SPI Slave with Single Port RAM

By

Youssef Mohamed Shaban

Contents

Lower Module (RAM)

RAM.v

```
1  //--------------------------- Single-Port Sync. RAM ---------------------------//
2
3  module RAM (din, clk, rst_n, rx_valid, dout, tx_valid);
4
5  parameter MEM_DEPTH = 256;
6  parameter ADDR_SIZE = 8;
7
8  input clk, rst_n, rx_valid;
9  input [ADDR_SIZE+1 : 0] din;
10
11 output reg tx_valid;
12 output reg [ADDR_SIZE-1 : 0] dout;
13
14 reg [7:0] RAM [MEM_DEPTH-1 : 0];
15
16 reg [ADDR_SIZE-1 : 0] rd_addr, wr_addr;
17
18 always @(posedge clk) begin
19     if (~rst_n) begin
20         dout     <= 0;
21         tx_valid <= 0;
22         rd_addr  <= 0;
23         wr_addr  <= 0;
24     end
25
26     else if (rx_valid) begin
27         case (din[ADDR_SIZE+1 : ADDR_SIZE])
28
29             2'b 00: begin
30                         wr_addr <= din[ADDR_SIZE-1 : 0];
31                         tx_valid <= 0;
32                     end
33
34             2'b 01: begin
35                         RAM[wr_addr] <= din[7 : 0];
36                         tx_valid <= 0;
37                     end
38
39             2'b 10: begin
40                         rd_addr <= din[ADDR_SIZE-1 : 0];
41                         tx_valid <= 0;
42                     end
43
```

```
43
44                  2'b 11: begin
45                              dout <= RAM[rd_addr];
46                              tx_valid <= 1;
47                          end
48              endcase
49          end
50      end
51      endmodule
```

Lower Module (SPI Slave Interface)

SPI_Slave.v

```
1   //------------------------------ SPI Slave Interface -------------------------------//
2
3   module SPI_Slave (MOSI, MISO, SS_n, clk, rst_n, rx_data, rx_valid, tx_data, tx_valid);
4
5   parameter IDLE      = 3'b 000;
6   parameter CHK_CMD   = 3'b 001;
7   parameter WRITE     = 3'b 010;
8   parameter READ_ADD  = 3'b 011;
9   parameter READ_DATA = 3'b 100;
10
11  input clk, rst_n, MOSI, SS_n, tx_valid;
12  input [7:0] tx_data;
13
14  output reg rx_valid, MISO;
15  output reg [9:0] rx_data;
16
17  reg [2:0] cs, ns;
18  reg [3:0] counter;
19  reg addr_or_data;
20
21  //State Memory
22  always @(posedge clk) begin
23      if (~rst_n)
24          cs <= IDLE;
25      else
26          cs <= ns;
27  end
28
29  //Next State Logic
30  always @(*) begin
31      case (cs)
32          IDLE: begin
33                  if (~SS_n) ns = CHK_CMD;
34                  else ns = IDLE;
35              end
36
```

```
        CHK_CMD: begin
                    if (~SS_n && ~MOSI) ns = WRITE;
                    else if (~SS_n &&  MOSI && ~addr_or_data) ns = READ_ADD;
                    else if (~SS_n &&  MOSI &&  addr_or_data) ns = READ_DATA;
                    else ns = IDLE;
        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
always @(posedge clk) begin
    if (~rst_n) begin
        MISO     <= 0;
        rx_data  <= 0;
        rx_valid <= 0;
        counter  <= 0;
        addr_or_data <=0;
    end

    else begin
        case (cs)
        IDLE, CHK_CMD: begin
                    rx_valid <= 0;
                    counter  <= 0;
                    MISO     <= 0;
                end

        WRITE: begin
                    if (counter <= 9) begin
                        rx_data <= {rx_data[8:0] , MOSI};
                        counter <= counter + 1;
                    end

                    if (counter >= 9) rx_valid <= 1;
                end
```

```
89
90          READ_ADD: begin
91                      if (counter <= 9) begin
92                          rx_data <= {rx_data[8:0] , MOSI};
93                          counter <= counter + 1;
94                      end
95
96                      if (counter >= 9) begin
97                          rx_valid <= 1;
98                          addr_or_data <= 1;
99                      end
100                 end
101
102         READ_DATA: begin
103                     if (~tx_valid && counter <= 9) begin
104                         rx_data <= {rx_data[8:0] , MOSI};
105                         counter <= counter + 1;
106                     end
107
108                     if (~tx_valid && counter >= 9) begin
109                         rx_valid <= 1;
110                         addr_or_data <= 0;
111                     end
112
113                     if (tx_valid && counter >= 3) begin //[10 - 3] --> [7] and so on ...
114                         MISO <= tx_data[counter-3];
115                         counter <= counter - 1;
116                     end
117                 end
118         endcase
119     end
120 end
121
122 endmodule
```

Top Module (SPI Wrapper)

```
//---------------------------------------- SPI Wrapper ----------------------------------------//

module SPI_Wrapper (MOSI, MISO, SS_n, clk, rst_n);

//Parameters For RAM
parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;

input clk, rst_n, SS_n, MOSI;
output MISO;

//Internal Wires
wire [ADDR_SIZE+1 : 0] rx_data_bus;
wire [ADDR_SIZE-1 : 0] tx_data_bus;
wire rx_valid, tx_valid;

RAM #(MEM_DEPTH, ADDR_SIZE) RAM_INSTANCE (rx_data_bus, clk, rst_n, rx_valid, tx_data_bus, tx_valid);

SPI_Slave SPI_INSATNCE (MOSI, MISO, SS_n, clk, rst_n, rx_data_bus, rx_valid, tx_data_bus, tx_valid);

endmodule
```

DO File

```
.main clear

vlib work

vlog RAM.v SPI_Slave.v SPI_Wrapper.v TB_2.v

vsim -voptargs=+acc work.Sh3ban_TB

add wave -radix unsigned -position insertpoint  \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/counter
add wave -position insertpoint  \
sim:/Sh3ban_TB/Parallel_ADDR \
sim:/Sh3ban_TB/Parallel_DATA
add wave -position insertpoint  \
sim:/Sh3ban_TB/rst_n
add wave -color magenta -position insertpoint  \
sim:/Sh3ban_TB/clk
add wave -position insertpoint  \
sim:/Sh3ban_TB/MISO \
sim:/Sh3ban_TB/SS_n
add wave -color gold -position insertpoint  \
sim:/Sh3ban_TB/MOSI
add wave -position insertpoint  \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/rx_data
add wave -position insertpoint  \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/rx_valid
#add wave -radix binary -position insertpoint  \
#sim:/Sh3ban_TB/DUT/RAM_INSTANCE/din
add wave -color cyan -position insertpoint  \
sim:/Sh3ban_TB/DUT/RAM_INSTANCE/wr_addr
add wave -position insertpoint  \
sim:/Sh3ban_TB/DUT/RAM_INSTANCE/rd_addr
add wave -position insertpoint  \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/tx_valid \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/tx_data
add wave -color cyan -position insertpoint  \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/cs
add wave -position insertpoint  \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/ns \
sim:/Sh3ban_TB/DUT/SPI_INSATNCE/addr_or_data
add wave -color Orchid -position insertpoint  \
{sim:/Sh3ban_TB/DUT/RAM_INSTANCE/RAM[169]}
add wave -position insertpoint  \
sim:/Sh3ban_TB/DUT/RAM_INSTANCE/RAM
add wave -position insertpoint  \
sim:/Sh3ban_TB/Expected_out

run -all
```

Testbench Code

- This testbench is:
  - A directed testbench.
  - Self-checking testbench, where it checks for:
    I. Current State after each expected state transition.
    II. Output **"MISO"** after being converted into serial, to check if it is the same as the intended RAM content.
    III. RAM content at the intended address after the expected writing operation.
    IV. The value in the internal register of RAM, after conversion from serial to parallel and after being read by RAM (in the case of writing address and the case of reading address).

- Since, the testbench checks for all cases in RAM & SPI slave interface
  It can be considered a testbench for RAM too.

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
//                    Self Checking Testbech for SPI & RAM                         //
//                                      BY                                         //
//                                Youssef Sha3ban                                  //
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

module Sh3ban_TB ();

parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;

//The following parameters are just for the self-checking testbench
parameter IDLE      = 3'b 000;
parameter CHK_CMD   = 3'b 001;
parameter WRITE     = 3'b 010;
parameter READ_ADD = 3'b 011;
parameter READ_DATA = 3'b 100;
//------------------------------------------------------------------

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

SPI_Wrapper #(MEM_DEPTH, ADDR_SIZE) DUT (MOSI, MISO, SS_n, clk, rst_n);

//The following regesiters are just for the self-checking testbench
reg [9:0] Parallel_ADDR; //Master Address Input (Parallel)
reg [9:0] Parallel_DATA; //Master Data Input (Parallel)
reg [7:0] Expected_out;
```

```
initial begin
    clk = 0;
    forever
        #10 clk = ~clk;
end

integer i;

initial begin
$readmemh("mem.dat", DUT.RAM_INSTANCE.RAM);

$display("\n----------------> \"Testing Reset\" <----------------\n");
rst_n = 0;
SS_n = 1;
MOSI = 0;
@(negedge clk);
rst_n = 1;
@(negedge clk);

//-------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != IDLE) begin        //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-------------------------------------------------//



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
 $display("\n----------------> \"Testing Write Address\" <----------------\n");
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

SS_n = 0;
@(negedge clk);

//-------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != CHK_CMD) begin     //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-------------------------------------------------//

MOSI = 0;
Parallel_ADDR = 10'b 00_1010_1001;
@(negedge clk);

//-------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != WRITE) begin       //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-------------------------------------------------//
```

```
for (i=9 ; i >= 0 ; i=i-1) begin
    MOSI = Parallel_ADDR[i];
    @(negedge clk);
end

@(negedge clk); //To let RAM Writes Address

if (DUT.RAM_INSTANCE.wr_addr != Parallel_ADDR[7:0]) begin
    $display("ERROR!! Incorrect Output :(");
    $stop;
end

$write("\n");
SS_n = 1;
@(negedge clk);

//------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != IDLE) begin        //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//------------------------------------------------//



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
    $display("\n----------------> \"Testing Write Data\" <----------------\n");
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

SS_n = 0;
@(negedge clk);

//------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != CHK_CMD) begin     //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//------------------------------------------------//

MOSI = 0;
Parallel_DATA = 10'b 01_0100_1101;
@(negedge clk);

//------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != WRITE) begin       //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//------------------------------------------------//
```

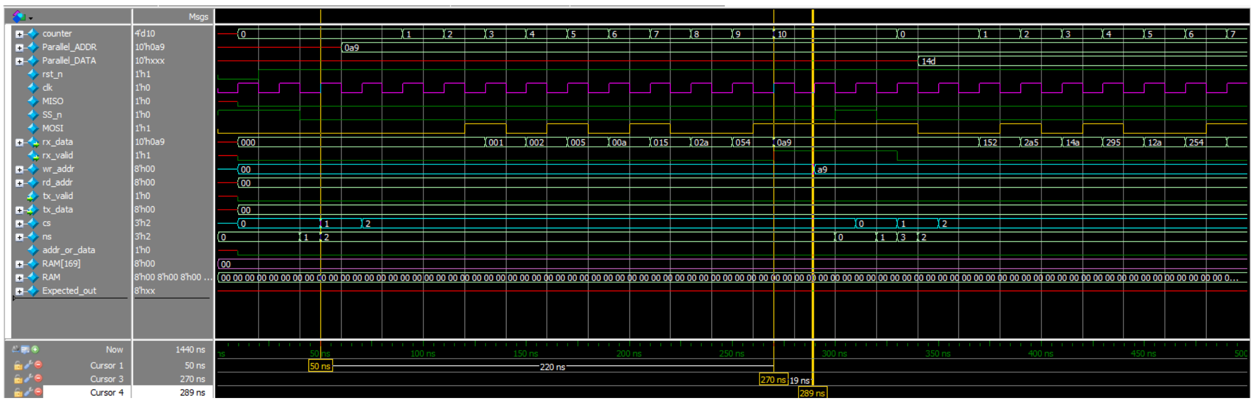
```
for (i=9 ; i >= 0 ; i=i-1) begin
    MOSI = Parallel_DATA[i];
    @(negedge clk);
end

@(negedge clk); //To let RAM Writes Data

if (DUT.RAM_INSTANCE.RAM[Parallel_ADDR[7:0]] != Parallel_DATA[7:0]) begin
    $display("ERROR!! Incorrect Output :(");
    $stop;
end

$write("\n");
SS_n = 1;
@(negedge clk);

//-----------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != IDLE) begin        //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-----------------------------------------------//



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
                    $display("\n \"Testing Read Address\" \n");
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

SS_n = 0;
@(negedge clk);

//-----------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != CHK_CMD) begin     //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-----------------------------------------------//

MOSI = 1;
Parallel_ADDR = 10'b 10_1010_1001;

@(negedge clk);

//-----------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != READ_ADD) begin    //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-----------------------------------------------//
```

```
for (i=9 ; i >= 0 ; i=i-1) begin
    MOSI = Parallel_ADDR[i];
    @(negedge clk);
end

@(negedge clk); //To let RAM Read Address

if (DUT.RAM_INSTANCE.rd_addr != Parallel_ADDR[7:0]) begin
    $display("ERROR!! Incorrect Output :(");
    $stop;
end

$write("\n");
SS_n = 1;
@(negedge clk);

//-------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != IDLE) begin        //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-------------------------------------------------//



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
    $display("\n----------------> \"Testing Read Data\" <----------------\n");
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

SS_n = 0;
@(negedge clk);

//-------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != CHK_CMD) begin     //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-------------------------------------------------//

MOSI = 1;
Parallel_DATA = 10'b 11_0111_1001;
@(negedge clk);

//-------------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != READ_DATA) begin   //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-------------------------------------------------//
```

```
for (i=9 ; i >= 0 ; i=i-1) begin
    MOSI = Parallel_DATA[i];
    @(negedge clk);
end

@(negedge clk); //To let RAM Writes Data
@(negedge clk);

repeat (8) begin
    Expected_out = {Expected_out[6:0] , MISO};
    @(negedge clk);
end

if (DUT.RAM_INSTANCE.RAM[Parallel_ADDR[7:0]] != Expected_out) begin
    $display("ERROR!! Incorrect Output :(");
    $stop;
end

$write("\n");
SS_n = 1;
@(negedge clk);

//-----------------------------------------------//
    if (DUT.SPI_INSATNCE.cs != IDLE) begin        //
        $display("ERROR!! Incorrect Output :(");  //
        $stop;                                    //
    end                                           //
//-----------------------------------------------//

repeat(5) @(negedge clk);

//If the simulation reached this line then no errors were found
$display("\n--->  NO ERRORS, All Outputs are Correct :)  <--- \n");
$stop;

end

initial begin
    $monitor("clk= %b, rst_n= %b, MISO= %b, SS_n= %b ,MOSI= %b, rx_data= %b_%b, rx_valid= %b, tx_data= %b, tx_valid= %b"
    ,clk, rst_n, MISO, SS_n, MOSI, DUT.rx_data_bus[9:8], DUT.rx_data_bus[7:0], DUT.rx_valid, DUT.tx_data_bus, DUT.tx_valid);
end

endmodule
```

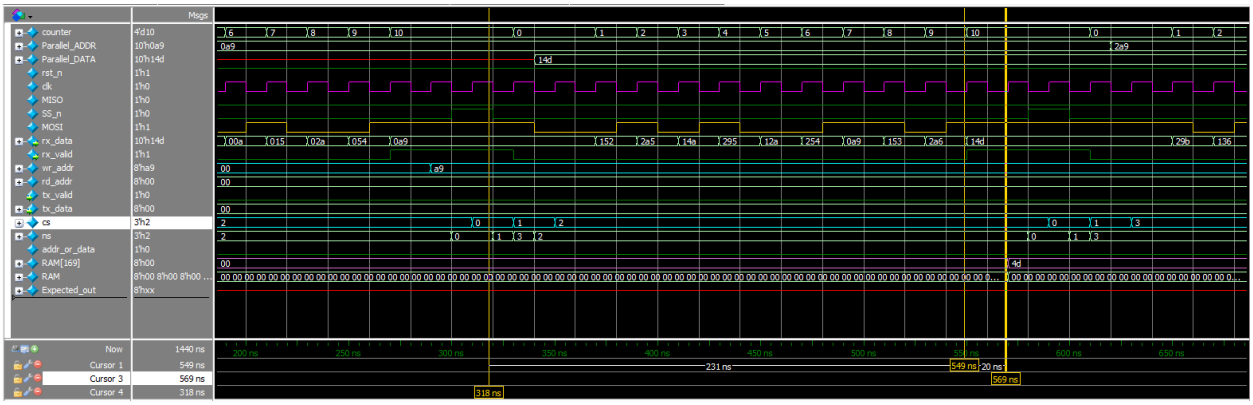
Result of the Testbench

- Testing reset in the first clock cycle.
- And then put (**SS_n = 0**) to make SPI Slave know that I want to start a connection and then **ns** will be **CHK_CMD (ns = 1)**, as shown at the first cursor.
- Then put (**MOSI = 0)** for one cycle, so ns will be **WRITE (ns = 2).**
- After 10 clock cycles, the 10 bits are converted to parallel & the **rx_valid** is high, as shown at the second cursor.
- After another clock cycle the data on **rx_data** is stored in the internal register (wr_addr) in the RAM, as shown at the third cursor.
- At the end put (**SS_n = 0**) to release the connection.



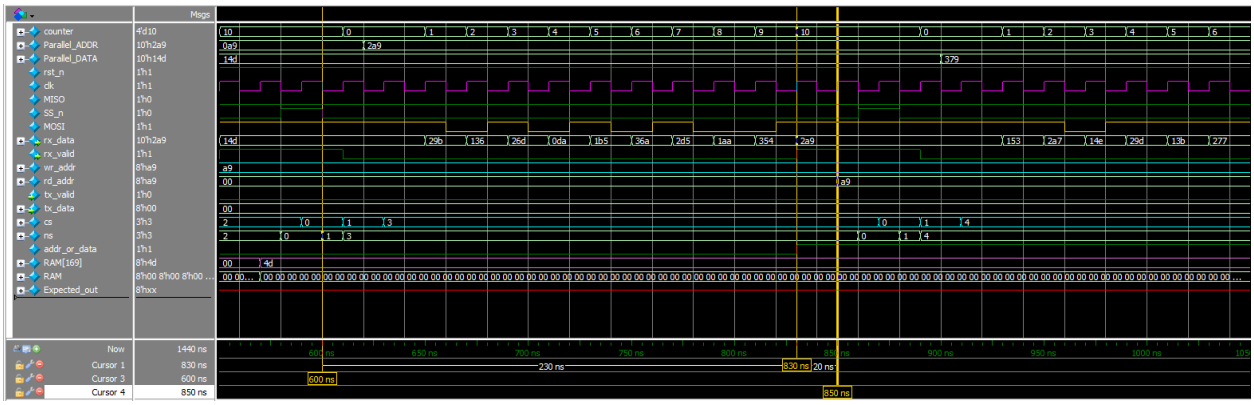

```
#
# ---------------> "Testing Reset" <---------------
#
# clk= 0, rst_n= 0, MISO= x, SS_n= 1 ,MOSI= 0, rx_data= xx_xxxxxxxx, rx_valid= x, tx_data= xxxxxxxx, tx_valid= x
# clk= 1, rst_n= 0, MISO= 0, SS_n= 1 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
#
# ---------------> "Testing Write Address" <---------------
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_00000000, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_00000001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_00000010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_00000101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00000101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00001010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_00001010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_00010101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00010101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_00101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_01010100, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_01010100, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
#
```

- Put (**SS_n = 1**) to make SPI Slave know that I want to start a connection and then **ns** will be **CHK_CMD (ns = 1)**, as shown at the first cursor.
- Then put (**MOSI = 0)** for one cycle, so ns will be **WRITE (ns = 2).**
- After 10 clock cycles, the 10 bits are converted to parallel & the **rx_valid** is high, as shown at the second cursor.
- After another clock cycle the data on **rx_data** is stored in the RAM at the location that was previously stored in (wr_addr), as shown at the third cursor.
- At the end put (**SS_n = 0**) to release the connection.



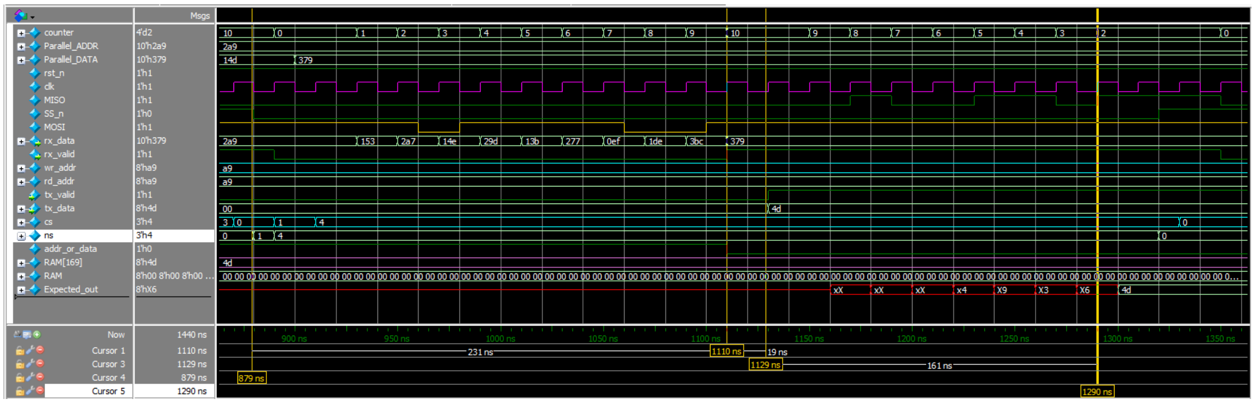

```
#
# ----------------> "Testing Write Data" <----------------
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_01010010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01010010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10100101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_10100101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_01001010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10010101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_10010101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_00101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_00101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_01010100, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_01010100, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01010011, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_01010011, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_10100110, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10100110, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 1, tx_data= 00000000, tx_valid= 0
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 1, tx_data= 00000000, tx_valid= 0
#
```

- Put (**SS_n = 1**) to make SPI Slave know that I want to start a connection and then **ns** will be **CHK_CMD (ns = 1)**, as shown at the first cursor.
- Then put (**MOSI = 1)** for one cycle, so ns will be **READ_ADD (ns = 3).**
- After 10 clock cycles, the 10 bits are converted to parallel & the **rx_valid** is high, as shown at the second cursor.
- After another clock cycle the data on **rx_data** is stored in the internal register (rd_addr) in the RAM, as shown at the third cursor.
- At the end put (**SS_n = 0**) to release the connection.




```
"
#  "Testing Read Address"
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01001101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10011011, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_10011011, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_00110110, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_00110110, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_01101101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_01101101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 00_11011010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 00_11011010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_10110101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_10110101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 11_01101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 11_01101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_11010101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_11010101, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_10101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 01_10101010, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 11_01010100, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 11_01010100, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
```

- Put (**SS_n = 1**) to make SPI Slave know that I want to start a connection and then **ns** will be **CHK_CMD (ns = 1)**, as shown at the first cursor.
- Then put (**MOSI = 1)** for one cycle, so ns will be **READ_DATA (ns = 4).**
- After 10 clock cycles, the 10 bits are converted into parallel data & the **rx_valid** is high, as shown at the second cursor. (however the other 8 bits are garbage and will be ignored in RAM).
- After another clock cycle the data on the data stored in the RAM at the location that was previously stored in (rd_addr) are available on **tx_data** and **tx_valid** is high, as shown at the third cursor.
- After another 8 clock cycles, the data that was on **tx_data** are converted into serial data (**MISO**), as shown at the fourth cursor.



```
#
# ---------------> "Testing Read Data" <---------------
#
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 1, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10101001, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01010011, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 01_01010011, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 1, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 1, rx_data= 10_10100111, rx_valid= 0, tx_data= 00000000, tx_valid= 0
# clk= 0, rst_n= 1, MISO= 0, SS_n= 0 ,MOSI= 0, rx_data= 10_10100111, rx_valid= 0, tx_data= 00000000, tx_valid= 0
```

```
# clk= 0, rst_n= 1, MISO= 1, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 1, tx_data= 01001101, tx_valid= 1
# clk= 1, rst_n= 1, MISO= 1, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 1, tx_data= 01001101, tx_valid= 1
# clk= 0, rst_n= 1, MISO= 1, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 1, tx_data= 01001101, tx_valid= 1
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 0, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
# clk= 1, rst_n= 1, MISO= 0, SS_n= 1 ,MOSI= 1, rx_data= 11_01111001, rx_valid= 0, tx_data= 01001101, tx_valid= 1
#
# ---> NO ERRORS, All Outputs are Correct :) <---
#
# ** Note: $stop    : TB_2.v(267)
#    Time: 1440 ns  Iteration: 1  Instance: /Sh3ban_TB
# Break in Module Sh3ban_TB at TB_2.v line 267
```

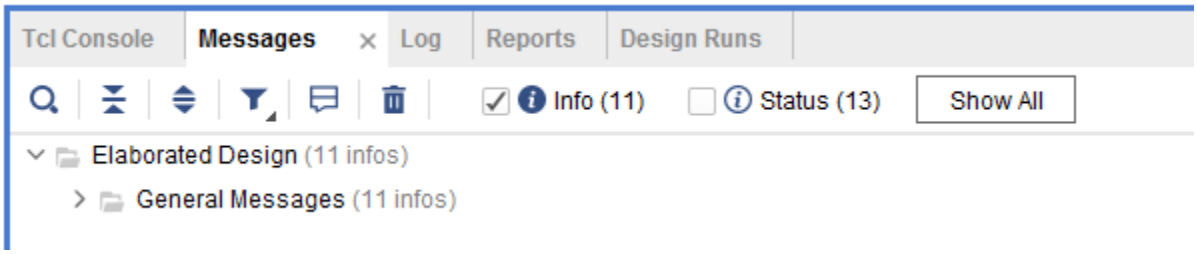
Constraint File

Constraints_basys3.xdc

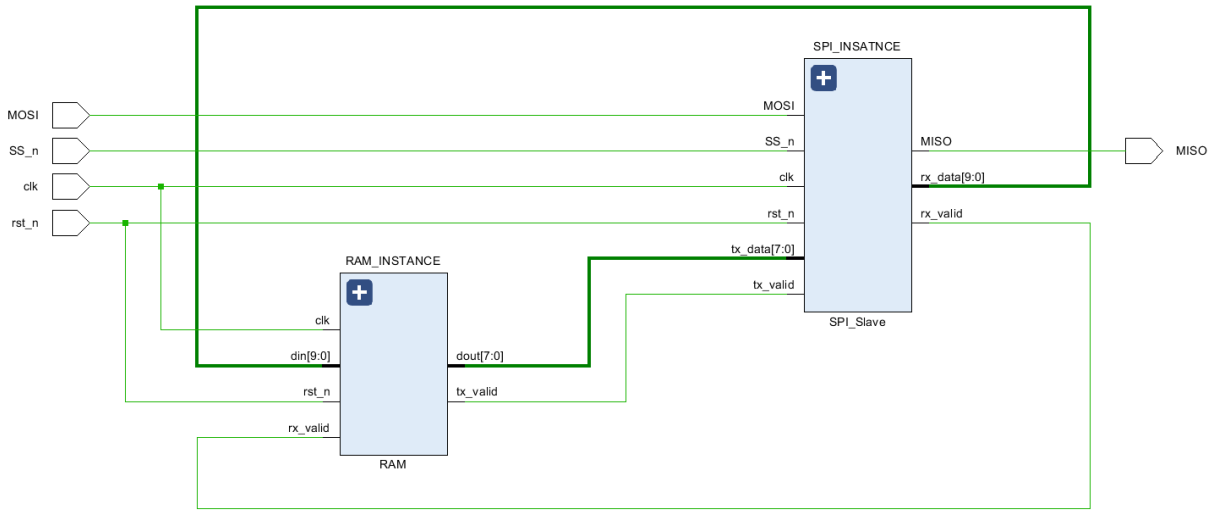
```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]


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


## LEDs
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports MISO]


##Buttons
# set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

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

create_debug_core u_ila_0 ila
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
```

Elaboration

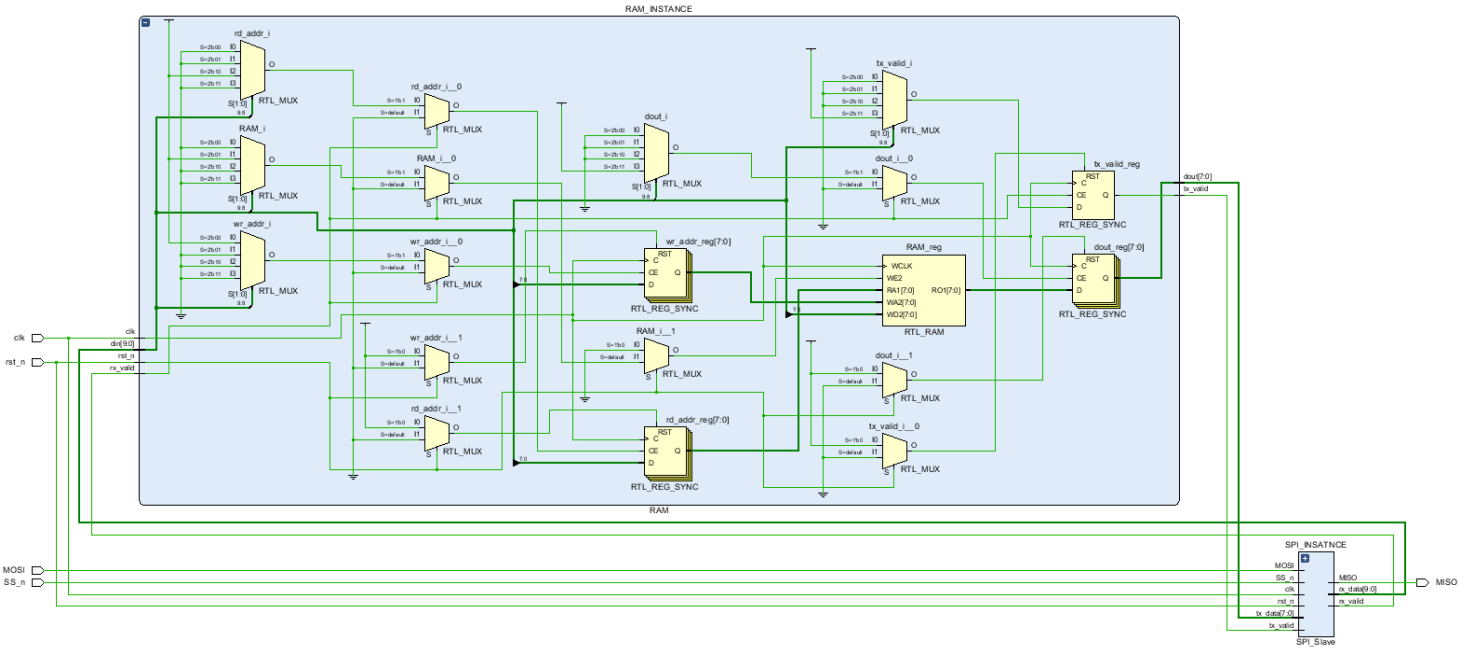
- Messages tab after elaboration.
- As shown in the following snippet → **No Critical Warnings or Errors**.



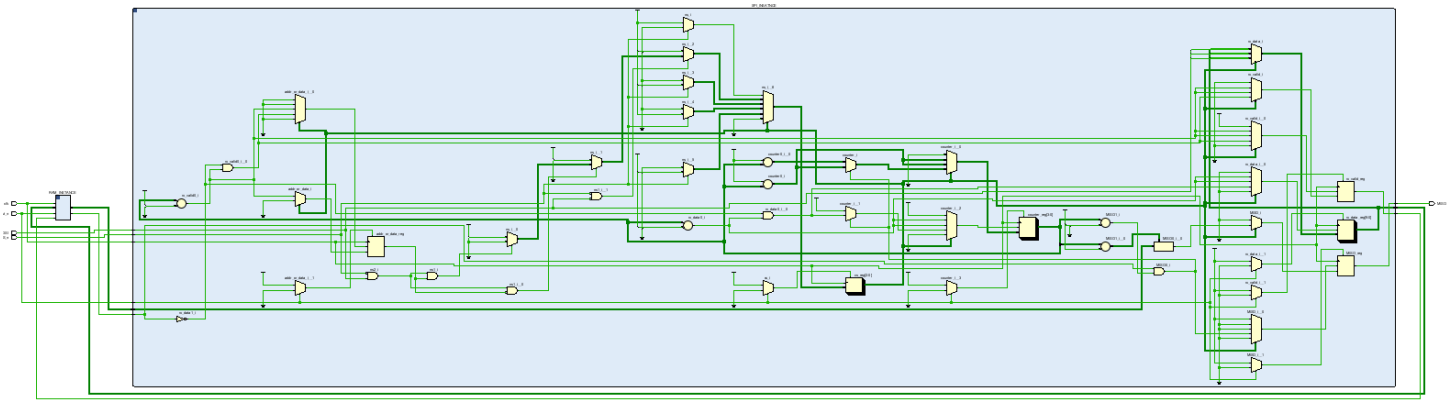
- Schematic after elaboration.



- Schematic for RAM instance (Lower Module).



- Schematic for SPI slave interface instance (Lower Module).

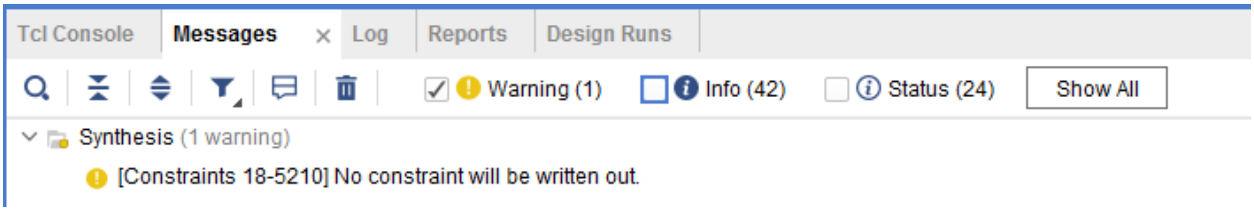


Synthesis (Sequential Encoding)

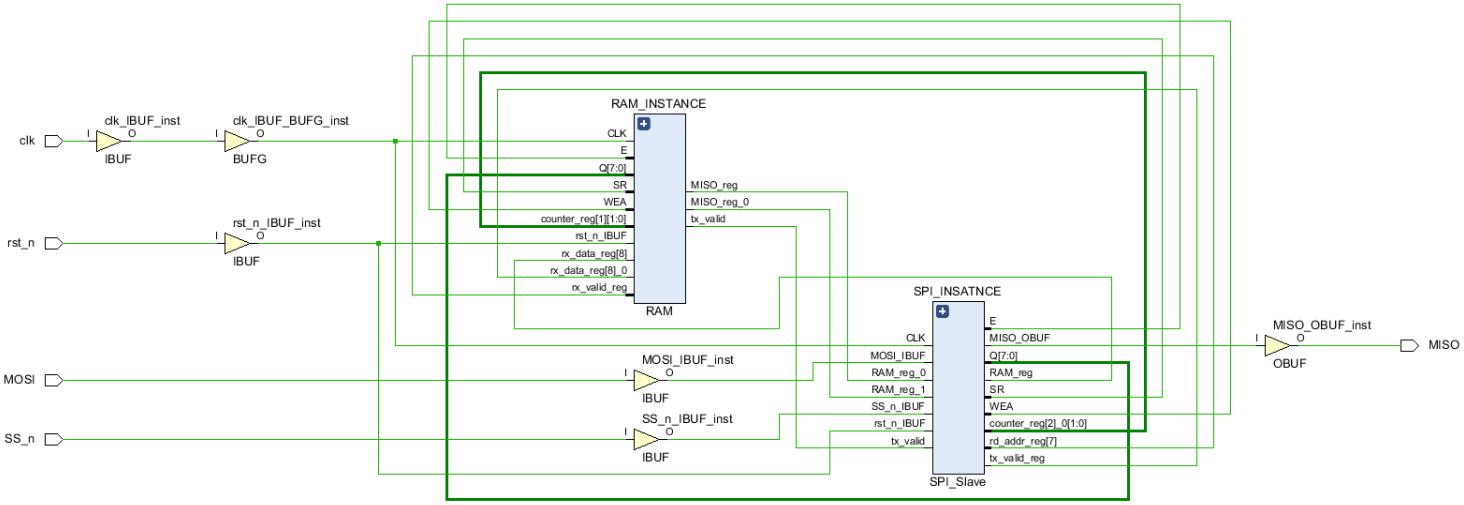
- Synthesis Report.

```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "sequential" *) [H:/Digital_Design_Diploma/Project 2/SPI_Slave.v:18]

---------------------------------------------------------------------------------
INFO: [Synth 8-802] inferred FSM for state register 'cs_reg' in module 'SPI_Slave'
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
---------------------------------------------------------------------------------------------------
                   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'
---------------------------------------------------------------------------------
```

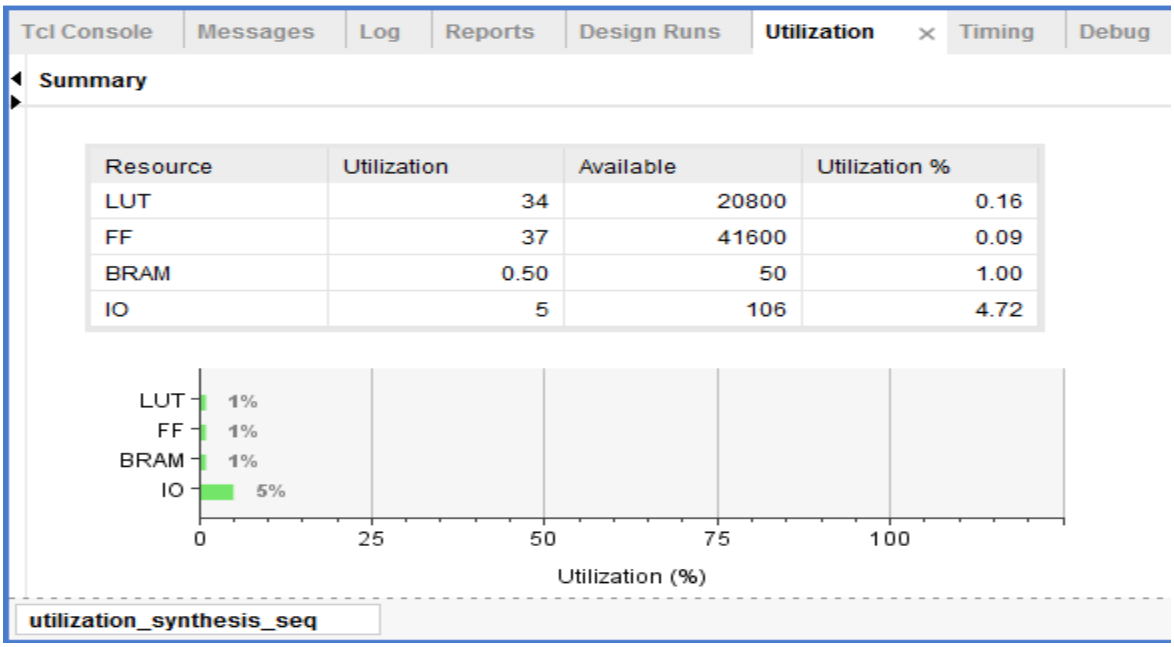
- Messages tab after running synthesis.
- As shown in the snippet → **No Critical Warnings or Errors**.



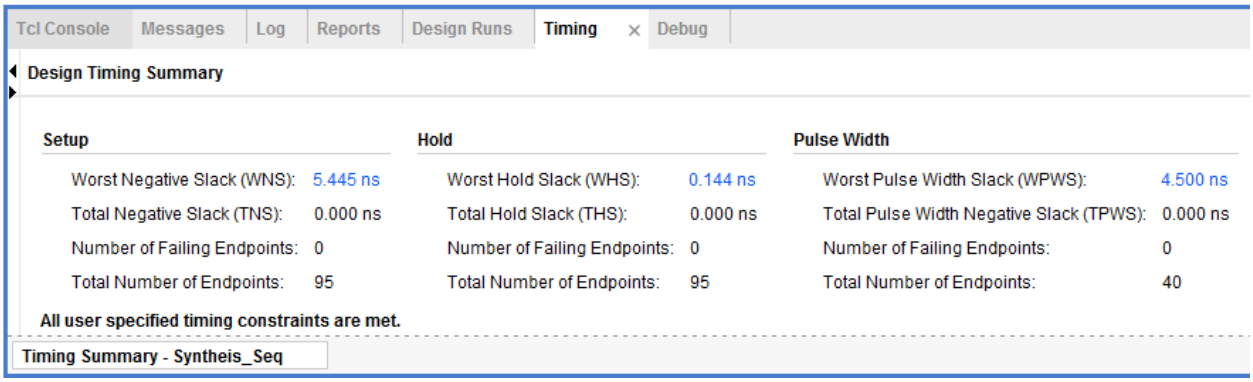
- Schematic after running synthesis.



- Utilization after running synthesis.



- Timing Summary after running synthesis.
- As shown in the snippet:
  - **Both setup time & hold time slacks are positive.**
  - **Which means that there no timing violations.**



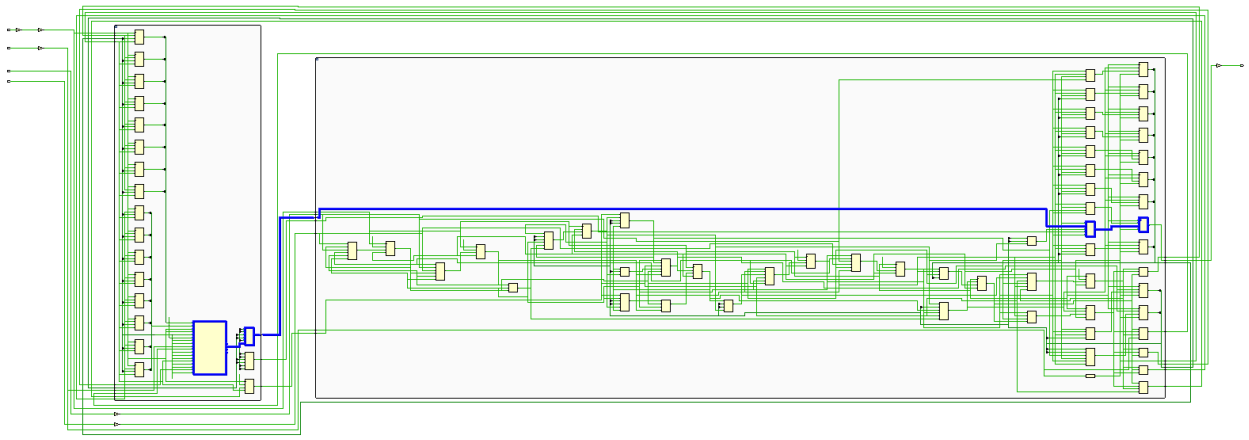
- The critical path.

Tcl Console | Messages | Log | Reports | Design Runs | Utilization | Timing | Debug

Intra-Clock Paths - sys_clk_pin - Setup

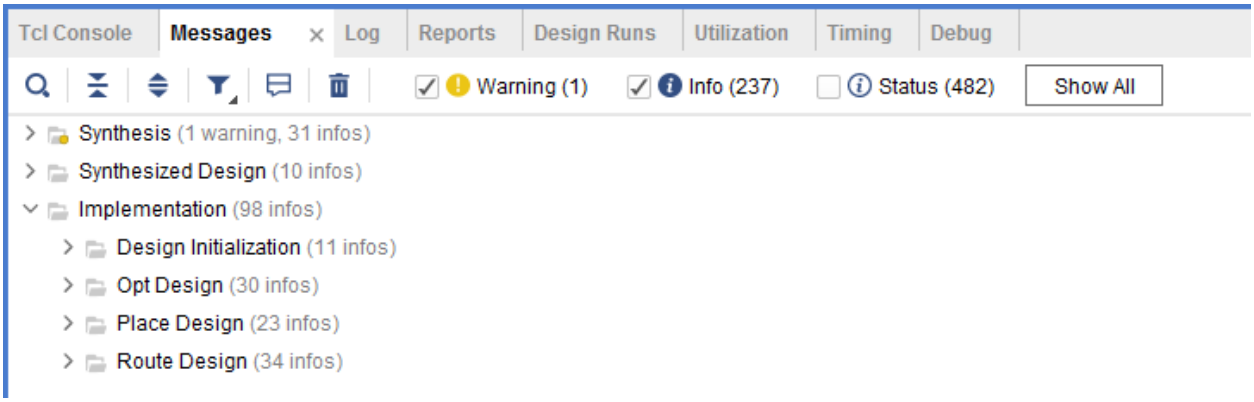
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Excep
Path 1	5.445	2	3	1	RAM_INSTANC...g/CLKBWRCLK	SPI_INSATNCE/MISO_reg/D	4.404	2.702	1.702	10.0	sys_clk_pin	sys_clk_pin	
Path 2	6.956	2	3	7	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/MISO_reg/CE	2.662	0.875	1.787	10.0	sys_clk_pin	sys_clk_pin	
Path 3	6.962	2	3	7	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[0]/CE	2.656	0.875	1.781	10.0	sys_clk_pin	sys_clk_pin	
Path 4	6.962	2	3	7	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[1]/CE	2.656	0.875	1.781	10.0	sys_clk_pin	sys_clk_pin	
Path 5	6.962	2	3	7	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[2]/CE	2.656	0.875	1.781	10.0	sys_clk_pin	sys_clk_pin	
Path 6	6.962	2	3	7	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[3]/CE	2.656	0.875	1.781	10.0	sys_clk_pin	sys_clk_pin	
Path 7	6.964	1	2	6	SPI_INSATNCE/rx_valid_reg/C	RAM_INSTANCE/RAM_reg/WEA[0]	2.324	0.751	1.573	10.0	sys_clk_pin	sys_clk_pin	
Path 8	6.964	1	2	6	SPI_INSATNCE/rx_valid_reg/C	RAM_INSTANCE/RAM_reg/WEA[1]	2.324	0.751	1.573	10.0	sys_clk_pin	sys_clk_pin	
Path 9	6.965	2	3	21	SPI_INSATNCE/F...l_cs_reg[1]/C	SPI_INSATNCE/rx_data_reg[0]/CE	2.653	0.875	1.778	10.0	sys_clk_pin	sys_clk_pin	
Path 10	6.965	2	3	21	SPI_INSATNCE/F...l_cs_reg[1]/C	SPI_INSATNCE/rx_data_reg[1]/CE	2.653	0.875	1.778	10.0	sys_clk_pin	sys_clk_pin	

Timing Summary - Syntheis_Seq

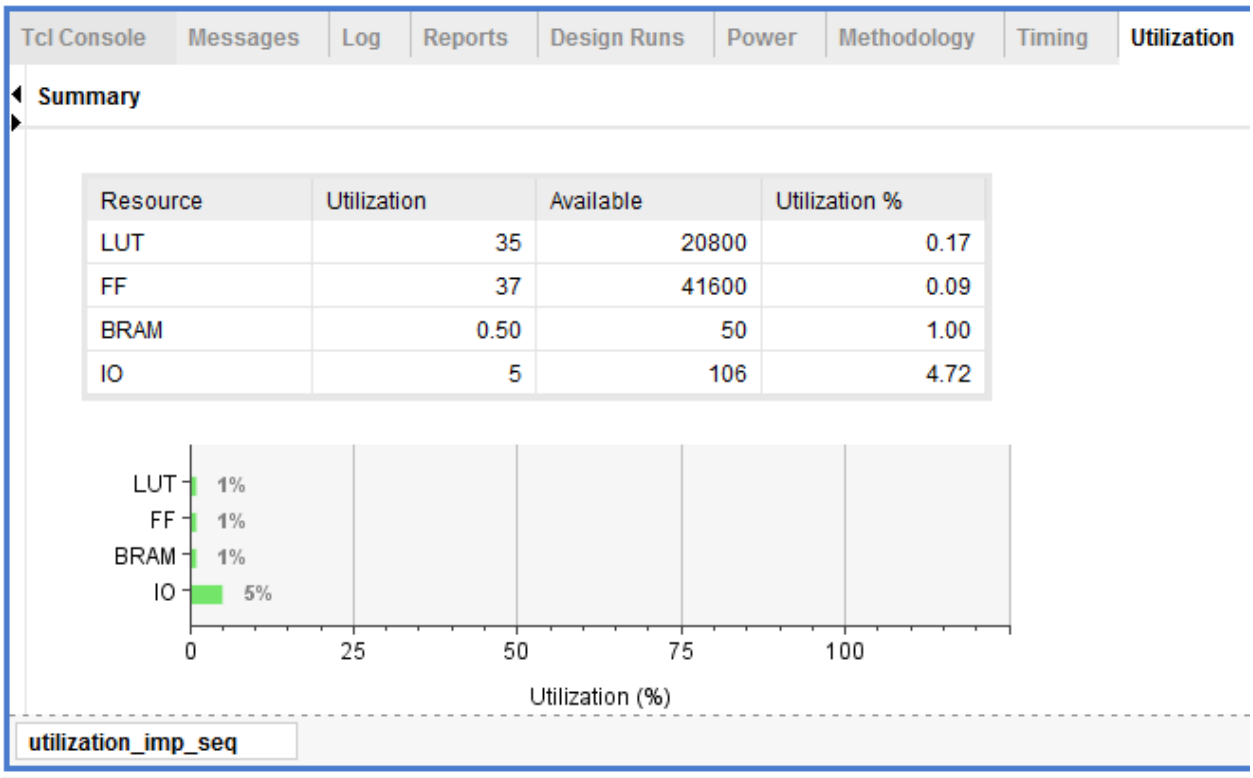


Implementation (Sequential Encoding)

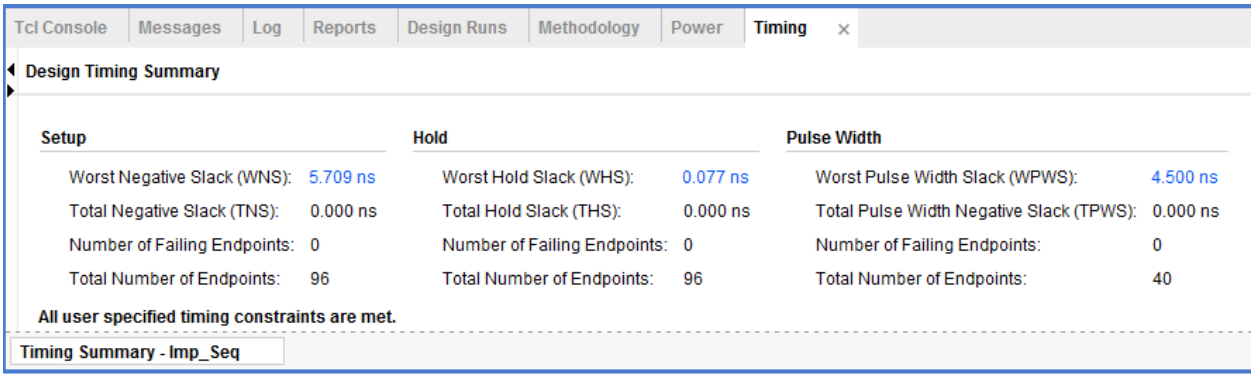
- Messages tab after running implementation.
- As shown in the snippet → **No Critical Warnings or Errors**.



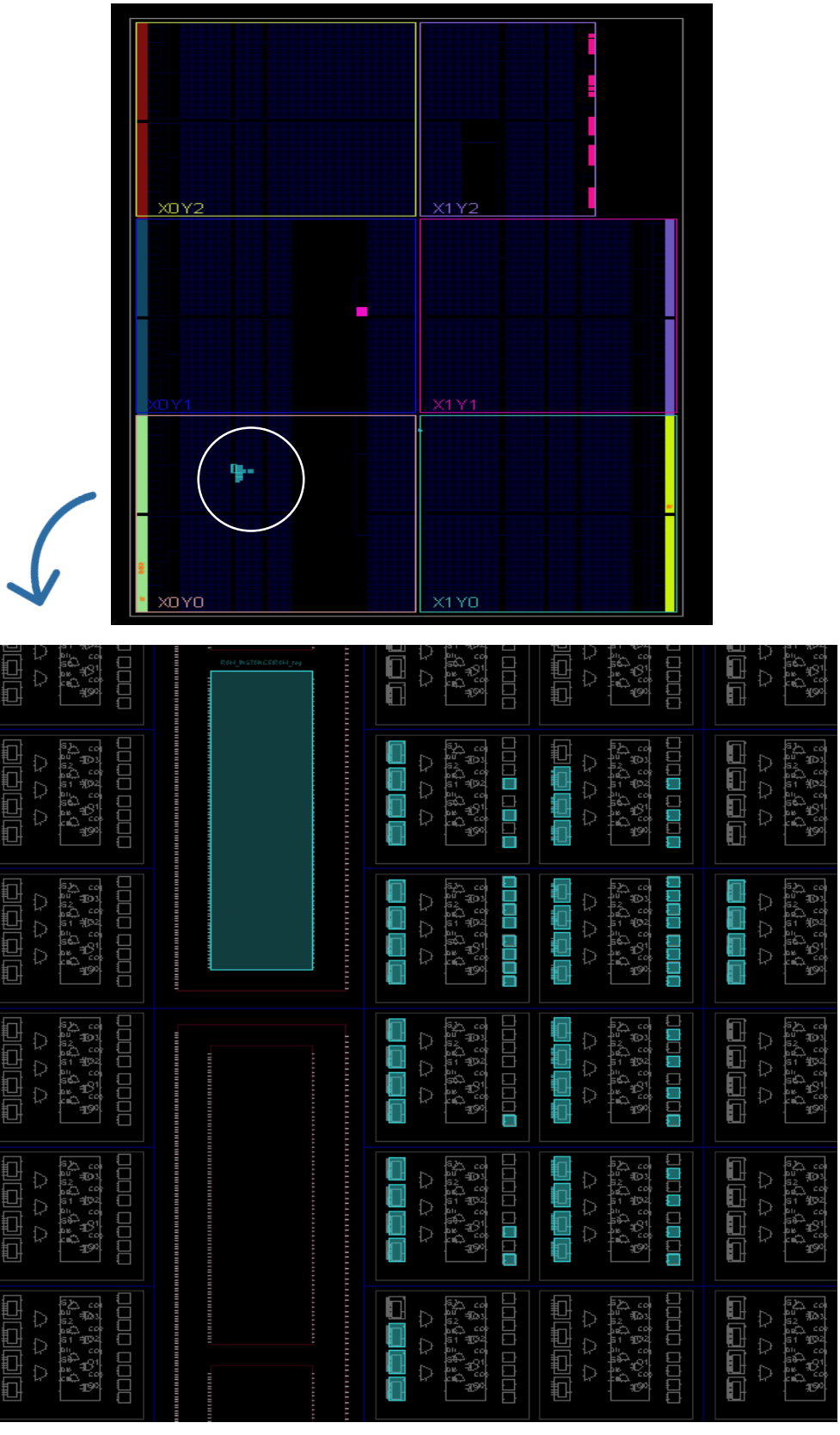
- Utilization after implementation.



- Timing Summary after implementation**.**
- As shown in the snippet:
  - **Both setup time & hold time slacks are positive.**
  - **Which means that there no timing violations.**



- FPGA Device.



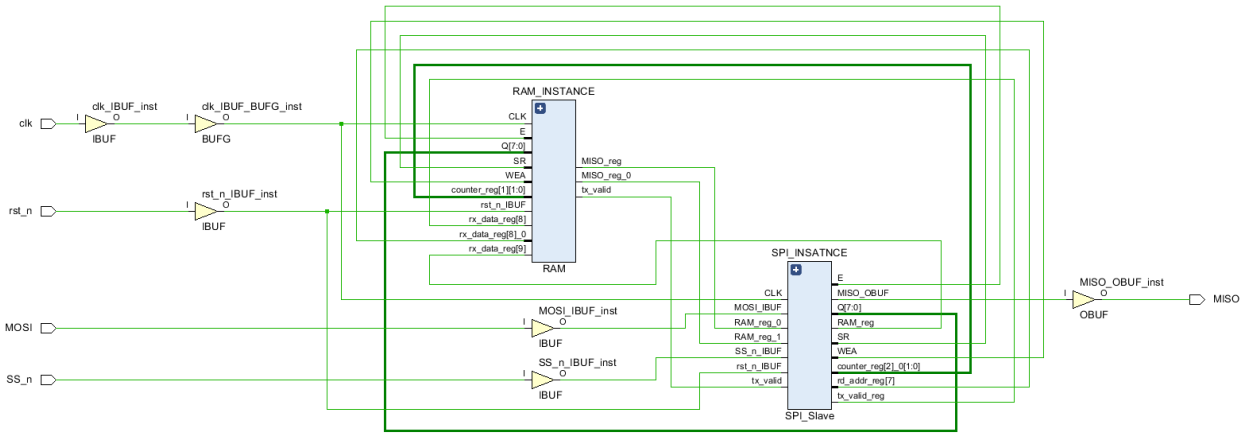
Synthesis (Gray Encoding)

- Synthesis Report.

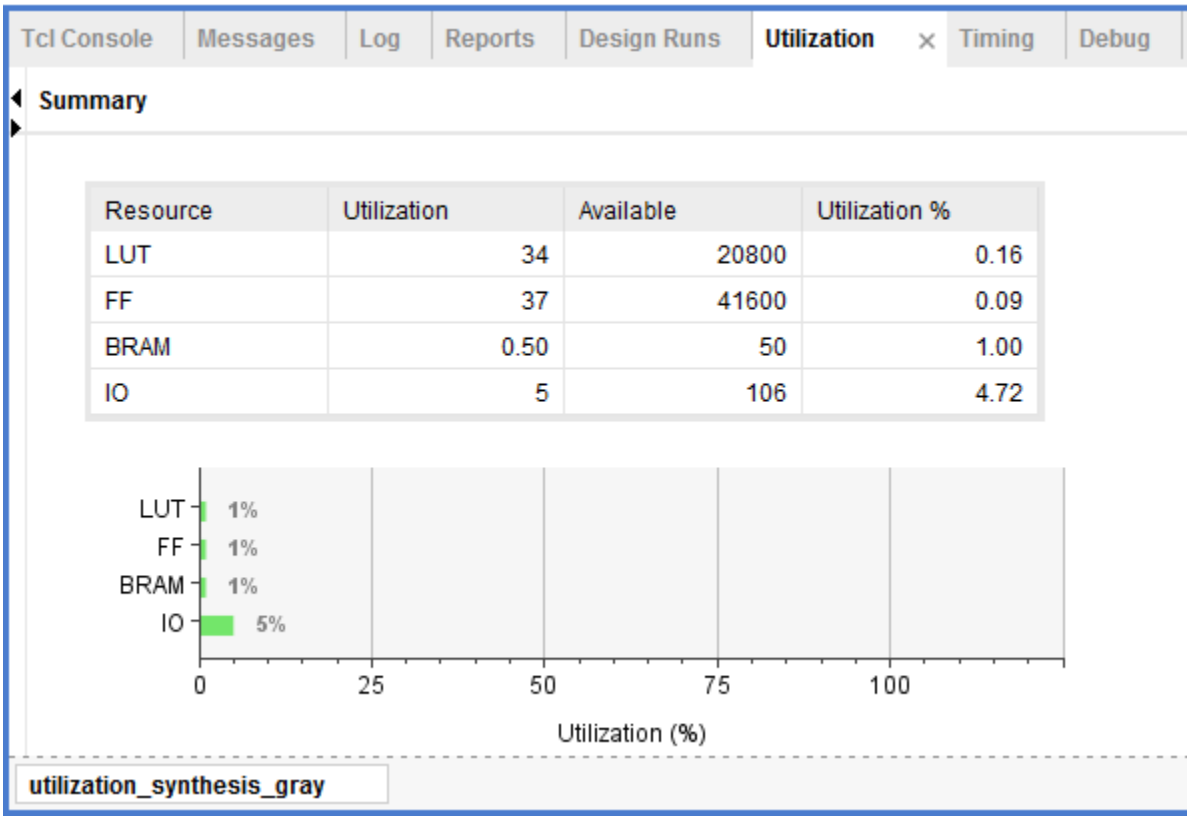
```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *) [H:/Digital_Design_Diploma/Project 2/SPI_Slave.v:18]
---------------------------------------------------------------------------------
INFO: [Synth 8-802] inferred FSM for state register 'cs_reg' in module 'SPI_Slave'
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
---------------------------------------------------------------------------------------------------
                   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'
---------------------------------------------------------------------------------
```

- Messages tab after running synthesis.
- As shown in the snippet → **No Critical Warnings or Errors**.

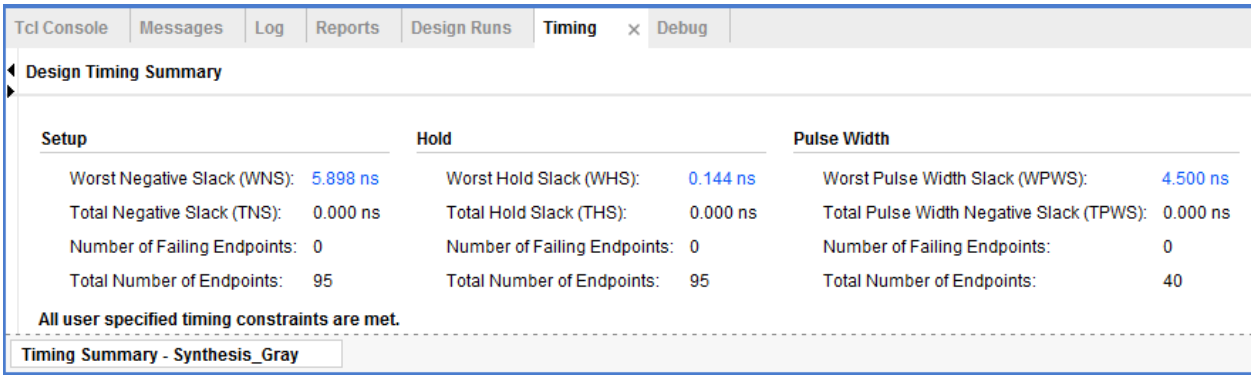
- Schematic after running synthesis.



- Utilization after running synthesis.



- Timing Summary after running synthesis.
- As shown in the snippet:
  - **Both setup time & hold time slacks are positive.**
  - **Which means that there no timing violations.**



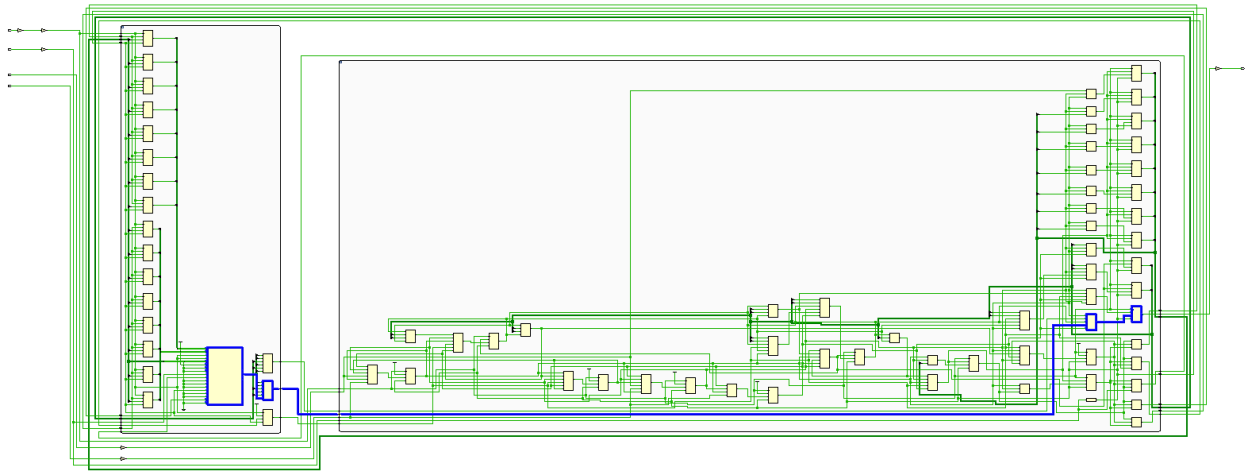
- The critical path.

Tcl Console | Messages | Log | Reports | Design Runs | Timing | Debug

Intra-Clock Paths - sys_clk_pin - Setup

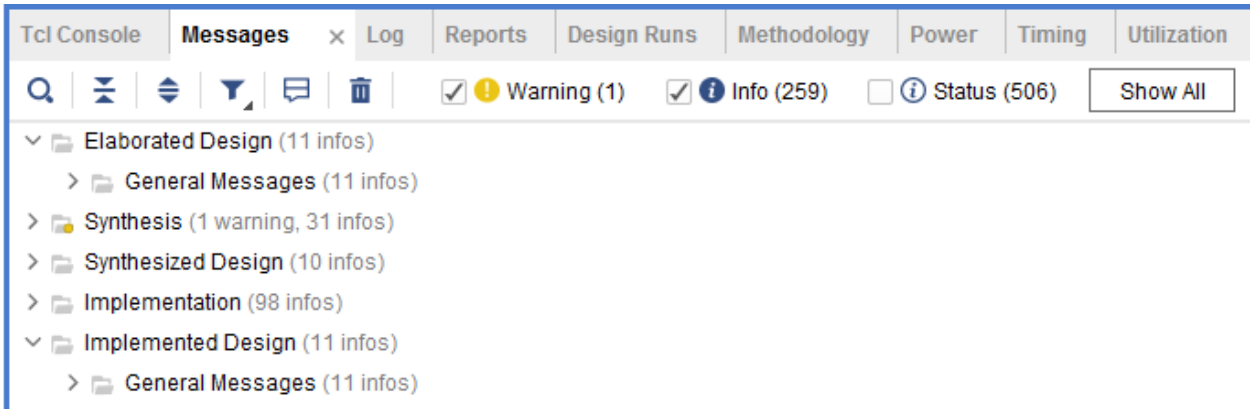
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	5.898	2	3	1	RAM_INSTANC...g/CLKBWRCLK	SPI_INSATNCE/MISO_reg/D	3.951	2.702	1.249	10.0	sys_clk_pin	sys_clk_pin
Path 2	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[0]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 3	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[1]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 4	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[2]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 5	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[3]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 6	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[4]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 7	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[5]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 8	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[6]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 9	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[7]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin
Path 10	6.962	2	3	10	SPI_INSATNCE/counter_reg[2]/C	SPI_INSATNCE/rx_data_reg[8]/CE	2.656	0.901	1.755	10.0	sys_clk_pin	sys_clk_pin

Timing Summary - Synthesis_Gray

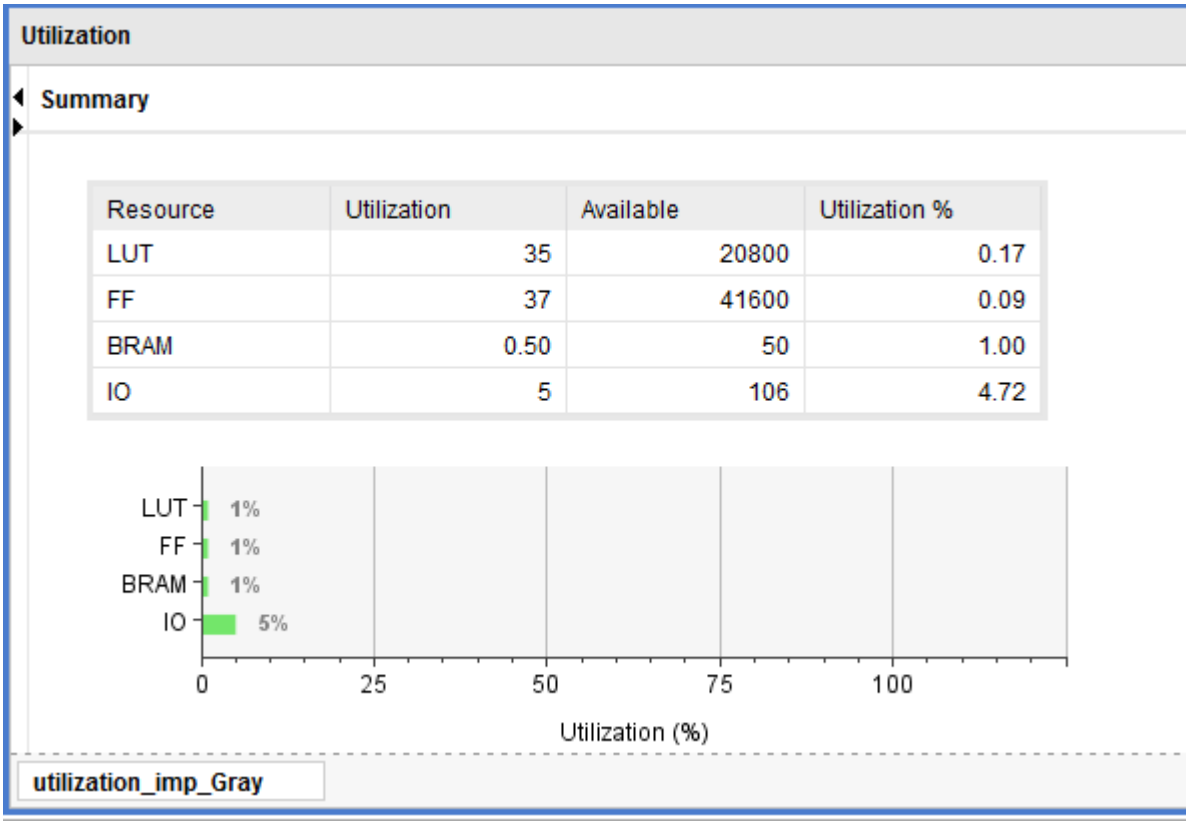


Implementation (Gray Encoding)

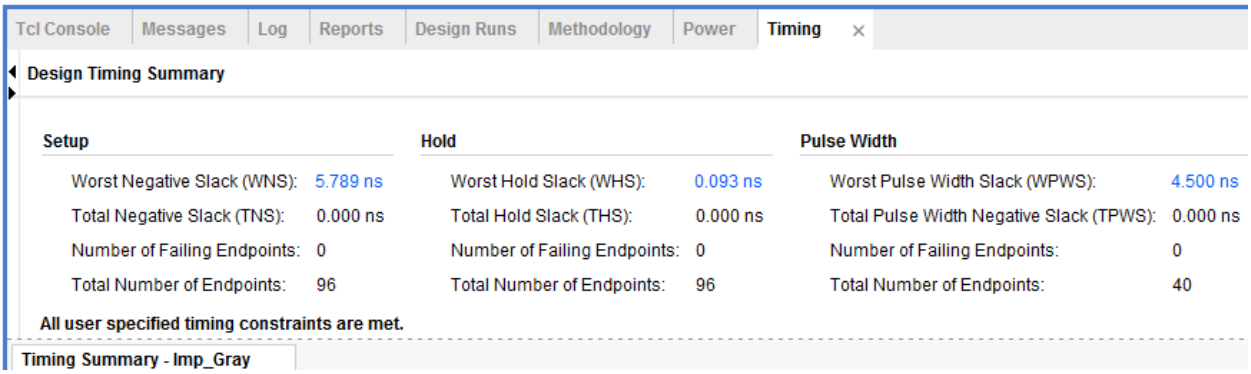
- Messages tab after running implementation.
- As shown in the snippet → **No Critical Warnings or Errors**.



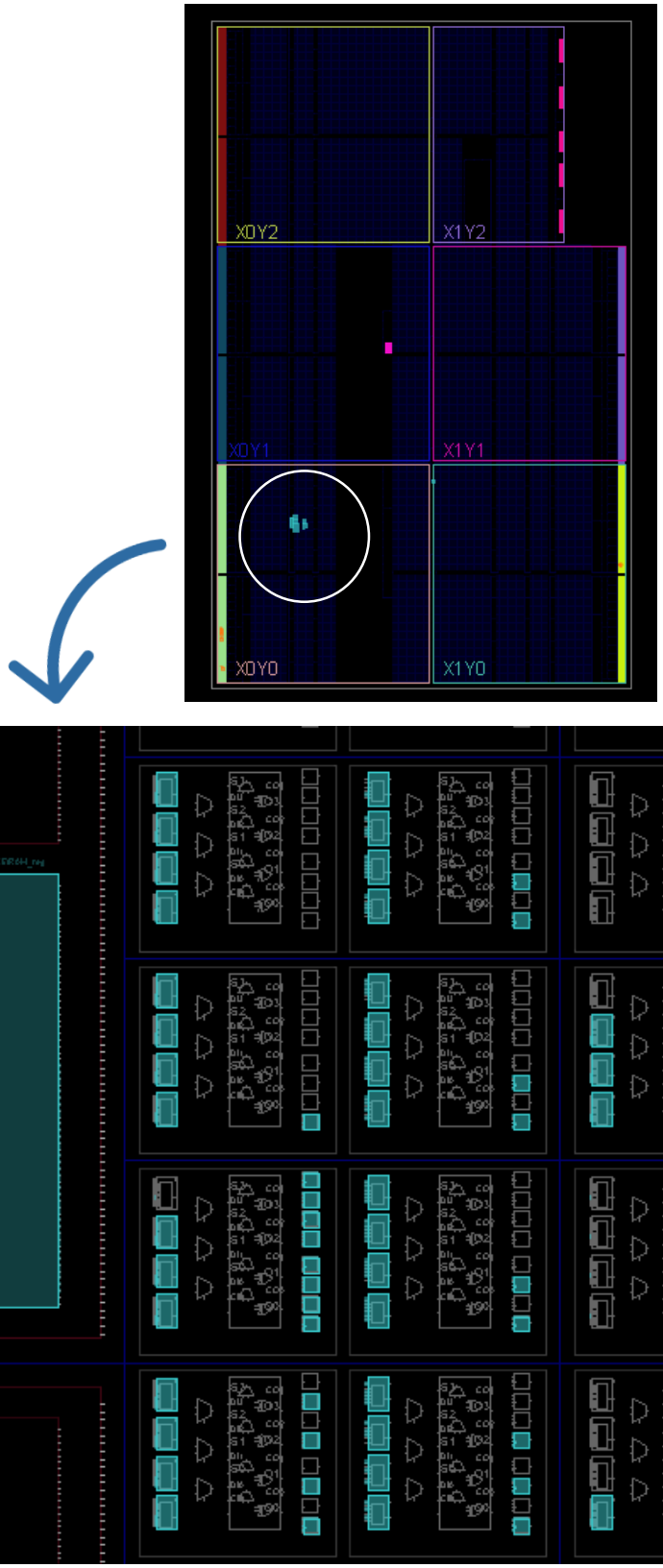
- Utilization after implementation.



- Timing Summary after implementation**.**
- As shown in the snippet:
  - **Both setup time & hold time slacks are positive.**
  - **Which means that there no timing violations.**



- FPGA Device.

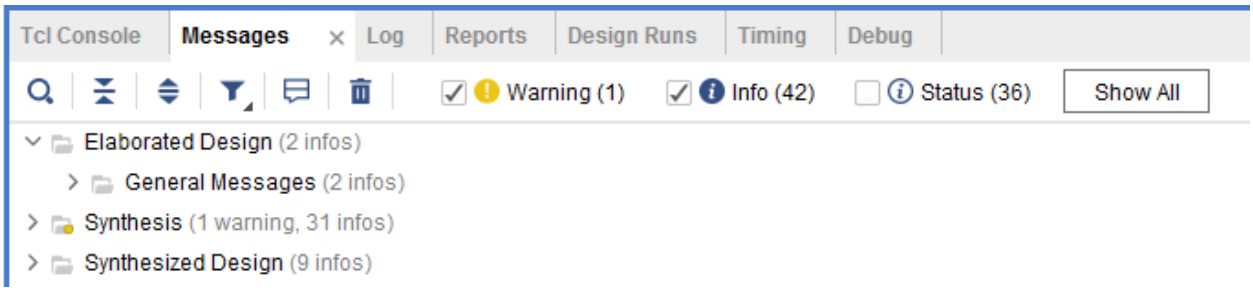


Synthesis (One-Hot Encoding)

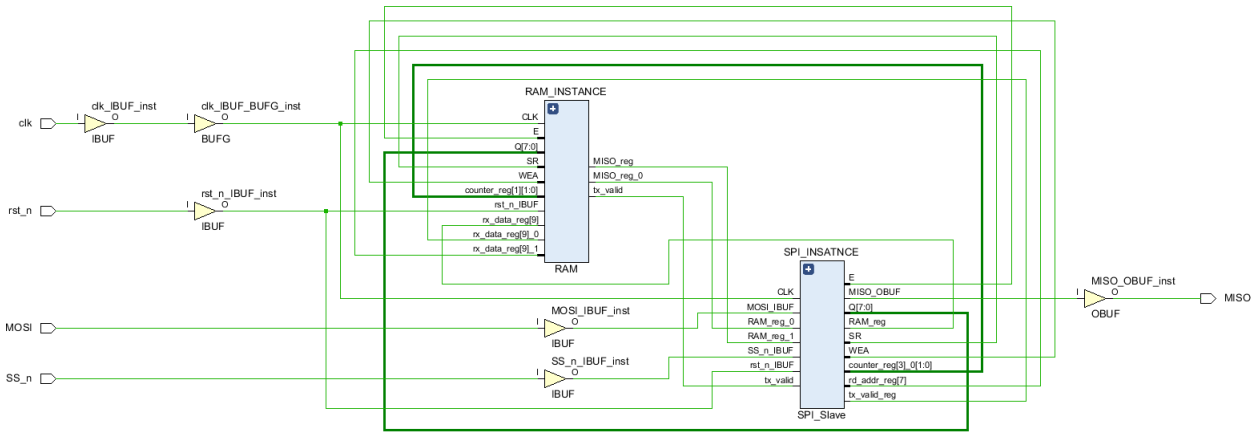
- Synthesis Report.

```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "one_hot" *) [H:/Digital_Design_Diploma/Project 2/SPI_Slave.v:18]
---------------------------------------------------------------------------------
INFO: [Synth 8-802] inferred FSM for state register 'cs_reg' in module 'SPI_Slave'
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
---------------------------------------------------------------------------------------------------
                   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'
---------------------------------------------------------------------------
```

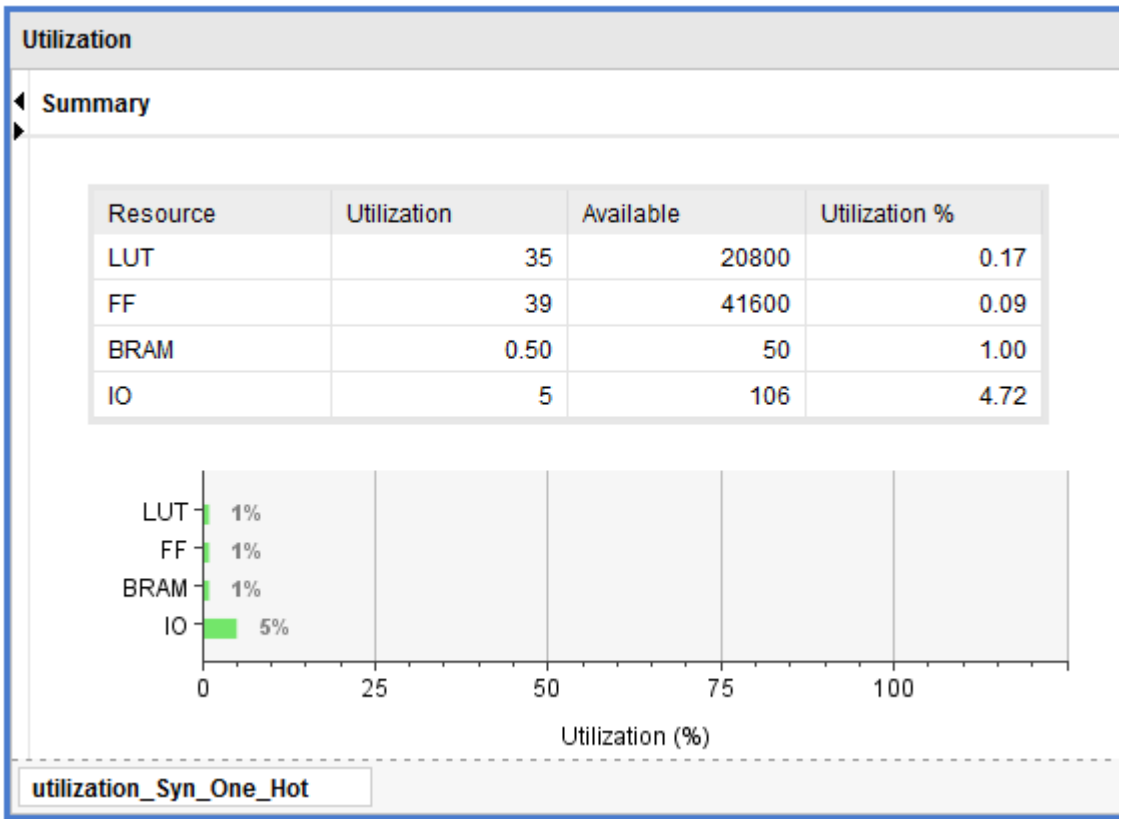
- Messages tab after running synthesis.
- As shown in the snippet → **No Critical Warnings or Errors**.



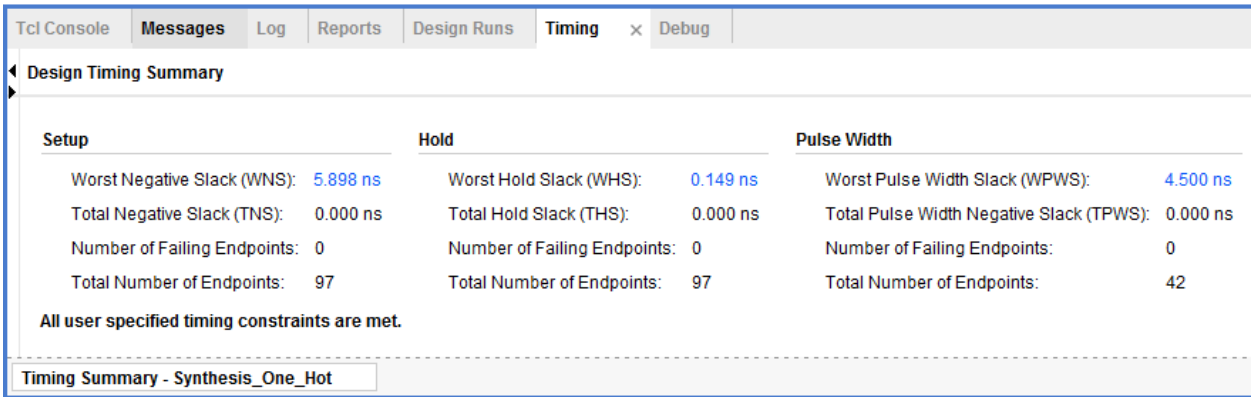
- Schematic after running synthesis.



- Utilization after running synthesis.



- Timing Summary after running synthesis.
- As shown in the snippet:
  - **Both setup time & hold time slacks are positive.**
  - **Which means that there no timing violations.**



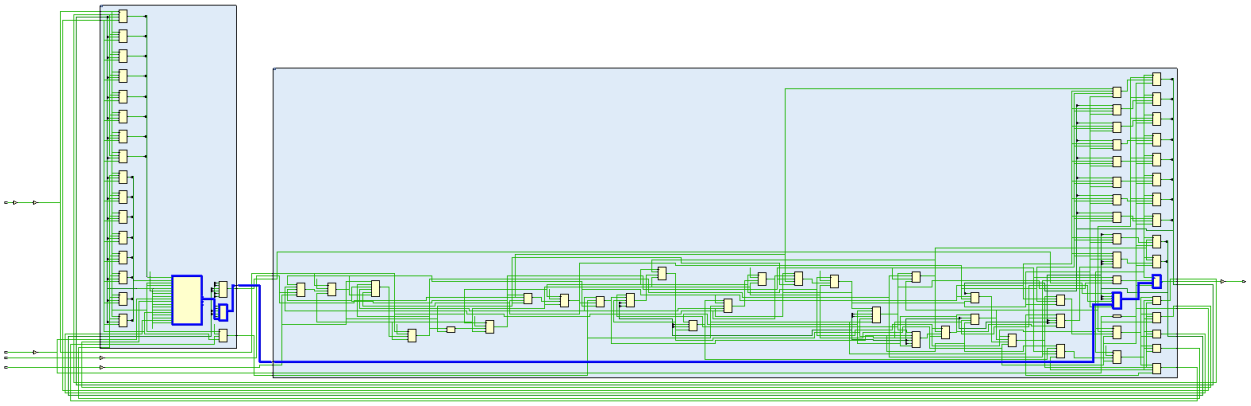
- The critical path.

Tcl Console | Messages | Log | Reports | Design Runs | Timing | Debug

Intra-Clock Paths - sys_clk_pin - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	5.898	2	3	1	RAM_INSTANC...g/CLKBWRCLK	SPI_INSATNCE/MISO_reg/D	3.951	2.702	1.249	10.0	sys_clk_pin	sys_clk_pin
Path 2	6.385	3	4	11	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[0]/CE	3.233	0.999	2.234	10.0	sys_clk_pin	sys_clk_pin
Path 3	6.385	3	4	11	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[1]/CE	3.233	0.999	2.234	10.0	sys_clk_pin	sys_clk_pin
Path 4	6.385	3	4	11	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[2]/CE	3.233	0.999	2.234	10.0	sys_clk_pin	sys_clk_pin
Path 5	6.385	3	4	11	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/counter_reg[3]/CE	3.233	0.999	2.234	10.0	sys_clk_pin	sys_clk_pin
Path 6	6.775	2	3	18	SPI_INSATNCE/F...t_cs_reg[4]/C	SPI_INSATNCE/MISO_reg/CE	2.843	0.875	1.968	10.0	sys_clk_pin	sys_clk_pin
Path 7	6.964	1	2	6	SPI_INSATNCE/rx_valid_reg/C	RAM_INSTANCE/RAM_reg/WEA[0]	2.324	0.751	1.573	10.0	sys_clk_pin	sys_clk_pin
Path 8	6.964	1	2	6	SPI_INSATNCE/rx_valid_reg/C	RAM_INSTANCE/RAM_reg/WEA[1]	2.324	0.751	1.573	10.0	sys_clk_pin	sys_clk_pin
Path 9	6.982	2	3	11	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/rx_data_reg[0]/CE	2.636	0.875	1.761	10.0	sys_clk_pin	sys_clk_pin
Path 10	6.982	2	3	11	RAM_INSTANCE/tx_valid_reg/C	SPI_INSATNCE/rx_data_reg[1]/CE	2.636	0.875	1.761	10.0	sys_clk_pin	sys_clk_pin

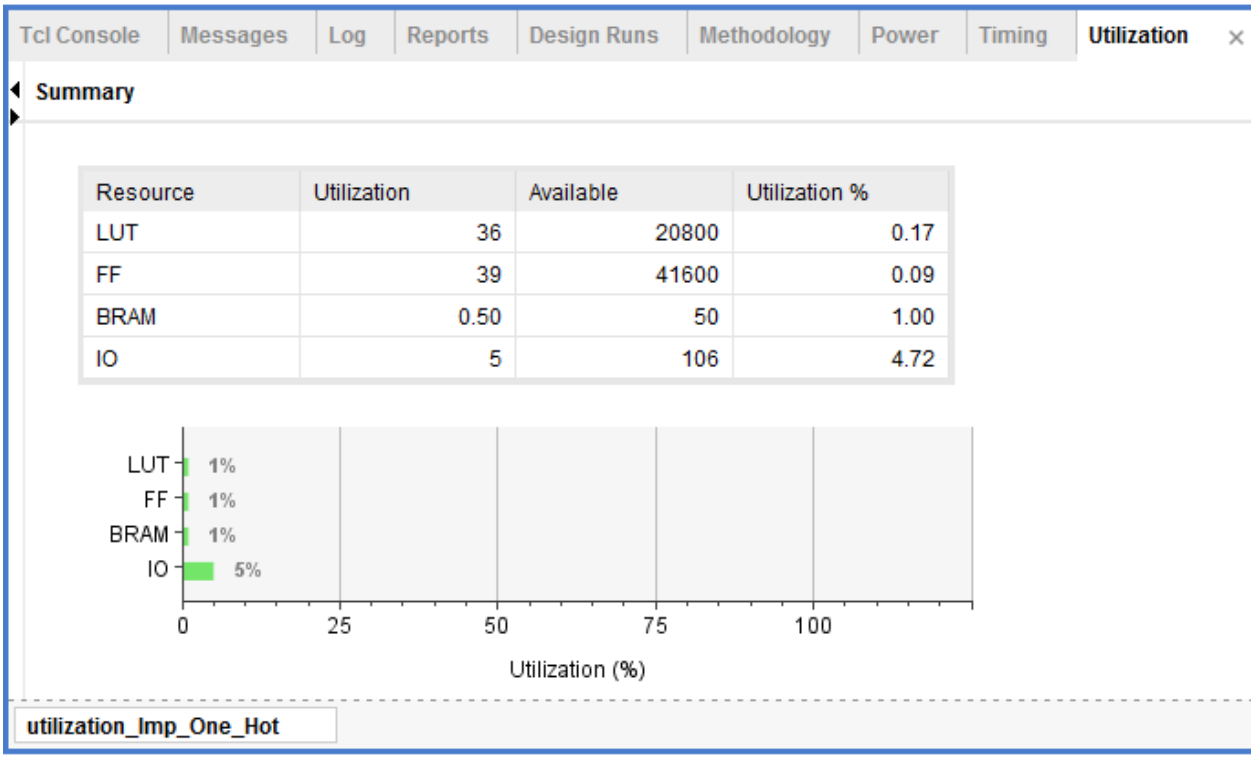
Timing Summary - Synthesis_One_Hot



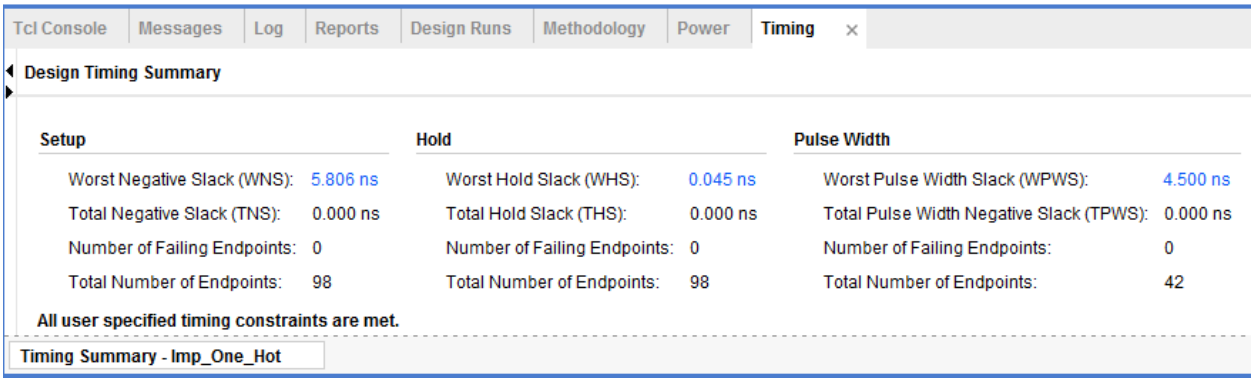
Implementation (One-Hot Encoding)

- Messages tab after running implementation.
- As shown in the snippet → **No Critical Warnings or Errors**.

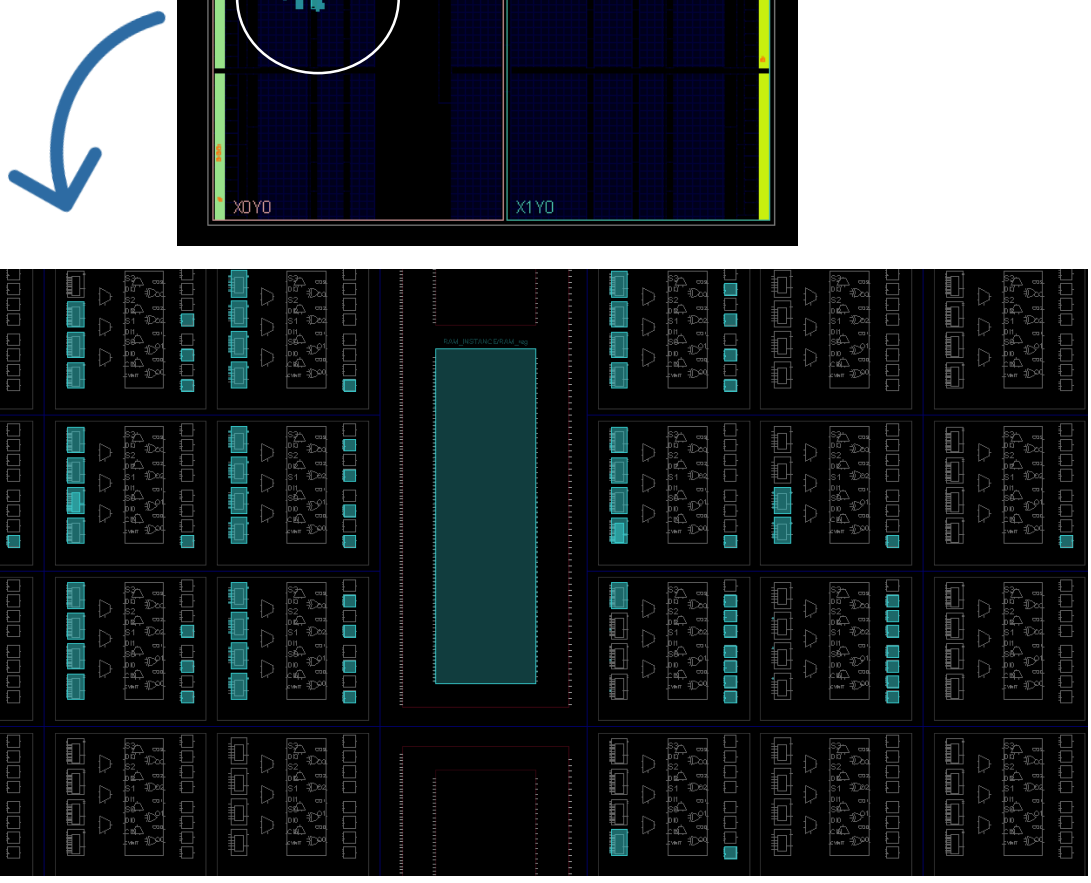
- Utilization after implementation.



- Timing Summary after implementation**.**
- As shown in the snippet:
  - **Both setup time & hold time slacks are positive.**
  - **Which means that there no timing violations.**

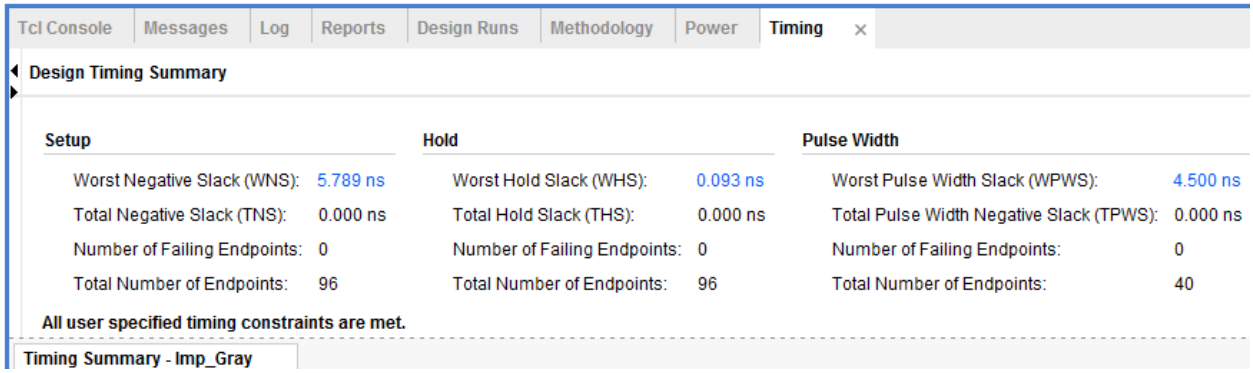


- FPGA Device.

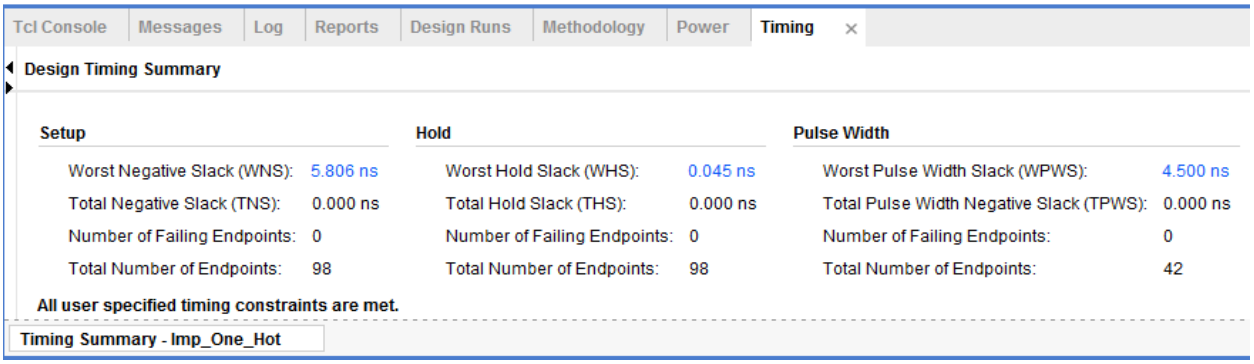


Comparing the Worst Slack

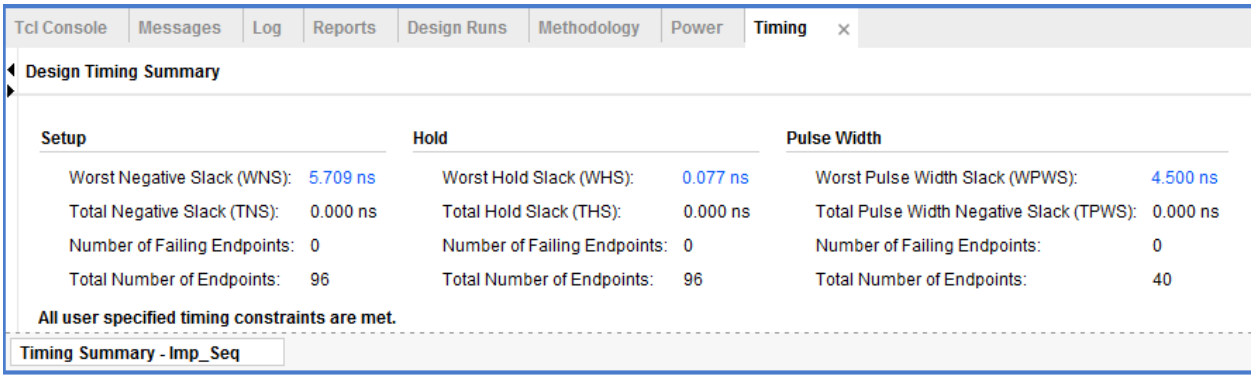
1. Gray Encoding:



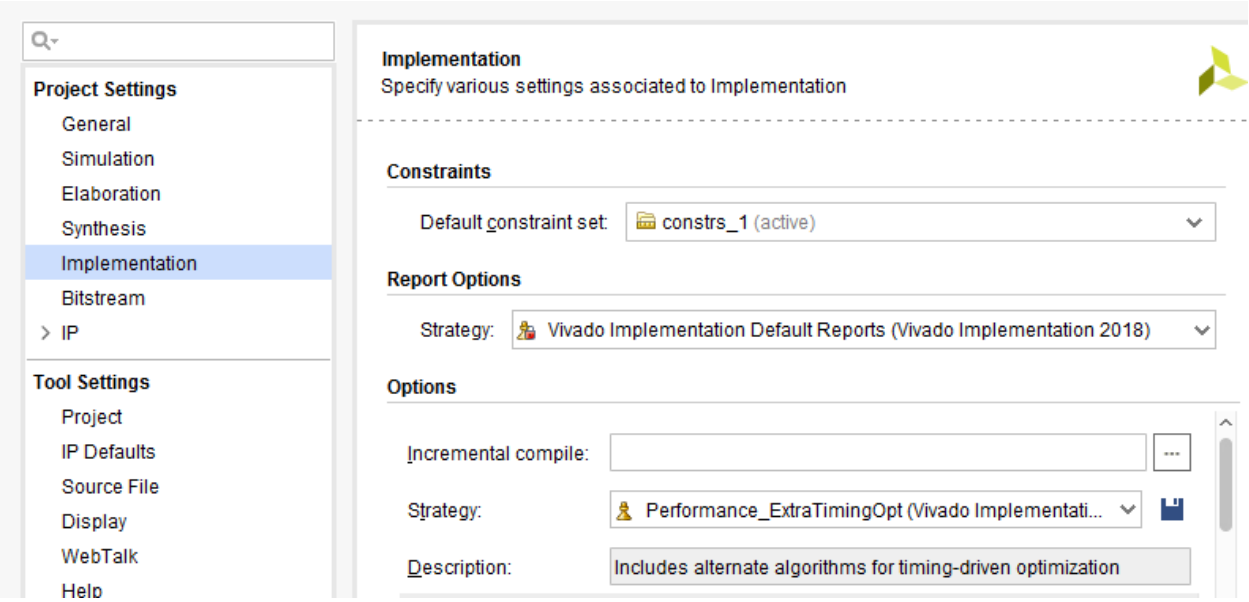
2. One-Hot Encoding:



3. Sequential Encoding:



- **Note:**
  The Previous implementations were optimized to get the best timing.



- **Table of timing summary for all encoding schemes:**

	Synthesis		Implementation	
	Setup	**Hold**	**Setup**	**Hold**
Gray	5.898	0.144	5.789	0.093
One Hot	5.898	0.149	5.806	0.045
Sequential	5.445	0.144	5.709	0.077

- **Conclusion:**

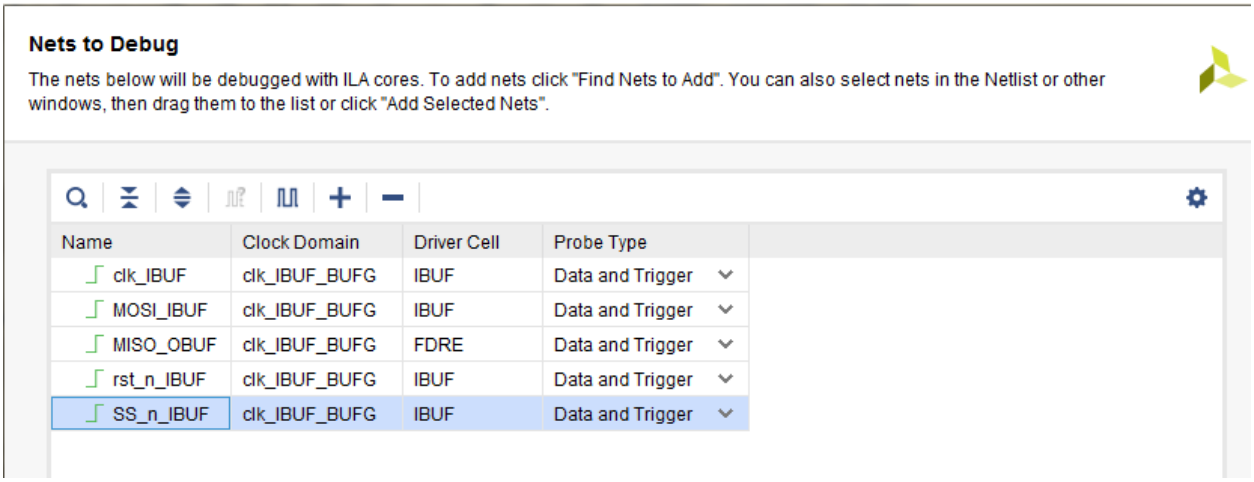
  - One-Hot encoding has largest setup time slack after implementation, so this encoding scheme achieves the highest frequency.
  - Sequential encoding has smallest setup time slack after implementation, so it is the encoding scheme that has the worst setup time slack (lowest frequency).
  - But all encoding schemes don't have any timing violations.

- **Note:**
  The Following Snippets are for **One-Hot Encoding.**

Set Up Debug

Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

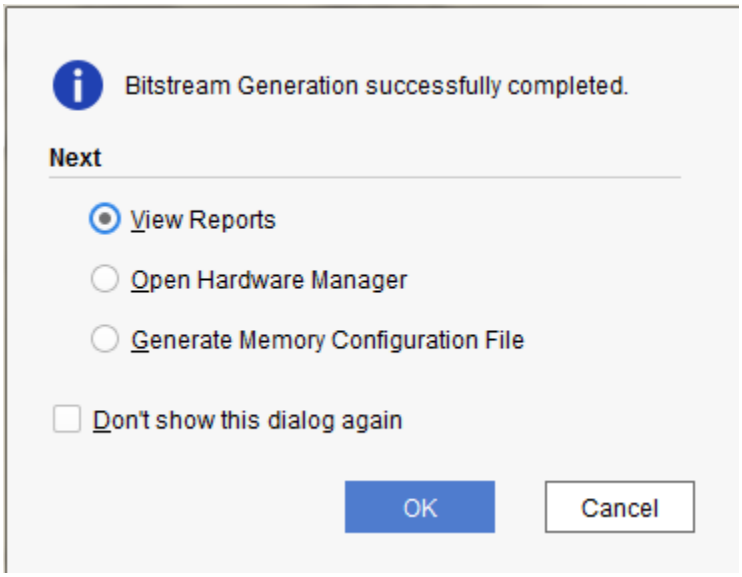


Name	Clock Domain	Driver Cell	Probe Type
clk_IBUF	clk_IBUF_BUFG	IBUF	Data and Trigger
MOSI_IBUF	clk_IBUF_BUFG	IBUF	Data and Trigger
MISO_OBUF	clk_IBUF_BUFG	FDRE	Data and Trigger
rst_n_IBUF	clk_IBUF_BUFG	IBUF	Data and Trigger
SS_n_IBUF	clk_IBUF_BUFG	IBUF	Data and Trigger

Netlist

```
// Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
// ----------------------------------------------------------------------------
// Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018
// Date         : Mon Aug  5 22:14:06 2024
// Host         : Ace-PC running 64-bit Service Pack 1  (build 7601)
// Command      : write_verilog {H:/Digital_Design_Diploma/Project 2/SPI_vivado/SPI_vivado.v}
// Design       : SPI_Wrapper
// Purpose      : This is a Verilog netlist of the current design or from a specific cell of the design. The output is an
//                IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input
//                design files.
// Device       : xc7a35ticpg236-1L
// ----------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module dbg_hub_CV
   (clk,
    sl_iport0_o,
    sl_oport0_i);
  input clk;
  output [0:36]sl_iport0_o;
  input [0:16]sl_oport0_i;


endmodule

module u_ila_0_CV
   (clk,
    probe0,
    SL_IPORT_I,
    SL_OPORT_O,
    probe1,
    probe2,
    probe3,
    probe4);
  input clk;
  input [0:0]probe0;
  input [0:36]SL_IPORT_I;
  output [0:16]SL_OPORT_O;
  input [0:0]probe1;
  input [0:0]probe2;
  input [0:0]probe3;
  input [0:0]probe4;


endmodule

module RAM
   (tx_valid,
    MISO_reg,
    MISO_reg_0,
    CLK,
    rst_n_IBUF,
```

Bitstream Generation

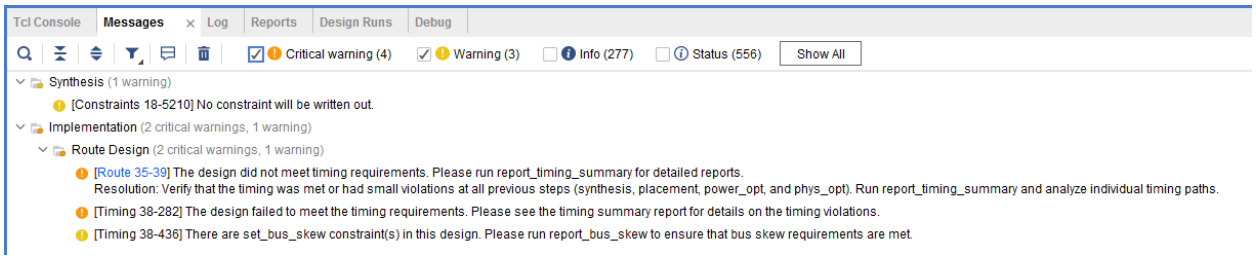


Constraints_basys3.xdc | SPI_Wrapper.bit

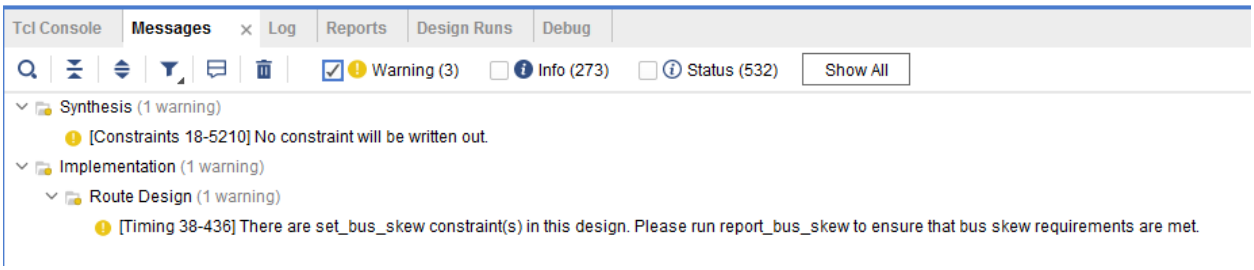
```
1  0009 0ff0 0ff0 0ff0 0ff0 0000 0161 003b
2  5350 495f 5772 6170 7065 723b 434f 4d50
3  5245 5353 3d54 5255 453b 5573 6572 4944
4  3d30 5846 4646 4646 4646 463b 5665 7273
5  696f 6e3d 3230 3138 2e32 0062 000d 3761
6  3335 7469 6370 6732 3336 0063 000b 3230
7  3234 2f30 382f 3035 0064 0009 3233 3a31
8  373a 3532 0065 0003 733c ffff ffff ffff
9  ffff ffff ffff ffff ffff ffff ffff ffff
10 ffff ffff ffff ffff ffff 0000 00bb 1122
11 0044 ffff ffff ffff ffff aa99 5566 2000
12 0000 3003 e001 0000 026b 3000 8001 0000
13 0012 2000 0000 3002 2001 0000 0000 3002
14 0001 0000 0000 3000 8001 0000 0000 2000
15 0000 3000 8001 0000 0007 2000 0000 2000
16 0000 3002 6001 0000 0000 3001 2001 0242
17 3fe5 3001 c001 0000 0000 3001 8001 0362
18 d093 3000 8001 0000 0009 2000 0000 3000
19 c001 0000 0401 3000 a001 0000 0501 3000
20 c001 0000 1000 3003 0001 0000 1000 2000
21 0000 2000 0000 2000 0000 2000 0000 2000
22 0000 2000 0000 2000 0000 2000 0000 3000
23 2001 0000 0000 3000 8001 0000 0001 2000
24 0000 3000 4065 0000 0000 0000 0000 0000
25 0000 0000 0000 0000 0000 0000 0000 0000
26 0000 0000 0000 0000 0000 0000 0000 0000
```

- **Note:**
  - **After adding the debug core** (before adding it there wasn't any critical warnings) using the following connections, the implementation was having critical warnings (timing violations).

Create a constraint file where the rst_n , SS_n & MOSI are connected to 3 switches, and the MISO to a led.



  - After connecting **"rst_n"** to a button, the problem was solved.

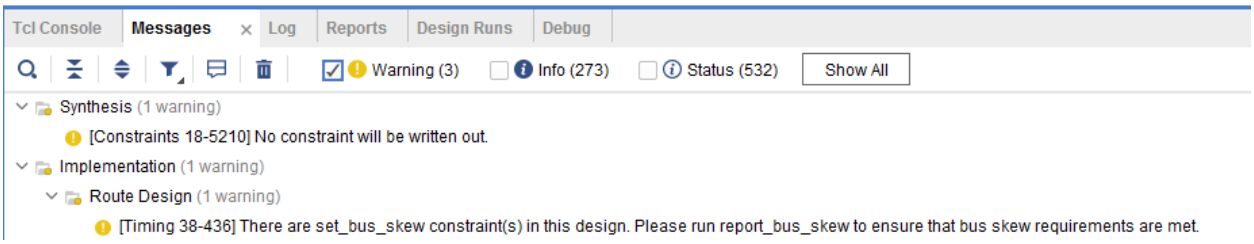


  - **But honestly, I am not sure if the push-buttons can be used as an active low input or not ?!!**

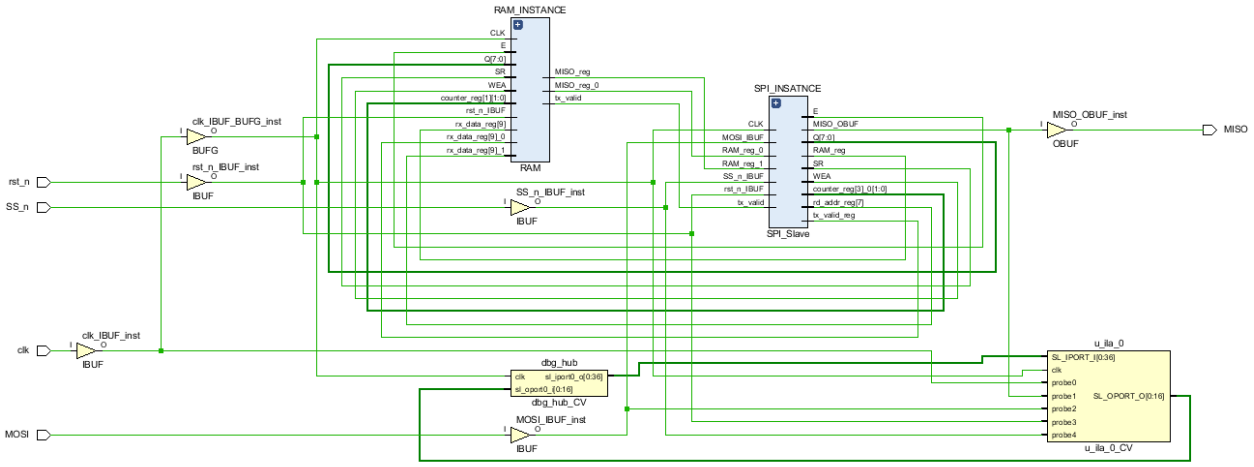
- **Note:**
  The following results are after connecting **“rst_n”** to a button.

Implementation (After Adding Debug Core)

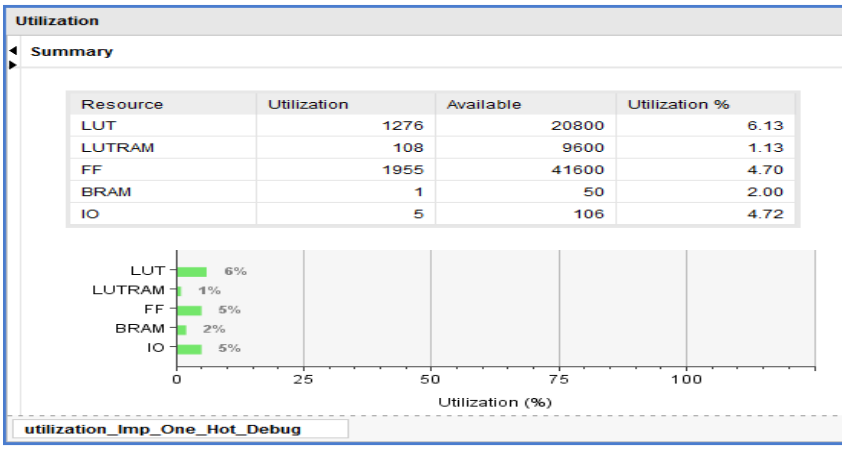
- Messages tab after running implementation.
- As shown in the snippet → **No Critical Warnings or Errors**.



- Schematic after running synthesis (with debug core).



- Utilization after implementation (with debug core).



- FPGA Device (with debug core).