

SPI SLAVE WITH SINGLE PORT RAM

Digital Designers: Ziad Kassem & Bassel Mahmoud



Digital Design Using Verilog & FPGA Flow Using Vivado
Diploma - V14
Under Supervision of Eng Kareem Wasseem

Table of Contents

Introduction :	3
Objectives of the Project:	3
SPI Working Principle:	3
How SPI Works?.....	3
Master Sends Clock Signal (SCLK)	3
Slave is Selected (SS/CS).....	3
Data Transfer (MOSI & MISO).....	3
Communication Ends	3
Why Use SPI?	4
Modules Design Code:	5
RAM Design Code:	5
SPI_SLAVE_DESIGN_CODE:	7
TOP_MODULE_DESIGN_CODE:	11
Test Bench:	12
Waveform :	14
Testing Write address functionality :	14
Testing Write data functionality :	14
Randomized reading form RAM :	15
Do File:	16
Constrain File:	16
Encoding:	17
ONE_HOT:.....	17
Elaborated Design:	17
Synthesis:.....	17
Implementation:	19
Bitstream File:.....	20
Gray_Encoding:	20
Elaborated Design:	20
Synthesis:.....	20
Implementation:	22



Bitstream File:	24
Sequential_Encoding:	24
Elaborated Design:	24
Synthesis:	24
Implementation:	26
Bitstream File:	28
Debug Core:	28

Introduction :

Welcome to **SPIder Web Project** – a digital design adventure where we bring **Serial Peripheral Interface (SPI) communication** to life!

In this project, we implement an **SPI Slave module** integrated with a **Single-Port RAM**, designed using **Verilog** and deployed on an **FPGA** using the **Vivado Design Suite**. This system enables seamless data transfer between an SPI Master and memory storage, making it a crucial building block in embedded systems and digital design.

you can check the Design with interactive file directories from our **Github** Repo : [SPI_Slave_With_Single_Port_RAM](#) 

Objectives of the Project:

- Develop an **SPI Slave** that can receive and process commands from an SPI Master.
- Implement a **256-depth Single-Port RAM** to store and retrieve data efficiently.
- Design a **Finite State Machine (FSM)** for SPI communication and control logic.
- Synthesize, implement, and test the design on an **FPGA** using **Vivado**.

SPI Working Principle:

The **Serial Peripheral Interface (SPI)** is a fast and simple way for electronic devices to communicate with each other. It follows a **Master-Slave** setup, where the **Master controls the communication**, and the **Slave responds**.

How SPI Works?

Master Sends Clock Signal (SCLK)

- The Master generates a **clock pulse** to synchronize communication with the Slave.
- The speed of data transfer depends on this clock signal.

Slave is Selected (SS/CS)

- Each Slave has a **Slave Select (SS)** pin.
- The Master pulls **SS LOW (0)** to select a specific Slave and start communication.

Data Transfer (MOSI & MISO)

- **MOSI (Master Out, Slave In):** Master sends data **bit by bit** to the Slave.
- **MISO (Master In, Slave Out):** At the same time, the Slave **sends data back** to the Master.
- This happens in **full-duplex mode** (both can send and receive at the same time).

Communication Ends

- Once data exchange is complete, the Master **pulls SS HIGH (1)**, and the Slave stops responding.



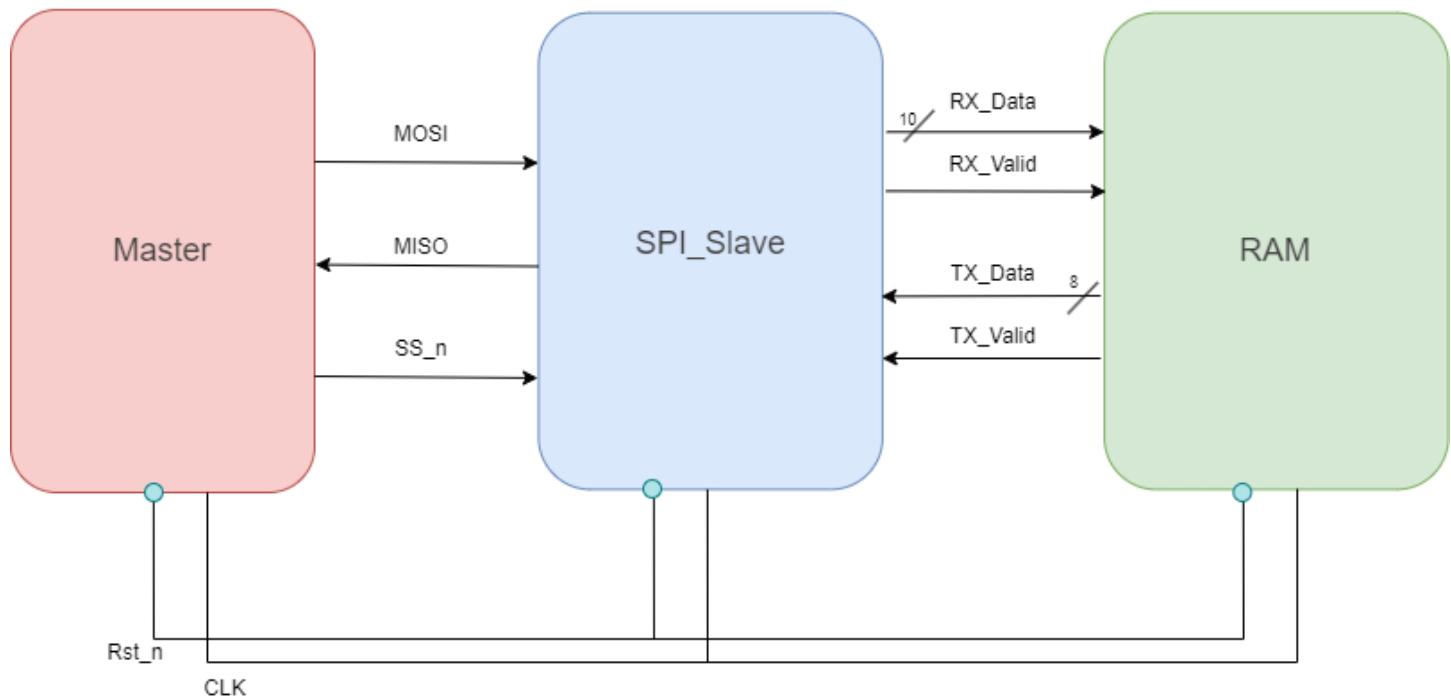
Why Use SPI?

Fast Data Transfer – Works much faster than I²C.

Simultaneous Communication – Master and Slave exchange data at the same time.

Easy to Use – Simple structure with only **4 main wires**.

Reliable – Works efficiently for **short-distance communication**.



Modules Design Code:

RAM Design Code:

```
module RAM #(
    parameter MEM_DEPTH = 256, // Defines the depth of memory (number of addresses available)
    parameter ADDR_SIZE = 8    // Address size (number of bits to address memory locations)
) (
    input [9:0] din,           // 10-bit data input (2 bits for control, 8 bits for address/data)
    input rx_valid, clk, rst_n, // Control signals: receive valid, clock, and active-low reset
    output reg [7:0] dout,     // 8-bit data output (memory read result)
    output reg tx_valid       // Transmit valid signal (indicates data is ready to be transmitted)
);

reg [7:0] memory [MEM_DEPTH-1:0]; // Memory array declaration
reg [ADDR_SIZE-1:0] WR_address_bus; // Write address register
reg [ADDR_SIZE-1:0] RD_address_bus; // Read address register

reg flag_read; // Flag to indicate a pending read operation
reg flag_write; // Flag to indicate a pending write operation

always @(posedge clk) begin
    if (~rst_n) begin // Active-low reset handling
        dout <= 0;
        tx_valid <= 0;
        flag_write <= 0;
        flag_read <= 0;
        RD_address_bus <= 0;
        WR_address_bus <= 0;
    end
    else begin
        tx_valid <= 0; // Ensure tx_valid is cleared at the start of each clock cycle
        case ({din[9], din[8]}) // Extracting the control bits from din
            2'b00: begin
                /*
                 * Case 00: Handle write address request from SPI slave
                 * If no previous write is pending, store the write address
                 */
                if (rx_valid && ~flag_write) begin
                    WR_address_bus <= din[7:0]; // Store write address
                    flag_write <= 1; // Indicate address is ready to be written on
                end
                // TODO: Implement SPI Slave communication to set SS_n = 1 after accepting the address
            end
            2'b01: begin
                /*
                 * Case 01: Handle writing data to memory
                 * Only write if a previous write address has been set
                 */
                if (rx_valid && flag_write) begin

```



```

        flag_write <= 0;
        memory[WR_address_bus] <= din[7:0]; // Write data to the address stored
    end
end
2'b10: begin
/*
 * Case 10: Handle read address request from SPI slave
 * If no previous read is pending, store the read address
 */
if (rx_valid && ~flag_read) begin
    flag_read <= 1;
    RD_address_bus <= din[7:0]; // Store read address
end
end
2'b11: begin
/*
 * Case 11: Handle memory read operation
 * Read data from stored read address and assert tx_valid
 */
if (rx_valid && flag_read) begin
    dout <= memory[RD_address_bus]; // Read data from memory
    flag_read <= 0;
    tx_valid <= 1; // Indicate data is ready to be sent
end
end
endcase
end
end
endmodule

```

Lint Checks

Filter: Type here Waived | Fixed | Pending | Uninspected | Bug | Verified | Total : 2 Selected : 0

Severity	Status	Check	Alias	Message	Module	Category	State
Info	Waived	always_signal_assign_large		Always block has more signal assignments than the specified limit. Total count 7, Specified limit 5, Module ... RAM	Rtl Design Style	open	
Info	Waived	multi_ports_in_single_line		Multiple ports are declared in one line. Module RAM, File D:/Kareem_Wassem/DESIGN/Projects/SPI_Slav... RAM	Rtl Design Style	open	



SPI_SLAVE_DESIGN_CODE:

```
module SPI_Slave #(
    //States
    parameter IDLE      ='b000 ,
    parameter CHK_CMD   ='b001 ,
    parameter WRITE     ='b010 ,
    parameter READ_ADD  ='b011 ,
    parameter READ_DATA ='b100
) (
    //Control Inputs
    input SS_n,CLK,rst_n,
    //TX Inputs
    input tx_valid,
    input [7:0] tx_data,
    //Protocol Lines
    input MOSI,
    output reg MISO,
    //RX Outputs
    output reg rx_valid,
    output reg [9:0] rx_data
);
//////////////////////////////////////////////////////////////// Variables /////////////////////////////////
// State Variables
reg [2:0] cs,ns;

//Global Variables
reg is_address_received;
reg data_received;
//Tasks Variables:
// 1)SER_TO_PAR
reg [3:0] cycle_counter_SER_TO_PAR;
// 2)PAR_TO_SER
reg [3:0] cycle_counter_PAR_TO_SER;
reg [7:0]Parallel_data_out;

//////////////////////////////////////////////////////////////// FSM Algorithm ///////////////////////////////
//State Memory
always @(posedge CLK) begin
    if(~rst_n)begin
        cs <= IDLE ;
    end
    else begin
        cs <= ns ;
    end
end
```

```

//Next State Logic
always @(*) begin
    case (cs)
        IDLE: begin
            if (SS_n) begin
                ns = IDLE ;
            end
            else begin
                ns = CHK_CMD ;
            end
        end

        CHK_CMD: begin
            if (SS_n) begin
                ns = IDLE ;
            end
            else if (MOSI == 0) begin
                ns = WRITE ;
            end
            else if (~is_address_received) begin
                ns = READ_ADD ;
            end
            else begin
                ns = READ_DATA ;
            end
        end

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

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

        READ_DATA: begin
            if (SS_n) begin
                ns = IDLE ;
            end else begin
                ns = READ_DATA ;
            end
        end
    endcase
end

```

```

        end
    default: ns = IDLE ;
endcase
end

//Output Logic
always @(posedge CLK) begin
    if(~rst_n)begin
        MISO      <= 0 ;
        rx_valid  <= 0 ;
        rx_data   <= 0 ;
        //some internal signals
        is_address_received <= 0 ;
        Parallel_data_out   <= 0 ;
        cycle_counter_PAR_TO_SER <= 0 ;
        cycle_counter_SER_TO_PAR <= 0 ;
    end
    else begin
        if (SS_n) begin
            communicationEnded();
        end else if (cs == CHK_CMD) begin
            SER_TO_PAR();
        end else if (cs == WRITE) begin
            SER_TO_PAR();
        end else if (cs == READ_ADD) begin
            SER_TO_PAR();
        end else if (cs == READ_DATA) begin
            READ_DATA_TASK();
        end
    end
end
end

/////////////////////////////// Tasks /////////////////////////////////
task SER_TO_PAR();
begin
    if (cycle_counter_SER_TO_PAR < 10) begin
        rx_data <= {rx_data [8:0] , MOSI} ;
        cycle_counter_SER_TO_PAR <= cycle_counter_SER_TO_PAR + 1 ;
    end
    if (cycle_counter_SER_TO_PAR == 9) begin
        rx_valid <= 1;
        if (cs == READ_ADD) is_address_received <= 1;
        else is_address_received <= 0;
    end
end
endtask

task PAR_TO_SER();

```



```

begin
    if (cycle_counter_PAR_TO_SER < 8) begin
        MISO <= Parallel_data_out[7];
        Parallel_data_out <= Parallel_data_out<<1;
        cycle_counter_PAR_TO_SER <= cycle_counter_PAR_TO_SER + 1 ;
    end
    if(cycle_counter_PAR_TO_SER == 7) begin
        data_received <= 0 ;
        is_address_received <= 0 ;
    end
end
endtask

task READ_DATA_TASK();
begin
    if(data_received)begin
        PAR_TO_SER();
    end
    else if (tx_valid) begin
        rx_valid <= 0;
        Parallel_data_out <= tx_data;
        data_received <= 1 ;
    end
    else begin
        SER_TO_PAR();
    end
end
endtask

// Handling any time SS_n = 1 (at the end of communication or accidentally)
task communication_ended();begin
    MISO          <= 0 ;
    cycle_counter_PAR_TO_SER <= 0 ;
    cycle_counter_SER_TO_PAR <= 0 ;
    rx_valid      <= 0 ;
    Parallel_data_out <= 0 ;
end
endtask

```

endmodule

Severity	Status	Check	Alias	Message	/	Module	Category	Stat
Info	?	task_sets_global_var		Task sets a global variable. Variable rx_data, Set at: Module SPI_Slave, File D:/Kareem_Wassem/DESIG...	SPI_Slave	Rtl Design Style open		
Info	?	task_sets_global_var		Task sets a global variable. Variable cycle_counter_SER_TO_PAR, Set at: Module SPI_Slave, File D:/Kar...	SPI_Slave	Rtl Design Style open		
Info	?	task_sets_global_var		Task sets a global variable. Variable rx_valid, Set at: Module SPI_Slave, File D:/Kareem_Wasse...	SPI_Slave	Rtl Design Style open		
Info	?	task_sets_global_var		Task sets a global variable. Variable is_address_received, Set at: Module SPI_Slave, File D:/Kareem_Was...	SPI_Slave	Rtl Design Style open		
Info	?	task_sets_global_var		Task sets a global variable. Variable MISO, Set at: Module SPI_Slave, File D:/Kareem_Wasse...	SPI_Slave	Rtl Design Style open		



TOP_MODULE DESIGN CODE:

```
module SPI#(
    parameter MEM_DEPTH = 256,
    parameter ADDR_SIZE = 8
) (
    input MOSI ,SS_n, CLK , rst_n,
    output MISO
);
    wire [7:0] tx_data ;
    wire [9:0] rx_data ;
    wire rx_valid, tx_valid;

    SPI_Slave Slave_handler (
        .SS_n(SS_n),
        .CLK(CLK),
        .rst_n(rst_n),
        .tx_valid(tx_valid),
        .tx_data(tx_data),
        .MOSI(MOSI),
        .MISO(MISO),
        .rx_valid(rx_valid),
        .rx_data(rx_data)
    );
    RAM #((
        .MEM_DEPTH(MEM_DEPTH),
        .ADDR_SIZE(ADDR_SIZE)
    ))RAM1 (
        .clk(CLK),
        .rst_n(rst_n),
        .din(rx_data),
        .rx_valid(rx_valid),
        .dout(tx_data),
        .tx_valid(tx_valid)
    );
endmodule
```



Test Bench:

```
module SPI_tb();
    bit MOSI_tb ,SS_n, CLK , rst_n;
    logic MISO_tb;
    SPI DUT (
        .SS_n(SS_n),
        .CLK(CLK),
        .rst_n(rst_n),
        .MOSI(MOSI_tb),
        .MISO(MISO_tb)
    );
    initial begin
        CLK = 0;
        forever #1 CLK = ~CLK;
    end
    logic [7:0] address,data;
    initial begin
        SS_n = 1;
        MOSI_tb = 0;
        // Reset the system
        reset();
        $readmemb("mem.dat",DUT.RAM1.memory);
    //////////////////////////////// Normal Full Scenario ///////////////////////
    // Test Case 1: Write Address (WRITE)
    SS_n = 0;
    @(negedge CLK);
    address = 8'b1111_0000;
    receive_from_master(10'b00_1111_0000); // Command 00 (WRITE) + 8-bit data
    SS_n = 1;
    @(negedge CLK);
    $stop;
    // Test Case 2: Write Data (WRITE)
    SS_n = 0;
    @(negedge CLK);
    address = 8'b1111_0000;
    receive_from_master(10'b01_1111_0000); // Command 01 (WRITE) + 8-bit data
    SS_n = 1;
    @(negedge CLK);
    $stop;
    // Test Case 3: Read Address (READ_ADD)
    SS_n = 0;
    @(negedge CLK);
    address = 8'b1111_0000;
    receive_from_master(10'b10_1111_0000); // Command 10 (READ_ADD) + 8-bit address
    SS_n = 1;
    @(negedge CLK);
```



```

$stop;
// Test Case 4: Read Data (READ_DATA)
SS_n = 0;
@(negedge CLK);
address = 8'b0000_0000;
receive_from_master(10'b11_0000_0000); // Command 11 (READ_DATA) + dummy data
@(negedge CLK); // wait for tx data from RAM
repeat(9) @(negedge CLK); // Wait for 8-bit serialization
SS_n = 1;
@(negedge CLK);
$display("Normal Scenerio Finished ,Now starting Just Reading with self checking");
/////////////////////////////// Just Reading with self checking /////////////////////
repeat(10)begin
    //Read Address (READ_ADD)
    SS_n = 0;
    @(negedge CLK);
    address = $random;
    receive_from_master({2'b10,address}); // Command 10 (READ_ADD) + 8-bit address
    SS_n = 1;
    @(negedge CLK);

    // Test Case 4: Read Data (READ_DATA)
    SS_n = 0;
    @(negedge CLK);
    receive_from_master(10'b11_0000_0000); // Command 11 (READ_DATA) + dummy data
    repeat(2)@(negedge CLK); // wait for tx_valid trigger+ another edge spi receive data
    if (DUT.RAM1.memory [address] != DUT.Slave_handler.Parallel_data_out) begin
        $display("Error in just reading with self checking at time=%d",$time);
        $stop;
    end
    repeat(9) @(negedge CLK); // Wait for 8-bit serialization
    SS_n = 1;
    @(negedge CLK);
end
$display("Just Reading with self checking Finished with no errors ,Now starting Just Writing with
self checking");
/////////////////////////////// Just Writing with self checking /////////////////////
repeat(10)begin
    //Read Address (Write_ADD)
    SS_n = 0;
    @(negedge CLK);
    address = $random;
    data = $random ;
    receive_from_master({2'b00,address}); // Command 10 (READ_ADD) + 8-bit address
    SS_n = 1;
    @(negedge CLK);

    // Test Case 4: Read Data (READ_DATA)
    SS_n = 0;

```



```

@(negedge CLK);
receive_from_master({2'b01,data}); // Command 11 (READ_DATA) + dummy data
@(negedge CLK); // wait for data to be written
if (DUT.RAM1.memory [address] != data ) begin
    $display("Error in just Writing with self checking at time =%d",$time);
    $stop;
end
SS_n = 1;
@(negedge CLK);
end
$display("Just Writing with self checking Finished with no errors");
$stop;
end

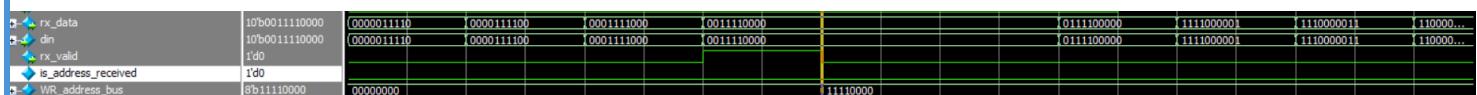
// Reset Task
task reset();
    rst_n = 1;
    @(negedge CLK);
    rst_n = 0;
    @(negedge CLK);
    rst_n = 1;
endtask
task receive_from_master(logic [9:0] Parallel_data_in_tb);
    for (int i = 9; i >= 0; i--) begin
        MOSI_tb = Parallel_data_in_tb[i];
        @(negedge CLK);
    end
endtask
endmodule

```

Waveform :

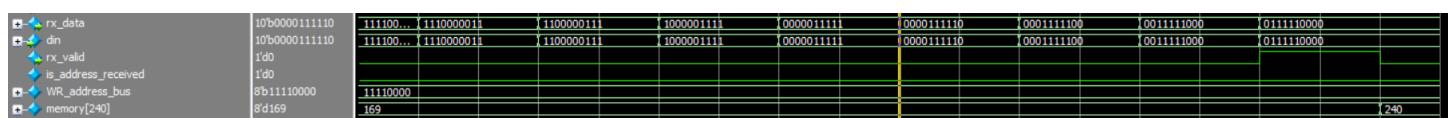
Testing Write address functionality :

Here we can see that the address has successfully been sent to the ram and the RAM internal signal WR_address_bus received it successfully



Testing Write data functionality :

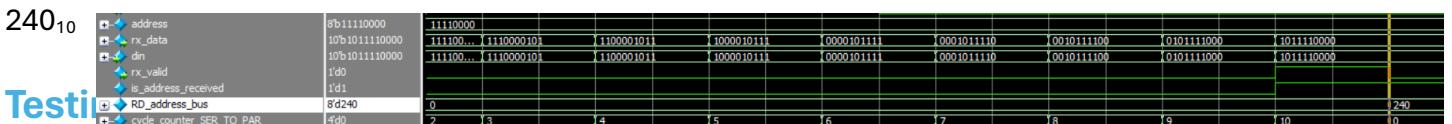
Data has been successfully written inside RAM with address 240_{10} and the data written is 240_{10}



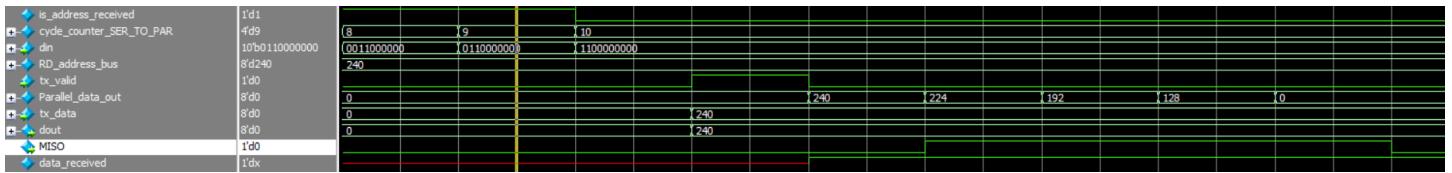
Testinng Read address functionality:



Data has been successfully sent to RAM to be saved in the internal signal RD_Address_bus with value 240₁₀

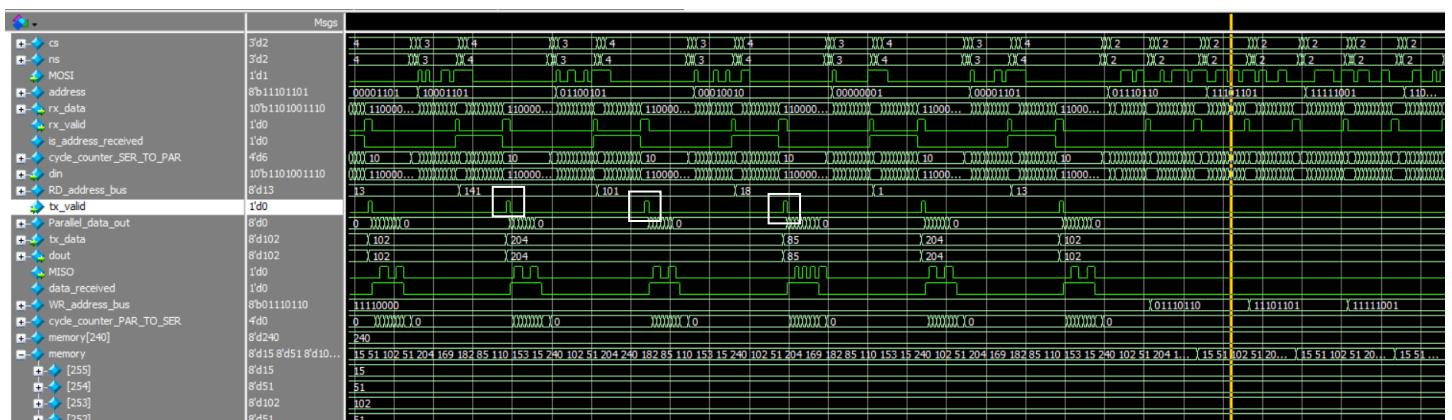


Data has been successfully read from address 240₁₀ and tx_valid is triggered and MISO begin to serialize data to the master



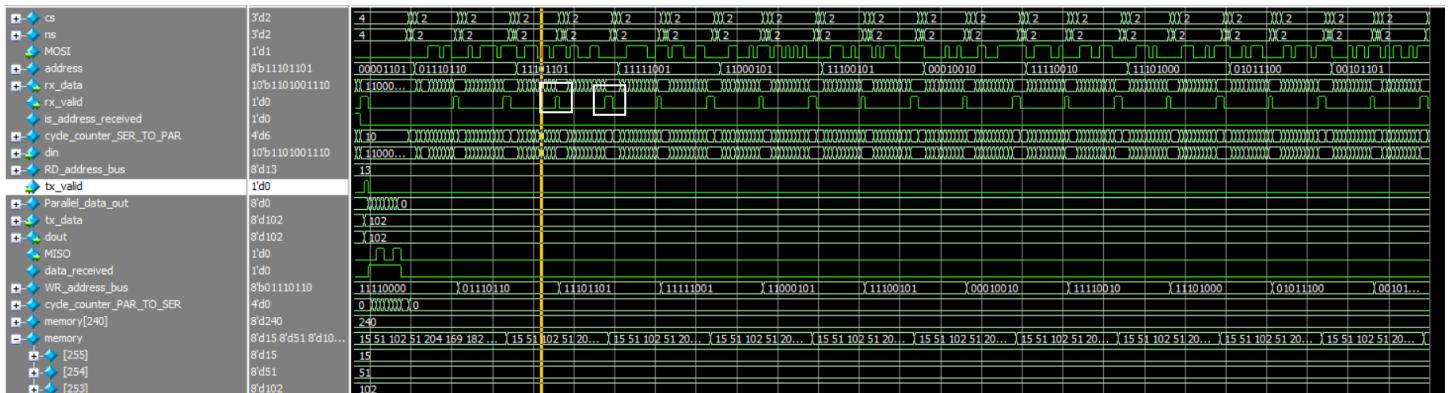
Randomized reading from RAM :

```
PSIM > run -continue
# Normal Scenario Finished ,Now starting Just Reading with self checking
# Just Reading with self checking Finished with no errors ,Now starting Just Writing with self checking
# Just Writing with self checking Finished with no errors
# ** Note: $stop : Main_module_tb.sv(107)
# Time: 1320 ns Iteration: 1 Instance: /SPI_tb
```



As we can see from the White boxes that the tx_valid is triggered to indicate that the message is sent successfully to the SPI_Slave form the RAM .

Random writing to RAM



As we can see from the White boxes that the rx_valid is triggered to indicate that the message is sent successfully from SPI_Slave to the RAM .



Do File:

```
vlib work
vlog Main_module.v Main_module_tb.sv SPI.v RAM.v
vsim -voptargs+=acc work.SPI_tb
add wave -position insertpoint \
sim:/SPI_tb/DUT/Slave_handler/CLK \
sim:/SPI_tb/DUT/Slave_handler/rst_n \
sim:/SPI_tb/DUT/Slave_handler/MOSI \
sim:/SPI_tb/DUT/Slave_handler/SS_n \
sim:/SPI_tb/DUT/Slave_handler/MISO \
sim:/SPI_tb/DUT/Slave_handler/tx_data \
sim:/SPI_tb/DUT/Slave_handler/rx_data \
sim:/SPI_tb/DUT/Slave_handler/tx_valid \
sim:/SPI_tb/DUT/Slave_handler/rx_valid \
run -all
```

Constrain File:

```
## Clock signal
set_property -dict { PACKAGE_PIN W5    IOSTANDARD LVCMS3 } [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 LVCMS3 } [get_ports {rst_n}]
set_property -dict { PACKAGE_PIN V16    IOSTANDARD LVCMS3 } [get_ports {SS_n}]
set_property -dict { PACKAGE_PIN W16    IOSTANDARD LVCMS3 } [get_ports {MOSI}]

## LEDs
set_property -dict { PACKAGE_PIN U16    IOSTANDARD LVCMS3 } [get_ports {MISO}]

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

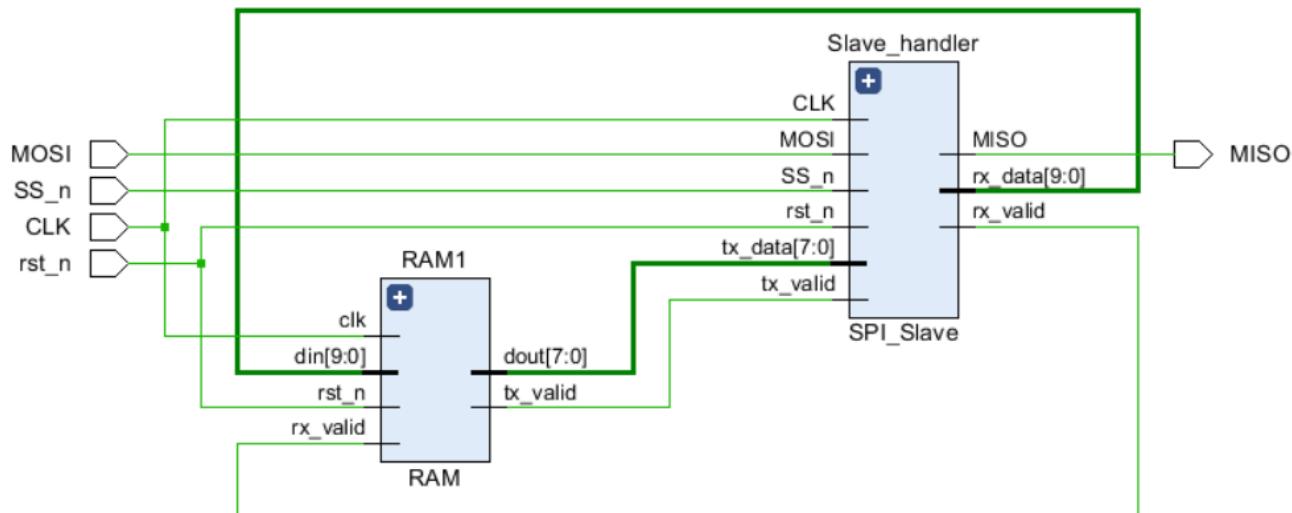
## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```



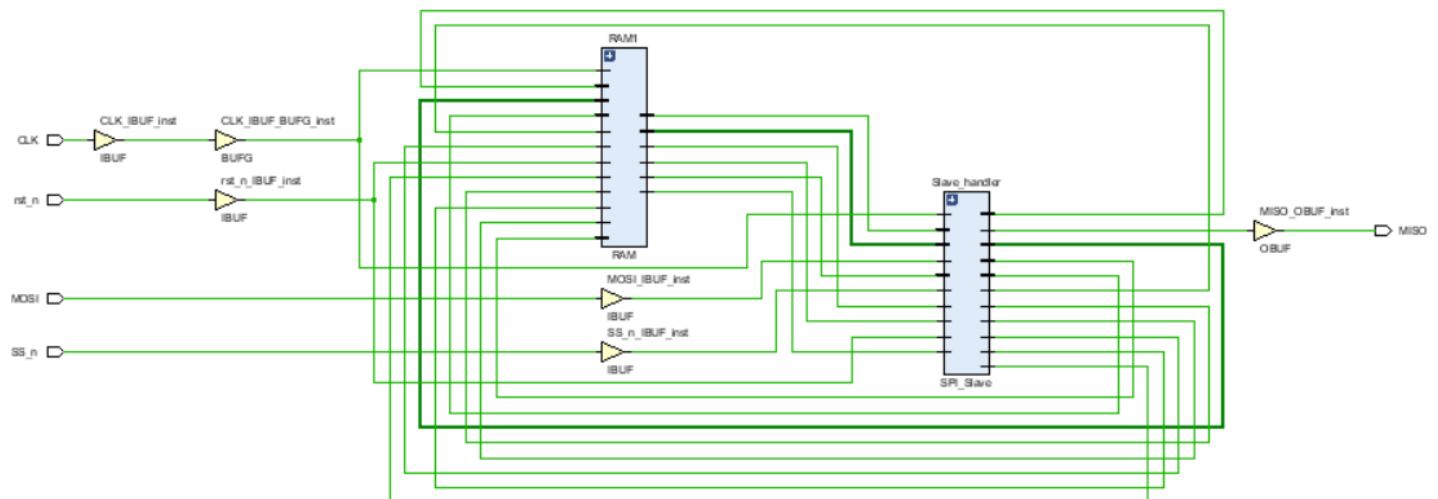
Encoding:

ONE_HOT:

Elaborated Design:



Synthesis:

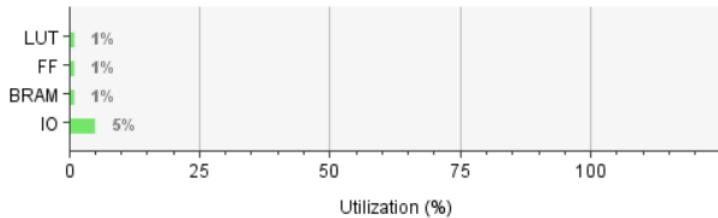


State	New Encoding	Previous Encoding
IDLE	00001	000
CHK_CMD	00010	001
WRITE	00100	010
READ_ADD	01000	011
READ_DATA	10000	100



Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	37	20800	0.18
FF	54	41600	0.13
BRAM	0.50	50	1.00
IO	5	106	4.72



Timing Report:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.471 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: 124	Total Number of Endpoints: 124	Total Number of Endpoints: 57

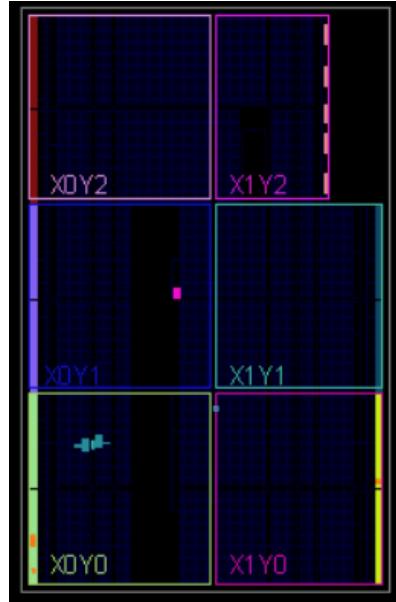
Netlist:

```
// Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.  
// -----  
// Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018  
// Date : Thu Mar 13 10:09:09 2025  
// Host : Ziad-Kassem running 64-bit major release (build 9200)  
// Command : write_verilog  
D:/Kareem_Wasseem/DESIGN/Projects/SPI_Slave_With_Single_Port_RAM/ONE_HOT_NETLIST.v  
// Design : SPI  
// Purpose : This is a Verilog netlist of the current design or from a specific cell of  
the design. The output is an  
// IEEE 1364-2001 compliant Verilog HDL file that contains netlist information  
obtained from the input  
// design files.  
// Device : xc7a35ticpg236-1L  
// -----
```

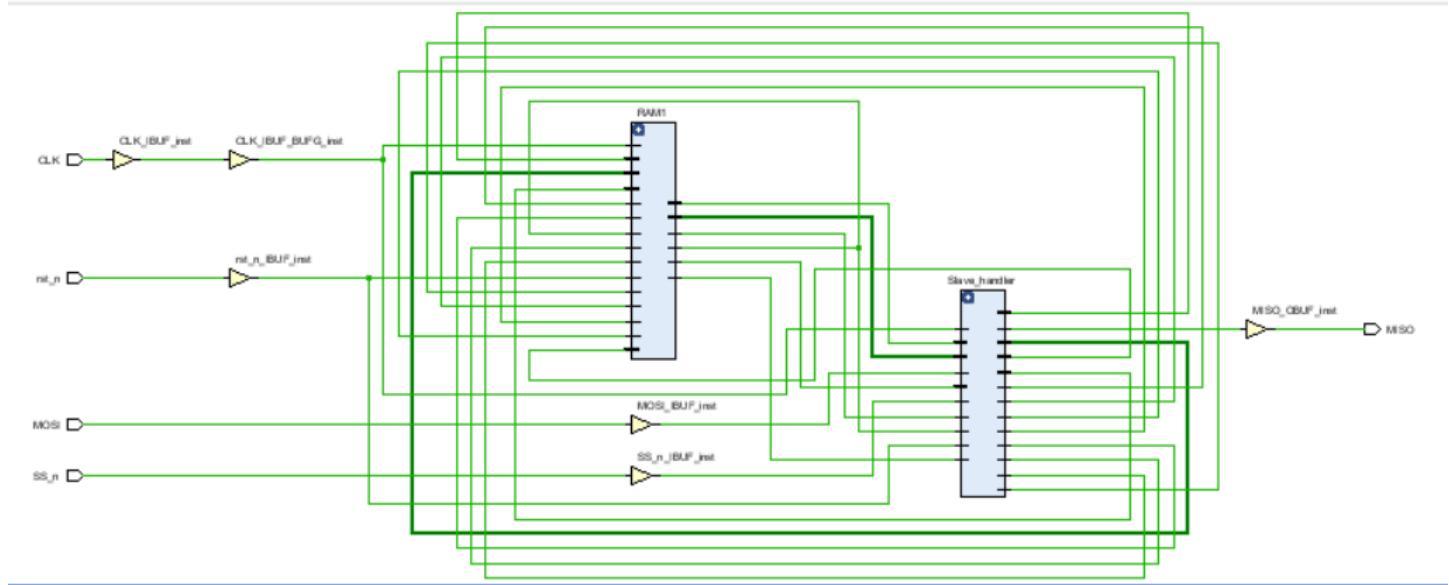
Full file is found including the files and named: ONE_HOT_NETLIST.v

Implementation:

FPGA:



Schematic:



Timing:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.327 ns	Worst Hold Slack (WHS): 0.052 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: 124	Total Number of Endpoints: 124	Total Number of Endpoints: 57

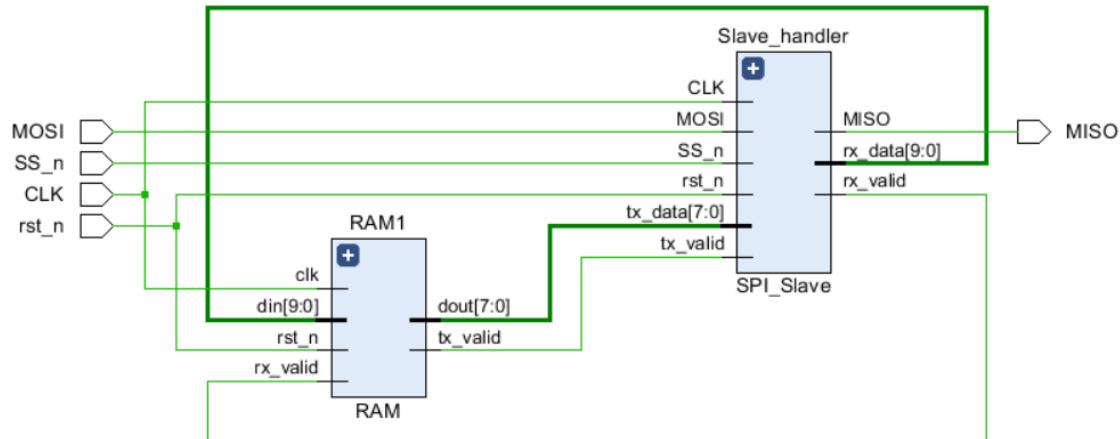
All user specified timing constraints are met.

Bitstream File:

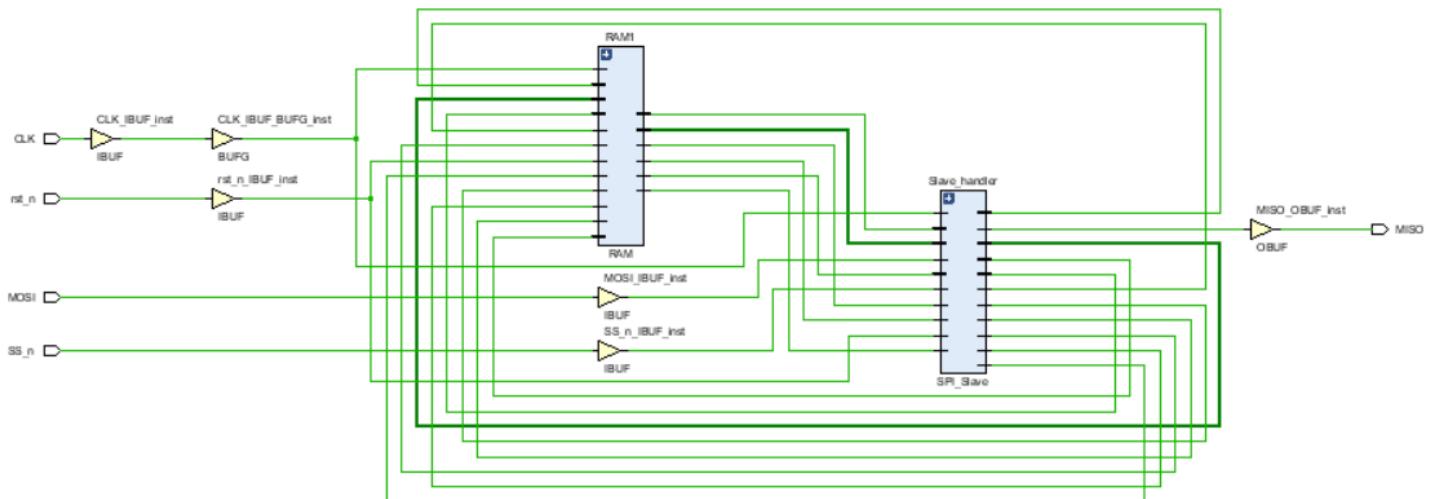
	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000000	0a	67	0a	0c	43	6f	6d	6d	61	6e	64	3a	20	25	73	0a
00000010	10	04	1a	04	31	38	37	30	2a	09	70	6c	61	6e	41	68
00000020	65	61	64	32	32	0a	1e	6f	70	65	6e	5f	63	68	65	63
00000030	6b	70	6f	69	6e	74	20	53	50	49	5f	72	6f	75	74	65
00000040	64	2e	64	63	70	32	07	64	65	66	61	75	6c	74	3a	07
00000050	64	65	66	61	75	6c	74	5a	07	31	32	2d	32	38	36	36
00000060	68	00	70	04	78	04	90	01	00	10	02	0a	5e	0a	12	0a
00000070	53	74	61	72	74	69	6e	67	20	25	73	20	54	61	73	6b
00000080	0a	10	04	1a	03	31	30	33	2a	0b	63	6f	6e	73	74	72
00000090	61	69	6e	74	73	32	23	0a	0f	6f	70	65	6e	5f	63	68
000000a0	65	63	6b	70	6f	69	6e	74	32	07	64	65	66	61	75	6c
000000b0	74	3a	07	64	65	66	61	75	6c	74	5a	06	31	38	2d	31
000000c0	30	33	68	00	70	04	78	04	90	01	00	10	02	0a	92	01
000000d0	0a	04	0a	25	73	0a	10	04	2a	0b	63	6f	6e	73	74	72
000000e0	61	69	6e	74	73	32	72	0a	5e	54	69	6d	65	20	28	73

Gray Encoding:

Elaborated Design:

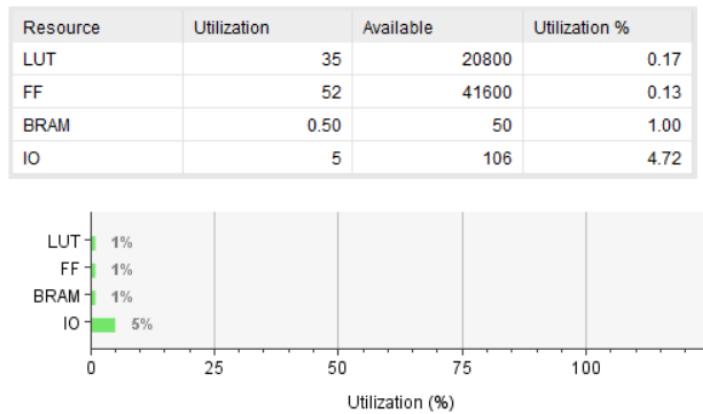


Synthesis:



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

Utilization:



Timing:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.471 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: 122	Total Number of Endpoints: 122	Total Number of Endpoints: 55

All user specified timing constraints are met.

A screenshot of a software interface titled "Messages". The window shows a toolbar with icons for search, filter, and message types (Warning, Info, Status). Below the toolbar, a section titled "Synthesis (1 warning)" is expanded, displaying a single warning message: "[Constraints 18-5210] No constraint will be written out.".



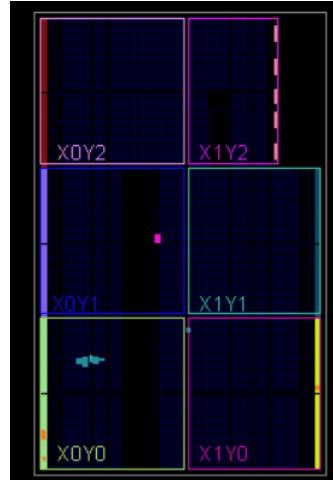
NETLIST Report:

```
// Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.  
// -----  
// Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018  
// Date : Thu Mar 13 10:45:28 2025  
// Host : Ziad-Kassem running 64-bit major release (build 9200)  
// Command : write_verilog  
D:/Kareem_Wasseem/DESIGN/Projects/SPI_Slave_With_Single_Port_RAM/GRAY_NETLIST.v  
// Design : SPI  
// Purpose : This is a Verilog netlist of the current design or from a specific cell of  
the design. The output is an  
// IEEE 1364-2001 compliant Verilog HDL file that contains netlist information  
obtained from the input  
// design files.  
// Device : xc7a35ticpg236-1L  
// -----
```

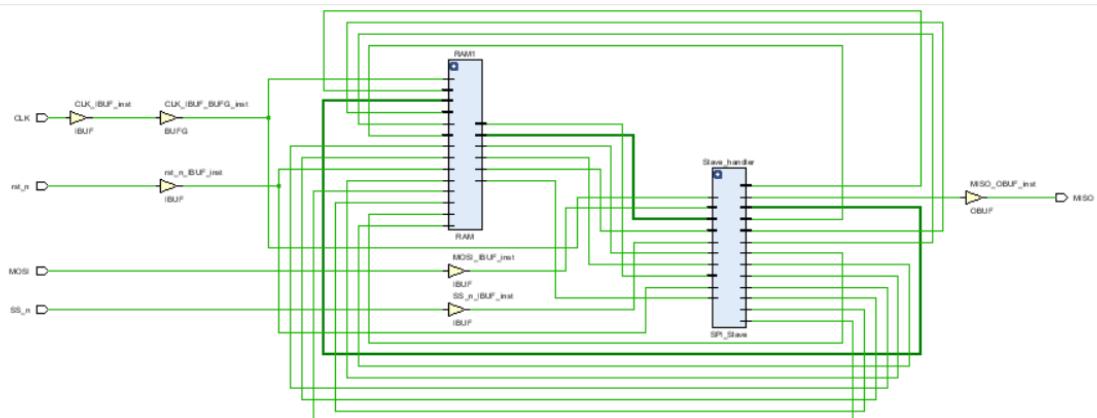
Full file is found including the files and named: GRAY_NETLIST.v

Implementation:

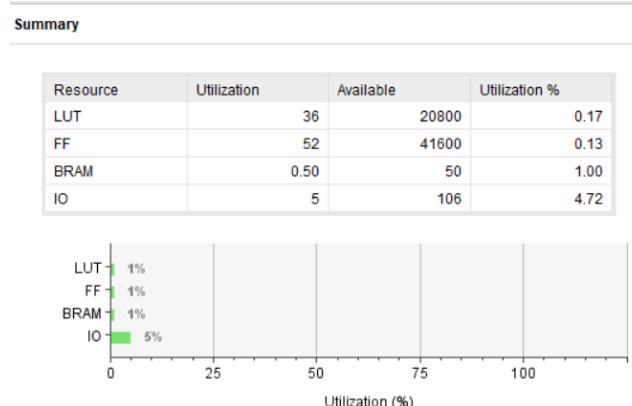
FPGA:



Schematic:



Utilization:

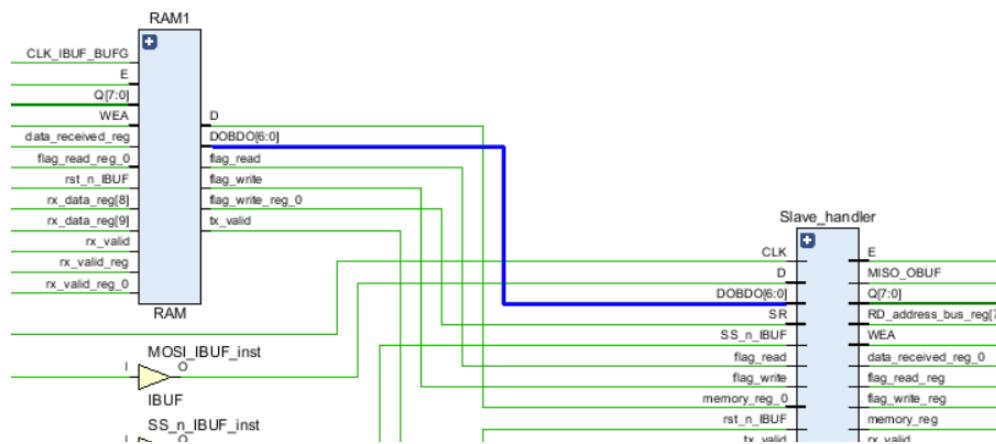


Timing:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.299 ns	Worst Hold Slack (WHS): 0.085 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: 122	Total Number of Endpoints: 122	Total Number of Endpoints: 55

All user specified timing constraints are met.

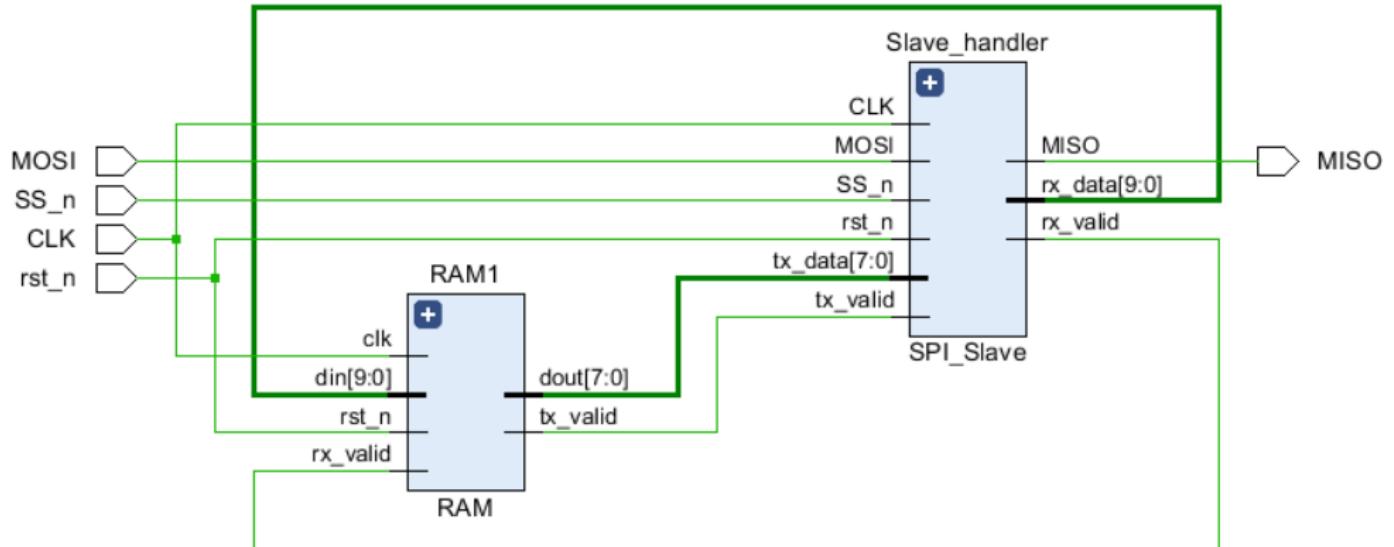


Bitstream File:

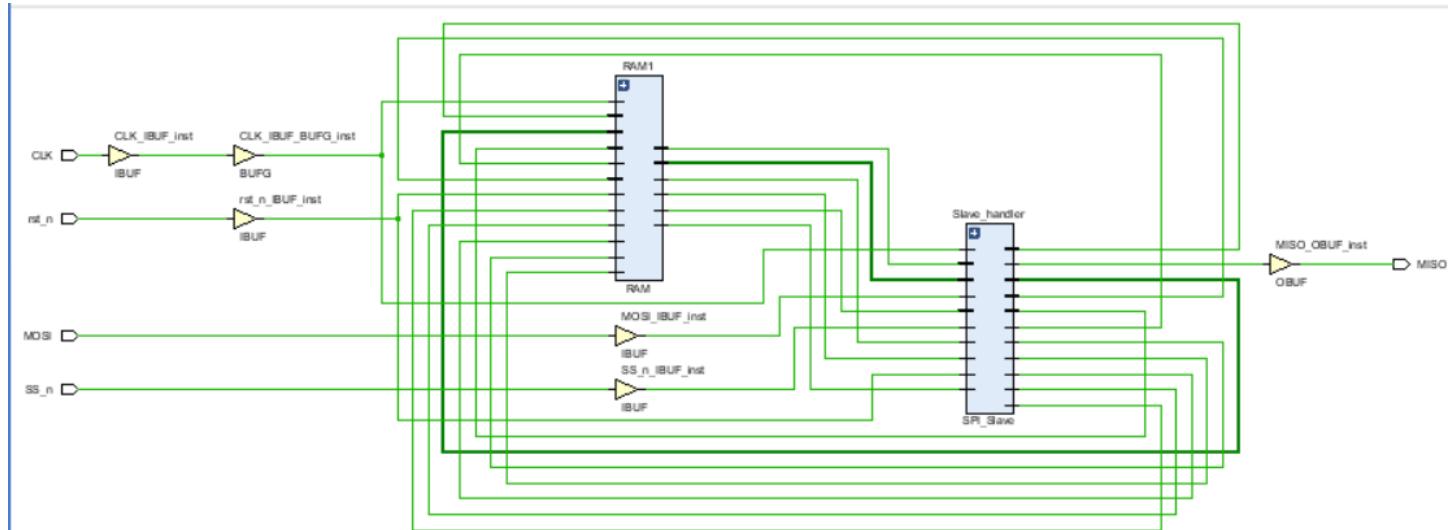
	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000000	0a	67	0a	0c	43	6f	6d	6d	61	6e	64	3a	20	25	73	0a
00000010	10	04	1a	04	31	38	37	30	2a	09	70	6c	61	6e	41	68
00000020	65	61	64	32	32	0a	1e	6f	70	65	6e	5f	63	68	65	63
00000030	6b	70	6f	69	6e	74	20	53	50	49	5f	72	6f	75	74	65
00000040	64	2e	64	63	70	32	07	64	65	66	61	75	6c	74	3a	07
00000050	64	65	66	61	75	6c	74	5a	07	31	32	2d	32	38	36	36
00000060	68	00	70	04	78	04	90	01	00	10	02	0a	5e	0a	12	0a
00000070	53	74	61	72	74	69	6e	67	20	25	73	20	54	61	73	6b
00000080	0a	10	04	1a	03	31	30	33	2a	0b	63	6f	6e	73	74	72
00000090	61	69	6e	74	73	32	23	0a	0f	6f	70	65	6e	5f	63	68
000000a0	65	63	6b	70	6f	69	6e	74	32	07	64	65	66	61	75	6c
000000b0	74	3a	07	64	65	66	61	75	6c	74	5a	06	31	38	2d	31
000000c0	30	33	68	00	70	04	78	04	90	01	00	10	02	0a	92	01
000000d0	0a	04	0a	25	73	0a	10	04	2a	0b	63	6f	6e	73	74	72
000000e0	61	69	6e	74	73	32	72	0a	5e	54	69	6d	65	20	28	73

Sequential_Encoding:

Elaborated Design:



Synthesis:

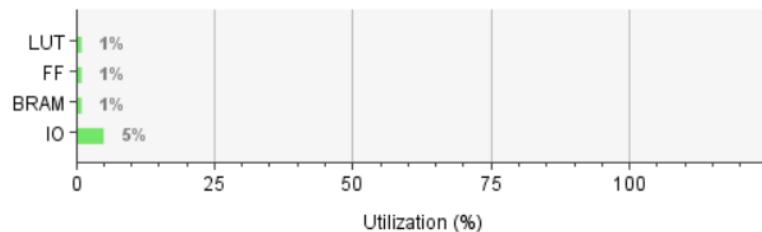


State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	001	001
WRITE	010	010
READ_ADD	011	011
READ_DATA	100	100

Utilization:

Summary

Resource	Utilization	Available	Utilization %
LUT	37	20800	0.18
FF	52	41600	0.13
BRAM	0.50	50	1.00
IO	5	106	4.72

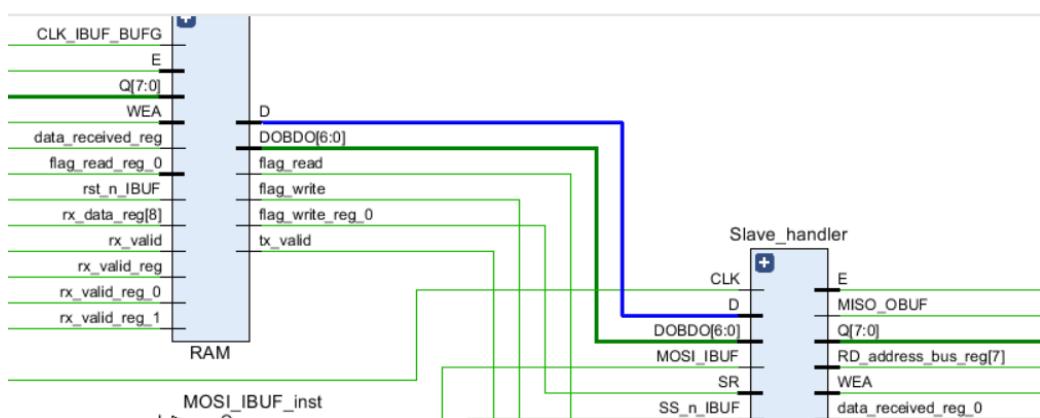


Timing:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.471 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: 122	Total Number of Endpoints: 122	Total Number of Endpoints: 55

All user specified timing constraints are met.



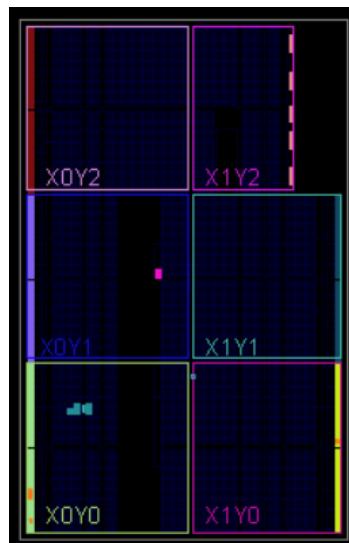
Netlist Report:

```
// Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.  
// -----  
// Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018  
// Date       : Thu Mar 13 11:04:24 2025  
// Host       : Ziad-Kassem running 64-bit major release (build 9200)  
// Command    : write_verilog  
D:/Kareem_Wasseem/DESIGN/Projects/SPI_Slave_With_Single_Port_RAM/PSEQUENTIAL_NETLIST.v  
// Design     : SPI  
// Purpose    : This is a Verilog netlist of the current design or from a specific cell of  
the design. The output is an  
//                 IEEE 1364-2001 compliant Verilog HDL file that contains netlist information  
obtained from the input  
//                 design files.  
// Device     : xc7a35ticpg236-1L  
// -----
```

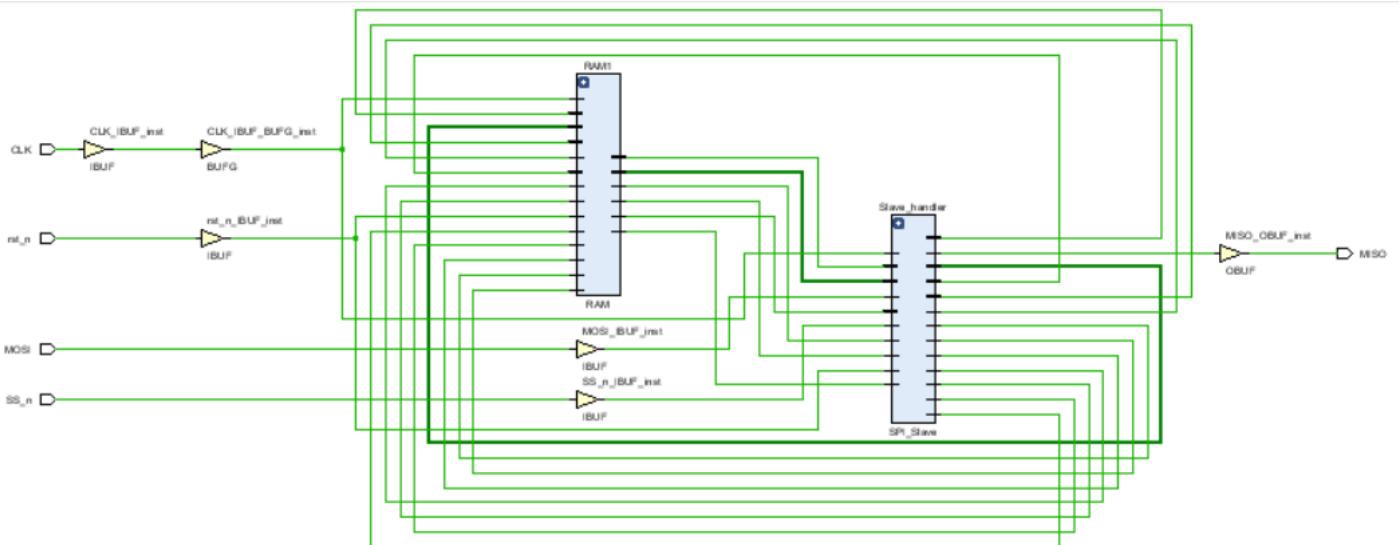
Full file is found including the files and named: PSEQUENTIAL_NETLIST.v

Implementation:

FPGA:



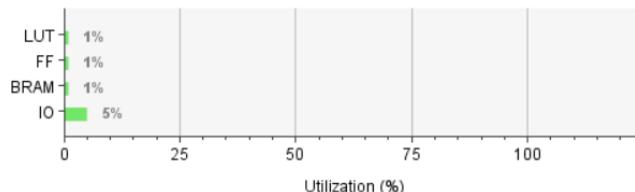
Schematic:



Utilization:

Summary

Resource	Utilization	Available	Utilization %
LUT	38	20800	0.18
FF	52	41600	0.13
BRAM	0.50	50	1.00
IO	5	106	4.72



Timing:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.149 ns	Worst Hold Slack (WHS): 0.044 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 (TPWWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 122	Total Number of Endpoints: 122	Total Number of Endpoints: 55

All user specified timing constraints are met.

Bitstream File:

00000000	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000000	0a	67	0a	0c	43	6f	6d	6d	61	6e	64	3a	20	25	73	0a
00000010	10	04	1a	04	31	38	37	30	2a	09	70	6c	61	6e	41	68
00000020	65	61	64	32	32	0a	1e	6f	70	65	6e	5f	63	68	65	63
00000030	6b	70	6f	69	6e	74	20	53	50	49	5f	72	6f	75	74	65
00000040	64	2e	64	63	70	32	07	64	65	66	61	75	6c	74	3a	07
00000050	64	65	66	61	75	6c	74	5a	07	31	32	2d	32	38	36	36
00000060	68	00	70	04	78	04	90	01	00	10	02	0a	5e	0a	12	0a
00000070	53	74	61	72	74	69	6e	67	20	25	73	20	54	61	73	6b
00000080	0a	10	04	1a	03	31	30	33	2a	0b	63	6f	6e	73	74	72
00000090	61	69	6e	74	73	32	23	0a	0f	6f	70	65	6e	5f	63	68
000000a0	65	63	6b	70	6f	69	6e	74	32	07	64	65	66	61	75	6c
000000b0	74	3a	07	64	65	66	61	75	6c	74	5a	06	31	38	2d	31
000000c0	30	33	68	00	70	04	78	04	90	01	00	10	02	0a	92	01
000000d0	0a	04	0a	25	73	0a	10	04	2a	0b	63	6f	6e	73	74	72
000000e0	61	69	6e	74	73	32	72	0a	5e	54	69	6d	65	20	28	73

Debug Core:

As we found that the synthesis timing for all types of encoding approximately the same , so we choose the gray as the gray won't need high power in switching case (just change 1 bit) compared to sequential ,and less area compared to One hot because One hot will add 1 more flipflop.

```
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]
```