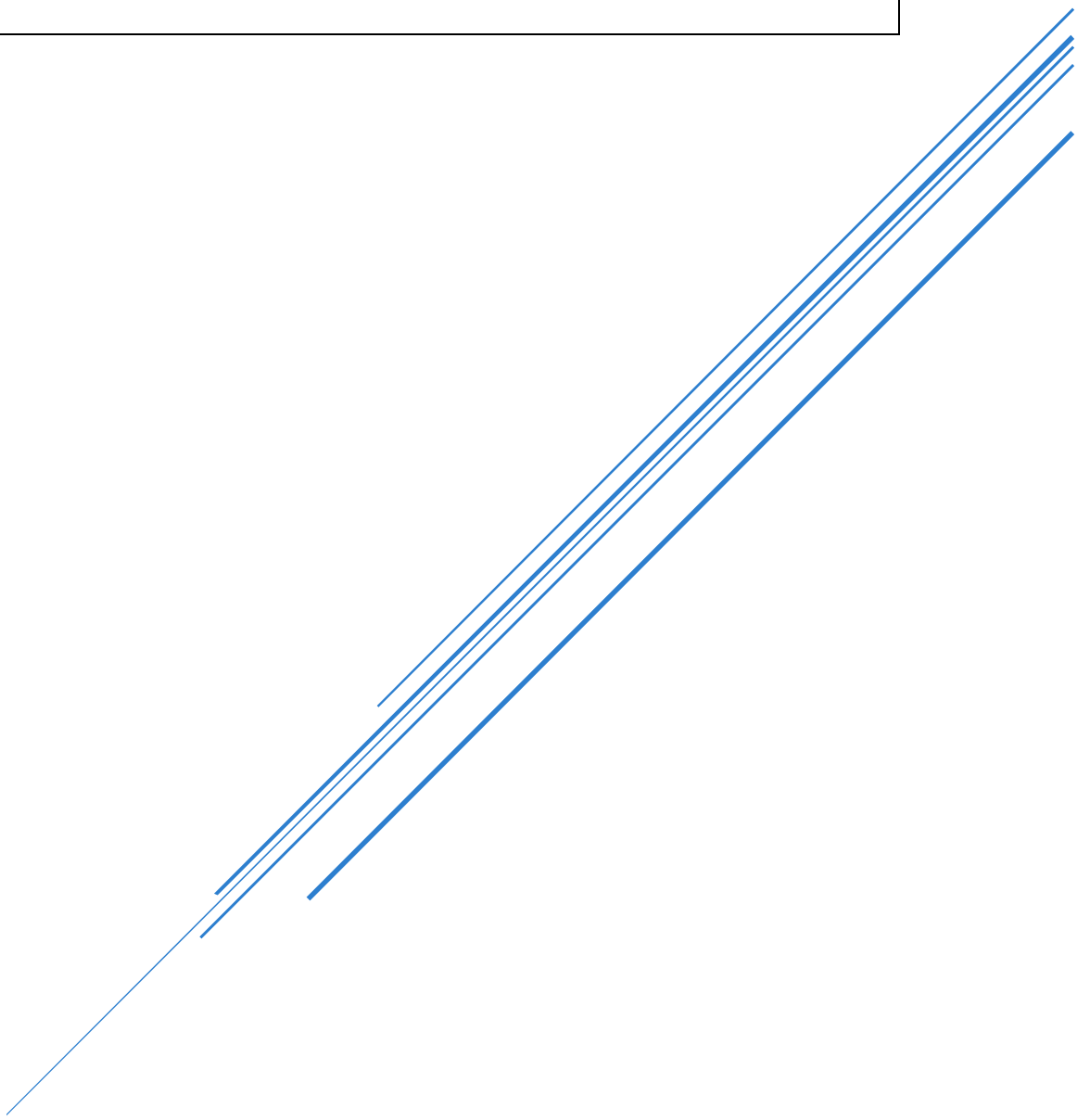
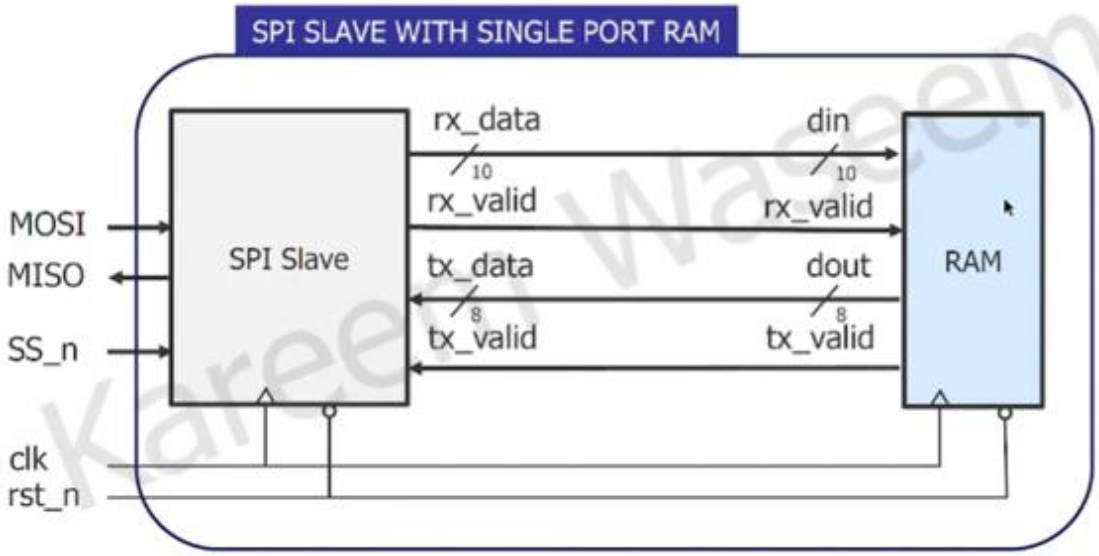
SPI SLAVE WITH SINGLE PORT RAM

Project 2

Bassam Hussam Mashaly
Ziad Tamer Ibrahim
Ebrahim BenBella Sayed Tawfik



Digital Design Diploma



1. RTL code:

a. Single port sync Ram

```
2. module ram #(
3.     parameter MEM_DEPTH = 256,
4.     parameter ADDR_SIZE = 8
5. )(
6.     input [9:0] din,
7.     input clk,
8.     input rst_n,
9.     input rx_valid,
10.    output reg [7:0] dout,
11.    output reg tx_valid
12.);
13.    reg [7:0] mem [0:MEM_DEPTH-1];
14.    reg [ADDR_SIZE-1:0] wr_addr;
15.    reg [ADDR_SIZE-1:0] rd_addr;
16.    always @(posedge clk) begin
17.        if(!rst_n) begin
18.            wr_addr <= 0;
19.            rd_addr <= 0;
20.            dout <= 0;
21.            tx_valid <= 0;
22.        end
23.        else begin
24.                case (din[9:8])
25.                    2'b00: if(rx_valid) begin tx_valid <= 0; wr_addr <= din[7:0]; end
26.                    2'b01: if(rx_valid) begin tx_valid <= 0; mem[wr_addr] <= din[7:0];
   end
27.                    2'b10: if(rx_valid) begin tx_valid <= 0; rd_addr <= din[7:0]; end
28.                    2'b11: begin
```

```
                        tx_valid <= 1'b1;
                        dout <= mem[rd_addr];
                    end
                    default: tx_valid<=0;
                endcase
        end
    end
endmodule
```

b. SPI slave

```
module spi_slave #(
    parameter IDLE = 3'b000,
    parameter CHK_CMD = 3'b001,
    parameter WRITE = 3'b010,
    parameter READ_ADD = 3'b011,
    parameter READ_DATA = 3'b100
)(
    input clk,
    input rst_n,
    input MOSI,
    output reg MISO,
    input SS_n,
    output reg [9:0] rx_data,
    output reg rx_valid,
    input [7:0] tx_data,
    input  tx_valid
);
    integer count;
 // try one_hot , gray , sequential.
    (* fsm_encoding = "sequential" *)
    reg [2:0] cs,ns;
    reg addr_read_check;
    always @(posedge clk) begin
        if (!rst_n)
            cs <= IDLE;
        else
            cs <= ns;
    end
    always @(*) begin
        case (cs)
            IDLE: begin
                if (!SS_n) ns = CHK_CMD;
                else ns = IDLE;
            end
            CHK_CMD: begin
                if (SS_n) ns = IDLE;
                else begin
                    if(MOSI) begin
                        if (addr_read_check) ns = READ_DATA;
                        else ns = READ_ADD;
```

```
                end
                else begin
                    ns = WRITE;
                end
            end
        end
        WRITE: begin
            if (!SS_n ) ns = WRITE;
            else if (SS_n) ns = IDLE;
        end
        READ_ADD: begin
            if (!SS_n ) ns = READ_ADD;
            else if (SS_n) ns = IDLE;
        end
        READ_DATA: begin
            if (!SS_n) ns = READ_DATA;
            else if (SS_n) ns = IDLE;
        end
    endcase
end
always @(posedge clk) begin
    if (!rst_n) begin
        count <= 0;
        rx_data <= 0;
        rx_valid <= 0;
        addr_read_check <= 0;
    end else begin
        case (cs)
        IDLE: begin
            rx_data <= 0;
        end
            WRITE: begin
                rx_valid <= 0;
                rx_data <= {rx_data[8:0], MOSI};
                count <= count + 1;
                if(count == 9) begin
                    rx_valid <= 1;
                end
                if(count==10) rx_data <= 0;
            end
            READ_ADD: begin
                rx_valid <= 0;
                rx_data <= {rx_data[8:0], MOSI};
                count <= count + 1;
                if(count == 9) begin
                    rx_valid <= 1;
                end
                if(count==10) rx_data <= 0;
                addr_read_check <= 1;
            end
```

```
                READ_DATA: begin
                    rx_valid <= 0;
                    rx_data <= {rx_data[8:0], MOSI};
                    if (tx_valid) begin
                        if (count == 8) count <= 0;
                        else begin
                            MISO <= tx_data[7-count];
                            count <= count + 1;
                        end
                    end
                    addr_read_check <= 0;
                end
                CHK_CMD: begin
                    count <= 0;
                    rx_valid <= 0;
                    rx_data <= 0;
                end
            endcase
        end
    end
endmodule
```

c. SPI_wrapper

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

    // Instantiate RAM
    ram #(
        .MEM_DEPTH(256),
        .ADDR_SIZE(8)
    ) u_ram (
        .din(rx_data),
        .clk(clk),
        .rst_n(rst_n),
        .rx_valid(rx_valid),
        .dout(tx_data),
        .tx_valid(tx_valid)
    );

    // Instantiate SPI Slave
    spi_slave u_spi_slave (
```

```
        .clk(clk),
        .rst_n(rst_n),
        .MOSI(MOSI),
        .MISO(MISO),
        .SS_n(SS_n),
        .rx_data(rx_data),
        .rx_valid(rx_valid),
        .tx_data(tx_data),
        .tx_valid(tx_valid)
    );
endmodule
```

d.Testbench

```
module spi_tb();
    reg rst_n,clk,SS_n;
    wire MISO,tx_valid,rx_valid;
    wire [7:0]dout; wire[9:0]din;
    integer i;
    reg MOSI;
    reg [9:0] in_data;
     ram #(
        .MEM_DEPTH(256),
        .ADDR_SIZE(8)
    ) Ram (
        .din(din),
        .clk(clk),
        .rst_n(rst_n),
        .rx_valid(rx_valid),
        .dout(dout),
        .tx_valid(tx_valid) );
    // Instantiate SPI Slave
    spi_slave u_spi_slave (
        .clk(clk),
        .rst_n(rst_n),
        .MOSI(MOSI),
        .MISO(MISO),
        .SS_n(SS_n),
        .rx_data(din),
        .rx_valid(rx_valid),
        .tx_data(dout),
        .tx_valid(tx_valid)
    );
    initial begin
        clk=0;
        forever #1 clk=~clk;
    end
    initial begin
        $readmemh("spi_mem.dat",Ram.mem);
        rst_n=0;
        SS_n=0;
```

```
        @(negedge clk);
        rst_n=1; MOSI=0; @(negedge clk);
   // Test 1: Valid write address command
        in_data=10'b00_10101011;  // {2'b00, 8'hAB}
 @(negedge clk);
            for (i = 9; i >= 0; i = i - 1) begin
                MOSI = in_data[i];
                @(negedge clk);  // Change data on falling edge
            end
   // Test 2: Valid write data command
        SS_n = 1;  // End transaction
        @(negedge clk);
        SS_n = 0; MOSI=0; @(negedge clk);
        in_data=10'b01_11010111;  // {2'b00, 8'hAB}
         @(negedge clk);
        for (i = 9; i >= 0; i = i - 1) begin
            MOSI = in_data[i];
            @(negedge clk);  // Change data on falling edge
        end
   // Test 3: Valid read address command
        SS_n = 1;  // End transaction
        @(negedge clk);
        SS_n = 0; MOSI=1; @(negedge clk);
        in_data=10'b10_10101011;
        @(negedge clk);
        for (i = 9; i >= 0; i = i - 1) begin
            MOSI = in_data[i];
            @(negedge clk);  // Change data on falling edge
        end
        SS_n = 1;
        @(negedge clk);
    // Test 4: Valid read data command
        SS_n = 0; MOSI=1; @(negedge clk);
        in_data = 10'b11_01010010;
        @(negedge clk);
        for (i = 9; i >= 0; i = i - 1) begin
            MOSI = in_data[i];
            @(negedge clk);
        end
        // Wait for MISO data to be read (8 clock cycles)
        repeat(9) @(negedge clk);
        SS_n = 1;
        @(negedge clk);
    $stop;
    end
    initial begin
        $monitor(" SS_n=%b, MOSI=%b, MISO=%b"
                , SS_n, MOSI, MISO);
    end
endmodule
```

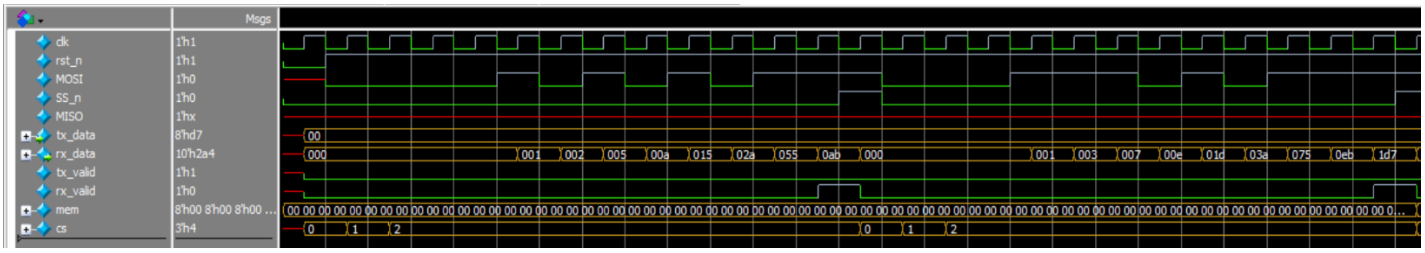
do file:

C: > Ebrahim > Digital_Design > Project2 > run.do

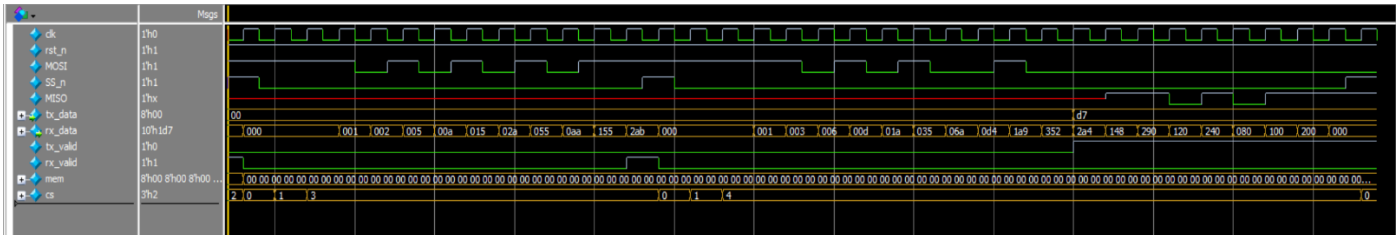
```
vlib work
vlog ram.v spi_slave.v spi_wrapper.v spi_tb.v
vsim -voptargs=+acc work.spi_tb
add wave *
add wave -position insertpoint  \
sim:/spi_tb/Ram/mem \
sim:/spi_tb/Ram/wr_addr \
sim:/spi_tb/Ram/rd_addr
add wave -position insertpoint  \
sim:/spi_tb/u_spi_slave/count \
sim:/spi_tb/u_spi_slave/cs
add wave -position insertpoint  \
sim:/spi_tb/u_spi_slave/rx_data
add wave -position insertpoint  \
sim:/spi_tb/u_spi_slave/ns
run -all
#quit -sim
```

1. QuestaSim snippets:

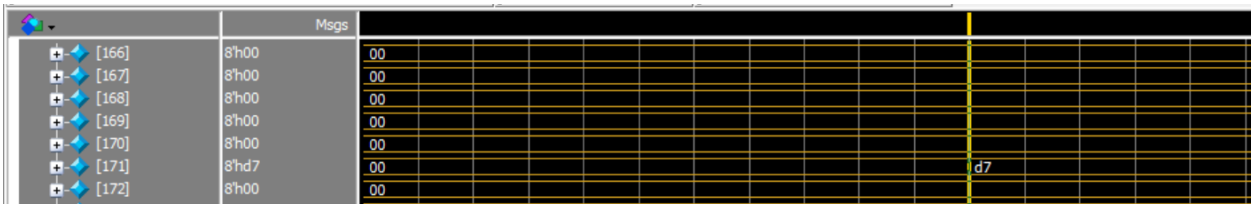
1st and 2nd states



3rd and 4th states

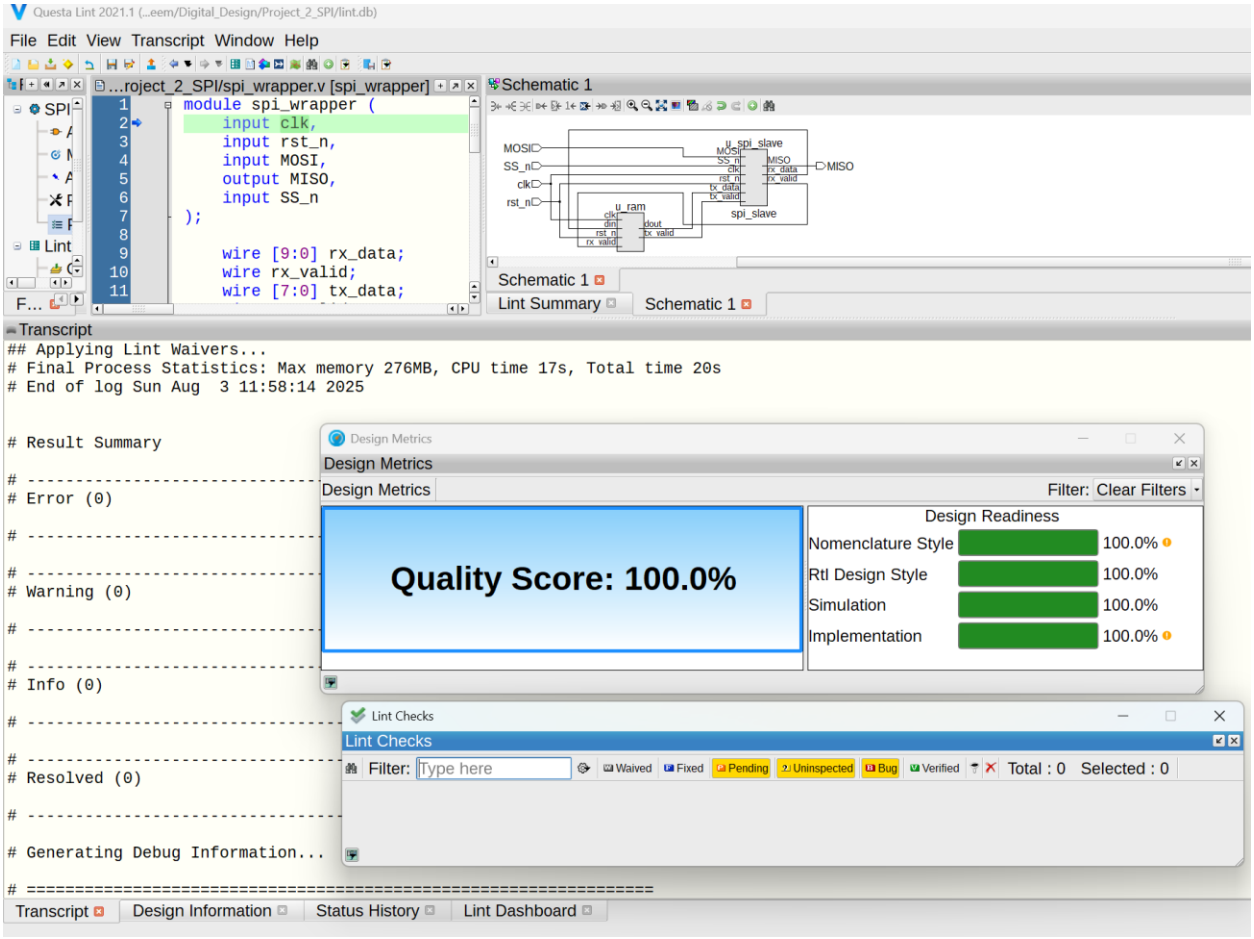


Ram written data



2. Questa lint

```
================================================================================================
Lint Check Report
Questa Lint  Version 2021.1 4558100 win64 28-Jan-2021

Timestamp            : Sun Aug  3 12:15:31 2025
Description          : Report for referring checks count, check violations details, and  design information
Design               : spi_wrapper
Database             : F:/KareemWaseem/Digital_Design/Project_2_SPI/lint.db
Design Quality Score : 100%

Sections:
   Section 1 : Check Summary
   Section 2 : Check Details
   Section 3 : Design Information
================================================================================================


================================================================================================
Section 1 : Check Summary
================================================================================================
-------------
| Error (0) |
-------------

---------------
| Warning (0) |
---------------

------------
| Info (0) |
------------

----------------
| Resolved (0) |
----------------


================================================================================================
Section 2 : Check Details
================================================================================================
-------------
| Error (0) |
-------------


---------------
| Warning (0) |
---------------


------------
| Info (0) |
------------


----------------
| Resolved (0) |
----------------



================================================================================================
Section 3 : Design Information
================================================================================================
-----------
| Summary |
-----------
  Register Bits                         : 81
  Latch Bits                            : 3
  User-specified Blackboxes             : 0
  Inferred Blackboxes                   : 0
  Empty Modules                         : 0
  Unresolved Modules                    : 0
  Hierarchical IPs                      : 0
```



Constraint file :

```
## Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [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}]
#set_property -dict {PACKAGE_PIN W17 IOSTANDARD LVCMOS33} [get_ports {in1[1]}]
#set_property -dict {PACKAGE_PIN W15 IOSTANDARD LVCMOS33} [get_ports {in2[0]}]
#set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {in2[1]}]
#set_property -dict {PACKAGE_PIN W14 IOSTANDARD LVCMOS33} [get_ports {in3[0]}]
#set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports {in3[1]}]
#set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports {B[2]}]
#set_property -dict {PACKAGE_PIN T3 IOSTANDARD LVCMOS33} [get_ports cin]
#set_property -dict {PACKAGE_PIN T2 IOSTANDARD LVCMOS33} [get_ports red_op_A]
#set_property -dict {PACKAGE_PIN R3 IOSTANDARD LVCMOS33} [get_ports red_op_B]
#set_property -dict {PACKAGE_PIN W2 IOSTANDARD LVCMOS33} [get_ports bypass_A]
#set_property -dict {PACKAGE_PIN U1 IOSTANDARD LVCMOS33} [get_ports bypass_B]
#set_property -dict {PACKAGE_PIN T1 IOSTANDARD LVCMOS33} [get_ports direction]
#set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports serial_in]


## LEDs
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {MISO}]
```

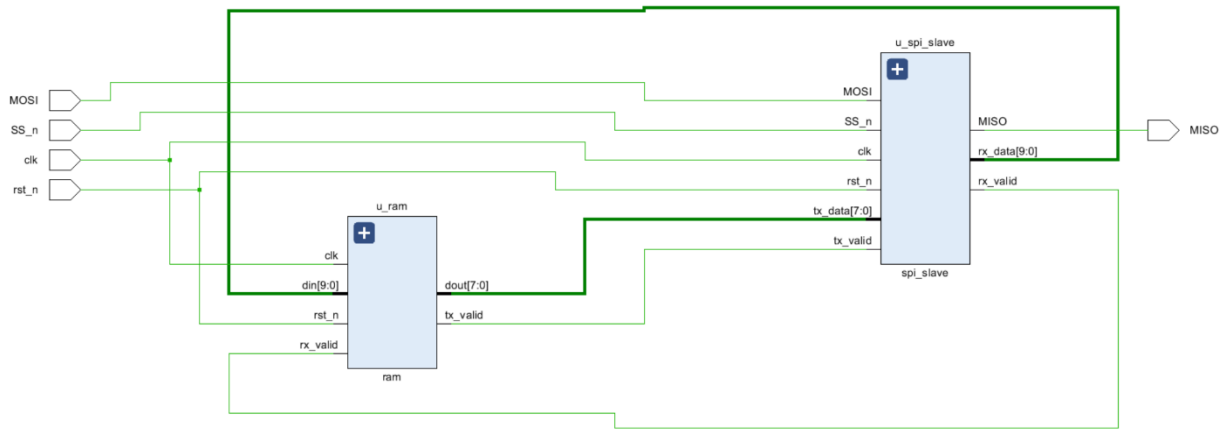
Constraint file after debugging

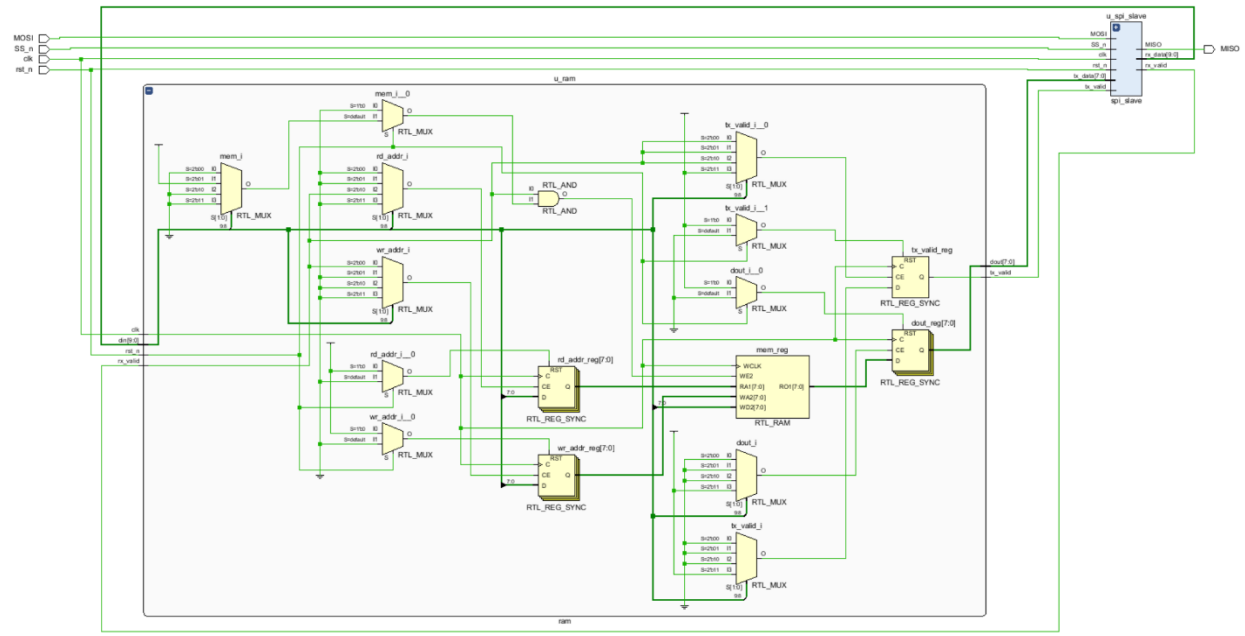
```
set_property MARK_DEBUG true [get_nets clk_IBUF]
set_property MARK_DEBUG true [get_nets clk_IBUF_BUFG]
set_property MARK_DEBUG true [get_nets MISO_OBUF]
set_property MARK_DEBUG true [get_nets MOSI_IBUF]
set_property MARK_DEBUG true [get_nets rst_n_IBUF]
set_property MARK_DEBUG true [get_nets SS_n_IBUF]

create_debug_core u_ila_0 ila
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk_IBUF_BUFG]]
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list clk_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
set_property port_width 1 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list MISO_OBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe2]
set_property port_width 1 [get_debug_ports u_ila_0/probe2]
connect_debug_port u_ila_0/probe2 [get_nets [list MOSI_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe3]
set_property port_width 1 [get_debug_ports u_ila_0/probe3]
connect_debug_port u_ila_0/probe3 [get_nets [list rst_n_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe4]
set_property port_width 1 [get_debug_ports u_ila_0/probe4]
connect_debug_port u_ila_0/probe4 [get_nets [list SS_n_IBUF]]
set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
connect_debug_port dbg_hub/clk [get_nets clk_IBUF_BUFG]
```

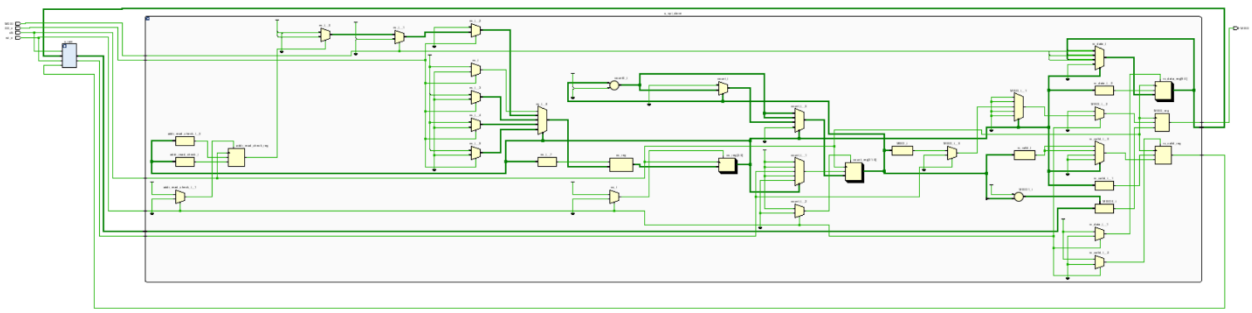
3.Synthesis snippets

3.1 Schematic after the elaboration & synthesis

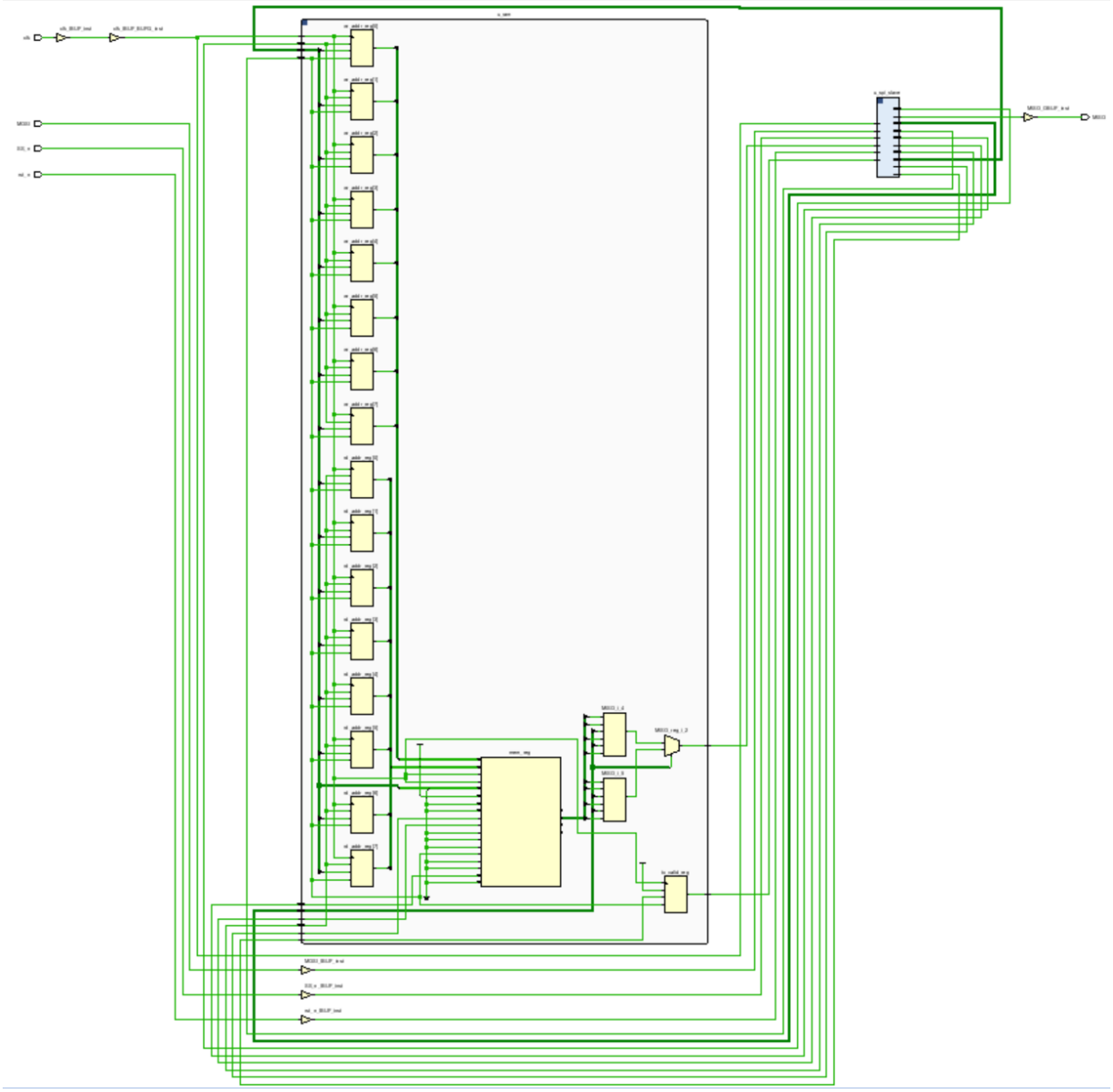
Gray code encoding elaboration

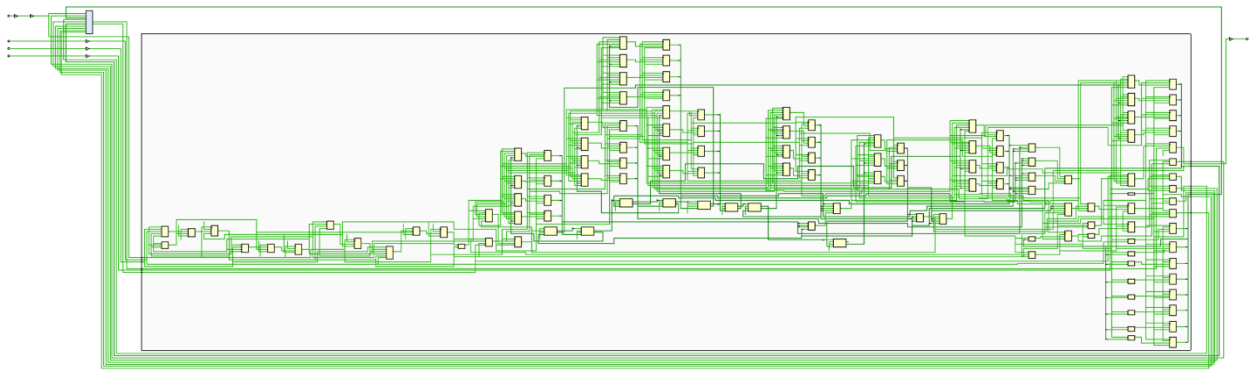






Gray encoding synthesis





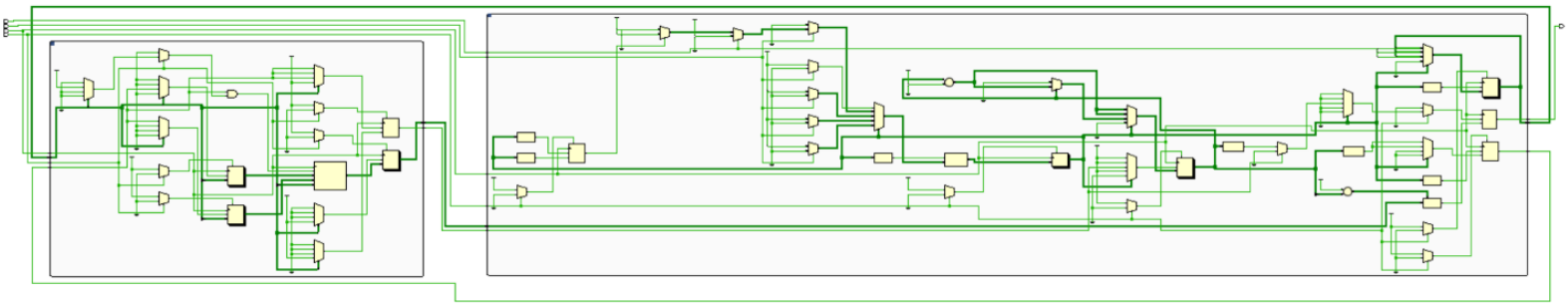
Gray code encoding timing

Design Timing Summary

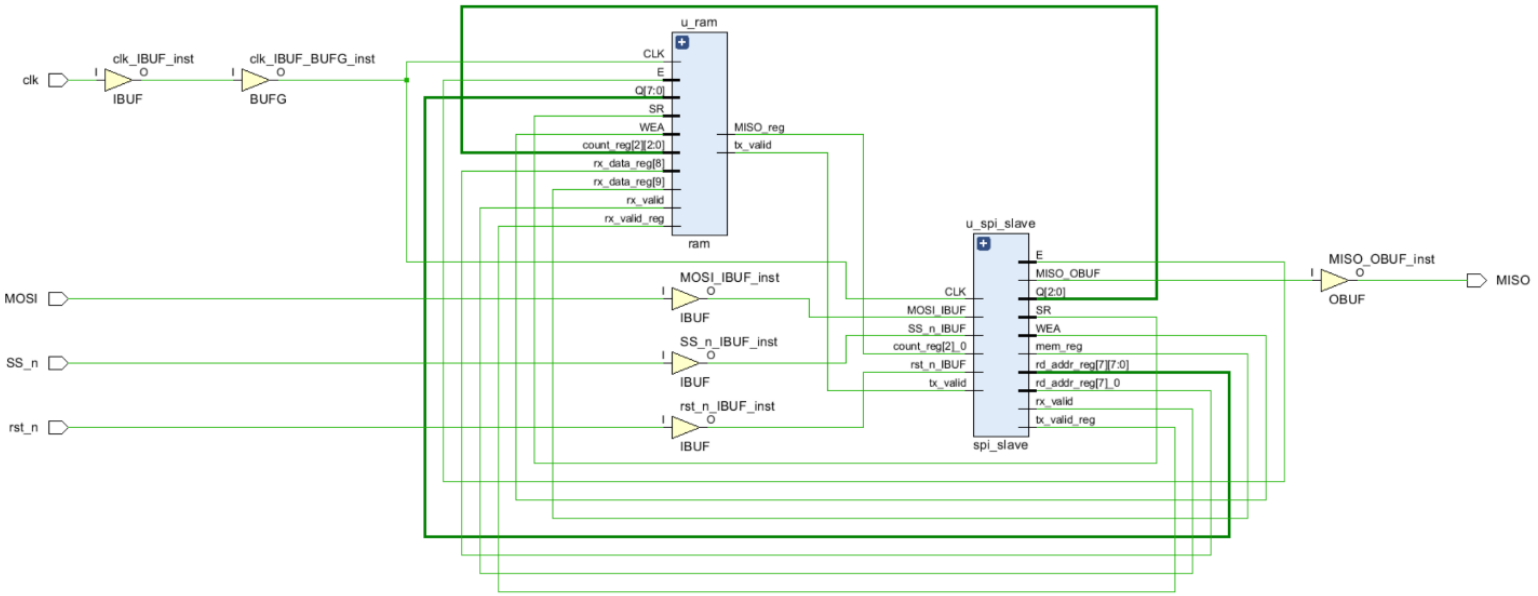
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.517 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	150	Total Number of Endpoints:	150	Total Number of Endpoints:	68

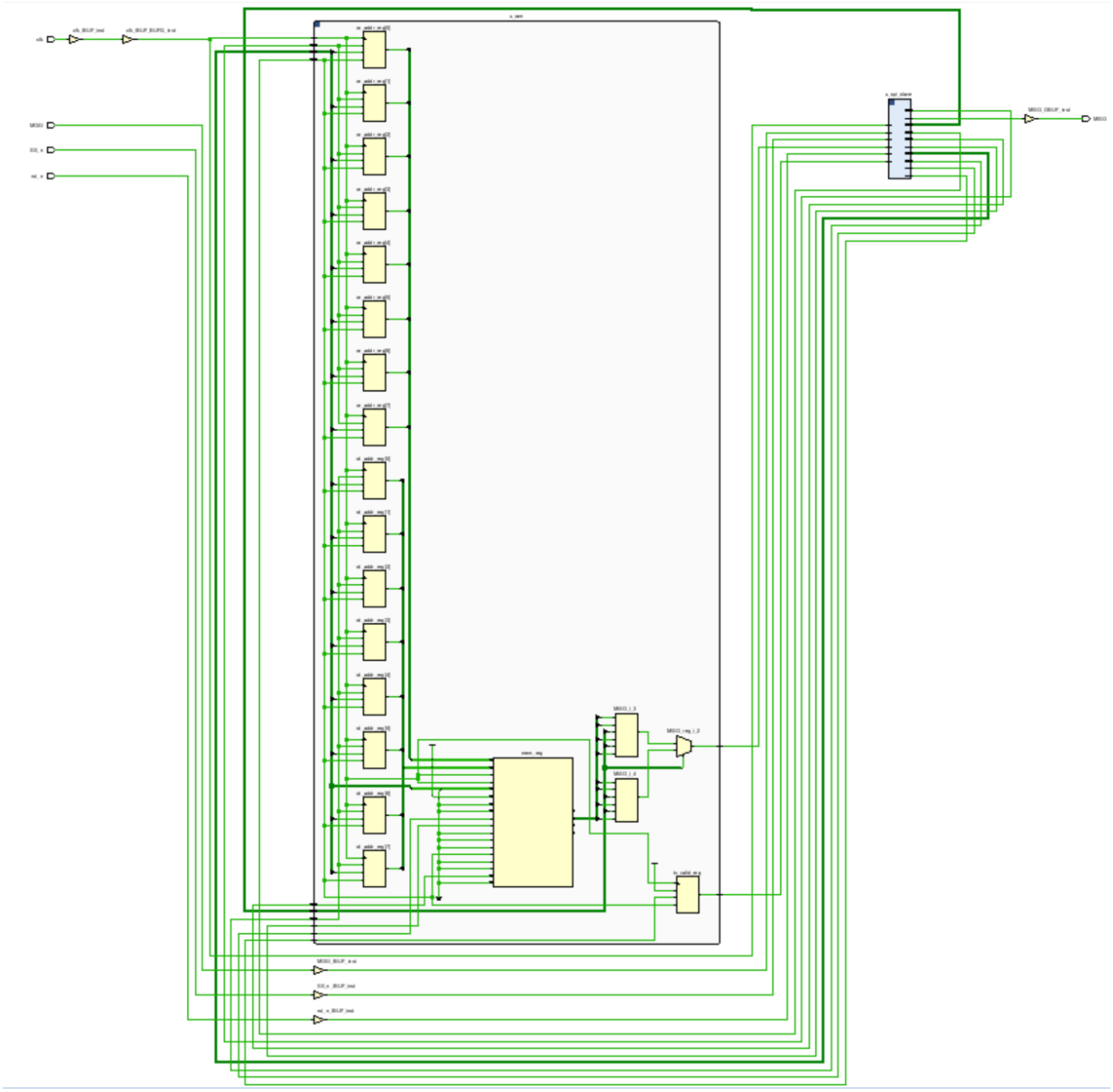
All user specified timing constraints are met.

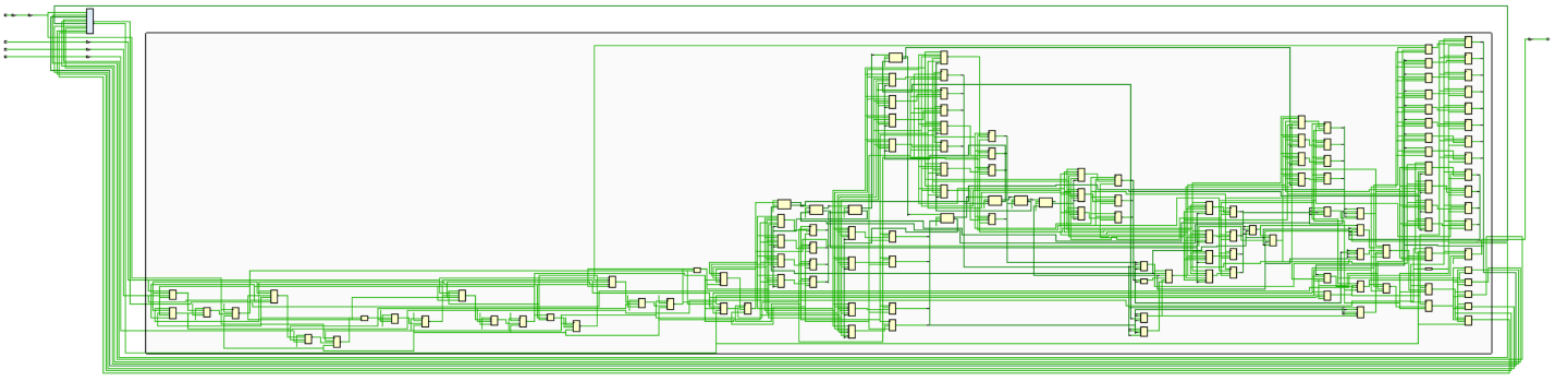
One hot encoding Elaboration



One hot encoding synthesis







one hot encoding timing report

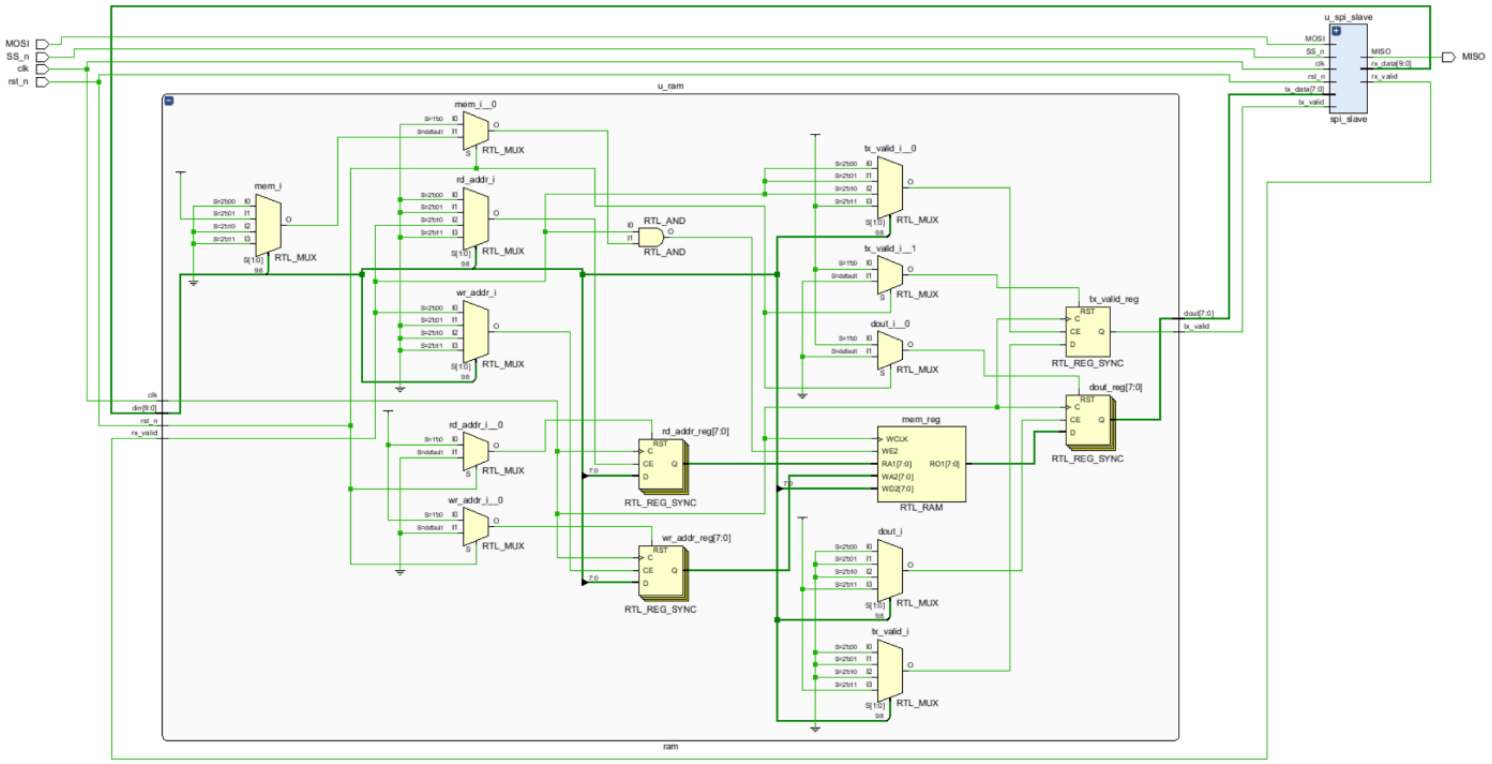
Design Timing Summary

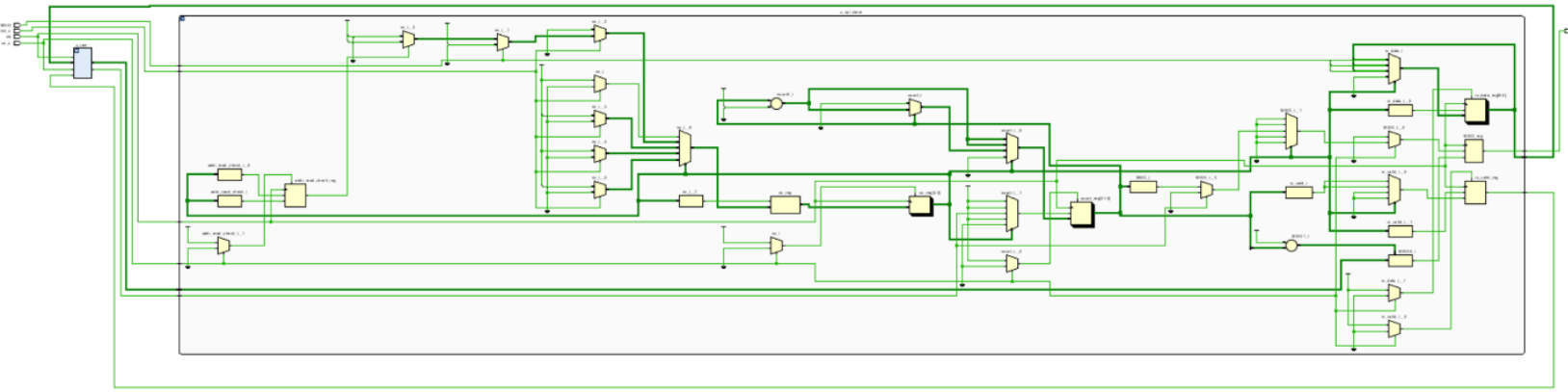
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.869 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	150	Total Number of Endpoints:	150	Total Number of Endpoints:	70

All user specified timing constraints are met.

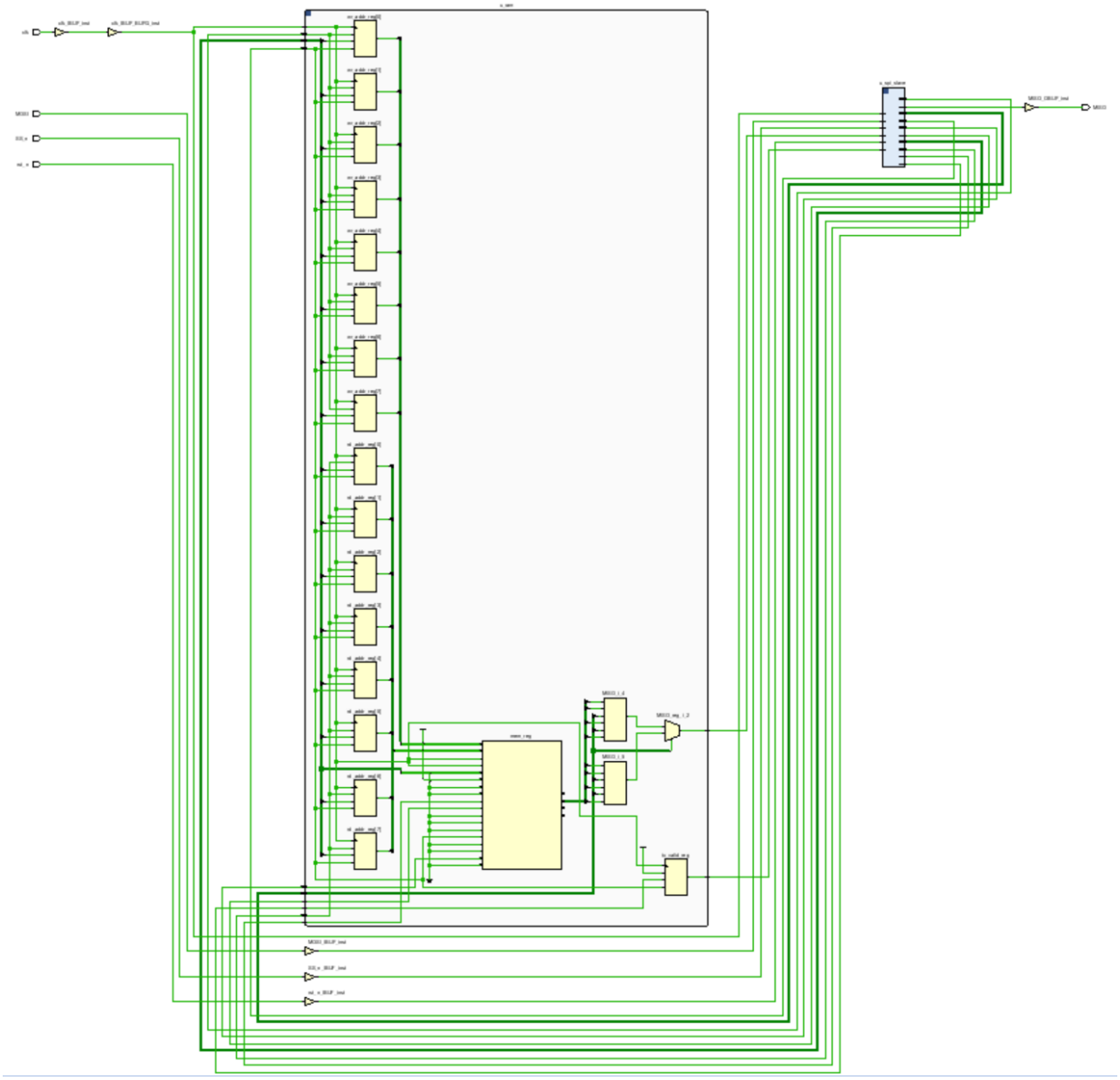
Sequential encoding

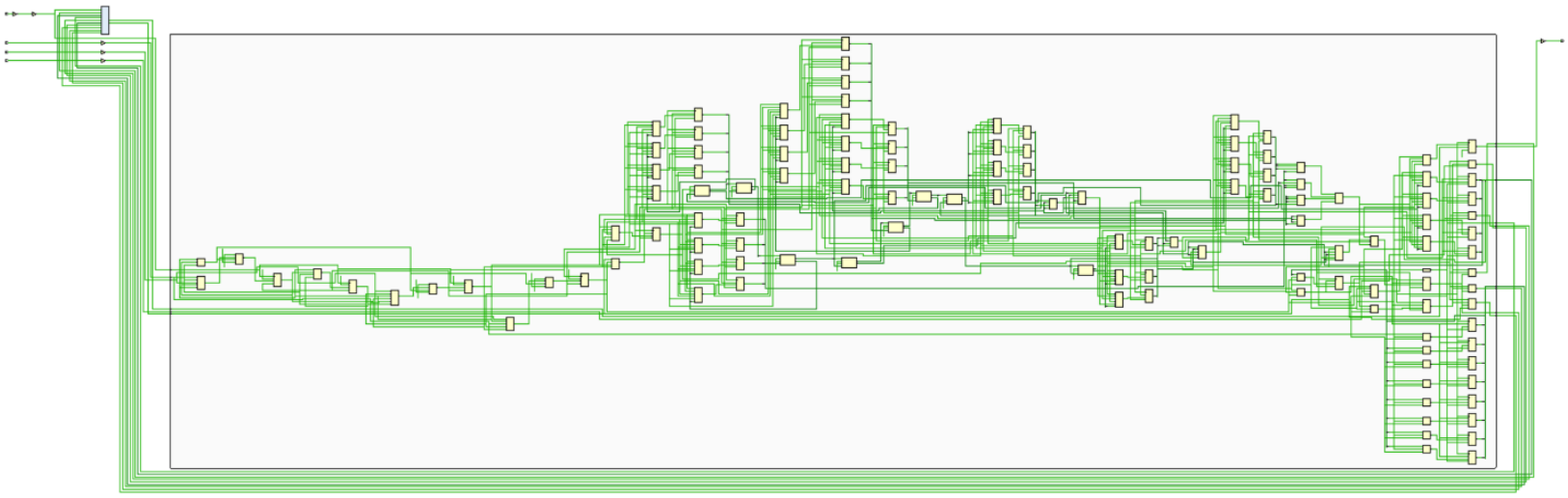
sequential encoding elaboration:





Sequential encoding synthesis





sequential encoding timing:

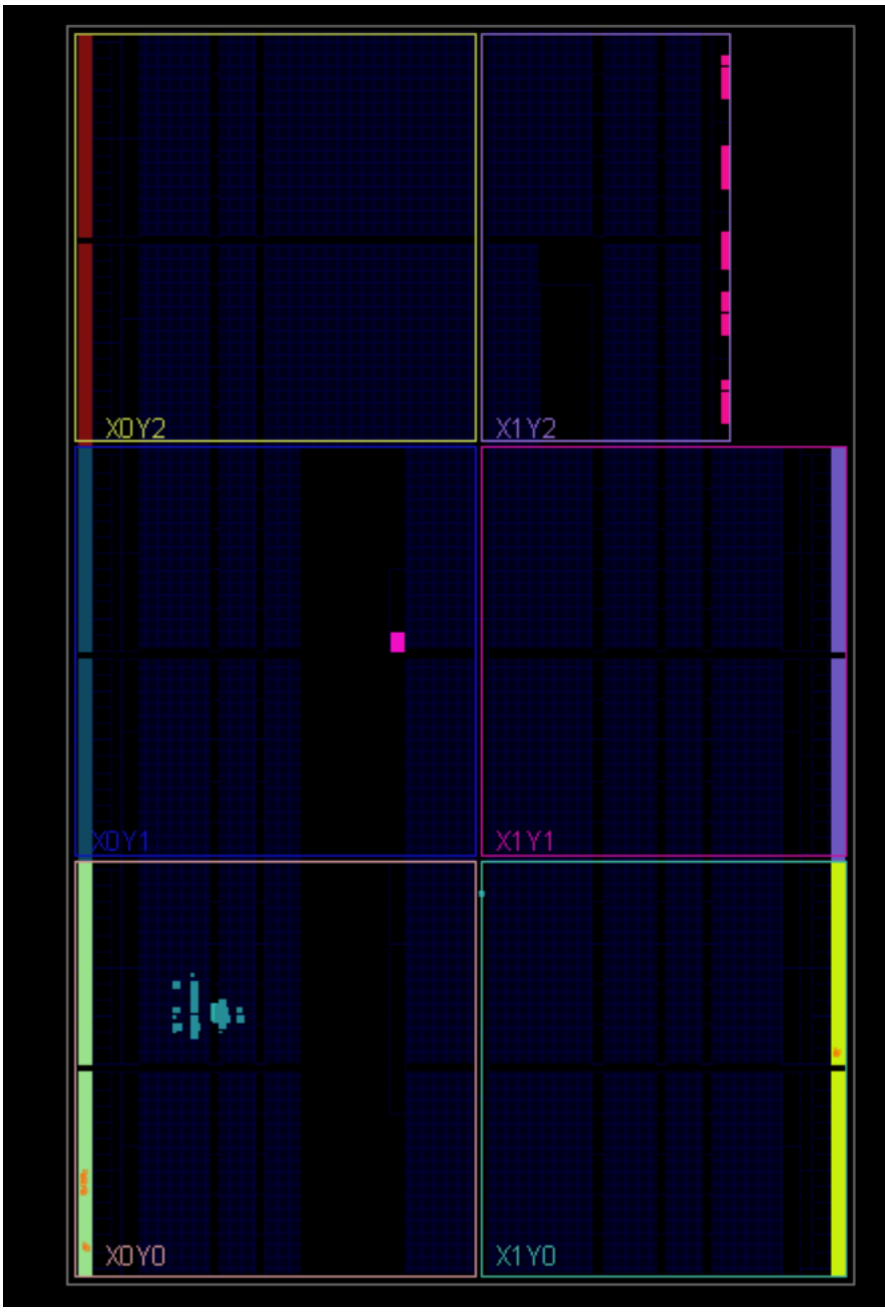
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.517 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:	150	Total Number of Endpoints:	150	Total Number of Endpoints:	68

All user specified timing constraints are met.

4. Implementation snippets:

Gray encoding Device



Gray encoding timing

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.947 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:	150	Total Number of Endpoints:	150	Total Number of Endpoints:	68

All user specified timing constraints are met.

Gray code encoding primitives

Primitives

Ref Name	Used	Functional Category
FDRE	65	Flop & Latch
LUT6	37	LUT
LUT4	12	LUT
LUT2	11	LUT
CARRY4	8	CarryLogic
LUT3	7	LUT
IBUF	4	IO
LUT5	3	LUT
LDCE	3	Flop & Latch
RAMB18E1	1	Block Memory
OBUF	1	IO
MUXF7	1	MuxFx
LUT1	1	LUT
BUFG	1	Clock

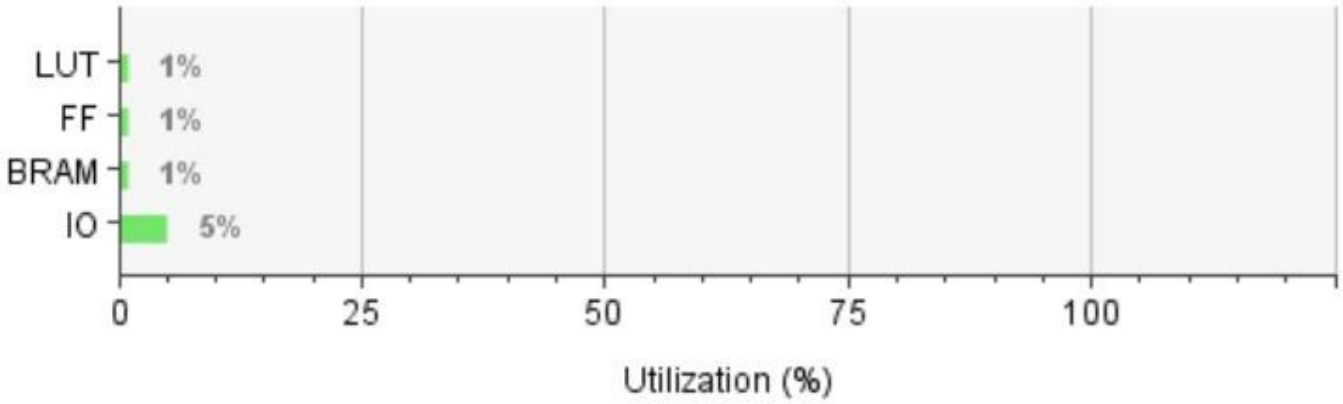
Gray code encoding hierarchy

Hierarchy

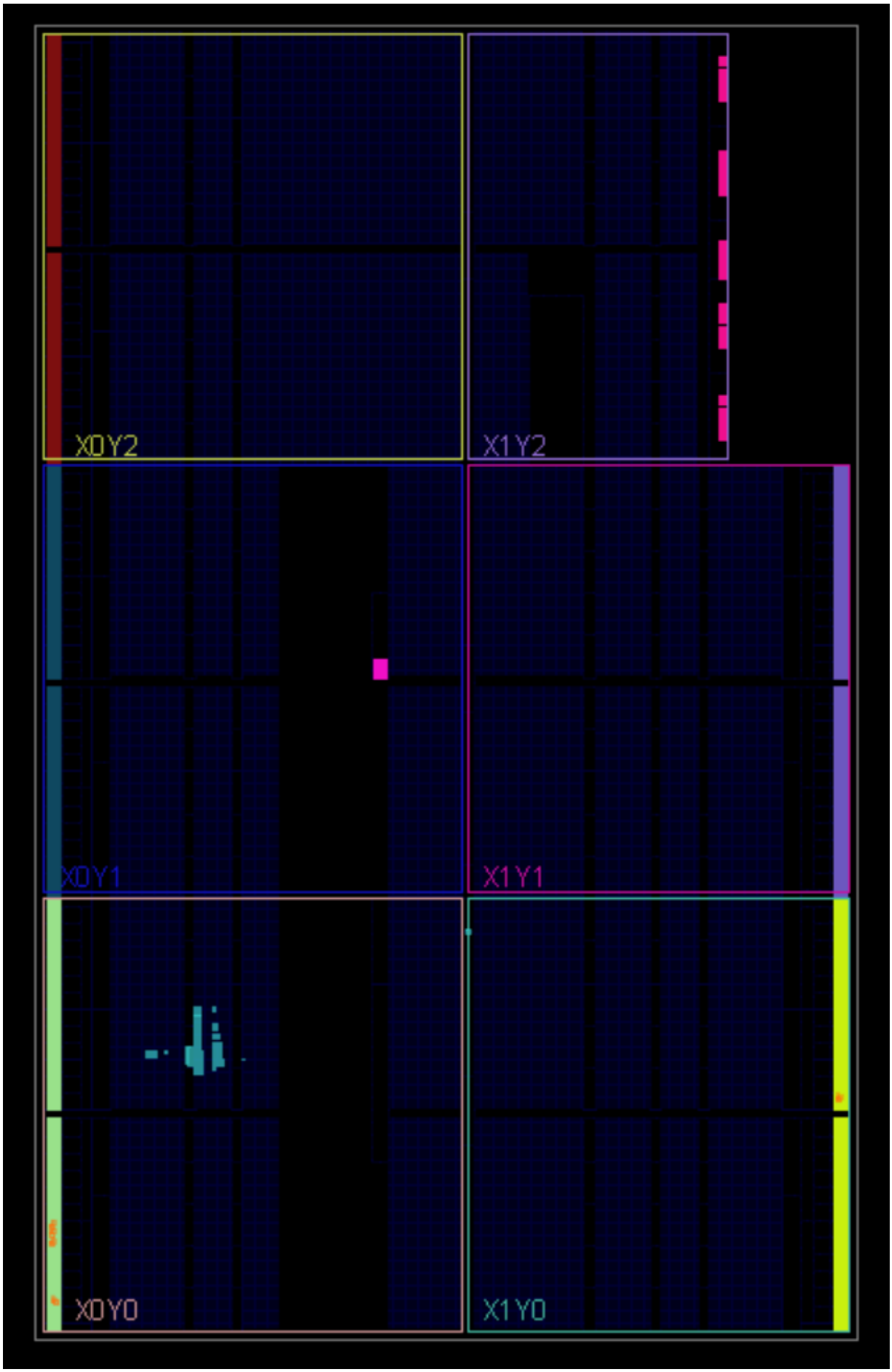
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_wrapper	70	68	1	0.5	5	1
u_ram (ram)	2	17	1	0.5	0	0
u_spi_slave (spi_slave)	68	51	0	0	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	70	20800	0.34
FF	68	41600	0.16
BRAM	0.50	50	1.00
IO	5	106	4.72



One hot encoding



One hot encoding Timing

Design Timing Summary

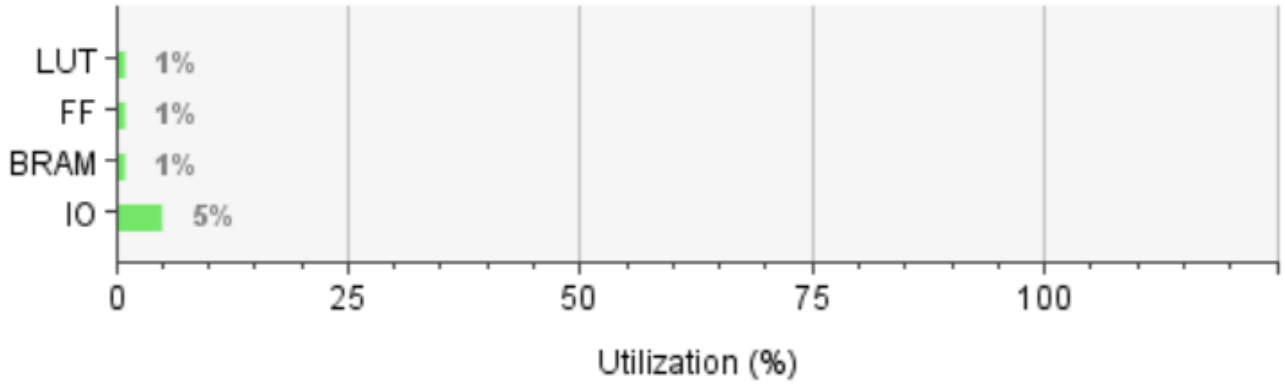
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.072 ns	Worst Hold Slack (WHS):	0.089 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	150	Total Number of Endpoints:	150	Total Number of Endpoints:	70

All user specified timing constraints are met.

One hot encoding utilization

Summary

Resource	Utilization	Available	Utilization %
LUT	69	20800	0.33
FF	68	41600	0.16
BRAM	0.50	50	1.00
IO	5	106	4.72



One hot encoding primitives :

Primitives

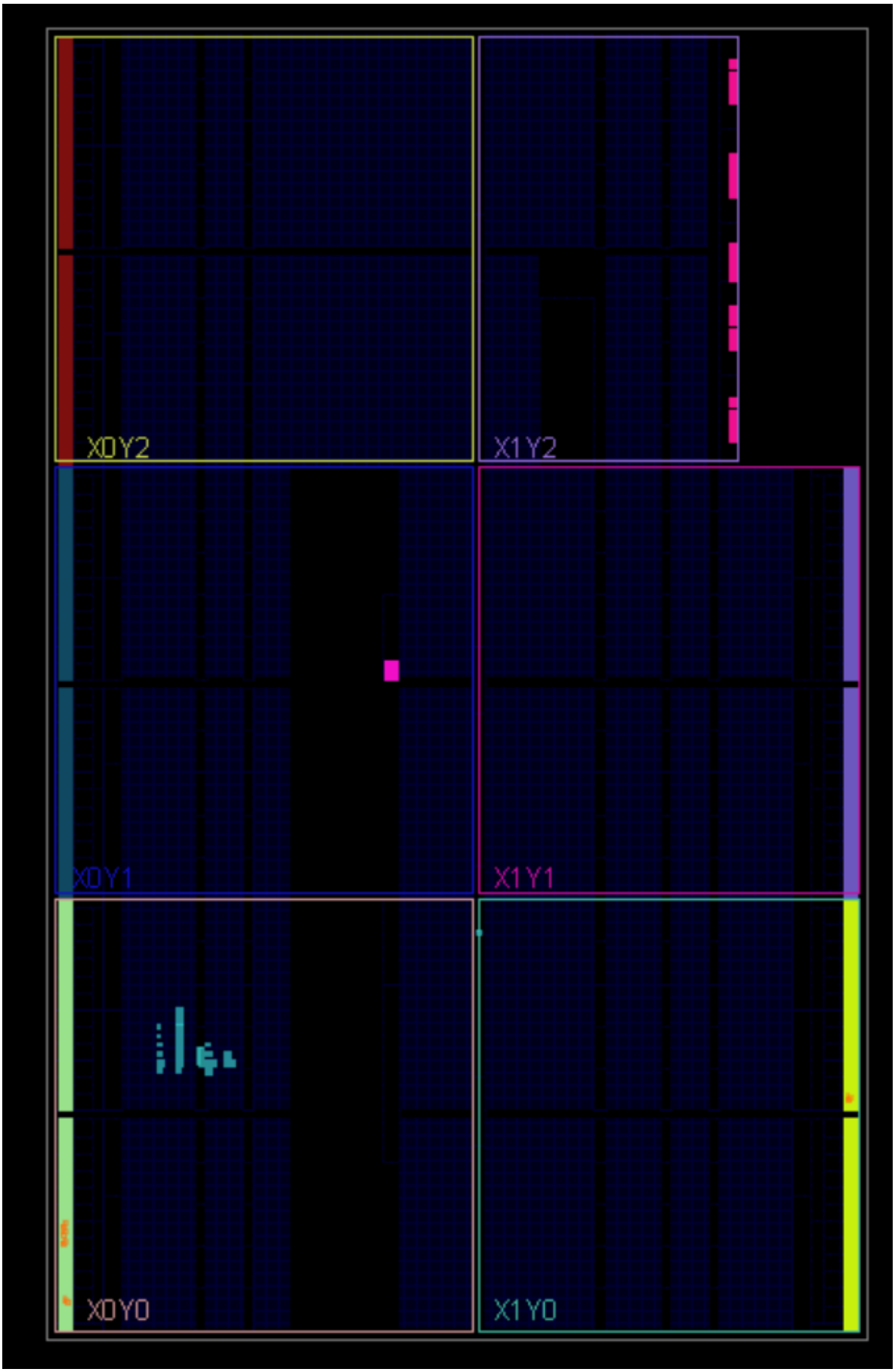
Ref Name	Used	Functional Category
FDRE	65	Flop & Latch
LUT6	36	LUT
LUT3	16	LUT
LUT4	12	LUT
CARRY4	8	CarryLogic
LUT5	5	LUT
IBUF	4	IO
LDCE	3	Flop & Latch
RAMB18E1	1	Block Memory
OBUF	1	IO
MUXF7	1	MuxFx
LUT1	1	LUT
BUFG	1	Clock

One hot encoding hierarchy

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_wrapper	71	72	1	0.5	5	1
u_ram (ram)	2	17	1	0.5	0	0
u_spi_slave (spi_slave)	69	55	0	0	0	0

Sequential encoding



Sequential encoding timing

Design Timing Summary

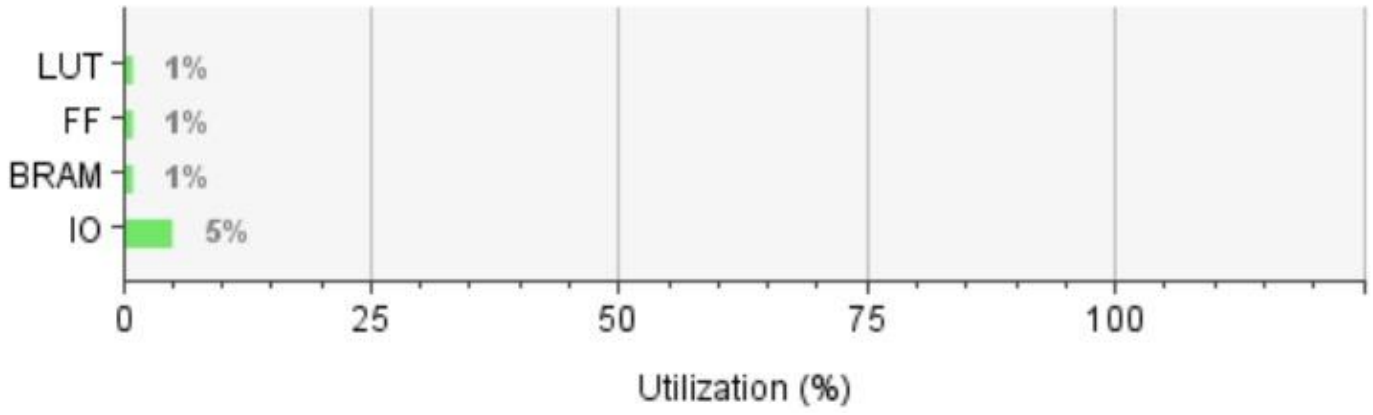
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.559 ns	Worst Hold Slack (WHS):	0.102 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	150	Total Number of Endpoints:	150	Total Number of Endpoints:	68

All user specified timing constraints are met.

Sequential encoding utilization

Summary

Resource	Utilization	Available	Utilization %
LUT	69	20800	0.33
FF	68	41600	0.16
BRAM	0.50	50	1.00
IO	5	106	4.72



Primitives

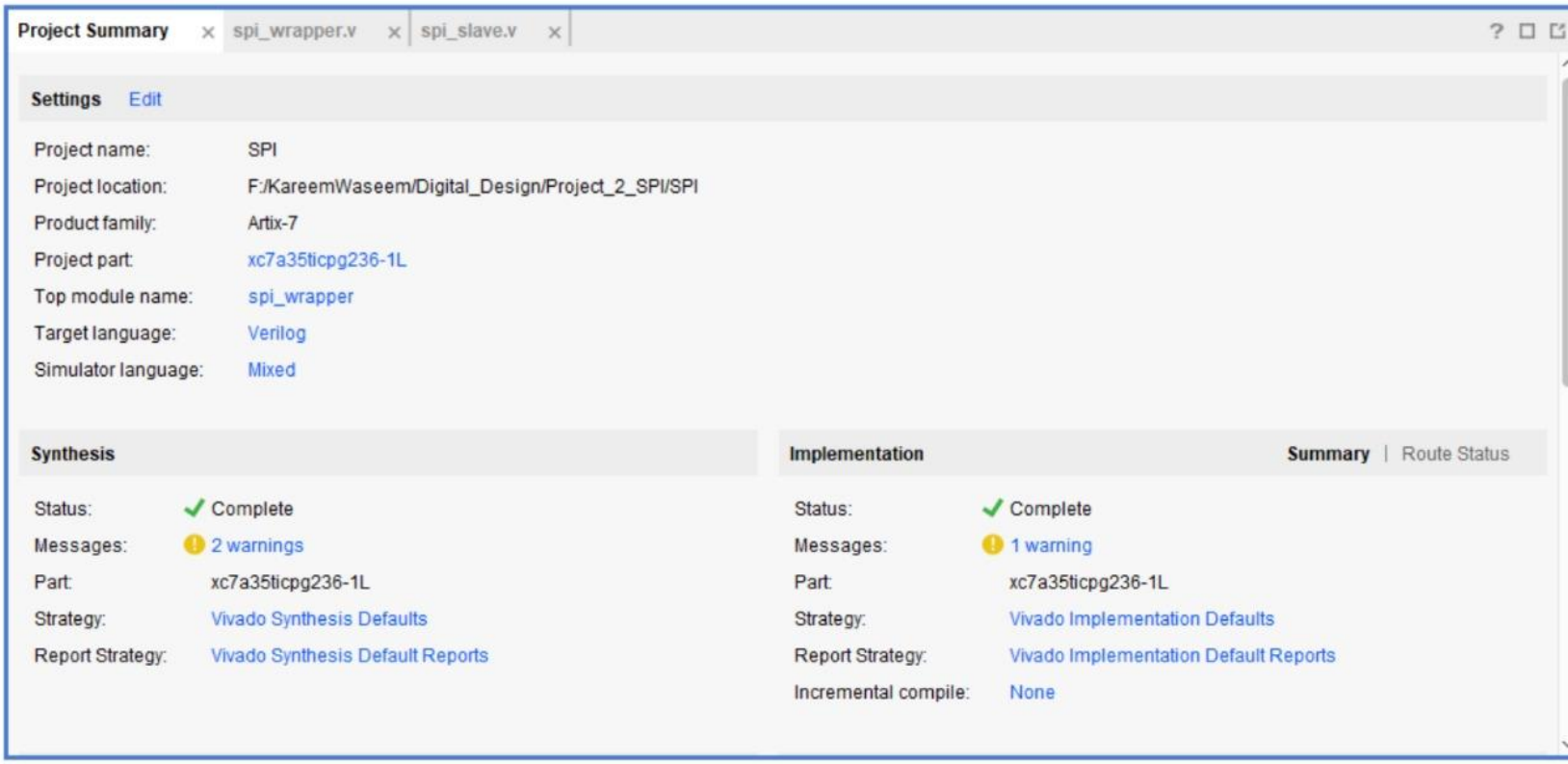
Ref Name	Used	Functional Category
FDRE	65	Flop & Latch
LUT6	36	LUT
LUT3	16	LUT
LUT4	12	LUT
CARRY4	8	CarryLogic
LUT5	5	LUT
IBUF	4	IO
LDCE	3	Flop & Latch
RAMB18E1	1	Block Memory
OBUF	1	IO
MUXF7	1	MuxFx
LUT1	1	LUT
BUFG	1	Clock

sequential encoding hierarchy

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_wrapper	69	68	1	0.5	5	1
u_ram (ram)	2	17	1	0.5	0	0
u_spi_slave (spi_slave)	67	51	0	0	0	0

Messages showing no Errors and critical warning



Important note:

- **We chose one_hot encoding over the other encoding style as it has the biggest worst negative slake so we can use larger frequency.**