SPI_PROJECT
(SPI SLAVE WITH SINGLE PORT RAM)

Submitted By:

- Mahmoud Badry Mahmoud Ahmed
- Ahmed Khaled Marzouk
- Ismail Galal Mohamed

Submitted To:

Eng. Kareem Wessam

Contents

1. Design Flow

a. RAM Code

```
module ram (din,dout,rx,tx,clk,rst_n);
        parameter MEM_DEPTH = 256;
        parameter ADDER_SIZE =8;
        input [ADDER_SIZE+1:0] din;
        output reg [ADDER_SIZE-1:0] dout;
        input rx,clk,rst_n;
        output  reg tx;
        reg [ADDER_SIZE-1:0] mem [MEM_DEPTH-1:0];
        reg [ADDER_SIZE-1:0] internal_addr;

        always @(posedge clk ) begin
        if (~rst_n)
        dout <=0;
        else begin
            if (din[9:8] == 2'b00 && rx== 1)begin
                internal_addr <= din[7:0];
                tx =0;
            end
            else if (din[9:8] == 2'b10 && rx== 1)begin
                internal_addr <= din[7:0];
                tx =0;
            end
            else if (din[9:8] == 2'b01 && rx== 1)begin
```

```
                mem[internal_addr] <= din[7:0];
                tx = 0;
            end
            else if (din[9:8] == 2'b11 && rx== 1)begin
                dout <= mem[internal_addr];
                tx =1;
            end
            else
                tx=0;
        end
    end
endmodule
```

b.SPI Code

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

    (* fsm_encoding = "gray" *)
    //Internal signals
    reg [2:0] cs,ns;  //cs-> current state, ns-> next state
    integer READ_FLAG = 0;

    integer counter_convert = 10;
```

```
/**********************Combinational -> Next State*********************
always@(*)begin
  case (cs)
        IDLE:
            if(SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        CHK_CMD:
            if(SS_n)
                ns = IDLE;
            else begin
                if(MOSI)begin
                    if(READ_FLAG == 0)
                        ns = READ_ADD;
                    else
                        ns = READ_DATA;
                end
                else
                    ns = WRITE;
            end
        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
    end

```

```
/*******************Sequential -> State Memory , rst_n is synch********************/
always@(posedge clk)begin
    if(!rst_n)begin
        cs <= IDLE;
        READ_FLAG <= 0;
    end
    else
        cs <= ns;
end

/******************************Sequential -> Output*************************/
always@(posedge clk )begin
   case (cs)
        IDLE:begin
            counter_convert <= 9;
        end
        CHK_CMD:begin rx_valid <= 0;  //data is unavailable
                rx_data[counter_convert] <= MOSI; //convert serial to parallel (MSB)

         end
        WRITE:begin
                rx_data[counter_convert-1] <= MOSI; //convert serial to parallel
                counter_convert <= counter_convert - 1;
```

```
                if ( counter_convert == 0)
                 rx_valid <= 1;  //data is available
                end
            READ_ADD:begin

                rx_data[counter_convert-1] <= MOSI; //convert serial to parallel
                counter_convert <= counter_convert - 1;

               if ( counter_convert == 0) begin
                rx_valid <= 1;  //data is available
                READ_FLAG <= 1; //go to read data state
               end
            end
```

```
            READ_DATA:begin
                if(tx_valid)begin

                   MISO <= tx_data[counter_convert -1];    //convert parallel to serial
                   counter_convert <= counter_convert - 1;
                end
                else begin

                    rx_data[counter_convert-1] <= MOSI;
                    counter_convert <= counter_convert - 1;

                  if ( counter_convert == 0) begin
                    rx_valid <= 1;
                    READ_FLAG <= 0; //to read address state
                    counter_convert <= 9; //Ram will take one clk cycle to raise tx_valid to 1, so counter = 9 not 8
                  end
                end
            end
        endcase
    end
endmodule
```

c. Wrapper Code

```
module Wrapper_module(MOSI,MISO,SS_n,clk,rst_n);
    //Input output ports
    input MOSI,SS_n,clk,rst_n;
    output MISO;

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


    //Instantiations modules
    ram M1(rx_data,tx_data,rx_valid,tx_valid,clk,rst_n);
    SPI_FSM_module M2(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data
        ,tx_valid);

endmodule
```

2. Verification Flow

a. Verification plan

Our verification plan is directed and since we didn't initialize the ram, we will verify the following cases:

- Reset functionality
- Write address state (11001100 location)
- End communication (SS_n =1)
- Write data state (11001100 data written)
- End communication (SS_n =1)
- Read address state (11001100 location same address of write state)
- End communication (SS_n =1)
- Read data state (11001100 are data redden which was same data have written)

b. Test bench code

```
module Wrapper_tb();
    //signal declairation
    reg MOSI,SS_n,clk,rst_n;
    wire MISO;
    integer i;
    //Module instantiations
    Wrapper_module DUT(MOSI,MISO,SS_n,clk,rst_n);

    //clock generation
    initial begin
        clk = 0;
        repeat (150)
            #1 clk = ~clk;
    end

    //test stimulus generation
    initial begin
        rst_n = 0;  //enable reset
        SS_n = 0;   //start communication
        @(negedge clk);
        rst_n = 1;
        for(i = 0;i<50;i = i+1)begin
            MOSI = 0;
            SS_n = 0;   //start communication (CHK state 1 clk)
            @(negedge clk);
```

```
MOSI = 0;   //write addr state  (10 clk)
@(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;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
SS_n = 1;   //end communication (IDEL state)
@(negedge clk); $stop;

SS_n = 0;   //start communication  (CHK state 1 clk)
MOSI = 0;
```

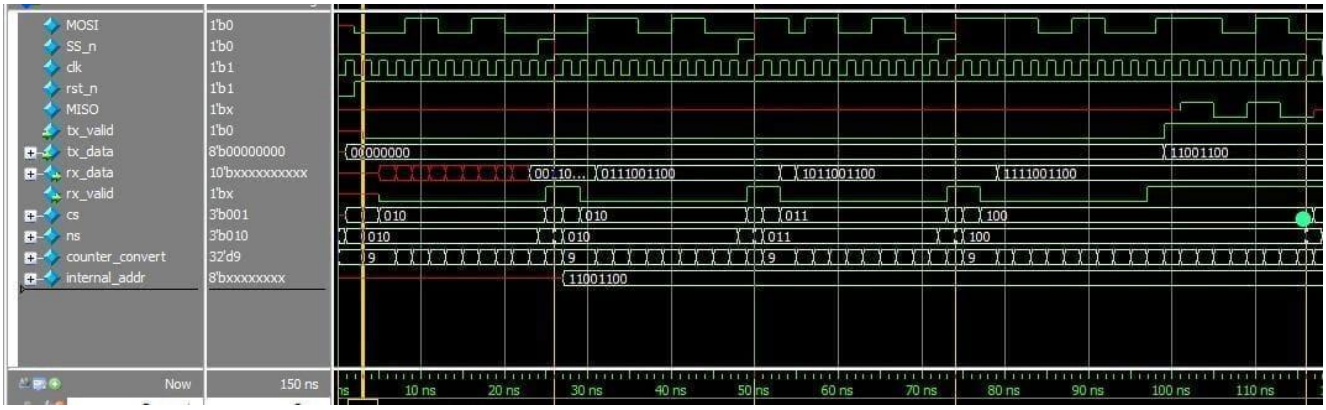
```
@(negedge clk);
MOSI = 0;  //write data state  (10 clk)
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
SS_n = 1;   //end communication
@(negedge clk); $stop;

SS_n = 0;   //start communication     (CHK state 1 clk)
```

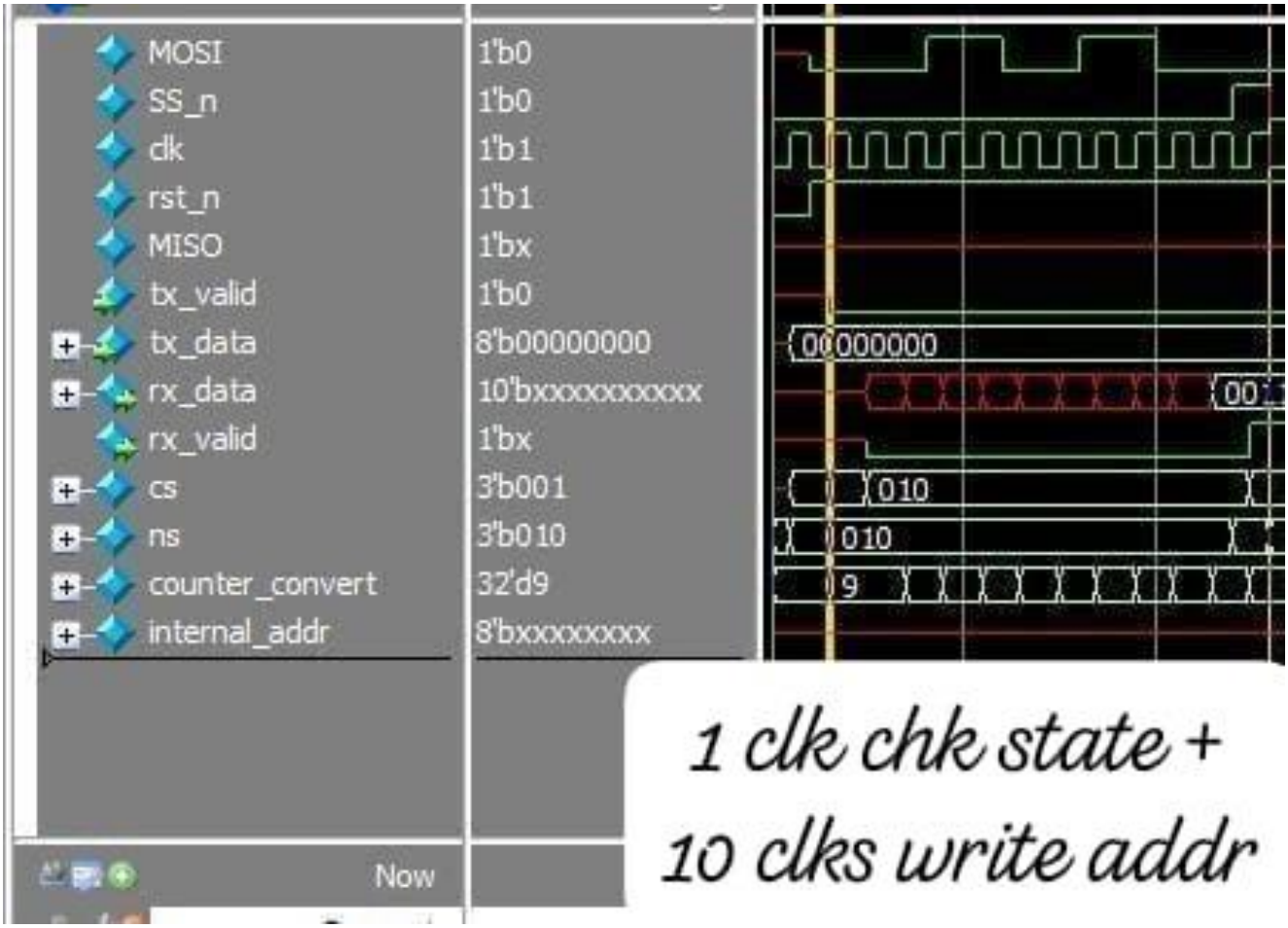
```
MOSI = 1;
@(negedge clk);
MOSI = 1;   //read addr state (same location @ which we have write 10 clk )
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
SS_n = 1;   //end communication (IDEL state)
@(negedge clk); $stop;
```

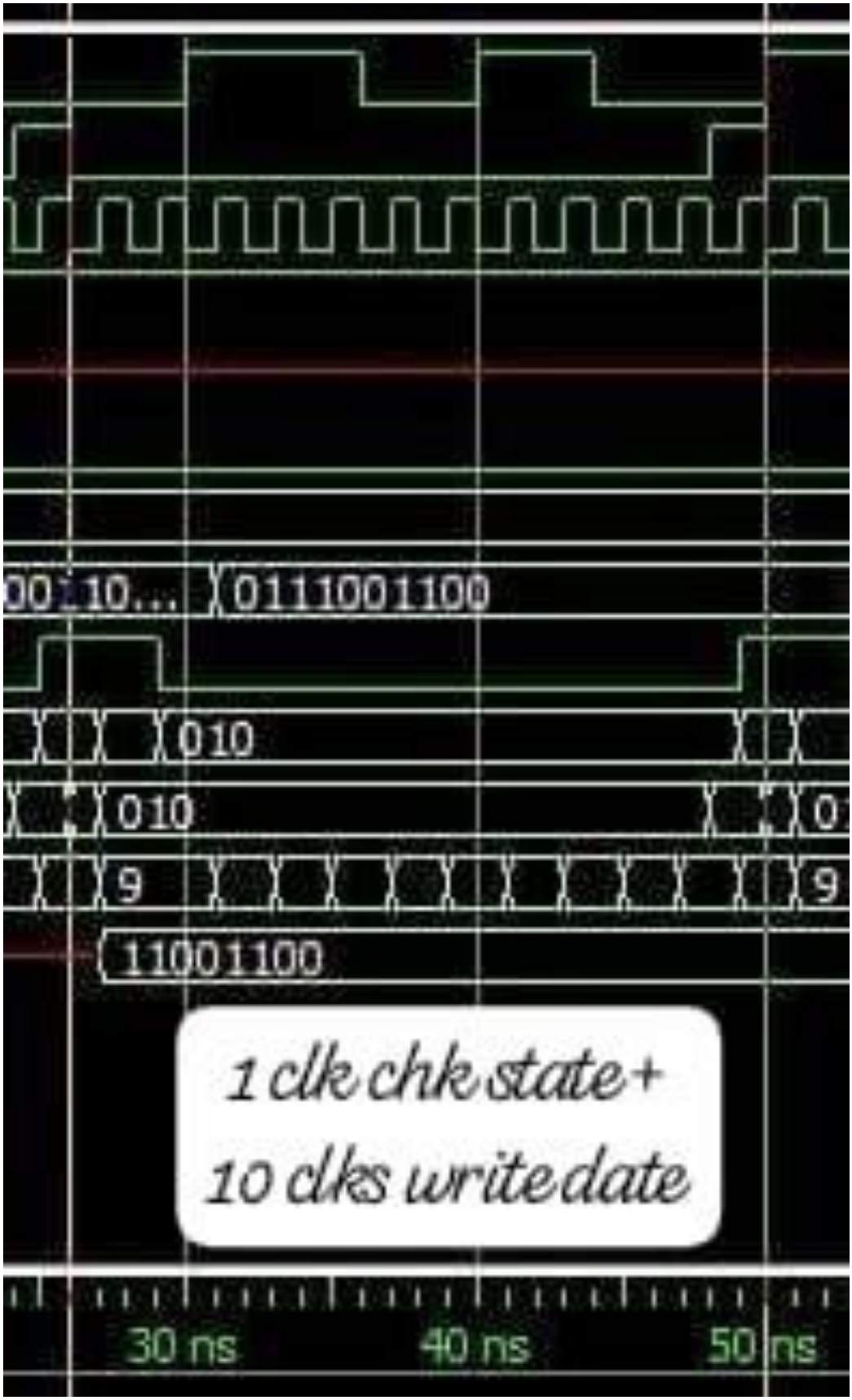
```
SS_n = 0;   //start communication (CHK state 1 clk)
MOSI = 1;
@(negedge clk);
MOSI = 1;  //read data state (same data which we have write 10 clk write + 10 clk read)
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 1;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
@(negedge clk);
MOSI = 0;
```

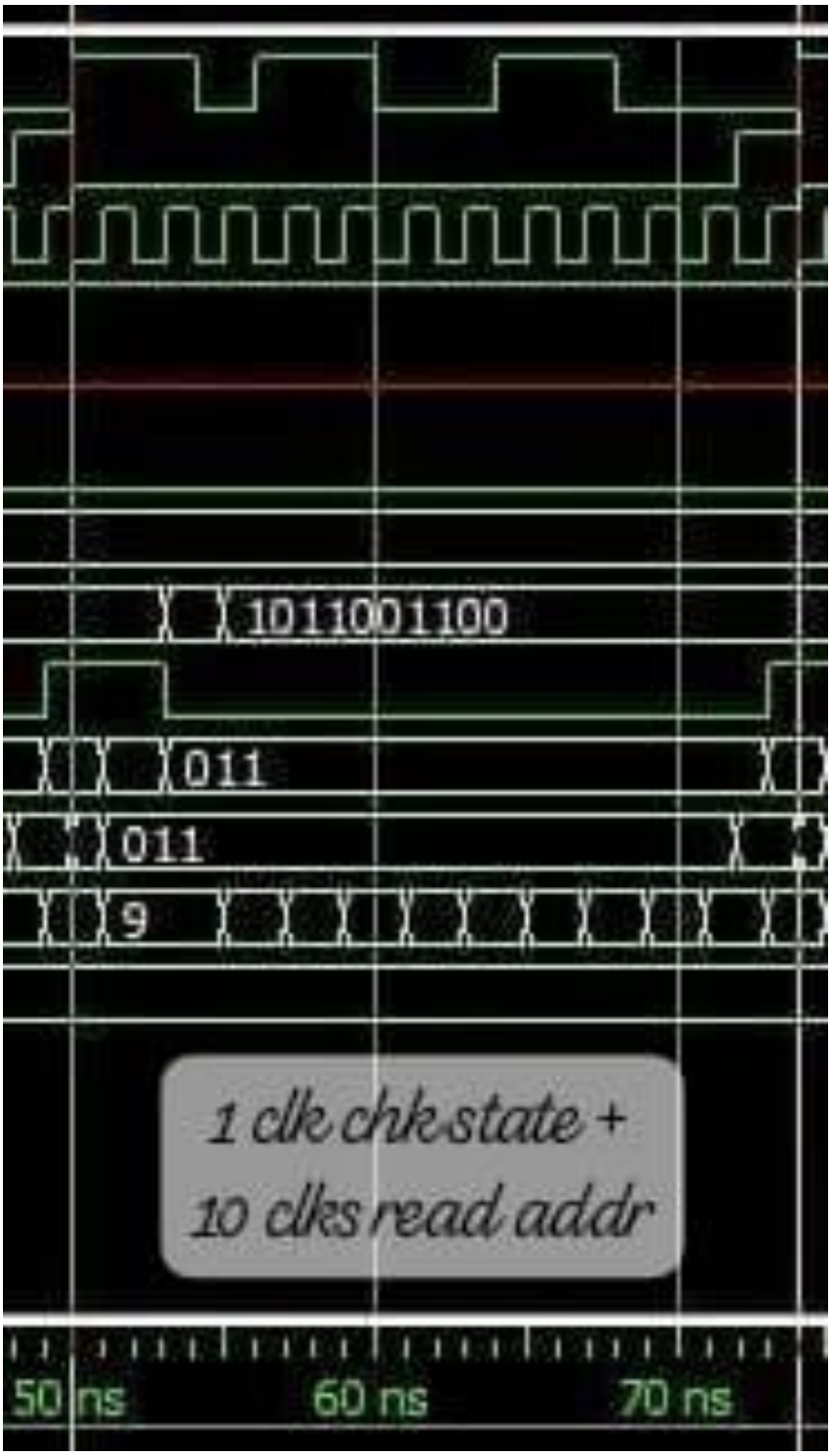
```
            @(negedge clk);

            MOSI = 1;
            @(negedge clk);
            MOSI = 1;
            @(negedge clk);
            MOSI = 1;
            @(negedge clk);
            MOSI = 1;
            @(negedge clk);
            MOSI = 0;
            @(negedge clk);
            MOSI = 0;
            @(negedge clk);
            MOSI = 1;
            @(negedge clk);
            MOSI = 1;
            @(negedge clk);
            MOSI = 0;
            @(negedge clk);
            SS_n = 1;          //end communication
            @(negedge clk);
        end
    end
endmodule
```

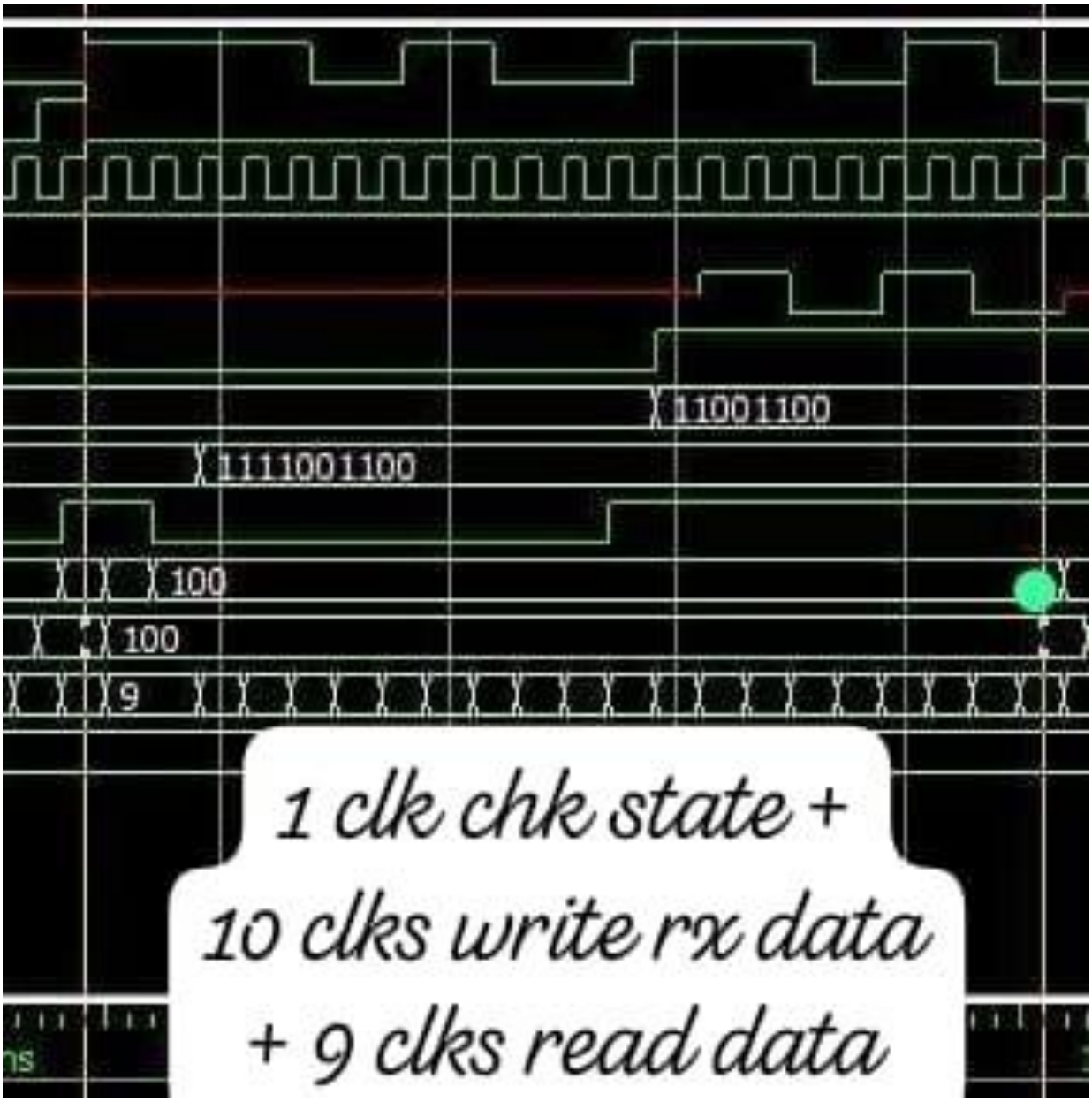
c. Waveforms


MOSI 1'b0
SS_n 1'b0
clk 1'b1
rst_n 1'b1
MISO 1'bx
tx_valid 1'b0
tx_data 8'b00000000
rx_data 10'bxxxxxxxxxx
rx_valid 1'bx
cs 3'b001
ns 3'b010
counter_convert 32'd9
internal_addr 8'bxxxxxxxx
00000000
0111001100
1011001100
1111001100
11001100
010
011
100
9
Now
150 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns


MOSI 1'b0
SS_n 1'b0
clk 1'b1
rst_n 1'b1
MISO 1'bx
tx_valid 1'b0
tx_data 8'b00000000
rx_data 10'bxxxxxxxxxx
rx_valid 1'bx
cs 3'b001
ns 3'b010
counter_convert 32'd9
internal_addr 8'bxxxxxxxx
00000000
010
010
9
Now
1 clk chk state +
10 clks write addr


00110...
0111001100
010
010
9
11001100
1 clk chk state+
10 clks write date
30 ns
40 ns
50 ns


1011001100
011
011
9
1 clk chk state +
10 clks read addr
50 ns
60 ns
70 ns



Notice that in the upper snippets we receive on MISO port the same pattern we have sent before (11001100).

3. Synthesis and Implementation Flow

a. Constrain file

```
## 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]
```

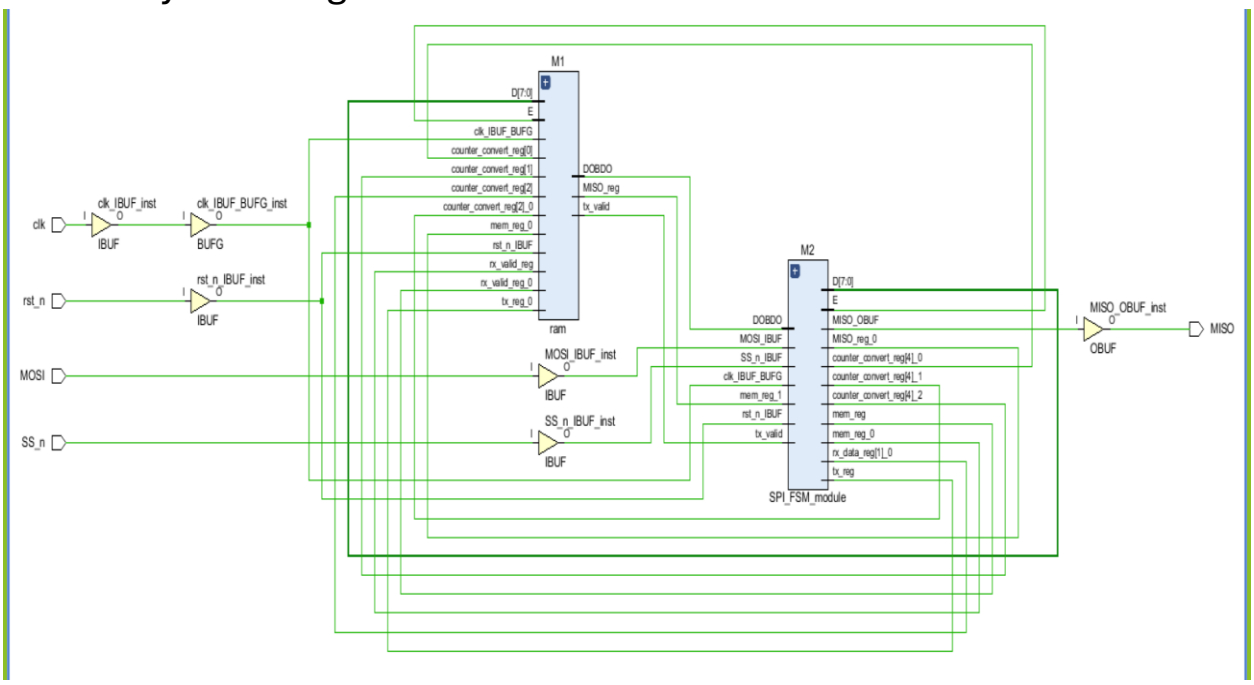
```
## Switches
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33}
[get_ports rst_n]
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33}
[get_ports SS_n]
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33}
[get_ports MOSI]
```

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

The snippets above shows the constrain file modifications in order to set input ,outputs and clks.

b. Synthesis Flow for encodings used

- Gray Encoding



Synthesis Schematic

```
    Parameter READ_DATA bound to: 3'b100
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *) [D:/KW Diploma/SPI Project/SPI_FSM_module.v:20]
INFO: [Synth 8-155] case statement is not full and has no default [D:/KW Diploma/SPI Project/SPI_FSM_module.v:77]
INFO: [Synth 8-6155] done synthesizing module 'SPI_FSM_module' (2#1) [D:/KW Diploma/SPI Project/SPI_FSM_module.v:3]
INFO: [Synth 8-6155] done synthesizing module 'Wrapper_module' (3#1) [D:/KW Diploma/SPI Project/Wrapper_module.v:3]
---------------------------------------------------------------------------------
Finished RTL Elaboration : Time (s): cpu = 00:00:07 ; elapsed = 00:00:07 . Memory (MB): peak = 410.633 ; gain = 153.902
---------------------------------------------------------------------------------
```

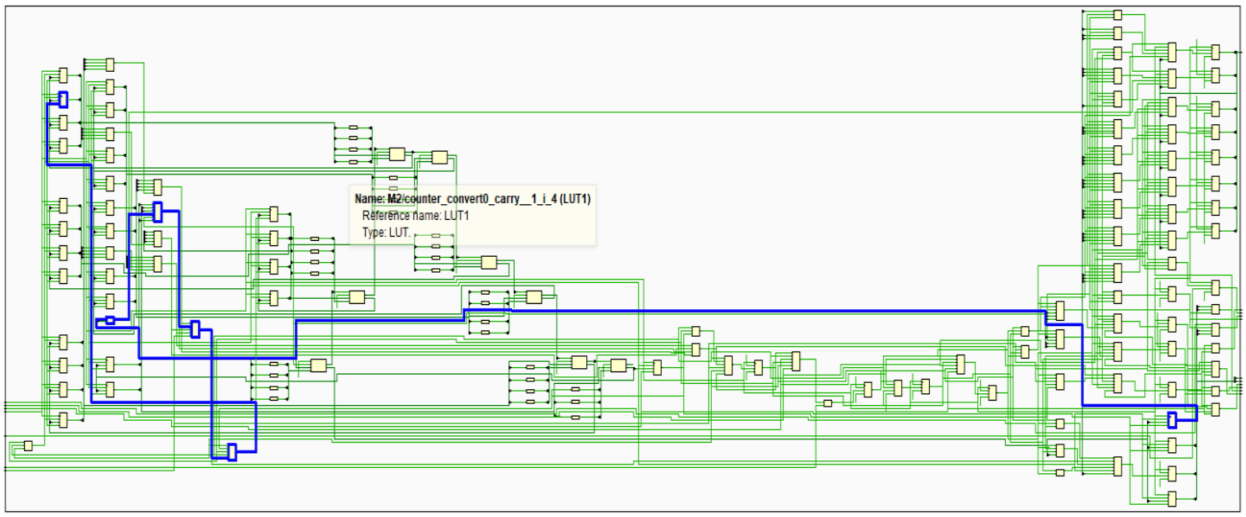
Synthesis report showing encoding used

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.059 ns	Worst Hold Slack (WHS):	0.074 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:	152	Total Number of Endpoints:	152	Total Number of Endpoints:	60

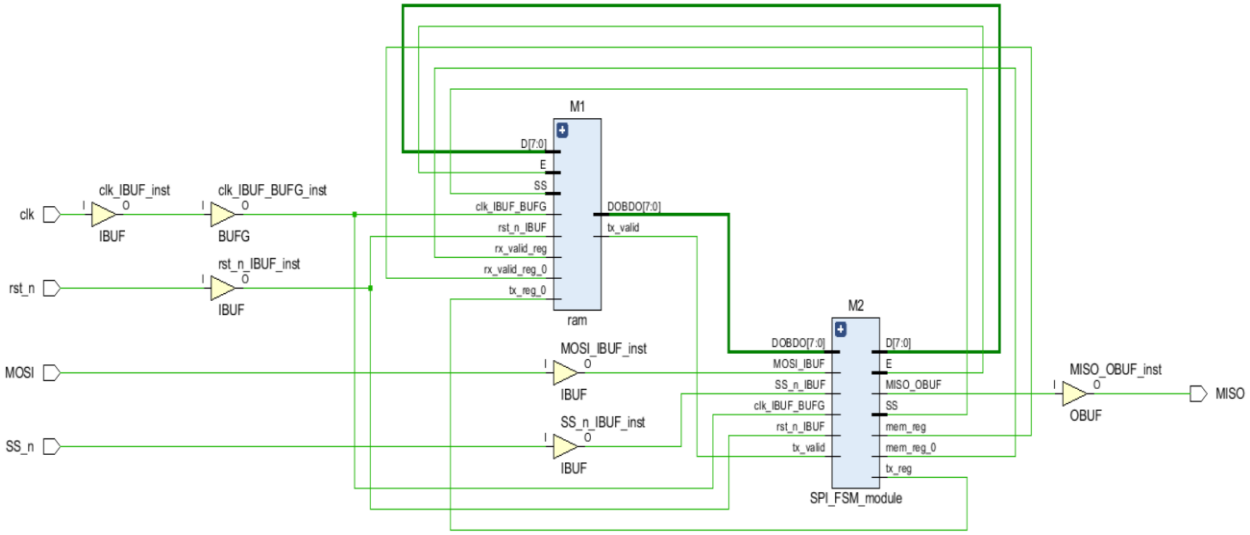
All user specified timing constraints are met.

Timing report



Schematic Showing Critical Path

- One_Hot Encoding



Synthesis Schematic

```
encoding    Next    Previous    Highlight    Match Case    Whole Words    4 Match(es)
32    Parameter READ_DATA bound to: 3'b100
33  INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "one_hot" *) [D:/KW Diploma/SPI Project/SPI_FSM_module.v:20]
34  INFO: [Synth 8-155] case statement is not full and has no default [D:/KW Diploma/SPI Project/SPI_FSM_module.v:77]
35  INFO: [Synth 8-6155] done synthesizing module 'SPI_FSM_module' (2#1) [D:/KW Diploma/SPI Project/SPI_FSM_module.v:3]
36  INFO: [Synth 8-6155] done synthesizing module 'Wrapper_module' (3#1) [D:/KW Diploma/SPI Project/Wrapper_module.v:3]
37  ---------------------------------------------------------------------------
38  Finished RTL Elaboration : Time (s): cpu = 00:00:07 ; elapsed = 00:00:09 . Memory (MB): peak = 409.496 ; gain = 152.586
39  ---------------------------------------------------------------------------
40
```

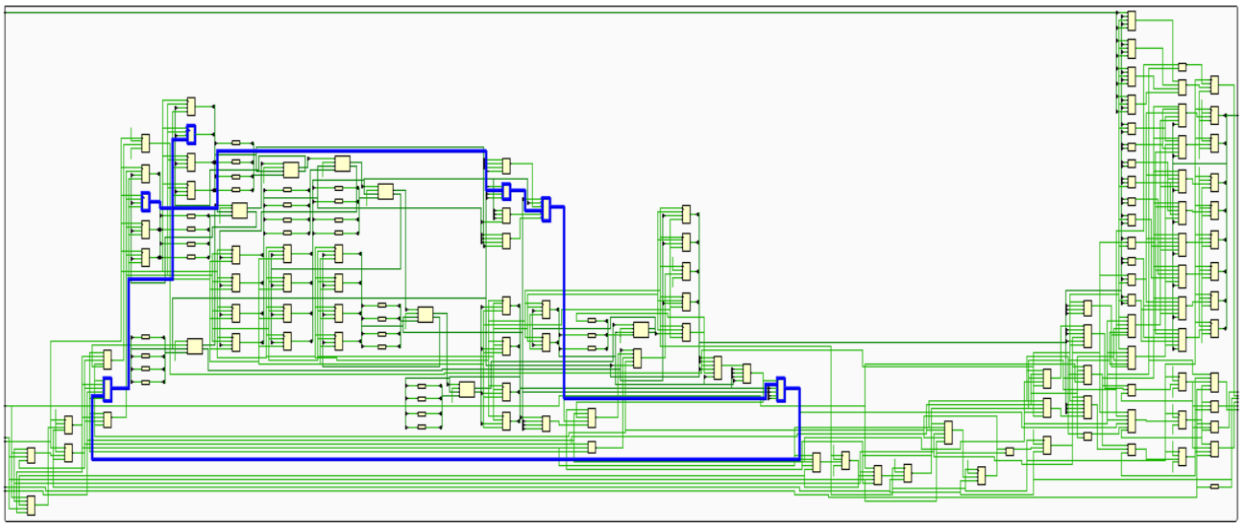
Synthesis report showing encoding used

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.830 ns	Worst Hold Slack (WHS):	0.074 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:	154	Total Number of Endpoints:	154	Total Number of Endpoints:	62

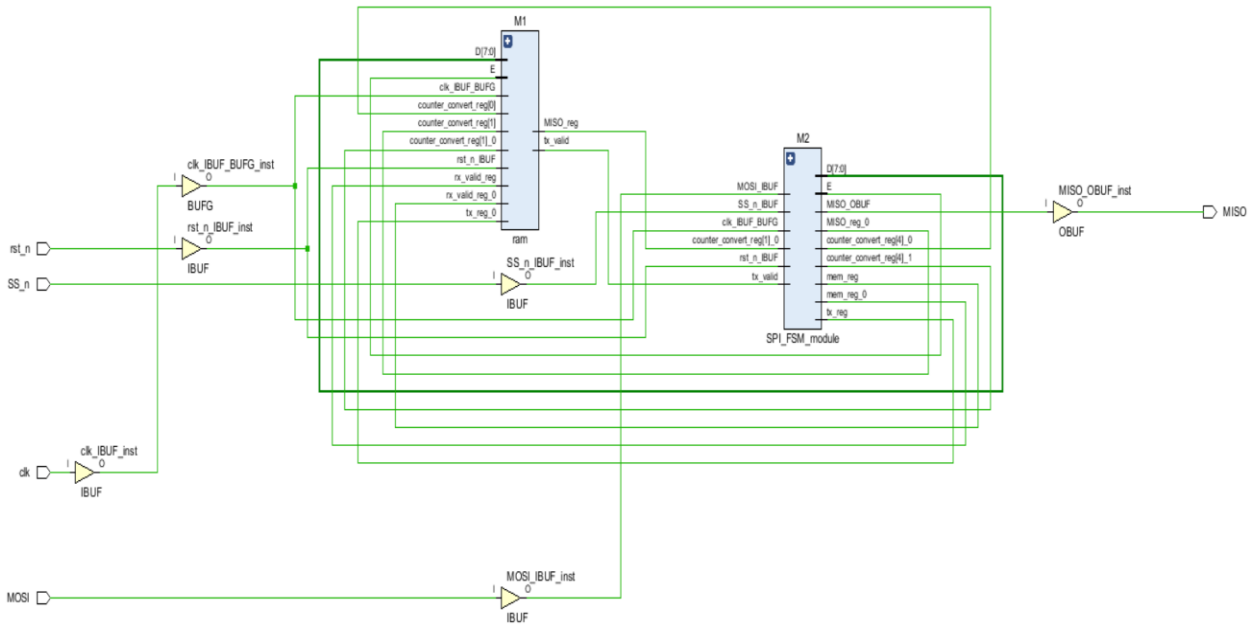
All user specified timing constraints are met.

Timing report



Schematic Showing Critical Path

- Sequential Encoding



Synthesis Schematic

```
    Parameter READ_DATA bound to: 3'b100
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "sequential" *) [D:/KW Diploma/SPI Project/SPI_FSM_module.v:20]
INFO: [Synth 8-155] case statement is not full and has no default [D:/KW Diploma/SPI Project/SPI_FSM_module.v:77]
INFO: [Synth 8-6155] done synthesizing module 'SPI_FSM_module' (2#1) [D:/KW Diploma/SPI Project/SPI_FSM_module.v:3]
INFO: [Synth 8-6155] done synthesizing module 'Wrapper_module' (3#1) [D:/KW Diploma/SPI Project/Wrapper_module.v:3]
---------------------------------------------------------------------------
Finished RTL Elaboration : Time (s): cpu = 00:00:04 ; elapsed = 00:00:04 . Memory (MB): peak = 410.824 ; gain = 153.348
---------------------------------------------------------------------------
```

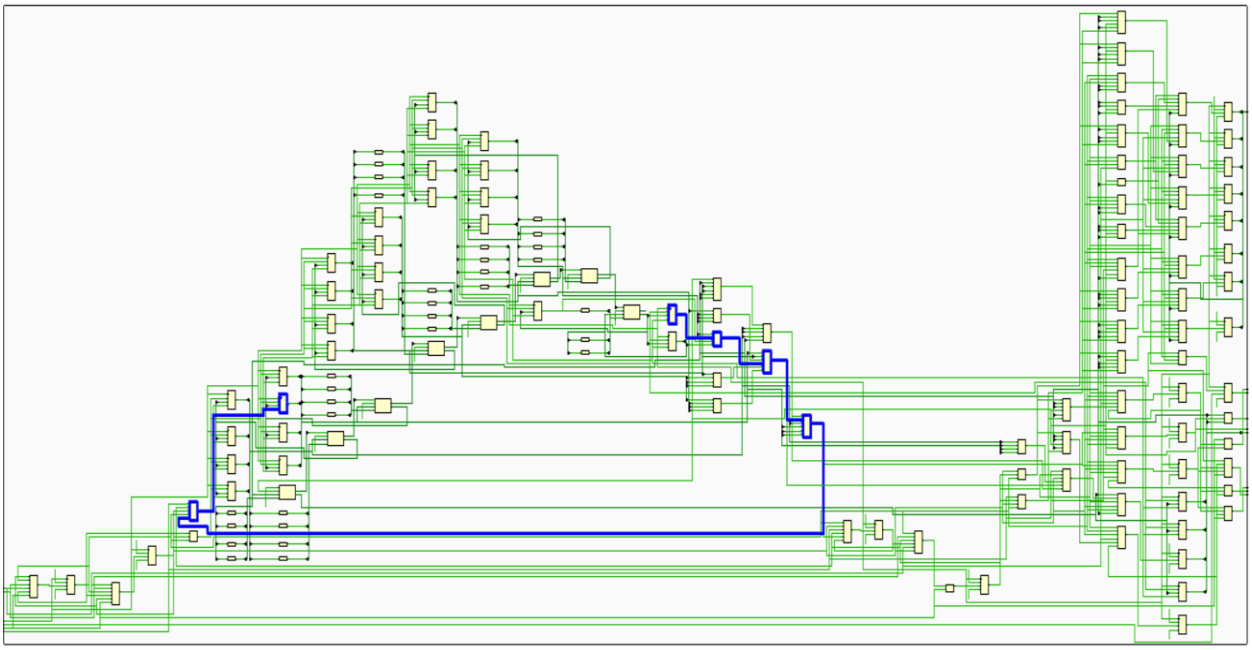
Synthesis report showing encoding used

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.824 ns	Worst Hold Slack (WHS):	0.074 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:	152	Total Number of Endpoints:	152	Total Number of Endpoints:	60

All user specified timing constraints are met.

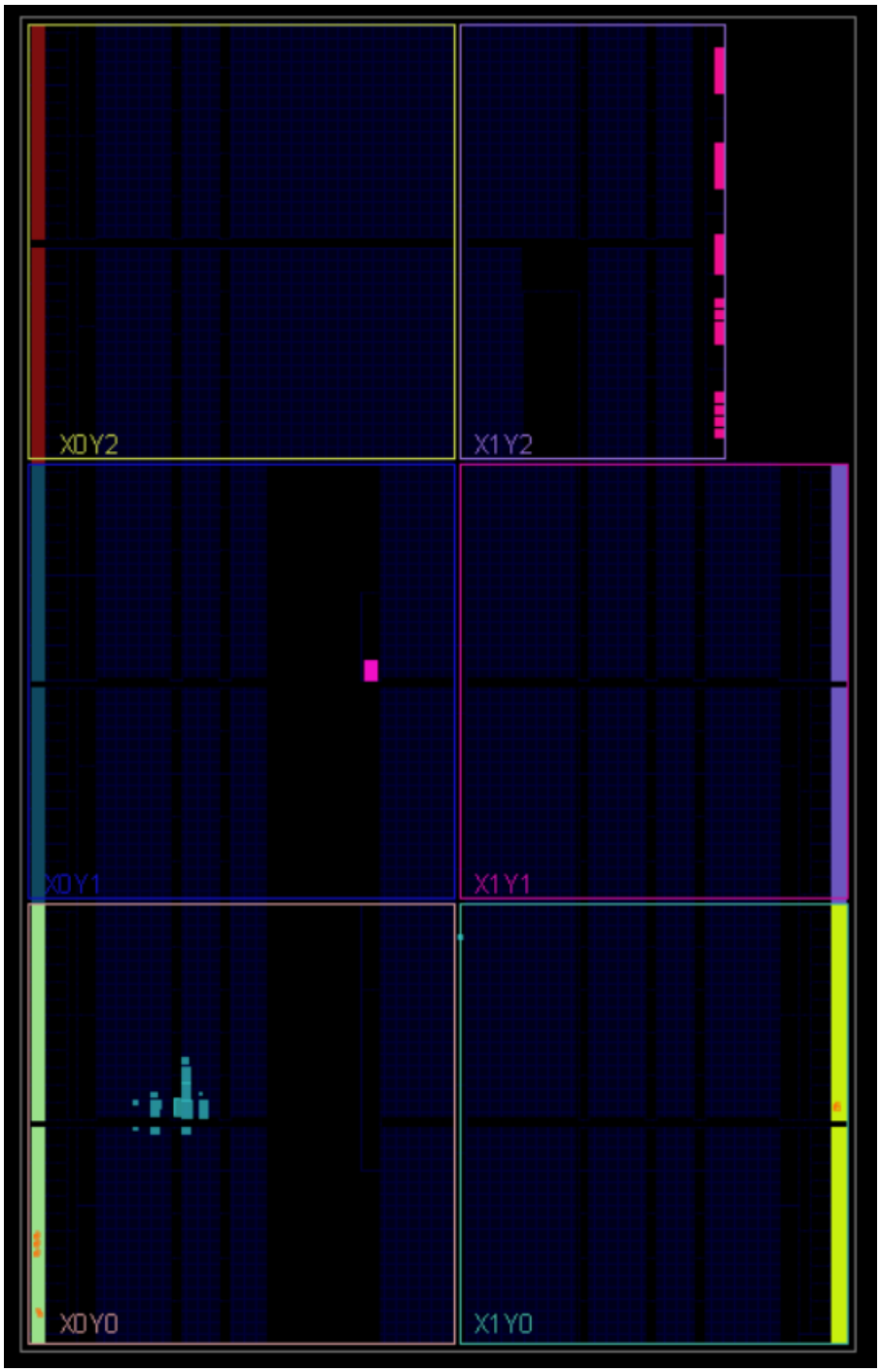
Timing report



Schematic Showing Critical Path

C. Implementation Flows for Encoding Used

- Gray Encoding



Device Implementation

Hierarchy

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)
Wrapper_module	88	57	30	88	49	0.5	5	1
M1 (ram)	4	9	6	4	0	0.5	0	0
M2 (SPI_FSM_module)	84	48	30	84	47	0	0	0

Utilization Report

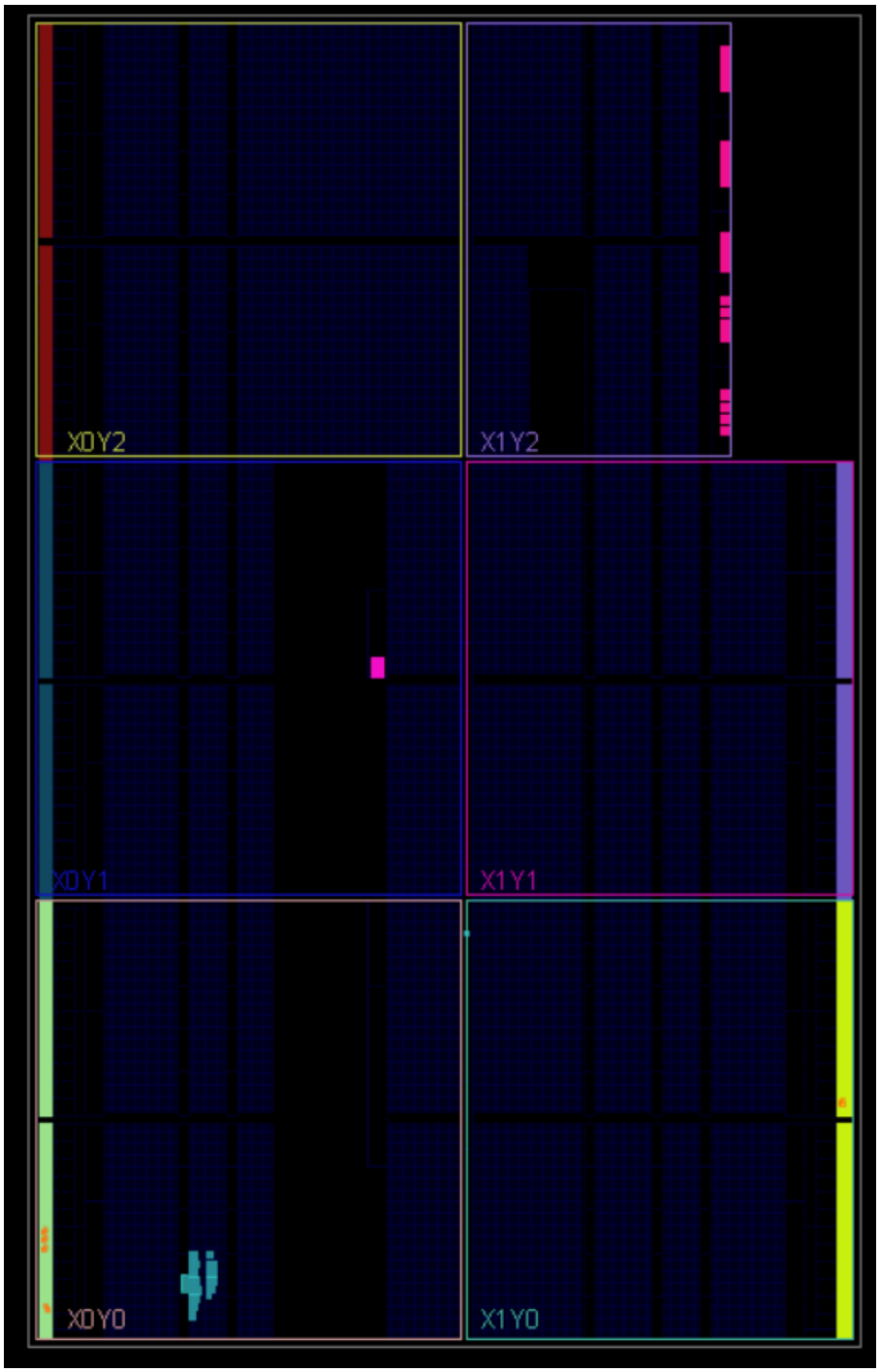
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.150 ns	Worst Hold Slack (WHS):	0.077 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:	152	Total Number of Endpoints:	152	Total Number of Endpoints:	60

All user specified timing constraints are met.

Timing Report

- One_Hot Encoding



Device Implementation

Hierarchy

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)
Wrapper_module	85	59	24	85	50	0.5	5	1
M1 (ram)	1	9	2	1	0	0.5	0	0
M2 (SPI_FSM_module)	84	50	24	84	49	0	0	0

Utilization Report

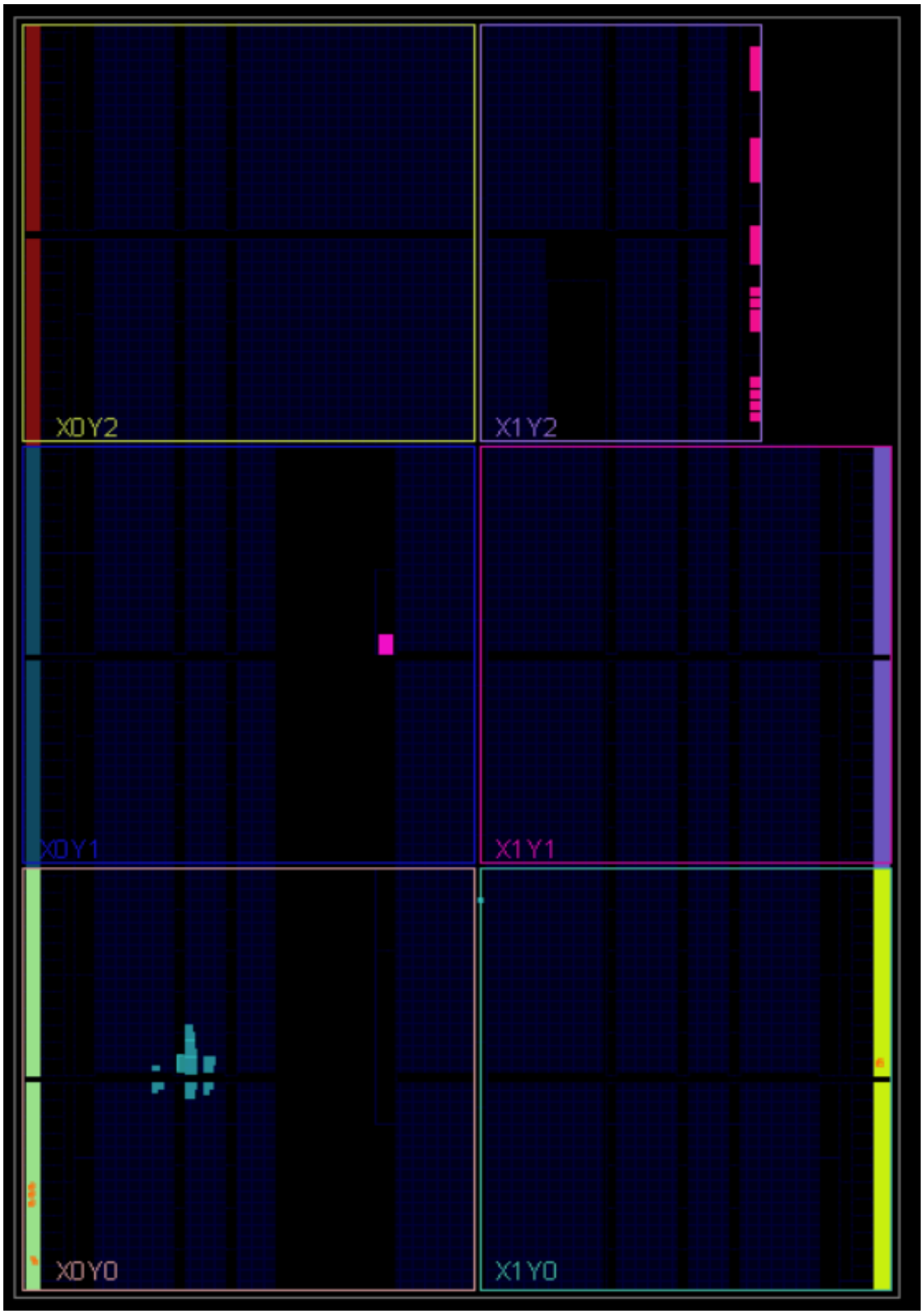
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.347 ns	Worst Hold Slack (WHS):	0.080 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:	154	Total Number of Endpoints:	154	Total Number of Endpoints:	62

All user specified timing constraints are met.

Timing Report

- Sequential Encoding



Device Implementation

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
Wrapper_module	85	57	1	28	85	49	0.5	5	1
M1 (ram)	3	9	1	3	3	0	0.5	0	0
M2 (SPI_FSM_module)	82	48	0	28	82	47	0	0	0

Utilization Report

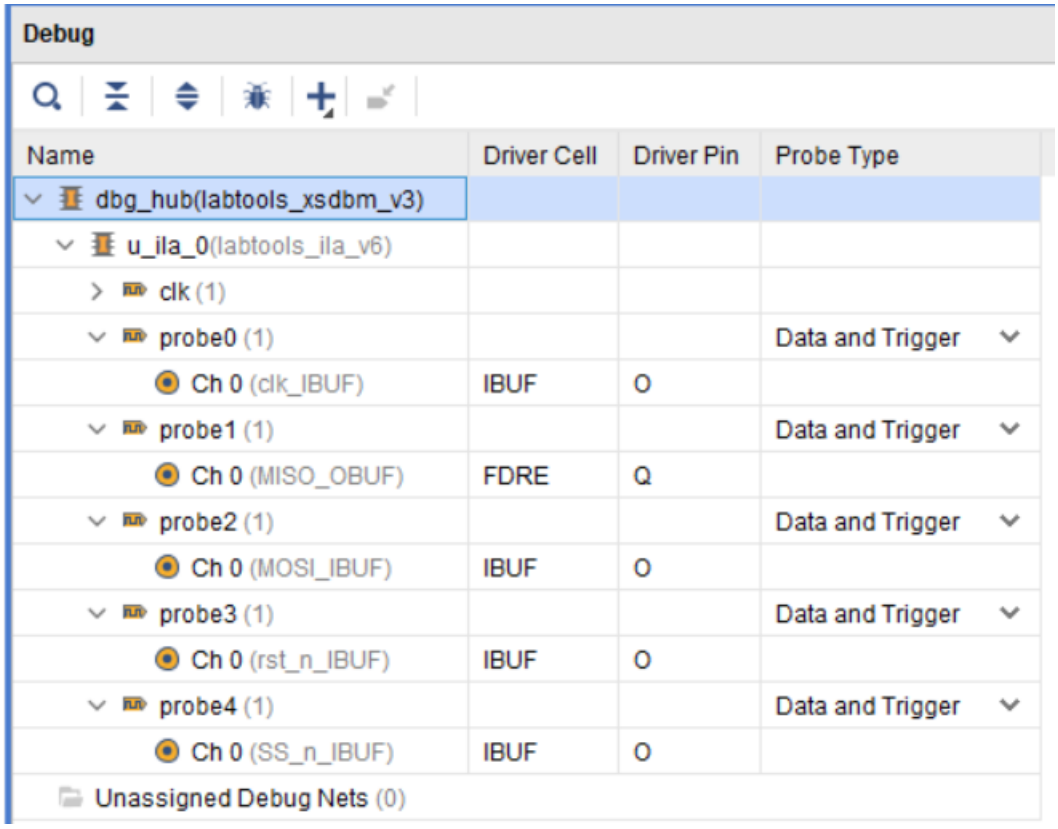
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.968 ns	Worst Hold Slack (WHS):	0.080 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	152	Total Number of Endpoints:	152	Total Number of Endpoints:	60

All user specified timing constraints are met.

Timing Report

After comparing timing reports, seeing that gray encoding has the highest setup/hold slack after implementation, proceeding in adding debugging core.

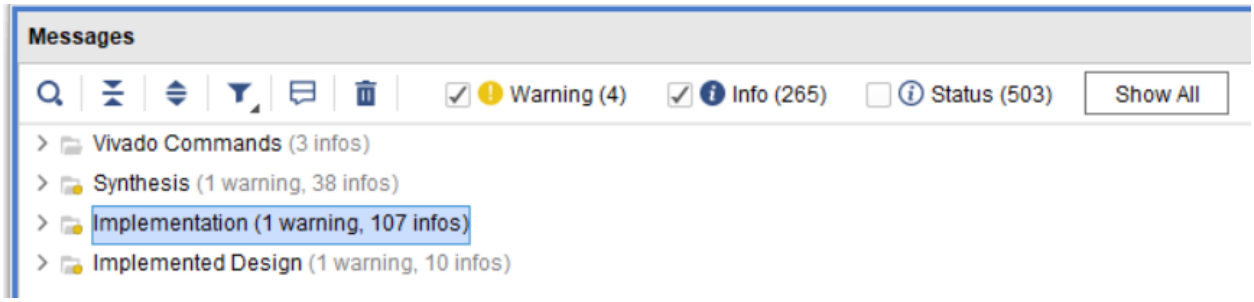
- Debugging core added



Debug

Name	Driver Cell	Driver Pin	Probe Type
dbg_hub(labtools_xsdbm_v3)			
u_ila_0(labtools_ila_v6)			
clk (1)			
probe0 (1)			Data and Trigger
Ch 0 (clk_IBUF)	IBUF	O	
probe1 (1)			Data and Trigger
Ch 0 (MISO_OBUF)	FDRE	Q	
probe2 (1)			Data and Trigger
Ch 0 (MOSI_IBUF)	IBUF	O	
probe3 (1)			Data and Trigger
Ch 0 (rst_n_IBUF)	IBUF	O	
probe4 (1)			Data and Trigger
Ch 0 (SS_n_IBUF)	IBUF	O	
Unassigned Debug Nets (0)			

- Messages Tab showing no critical warnings or errors



Bitstream Generation