SPI Slave with Single Port RAM

Project_2

==========

Name: Ahmed Nour

Email: ahmedmohamednoour@gmail.com

==========

Table of Contents

1. RTL code

1. TOP_Module

```rtl
module SPI_SLAVE_WITH_SINGLE_PORT_RAM (MISO, MOSI, clk, rst_n, SS_n);
    /*==============================================================================*/
    //Input Ports of SPI_SLAVE_WITH_SINGLE_PORT_RAM
    /*==============================================================================*/
    //Data ports
    input MOSI  ;

    //clock ports
    input clk   ;

    //control ports
    input SS_n  ;
    input rst_n ;

    /*==============================================================================*/
    //Output Ports of SPI_SLAVE_WITH_SINGLE_PORT_RAM
    /*==============================================================================*/
    //Data ports
    output MISO ;

    /*==============================================================================*/
    //Internal Ports of SPI_SLAVE_WITH_SINGLE_PORT_RAM
    /*==============================================================================*/
    //Internal Data ports
    wire [9:0] RAM_IN   ;
    wire [7:0] RAM_OUT  ;

    wire       rx_valid ;
    wire       tx_valid ;

    /*==============================================================================*/
    // Instantiation of Modules
    /*==============================================================================*/
    //SPI Slave
    SPI_Slave       SPI1 (.clk(clk), .rst_n(rst_n), .SS_n(SS_n), .tx_valid(tx_valid), .MOSI(MOSI),
                          .MISO(MISO), .tx_data(RAM_OUT), .rx_valid(rx_valid), .rx_data(RAM_IN));

    //Single Port RAM
    SINGLE_PORT_RAM RAM1 (.clk(clk), .rst_n(rst_n), .din(RAM_IN),
                          .rx_valid(rx_valid), .dout(RAM_OUT), .tx_valid(tx_valid));

endmodule
```

2. SPI_MODULE

```
module SPI_Slave (clk, rst_n, SS_n, tx_valid, MOSI, MISO, tx_data, rx_valid, rx_data);
/*==============================================*/
// states
/*==============================================*/
parameter IDLE_STATE     = 3'b000;
parameter CMD_CHECK      = 3'b001;
parameter WRITE_MODE     = 3'b010;
parameter READ_ADDRESS   = 3'b011;
parameter READ_DATA      = 3'b100;
/*==================================================================================*/
// Input Ports
/*==================================================================================*/
input           clk      ;
input           rst_n    ;
input           SS_n     ;
input           tx_valid ;
input           MOSI     ;
input     [7:0] tx_data  ;
/*==================================================================================*/
// Output Ports
/*==================================================================================*/
output           MISO     ;
output reg       rx_valid ;
output reg [9:0] rx_data  ;
/*==================================================================================*/
// Internals
/*==================================================================================*/
reg [2:0] CS          ;
reg [2:0] NS          ;
reg       read_flag   ;
reg [3:0] counter     ;
```

```
/*==================================================================================*/
// State Transition
/*==================================================================================*/
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        CS <= IDLE_STATE;
    else
        CS <= NS;
end
```

```
/*==============================================================================*/
// Next State Logic
/*==============================================================================*/
always @(*) begin
    case (CS)
        IDLE_STATE: begin
            if (!SS_n)
                NS = CMD_CHECK;
            else
                NS = IDLE_STATE;
        end

        CMD_CHECK: begin
            if (SS_n)
                NS = IDLE_STATE;
            else if (MOSI == 0)
                NS = WRITE_MODE;
            else if (MOSI == 1 && !read_flag)
                NS = READ_ADDRESS;
            else
                NS = READ_DATA;
        end

        WRITE_MODE: begin
            if (SS_n)
                NS = IDLE_STATE;
            else
                NS = WRITE_MODE;
        end

        READ_ADDRESS: begin
            if (SS_n)
                NS = IDLE_STATE;
            else
                NS = READ_ADDRESS;
        end

        READ_DATA: begin
            if (SS_n)
                NS = IDLE_STATE;
            else
                NS = READ_DATA;
        end

        default: NS = IDLE_STATE;

    endcase
end
```

```
/*==============================================================================*/
// Output logic
/*==============================================================================*/
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rx_data   <= 10'b0;
        rx_valid  <= 1'b0;
        counter   <= 4'b0;
        read_flag <= 1'b0;
    end
    else begin
        if (!SS_n) begin
            rx_valid <= 1'b0;
            if (CS == WRITE_MODE || CS == READ_ADDRESS) begin
                rx_data <= {rx_data[8:0], MOSI};
                if (counter == 9) begin
                    rx_valid  <= 1'b1;
                    counter   <= 4'b0;
                    if (CS == READ_ADDRESS)
                        read_flag <= 1'b1;
                end
                else begin
                    counter <= counter + 1;
                end
            end
            else if (CS == READ_DATA) begin
                if (!tx_valid) begin
                    rx_data <= {rx_data[8:0], MOSI};
                    if (counter == 9) begin
                        rx_valid <= 1'b1;
                        counter  <= 4'b0;
                    end
                    else begin
                        counter <= counter + 1;
                    end
                end
                else begin
                    if (counter > 7) begin
                        counter   <= 4'b0;
                        read_flag <= 1'b0;
                    end
                    else begin
                        counter <= counter + 1;
                    end
                end
            end
        end
        else begin
            rx_valid <= 1'b0;
        end
    end
end

assign MISO = (tx_valid && CS == READ_DATA) ? tx_data[8 - counter] : 0;

endmodule
```

3. RAM_MODULE

```
module SINGLE_PORT_RAM (din,rx_valid,clk,rst_n,dout,tx_valid);
/*==================================================*/
// Input Ports
/*==================================================*/
input  [9:0] din;
input  clk;
input  rst_n;
input  rx_valid;

/*==================================================*/
// Output Ports
/*==================================================*/
output reg [7:0] dout;
output reg tx_valid;

/*==================================================*/
// Parameters
/*==================================================*/
parameter MEM_DEPTH = 256;
parameter WIDTH = 8;

/*==================================================*/
// Internal Registers
/*==================================================*/
reg [WIDTH-1:0] WRITE;
reg [WIDTH-1:0] READ;
reg [7:0] mem [0:MEM_DEPTH-1];
```

```
/*==============================================*/
// Sequential Logic
/*==============================================*/
always @(posedge clk) begin
    if (!rst_n) begin
        dout <= 8'd0;
        tx_valid <= 1'b0;
        WRITE <= 0;
        READ <= 0;
    end

    else begin
        if (rx_valid) begin
            case (din[9:8])
                2'b00: begin
                    WRITE <= din[7:0];
                    tx_valid <= 0;
                end

                2'b01: begin
                    mem[WRITE] <= din[7:0];
                    tx_valid <= 0;
                end

                2'b10: begin
                    READ <= din[7:0];
                    tx_valid <= 0;
                end

                2'b11: begin
                    dout <= mem[READ];
                    tx_valid <= 1;
                end
            endcase
        end
    end
end

endmodule
```

2. Testbench code

```
module SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb();
/*==============================================================================*/
// Internal Signals
/*==============================================================================*/
reg mosi_tb;
reg clk_tb;
reg ss_n_tb;
reg rst_n_tb;

wire miso_tb;

integer i;

/*==============================================================================*/
// DUT Instantiation
/*==============================================================================*/
SPI_SLAVE_WITH_SINGLE_PORT_RAM DUT (.MISO(miso_tb), .MOSI(mosi_tb), .clk(clk_tb),
                                    .rst_n(rst_n_tb), .SS_n(ss_n_tb));

/*==============================================================================*/
// Clock Generation
/*==============================================================================*/
initial begin
    clk_tb = 0;
    forever #1 clk_tb = ~clk_tb;
end
/*==============================================================================*/
// Stimulus
/*==============================================================================*/
initial begin
/*==============================================================================*/
// Reset
/*==============================================================================*/
    $readmemh("mem.dat", DUT.RAM1.mem);
    for (i = 0; i < 256; i = i + 1)
        DUT.RAM1.mem[i] <= 8'd0;

    rst_n_tb = 0;
    mosi_tb  = 1;
    ss_n_tb  = 1;
    @(negedge clk_tb);@(negedge clk_tb);
```

```
/*==============================================================================*/
// Write Address
/*==============================================================================*/
    rst_n_tb = 1; @(negedge clk_tb);
    ss_n_tb  = 0;
    @(negedge clk_tb); @(negedge clk_tb);

    //00111_11111
    mosi_tb = 0; @(negedge clk_tb); mosi_tb = 0; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);


    ss_n_tb = 1;
    @(negedge clk_tb);
    @(negedge clk_tb);

/*==============================================================================*/
// Write Data
/*==============================================================================*/
    ss_n_tb = 0;
    @(negedge clk_tb); @(negedge clk_tb);
    //01101_01010
    mosi_tb = 0; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 0; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 0; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 0; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 0; @(negedge clk_tb);


    ss_n_tb = 1;
    @(negedge clk_tb);
    @(negedge clk_tb);

/*==============================================================================*/
// Read Address
/*==============================================================================*/
    ss_n_tb = 0;
    @(negedge clk_tb); @(negedge clk_tb);
    //1010101101
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 0; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);
    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);



    ss_n_tb = 1;
    @(negedge clk_tb);
    @(negedge clk_tb);

/*==============================================================================*/
// Read Data
/*==============================================================================*/
    ss_n_tb = 0;
    @(negedge clk_tb); @(negedge clk_tb);

    mosi_tb = 1; @(negedge clk_tb); mosi_tb = 1; @(negedge clk_tb);

    repeat (16)begin
        @(negedge clk_tb);
    end

    ss_n_tb = 1;
    @(negedge clk_tb);
    @(negedge clk_tb);

    $display("Test Done");
    $finish;
end

endmodule
```

3. Do file

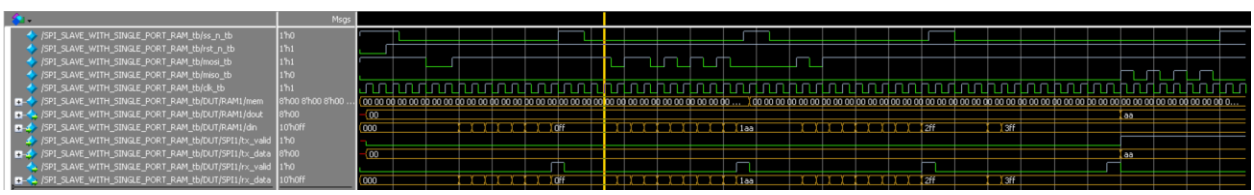
```
vlib work
vlog SPI_Slave.v SINGLE_PORT_RAM.v SPI_SLAVE_WITH_SINGLE_PORT_RAM.v SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb.v

vsim -voptargs=+acc work.SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb

add wave -position insertpoint \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/ss_n_tb \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/rst_n_tb \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/mosi_tb \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/miso_tb \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/clk_tb

add wave -position insertpoint \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/RAM1/mem \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/RAM1/dout \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/RAM1/din

add wave -position insertpoint \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/SPI1/tx_valid \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/SPI1/tx_data \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/SPI1/rx_valid \
    sim:/SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb/DUT/SPI1/rx_data

run -all
#quit -sim
```

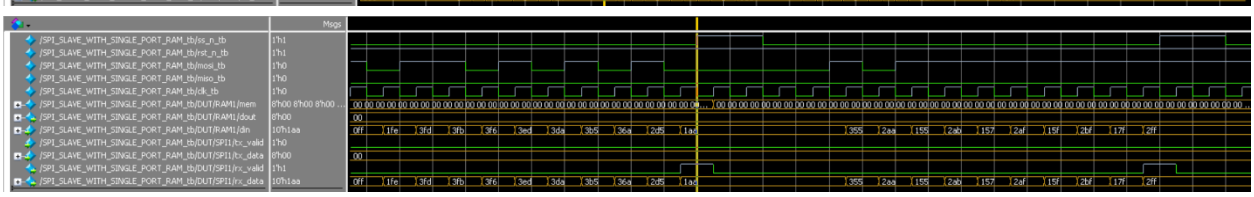
4. QuestaSim Snippets

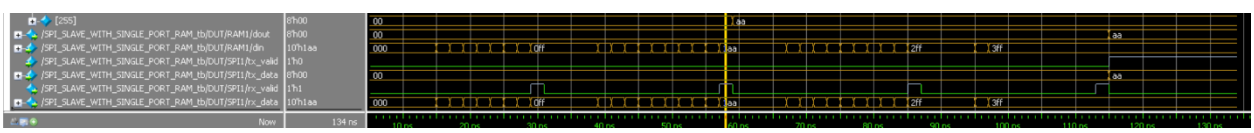
1. Messages Snippet

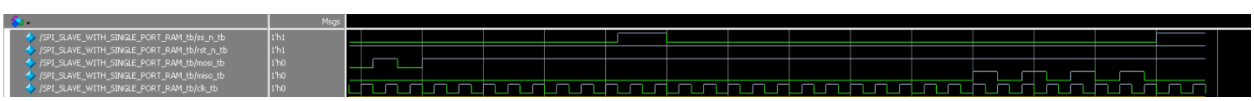
```
# Top level modules:
#       SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb
# End time: 17:19:51 on Aug 20,2025, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" work.SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb
# Start time: 17:19:51 on Aug 20,2025
# ** Note: (vsim-3812) Design is being optimized...
# ** Note: (vopt-143) Recognized 1 FSM in module "SPI_Slave(fast)".
# Loading work.SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb(fast)
# Loading work.SPI_SLAVE_WITH_SINGLE_PORT_RAM(fast)
# Loading work.SPI_Slave(fast)
# Loading work.SINGLE_PORT_RAM(fast)
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#           File in use by: nour  Hostname: NOUR  ProcessID: 29948
#           Attempting to use alternate WLF file "./wlft0rl8k7".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlft0rl8k7
# ** Warning: (vsim-7) Failed to open readmem file "mem.dat" in read mode.
# No such file or directory. (errno = ENOENT)    : SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb.v(34)
#    Time: 0 ns  Iteration: 0  Instance: /SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb
# Test Done
# ** Note: $finish    : SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb.v(114)
#    Time: 134 ns  Iteration: 1  Instance: /SPI_SLAVE_WITH_SINGLE_PORT_RAM_tb
```

2. Waveform Snippet









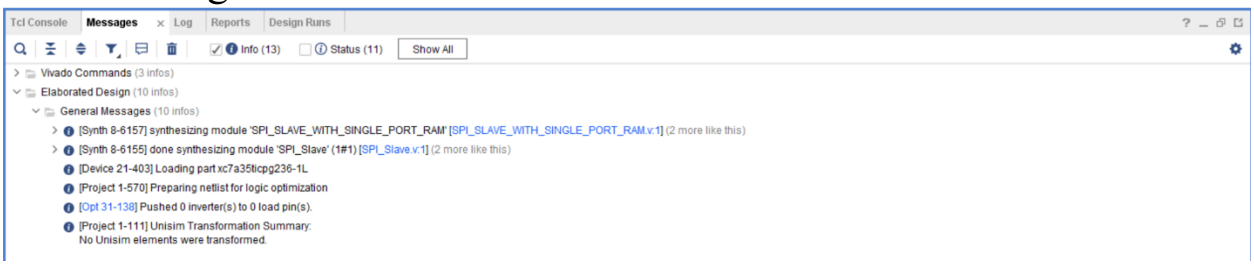
5. Constraint File

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


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

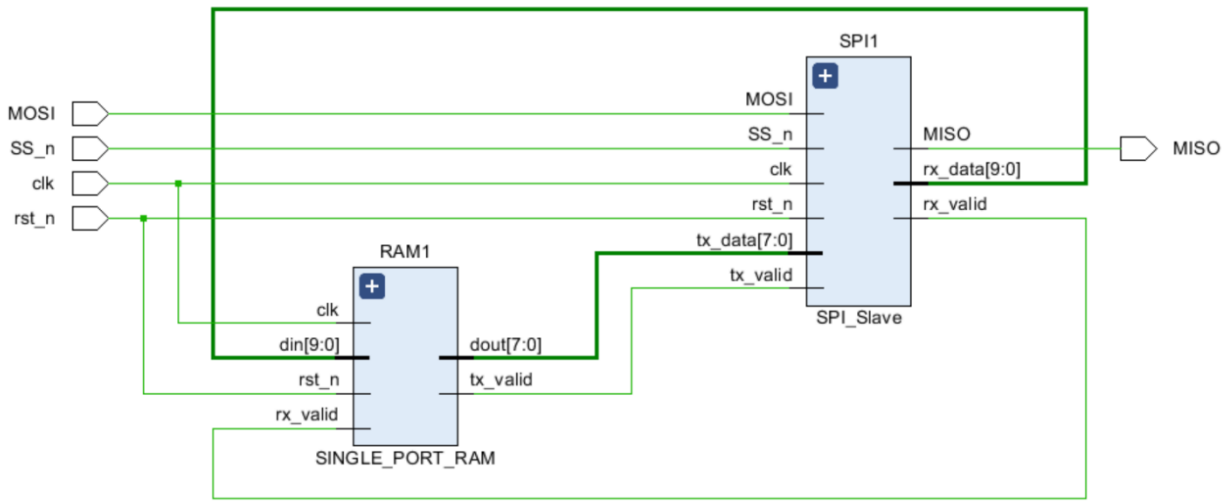
```
## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]



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

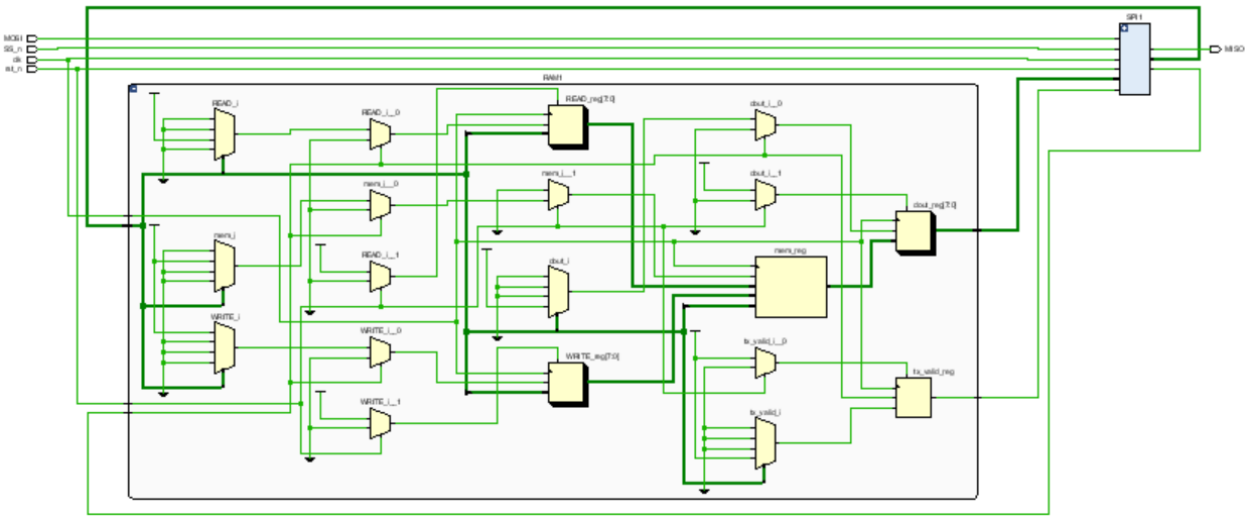
6. Elaboration

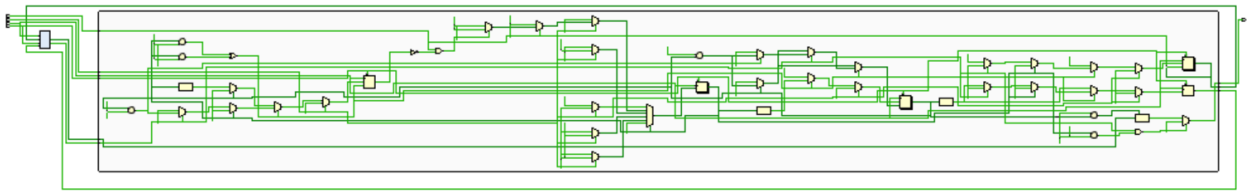
1. Message tab



2. Schematic

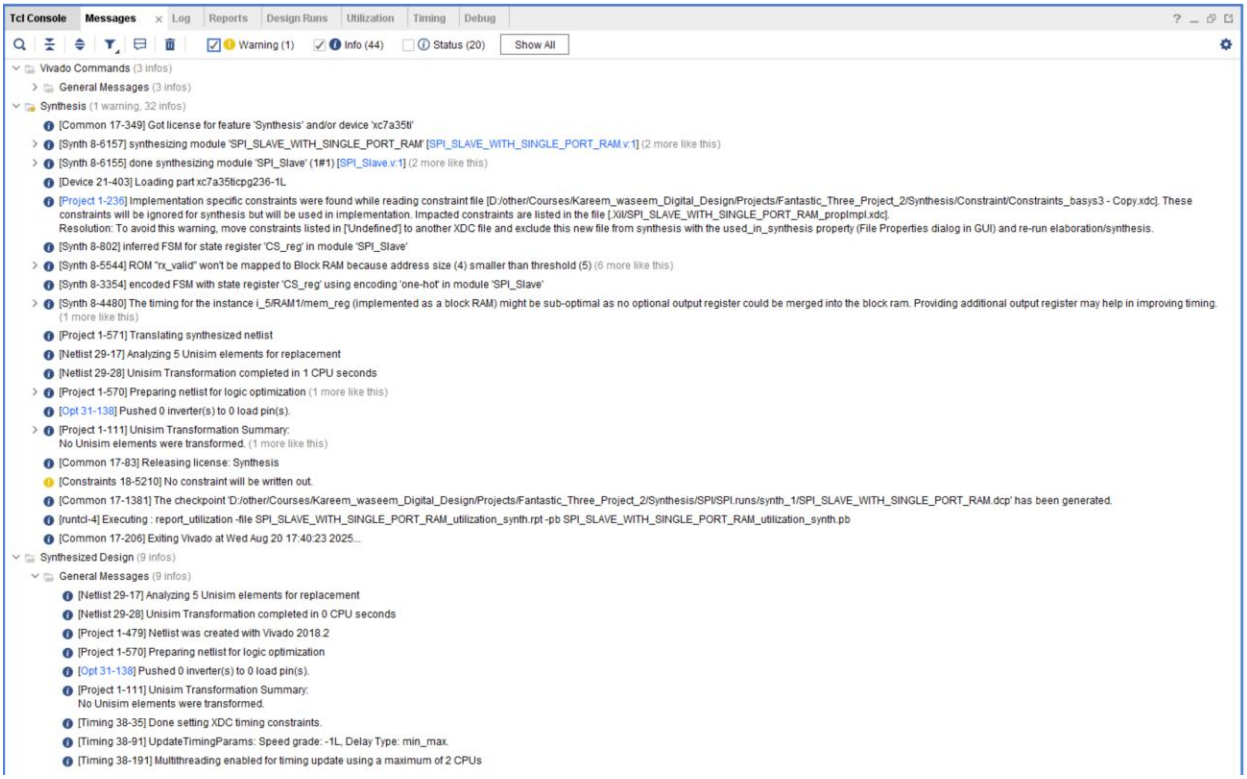




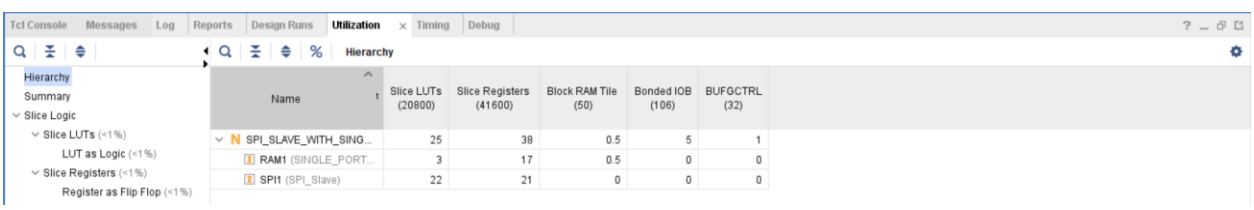


7. Synthesis

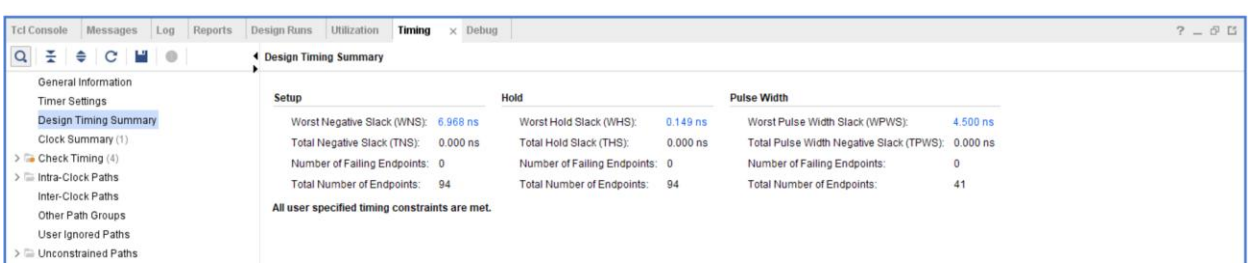
1. Message Tab



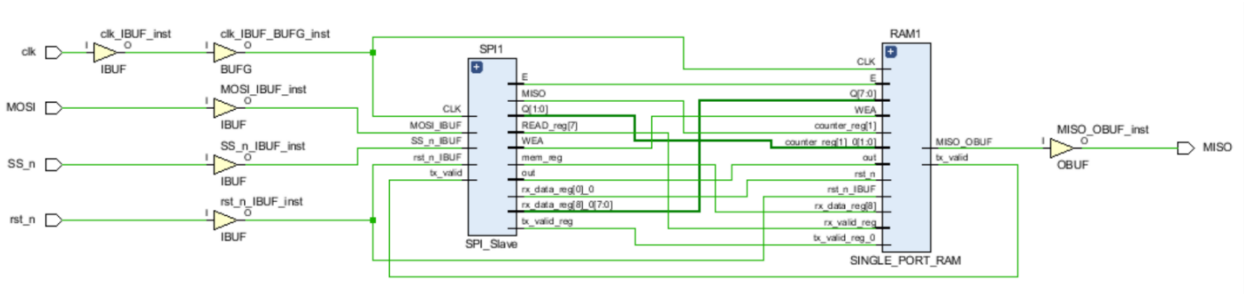
2. Utilization report

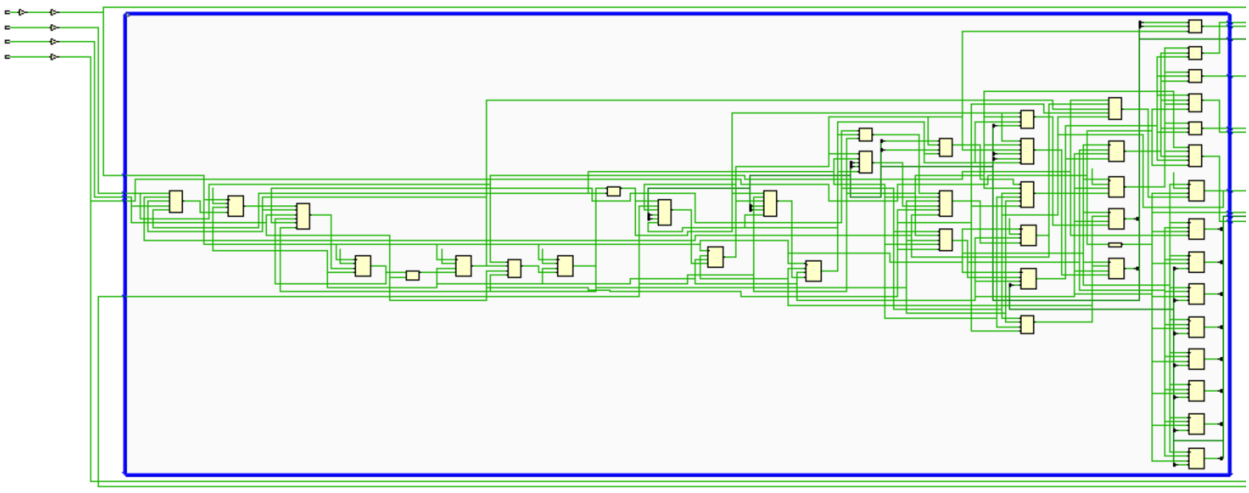


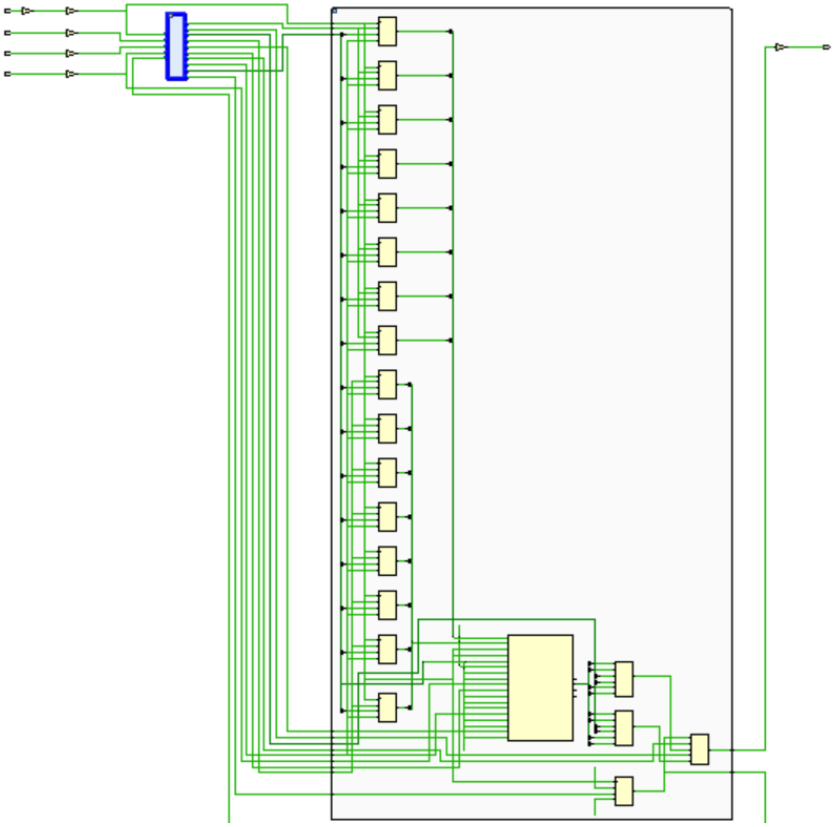
3. Timing report



4. Schematic


clk
clk_IBUF_inst
IBUF
clk_IBUF_BUFG_inst
BUFG
MOSI
MOSI_IBUF_inst
IBUF
SS_n
SS_n_IBUF_inst
IBUF
rst_n
rst_n_IBUF_inst
IBUF
SPI1
CLK
MOSI_IBUF
SS_n_IBUF
rst_n_IBUF
tx_valid
E
MISO
Q[1:0]
READ_reg[7]
WEA
mem_reg
out
rx_data_reg[0]_0
rx_data_reg[8]_0[7:0]
tx_valid_reg
SPI_Slave
RAM1
CLK
E
Q[7:0]
WEA
counter_reg[1]
counter_reg[1]_0[1:0]
out
rst_n
rst_n_IBUF
rx_data_reg[8]
rx_valid_reg
tx_valid_reg_0
MISO_OBUF
tx_valid
SINGLE_PORT_RAM
MISO_OBUF_inst
OBUF
MISO





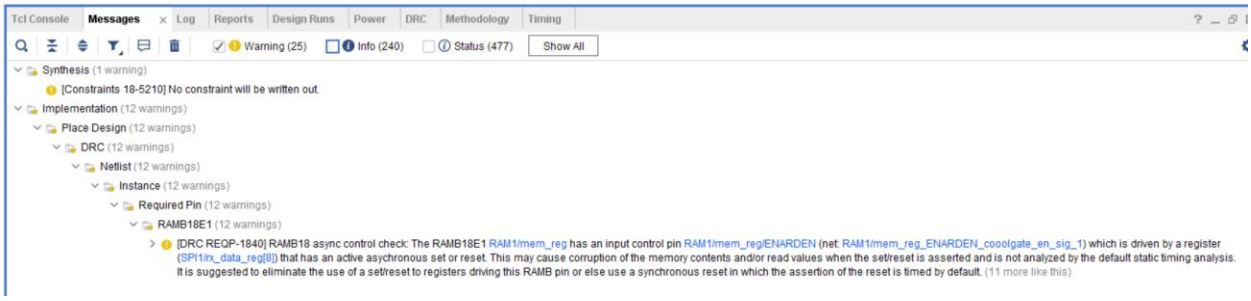
5. Netlist

```
`timescale 1 ps / 1 ps

module SINGLE_PORT_RAM
  (tx_valid,
   MISO_OBUF,
   CLK,
   rst_n_IBUF,
   \rx_data_reg[8] ,
   rst_n,
   Q,
   WEA,
   tx_valid_reg_0,
   out,
   \counter_reg[1] ,
   \counter_reg[1]_0 ,
   E,
   rx_valid_reg);
  output tx_valid;
  output MISO_OBUF;
  input CLK;
  input rst_n_IBUF;
  input \rx_data_reg[8] ;
  input rst_n;
  input [7:0]Q;
  input [0:0]WEA;
  input tx_valid_reg_0;
  input [0:0]out;
```

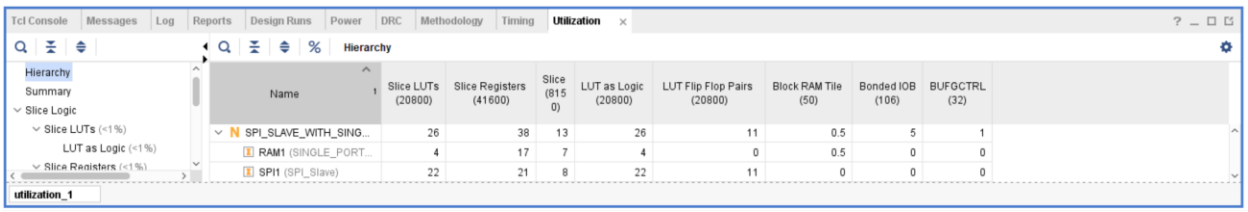
```
    .INIT(64'h00000000AAAAAA20))
    rx_valid_i_1
      (.I0(rx_valid2_in),
       .I1(tx_valid),
       .I2(out),
       .I3(\FSM_onehot_CS_reg_n_0_[3] ),
       .I4(\FSM_onehot_CS_reg_n_0_[2] ),
       .I5(SS_n_IBUF),
       .O(rx_valid_i_1_n_0));
  (* SOFT_HLUTNM = "soft_lutpair1" *)
  LUT4 #(
    .INIT(16'h1000))
    rx_valid_i_2
      (.I0(Q[1]),
       .I1(\counter_reg_n_0_[2] ),
       .I2(Q[0]),
       .I3(counter1),
       .O(rx_valid2_in));
  FDCE #(
    .INIT(1'b0))
    rx_valid_reg
      (.C(CLK),
       .CE(\<const1> ),
       .CLR(\rx_data_reg[0]_0 ),
       .D(rx_valid_i_1_n_0),
       .Q(rx_valid));
  (* SOFT_HLUTNM = "soft_lutpair0" *)
  LUT5 #(
    .INIT(32'hE2220000))
    tx_valid_i_1
      (.I0(tx_valid),
       .I1(rx_valid),
       .I2(RAM_IN[9]),
       .I3(RAM_IN[8]),
       .I4(rst_n_IBUF),
       .O(tx_valid_reg));
endmodule
```

8. Implementation

1. Message Tab

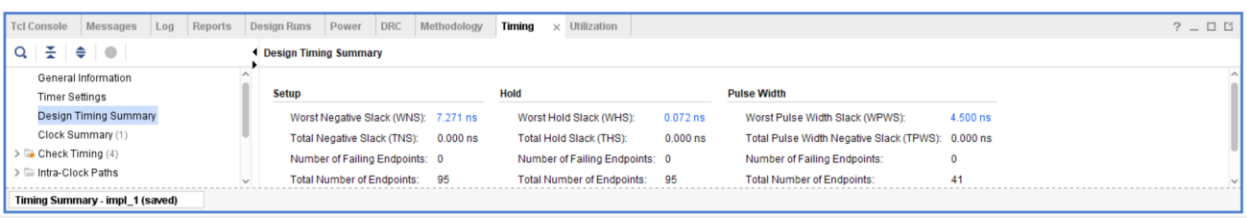


2. Utilization report



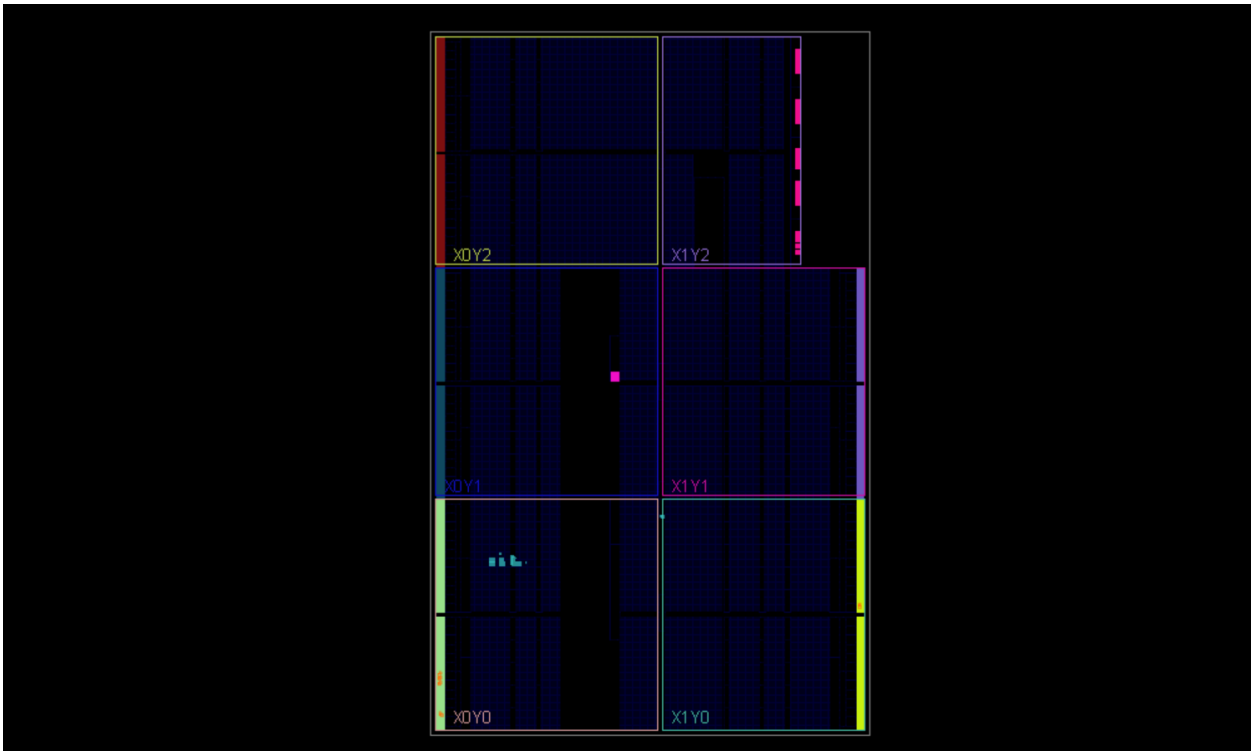
Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_SLAVE_WITH_SING...	26	38	13	26	11	0.5	5	1
RAM1 (SINGLE_PORT...	4	17	7	4	0	0.5	0	0
SPI1 (SPI_Slave)	22	21	8	22	11	0	0	0

3. Timing report

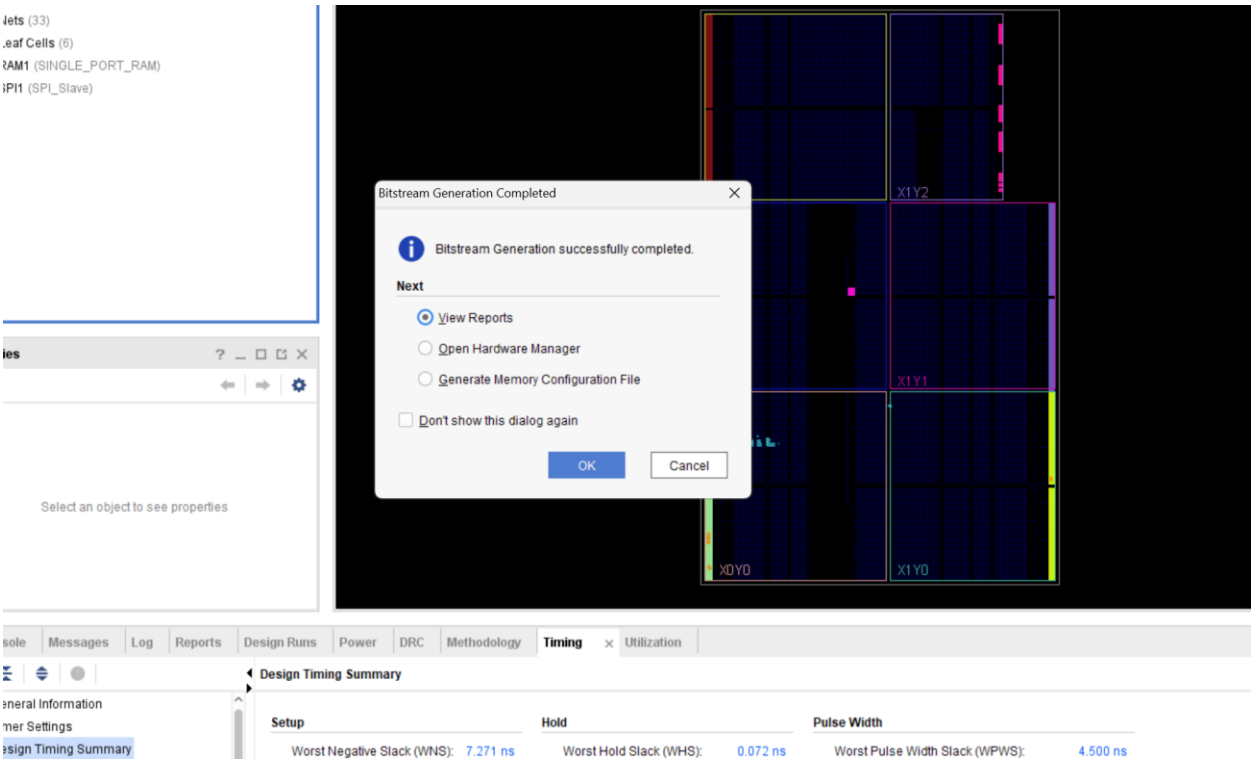


Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.271 ns	Worst Hold Slack (WHS):	0.072 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	95	Total Number of Endpoints:	95	Total Number of Endpoints:	41

4. Device

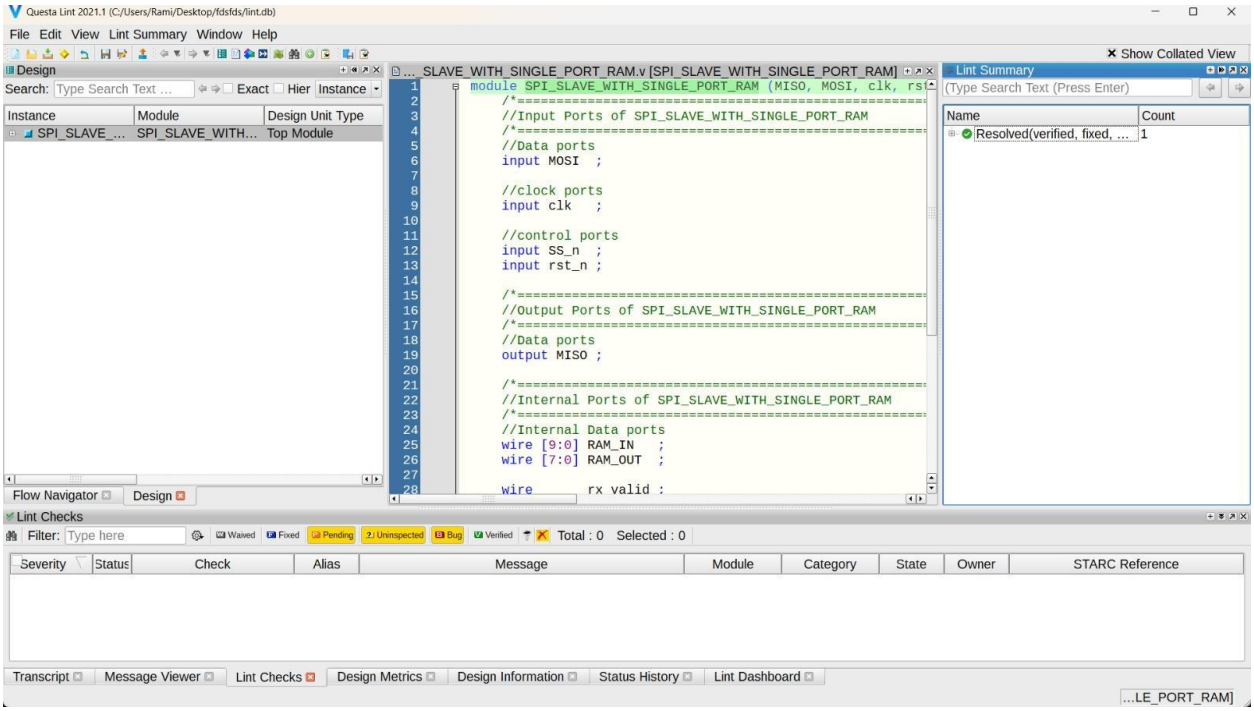


5. Bitstream

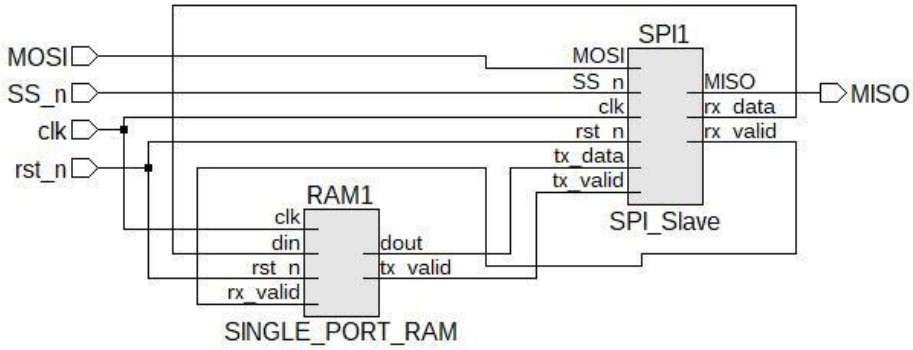


9. Linting

1. Lint snippet

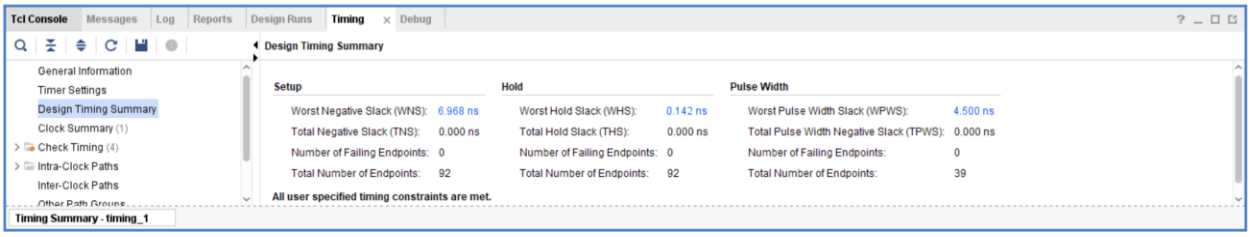


2. Lint Schematic

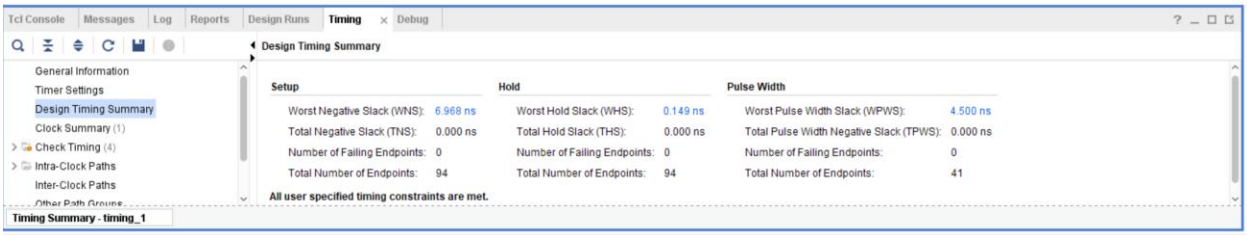


10. Comment

Gray encoding:



One_hot encoding:



So, in terms of the highest frequency possible it would be gray encoding.