

# **SPI SLAVE WITH SINGLE PORT RAM**

Prepared By  
**Abdalrhman Mohy**

# Table of Contents

<b>INTRODUCTION.....</b>	3
Project Overview.....	3
<b>SPI Architecture.....</b>	5
<b>SPI State diagram .....</b>	5
<b>RTL Code.....</b>	6
Single port RAM .....	6
SPI.....	7
SPI wrapper.....	9
Test bench.....	10
<b>Wave form.....</b>	2
<b>Linting .....</b>	14
<b>One hot encoding.....</b>	15
Elaboration .....	15
Synthesize .....	16
Implementation.....	18
<b>Gray.....</b>	19
Elaboration .....	19
Synthesize .....	20
Implementation.....	22
<b>Sequential.....</b>	23
Elaboration .....	23
Synthesize .....	24
Implementation.....	26

# 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.

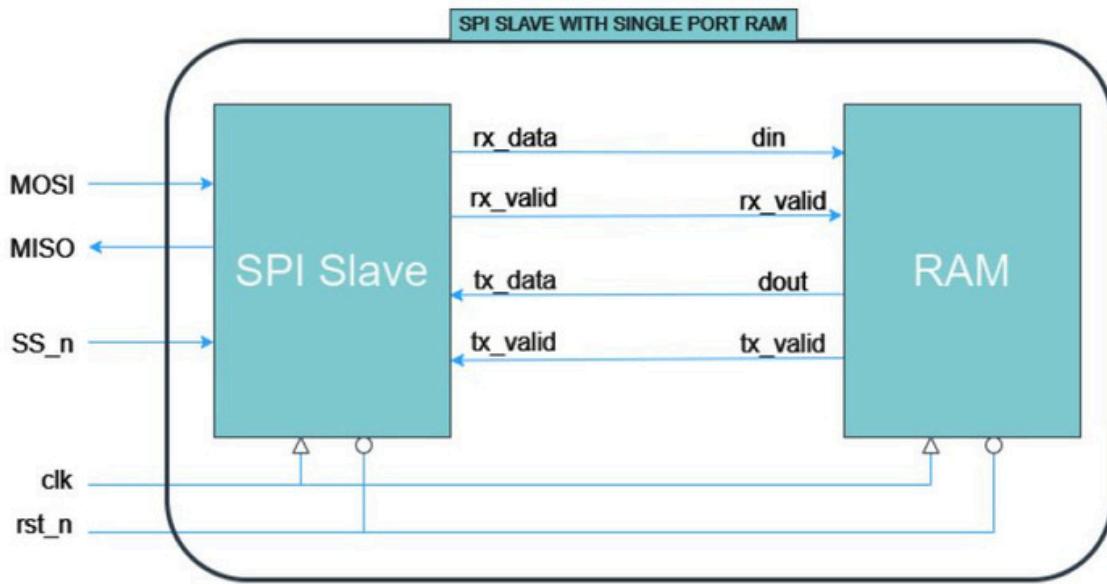
## Communication Workflow

- **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.

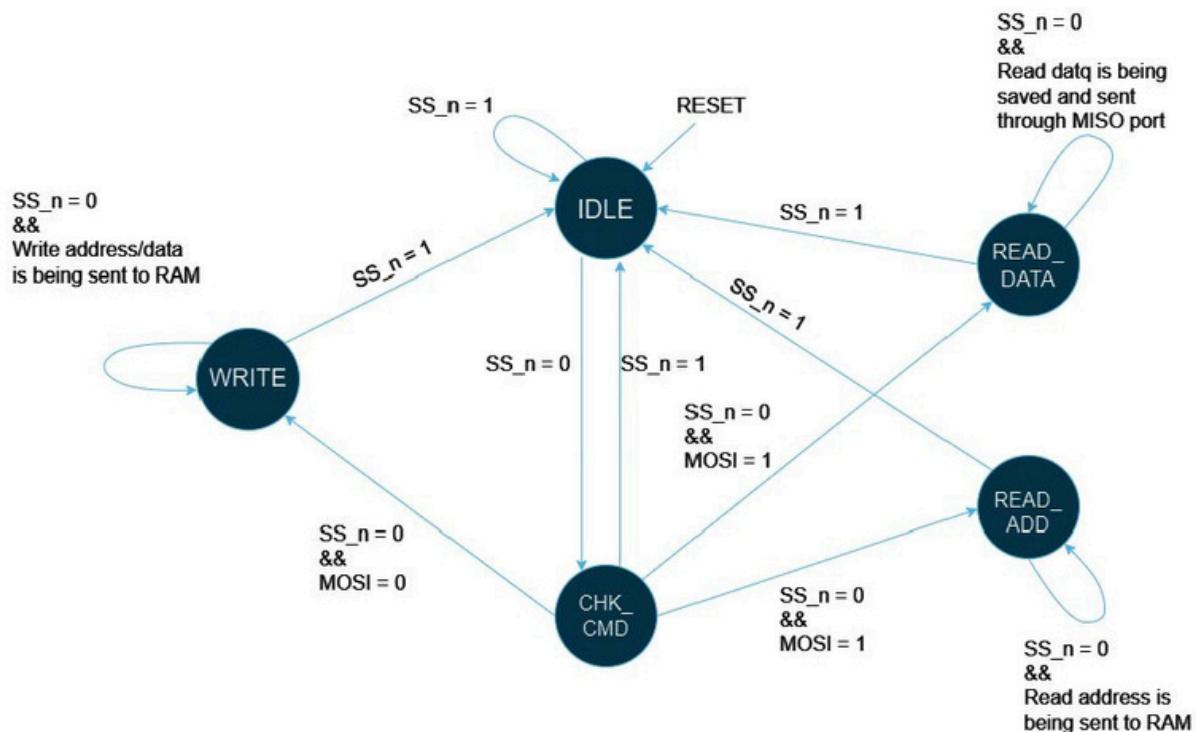
## Applications:

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

# SPI Architecture



# SPI State diagram



# RTL Code

## Single port RAM



```
1 module RAM (
2     input clk ,
3     input rst_n ,
4     input [9:0] din ,
5     input rx_valid ,
6     output reg [7:0] dout ,
7     output reg tx_valid
8 );
9
10 parameter MEM_DEPTH = 256 ;
11 parameter ADDR_SIZE = 8 ;
12
13 reg [7:0] mem [MEM_DEPTH - 1 : 0] ;
14 reg [ADDR_SIZE - 1: 0] read_add ;
15 reg [ADDR_SIZE - 1: 0] write_add ;
16
17 always @(posedge clk ) begin
18     if (!rst_n) begin
19         dout <= 0 ;
20         tx_valid <= 0 ;
21     end
22     else begin
23         tx_valid <= 0 ;
24         if (rx_valid) begin
25             case (din [9:8])
26                 2'b00 : begin
27                     write_add <= din [7:0] ;
28                 end
29                 2'b01 : begin
30                     mem [write_add] <= din [7:0] ;
31                 end
32                 2'b10 : begin
33                     read_add <= din [7:0] ;
34                 end
35                 2'b11 : begin
36                     dout <= mem [read_add] ;
37                     tx_valid <= 1 ;
38                 end
39             endcase
40         end
41     end
42 end
43 endmodule
```

# SPI

```
● ● ●  
1 module SPI (  
2     input MOSI ,  
3     input SS_n ,  
4     input clk ,  
5     input rst_n ,  
6     output reg [9:0] rx_data ,  
7     output reg rx_valid ,  
8     input tx_valid ,  
9     input [7:0] tx_data ,  
10    output reg MISO  
11 );  
12  
13 parameter IDLE = 3'b000 ;  
14 parameter WRITE = 3'b001 ;  
15 parameter CHK_CMD= 3'b010 ;  
16 parameter READ_ADD = 3'b011 ;  
17 parameter READ_DATA = 3'b100 ;  
18  
19 reg addr_check ;           // To check if available read address  
20 reg [3:0] counter_up ;  
21 reg [7:0] tx_data_reg ;  
22 reg output_available ;      // High during reading data  
(* fsm_encoding = "one_hot" *)  
24 reg [2:0] cs , ns ;        // Current and next state  
25  
26 always @ (posedge clk) begin    // State memory  
27     if (!rst_n)  
28         cs <= IDLE ;  
29     else  
30         cs <= ns ;  
31 end  
32  
33 always @ (*) begin          // Next state logic  
34     case (cs)  
35         IDLE : begin  
36             if (SS_n)  
37                 ns = IDLE ;  
38             else  
39                 ns = CHK_CMD ;  
40         end  
41         CHK_CMD : begin  
42             if (MOSI) begin  
43                 if (addr_check)  
44                     ns = READ_DATA ;  
45                 else  
46                     ns = READ_ADD ;  
47             end  
48             else  
49                 ns = WRITE ;  
50         end  
51         WRITE : begin  
52             if (SS_n)  
53                 ns = IDLE ;  
54             else  
55                 ns = WRITE ;  
56         end  
57         READ_ADD : begin  
58             if (SS_n)  
59                 ns = IDLE ;  
60             else  
61                 ns = READ_ADD ;  
62         end  
63         READ_DATA : begin  
64             if (SS_n)  
65                 ns = IDLE ;  
66             else  
67                 ns = READ_DATA ;  
68         end  
69         default : ns = IDLE ;  
70     endcase  
71 end
```

```

1  always @ (posedge clk ) begin //new output logic
2    case (cs)
3      IDLE : begin
4        rx_valid <= 0 ;
5        rx_data <= 0 ;
6        counter_up <= 0 ;
7        output_available <= 0 ;
8        MISO <= 0 ;
9      end
10     WRITE : begin
11       if (counter_up == 9 ) begin
12         counter_up <= 0 ;
13         rx_valid <= 1 ;
14       end
15       else begin
16         rx_data [counter_up] <= MOSI ;
17         counter_up <= counter_up + 1 ;
18         rx_valid <= 0 ;
19       end
20     end
21     READ_ADD : begin
22       if (counter_up == 9 ) begin
23         rx_data [counter_up] <= MOSI ;
24         counter_up <= 0 ;
25         rx_valid <= 1 ;
26       end
27       else begin
28         rx_data [counter_up] <= MOSI ;
29         counter_up <= counter_up + 1 ;
30         rx_valid <= 0 ;
31         addr_check <= 1 ;
32       end
33     end
34     READ_DATA : begin
35       if (tx_valid ) begin
36         output_available <= 1 ;
37         tx_data_reg <= tx_data ;
38         counter_up <= 0 ;
39       end
40       else if (output_available)begin
41         if (counter_up == 7 ) begin
42           MISO <= tx_data_reg [7 - counter_up ] ;
43           counter_up <= 0 ;
44         end
45         else begin
46           MISO <= tx_data_reg [7 - counter_up ] ;
47           counter_up <= counter_up + 1 ;
48         end
49       end
50       else begin
51         if (counter_up == 9 ) begin
52           rx_data [counter_up] <= MOSI ;
53           counter_up <= 0 ;
54           rx_valid <= 1 ;
55         end
56         else begin
57           rx_data [counter_up] <= MOSI ;
58           counter_up <= counter_up + 1 ;
59           rx_valid <= 0 ;
60           addr_check <= 0 ;
61         end
62       end
63     end
64   endcase
65
66 end
67 endmodule

```

# SPI Wrapper (TOP module )



```
1 module SPI_WRAPPER (
2     input MOSI ,
3     input clk ,
4     input rst_n ,
5     input SS_n ,
6     output MISO
7 );
8
9 parameter MEM_DEPTH = 256 ;
10 parameter ADDR_SIZE = 8 ;
11
12 wire [9:0] rx_data ;
13 wire rx_valid ;
14 wire [7:0] tx_data ;
15 wire tx_valid ;
16
17 SPI SPI_SLAVE_inst (
18     .clk(clk),
19     .rst_n(rst_n) ,
20     .SS_n(SS_n) ,
21     .MOSI(MOSI) ,
22     .rx_data(rx_data) ,
23     .rx_valid (rx_valid) ,
24     .tx_valid (tx_valid) ,
25     .tx_data (tx_data) ,
26     .MISO (MISO)
27 );
28
29 RAM RAM_inst #(.MEM_DEPTH(MEM_DEPTH) ,.ADDR_SIZE(ADDR_SIZE) )( 
30     .clk(clk) ,
31     .rst_n (rst_n) ,
32     .din(rx_data) ,
33     .rx_valid(rx_valid) ,
34     .dout(tx_data) ,
35     .tx_valid(tx_valid)
36 );
37
38 endmodule
```

## Test bench

```
● ● ●  
1 module tb2 () ;  
2  
3 reg clk ;  
4 reg rst_n ;  
5 wire MISO ;  
6 reg MOSI , SS_n ;  
7  
8 SPI_WRAPPER DUT (  
9     MOSI , clk , rst_n , SS_n , MISO  
10 ) ;  
11  
12 initial begin  
13     clk = 0 ;  
14     forever begin  
15         #1 clk = ~clk ;  
16     end  
17 end  
18  
19 initial begin  
20     $readmemh ("mem.dat" , DUT.RAM_inst.mem) ;  
21     rst_n = 0 ;  
22     @(negedge clk ) ;  
23     rst_n = 1 ;  
24     SS_n = 0 ;  
25     @(negedge clk ) ;  
26     MOSI = 0 ;           //writte mode  
27     @(negedge clk ) ;  
28     repeat (8) begin    // writing address  
29         MOSI = 1 ;  
30         @(negedge clk ) ;  
31     end  
32     repeat (2) begin  
33         MOSI = 0 ;  
34         @ (negedge clk );  
35     end
```

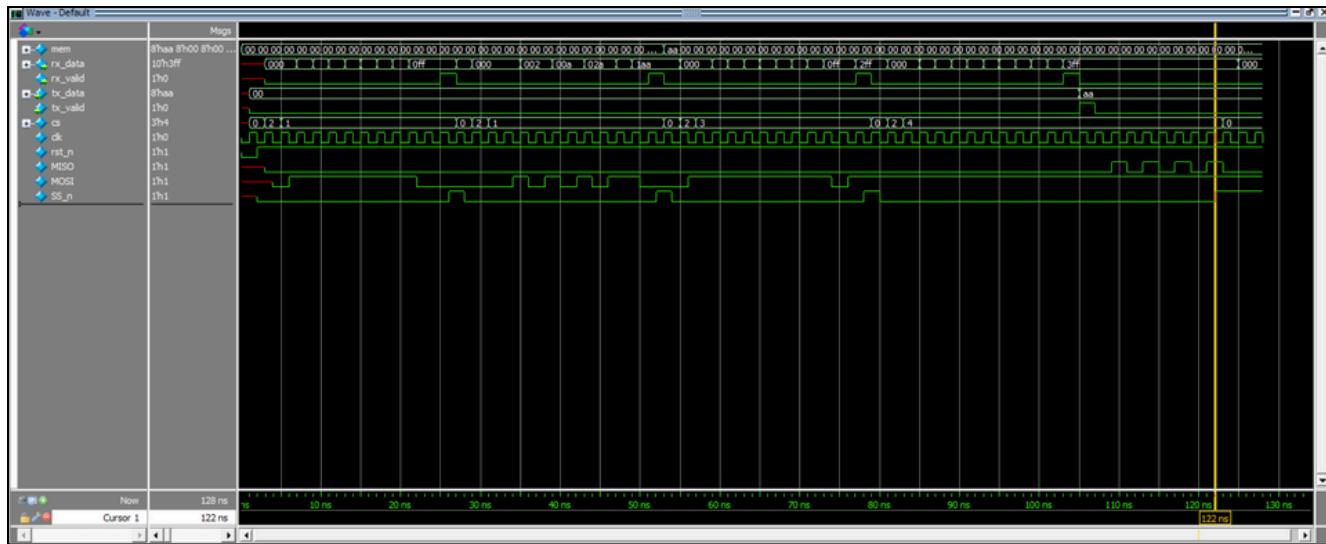
```

1  SS_n = 1 ;
2      @ (negedge clk) ;
3  SS_n = 0 ;
4      @ (negedge clk) ;
5  MOSI = 0 ;           //writte mode
6      @(negedge clk) ;
7  repeat (4) begin    //write data
8      MOSI = 0 ;
9      @(negedge clk) ;
10     MOSI = 1 ;
11      @(negedge clk) ;
12  end
13  MOSI = 1 ;
14  @ (negedge clk) ;
15  MOSI = 0 ;
16  @ (negedge clk) ;
17  SS_n = 1 ;
18  @ (negedge clk) ;
19  SS_n = 0 ;
20  @ (negedge clk) ;
21  MOSI = 1 ;           // reading mode
22  @ (negedge clk) ;
23  repeat (8) begin    // reading address
24      MOSI = 1 ;
25      @(negedge clk) ;
26  end
27  MOSI = 0 ;
28  @ (negedge clk) ;
29  MOSI = 1 ;
30  @ (negedge clk) ;
31  SS_n = 1 ;
32  @ (negedge clk) ;
33  SS_n = 0 ;
34  @ (negedge clk) ;
35  MOSI = 1 ;           // reading mode
36  repeat (10) begin   // reading dummies
37      MOSI = 1 ;
38      @(negedge clk) ;
39  end
40  repeat (10) @(negedge clk) ;
41  SS_n = 1 ;
42  repeat(3)@ (negedge clk) ;
43  $stop ;
44 end
45 endmodule

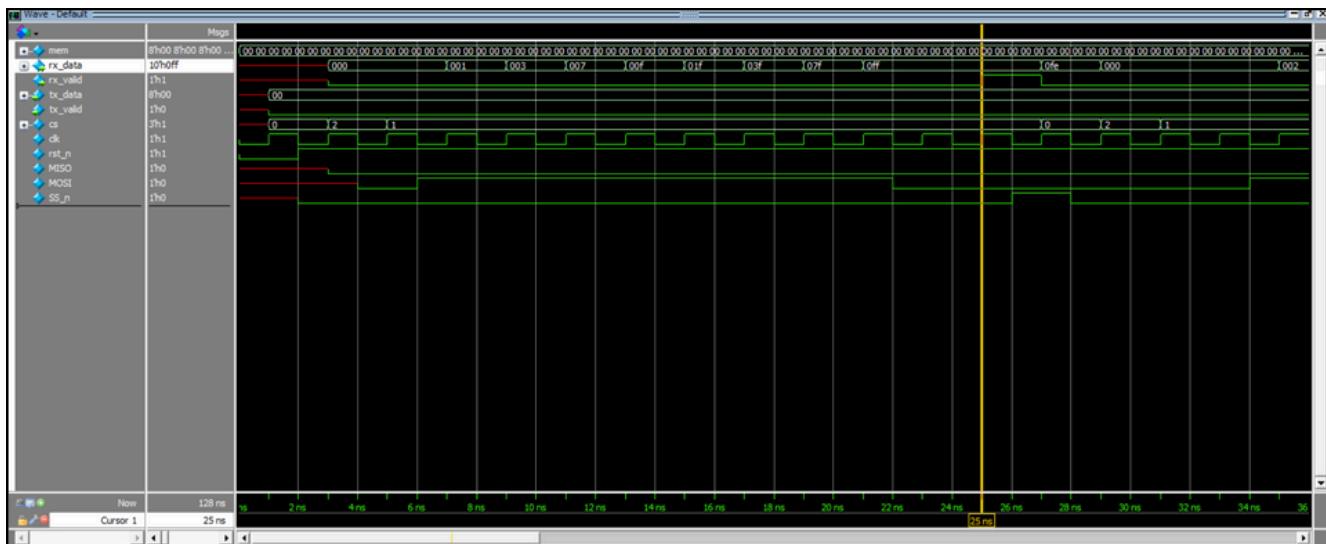
```

# Waveform

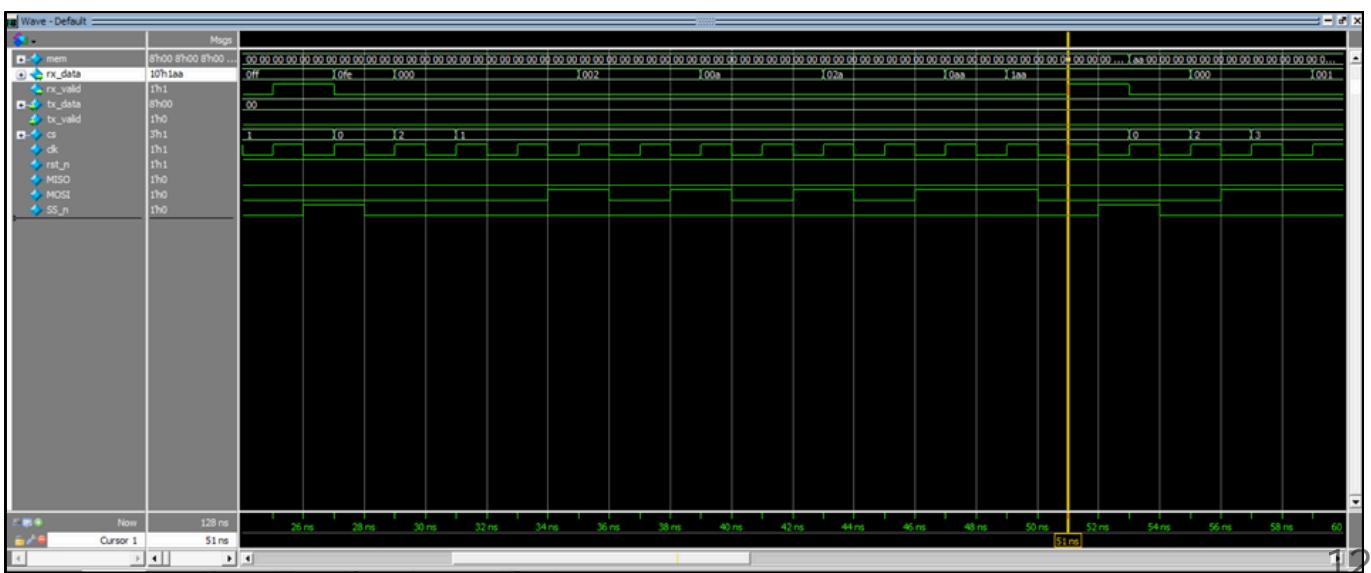
## Overall waveform



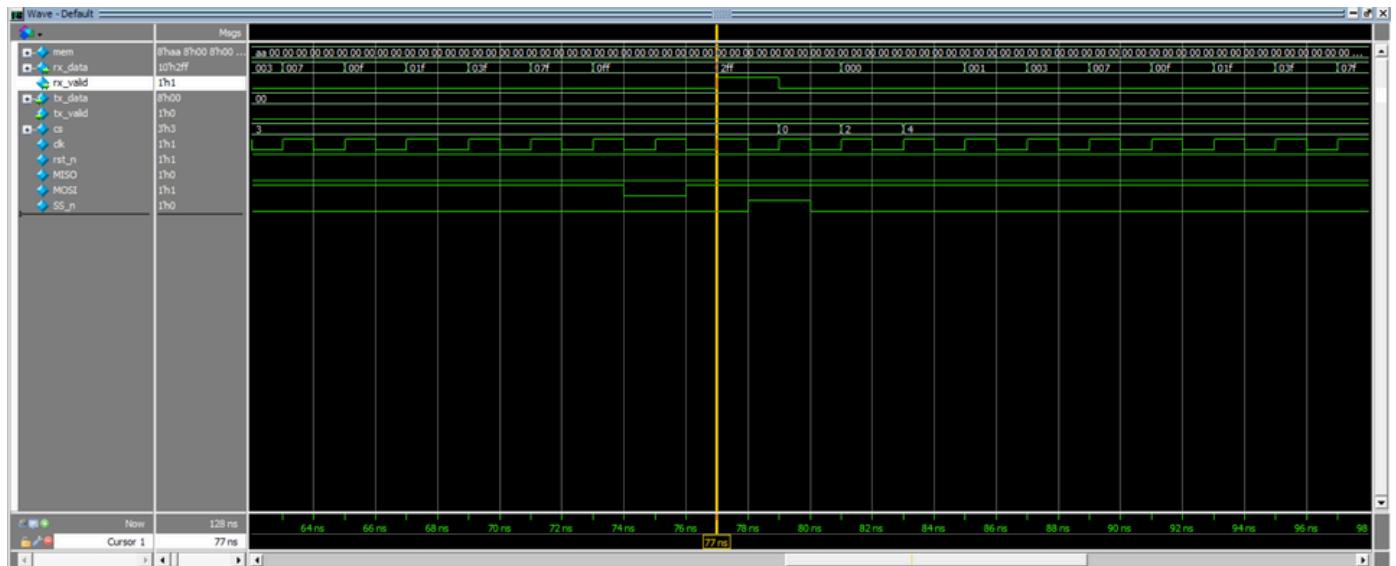
Write address : 0xff



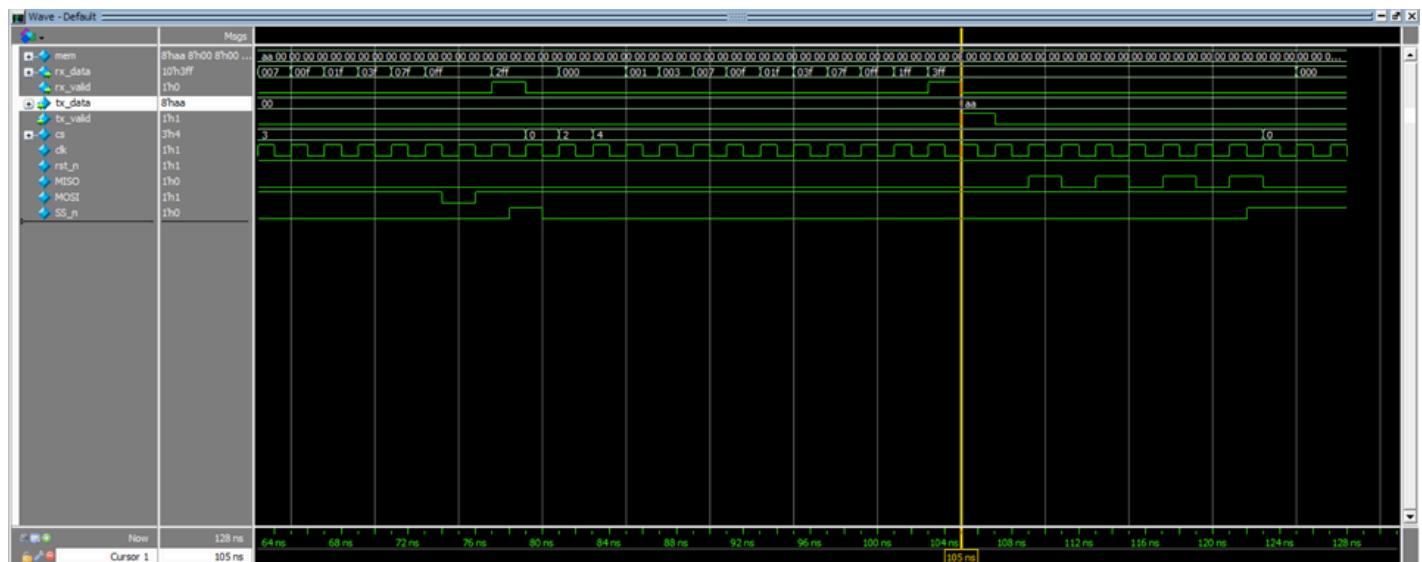
Write data : 0xaa



## Read address : 0xff



## Read data : 0xaa



# Linting

The screenshot shows the Questa Lint 2021.1 interface. At the top, there's a menu bar with File, Edit, View, Lint Checks, Window, Help. Below it is a toolbar with various icons. The main window has a search bar "Search: Type Search Text (Press Enter)". A table displays "Design Unit Type" information: Instance SPI\_WRAPPER, Module SPI\_WRAPPER, Design Unit Type Top Module, State Bits 54, Memory bits 2048, State Bits Coverage. To the right, a code editor window titled "...[SPI\_WRAPPER]" shows a portion of Verilog code:

```
1 module SPI;
2   input;
3   input;
4   input;
5   input;
6   output;
7 );
8 parameter;
9 parameter;
10 wire [9:0];
11 wire RX_V;
12 wire [7:0];
13 endmodule
```

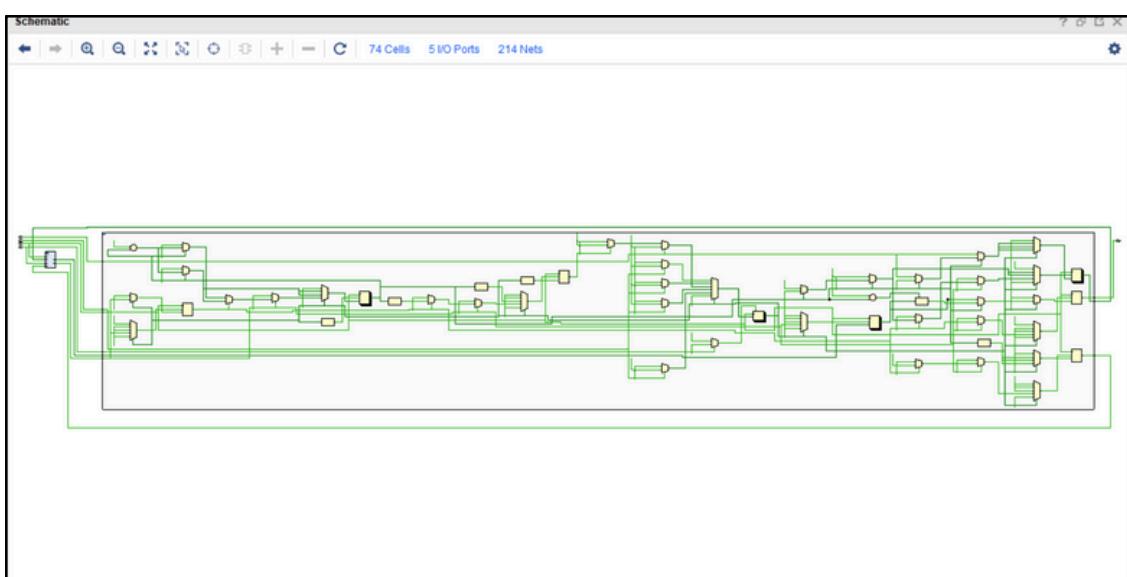
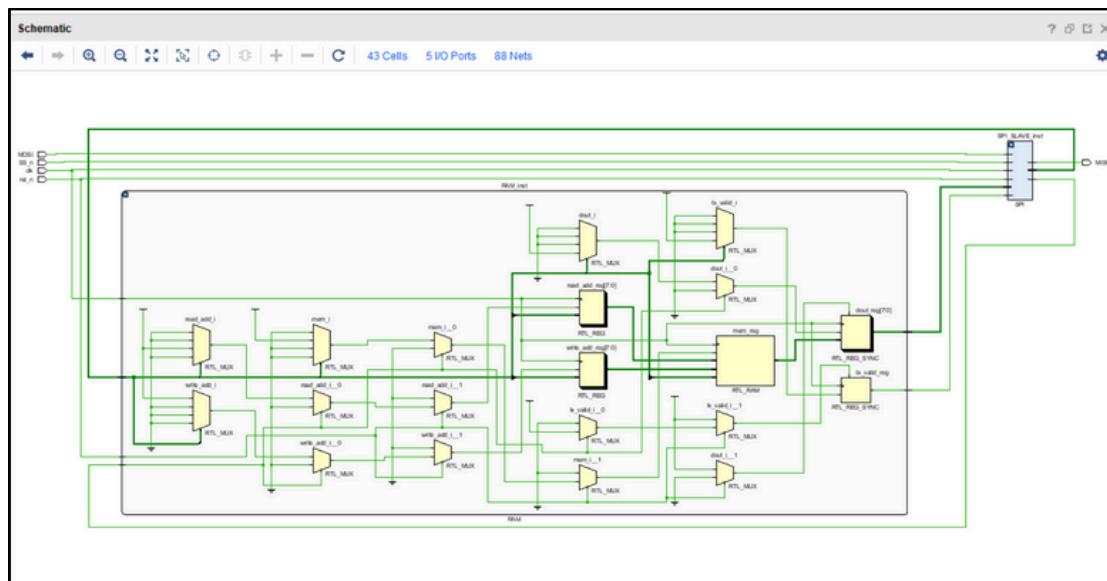
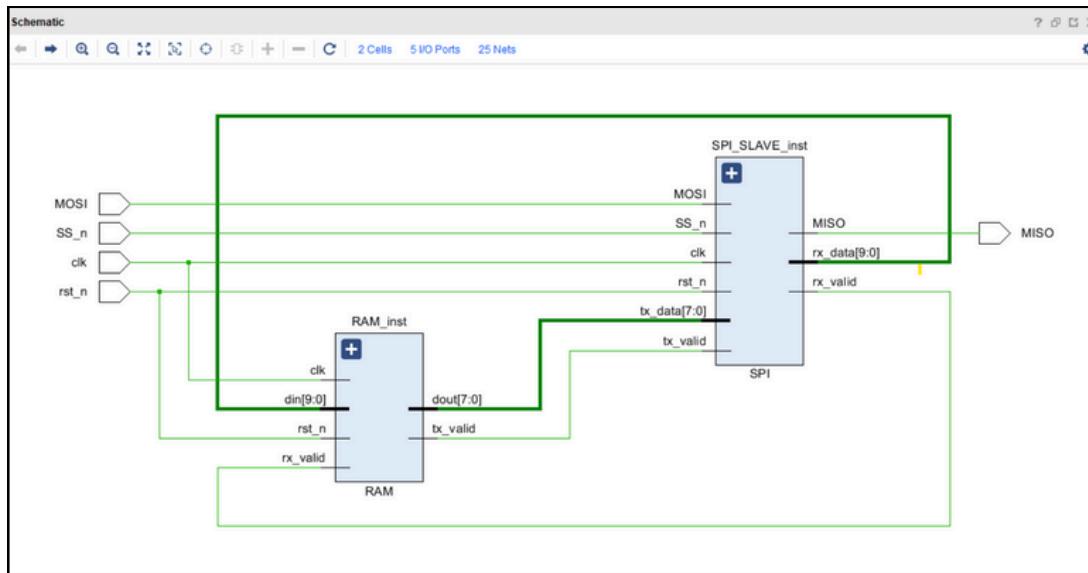
Below the code editor is a "Lint Checks" section with a table:

Severity	Status	Check	Alias	Message	Module	Category
Info	W	parameter_name_duplicate		Same parameter name is used in more than one module. Parameter MEM_DEPTH, Total count 2, First modul...	RAM	Nomenclature...
Info	W	parameter_name_duplicate		Same parameter name is used in more than one module. Parameter ADDR_SIZE, Total count 2, First modul...	RAM	Nomenclature...
Warning	W	always_signal_assign_large		Always block has more signal assignments than the specified limit. Total count 7, Specified limit 5, Module S...	SPI	Rtl Design Style

- 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.

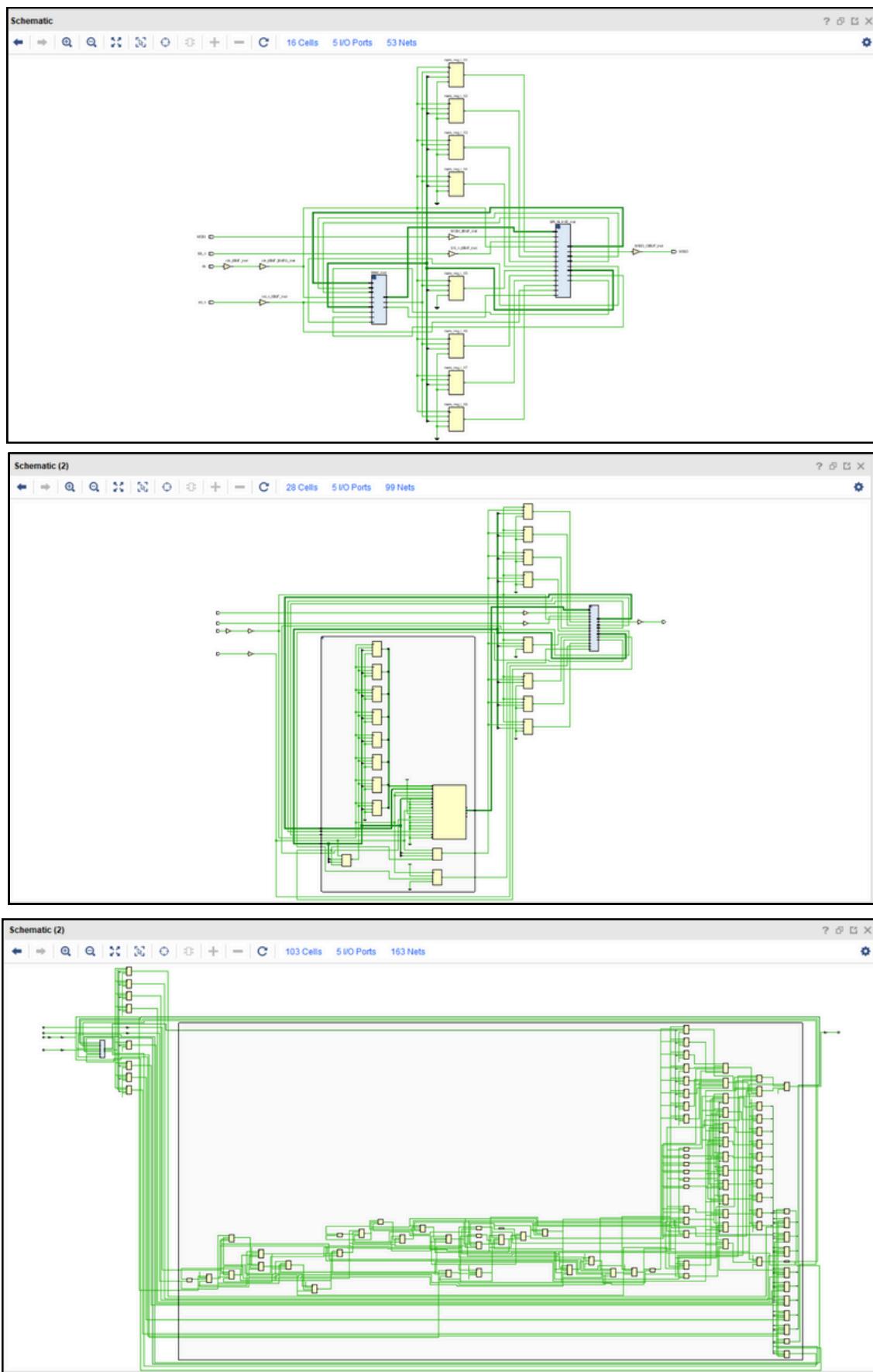
# Elaboration (one hot encoding )

## 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'

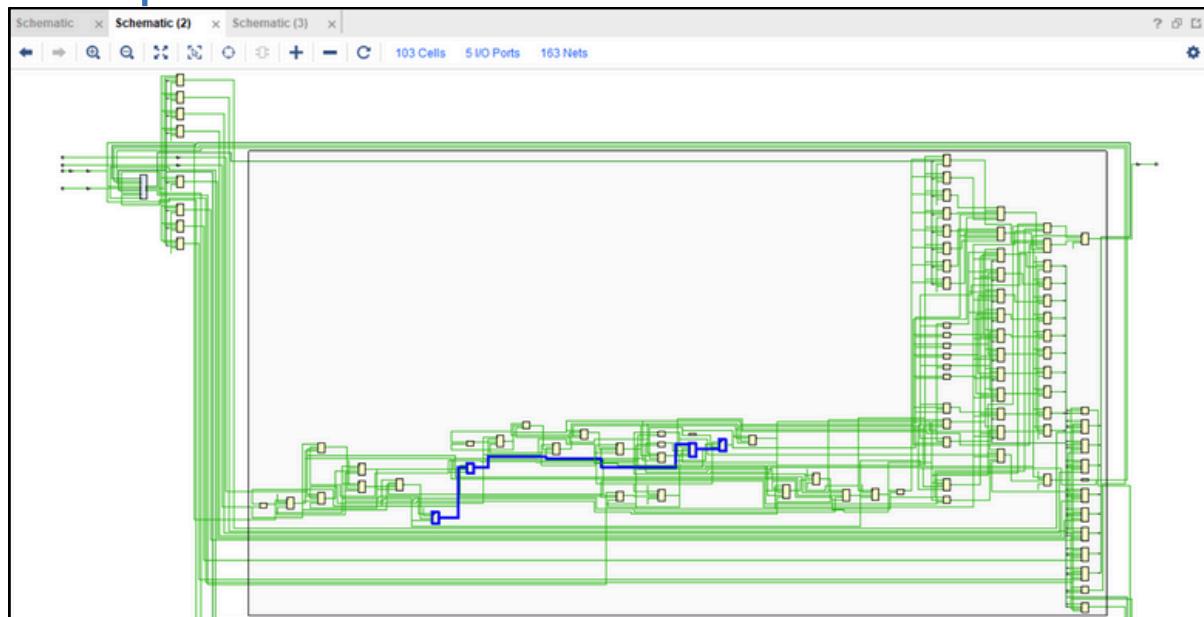
# Timing report



# Utilization report

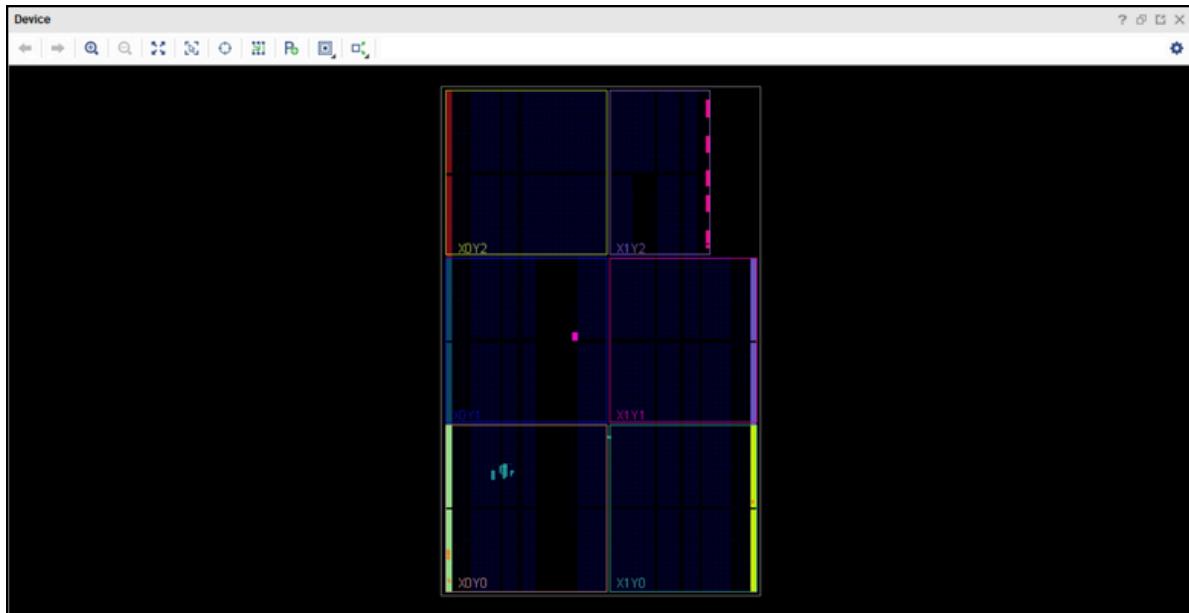
Name	1	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N SPI_WRAPPER	51	48	0.5	5	1	
RAM_inst (RAM)	2	9	0.5	0	0	
SPI_SLAVE_inst (SPI)	49	31	0	0	0	

# Critical path



# Implementation (one hot encoding)

## FPGA device snippet



## Utilization report

Name	1	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)
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

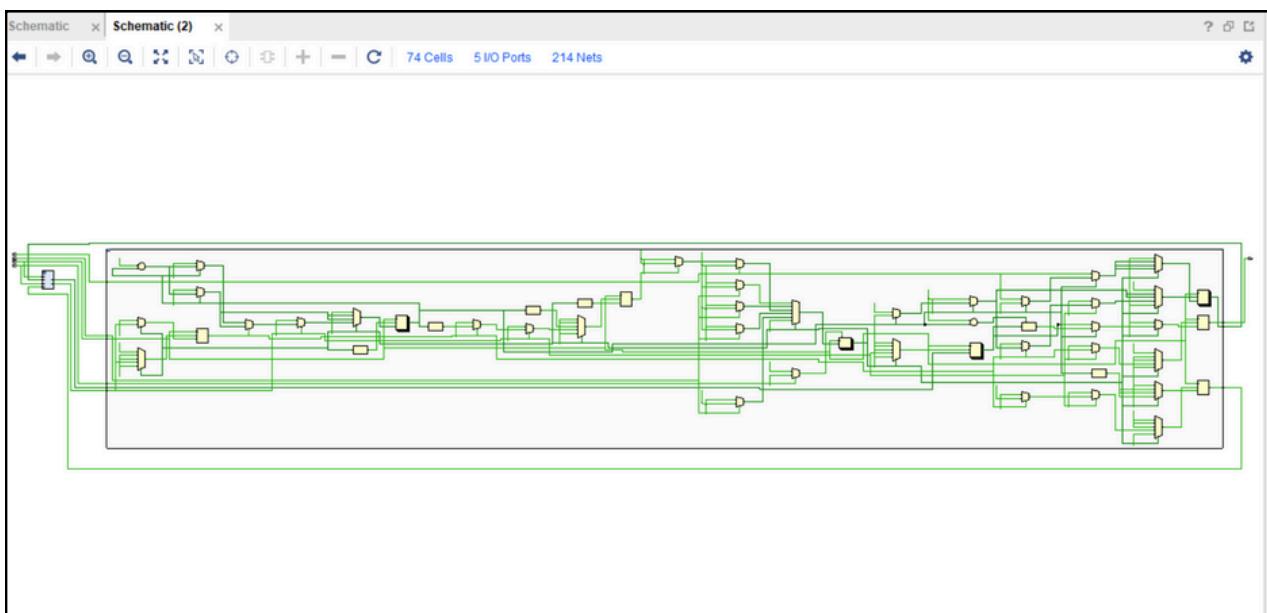
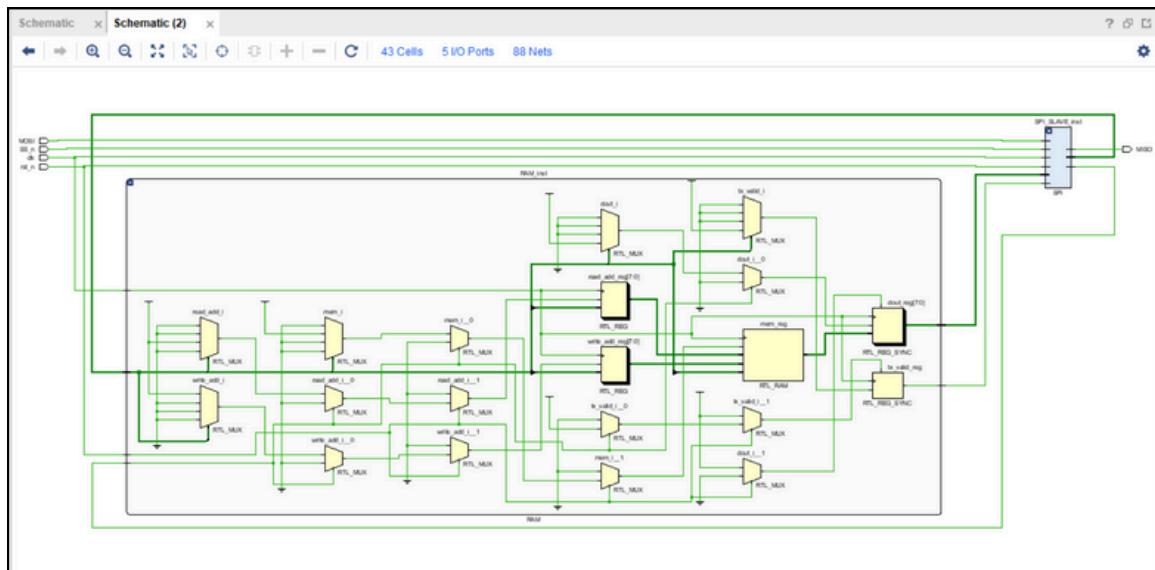
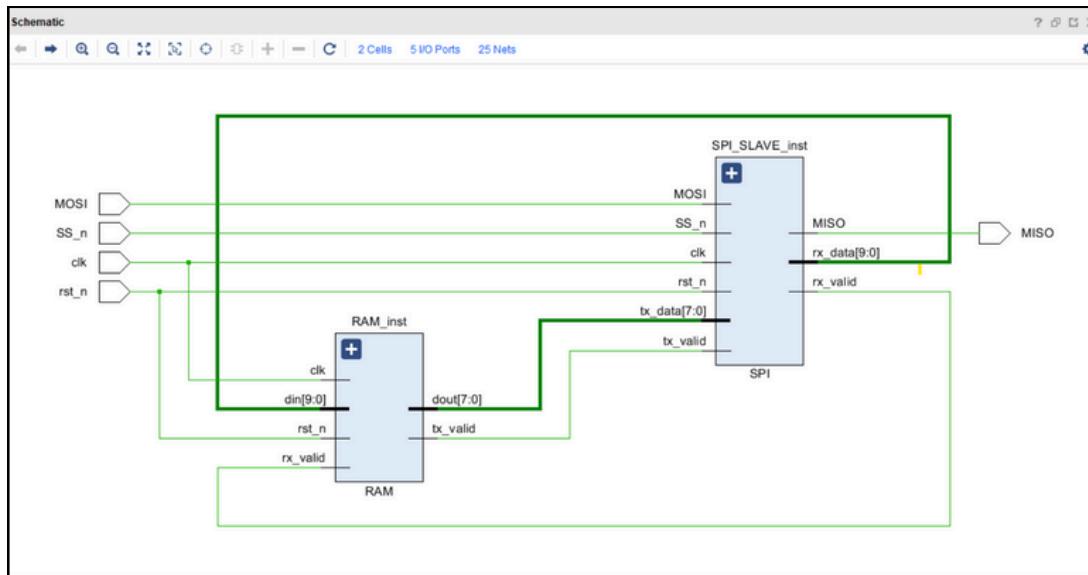
All user specified timing constraints are met.

## Messages tab

YCI Console									
Messages									
Log	Reports	Design Runs	Power	Methodology	Timing	Utilization			
Search	Filter	Search	Log	Reports	Design Runs	Power	Methodology	Timing	Utilization
<input checked="" type="checkbox"/> Warning (1) <input type="checkbox"/> Info (269) <input type="checkbox"/> Status (503) Show All									
Synthesis (1 warning)									
[Constraints 18-5210] No constraint will be written out.									

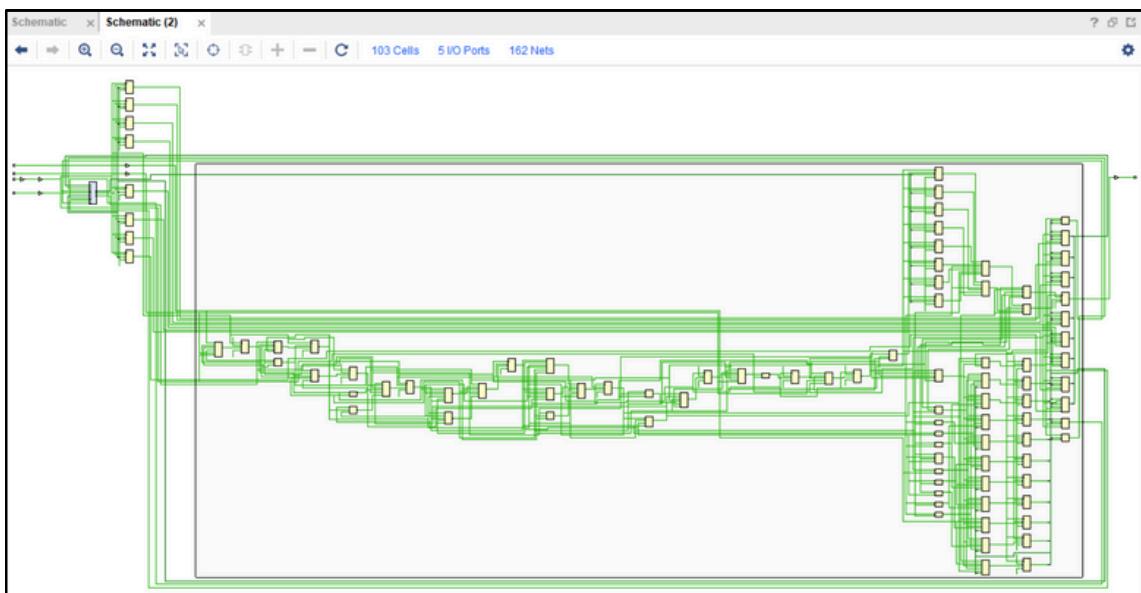
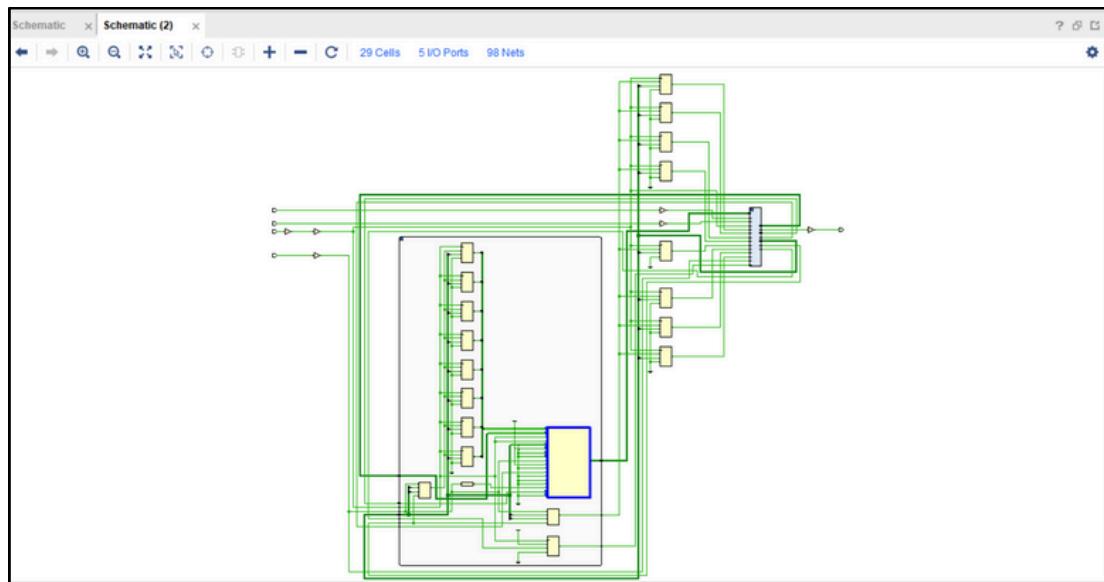
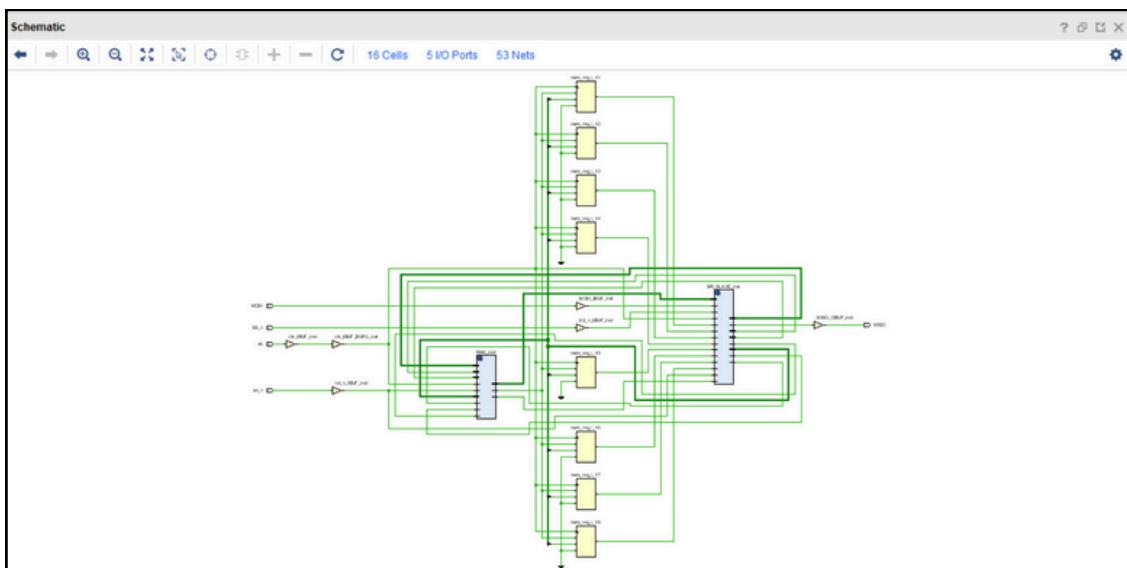
# Elaboration (gray encoding )

## 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

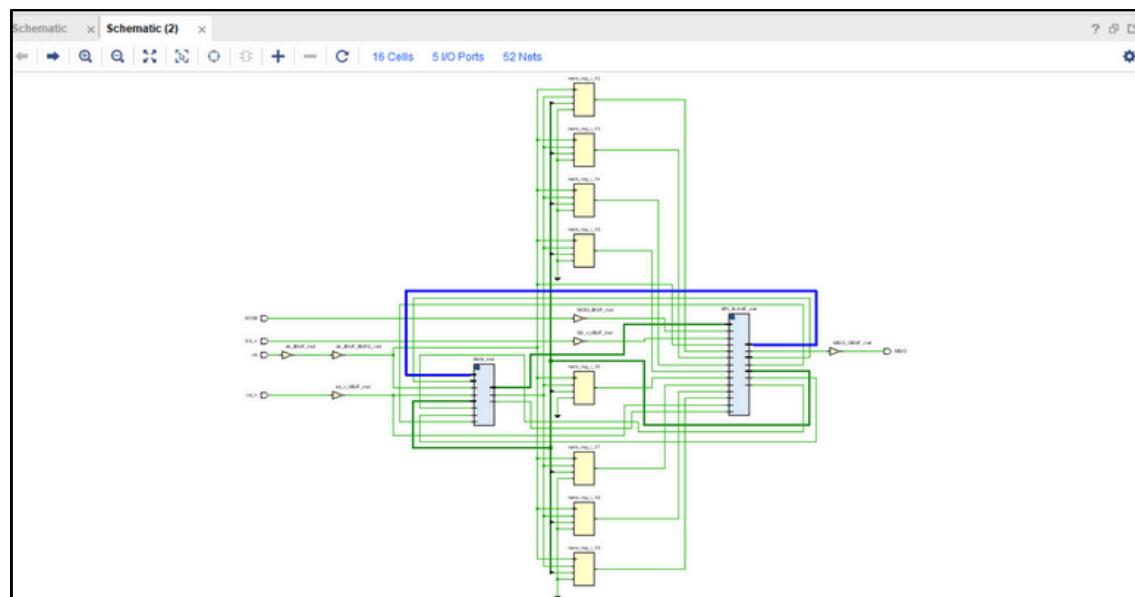
Design Timing Summary						
Setup	Hold		Pulse Width			
Worst Negative Slack (WNS): 6.710 ns		Worst Hold Slack (WHS): 0.149 ns		Worst Pulse Width Slack (WPWS): 4.500 ns		
Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns		Total Pulse Width Negative Slack (TPWS): 0.000 ns		
Number 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

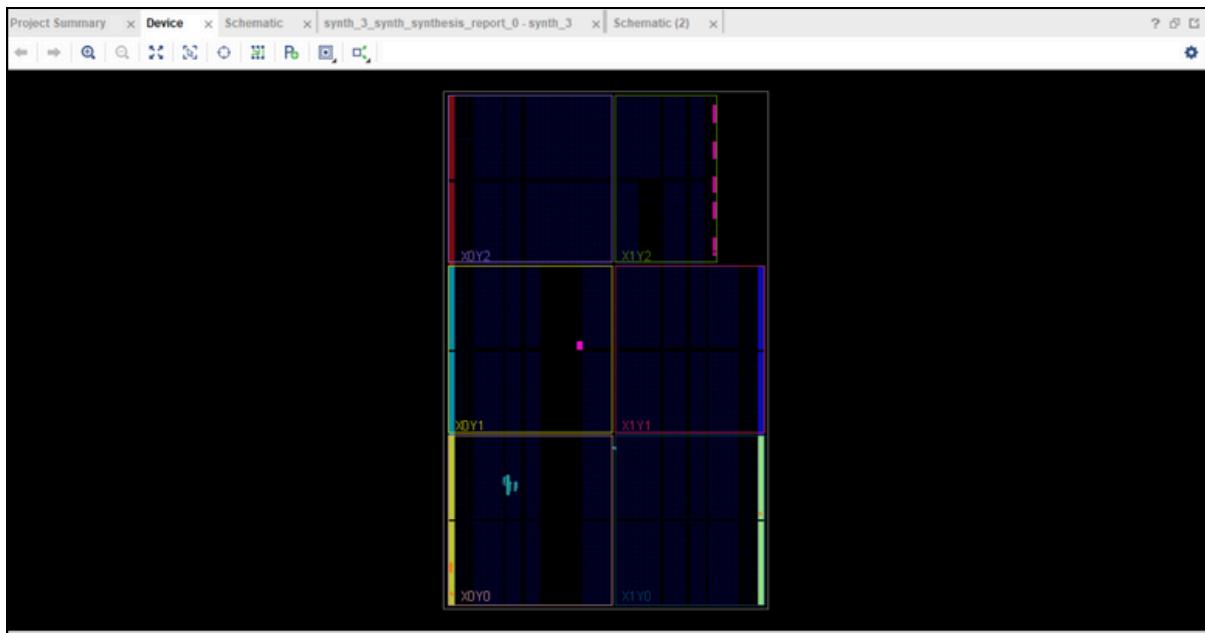
Name	1	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	
N SPI_WRAPPER		51	48	0.5	5	1	
RAM_inst (RAM)		2	9	0.5	0	0	
SPI_SLAVE_inst (SPI)		49	31	0	0	0	

# Critical path



# Implementation (gray encoding)

## FPGA device snippet



## Utilization report

Name	1	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

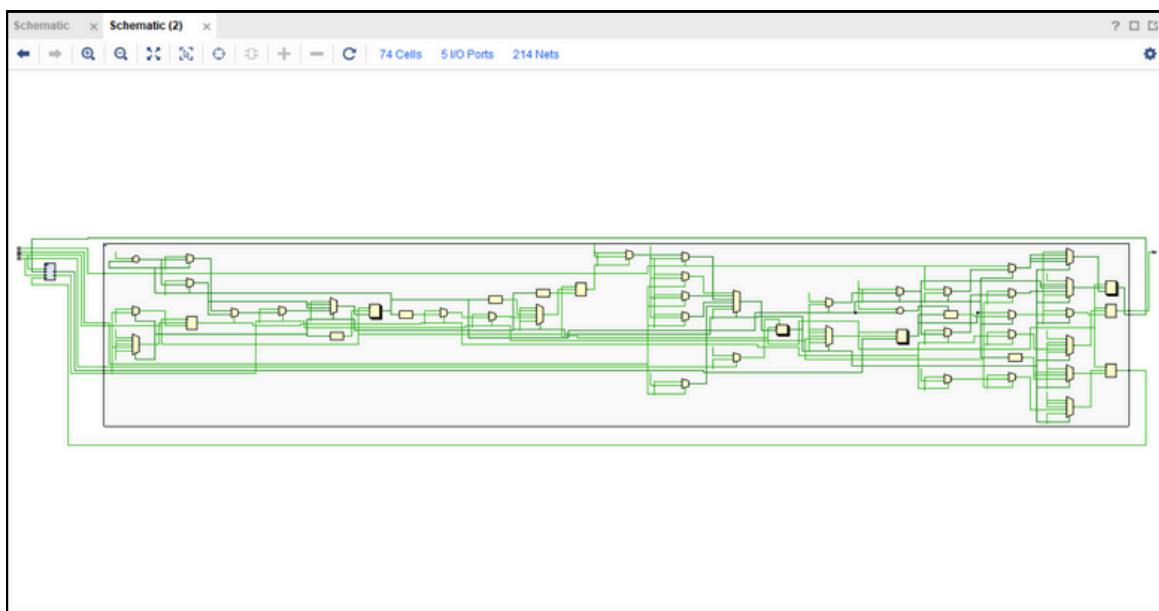
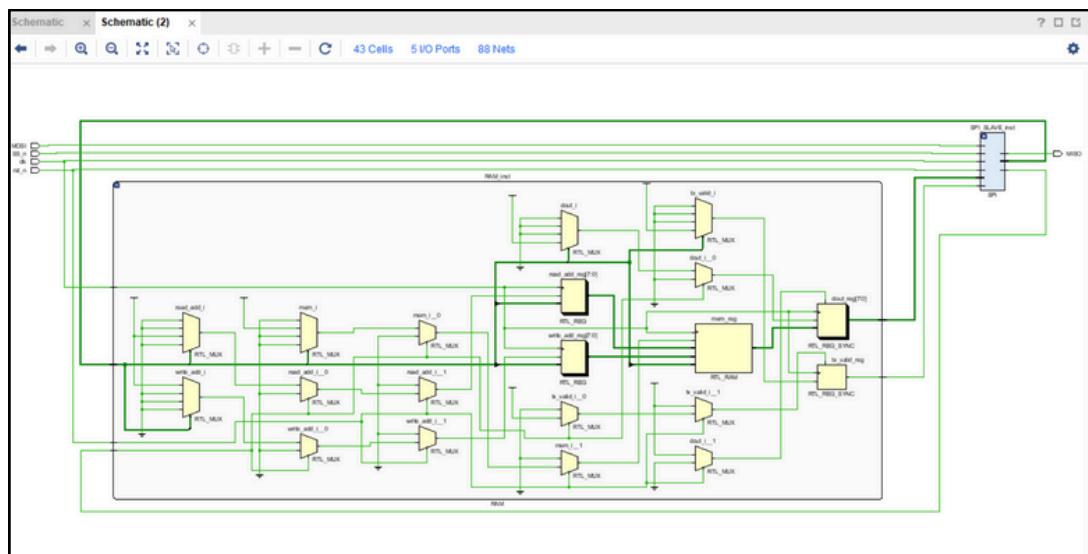
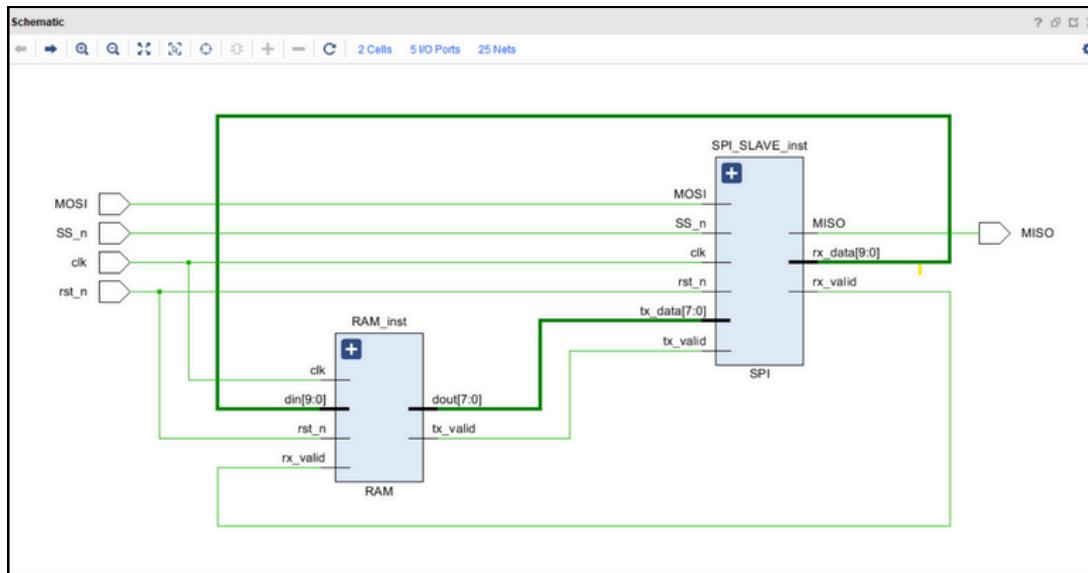
Design Timing Summary			
Setup		Hold	
Worst Negative Slack (WNS):	6.580 ns	Worst Hold Slack (WHS):	0.105 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	103	Total Number of Endpoints:	103
All user specified timing constraints are met.			

## Messages tab

Utilization	Tcl Console	Messages	x	Log	Reports	Design Runs	Power	Methodology	Timing
<input checked="" type="checkbox"/> Warning (1) <input type="checkbox"/> Info (267) <input type="checkbox"/> Status (491) <a href="#">Show All</a>									
Synthesis (1 warning) [Constraints 18-5210] No constraint will be written out.									

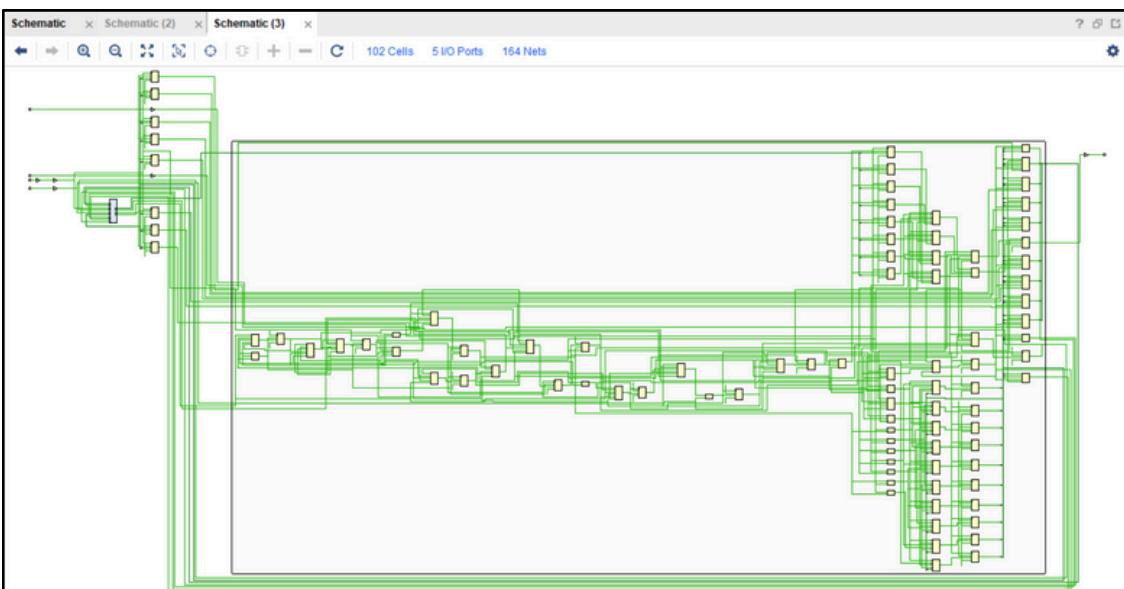
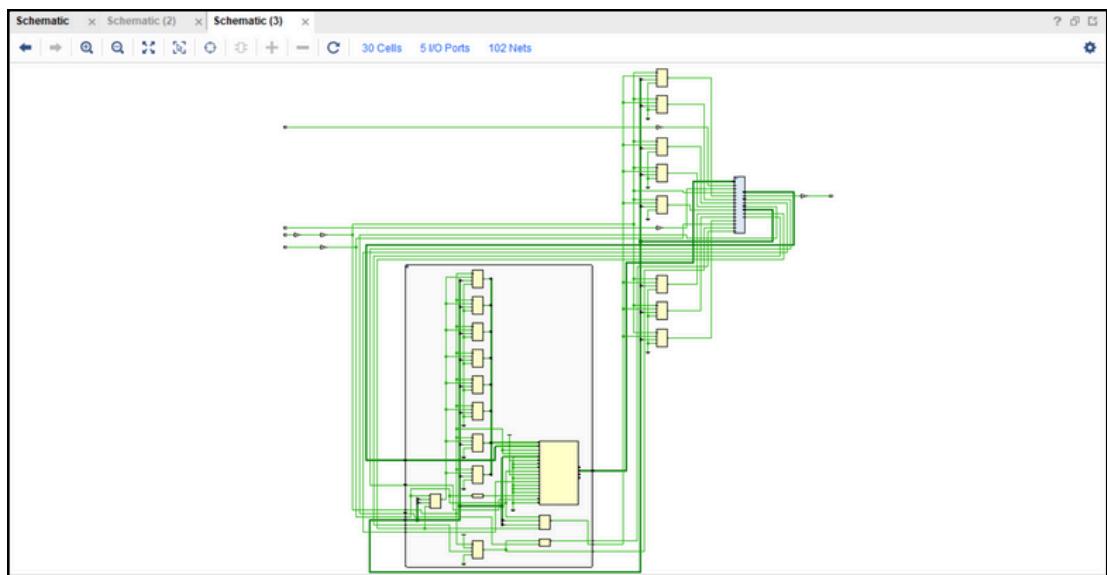
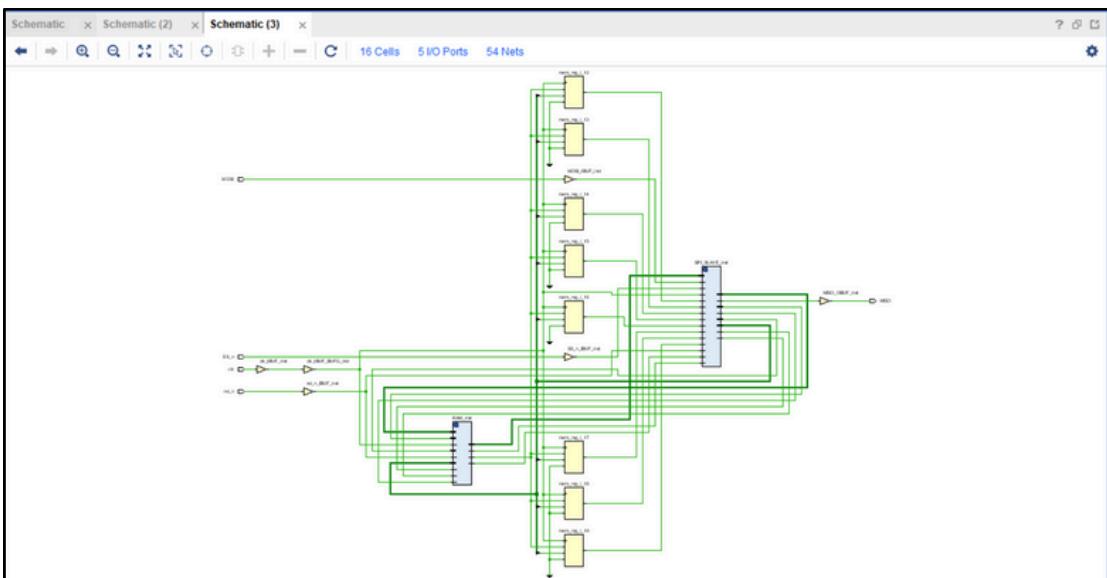
# Elaboration (sequential encoding)

## 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'  
INFO: [Synth 8-3401] The initial RAM was not mapped to memory controller if need, and using direct address mapping.

# Timing report

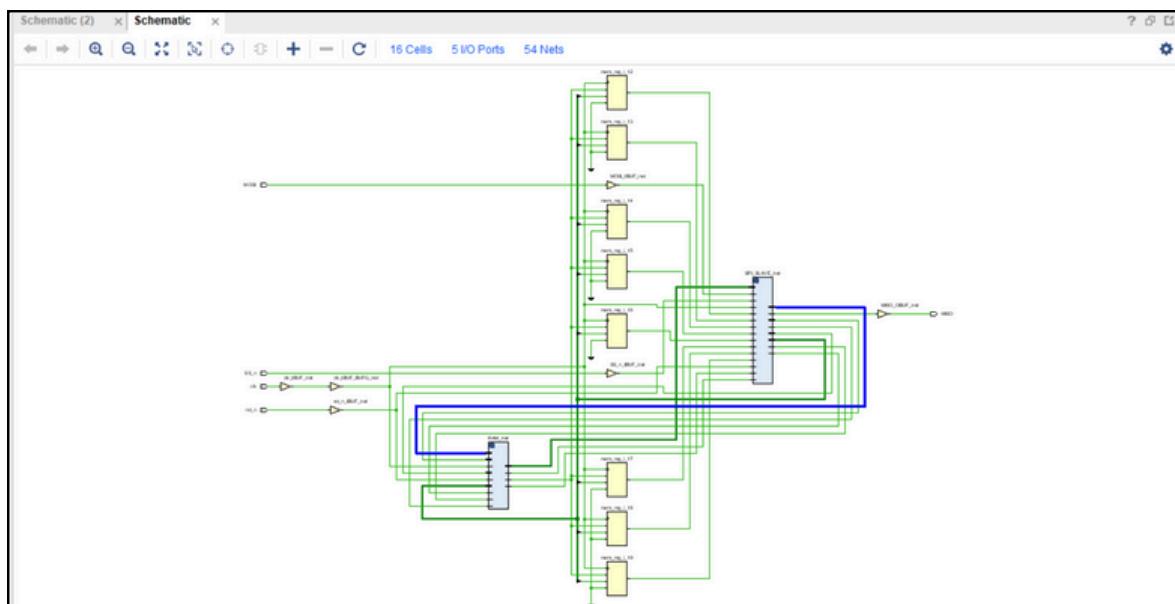
Design Timing Summary						
Setup	Hold		Pulse Width			
Worst Negative Slack (WNS): 6.710 ns		Worst Hold Slack (WHS): 0.150 ns		Worst Pulse Width Slack (WPWS): 4.500 ns		
Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns		Total Pulse Width Negative Slack (TPWS): 0.000 ns		
Number 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

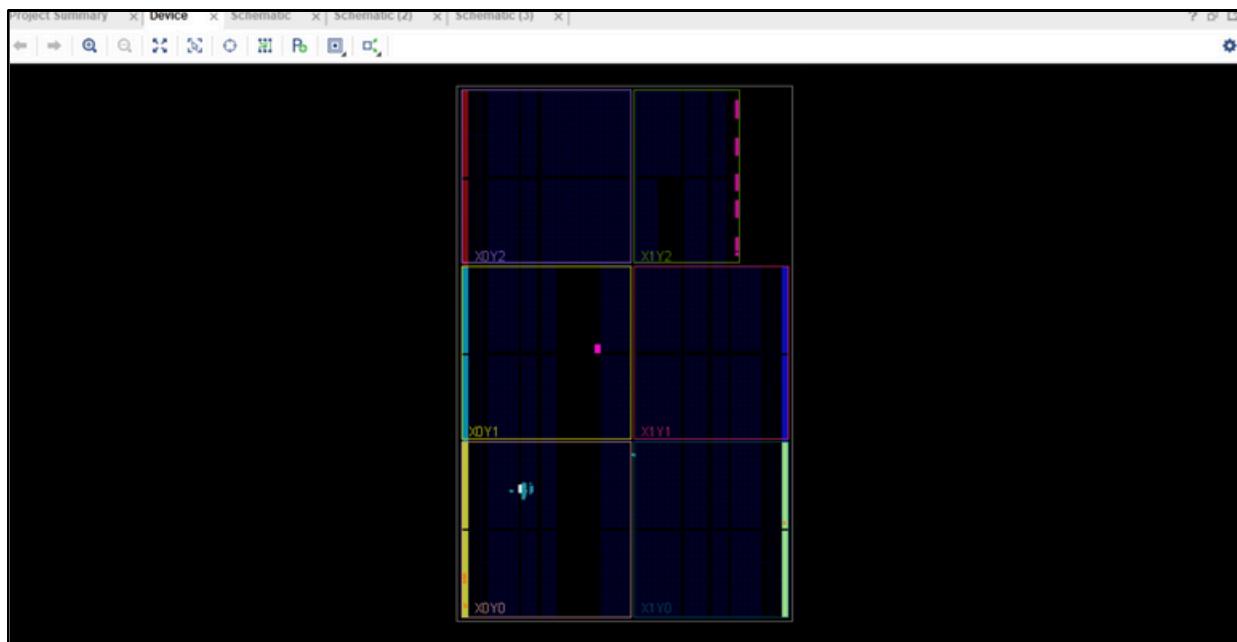
Name	1	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_wrapper	56	46	0.5	5	1	
RAM_inst (RAM)	4	9	0.5	0	0	
SPI_SLAVE_inst (SPI)	52	29	0	0	0	

# Critical path



# Implementation (sequential encoding)

## FPGA device snippet



## Utilization report

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (815 0)	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

Utilization	Tcl Console	Messages	Log	Reports	Design Runs	Power	Methodology	Timing
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<p>Q   E   D   T   M   C   W   I   S   Warning (1)   Info (253)   Status (486)   Show All</p> <p>↳ Synthesis (1 warning) ⌚ [Constraints 18-5210] No constraint will be written out.</p>								

## 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