SPI SLAVE WITH SINGLE PORT RAM

Prepared By

Abdalrhman Mohy

Table of Contents

Introduction

Description

The SPI (Serial Peripheral Interface) Slave with Single Port RAM is a digital hardware module designed to facilitate serial communication between a master device and a memory-mapped slave. This design integrates an SPI slave controller with a single-port RAM module, allowing the SPI master to read and write data to the RAM in a controlled and synchronized manner. The SPI slave interface handles the serial protocol—receiving commands and data on the MOSI line, synchronizing with the master through the CLK line, and responding via the MISO line. The Slave Select (SS) signal from the master enables the slave device for communication. Once activated, the slave decodes incoming instructions and interacts with its internal RAM accordingly. This module is written in Verilog HDL and synthesized using Xilinx Vivado, making it suitable for FPGA-based designs requiring compact and efficient memory interfacing.

Features

- SPI Slave Interface : Fully compliant SPI slave logic supporting standard SPI modes.
- Integrated Memory : Single-port RAM enables internal data storage with read/write access.
- Synchronous Operation : All data transactions are synchronized with the SPI clock for timing accuracy.
- Bidirectional Communication : Simultaneous transmission and reception of data over MISO and MOSI.
- Compact Design : Minimal signal line requirements make it suitable for pin-limited devices.
- Scalable : RAM depth can be expanded for additional functionality.

Signal Lines

- MOSI (Master Out Slave In) – Transmits data from the master to the slave.
- MISO (Master In Slave Out) – Transmits data from the slave to the master.
- CLK – Clock signal from the master used to synchronize data transfers.
- SS (Slave Select) – signal used by the master to activate the slave device.

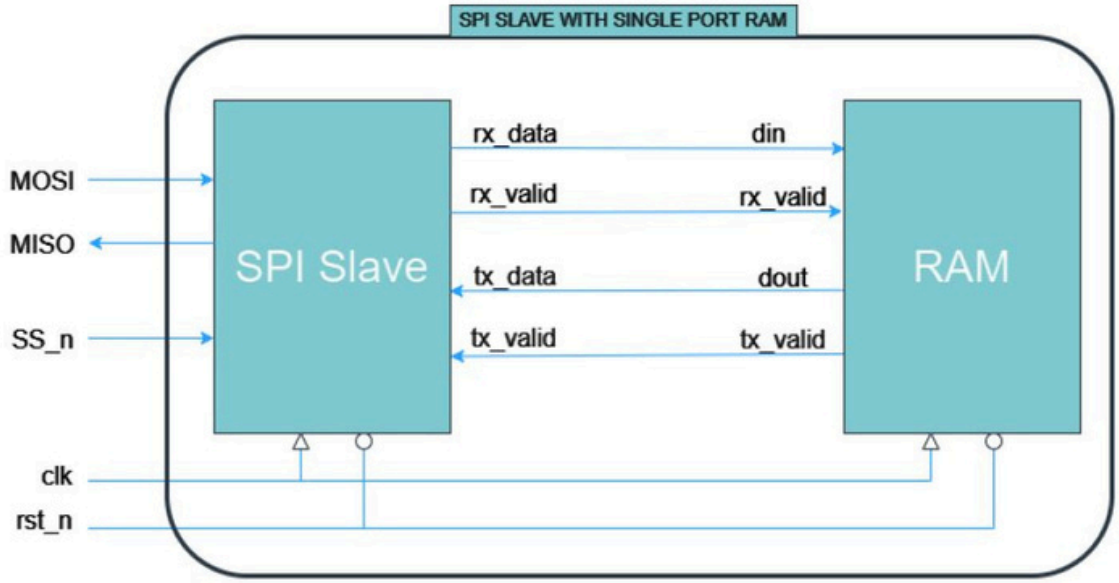
Communication Workflow

- **Initialization** : The SPI master asserts the SS line to enable the slave and sets up SPI mode parameters
- **Command Reception** : The master sends a command over MOSI (e.g., WRITE or READ), which the slave decodes to determine the intended operation.
- **Address & Data Phase** : For memory operations, the master sends a target address followed by data (in the case of a write) or requests data (in the case of a read).
- **RAM Access** : The slave performs the corresponding read or write operation on its internal single-port RAM.
- **Data Transmission** : If reading, the slave sends data back to the master via the MISO line, synchronized with the clock.
- **Transaction Completion** : The master deasserts SS, ending the communication cycle.

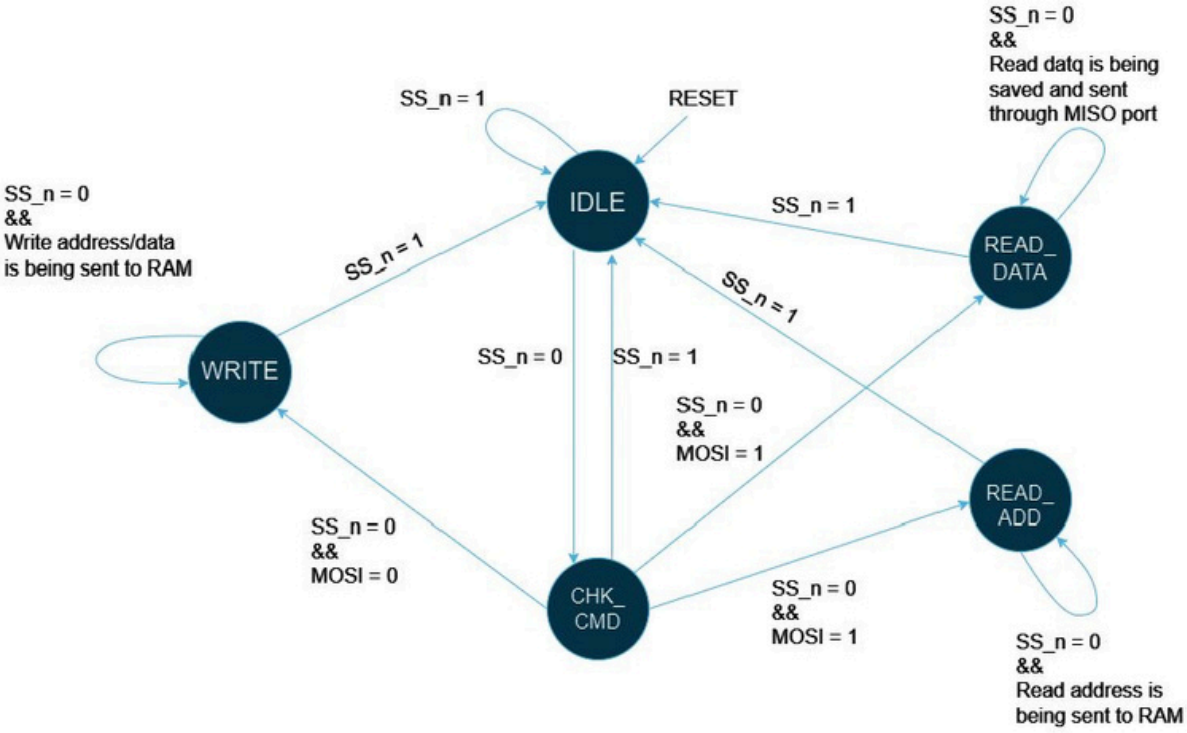
Applications:

- SPI-Controlled RAM Buffers
- Custom Memory-Mapped Peripherals
- Microcontroller–FPGA Communication Interface

SPI Architecture



SPI State diagram



RTL Code

Single port RAM

```
module RAM (
    input clk ,
    input rst_n ,
    input [9:0] din ,
    input rx_valid ,
    output reg [7:0] dout ,
    output reg tx_valid
) ;

parameter  MEM_DEPTH = 256 ;
parameter ADDR_SIZE = 8 ;

reg [7:0] mem [MEM_DEPTH - 1 : 0] ;
reg [ADDR_SIZE - 1: 0] read_add ;
reg [ADDR_SIZE - 1: 0] write_add ;

always @(posedge clk ) begin
    if (!rst_n) begin
        dout <= 0 ;
        tx_valid <= 0 ;
    end
    else begin
        tx_valid <= 0 ;
        if (rx_valid) begin
            case (din [9:8])
                2'b00 : begin
                    write_add <= din [7:0] ;
                end
                2'b01 : begin
                    mem [write_add] <= din [7:0] ;
                end
                2'b10 : begin
                    read_add <= din [7:0] ;
                end
                2'b11 : begin
                    dout <= mem [read_add] ;
                    tx_valid <= 1 ;
                end
            endcase
        end
    end
end
endmodule
```

SPI

```
module SPI (
    input MOSI ,
    input SS_n ,
    input clk ,
    input rst_n ,
    output reg [9:0] rx_data ,
    output reg rx_valid ,
    input  tx_valid ,
    input  [7:0] tx_data ,
    output reg MISO
) ;

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

reg addr_check ;           // To check if available read address
reg [3:0] counter_up  ;
reg [7:0] tx_data_reg ;
reg output_available ;     // High during reading data
(* fsm_encoding = "one_hot" *)
reg [2:0] cs , ns ;       // Current and nesxt state

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

always @ (*) begin        // Next state logic
    case (cs)
        IDLE : begin
            if (SS_n)
                ns = IDLE ;
            else
                ns = CHK_CMD ;
        end
        CHK_CMD : begin
            if (MOSI) begin
                if (addr_check)
                    ns = READ_DATA ;
                else
                    ns = READ_ADD ;
            end
            else
                ns = WRITE ;
        end
        WRITE : begin
            if (SS_n)
                ns = IDLE ;
            else
                ns = WRITE ;
        end
        READ_ADD : begin
            if (SS_n)
                ns = IDLE ;
            else
                ns = READ_ADD ;
        end
        READ_DATA : begin
            if (SS_n)
                ns = IDLE ;
            else
                ns = READ_DATA ;
        end
        default : ns = IDLE ;
    endcase
end
```

```
always @ (posedge clk ) begin   //new output logic
    case (cs)
        IDLE : begin
            rx_valid <= 0 ;
            rx_data <= 0 ;
            counter_up <= 0 ;
            output_available <= 0 ;
            MISO <= 0 ;
        end
        WRITE : begin
            if (counter_up == 9 )  begin
                counter_up <= 0 ;
                rx_valid <= 1 ;
            end
            else begin
                rx_data [counter_up] <= MOSI ;
                counter_up <= counter_up + 1 ;
                rx_valid <= 0 ;
            end
        end
        READ_ADD : begin
            if (counter_up == 9 )  begin
                rx_data [counter_up] <= MOSI ;
                counter_up <= 0 ;
                rx_valid <= 1 ;
            end
            else begin
                rx_data [counter_up] <= MOSI ;
                counter_up <= counter_up + 1 ;
                rx_valid <= 0 ;
                addr_check <= 1 ;
            end
        end
        READ_DATA : begin
            if (tx_valid ) begin
                output_available <= 1 ;
                tx_data_reg <= tx_data ;
                counter_up <= 0 ;
            end
            else if (output_available)begin
                if (counter_up == 7 )  begin
                    MISO <= tx_data_reg  [7 - counter_up ] ;
                    counter_up <= 0 ;
                end
                else begin
                    MISO <= tx_data_reg  [7 - counter_up ] ;
                    counter_up <= counter_up + 1 ;
                end
            end
            else begin
                if (counter_up == 9 )  begin
                    rx_data [counter_up] <= MOSI ;
                    counter_up <= 0 ;
                    rx_valid <= 1 ;
                end
                else begin
                    rx_data [counter_up] <= MOSI ;
                    counter_up <= counter_up + 1 ;
                    rx_valid <= 0 ;
                    addr_check <= 0 ;
                end
            end
        end
    endcase

end
endmodule
```

SPI Wrapper （TOP module ）

```
module SPI_WRAPPER (
    input MOSI ,
    input clk  ,
    input rst_n ,
    input SS_n ,
    output MISO
) ;

parameter  MEM_DEPTH = 256 ;
parameter ADDR_SIZE = 8 ;

wire [9:0] rx_data ;
wire rx_valid ;
wire [7:0] tx_data ;
wire tx_valid ;

SPI SPI_SLAVE_inst (
    .clk(clk),
    .rst_n(rst_n) ,
    .SS_n(SS_n) ,
    .MOSI(MOSI) ,
    .rx_data(rx_data) ,
    .rx_valid (rx_valid) ,
    .tx_valid (tx_valid) ,
    .tx_data (tx_data) ,
    .MISO (MISO)
) ;

RAM RAM_inst #(.MEM_DEPTH(MEM_DEPTH ) ,.ADDR_SIZE(ADDR_SIZE) ) (
    .clk(clk) ,
    .rst_n (rst_n) ,
    .din(rx_data) ,
    .rx_valid(rx_valid) ,
    .dout(tx_data) ,
    .tx_valid(tx_valid)
) ;

endmodule
```

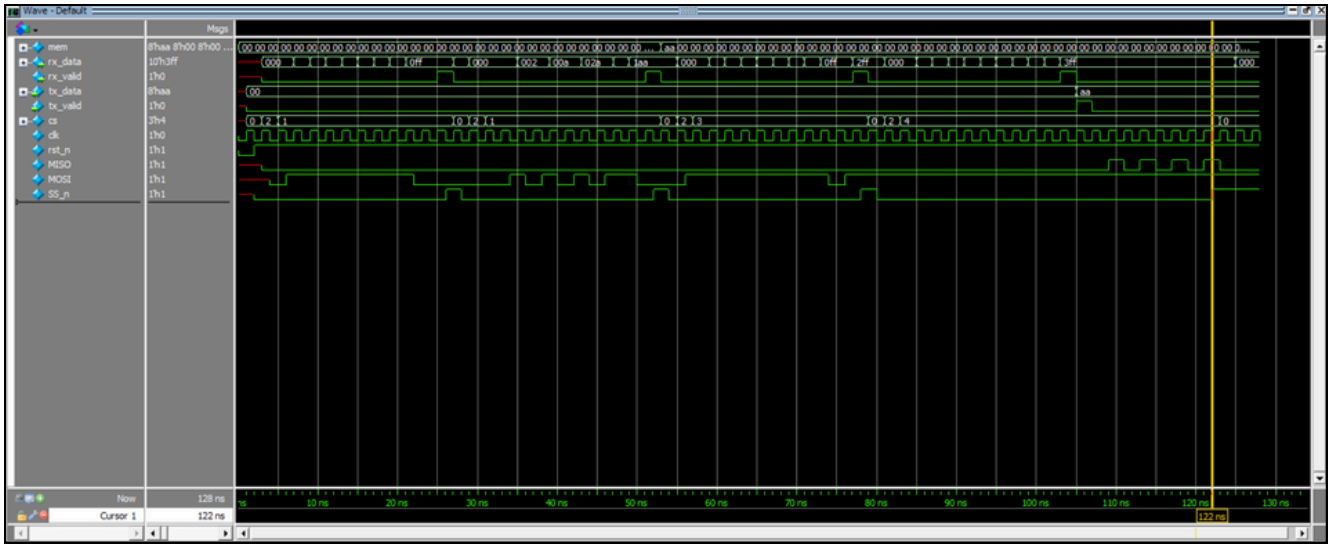
Test bench

```
module tb2 () ;

reg clk ;
reg rst_n ;
wire MISO ;
reg MOSI , SS_n ;

SPI_WRAPPER DUT (
    MOSI , clk ,  rst_n , SS_n , MISO
) ;

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

initial begin
    $readmemh ("mem.dat" , DUT.RAM_inst.mem) ;
    rst_n = 0 ;
    @(negedge clk ) ;
    rst_n = 1 ;
    SS_n = 0 ;
    @(negedge clk ) ;
    MOSI = 0 ;             //writte mode
    @(negedge clk ) ;
    repeat (8) begin       // writing address
        MOSI = 1 ;
        @(negedge clk ) ;
    end
    repeat (2) begin
        MOSI = 0 ;
        @ (negedge clk );
    end
```

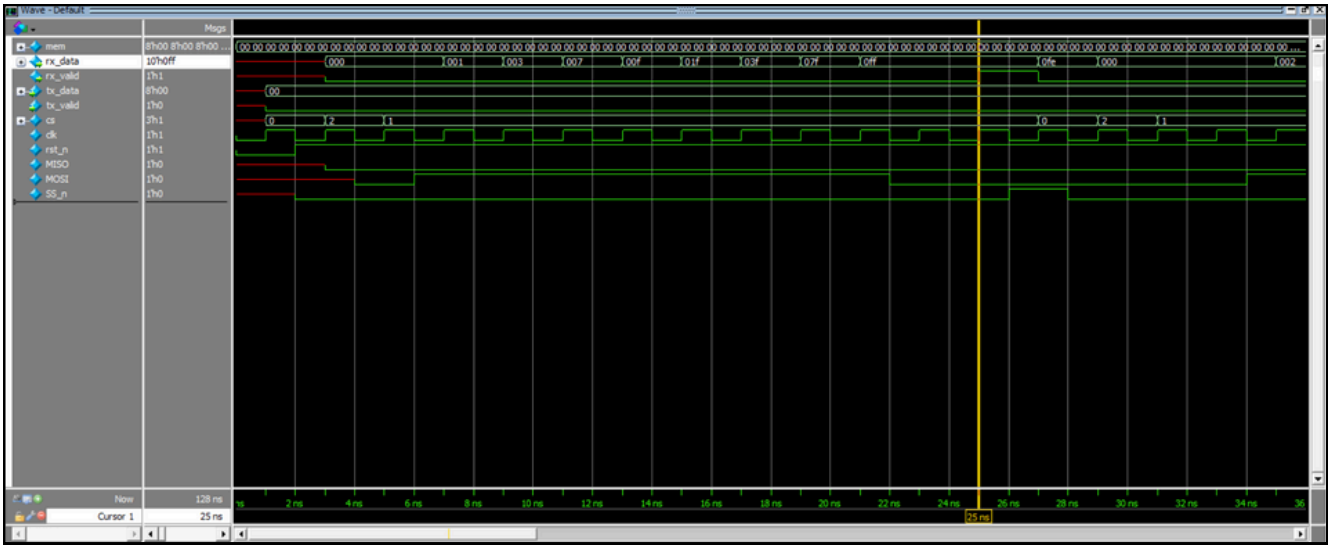
```
SS_n = 1 ;
    @ (negedge clk ) ;
    SS_n = 0 ;
    @ (negedge clk ) ;
    MOSI = 0 ;            //writte mode
    @(negedge clk ) ;
    repeat (4) begin      //write data
        MOSI = 0 ;
        @(negedge clk ) ;
        MOSI = 1 ;
        @(negedge clk ) ;
    end
    MOSI = 1 ;
    @ (negedge clk ) ;
    MOSI = 0 ;
    @ (negedge clk ) ;
    SS_n = 1  ;
    @ (negedge clk) ;
    SS_n = 0 ;
    @ (negedge clk) ;
    MOSI = 1 ;           // reading mode
    @ (negedge clk) ;
    repeat (8) begin     // reading address
        MOSI = 1 ;
        @(negedge clk ) ;
    end
    MOSI = 0 ;
    @ (negedge  clk ) ;
    MOSI = 1 ;
    @ (negedge clk) ;
    SS_n = 1  ;
    @ (negedge clk) ;
    SS_n = 0 ;
    @ (negedge clk) ;
    MOSI = 1 ;          // reading mode
    repeat (10) begin   // reading dummies
        MOSI = 1 ;
        @(negedge clk ) ;
    end
    repeat (10) @(negedge clk) ;
     SS_n = 1  ;
    repeat(3)@ (negedge clk) ;
    $stop ;
end
endmodule
```

Waveform

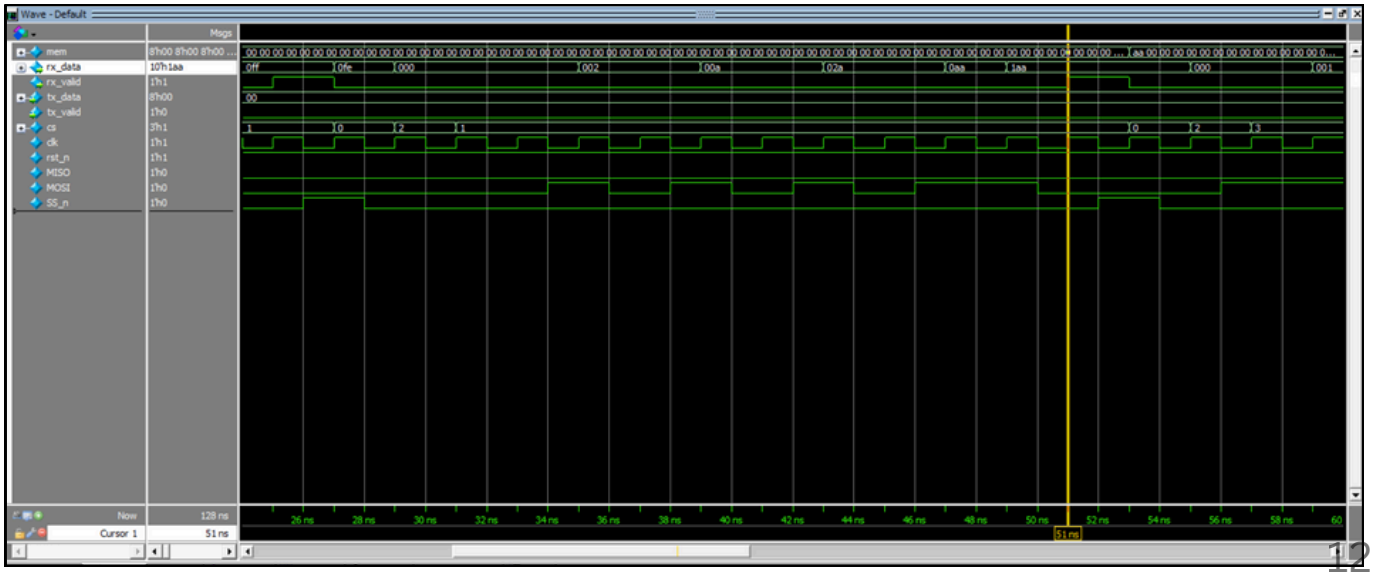
Overall waveform



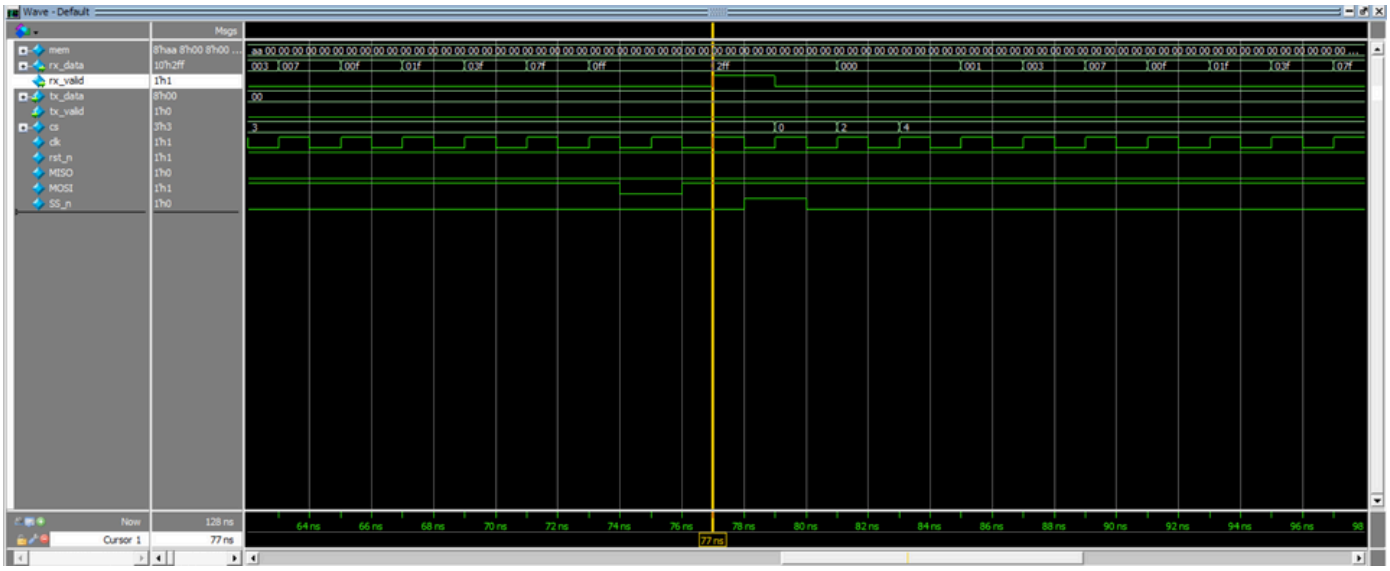
Write address : 0xff



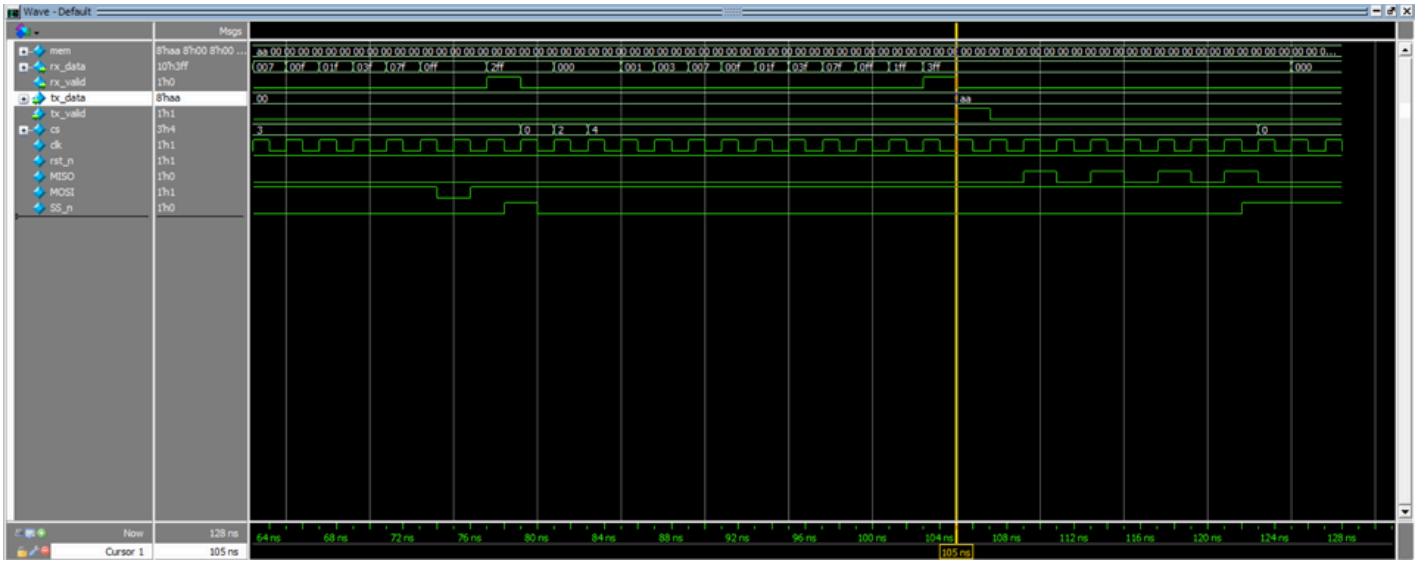
Write data : 0xaa



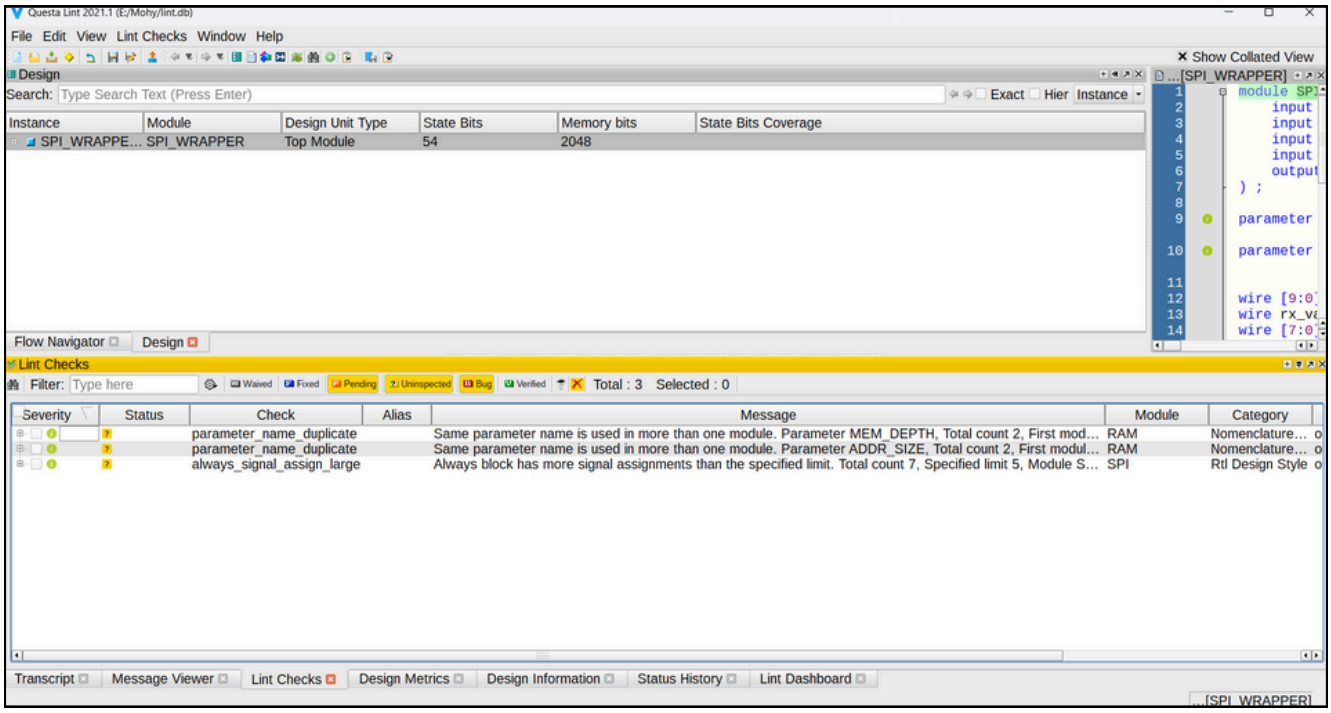
Read address : 0xff



Read data : 0xaa



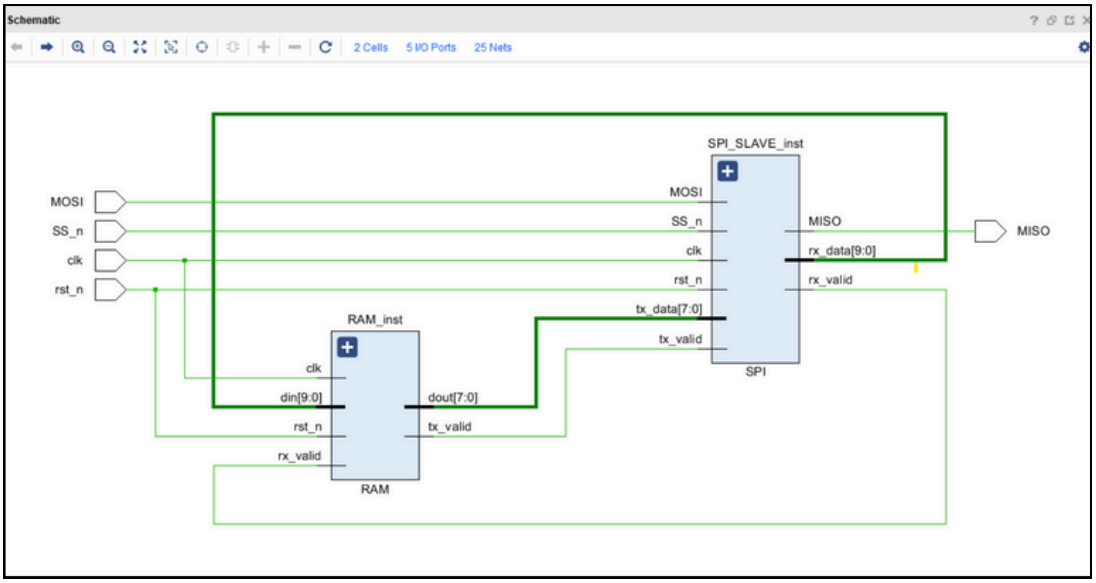
Linting

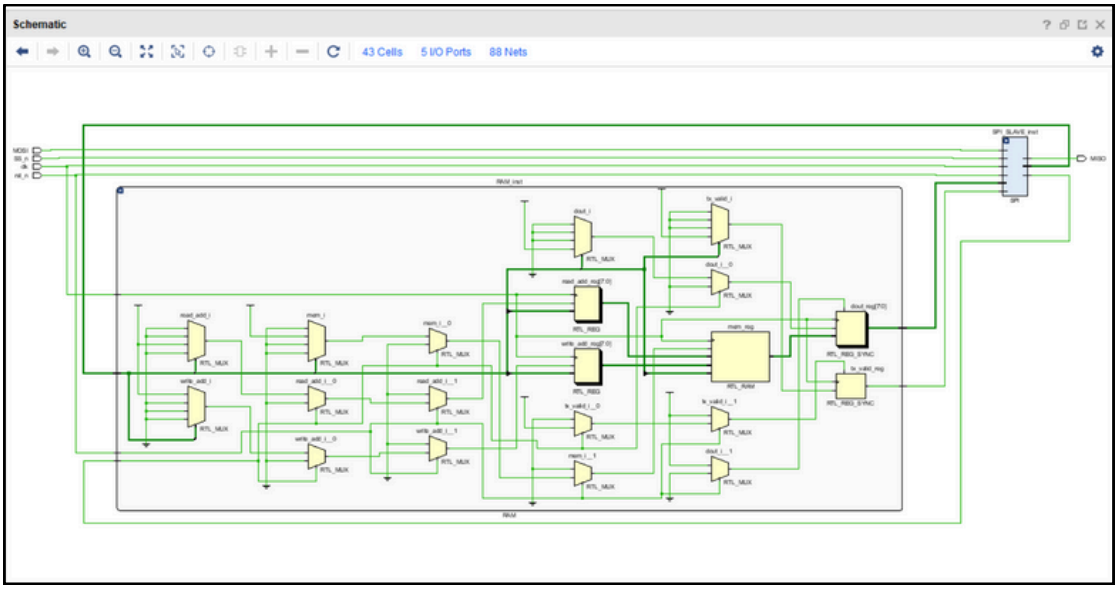


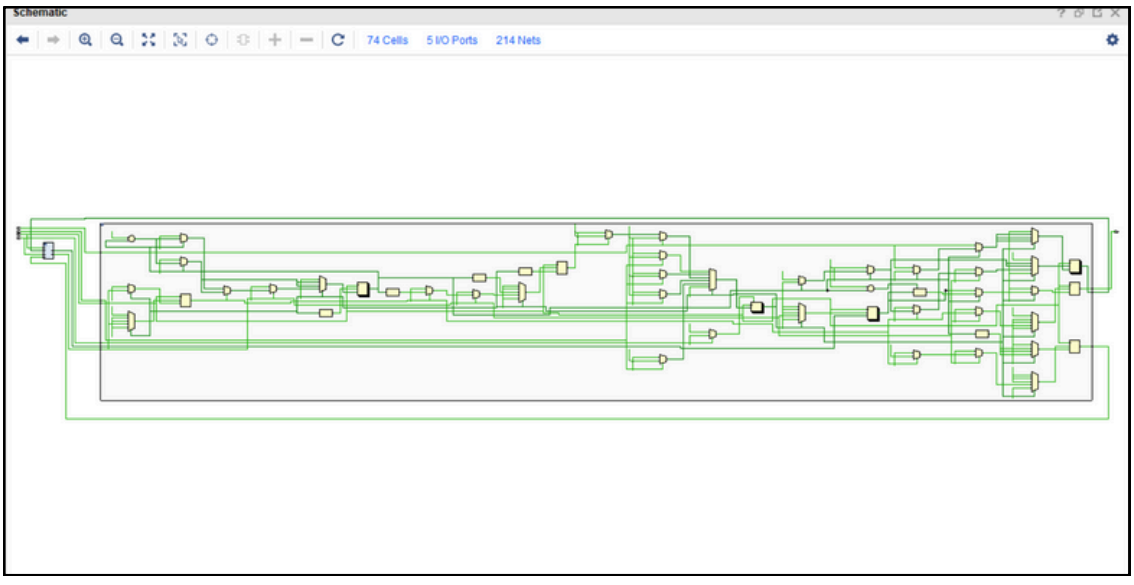
- To achieve optimal performance, the design will be evaluated under different encoding schemes. The final selection will be made based on timing analysis results . The encoding that provided the highest setup and hold slack will be chosen, ensuring the design operates reliably at the maximum achievable frequency within the target FPGA constraints.

Elaboration (one hot encoding)

Schematic

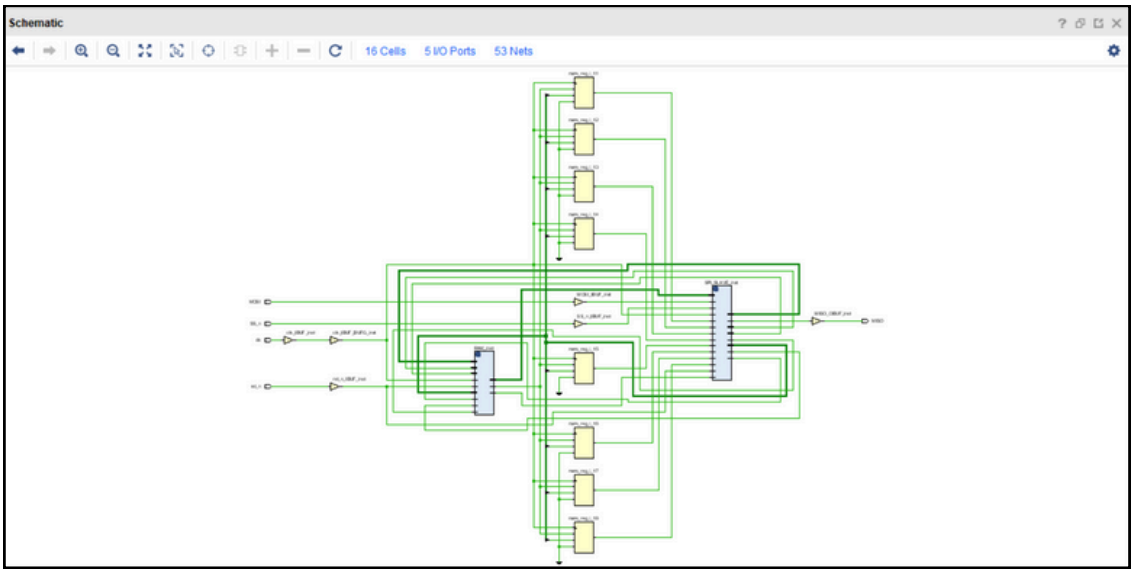


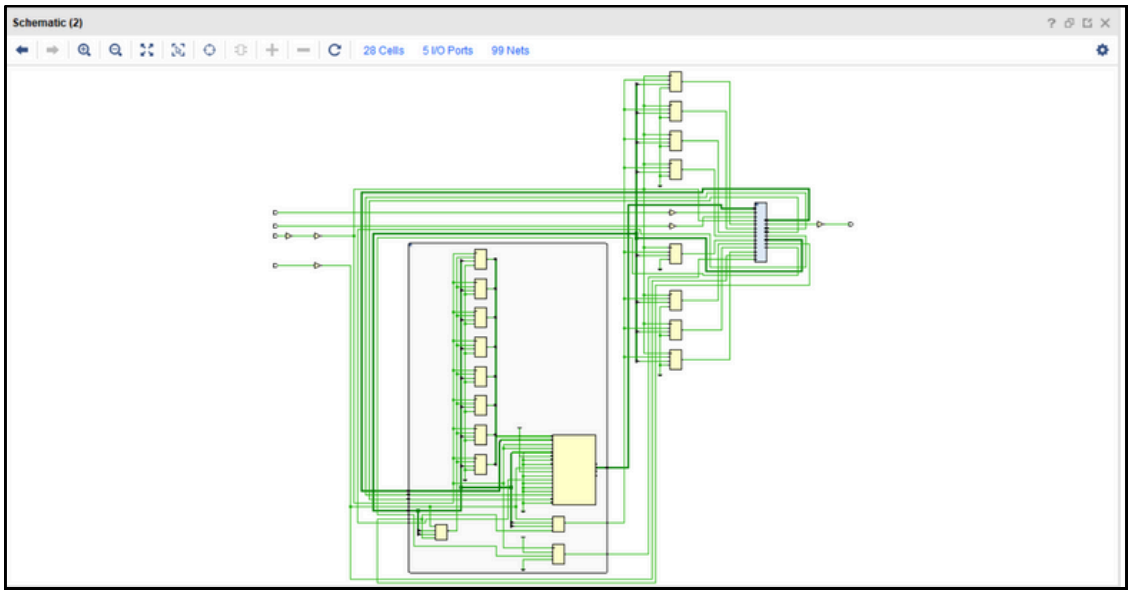


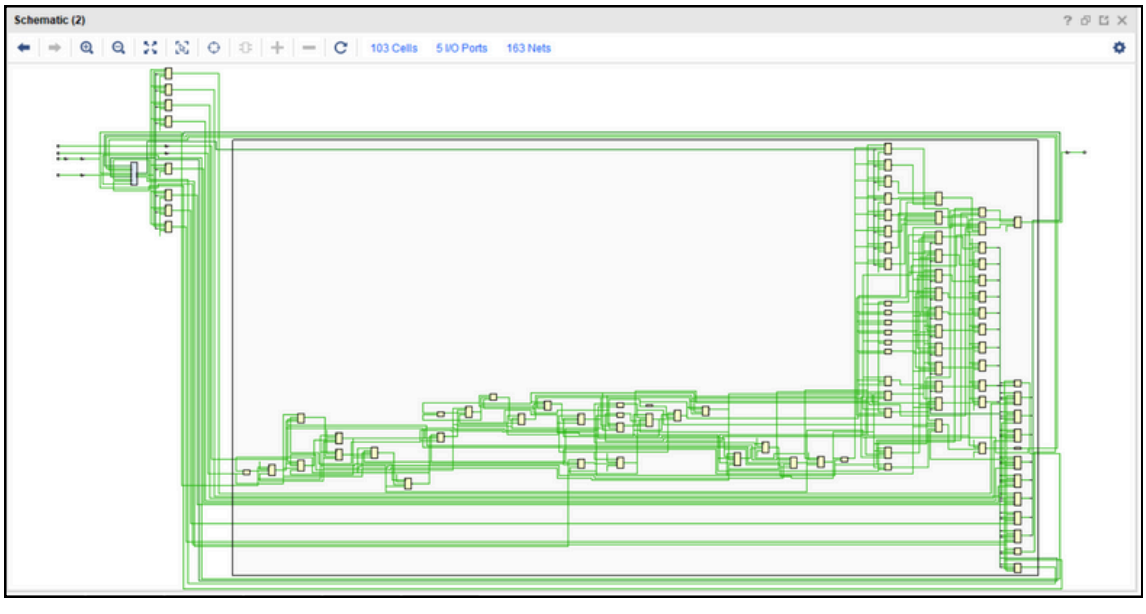


Synthesis (one hot encoding)

Schematic



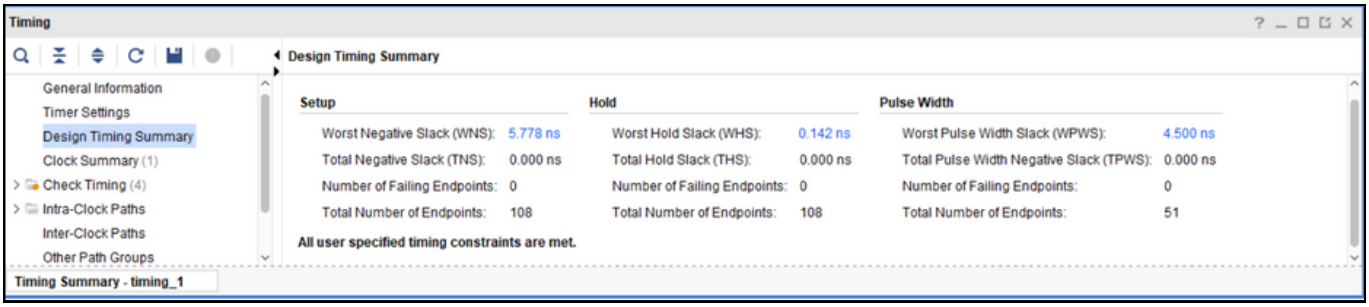




Synthesis report

```
-------------------------------------------------------------------------------------------------------
               State |                     New Encoding |                Previous Encoding
-------------------------------------------------------------------------------------------------------
                IDLE |                            00001 |                             000
             CHK_CMD |                            00010 |                             010
           READ_DATA |                            00100 |                             100
            READ_ADD |                            01000 |                             011
               WRITE |                            10000 |                             001
-------------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'one-hot' in module 'SPI'
```

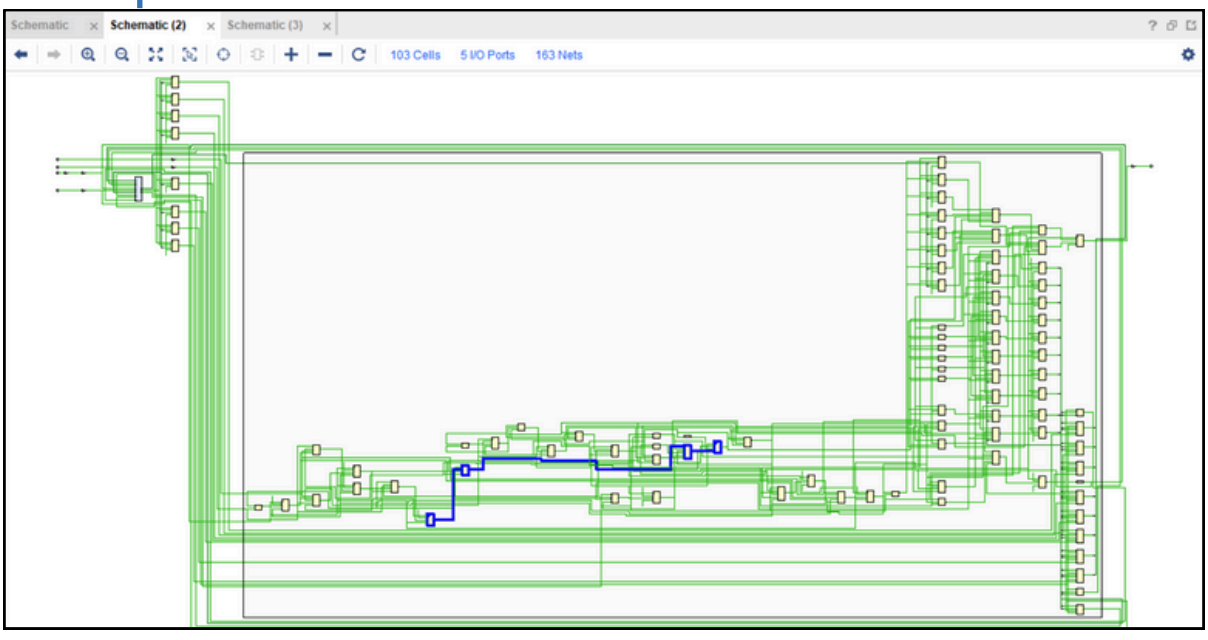
Timing report



Utilization report

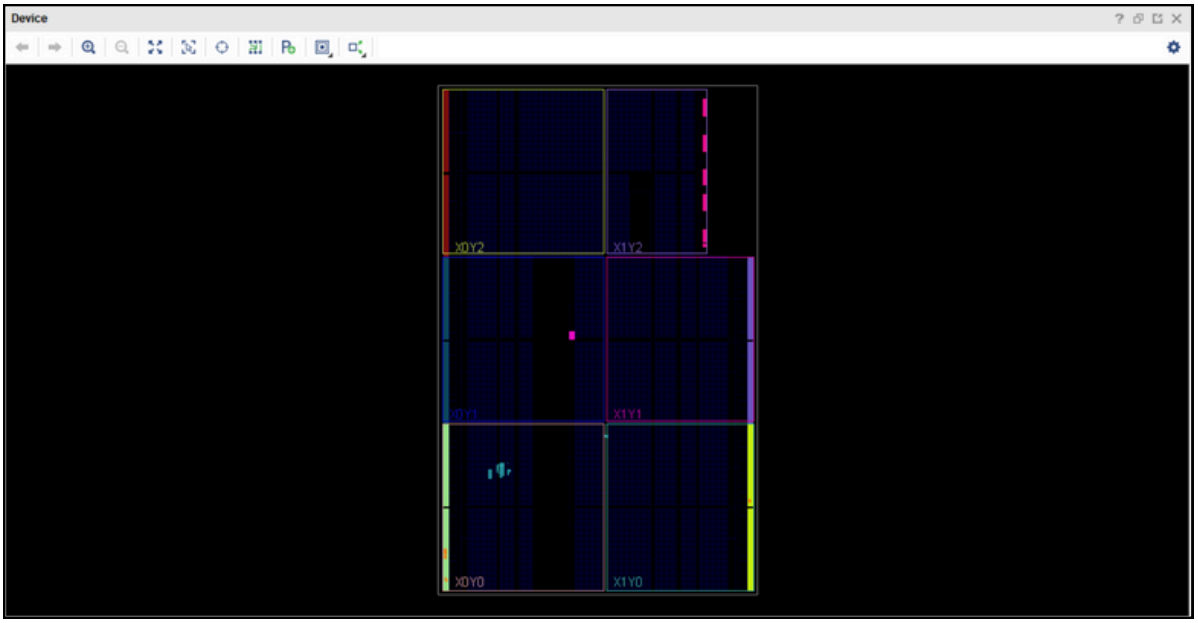
Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_WRAPPER	51	48	0.5	5	1
RAM_inst (RAM)	2	9	0.5	0	0
SPI_SLAVE_inst (SPI)	49	31	0	0	0

Critical path



Implementation (one hot encoding)

FPGA device snippet



Utilization report

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_WRAPPER	51	48	16	51	22	0.5	5	1
RAM_inst (RAM)	3	9	6	3	0	0.5	0	0
SPI_SLAVE_inst (SPI)	48	31	15	48	21	0	0	0

Timing report

Design Timing Summary

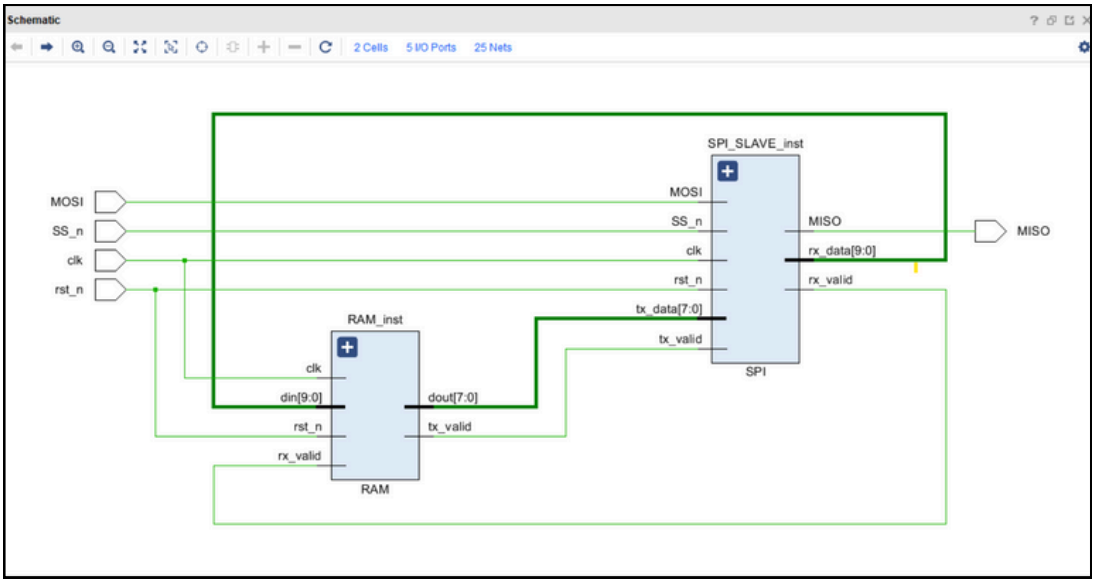
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.645 ns	Worst Hold Slack (WHS):	0.075 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	109	Total Number of Endpoints:	109	Total Number of Endpoints:	51

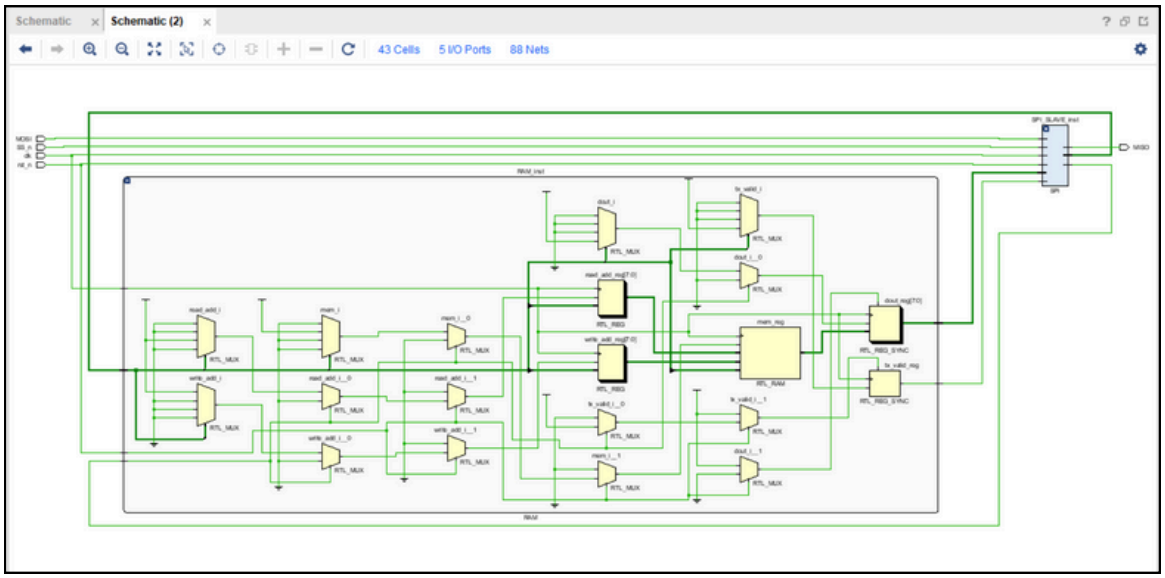
All user specified timing constraints are met.

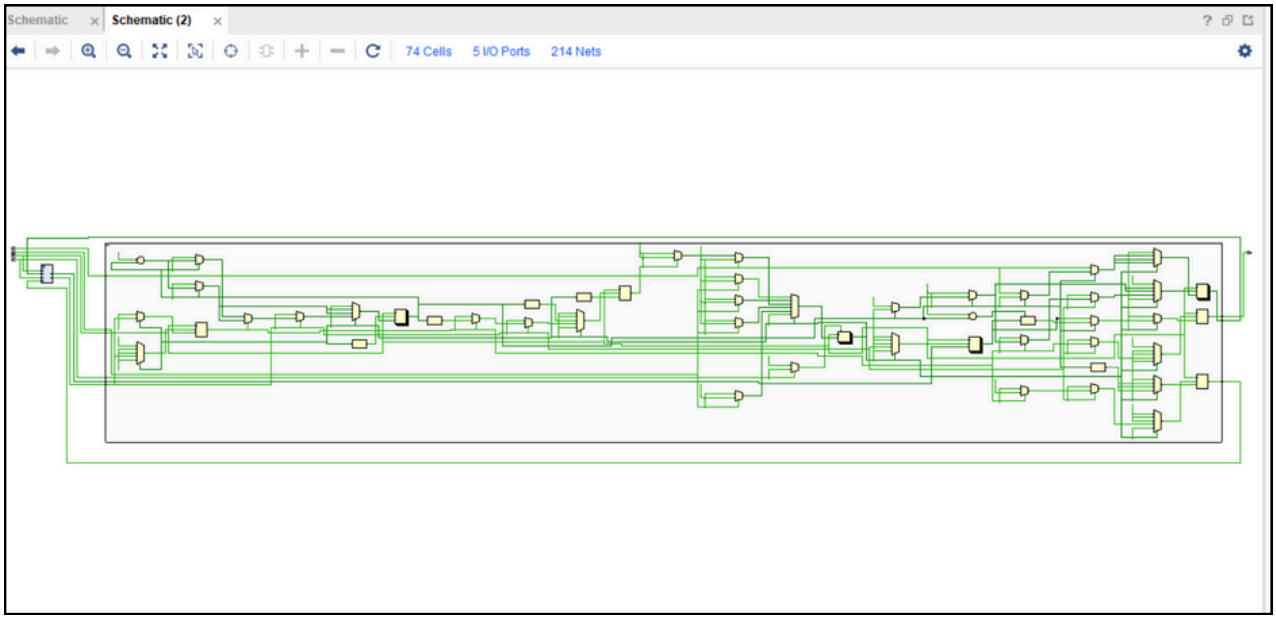
Messages tab

Elaboration (gray encoding)

Schematic

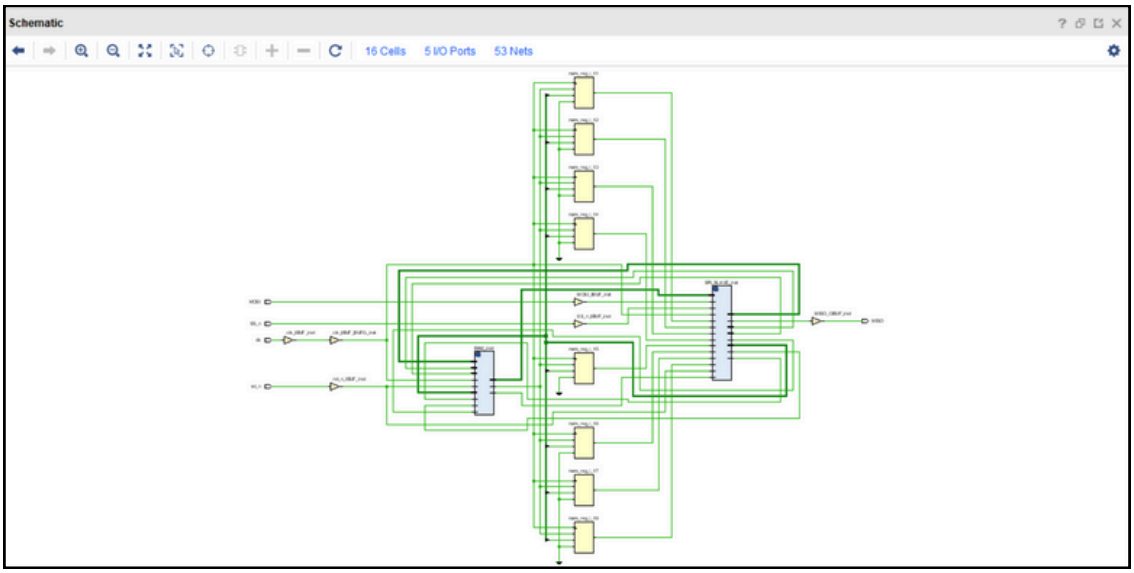


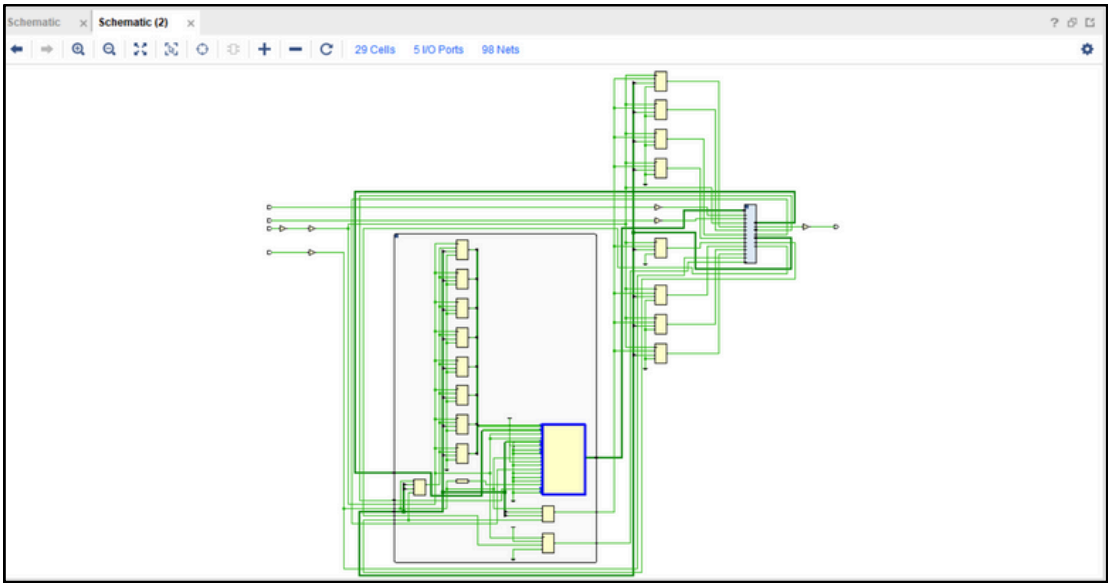


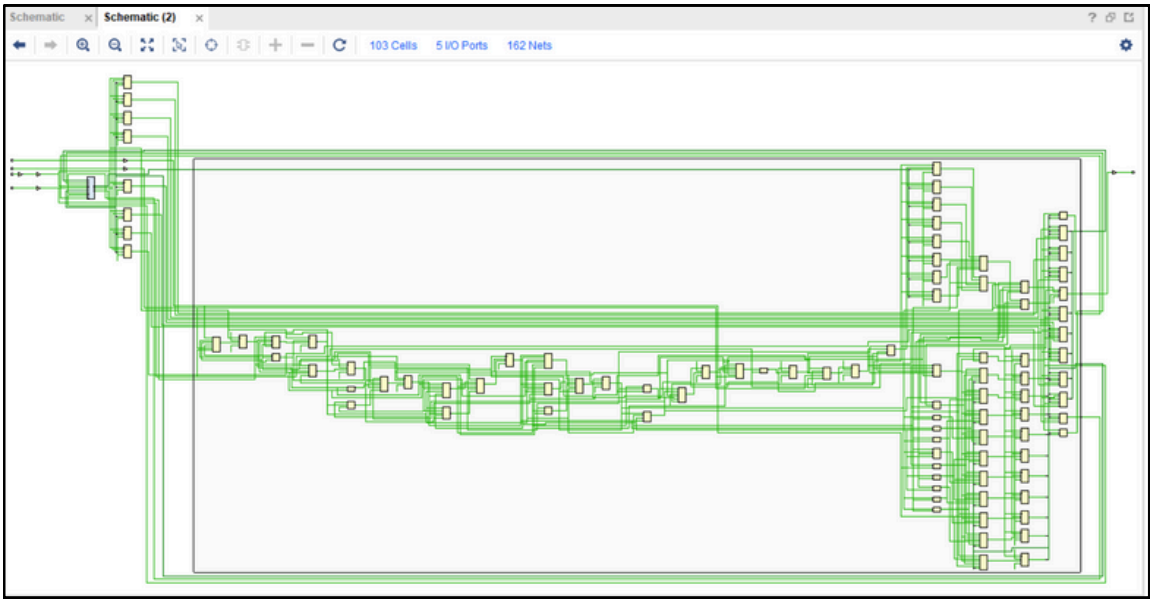


Synthesis (gray encoding)

Schematic







Synthesis report

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

INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'gray' in module 'SPI'

Timing report

Design Timing Summary

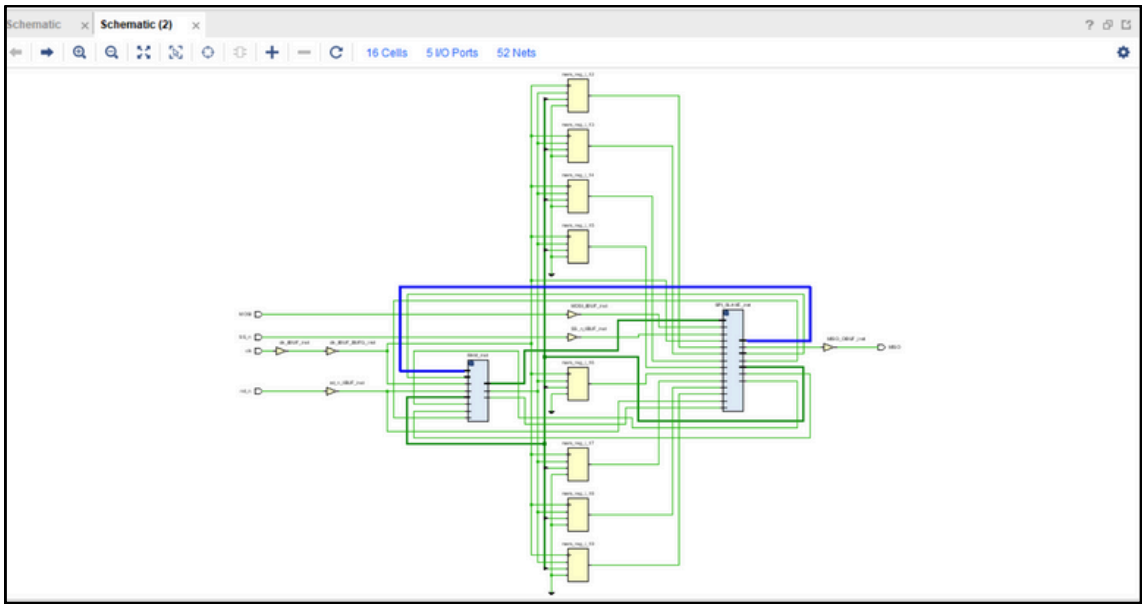
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.710 ns	Worst Hold Slack (WHS):	0.149 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	102	Total Number of Endpoints:	102	Total Number of Endpoints:	49

All user specified timing constraints are met.

Utilization report

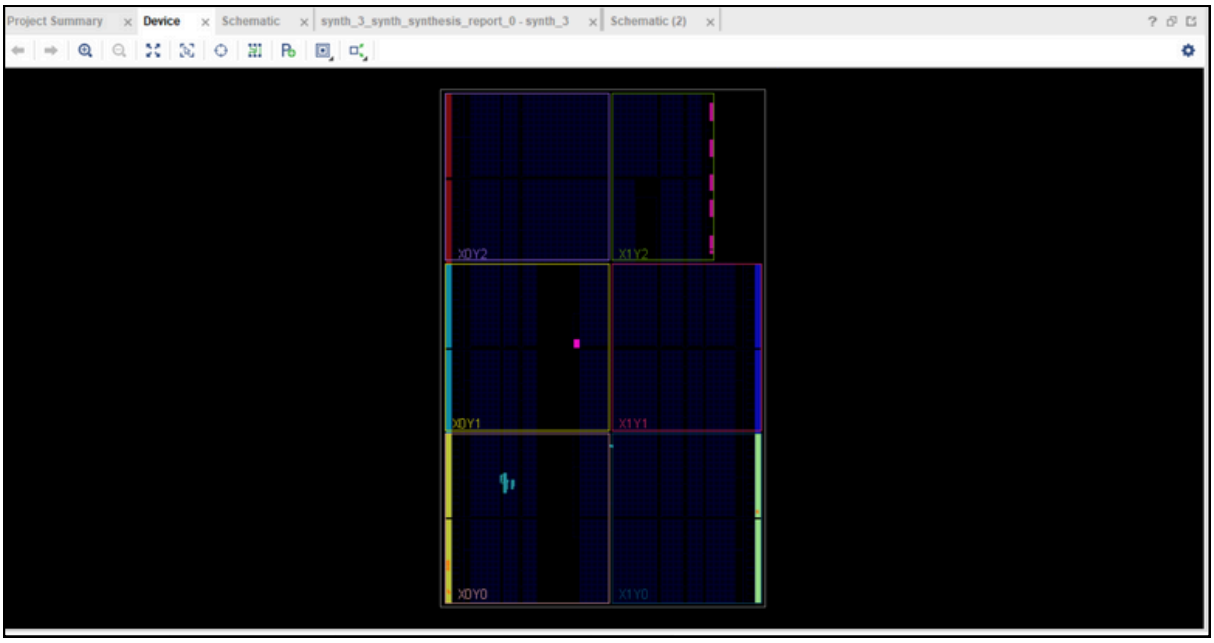
Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_WRAPPER	51	48	0.5	5	1
RAM_inst (RAM)	2	9	0.5	0	0
SPI_SLAVE_inst (SPI)	49	31	0	0	0

Critical path



Implementation (gray encoding)

FPGA device snippet



Utilization report

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_WRAPPER	55	46	17	55	21	0.5	5	1
RAM_inst (RAM)	3	9	3	3	0	0.5	0	0
SPI_SLAVE_inst (SPI)	52	29	16	52	21	0	0	0

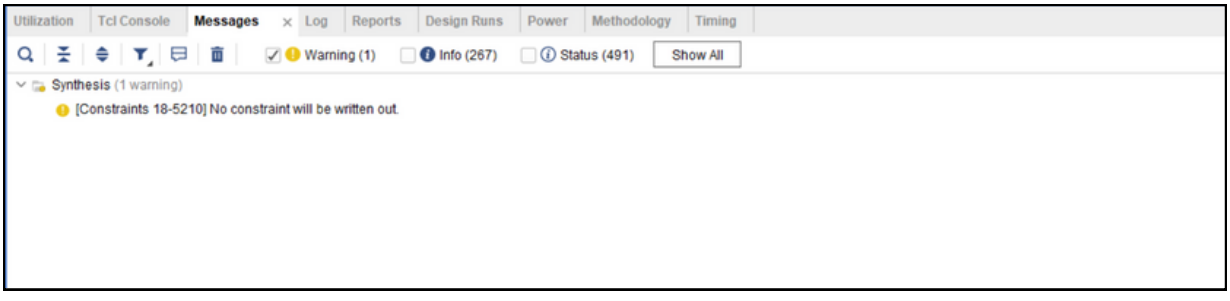
Timing report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.580 ns	Worst Hold Slack (WHS):	0.105 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:	49

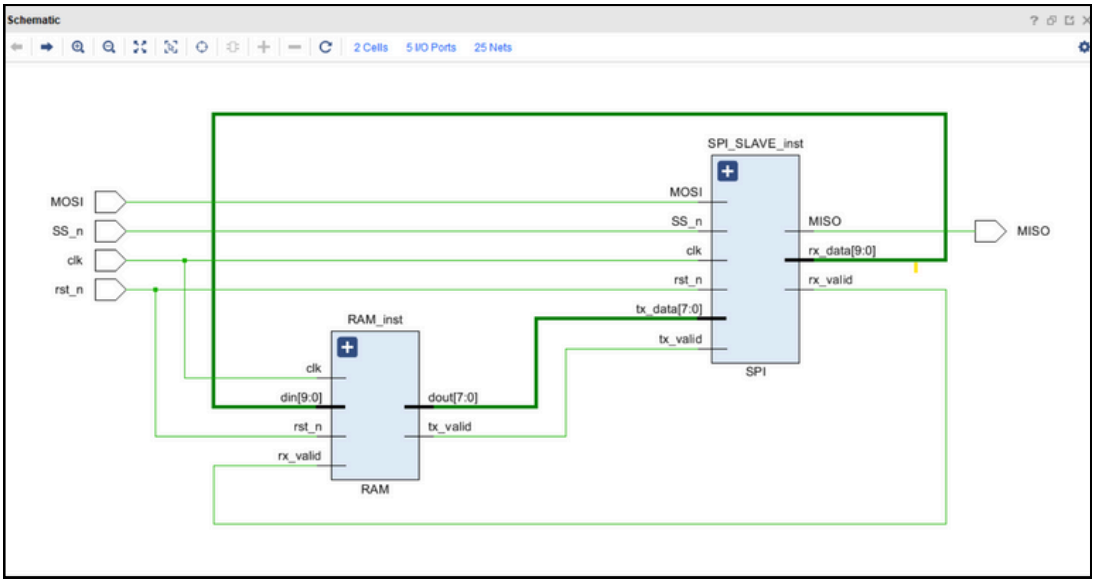
All user specified timing constraints are met.

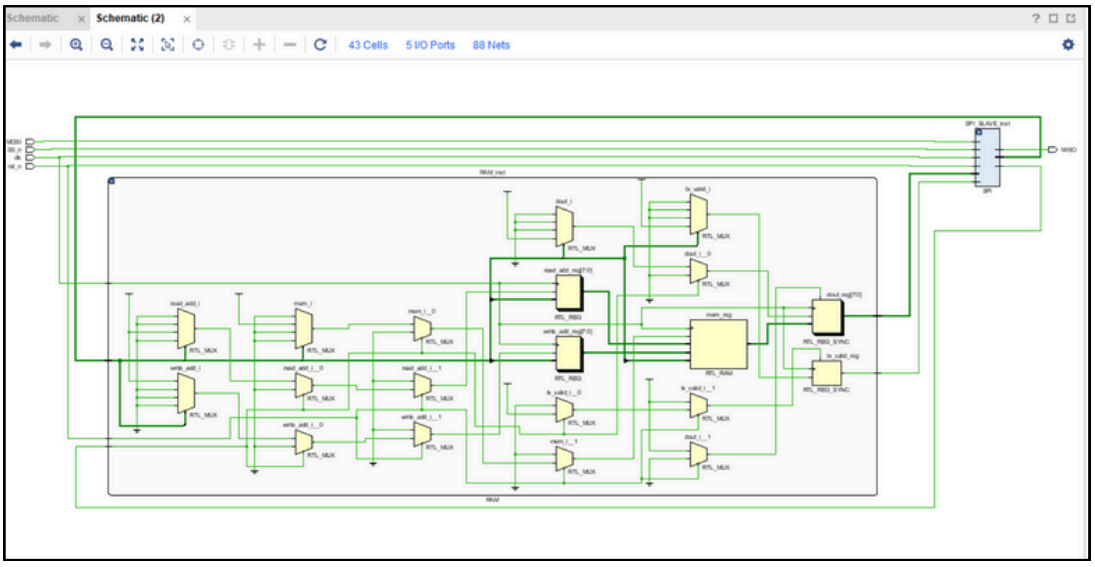
Messages tab

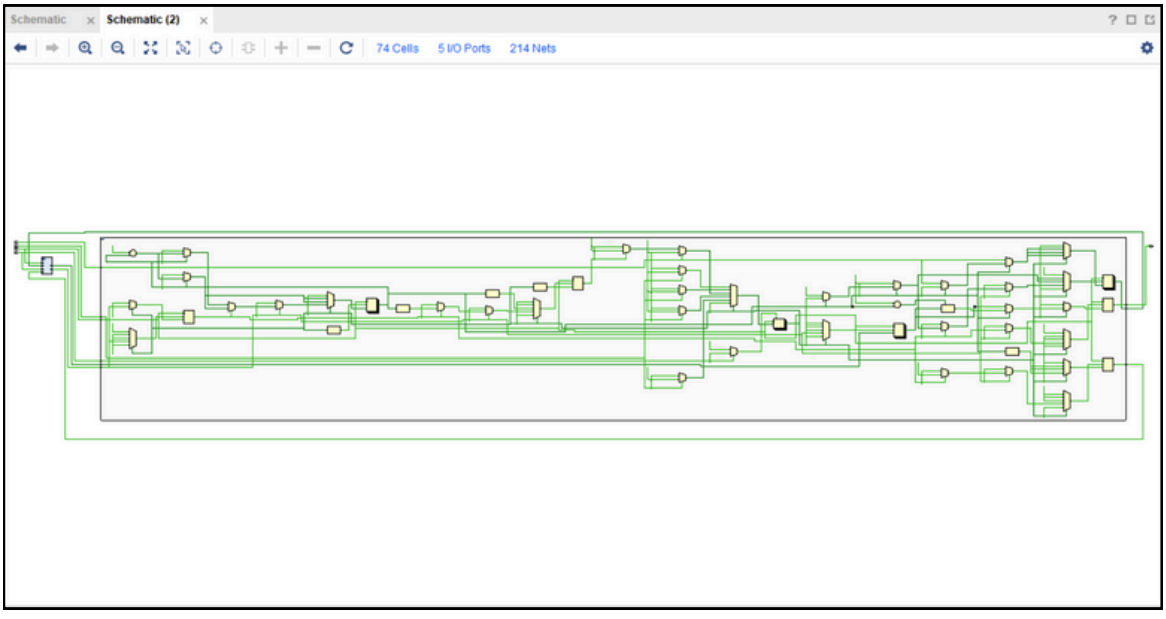


Elaboration (sequential encoding)

Schematic

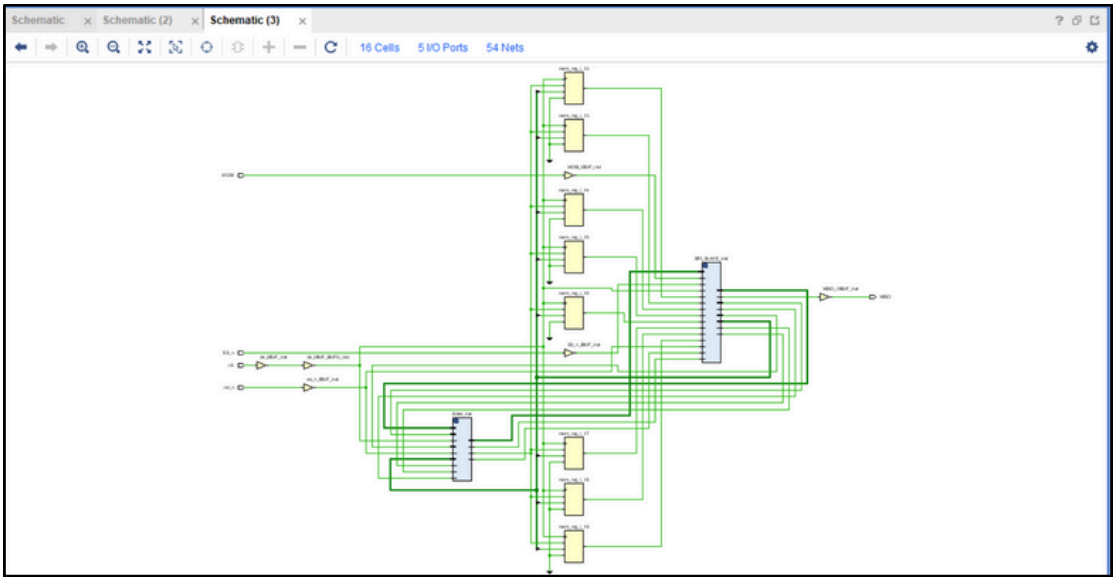


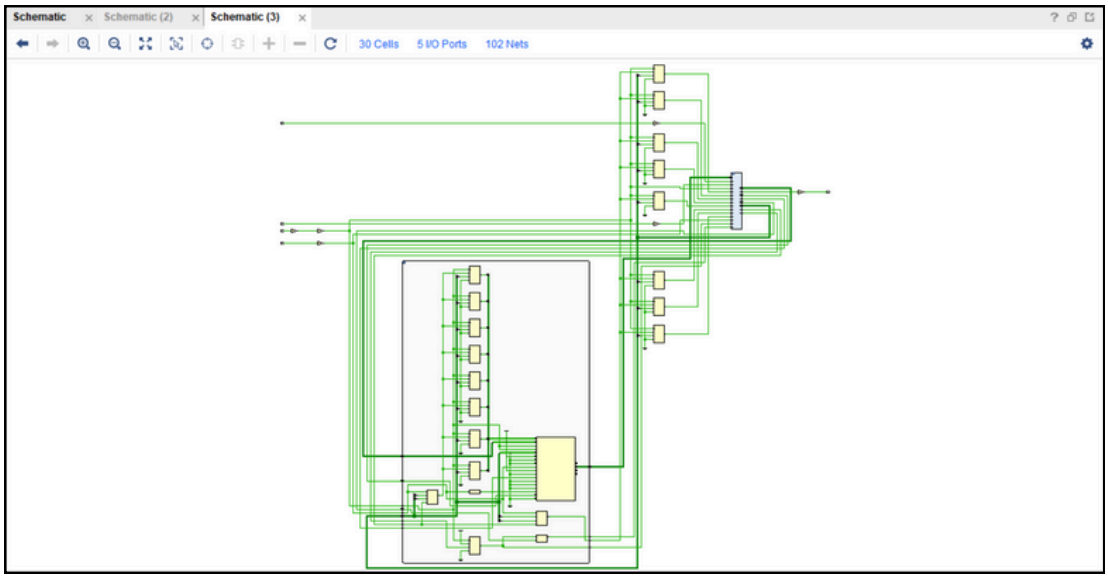


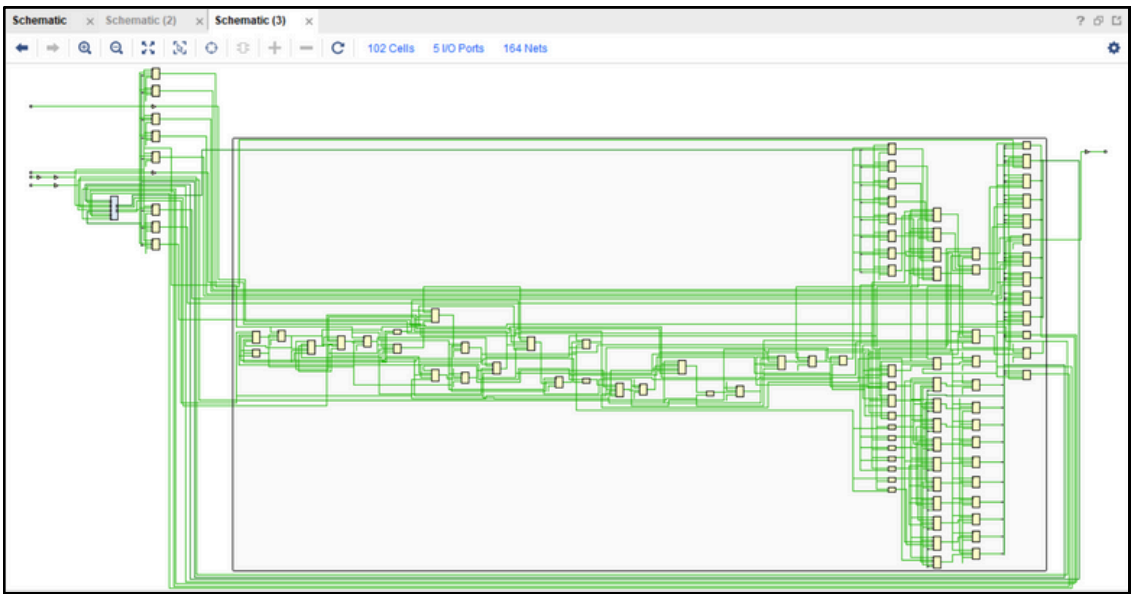


Synthesis (sequential encoding)

Schematic







Synthesis report

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

INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'sequential' in module 'SPI'

Timing report

Design Timing Summary

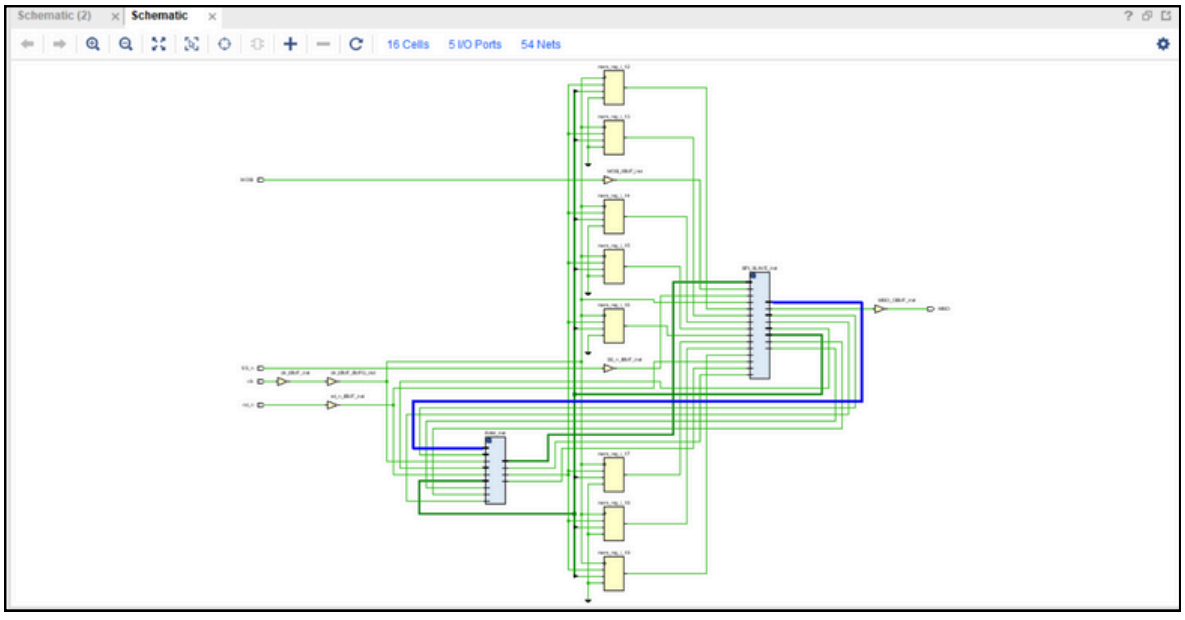
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.710 ns	Worst Hold Slack (WHS):	0.150 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	102	Total Number of Endpoints:	102	Total Number of Endpoints:	49

All user specified timing constraints are met.

Utilization report

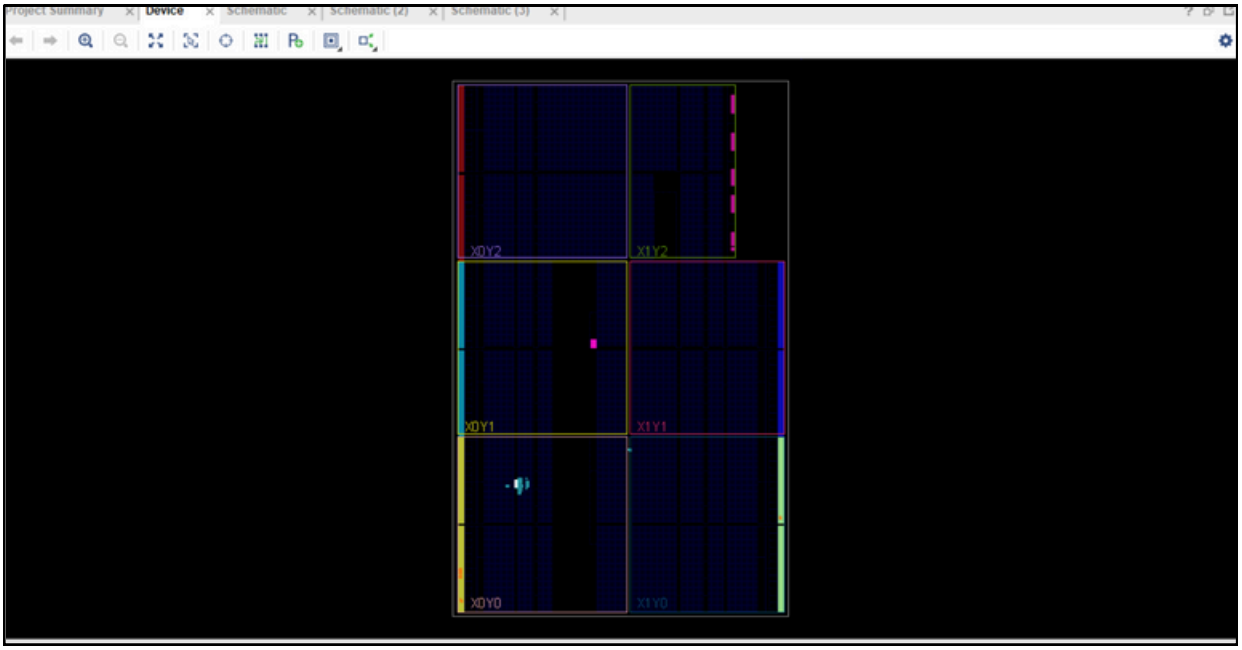
Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
SPI_WRAPPER	56	46	0.5	5	1
RAM_inst (RAM)	4	9	0.5	0	0
SPI_SLAVE_inst (SPI)	52	29	0	0	0

Critical path



Implementation (sequential encoding)

FPGA device snippet



Utilization report

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_WRAPPER	56	46	18	56	21	0.5	5	1
RAM_inst (RAM)	4	9	5	4	0	0.5	0	0
SPI_SLAVE_inst (SPI)	52	29	18	52	21	0	0	0

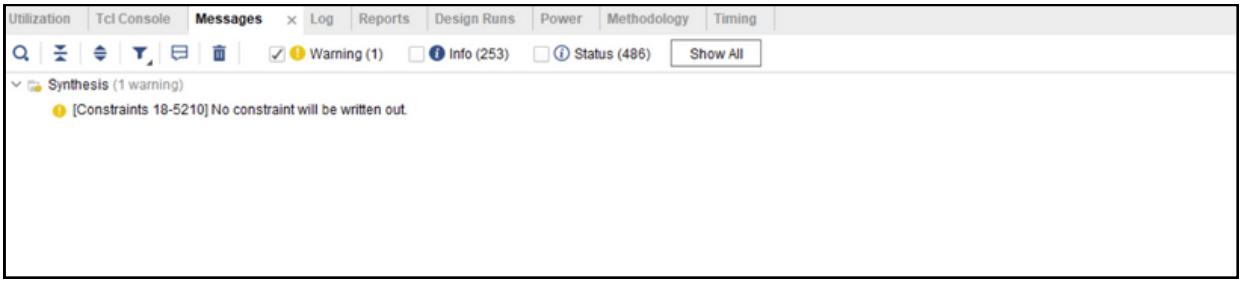
Timing report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.549 ns	Worst Hold Slack (WHS):	0.057 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:	49

All user specified timing constraints are met.

Messages tab



Comparison between different encoding :

Encoding Technique	Setup time slack	Hold time slack
one hot	6.645 ns	0.075 ns
gray	6.580 ns	0.105 ns
sequential	6.549 ns	0.057 ns

- One hot encoding gives the best setup and hold time slack allowing to operate at highest frequency possible