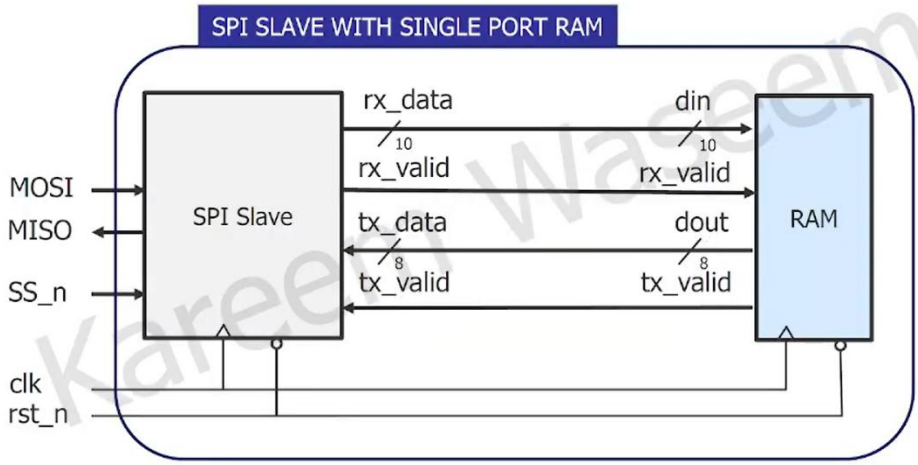
SPI Slave with Single Port RAM

Prepared By :

Mazen Mohamed Hemdan

Menna Allah Mohamed Farag Mahmoud



SPI Interface

- **Protocol:** SPI (Serial Peripheral Interface)
- **Wires:**
  - MOSI – Master Out, Slave In (for sending data/commands to the slave)
  - MISO – Master In, Slave Out (for receiving data from the slave)
  - SCK – Serial Clock (synchronizes data transfer)
  - SS_n – Active-low Slave Select
- Finite State Machine (FSM) based design with states like :

  IDLE, CHK_CMD, WRITE, READ_ADD, READ_DATA

RAM Module

- **Type:** Single-port synchronous RAM
- **Parameters:**
  - MEM_DEPTH = 256
  - ADDR_SIZE = 8 bits
- **Ports:**
  - din (10 bits) – input data
  - dout (8 bits) – output data
  - rx_valid, tx_valid – control signals
  - clk, rst_n – clock and active-low reset

➢ RTL Verilog Code :

- RAM Design :

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

parameter MEM_WIDTH = 8;
parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;

input clk,rst_n,rx_valid;
input [9:0] din;
output reg tx_valid;
output reg [7:0] dout;
reg [ADDR_SIZE-1:0] addr;

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

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

- SPI Slave Design :

✓ State Memory :

```
module SPI_SLAVE(clk,rst_n,SS_n,rx_data,rx_valid,tx_data,tx_valid,MOSI,MISO);
parameter IDLE = 3'b000;
parameter CHK_CMD = 3'b001;
parameter WRITE = 3'b010;
parameter READ_ADD = 3'b011;
parameter READ_DATA = 3'b100;


input clk,rst_n,SS_n,MOSI,tx_valid;
input [7:0] tx_data;
output reg MISO ,rx_valid;
output [9:0] rx_data;
reg [9:0] data;
(* fsm_encoding = "sequential" *)
reg [2:0] cs,ns;
reg read_state;
reg [4:0] counter;

// STATE MEMORY
always@(posedge clk)begin
    if(!rst_n) begin cs <= IDLE; read_state <= 0; end
    else begin
        cs <= ns;
        if ((cs == READ_ADD || cs == READ_DATA) && SS_n)
        read_state <= ~read_state;
    end
end
```

✓ Next State :

```
// NEXT STATE
always@(cs,SS_n,MOSI)begin
    case(cs)
    IDLE : begin
        if(SS_n) ns=IDLE;
        else ns=CHK_CMD;
    end
    CHK_CMD : begin
        if(SS_n) ns=IDLE;
        else begin
            if(!MOSI) ns=WRITE;
            else if(!read_state ) ns=READ_ADD;
            else ns=READ_DATA;
        end
    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 : ns = IDLE;
    endcase
end
```

✓ Output Logic :

```
// OUTPUT LOGIC
always@(posedge clk)begin
    if (!rst_n) begin
        rx_valid <= 0;
        counter <= 0;
        MISO <= 0;
    end else begin
    case(cs)
    IDLE : begin rx_valid<=0; counter<=0; end
    WRITE : begin
        counter <=counter+1; data[9-counter] <= MOSI; rx_valid <=0;if(counter ==9)rx_valid <=1;
    end
    READ_ADD : begin
        counter <=counter+1; data[9-counter] <= MOSI; rx_valid <=0;if(counter ==9)rx_valid <=1;
    end
    READ_DATA : begin
        if(!tx_valid)begin
            if(counter == 10)begin counter <=0;  end
            else if(counter < 10 && !rx_valid)begin counter <=counter+1; data[9-counter] <= MOSI; rx_valid <=0;if(counter ==9)rx_valid <=1; end
        end
        else if(tx_valid)begin
            if(counter == 8)begin counter <=0; end
            else if(counter < 8)begin counter <=counter+1; MISO <= tx_data[7-counter];rx_valid<=0;end
        end
    end
    endcase
    end
end
assign rx_data = (rx_valid)?data:rx_data;
endmodule
```

- SPI Wrapper Desgin :

```
module SPI_WRAPPER(
    input clk,
    input rst_n,
    input SS_n,
    input MOSI,
    output MISO
);

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

    // SPI_SLAVE instantiation
    SPI_SLAVE spi_slave_inst (
        .clk(clk),
        .rst_n(rst_n),
        .SS_n(SS_n),
        .rx_data(rx_data),
        .rx_valid(rx_valid),
        .tx_data(tx_data),
        .tx_valid(tx_valid),
        .MOSI(MOSI),
        .MISO(MISO)
    );

    // RAM instantiation
    RAM ram_inst (
        .clk(clk),
        .rst_n(rst_n),
        .rx_valid(rx_valid),
        .din(rx_data),
        .tx_valid(tx_valid),
        .dout(tx_data)
    );

endmodule
```

➢ Testbench Code :

- SPI Wrapper Instantiation and Clock Generation :

```
module SPI_WRAPPER_tb;

    // Testbench signals
    reg clk;
    reg rst_n;
    reg SS_n;
    reg MOSI;
    wire MISO;


    // Instantiate SPI_WRAPPER
    SPI_WRAPPER uut (
        .clk(clk),
        .rst_n(rst_n),
        .SS_n(SS_n),
        .MOSI(MOSI),
        .MISO(MISO)
    );


    // Internal signals for self checking
    reg [7:0] address;
    reg [7:0] data;
    reg [7:0] read_data;


    // Clock generation
    initial begin
        clk = 0;
        forever #1 clk = ~clk;
    end
```

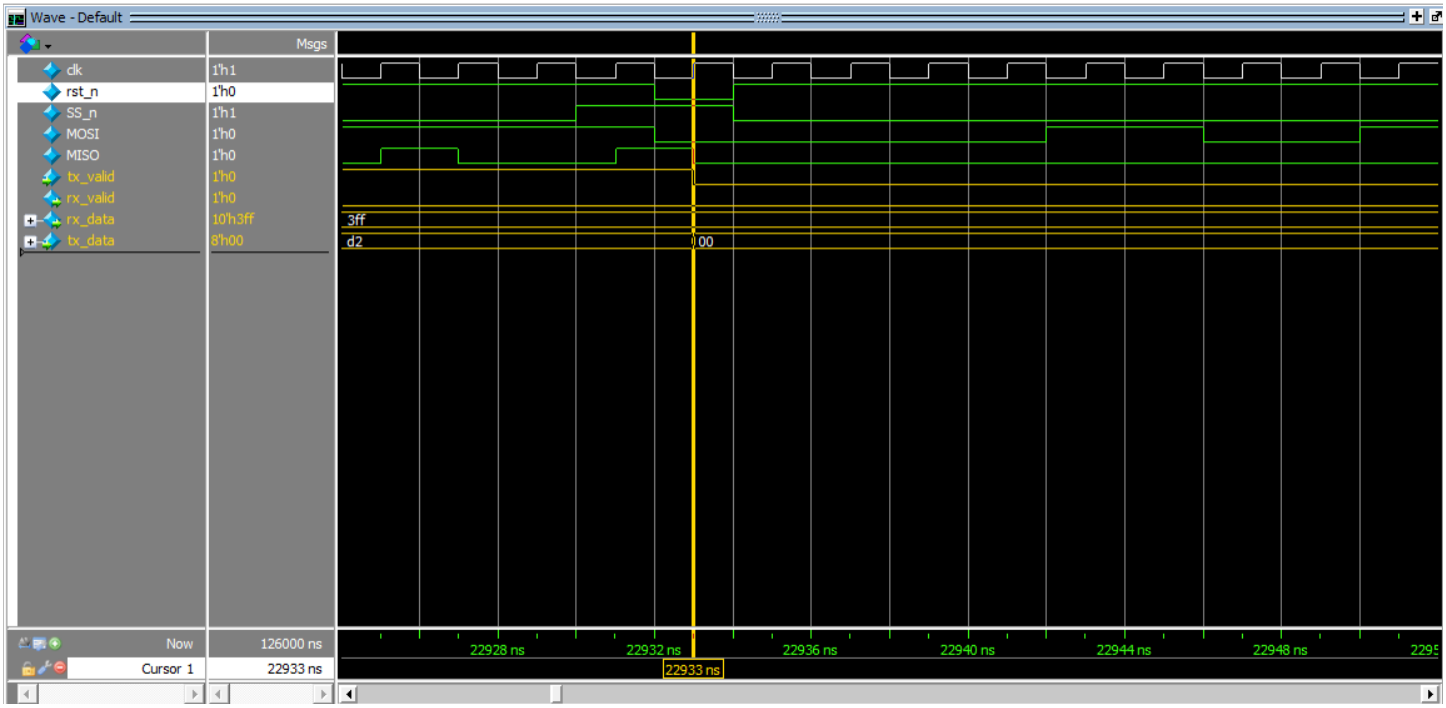
- Memory Instantiation with zeros :

```
integer i;
initial begin
    $readmemh("mem.dat",uut.ram_inst.mem);
```

- RST check :

Each iteration checks on the 4 states :

```
repeat(1000)begin
            // rst check
            rst_n = 0;
            MOSI = 0;
            address = 0;
            data = 0;
            read_data = 0;
            @(negedge clk);
            if (MISO!=0) begin
                $display("rst check failed");
                $stop;
            end
            else
                $display("rst check passed");

            rst_n = 1;
```



- Write Address Check :

```
//Write Address check
SS_n = 0;
@(negedge clk);
// CHK_CMD
MOSI = 0;
repeat(3)@(negedge clk);
// receive wr_address
for(i=7;i>=0;i=i-1)begin
    MOSI = $random;
    address[i] = MOSI;
    @(negedge clk);
end
SS_n = 1;
@(negedge clk);
// IDLE
if (uut.ram_inst.addr != address) begin
    $display("write address check failed");
    $stop;
end
else
    $display("write address check passed");
```

- Write Data Check :

```
//Write Data check
SS_n = 0;
@(negedge clk);
// CHK_CMD
MOSI = 0;
repeat(2)@(negedge clk);
MOSI = 1;
@(negedge clk);
// Receive wr_data
for(i=7;i>=0;i=i-1)begin
    MOSI = $random;
    data[i] = MOSI;
    @(negedge clk);
end
SS_n = 1;
@(negedge clk);
// IDLE
if (uut.ram_inst.mem[address] != data) begin
    $display("write data check failed");
    $stop;
end
else
    $display("write data check passed");
```

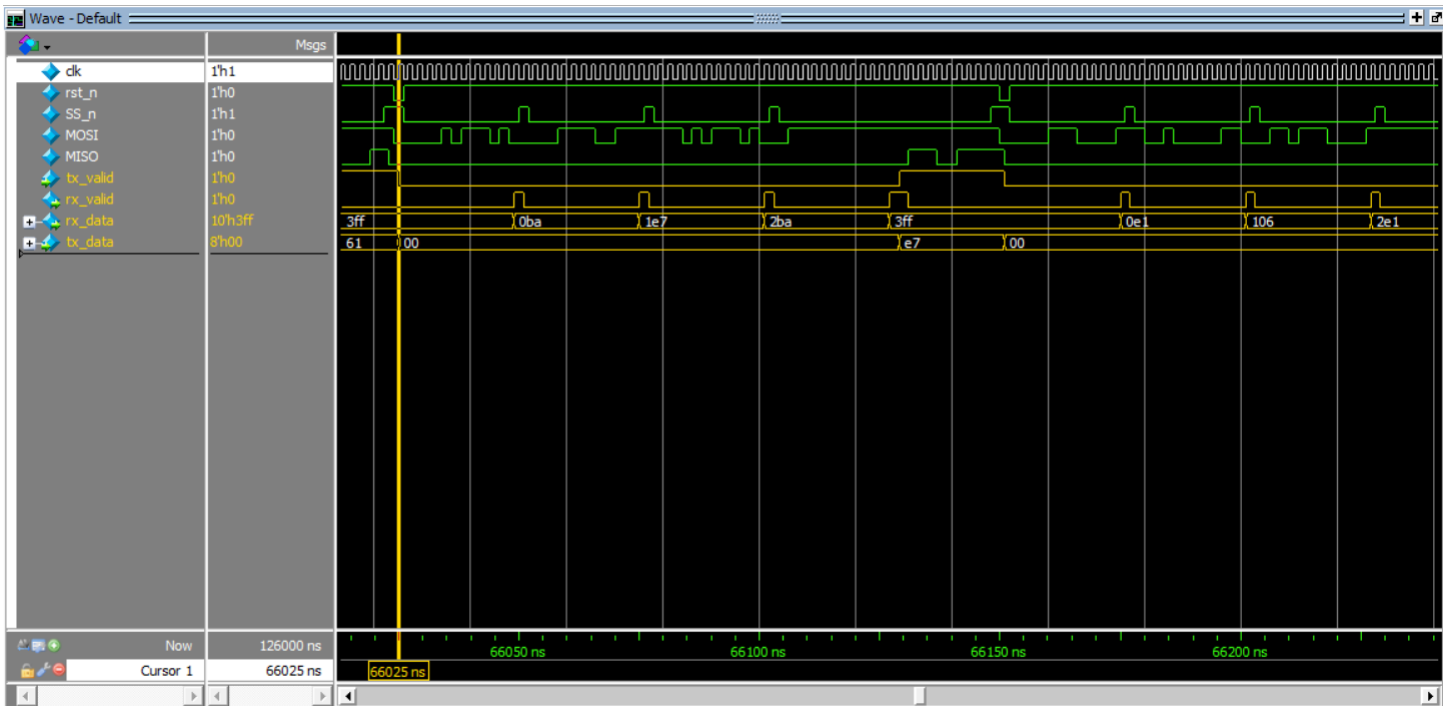
- Read Address Check :

```
//Read Address check
SS_n = 0;
@(negedge clk);
// CHK_CMD
MOSI = 1;
repeat(2)@(negedge clk);
MOSI = 0;
@(negedge clk);
// send rd_address
for(i=7;i>=0;i=i-1)begin
    MOSI = address[i];
    @(negedge clk);
end
SS_n = 1;
@(negedge clk);
// IDLE
if (uut.ram_inst.addr != address) begin
    $display("read address check failed");
    $stop;
end
else
    $display("read address check passed");
```

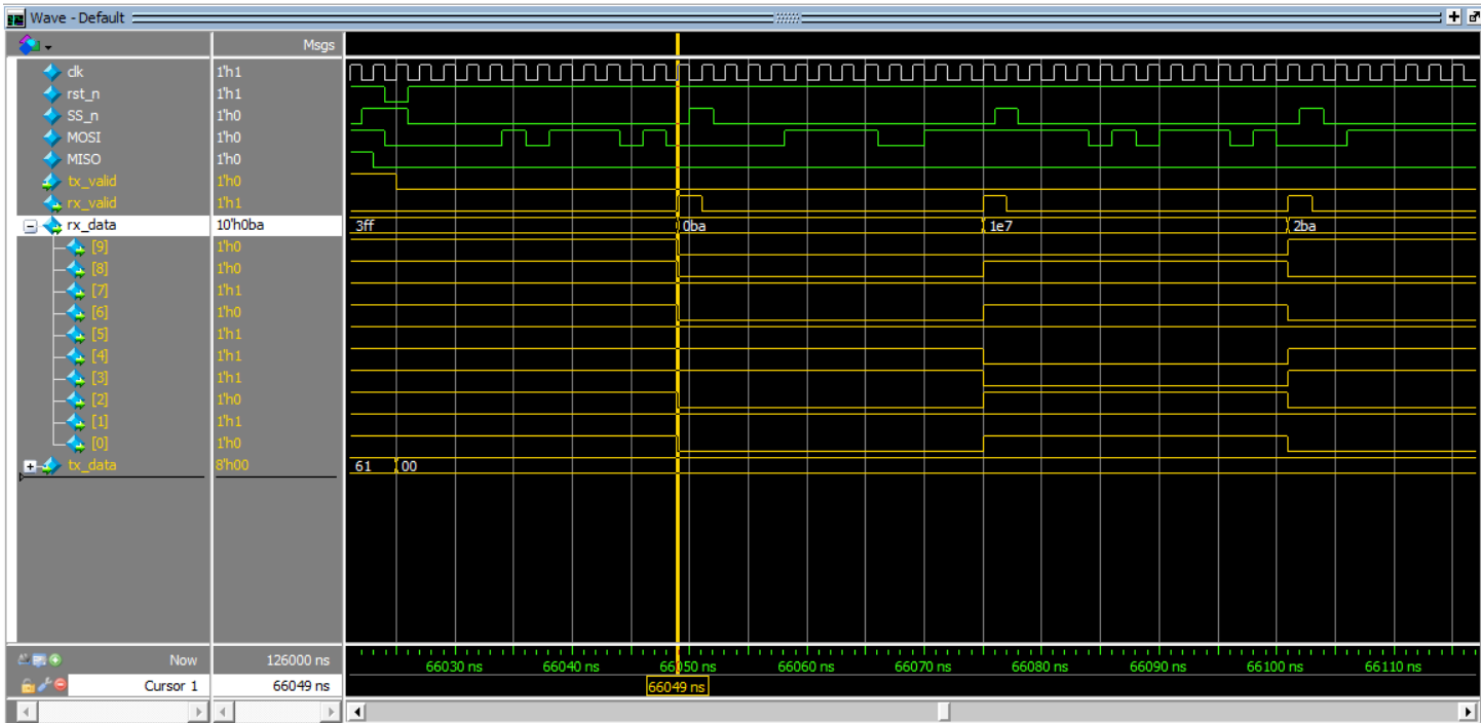
- Read Data Check :

```
        //Read Data check
        SS_n = 0;
        @(negedge clk);
        // CHK_CMD
        MOSI = 1;
        repeat(3)@(negedge clk);
        // read dumy 8 bits
        repeat(8)begin
            @(negedge clk);
        end
        // read rd_data
        repeat(2)@(negedge clk);
        for(i=7;i>=0;i=i-1)begin
            read_data[i] = MISO;
            @(negedge clk);
        end
        SS_n = 1;
        @(negedge clk);
        // IDLE
        if (read_data != uut.ram_inst.mem[address]) begin
            $display("read data check failed");
            $stop;
        end
        else
            $display("read data check passed");

    end
    $stop;
end

endmodule
```

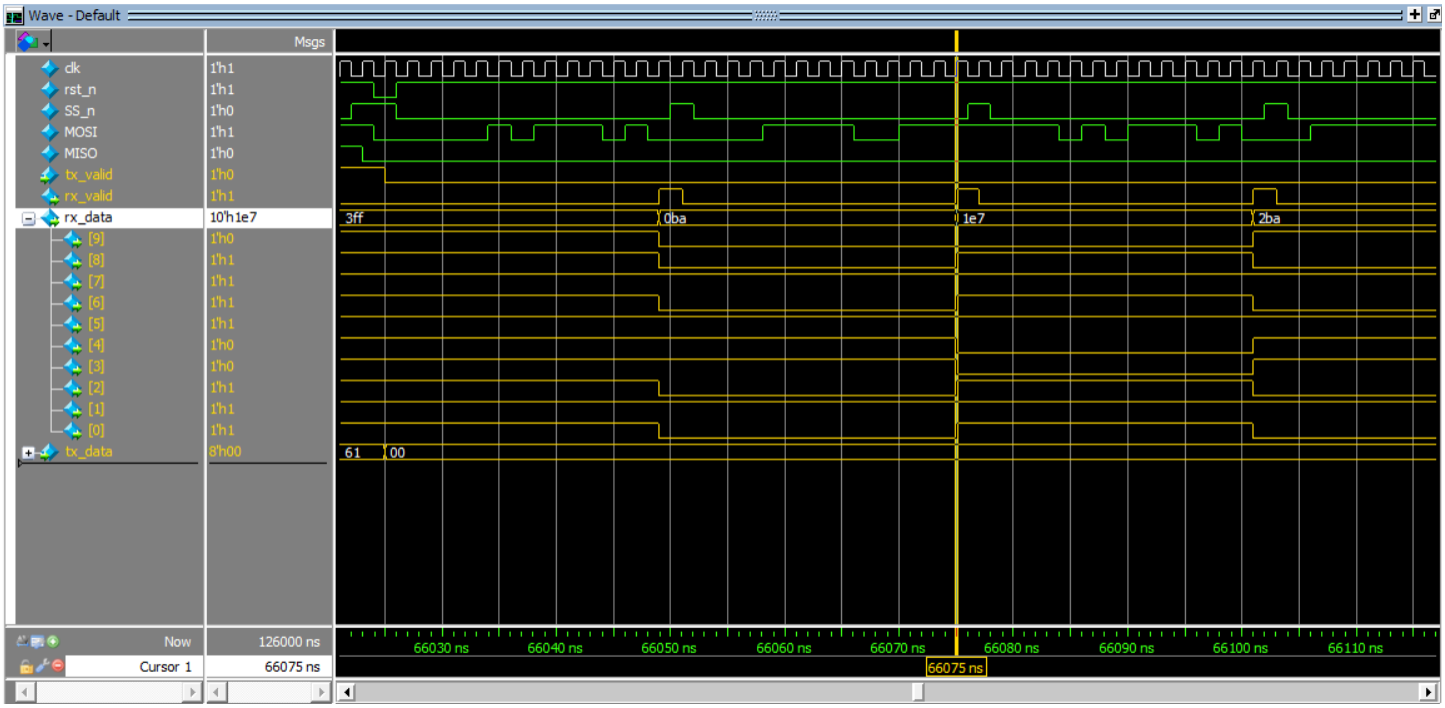
➢ Waveforms Snippets captured from QuestaSim :



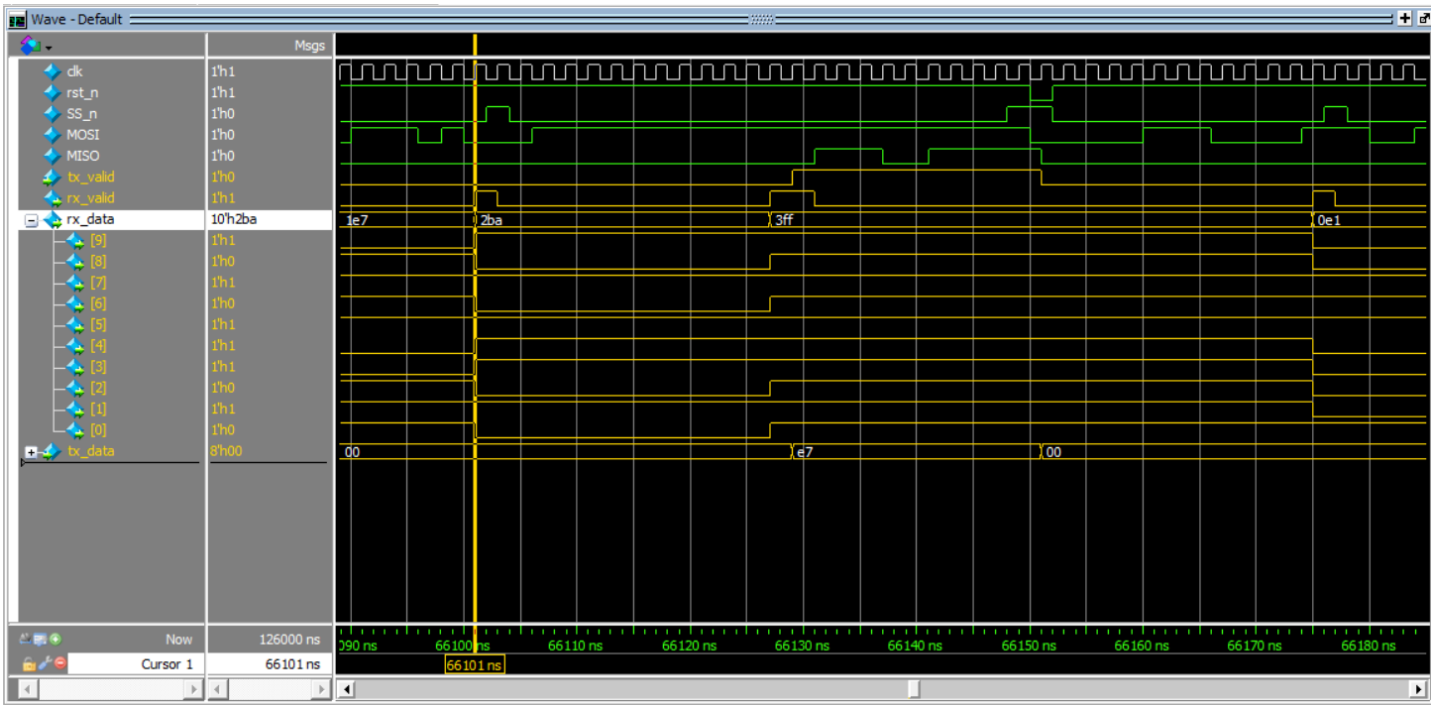
- After the first write state address is being written in rx_data as well as most significant 2 bits which are 2'b00 to declare that the data is address (ba) :



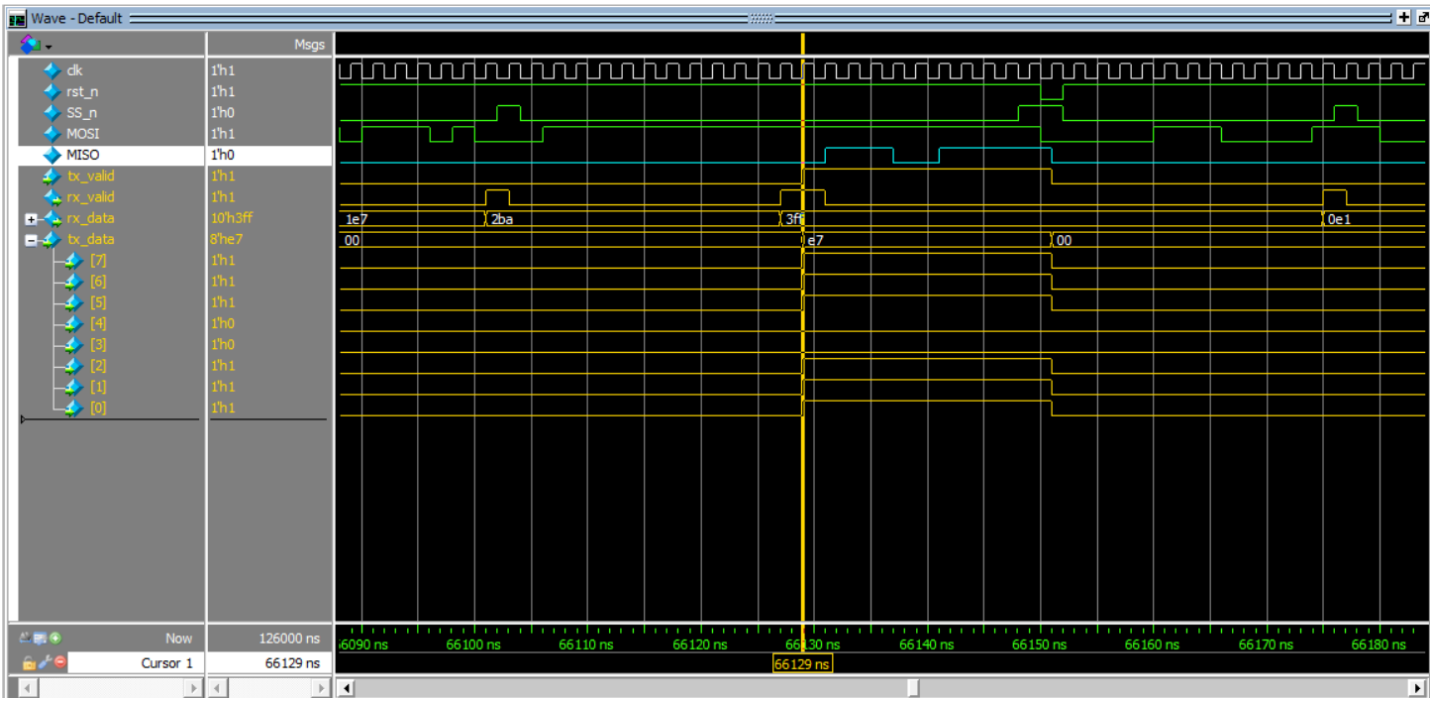
- After the second write state data is being written in rx_data as well as most significant 2 bits which are 2’b01 to declare that the data is wr_data (e7) :



- After the first read state data is being written in rx_data as well as most significant 2 bits which are 2’b10 to declare that the data is rd_add (ba)(same as the data we just wrote in) :

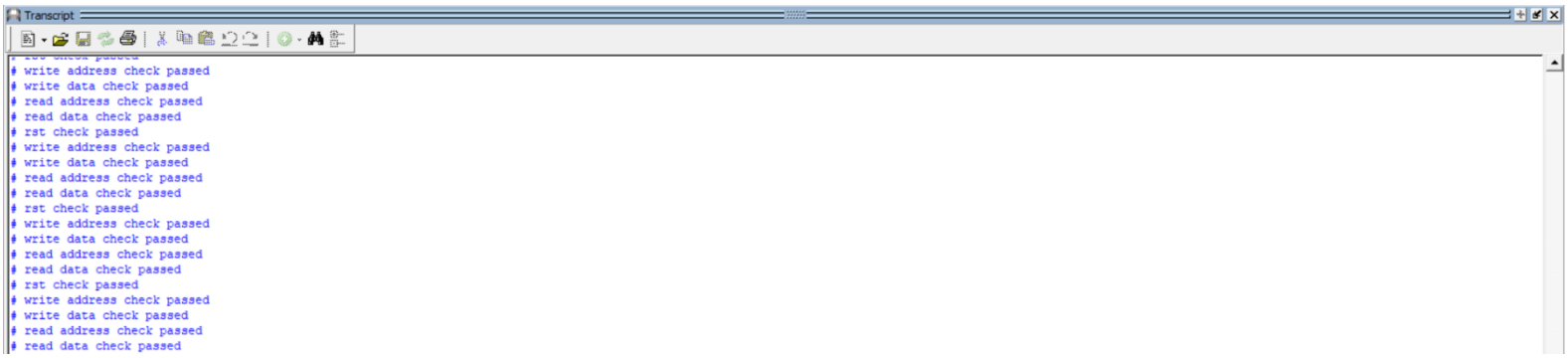


- After the second read state data is being written in rx_data as well as most significant 2 bits which are 2'b11 but the rest 8 data are dumy after that tx_valid is raised so tx_data could read the data being written in (ba) address (same as the data we just wrote in(e7)) :



- Also MISO is being transmitted bit by bit (1110 0111)

✓ Transcript Output :

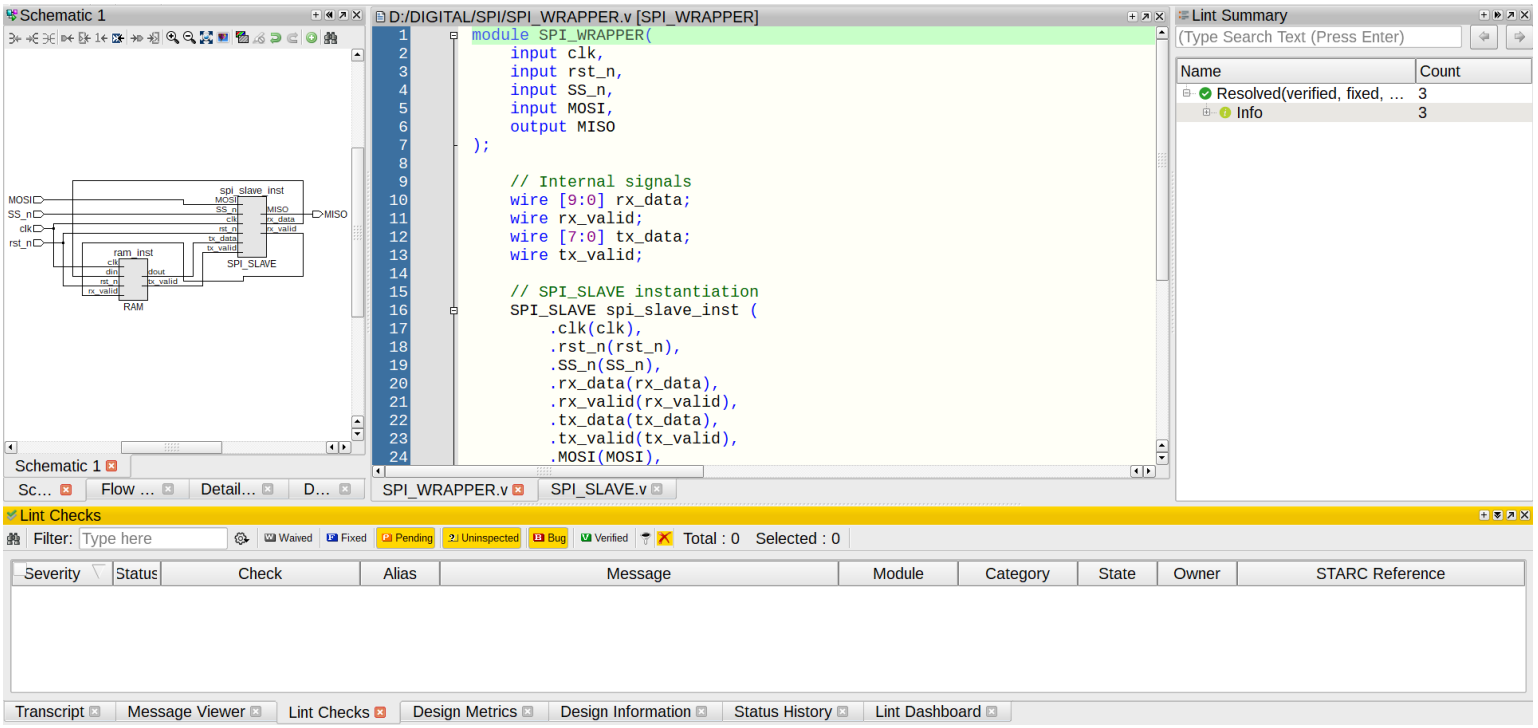


➢ DO File :

```
vlib work

vlog RAM.v SPI_SLAVE.v SPI_WRAPPER.v SPI_WRAPPER_tb.v

vsim -voptargs=+acc work.SPI_WRAPPER_tb

add wave -position insertpoint  \

sim:/SPI_WRAPPER_tb/clk \

sim:/SPI_WRAPPER_tb/rst_n \

sim:/SPI_WRAPPER_tb/SS_n \

sim:/SPI_WRAPPER_tb/MOSI \

sim:/SPI_WRAPPER_tb/MISO \

sim:/SPI_WRAPPER_tb/uut/spi_slave_inst/tx_valid \

sim:/SPI_WRAPPER_tb/uut/spi_slave_inst/rx_valid \

sim:/SPI_WRAPPER_tb/uut/spi_slave_inst/rx_data \

sim:/SPI_WRAPPER_tb/uut/spi_slave_inst/tx_data

run -all
```

```
VSIM 83> do run_spi.do
# ** Warning: (vlib-34) Library already exists at "work".
# Errors: 0, Warnings: 1
# QuestaSim-64 vlog 2021.1 Compiler 2021.01 Jan 19 2021
# Start time: 23:00:42 on Jul 30,2025
# vlog -reportprogress 300 RAM.v SPI_SLAVE.v SPI_WRAPPER.v SPI_WRAPPER_tb.v
# -- Compiling module RAM
# -- Compiling module SPI_SLAVE
# -- Compiling module SPI_WRAPPER
# -- Compiling module SPI_WRAPPER_tb
#
# Top level modules:
#       SPI_WRAPPER_tb
# End time: 23:00:42 on Jul 30,2025, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# ** Error: 栝散獮桒□敤楮潴Ø
#
#
#        Unable to replace existing ini file (D:/DIGITAL/SPI/SPI.mpf).  File can not be renamed.
# ** Error: 栝散獮桒□敤楮潴Ø
#
#
#        Unable to replace existing ini file (D:/DIGITAL/SPI/SPI.mpf).  File can not be renamed.
# End time: 23:00:47 on Jul 30,2025, Elapsed time: 0:01:31
# Errors: 2, Warnings: 1
# vsim -voptargs="+acc" work.SPI_WRAPPER_tb
# Start time: 23:00:47 on Jul 30,2025
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading work.SPI_WRAPPER_tb(fast)
# Loading work.SPI_WRAPPER(fast)
# Loading work.SPI_SLAVE(fast)
# Loading work.RAM(fast)
# ** Note: $stop    : SPI_WRAPPER_tb.v(153)
#    Time: 126 us  Iteration: 1  Instance: /SPI_WRAPPER_tb
# Break in Module SPI_WRAPPER_tb at SPI_WRAPPER_tb.v line 153
```

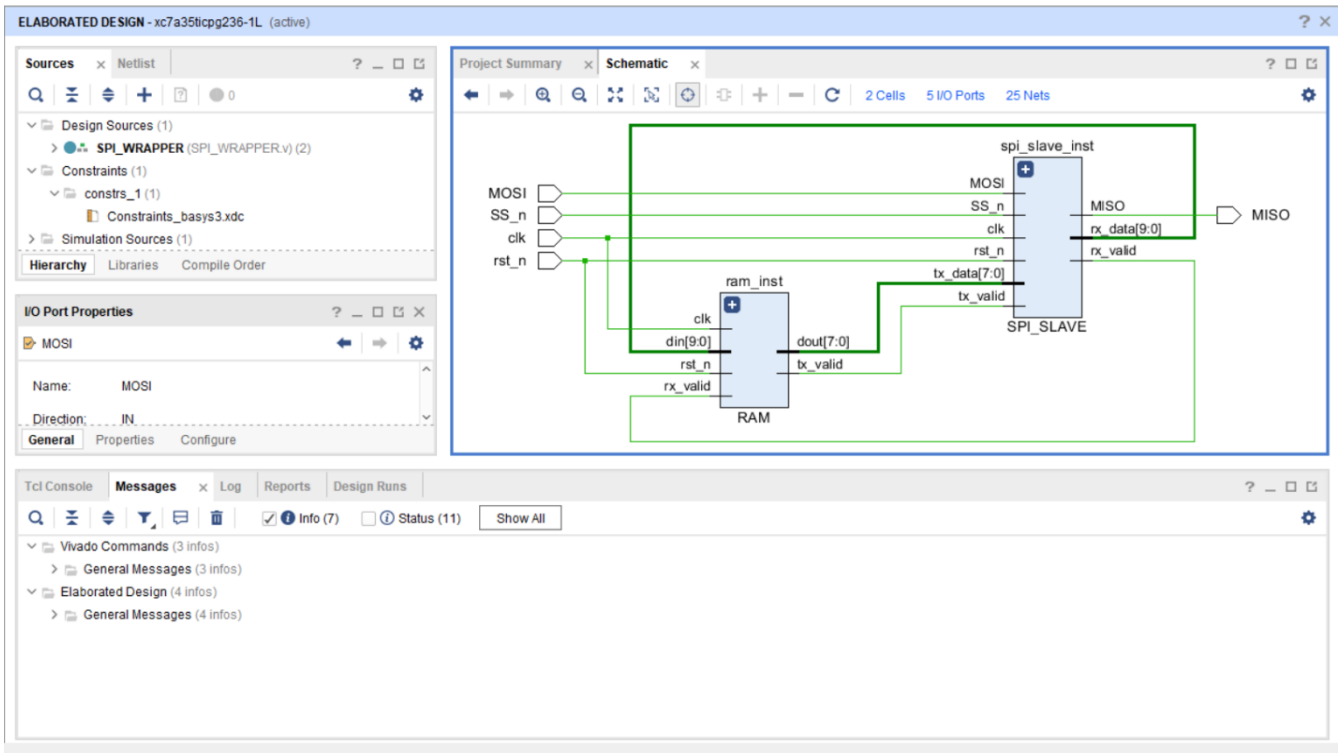
➢ QuestaLint Snippets Showing 0 Warning or Error :



➢ 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 E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
#set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
```

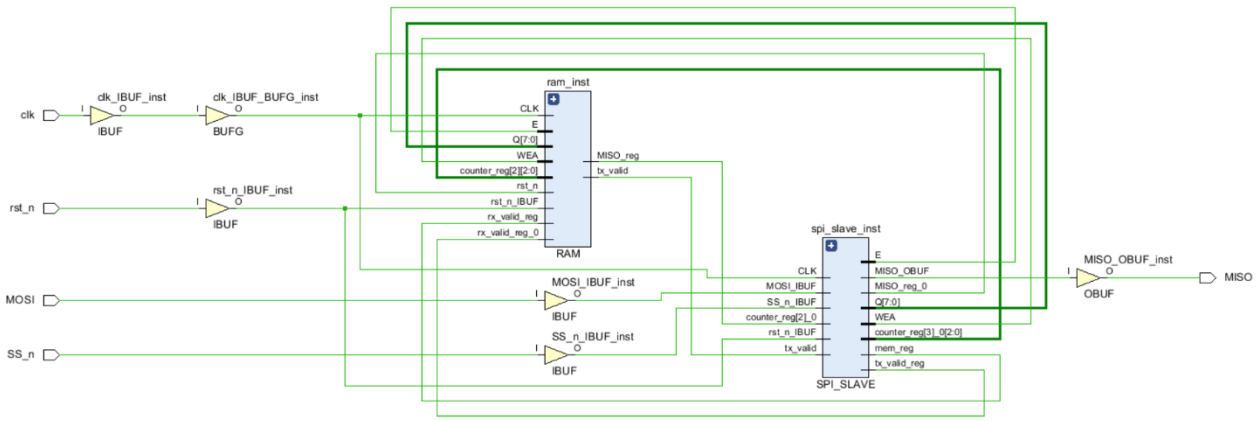
➢ RTL Schematic + Message :



➢ Synthesis Snippets :

○ Sequential Encoding :

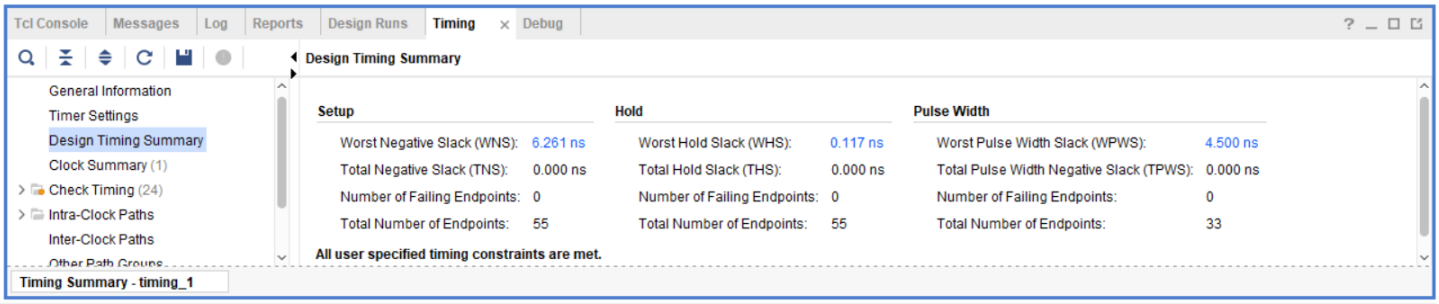
✓ Schematic :

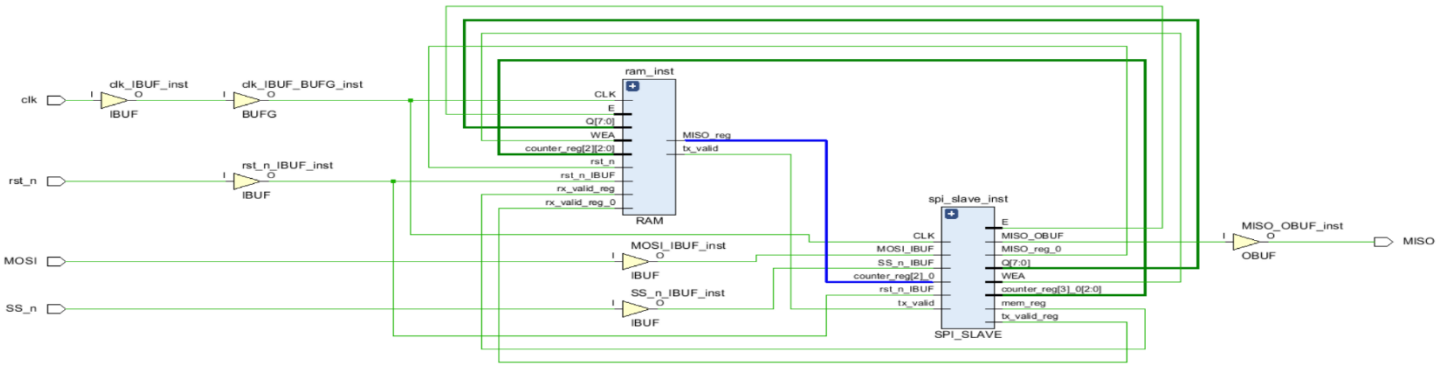


✓ Synthesis report showing the encoding used :

```
-------------------------------------------------------------------------------------------------
                State |                     New Encoding |                Previous Encoding
-------------------------------------------------------------------------------------------------
                 IDLE |                              000 |                             000
              CHK_CMD |                              001 |                             001
                WRITE |                              010 |                             010
             READ_ADD |                              011 |                             011
            READ_DATA |                              100 |                             100
-------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'sequential' in module 'SPI_SLAVE'
WARNING: [Synth 8-327] inferring latch for variable 'rx_data' [D:/DIGITAL/SPI/SPI_SLAVE.v:88]
---------------------------------------------------------------------------------
```

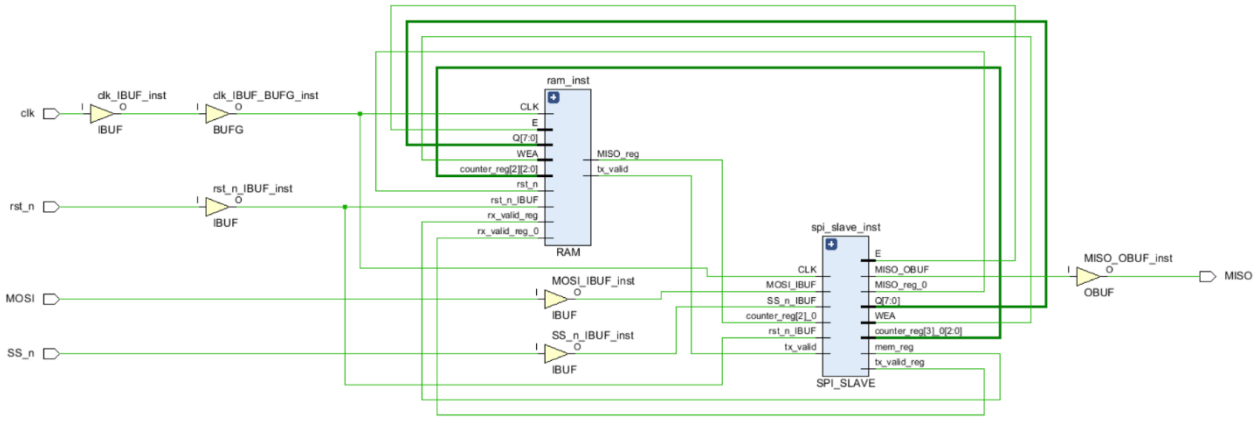
✓ Timing report & Snippet of the critical path :





○ Gray Encoding :

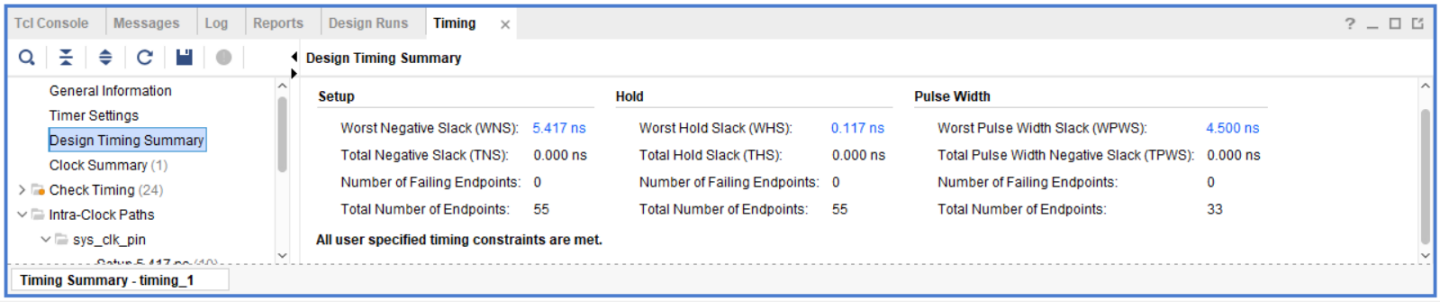
✓ Schematic :

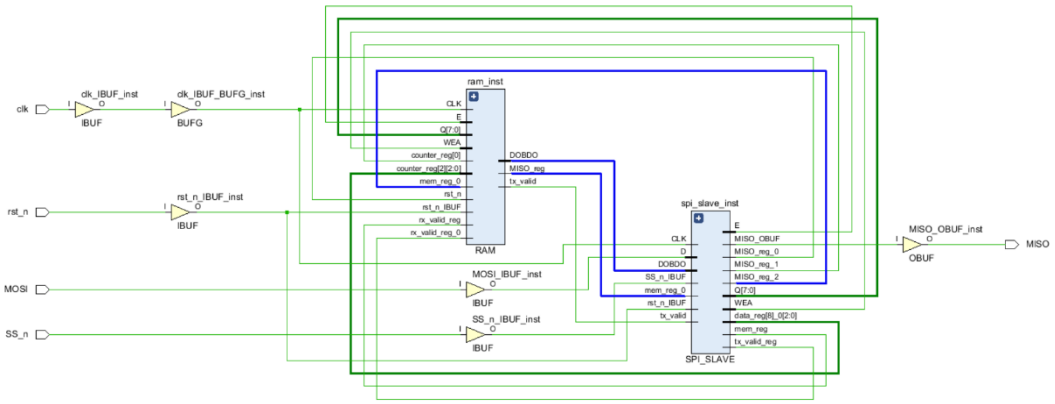


✓ Synthesis report showing the encoding used :

```
-----------------------------------------------------------------------------------------------------
                   State |                     New Encoding |                Previous Encoding
-----------------------------------------------------------------------------------------------------
                    IDLE |                              000 |                             000
                 CHK_CMD |                              001 |                             001
                   WRITE |                              011 |                             010
                READ_ADD |                              010 |                             011
               READ_DATA |                              111 |                             100
-----------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'gray' in module 'SPI_SLAVE'
WARNING: [Synth 8-327] inferring latch for variable 'rx_data' [D:/DIGITAL/SPI/SPI_SLAVE.v:88]
-----------------------------------------------------------------------------------------------------
```

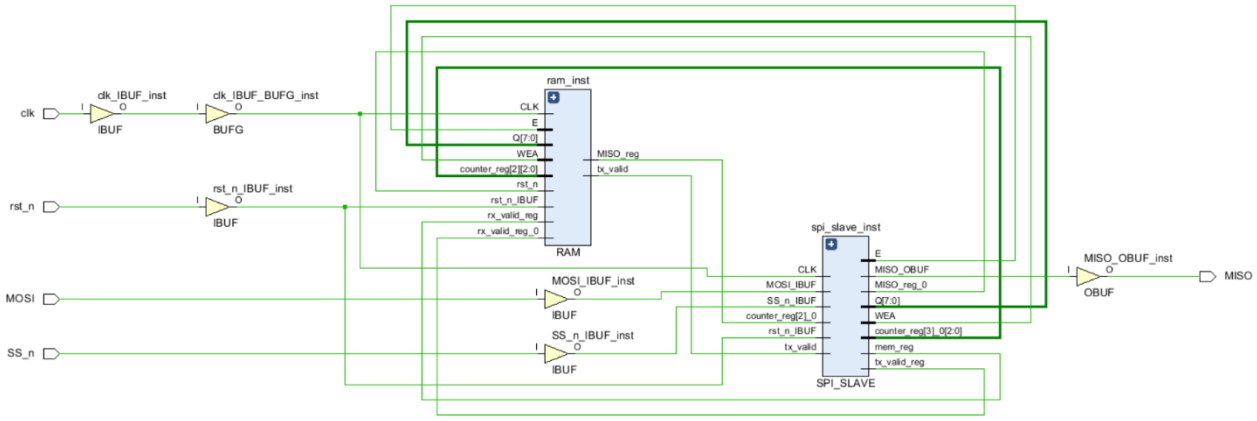
✓ Timing report & Snippet of the critical path :





○ One_Hot Encoding :

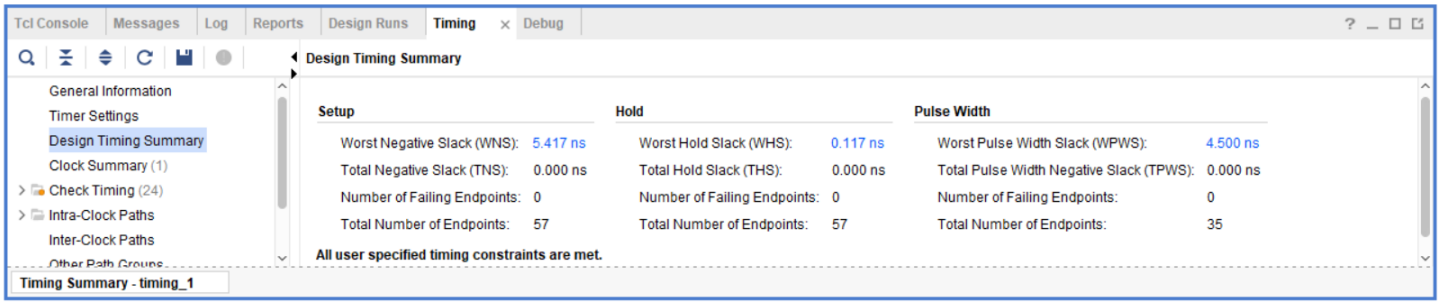
✓ Schematic :

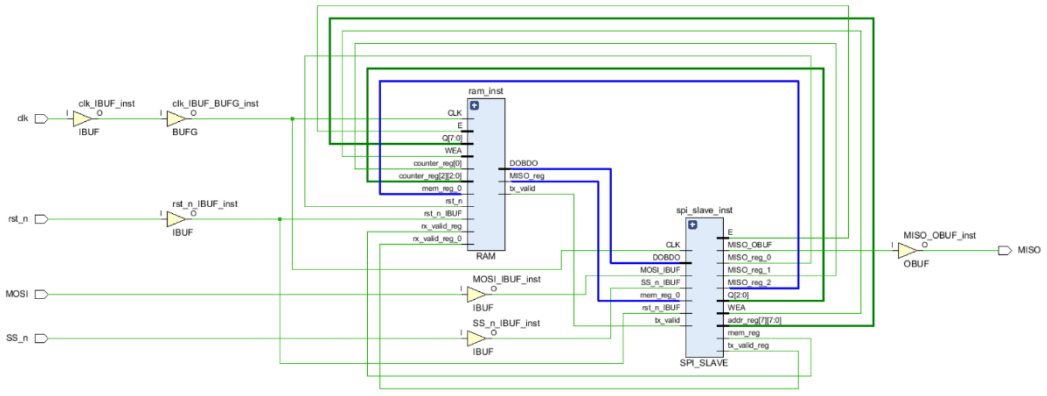


✓ Synthesis report showing the encoding used :

```
-------------------------------------------------------------------------------------------------
               State |                     New Encoding |                Previous Encoding
-------------------------------------------------------------------------------------------------
                IDLE |                            00001 |                             000
              CHK_CMD |                            00010 |                             001
               WRITE |                            00100 |                             010
            READ_ADD |                            01000 |                             011
           READ_DATA |                            10000 |                             100
-------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'one-hot' in module 'SPI_SLAVE'
WARNING: [Synth 8-327] inferring latch for variable 'rx_data' [D:/DIGITAL/SPI/SPI_SLAVE.v:88]
---------------------------------------------------------------------------
```

✓ Timing report & Snippet of the critical path :



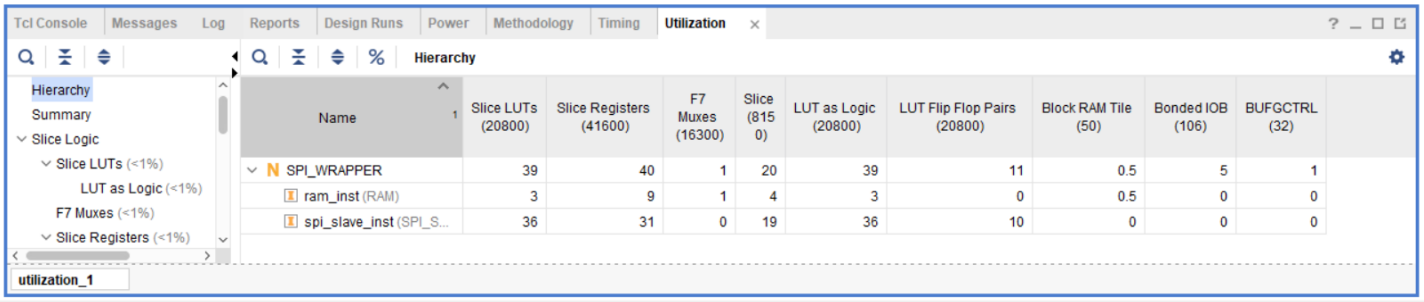


->We can tell from the previous timing report that sequential <- encoding is the best in terms of frequency

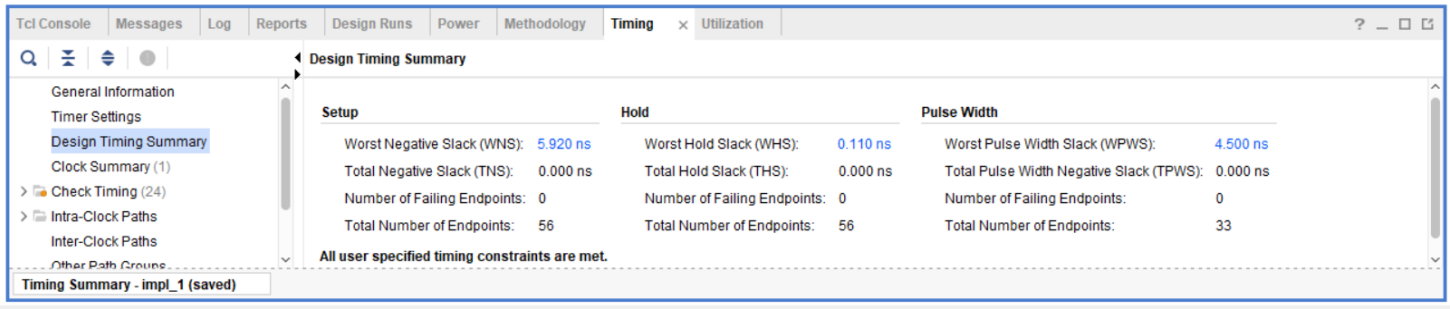
➢ Implementation Snippets :

○ Sequential Encoding :

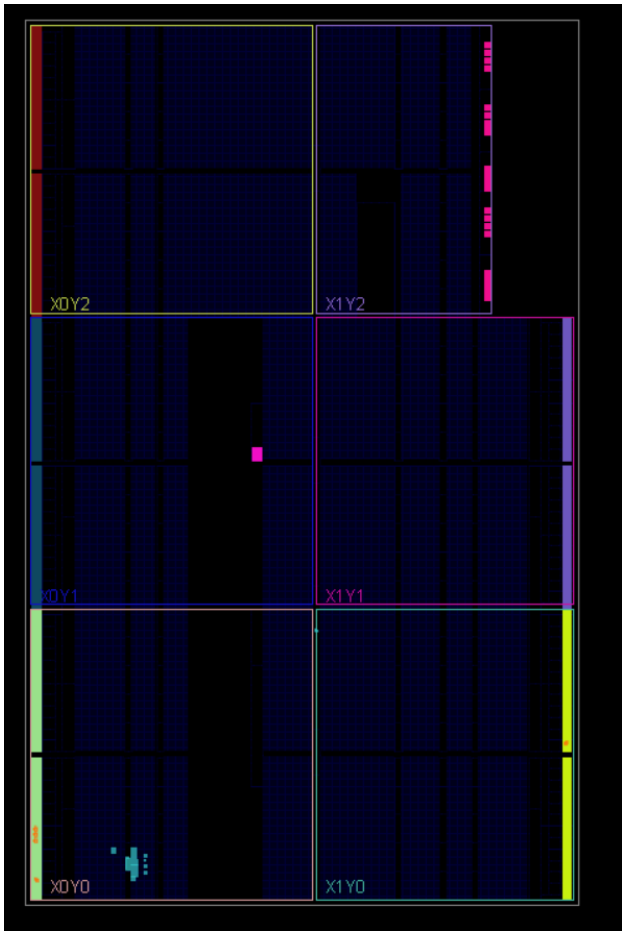
✓ Utilization report :



✓ Timing report :

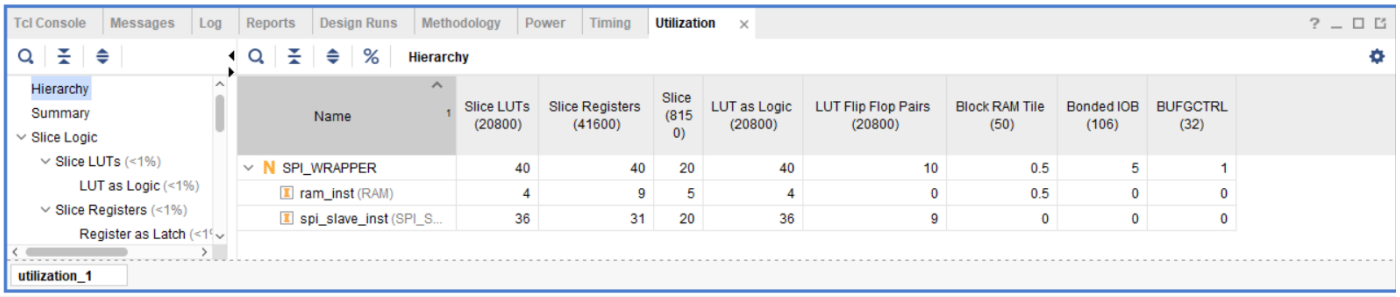


✓ FPGA device snippet :

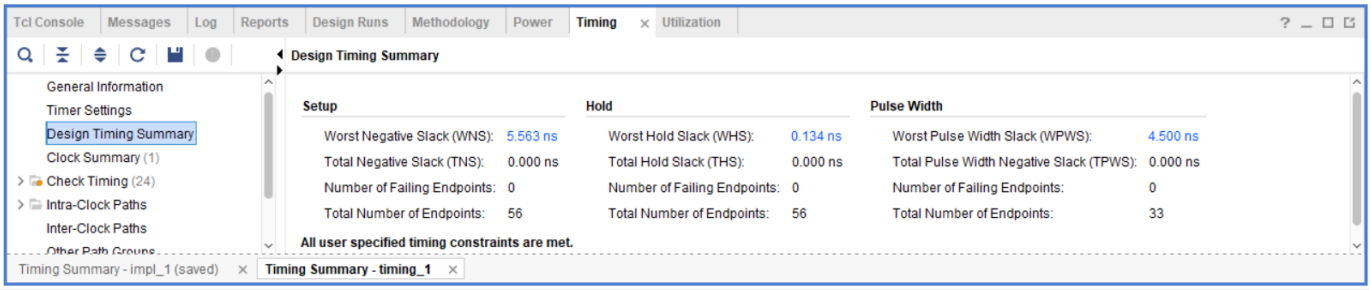


○ Gray Encoding :

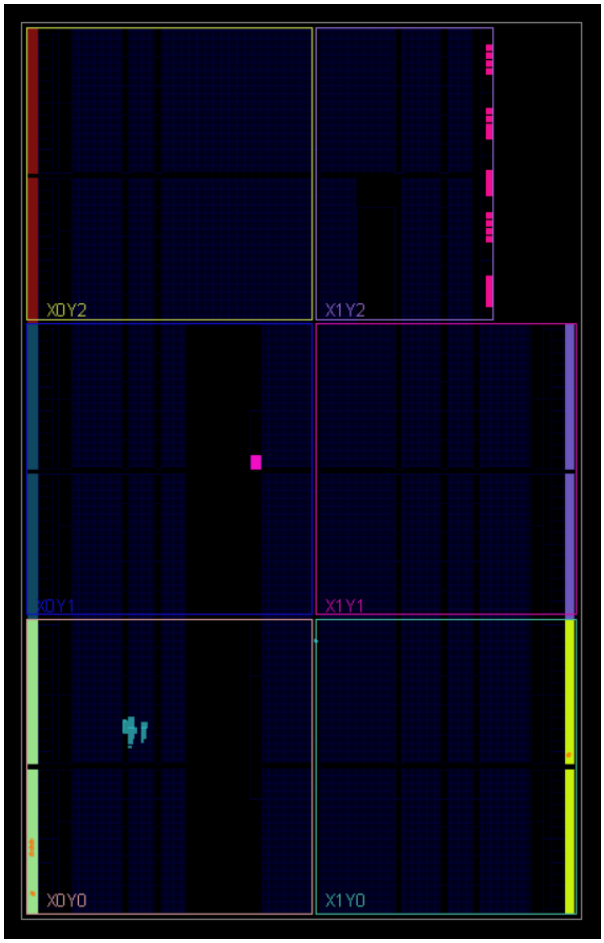
✓ Utilization report :



✓ Timing report :

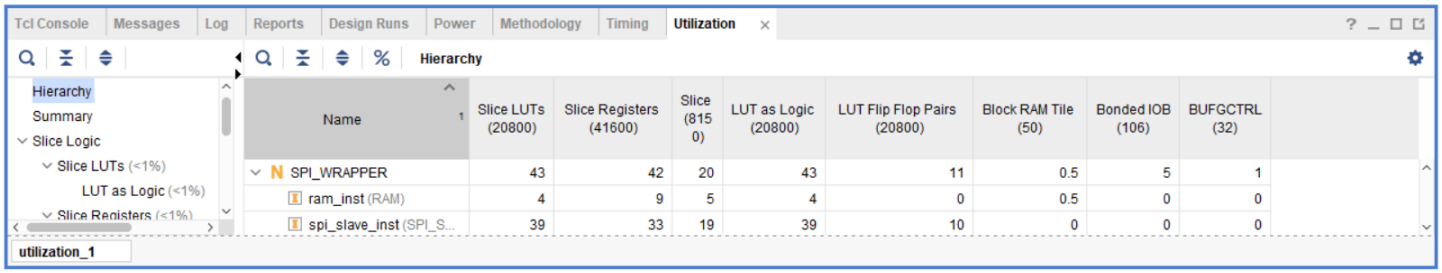


✓ FPGA device snippet :

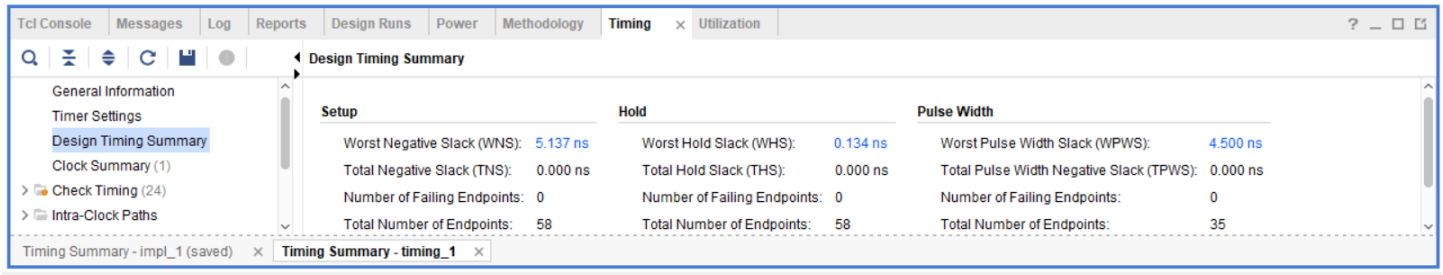


○ One_Hot Encoding :

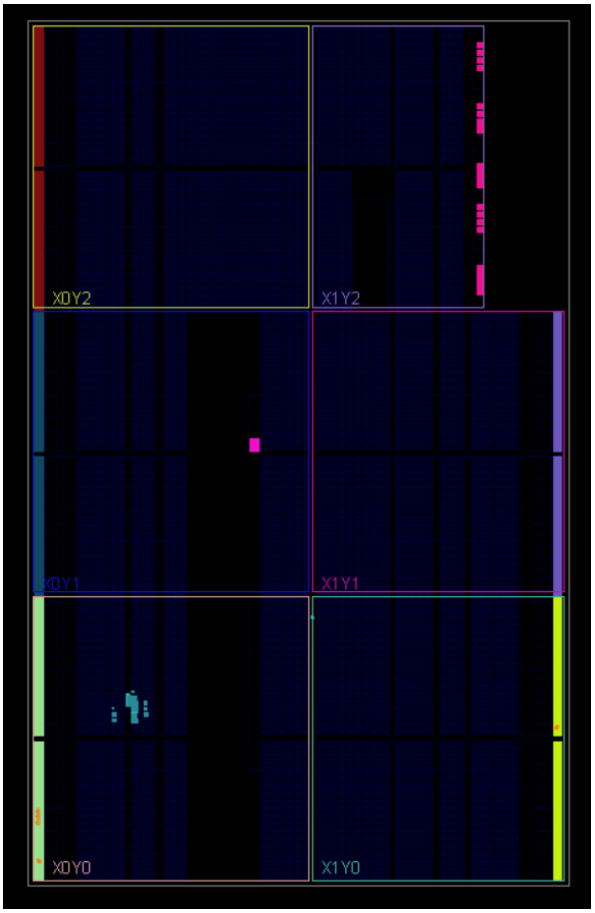
✓ Utilization report :



✓ Timing report :



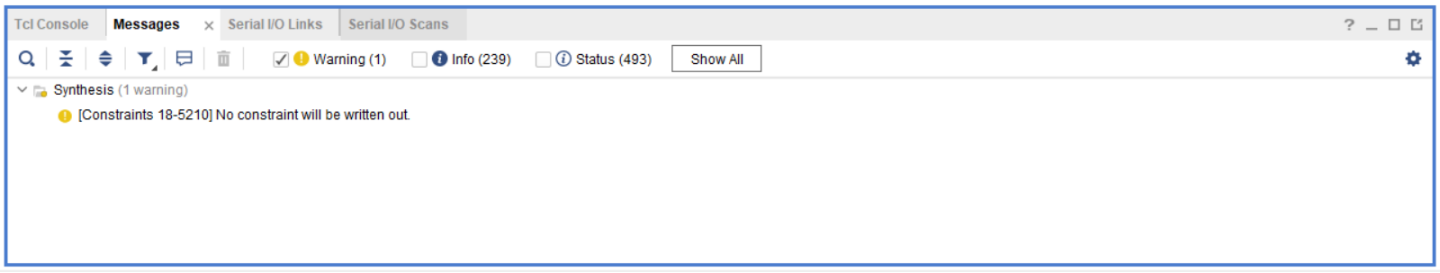
✓ FPGA device snippet :



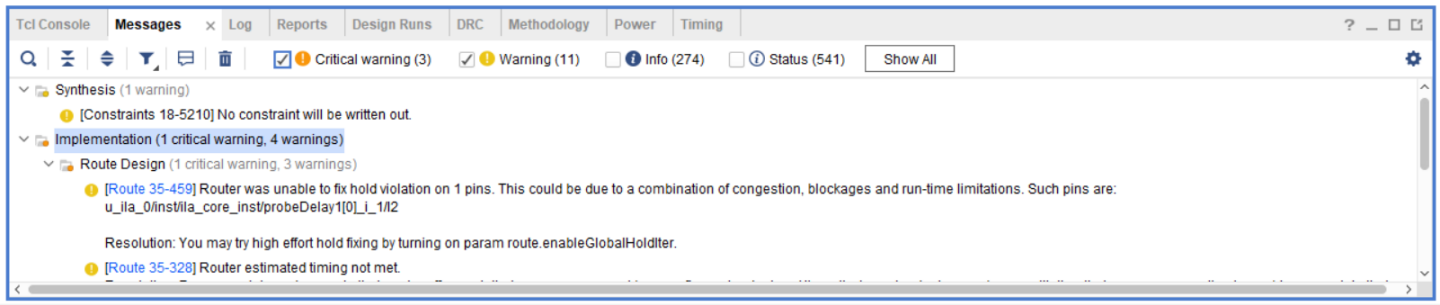
->Largest area was for the encoding with the least setup time slack<- which is the one_hot encoding

➢ Snippet of the “Messages” tab after bitstream generation :

Before setup debug :



After setup debug there will be hold time violation at the implementation and the bitstream generation :



➢ Constrain File after debug :

```
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]
```