ECE 6443, VLSI Systems & Architecture

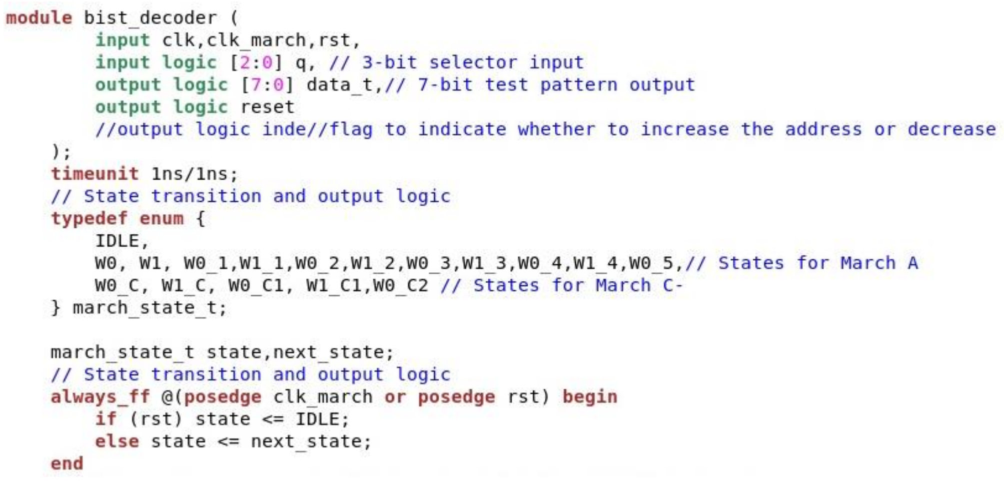
Project MBIST

hl5639 Hongrui Liu

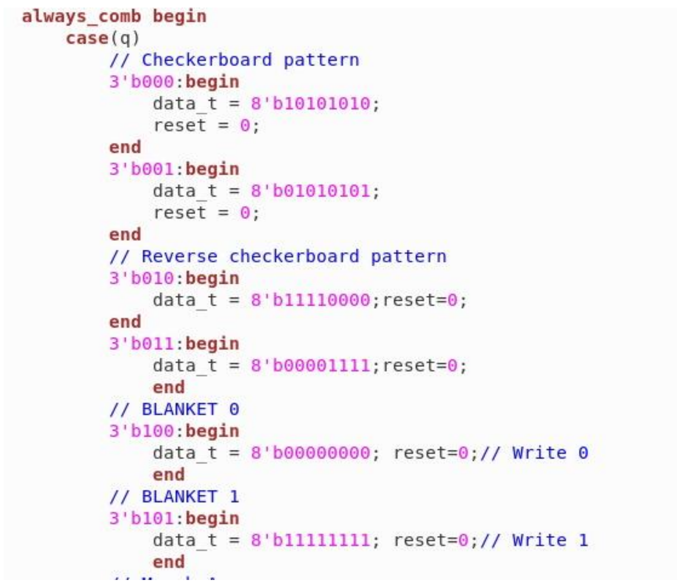
A. Verification of each component

1. Decoder

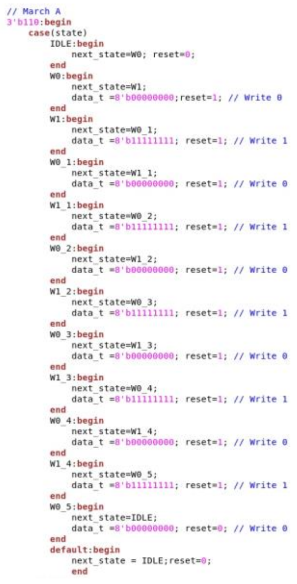
Code for Decoder and its testbench:



```
module bist_decoder (
        input clk,clk_march,rst,
        input logic [2:0] q, // 3-bit selector input
        output logic [7:0] data_t,// 7-bit test pattern output
        output logic reset
        //output logic inde//flag to indicate whether to increase the address or decrease
    );
    timeunit 1ns/1ns;
    // State transition and output logic
    typedef enum {
        IDLE,
        W0, W1, W0_1,W1_1,W0_2,W1_2,W0_3,W1_3,W0_4,W1_4,W0_5,// States for March A
        W0_C, W1_C, W0_C1, W1_C1,W0_C2 // States for March C-
    } march_state_t;

    march_state_t state,next_state;
    // State transition and output logic
    always_ff @(posedge clk_march or posedge rst) begin
        if (rst) state <= IDLE;
        else state <= next_state;
    end
```

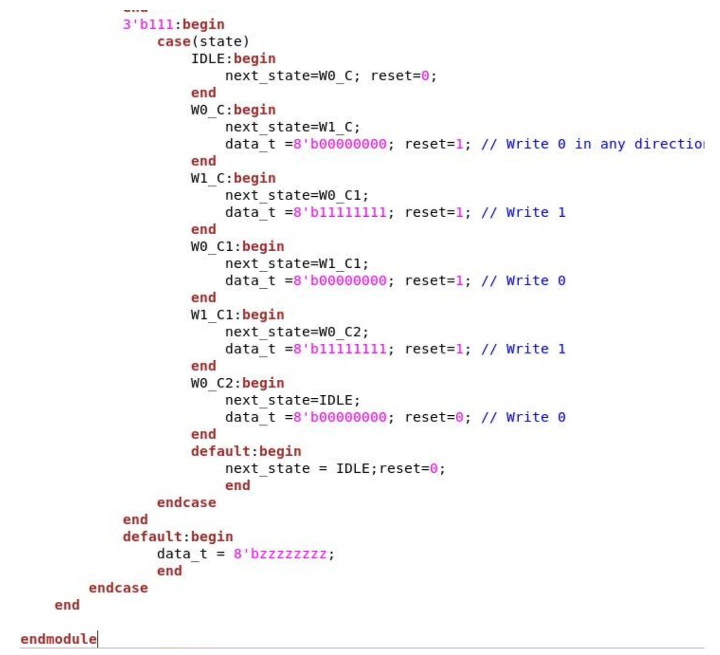


```
 always_comb begin
     case(q)
         // Checkerboard pattern
         3'b000:begin
             data_t = 8'b10101010;
             reset = 0;
         end
         3'b001:begin
             data_t = 8'b01010101;
             reset = 0;
         end
         // Reverse checkerboard pattern
         3'b010:begin
             data_t = 8'b11110000;reset=0;
         end
         3'b011:begin
             data_t = 8'b00001111;reset=0;
             end
         // BLANKET 0
         3'b100:begin
             data_t = 8'b00000000; reset=0;// Write 0
             end
         // BLANKET 1
         3'b101:begin
             data_t = 8'b11111111; reset=0;// Write 1
             end
```

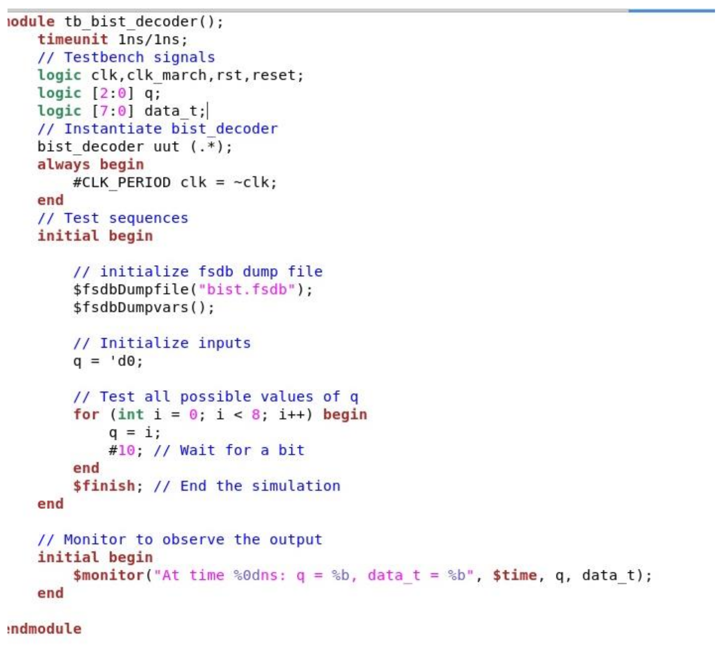


```
// March A
3'b110:begin
    case(state)
        IDLE:begin
            next_state=W0; reset=0;
        end
        W0:begin
            next_state=W1;
            data_t =8'b00000000;reset=1; // Write 0
        end
        W1:begin
            next_state=W0_1;
            data_t =8'b11111111; reset=1; // Write 1
        end
        W0_1:begin
            next_state=W1_1;
            data_t =8'b00000000; reset=1; // Write 0
        end
        W1_1:begin
            next_state=W0_2;
            data_t =8'b11111111; reset=1; // Write 1
        end
        W0_2:begin
            next_state=W1_2;
            data_t =8'b00000000; reset=1; // Write 0
        end
        W1_2:begin
            next_state=W0_3;
            data_t =8'b11111111; reset=1; // Write 1
        end
        W0_3:begin
            next_state=W1_3;
            data_t =8'b00000000; reset=1; // Write 0
        end
        W1_3:begin
            next_state=W0_4;
            data_t =8'b11111111; reset=1; // Write 1
        end
        W0_4:begin
            next_state=W1_4;
            data_t =8'b00000000; reset=1; // Write 0
        end
        W1_4:begin
            next_state=W0_5;
            data_t =8'b11111111; reset=1; // Write 1
        end
        W0_5:begin
            next_state=IDLE;
            data_t =8'b00000000; reset=0; // Write 0
        end
        default:begin
            next_state = IDLE;reset=0;
            end
```

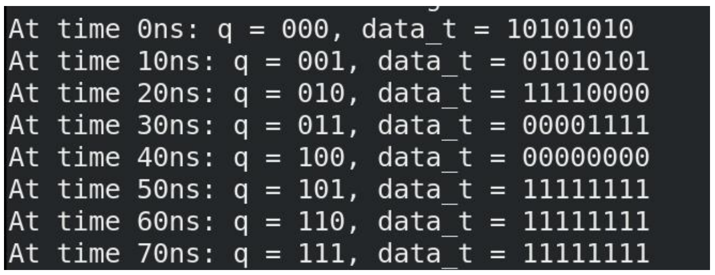
It can be found that there is a state machine created for March pattern to implement certain write 1s and 0s alternatively. Also, **reset** is introduced as a signal to indicates that it is in the March pattern. This is how it works: For other patterns, reset is 0, it mains that it normally counting up. When it reaches March patterns, it set to 1 which the counter is received it as a input signal that it counter in the state from 8'b0000000 to 8'hff with other bits of q is still remained. This ensures that the pattern is still remains in March C- and March A, where the counter is counting.



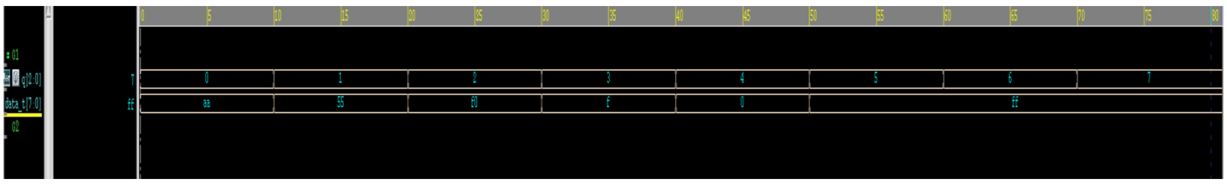
```
            3'b111:begin
                case(state)
                    IDLE:begin
                        next_state=W0_C; reset=0;
                    end
                    W0_C:begin
                        next_state=W1_C;
                        data_t =8'b00000000; reset=1; // Write 0 in any directio
                    end
                    W1_C:begin
                        next_state=W0_C1;
                        data_t =8'b11111111; reset=1; // Write 1
                    end
                    W0_C1:begin
                        next_state=W1_C1;
                        data_t =8'b00000000; reset=1; // Write 0
                    end
                    W1_C1:begin
                        next_state=W0_C2;
                        data_t =8'b11111111; reset=1; // Write 1
                    end
                    W0_C2:begin
                        next_state=IDLE;
                        data_t =8'b00000000; reset=0; // Write 0
                    end
                    default:begin
                        next_state = IDLE;reset=0;
                        end
                endcase
            end
            default:begin
                data_t = 8'bzzzzzzzz;
                end
        endcase
    end

endmodule
```



```
odule tb_bist_decoder();
    timeunit 1ns/1ns;
    // Testbench signals
    logic clk,clk_march,rst,reset;
    logic [2:0] q;
    logic [7:0] data_t;
    // Instantiate bist_decoder
    bist_decoder uut (.*);
    always begin
        #CLK_PERIOD clk = ~clk;
    end
    // Test sequences
    initial begin

        // initialize fsdb dump file
        $fsdbDumpfile("bist.fsdb");
        $fsdbDumpvars();

        // Initialize inputs
        q = 'd0;

        // Test all possible values of q
        for (int i = 0; i < 8; i++) begin
            q = i;
            #10; // Wait for a bit
        end
        $finish; // End the simulation
    end

    // Monitor to observe the output
    initial begin
        $monitor("At time %0dns: q = %b, data_t = %b", $time, q, data_t);
    end

ndmodule
```

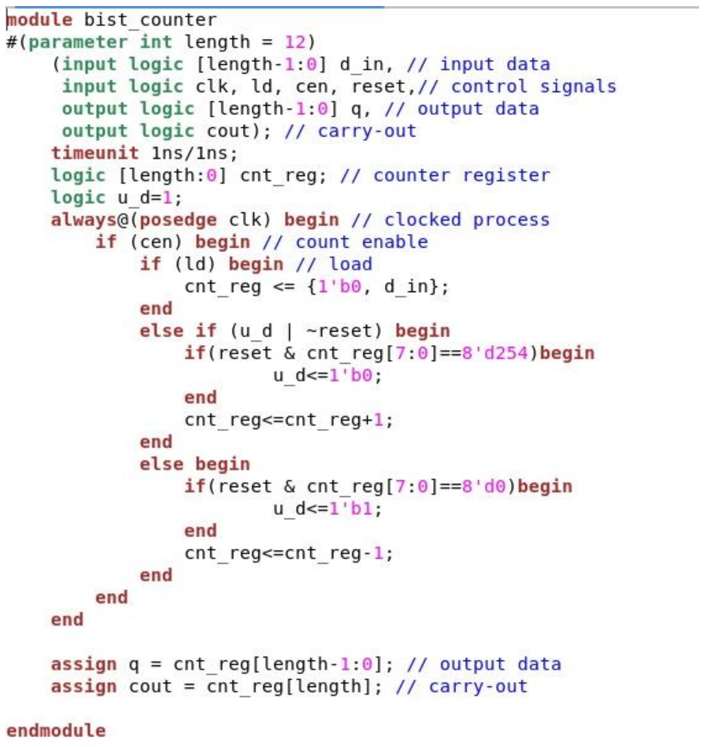


```
At time 0ns: q = 000, data_t = 10101010
At time 10ns: q = 001, data_t = 01010101
At time 20ns: q = 010, data_t = 11110000
At time 30ns: q = 011, data_t = 00001111
At time 40ns: q = 100, data_t = 00000000
At time 50ns: q = 101, data_t = 11111111
At time 60ns: q = 110, data_t = 11111111
At time 70ns: q = 111, data_t = 11111111
```



2. Counter

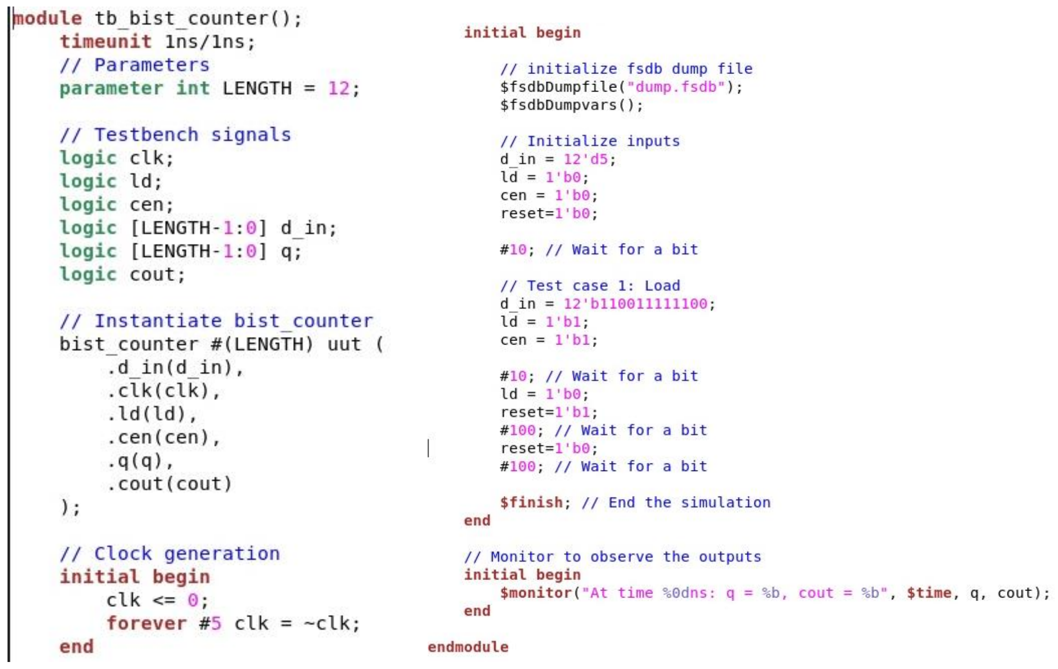
Code for counter:



```
module bist_counter
#(parameter int length = 12)
    (input logic [length-1:0] d_in, // input data
     input logic clk, ld, cen, reset,// control signals
     output logic [length-1:0] q, // output data
     output logic cout); // carry-out
    timeunit 1ns/1ns;
    logic [length:0] cnt_reg; // counter register
    logic u_d=1;
    always@(posedge clk) begin // clocked process
        if (cen) begin // count enable
            if (ld) begin // load
                cnt_reg <= {1'b0, d_in};
            end
            else if (u_d | ~reset) begin
                if(reset & cnt_reg[7:0]==8'd254)begin
                        u_d<=1'b0;
                end
                cnt_reg<=cnt_reg+1;
            end
            else begin
                if(reset & cnt_reg[7:0]==8'd0)begin
                        u_d<=1'b1;
                end
                cnt_reg<=cnt_reg-1;
            end
        end
    end

    assign q = cnt_reg[length-1:0]; // output data
    assign cout = cnt_reg[length]; // carry-out

endmodule
```

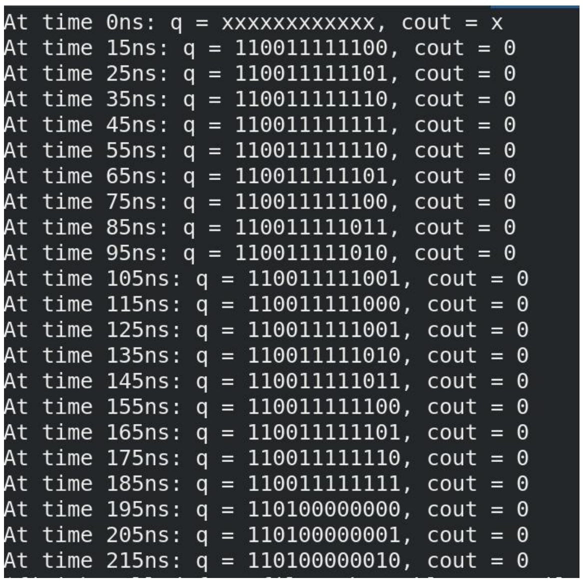
I changed some logic in the counter to make sure that it satisfied operating in different test patterns. The **reset** signal is a signal that came out from decoder as mentioned above. What specially for March A and March C- to enable them to loop over again and again writing 0 and 1s. As it reaches the top limit of the ram address the counter decreased, and if they reached the lower limit, it will increase. This makes sure that the q [11:9] stay in its status. For other patterns, it just normally increasing.

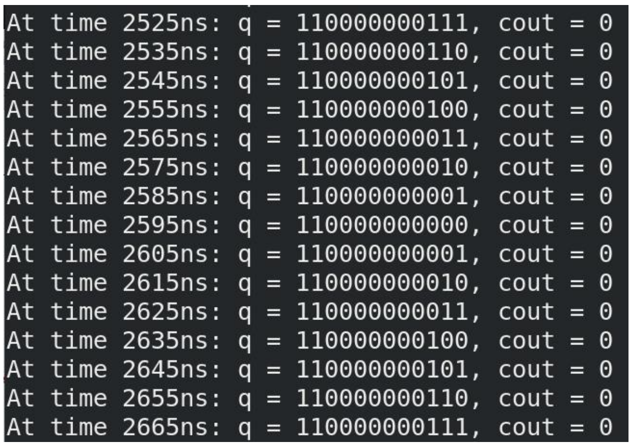
Testbench for the counter:



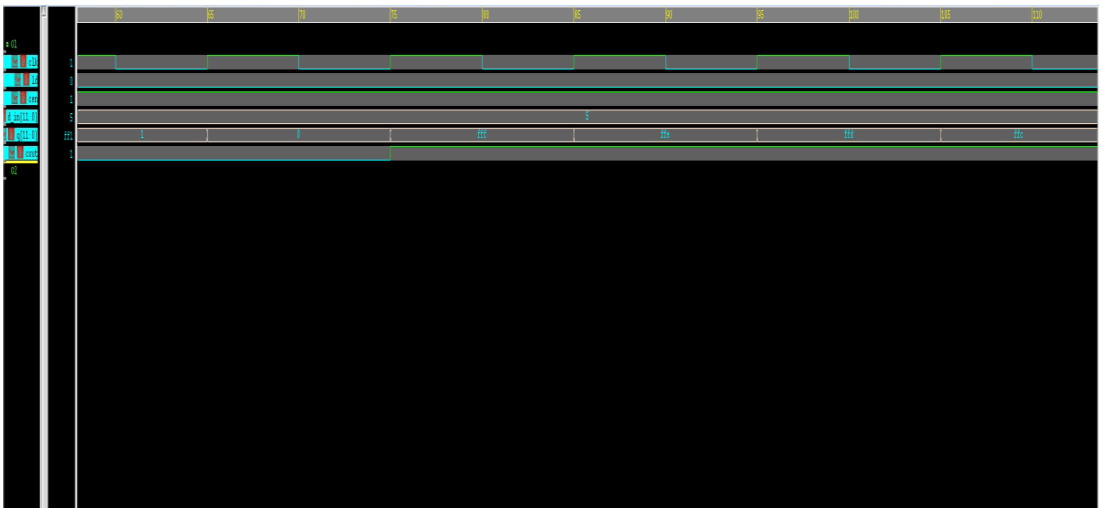
```
module tb_bist_counter();
    timeunit 1ns/1ns;
    // Parameters
    parameter int LENGTH = 12;

    // Testbench signals
    logic clk;
    logic ld;
    logic cen;
    logic [LENGTH-1:0] d_in;
    logic [LENGTH-1:0] q;
    logic cout;

    // Instantiate bist_counter
    bist_counter #(LENGTH) uut (
        .d_in(d_in),
        .clk(clk),
        .ld(ld),
        .cen(cen),
        .q(q),
        .cout(cout)
    );

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

    initial begin

        // initialize fsdb dump file
        $fsdbDumpfile("dump.fsdb");
        $fsdbDumpvars();

        // Initialize inputs
        d_in = 12'd5;
        ld = 1'b0;
        cen = 1'b0;
        reset=1'b0;

        #10; // Wait for a bit

        // Test case 1: Load
        d_in = 12'b110011111100;
        ld = 1'b1;
        cen = 1'b1;

        #10; // Wait for a bit
        ld = 1'b0;
        reset=1'b1;
        #100; // Wait for a bit
        reset=1'b0;
        #100; // Wait for a bit

        $finish; // End the simulation
    end

    // Monitor to observe the outputs
    initial begin
        $monitor("At time %0dns: q = %b, cout = %b", $time, q, cout);
    end

endmodule
```

Simulation output and Waveform:


```
At time 0ns: q = xxxxxxxxxxxx, cout = x
At time 15ns: q = 110011111100, cout = 0
At time 25ns: q = 110011111101, cout = 0
At time 35ns: q = 110011111110, cout = 0
At time 45ns: q = 110011111111, cout = 0
At time 55ns: q = 110011111110, cout = 0
At time 65ns: q = 110011111101, cout = 0
At time 75ns: q = 110011111100, cout = 0
At time 85ns: q = 110011111011, cout = 0
At time 95ns: q = 110011111010, cout = 0
At time 105ns: q = 110011111001, cout = 0
At time 115ns: q = 110011111000, cout = 0
At time 125ns: q = 110011111001, cout = 0
At time 135ns: q = 110011111010, cout = 0
At time 145ns: q = 110011111011, cout = 0
At time 155ns: q = 110011111100, cout = 0
At time 165ns: q = 110011111101, cout = 0
At time 175ns: q = 110011111110, cout = 0
At time 185ns: q = 110011111111, cout = 0
At time 195ns: q = 110100000000, cout = 0
At time 205ns: q = 110100000001, cout = 0
At time 215ns: q = 110100000010, cout = 0
```

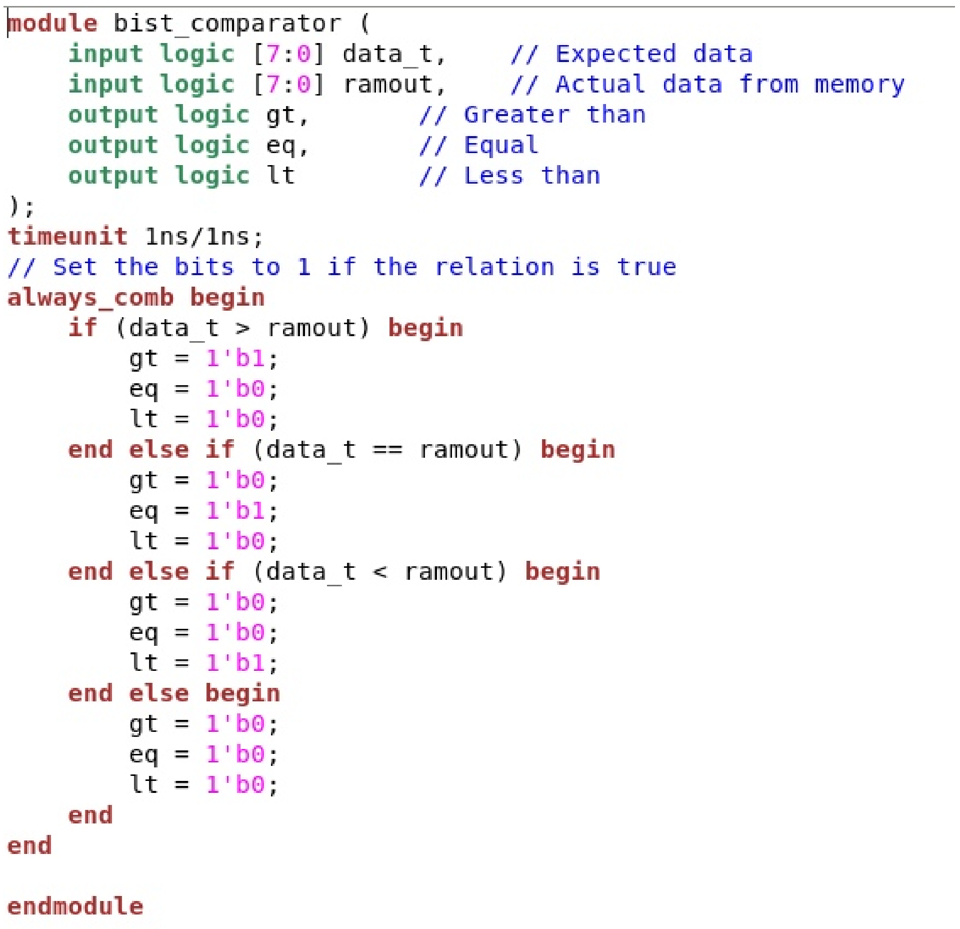

```
At time 2525ns: q = 110000000111, cout = 0
At time 2535ns: q = 110000000110, cout = 0
At time 2545ns: q = 110000000101, cout = 0
At time 2555ns: q = 110000000100, cout = 0
At time 2565ns: q = 110000000011, cout = 0
At time 2575ns: q = 110000000010, cout = 0
At time 2585ns: q = 110000000001, cout = 0
At time 2595ns: q = 110000000000, cout = 0
At time 2605ns: q = 110000000001, cout = 0
At time 2615ns: q = 110000000010, cout = 0
At time 2625ns: q = 110000000011, cout = 0
At time 2635ns: q = 110000000100, cout = 0
At time 2645ns: q = 110000000101, cout = 0
At time 2655ns: q = 110000000110, cout = 0
At time 2665ns: q = 110000000111, cout = 0
```

We can find that the counter reached 8’b11111111 in reset=1 mode which simulates the march pattern, it will decrease, and it will decrease until reaches 8’b00000000 and start to increase again within reset 1 mode. And when it changes to reset=0 mode, it will just increase normally.

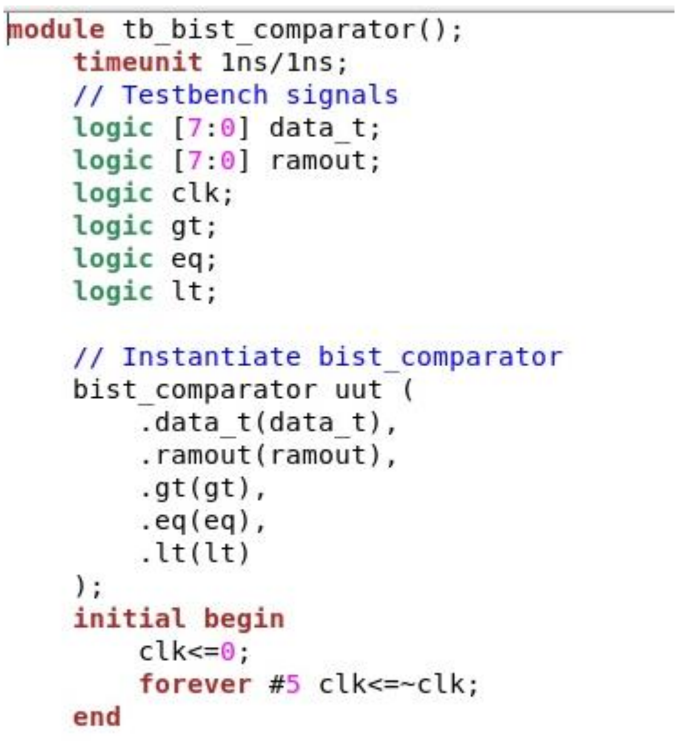


3. Comparator

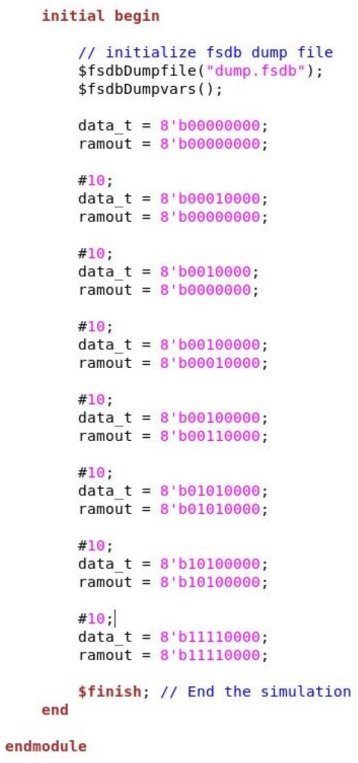
Code for Comparator and its testbench:



```
module bist_comparator (
    input logic [7:0] data_t,    // Expected data
    input logic [7:0] ramout,    // Actual data from memory
    output logic gt,        // Greater than
    output logic eq,        // Equal
    output logic lt         // Less than
);
timeunit 1ns/1ns;
// Set the bits to 1 if the relation is true
always_comb begin
    if (data_t > ramout) begin
        gt = 1'b1;
        eq = 1'b0;
        lt = 1'b0;
    end else if (data_t == ramout) begin
        gt = 1'b0;
        eq = 1'b1;
        lt = 1'b0;
    end else if (data_t < ramout) begin
        gt = 1'b0;
        eq = 1'b0;
        lt = 1'b1;
    end else begin
        gt = 1'b0;
        eq = 1'b0;
        lt = 1'b0;
    end
end

endmodule
```

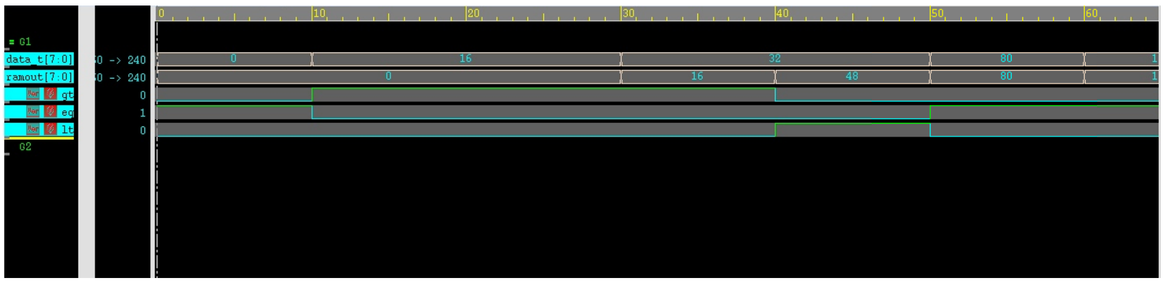


```
module tb_bist_comparator();
    timeunit 1ns/1ns;
    // Testbench signals
    logic [7:0] data_t;
    logic [7:0] ramout;
    logic clk;
    logic gt;
    logic eq;
    logic lt;

    // Instantiate bist_comparator
    bist_comparator uut (
        .data_t(data_t),
        .ramout(ramout),
        .gt(gt),
        .eq(eq),
        .lt(lt)
    );
    initial begin
        clk<=0;
        forever #5 clk<=~clk;
    end
```



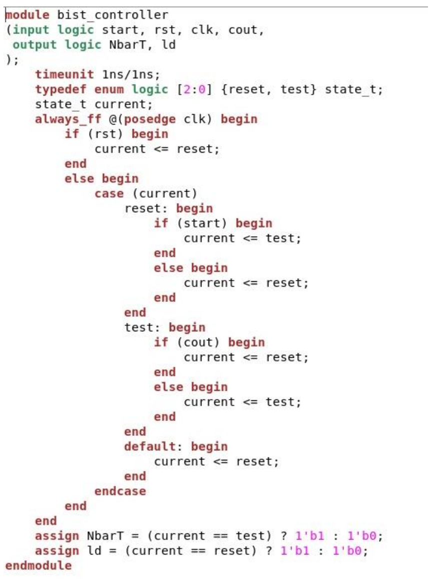
```
    initial begin

        // initialize fsdb dump file
        $fsdbDumpfile("dump.fsdb");
        $fsdbDumpvars();

        data_t = 8'b00000000;
        ramout = 8'b00000000;

        #10;
        data_t = 8'b00010000;
        ramout = 8'b00000000;

        #10;
        data_t = 8'b0010000;
        ramout = 8'b0000000;

        #10;
        data_t = 8'b00100000;
        ramout = 8'b00010000;

        #10;
        data_t = 8'b00100000;
        ramout = 8'b00110000;

        #10;
        data_t = 8'b01010000;
        ramout = 8'b01010000;

        #10;
        data_t = 8'b10100000;
        ramout = 8'b10100000;

        #10;
        data_t = 8'b11110000;
        ramout = 8'b11110000;

        $finish; // End the simulation
    end

endmodule
```

Simulation output and Waveform:

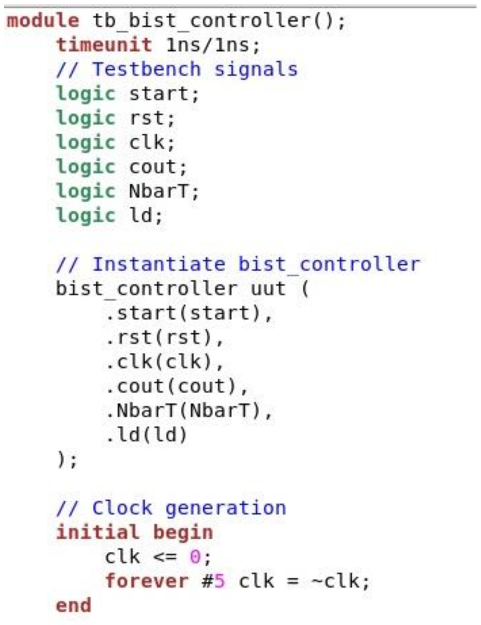


4. Controller

Code for Controller and its testbench:



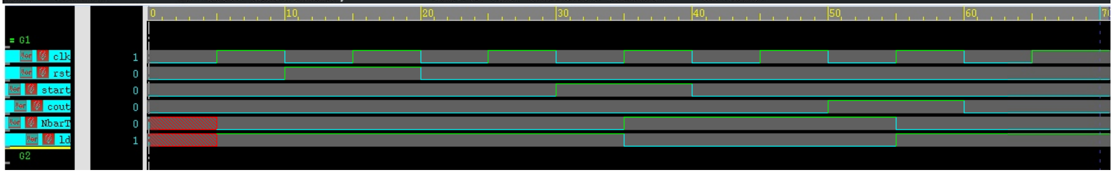
```
module bist_controller
(input logic start, rst, clk, cout,
 output logic NbarT, ld
);
    timeunit 1ns/1ns;
    typedef enum logic [2:0] {reset, test} state_t;
    state_t current;
    always_ff @(posedge clk) begin
        if (rst) begin
            current <= reset;
        end
        else begin
            case (current)
                reset: begin
                    if (start) begin
                        current <= test;
                    end
                    else begin
                        current <= reset;
                    end
                end
                test: begin
                    if (cout) begin
                        current <= reset;
                    end
                    else begin
                        current <= test;
                    end
                end
                default: begin
                    current <= reset;
                end
            endcase
        end
    end
    assign NbarT = (current == test) ? 1'b1 : 1'b0;
    assign ld = (current == reset) ? 1'b1 : 1'b0;
endmodule
```



```
module tb_bist_controller();
    timeunit 1ns/1ns;
    // Testbench signals
    logic start;
    logic rst;
    logic clk;
    logic cout;
    logic NbarT;
    logic ld;

    // Instantiate bist_controller
    bist_controller uut (
        .start(start),
        .rst(rst),
        .clk(clk),
        .cout(cout),
        .NbarT(NbarT),
        .ld(ld)
    );

    // Clock generation
    initial begin
        clk <= 0;
        forever #5 clk = ~clk;
    end
    // Test sequences
    initial begin

        // initialize fsdb dump file
        $fsdbDumpfile("dump.fsdb");
        $fsdbDumpvars();

        // Initialize inputs
        start = 1'b0;
        rst = 1'b0;
        cout = 1'b0;

        #10; // Wait for a bit
        // Test case 1: Test reset
        rst = 1'b1;
        #10; // Wait for a bit
        rst = 1'b0;
        #10; // Wait for a bit

        // Test case 2: Test start
        start = 1'b1;
        #10; // Wait for a bit
        start = 1'b0;
        #10; // Wait for a bit

        // Test case 3: Test cout
        cout = 1'b1;
        #10; // Wait for a bit
        cout = 1'b0;
        #10; // Wait for a bit

        $finish; // End the simulation
    end

    // Monitor to observe the outputs
    initial begin
        $monitor("At time %0dns: NbarT = %b, ld = %b", $time, NbarT, ld);
    end

endmodule
```

Simulation output and Waveform:

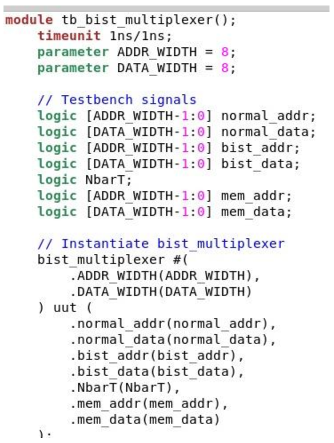
```
At time 0ns: NbarT = x, ld = x
At time 5ns: NbarT = 0, ld = 1
At time 35ns: NbarT = 1, ld = 0
At time 55ns: NbarT = 0, ld = 1
$finish called from file "/home/hl5639/verilog_files/project/tb_bist_controller.sv", line 58.
$finish at simulation time                70
           V C S   S i m u l a t i o n   R e p o r t
Time: 70 ns
CPU Time:      0.300 seconds;       Data structure size:   0.0Mb
```



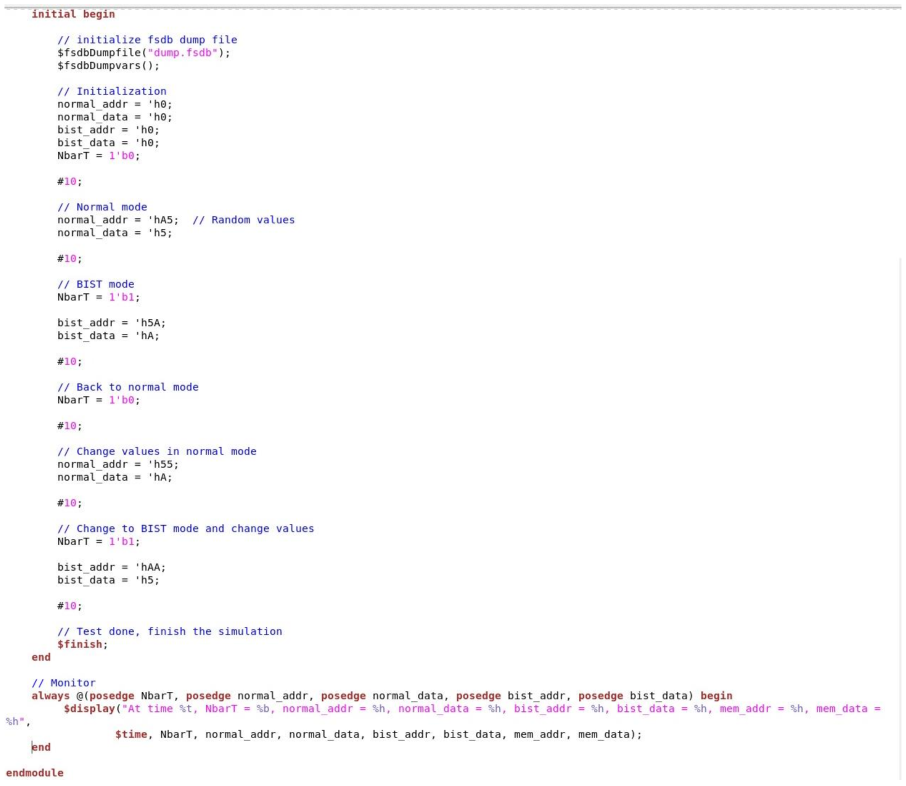
5. Multiplexer

Code for Multiplexer and its testbench:

```
module bist_multiplexer #(
    parameter int ADDR_WIDTH = 8,
    parameter int DATA_WIDTH = 8
) (
    input logic [ADDR_WIDTH-1:0] normal_addr,   // Input address for normal mode
    input logic [DATA_WIDTH-1:0] normal_data,   // Input data for normal mode
    input logic [ADDR_WIDTH-1:0] bist_addr,     // Input address for BIST mode
    input logic [DATA_WIDTH-1:0] bist_data,     // Input data for BIST mode
    input logic NbarT,                          // 0: normal mode, 1: test mode
    output logic [ADDR_WIDTH-1:0] mem_addr,     // Output address for memory
    output logic [DATA_WIDTH-1:0] mem_data      // Output data for memory
);
timeunit 1ns/1ns;
// Select between normal and BIST inputs based on the value of NbarT
assign mem_addr = (NbarT == 1'b0) ? normal_addr : bist_addr;
assign mem_data = (NbarT == 1'b0) ? normal_data : bist_data;

endmodule
```

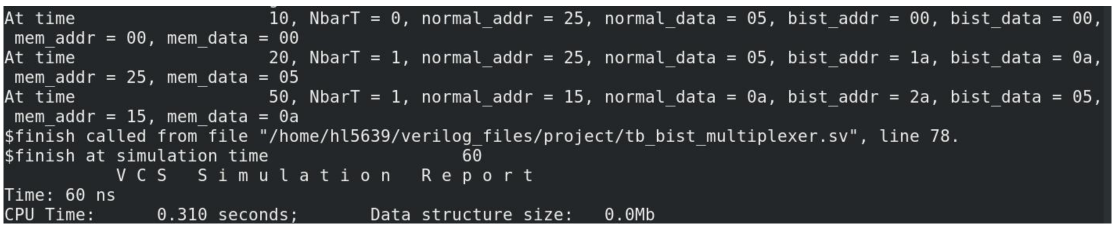


```
module tb_bist_multiplexer();
    timeunit 1ns/1ns;
    parameter ADDR_WIDTH = 8;
    parameter DATA_WIDTH = 8;

    // Testbench signals
    logic [ADDR_WIDTH-1:0] normal_addr;
    logic [DATA_WIDTH-1:0] normal_data;
    logic [ADDR_WIDTH-1:0] bist_addr;
    logic [DATA_WIDTH-1:0] bist_data;
    logic NbarT;
    logic [ADDR_WIDTH-1:0] mem_addr;
    logic [DATA_WIDTH-1:0] mem_data;

    // Instantiate bist_multiplexer
    bist_multiplexer #(
        .ADDR_WIDTH(ADDR_WIDTH),
        .DATA_WIDTH(DATA_WIDTH)
    ) uut (
        .normal_addr(normal_addr),
        .normal_data(normal_data),
        .bist_addr(bist_addr),
        .bist_data(bist_data),
        .NbarT(NbarT),
        .mem_addr(mem_addr),
        .mem_data(mem_data)
    );
```

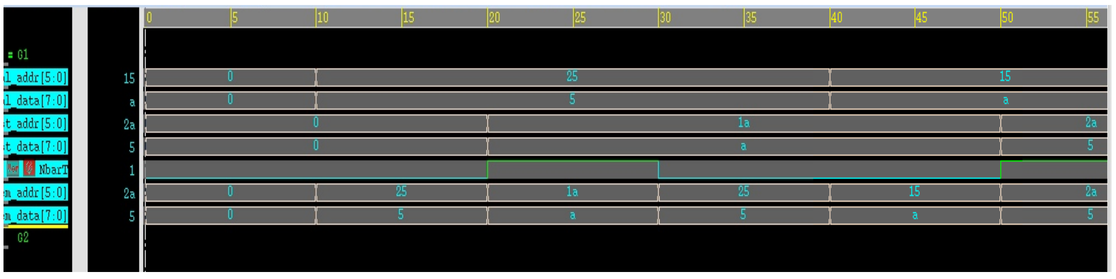


```
    initial begin

        // initialize fsdb dump file
        $fsdbDumpfile("dump.fsdb");
        $fsdbDumpvars();

        // Initialization
        normal_addr = 'h0;
        normal_data = 'h0;
        bist_addr = 'h0;
        bist_data = 'h0;
        NbarT = 1'b0;

        #10;

        // Normal mode
        normal_addr = 'hA5;  // Random values
        normal_data = 'h5;

        #10;

        // BIST mode
        NbarT = 1'b1;

        bist_addr = 'h5A;
        bist_data = 'hA;

        #10;

        // Back to normal mode
        NbarT = 1'b0;

        #10;

        // Change values in normal mode
        normal_addr = 'h55;
        normal_data = 'hA;

        #10;

        // Change to BIST mode and change values
        NbarT = 1'b1;

        bist_addr = 'hAA;
        bist_data = 'h5;

        #10;

        // Test done, finish the simulation
        $finish;
    end

    // Monitor
    always @(posedge NbarT, posedge normal_addr, posedge normal_data, posedge bist_addr, posedge bist_data) begin
         $display("At time %t, NbarT = %b, normal_addr = %h, normal_data = %h, bist_addr = %h, bist_data = %h, mem_addr = %h, mem_data =
%h",
                    $time, NbarT, normal_addr, normal_data, bist_addr, bist_data, mem_addr, mem_data);
    end

endmodule
```

Simulation output and Waveform:

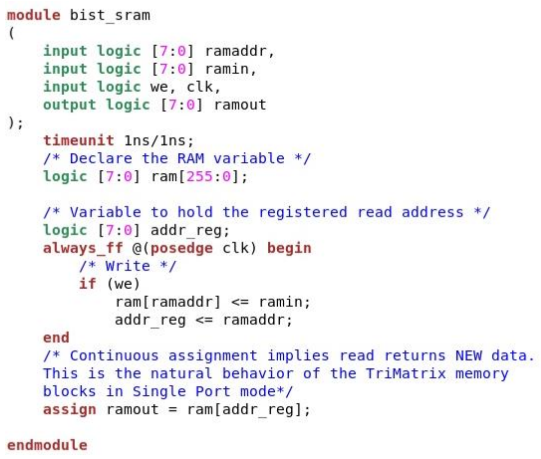


```
At time                   10, NbarT = 0, normal_addr = 25, normal_data = 05, bist_addr = 00, bist_data = 00,
 mem_addr = 00, mem_data = 00
At time                   20, NbarT = 1, normal_addr = 25, normal_data = 05, bist_addr = 1a, bist_data = 0a,
 mem_addr = 25, mem_data = 05
At time                   50, NbarT = 1, normal_addr = 15, normal_data = 0a, bist_addr = 2a, bist_data = 05,
 mem_addr = 15, mem_data = 0a
$finish called from file "/home/hl5639/verilog_files/project/tb_bist_multiplexer.sv", line 78.
$finish at simulation time                 60
           V C S   S i m u l a t i o n   R e p o r t
Time: 60 ns
CPU Time:      0.310 seconds;       Data structure size:   0.0Mb
```

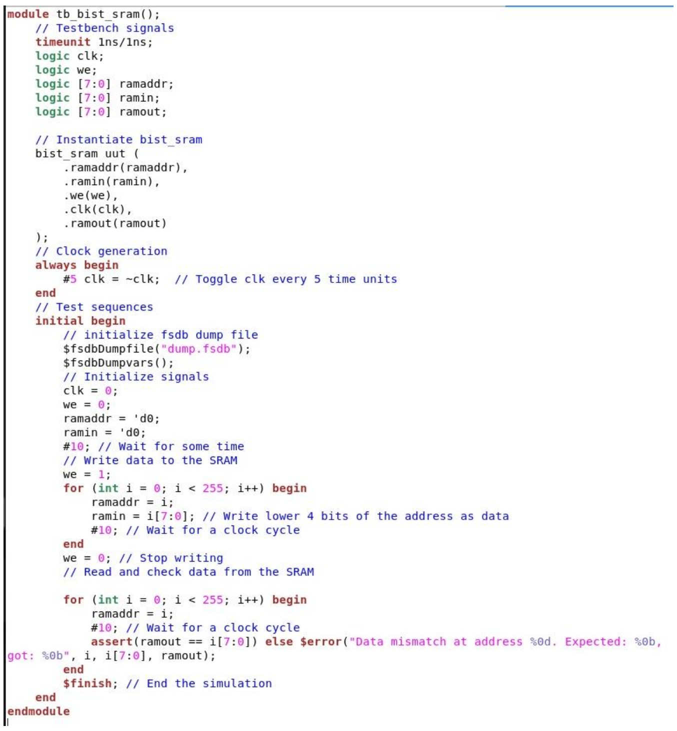


6. SRAM

Code for SRAM and its testbench:

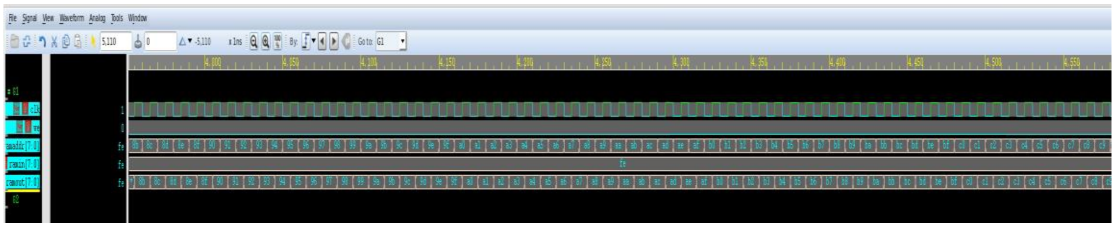


```
module bist_sram
(
    input logic [7:0] ramaddr,
    input logic [7:0] ramin,
    input logic we, clk,
    output logic [7:0] ramout
);
    timeunit 1ns/1ns;
    /* Declare the RAM variable */
    logic [7:0] ram[255:0];

    /* Variable to hold the registered read address */
    logic [7:0] addr_reg;
    always_ff @(posedge clk) begin
        /* Write */
        if (we)
            ram[ramaddr] <= ramin;
            addr_reg <= ramaddr;
    end
    /* Continuous assignment implies read returns NEW data.
    This is the natural behavior of the TriMatrix memory
    blocks in Single Port mode*/
    assign ramout = ram[addr_reg];

endmodule
```



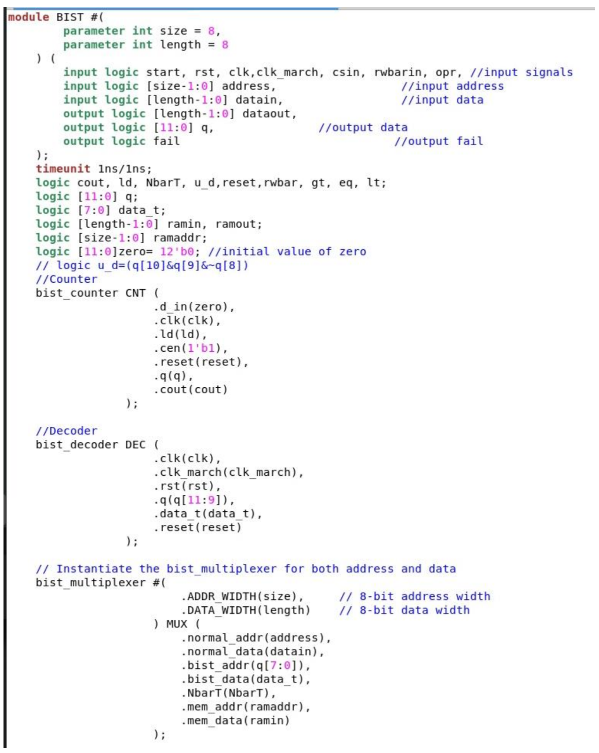
```
module tb_bist_sram();
    // Testbench signals
    timeunit 1ns/1ns;
    logic clk;
    logic we;
    logic [7:0] ramaddr;
    logic [7:0] ramin;
    logic [7:0] ramout;

    // Instantiate bist_sram
    bist_sram uut (
        .ramaddr(ramaddr),
        .ramin(ramin),
        .we(we),
        .clk(clk),
        .ramout(ramout)
    );
    // Clock generation
    always begin
        #5 clk = ~clk;  // Toggle clk every 5 time units
    end
    // Test sequences
    initial begin
        // initialize fsdb dump file
        $fsdbDumpfile("dump.fsdb");
        $fsdbDumpvars();
        // Initialize signals
        clk = 0;
        we = 0;
        ramaddr = 'd0;
        ramin = 'd0;
        #10; // Wait for some time
        // Write data to the SRAM
        we = 1;
        for (int i = 0; i < 255; i++) begin
            ramaddr = i;
            ramin = i[7:0]; // Write lower 4 bits of the address as data
            #10; // Wait for a clock cycle
        end
        we = 0; // Stop writing
        // Read and check data from the SRAM

        for (int i = 0; i < 255; i++) begin
            ramaddr = i;
            #10; // Wait for a clock cycle
            assert(ramout == i[7:0]) else $error("Data mismatch at address %0d. Expected: %0b,
got: %0b", i, i[7:0], ramout);
        end
        $finish; // End the simulation
    end
endmodule
```

Simulation output and Waveform:



7. Top module - March A and March C

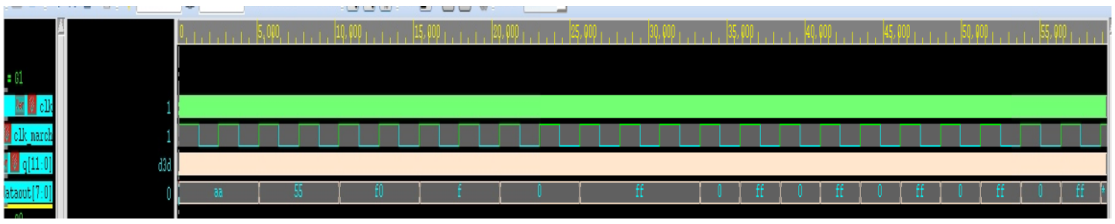
Top module RTL codes and testbench:

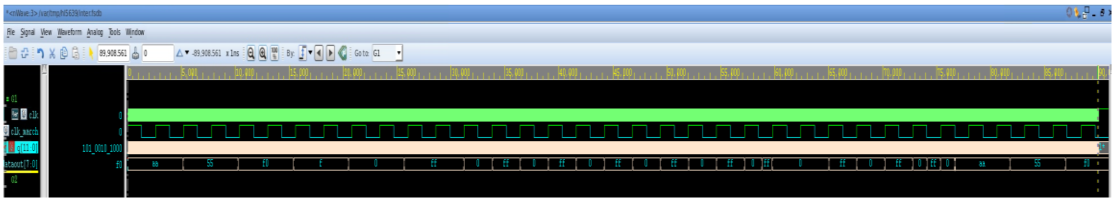


```
module BIST #(
        parameter int size = 8,
        parameter int length = 8
    ) (
        input logic start, rst, clk,clk_march, csin, rwbarin, opr, //input signals
        input logic [size-1:0] address,                  //input address
        input logic [length-1:0] datain,                 //input data
        output logic [length-1:0] dataout,
        output logic [11:0] q,                  //output data
        output logic fail                                //output fail
    );
    timeunit 1ns/1ns;
    logic cout, ld, NbarT, u_d,reset,rwbar, gt, eq, lt;
    logic [11:0] q;
    logic [7:0] data_t;
    logic [length-1:0] ramin, ramout;
    logic [size-1:0] ramaddr;
    logic [11:0]zero= 12'b0; //initial value of zero
    // logic u_d=(q[10]&q[9]&~q[8])
    //Counter
    bist_counter CNT (
                    .d_in(zero),
                    .clk(clk),
                    .ld(ld),
                    .cen(1'b1),
                    .reset(reset),
                    .q(q),
                    .cout(cout)
                );

    //Decoder
    bist_decoder DEC (
                    .clk(clk),
                    .clk_march(clk_march),
                    .rst(rst),
                    .q(q[11:9]),
                    .data_t(data_t),
                    .reset(reset)
                );

    // Instantiate the bist_multiplexer for both address and data
    bist_multiplexer #(
                        .ADDR_WIDTH(size),     // 8-bit address width
                        .DATA_WIDTH(length)    // 8-bit data width
                    ) MUX (
                        .normal_addr(address),
                        .normal_data(datain),
                        .bist_addr(q[7:0]),
                        .bist_data(data_t),
                        .NbarT(NbarT),
                        .mem_addr(ramaddr),
                        .mem_data(ramin)
                    );
```

```
    //BIST Controller
    bist_controller CNTRL (
                        .start(start),
                        .rst(rst),
                        .clk(clk),
                        .cout(cout),
                        .NbarT(NbarT),
                        .ld(ld)
                    );

    assign rwbar = (~NbarT) ? rwbarin : q[8];

    //RAM
    bist_sram MEM (
                    .ramaddr(ramaddr),
                    .ramin(ramin),
                    .we(~rwbar),
                    .clk(clk),
                    .ramout(ramout)
                );

    //Comparator
    bist_comparator CMP (
                        .data_t(data_t),
                        .ramout(ramout),
                        .eq(eq),
                        .gt(gt),
                        .lt(lt)
                    );

    always_ff @(posedge clk) begin
        if (NbarT && rwbar && opr && ~eq) begin
            fail <= 1'b1;
        end
        else begin
            fail <= 1'b0;
        end
    end

    assign dataout = ramout;

endmodule
```

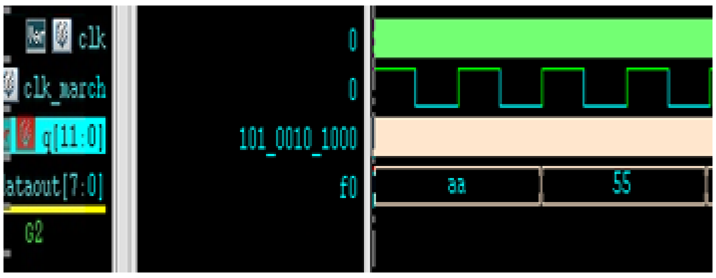
Final Waveform:



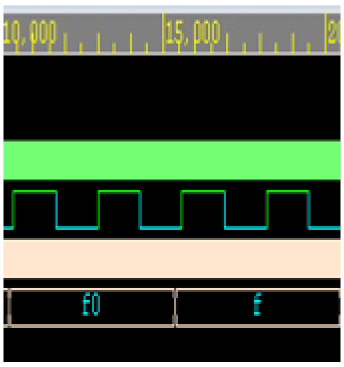


It can be found from the waveform that Our MBIST running automatically checking all the memory cells and read out the data in different patterns: Checkerboard stands for 8'haa and 8'h55, Reverse checkerboard stands for 8'df0 and 8'd0f, blanket 0 and blanket 1 stands for 8'h00 and 8'hff, and finally is the **March A and March C**- with alternatively writing 8'h00 and 8'hff which can be observed above on the waveform.

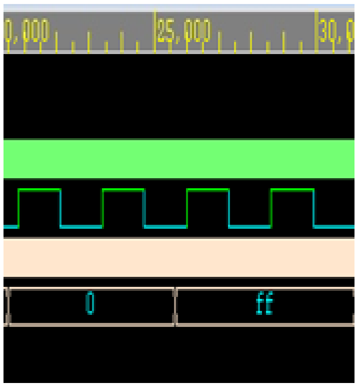
Checkboard:



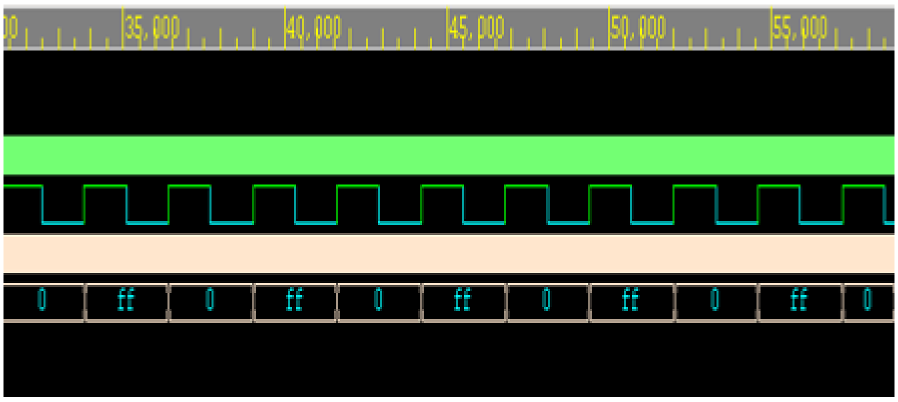
Reverse checkerboard:



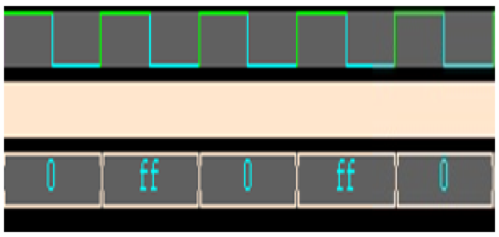
Blanket 0 and 1:



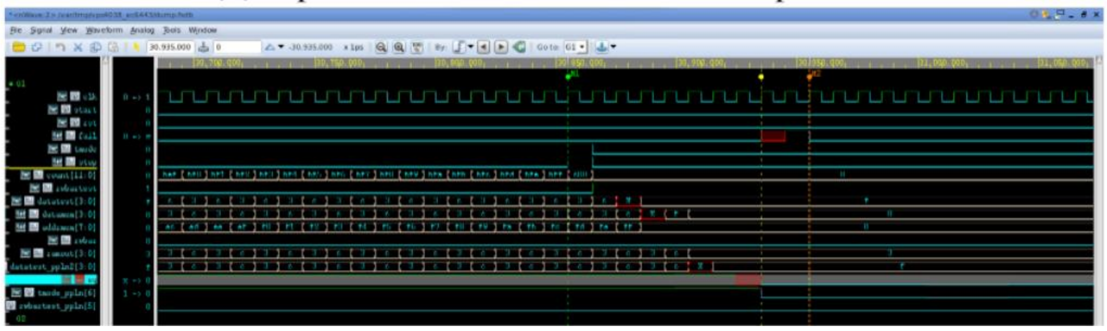
March A:



March C-:



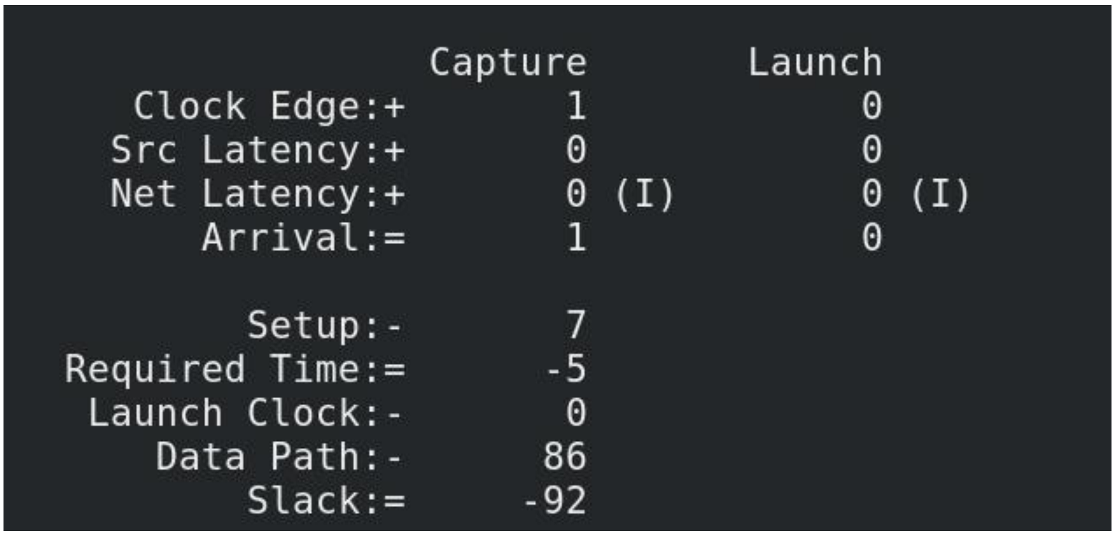
Also, we can check whether there is a fault generated and stop the test.



B. *Constraint and synthesis*

1. Counter

Checking the time violation:

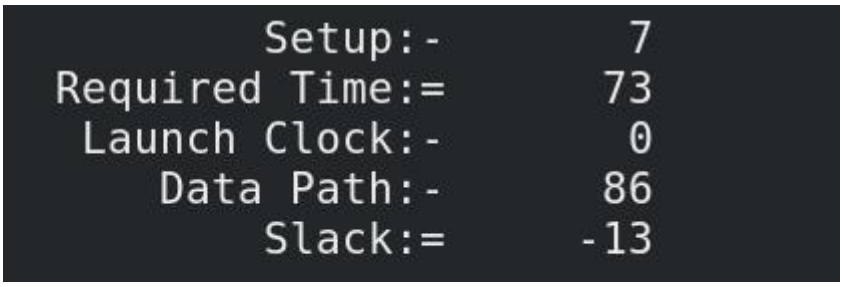


```
                    Capture         Launch
     Clock Edge:+         1              0
    Src Latency:+         0              0
    Net Latency:+         0 (I)          0 (I)
        Arrival:=         1              0

          Setup:-         7
  Required Time:=        -5
   Launch Clock:-         0
      Data Path:-        86
          Slack:=       -92
```

It can be found that the slack is much lower than 0, so we have to modify that to become a positive slack.

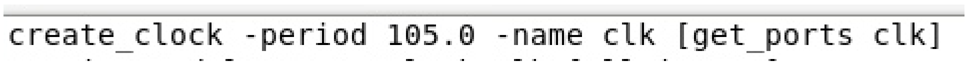
```
create_clock  -period 80 -name clk [get_ports clk]
set_input_delay 0.1 -clock clk [all_inputs]
set_output_delay 0.15 -clock clk [all_outputs]
set_load 0.1 [all_outputs]
set_max_fanout 1 [all_inputs]
set_fanout_load 8 [all_outputs]
set_clock_uncertainty .01 [all_clocks ]
set_clock_latency 0.01 -source [get_ports clk]
```

Increasing the time period of the clk which I do it for 80 to check the slack again:

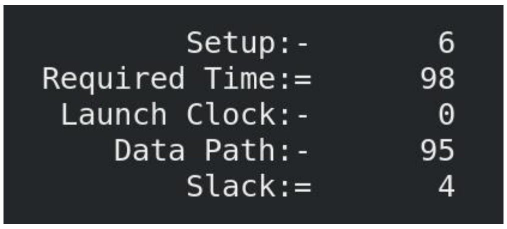


```
          Setup:-         7
  Required Time:=        73
   Launch Clock:-         0
      Data Path:-        86
          Slack:=       -13
```

Which still has a slight violation and then try to change the period to **100**:

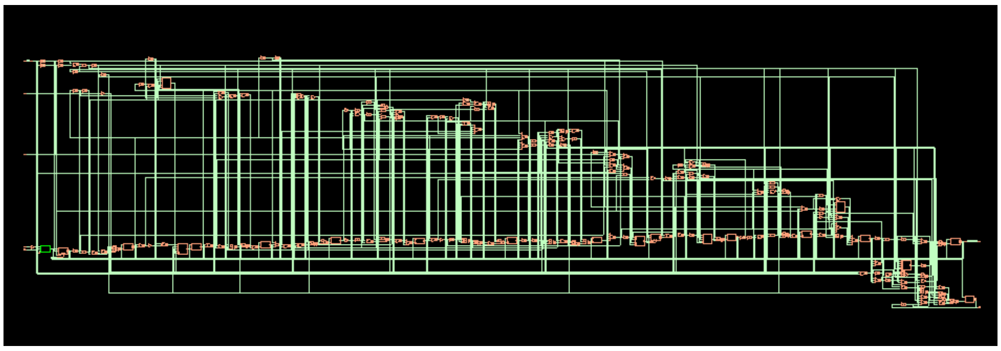


```
create_clock -period 105.0 -name clk [get_ports clk]
```



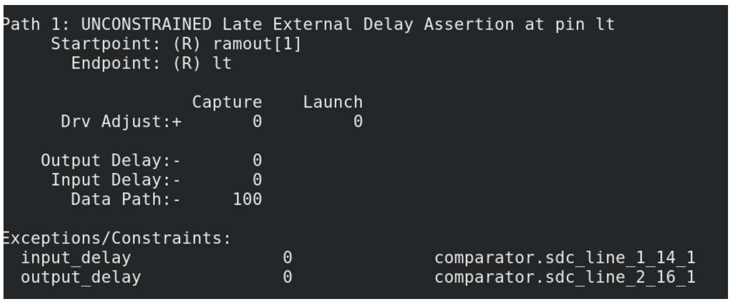
```
           Setup:-        6
   Required Time:=       98
    Launch Clock:-        0
       Data Path:-       95
          Slack:=         4
```

Finally, find that the slack is a positive number and with a small number. And we can synthesis the RTL level diagram:



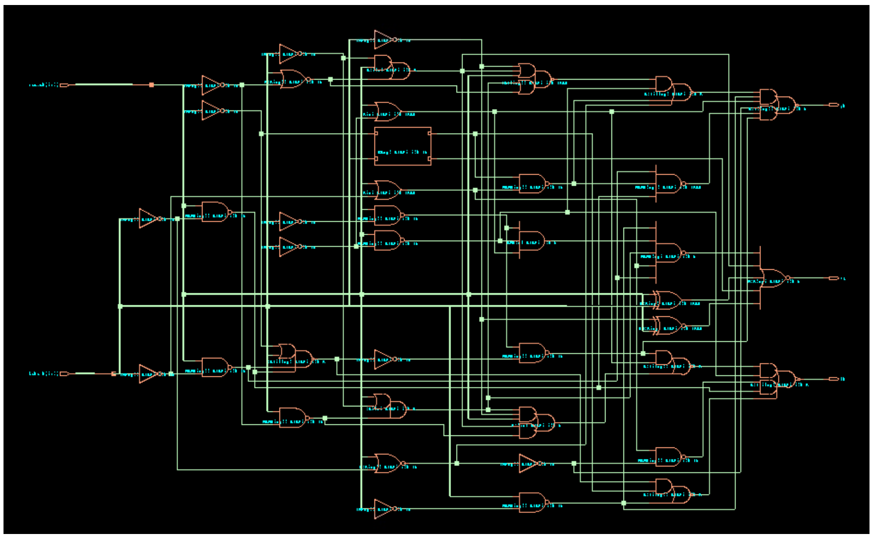
2. Comparator

Comparator is a combinational logic component which does not has a clk signal, so STA we only check the data path and input output delay:



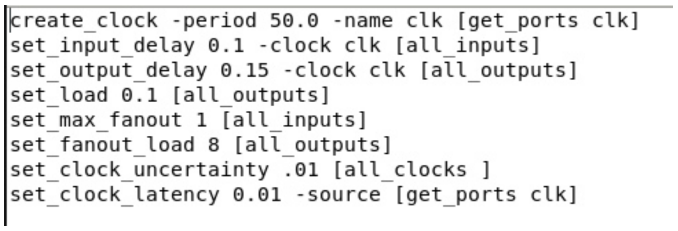
```
Path 1: UNCONSTRAINED Late External Delay Assertion at pin lt
     Startpoint: (R) ramout[1]
       Endpoint: (R) lt

                    Capture     Launch
      Drv Adjust:+        0          0

    Output Delay:-        0
     Input Delay:-        0
       Data Path:-      100

Exceptions/Constraints:
  input_delay                 0              comparator.sdc_line_1_14_1
  output_delay                0              comparator.sdc_line_2_16_1
```

Here's the synthesis:

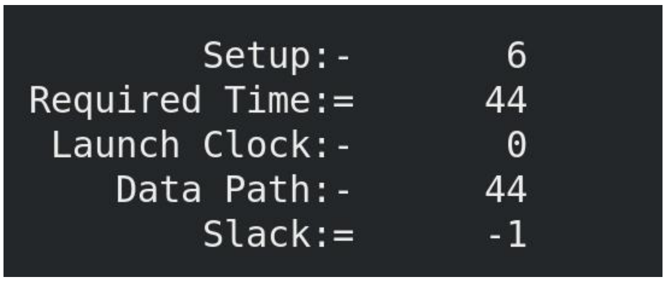


3. Controller

Checking the time violation, where given the clk period is 50:

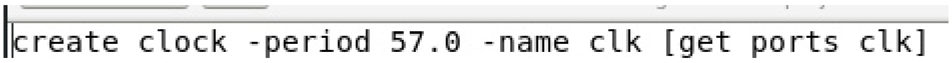


```
create_clock -period 50.0 -name clk [get_ports clk]
set_input_delay 0.1 -clock clk [all_inputs]
set_output_delay 0.15 -clock clk [all_outputs]
set_load 0.1 [all_outputs]
set_max_fanout 1 [all_inputs]
set_fanout_load 8 [all_outputs]
set_clock_uncertainty .01 [all_clocks ]
set_clock_latency 0.01 -source [get_ports clk]
```

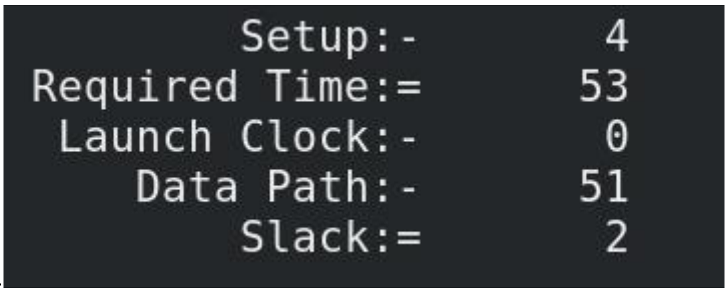


```
        Setup:-        6
Required Time:=       44
 Launch Clock:-        0
    Data Path:-       44
        Slack:=       -1
```

Increasing the period to 57 see whether the slack meet the requirement:

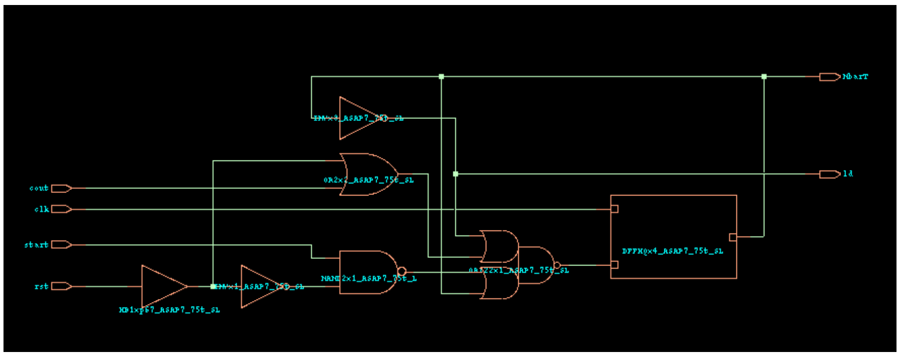


```
create clock -period 57.0 -name clk [get ports clk]
```



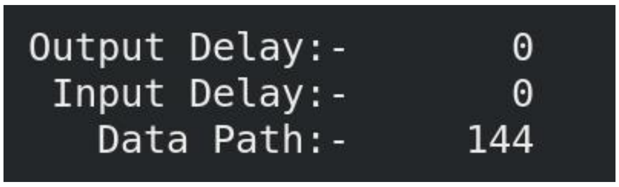
```
        Setup:-        4
Required Time:=       53
 Launch Clock:-        0
    Data Path:-       51
        Slack:=        2
```

Finally, find that the slack is a positive number and with a small number. And we can synthesis the RTL level diagram:



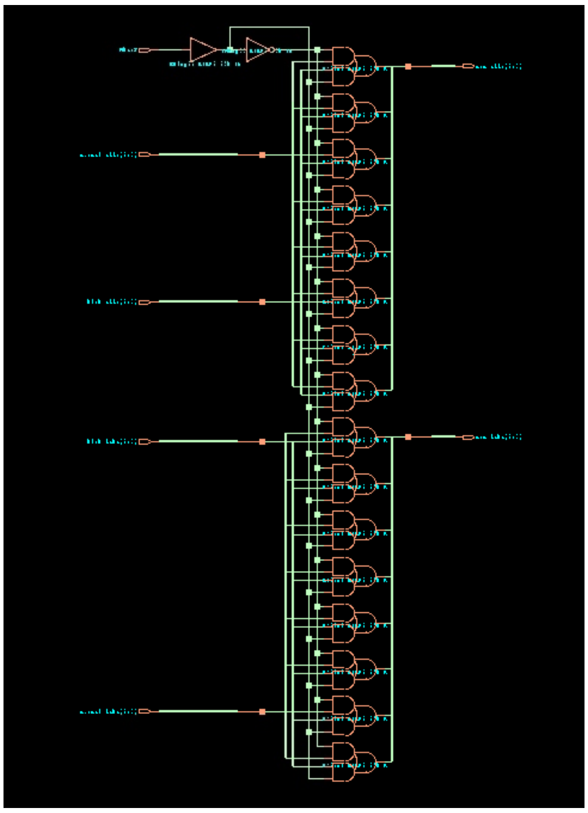
4. Multiplexer

Comparator is a combinational logic component which does not has a clk signal, so STA we only check the data path and input output delay:



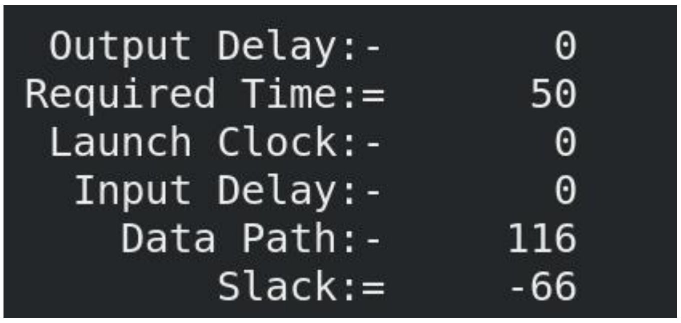
```
Output Delay:-        0
 Input Delay:-        0
   Data Path:-      144
```

Here's the synthesis:



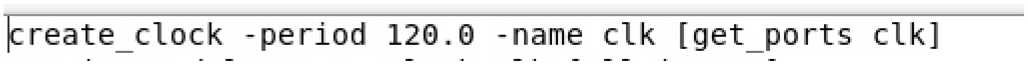
5. Decoder

Checking the time violation, where given the clk period is 50:

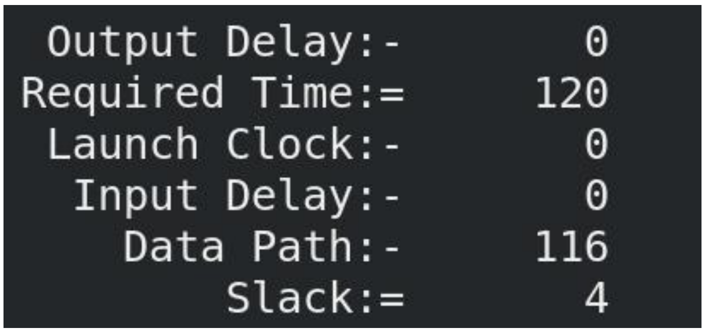


```
  Output Delay:-       0
Required Time:=       50
  Launch Clock:-       0
   Input Delay:-       0
     Data Path:-     116
         Slack:=     -66
```

Increasing the period to 120 see whether the slack meet the requirement:

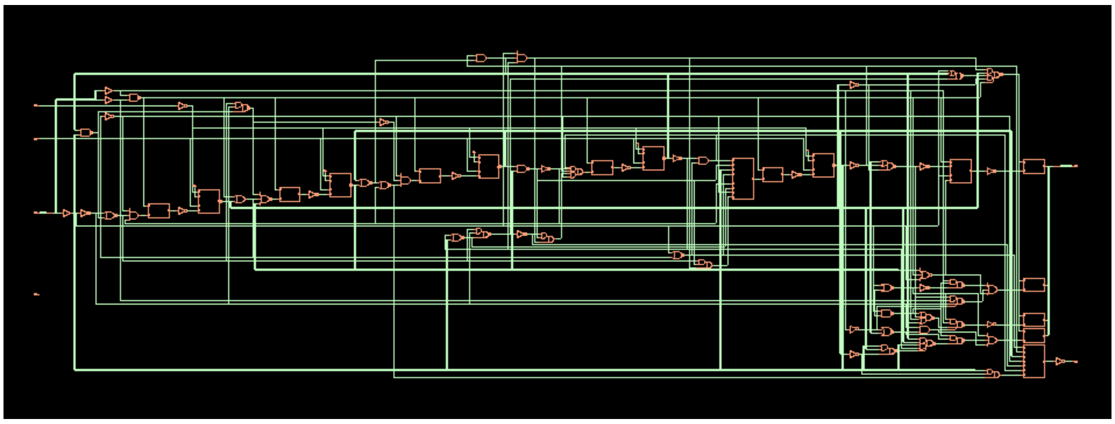


```
create_clock -period 120.0 -name clk [get_ports clk]
```



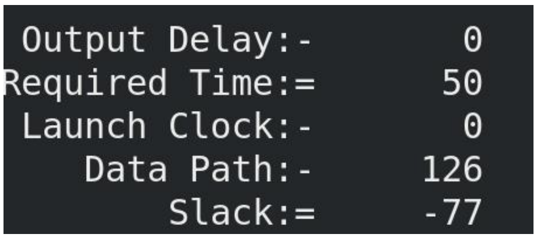
```
  Output Delay:-       0
Required Time:=      120
  Launch Clock:-       0
   Input Delay:-       0
     Data Path:-     116
         Slack:=       4
```

Here is the synthesis:



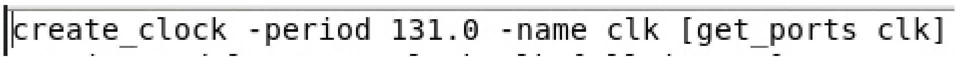
6. SRAM

Checking the time violation, where given the clk period is 50:

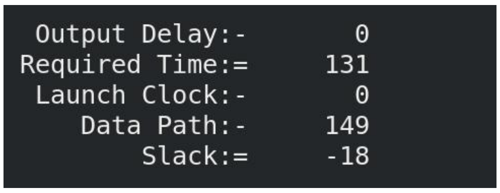


```
 Output Delay:-        0
Required Time:=       50
 Launch Clock:-        0
    Data Path:-      126
        Slack:=      -77
```

We can find that the clk period is 131, it still gets a -18 slack, so just increase a little more:

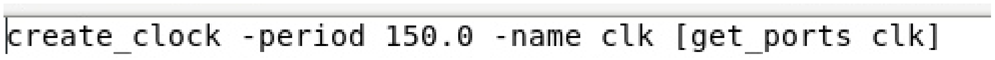


```
create_clock -period 131.0 -name clk [get_ports clk]
```

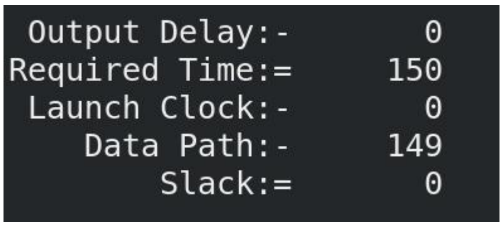


```
 Output Delay:-        0
Required Time:=      131
 Launch Clock:-        0
    Data Path:-      149
        Slack:=      -18
```

When it becomes 150, the slack become 0:

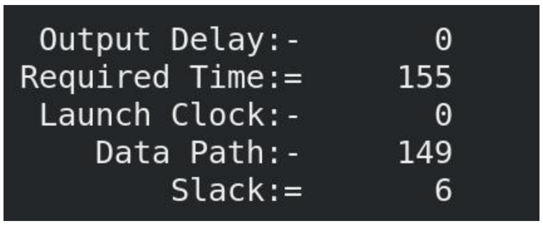


```
create_clock -period 150.0 -name clk [get_ports clk]
```



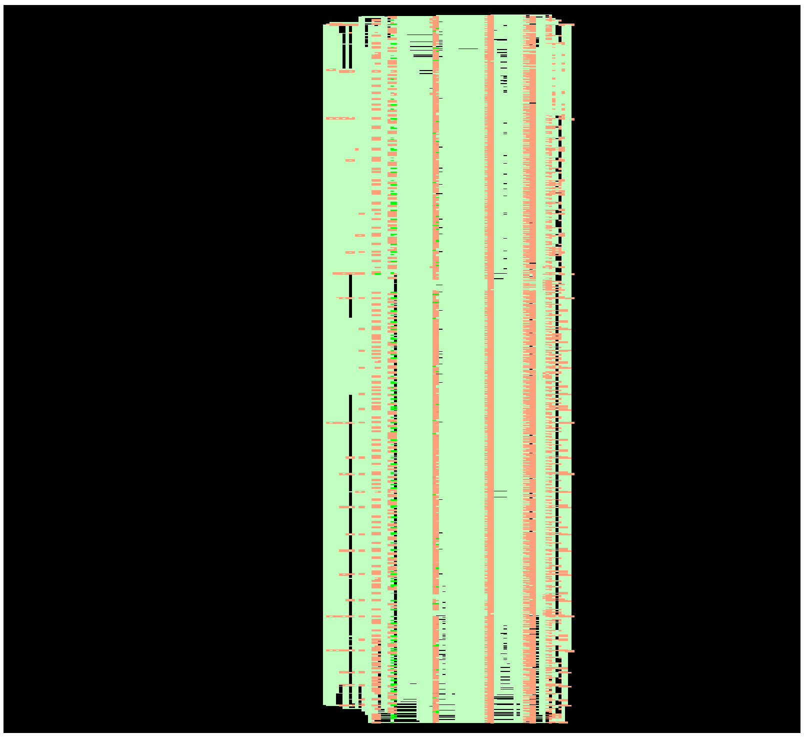
```
 Output Delay:-        0
Required Time:=      150
 Launch Clock:-        0
    Data Path:-      149
        Slack:=        0
```

Make it 155:



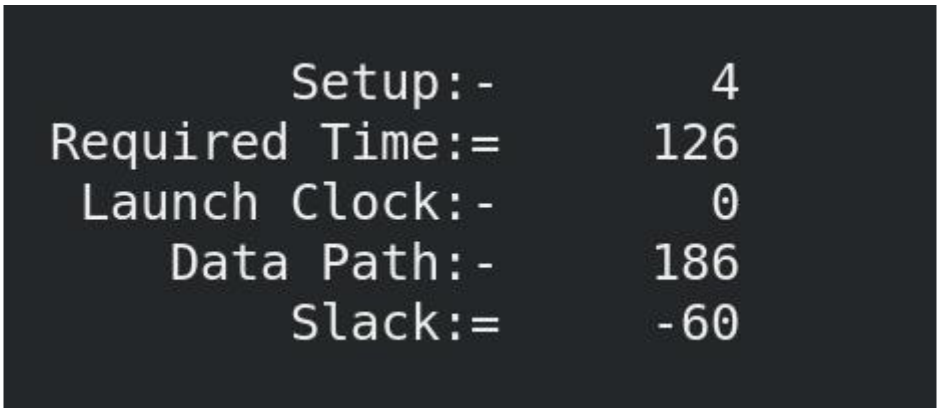
```
 Output Delay:-        0
Required Time:=      155
 Launch Clock:-        0
    Data Path:-      149
        Slack:=        6
```

Finally, find that the slack is a positive number and with a small number. And we can synthesis the RTL level diagram:



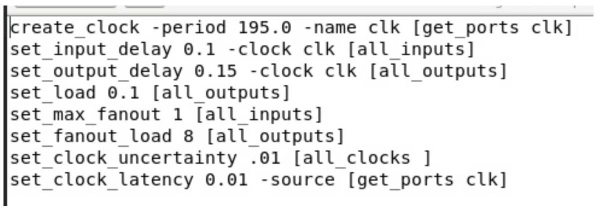
7. Top

Checking the time violation, where given the clk period is 130:

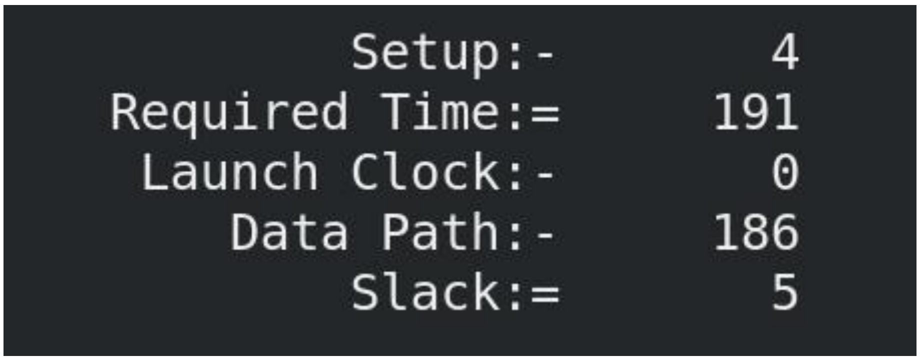


```
        Setup:-        4
Required Time:=      126
 Launch Clock:-        0
    Data Path:-      186
        Slack:=      -60
```

Then we increase the clk period to 200, it has a slack of 10, as we changed it to 195:



```
create_clock -period 195.0 -name clk [get_ports clk]
set_input_delay 0.1 -clock clk [all_inputs]
set_output_delay 0.15 -clock clk [all_outputs]
set_load 0.1 [all_outputs]
set_max_fanout 1 [all_inputs]
set_fanout_load 8 [all_outputs]
set_clock_uncertainty .01 [all_clocks ]
set_clock_latency 0.01 -source [get_ports clk]
```



```
        Setup:-        4
Required Time:=      191
 Launch Clock:-        0
    Data Path:-      186
        Slack:=        5
```

Finally, find that the slack is a positive number and with a small number. And we can synthesis the RTL level diagram: