Table of Contents

[Modules Design Code: 2](#_Toc192758216)

[RAM Design Code: 2](#_Toc192758217)

[SPI\_SLAVE\_DESIGN\_CODE: 4](#_Toc192758218)

[TOP\_MODULE\_DESIGN\_CODE: 8](#_Toc192758219)

[Test Bench: 9](#_Toc192758220)

[Do File: 12](#_Toc192758221)

[Constrain File: 12](#_Toc192758222)

[Encoding: 13](#_Toc192758223)

[ONE\_HOT: 13](#_Toc192758224)

[Elaborated Design: 13](#_Toc192758225)

[Synthesis: 13](#_Toc192758226)

[Implementation: 15](#_Toc192758227)

[Bitstream File: 16](#_Toc192758228)

[Gray\_Encoding: 16](#_Toc192758229)

[Elaborated Design: 16](#_Toc192758230)

[Synthesis: 16](#_Toc192758231)

[Implementation: 18](#_Toc192758232)

[Bitstream File: 19](#_Toc192758233)

[Sequential\_Encoding: 20](#_Toc192758234)

[Elaborated Design: 20](#_Toc192758235)

[Synthesis: 20](#_Toc192758236)

[Implementation: 22](#_Toc192758237)

[Bitstream File: 24](#_Toc192758238)

[Debug Core: 24](#_Toc192758239)

# 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

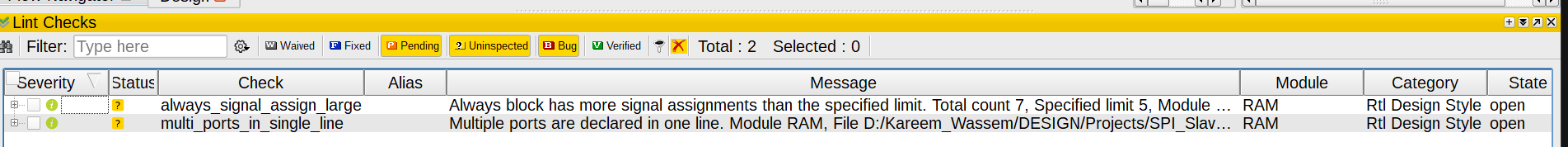
                end

            endcase

        end

    end

endmodule



## 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

        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

            communication\_ended();

        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

///////////////////////////////////////////// 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 accidentaly)

task communication\_ended();begin

    MISO                        <= 0 ;

    cycle\_counter\_PAR\_TO\_SER    <= 0 ;

    cycle\_counter\_SER\_TO\_PAR    <= 0 ;

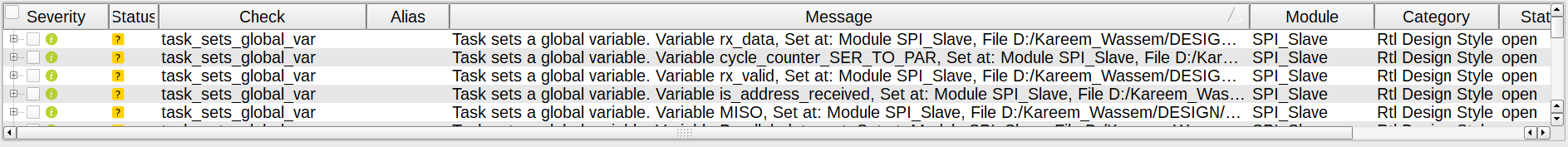
    rx\_valid                    <= 0 ;

    Parallel\_data\_out           <= 0 ;

end

endtask

endmodule



## 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

# A screenshot of a computer program AI-generated content may be incorrect.Do File:

# Constrain File:

## Clock signal

set\_property -dict { PACKAGE\_PIN W5   IOSTANDARD LVCMOS33 } [get\_ports CLK]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports CLK]

## Switches

set\_property -dict { PACKAGE\_PIN V17   IOSTANDARD LVCMOS33 } [get\_ports {rst\_n}]

set\_property -dict { PACKAGE\_PIN V16   IOSTANDARD LVCMOS33 } [get\_ports {SS\_n}]

set\_property -dict { PACKAGE\_PIN W16   IOSTANDARD LVCMOS33 } [get\_ports {MOSI}]

## LEDs

set\_property -dict { PACKAGE\_PIN U16   IOSTANDARD LVCMOS33 } [get\_ports {MISO}]

## Configuration options, can be used for all designs

set\_property CONFIG\_VOLTAGE 3.3 [current\_design]

set\_property CFGBVS VCCO [current\_design]

## SPI configuration mode options for QSPI boot, can be used for all designs

set\_property BITSTREAM.GENERAL.COMPRESS TRUE [current\_design]

set\_property BITSTREAM.CONFIG.CONFIGRATE 33 [current\_design]

set\_property CONFIG\_MODE SPIx4 [current\_design]

# Encoding:

## ONE\_HOT:

### Elaborated Design:

A diagram of a computer

AI-generated content may be incorrect.

### Synthesis:

A diagram of a computer

AI-generated content may be incorrect.

A white paper with black text and numbers

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#### A screenshot of a graph AI-generated content may be incorrect.Utilization Report:

#### Timing Report:

A screenshot of a computer

AI-generated content may be incorrect.

#### 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\_Wassem/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:

#### A screenshot of a game AI-generated content may be incorrect.FPGA:

#### A diagram of a machine AI-generated content may be incorrect.Schematic:

#### Timing:

A screenshot of a computer

AI-generated content may be incorrect.

### A screenshot of a computer screen AI-generated content may be incorrect.Bitstream File:

## Gray\_Encoding:

### Elaborated Design:

A computer screen shot of a computer

AI-generated content may be incorrect.

### Synthesis:

A diagram of a computer

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.

#### Utilization:

A screenshot of a graph

AI-generated content may be incorrect.

#### Timing:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#### 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\_Wassem/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

### A screenshot of a computer screen AI-generated content may be incorrect.Implementation:

#### FPGA:

#### A diagram of a computer AI-generated content may be incorrect.Schematic:

#### A screenshot of a graph AI-generated content may be incorrect.Utilization:

#### Timing:

A screenshot of a computer

AI-generated content may be incorrect.

A blue line with green lines

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

### A screenshot of a computer screen AI-generated content may be incorrect.Bitstream File:

## Sequential\_Encoding:

### Elaborated Design:

A diagram of a computer

AI-generated content may be incorrect.

### Synthesis:

A diagram of a computer

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.

#### A screenshot of a graph AI-generated content may be incorrect.Utilization:

#### Timing:

A computer screen shot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

#### 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\_Wassem/DESIGN/Projects/SPI\_Slave\_With\_Single\_Port\_RAM/PSEQENTIAL\_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:

A screenshot of a computer screen

AI-generated content may be incorrect.

#### Schematic:

A diagram of a computer

AI-generated content may be incorrect.

#### Utilization:

A screenshot of a graph

AI-generated content may be incorrect.

#### Timing:

A screenshot of a computer

AI-generated content may be incorrect.

### Bitstream File:

A screenshot of a computer screen

AI-generated content may be incorrect.

## 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]