Temon & Bomba Team

Digital Design

SPI Project

Provided by

Saeed Nabawy

Zeyad Abdallah Shaban

Mohamed Hassan Abdelkhalek

Submitted to

ENG / Kareem Waseem

Table of Contents

Introduction 5

Project Description 6

Overview 6

Top-Level Module: SPI_Wrapper 6

SPI_Slave Module 7

RAM Module 7

Data Conversion Modules 7

Serial_to_Parallel 7

parallel_to_serial 7

SPI_Wrapper_tb (Testbench) 8

Modules Code 9

Serial_to_Parallel Module 9

Parallel_to_Serial Module 10

RAM Module 11

SPI_Slave Module 12

SPI_Wrapper Module 14

Testbench 15

Do File 16

QuestaSim Simulation 17

Write Address 17

Write Data 17

Data Modified in Memory 18

Read Address 18

Data in Memory Address ‘h255 (‘d85) 18

Read Data & MISO output 19

Linting 19

Elaboration (seq encoding) 20

Message 20

Schematic 20
RAM Schematic 21
SPI Schematic 21
Synthesis (seq encoding) 22
Schematic 22
SPI_Slave Schematic 23
Message 24
Utilization Report 24
Timing Report 24
Critical path highlighted in the schematic 25
Implementation (seq encoding) 26
Device 26
Message 27
Utilization Report 27
Timing Report 27
Constrain File 28
Elaboration (gray encoding) 29
Message 29
Schematic 29
Synthesis (gray encoding) 30
Schematic 30
SPI_Slave Schematic 31
Message 32
Utilization Report 32
Timing Report 32
Implementation (gray encoding) 33
Device 33
Message 34
Utilization Report 34

Timing Report 34
Elaboration (one_hot encoding) 35
Message 35
Schematic 35
Synthesis (one_hot encoding) 36
Schematic 36
SPI_Slave Schematic 37
Message 38
Utilization Report 38
Timing Report 38
Implementation (one_hot encoding) 39
Device 39
Message 40
Utilization Report 40
Timing Report 40

Introduction

In the ever-evolving world of embedded systems, devices must speak a common language — one not built on human words, but on precise timing, pulses, and codes. Deep within the silicon silence of microcontrollers and peripheral chips, the Serial Peripheral Interface (SPI) is that language: a minimalist protocol, sharp and efficient, whispering commands and responses in synchronous rhythm.

This project delves into the hidden patterns of this digital dialogue. We design and implement a custom SPI Slave system in Verilog — a silent listener that not only interprets incoming serial data but acts on it intelligently. Like a watchful gatekeeper, it receives instructions, deciphers intent, and manipulates memory with precision.

The design merges state-machine logic with serial-to-parallel and parallel-to-serial data conversion. It features an internal RAM array, capable of storing and retrieving data based on encoded commands, all processed in real time through carefully synchronized hardware logic.

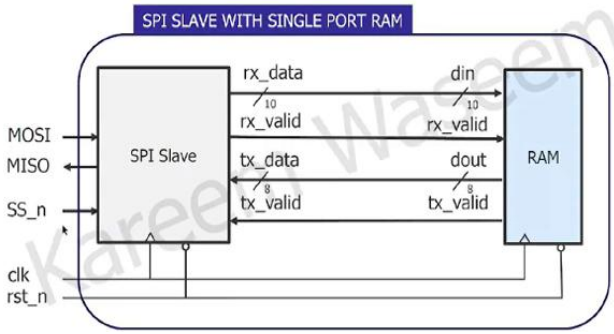
What begins as a stream of shifting bits over a single wire becomes, in this system, a fully responsive, memory-aware peripheral — crafted not just to understand the SPI protocol, but to embody it.

Project Description

Overview

This project implements a **complete SPI Slave system in Verilog**, with a built-in memory subsystem and full support for read/write communication initiated by an SPI Master. It includes all necessary components for:

- Serial-to-parallel conversion (to interpret incoming SPI data),
- Parallel-to-serial conversion (to send responses back),
- Command decoding,
- Addressable memory storage (RAM),
- And a testbench to simulate real-world usage.



Top-Level Module: SPI_Wrapper

This module acts as the central unit. It wires together:

- The SPI_Slave module, which handles protocol logic.
- The RAM module, which stores and retrieves data based on SPI commands.

Ports:

- Inputs: MOSI, SS_n, clk, rst_n
- Output: MISO

Internal Signals:

- rx_data [9:0] – full 10-bit command/data received from SPI
- rx_valid – signals when a valid 10-bit word has been received
- tx_data [7:0] – data to be transmitted over SPI
- tx_valid – signals when data is ready to be transmitted

SPI_Slave Module

This is the protocol engine. It:

- Uses an FSM with five states: IDLE, CHK_CMD, WRITE, READ_ADD, READ_DATA
- Receives 10-bit commands using the Serial_to_Parallel module
- Transmits 8-bit data using the parallel_to_serial module
- Signals data readiness with rx_valid, and waits for tx_valid to begin transmission

RAM Module

A simple 256-byte memory (8-bit wide, 8-bit addressable), supporting:

- Address selection via SPI
- Data storage and retrieval
- Behavioral simulation support via $readmemh

It interprets the rx_data coming from the slave:

- 00 opcode sets the internal write address
- 01 opcode writes data to that address
- 10 opcode sets the read address
- 11 opcode reads data from that address and outputs it on dout

Data Conversion Modules

Serial_to_Parallel

- Converts incoming SPI stream into 10-bit parallel words
- Detects transaction completion and triggers done pulse

parallel_to_serial

- Converts 8-bit parallel data to serial format
- Manages busy signal to indicate transmission state

SPI_Wrapper_tb (Testbench)

The testbench simulates:

- Clock and reset behavior
- Command sequences for write and read operations
- Realistic SPI timing (bit-wise shifting over MOSI)
- Memory preload using $readmemh("mem.dat", ...)

Test Phases:

1. Reset system
2. Write address
3. Write data
4. Read address
5. Read data
6. Wait for response, verify behavior

Modules Code

Serial_to_Parallel Module

```
module Serial_to_Parallel(
    input clk,
    input rst_n,
    input serial_in,
    input enable,
    output reg [9:0] parallel_out,
    output reg done
);



    reg [3:0] bit_count;
    reg [9:0] shift_reg;

    always @(posedge clk) begin
        if (!rst_n) begin
            shift_reg    <= 10'b0;
            bit_count    <= 4'b0;
            done         <= 1'b0;
            parallel_out <= 10'b0;
        end
        else if (enable) begin
            shift_reg <= {shift_reg[8:0], serial_in};
            bit_count <= bit_count + 1;

            if (bit_count == 4'b1001) begin
                parallel_out <= {shift_reg[8:0], serial_in};
                shift_reg <= 10'b0;
                done <= 1'b1;
                bit_count <= 4'b0;
            end else begin
                done <= 1'b0;
            end
        end else begin
            done <= 1'b0;
        end
    end

endmodule

```

Parallel_to_Serial Module

```
module parallel_to_serial (
    input wire clk,
    input wire rst_n,
    input wire [7:0] parallel_in,
    input wire load,
    output reg serial_out,
    output reg busy
);

    reg [7:0] shift_reg;
    reg [2:0] bit_count;

    always @(posedge clk) begin
        if (!rst_n) begin
            shift_reg  <= 8'd0;
            bit_count  <= 3'd0;
            serial_out <= 1'bz;
            busy       <= 1'b0;
        end else if (load && !busy) begin
            shift_reg  <= parallel_in;
            bit_count  <= 3'd0;
            busy       <= 1'b1;
        end
        else if (busy) begin
            serial_out <= shift_reg[7];
            shift_reg  <= {shift_reg[6:0], 1'b0};
            bit_count  <= bit_count + 1;

            if (bit_count == 3'd7) begin
                busy <= 1'b0;
            end
        end
        else begin
            serial_out <= 1'bz;
            busy       <= 1'b0;
        end
    end

endmodule

```

RAM Module

```
module RAM (
    input [9:0] din,
    input rx_valid,clk,rst_n,
    output reg tx_valid,
    output reg [7:0] dout
);

    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

    reg [ADDR_SIZE-1:0] write_address;
    reg [ADDR_SIZE-1:0] read_address;

    reg [7:0] mem [0:MEM_DEPTH-1];


    always @(posedge clk) begin
        if(!rst_n) begin
            dout <= 0;
            tx_valid <= 0;
        end
        else begin
            if(rx_valid) begin
                case(din[9:8])
                    2'b00 : begin
                        write_address <= din[7:0];
                        dout <= 8'bzzzzzzzz;
                        tx_valid <= 1'bz;
                    end
                    2'b01 : begin
                        mem [write_address] <= din[7:0];
                        dout <= 8'bzzzzzzzz;
                        tx_valid <= 1'bz;
                    end
                    2'b10 : begin
                        read_address <= din[7:0];
                        dout <= 8'bzzzzzzzz;
                        tx_valid <= 1'bz;
                    end
                    2'b11 : begin
                        dout <= mem [read_address];
                        tx_valid <= 1;
                    end
                    default : begin
                        dout <= 8'bzzzzzzzz;
                        tx_valid <= 1'bz;
                    end
                endcase
            end
            else begin
                dout <= 0;
                tx_valid <= 0;
            end
        end
    end

endmodule //RAM
```

SPI_Slave Module

```
module SPI_Slave(
    input MOSI,
    input SS_n,
    input clk,
    input rst_n,
    input [7:0] tx_data,
    input tx_valid,
    output MISO,
    output reg [9:0] rx_data,
    output reg rx_valid
    );

    parameter IDLE = 5'b000;
    parameter CHK_CMD = 5'b001;
    parameter WRITE = 5'b010;
    parameter READ_ADD = 5'b011;
    parameter READ_DATA = 5'b100;

    // (* fsm_encoding = "gray" *)
    reg [4:0] cs,ns;
    reg addr_or_data ;
    reg STP_enable;
    wire STP_Done;
    wire [9:0] STP_out;
    wire PTS_busy, PTS_out;

    Serial_to_Parallel STP (.clk(clk), .rst_n(rst_n), .serial_in(MOSI),
    .enable(STP_enable), .parallel_out(STP_out), .done(STP_Done));

    parallel_to_serial PTS (.clk(clk), .rst_n(rst_n), .parallel_in(tx_data),
    .load(tx_valid), .serial_out_buff(MISO), .busy(PTS_busy));

    always @(cs,SS_n) begin
        case(cs)
            IDLE: begin
                if(SS_n) begin
                    ns = IDLE;
                end
                else begin
                    ns = CHK_CMD;
                end
            end
            CHK_CMD: begin
                if(SS_n) begin
                    ns = IDLE;
                end
                else begin
                    if(MOSI) begin
                        if(addr_or_data) begin
                            ns = READ_DATA;
                        end
                        else begin
                            ns = READ_ADD;
                        end
                    end
                    else begin
                        ns = WRITE;
                    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

```

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

    always @(posedge clk) begin
        if(cs == IDLE || cs == CHK_CMD) begin
            STP_enable <= 0;
            rx_data<= {10{1'bz}};
            rx_valid <= 1'b0;
        end
        else if(cs == WRITE) begin
            STP_enable <= 1;
            if(STP_Done) begin
                rx_valid <= 1;
                rx_data <= STP_out;
                STP_enable <= 0;
            end
        end
        else if(cs == READ_ADD) begin
            STP_enable <= 1;
            if(STP_Done) begin
                rx_valid <= 1;
                rx_data <= STP_out;
                addr_or_data <= 1;
                STP_enable <= 0;
            end
        end
        else if(cs == READ_DATA) begin
            STP_enable <= 1;
            if(STP_Done) begin
                rx_valid <= 1;
                rx_data<= STP_out;
                addr_or_data <= 0;
                STP_enable <= 0;
            end
        end
    end
endmodule
```

SPI_Wrapper Module

```
module SPI_Wrapper(
    input MOSI,
    input SS_n,
    input clk,
    input rst_n,
    output MISO
    );
    wire [9:0] rx_data;
    wire rx_valid;
    wire [7:0] tx_data;
    wire tx_valid;

    SPI_Slave SPI (.MOSI(MOSI),.SS_n(SS_n),.clk(clk),.rst_n(rst_n),
    .MISO(MISO),.rx_data(rx_data),.rx_valid(rx_valid),.tx_data(tx_data),
    .tx_valid(tx_valid));

    RAM u_ram(.din(rx_data),.rx_valid(rx_valid),.dout(tx_data),
    .tx_valid(tx_valid),.clk(clk),.rst_n(rst_n));

endmodule
```

Testbench

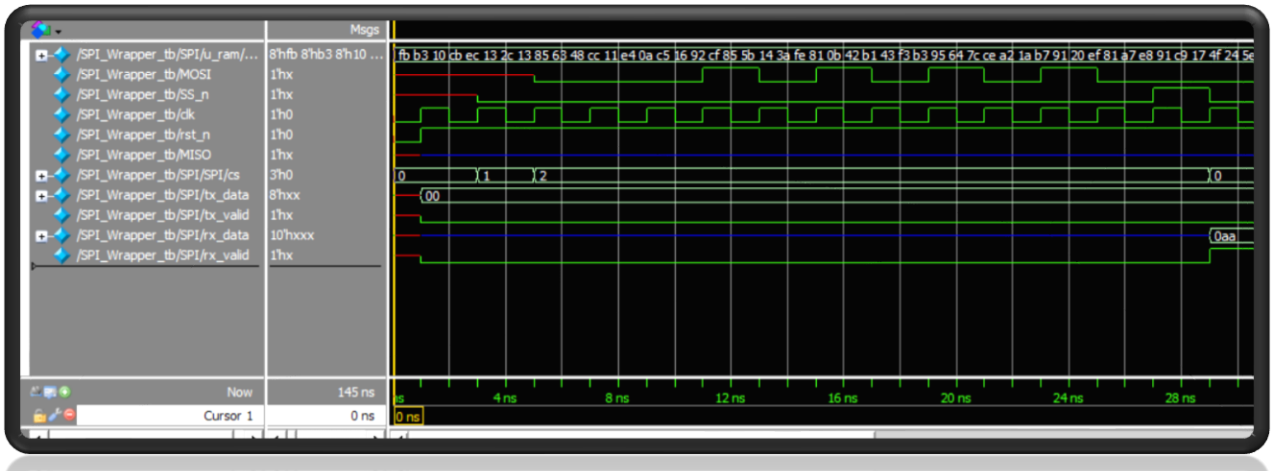
```
module SPI_Wrapper_tb();

    reg MOSI,SS_n,clk,rst_n;
    wire MISO;

    SPI_Wrapper SPI(.MOSI(MOSI),.SS_n(SS_n),.clk(clk),
    .rst_n(rst_n),.MISO(MISO));

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

    integer i;
    initial begin
        $readmemh("mem.dat", SPI.u_ram.mem);
        // resq test
        rst_n = 0;
        @(posedge clk);
        rst_n = 1;

        // Write Address
        @(posedge clk);
        SS_n = 0;
        @(posedge clk);
        MOSI = 0;
        @(posedge clk);
        for(i = 0; i < 10; i = i + 1) begin
            if(i < 2) begin
                MOSI = 0;
            end
            else begin
                MOSI = ~MOSI;
            end
            @(posedge clk);

        end
        SS_n = 1;

        // Write Data
        @(posedge clk);
        SS_n = 0;
        @(posedge clk);
        MOSI = 0;
        @(posedge clk);
        for(i = 0; i < 10; i = i + 1) begin
            if(i == 0) begin
                MOSI = 0;
            end begin
                MOSI = 1;
            end
            @(posedge clk);
        end
        SS_n = 1;

        // Read Address
        @(posedge clk);
        SS_n = 0;
        @(posedge clk);
        MOSI = 1;
        @(posedge clk);
        for(i = 0; i < 10; i = i + 1) begin
            if(i == 0) begin
                MOSI = 1;
            end
            else if(i == 1) begin
                MOSI = 0;
            end
            else begin
                MOSI = ~MOSI;
            end
            @(posedge clk);
        end
        SS_n = 1;

        // Read Data
        @(posedge clk);
        SS_n = 0;
        @(posedge clk);
        MOSI = 1;
        @(posedge clk);
        for(i = 0; i < 10; i = i + 1) begin
            MOSI = 1;
            @(posedge clk);
        end
        for(i = 0; i < 20; i = i + 1) begin
            @(posedge clk);
        end
        SS_n = 1;
        $stop;
    end

endmodule
```

Do File

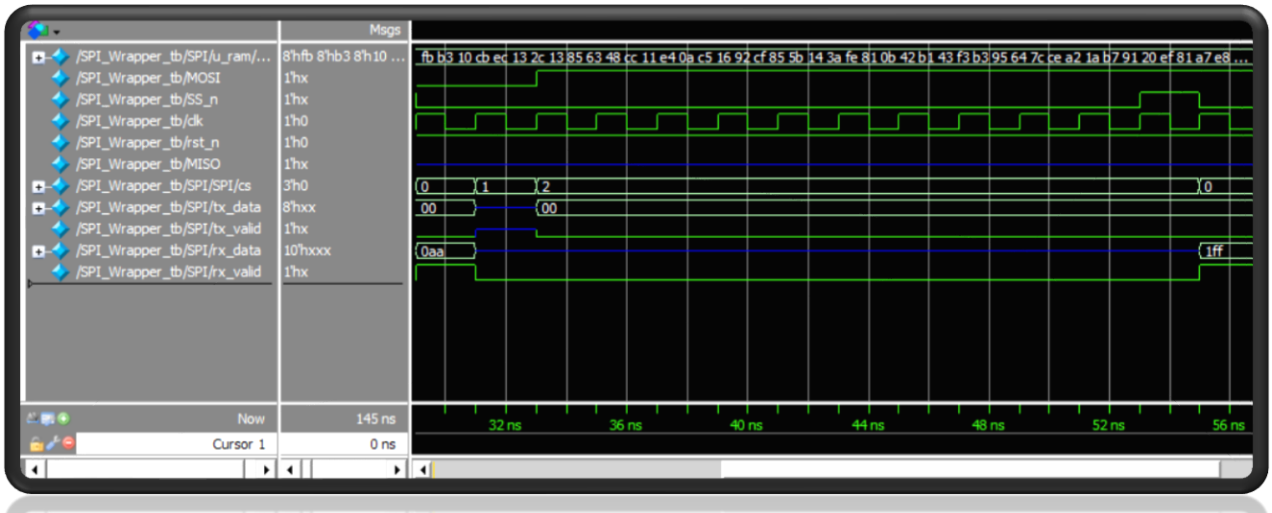
```
vlib work
vlog RAM.v Serial_to_Parallel.v Parallel_to_Serial.v SPI_Slave.v SPI_Wrapper.v
vsim -voptargs=+acc work.SPI_Wrapper_tb
add wave -position insertpoint sim:/SPI_Wrapper_tb/*
add wave -position insertpoint  \
sim:/SPI_Wrapper_tb/SPI/rx_data \
sim:/SPI_Wrapper_tb/SPI/rx_valid \
sim:/SPI_Wrapper_tb/SPI/tx_data \
sim:/SPI_Wrapper_tb/SPI/tx_valid
add wave -position insertpoint  \
sim:/SPI_Wrapper_tb/SPI/SPI/cs
add wave -position 0  sim:/SPI_Wrapper_tb/SPI/u_ram/mem
run -all
#quit -sim
```

QuestaSim Simulation

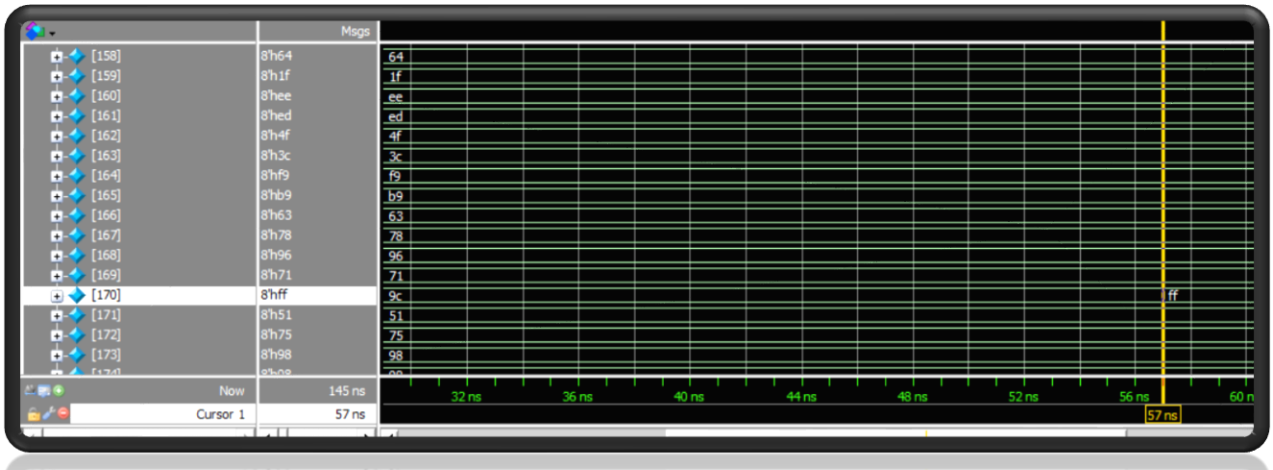
Write Address



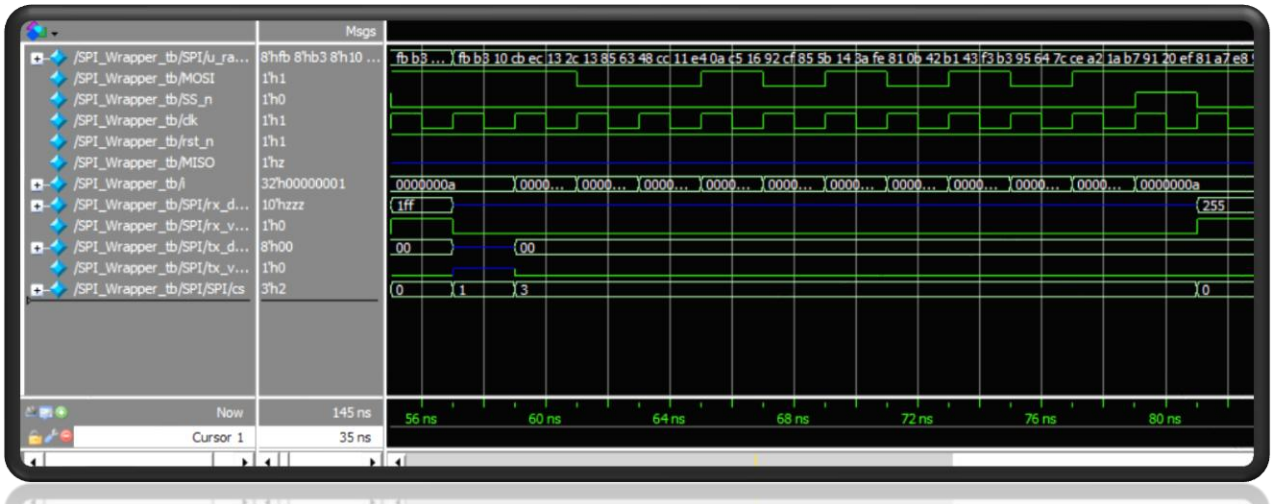
Write Data



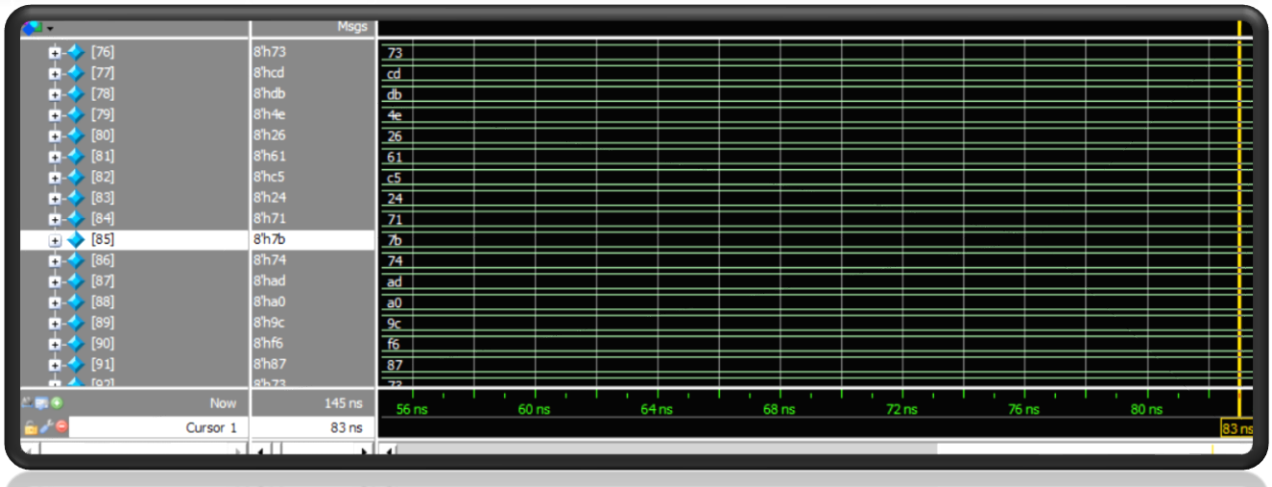
Data Modified in Memory



Read Address

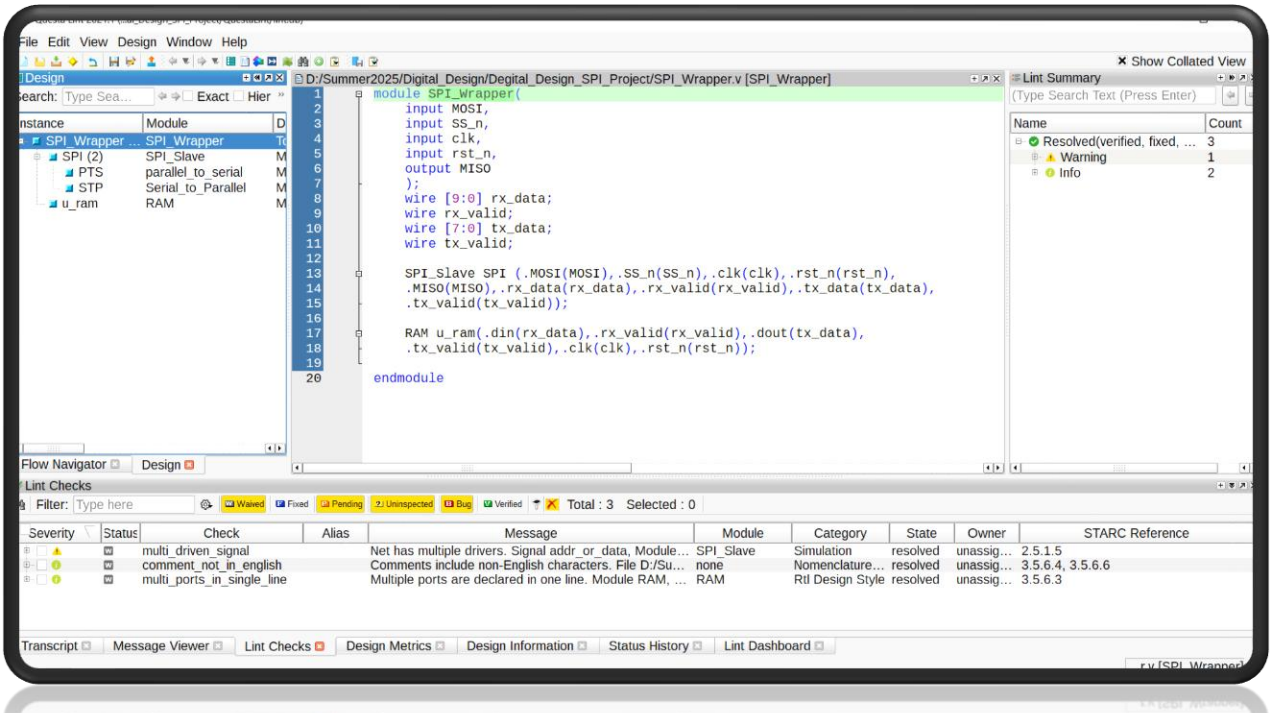


Data in Memory Address ‘h255 (‘d85)



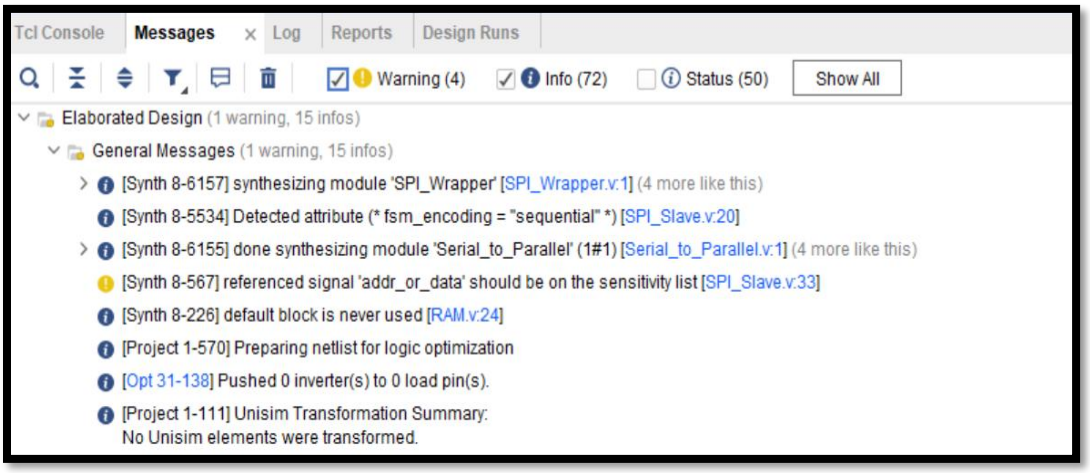
Read Data & MISO output

Linting

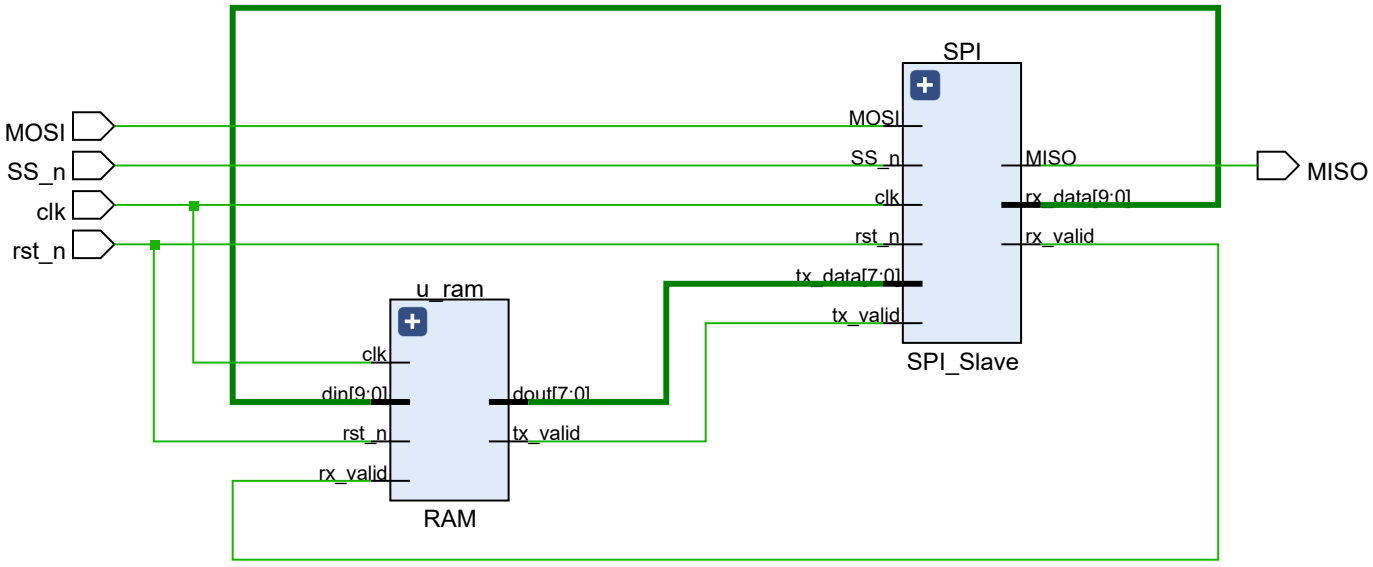


Elaboration (seq encoding)

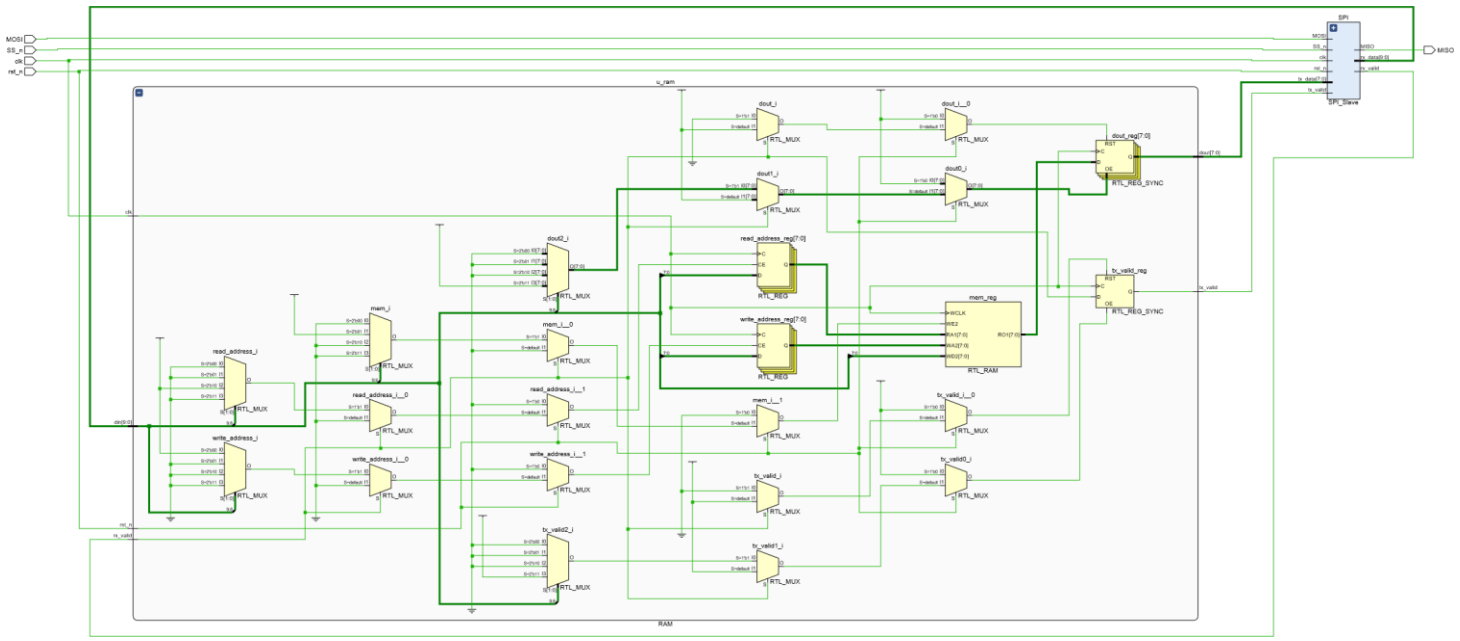
Message



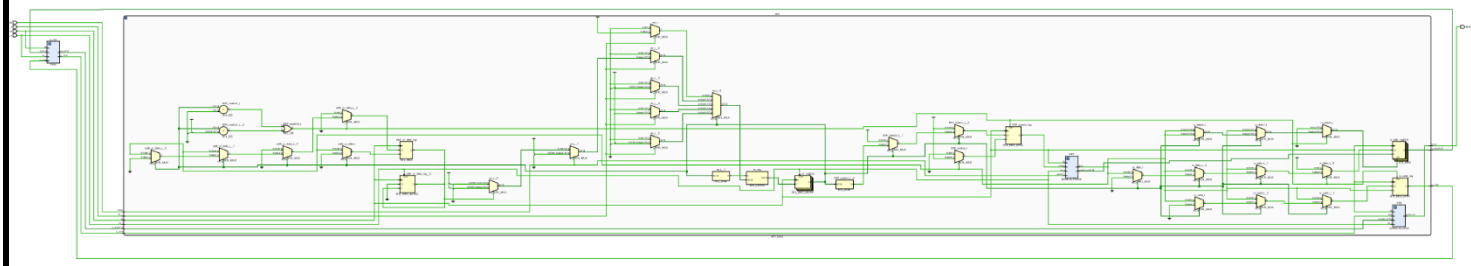
Schematic



RAM Schematic



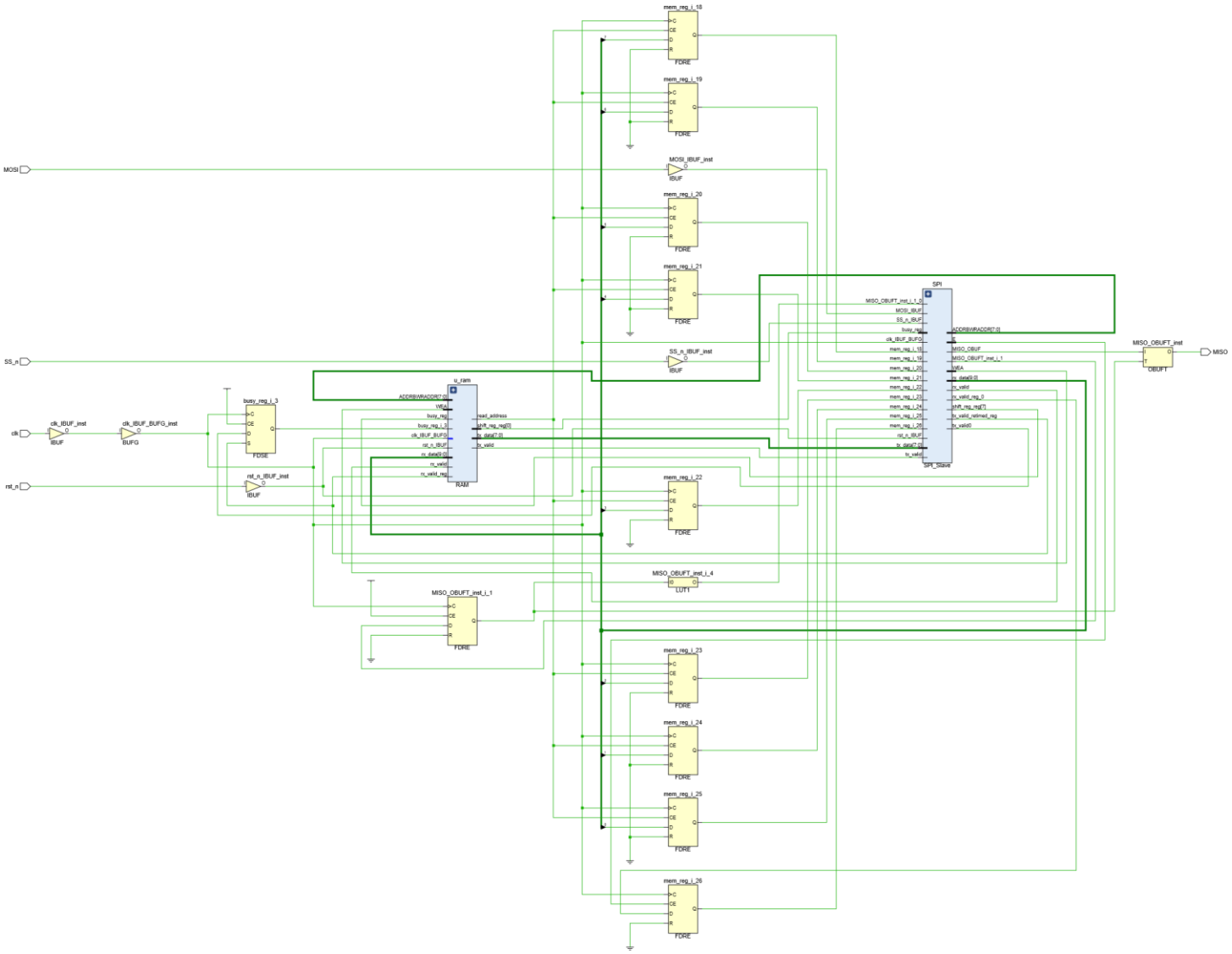
SPI Schematic



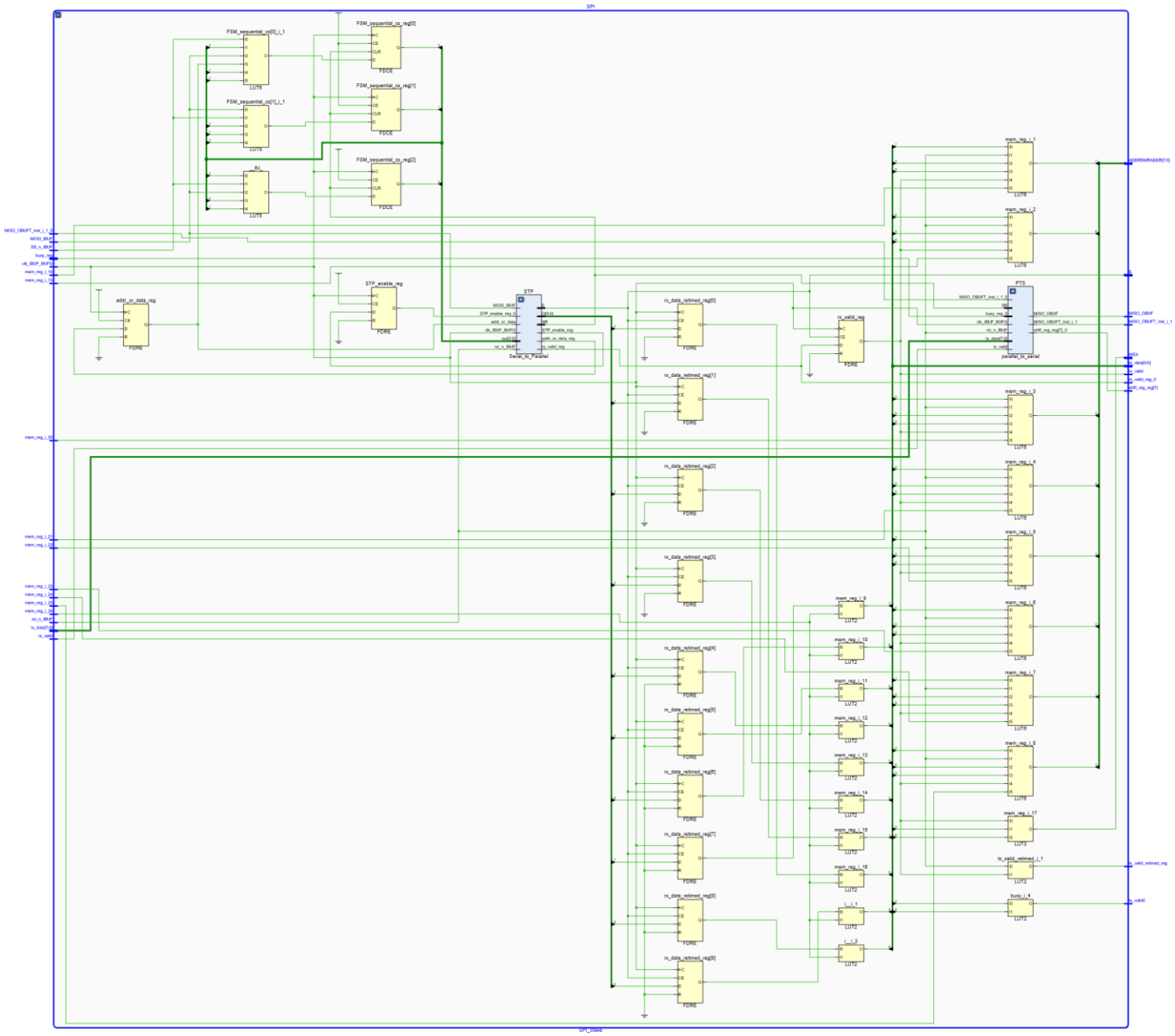
Synthesis (seq encoding)

State	Encoding
IDLE	5’b000
CHK_CMD	5’b001
WRITE	5’b010
READ_ADD	5’b011
READ_DATA	5’b100

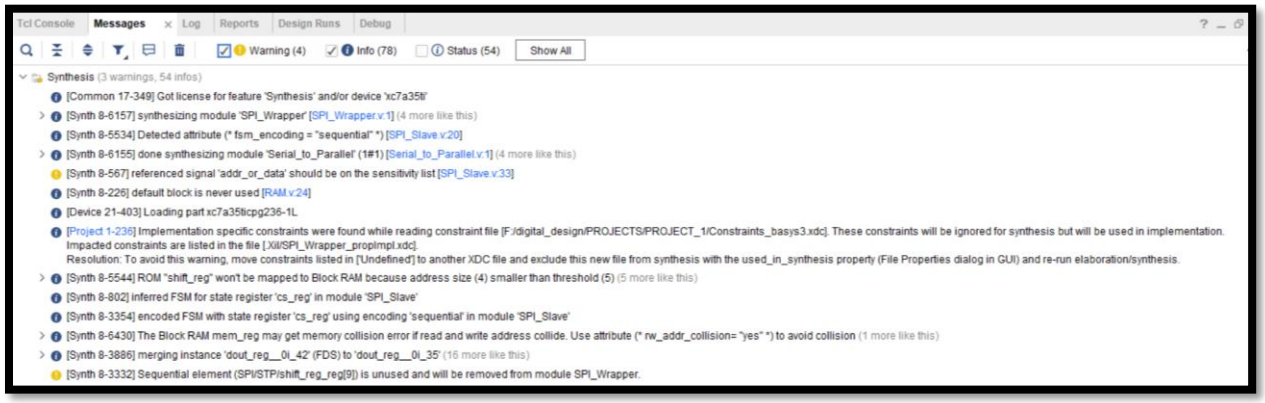
Schematic



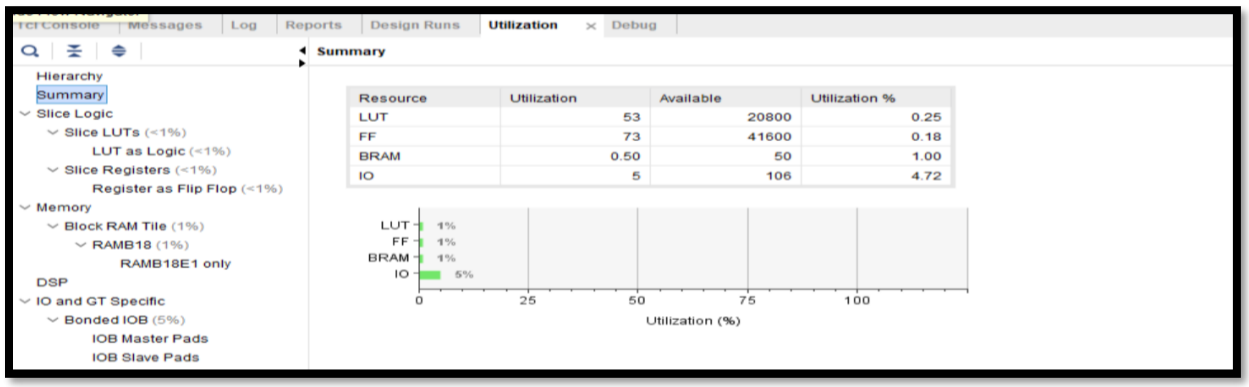
SPI_Slave Schematic



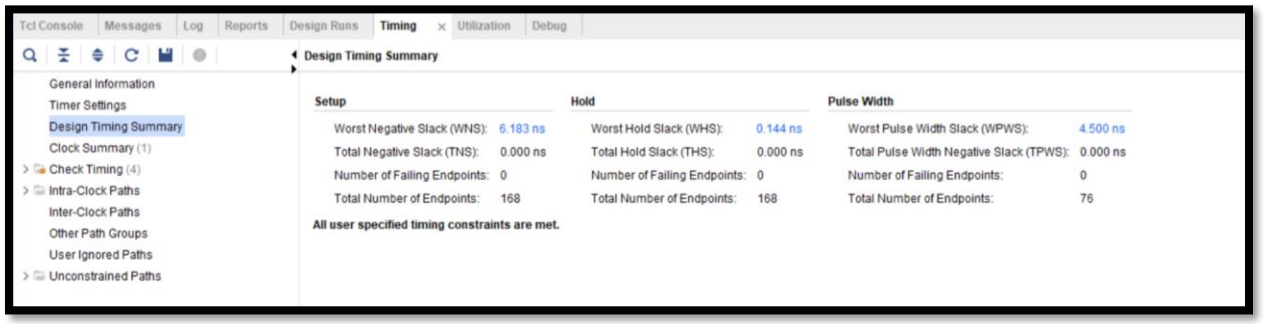
Message



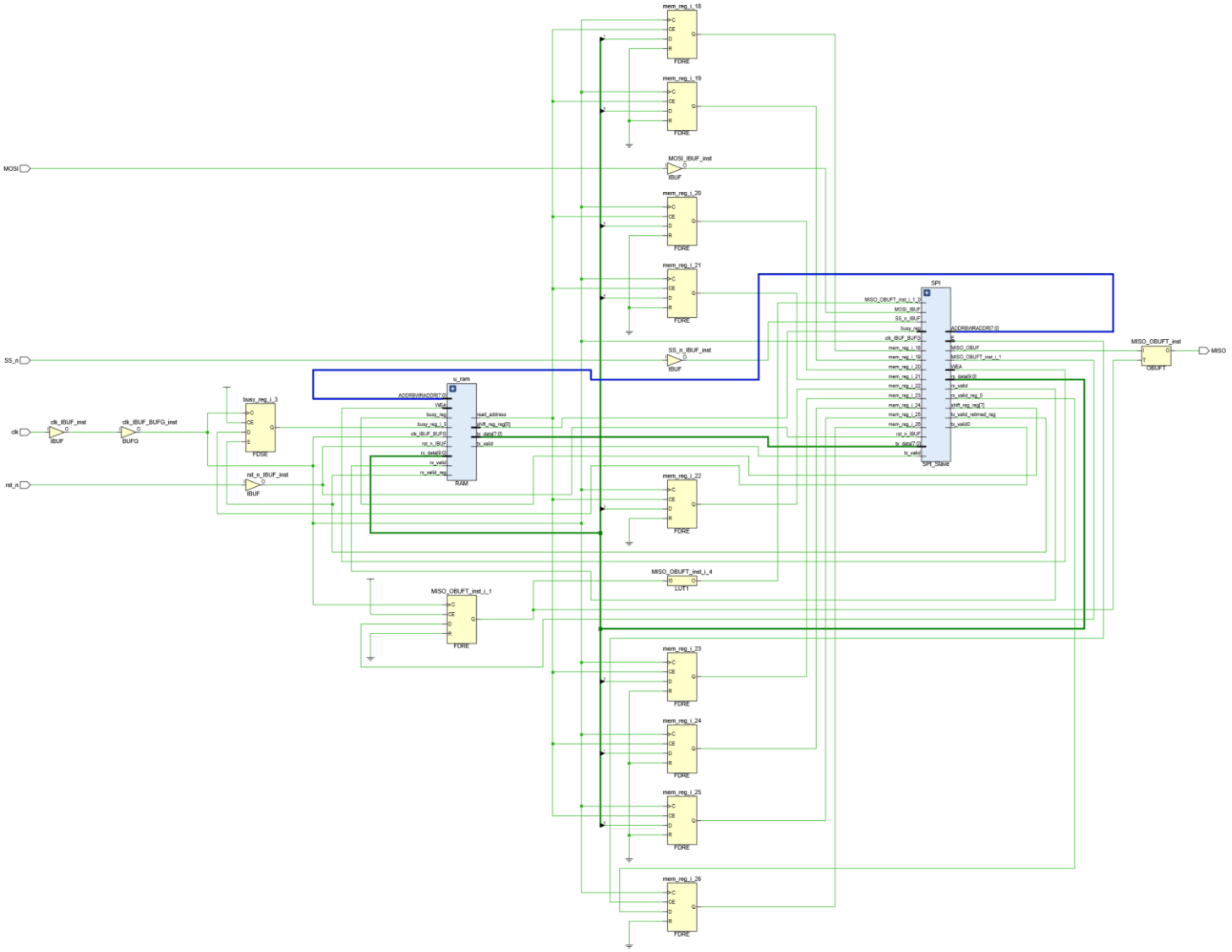
Utilization Report



Timing Report

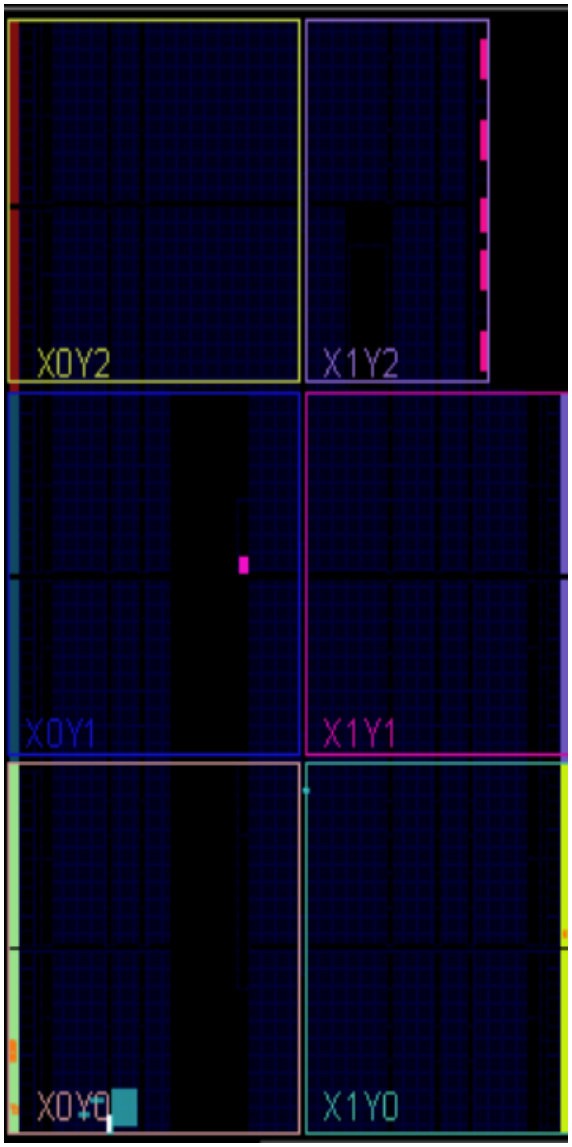


Critical path highlighted in the schematic

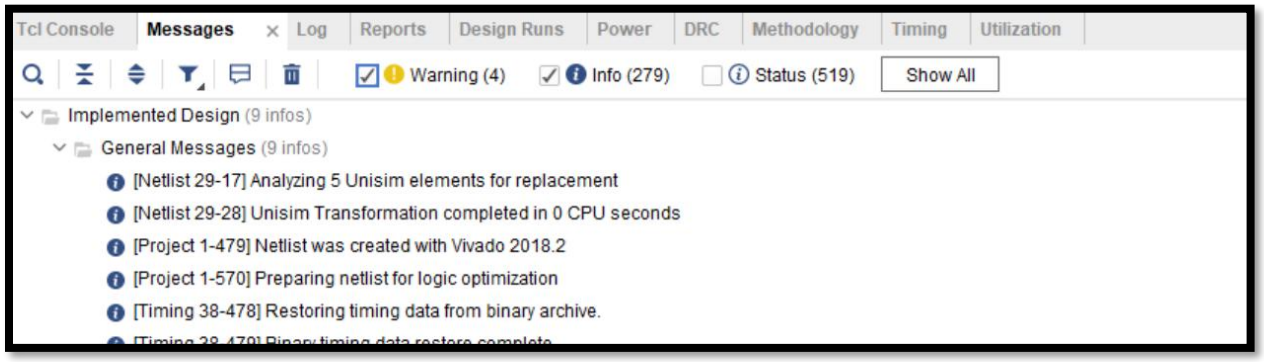


Implementation (seq encoding)

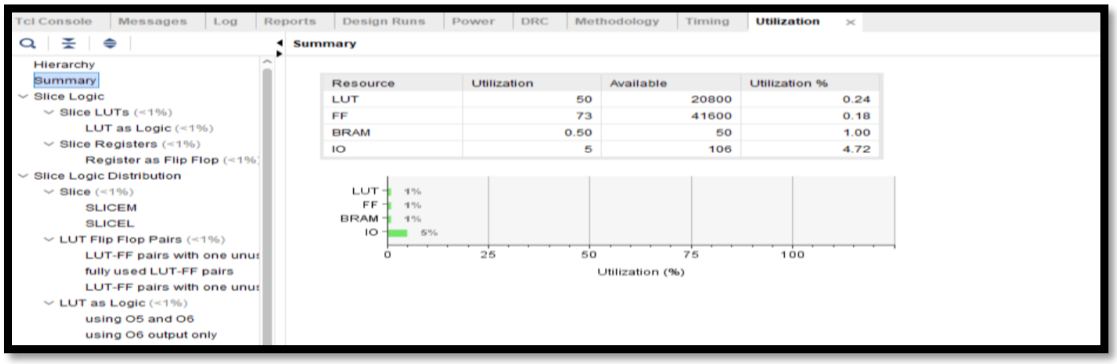
Device



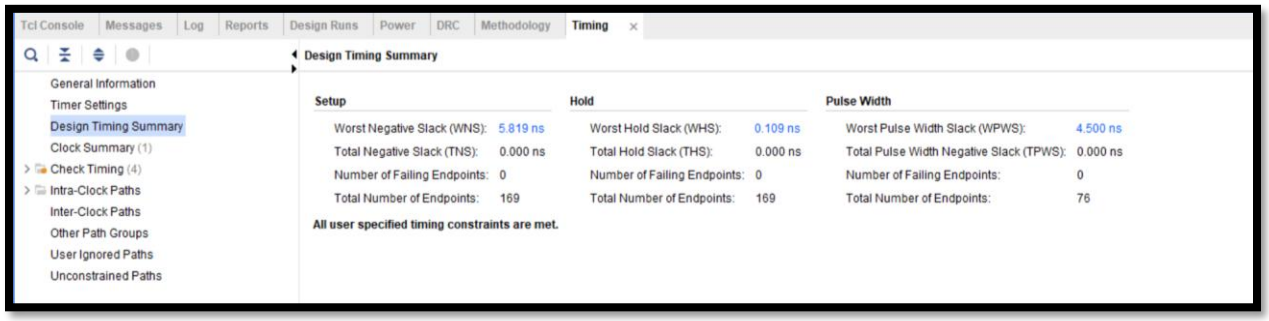
Message



Utilization Report



Timing Report

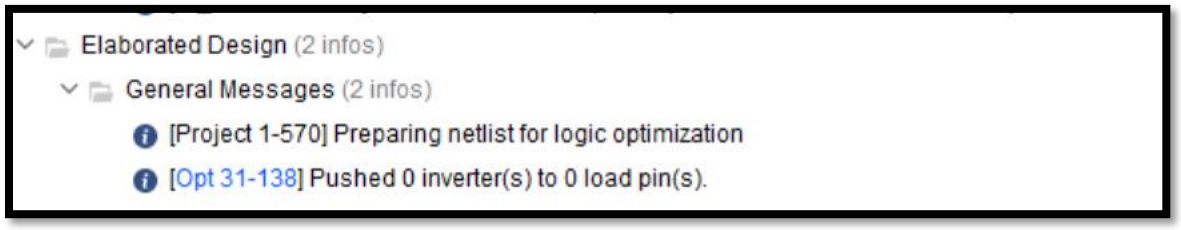


Constrain File

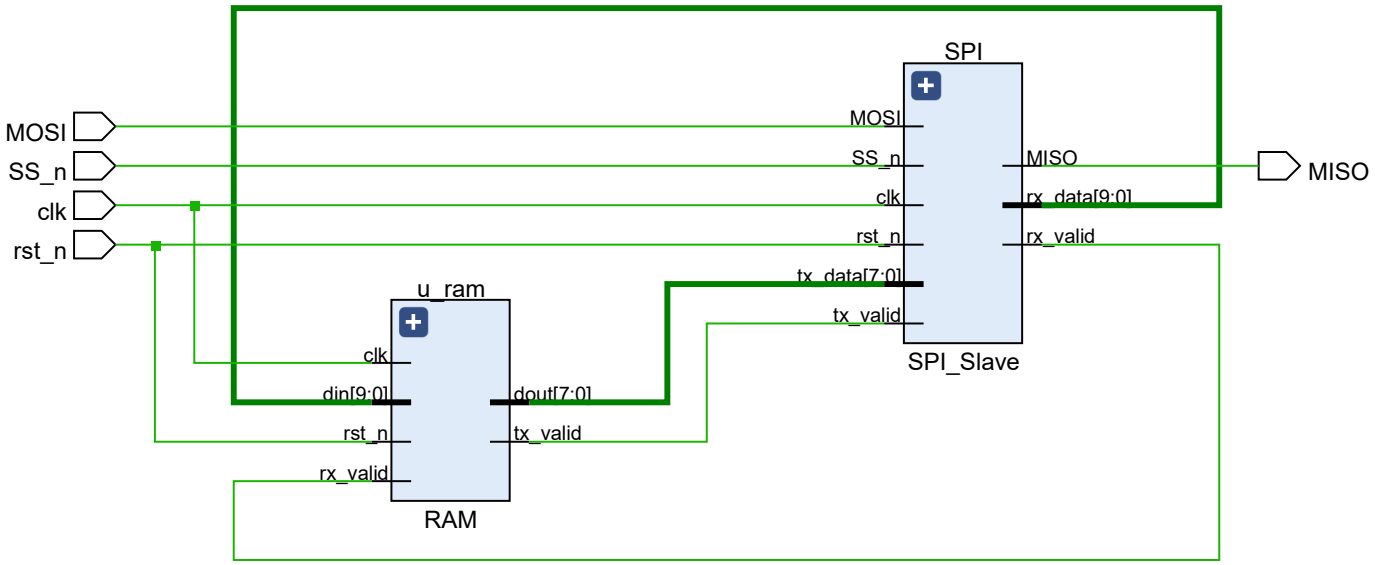
```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

# Clock signal
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}]
#set property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
#set property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
#set property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {sw[5]}]
#set property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {sw[6]}]
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {sw[7]}]
#set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports {sw[8]}]
#set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports {sw[9]}]
#set_property  dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports {sw[10]}]
#set_property  dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports {sw[11]}]
#set_property  dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports {sw[12]}]
#set_property  dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports {sw[13]}]
#set_property  dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports {sw[14]}]
#set_property  dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports {sw[15]}]


## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {MISO}]
#set property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
#set property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
#set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
#set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
#set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {led[5]}]
#set_property  dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {led[6]}]
#set_property  dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {led[7]}]
#set_property  dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports {led[8]}]
#set_property  dict { PACKAGE_PIN V3    IOSTANDARD LVCMOS33 } [get_ports {led[9]}]
#set_property  dict { PACKAGE_PIN W3    IOSTANDARD LVCMOS33 } [get_ports {led[10]}]
#set_property -dict { PACKAGE_PIN U3    IOSTANDARD LVCMOS33 } [get_ports {led[11]}]
#set_property -dict { PACKAGE_PIN P3    IOSTANDARD LVCMOS33 } [get_ports {led[12]}]
#set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports {led[13]}]
#set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports {led[14]}]
#set_property -dict { PACKAGE_PIN L1    IOSTANDARD LVCMOS33 } [get_ports {led[15]}]


##7 Segment Display
#set_property -dict { PACKAGE_PIN W7    IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
#set_property  dict { PACKAGE_PIN W6    IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
#set_property  dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
#set_property  dict { PACKAGE_PIN V8    IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
#set_property  dict { PACKAGE_PIN U5    IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
#set_property  dict { PACKAGE_PIN V5    IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
#set_property -dict { PACKAGE_PIN U7    IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]

#set property -dict { PACKAGE_PIN V7    IOSTANDARD LVCMOS33 } [get ports dp]

#set property -dict { PACKAGE_PIN U2    IOSTANDARD LVCMOS33 } [get_ports {an[0]}]
#set_property -dict { PACKAGE_PIN U4    IOSTANDARD LVCMOS33 } [get_ports {an[1]}]
#set_property -dict { PACKAGE_PIN V4    IOSTANDARD LVCMOS33 } [get_ports {an[2]}]
#set_property -dict { PACKAGE_PIN W4    IOSTANDARD LVCMOS33 } [get_ports {an[3]}]


##Buttons
# set_property  dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst_n]
#set_property  dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btnU]
#set_property  dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btnL]
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btnR]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]


##Pmod Header JA
#set_property -dict { PACKAGE_PIN J1    IOSTANDARD LVCMOS33 } [get_ports {JA[0]}];#Sch name = JA1
#set_property -dict { PACKAGE_PIN L2    IOSTANDARD LVCMOS33 } [get_ports {JA[1]}];#Sch name = JA2
#set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33 } [get_ports {JA[2]}];#Sch name = JA3
#set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33 } [get_ports {JA[3]}];#Sch name = JA4
#set_property -dict { PACKAGE_PIN H1    IOSTANDARD LVCMOS33 } [get_ports {JA[4]}];#Sch name = JA7
#set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports {JA[5]}];#Sch name = JA8
#set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33 } [get_ports {JA[6]}];#Sch name = JA9
#set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33 } [get_ports {JA[7]}];#Sch name = JA10


##Pmod Header JB
#set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {JB[0]}];#Sch name = JB1
#set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {JB[1]}];#Sch name = JB2
#set_property -dict { PACKAGE_PIN B15   IOSTANDARD LVCMOS33 } [get_ports {JB[2]}];#Sch name = JB3
#set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {JB[3]}];#Sch name = JB4
#set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports {JB[4]}];#Sch name = JB7
#set_property -dict { PACKAGE_PIN A17   IOSTANDARD LVCMOS33 } [get_ports {JB[5]}];#Sch name = JB8
#set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports {JB[6]}];#Sch name = JB9
#set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports {JB[7]}];#Sch name = JB10

##Pmod Header JC
#set_property  dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports {JC[0]}];#Sch name = JC1
#set_property  dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports {JC[1]}];#Sch name = JC2
#set_property  dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports {JC[2]}];#Sch name = JC3
#set_property  dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports {JC[3]}];#Sch name = JC4
#set_property  dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports {JC[4]}];#Sch name = JC7
#set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports {JC[5]}];#Sch name = JC8
#set property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get ports {JC[6]}];#Sch name = JC9
#set property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get ports {JC[7]}];#Sch name = JC10


##Pmod Header JXADC
#set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports {JXADC[0]}];#Sch name = XA1_P
#set_property -dict { PACKAGE_PIN L3    IOSTANDARD LVCMOS33 } [get_ports {JXADC[1]}];#Sch name = XA2_P
#set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports {JXADC[2]}];#Sch name = XA3_P
#set_property -dict { PACKAGE_PIN N2    IOSTANDARD LVCMOS33 } [get_ports {JXADC[3]}];#Sch name = XA4_P
#set_property  dict { PACKAGE_PIN K3    IOSTANDARD LVCMOS33 } [get_ports {JXADC[4]}];#Sch name = XA1_N
#set_property  dict { PACKAGE_PIN M3    IOSTANDARD LVCMOS33 } [get_ports {JXADC[5]}];#Sch name = XA2_N
#set_property  dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports {JXADC[6]}];#Sch name = XA3_N
#set_property  dict { PACKAGE_PIN N1    IOSTANDARD LVCMOS33 } [get_ports {JXADC[7]}];#Sch name = XA4_N


##VGA Connector
#set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[0]}]
#set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[1]}]
#set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[2]}]
#set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[3]}]
#set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[0]}]
#set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[1]}]
#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[2]}]
#set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[3]}]
#set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[0]}]
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[1]}]
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[2]}]
#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[3]}]
#set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports Hsync]
#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports Vsync]


##USB-RS232 Interface
#set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports RsRx]
#set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports RsTx]


##USB HID (PS/2)
#set_property  dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33   PULLUP true } [get_ports PS2Clk]
#set_property  dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33   PULLUP true } [get_ports PS2Data]


##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the
##STARTUPE2 primitive.
#set property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get ports {QspiDB[0]}]
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[1]}]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[2]}]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[3]}]
#set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports QspiCSn]


## 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]
```

Elaboration (gray encoding)

Message



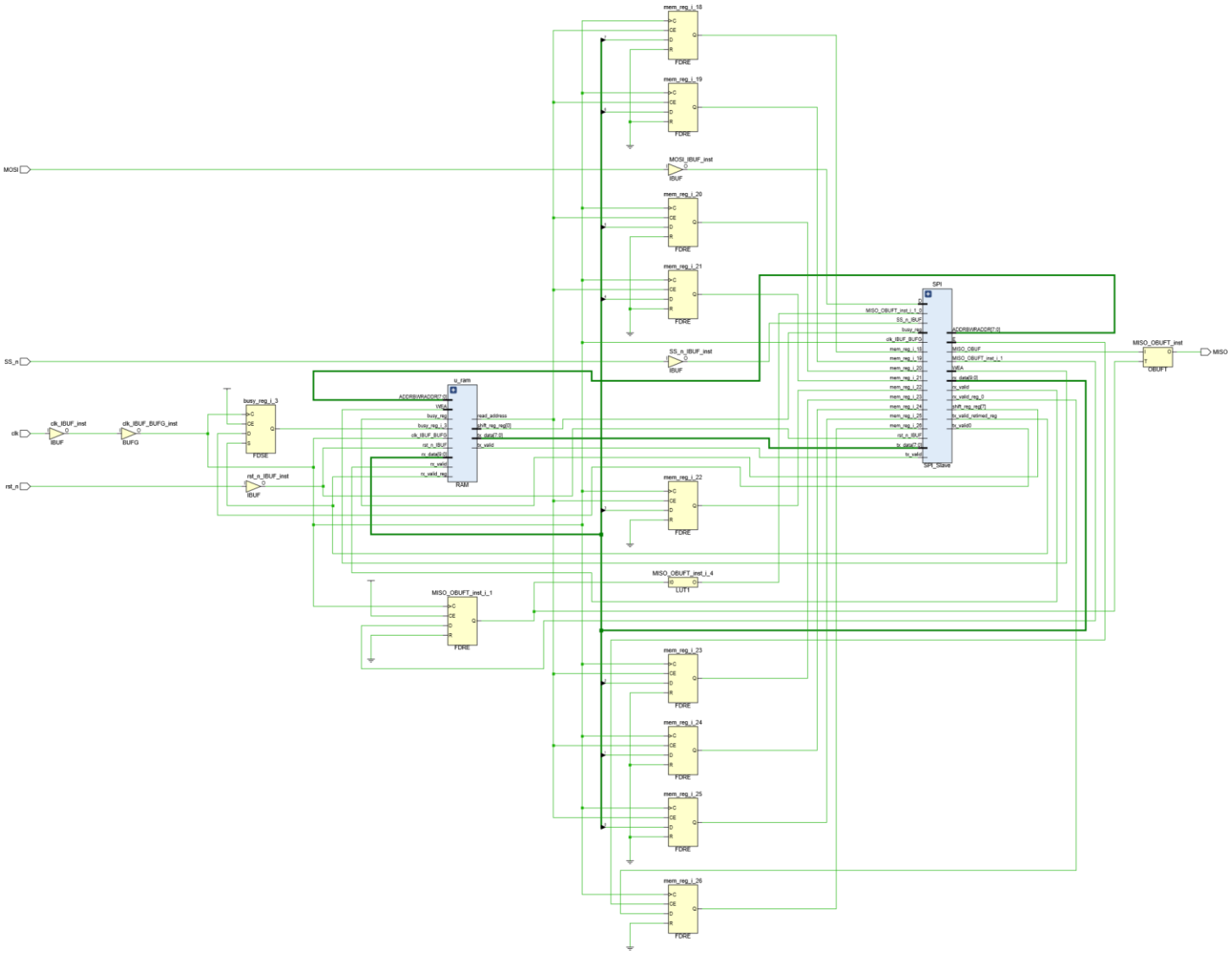
Schematic



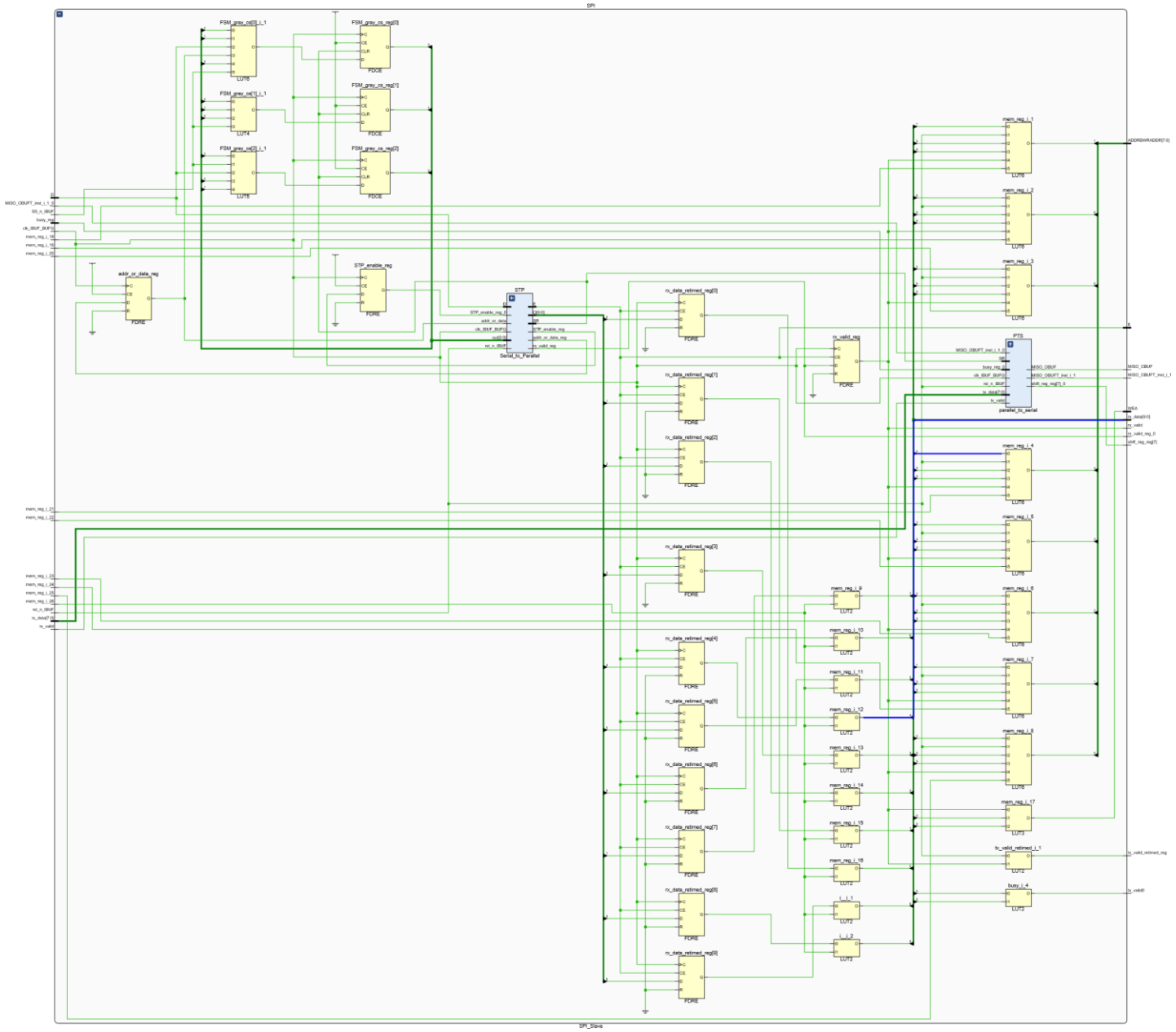
Synthesis (gray encoding)

State	Encoding
IDLE	5’b000
CHK_CMD	5’b001
WRITE	5’b011
READ_ADD	5’b010
READ_DATA	5’b110

Schematic

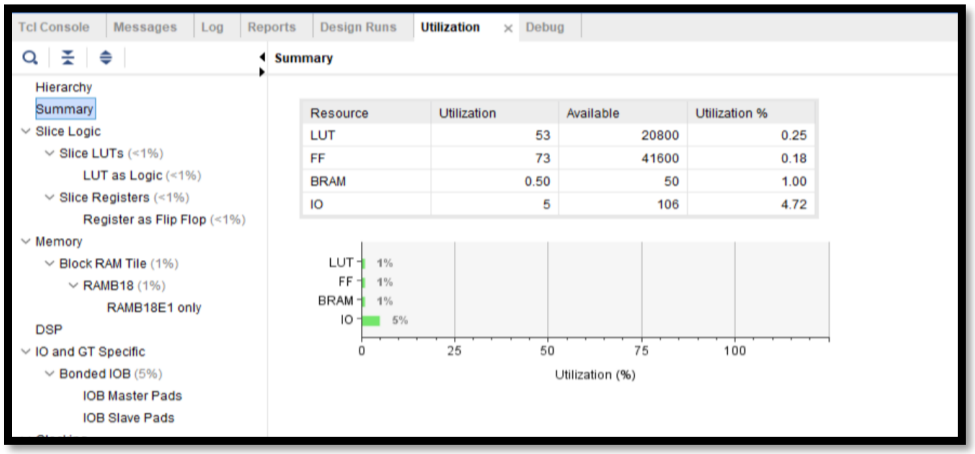


SPI_Slave Schematic

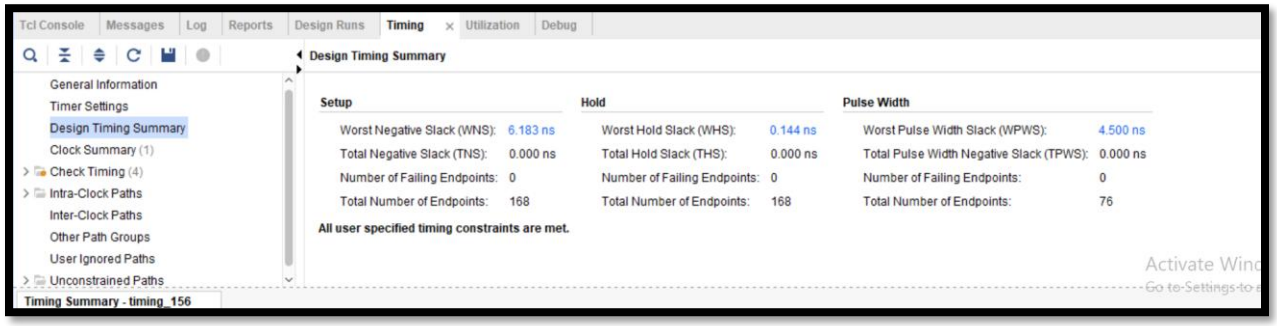


Message

Utilization Report

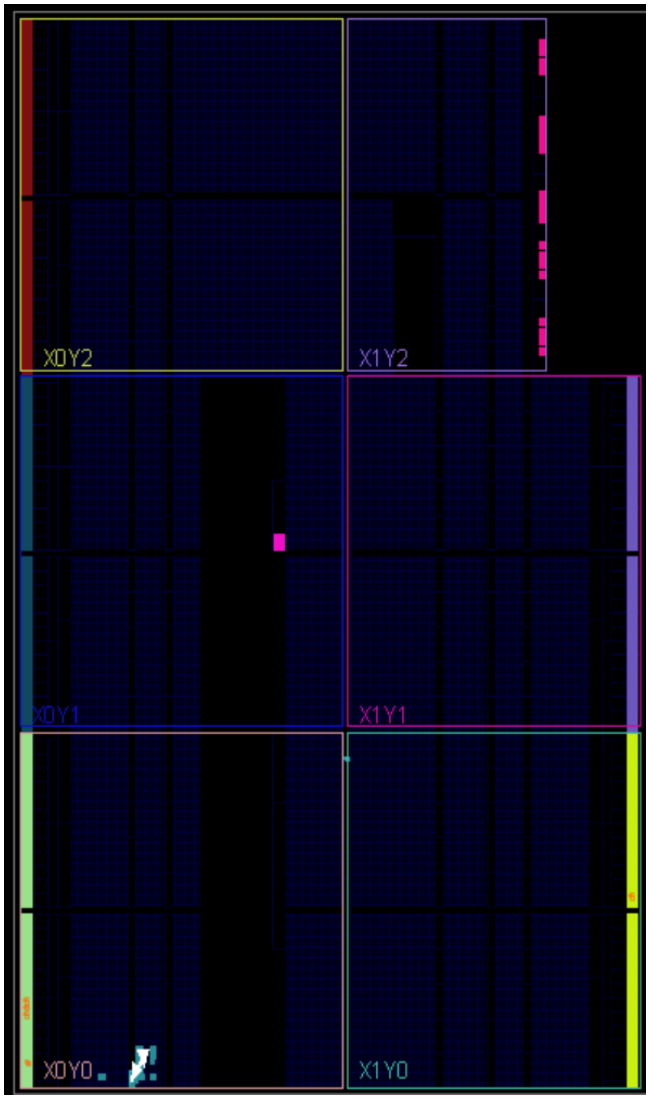


Timing Report

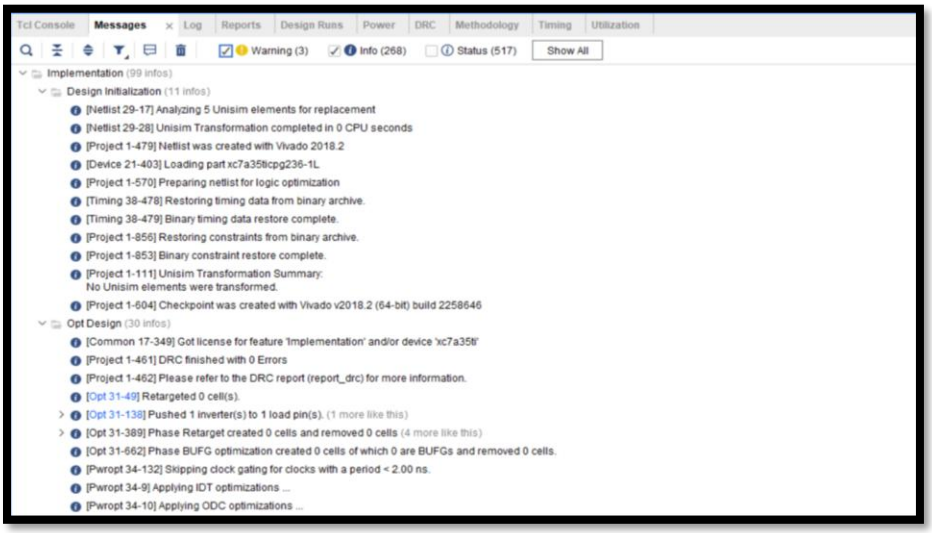


Implementation (gray encoding)

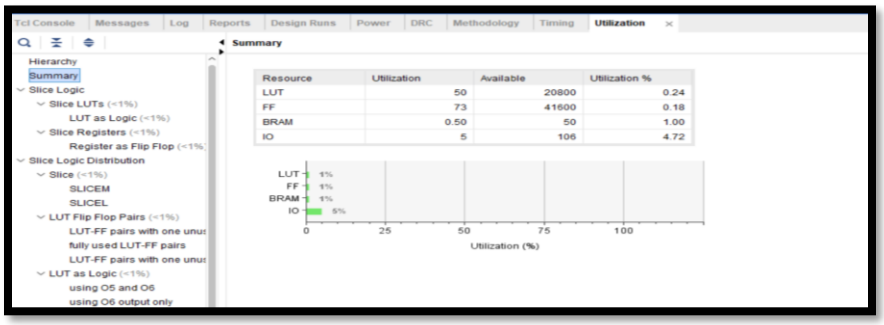
Device



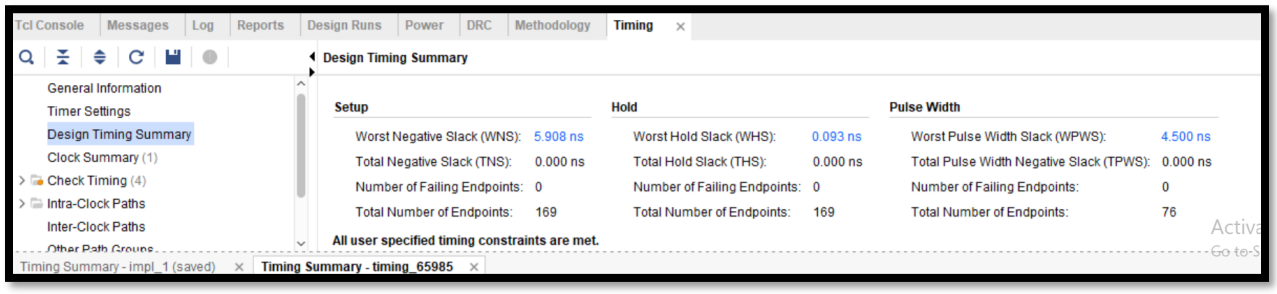
Message



Utilization Report



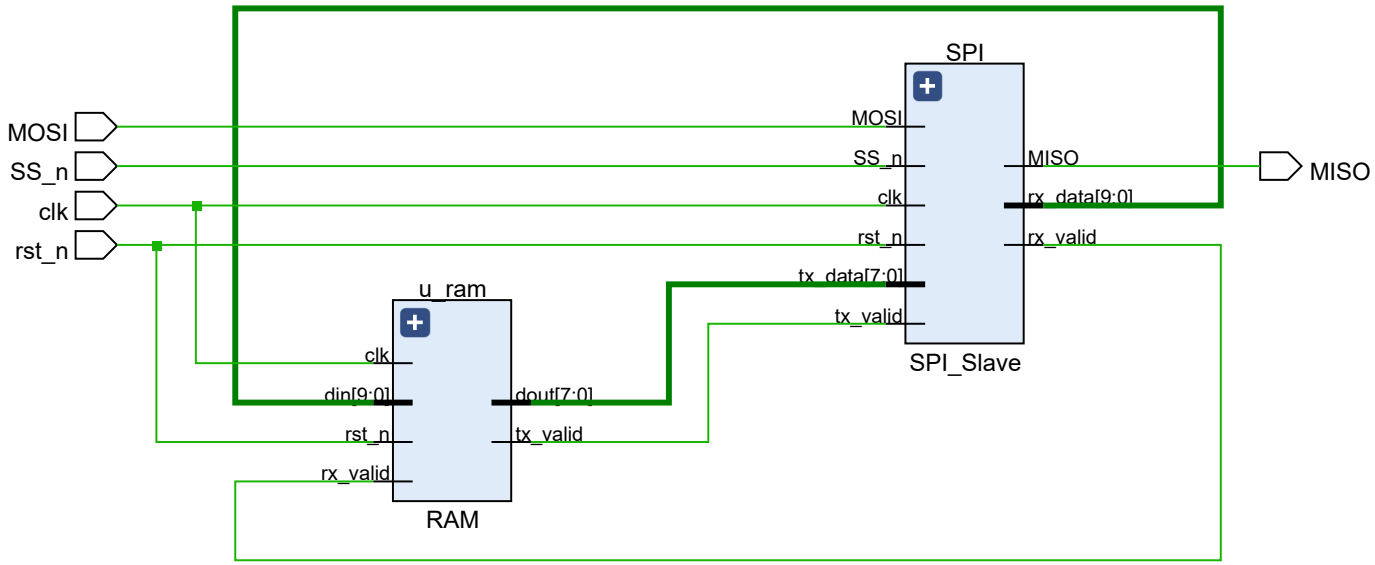
Timing Report



Elaboration (one_hot encoding)

Message

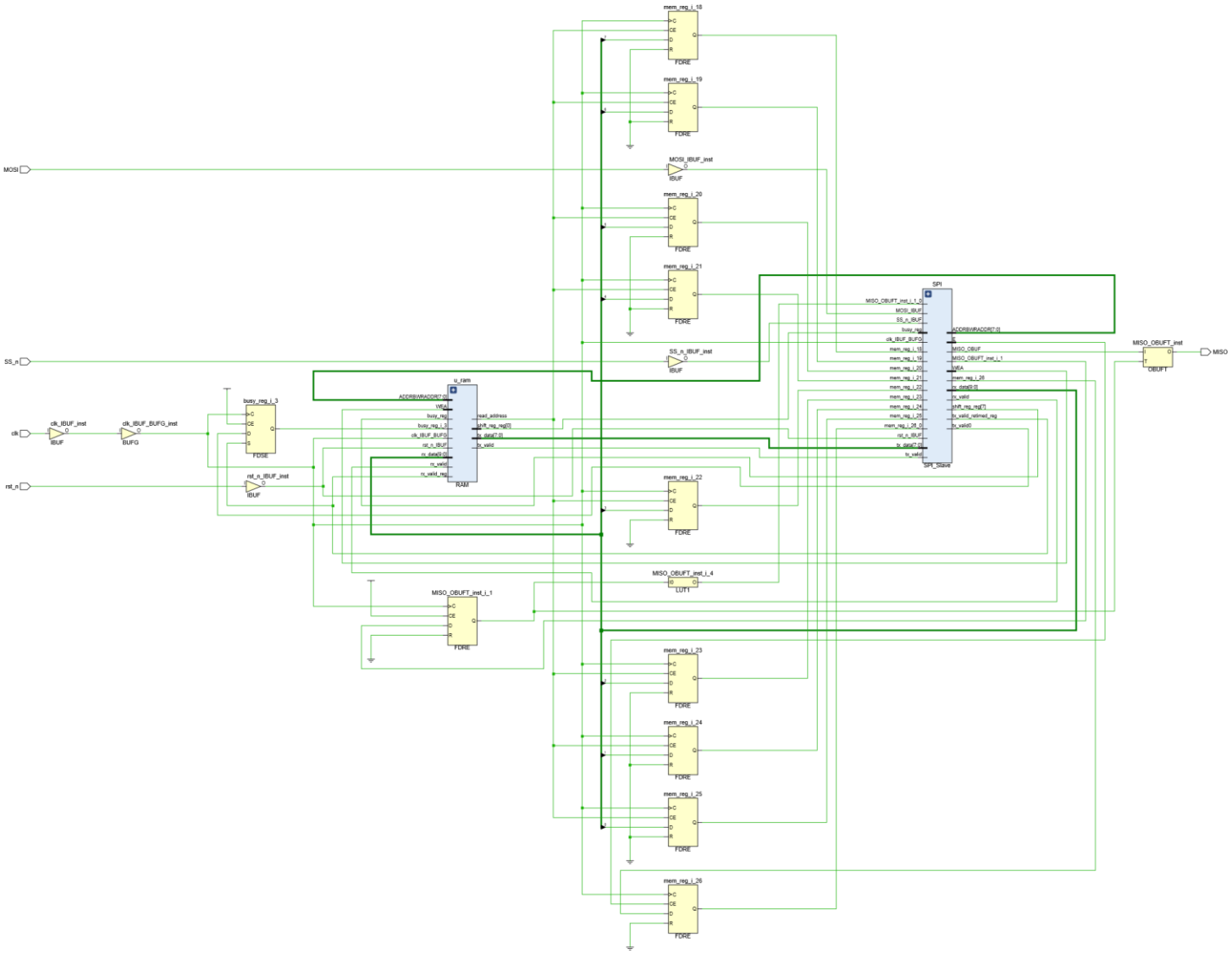
Schematic



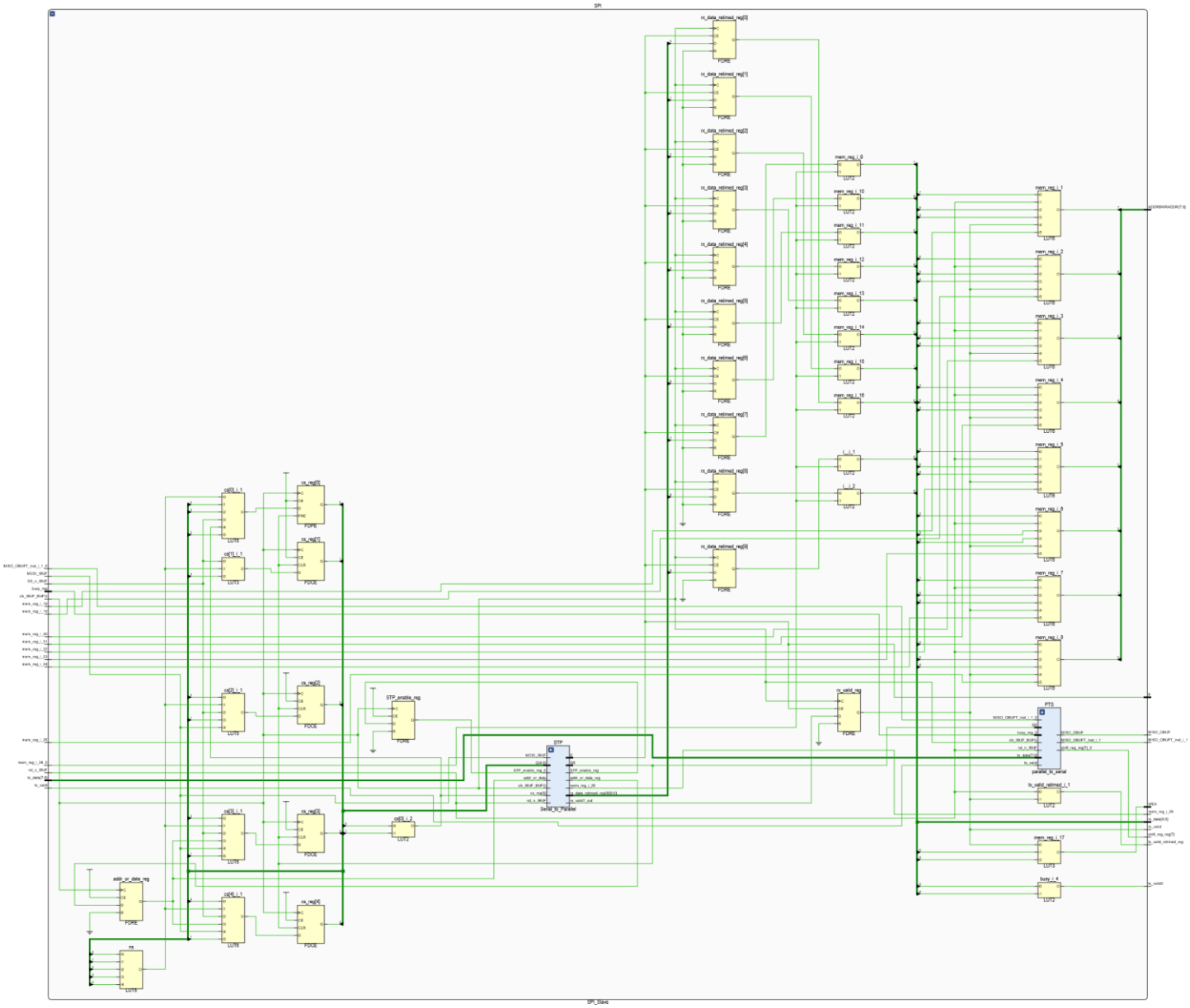
Synthesis (one_hot encoding)

State	Encoding
IDLE	5’b00001
CHK_CMD	5’b00010
WRITE	5’b00100
READ_ADD	5’b01000
READ_DATA	5’b10000

Schematic

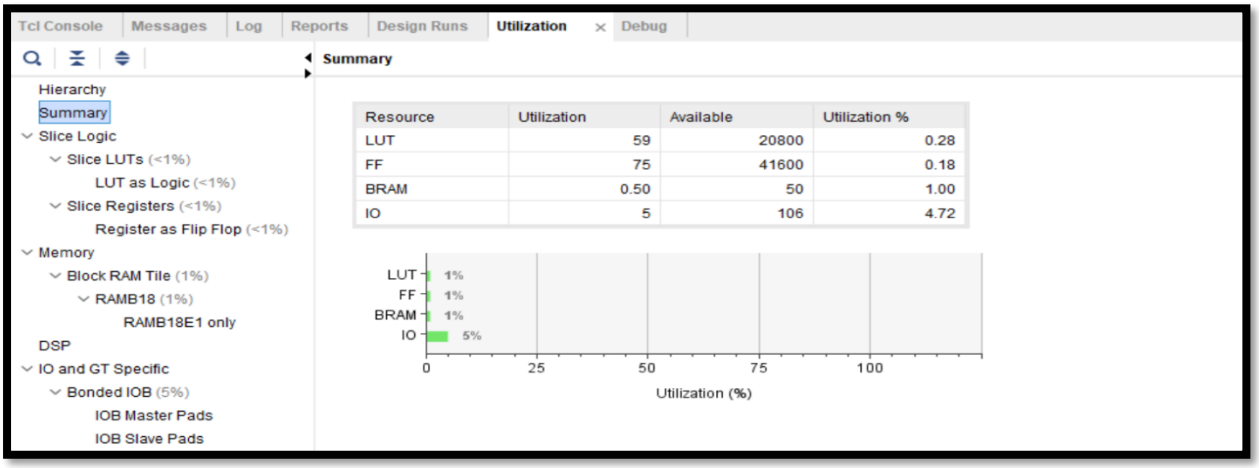


SPI_Slave Schematic

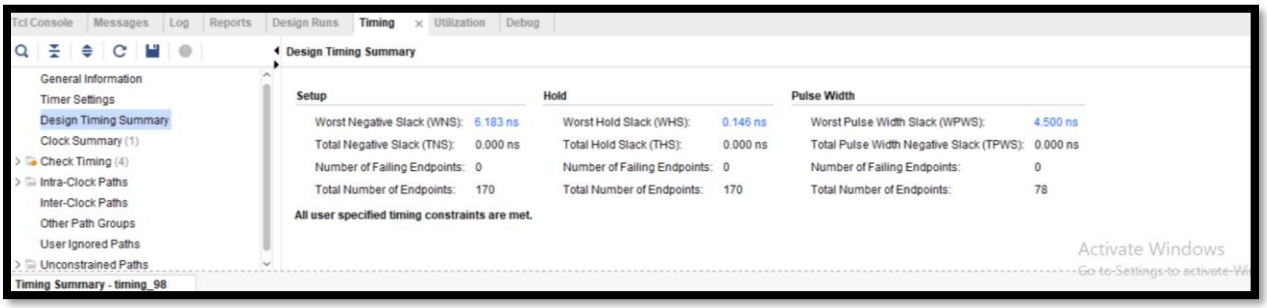


Message

Utilization Report

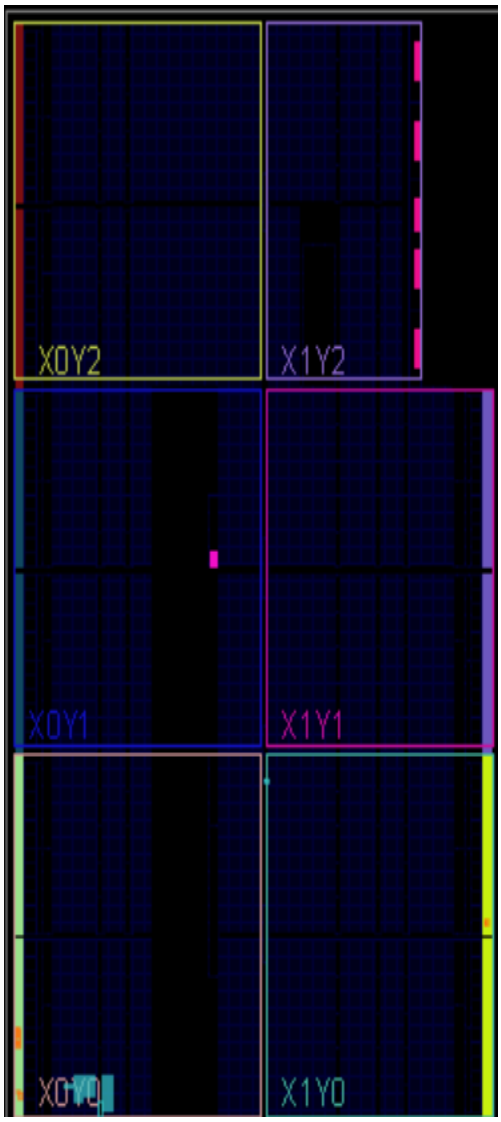


Timing Report

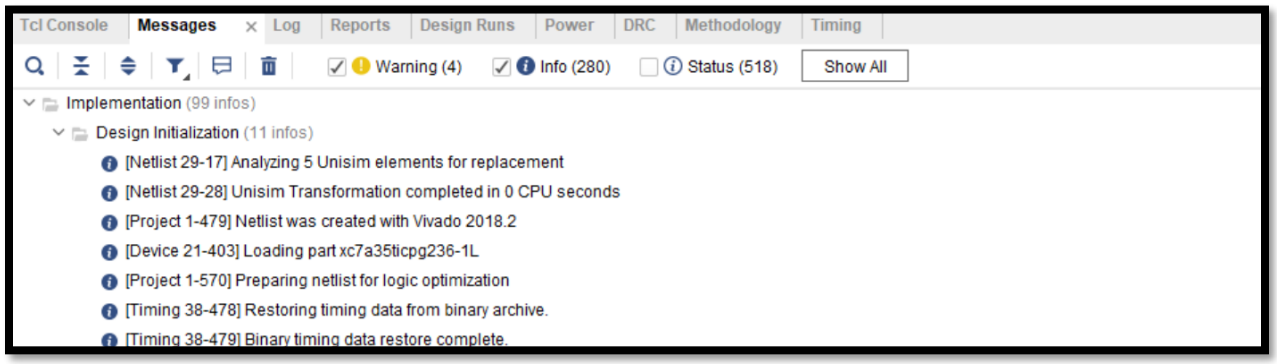


Implementation (one_hot encoding)

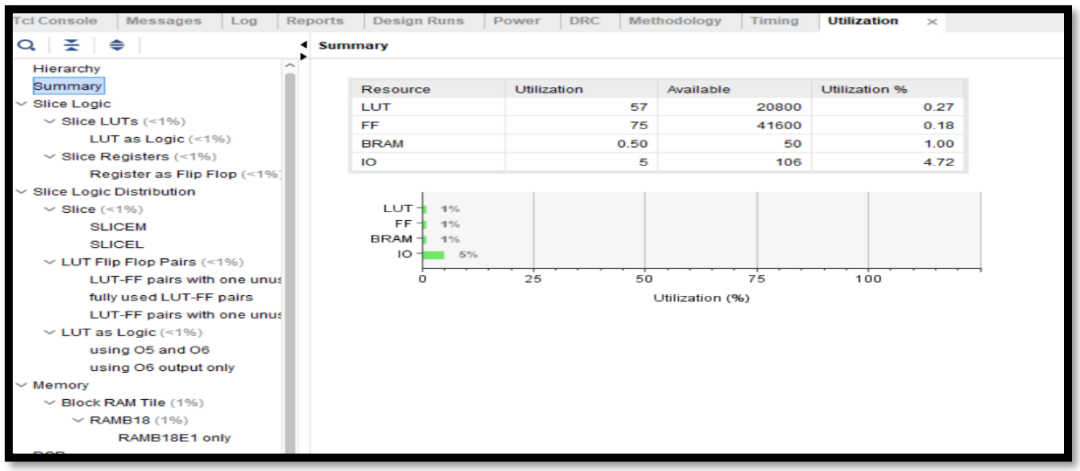
Device



Message



Utilization Report



Timing Report