Elec 527 Homework 2

Problem 1: Verilog and Standard Cells for Combinational Logic

1. **Verilog input module file**

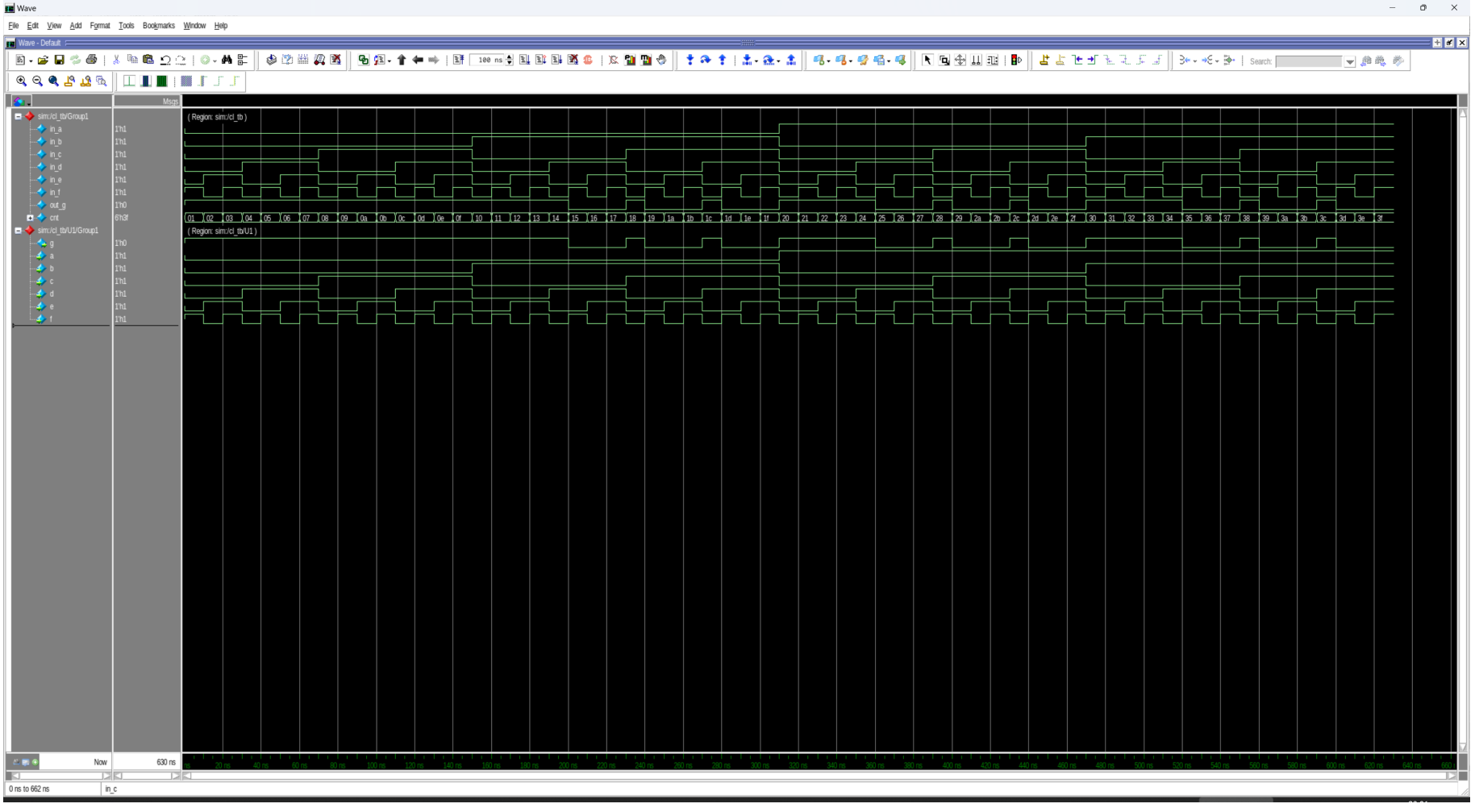
```
module cl (g, a, b, c, d, e, f);
// declare port signals
output g;
input a, b, c, d, e, f;
// logic statements
assign g = !((a|b)&(c|d)&(e|f)); //g is output, a, b, c, d, e and f are inputs
endmodule
```

2. **Verilog testbench file**

```
module cl_tb();

reg in_a, in_b, in_c, in_d, in_e, in_f;

wire out_g;

reg [5:0] cnt;

//create a Combinational Logic instance.
cl U1 (     .g (out_g),
            .a (in_a),
            .b (in_b),
            .c (in_c),
            .d (in_d),
            .e (in_e),
            .f (in_f)
          );

// Test circuit with truth table inputs and delay between value changes.
initial begin
  in_a = 0;
  in_b = 0;
  in_c = 0;
  in_d = 0;
  in_e = 0;
  in_f = 0;
  cnt = 6'b0;

  repeat (63) begin
    cnt = cnt + 6'b1;
    in_a = cnt[5];
    in_b = cnt[4];
    in_c = cnt[3];
    in_d = cnt[2];
    in_e = cnt[1];
    in_f = cnt[0];
    #10;
  end

    $dumpfile ("cl_tb.vcd");
    $dumpvars;

    $display("\t\t in_a,\t in_b,,\t in_c,\t in_d,\t in_e,\t in_f \t out_g");
$stop;
  end

endmodule
```

3. **Modelsim output plot showing correct function both before and after synthesis**

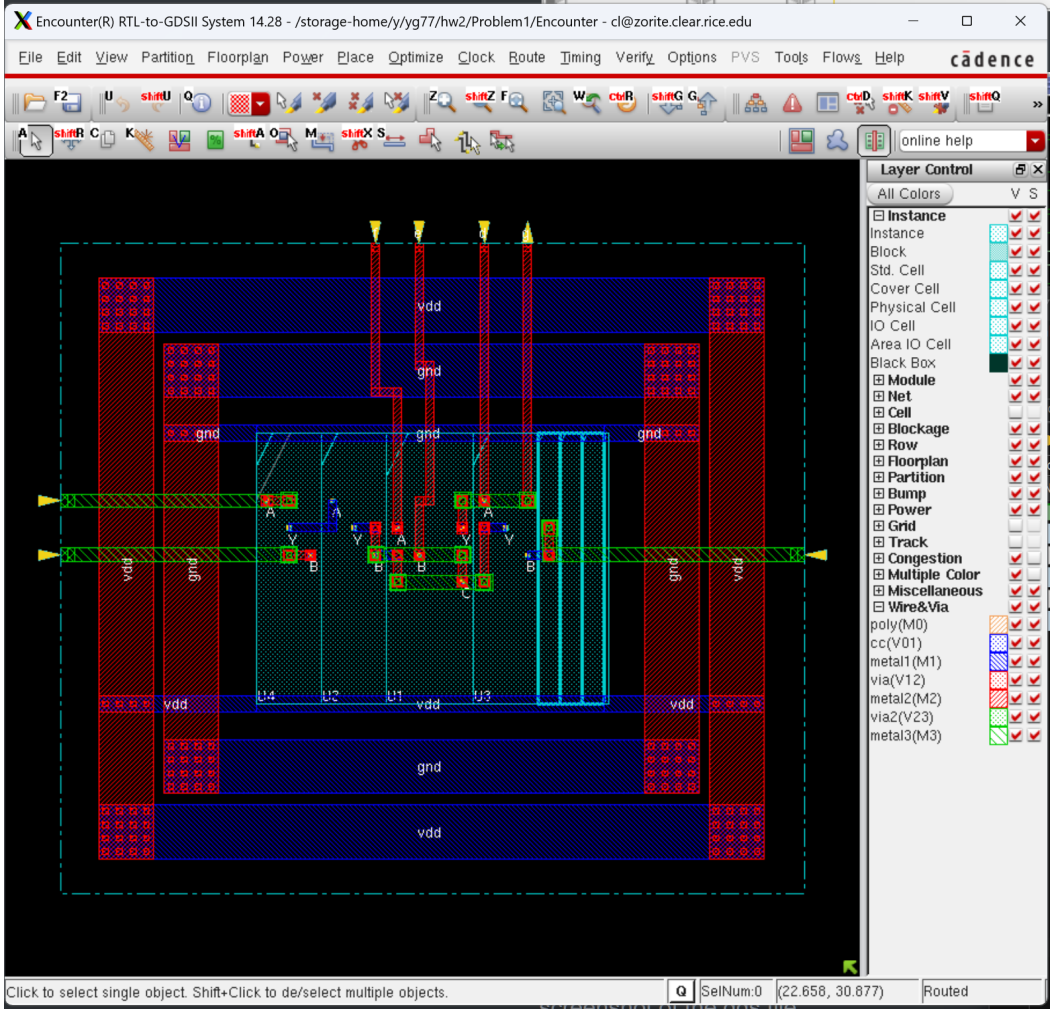
   before synthesis



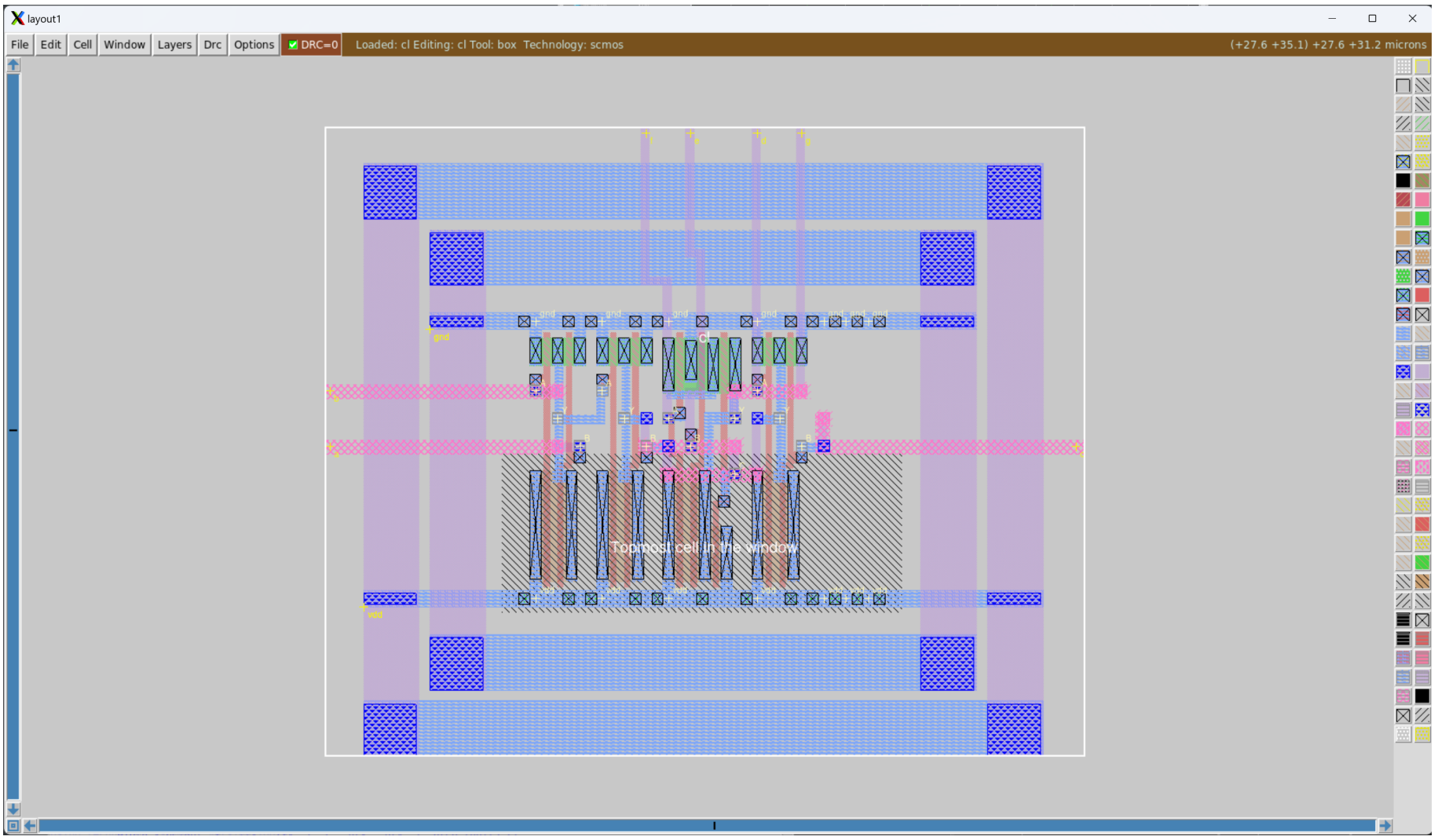
after synthesis

4. **Sceenshot from Encounter and from Magic - upload magic files**

screenshot from Encounter



screenshot of magic window



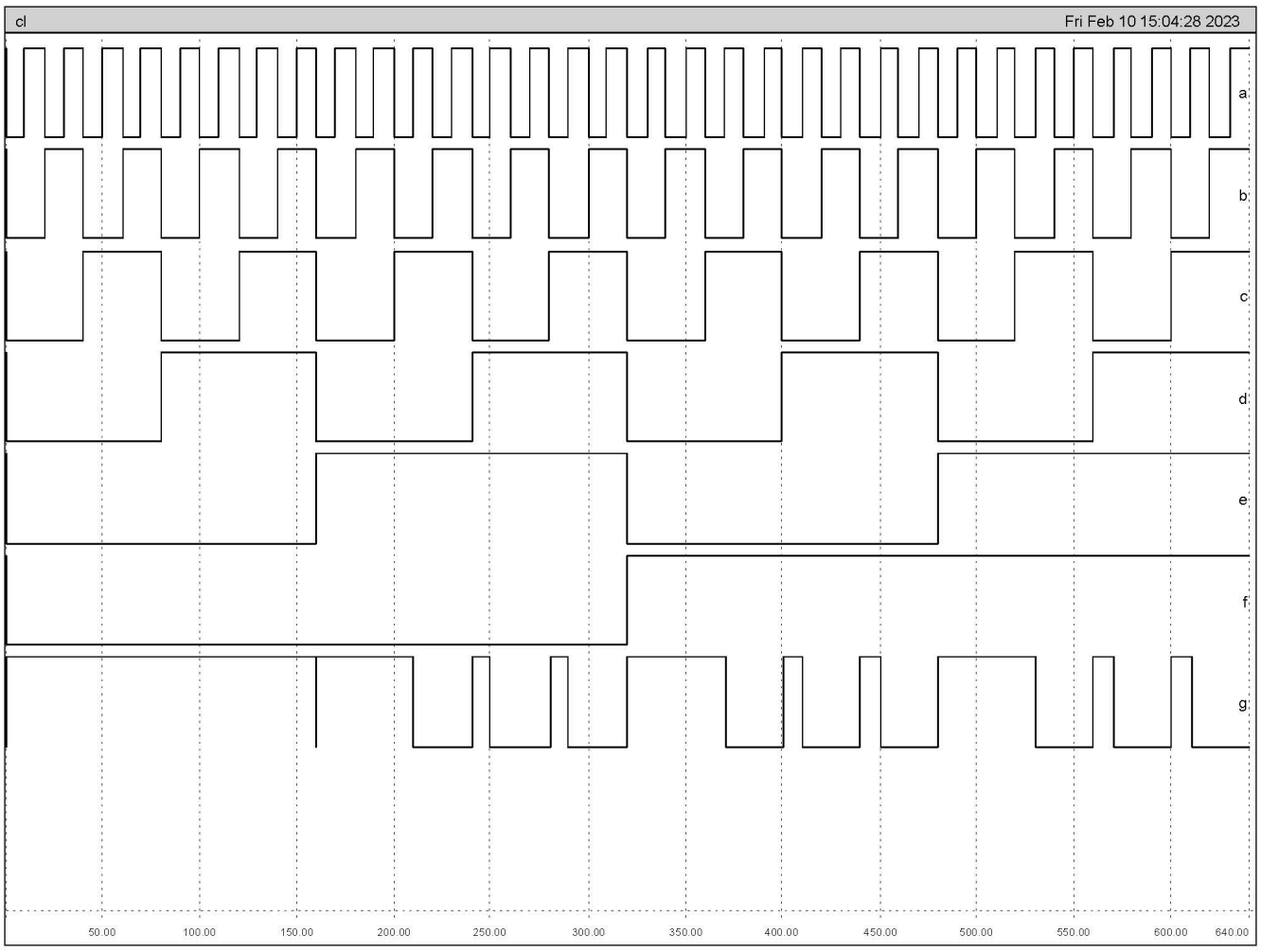
5. **Irsim test vector file and output plot showing correct function. Include .sim file.**

Irsim test vector file

```
|
| cl.cmd
|
| Comments
| Elec422/527 HW2 Problem1
| CMOS Combinational Logic
|
logfile cl.log
ana a b c d e f g
w   a b c d e f g
v    a 0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
v    b 0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
v    c 0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
v    d 0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
v    e 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
v    f 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
clock vdd 1
```

```
R
```

a plot of the Irsim results



6. **How many transistors (p and n) lines were there in your homework 1 solution and in your homework 2 solution in the .sim files? Compare and discuss.**

   There are 6 transistors lines in my homework 1 solution.

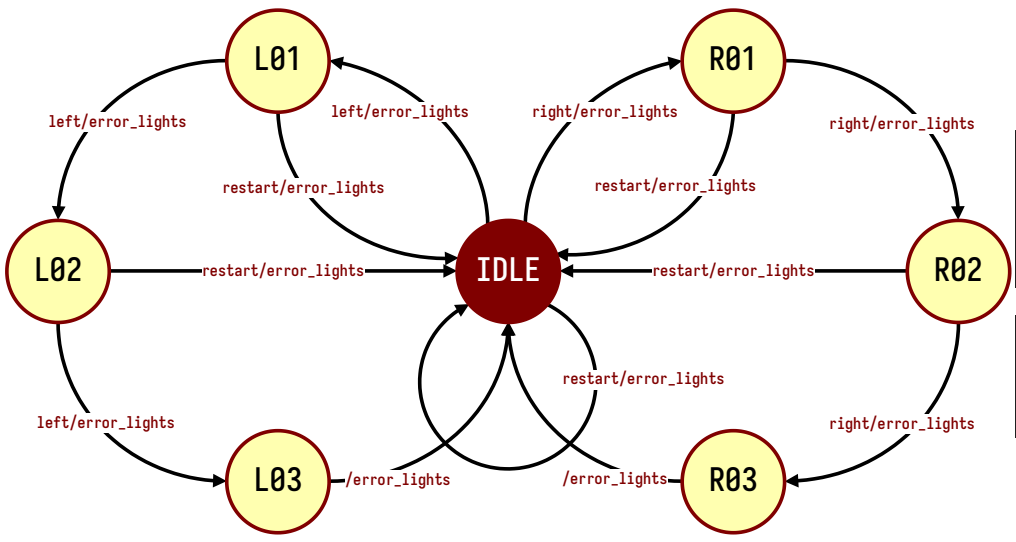
   There are 18 transistors lines in my homework 2 solution.

   In homework 1, we customized the module according to the logical expression, and in homework 2, we used the verilog language to describe the logical expression, and used the synthesis tool to generate it according to the standard library.

   Since there is no module in the standard library that completely corresponds to the logical expression of the topic, it is necessary to split the logical expression and splicing it with different gate circuits, and the cascading of input and output between different modules will also take up space. Therefore, it is not as compact as the design in homework 1.

Problem 2: Automobile Turn Signal Controller - FSM

1. **State transition diagram**




```
IDLE: error_lights <= {!and_lights[0],((6'b000_000)|or_lights)&and_lights};
R01:  error_lights <= {!and_lights[0],((6'b000_100)|or_lights)&and_lights};
R02:  error_lights <= {!and_lights[0],((6'b000_110)|or_lights)&and_lights};
R03:  error_lights <= {!and_lights[0],((6'b000_111)|or_lights)&and_lights};
L01:  error_lights <= {!and_lights[0],((6'b001_000)|or_lights)&and_lights};
L02:  error_lights <= {!and_lights[0],((6'b011_000)|or_lights)&and_lights};
L03:  error_lights <= {!and_lights[0],((6'b111_000)|or_lights)&and_lights};
```

```
//----------signal delay------------
always @ (negedge clka)
begin
  or_lights <=  (left|right)?({6{brake}}&{{3{right}},{3{left}}}):{6{brake}};
  and_lights <= (left&right) ? 6'b000000 : 6'b111111;
end
```

2. **Verilog input module file using two phase clocking**

```
//-------------------------------------------------------
// File Name   : Automobile Turn Signal Controller - FSM
// with count hold input and odd output
//-------------------------------------------------------
module atsc (clka, clkb, restart, brake, right, left, L0, L1, L2, R0, R1, R2, error, state);
//------Internal Constants-------------------------
parameter SIZE = 4;
parameter IDLE = 4'd0,                          // IDLE
          R01  = 4'd1, R02  = 4'd2, R03 = 4'd3,   // Turn right
          L01  = 4'd4, L02  = 4'd5, L03 = 4'd6;   // Turn left
//-------------Input Ports-----------------------------
input   clka, clkb, restart, brake, right, left;
//-------------Output Ports----------------------------
output  L0, L1, L2, R0, R1, R2, error;
output  [SIZE-1:0]state;
```

```
//-------------Input ports Data Type-------------------
wire    clka, clkb, restart, brake, right, left;
//-------------Output Ports Data Type------------------
wire    L0, L1, L2, R0, R1, R2;
//-------------Internal Variables----------------------------
reg   [SIZE-1:0]          state        ;// Seq part of the FSM
wire  [SIZE-1:0]          temp_state   ;// Internal state reg
reg   [SIZE-1:0]          next_state   ;// combo part of FSM
reg   [6:0]               error_lights   ;
wire  error                            ;
reg   [5:0]               or_lights    ;
reg   [5:0]               and_lights   ;
//----------Code startes Here------------------------
assign temp_state = fsm_function(state, brake, right, left, error);
assign error = error_lights[6];
assign L2 = error_lights[5];
assign L1 = error_lights[4];
assign L0 = error_lights[3];
assign R0 = error_lights[2];
assign R1 = error_lights[1];
assign R2 = error_lights[0];
//----------Function for Combo Logic-----------------
function [SIZE-1:0] fsm_function;
  input  [SIZE-1:0] state;
  input brake, right, left, error;
  case(state)
    // IDLE status
    IDLE: begin
  if((right)&(!left))
    fsm_function = R01;
  else if((left)&(!right))
    fsm_function = L01;
end
// Turn right status
R01: begin
  if((right)&(!left))
    fsm_function = R02;
  else
    fsm_function = IDLE;
end
R02: begin
  if((right)&(!left))
    fsm_function = R03;
  else
    fsm_function = IDLE;
end
R03: begin
    fsm_function = IDLE;
end
// Turn left status
L01: begin
  if((left)&(!right))
    fsm_function = L02;
  else
    fsm_function = IDLE;
end
L02: begin
      if((left)&(!right))
        fsm_function = L03;
      else
        fsm_function = IDLE;
    end
    L03: fsm_function = IDLE;
    //Brake status
    default : fsm_function = IDLE;
    endcase
endfunction
//----------Seq Logic-----------------------------
always @ (negedge clka)
begin : FSM_SEQ
  if (restart == 1'b1) begin
    next_state <= IDLE;
  end else begin
    next_state <= temp_state;
  end
end
//----------Output Logic--------------
always @ (negedge clkb)
begin : OUTPUT_LOGIC
  state <= next_state;
  case(next_state)
  IDLE: error_lights <= {!and_lights[0],((6'b000_000)|or_lights)&and_lights};
  R01:  error_lights <= {!and_lights[0],((6'b000_100)|or_lights)&and_lights};
  R02:  error_lights <= {!and_lights[0],((6'b000_110)|or_lights)&and_lights};
```

```
  R03:  error_lights <= {!and_lights[0],((6'b000_111)|or_lights)&and_lights};
  L01:  error_lights <= {!and_lights[0],((6'b001_000)|or_lights)&and_lights};
  L02:  error_lights <= {!and_lights[0],((6'b011_000)|or_lights)&and_lights};
  L03:  error_lights <= {!and_lights[0],((6'b111_000)|or_lights)&and_lights};
  endcase
end // End Of Block OUTPUT_LOGIC
//----------signal delay--------------
always @ (negedge clka)
begin
  or_lights <=  (left|right)?({6{brake}}&{{3{right}},{3{left}}}):{6{brake}};
  and_lights <= (left&right) ? 6'b000000 : 6'b111111;
end

endmodule // End of Module atsc
```

3. Verilog testbench file covering the test sequence in the problem statement

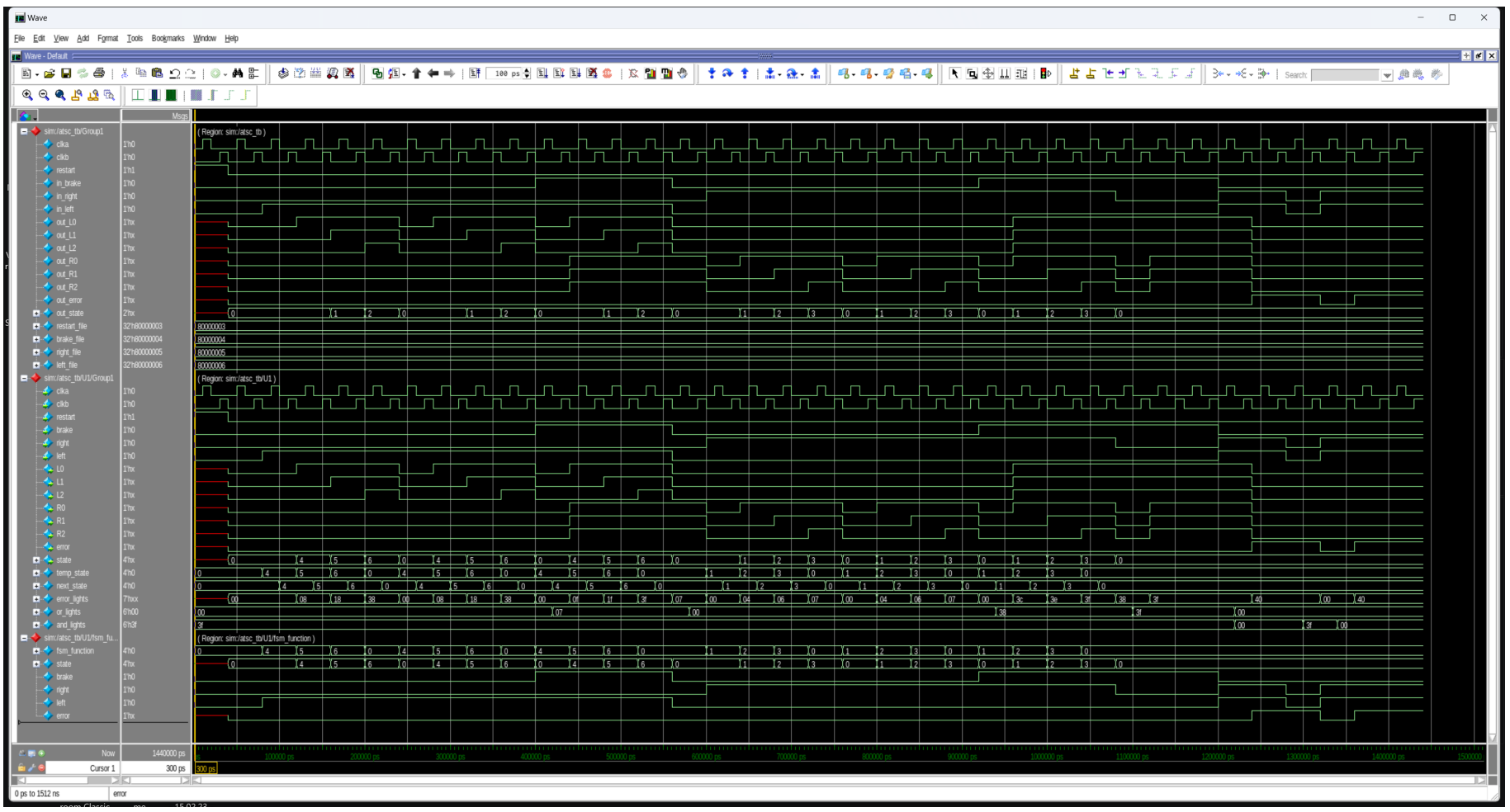
```
`timescale 1ns/100ps

module atsc_tb();

reg clka, clkb, restart, in_brake, in_right, in_left;
wire out_L0, out_L1, out_L2, out_R0, out_R1, out_R2, out_error;
wire [1:0] out_state;

integer restart_file;
integer brake_file;
integer right_file;
integer left_file;

initial begin
  restart_file = $fopen("./restart_file.txt", "w");
  brake_file = $fopen("./brake_file.txt", "w");
  right_file = $fopen("./right_file.txt", "w");
  left_file = $fopen("./left_file.txt", "w");
end

always @(negedge clkb) begin
    $fwrite(restart_file, "%d ", restart);
    $fwrite(brake_file, "%d ", in_brake);
    $fwrite(right_file, "%d ", in_right);
    $fwrite(left_file, "%d ", in_left);
end

//create a atsc instance.
atsc U1 (   .clka        (clka),
            .clkb        (clkb),
            .restart     (restart),
            .brake       (in_brake),
            .right       (in_right),
            .left        (in_left),
            .L0          (out_L0),
            .L1          (out_L1),
            .L2          (out_L2),
            .R0          (out_R0),
            .R1          (out_R1),
            .R2          (out_R2),
            .error       (out_error),
            .state       (out_state)
        );

initial
begin

// cycle 0
// RESTART(1)
restart = 1;
in_brake = 0;
in_right = 0;
in_left = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// cycle 1
// idle(1)
restart = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// cycle 2 - 9
```

```
// LEFT(8)
in_brake = 0;
in_right = 0;
in_left = 1;
repeat (8) begin
    clka = 0; clkb = 0; #10;
    clka = 1; clkb = 0; #10;
    clka = 0; clkb = 0; #10;
    clka = 0; clkb = 1; #10;
end

// cycle 10 - 13
// LEFT and BRAKE (4)
in_brake = 1;
in_right = 0;
in_left = 1;
repeat (4) begin
    clka = 0; clkb = 0; #10;
    clka = 1; clkb = 0; #10;
    clka = 0; clkb = 0; #10;
    clka = 0; clkb = 1; #10;
end

// cycle 14
// idle(1)
in_brake = 0;
in_right = 0;
in_left = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;


// cycle 15 - 22
// RIGHT(8)
in_brake = 0;
in_right = 1;
in_left = 0;
repeat (8) begin
    clka = 0; clkb = 0; #10;
    clka = 1; clkb = 0; #10;
    clka = 0; clkb = 0; #10;
    clka = 0; clkb = 1; #10;
end

// cycle 23 - 26
// RIGHT and BRAKE (4)
in_brake = 1;
in_right = 1;
in_left = 0;
repeat (4) begin
    clka = 0; clkb = 0; #10;
    clka = 1; clkb = 0; #10;
    clka = 0; clkb = 0; #10;
    clka = 0; clkb = 1; #10;
end

// cycle 27 - 29
// BRAKE (3)
in_brake = 1;
in_right = 0;
in_left = 0;
repeat (3) begin
    clka = 0; clkb = 0; #10;
    clka = 1; clkb = 0; #10;
    clka = 0; clkb = 0; #10;
    clka = 0; clkb = 1; #10;
end

// cycle 30 - 31
// LEFT and RIGHT (2)
in_brake = 0;
in_right = 1;
in_left = 1;
repeat (2) begin
    clka = 0; clkb = 0; #10;
    clka = 1; clkb = 0; #10;
    clka = 0; clkb = 0; #10;
    clka = 0; clkb = 1; #10;
end

// cycle 32
// idle(1)
```

```
in_brake = 0;
in_right = 0;
in_left = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// cycle 33
// LEFT and RIGHT (2)
in_brake = 0;
in_right = 1;
in_left = 1;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// cycle 34
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// cycle 35
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

$dumpfile ("atsc_tb.vcd");
$dumpvars;


$stop;
end

endmodule
```

4. **Modelsim output plot showing correct function both before and after synthesis**

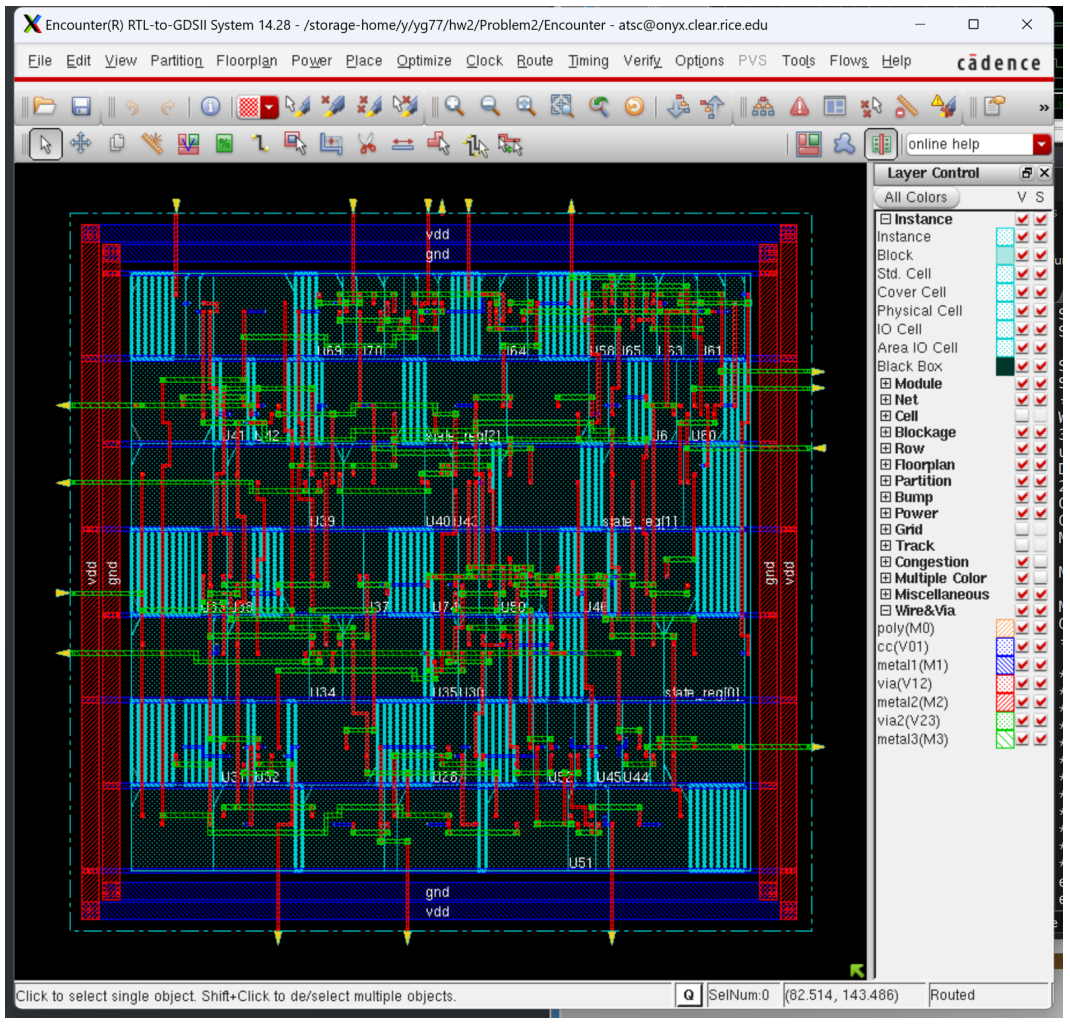
   before synthesis



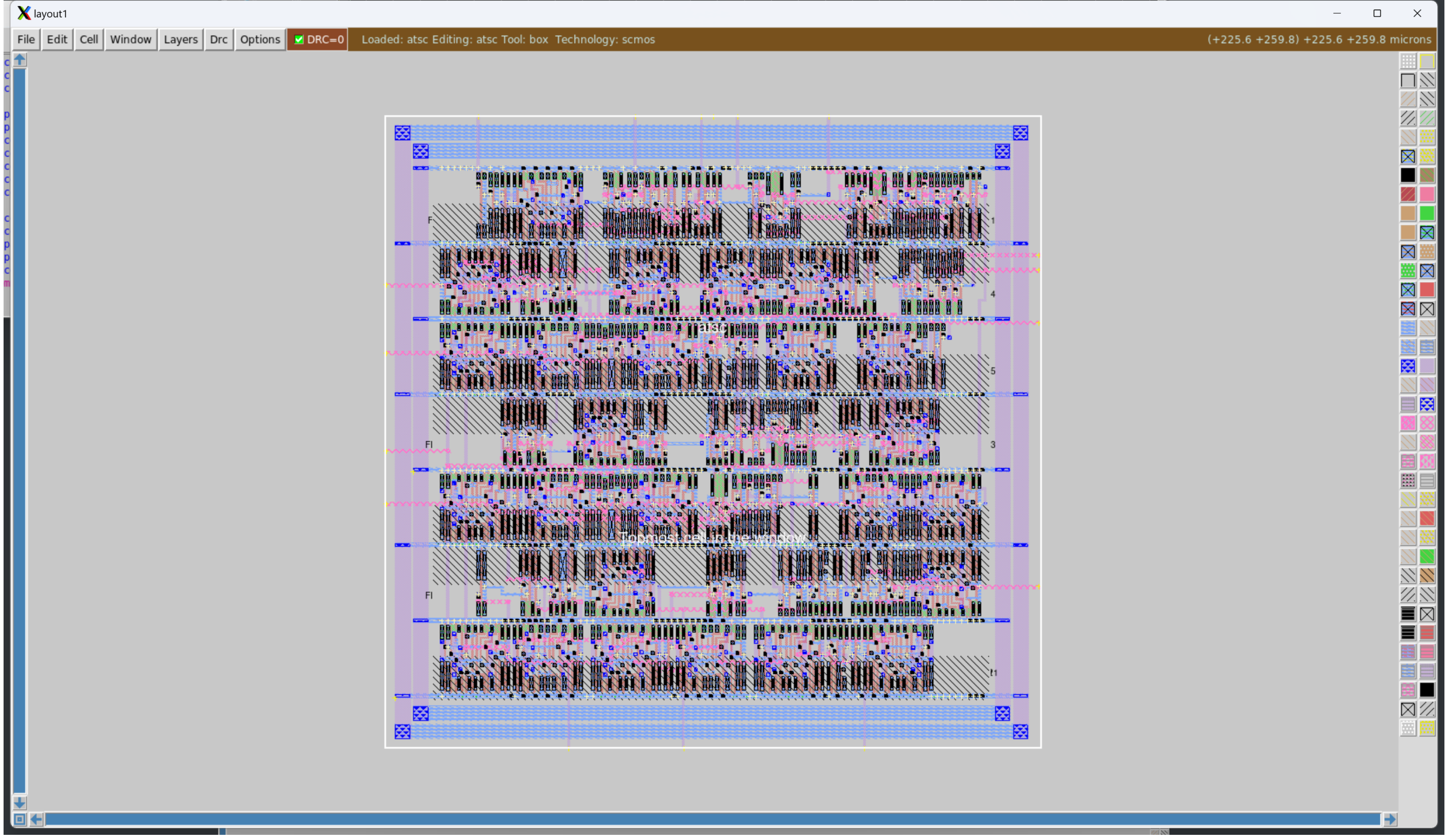
   after synthesis

5. Sceenshot from Encounter and from Magic - upload magic files - report any label or DRC errors

screenshot from Encounter



screenshot of magic window

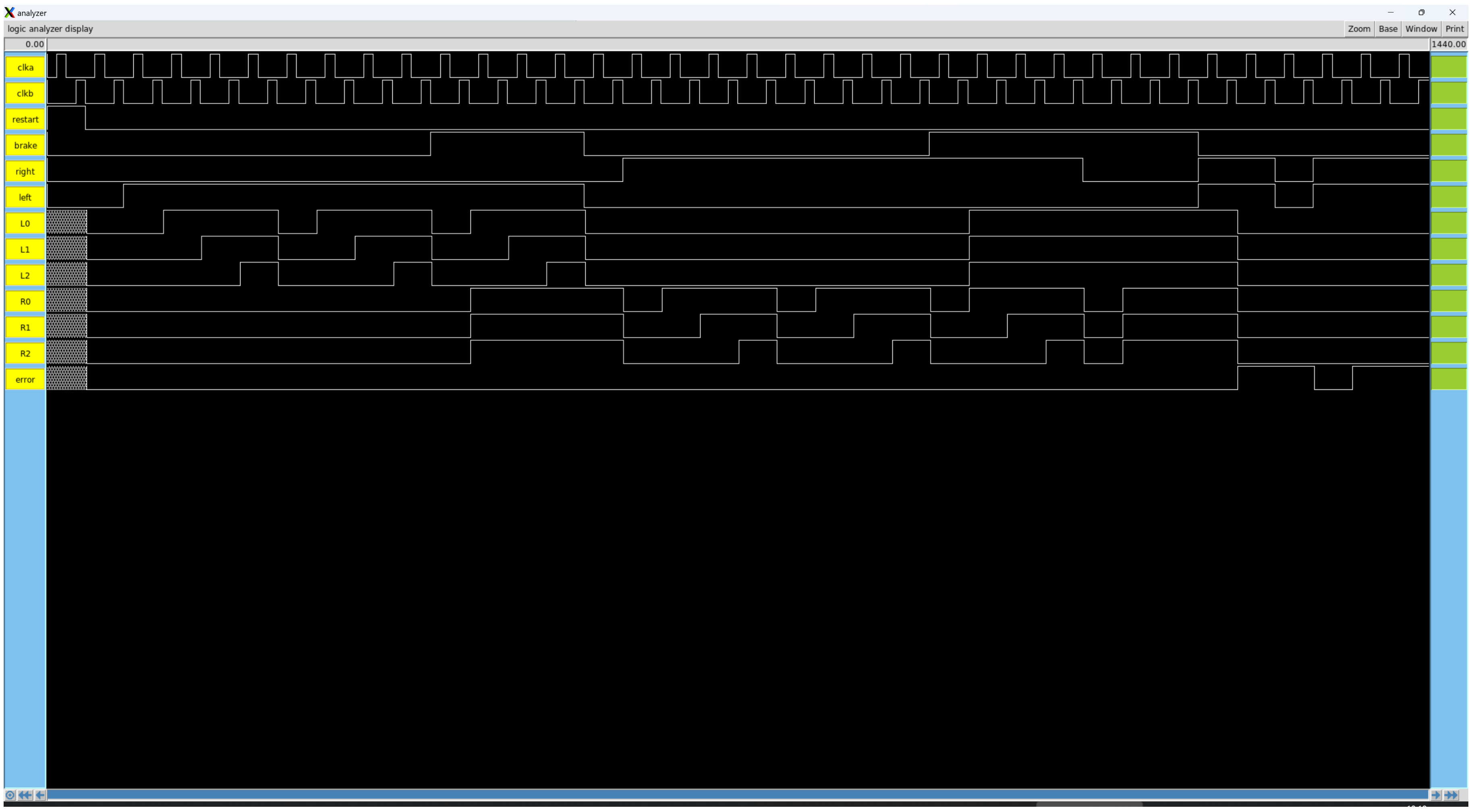


6. **Irsim test vector file covering the test sequence in the problem statement and output plot showing correct function. Include .sim file.**

Irsim test vector file

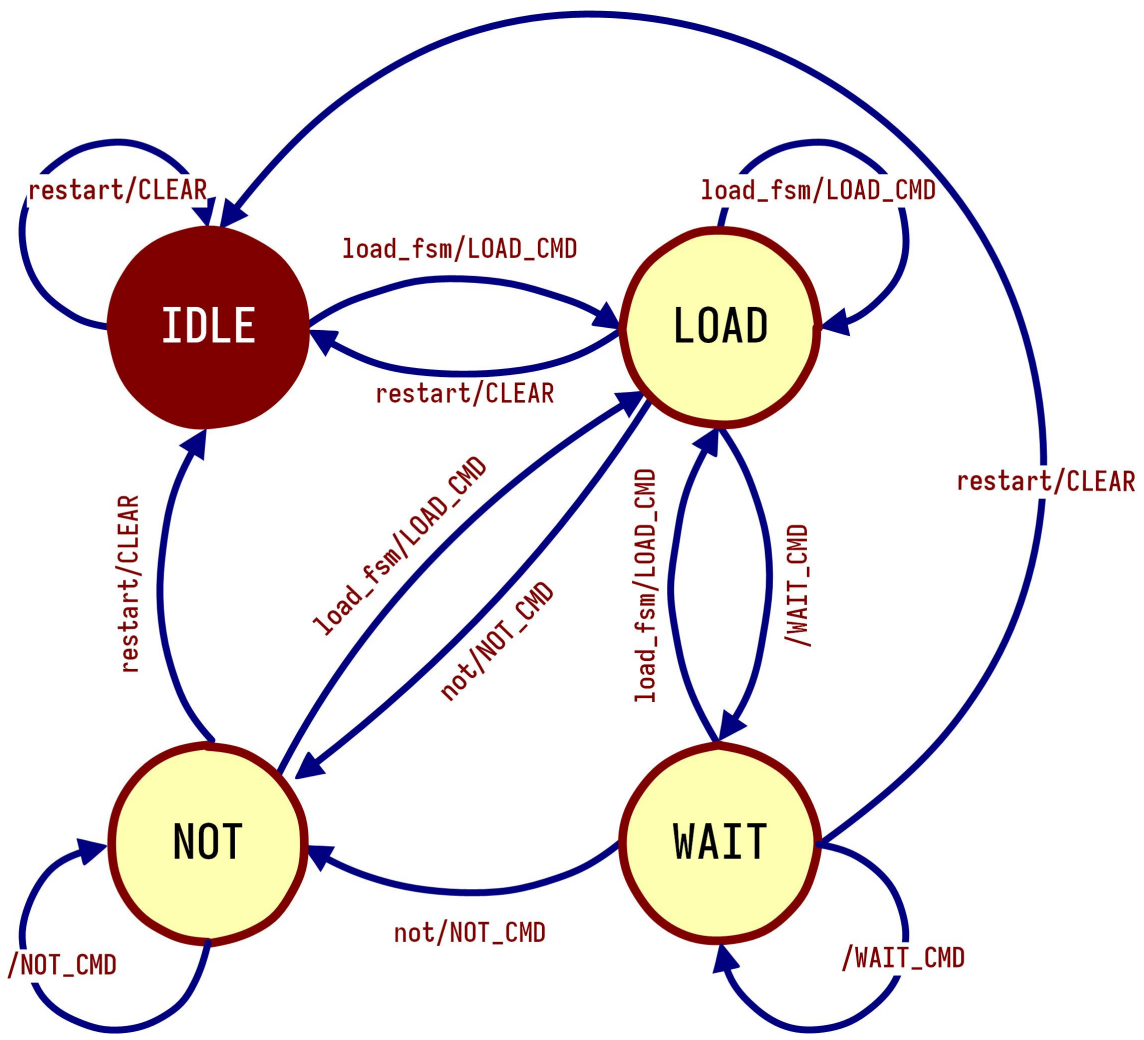
```
|
| atsc.cmd.txt
|
| Comments
| Elec422/527 Problem2 atsc
| define vectors for easier display
ana clka clkb restart brake right left L0 L1 L2 R0 R1 R2 error
V restart 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 
V brake   0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
V right   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1
V left    0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1
| Two phase clock with non-overlap period - same as Modelsim testbench
clock clka 0 1 0 0
clock clkb 0 0 0 1
R
```

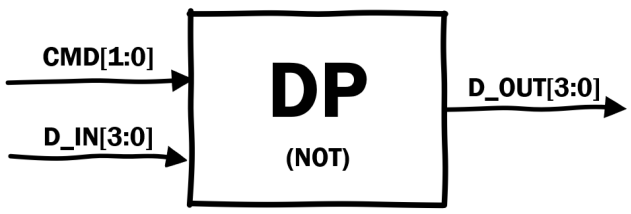
a plot of Irsim result



3. Problem 3: Finite State Machine with Datapath

1. **State transition diagram and block diagram showing datapath interconnections**





2. **Verilog input module file for top, FSM, and datapath using two phase clocking**

```
//-----------------------------------------------------
// Design Name : main_FSM
// File Name   : main_FSM.v
// Function    : 2 Phase Clock main_FSM
//       Controller processes load input and issues
//       load signal to data path dp.
//-----------------------------------------------------
module main_FSM (clka, clkb, restart, in_load, in_not, out_state, out_cmd);
//-------------Input Ports-----------------------------
input wire clka, clkb, restart, in_load, in_not;
//-------------Output Ports----------------------------
output reg [1:0] out_cmd;
output reg [1:0] out_state;
//-------------Internal Constants----------------------
parameter IDLE  = 2'b00, LOAD = 2'b01, WAIT = 2'b10, NOT = 2'b11;
parameter CLEAR_CMD  = 2'b00, LOAD_CMD = 2'b01, WAIT_CMD = 2'b10, NOT_CMD = 2'b11;
//-------------Internal Variables----------------------------
wire  [1:0]          temp_state;    // Internal wire for output of function for setting next state
reg   [1:0]          next_state;    // Temporary reg to hold next state to update state on output
reg                  load_delay;
wire                 load_fsm;
//----------Code startes Here------------------------
// Delay in_load for one cycle
always @ (negedge clka) begin
  load_delay <= in_load;
end
// Detect the rising edge of in_load
assign load_fsm = in_load?((!load_delay)?1:0):0;
assign temp_state = fsm_function(out_state, load_fsm, in_not);
//----------Function for Combinational Logic to read inputs -----------
function [1:0] fsm_function;
  input [1:0] state;
  input load_fsm, in_not;
  case(out_state)
    IDLE: begin
      if(load_fsm == 1'b1) begin
        fsm_function = LOAD;
      end else begin
        fsm_function = IDLE;
      end
    end
    LOAD: begin
      if(load_fsm == 1'b1) begin
```

```
        fsm_function = LOAD;
      end else if (in_not == 1'b1) begin
        fsm_function = NOT;
      end else begin
        fsm_function = WAIT;
      end
    end
    WAIT: begin
      if(load_fsm == 1'b1) begin
        fsm_function = LOAD;
      end else if (in_not == 1'b1) begin
        fsm_function = NOT;
      end
    end
    NOT: begin
      if(load_fsm == 1'b1) begin
        fsm_function = LOAD;
      end else begin
        fsm_function = NOT;
      end
    end
  endcase
endfunction

//----------Seq Logic-----------------------------
always @ (negedge clka)
begin : FSM_SEQ
  if (restart == 1'b1) begin
    next_state <= IDLE;
  end else begin
    next_state <= temp_state;
  end
end
//----------Output Logic--------------
always @ (negedge clkb)
begin : OUTPUT_LOGIC
  case(next_state)
  IDLE: begin
          out_state <= next_state;
          out_cmd <= CLEAR_CMD;
        end
  LOAD: begin
          out_state <= next_state;
          out_cmd <= LOAD_CMD;
        end
  WAIT: begin
          out_state <= next_state;
          out_cmd <= WAIT_CMD;
        end
  NOT: begin
          out_state <= next_state;
          out_cmd <= NOT_CMD;
        end
  default: begin
          out_state <= next_state;
          out_cmd <= IDLE;
         end
  endcase
end // End Of Block OUTPUT_LOGIC

endmodule // End of Module main_FSM
```

```
//------------------------------------------------------
// Design Name : dp
// File Name   : dp.v
// Function    : Data path
//------------------------------------------------------
//
//
module dp (clka, clkb, in_cmd, in_d1, out_d);
//-----------Input Ports---------------
input clka, clkb;
input [1:0] in_cmd;
input [3:0] in_d1;
//-----------Output Ports---------------
output reg [3:0] out_d;
//------------Internal Variables--------
reg  [3:0] temp_1;
//-------------Internal Constants-------
parameter CLEAR_CMD  = 2'b00, LOAD_CMD = 2'b01, WAIT_CMD = 2'b10, NOT_CMD = 2'b11;
//-------------Code Starts Here---------
// Qualify the control signal by clka and clkb for the d1 and d_out registers
always @ (negedge clka)
begin
```

```
if (in_cmd == CLEAR_CMD) begin
   temp_1 = 4'b0000;
   end else if (in_cmd == LOAD_CMD) begin
   temp_1 = in_d1;
   end
end

always @ (negedge clkb)
begin
if (in_cmd == NOT_CMD) begin
out_d  = ~temp_1;
end else begin
out_d  = temp_1;
end
end

endmodule //End Of Module dp  datapath
```

```
//-----------------------------------------------------
// Design Name : top_module
// File Name   : top_module.v
//
// Function    : top file call main_FSM controller.
//               When in_load goes high as Stable_A1, main_FSM issues start as
//               Valid_B1_Stable_A2 (A of next cycle)
//               to load the two input registers.
//               The dp datapath then receives the start Valid_B1_Stable_A2
//               and expects data to be Stable_A2. The datapath then adds the
//               two register values and stores the result in the output latch
//               on clkb which is on clock cycle 2 so that the output is
//               Valid_B2_Stable_A3. The output should not change until a new
//               in_load is received.
//               The dp makes the output latch value visible through
//               this top module.
//-----------------------------------------------------
module top_module (clka, clkb, restart, in_load, in_not, out_state, in_d1, out_d);
//-------------Input Ports-----------------------------
input   clka, clkb, restart, in_load, in_not, in_d1;
//-------------Output Ports----------------------------
output [1:0] out_state;
output [3:0] out_d;
//-------------Input ports Data Type-------------------
wire   clka, clkb, restart, in_load, in_not;
wire   [3:0] in_d1;
//-------------Output Ports Data Type------------------
wire   [1:0] out_state;
wire   [3:0] out_d;
//
wire   [1:0] cmd;
//----------Code startes Here------------------------
main_FSM main_FSM_ins(  .clka       (clka)          ,
                        .clkb       (clkb)          ,
                        .restart    (restart)       ,
                        .in_load    (in_load)       ,
                        .in_not     (in_not)        ,
                        .out_state  (out_state)     ,
                        .out_cmd    (cmd)
                        );

dp dp_ins           (   .clka       (clka)          ,
                        .clkb       (clkb)          ,
                        .in_cmd     (cmd)           ,
                        .in_d1      (in_d1)         ,
                        .out_d      (out_d)
                        );

endmodule // End of Module top_module
```

3. Verilog testbench file covering the test sequence in the problem statement

```
//-----------------------------------------------------
//// Design Name : Testbench for top_module
//// File Name   : top_module_tb.v
//// Function    : Testbench for top module and total project.
////-----------------------------------------------------
//

module top_module_tb();

// Inputs to top_module
reg  clka, clkb, restart, in_load, in_not;
reg  [3:0] in_d1;
// Outputs from top_module
```

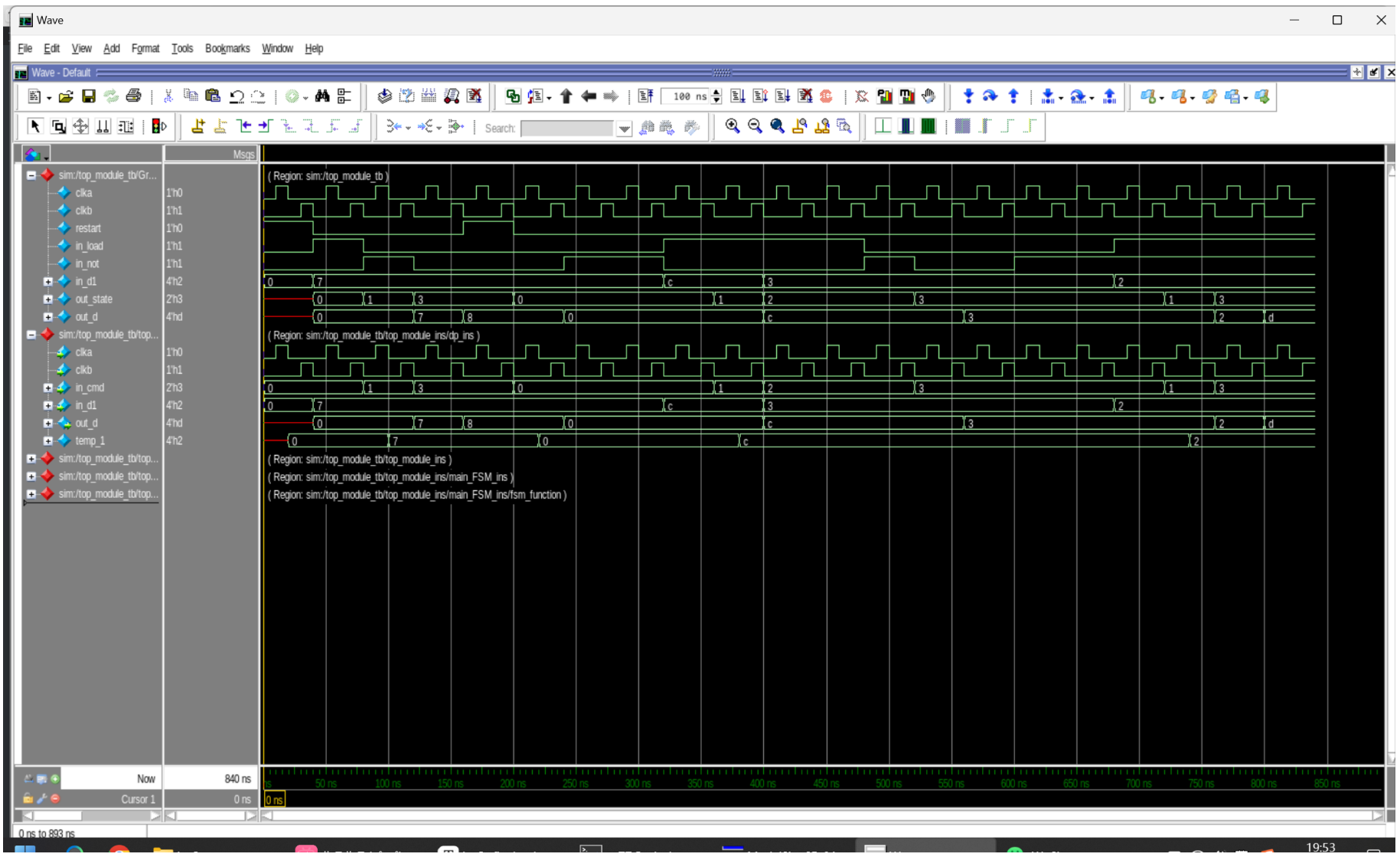
```
wire [1:0] out_state;
wire [3:0] out_d;

integer restart_file;
integer in_load_file;
integer in_not_file;
integer in_d1_00_file;
integer in_d1_01_file;
integer in_d1_02_file;
integer in_d1_03_file;

initial begin
  restart_file = $fopen("./restart_file.txt", "w");
  in_load_file = $fopen("./in_load_file.txt", "w");
  in_not_file = $fopen("./in_not_file.txt", "w");
  in_d1_00_file = $fopen("./in_d1_00_file.txt", "w");
  in_d1_01_file = $fopen("./in_d1_01_file.txt", "w");
  in_d1_02_file = $fopen("./in_d1_02_file.txt", "w");
  in_d1_03_file = $fopen("./in_d1_03_file.txt", "w");
end

always @(negedge clkb) begin
    $fwrite(restart_file, "%d ", restart);
    $fwrite(in_load_file, "%d ", in_load);
    $fwrite(in_not_file, "%d ", in_not);
    $fwrite(in_d1_00_file, "%d ", in_d1[0]);
    $fwrite(in_d1_01_file, "%d ", in_d1[1]);
    $fwrite(in_d1_02_file, "%d ", in_d1[2]);
    $fwrite(in_d1_03_file, "%d ", in_d1[3]);
end

//create a top FSM system instance.
top_module top_module_ins(  .clka       (clka),
                            .clkb       (clkb),
                            .restart    (restart),
                            .in_load    (in_load),
                            .in_not     (in_not),
                            .out_state  (out_state),
                            .in_d1      (in_d1),
                            .out_d      (out_d)
);

initial
begin

// Cycle 0
restart = 1;
in_load = 0;
in_not = 0;
in_d1 = 4'd0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10

// Cycle 1
restart = 0;
in_load = 1;
in_d1 = 4'd7;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10

// Cycle 2
in_load = 0;
in_not = 1;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10

// Cycle 3
in_not = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10

// Cycle 4
restart = 1;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
```

```
clka = 0; clkb = 1; #10

// Cycle 5
restart = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10

// Cycle 6
in_not = 1;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10

// Cycle 7
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 8
in_not = 0;
in_load = 1;
in_d1 = 4'd12;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 9
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 10
in_d1 = 4'd3;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 11
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 12
in_load = 0;
in_not = 1;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 13
in_not = 0;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;


// Cycle 14
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 15
in_not = 1;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 16
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;
```

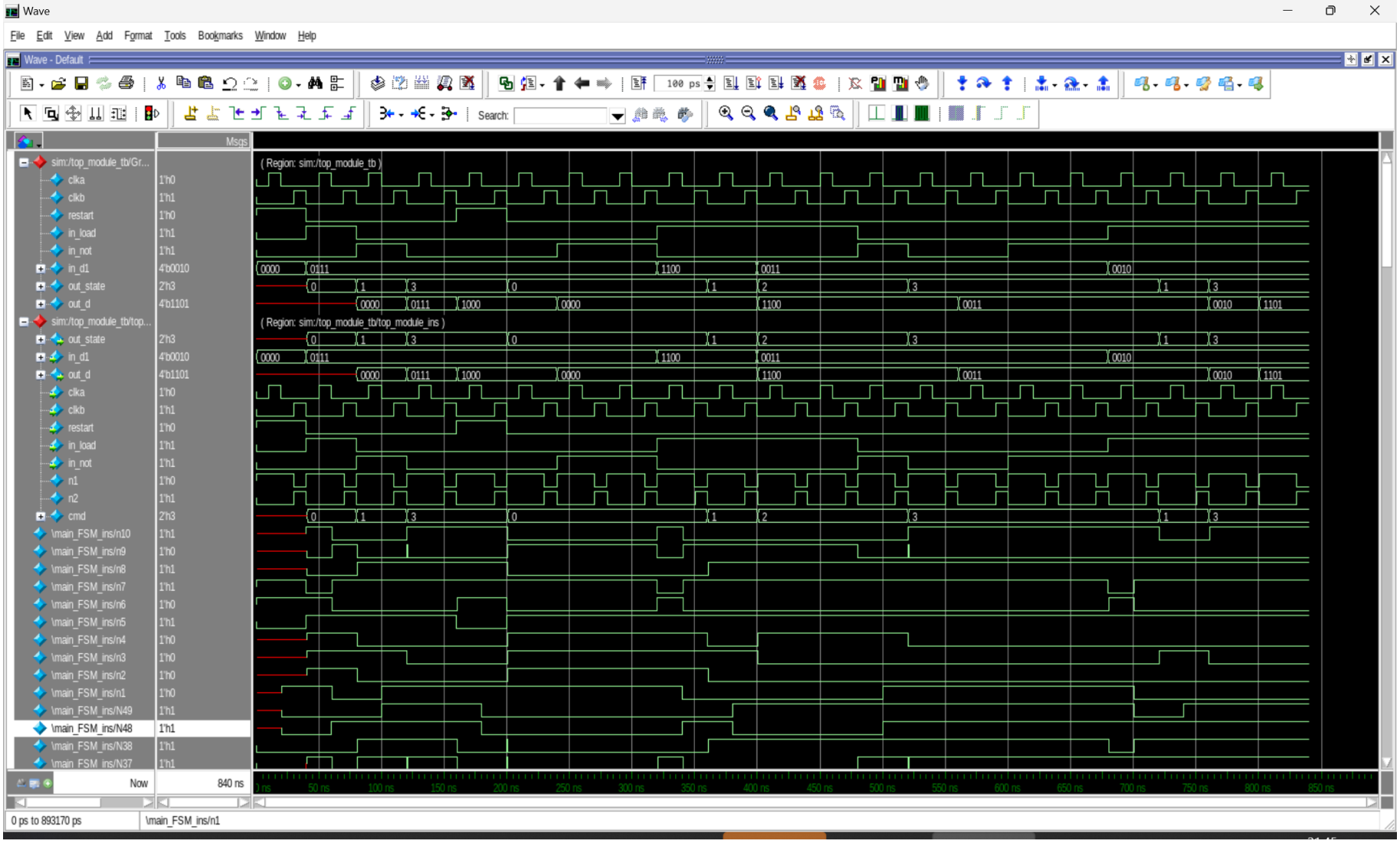
```
// Cycle 17
in_load = 1;
in_d1 = 4'd2;
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 18
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 19
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

// Cycle 19
clka = 0; clkb = 0; #10;
clka = 1; clkb = 0; #10;
clka = 0; clkb = 0; #10;
clka = 0; clkb = 1; #10;

$dumpfile ("top_module_tb.vcd");
$dumpvars;


$stop;
end

endmodule
```

4. **Modelsim output plot showing correct function both before and after synthesis**

   before synthesis



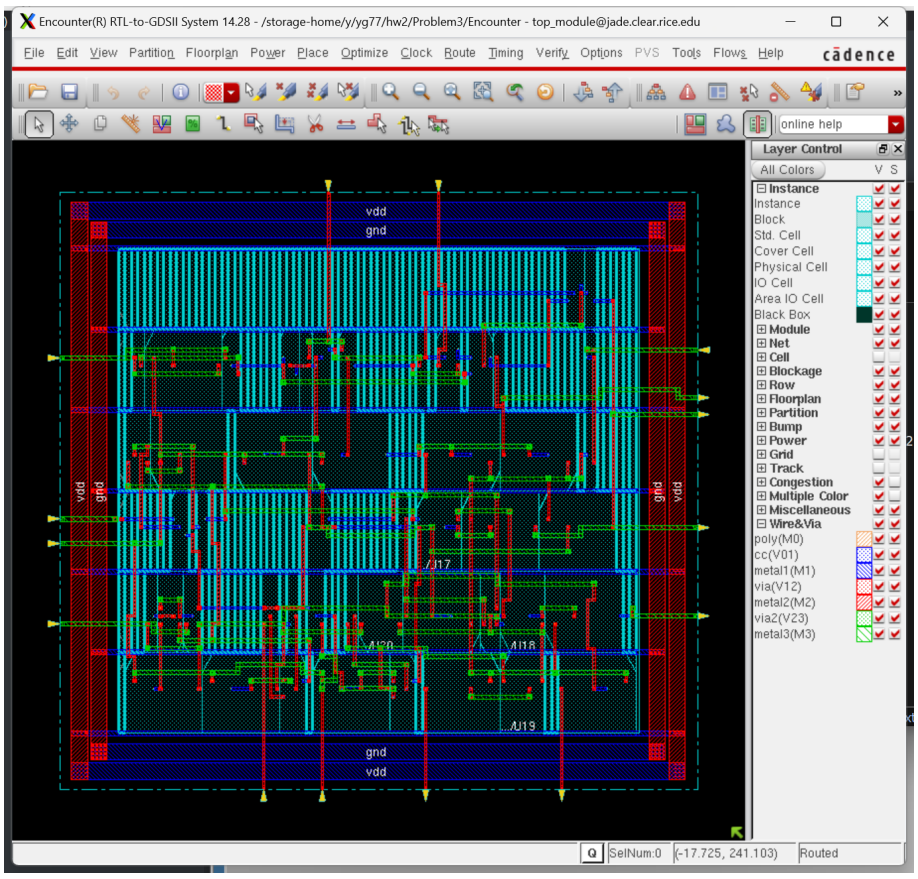
   after synthesis



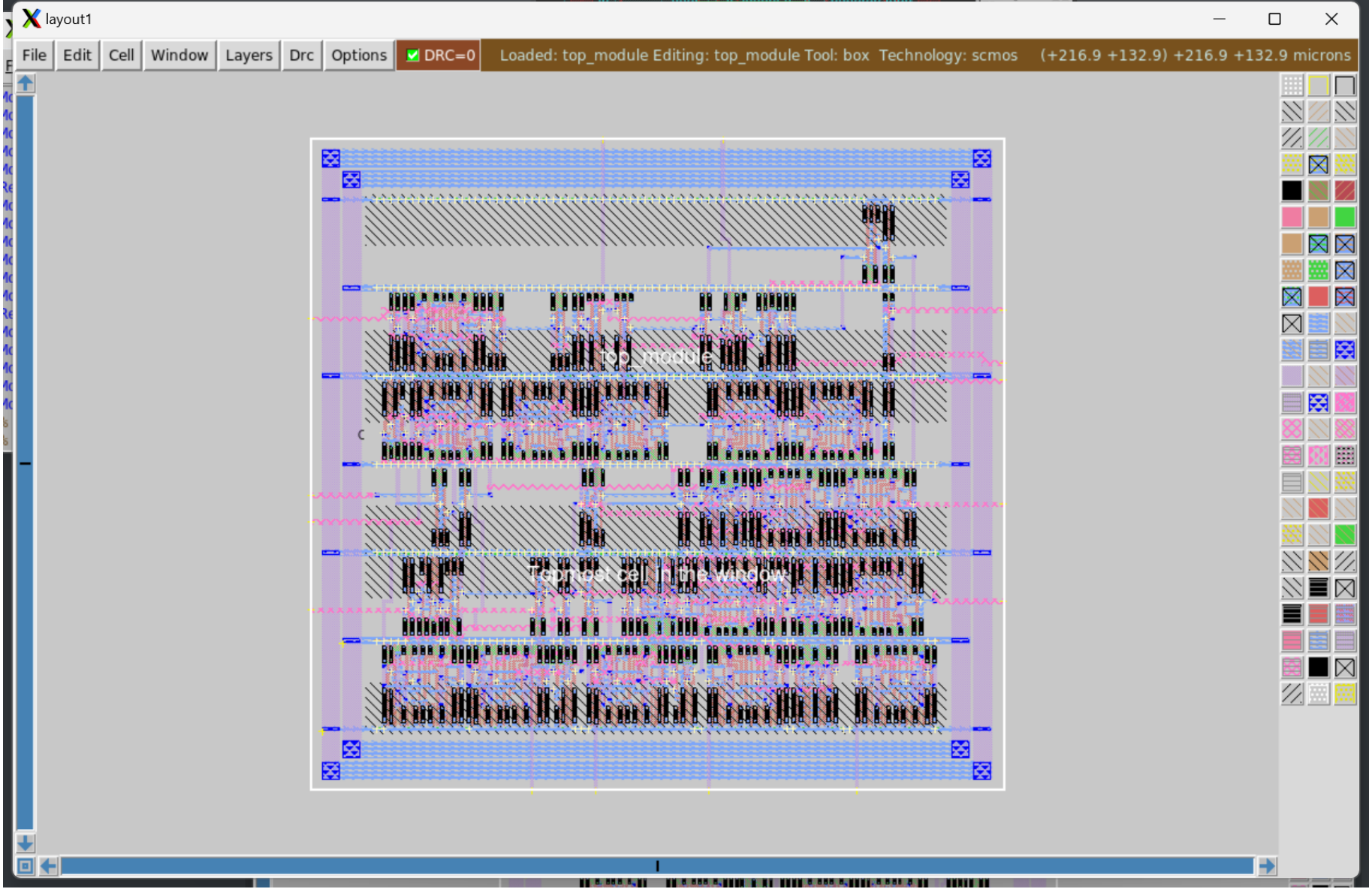
5. **Sceenshot from Encounter and from Magic - upload magic files - report any label or DRC errors**

There is no reported error.

Sceenshot from Encounter



Sceenshot of Magic window

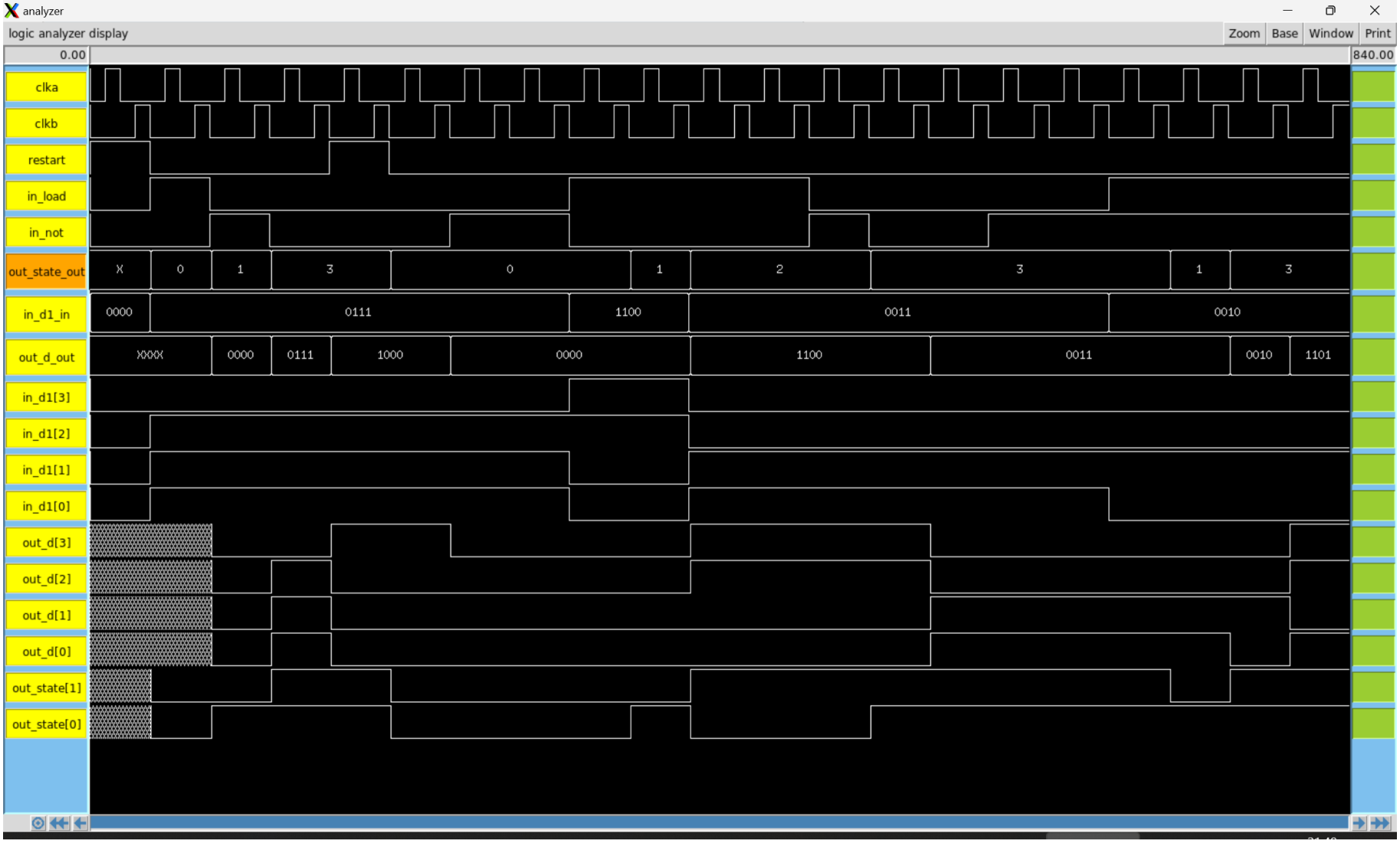


6. Irsim test vector file covering the test sequence in the problem statement and output plot showing correct function. Include .sim file.

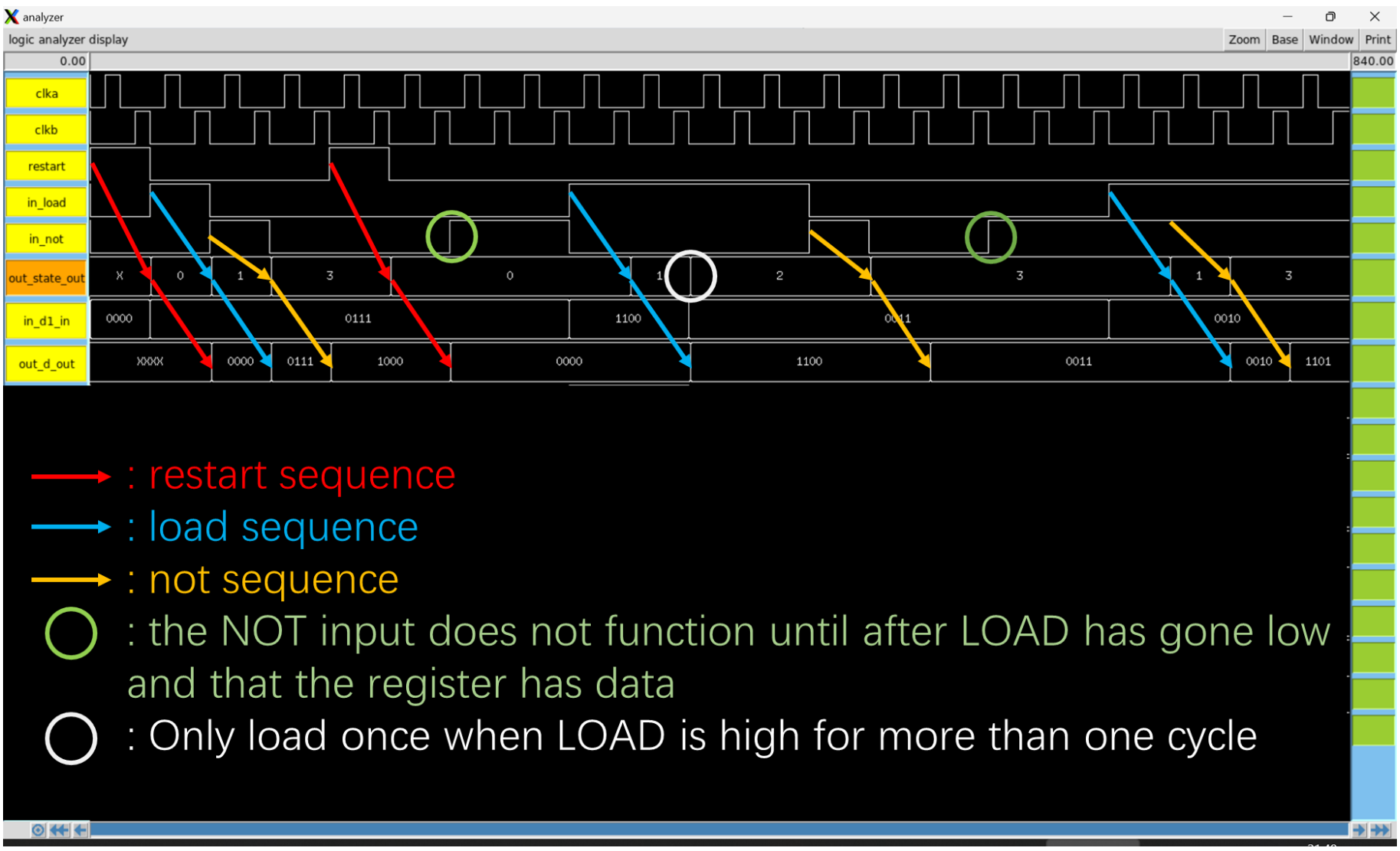
   Irsim test vector file

```
|
| top_module.cmd
|
| Comments
| Elec422/527 Problem3 FSM_DP
|
| define vectors for easier display
vector in_d1_in         in_d1\[3\] in_d1\[2\] in_d1\[1\] in_d1\[0\]
vector out_d_out        out_d\[3\] out_d\[2\] out_d\[1\] out_d\[0\]
vector out_state_out    out_state\[1\] out_state\[0\]
|
logfile twoP_FSM.log
ana clka clkb restart in_load in_not
ana in_d1_in out_d_out out_state_out
ana in_d1\[3\] in_d1\[2\] in_d1\[1\] in_d1\[0\]
ana out_d\[3\] out_d\[2\] out_d\[1\] out_d\[0\]
ana out_state\[1\] out_state\[0\]
|   cycle         1 2 3 4 5 6 7 8 9 10111213141516171819202l
v   restart       1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
v   in_load       0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1
v   in_not        0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 1 1 1 1
v   in_d1\[3\]    0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
v   in_d1\[2\]    0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
v   in_d1\[1\]    0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
v   in_d1\[0\]    0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0
| Two phase clock with non-overlap period - same as Modelsim testbench
clock clka 0 1 0 0
clock clkb 0 0 0 1
R
```

   a plot of Irsim result


analyzer
logic analyzer display
Zoom
Base
Window
Print
0.00
840.00
clka
clkb
restart
in_load
in_not
out_state_out
X 0 1 3 0 1 2 3 1 3
in_d1_in
0000 0111 1100 0011 0010
out_d_out
XXXX 0000 0111 1000 0000 1100 0011 0010 1101
in_d1[3]
in_d1[2]
in_d1[1]
in_d1[0]
out_d[3]
out_d[2]
out_d[1]
out_d[0]
out_state[1]
out_state[0]

Annotation:


analyzer
logic analyzer display
Zoom
Base
Window
Print
0.00
840.00
clka
clkb
restart
in_load
in_not
out_state_out
X 0 1 3 0 1 2 3 1 3
in_d1_in
0000 0111 1100 0011 0010
out_d_out
XXXX 0000 0111 1000 0000 1100 0011 0010 1101
: restart sequence
: load sequence
: not sequence
: the NOT input does not function until after LOAD has gone low and that the register has data
: Only load once when LOAD is high for more than one cycle