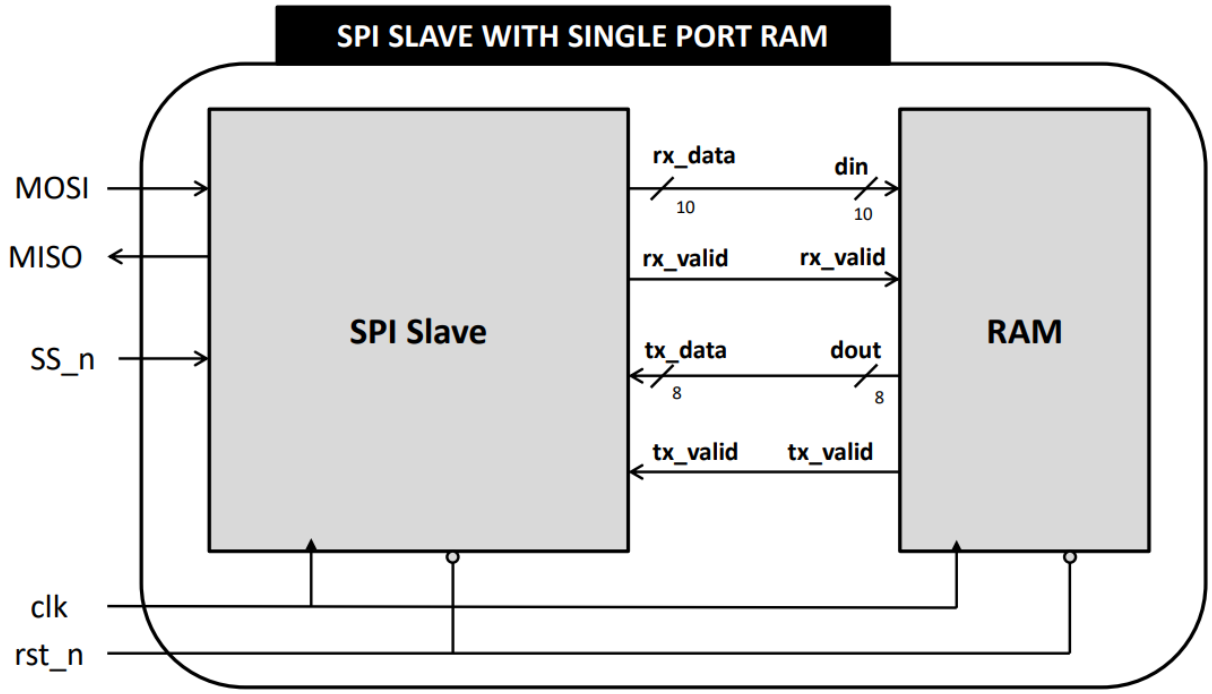
SPI Slave with Single Port RAM



By:

Roaa Atef Mohamed

Salah Eldin Hassen Hassen

Shahd Gamal Mahmoud

Table of Contents

1. RTL Code

➢ Single Port RAM Code

```
module single_port_async_ram
    #(
        parameter MEM_DEPTH = 256,
        parameter ADDR_SIZE = 8,
        parameter INPUT_SIZE = 10,
        parameter WORD_SIZE = 8
    )
    (
        input wire [INPUT_SIZE-1:0] din,    // Data input most 2 significant bits defines the operation
        /**
         * # din[9:8] Operation (most 2 significant bits)
         * ? Write operation:
         * * 00: Hold din as write address
         * * 01: Write din as data in write address
         * ? Read operation:
         * * 10: Hold din as read address
         * * 11: Read data from read address and tx_valid should be HIGH

         * ? Note that the most significant bit determines the operation if read or write.
        */

        input wire clk,                     // Clock
        input wire rst_n,                   // Active low synchronous reset
        input wire rx_valid,                // If HIGH: accept din to save address or write memory
        output reg [WORD_SIZE-1:0] dout,    // Data output
        output reg tx_valid                 // Becomes HIGH when data is ready to be read
    );

    // Memory
    reg [WORD_SIZE-1:0] mem [0:MEM_DEPTH-1];

    // Address registers
    reg [ADDR_SIZE-1:0] addr_wr_reg;
    reg [ADDR_SIZE-1:0] addr_rd_reg;

    always @(posedge clk) begin
        if (~rst_n) begin
            addr_wr_reg <= 0;
            addr_rd_reg <= 0;
            dout <= 0;
            tx_valid <= 0;
        end else begin
            if(rx_valid)  // If rx_valid is HIGH, then do the operation
            begin
                case (din[INPUT_SIZE-1:INPUT_SIZE-2])
                    2'b00: begin
                        addr_wr_reg <= din[WORD_SIZE-1:0];
                        tx_valid <= 0;
                    end
                    2'b01: begin
                        mem[addr_wr_reg] <= din[WORD_SIZE-1:0];
                        tx_valid <= 0;
                    end
                    2'b10: begin
                        addr_rd_reg <= din[WORD_SIZE-1:0];
                        tx_valid <= 0;
                    end
                    2'b11: begin
                        dout <= mem[addr_rd_reg];
                        tx_valid <= 1;
                    end
                endcase
            end
        end
    end

endmodule
```

➢ Design SPI Interface

```
module spi_slave_interface
    #(
        parameter MEM_DEPTH = 256,
        parameter MEM_ADDR_SIZE = 8,
        parameter MEM_INPUT_SIZE = 10,
        parameter MEM_WORD_SIZE = 8
    )
    (
        input wire MOSI,    // Master Out Slave In
        output reg MISO,    // Master In Slave Out
        input wire clk,     // Serial Clock
        input wire SS_n,    // Slave Select
        input wire rst_n    // Active low synchronous reset
    );

    //// ! Ram Signals
    wire tx_valid;                      // Becomes HIGH when data is ready to be read
    wire [MEM_WORD_SIZE-1:0] tx_data;   // Data input from Memory
    reg rx_valid;                       // Becoms HIGH when sending data to Memory
    reg [MEM_INPUT_SIZE-1:0] rx_data;   // Data output to Memory

    //// ! RAM
    single_port_async_ram #(
                                .MEM_DEPTH(MEM_DEPTH),
                                .ADDR_SIZE(MEM_ADDR_SIZE),
                                .INPUT_SIZE(MEM_INPUT_SIZE),
                                .WORD_SIZE(MEM_WORD_SIZE)
                            ) ram (
                                .din(rx_data),
                                .clk(clk),
                                .rst_n(rst_n),
                                .rx_valid(rx_valid),
                                .dout(tx_data),
                                .tx_valid(tx_valid)
                            );

```

```
    //// ! SPI Slave Interface

    // States
    localparam IDLE         = 0;    // Idle state
    localparam CHK_CMD      = 1;    // Takes one MOSI input
    localparam WRITE        = 2;    // Writes address/data is being sent to RAM
    localparam READ_ADD     = 3;    // Read address is being sent to RAM
    localparam READ_DATA    = 4;    // Read data is being saved then sent to MISO

    // Registers for current state (cs) and next state (ns)
    (* fsm_encoding = "sequential" *)
    reg [2:0] cs, ns;

    // Counter to keep track of bits received or transmitted
    reg [4:0] counter;

    // Flag to indicate a read operation
    reg READ_OP;

```

```
    // # State Memory: Sequential logic to update the current state
    always @(posedge clk)
    begin
        if(~rst_n)
            cs <= IDLE;  // Reset to IDLE state
        else
            cs <= ns;    // Update to next state
    end

```

```
    // # Next State Logic: Combinational logic to determine the next state
    always @(*)
    begin
        if(~rst_n) begin
            ns = IDLE;                      // Reset to IDLE state
        end
        else begin
            case(cs)
                IDLE: begin
                    if(~SS_n)
                        ns = CHK_CMD;       // Move to CHK_CMD state if Slave Select is active
                    else
                        ns = IDLE;          // Remain in IDLE state
                end
                CHK_CMD: begin
                    if(MOSI == 0)
                        ns = WRITE;         // Move to WRITE state if MOSI is 0 (Write operation)
                    else begin
                        if(READ_OP)
                            ns = READ_DATA; // Move to READ_DATA state if READ_OP is set (Address received before)
                        else
                            ns = READ_ADD;  // Move to READ_ADD state otherwise (Address receiveing)
                    end
                end
                WRITE: begin
                    if(SS_n)
                        ns = IDLE;          // Move to IDLE state if Slave Select is inactive
                    else
                        ns = WRITE;         // Remain in WRITE state
                end
                READ_ADD: begin
                    if(SS_n)
                        ns = IDLE;          // Move to IDLE state if Slave Select is inactive
                    else
                        ns = READ_ADD;      // Remain in READ_ADD state
                end
                READ_DATA: begin
                    if(SS_n)
                        ns = IDLE;          // Move to IDLE state if Slave Select is inactive
                    else
                        ns = READ_DATA;     // Remain in READ_DATA state
                end
                default: ns = IDLE;         // Default to IDLE state
            endcase
        end
    end

```

```
    // # Output Logic: Sequential logic to generate outputs based on the current state
    always @(posedge clk) begin
        if(~rst_n)begin
            counter <= 4'b0;
            MISO <= 0;
            READ_OP <= 0;
        end
        else begin
            MISO <= 0;                  // Default MISO to 0
            rx_valid <= 0;              // Default rx_valid to 0
            case(cs)
                IDLE: begin
                    // No operation in IDLE state
                end
                CHK_CMD: begin
                    counter <= 0;       // Reset counter in CHK_CMD state
                end
                WRITE: begin
                    if(counter >= 10)
                        rx_valid <= 1;  // Set rx_valid when 10 bits are received
                    else begin
                        rx_data <= {rx_data[MEM_INPUT_SIZE-2:0], MOSI};  // Shift in MOSI data
                        rx_valid <= 0;
                        counter <= counter + 1;
                    end
                end
```

```
                READ_ADD: begin
                    READ_OP <= 1;        // Set READ_OP flag as Address is received
                    if(counter >= 10) begin
                        rx_valid <= 1;  // Set rx_valid when 10 bits are received
                    end
                    else begin
                        rx_data <= {rx_data[MEM_INPUT_SIZE-2:0], MOSI};  // Shift in MOSI data
                        rx_valid <= 0;
                        counter <= counter + 1;
                    end
                end
                READ_DATA: begin
                    READ_OP <= 0;        // Clear READ_OP flag as Data is received
                    if(counter >= 18) begin
                        // No operation when counter exceeds 18
                    end
                    else if(counter >= 10) begin
                        rx_valid <= 1;      // Set rx_valid when 10 bits are received
                        if(tx_valid) begin  // If tx_valid is HIGH (Data received from RAM), then send data to MISO
                            counter <= counter + 1;
                            MISO <= tx_data[17-counter];  // Send data on MISO
                        end
                    end
                    else begin
                        rx_data <= {rx_data[MEM_INPUT_SIZE-2:0], MOSI};  // Shift in MOSI data
                        counter <= counter + 1;
                        if(counter >= 10)
                            rx_valid <= 1;  // Set rx_valid when 10 bits are received to pass to RAM
                    end
                end
                default: begin
                    rx_valid <= 0;
                    MISO <= 0;
                    counter <= 0;
                end
            endcase
        end
    end

endmodule

```

2. Testbench Code

- Single Port RAM Testbench

```
module single_port_async_ram_tb;

    // Parameters
    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;
    parameter INPUT_SIZE = 10;
    parameter WORD_SIZE = 8;

    // Signals
    reg [INPUT_SIZE-1:0] din;
    reg clk;
    reg rst_n;
    reg rx_valid;
    wire [WORD_SIZE-1:0] dout;
    wire tx_valid;

    // Instantiate the DUT (Device Under Test)
    single_port_async_ram #(
                                .MEM_DEPTH(MEM_DEPTH),
                                .ADDR_SIZE(ADDR_SIZE),
                                .INPUT_SIZE(INPUT_SIZE),
                                .WORD_SIZE(WORD_SIZE)
                            ) dut (
                                .din(din),
                                .clk(clk),
                                .rst_n(rst_n),
                                .rx_valid(rx_valid),
                                .dout(dout),
                                .tx_valid(tx_valid)
                            );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
```

```
    // Test sequence
    initial begin
        // Initialize signals
        rst_n = 0;
        din = 0;
        rx_valid = 0;

        // Reset the DUT
        #10;
        rst_n = 1;

        // Write address 0x01
        #10;
        din = 10'b0000000001; // Write address 0x01
        rx_valid = 1;
        #10;
        rx_valid = 0;
```

```
        // Write data 0xAA to address 0x01
        #10;
        din = 10'b0100001010; // Write data 0x0A
        rx_valid = 1;
        #10;
        rx_valid = 0;

        // Read address 0x01
        #10;
        din = 10'b1000000001; // Read address 0x01
        rx_valid = 1;
        #10;
        rx_valid = 0;

        // Read data from address 0x01
        #10;
        din = 10'b1100000000; // Read data
        rx_valid = 1;
        #10;
        rx_valid = 0;

        // Wait for tx_valid to go high and check the output
        #10;
        if (tx_valid && dout == 8'h0A) begin
            $display("Test Passed: Data read correctly from address 0x01");
        end else begin
            $display("Test Failed: Incorrect data read from address 0x01");
        end

        // Finish the simulation
        #10;
        $stop;
    end

endmodule

```

➢ SPI Interface Testbench

```
module spi_slave_interface_tb();
    //parameters
    parameter MEM_DEPTH = 256;
    parameter MEM_ADDR_SIZE = 8;
    parameter MEM_INPUT_SIZE = 10;
    parameter MEM_WORD_SIZE = 8;
    ////////////////////////////////
    //input and output signals
    reg MOSI_tb;
    wire MISO_tb;
    reg clk_tb;
    reg SS_n_tb;
    reg rst_n_tb;
    ////////////////////////////////////
    //instantiation of the module
    spi_slave_interface spi_dut (
                                MOSI_tb,
                                MISO_tb,
                                clk_tb,
                                SS_n_tb,
                                rst_n_tb
                            );
    ////////////////////////////////
    reg [9:0] send_data;
    reg [7:0] address;
    reg [7:0] expected_data;
    reg [7:0] received_data;

    ////////////////////////////////////////
    //clk generation
    initial begin
        clk_tb=0;
        forever
            #5 clk_tb=~clk_tb;
    end
    ////////////////////////////////////////
    initial begin
        // test reset
        rst_n_tb = 0;
        SS_n_tb = 1;
        MOSI_tb = 0;
        send_data = 0;
        @(negedge clk_tb);
        ////////////////////////////////////////////

        repeat (40) begin
            //test sending address for write
            rst_n_tb = 1;
            SS_n_tb = 0;
            send_data = $random;
            send_data[9:8] = 2'b00;
            address = send_data[7:0];
            send_data_task();
            @(negedge clk_tb);
            @(negedge clk_tb);
            SS_n_tb = 1;
            @(negedge clk_tb);
            @(negedge clk_tb);

```

```
            // test writing data
            SS_n_tb = 0;
            send_data = $random;
            send_data[9:8] = 2'b01;
            expected_data = send_data[7:0];
            send_data_task();
            @(negedge clk_tb);
            @(negedge clk_tb);
            SS_n_tb = 1;
            @(negedge clk_tb);
            @(negedge clk_tb);

            //test writing read address
            SS_n_tb = 0;
            send_data = address;
            send_data[9:8] = 2'b10;
            send_data_task();
            @(negedge clk_tb);
            @(negedge clk_tb);
            SS_n_tb = 1;
            @(negedge clk_tb);
            @(negedge clk_tb);

            //read data
            SS_n_tb = 0;
            send_data = 10'b1100000000;
            send_data_task();
            @(negedge clk_tb);
            @(negedge clk_tb);
            get_data_task();
            SS_n_tb = 1;
            @(negedge clk_tb);
        end

        @(negedge clk_tb);
        $stop;
    end
```

```
    // Task to send data to the DUT
    task send_data_task();
        integer i;
        begin
            @(negedge clk_tb);
            MOSI_tb = send_data[9];
            @(negedge clk_tb);
            for (i = 9; i >= 0; i = i - 1) begin
                MOSI_tb = send_data[i];
                @(negedge clk_tb);
            end
            MOSI_tb = 0;
        end
    endtask

    // Task to get data from the DUT
    task get_data_task();
        integer i;
        begin
            for (i = 7; i >= 0; i = i - 1) begin
                @(negedge clk_tb);
                received_data[i] = MISO_tb;
            end
            if(expected_data != received_data) begin
                $display("Test Failed: Expected Data = %h, Received Data = %h, Address = %h", expected_data, received_data, address);
            end
            else begin
                $display("Test Passed: Expected Data = %h, Received Data = %h, Address = %h", expected_data, received_data, address);
            end
        end
    endtask
    ////////////////////////////
endmodule
```

3. Do File

- Single Port RAM

```
vlib work
vlog single_port_async_ram.v single_port_async_ram_tb.v
vsim -voptargs=+acc work.single_port_async_ram_tb
add wave *
run -all

```

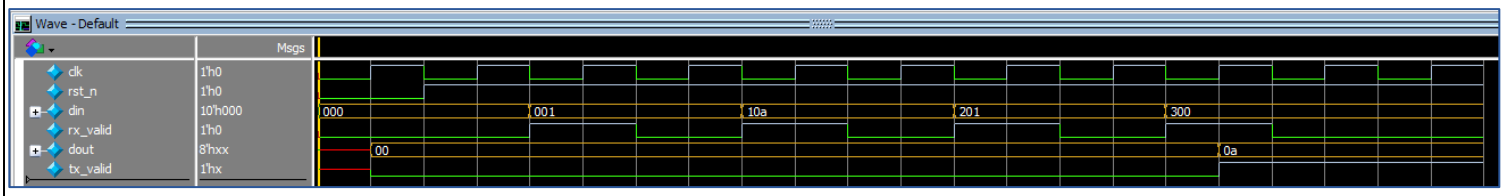
- SPI Interface

```
vlib work
vlog single_port_async_ram.v spi_slave_interface.v spi_slave_interface_tb.v
vsim -voptargs=+acc work.spi_slave_interface_tb
add wave *
run -all

```

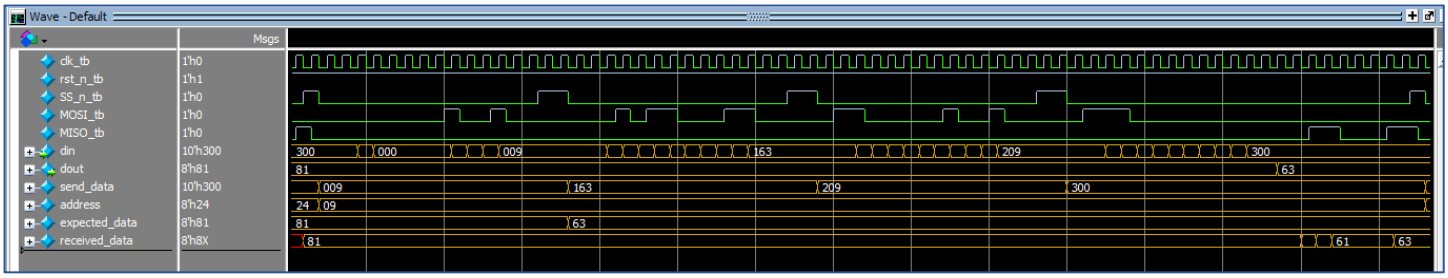
4. QuestaSim Snippets

- Single Port RAM Wave

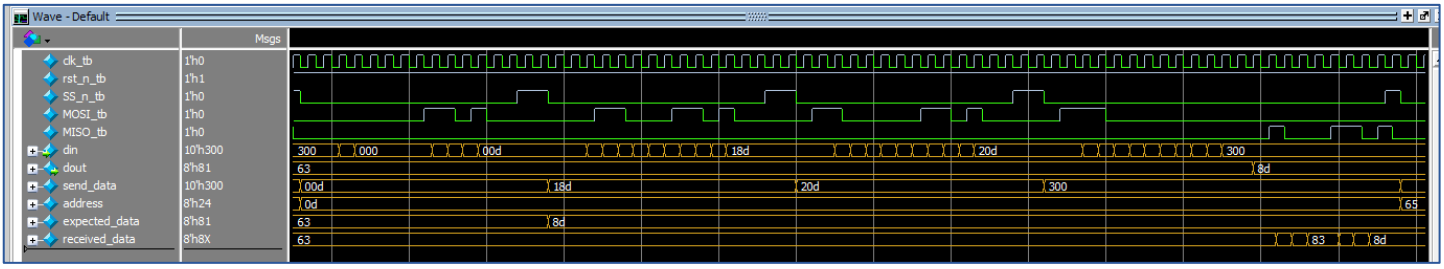


- SPI Interface Wave

Example 1



Example 2



5. Constraint File

```
## FPGA part: xc7a35ticpg236-1L

## 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 MOSI]
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports SS_n]
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports rst_n]

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

## Debug Core
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]
```

6. Encoding

1. Gray Encoding

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

2. OneHot Encoding

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

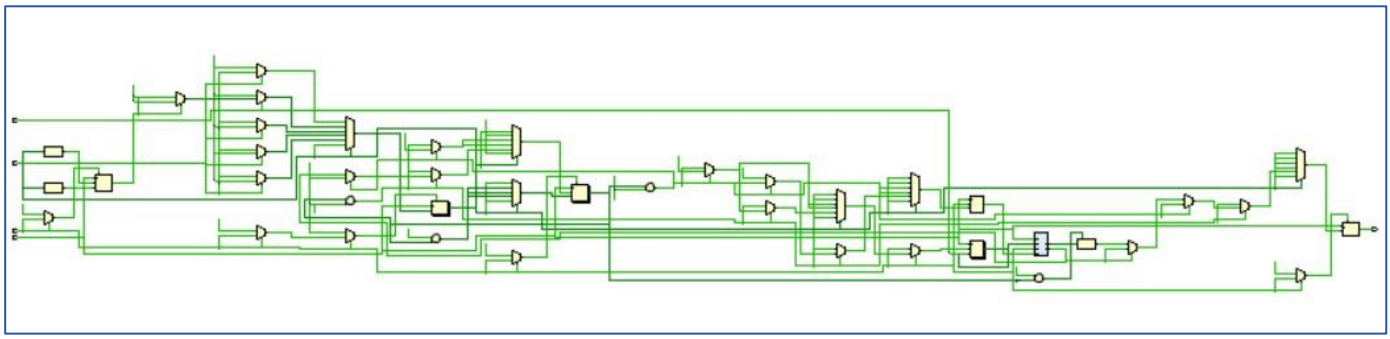
3. Sequential Encoding

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

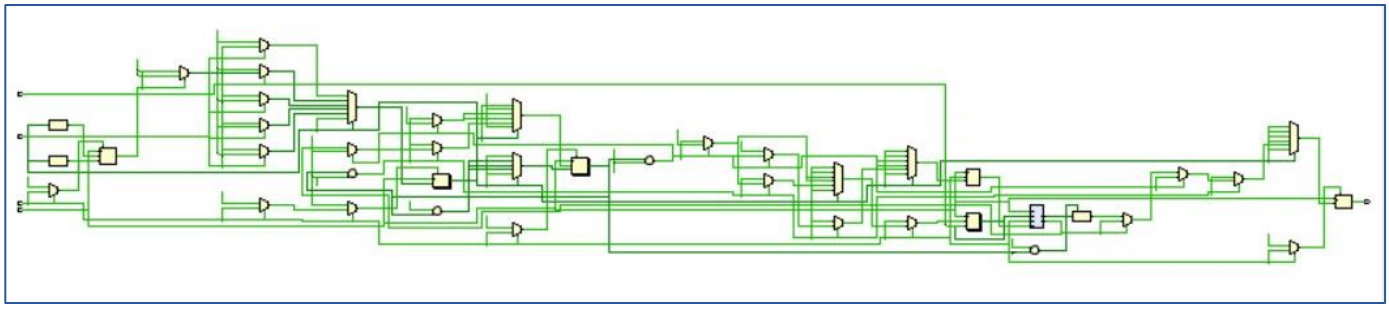
7. Elaboration

- **Schematic Snippets**

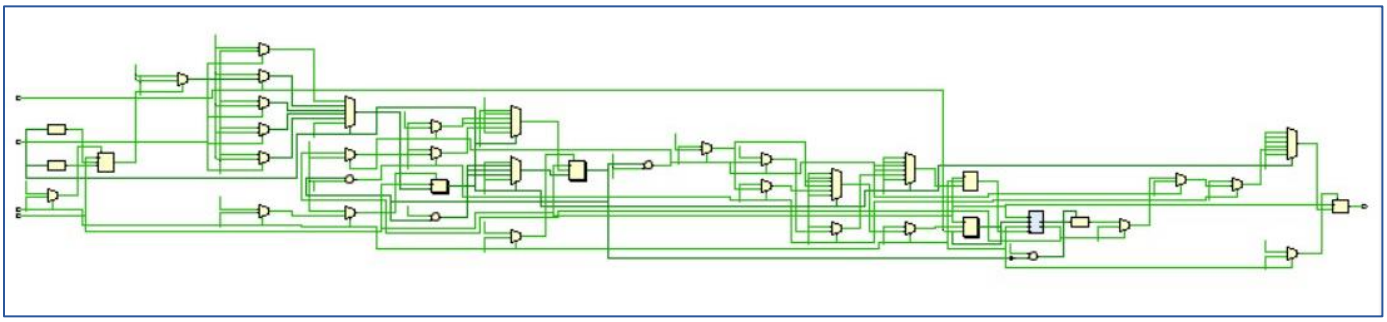
1. Gray Encoding



2. OneHot Encoding



3. Sequential Encoding

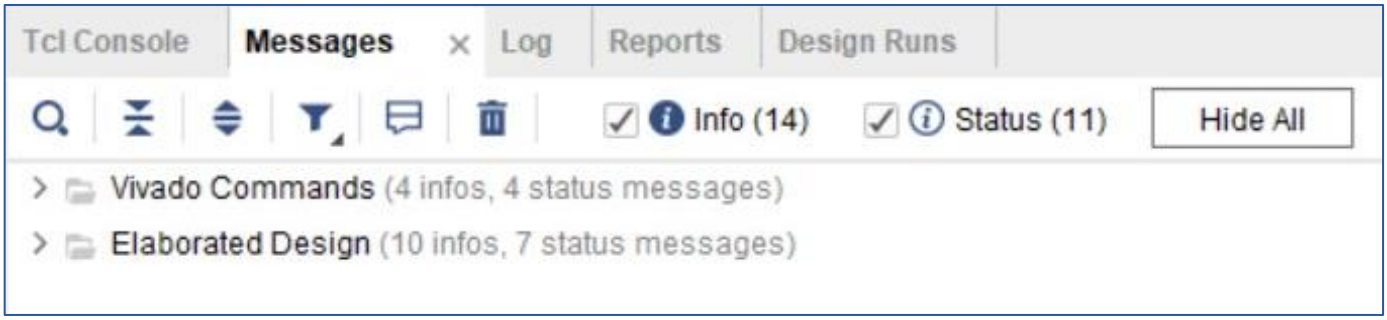


- **Messages (No Errors)**

1. Gray Encoding

2. OneHot Encoding

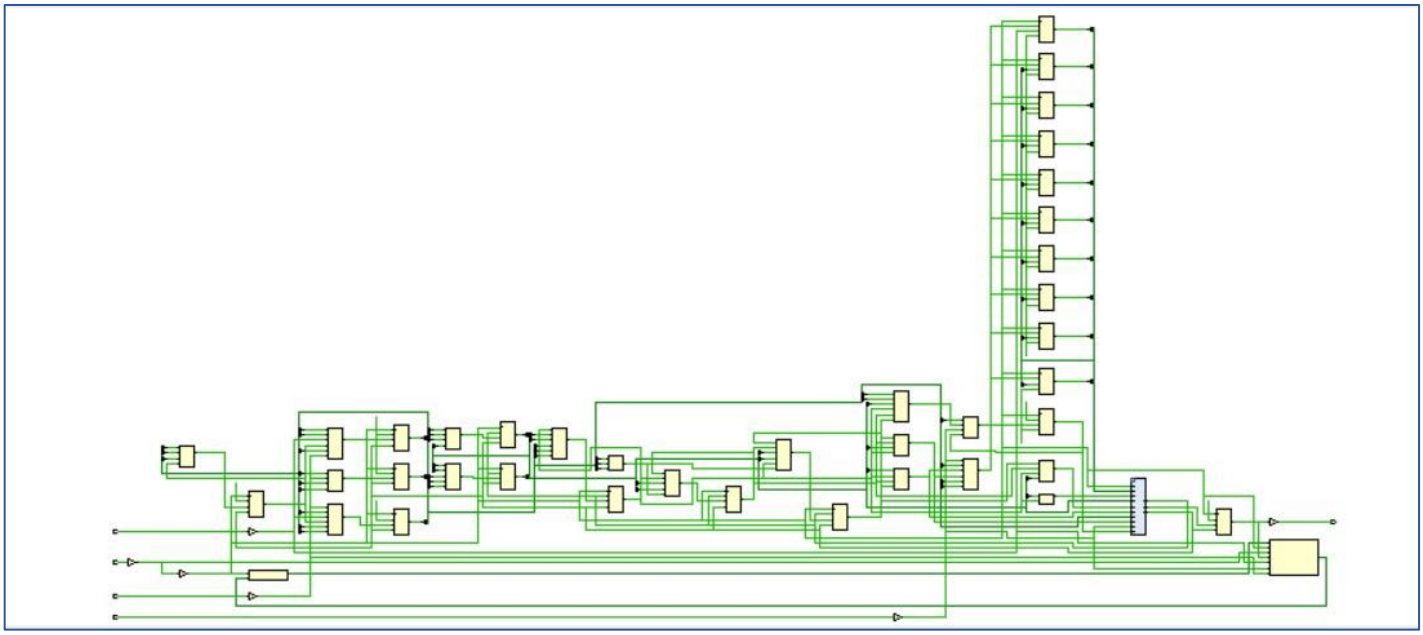
3. Sequential Encoding



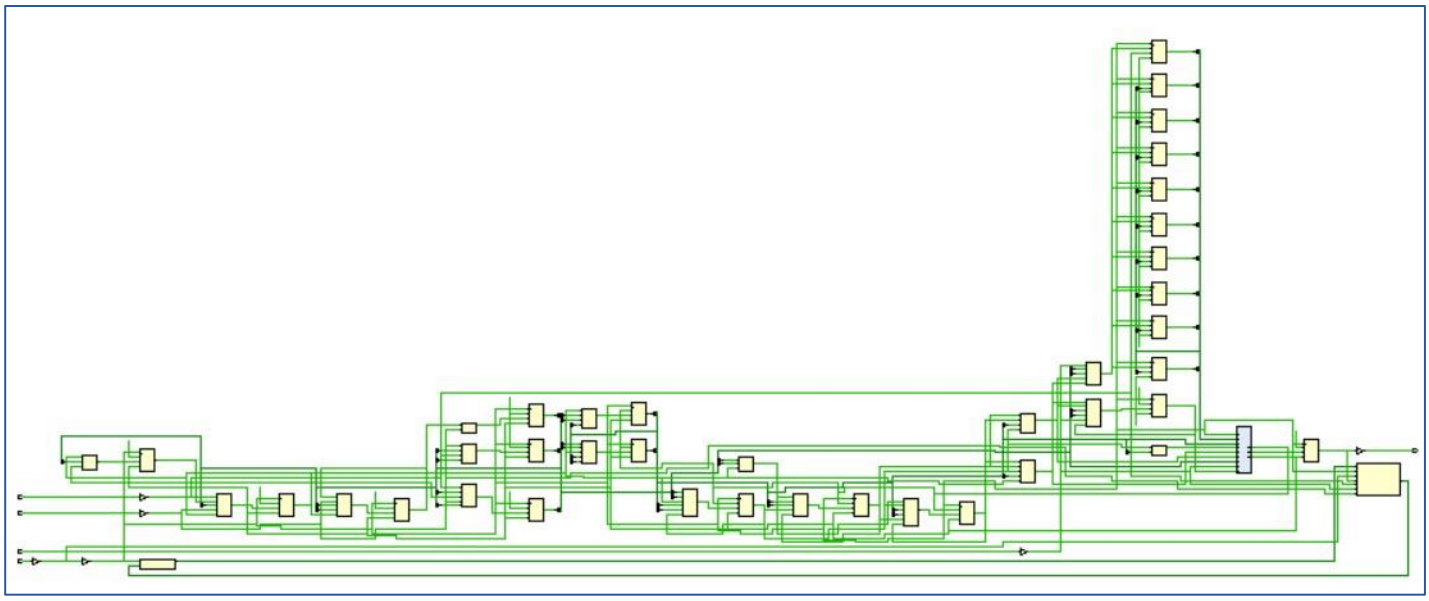
8. Synthesis

- **Schematic Snippets**

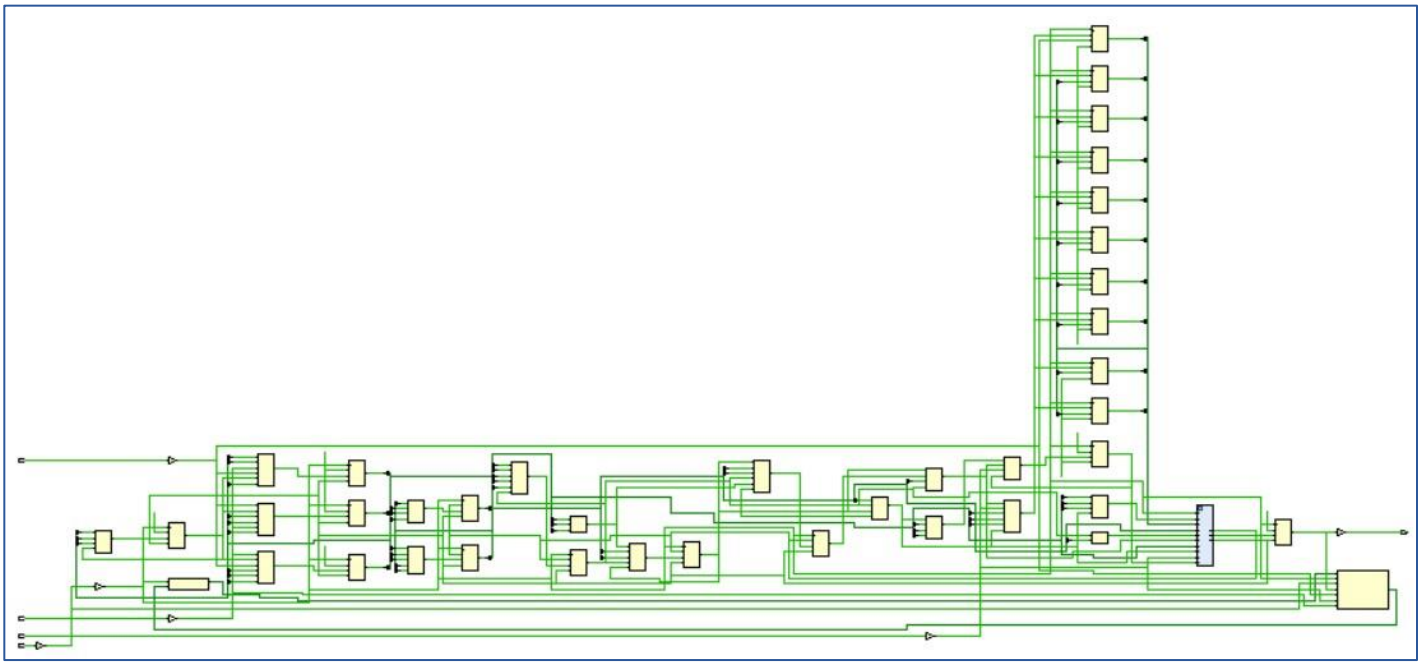
1. Gray Encoding



2. OneHot Encoding

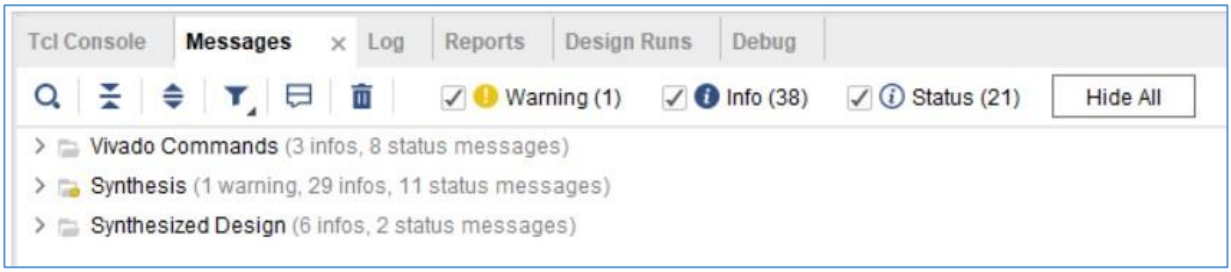


3. Sequential Encoding

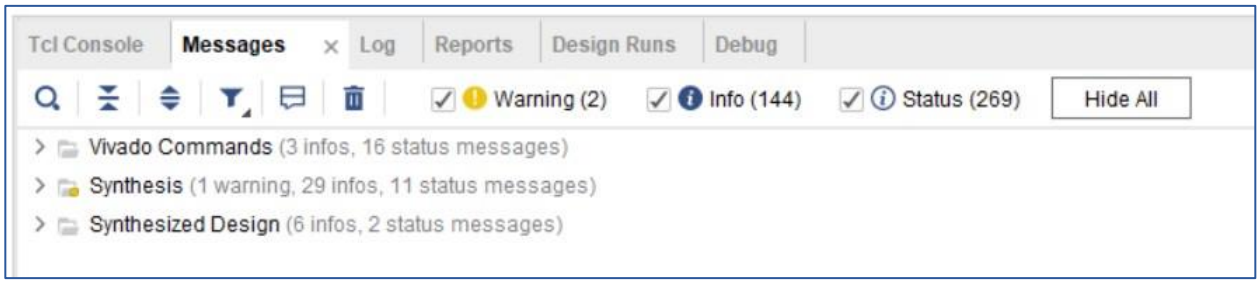


- **Messages (No Errors)**

1. Gray Encoding



2. OneHot Encoding



3. Sequential Encoding

- **Utilization report**

1. Gray Encoding

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_slave_interface	26	38	1	0.5	5	1
dbg_hub (dbg_hub_CV)	0	0	0	0	0	0
ram (single_port_asyn...	10	17	1	0.5	0	0
u_ila_0 (u_ila_0_CV)	0	0	0	0	0	0

2. OneHot Encoding

Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_slave_interface	26	40	0.5	5	1
dbg_hub (dbg_hub_CV)	0	0	0	0	0
ram (single_port_asyn...	10	17	0.5	0	0
u_ila_0 (u_ila_0_CV)	0	0	0	0	0

3. Sequential Encoding

Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
spi_slave_interface	25	38	0.5	5	1
dbg_hub (dbg_hub_CV)	0	0	0	0	0
ram (single_port_asyn...	10	17	0.5	0	0
u_ila_0 (u_ila_0_CV)	0	0	0	0	0

➢ Time report

1. Gray Encoding

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.476 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:	95	Total Number of Endpoints:	95	Total Number of Endpoints:	41

All user specified timing constraints are met.

2. OneHot Encoding

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.898 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:	97	Total Number of Endpoints:	97	Total Number of Endpoints:	43

All user specified timing constraints are met.

3. Sequential Encoding

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.445 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:	95	Total Number of Endpoints:	95	Total Number of Endpoints:	41

All user specified timing constraints are met.

➢ Power report

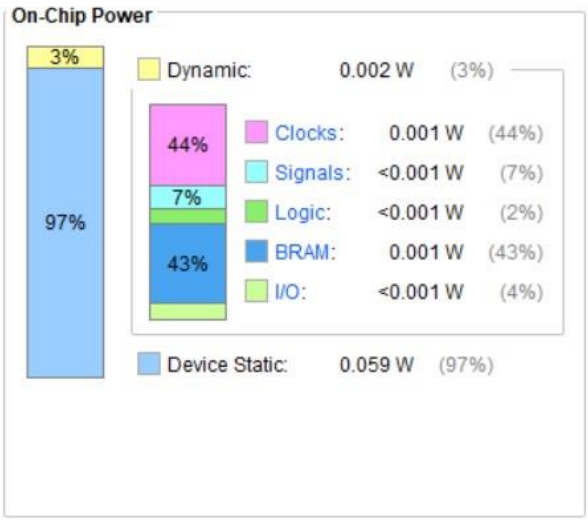
1. Gray Encoding

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.061 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (14.9 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



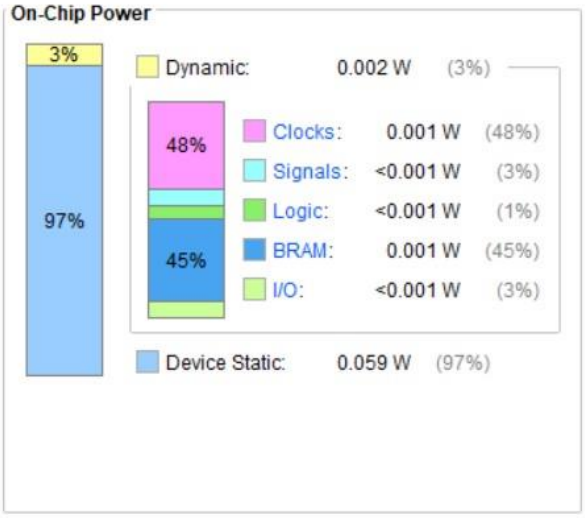
2. OneHot Encoding

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.061 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (14.9 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



3. Sequential Encoding

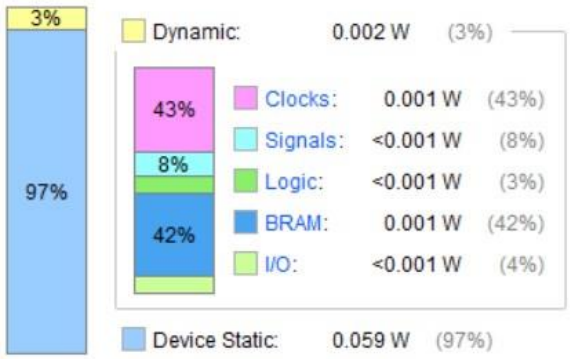
Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.061 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (14.9 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



9. Debug Cores

1. Gray Encoding

Name	Debug Core Instance	Debug Core Type	Debug Port	Driver Cell	Driver Name	Driver Pin
Assigned Debug Nets						
clk_IBUF	u_ila_0	labtools_ila_v6	probe0	IBUF	clk_IBUF_inst	O
MISO_OBUF	u_ila_0	labtools_ila_v6	probe1	FDRE	MISO_reg	Q
MOSI_IBUF	u_ila_0	labtools_ila_v6	probe2	IBUF	MOSI_IBUF_inst	O
rst_n_IBUF	u_ila_0	labtools_ila_v6	probe3	IBUF	rst_n_IBUF_inst	O
SS_n_IBUF	u_ila_0	labtools_ila_v6	probe4	IBUF	SS_n_IBUF_inst	O
Unassigned Debug Nets (0)						

2. OneHot Encoding

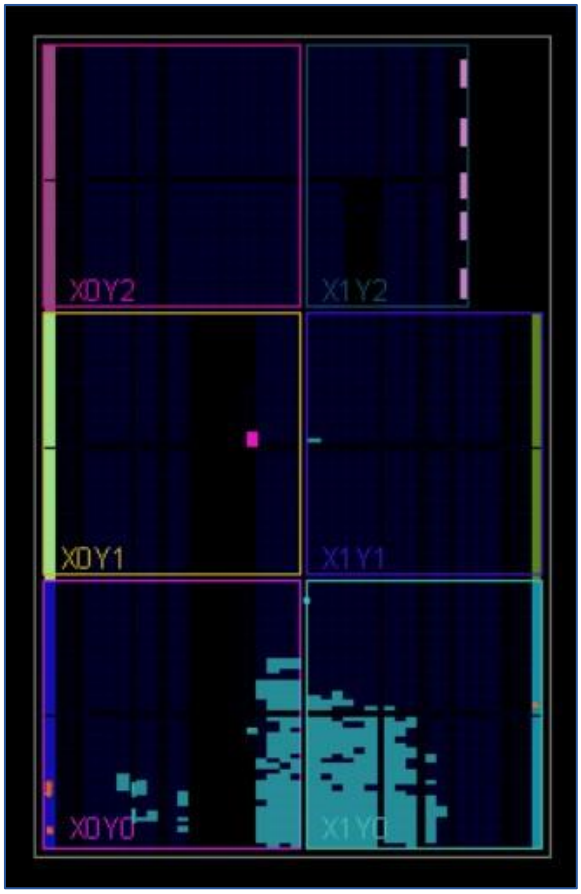
Name	Debug Core Instance	Debug Core Type	Debug Port	Driver Cell	Driver Name	Driver Pin
Assigned Debug Nets						
clk_IBUF	u_ila_0	labtools_ila_v6	probe0	IBUF	clk_IBUF_inst	O
MISO_OBUF	u_ila_0	labtools_ila_v6	probe1	FDRE	MISO_reg	Q
MOSI_IBUF	u_ila_0	labtools_ila_v6	probe2	IBUF	MOSI_IBUF_inst	O
rst_n_IBUF	u_ila_0	labtools_ila_v6	probe3	IBUF	rst_n_IBUF_inst	O
SS_n_IBUF	u_ila_0	labtools_ila_v6	probe4	IBUF	SS_n_IBUF_inst	O
Unassigned Debug Nets (0)						

3. Sequential Encoding

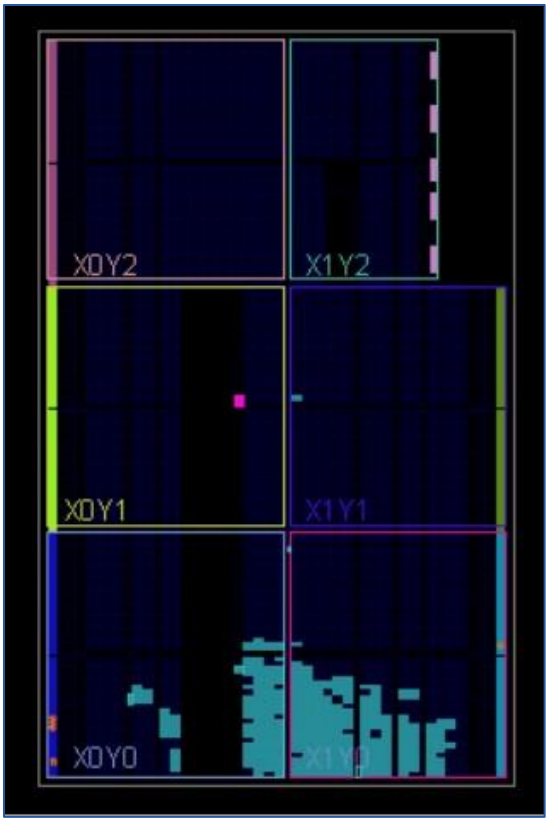
Name	Debug Core Instance	Debug Core Type	Debug Port	Driver Cell	Driver Name	Driver Pin
Assigned Debug Nets						
clk_IBUF	u_ila_0	labtools_ila_v6	probe0	IBUF	clk_IBUF_inst	O
MISO_OBUF	u_ila_0	labtools_ila_v6	probe1	FDRE	MISO_reg	Q
MOSI_IBUF	u_ila_0	labtools_ila_v6	probe2	IBUF	MOSI_IBUF_inst	O
rst_n_IBUF	u_ila_0	labtools_ila_v6	probe3	IBUF	rst_n_IBUF_inst	O
SS_n_IBUF	u_ila_0	labtools_ila_v6	probe4	IBUF	SS_n_IBUF_inst	O
Unassigned Debug Nets (0)						

10. Implementation

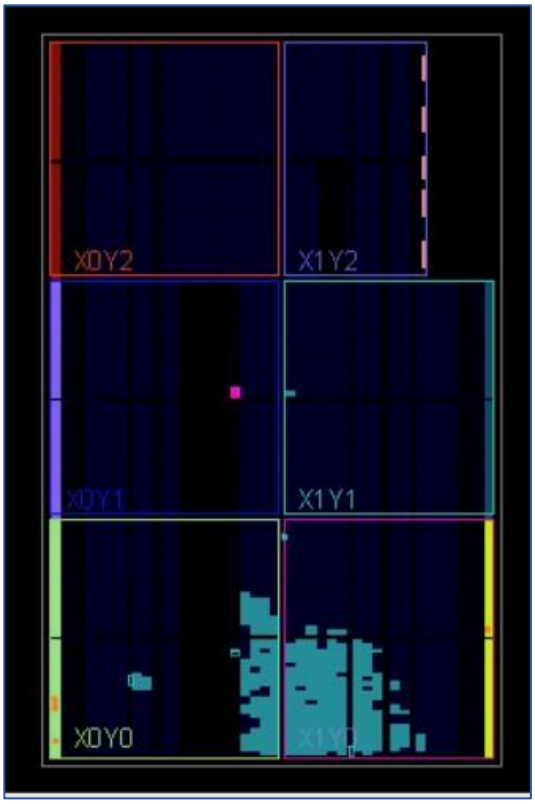
- **Device**
  1. Gray Encoding



  2. OneHot Encoding



3. Sequential Encoding

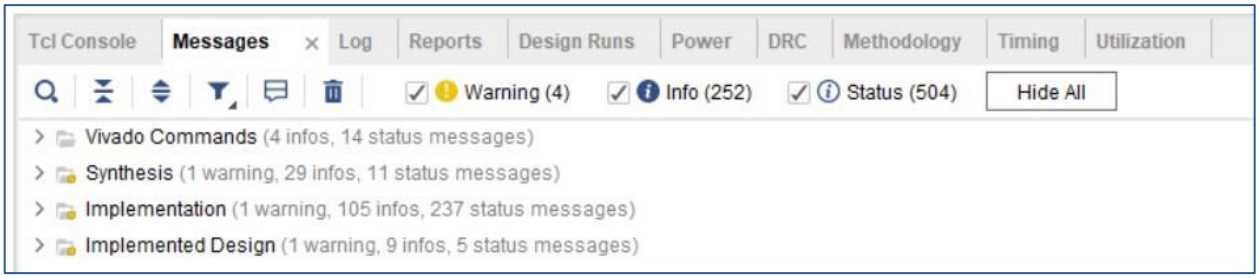


- **Messages (No Errors)**

1. Gray Encoding

2. OneHot Encoding

3. Sequential Encoding



➢ Utilization report

1. Gray Encoding

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
spi_slave_interface	1264	1954	11	626	1156	108	749	1	5	2	1
dbg_hub (dbg_hub)	475	727	0	238	451	24	312	0	0	1	1
ram (single_port_asyn...	10	17	1	7	10	0	1	0.5	0	0	0
u_ila_0 (u_ila_0)	763	1189	10	380	679	84	425	0.5	0	0	0

2. OneHot Encoding

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
spi_slave_interface	1264	1956	10	618	1156	108	746	1	5	2	1
dbg_hub (dbg_hub)	475	727	0	234	451	24	308	0	0	1	1
ram (single_port_asyn...	10	17	0	8	10	0	1	0.5	0	0	0
u_ila_0 (u_ila_0)	763	1189	10	378	679	84	421	0.5	0	0	0

3. Sequential Encoding

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
spi_slave_interface	1265	1954	10	613	1157	108	735	1	5	2	1
dbg_hub (dbg_hub)	476	727	0	238	452	24	303	0	0	1	1
ram (single_port_asyn...	10	17	0	4	10	0	1	0.5	0	0	0
u_ila_0 (u_ila_0)	764	1189	10	370	680	84	420	0.5	0	0	0

➢ Time report

1. Gray Encoding

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.310 ns	Worst Hold Slack (WHS):	0.040 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	3880	Total Number of Endpoints:	3864	Total Number of Endpoints:	2141

All user specified timing constraints are met.

2. OneHot Encoding

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.447 ns	Worst Hold Slack (WHS):	0.025 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	3882	Total Number of Endpoints:	3866	Total Number of Endpoints:	2143

All user specified timing constraints are met.

3. Sequential Encoding

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.460 ns	Worst Hold Slack (WHS):	0.049 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	3880	Total Number of Endpoints:	3864	Total Number of Endpoints:	2141

All user specified timing constraints are met.

As shown, **Sequential Encoding** has the highest setup/hold slack after implementation

➢ Power report

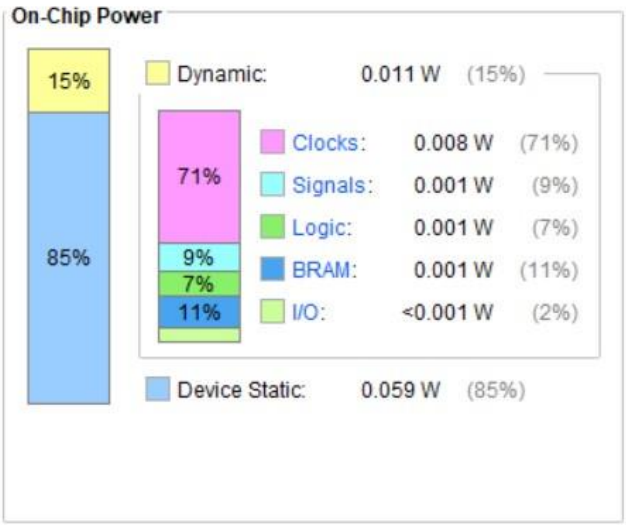
1. Gray Encoding

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.069 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (14.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



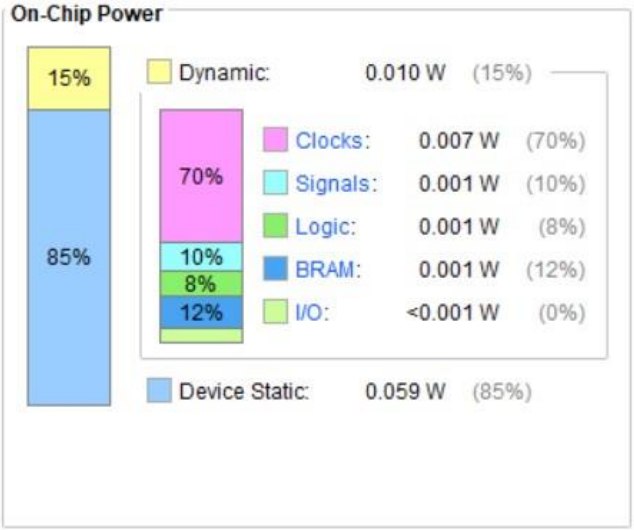
2. OneHot Encoding

Summary

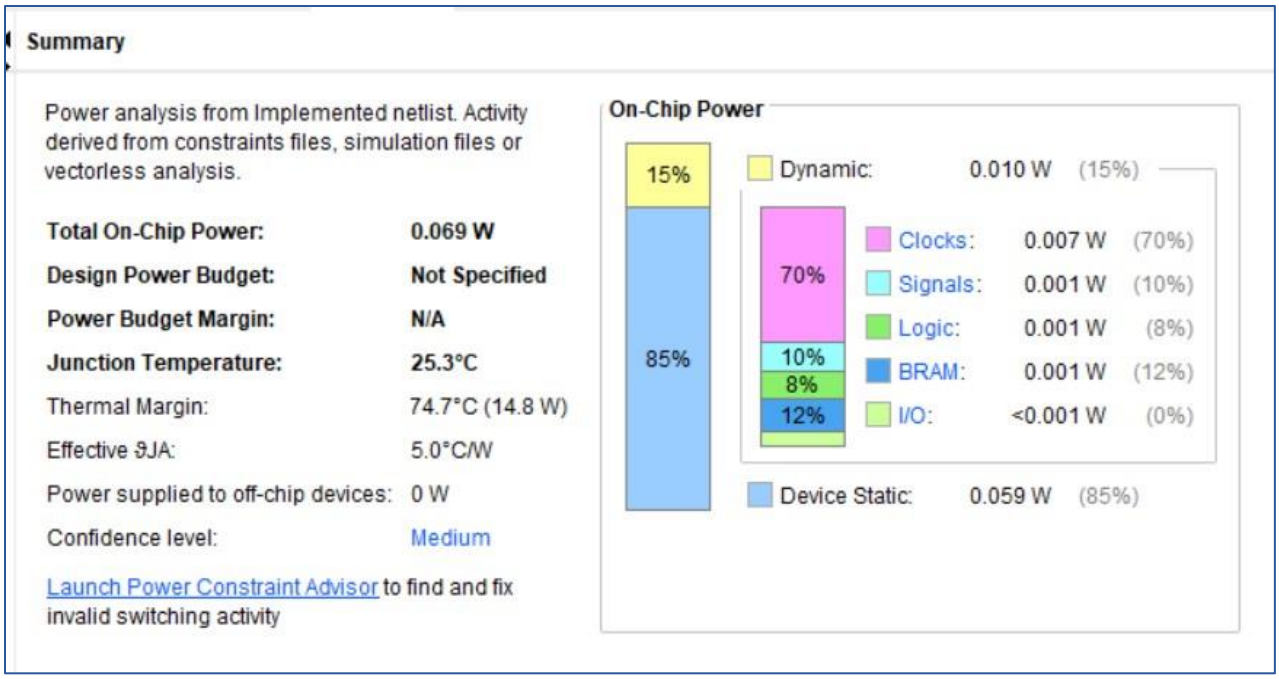
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.069 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (14.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



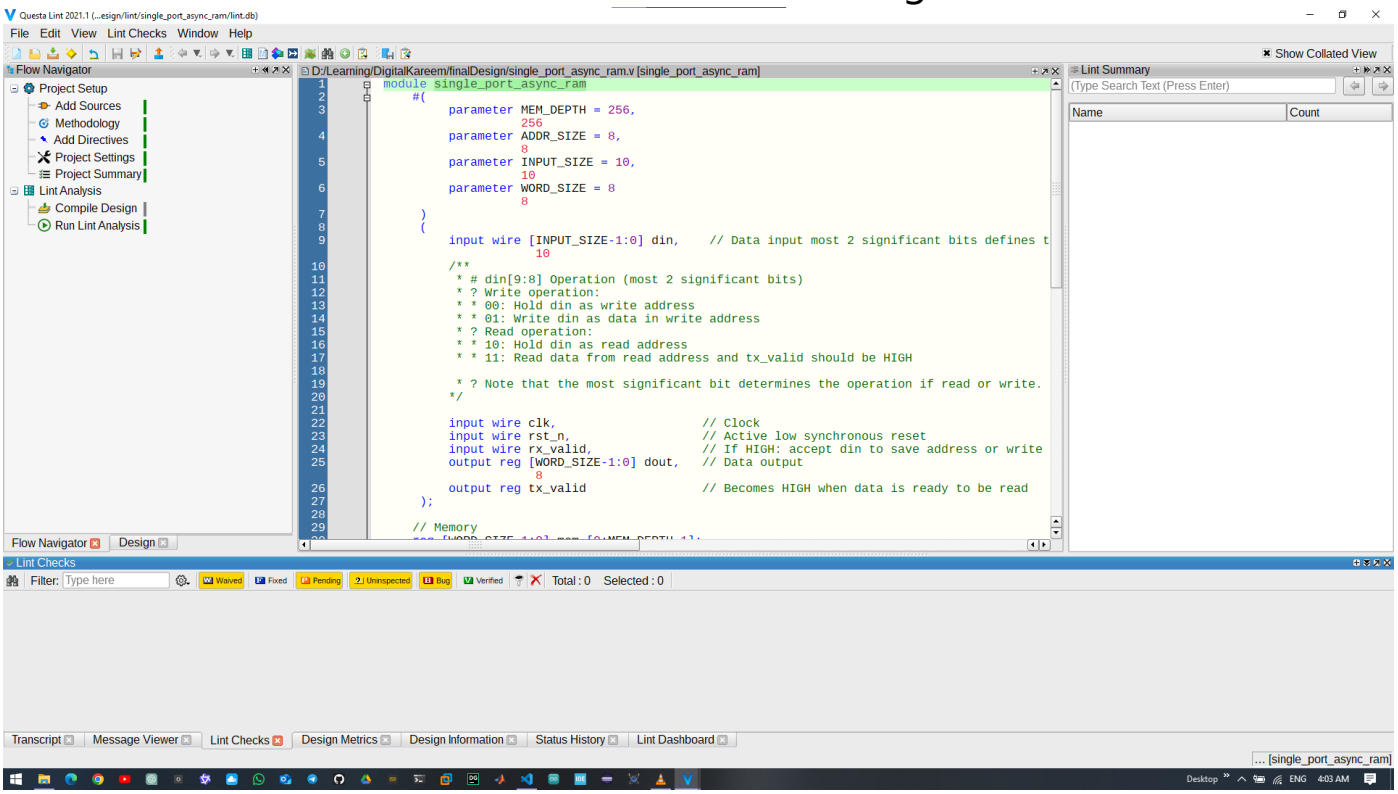
3. Sequential Encoding



11. Linting (Sequential Encoding)

Lint Checks (No Errors or Warnings)

- **Single Port RAM**
  - Check No Errors or Warnings

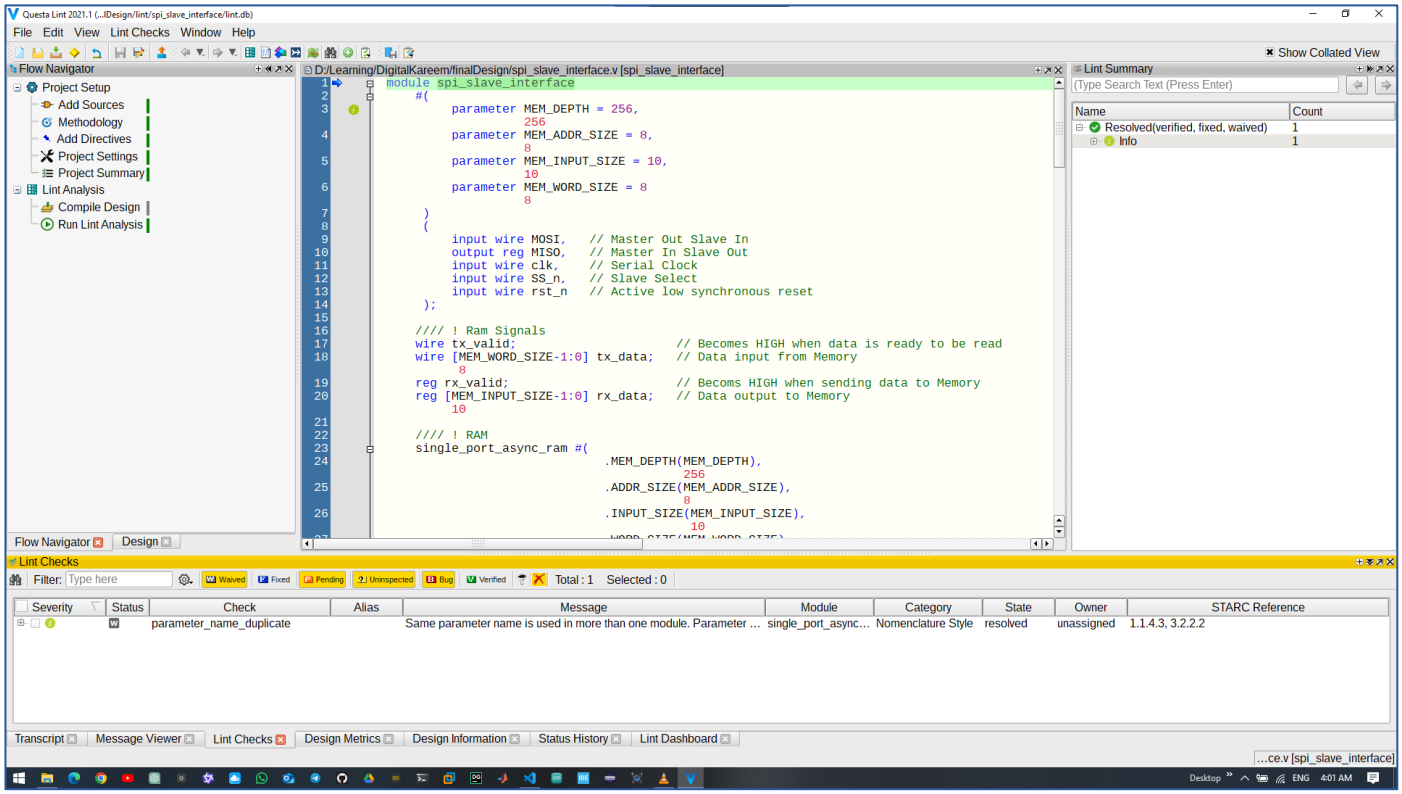


  - Summary

```
# Result Summary
# ------------------------------------------------------------
# Error (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Warning (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Info (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Resolved (0)
# ------------------------------------------------------------
#
# Generating Debug Information...
#
```

➢ SPI Interface

- Check No Errors or Warnings



- Summary

```
# Result Summary
# ------------------------------------------------------------
# Error (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Warning (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Info (1)
# ------------------------------------------------------------
#   parameter_name_duplicate                :1
#
# ------------------------------------------------------------
# Resolved (0)
# ------------------------------------------------------------
#
# Generating Debug Information...
#
```