Diploma Project

SPI Slave with Single Port RAM

Digital Ultras Team

⇨ ***Team members:***

1) Shehab Eldeen Khaled Mabrouk
2) Mohamed Sameh Mohamed Kamel
3) shurooq mohammad alshal

- ***Design Code:***

1) SPI_Slave:

```
module SPI_Slave
(clk,rst_n,ss_n,MOSI,tx_valid,tx_data,MISO,rx_data,rx_valid);
    //FSM states
    parameter IDLE = 3'b000;
    parameter CHK_CMD = 3'b001;
    parameter WRITE = 3'b010;
    parameter READ_ADD = 3'b011;
    parameter READ_DATA = 3'b100;

    // signal inout
    input clk , rst_n , ss_n , MOSI , tx_valid;
    input [7:0] tx_data;
    output reg MISO , rx_valid;
    output reg [9:0] rx_data;
    (* fsm_encoding = "one_hot"*) //  or any other encoding type
    reg [2:0] cs,ns;
    reg [3:0] count_s_to_p , count_p_to_s;
    reg address_recived;

    // 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: begin
                if (!ss_n) begin
                    ns = CHK_CMD;
                end
                else begin
                    ns = IDLE;
                end
            end
```

```
            CHK_CMD: begin
                if (!ss_n) begin
                    if (!MOSI) begin  //write
                        ns = WRITE;
                    end
                    else if (MOSI) begin  //read
                        if (address_recived) begin
                            ns = READ_DATA;
                        end
                        else begin
                            ns = READ_ADD;
                        end
                    end
                end
                else begin
                    ns = IDLE;
                end
            end

            WRITE: begin
                if (!ss_n) begin
                    ns = WRITE;
                end
                else begin
                    ns = IDLE;
                end
            end

            READ_ADD: begin
                if (!ss_n) begin
                    ns = READ_ADD;
                end
                else begin
                    ns = IDLE;
                end
            end

            READ_DATA: begin
                if (!ss_n) begin
                    ns = READ_DATA;
                end
                else begin
                    ns = IDLE;
                end
            end
```

```
            default: ns = IDLE;
        endcase
    end

    // output logic
    always @(posedge clk) begin
        if(!rst_n) begin
            MISO <= 0;
            rx_data <= 0;
            rx_valid <= 0;
            count_s_to_p <= 1;
            count_p_to_s <= 1;
            address_recived <= 0;
        end
        else begin
            case (cs)
                WRITE: begin
                    if(count_s_to_p == 10) begin
                        rx_valid <= 1;
                        count_s_to_p <= 1;
                    end
                    else if (count_s_to_p != 10) begin
                        rx_valid <= 0;
                    end
                    if (count_s_to_p < 11) begin
                        rx_data[10-count_s_to_p] <= MOSI;
                        count_s_to_p <= count_s_to_p + 1;
                    end
                end

                READ_ADD: begin
                    if(count_s_to_p == 10) begin
                        rx_valid <= 1;
                        count_s_to_p <= 1;
                    end
                    else if (count_s_to_p != 10) begin
                        rx_valid <= 0;
                    end
                    if (count_s_to_p < 11) begin
                        rx_data[10-count_s_to_p] <= MOSI;
                        count_s_to_p <= count_s_to_p + 1;
                    end
                    address_recived <= 1;
                end
```

```
                READ_DATA: begin
                    if (tx_valid) begin
                        if (count_p_to_s < 9) begin
                            MISO <= tx_data[8-count_p_to_s];
                            count_p_to_s <= count_p_to_s + 1;
                            count_s_to_p <= 1;
                        end
                        if (count_p_to_s == 8) begin
                            count_s_to_p <= 1;
                            count_p_to_s <= 1;
                        end
                    end

                    else begin
                        MISO <= 0;
                        if(count_s_to_p == 10) begin
                            rx_valid <= 1;
                            count_s_to_p <= 1;
                            count_p_to_s <= 1;
                        end
                        else if (count_s_to_p != 10) begin
                            rx_valid <= 0;
                        end
                        if (count_s_to_p < 11) begin
                            rx_data[10-count_s_to_p] <= MOSI;
                            count_s_to_p <= count_s_to_p + 1;
                            count_p_to_s <= 1;
                        end
                    end
                    address_recived <= 0;
                end
                default: begin
                    count_s_to_p <= 1;
                    count_p_to_s <= 1;
                    rx_valid <= 0;
                    MISO <= 0;
                end
            endcase
        end
    end
endmodule
```

2) ***RAM:***

```
module RAM(clk, rst_n, rx_valid, din, tx_valid, dout);
    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

    input clk, rst_n, rx_valid;
    input [ADDR_SIZE+1:0] din;

    output reg tx_valid;
    output reg [ADDR_SIZE-1:0] dout;
    wire [ADDR_SIZE-1:0] data;
    wire [1:0] signal;

    reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
    reg [ADDR_SIZE-1:0] wr_addr, rd_addr;

    assign signal = din[ADDR_SIZE+1:ADDR_SIZE];
    assign data = din[ADDR_SIZE-1:0];
    always @(posedge clk) begin
        if(~rst_n) begin
            tx_valid <= 0;
            dout <= 0;
            wr_addr <= 0;
            rd_addr <= 0;
        end
        else if (rx_valid) begin
            case(signal)
            2'b00: begin
                wr_addr <= data;
                tx_valid <= 0;
            end
            2'b01: begin
                mem[wr_addr] <= data;
                tx_valid <= 0;
            end
            2'b10: begin
                rd_addr <= data;
                tx_valid <= 0;
            end
            2'b11: begin
                dout <= mem[rd_addr];
                tx_valid <= 1;
            end
            default: tx_valid <= 0;
```

```
            endcase
        end
    end
endmodule
```

3) Top module (SPI_Wrapper):

```
module SPI_Wrapper (clk,rst_n,MOSI,ss_n,MISO);
    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;
    parameter IDLE = 3'b000;
    parameter CHK_CMD = 3'b001;
    parameter WRITE = 3'b010;
    parameter READ_ADD = 3'b011;
    parameter READ_DATA = 3'b100;

    input clk , rst_n , MOSI , ss_n;
    output MISO;
    wire rx_valid , tx_valid;
    wire [7:0] tx_data;
    wire [9:0] rx_data;

    SPI_Slave #(IDLE,CHK_CMD,WRITE,READ_ADD,READ_DATA)
SPI(clk,rst_n,ss_n,MOSI,tx_valid,tx_data,MISO,rx_data,rx_valid);
    RAM #(MEM_DEPTH,ADDR_SIZE) ram(clk, rst_n, rx_valid, rx_data, tx_valid,
tx_data);

endmodule
```

⇨ ***Testbench Code:***

```
module SPI_Wrapper_tb();

    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;
    parameter IDLE = 3'b000;
    parameter CHK_CMD = 3'b001;
    parameter WRITE = 3'b010;
    parameter READ_ADD = 3'b011;
    parameter READ_DATA = 3'b100;

    // signal declaration
    reg clk , rst_n , MOSI , ss_n;
    wire MISO;

    integer i = 0;

    // Module instantiation
    SPI_Wrapper #(MEM_DEPTH,ADDR_SIZE,IDLE,CHK_CMD,WRITE,READ_ADD,READ_DATA)
SPI_Wrapper_DUT (clk,rst_n,MOSI,ss_n,MISO);

    // clock generation
    initial begin
        clk=0;
        forever
        #1 clk=~clk;
    end

    initial begin
        $readmemh("mem.dat" ,SPI_Wrapper_DUT.ram.mem);
        rst_n=0;  // initialize Design
        ss_n=1;
        MOSI=1;
        @(negedge clk);
        rst_n=1;
        ss_n=0;

        // Write Address 1
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=0;
```

```
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        ss_n=1;

        // Write Data 1
        @(negedge clk);
        ss_n=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        ss_n=1;

        // Read Address 1
        @(negedge clk);
        ss_n=0;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        ss_n=1;

        // Read Data 1
        @(negedge clk);
        ss_n=0;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
```

```
        MOSI=1;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        repeat(8) @(negedge clk);

        ss_n=1;
        @(negedge clk);

        // write address 2
        ss_n=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=1;
            @(negedge clk);
        end
        ss_n=1;
        @(negedge clk);

       // Write Data 2
        ss_n=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        ss_n=1;
        @(negedge clk);
```

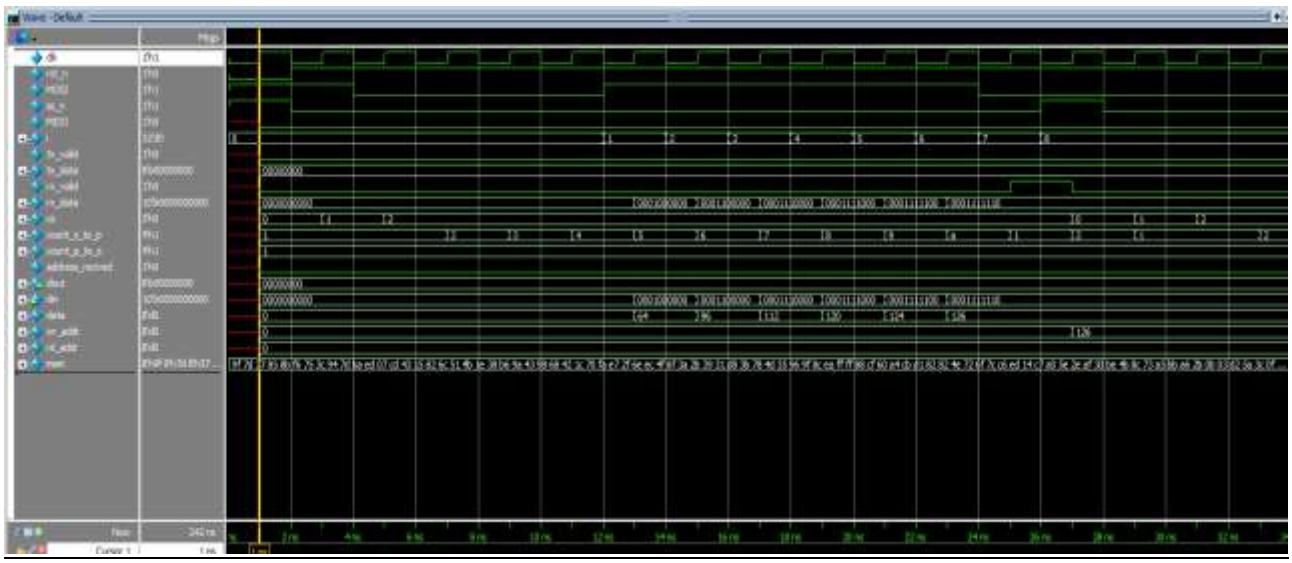
```
        // Read Address 2
        ss_n=0;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        MOSI=0;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        ss_n=1;
        @(negedge clk);

        // Read Data 2
        ss_n=0;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        MOSI=1;
        @(negedge clk);
        for(i=0;i<8;i=i+1) begin
            MOSI=$random;
            @(negedge clk);
        end
        repeat(8) @(negedge clk);
        ss_n=1;
        @(negedge clk);
        $stop;
    end

    endmodule
```

⇨ ***Do file:***

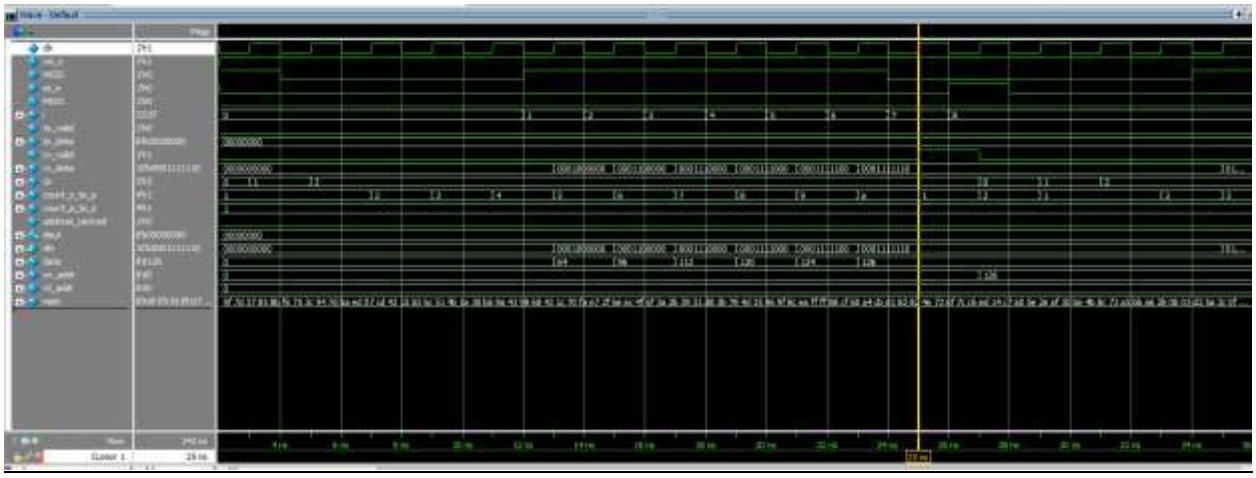
```
vlib work
vlog SPI_Slave.v RAM.v SPI_Wrapper.v SPI_Wrapper_tb.v
vsim -voptargs=+acc work.SPI_Wrapper_tb
add wave *
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/tx_valid
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/tx_data
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/rx_valid
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/rx_data
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/SPI/cs
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/SPI/count_s_to_p
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/SPI/count_p_to_s
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/SPI/address_recived
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/ram/dout
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/ram/din
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/ram/data
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/ram/wr_addr
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/ram/rd_addr
add wave /SPI_Wrapper_tb/SPI_Wrapper_DUT/ram/mem
run -all
#quit -sim
```

⇨ ***Simulation on Questasim:***

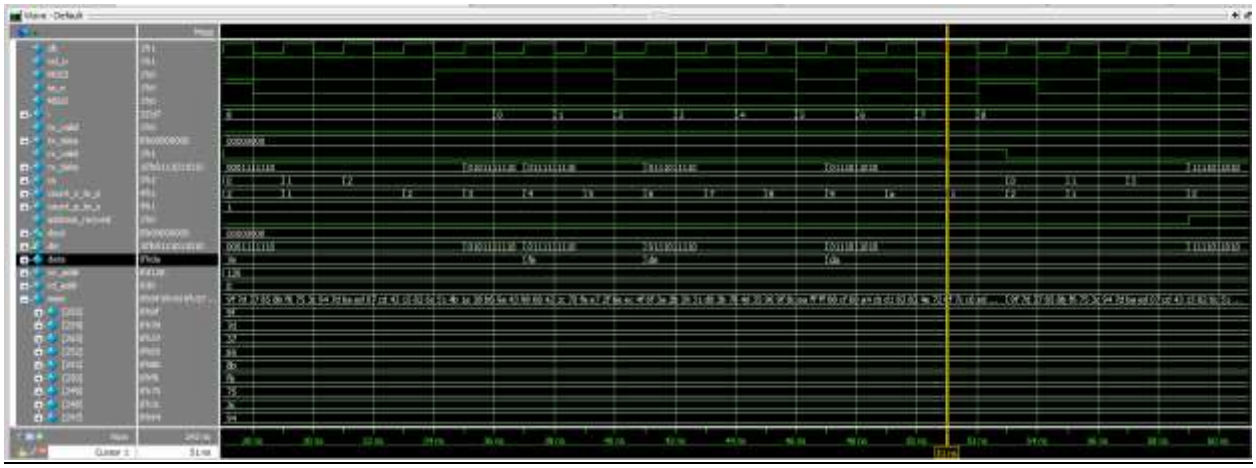
1) Reset case:

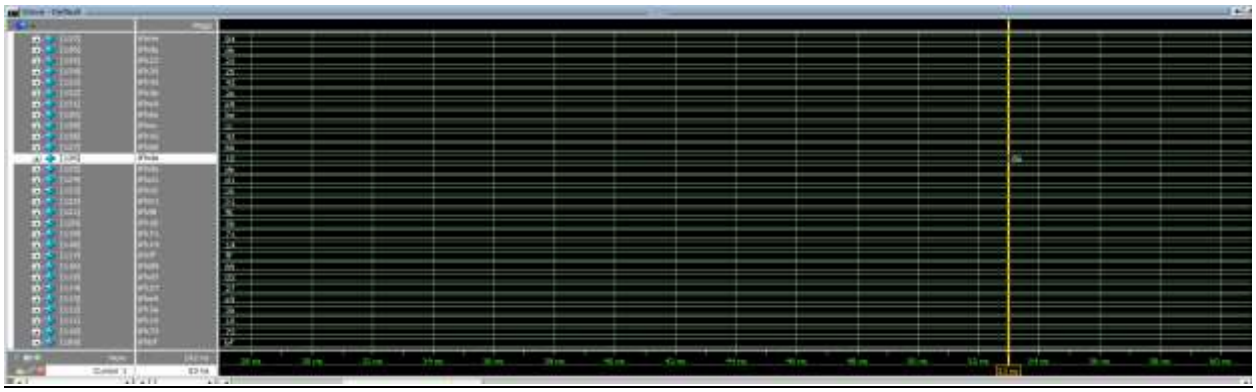


2) ***Write address case:***

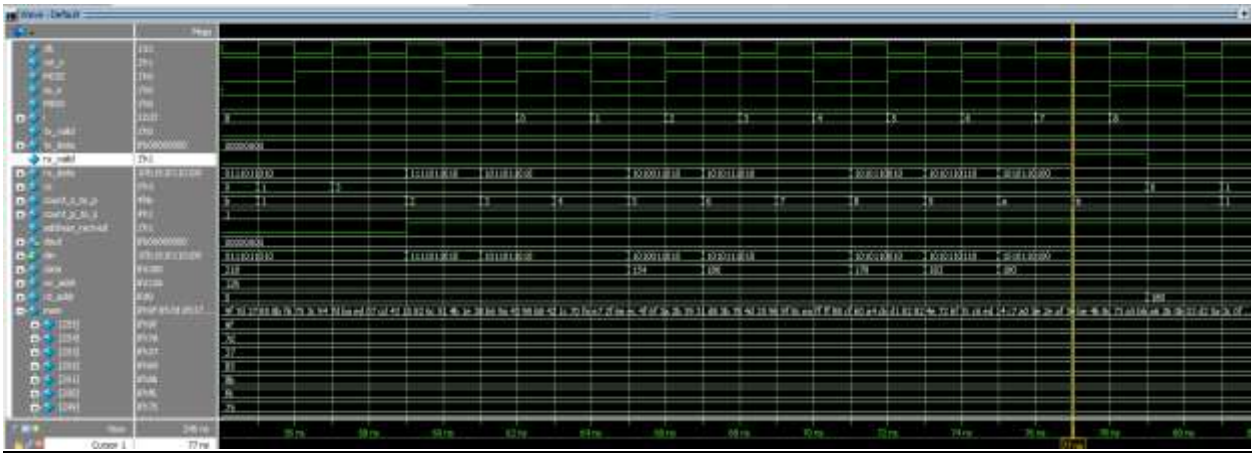


3) ***Write data case:***

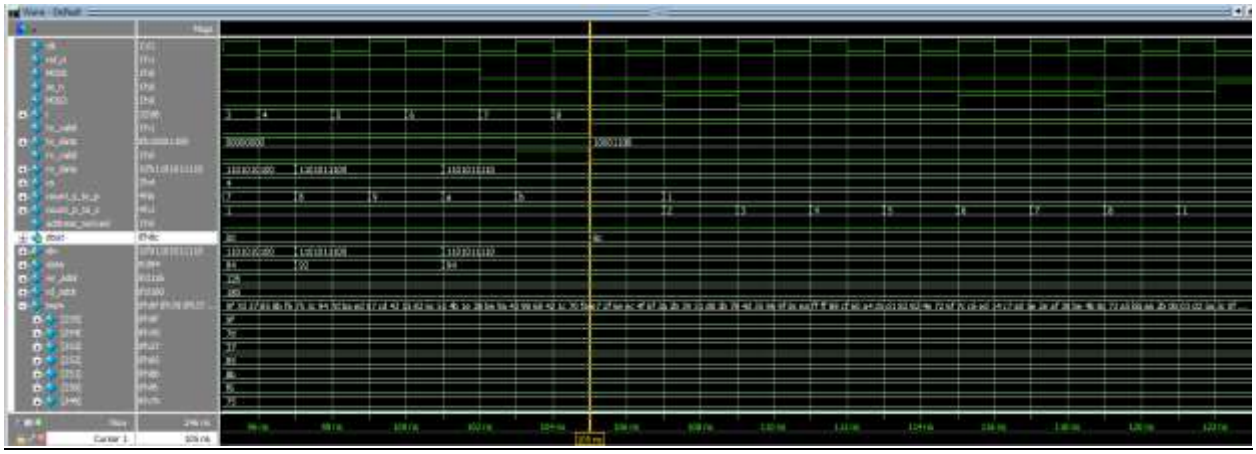


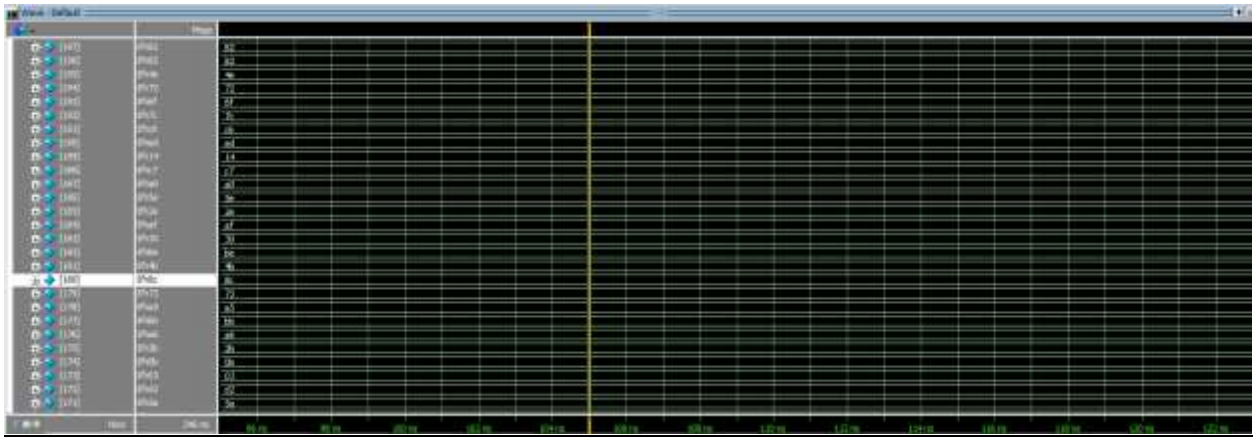


4) Read address case:



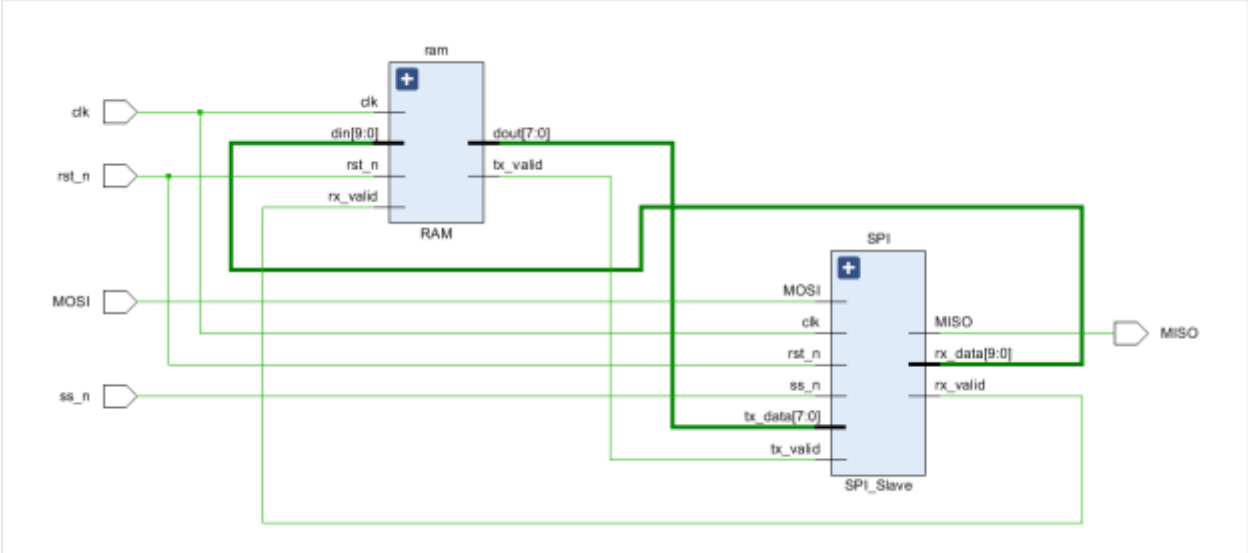
5) Read data case:



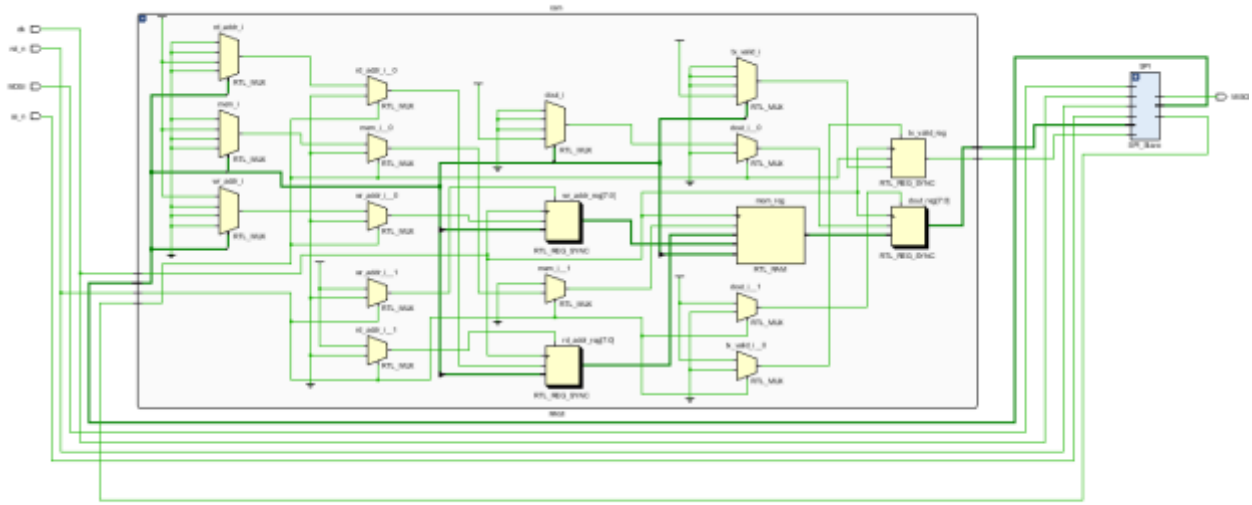


⇨ ***Elaboration:***

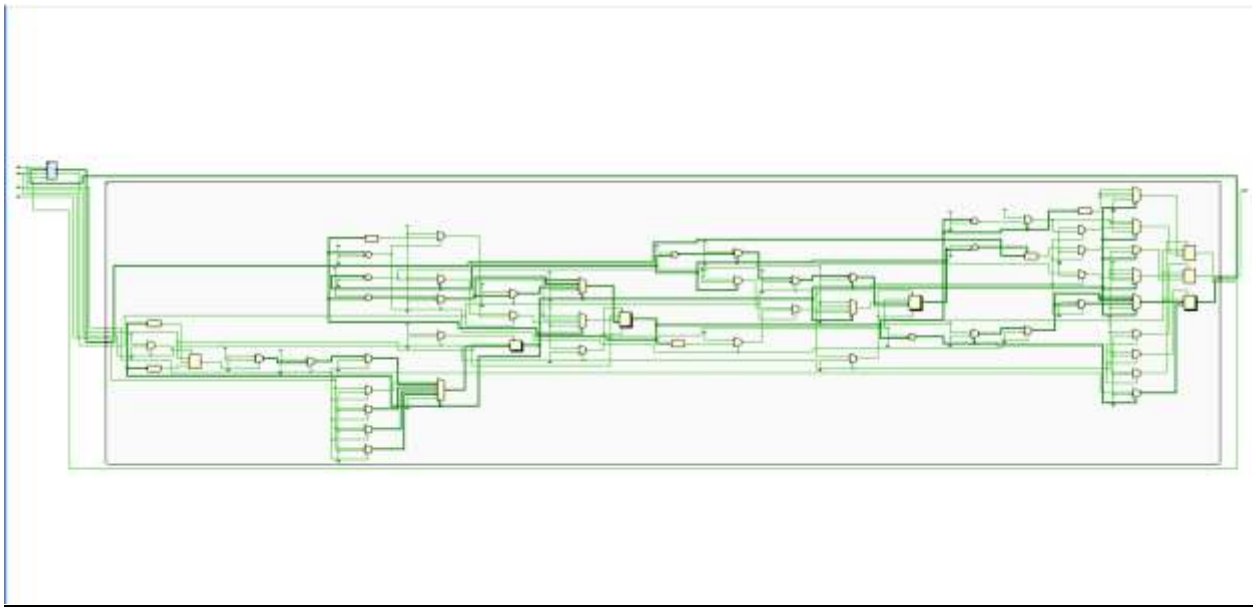
1) Schematic:



⇨ ***RAM***



⇨ ***SPI Slave:***



2) Message tab:

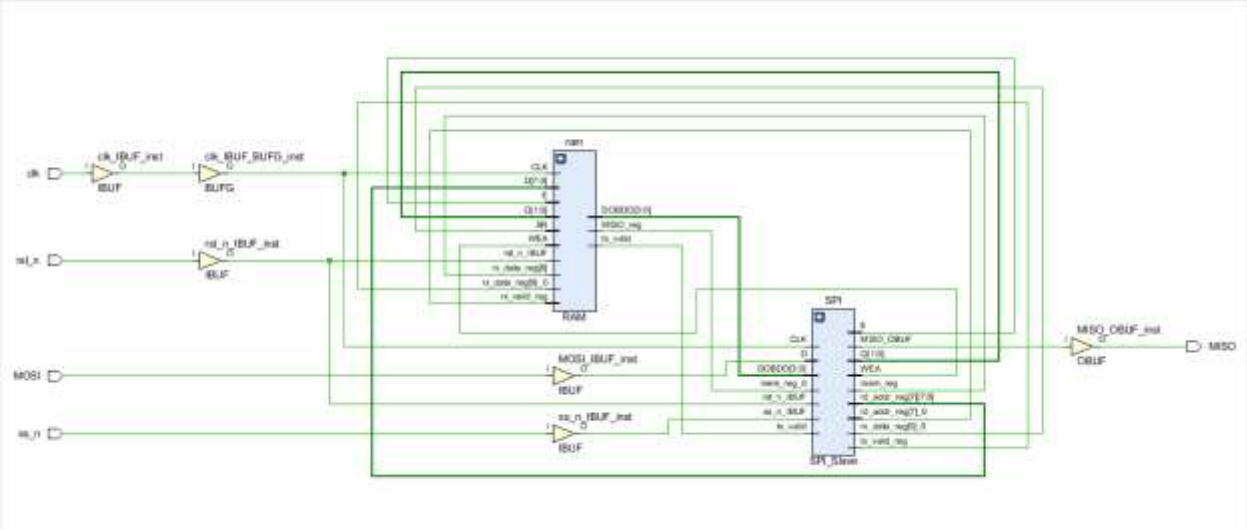
Tcl Console | **Messages** | Log | Reports | Design Runs

Info (15) | Status (11) | Show All

- Vivado Commands (3 infos)
  - General Messages (3 infos)
    - [IP_Flow 19-234] Refreshing IP repositories
    - [IP_Flow 19-1704] No user IP repositories specified
    - [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2018.2/data/ip'.
- Elaborated Design (12 infos)
  - General Messages (12 infos)
    - [Synth 8-6157] synthesizing module 'SPI_Wrapper' [SPI_Wrapper.v:1] (2 more like this)
    - [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *) [SPI_Slave.v:15]
    - [Synth 8-226] default block is never used [RAM.v:26]
    - [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 1-570] 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.

⇨ ***RTL Synthesis for gray encoding:***

1) schematic



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

2) Message tab:

3) Timing report:

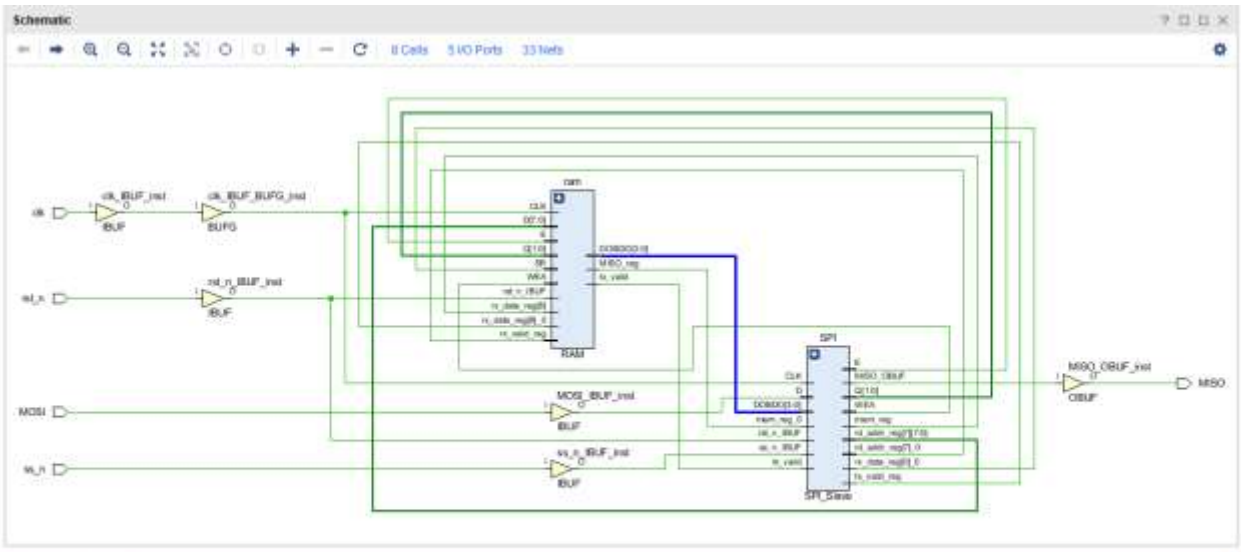
Runs | Timing | Debug

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.236 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.

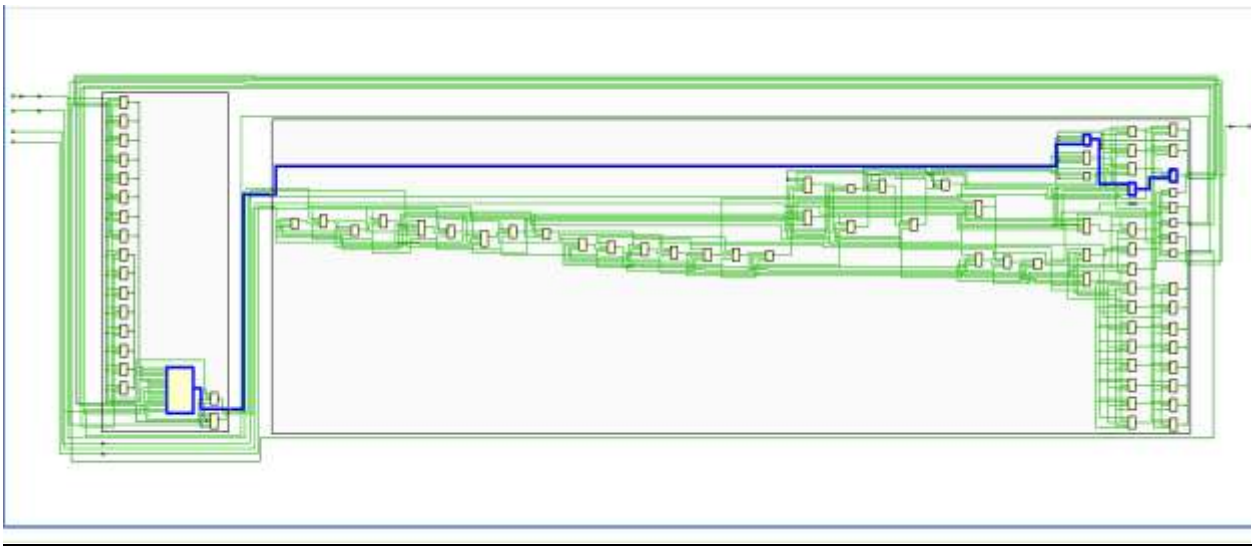
4) ***Critical path:***



Timing | Debug

Intra-Clock Paths - sys_clk_pin - Setup

	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	5.236	2	1	ram/mem_reg/CLKBWRCLK	SPI/MISO_reg/D	4.613	2.702	1.911	10.000	sys_clk_pin	sys_clk_pin	



5) ***Utilization report:***

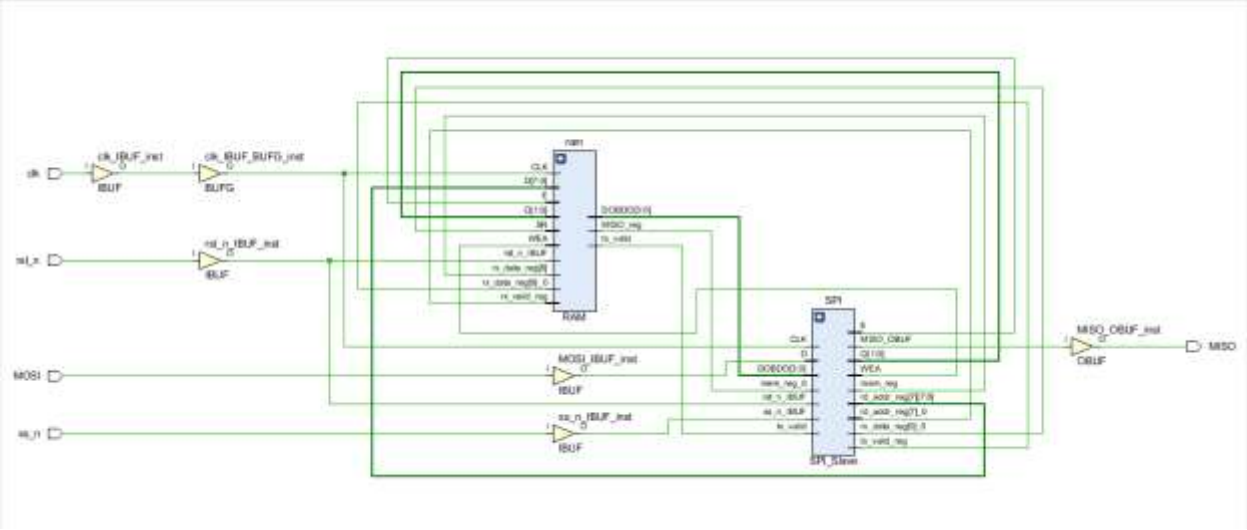
Design Runs | **Utilization** | Timing | Debug

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_Wrapper	36	41	0.5	5	1
ram (RAM)	1	17	0.5	0	0
SPI (SPI_Slave)	35	24	0	0	0

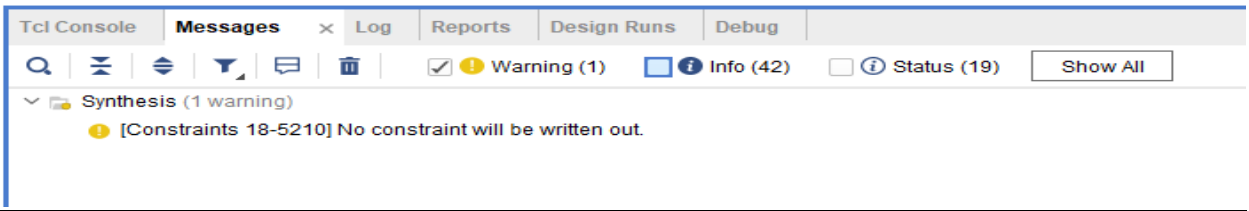
⇨ ***RTL Synthesis for one hot encoding:***

1) schematic



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

2) Message tab:



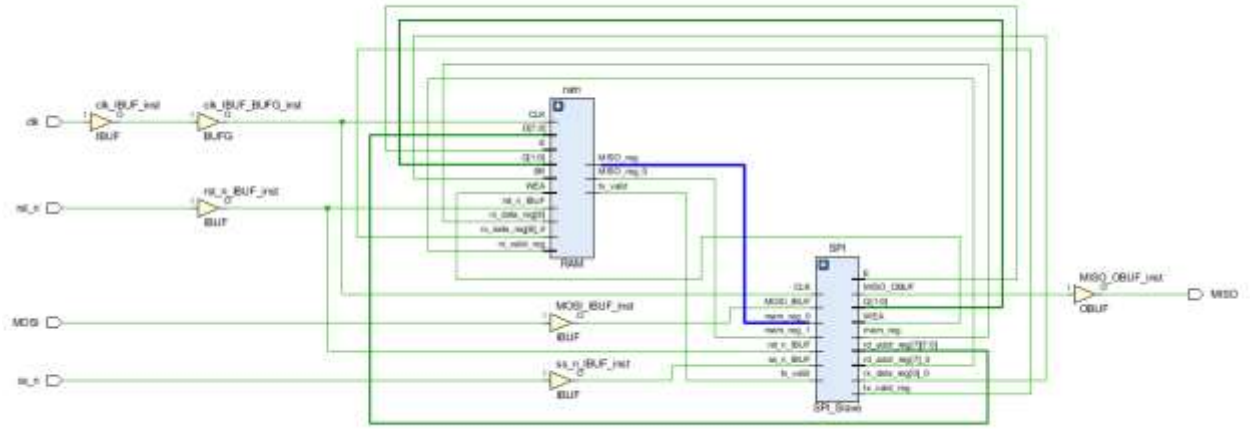
3) Timing report:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.445 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:	95	Total Number of Endpoints:	95	Total Number of Endpoints:	46

All user specified timing constraints are met.

4) ***Critical path:***

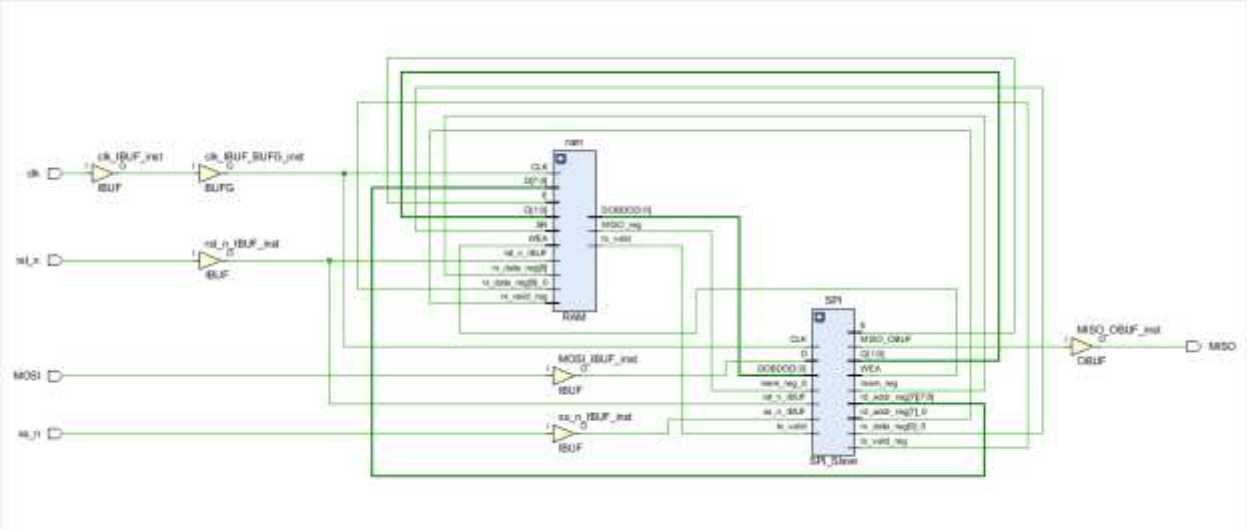


5) ***Utilization report:***

Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_Wrapper	38	43	0.5	5	1
ram (RAM)	2	17	0.5	0	0
SPI (SPI_Slave)	36	26	0	0	0

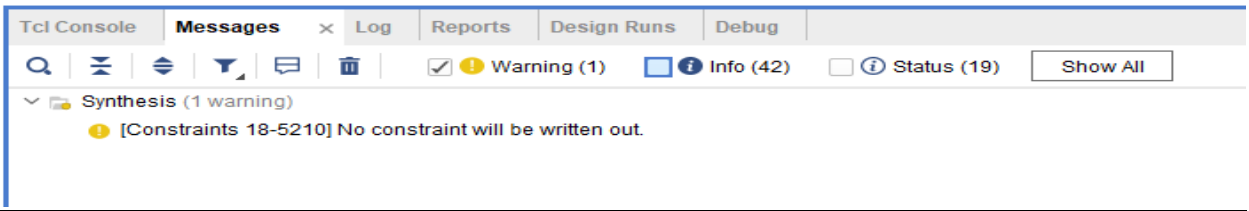
⇨ ***RTL Synthesis for sequential encoding:***

1) schematic



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

2) Message tab:

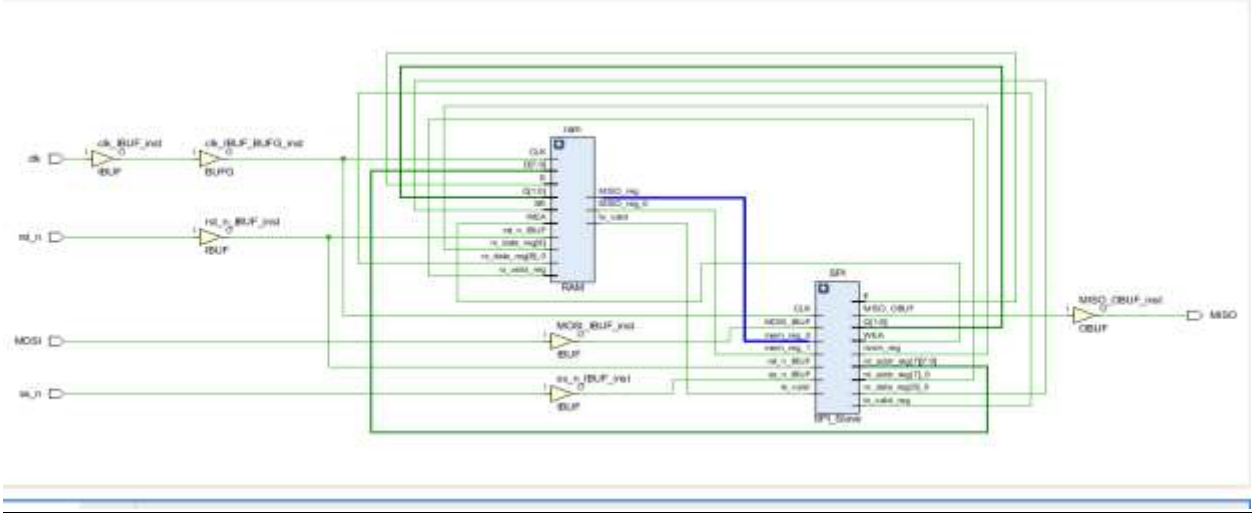


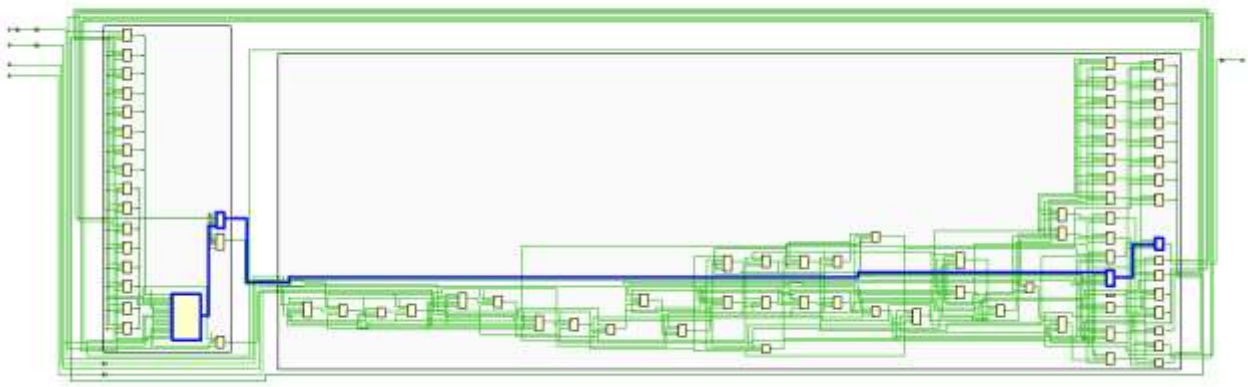
3) Timing report:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.445 ns	Worst Hold Slack (WHS):	0.144 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.

4) ***Critical path:***





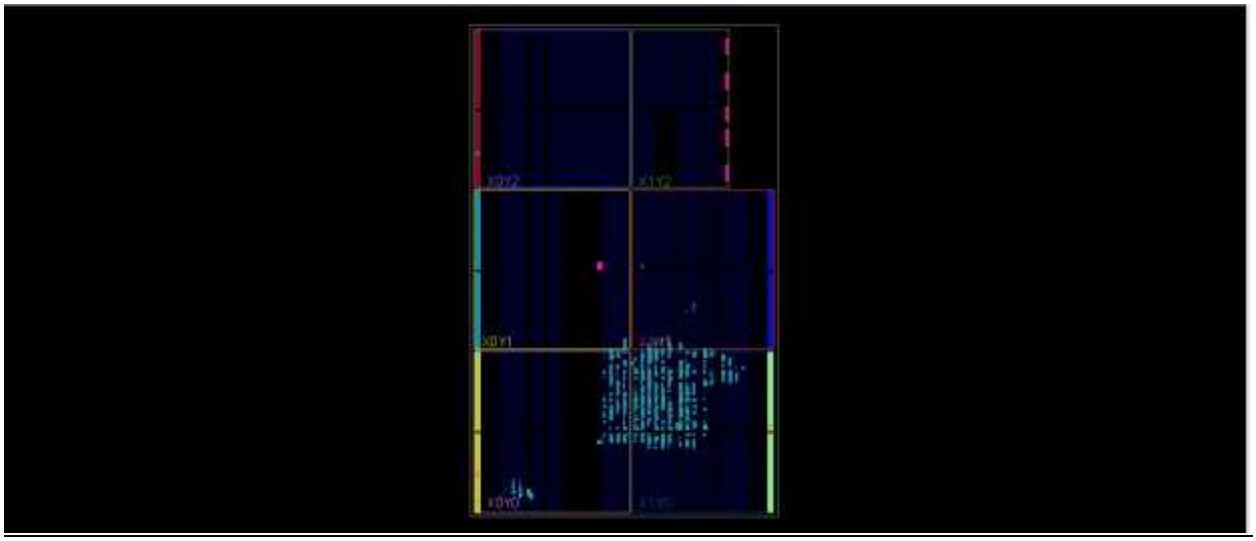
5) ***Utilization report:***

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_Wrapper	35	41	0.5	5	1
ram (RAM)	2	17	0.5	0	0
SPI (SPI_Slave)	33	24	0	0	0

⇨ ***Implementation for one hot encoding:***

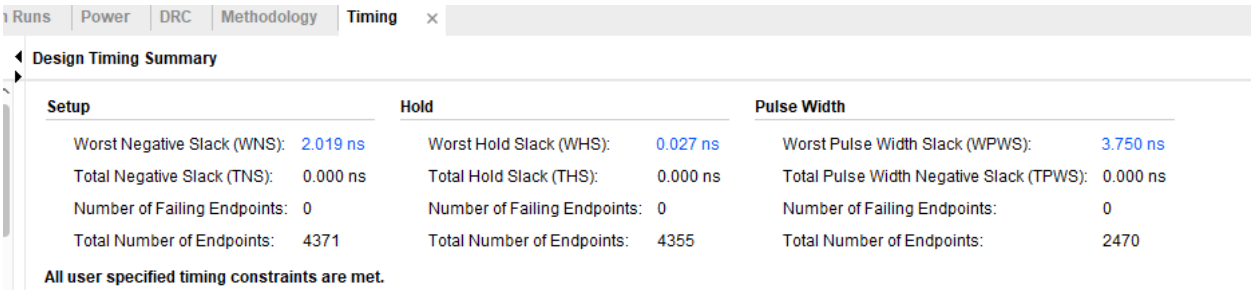
1) Device:



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

2) Message tab:

3) Timing report:

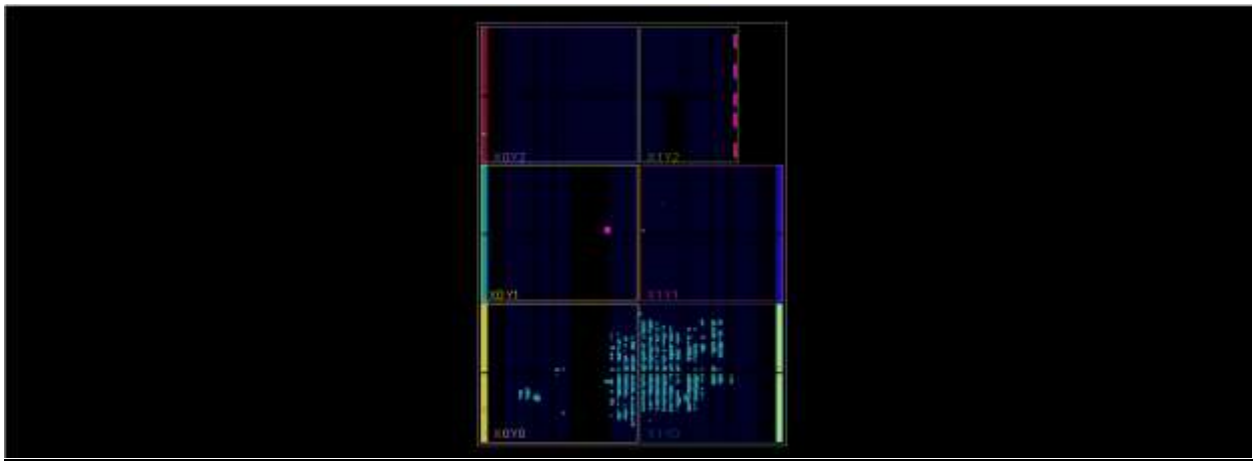


4) Utilization report:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (105)	BUFGCTRL (32)	BSCANE2 (4)
SPI_Wrapper	1469	2225	17	725	1330	139	892	1	5	2	1
dbg_hub (dbg_hub)	475	727	0	232	451	24	305	0	0	1	1
ram (RAM)	2	17	0	6	2	0	0	0.5	0	0	0
SPI (SPI_Slave)	40	26	0	20	40	0	16	0	0	0	0
u_ila_0 (u_ila_0)	952	1455	17	473	837	115	570	0.5	0	0	0

⇨ Implementation for gray encoding:

1) Device:



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

2) Message tab:

3) Timing report:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.857 ns	Worst Hold Slack (WHS):	0.040 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	4274	Total Number of Endpoints:	4258	Total Number of Endpoints:	2405

All user specified timing constraints are met.

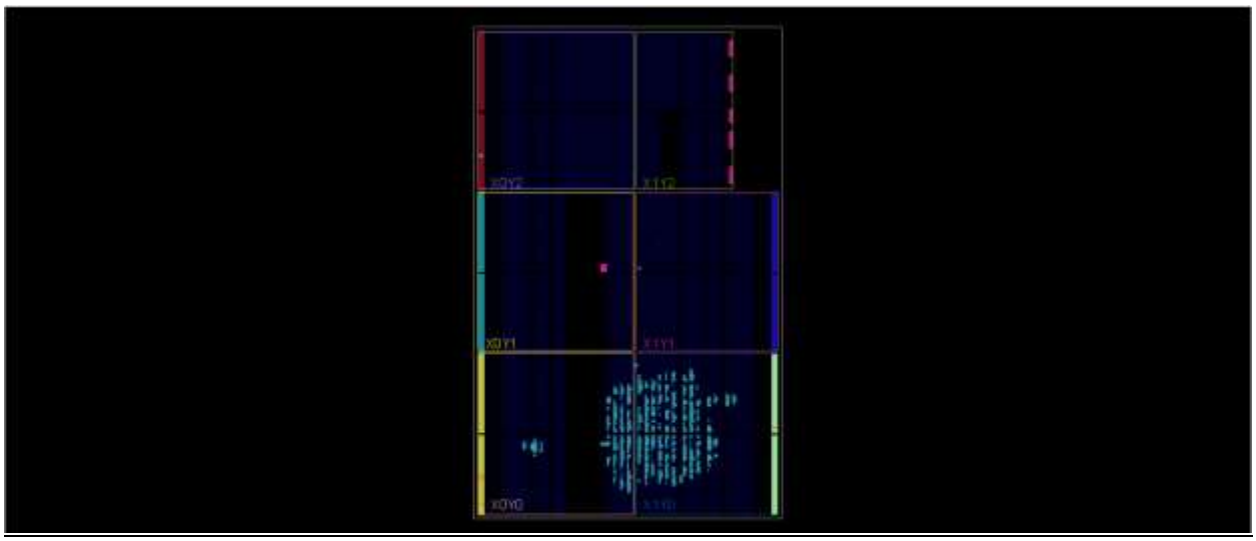
---

4) Utilization report:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
SPI_Wrapper	1443	2170	16	727	1310	133	869	1	5	2	1
dbg_hub (dbg_hub)	475	727	0	244	451	24	311	0	0	1	1
ram (RAM)	1	17	0	6	1	0	0	0.5	0	0	0
SPI (SPI_Slave)	41	24	0	20	41	0	14	0	0	0	0
u_ila_0 (u_ila_0)	826	1402	16	462	817	109	543	0.5	0	0	0

⇨ Implementation for sequential encoding:

1) Device:



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

2) Message tab:

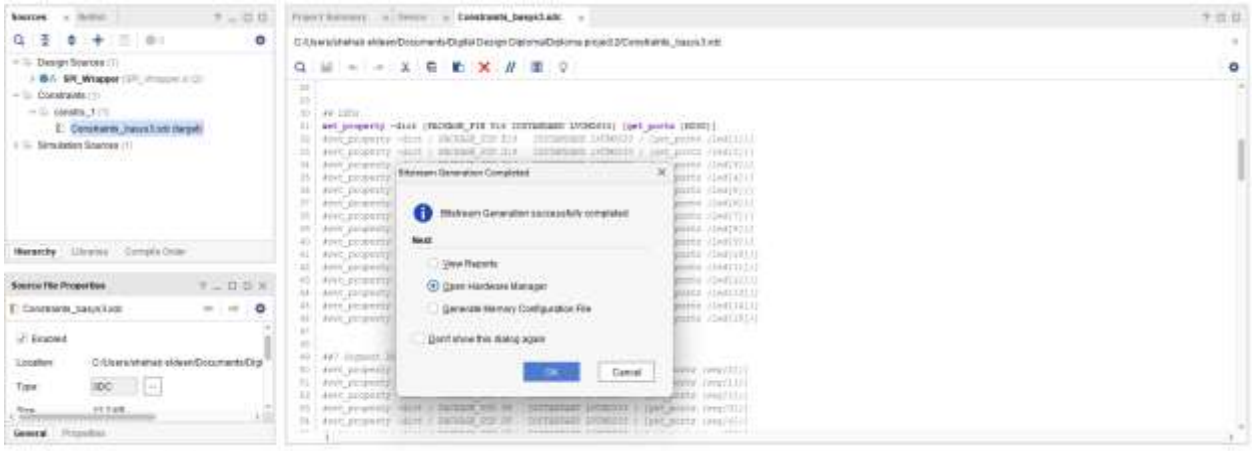
3) Timing report:

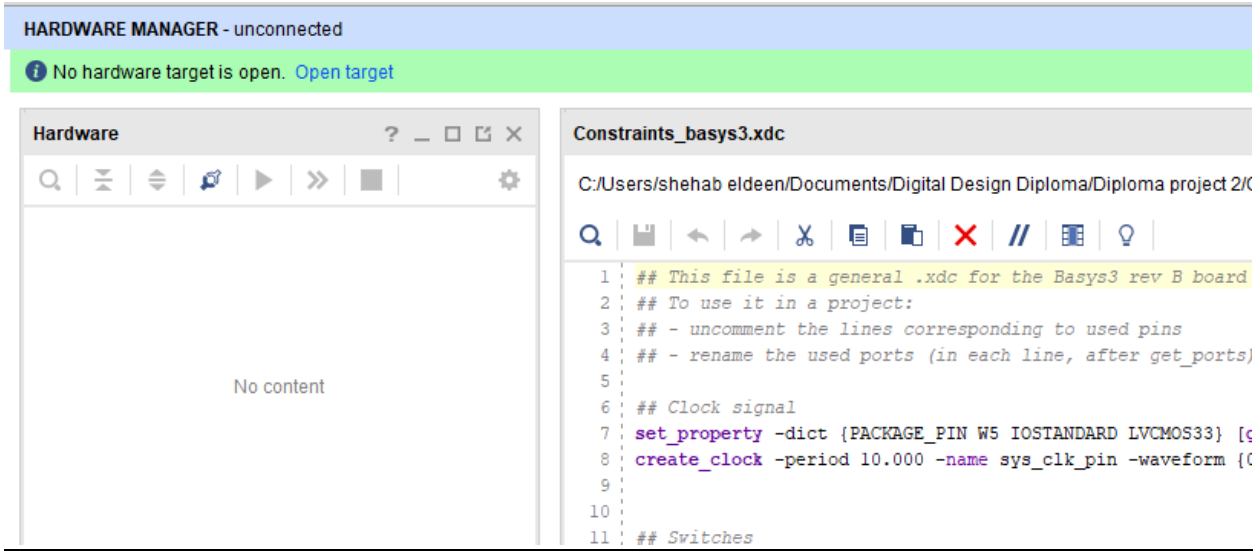
4) Utilization report:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
SPI_Wrapper	1442	2170	16	734	1309	133	672	1	5	2	1
dbg_hub	475	727	0	240	451	24	310	0	0	1	1
ram	2	17	0	4	2	0	0	0.5	0	0	0
SPI	38	24	0	21	38	0	14	0	0	0	0
u_ila_0	927	1402	16	477	818	109	547	0.5	0	0	0

Comment : we will choose one hot encoding as the best one to operate at highest frequency as it has highest setup time slack

⇨ ***Generate bit stream file:***





⇨ ***Netlist Verilog code:***