SPI Slave with Single Port RAM

RTL and FPGA design

Ahmed Mohamed Abdellatif

Contents

Overview

- Objective

To design and implement an SPI communication protocol in Verilog HDL. The system features an SPI master, an integrated SPI wrapper module that combines the SPI slave and RAM functionalities.

- Applications

An SPI Slave with Single Port RAM is useful for applications in embedded systems, industrial automation, consumer electronics, IoT devices, networking, medical devices, memory expansion, and data storage. It provides efficient communication, real-time data buffering, and temporary storage in low-power, constrained environments.

- System Design

1. SPI Slave Module

Handles communication with the master device using the SPI protocol, receiving and transmitting data to and from the RAM module.

2. Single Port RAM Module

Provides memory storage, allowing for efficient read/write operations of data received via the SPI interface.

3. SPI Wrapper Module

Integrates the SPI Slave and RAM, managing data flow and ensuring seamless communication between them.

RTL design

- Spi slave

```
module SPI_Slave (mosi, ss_n, clk, rst_n, tx_valid, tx_data, miso, rx_valid, rx_data);

//State parameters
parameter IDLE=0;
parameter CHK_CMD=1;
parameter WRITE=2;
parameter READ_ADD=3;
parameter READ_DATA=4;

//Input ports
input mosi;                         //Master_out_slave_in input
input ss_n, clk, rst_n, tx_valid;   //Control input signals
input[7:0] tx_data;                 //The parallel data sent from the RAM to SPI

//Output ports
output reg miso;                    //Master_in_slave_out output
output reg rx_valid;                //Control signal
output reg[9:0] rx_data;            //The parallel data sent to RAM

//Internal signals
reg[3:0] counter,counter_rd_data;   //Counter for serial / parallel tranformations
reg[2:0] cs,ns;                     //State rergisters
reg end_counter_rd_data;            //Internal control signals

//State memory
always @(posedge clk) begin
    if(~rst_n)
        cs<=IDLE;
    else
        cs<=ns;
end
```

```
//Next state combinational logic
always @(*) begin
    case (cs)
        IDLE: begin
            if (ss_n == 0)
                ns = CHK_CMD;
            else
                ns = IDLE;
        end
        CHK_CMD: begin
            if ((ss_n == 0) && rx_valid && (rx_data[9] == 0))
                ns = WRITE;
            else if ((ss_n == 0) && rx_valid && (rx_data[9:8] == 2'b10))
                ns = READ_ADD;
            else if ((ss_n == 0) && rx_valid && (rx_data[9:8] == 2'b11))
                ns = READ_DATA;
            else if(rx_valid == 0)
                ns = CHK_CMD;
            else
                ns = IDLE;
        end
        WRITE: begin
            if (ss_n == 1)
                ns = IDLE;
            else
                ns = WRITE;
        end
```

```
        end
        READ_ADD: begin
            if (ss_n == 1)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA: begin
            if (ss_n == 1)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
        default: ns = IDLE;
    endcase
end
```

```
75
76  //Serial mosi to parallel rx_data (from SPI to RAM)
77  always @(posedge clk) begin
78      if(~rst_n) begin
79          rx_data <= 0;
80          counter <= 0;
81          rx_valid <= 0;
82      end
83      else if (counter >= 10) begin
84          counter <= 0;
85          rx_valid <= 1;
86      end
87      else if((cs == CHK_CMD) && !rx_valid) begin
88          rx_data[9-counter] <= mosi;
89          counter <= counter + 1;
90      end
91      else if (cs == IDLE)
92          rx_valid <= 0;
93  end
```

```
94
95  //Parallel tx_data to serial miso (read data from RAM to SPI)
96  always @(posedge clk) begin
97      if(~rst_n) begin
98          miso <= 0;
99          counter_rd_data <= 0;
100         end_counter_rd_data <= 0;
101     end
102     else if (counter_rd_data >= 8) begin
103         counter_rd_data <= 0;
104         end_counter_rd_data <= 1;
105     end
106     else if ((cs == READ_DATA) && tx_valid && !end_counter_rd_data) begin
107         miso <= tx_data[7-counter_rd_data];
108         counter_rd_data <= counter_rd_data+1;
109     end
110     else if (cs == IDLE) begin
111         end_counter_rd_data <= 0;
112     end
113 end
114 endmodule
```

- Testbench for the SPI slave

```
module SPI_Slave_tb ();

    reg mosi,ss_n,clk,rst_n,tx_valid;
    reg[7:0] tx_data;
    wire miso,rx_valid; // rx_valid show the counter ends
    wire[9:0] rx_data;

    SPI_Slave dut0 (mosi,ss_n,clk,rst_n,tx_valid,tx_data,miso,rx_valid,rx_data);

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

    initial begin
//1
        rst_n=0; mosi=$random; ss_n=1; tx_valid=1; tx_data=$random;
        @(negedge clk);
//2
        rst_n=1; mosi=$random; ss_n=1; tx_valid=1; tx_data=$random;
        @(negedge clk);
//3
        ss_n=0; tx_data=$random;  tx_valid=1;
        repeat(30) begin
            mosi=$random; ss_n=0;
            @(negedge clk);
        end
        ss_n=1;
        @(negedge clk);
```

```
//4
        ss_n=0; tx_data=$random; tx_valid=1;
        repeat(30) begin
            mosi=$random; ss_n=0;
            @(negedge clk);
        end
        ss_n=1;
        @(negedge clk);

//5
        ss_n=0; tx_data=$random; tx_valid=1;
        repeat(30) begin
            mosi=$random; ss_n=0;
            @(negedge clk);
        end
        ss_n=1;
        @(negedge clk);

        $stop;
    end

    initial begin
        $monitor("rst_n=%b ,ss_n=%b ,mosi=%b ,rx_valid=%b ,rx_data=%b ,tx_valid=%b ,tx_data=%b ,miso=%b",rst_n,ss_n,mosi,rx_vali
    end

endmodule
```

- RAM

```
input clk, rst_n, rx_valid;     //Control input signals

//Output ports
output reg [ADDR_SIZE - 1 : 0]dout; //8-bits Output data from the RAM
output reg tx_valid;                //Control output signal

reg [ADDR_SIZE - 1 : 0] mem [MEM_DEPTH - 1 : 0];    //Memory
reg [ADDR_SIZE - 1 : 0] w_addr,r_addr;              //Internal registers to store the read and write addresses

//RAM sequential logic
always @(posedge clk) begin
    //Active low async reset
    if(~rst_n)  begin
        dout <= 0;
        tx_valid <= 0;
        w_addr <= 0;
        r_addr <=0;
    end
    //rx_valid refers to that there is data on din ready to be read
    else if (rx_valid) begin
        //The MSB 2-bits are control signals
        case (din[9:8])
            2'b00: w_addr <= din[ADDR_SIZE-1 : 0];      //Store the Write addres
            2'b01: mem[w_addr] <= din[ADDR_SIZE-1 : 0]; //Write the data in the stored write address
            2'b10: r_addr <= din[ADDR_SIZE-1 : 0];      //Store the Read address
            2'b11: begin                                //Read the data stored in memory at read address
                dout <= mem[r_addr];
                tx_valid <= 1;
            end
        endcase
    end
end
endmodule
```

- Testbench for the RAM part

```
module RAM_tb ();
    reg [9:0] din;
    reg clk,rst_n,rx_valid;
    wire [7:0] dout;
    wire tx_valid;
    RAM dut (clk,rst_n,din,rx_valid,dout,tx_valid);
    initial begin
        clk=0;
        forever begin
        #5; clk=~clk;
        end
    end
    initial begin
        $readmemh("mem.dat",dut.mem);
        rst_n=0; rx_valid=0; din=$random();
        @(negedge clk);
        rst_n=1; rx_valid=1;
        @(negedge clk);
        repeat(1000) begin
            din=$random();
            @(negedge clk);
        end
        $stop;
    end
endmodule
```

- Top module

```
module SPI_top_module (mosi, ss_n, clk, rst_n, miso);

//Input ports
input clk, rst_n, ss_n; //Control signals
input mosi;             //Master_out_slave_in input

//Output ports
output miso;            //Master_in_slave_out output

//Internal signals
wire[9:0] rx_data;          //The parallel data sent to RAM
wire rx_valid, tx_valid;    //Control signals for the read and write
wire[7:0] tx_data;          //The parallel data sent from the RAM to SPI

//SPI module
SPI_Slave dut0 (mosi, ss_n, clk,rst_n, tx_valid, tx_data, miso, rx_valid, rx_data);

//RAM module
RAM dut1 (.clk(clk), .rst_n(rst_n), .rx_valid(rx_valid), .tx_valid(tx_valid), .din(rx_data), .dout(tx_data));
endmodule
```

Testbench

```
module SPI_top_module_tb ();

 //Ports
 reg mosi, ss_n, clk,rst_n;
 wire miso;

 //The SPI module
 SPI_top_module dut (mosi, ss_n, clk, rst_n, miso);

 //Initiate the clock
 initial begin
     clk=0;
     forever
         #2 clk=~clk;
 end
```

```
integer i;
initial begin

    //Test reset & intiate signals
    mosi = 0;   ss_n = 1;
    rst_n = 0;
    @(negedge clk);
    if (miso != 0) begin
        $display("there was an error");
        $stop;
    end

    rst_n = 1;  //Release the reset button

    //Test write & its address
    repeat (1000) begin
        ss_n = 0;                           //Release the ss_n to start communication
        @(negedge clk);

        //Sends the Write address
        for (i = 0; i < 10; i = i + 1) begin
            if (i < 2) begin
                mosi = 0;                   //first 2 bits are 00 to intiate write address
            end
            else
                mosi = $random;             //sends 8 bits to be stored in the write address
            @(negedge clk);
        end
```

```
        repeat(2) @(negedge clk);           //Wait for the write address to read din in 2 clocks

        ss_n = 1;                           //Set ss_n to stop the communication and start new one
        @(negedge clk);

        ss_n = 0;                           //Release the ss_n to start new task
        @(negedge clk);

        //Sends the data to be stored
        for (i = 0; i < 10; i = i + 1) begin
            if (i == 0) begin
                mosi = 0;                   //first 2 bits are 01 to intiate write data
            end
            else if (i == 1) begin
                mosi = 1;
            end
            else
                mosi = $random;             //sends 8 bits to be stored in the memory
            @(negedge clk);
        end

        repeat(2) @(negedge clk);           //Wait for the RAM to store din in 2 clocks

        ss_n = 1;                           //Set ss_n to stop the communication and start new one
        @(negedge clk);
    end
    $stop;                                  //Pause the sim after writing the memory
```

```
    //test Read & its address
    repeat (1000) begin
        ss_n = 0;                           //Release the ss_n to start communication
        @(negedge clk);

        // sends the Read address
        for (i = 0; i < 10; i = i + 1) begin
            if (i == 0) begin
                mosi = 1;                   //first 2 bits are 10 to intiate Read address
            end
            else if (i == 1) begin
                mosi = 0;
            end
            else
                mosi = $random;             //sends 8 bits to be stored in the Read address
            @(negedge clk);
        end

        repeat(2) @(negedge clk);           //Wait for the read address to store din in 2 clocks

        ss_n = 1;                           //Set ss_n to stop the communication and start new one
        @(negedge clk);

        ss_n = 0;                           //Release the ss_n to start a new task
        @(negedge clk);
```

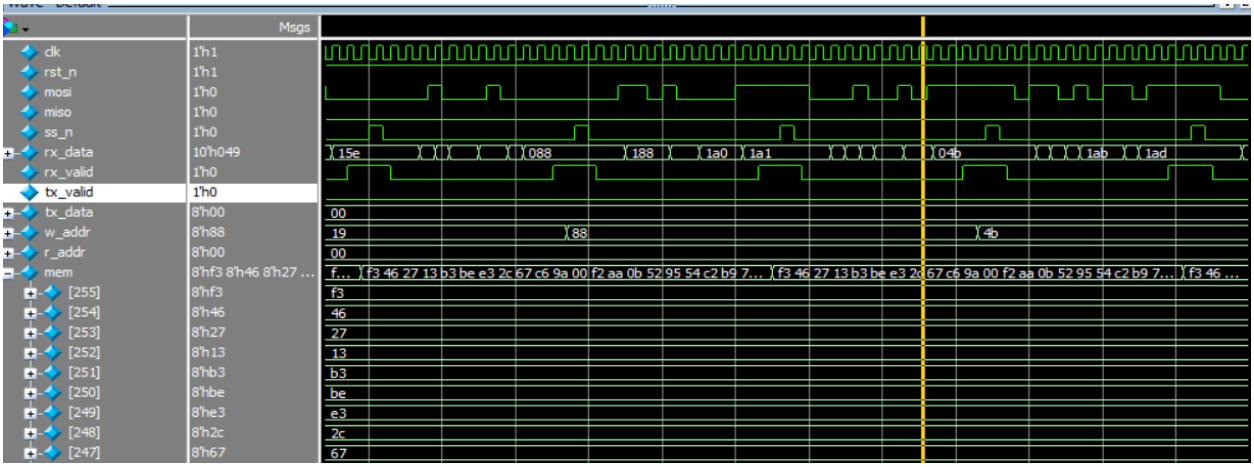
```
        //Receives the read data
        for (i = 0; i < 10; i = i + 1) begin
            if (i < 2) begin
                mosi = 1;                   //first 2 bits are 11 to intiate read data
            end
            else
                mosi = $random;             //the rest of the 8 bits are dummy data
            @(negedge clk);
        end

        repeat(2) @(negedge clk);           //Wait for the RAM to send the data on TX_data from mem read address stored

        repeat(9) @(negedge clk);           //Wait for the SPI slave to transfer the TX_data from parallel to series on the mis

        ss_n = 1;                           //Set ss_n to stop the communication and start new one
        @(negedge clk);
    end
    $stop;                                  //End of simulation
end
endmodule
```

DO file

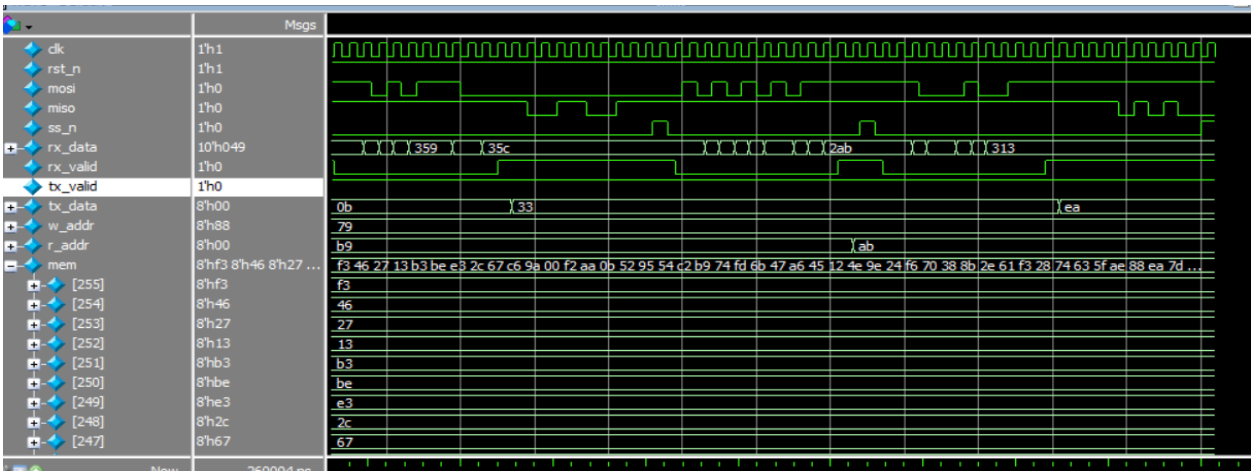
```
vlib work
vlog RAM.v SPI_Slave.v SPI_top_module.v SPI_tb.v
vsim -voptargs=+acc work.SPI_top_module_tb
add wave *
add wave -position insertpoint  \
sim:/SPI_top_module_tb/dut/rx_data \
sim:/SPI_top_module_tb/dut/rx_valid \
sim:/SPI_top_module_tb/dut/tx_valid \
sim:/SPI_top_module_tb/dut/tx_data
add wave -position insertpoint  \
sim:/SPI_top_module_tb/dut/dut1/mem \
sim:/SPI_top_module_tb/dut/dut1/w_addr \
sim:/SPI_top_module_tb/dut/dut1/r_addr
run -all
#quit -sim
```

Simulation

- Write operation



- Read operation



Constrain file

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

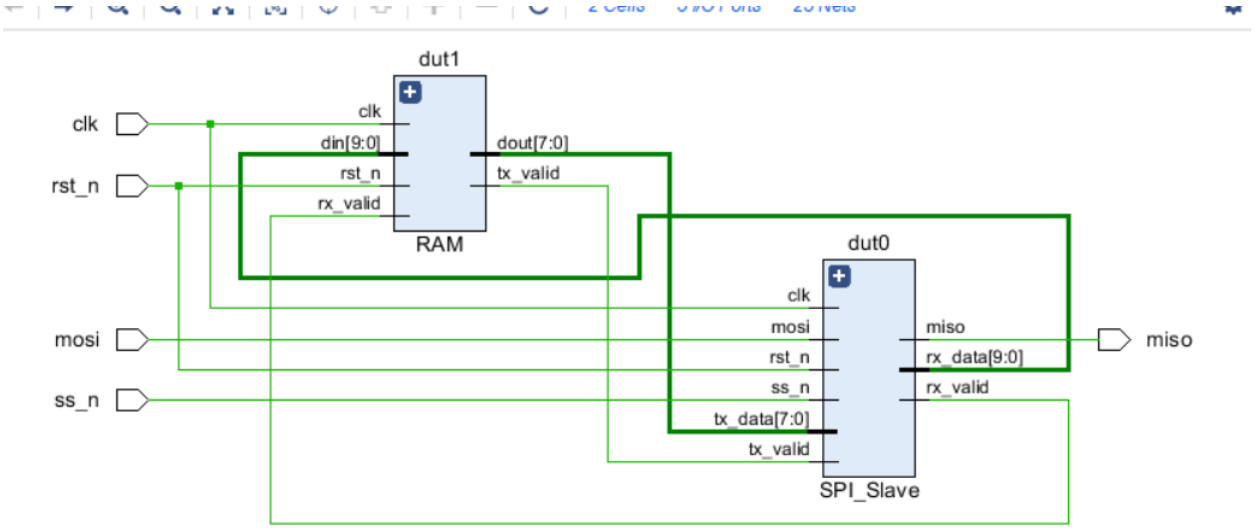
```
10
11  ## Switches
12  set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {rst_n}]
13  set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {ss_n}]
14  set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {mosi}]
15  #set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
16  #set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
17  #set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {sw[5]}]
18  #set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {sw[6]}]
```

```
9
0   ## LEDs
1   set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {miso}]
2   #set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
```

We will try Elaboration, Synthesis and Implementation using different encoding techniques.

Gray encoding:-

- Elaboration



[filemgmt 20-348] Importing the appropriate files for fileset: 'sources_1'

Elaborated Design (11 infos)

General Messages (11 infos)

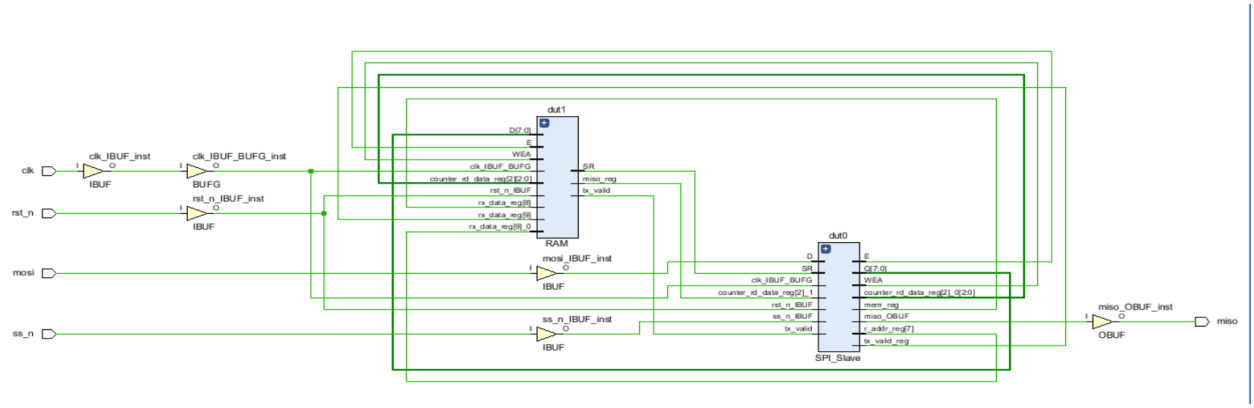
[Synth 8-5534] Detected attribute (fsm_encoding = "gray" *) [SPI_top_module.v:13]*

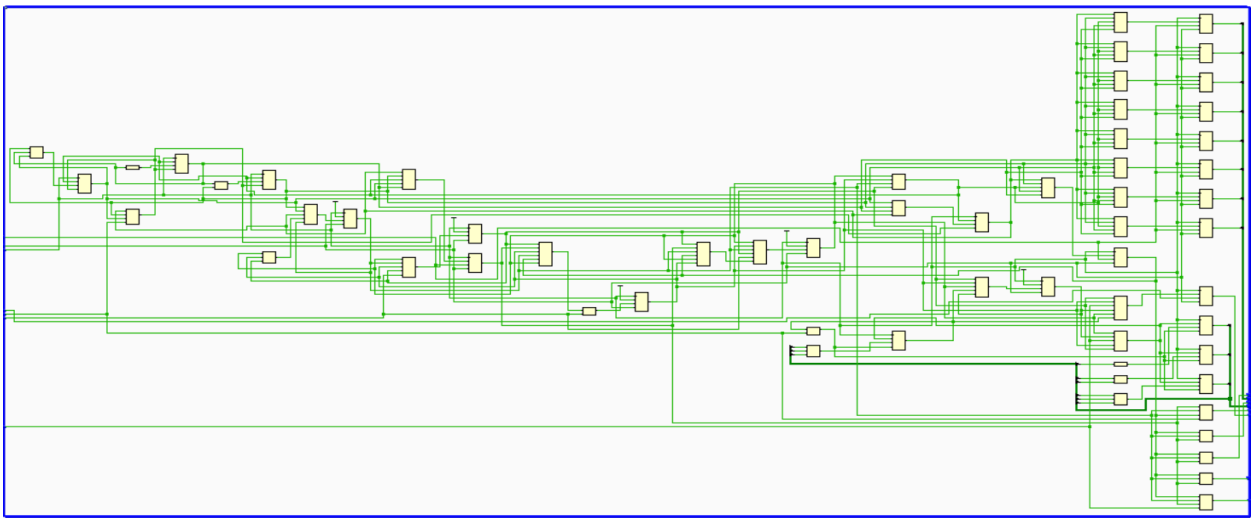
[Synth 8-6157] synthesizing module 'SPI_top_module' [SPI_top_module.v:1] (2 more like this)

[Synth 8-6155] done synthesizing module 'SPI_Slave' (1#1) [SPI_Slave.v:1] (2 more like this)

[Device 21-403] Loading part xc7a35ticpg236-1L

- Synthesis
  - Schematic





  - Messages

Synthesis (30 infos)

- [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7a35ti'
- [Synth 8-6157] synthesizing module 'SPI_top_module' [SPI_top_module.v:1] (2 more like this)
- [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *) [SPI_Slave.v:23]
- [Synth 8-6155] done synthesizing module 'SPI_Slave' (1#1) [SPI_Slave.v:1] (2 more like this)
- [Device 21-403] Loading part xc7a35ticpg236-1L
- [Project 1-236] Implementation specific constraints were found while reading constraint file [C:/Users/elnoor/Desktop/SPI/project_1.srcs/constrs_1/imports/SPI/Constraints_basys3.xdc]. These constraints will be ignored for synthesis but will be used in implementation. Impacted constraints are listed in the file [.Xil/SPI_top_module_propImpl.xdc].
  Resolution: To avoid this warning, move constraints listed in ['Undefined'] to another XDC file and exclude this new file from synthesis with the used_in_synthesis property (File Properties dialog in GUI) and re-run elaboration/synthesis.
- [Synth 8-802] inferred FSM for state register 'cs_reg' in module 'SPI_Slave'
- [Synth 8-5544] ROM 'rx_valid' won't be mapped to Block RAM because address size (3) smaller than threshold (5) (3 more like this)
- [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'gray' in module 'SPI_Slave'

  - Utilization

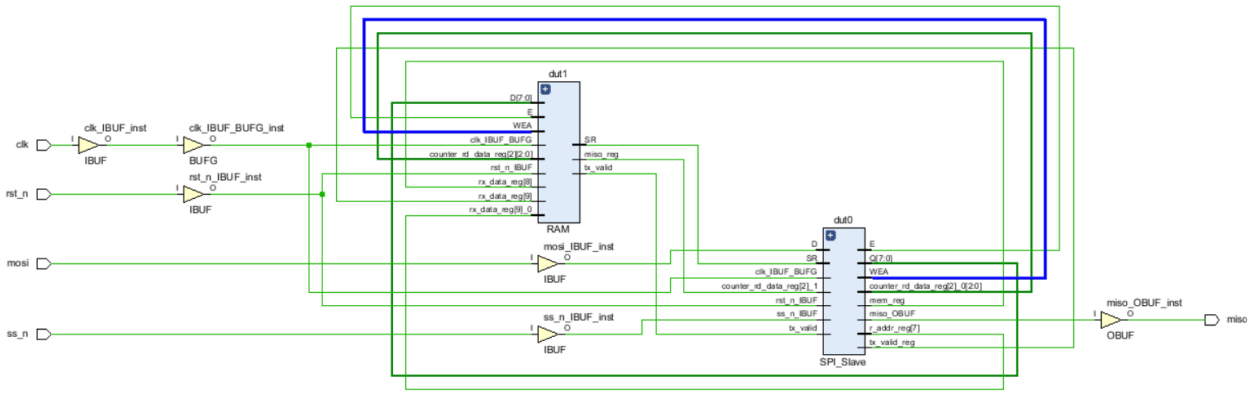
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_top_module	34	41	1	0.5	5	1
dut0 (SPI_Slave)	31	24	0	0	0	0
dut1 (RAM)	3	17	1	0.5	0	0

- Timing

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.261 ns	Worst Hold Slack (WHS):	0.142 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:	101	Total Number of Endpoints:	101	Total Number of Endpoints:	44

All user specified timing constraints are met.

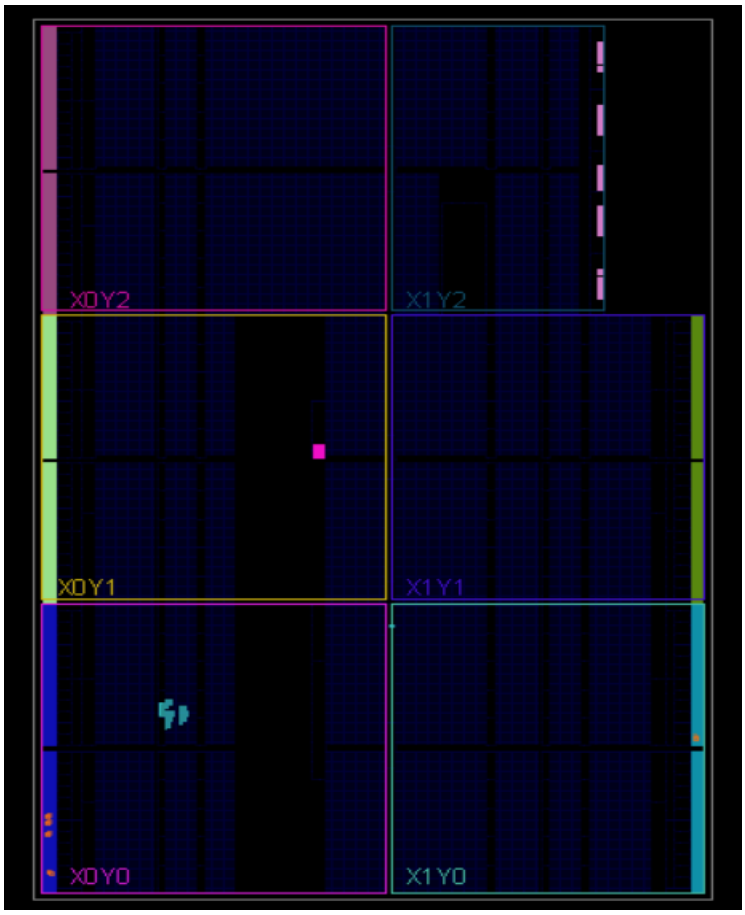
- Critical path



- Synthesis summary (encoding technique)

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

- Implementation
  - Device



  - Utilization

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)
N SPI_top_module	35	41	1	17	35	10	0.5	5	1
dut0 (SPI_Slave)	31	24	0	16	31	9	0	0	0
dut1 (RAM)	4	17	1	6	4	0	0.5	0	0

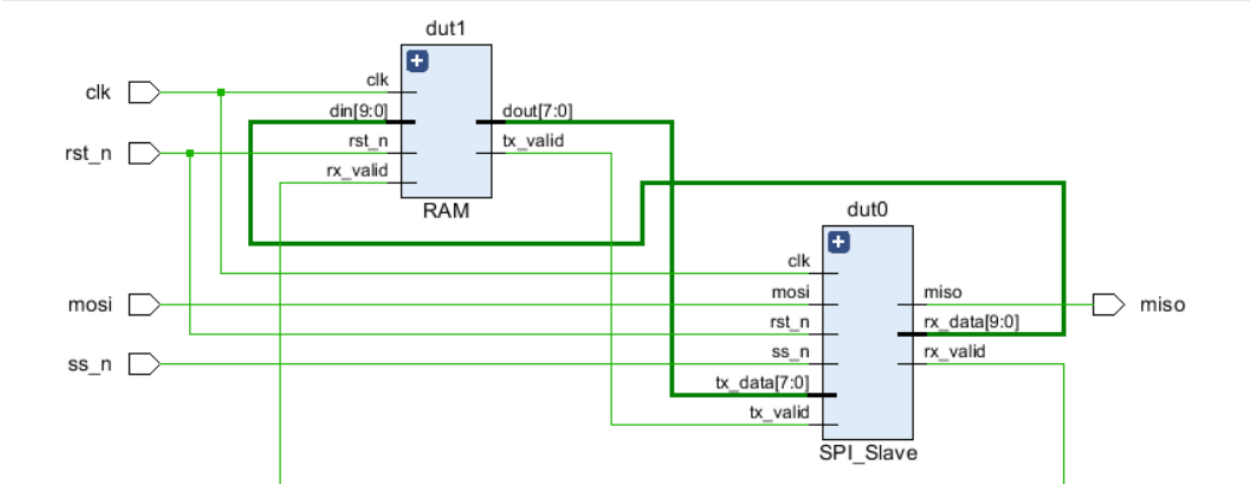
  - Timing

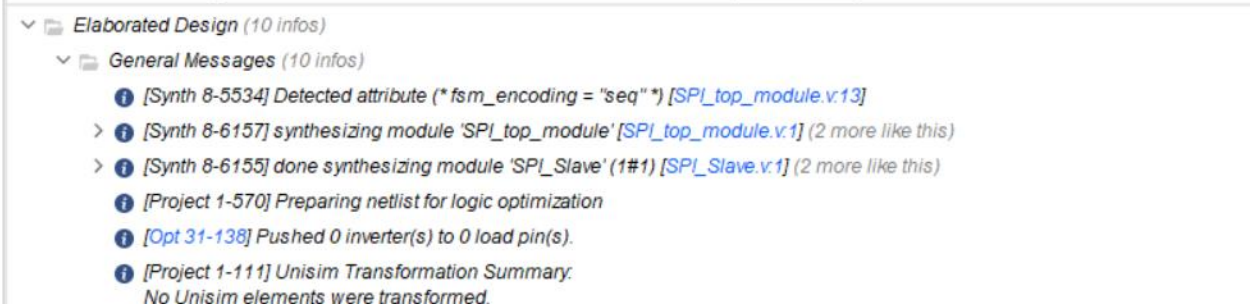
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.049 ns	Worst Hold Slack (WHS):	0.110 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	102	Total Number of Endpoints:	102	Total Number of Endpoints:	44

All user specified timing constraints are met.

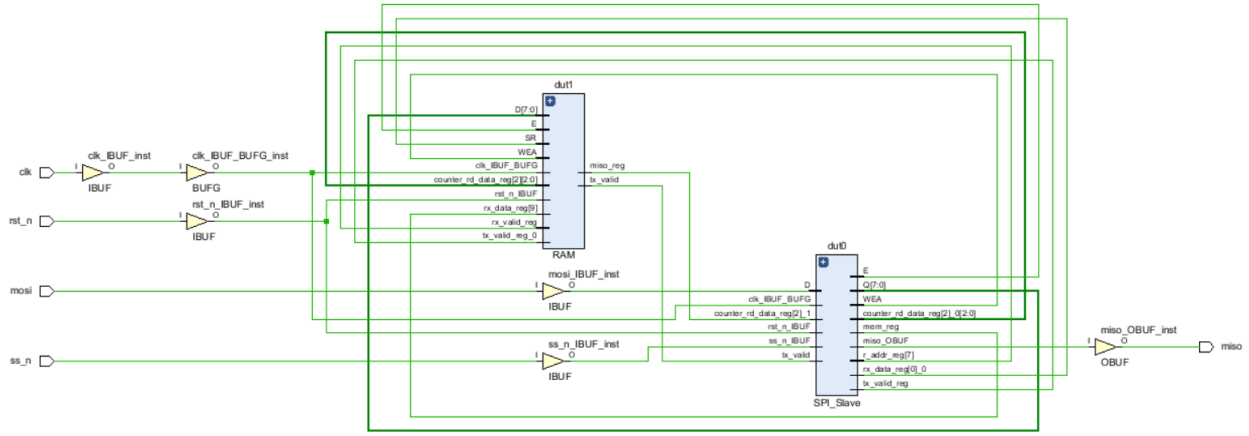
Seq encoding:-

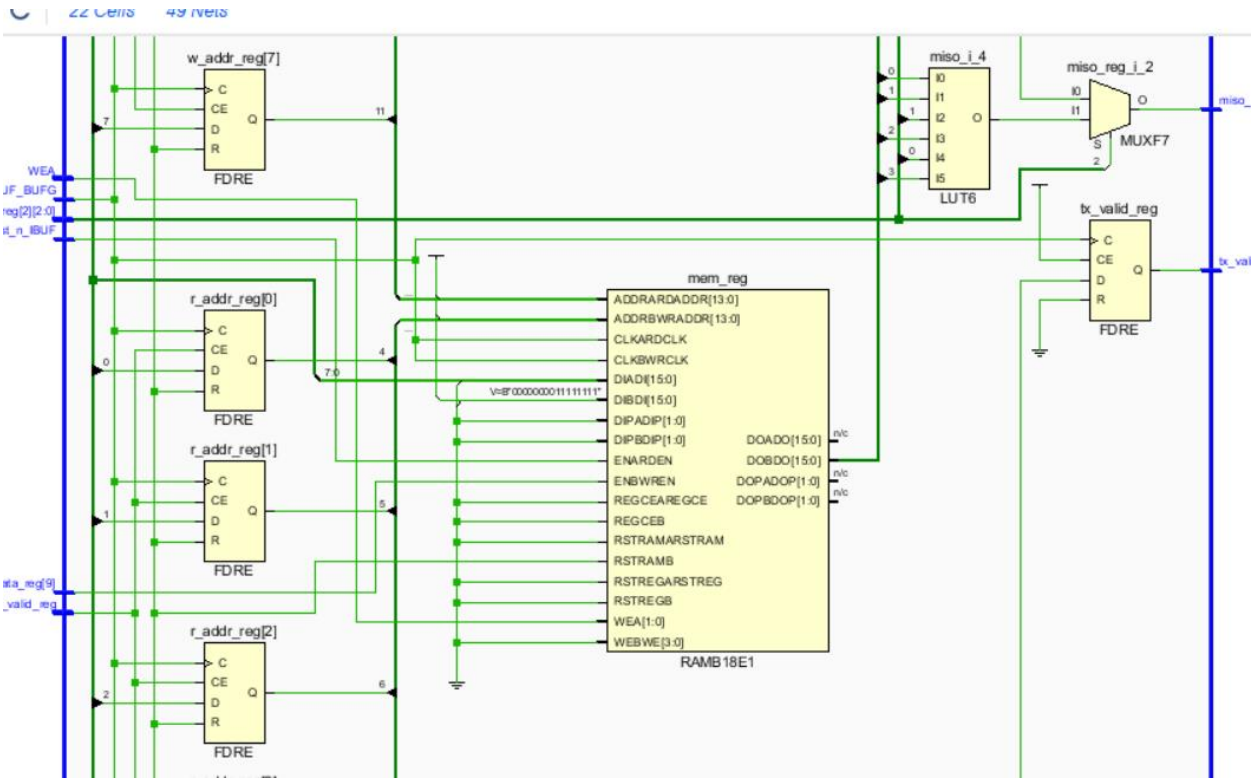
- Elaboration

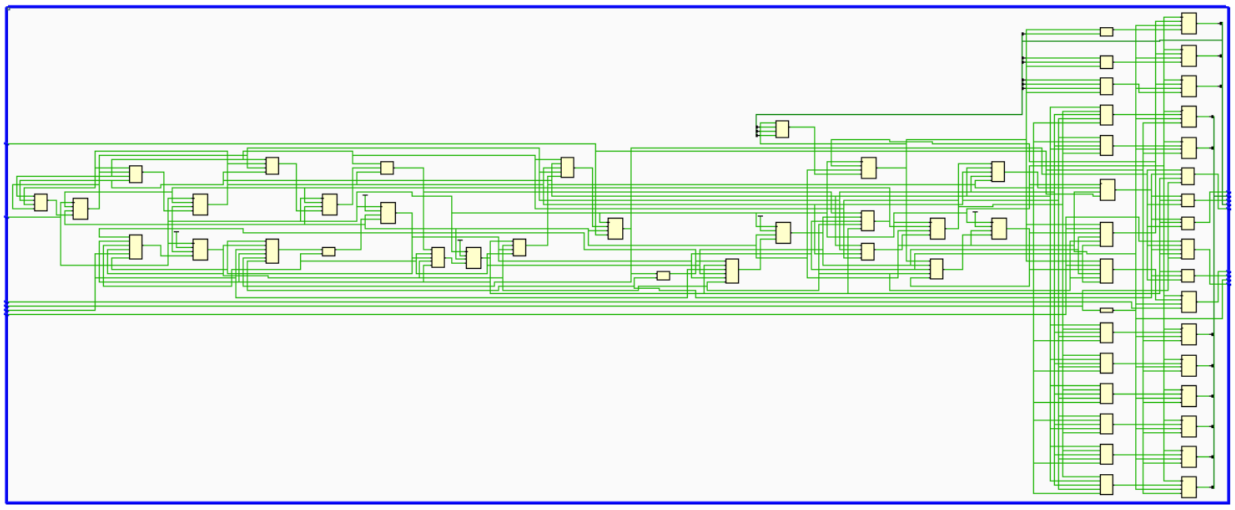




- Synthesis
- Schematic







- Messages

Synthesized Design (6 infos)
General Messages (6 infos)
[Netlist 29-17] Analyzing 6 Unisim elements for replacement
[Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
[Project 1-479] Netlist was created with Vivado 2018.2
[Project 1-570] Preparing netlist for logic optimization
[Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
[Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

- Utilization

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_top_module	32	41	1	0.5	5	1
dbg_hub (dbg_hub_CV)	0	0	0	0	0	0
dut0 (SPI_Slave)	30	24	0	0	0	0
dut1 (RAM)	2	17	1	0.5	0	0

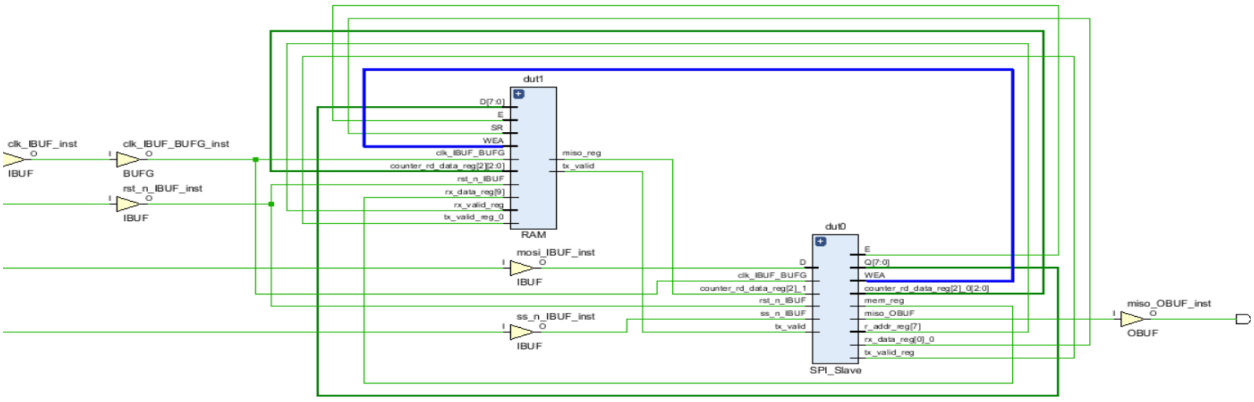
- Timing

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.261 ns	Worst Hold Slack (WHS):	0.142 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:	93	Total Number of Endpoints:	93	Total Number of Endpoints:	44

All user specified timing constraints are met

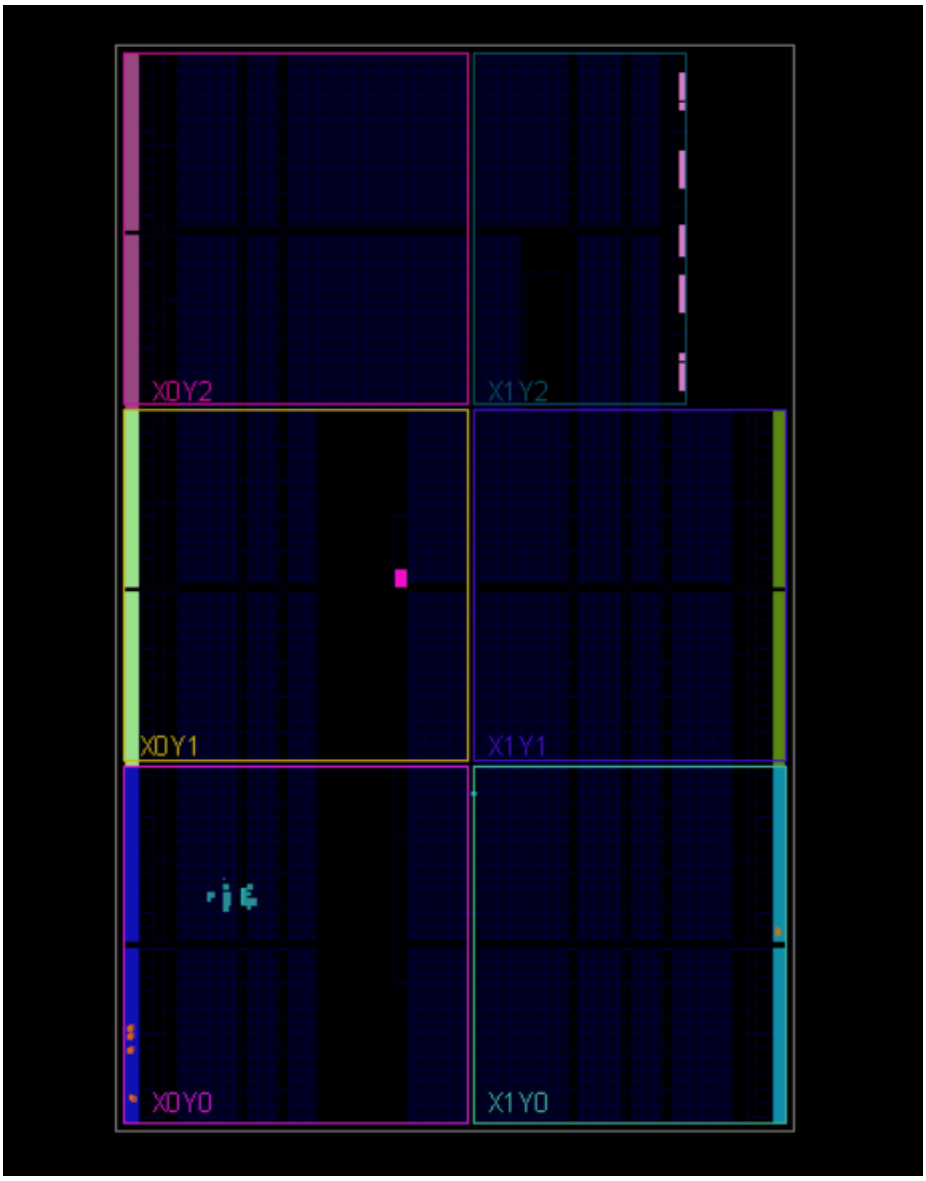
- Critical path

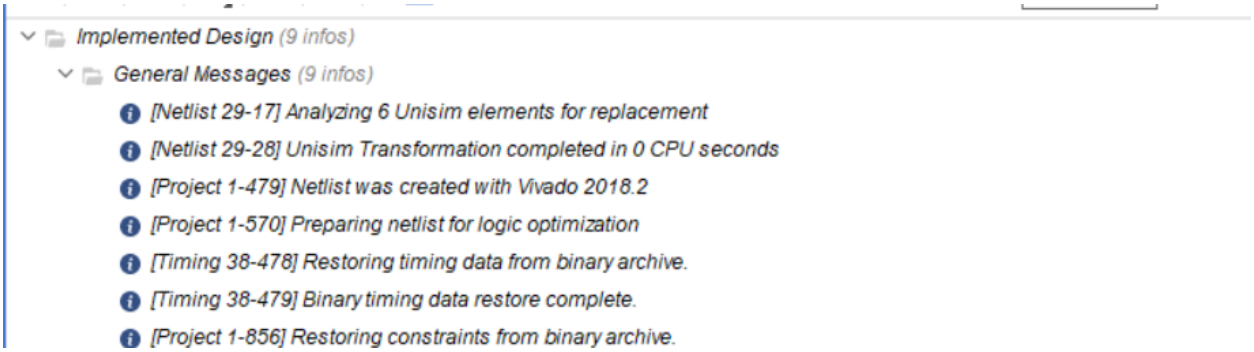


- Synthesis summary (encoding technique)

```
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
```

- Implementation
  - Device



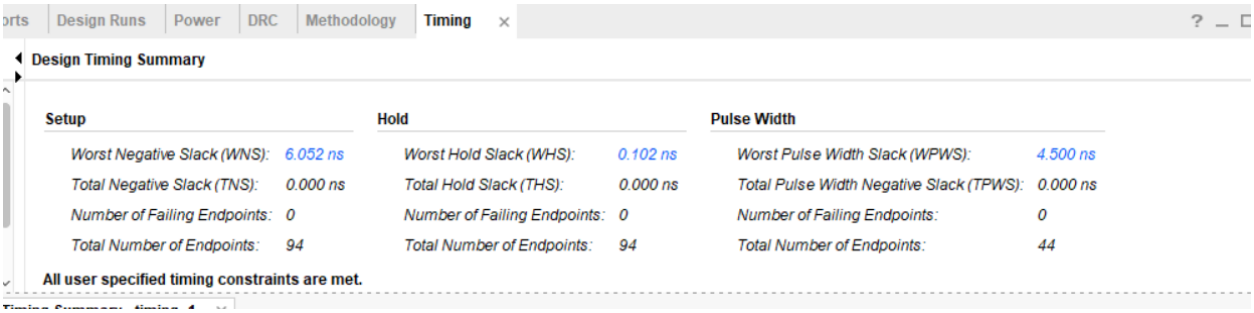


- Utilization

Hierarchy

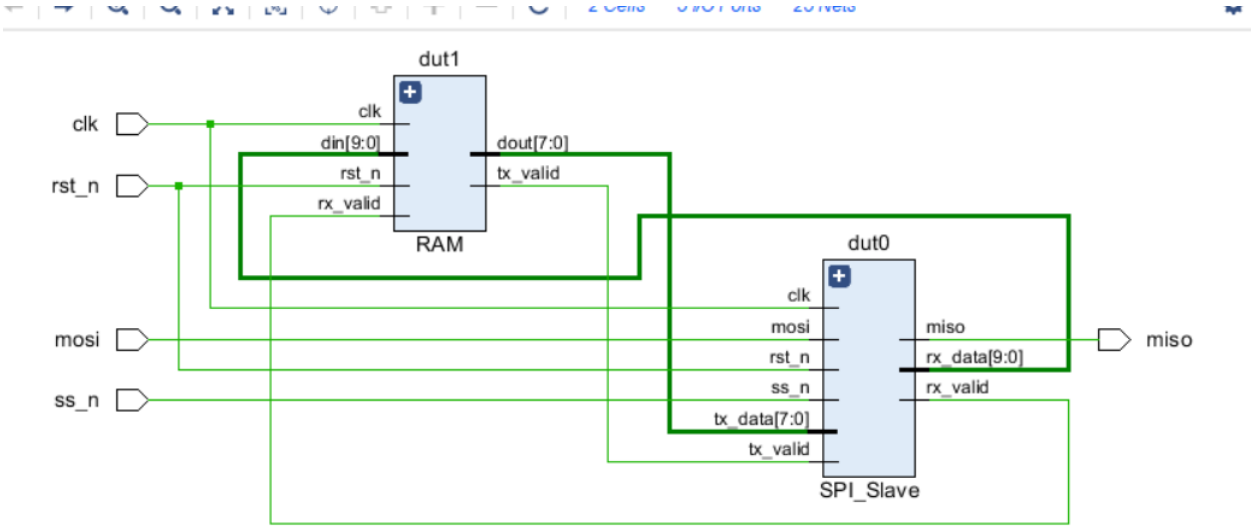
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_top_module	32	41	1	0.5	5	1
dut0 (SPI_Slave)	30	24	0	0	0	0
dut1 (RAM)	2	17	1	0.5	0	0

- Timing



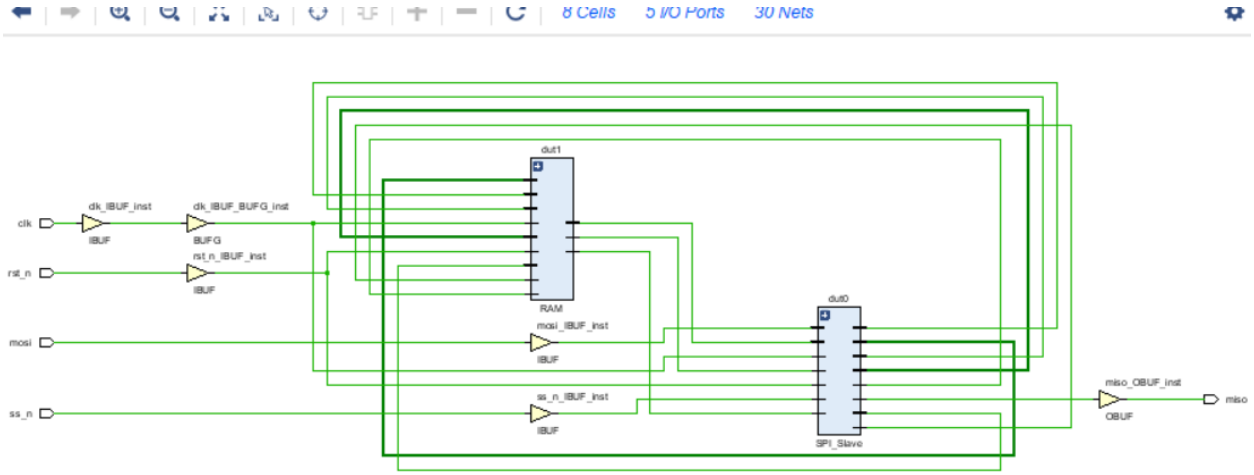
One_hot encoding:-

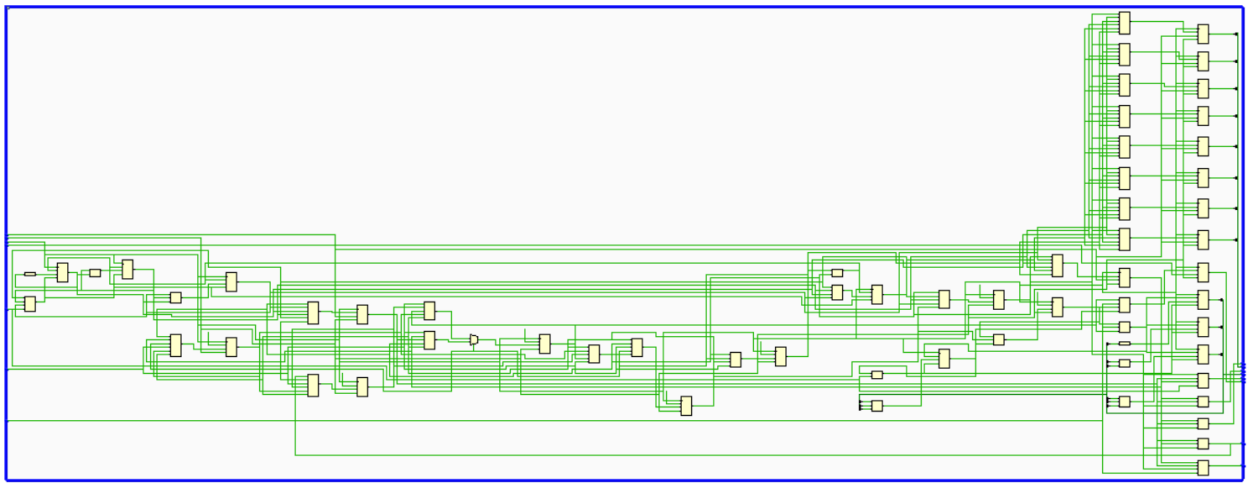
- Elaboration



- Synthesis

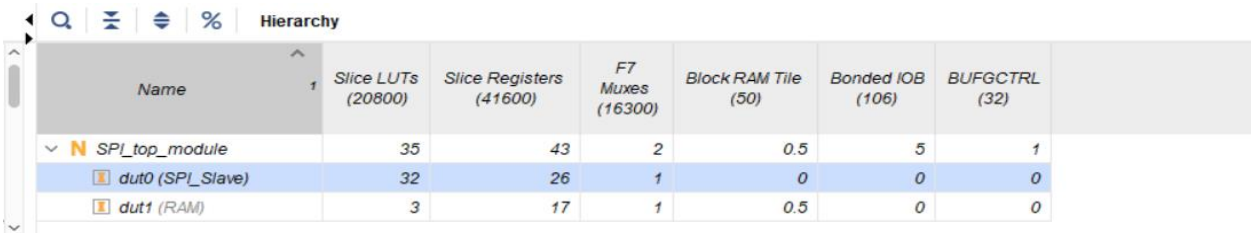
- Schematic





- Messages

- Utilization



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_top_module	35	43	2	0.5	5	1
dut0 (SPI_Slave)	32	26	1	0	0	0
dut1 (RAM)	3	17	1	0.5	0	0

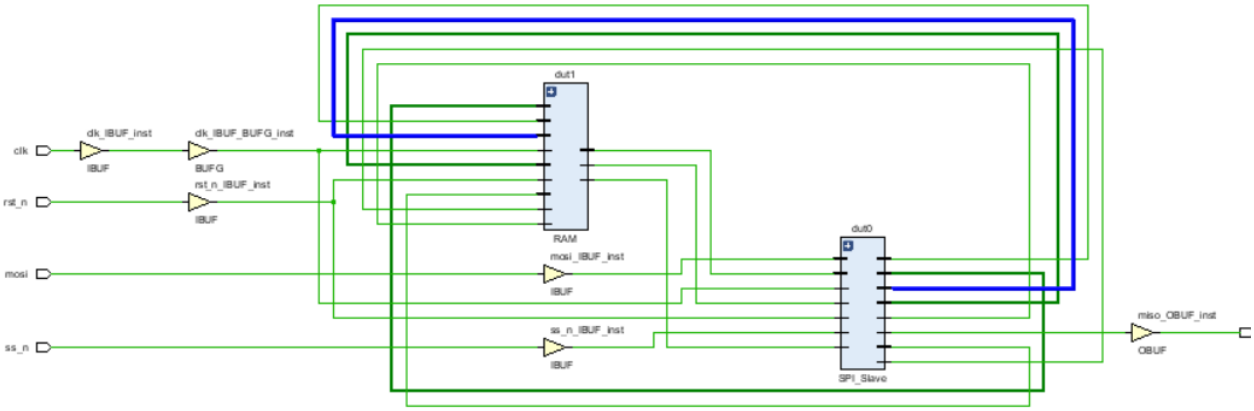
- Timing

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.261 ns	Worst Hold Slack (WHS):	0.142 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:	103	Total Number of Endpoints:	103	Total Number of Endpoints:	46

All user specified timing constraints are met.

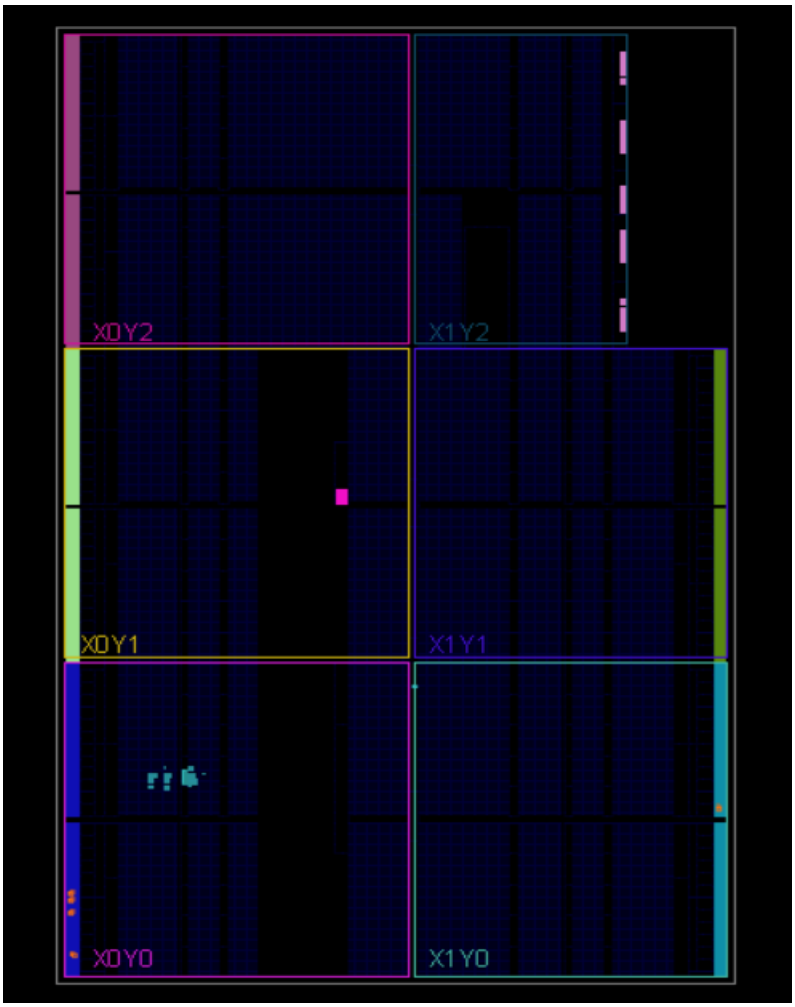
- Critical path



- Synthesis summary (encoding technique)

```
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'
-----------------------------------------------------------------------------------------------
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:32 ; elapsed = 00:00:35 . Memory (MB): peak = 765.305 ; ga
-----------------------------------------------------------------------------------------------
```

- Implementation
  - Device



Implemented Design (10 infos)

General Messages (10 infos)

- *[Netlist 29-17] Analyzing 7 Unisim elements for replacement*
- *[Netlist 29-28] Unisim Transformation completed in 0 CPU seconds*
- *[Project 1-479] Netlist was created with Vivado 2018.2*
- *[Device 21-403] Loading part xc7a35ticpg236-1L*
- *[Project 1-570] Preparing netlist for logic optimization*
- *[Timing 38-478] Restoring timing data from binary archive.*
- *[Timing 38-479] Binary timing data restore complete.*
- *[Project 1-856] Restoring constraints from binary archive.*
- *[Project 1-853] Binary constraint restore complete.*
- *[Project 1-111] Unisim Transformation Summary:*
  No Unisim elements were transformed.

- Utilization

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)
N SPI_top_module	36	43	2	19	36	12	0.5	5	1
dut0 (SPI_Slave)	32	26	1	16	32	11	0	0	0
dut1 (RAM)	4	17	1	6	4	0	0.5	0	0

- Timing

Setup

Worst Negative Slack (WNS):	6.212 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	104

Hold

Worst Hold Slack (WHS):	0.070 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	104

Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	46

All user specified timing constraints are met.

Comparison between techniques

Technique	Setup slack	Hold slack
Gray	6.049 ns	0.110 ns
Seq	6.052 ns	0.102 ns
One_hot	6.212 ns	0.070 ns

So as we considered gray encoding gives the best in the worst hold slack and one_hot encoding gives the best in the worst setup slack.