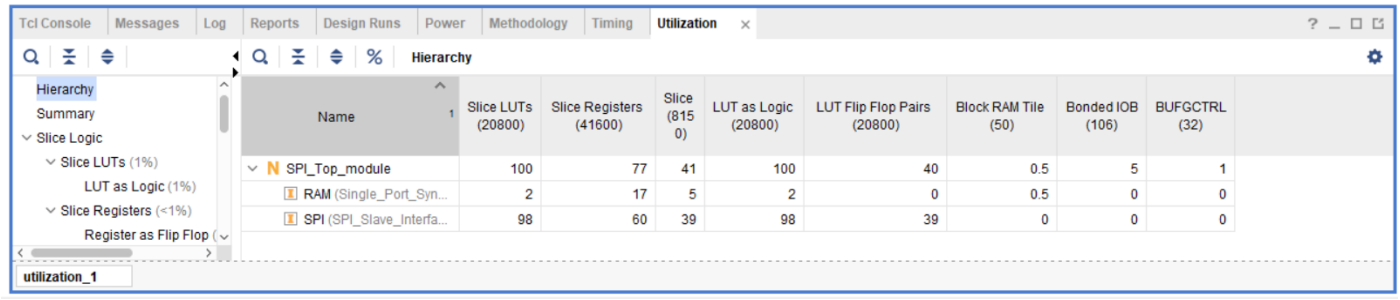

Figure 29: Sequential FSM-encoded Hierarchy Utilization Report

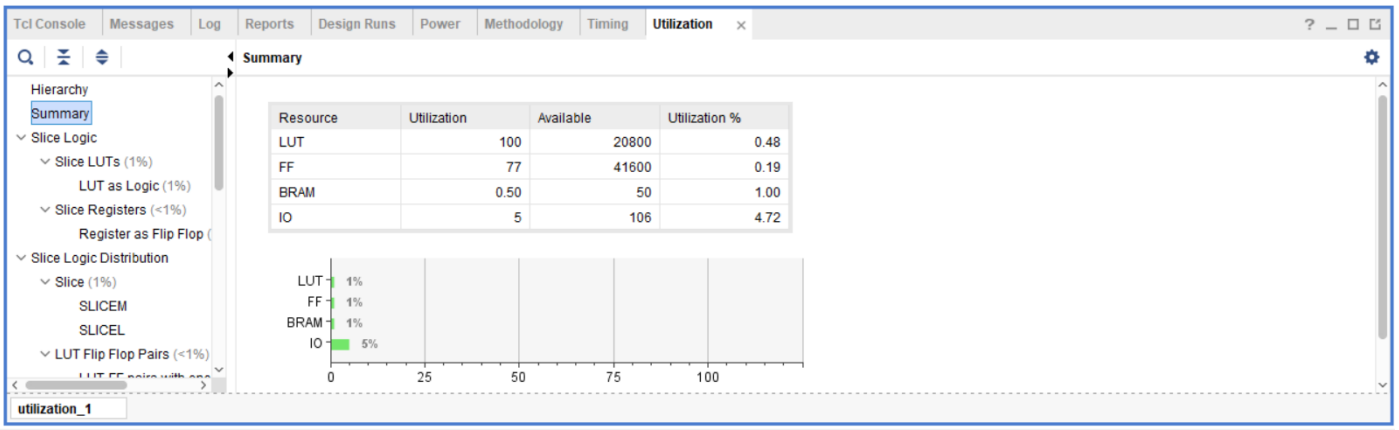

Figure 30: Sequential FSM-encoded Summary Utilization Report

8.7 Implementation Timing Report

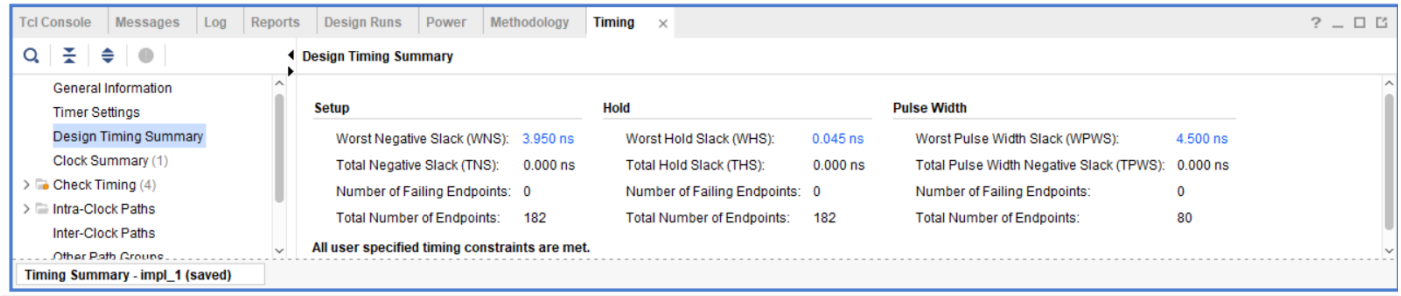

Figure 31: One-Hot FSM-encoded Implementation Timing Report

8.8 Messages Tab

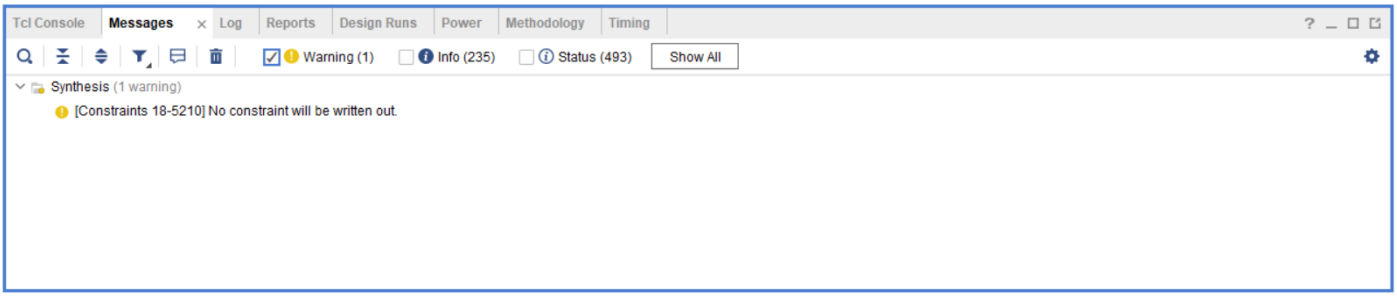

Figure 32: One-Hot FSM-encoded Messages Tab

9 Gray FSM-encoded Design

9.1 Elaboration Schematic

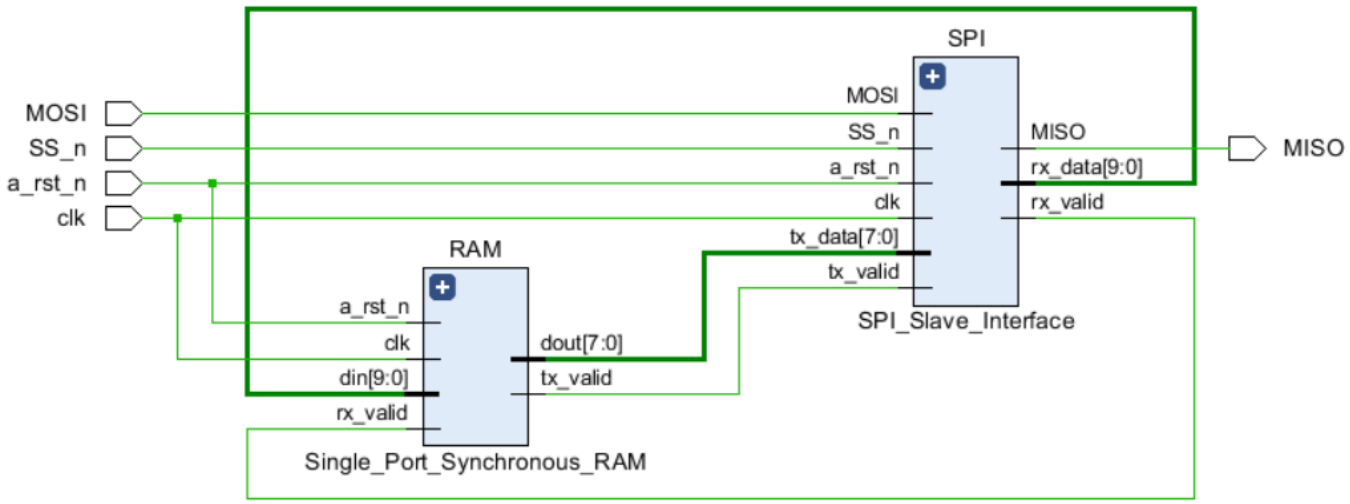

Figure 33: Gray FSM-encoded Elaboration Schematic

9.2 Synthesis Schematic

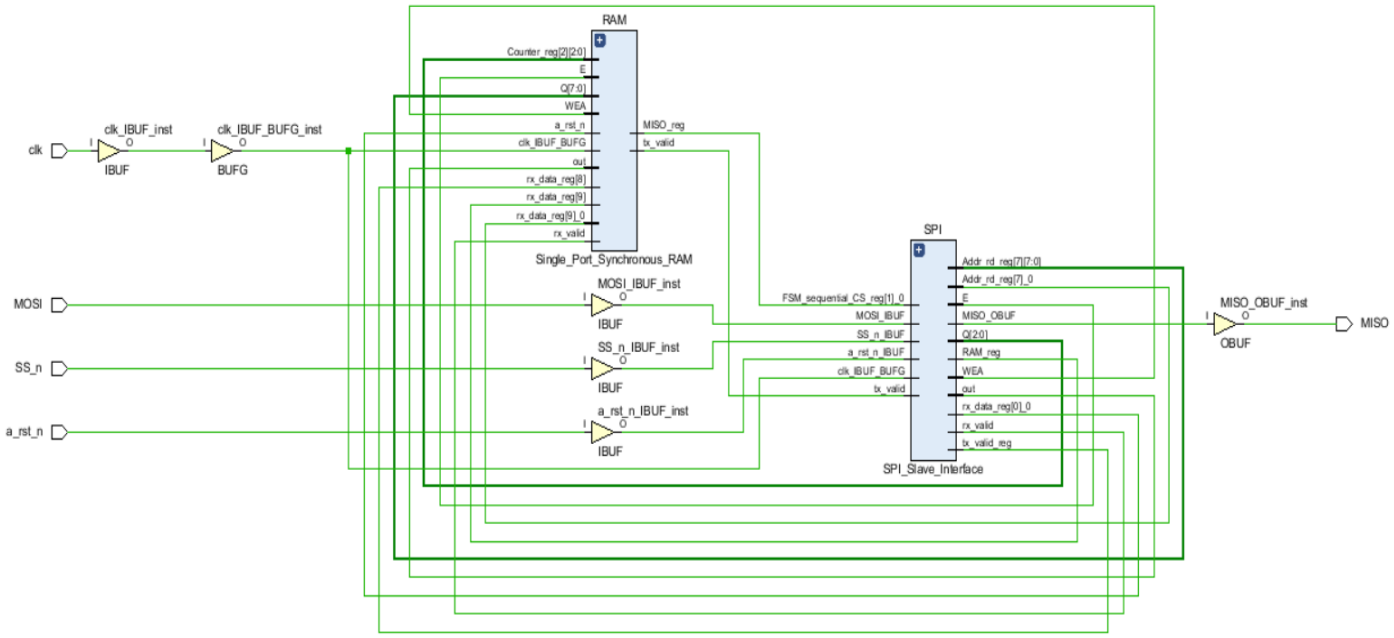

Figure 34: Gray FSM-encoded Synthesis Schematic

9.3 Synthesis Report

```
95  ----------------------------------------------------------------------------------------------------
96  |                   State |                     New Encoding |                Previous Encoding
97  ----------------------------------------------------------------------------------------------------
98  |                    IDLE |                              000 |                             001
99  |                 CHK_CMD |                              001 |                             010
100 |                   WRITE |                              011 |                             100
101 |               READ_DATA |                              010 |                             101
102 |                READ_ADD |                              111 |                             110
103 ----------------------------------------------------------------------------------------------------
104 INFO: [Synth 8-3354] encoded FSM with state register 'CS_reg' using encoding 'gray' in module 'SPI_Slave_Interface'
```

Figure 35: Gray FSM-encoded Synthesis Report

9.4 Synthesis Timing Report

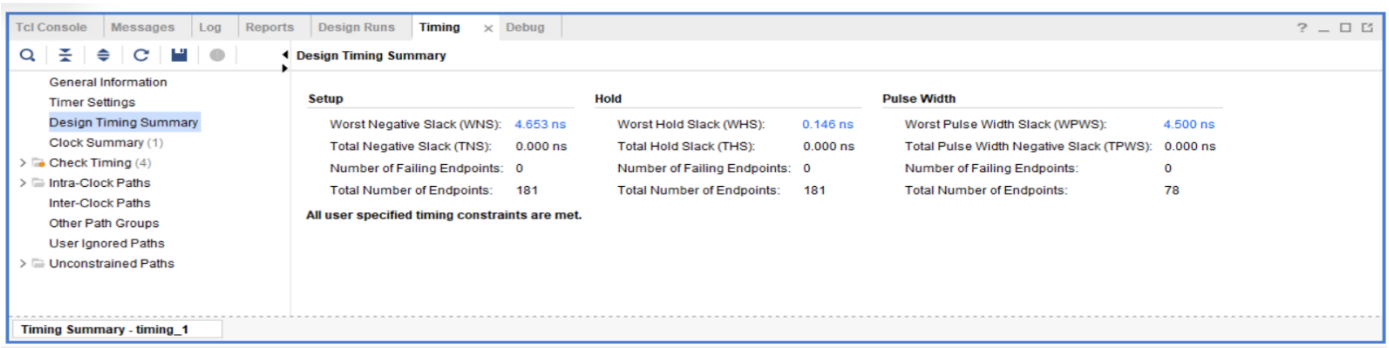

Figure 36: Gray FSM-encoded Timing Report

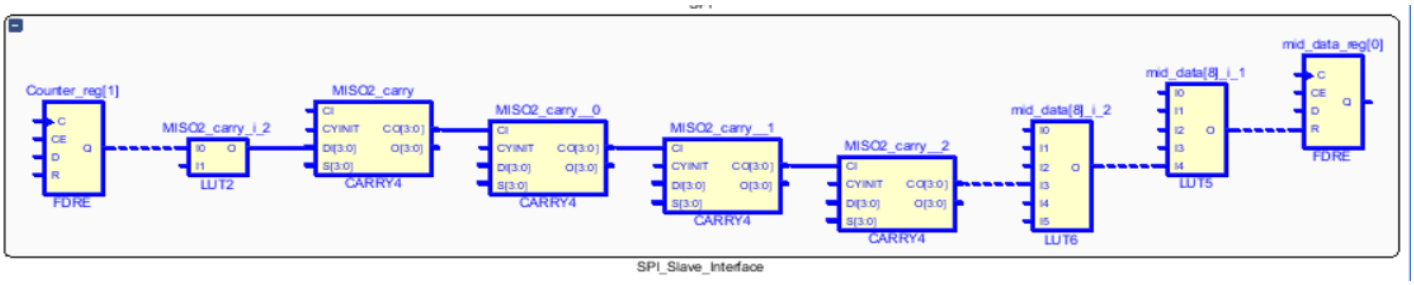

Figure 37: Gray FSM-encoded Critical Path

9.5 Implementation Device

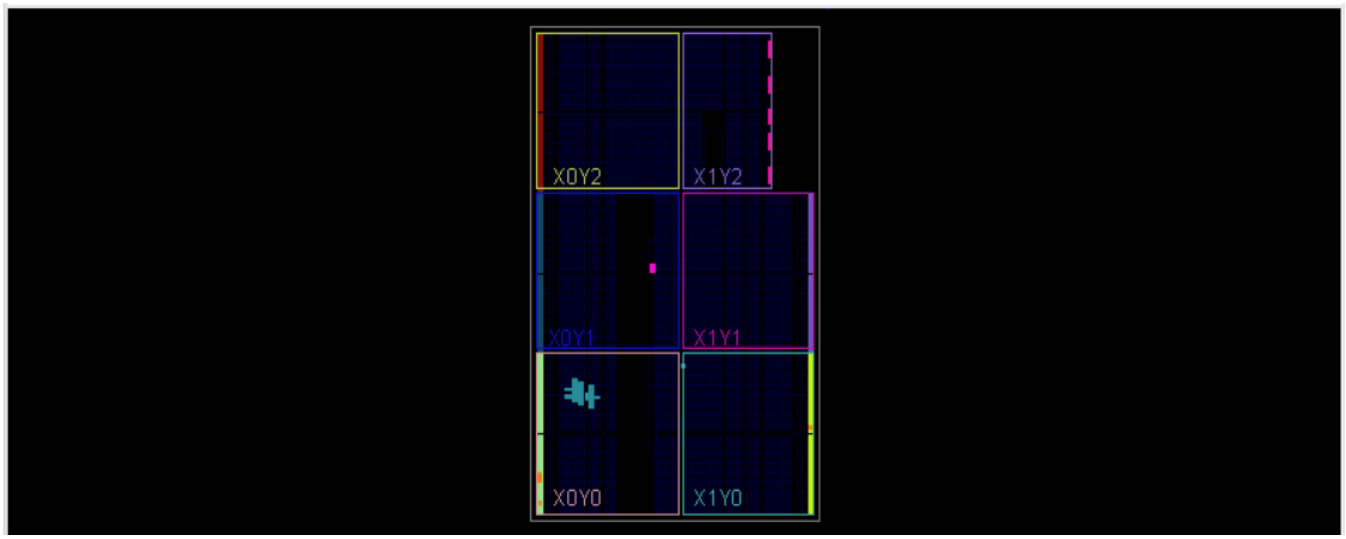

Figure 38: Gray FSM-encoded Implementation Device

9.6 Implementation Utilization Report

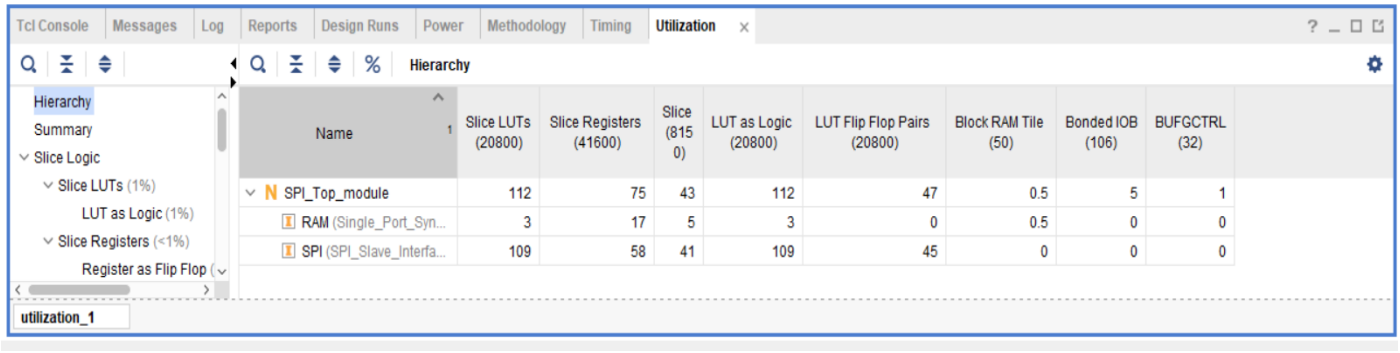

Figure 39: Gray FSM-encoded Hierarchy Utilization Report

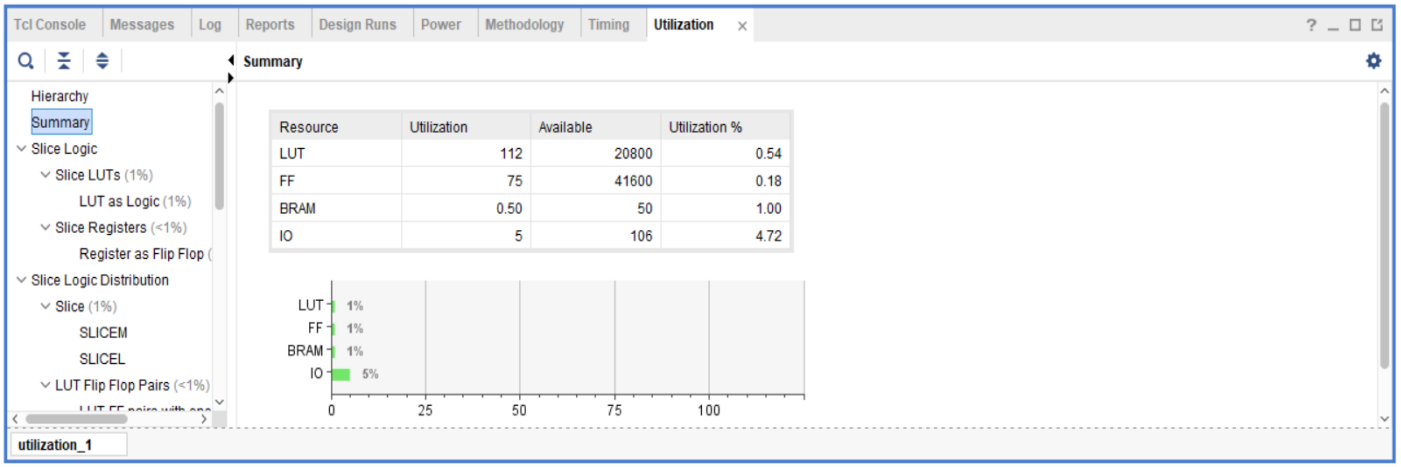

Figure 40: Gray FSM-encoded Summary Utilization Report

9.7 Implementation Timing Report

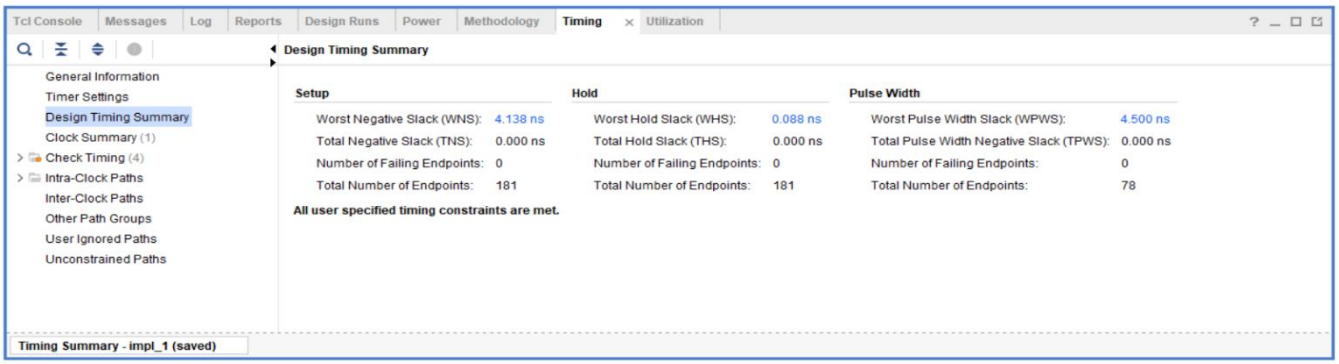

Figure 41: Gray FSM-encoded Implementation Timing Report

9.8 Messages Tab

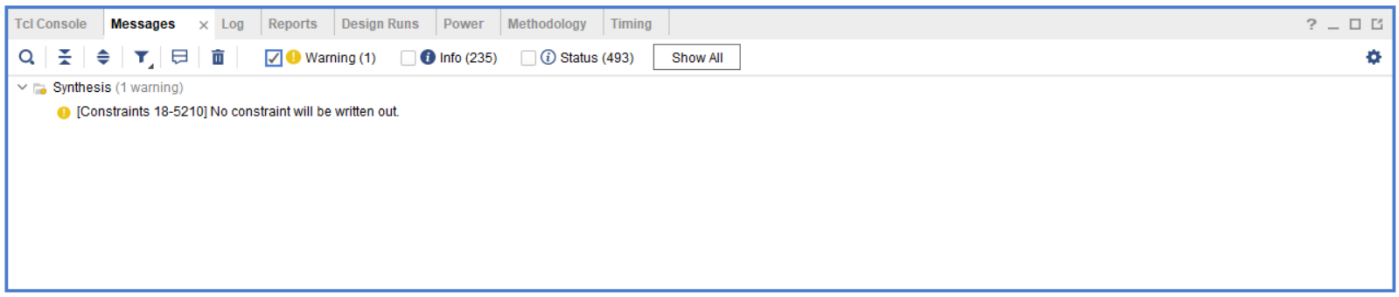

Figure 42: Gray FSM-encoded Messages Tab