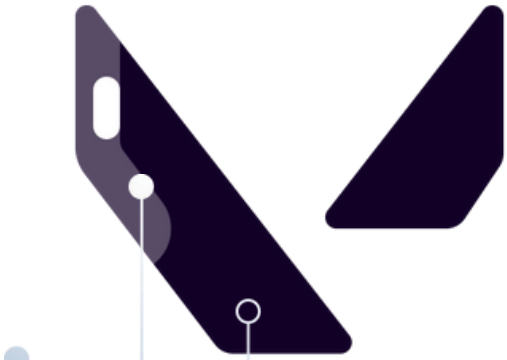
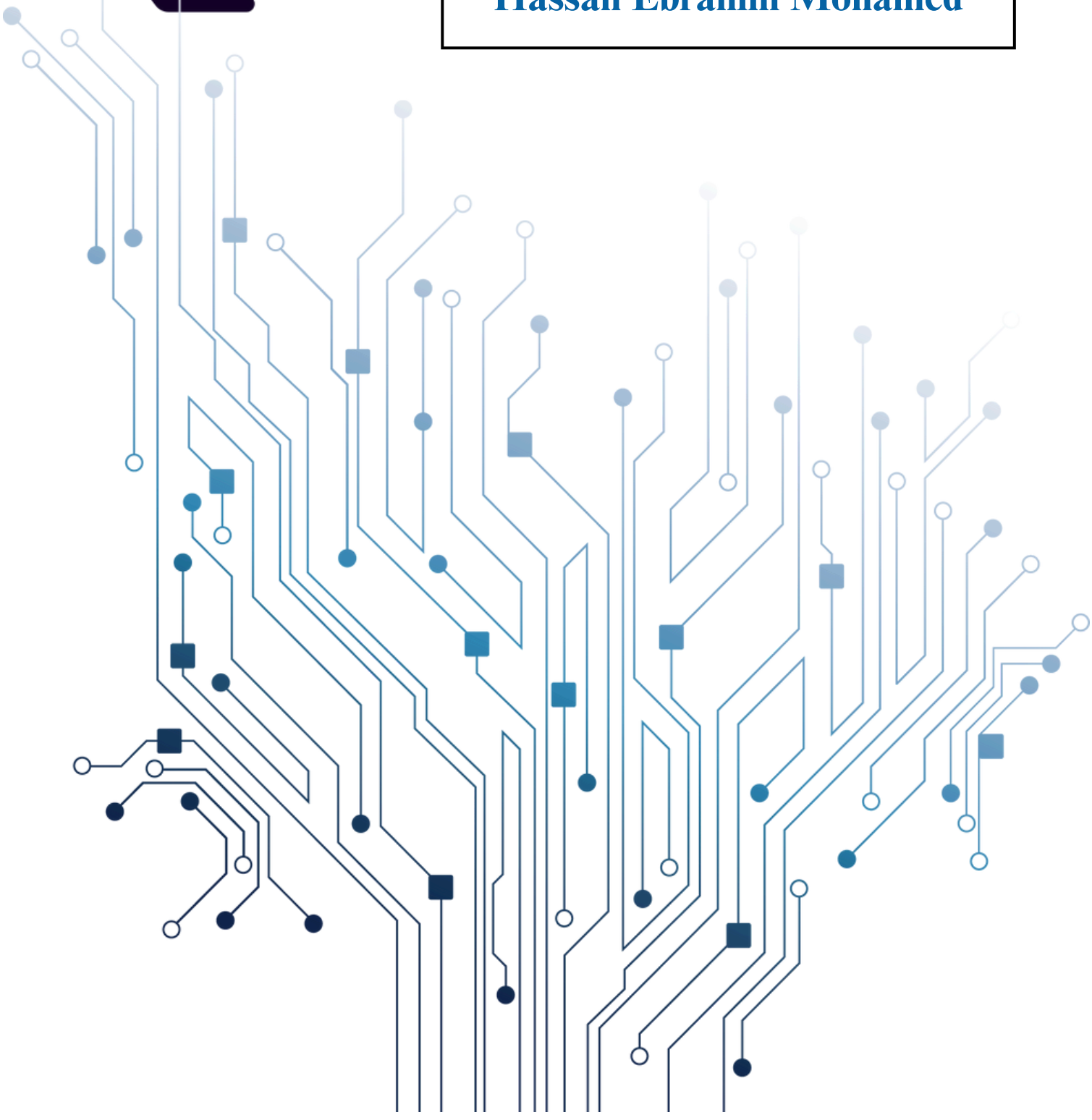


SPI Slave with Single Port RAM

Hassan Ebrahim Mohamed



Design

```
module Wrapper (MOSI , MISO , clk , rst_n , SS_n);


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

wire tx_valid  ;
wire [7:0] tx_data ;

wire rx_valid ;
wire [9:0] rx_data ;

wire [9 : 0] din ;

wire [7:0] dout ;


assign din = rx_data;
assign tx_data = dout ;

RAM RAM_inst (din , rx_valid , clk , rst_n , dout , tx_valid);
SPI_Slave SPI_Slave_inst (MOSI , tx_valid , clk , rst_n,SS_n,tx_data,rx_valid,rx_data , MISO);

endmodule
```

Instantiated Modules

1 → Slave

```
module SPI_Slave (MOSI , tx_valid , clk , rst_n,SS_n,tx_data,rx_valid,rx_data , MISO);
parameter IDLE = 3'b000 ;
parameter CHK_CMD = 3'b001 ;
parameter WRITE = 3'b010 ;
parameter READ_DATA = 3'b011;
parameter READ_ADD = 3'b100 ;
input MOSI , tx_valid , clk , rst_n ;
input SS_n ;
input [7:0] tx_data ;
output reg MISO , rx_valid ;
output reg [9:0] rx_data ;

reg [9:0] data_out ; // from slave to RAM
reg [7:0] data_in ; // from RAM to slave
reg [4:0] counter ;
reg Read_en ;

reg[2:0] cs , ns ;


// State Memory

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

// Next State Logic

always @(*) begin
    case (cs)
        IDLE :
            if (!SS_n)
                ns = CHK_CMD ;
            else
                ns = IDLE ;

        CHK_CMD :
            if (SS_n == 0 && MOSI == 0)
                ns = WRITE ;
            else if (SS_n == 0 && MOSI == 1)begin
                if (!Read_en)
                    ns = READ_ADD;
                else
                    ns = READ_DATA;
                // Read_en = 1 ;
            end else
                ns = IDLE ;

        WRITE :
            if (SS_n)
                ns = IDLE ;
            else
                ns = WRITE ;

        READ_ADD :
            if (SS_n)
                ns = IDLE ;
            else
                ns = READ_ADD ;

        READ_DATA :
            if (SS_n)
                ns = IDLE ;
            else
                ns = READ_DATA ;

        default : ns = IDLE ;
```

```
    endcase // cs
end



// Output Logic
always @(posedge clk ) begin
    if (!rst_n) begin
        MISO        <= 0 ;
        data_out    <= 0 ;
        data_in     <= 0 ;
        rx_valid    <= 0 ;
        rx_data     <= 0 ;
        counter     <= 0 ;
        Read_en     <= 0 ;
    end
    else begin
        case (cs)
            WRITE :
                    begin
                        if (counter < 10)begin
                            data_out <= {data_out , MOSI};
                        end else begin
                            rx_data <= data_out ;
                            if(!data_out[9])begin
                                rx_valid    <= 1 ;
                            end else begin
                                rx_valid    <= 0 ;
                            end
                        end
                        counter <= counter + 1 ;
                     end
            READ_ADD :
                    begin
                        if (counter < 10)begin
                            data_out <= {data_out , MOSI};
                        end else begin
                            rx_data <= data_out ;
                            if(data_out[9:8]==2'b10)begin
                                rx_valid    <= 1 ;
                                Read_en     <= 1 ;
                            end else begin
                                rx_valid    <= 0 ;
                            end
                        end
                        counter <= counter + 1 ;
                     end
            READ_DATA : /* SPI call RAM --- SPI and RAM are Active --- SPI receive Data and serialized it */
                    begin
                        if (counter < 10)begin
                            data_out <= {data_out , MOSI};
                        end else if (counter == 10) begin
                            rx_data <= data_out ;
                            if(data_out[9:8]==2'b11)begin
                                rx_valid    <= 1 ;
                                Read_en     <= 0 ;
                            end else begin
                                rx_valid    <= 0 ;
                            end
                        end else if (counter == 11 && !tx_valid)begin
                            rx_valid    <= 0 ;
                        end else if (counter == 12 && tx_valid)begin
                            data_in     <= tx_data  ;
                        end else if (counter <= 20 && counter > 12 && !tx_valid)begin
                            MISO    <= data_in[7];
                            data_in <= {data_in , 1'b0};
                        end else begin
                            MISO <= 0 ;
                        end
                        counter <= counter + 1 ;
                     end
            default :
                    begin
```

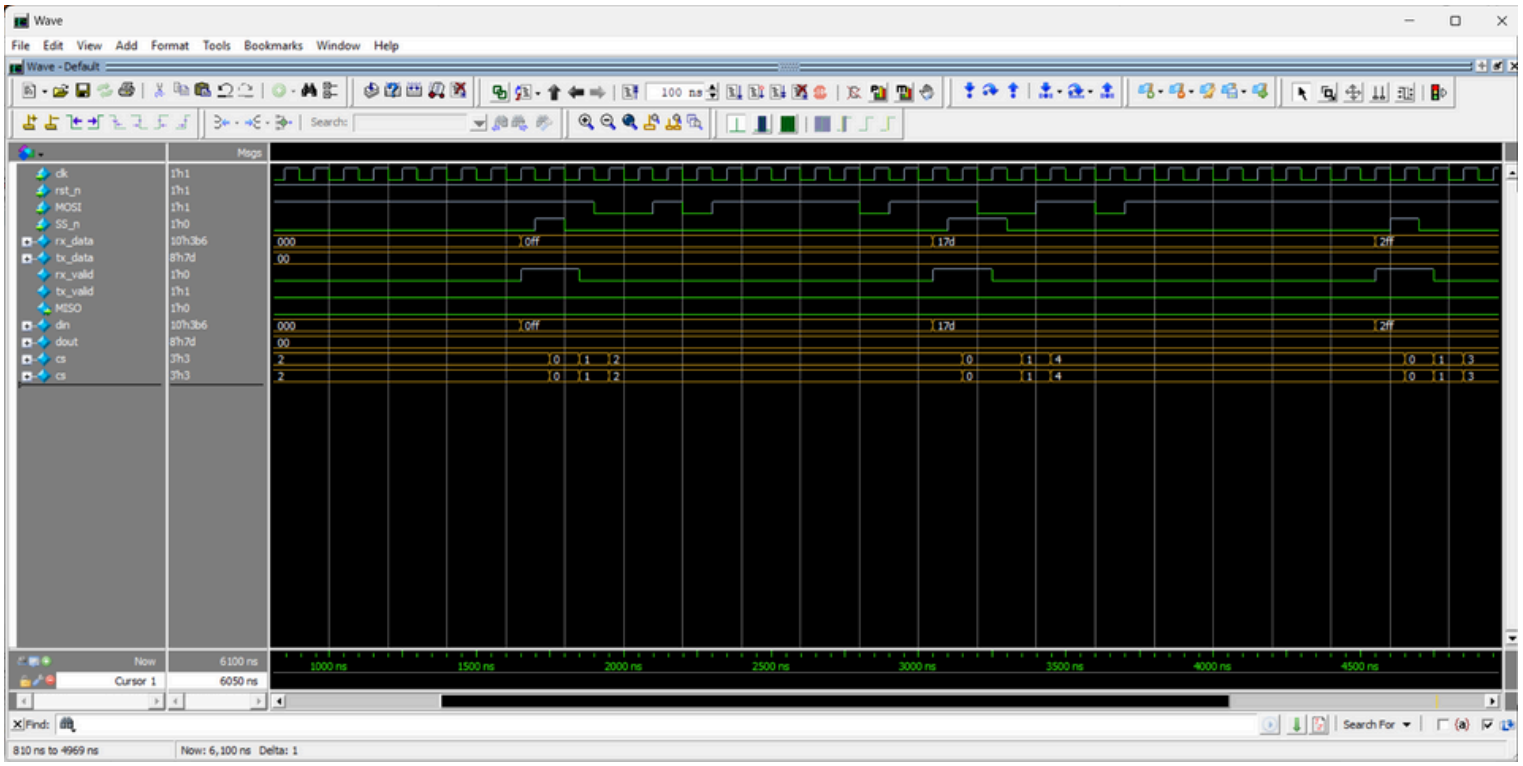
```
                        MISO        <= 0 ;
                        data_out    <= 0 ;
                        data_in     <= 0 ;
                        rx_valid    <= 0 ;
                        counter     <= 0 ;
                    end
        endcase
    end
end


endmodule
```

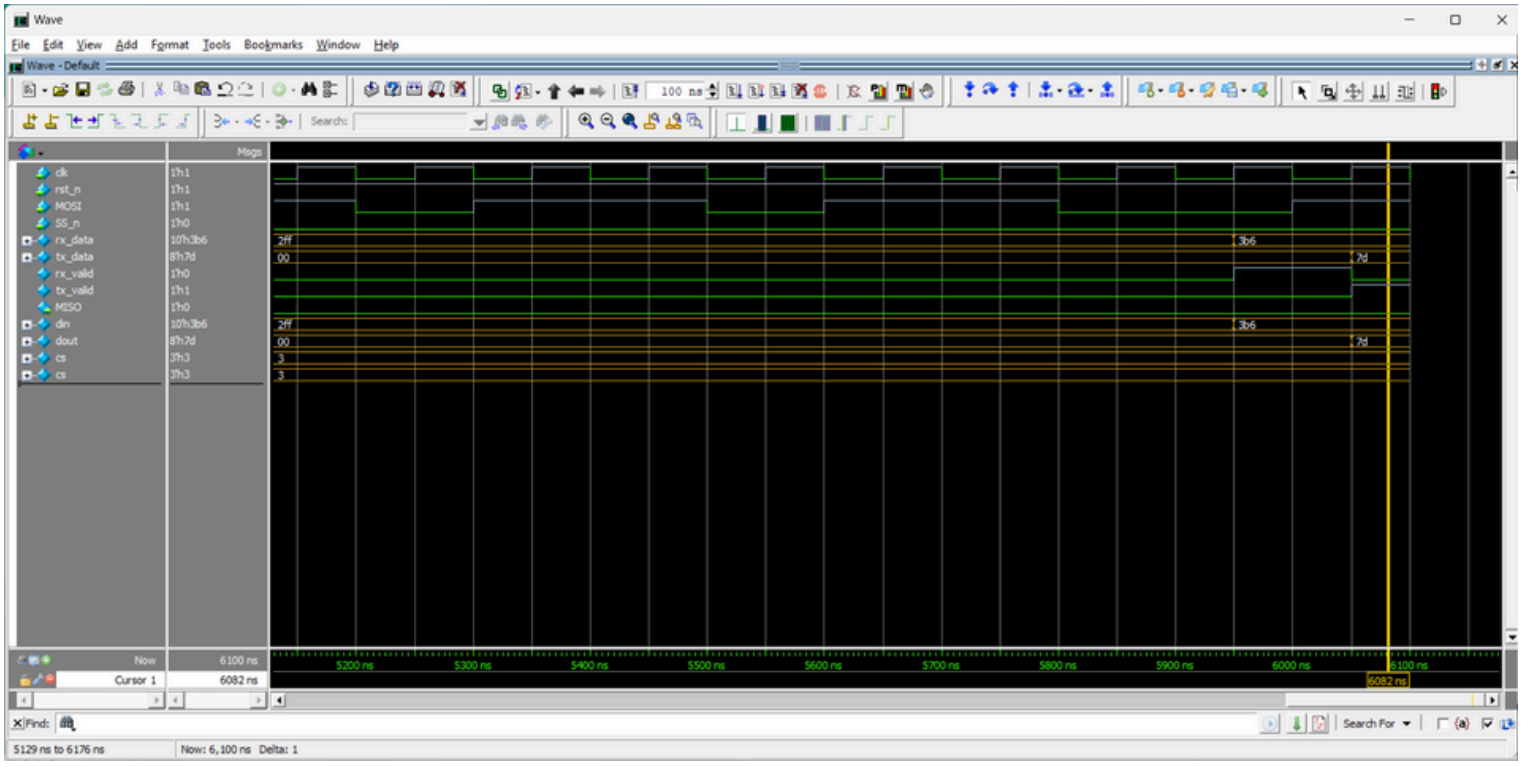
Instantiated Modules

2 → RAM

```
module RAM (din , rx_valid , clk , rst_n , dout , tx_valid);
parameter MEM_DEPTH = 256 ;
parameter ADDR_SIZE = $clog2(MEM_DEPTH) ;   // log2(MEM_DEPTH)

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

reg [9 : 0] mem [MEM_DEPTH - 1 :0];

wire [1:0] sel ;
reg [ADDR_SIZE - 1 : 0] addr_wr , addr_rd ;


assign sel = din [9 : 8] ;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0 ;
        tx_valid <= 0 ;
    end
    else begin
        if (rx_valid) begin
            if (sel  == 2'b00) begin
                addr_wr <= din[7 : 0];
                tx_valid <= 0 ;
            end
            else if (sel  == 2'b01)begin
                    mem[addr_wr] <= din[7 : 0];
                    tx_valid <= 0 ;
            end
            else if (sel  == 2'b10)begin
                    addr_rd <= din[7 : 0];
                    tx_valid <= 0 ;
            end
            else if (sel  == 2'b11)begin
                    dout <= mem [addr_rd];
                    tx_valid <= 1 ;
            end
        end else begin
            tx_valid <= 0 ;
        end

    end
end


endmodule
```

Questa Sim


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
clk
rst_n
MOSI
SS_n
rx_data
tx_data
rx_valid
tx_valid
MISO
din
dout
cs
cs
Now
Cursor 1
1000 ns
1500 ns
2000 ns
2500 ns
3000 ns
3500 ns
4000 ns
4500 ns
Find:
Search For
810 ns to 4969 ns
Now: 6,100 ns Delta: 1

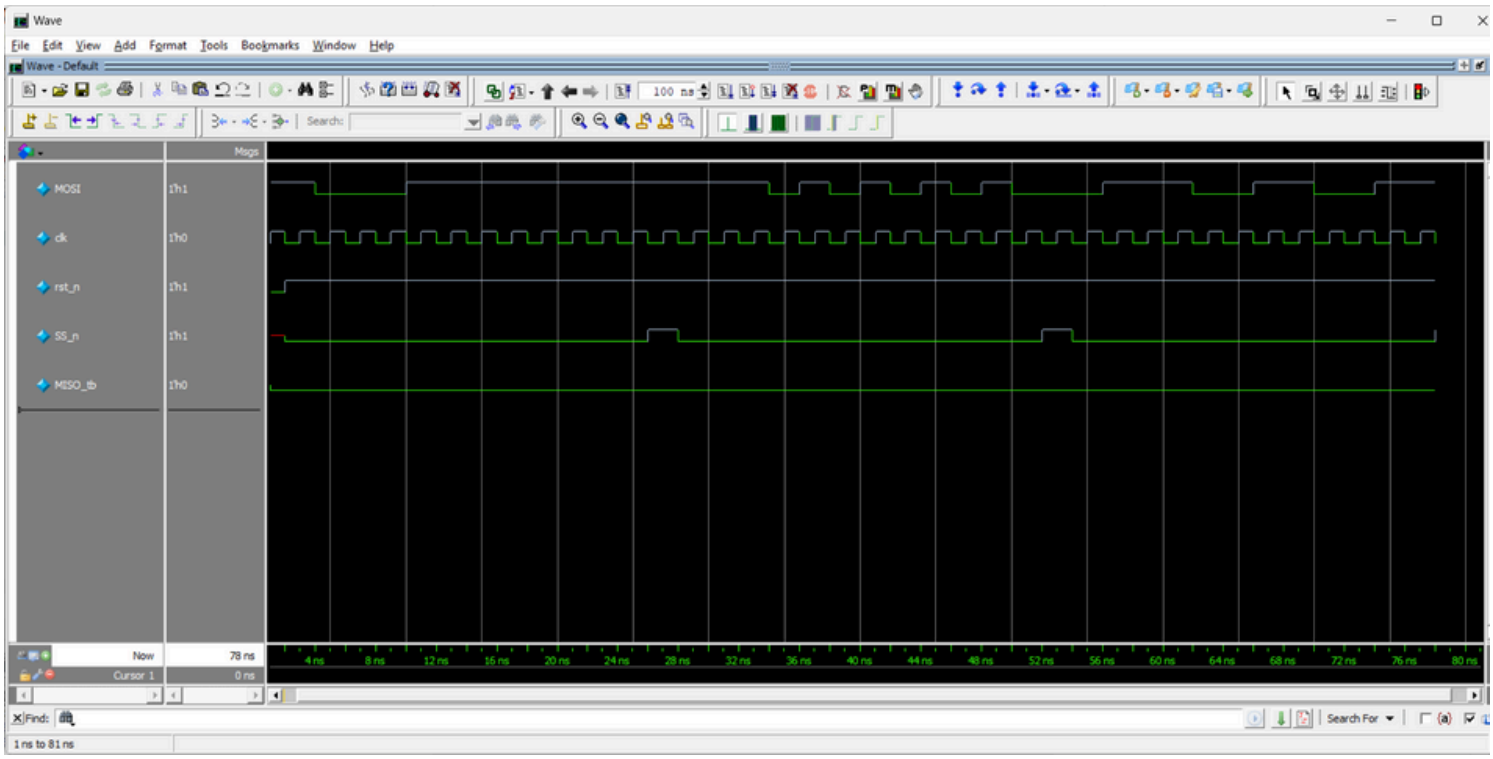

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
clk
rst_n
MOSI
SS_n
rx_data
tx_data
rx_valid
tx_valid
MISO
din
dout
cs
cs
Now
Cursor 1
5200 ns
5300 ns
5400 ns
5500 ns
5600 ns
5700 ns
5800 ns
5900 ns
6000 ns
Find:
Search For
5129 ns to 6176 ns
Now: 6,100 ns Delta: 1

Testbench

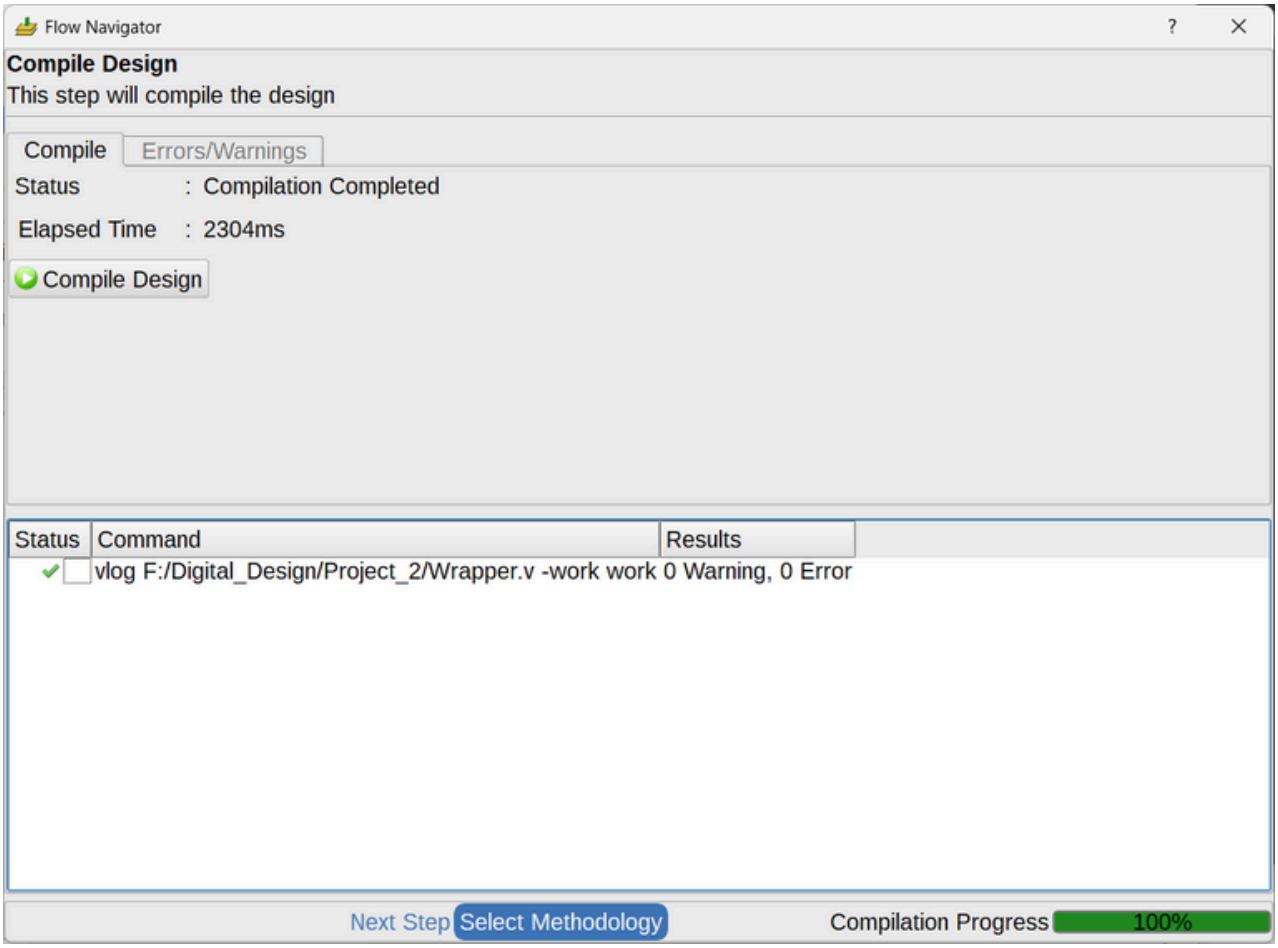
```
module Wrapper_tb ();
reg MOSI ,  clk , rst_n , SS_n  ;
wire  MISO_tb ;

Wrapper DUE (MOSI , MISO_tb , clk , rst_n , SS_n);

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

initial begin
    rst_n = 0 ;
    MOSI = 1 ;
    @(negedge clk)
    rst_n = 1 ;
    SS_n = 0 ;      // go to CHK_CMD
    @(negedge clk)
    MOSI = 0 ;      // go to WRITE
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 0 ;      // the first 2 bits = 00

    // write in the address : 1111_1111
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;          // end

    @(negedge clk)
    SS_n = 1 ;          // back to IDEAL

    @(negedge clk)
    SS_n = 0 ;          // go to CHK_CMD

    @(negedge clk)
    MOSI =  1 ;         // Read_add
    @(negedge clk)
    MOSI =  1 ;
    @(negedge clk)
```

```
    MOSI =  0 ;          // the first 2 bits = 10
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 0 ;           // end
    // write in the address : 1010_1010

    @(negedge clk)
    SS_n = 1 ;           // back to IDEAL
    @(negedge clk)
    SS_n = 0 ;           // go to CHK_CMD

    @(negedge clk)
    MOSI =  1 ;          // Read_data
    @(negedge clk)
    MOSI =  1 ;
    @(negedge clk)
    MOSI =  1 ;          // the first 2 bits = 11

    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 0 ;
    @(negedge clk)
    MOSI = 1 ;
    @(negedge clk)
    MOSI = 1 ;           // end
    // write in the address : 0011_0011
    @(negedge clk)
    SS_n = 1 ;           // back to IDEAL
$stop;
end

endmodule
```

Questa Sim


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
MOSI
clk
rst_n
SS_n
MISO_tb
Now
Cursor 1
4 ns 8 ns 12 ns 16 ns 20 ns 24 ns 28 ns 32 ns 36 ns 40 ns 44 ns 48 ns 52 ns 56 ns 60 ns 64 ns 68 ns 72 ns 76 ns 80 ns
Find:
Search For
1 ns to 81 ns

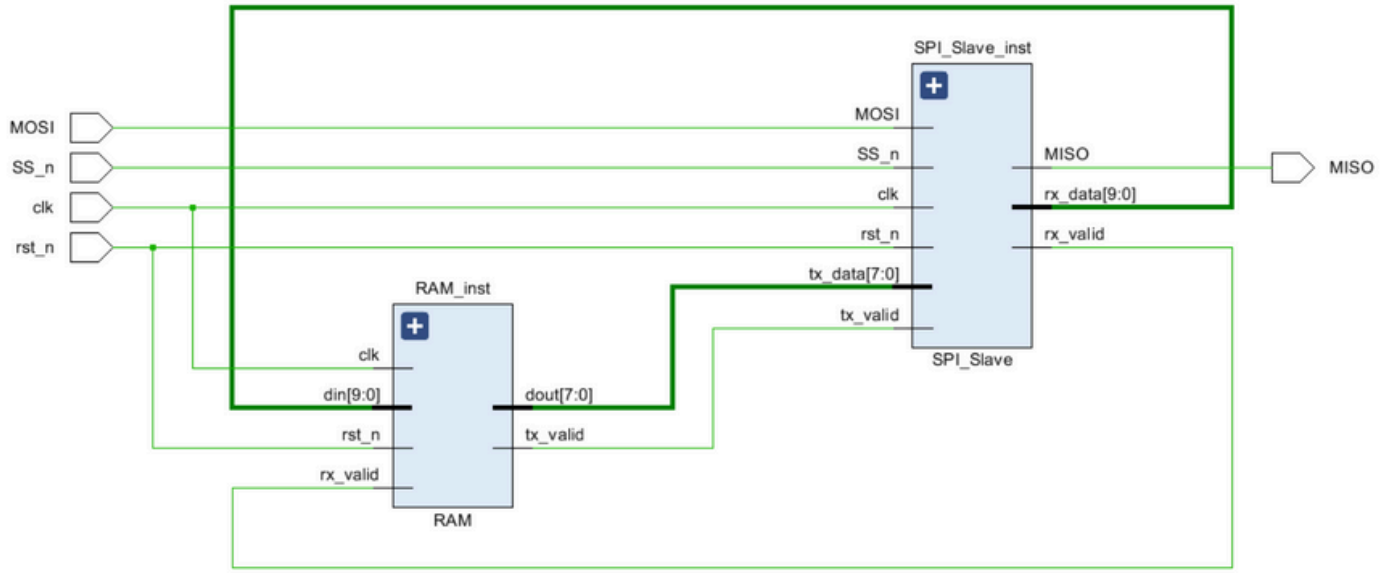
QuestaLint


Flow Navigator
Compile Design
This step will compile the design
Compile
Errors/Warnings
Status : Compilation Completed
Elapsed Time : 2304ms
Compile Design
Status
Command
Results
vlog F:/Digital_Design/Project_2/Wrapper.v -work work 0 Warning, 0 Error
Next Step
Select Methodology
Compilation Progress
100%

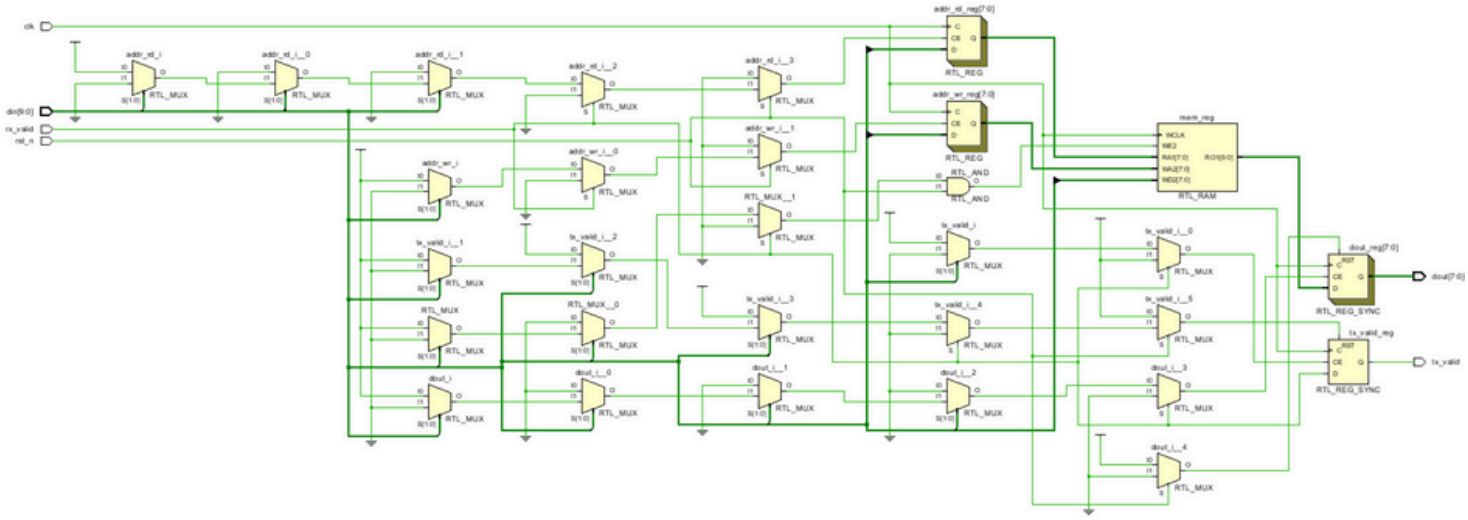
Vivado

Elaboration:

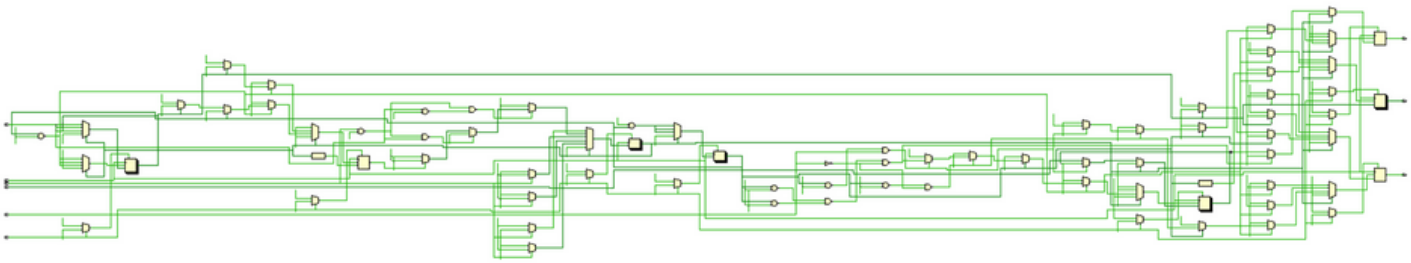
→Wrapper



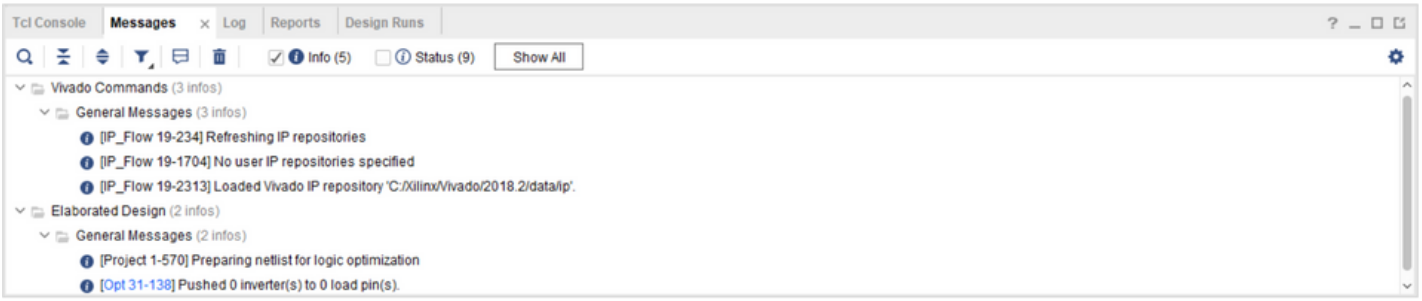
→RAM



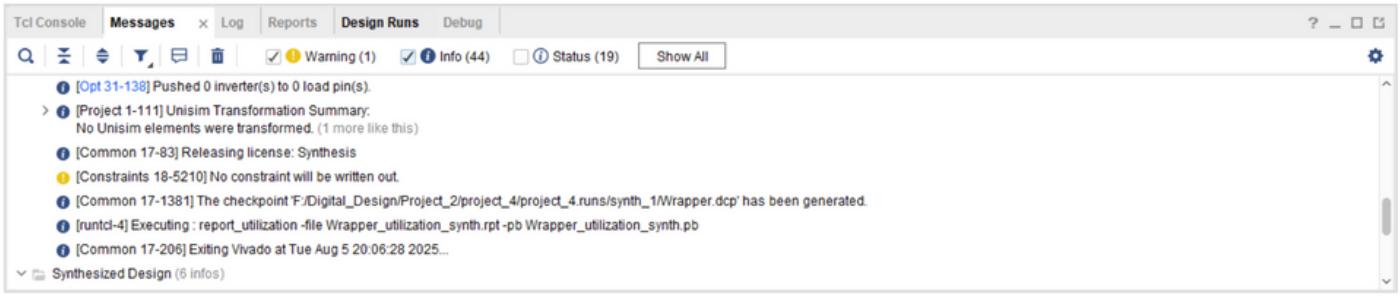
→SPI



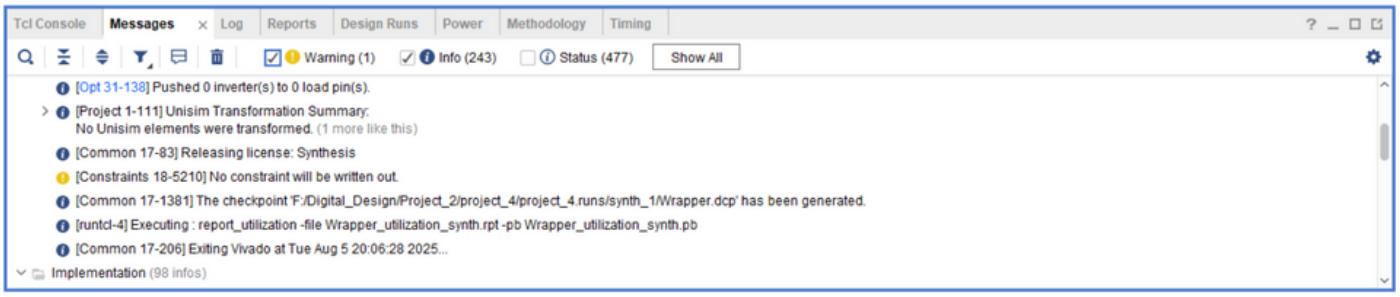
Elaboration:



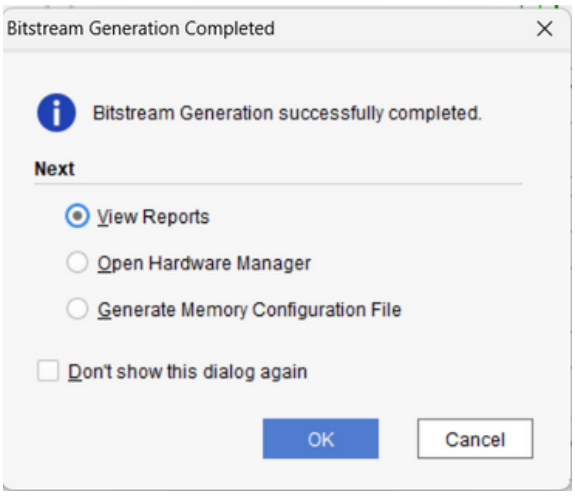
Synthesis:



Implementation:



Bitstream generation:

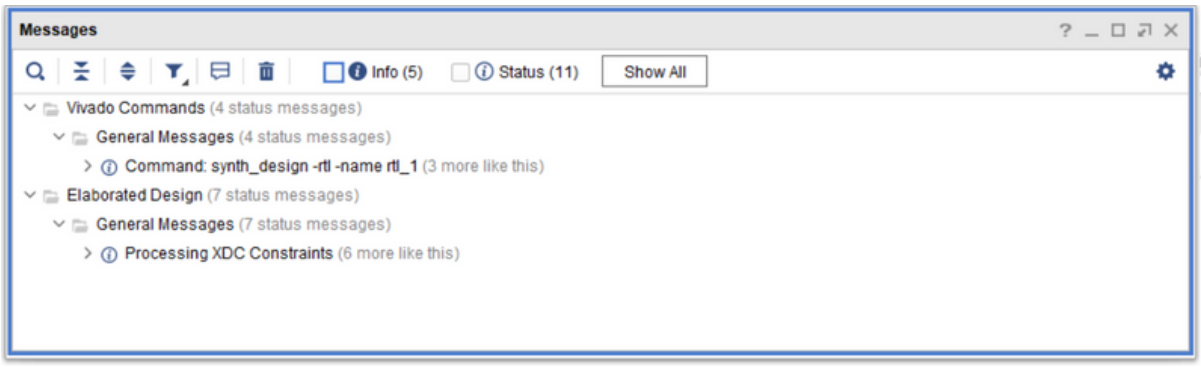


Synthesis snippets for each encoding

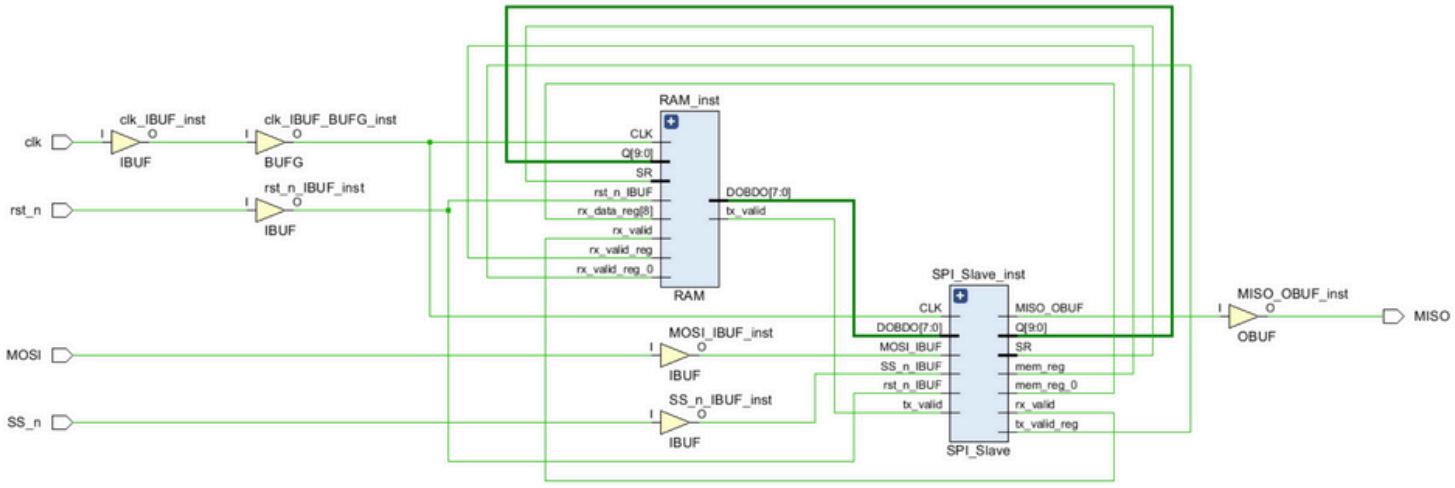
Seq:

Elaboration:

→Message



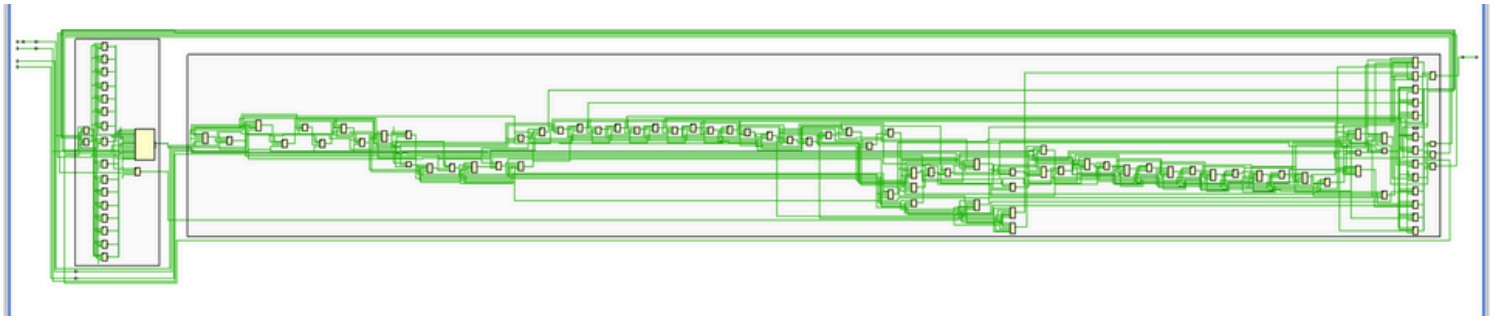
→Schematic



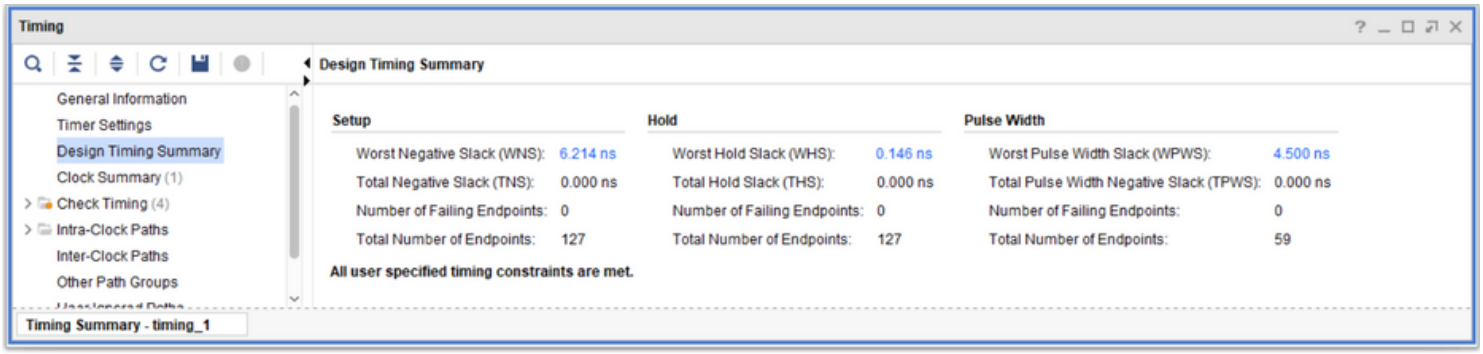

```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "sequential" *) [F:/Digital_Design/Project_2/Slave.v:20]
INFO: [Synth 8-6155] done synthesizing module 'SPI_Slave' (2#1) [F:/Digital_Design/Project_2/Slave.v:1]
INFO: [Synth 8-6155] done synthesizing module 'Wrapper' (3#1) [F:/Digital_Design/Project_2/Wrapper.v:1]
```

Synthesis:

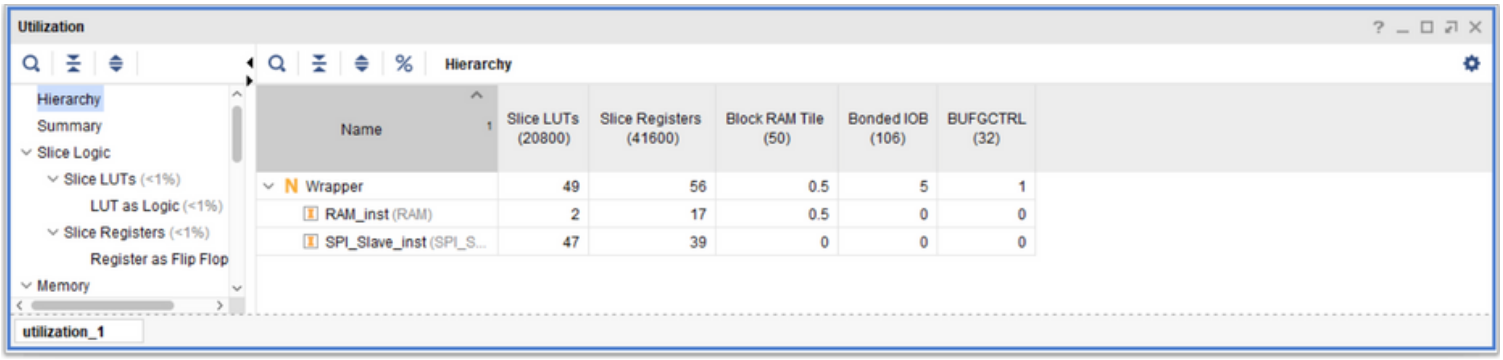
→Schematic



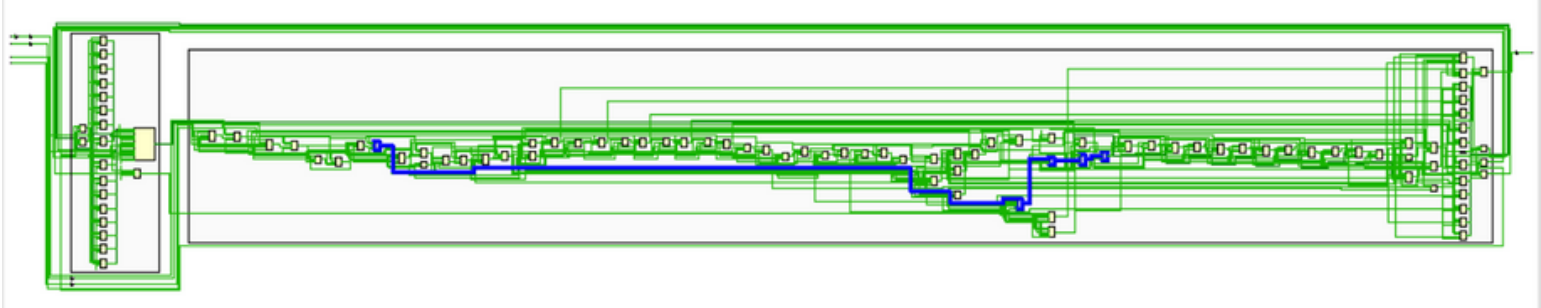
→Timing report



→Utilization report

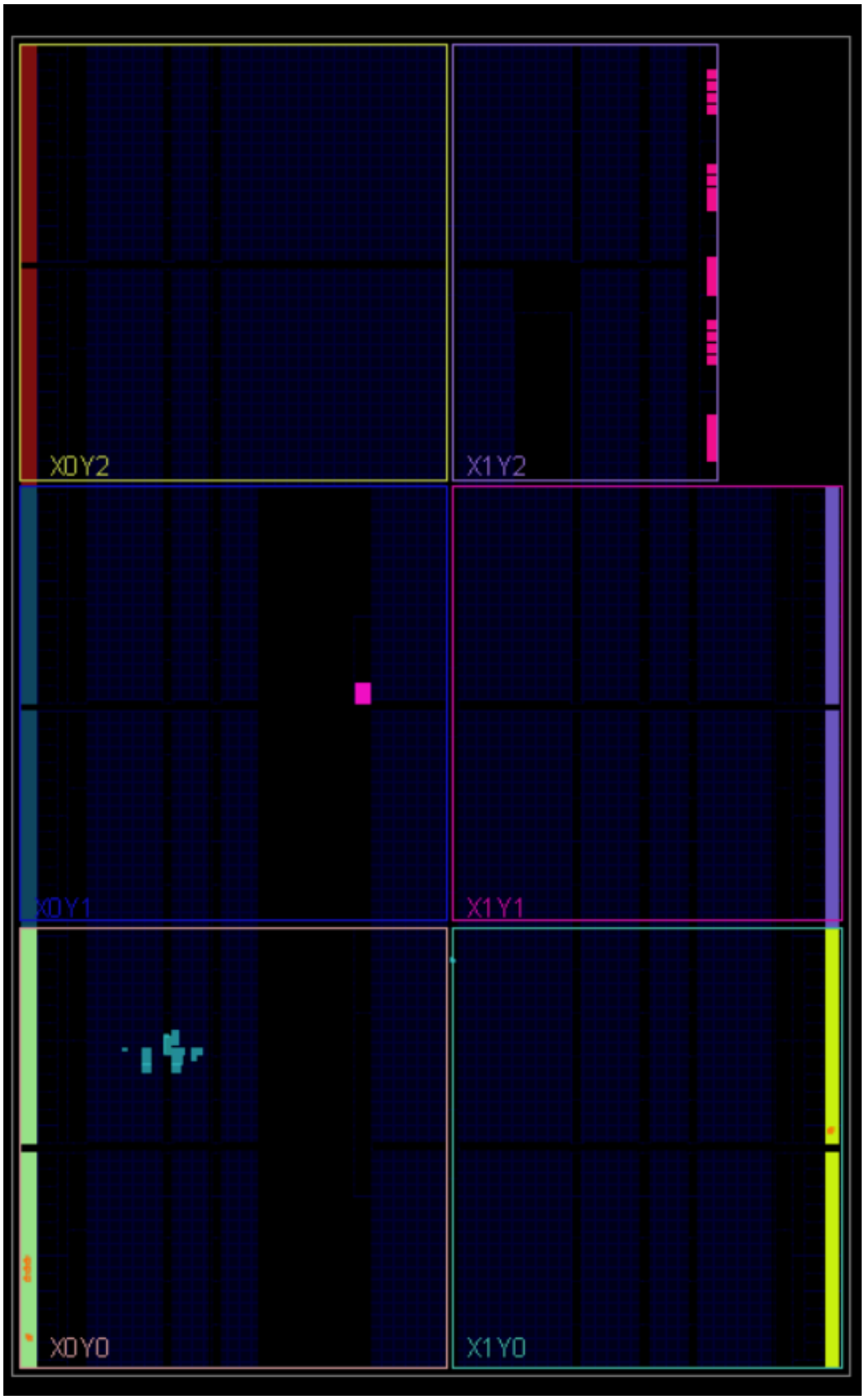


→ The critical path



Implementation:

→Device



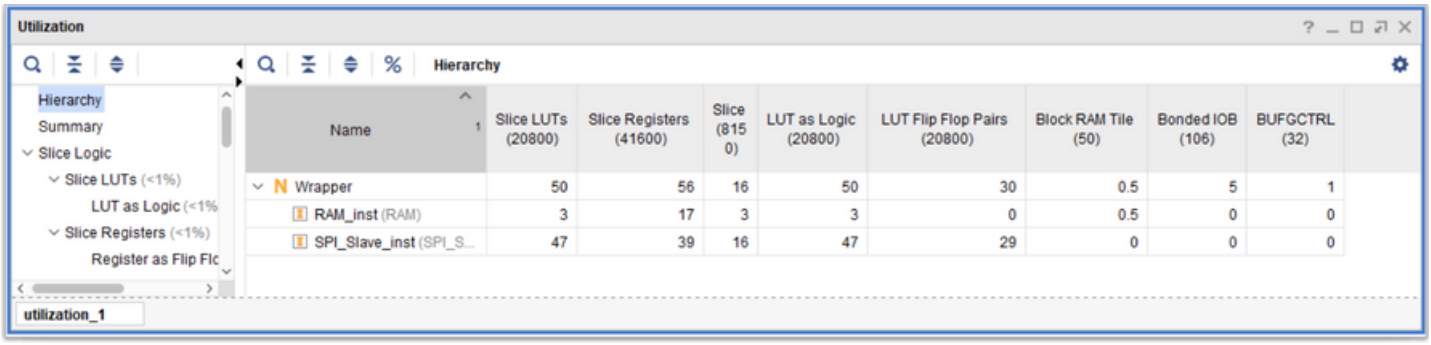
→Timing report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.335 ns	Worst Hold Slack (WHS):	0.044 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:	127	Total Number of Endpoints:	127	Total Number of Endpoints:	59

All user specified timing constraints are met.

→Utilization report



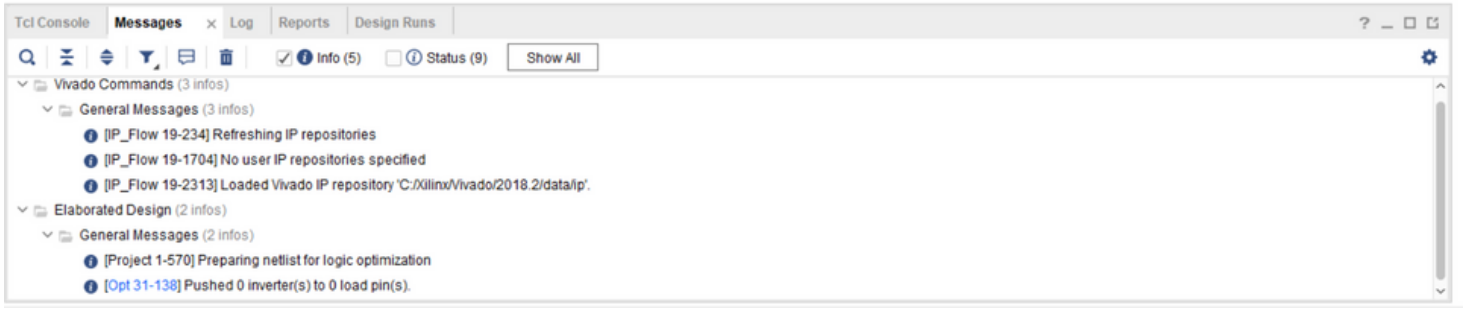
State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	001	001
WRITE	010	010
READ_ADD	011	100
READ_DATA	100	011

Synthesis snippets for each encoding

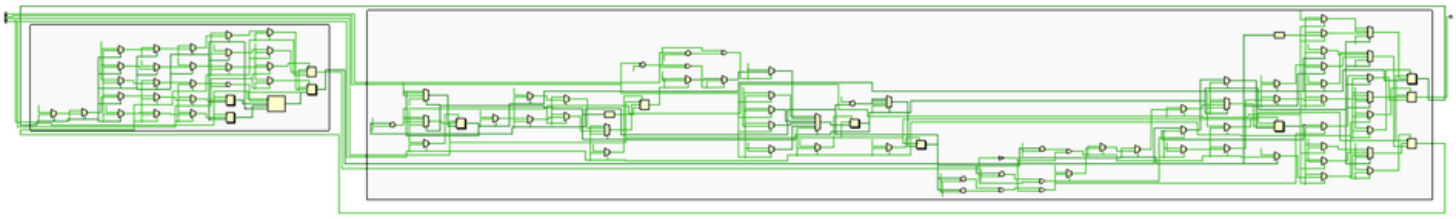
Gray:

Elaboration:

→Message



→Schematic

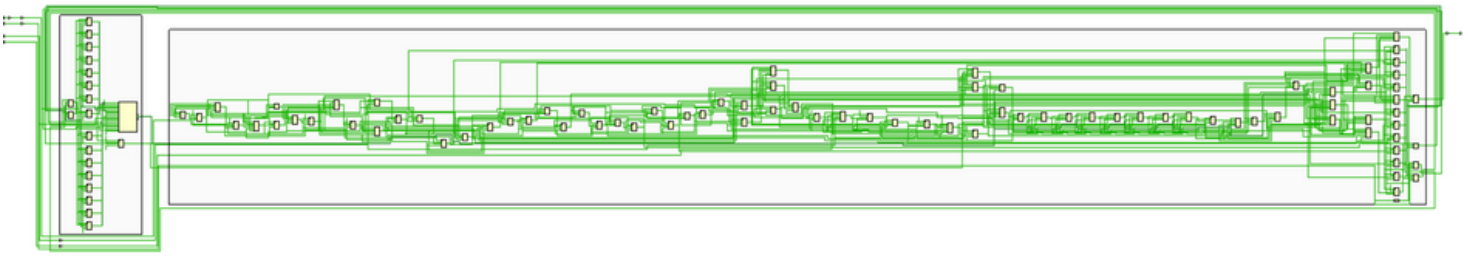


```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *) [F:/Digital_Design/Project_2/gray.v:20]
INFO: [Synth 8-6155] done synthesizing module 'SPI_Slave' (2#1) [F:/Digital_Design/Project_2/gray.v:1]
INFO: [Synth 8-6155] done synthesizing module 'Wrapper' (3#1) [F:/Digital_Design/Project_2/Wrapper.v:1]
```

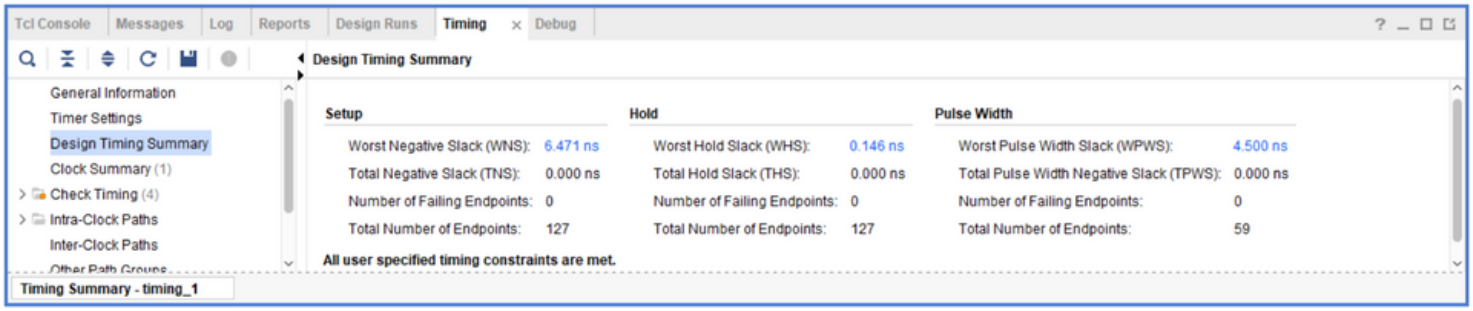
State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	001	001
WRITE	011	010
READ_ADD	010	100
READ_DATA	111	011

Synthesis:

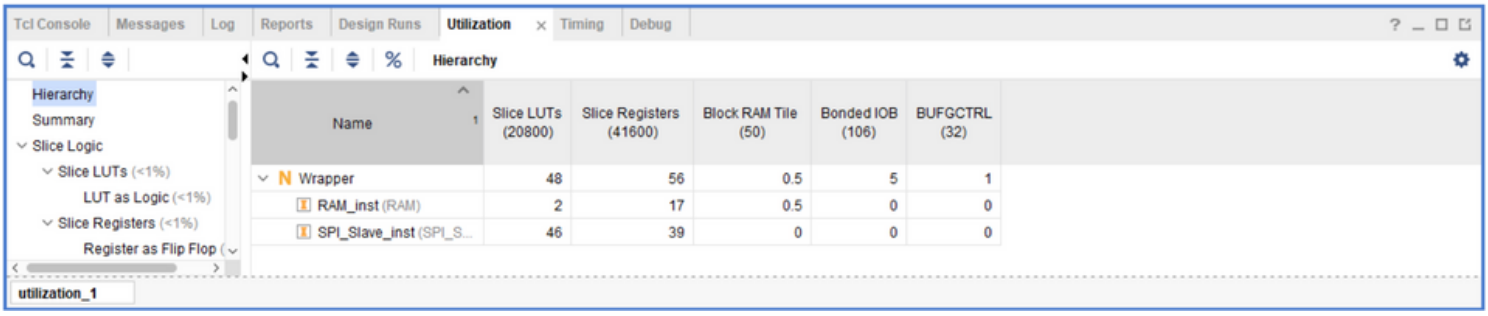
→Schematic



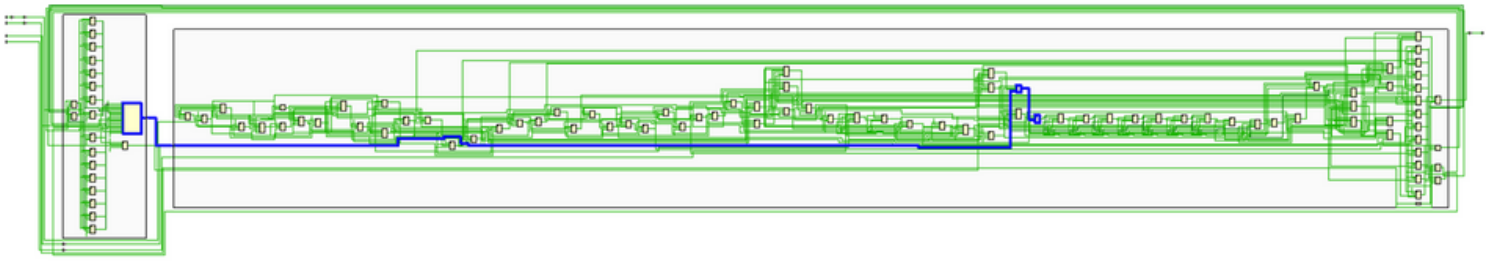
→Timing report



→Utilization report

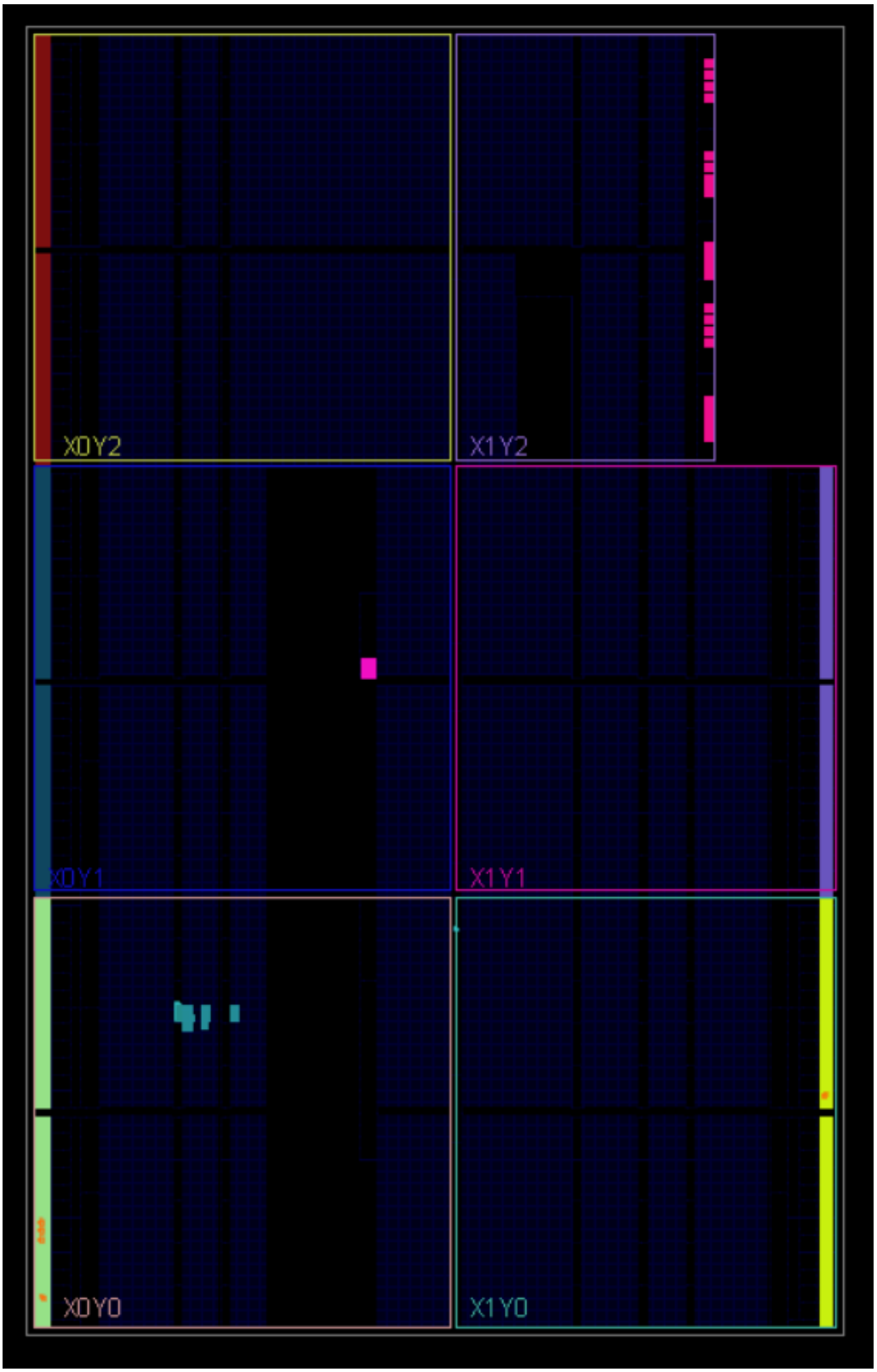


→ The critical path

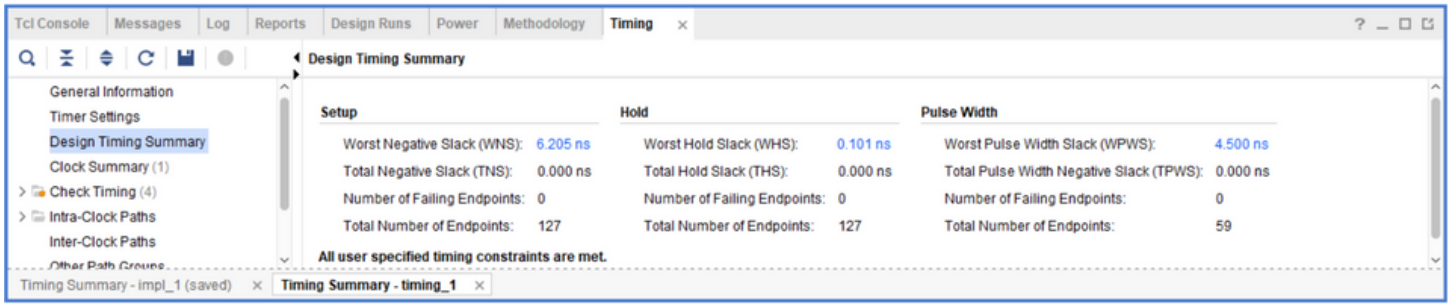


Implementation:

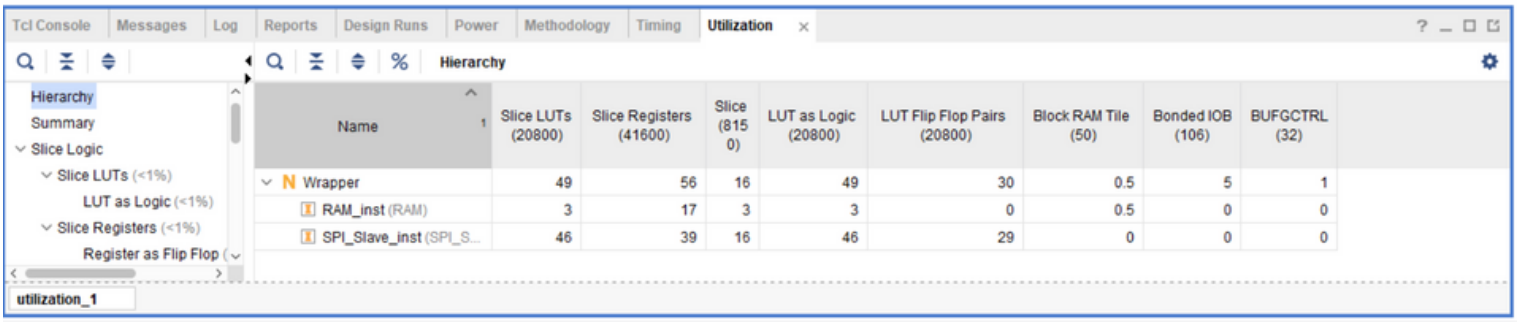
→Device



→Timing report



→Utilization report

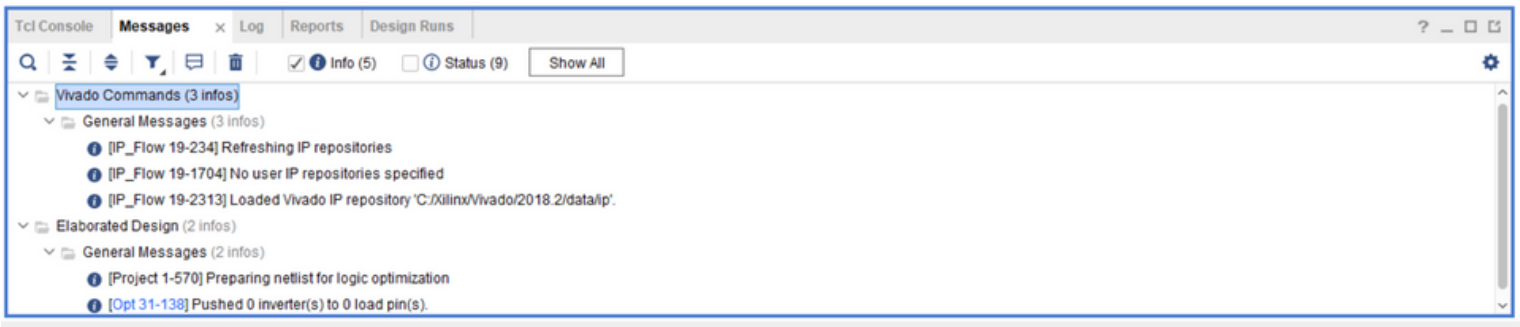


Synthesis snippets for each encoding

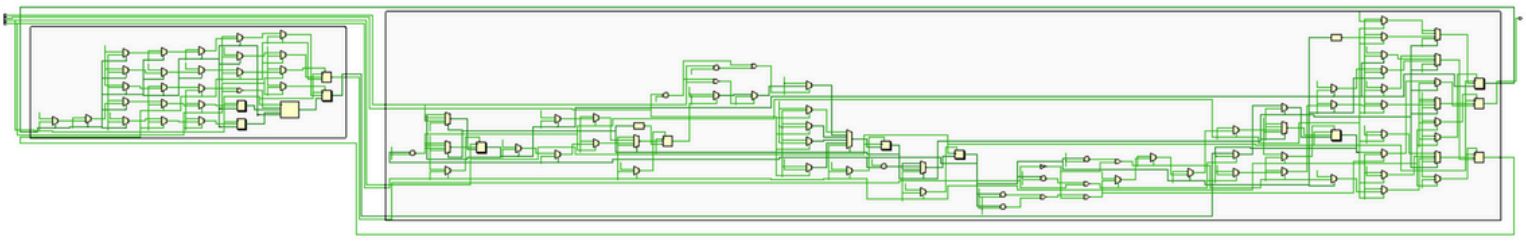
One hot:

Elaboration:

→Message



→Schematic



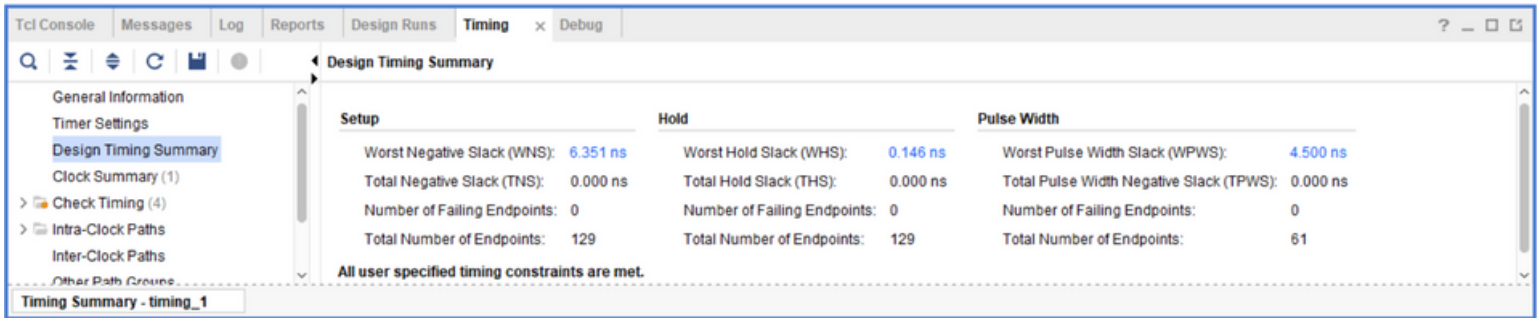
```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "one_hot" *) [F:/Digital_Design/Project_2/one_hot.v:20]
INFO: [Synth 8-6155] done synthesizing module 'SPI_Slave' (2#1) [F:/Digital_Design/Project_2/one_hot.v:1]
INFO: [Synth 8-6155] done synthesizing module 'Wrapper' (3#1) [F:/Digital_Design/Project_2/Wrapper.v:1]
```

State	New Encoding	Previous Encoding
IDLE	00001	000
CHK_CMD	00010	001
WRITE	00100	010
READ_ADD	01000	100
READ_DATA	10000	011

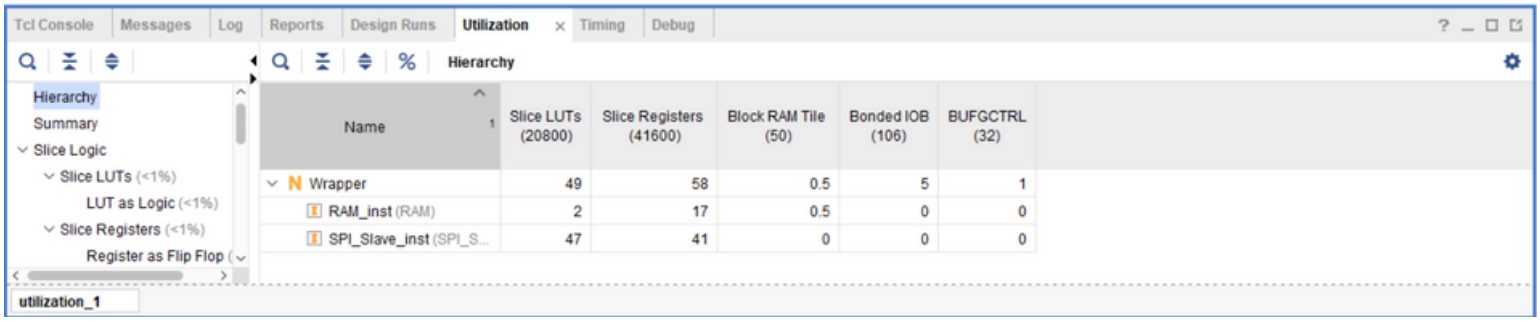
Synthesis:

→Schematic

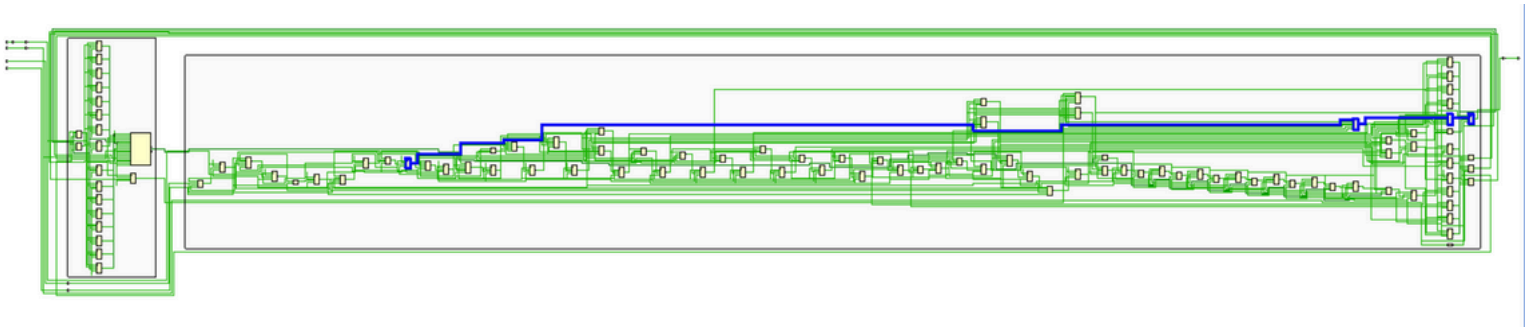
→Timing report



→Utilization report

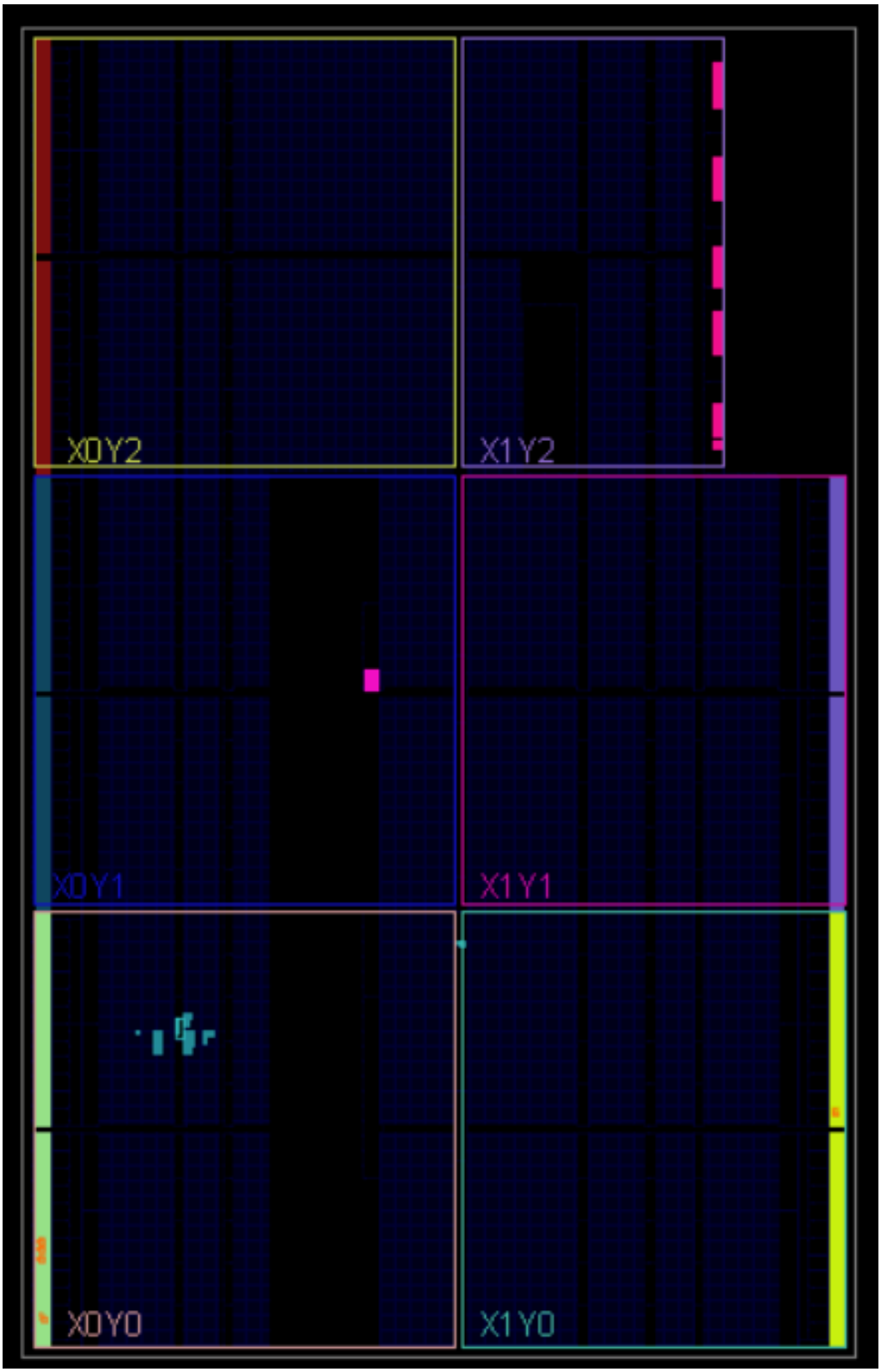


→ The critical path

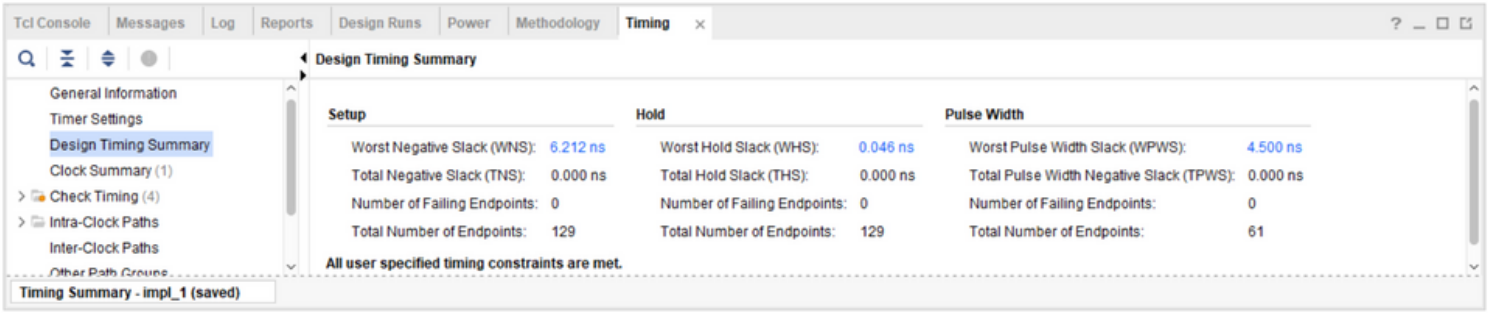


Implementation:

→Device



→Timing report



→Utilization report