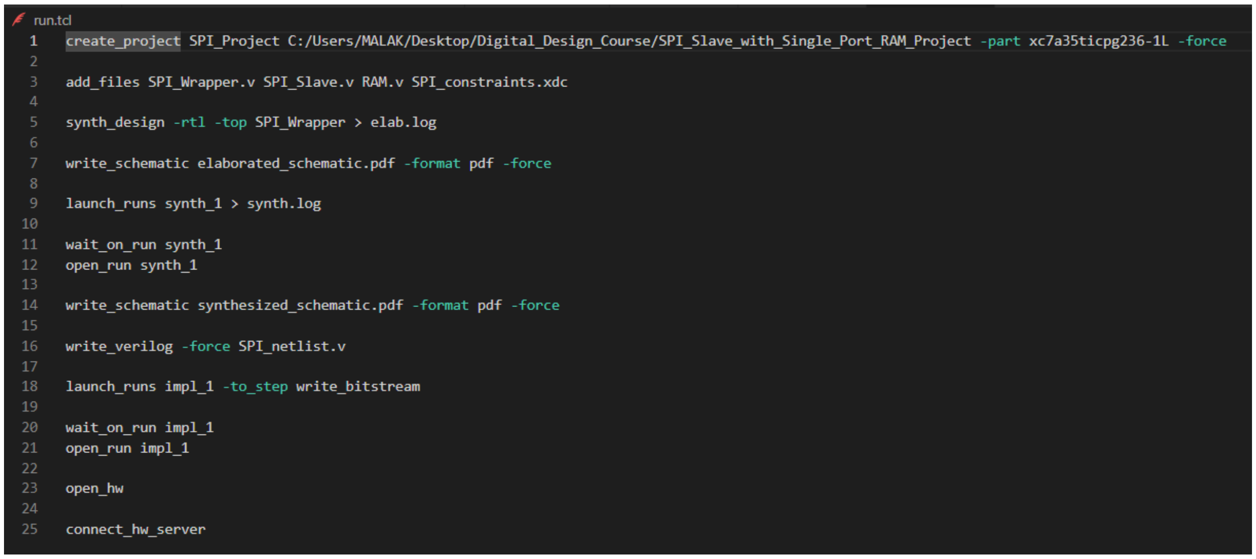
SPI SLAVE WITH SINGLE PORT RAM

Digital Design Course

Malak Hamed Mokhtar
22 – 08 -2023

SPI Slave with Single Port RAM

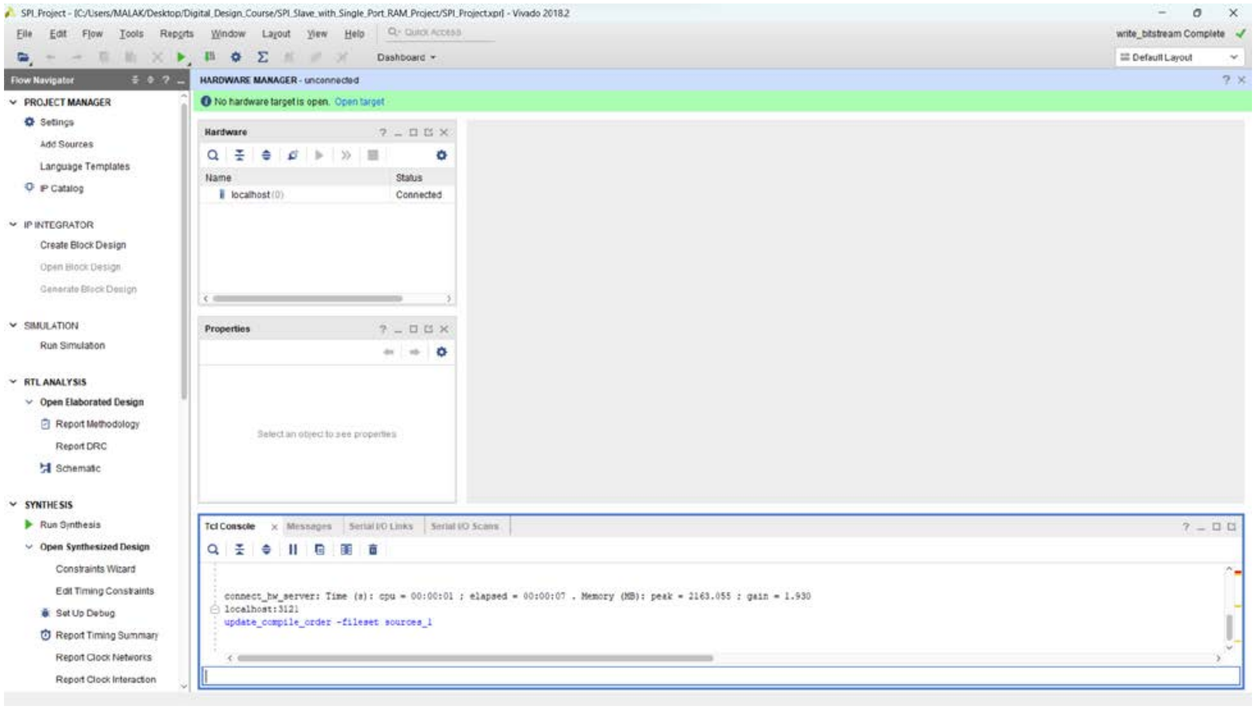
- Create Tcl file to create the vivado project and run the full design flow on Vivado using the Tcl file



run.tcl

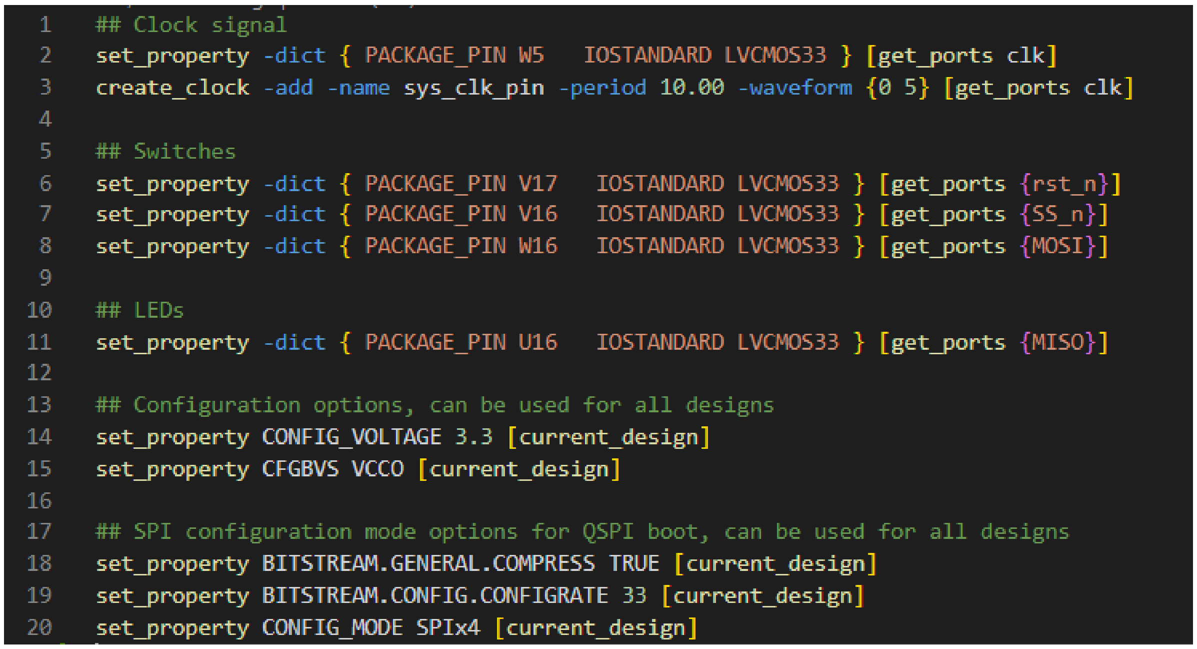
```
create_project SPI_Project C:/Users/MALAK/Desktop/Digital_Design_Course/SPI_Slave_with_Single_Port_RAM_Project -part xc7a35ticpg236-1L -force

add_files SPI_Wrapper.v SPI_Slave.v RAM.v SPI_constraints.xdc

synth_design -rtl -top SPI_Wrapper > elab.log

write_schematic elaborated_schematic.pdf -format pdf -force

launch_runs synth_1 > synth.log

wait_on_run synth_1
open_run synth_1

write_schematic synthesized_schematic.pdf -format pdf -force

write_verilog -force SPI_netlist.v

launch_runs impl_1 -to_step write_bitstream

wait_on_run impl_1
open_run impl_1

open_hw

connect_hw_server
```

Window shown after running run.tcl :



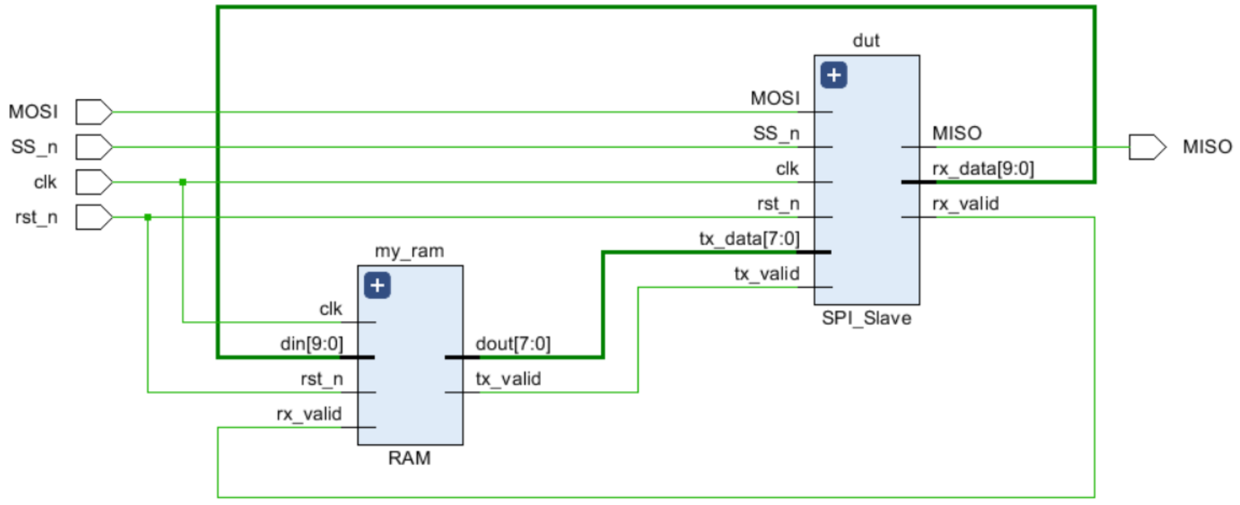
```
connect_hw_server: Time (s): cpu = 00:00:01 ; elapsed = 00:00:07 . Memory (MB): peak = 2163.055 : gain = 1.930
localhost:3121
update_compile_order -fileset sources_1
```

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

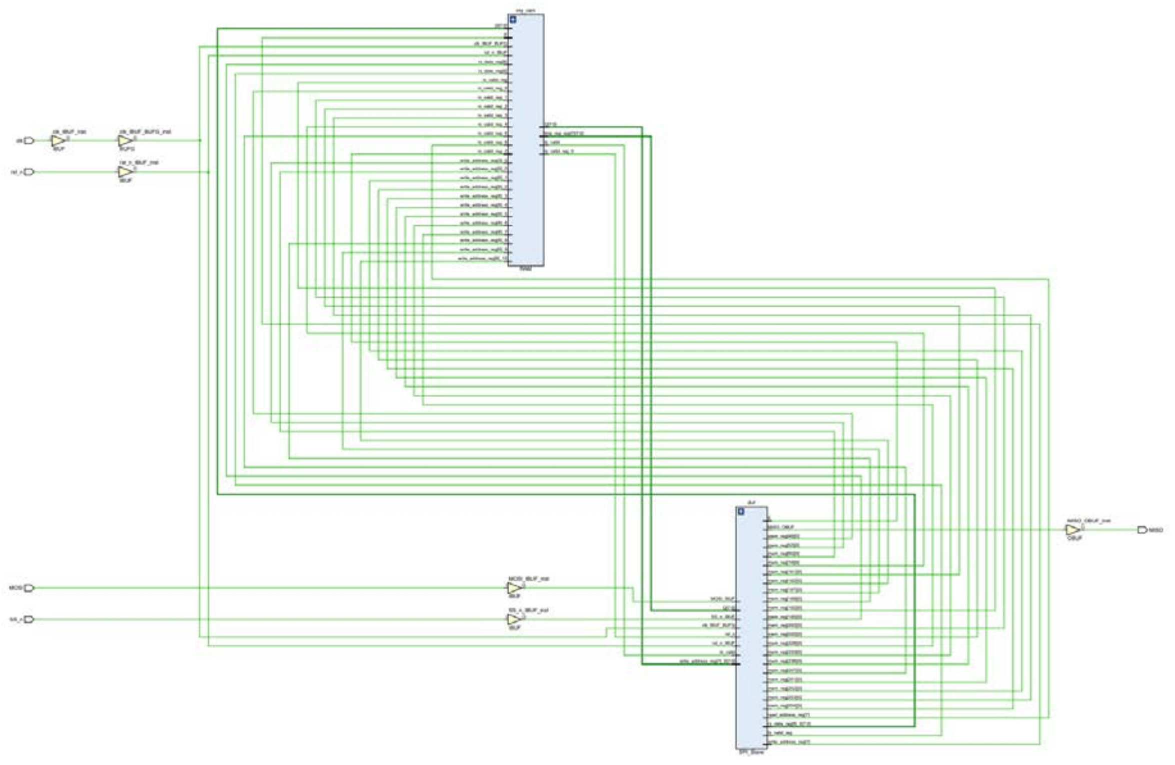


```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## 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}]

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

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

SPI Wrapper elaborated schematic:

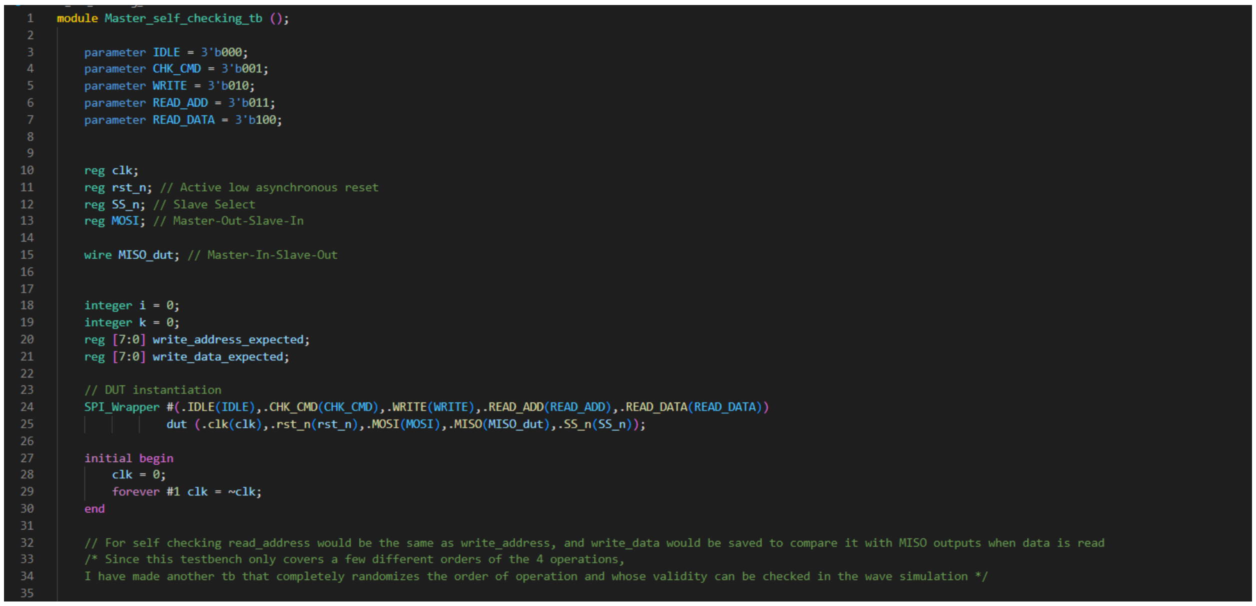


SPI Wrapper synthesized schematic:

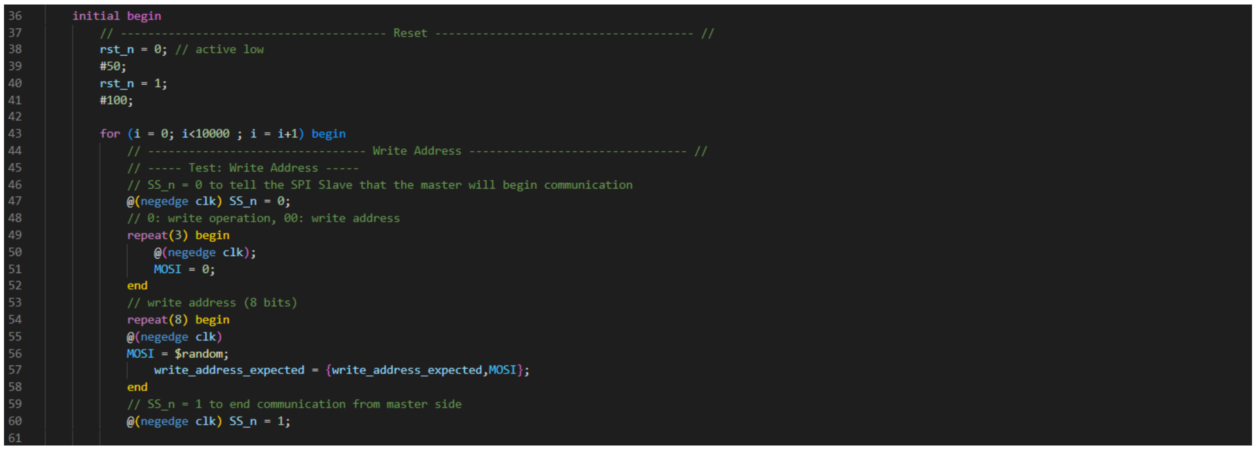


1) Snippets from the waveforms captured from QuestaSim for the design with inputs assigned values and output values visible.

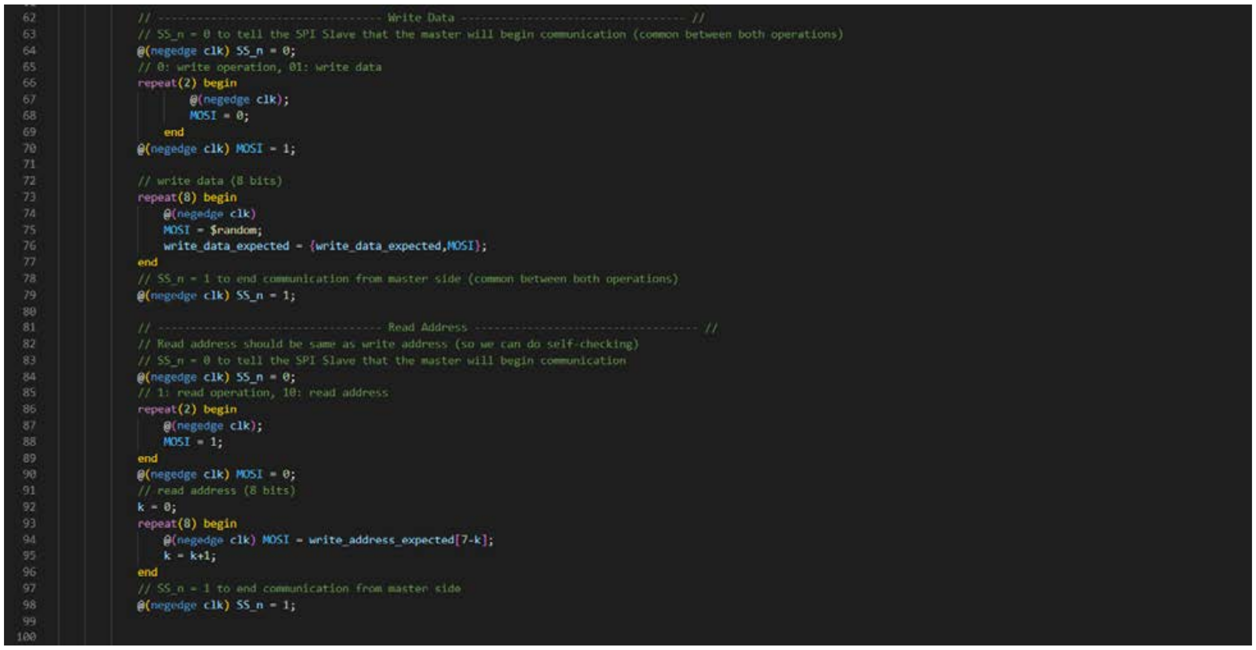
Using Master_self_checking_tb (DoFile2):



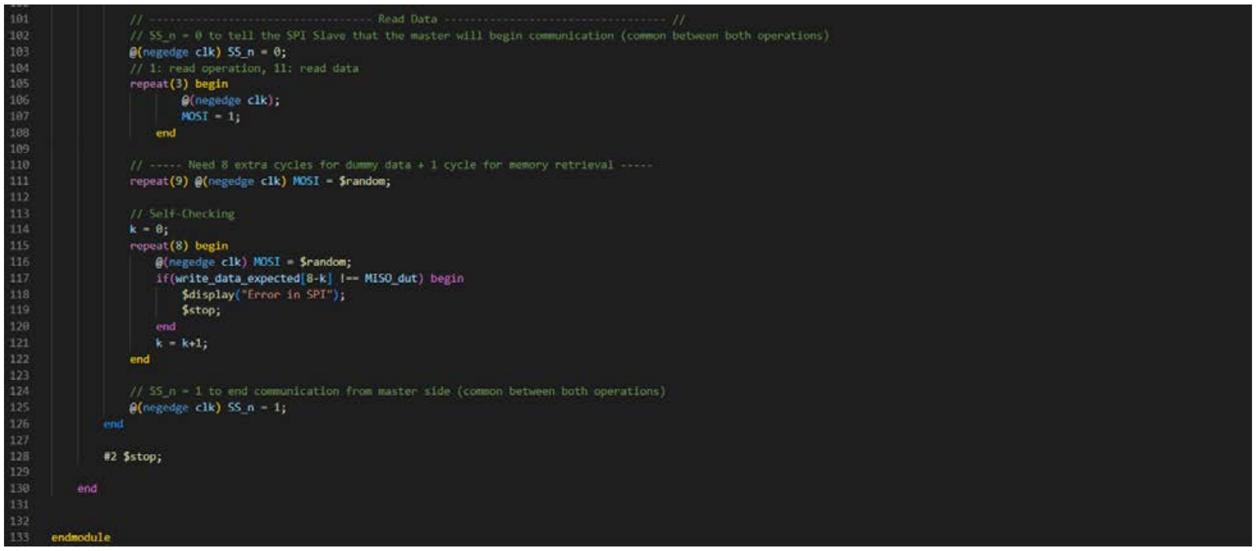
```verilog
module Master_self_checking_tb ();

    parameter IDLE = 3'b000;
    parameter CHK_CMD = 3'b001;
    parameter WRITE = 3'b010;
    parameter READ_ADD = 3'b011;
    parameter READ_DATA = 3'b100;


    reg clk;
    reg rst_n; // Active low asynchronous reset
    reg SS_n; // Slave Select
    reg MOSI; // Master-Out-Slave-In

    wire MISO_dut; // Master-In-Slave-Out


    integer i = 0;
    integer k = 0;
    reg [7:0] write_address_expected;
    reg [7:0] write_data_expected;

    // DUT instantiation
    SPI_Wrapper #(.IDLE(IDLE),.CHK_CMD(CHK_CMD),.WRITE(WRITE),.READ_ADD(READ_ADD),.READ_DATA(READ_DATA))
                dut (.clk(clk),.rst_n(rst_n),.MOSI(MOSI),.MISO(MISO_dut),.SS_n(SS_n));

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

    // For self checking read_address would be the same as write_address, and write_data would be saved to compare it with MISO outputs when data is read
    /* Since this testbench only covers a few different orders of the 4 operations,
    I have made another tb that completely randomizes the order of operation and whose validity can be checked in the wave simulation */

```



```verilog
    initial begin
        // ------------------------------------- Reset ------------------------------------- //
        rst_n = 0; // active low
        #50;
        rst_n = 1;
        #100;

        for (i = 0; i<10000 ; i = i+1) begin
            // -------------------------------- Write Address -------------------------------- //
            // ----- Test: Write Address -----
            // SS_n = 0 to tell the SPI Slave that the master will begin communication
            @(negedge clk) SS_n = 0;
            // 0: write operation, 00: write address
            repeat(3) begin
                @(negedge clk);
                MOSI = 0;
            end
            // write address (8 bits)
            repeat(8) begin
            @(negedge clk)
            MOSI = $random;
                write_address_expected = {write_address_expected,MOSI};
            end
            // SS_n = 1 to end communication from master side
            @(negedge clk) SS_n = 1;

```

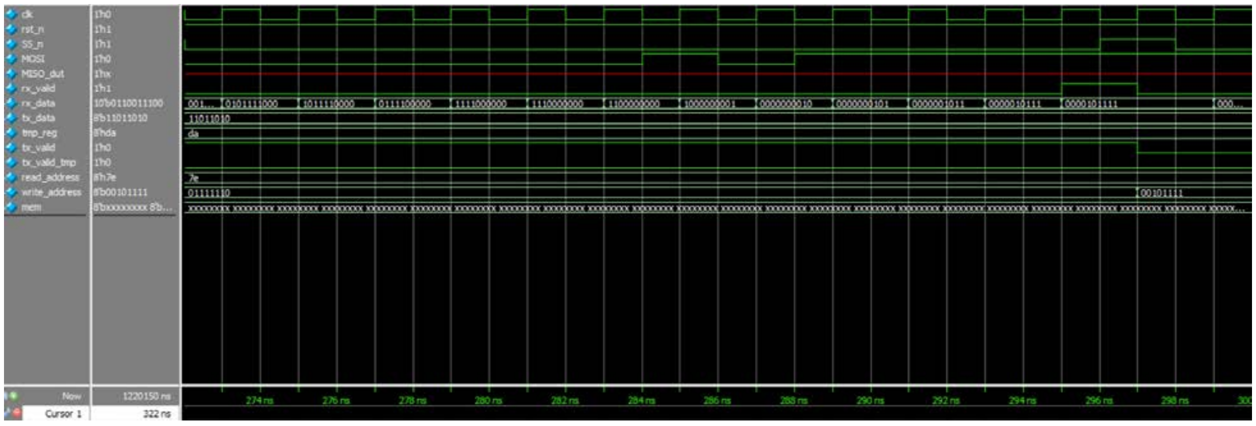


```verilog
            // ---------------------------------- Write Data ---------------------------------- //
            // SS_n = 0 to tell the SPI Slave that the master will begin communication (common between both operations)
            @(negedge clk) SS_n = 0;
            // 0: write operation, 01: write data
            repeat(2) begin
                    @(negedge clk);
                    MOSI = 0;
                end
            @(negedge clk) MOSI = 1;

            // write data (8 bits)
            repeat(8) begin
                @(negedge clk)
                MOSI = $random;
                write_data_expected = {write_data_expected,MOSI};
            end
            // SS_n = 1 to end communication from master side (common between both operations)
            @(negedge clk) SS_n = 1;

            // ---------------------------------- Read Address ---------------------------------- //
            // Read address should be same as write address (so we can do self-checking)
            // SS_n = 0 to tell the SPI Slave that the master will begin communication
            @(negedge clk) SS_n = 0;
            // 1: read operation, 10: read address
            repeat(2) begin
                @(negedge clk);
                MOSI = 1;
            end
            @(negedge clk) MOSI = 0;
            // read address (8 bits)
            k = 0;
            repeat(8) begin
                @(negedge clk) MOSI = write_address_expected[7-k];
                k = k+1;
            end
            // SS_n = 1 to end communication from master side
            @(negedge clk) SS_n = 1;


```



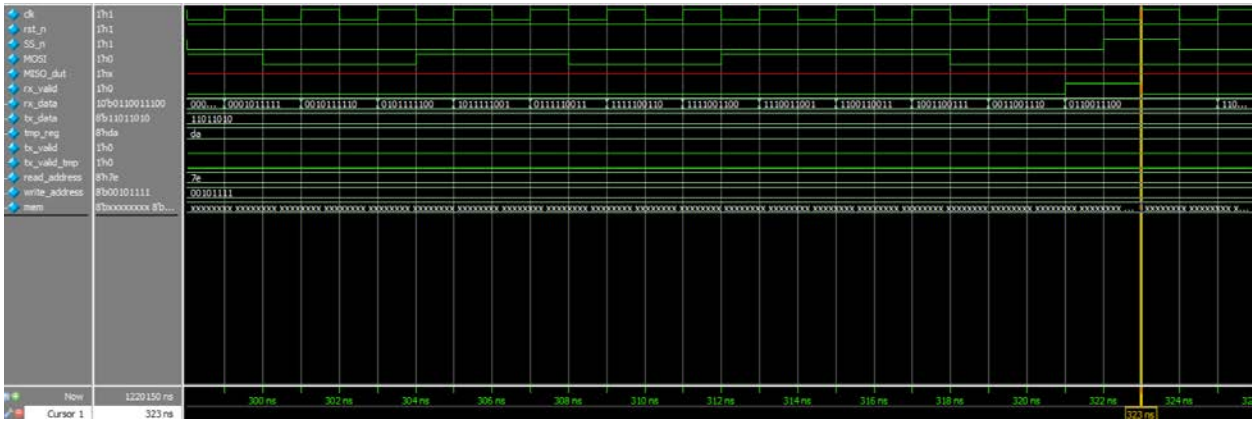
```verilog
            // ---------------------------------- Read Data ---------------------------------- //
            // SS_n = 0 to tell the SPI Slave that the master will begin communication (common between both operations)
            @(negedge clk) SS_n = 0;
            // 1: read operation, 11: read data
            repeat(3) begin
                    @(negedge clk);
                    MOSI = 1;
                end

            // ----- Need 8 extra cycles for dummy data + 1 cycle for memory retrieval -----
            repeat(9) @(negedge clk) MOSI = $random;

            // Self-Checking
            k = 0;
            repeat(8) begin
                @(negedge clk) MOSI = $random;
                if(write_data_expected[8-k] !== MISO_dut) begin
                    $display("Error in SPI");
                    $stop;
                end
                k = k+1;
            end

            // SS_n = 1 to end communication from master side (common between both operations)
            @(negedge clk) SS_n = 1;
        end

        #2 $stop;

    end


endmodule
```

Write address operation (starting from one clk cycle for SS_n = 0):

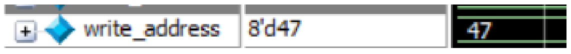


```
clk            1'h0
rst_n          1'h1
SS_n           1'h1
MOSI           1'h0
MISO_dut       1'hx
rx_valid       1'h1
rx_data        10'b0110011100
tx_data        8'b11011010
tmp_reg        8'hda
tx_valid       1'h0
tx_valid_tmp   1'h0
read_address   8'h7e
write_address  8'b00101111
mem            8'bxxxxxxxx 8'b...
Now            1220150 ns
Cursor 1       322 ns
```

Write data operation (starting from one clk cyle for SS_n = 0):

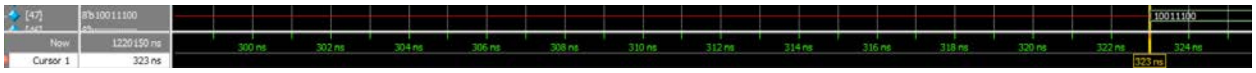


```
clk            1'h1
rst_n          1'h1
SS_n           1'h1
MOSI           1'h0
MISO_dut       1'hx
rx_valid       1'h0
rx_data        10'b0110011100
tx_data        8'b11011010
tmp_reg        8'hda
tx_valid       1'h0
tx_valid_tmp   1'h0
read_address   8'h7e
write_address  8'b00101111
mem            8'bxxxxxxxx 8'b...
Now            1220150 ns
Cursor 1       323 ns
```

Write address in decimal: 

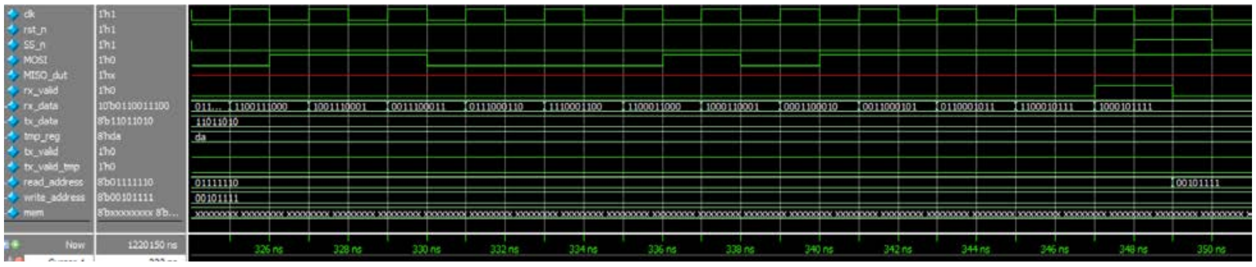
```
write_address  8'd47  47
```

Evidence data was written to memory in address 47:



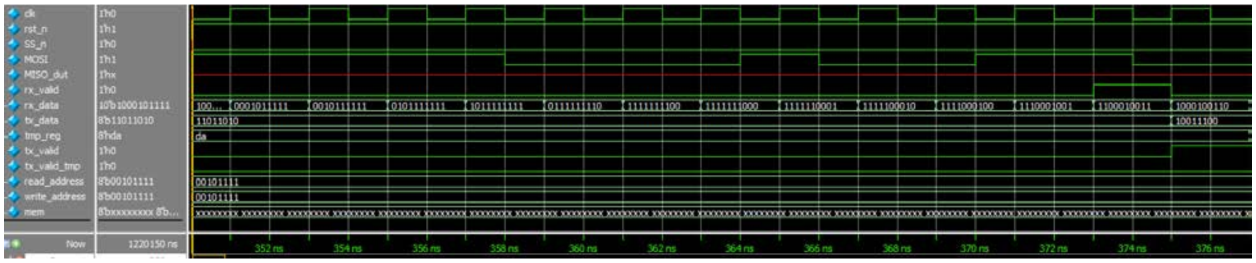
```
[47]      8'b10011100
Now       1220150 ns
Cursor 1  323 ns
```

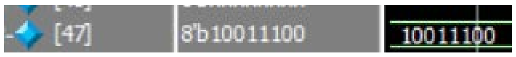
Read address operation: (starting from one clk cyle for SS_n = 0):



```
clk            1'h1
rst_n          1'h1
SS_n           1'h1
MOSI           1'h0
MISO_dut       1'hx
rx_valid       1'h0
rx_data        10'b0110011100
tx_data        8'b11011010
tmp_reg        8'hda
tx_valid       1'h0
tx_valid_tmp   1'h0
read_address   8'b01111110
write_address  8'b00101111
mem            8'bxxxxxxxx 8'b...
Now            1220150 ns
```

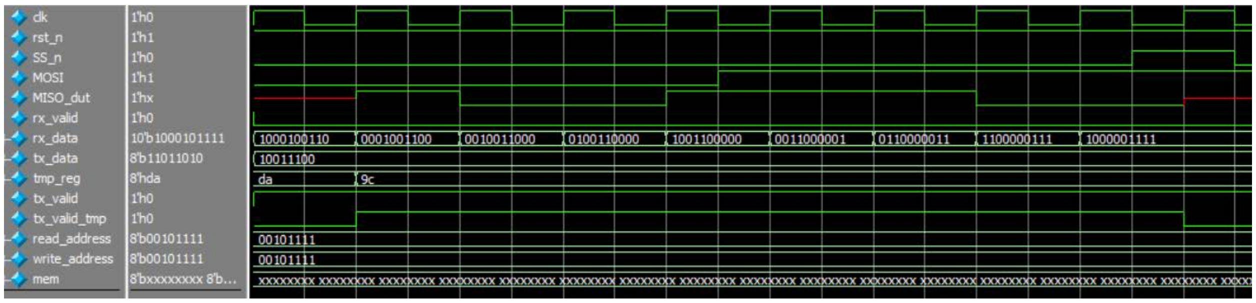
Read data operation: (starting from one clk cyle for SS_n = 0) (8 dummy cycles shown):



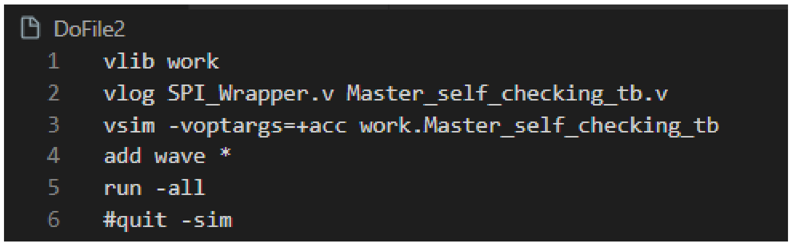
Read data: 

[47] 8'b10011100 10011100

Data is output on MISO port and then SS_n is raised:



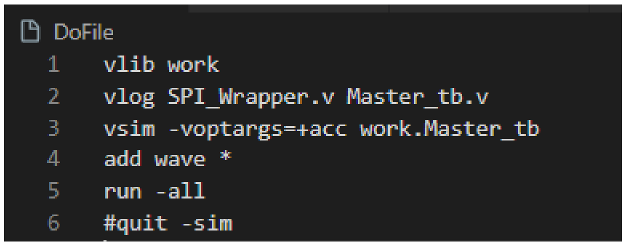
Do file for self-checking testbench:



DoFile2

```
vlib work
vlog SPI_Wrapper.v Master_self_checking_tb.v
vsim -voptargs=+acc work.Master_self_checking_tb
add wave *
run -all
#quit -sim
```

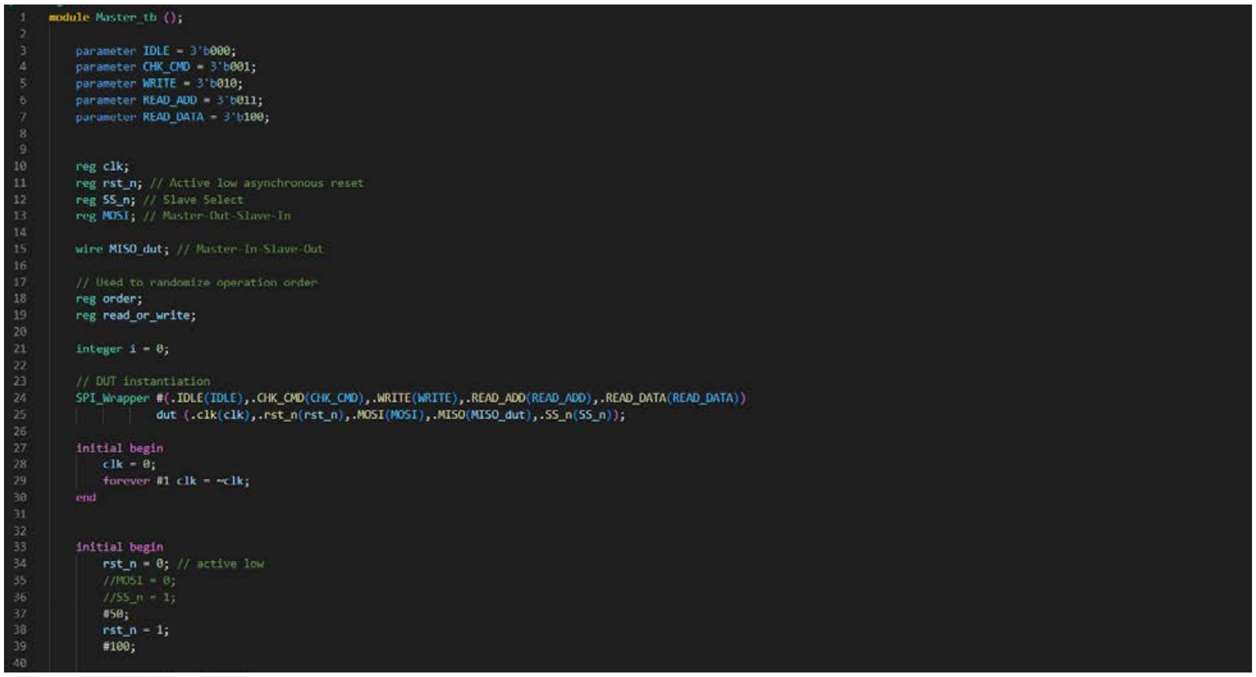
Do file for more randomized testbench that would be found next page:



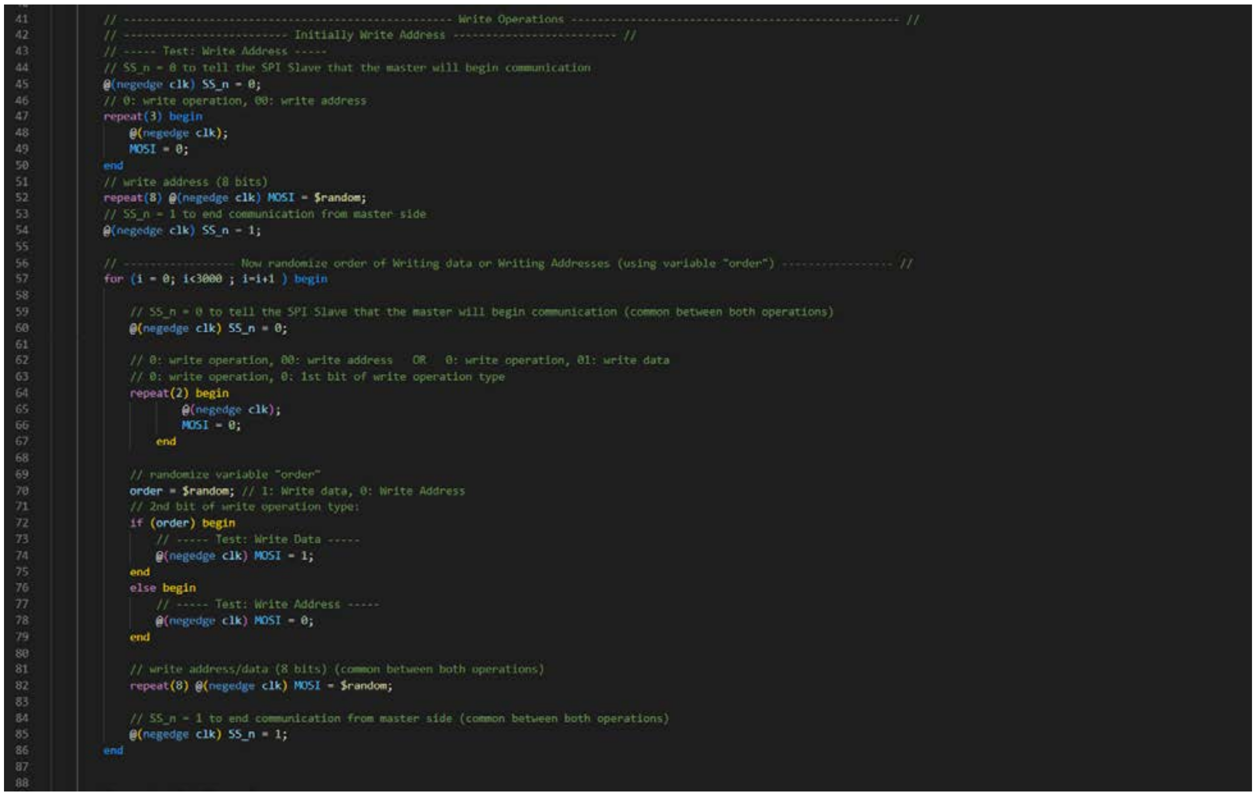
DoFile

```
vlib work
vlog SPI_Wrapper.v Master_tb.v
vsim -voptargs=+acc work.Master_tb
add wave *
run -all
#quit -sim
```

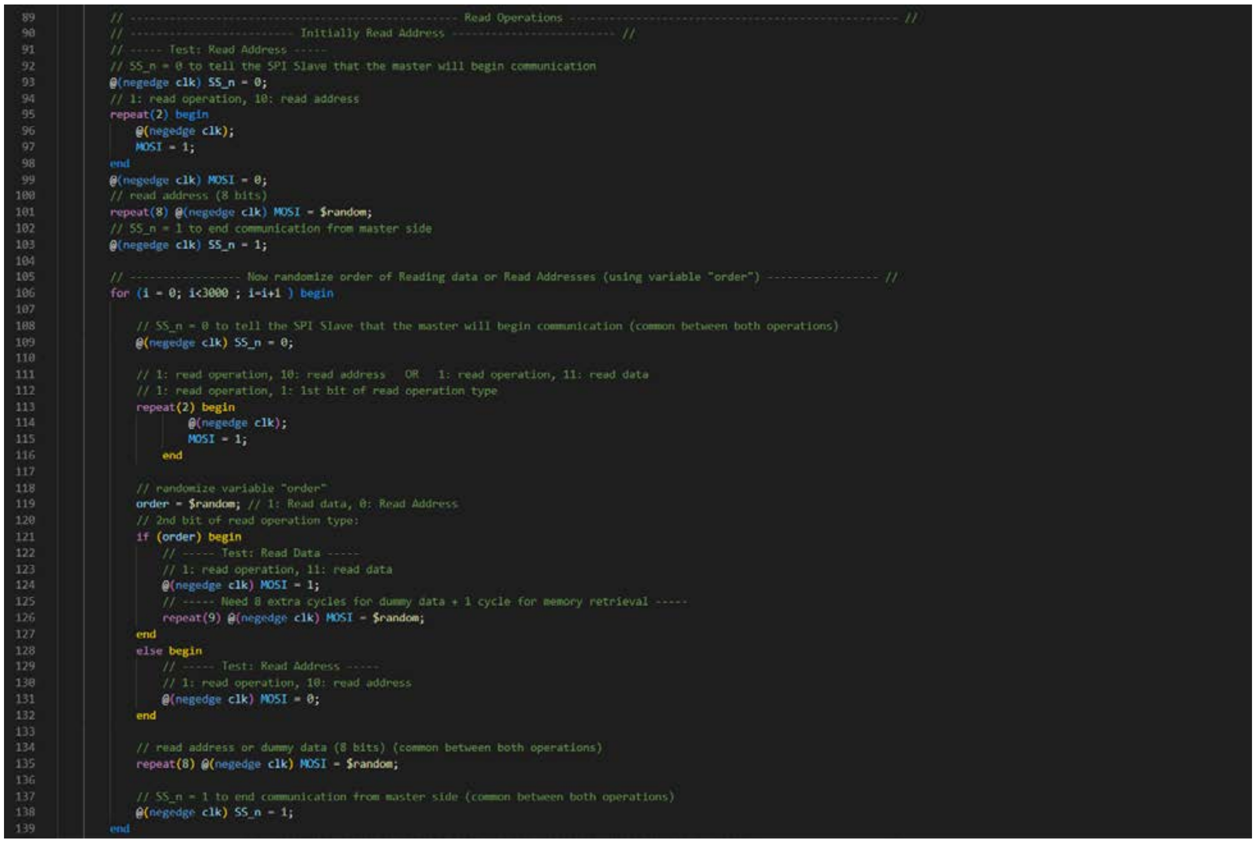
Master_tb, that is not self-checking but tries random operation orders:



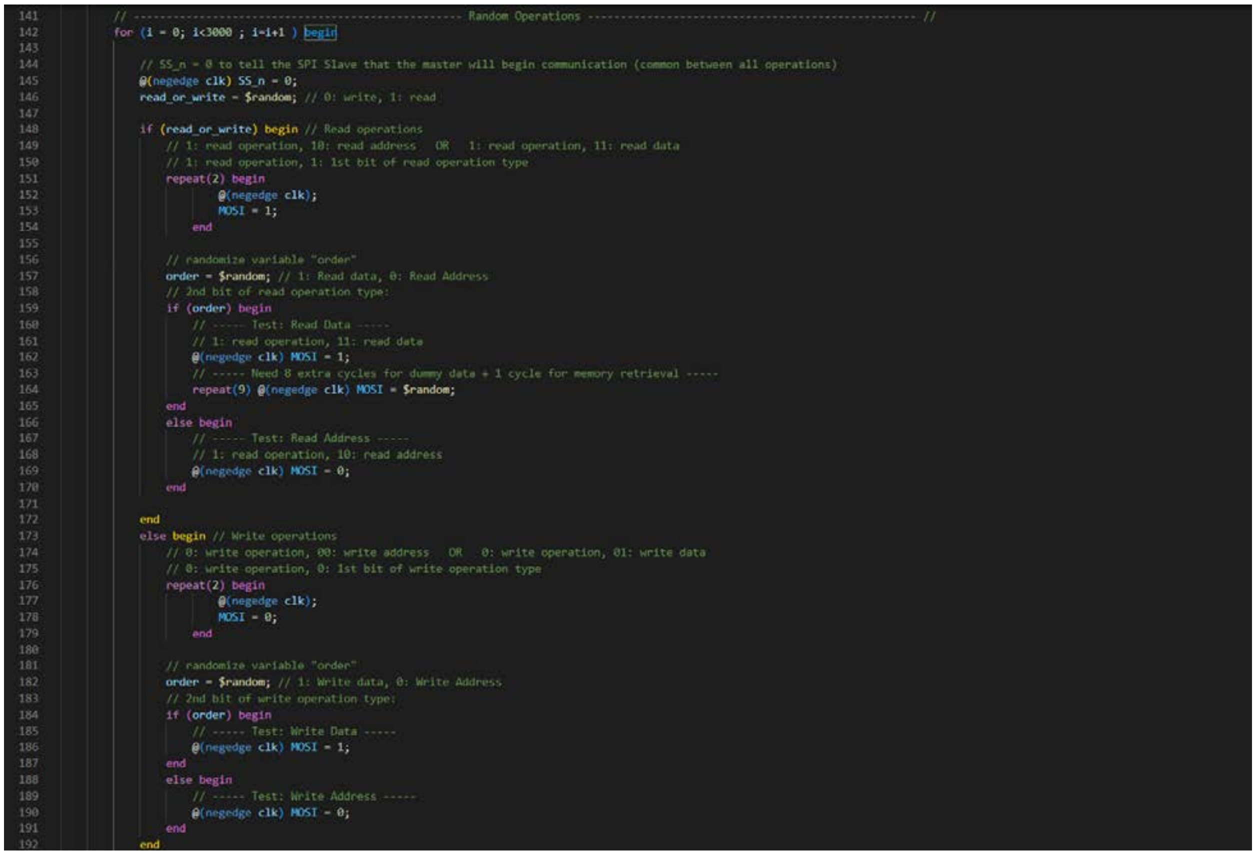
```verilog
module Master_tb ();

    parameter IDLE = 3'b000;
    parameter CHK_CMD = 3'b001;
    parameter WRITE = 3'b010;
    parameter READ_ADD = 3'b011;
    parameter READ_DATA = 3'b100;


    reg clk;
    reg rst_n; // Active low asynchronous reset
    reg SS_n; // Slave Select
    reg MOSI; // Master-Out-Slave-In

    wire MISO_dut; // Master-In-Slave-Out

    // Used to randomize operation order
    reg order;
    reg read_or_write;

    integer i = 0;

    // DUT instantiation
    SPI_Wrapper #(.IDLE(IDLE),.CHK_CMD(CHK_CMD),.WRITE(WRITE),.READ_ADD(READ_ADD),.READ_DATA(READ_DATA))
                dut (.clk(clk),.rst_n(rst_n),.MOSI(MOSI),.MISO(MISO_dut),.SS_n(SS_n));

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


    initial begin
        rst_n = 0; // active low
        //MOSI = 0;
        //SS_n = 1;
        #50;
        rst_n = 1;
        #100;

```



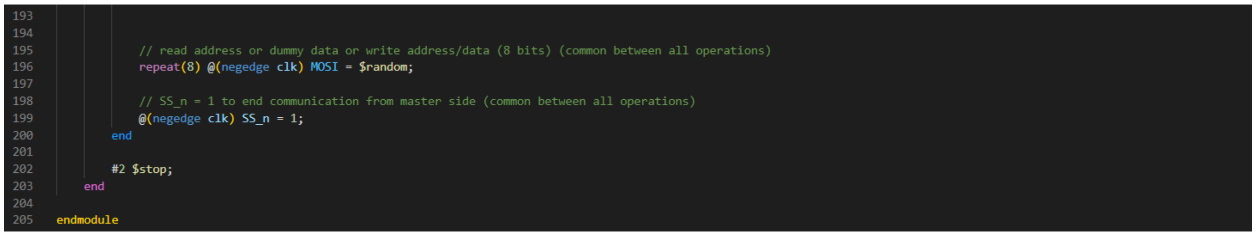
```verilog
        // ------------------------------------------------- Write Operations ------------------------------------------------- //
        // ------------------------- Initially Write Address ------------------------- //
        // ----- Test: Write Address -----
        // SS_n = 0 to tell the SPI Slave that the master will begin communication
        @(negedge clk) SS_n = 0;
        // 0: write operation, 00: write address
        repeat(3) begin
            @(negedge clk);
            MOSI = 0;
        end
        // write address (8 bits)
        repeat(8) @(negedge clk) MOSI = $random;
        // SS_n = 1 to end communication from master side
        @(negedge clk) SS_n = 1;

        // ---------------- Now randomize order of Writing data or Writing Addresses (using variable "order") ---------------- //
        for (i = 0; i<3000 ; i=i+1 ) begin

            // SS_n = 0 to tell the SPI Slave that the master will begin communication (common between both operations)
            @(negedge clk) SS_n = 0;

            // 0: write operation, 00: write address   OR   0: write operation, 01: write data
            // 0: write operation, 0: 1st bit of write operation type
            repeat(2) begin
                    @(negedge clk);
                    MOSI = 0;
                end

            // randomize variable "order"
            order = $random; // 1: Write data, 0: Write Address
            // 2nd bit of write operation type:
            if (order) begin
                // ----- Test: Write Data -----
                @(negedge clk) MOSI = 1;
            end
            else begin
                // ----- Test: Write Address -----
                @(negedge clk) MOSI = 0;
            end

            // write address/data (8 bits) (common between both operations)
            repeat(8) @(negedge clk) MOSI = $random;

            // SS_n = 1 to end communication from master side (common between both operations)
            @(negedge clk) SS_n = 1;
        end

```



```verilog
// ---------------------------------------------------- Read Operations ---------------------------------------------------- //
// ------------------------ Initially Read Address ------------------------ //
// ----- Test: Read Address -----
// SS_n = 0 to tell the SPI Slave that the master will begin communication
@(negedge clk) SS_n = 0;
// 1: read operation, 10: read address
repeat(2) begin
    @(negedge clk);
    MOSI = 1;
end
@(negedge clk) MOSI = 0;
// read address (8 bits)
repeat(8) @(negedge clk) MOSI = $random;
// SS_n = 1 to end communication from master side
@(negedge clk) SS_n = 1;

// ---------------- Now randomize order of Reading data or Read Addresses (using variable "order") ---------------- //
for (i = 0; i<3000 ; i=i+1 ) begin

    // SS_n = 0 to tell the SPI Slave that the master will begin communication (common between both operations)
    @(negedge clk) SS_n = 0;

    // 1: read operation, 10: read address   OR   1: read operation, 11: read data
    // 1: read operation, 1: 1st bit of read operation type
    repeat(2) begin
            @(negedge clk);
            MOSI = 1;
        end

    // randomize variable "order"
    order = $random; // 1: Read data, 0: Read Address
    // 2nd bit of read operation type:
    if (order) begin
        // ----- Test: Read Data -----
        // 1: read operation, 11: read data
        @(negedge clk) MOSI = 1;
        // ----- Need 8 extra cycles for dummy data + 1 cycle for memory retrieval -----
        repeat(9) @(negedge clk) MOSI = $random;
    end
    else begin
        // ----- Test: Read Address -----
        // 1: read operation, 10: read address
        @(negedge clk) MOSI = 0;
    end

    // read address or dummy data (8 bits) (common between both operations)
    repeat(8) @(negedge clk) MOSI = $random;

    // SS_n = 1 to end communication from master side (common between both operations)
    @(negedge clk) SS_n = 1;
end
```



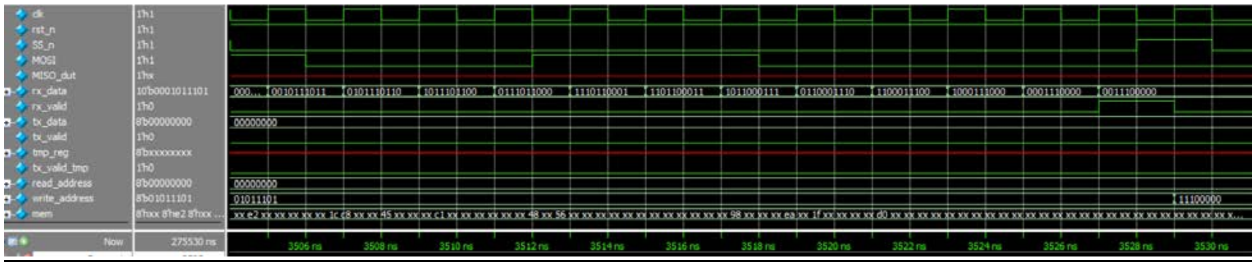
```verilog
// ---------------------------------------------------- Random Operations ---------------------------------------------------- //
for (i = 0; i<3000 ; i=i+1 ) begin

    // SS_n = 0 to tell the SPI Slave that the master will begin communication (common between all operations)
    @(negedge clk) SS_n = 0;
    read_or_write = $random; // 0: write, 1: read

    if (read_or_write) begin // Read operations
        // 1: read operation, 10: read address   OR   1: read operation, 11: read data
        // 1: read operation, 1: 1st bit of read operation type
        repeat(2) begin
                @(negedge clk);
                MOSI = 1;
            end

        // randomize variable "order"
        order = $random; // 1: Read data, 0: Read Address
        // 2nd bit of read operation type:
        if (order) begin
            // ----- Test: Read Data -----
            // 1: read operation, 11: read data
            @(negedge clk) MOSI = 1;
            // ----- Need 8 extra cycles for dummy data + 1 cycle for memory retrieval -----
            repeat(9) @(negedge clk) MOSI = $random;
        end
        else begin
            // ----- Test: Read Address -----
            // 1: read operation, 10: read address
            @(negedge clk) MOSI = 0;
        end

    end
    else begin // Write operations
        // 0: write operation, 00: write address   OR   0: write operation, 01: write data
        // 0: write operation, 0: 1st bit of write operation type
        repeat(2) begin
                @(negedge clk);
                MOSI = 0;
            end

        // randomize variable "order"
        order = $random; // 1: Write data, 0: Write Address
        // 2nd bit of write operation type:
        if (order) begin
            // ----- Test: Write Data -----
            @(negedge clk) MOSI = 1;
        end
        else begin
            // ----- Test: Write Address -----
            @(negedge clk) MOSI = 0;
        end
    end
```

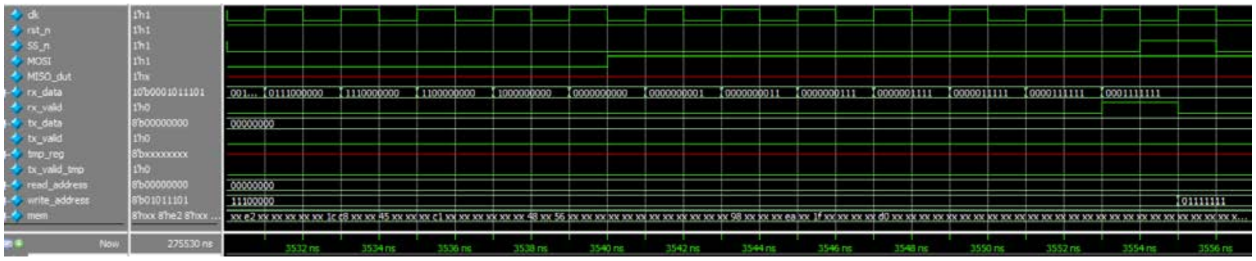


```verilog

            // read address or dummy data or write address/data (8 bits) (common between all operations)
            repeat(8) @(negedge clk) MOSI = $random;

            // SS_n = 1 to end communication from master side (common between all operations)
            @(negedge clk) SS_n = 1;
        end

        #2 $stop;
    end

endmodule
```

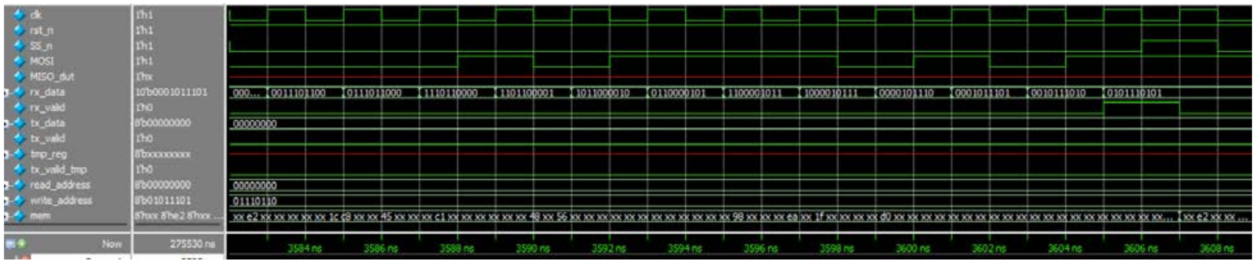
Wave simulations

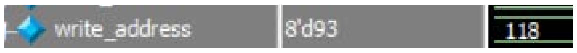
Write address operation and then another write address operation:



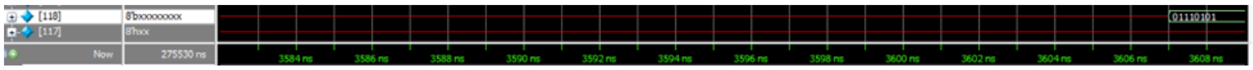


Write data operation and then another write data operation:

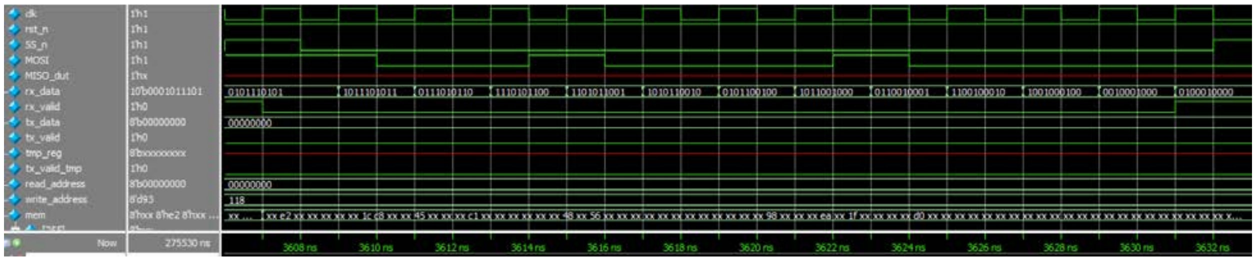


Write address in decimal: 

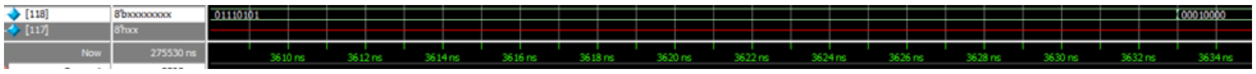
Evidence that data was written to memory:



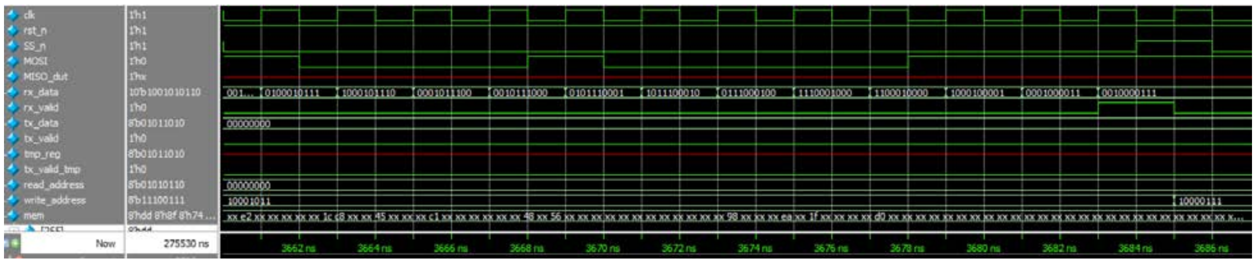
SS_n is raised and then master begins the second write data operation:



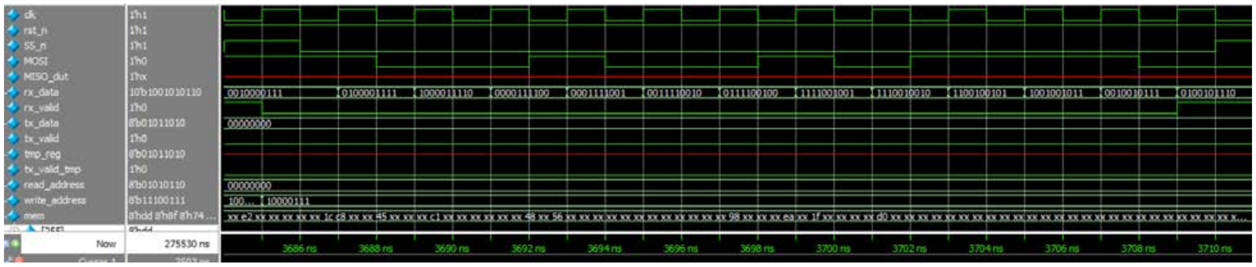
Evidence that write data was changed:



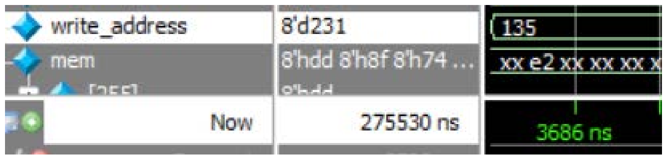
Write address operation and then write data operation:



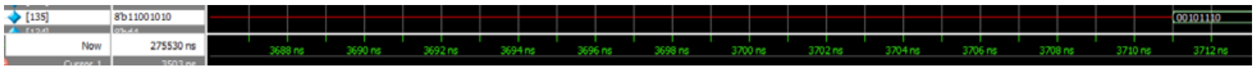
SS_n is raised and then master begins the write data operation:



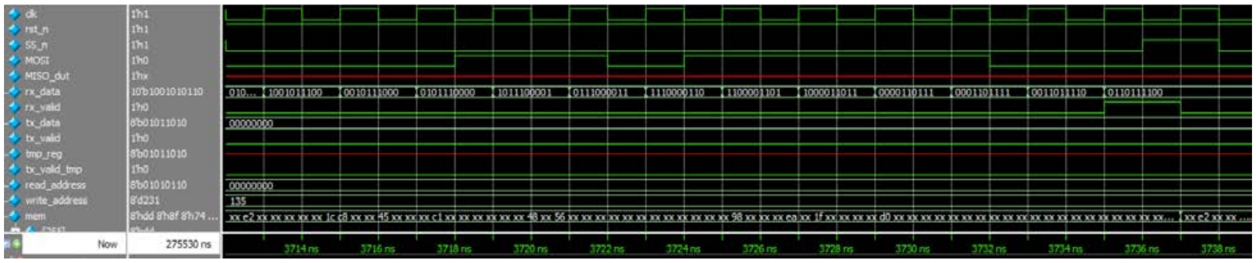
Write address in decimal:



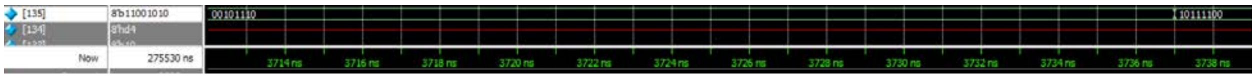
Evidence that write data was changed:



Write data operation and then write address operation:

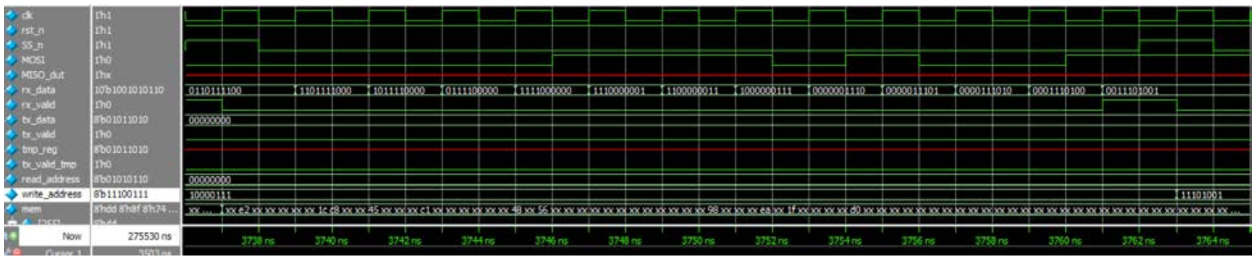


```
clk            1'h1
rst_n          1'h1
SS_n           1'h1
MOSI           1'h0
MISO_dut       1'hx
rx_data        10'b1001010110
rx_valid       1'h0
tx_data        8'b01011010
tx_valid       1'h0
tmp_reg        8'b01011010
tx_valid_tmp   1'h0
read_address   8'b01010110
write_address  8'd231
mem            8'hdd 8'h8f 8'h74 ...
Now            275530 ns
```



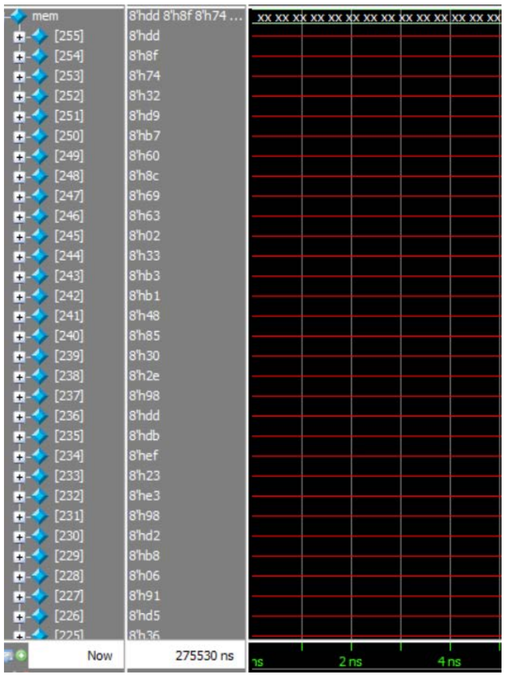
```
[135]   8'b11001010
[134]   8'hd4
Now     275530 ns
```

SS_n is raised and then master begins the write address operation:



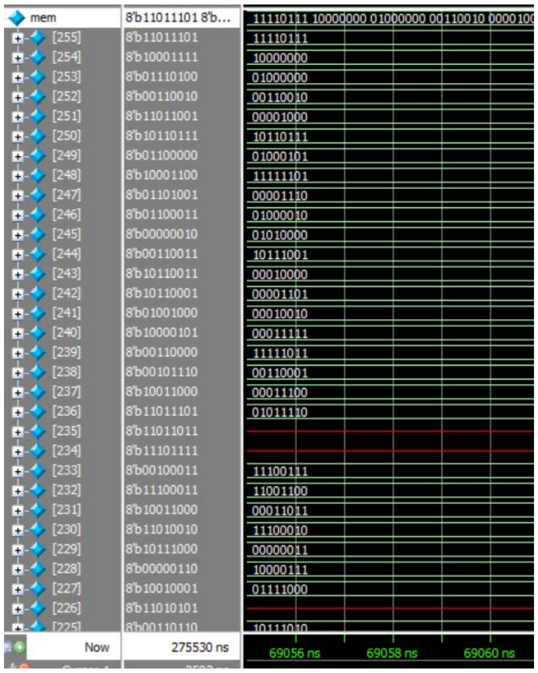
```
clk            1'h1
rst_n          1'h1
SS_n           1'h1
MOSI           1'h0
MISO_dut       1'hx
rx_data        10'b1001010110
rx_valid       1'h0
tx_data        8'b01011010
tx_valid       1'h0
tmp_reg        8'b01011010
tx_valid_tmp   1'h0
read_address   8'b01010110
write_address  8'b11100111
mem            8'hdd 8'h8f 8'h74 ...
Now            275530 ns
```

Partial screenshot of memory initially:



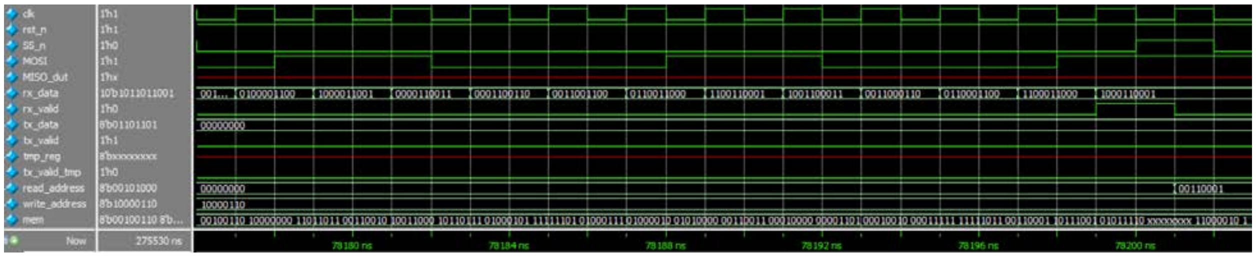
mem	8'hdd 8'h8f 8'h74 ...
[255]	8'hdd
[254]	8'h8f
[253]	8'h74
[252]	8'h32
[251]	8'hd9
[250]	8'hb7
[249]	8'h60
[248]	8'h8c
[247]	8'h69
[246]	8'h63
[245]	8'h02
[244]	8'h33
[243]	8'hb3
[242]	8'hb1
[241]	8'h48
[240]	8'h85
[239]	8'h30
[238]	8'h2e
[237]	8'h98
[236]	8'hdd
[235]	8'hdb
[234]	8'hef
[233]	8'h23
[232]	8'he3
[231]	8'h98
[230]	8'hd2
[229]	8'hb8
[228]	8'h06
[227]	8'h91
[226]	8'hd5
[225]	8'h36
Now	275530 ns

Partial screenshot after randomized write operations:

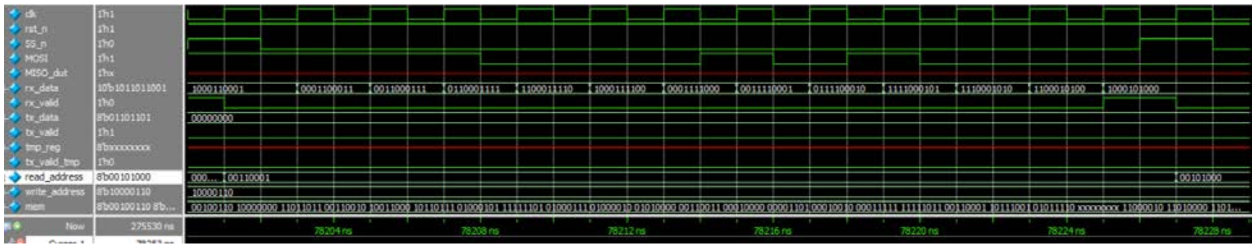


mem	8'b11011101 8'b...	11110111 10000000 01000000 00110010 0000100
[255]	8'b11011101	11110111
[254]	8'b10001111	10000000
[253]	8'b01110100	01000000
[252]	8'b00110010	00110010
[251]	8'b11011001	00001000
[250]	8'b10110111	10110111
[249]	8'b01100000	01000101
[248]	8'b10001100	11111101
[247]	8'b01101001	00001110
[246]	8'b01100011	01000010
[245]	8'b00000010	01010000
[244]	8'b00110011	10111001
[243]	8'b10110011	00010000
[242]	8'b10110001	00001101
[241]	8'b01001000	00010010
[240]	8'b10000101	00011111
[239]	8'b00110000	11111011
[238]	8'b00101110	00110001
[237]	8'b10011000	00011100
[236]	8'b11011101	01011110
[235]	8'b11011011	
[234]	8'b11101111	
[233]	8'b00100011	11100111
[232]	8'b11100011	11001100
[231]	8'b10011000	00011011
[230]	8'b11010010	11100010
[229]	8'b10111000	00000011
[228]	8'b00000110	10000111
[227]	8'b10010001	01111000
[226]	8'b11010101	
[225]	8'b00110110	10111010
Now	275530 ns	69056 ns 69058 ns 69060 ns

Read address operation and then another read address operation:



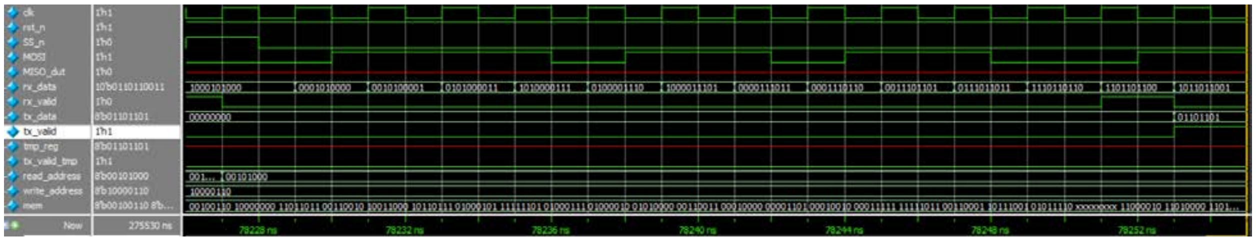
SS_n is raised and then master begins the second read address operation:



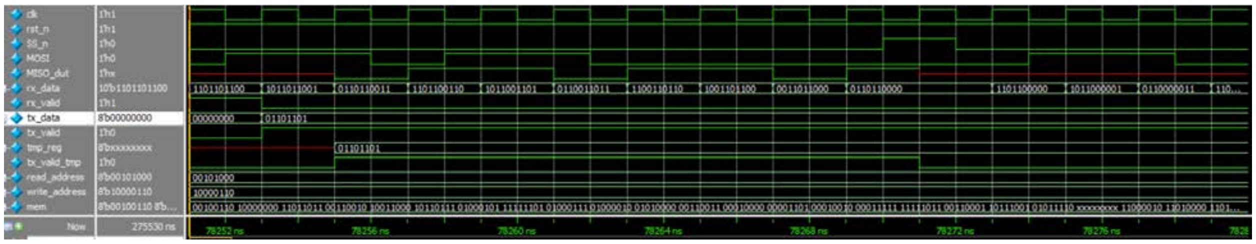
Read address operation and then read data operation:

The previous read address operation was followed by this read data operation:

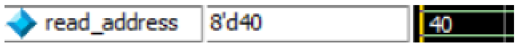
SS_n is raised and then master begins the read data operation: (8 cycles of dummy data after which RAM returns tx_valid as raised)



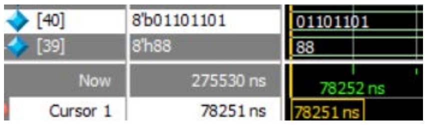
Tx_Data is output on MISO:



Evidence that tx_data is the data in the read address:

Read address in decimal: 

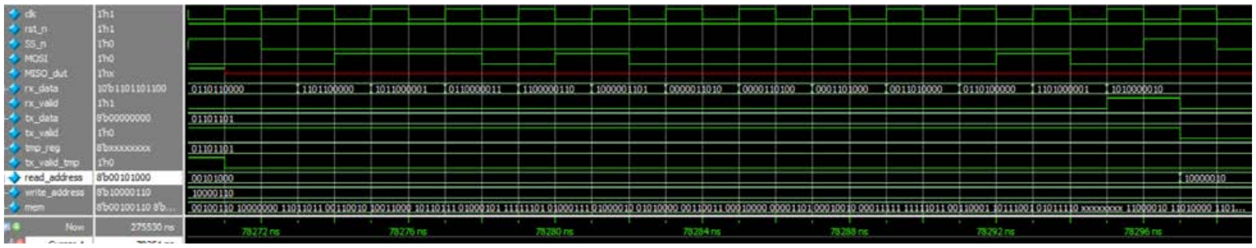
read_address 8'd40 40



```
[40]      8'b01101101   01101101
[39]      8'h88         88
Now       275530 ns     78252 ns
Cursor 1  78251 ns      78251 ns
```

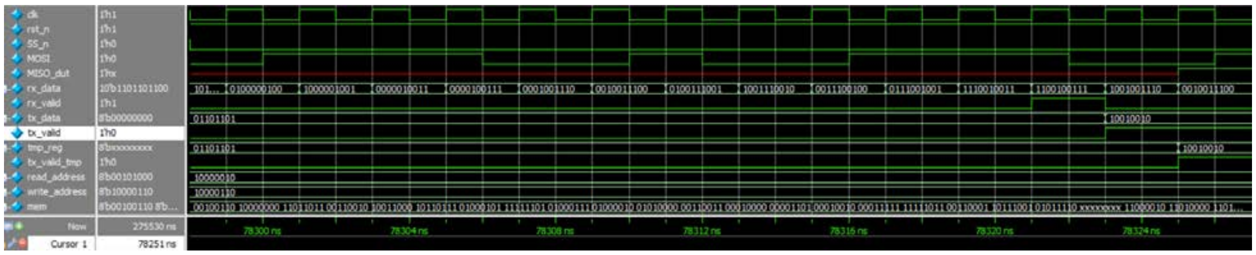
<u>Read data operation and then read address operation:</u>

The previous read data operation was followed by this read address operation:

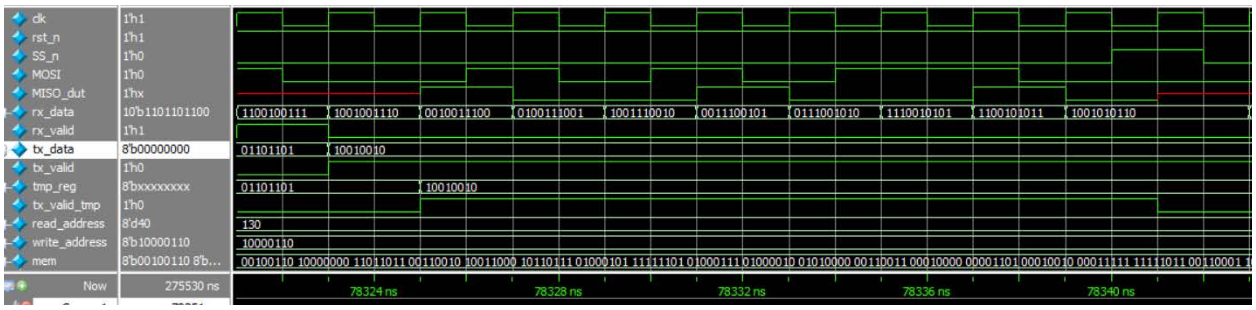


<u>Read data operation and then another read data operation:</u>

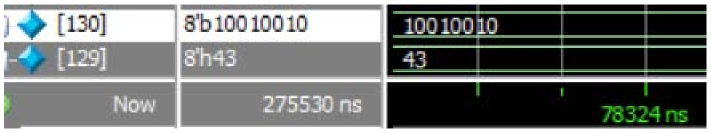
Read data operation 1 (8 cycles of dummy data after which RAM returns tx_valid as raised)



Tx_Data is output on MISO:



Evidence that tx_data is the data in the read address:

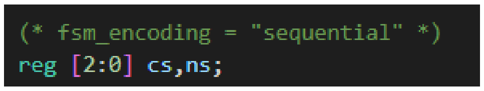


Then there is a for loop that generates random order of operations between the 4 operations. This can be observed on the wave simulation when running Master_tb.

- The SPI slave implementation is done using FSM, we shall try out three different encoding (gray, one_hot or seq) using the following vivado attribute in your Verilog code

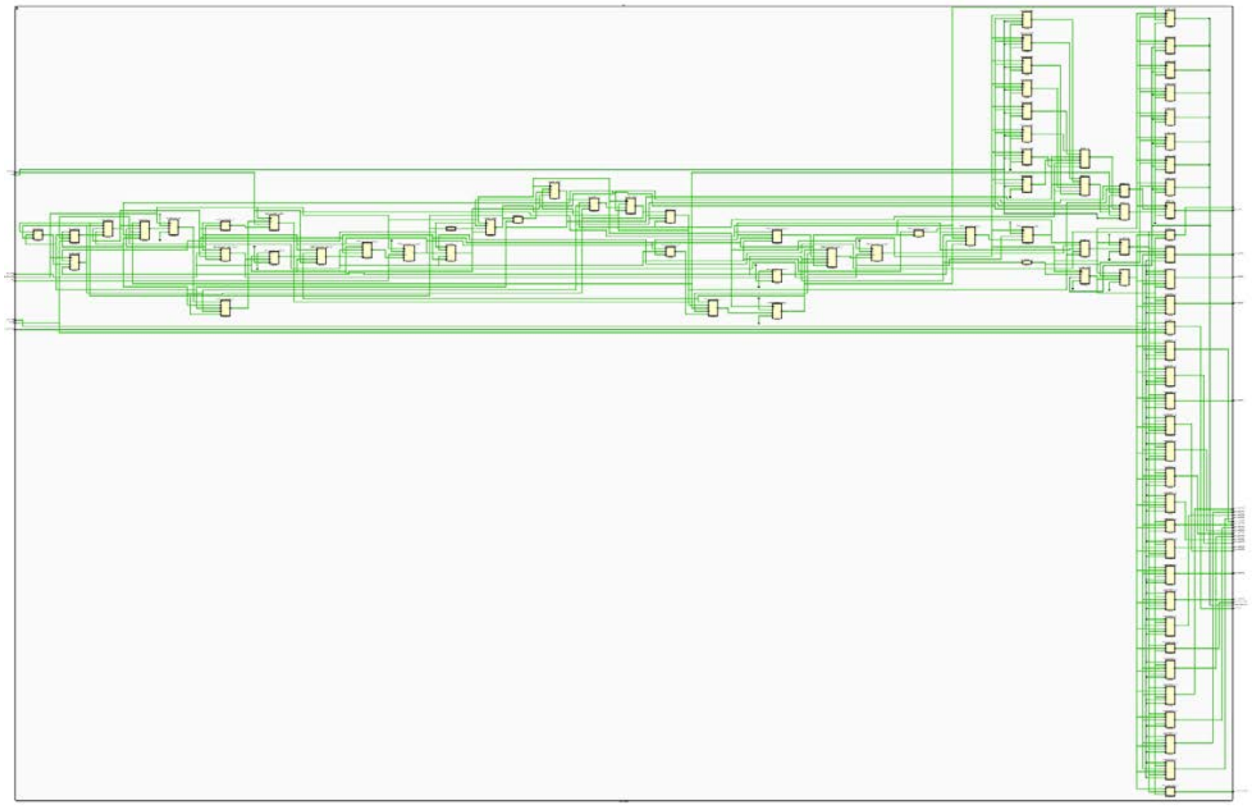
  (* fsm_encoding = "gray" *)

2) Synthesis snippets for each encoding



```
(* fsm_encoding = "sequential" *)
reg [2:0] cs,ns;
```

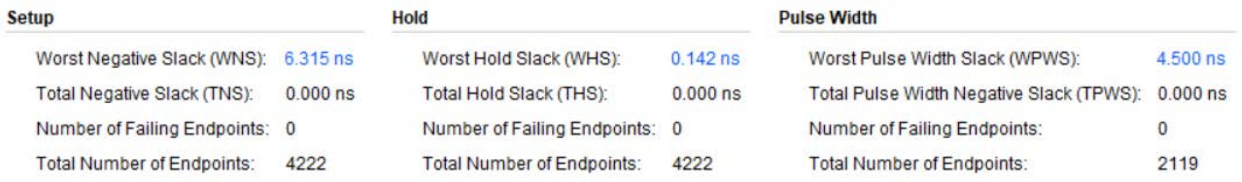
- Schematic after the elaboration & synthesis



- Synthesis report showing the encoding used

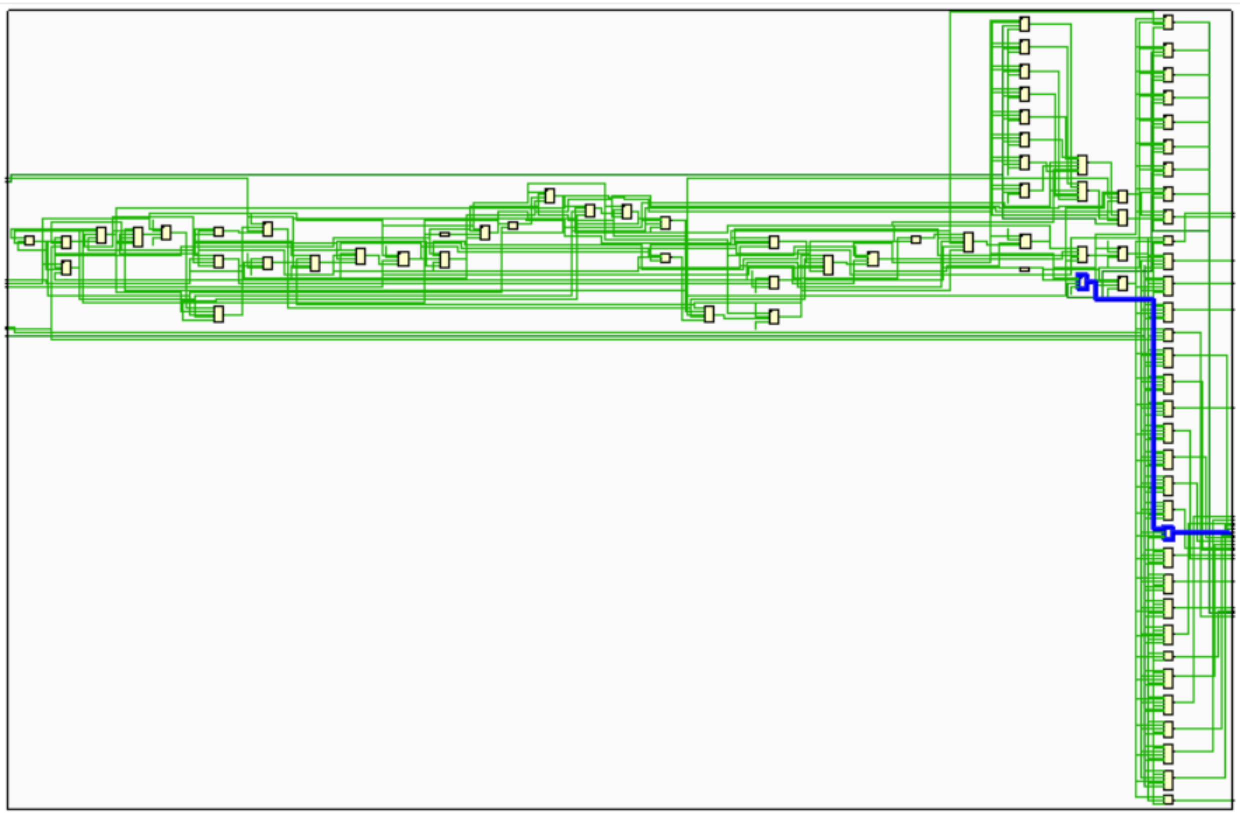
```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "sequential" *)
     State |                     New Encoding |                Previous Encoding
----------------------------------------------------------------------------------------
      IDLE |                              000 |                              000
   CHK_CMD |                              001 |                              001
     WRITE |                              010 |                              010
  READ_ADD |                              011 |                              011
 READ_DATA |                              100 |                              100
```

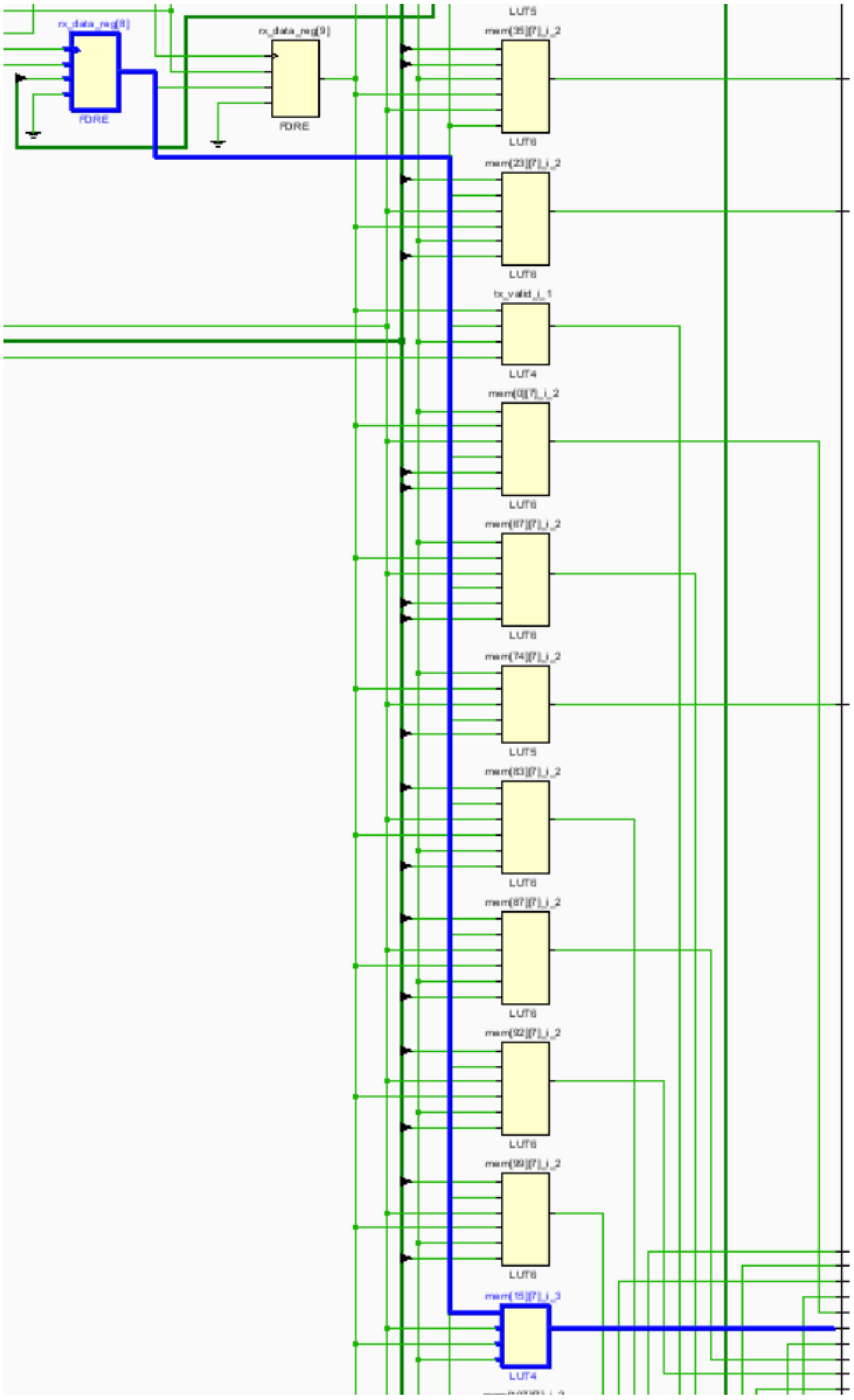
- Timing report snippet

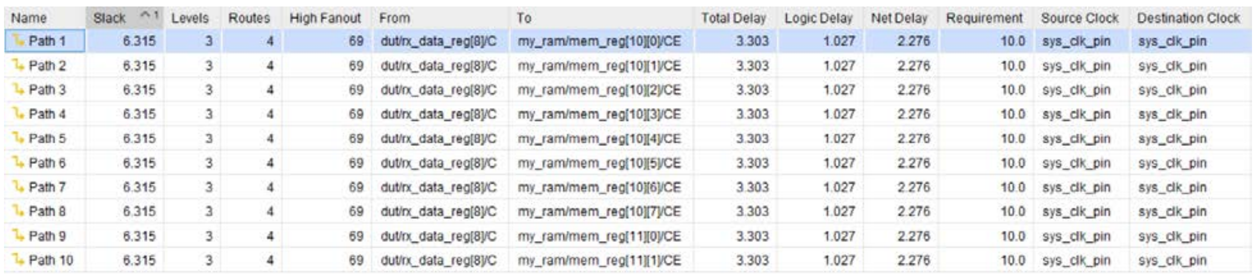


Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.315 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4222	Total Number of Endpoints:	4222	Total Number of Endpoints:	2119

- Snippet of the critical path highlighted in the schematic



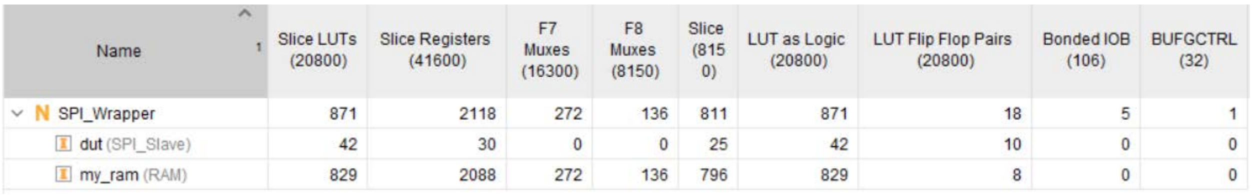

LUT5
rx_data_reg[8]
FDRE
rx_data_reg[9]
FDRE
mem[35][7]_i_2
LUT6
mem[23][7]_i_2
LUT6
tx_valid_i_1
LUT4
mem[0][7]_i_2
LUT6
mem[67][7]_i_2
LUT6
mem[74][7]_i_2
LUT5
mem[83][7]_i_2
LUT6
mem[87][7]_i_2
LUT6
mem[92][7]_i_2
LUT6
mem[99][7]_i_2
LUT6
mem[15][7]_i_3
LUT4



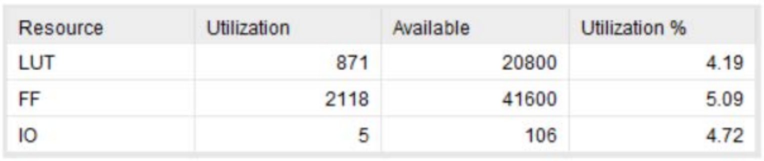
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][0]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 2	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][1]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 3	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][2]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 4	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][3]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 5	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][4]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 6	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][5]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 7	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][6]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 8	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][7]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 9	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[11][0]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin
Path 10	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[11][1]/CE	3.303	1.027	2.276	10.0	sys_clk_pin	sys_clk_pin

3) Implementation snippets for each encoding

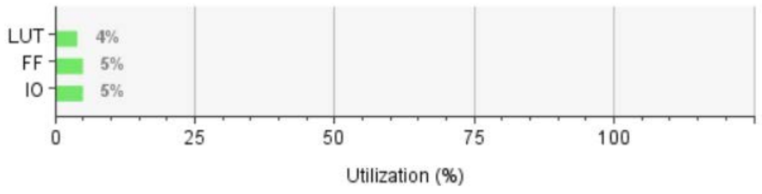
- Utilization report



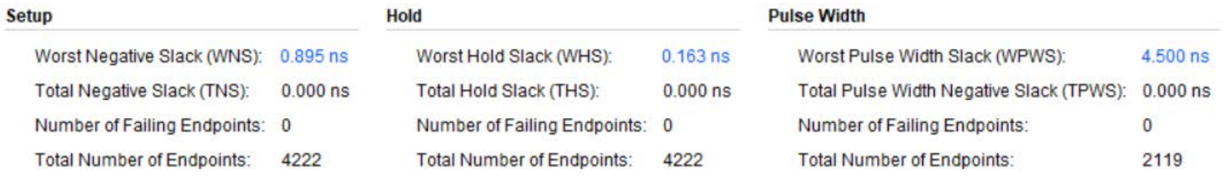
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
SPI_Wrapper	871	2118	272	136	811	871	18	5	1
dut (SPI_Slave)	42	30	0	0	25	42	10	0	0
my_ram (RAM)	829	2088	272	136	796	829	8	0	0



Resource	Utilization	Available	Utilization %
LUT	871	20800	4.19
FF	2118	41600	5.09
IO	5	106	4.72

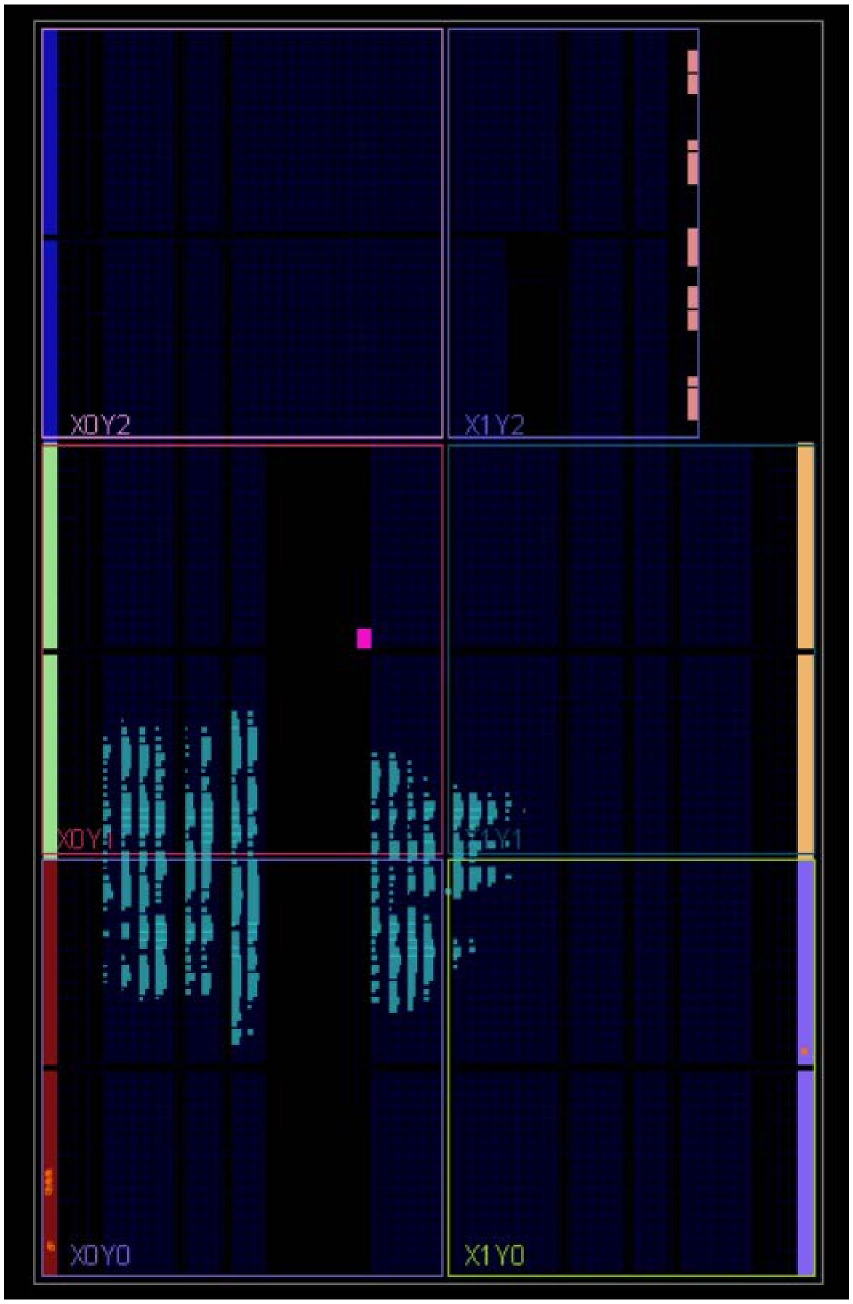


- Timing report snippet

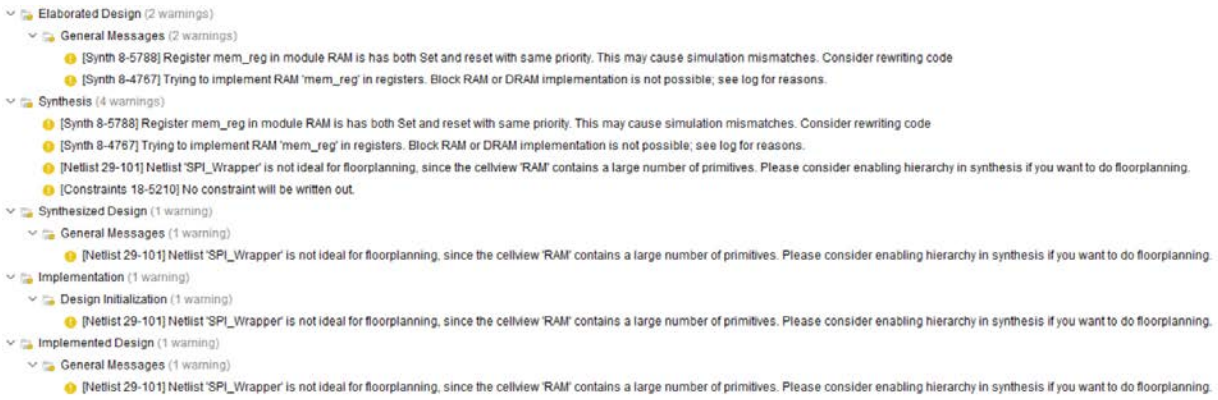


Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.895 ns	Worst Hold Slack (WHS):	0.163 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4222	Total Number of Endpoints:	4222	Total Number of Endpoints:	2119

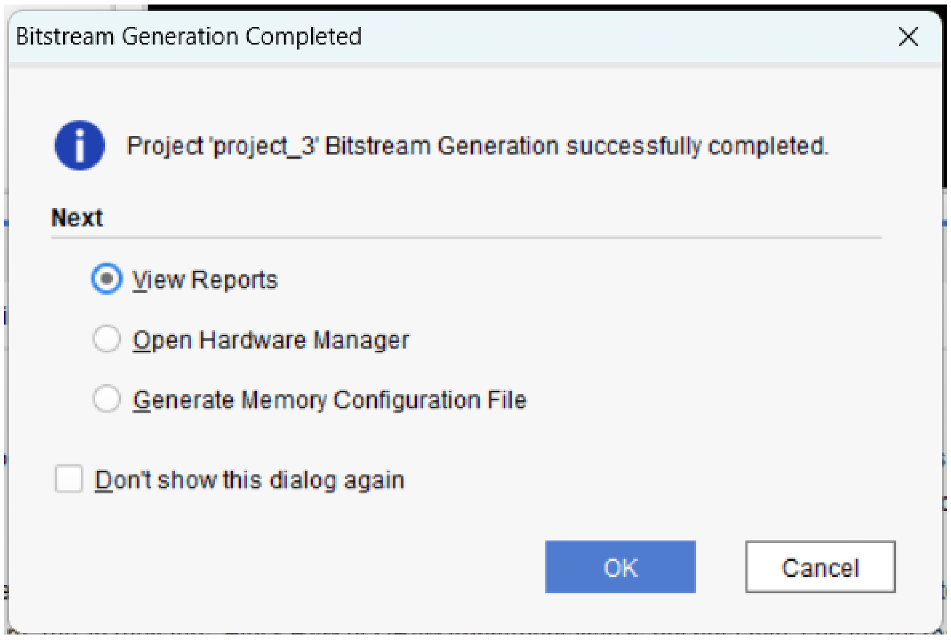
- FPGA device snippet



4) Snippet of the “Messages” tab showing no critical warnings or errors after running elaboration, synthesis, implementation and a successful bitstream generation.



```
Elaborated Design (2 warnings)
  General Messages (2 warnings)
    [Synth 8-5788] Register mem_reg in module RAM is has both Set and reset with same priority. This may cause simulation mismatches. Consider rewriting code
    [Synth 8-4767] Trying to implement RAM 'mem_reg' in registers. Block RAM or DRAM implementation is not possible; see log for reasons.
Synthesis (4 warnings)
  [Synth 8-5788] Register mem_reg in module RAM is has both Set and reset with same priority. This may cause simulation mismatches. Consider rewriting code
  [Synth 8-4767] Trying to implement RAM 'mem_reg' in registers. Block RAM or DRAM implementation is not possible; see log for reasons.
  [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
  [Constraints 18-5210] No constraint will be written out.
Synthesized Design (1 warning)
  General Messages (1 warning)
    [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
Implementation (1 warning)
  Design Initialization (1 warning)
    [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
Implemented Design (1 warning)
  General Messages (1 warning)
    [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
```



Bitstream Generation Completed

Project 'project_3' Bitstream Generation successfully completed.

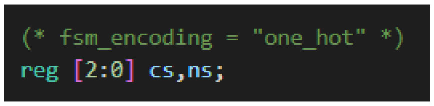
Next

- [✓ View Reports]
- Open Hardware Manager
- Generate Memory Configuration File

☐ Don't show this dialog again

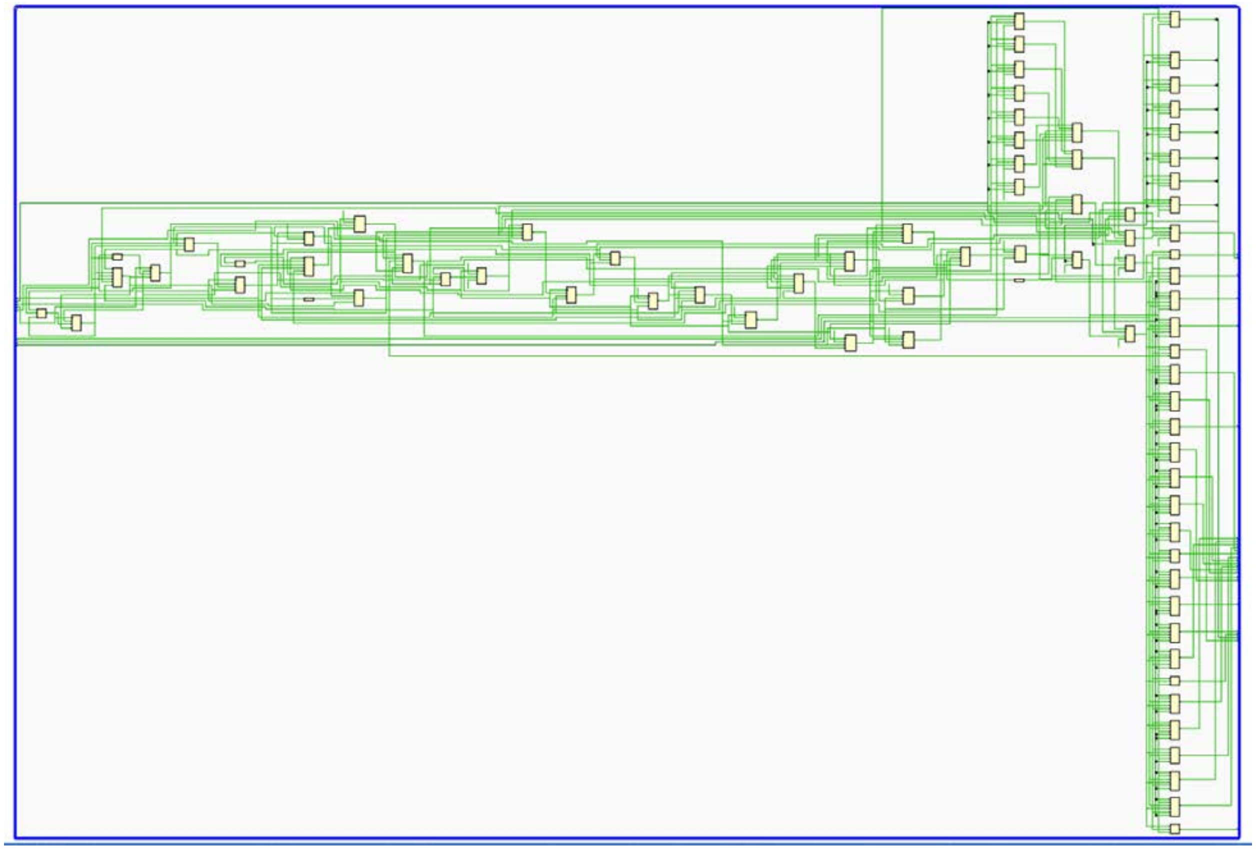
OK | Cancel

2) Synthesis snippets for each encoding

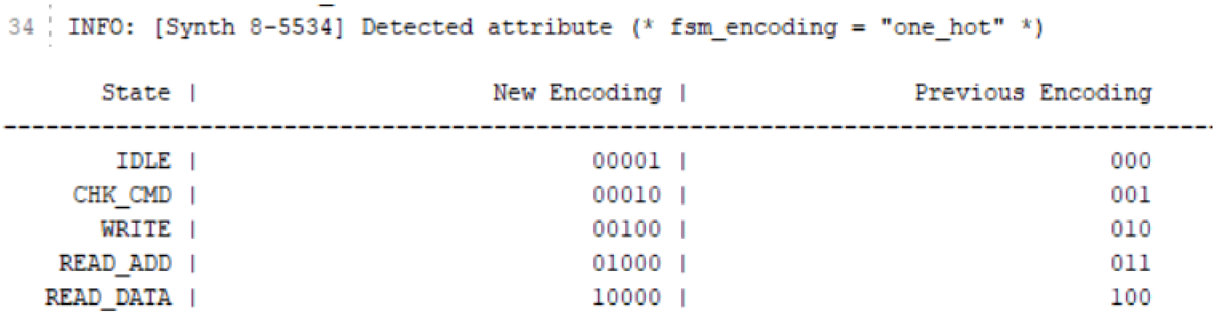


```
(* fsm_encoding = "one_hot" *)
reg [2:0] cs,ns;
```

- Schematic after the elaboration & synthesis

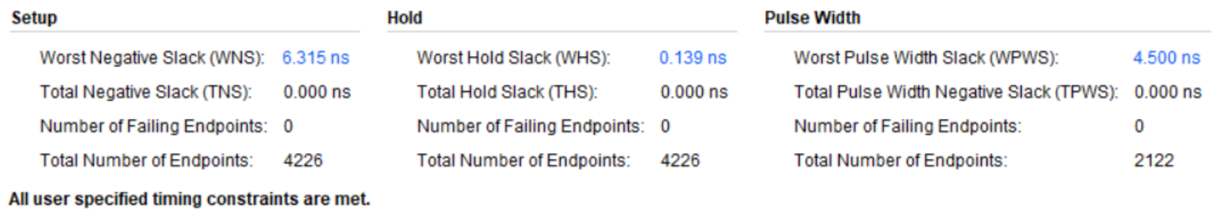


- Synthesis report showing the encoding used



```
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "one_hot" *)

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

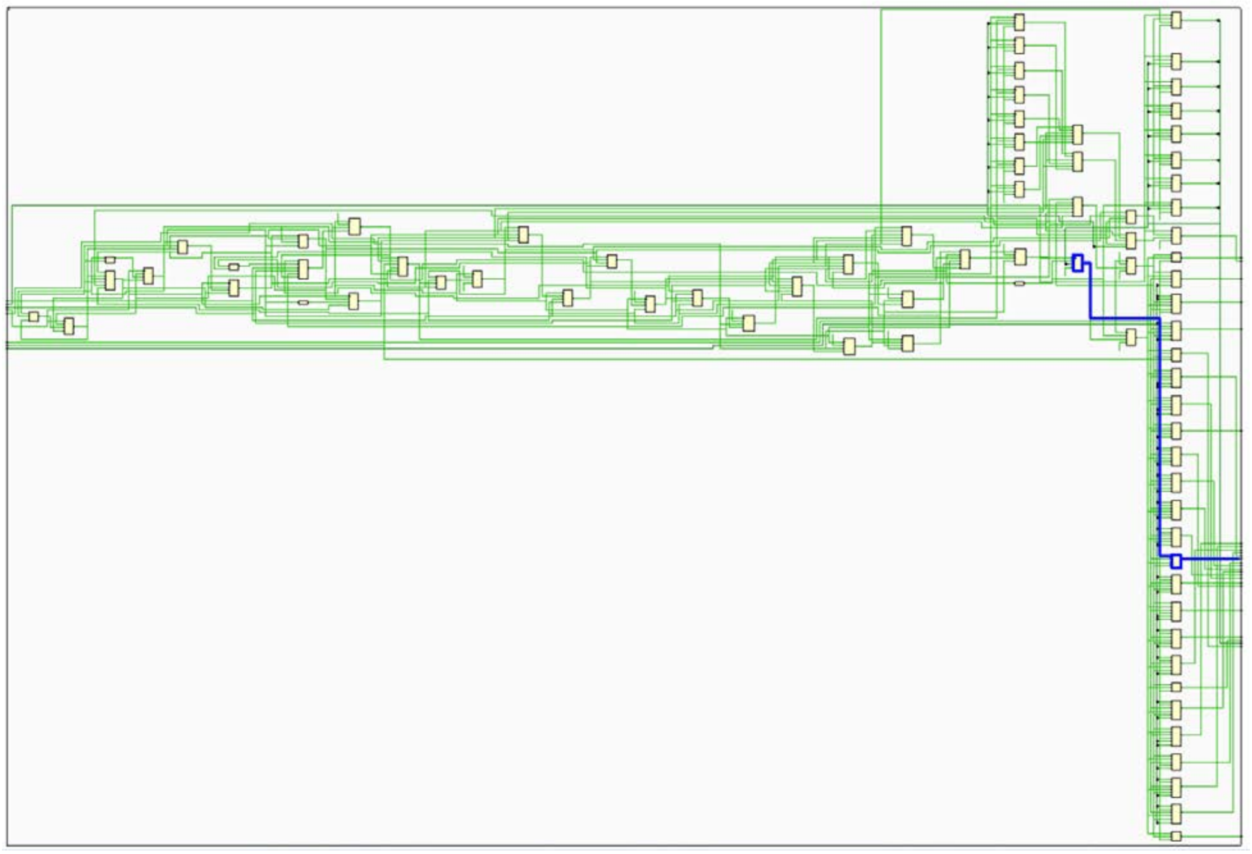
- Timing report snippet

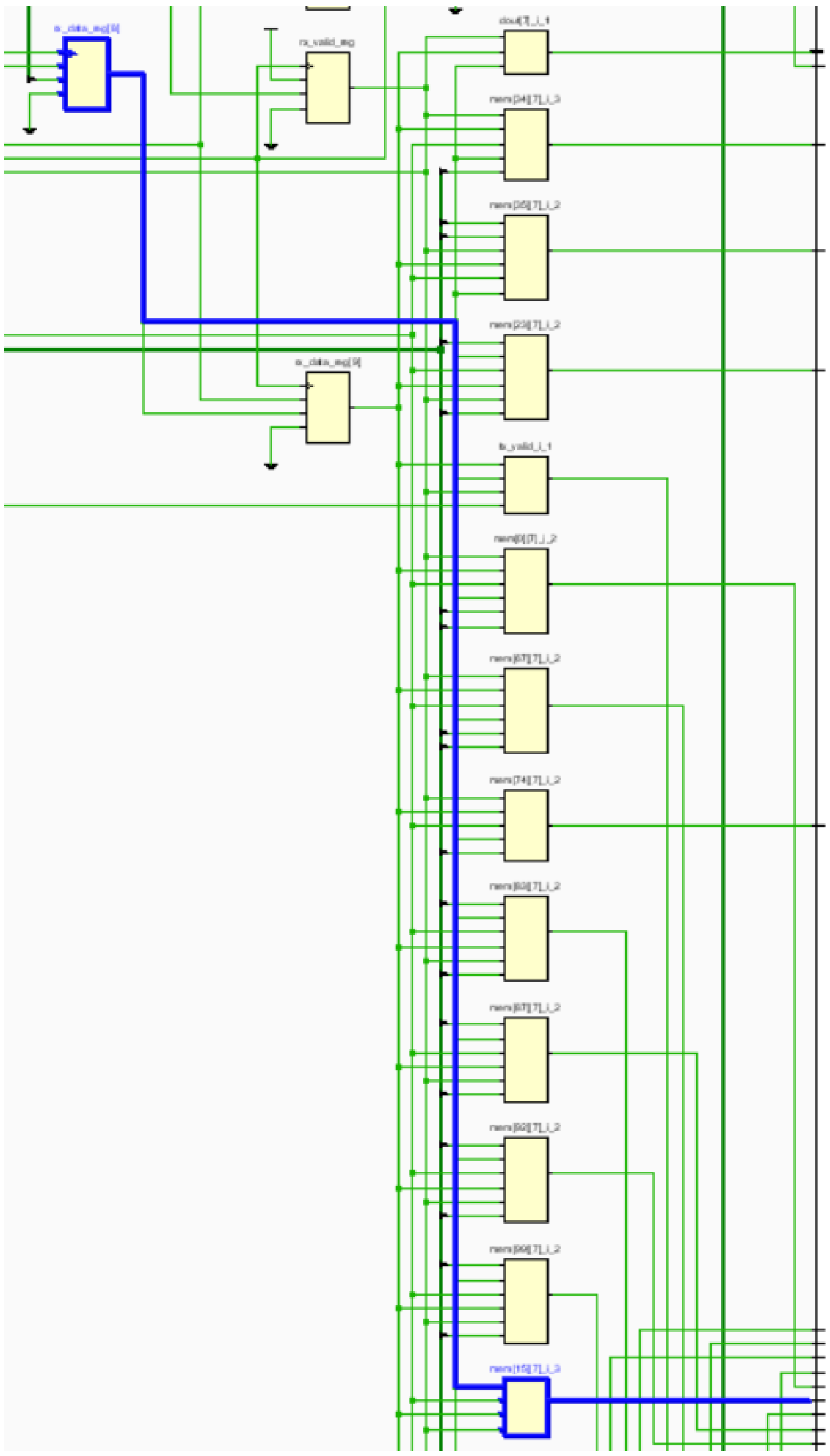


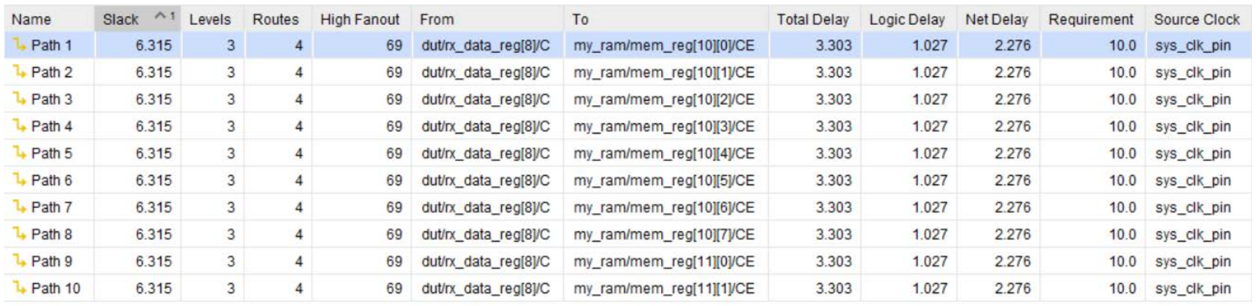
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.315 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4226	Total Number of Endpoints:	4226	Total Number of Endpoints:	2122

All user specified timing constraints are met.

- Snippet of the critical path highlighted in the schematic



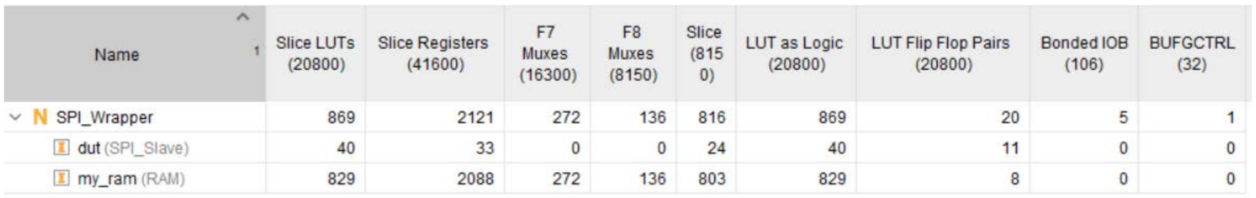

o_data_reg[9]
rx_valid_reg
dout[7]_i_1
mem[24][7]_i_3
mem[35][7]_i_2
mem[23][7]_i_2
o_data_reg[3]
tx_valid_i_1
mem[0][7]_i_2
mem[67][7]_i_2
mem[74][7]_i_2
mem[63][7]_i_2
mem[67][7]_i_2
mem[92][7]_i_2
mem[99][7]_i_2
mem[15][7]_i_3



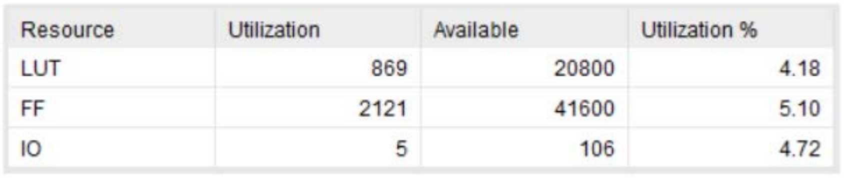
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][0]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 2	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][1]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 3	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][2]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 4	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][3]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 5	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][4]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 6	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][5]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 7	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][6]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 8	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][7]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 9	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[11][0]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 10	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[11][1]/CE	3.303	1.027	2.276	10.0	sys_clk_pin

3) Implementation snippets for each encoding

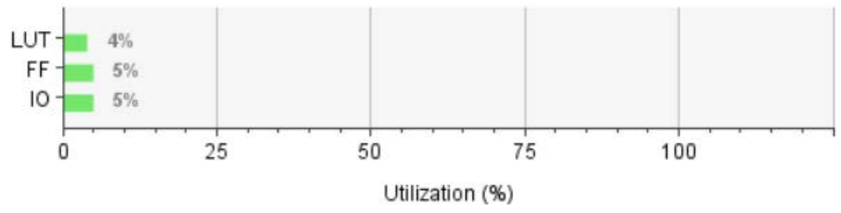
- Utilization report



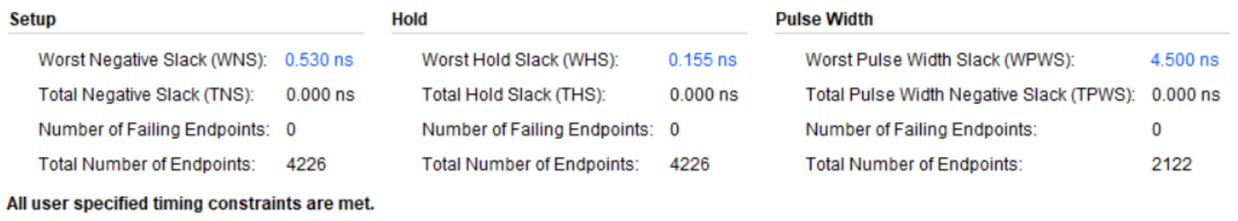
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
SPI_Wrapper	869	2121	272	136	816	869	20	5	1
dut (SPI_Slave)	40	33	0	0	24	40	11	0	0
my_ram (RAM)	829	2088	272	136	803	829	8	0	0



Resource	Utilization	Available	Utilization %
LUT	869	20800	4.18
FF	2121	41600	5.10
IO	5	106	4.72



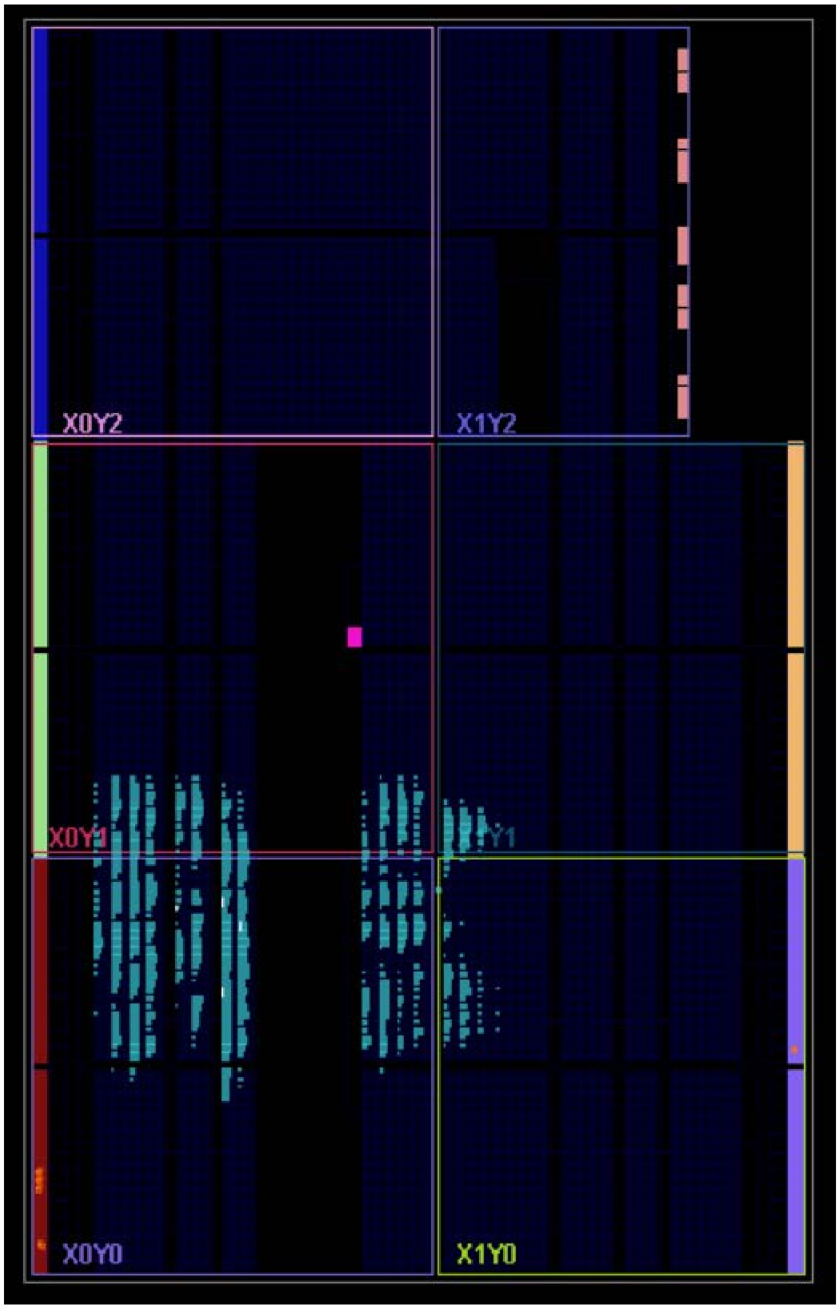
- Timing report snippet



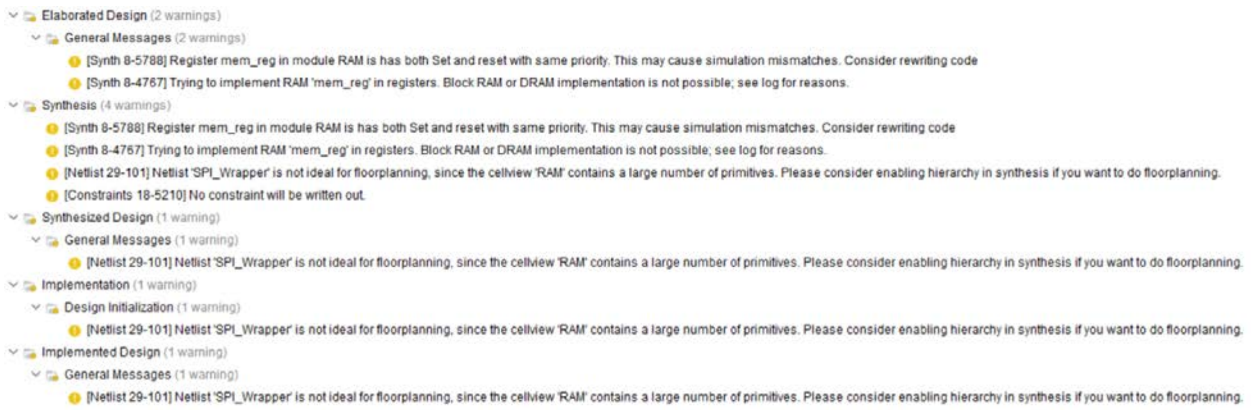
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.530 ns	Worst Hold Slack (WHS):	0.155 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4226	Total Number of Endpoints:	4226	Total Number of Endpoints:	2122

All user specified timing constraints are met.

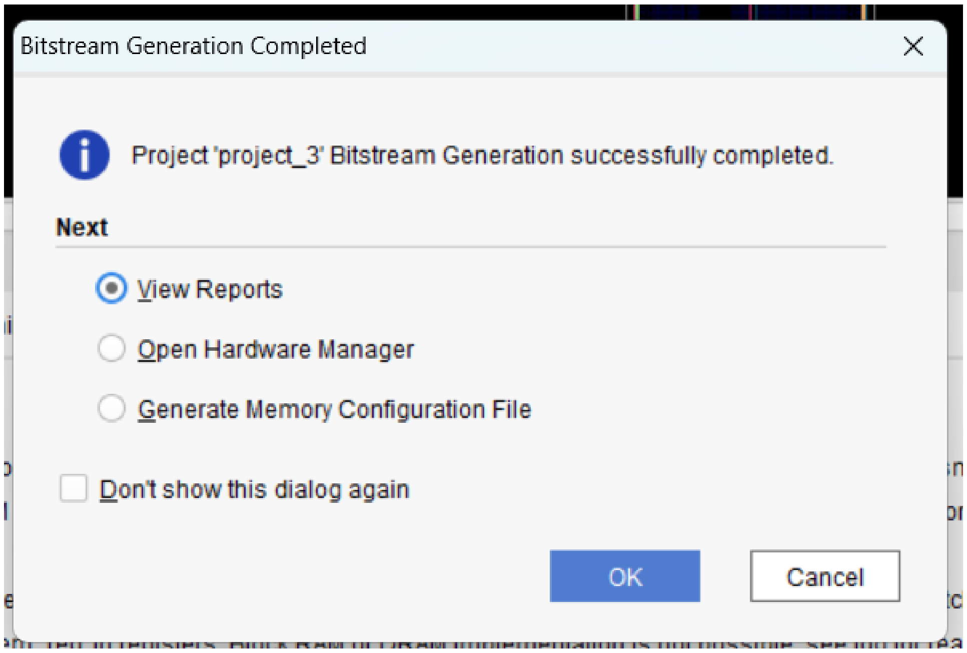
- FPGA device snippet



4) Snippet of the “Messages” tab showing no critical warnings or errors after running elaboration, synthesis, implementation and a successful bitstream generation.



```
Elaborated Design (2 warnings)
    General Messages (2 warnings)
        [Synth 8-5788] Register mem_reg in module RAM is has both Set and reset with same priority. This may cause simulation mismatches. Consider rewriting code
        [Synth 8-4767] Trying to implement RAM 'mem_reg' in registers. Block RAM or DRAM implementation is not possible; see log for reasons.
Synthesis (4 warnings)
    [Synth 8-5788] Register mem_reg in module RAM is has both Set and reset with same priority. This may cause simulation mismatches. Consider rewriting code
    [Synth 8-4767] Trying to implement RAM 'mem_reg' in registers. Block RAM or DRAM implementation is not possible; see log for reasons.
    [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
    [Constraints 18-5210] No constraint will be written out.
Synthesized Design (1 warning)
    General Messages (1 warning)
        [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
Implementation (1 warning)
    Design Initialization (1 warning)
        [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
Implemented Design (1 warning)
    General Messages (1 warning)
        [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
```



Bitstream Generation Completed

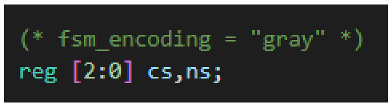
Project 'project_3' Bitstream Generation successfully completed.

Next

- [✓ View Reports]
- Open Hardware Manager
- Generate Memory Configuration File

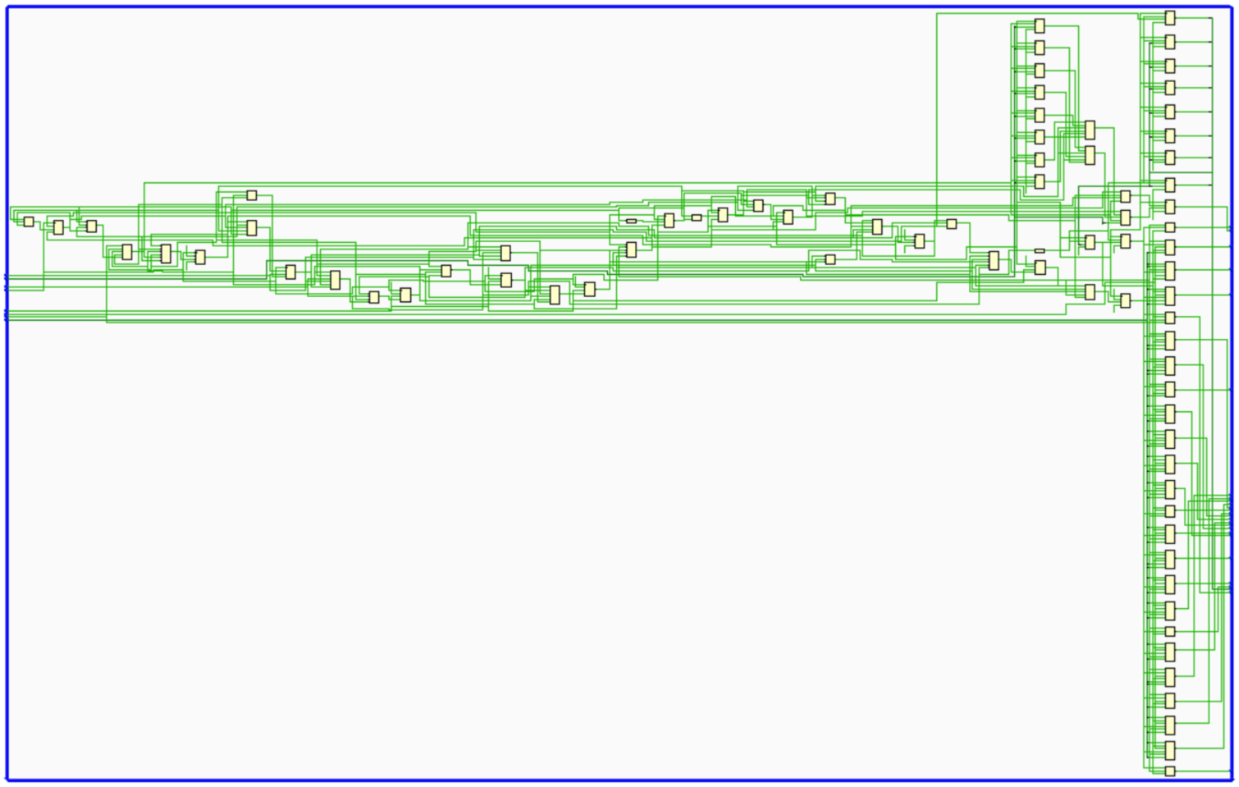
Don't show this dialog again

2) Synthesis snippets for each encoding

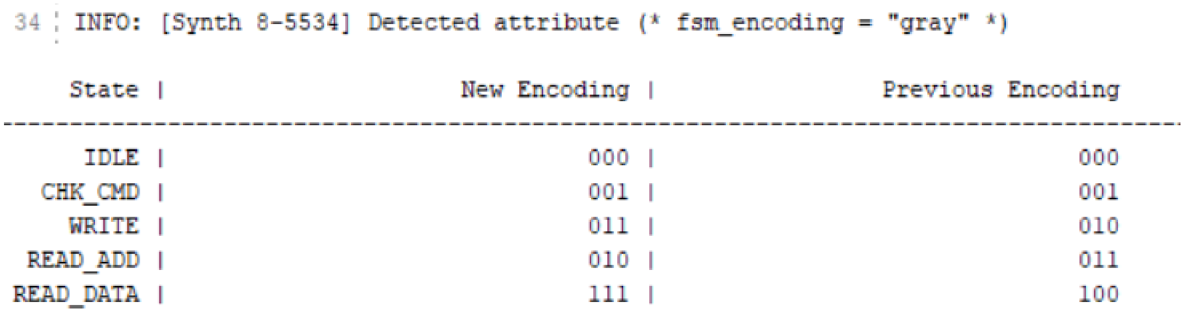


```
(* fsm_encoding = "gray" *)
reg [2:0] cs,ns;
```

- Schematic after the elaboration & synthesis

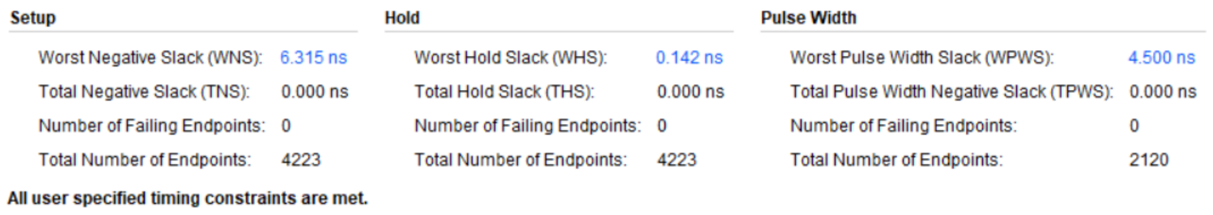


- Synthesis report showing the encoding used



```
34 | INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *)

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

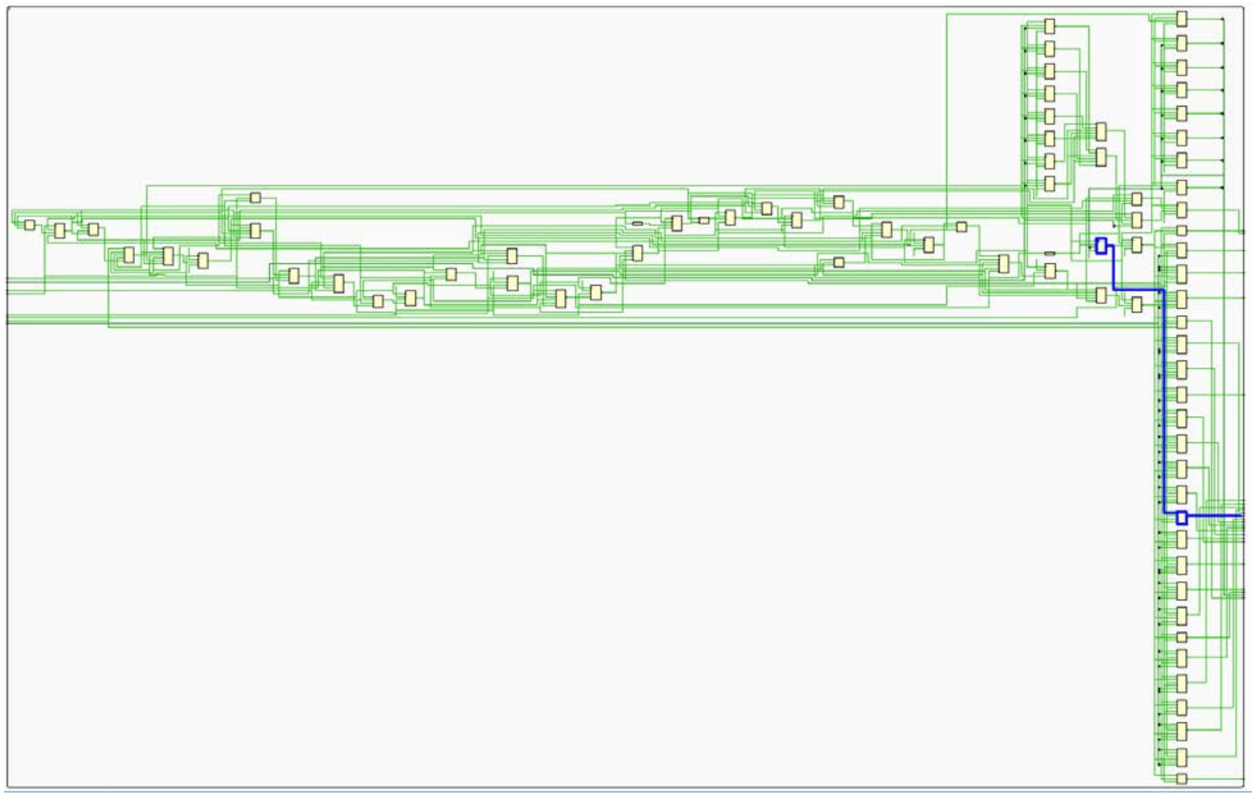
- Timing report snippet

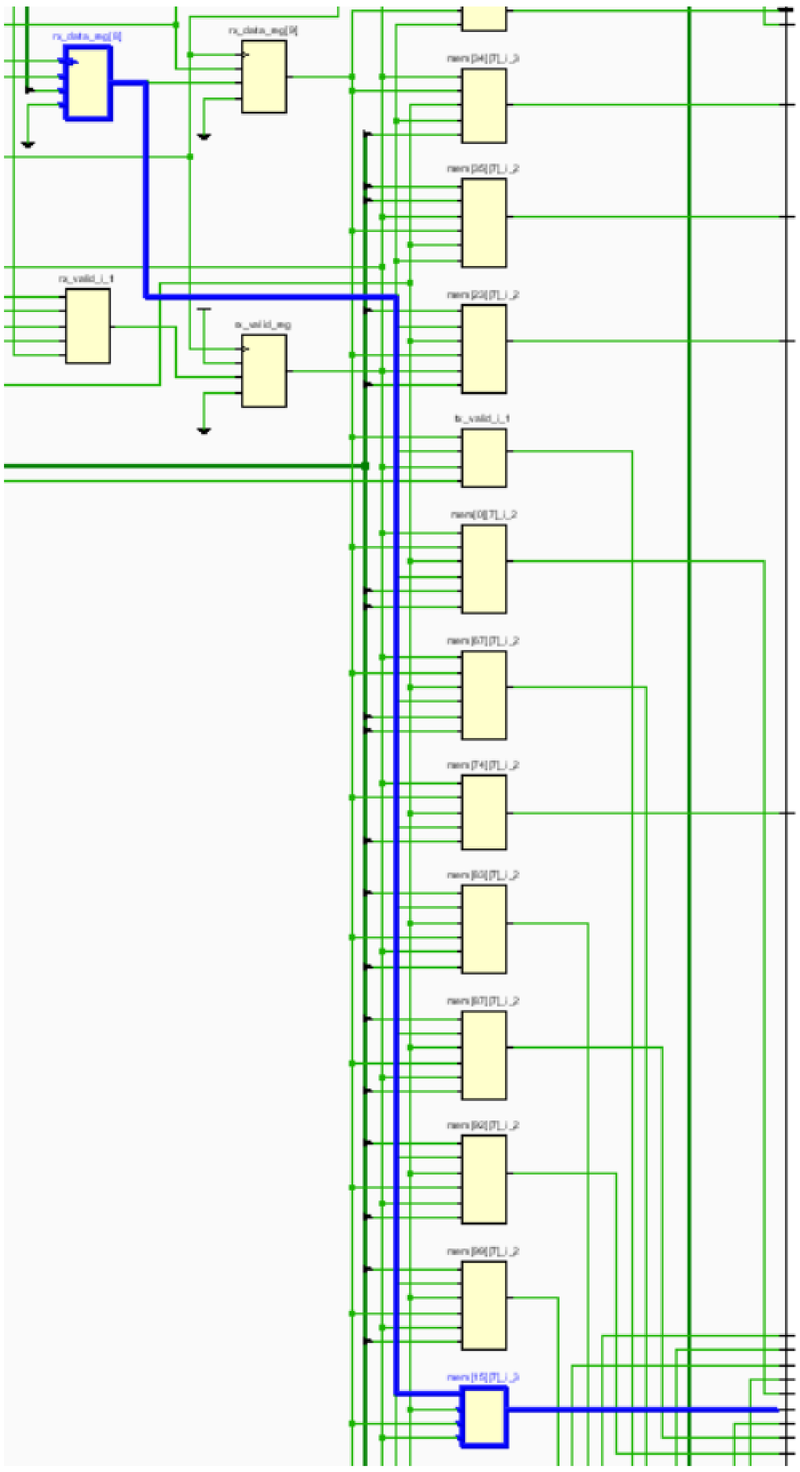


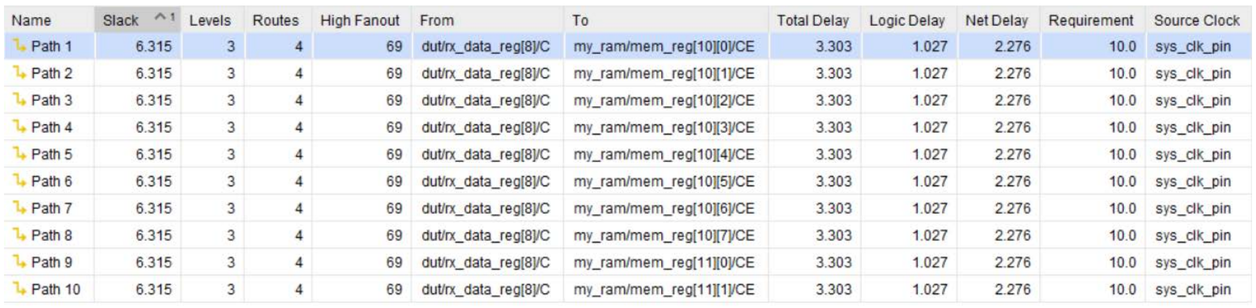
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.315 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4223	Total Number of Endpoints:	4223	Total Number of Endpoints:	2120

All user specified timing constraints are met.

- Snippet of the critical path highlighted in the schematic



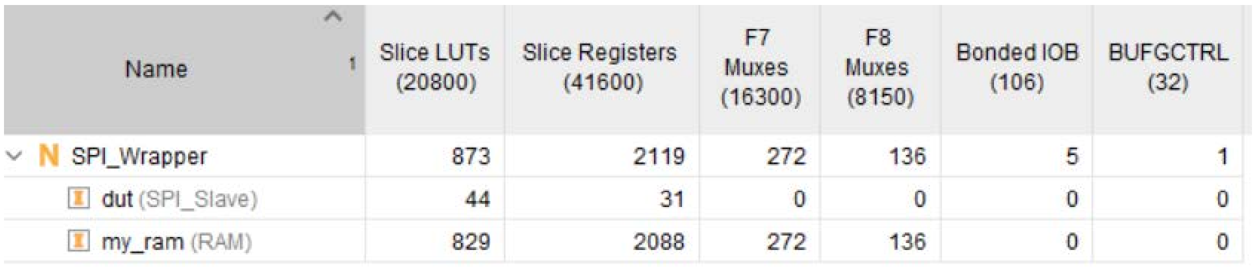

rx_data_reg[8]
rx_data_reg[9]
mem[24][7]_i_3
mem[25][7]_i_2
rx_valid_i_1
mem[23][7]_i_2
rx_valid_reg
tx_valid_i_1
mem[0][7]_i_2
mem[67][7]_i_2
mem[74][7]_i_2
mem[83][7]_i_2
mem[87][7]_i_2
mem[92][7]_i_2
mem[99][7]_i_2
mem[15][7]_i_3



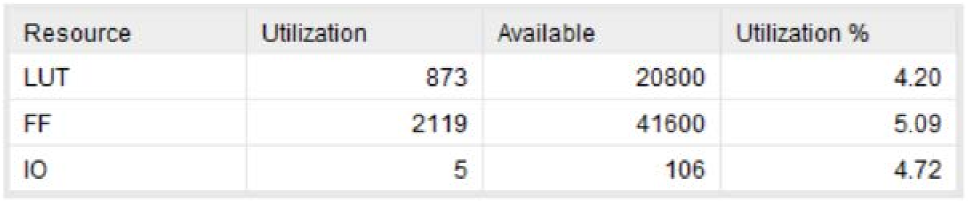
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][0]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 2	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][1]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 3	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][2]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 4	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][3]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 5	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][4]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 6	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][5]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 7	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][6]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 8	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[10][7]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 9	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[11][0]/CE	3.303	1.027	2.276	10.0	sys_clk_pin
Path 10	6.315	3	4	69	dut/rx_data_reg[8]/C	my_ram/mem_reg[11][1]/CE	3.303	1.027	2.276	10.0	sys_clk_pin

3) Implementation snippets for each encoding

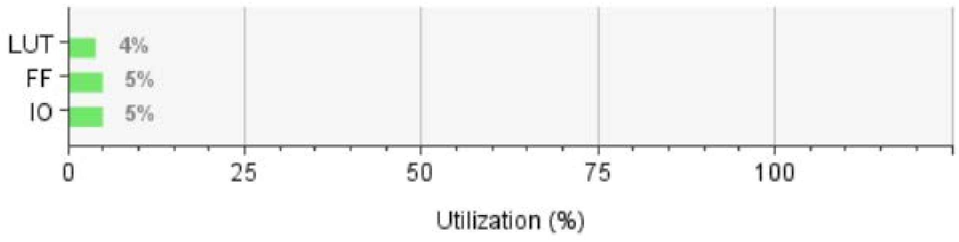
- Utilization report



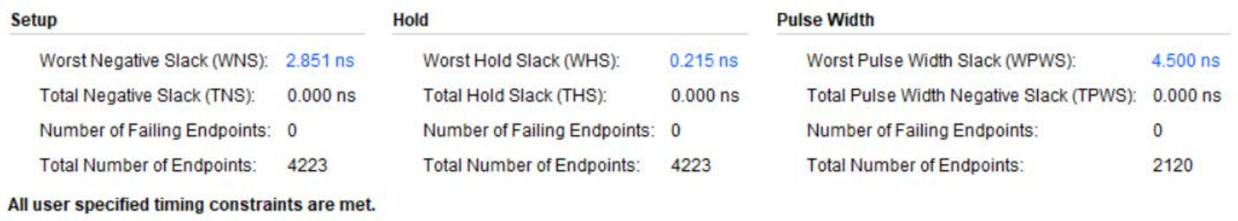
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (106)	BUFGCTRL (32)
SPI_Wrapper	873	2119	272	136	5	1
dut (SPI_Slave)	44	31	0	0	0	0
my_ram (RAM)	829	2088	272	136	0	0



Resource	Utilization	Available	Utilization %
LUT	873	20800	4.20
FF	2119	41600	5.09
IO	5	106	4.72



- Timing report snippet



Setup

Worst Negative Slack (WNS):	2.851 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	4223

Hold

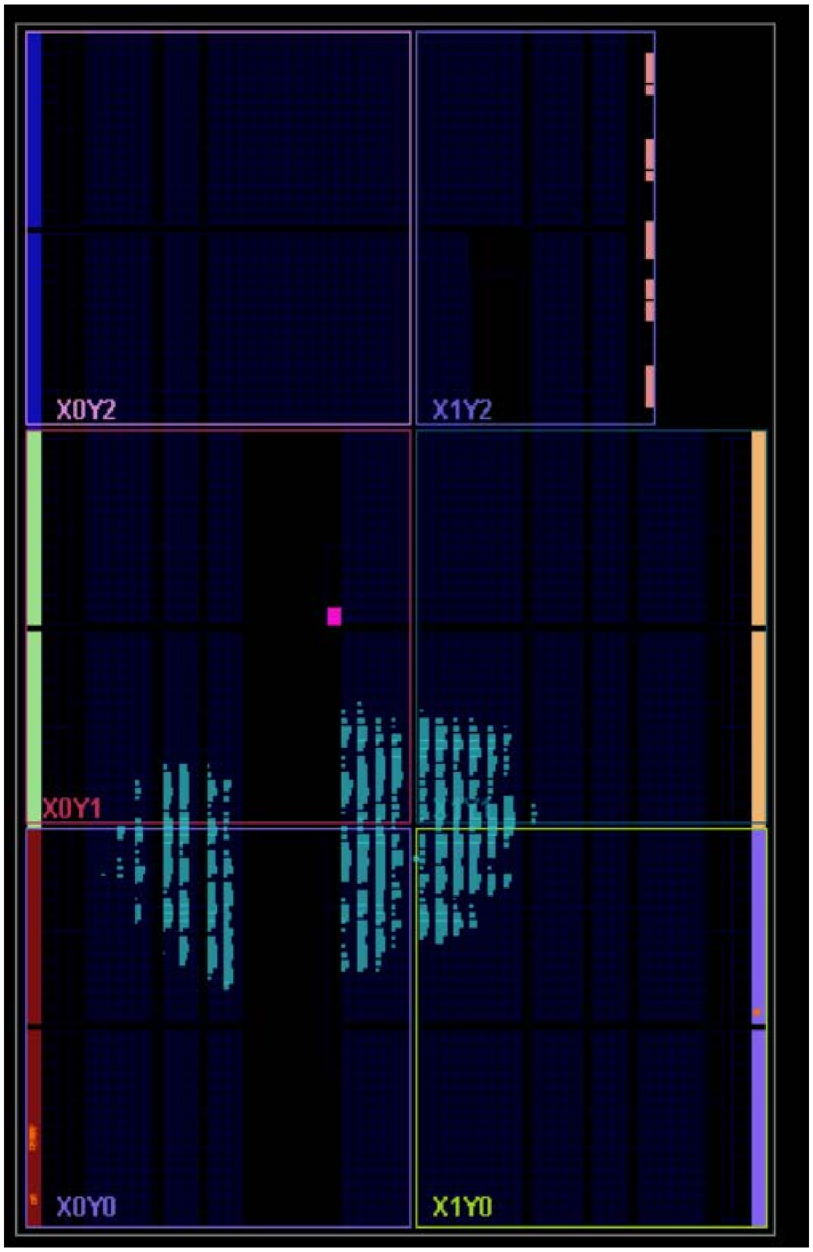
Worst Hold Slack (WHS):	0.215 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	4223

Pulse Width

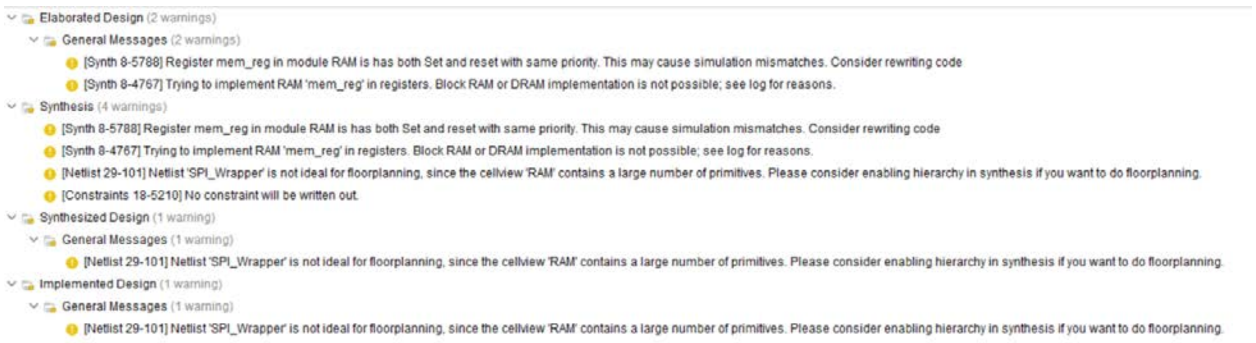
Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	2120

All user specified timing constraints are met.

- FPGA device snippet



4) Snippet of the “Messages” tab showing no critical warnings or errors after running elaboration, synthesis, implementation and a successful bitstream generation.

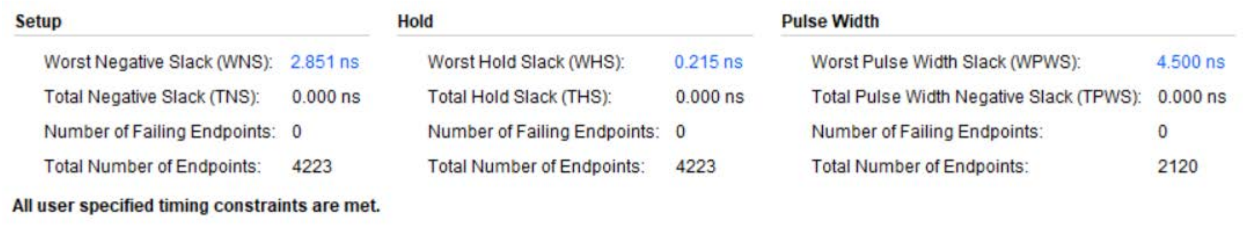


```
Elaborated Design (2 warnings)
  General Messages (2 warnings)
    [Synth 8-5788] Register mem_reg in module RAM is has both Set and reset with same priority. This may cause simulation mismatches. Consider rewriting code
    [Synth 8-4767] Trying to implement RAM 'mem_reg' in registers. Block RAM or DRAM implementation is not possible; see log for reasons.
Synthesis (4 warnings)
  [Synth 8-5788] Register mem_reg in module RAM is has both Set and reset with same priority. This may cause simulation mismatches. Consider rewriting code
  [Synth 8-4767] Trying to implement RAM 'mem_reg' in registers. Block RAM or DRAM implementation is not possible; see log for reasons.
  [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
  [Constraints 18-5210] No constraint will be written out.
Synthesized Design (1 warning)
  General Messages (1 warning)
    [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
Implemented Design (1 warning)
  General Messages (1 warning)
    [Netlist 29-101] Netlist 'SPI_Wrapper' is not ideal for floorplanning, since the cellview 'RAM' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.
```

- We wish to operate at the highest frequency possible and so you shall choose the encoding based on the best timing report that gives the high setup slack after implementation.

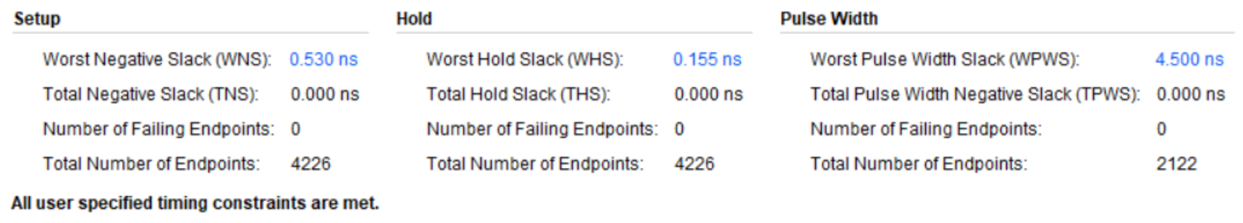
(Copied timing reports here for ease of comparison)

```
(* fsm_encoding = "gray" *)
reg [2:0] cs,ns;
```



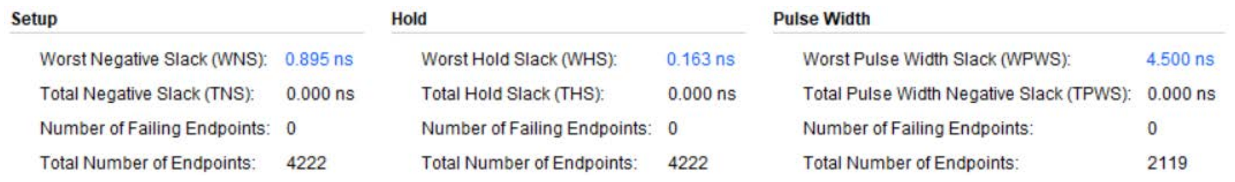
```
Setup                                          Hold                                          Pulse Width

  Worst Negative Slack (WNS):     2.851 ns       Worst Hold Slack (WHS):         0.215 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:      4223           Total Number of Endpoints:      4223           Total Number of Endpoints:                  2120

All user specified timing constraints are met.
```

```
(* fsm_encoding = "one_hot" *)
reg [2:0] cs,ns;
```



```
Setup                                          Hold                                          Pulse Width

  Worst Negative Slack (WNS):     0.530 ns       Worst Hold Slack (WHS):         0.155 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:      4226           Total Number of Endpoints:      4226           Total Number of Endpoints:                  2122

All user specified timing constraints are met.
```

```
(* fsm_encoding = "seq" *)
reg [2:0] cs,ns;
```

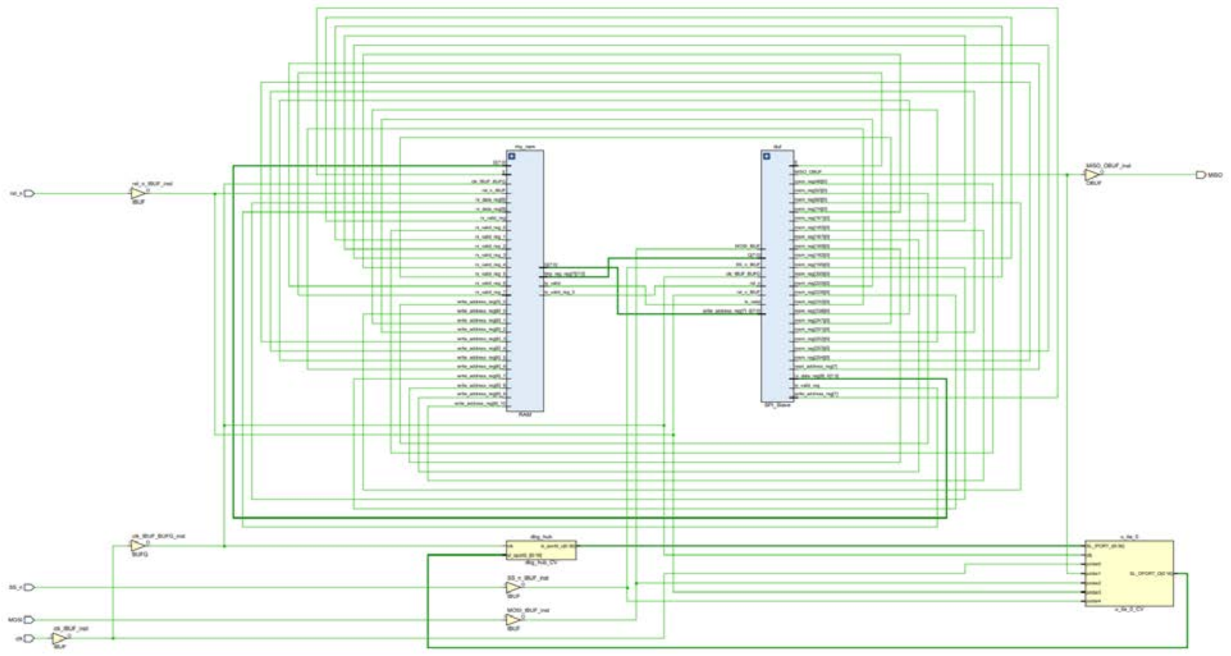


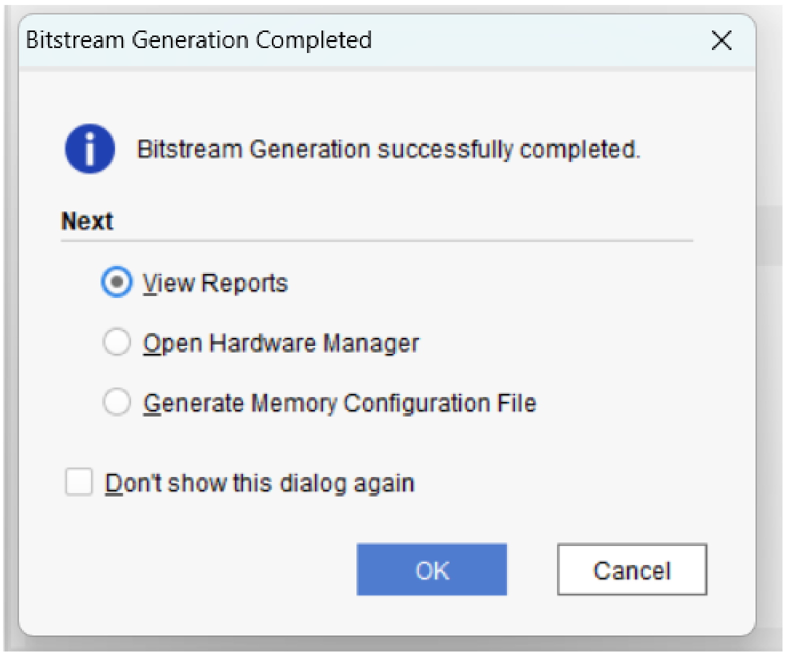
```
Setup                                          Hold                                          Pulse Width

  Worst Negative Slack (WNS):     0.895 ns       Worst Hold Slack (WHS):         0.163 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:      4222           Total Number of Endpoints:      4222           Total Number of Endpoints:                  2119
```

Highest setup slack is observed to be in the case of using **gray encoding**.

- After choosing the best encoding, add a debug core such that all internals (MISO, MOSI, SS_n, rst_n & clk) can be analyzed and then generate a bitstream file

Schematic after added debug core:





Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

[✓ View Reports]

Open Hardware Manager

Generate Memory Configuration File

Don't show this dialog again