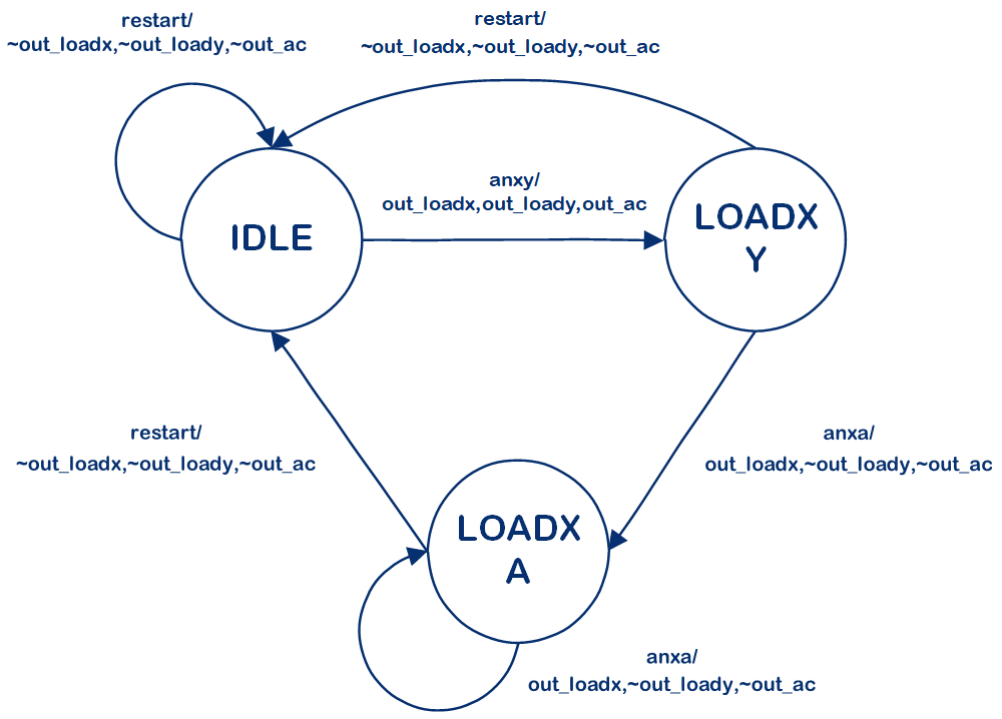
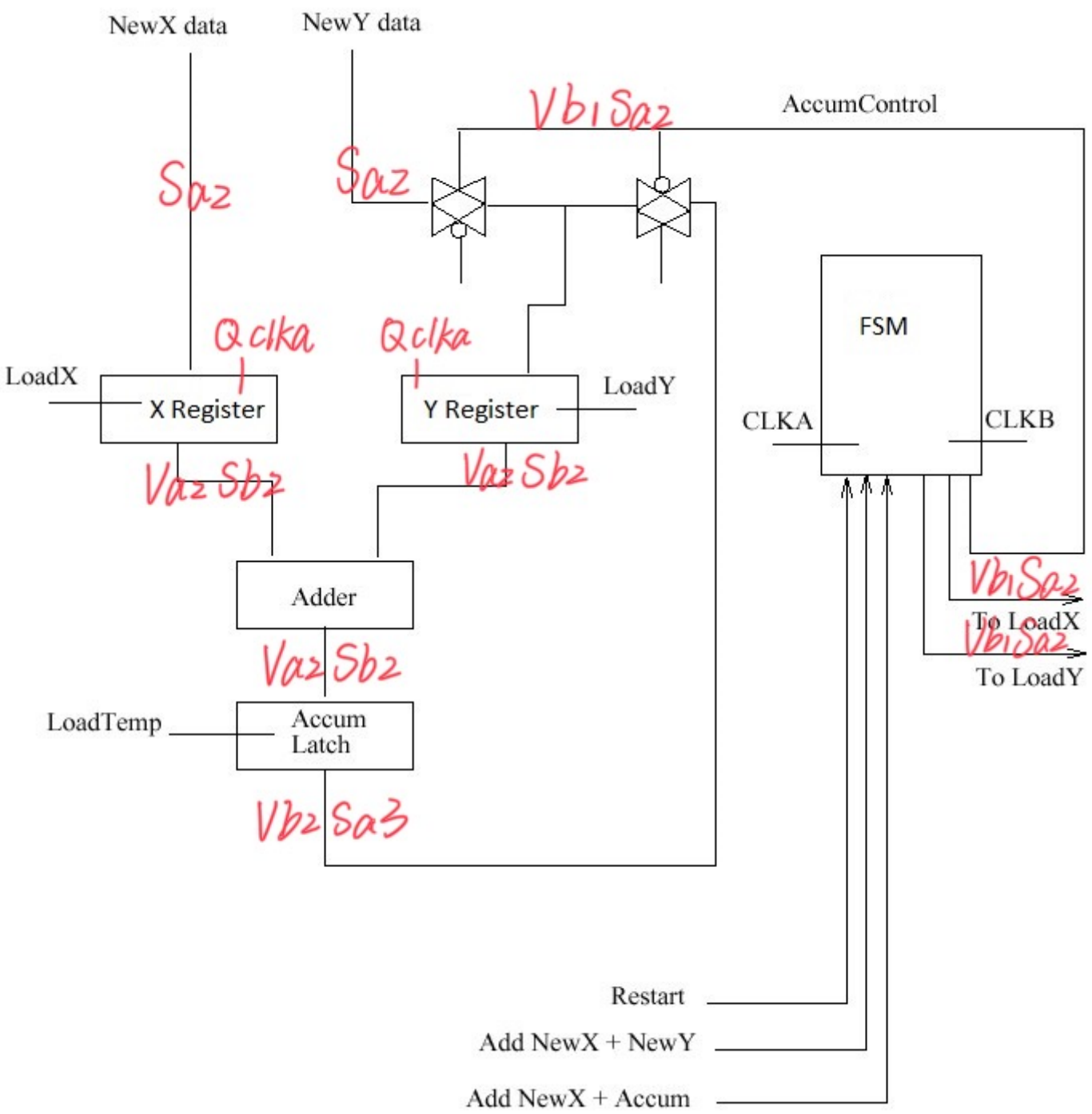
ELEC 527 Homework 3

Name: Yufei Gu NetID: yg77

1. **Problem 1: Accumulator Based System**
    - Timing analysis V,S labels on diagram
        - FSM diagram



        - DP diagram



    - Verilog input module file using two phase clocking and datapath
        - FSM

```
```

```
module main_FSM (in_clka, in_clkb, in_restart, in_anxy, in_anxa, out_state,
out_loadx, out_loady, out_ac);
  // Internal Constants
  parameter SIZE = 2;
  parameter IDLE = 2'b00, LOADXY = 2'b01, LOADXA = 2'b10;
  // Input Ports
  input wire in_clka, in_clkb, in_restart, in_anxy, in_anxa;
  // Output Ports
  output reg [SIZE-1:0] out_state;
  output reg out_loadx, out_loady, out_ac;
  // Internal Variables
  wire [SIZE-1:0] temp_out_state;
  reg [SIZE-1:0] next_out_state;
  // Code startes Here
  assign temp_out_state = fsm_function(out_state, in_anxy, in_anxa);
  // Function for Combinational Logic to read inputs
  function [SIZE-1:0] fsm_function;
    input  [SIZE-1:0] out_state;
    input in_anxy;
    input in_anxa;

  case(out_state)
    IDLE: begin
      if (in_anxy) begin
        fsm_function = LOADXY;
      end else begin
        fsm_function = IDLE;
      end
    end
    LOADXY: begin
      if (in_anxy) begin
        fsm_function = LOADXY;
      end else if (in_anxa) begin
        fsm_function = LOADXA;
      end else begin
        fsm_function = IDLE;
      end
    end
    LOADXA: begin
      if (in_anxy) begin
        fsm_function = LOADXY;
      end else if (in_anxa) begin
        fsm_function = LOADXA;
      end else begin
        fsm_function = IDLE;
      end
    end
    default: fsm_function = IDLE;
    endcase
  endfunction
  //-------------Seq Logic-----------------------------------
  always @ (negedge in_clka)
  begin : FSM_SEQ
```

```
    if (in_restart) begin
      next_out_state <= IDLE;
    end else begin
      next_out_state <= temp_out_state;
    end
  end
  //-------------Output Logic------------------------------
  always @ (negedge in_clkb)
  begin : OUTPUT_LOGIC
    case(next_out_state)
    IDLE: begin
            out_state <= next_out_state;
            out_loadx <= 1'b0;
            out_loady <= 1'b0;
            out_ac <= 1'b0;
          end
    LOADXY: begin
            out_state <= next_out_state;
            out_loadx <= 1'b1;
            out_loady <= 1'b1;
            out_ac <= 1'b1;
          end
    LOADXA: begin
            out_state <= next_out_state;
            out_loadx <= 1'b1;
            out_loady <= 1'b1;
            out_ac <= 1'b0;
            end
    default: begin
            out_state <= next_out_state;
            out_loadx <= 1'b0;
            out_loady <= 1'b0;
            out_ac <= 1'b0;
           end
    endcase
  end // End Of Block OUTPUT_LOGIC


  endmodule  // End of Module main_FSM
```

- DP

```
module dp (in_clka, in_clkb, in_restart, in_loadx, in_loady, in_ac, in_newx,
in_newy, out_al);
// Input Ports
input wire in_clka, in_clkb, in_restart, in_loadx, in_loady, in_ac;
input wire [3:0] in_newx, in_newy;
// Output Ports
output reg [3:0] out_al; //in_accum Latch
// Internal Variables
reg [3:0] xlatch, ylatch;
wire [3:0] tempres, tempy;
// Code Starts Here
assign tempy = in_ac? in_newy:out_al;
assign tempres = xlatch + ylatch;
```

```
always @ (negedge in_clka)
begin
if (in_restart) begin
   xlatch = 4'b0000;
   ylatch = 4'b0000;
   end else if (in_loadx & in_loady == 1'b1) begin
   xlatch = in_newx;
   ylatch = tempy;
   end else begin
   xlatch = xlatch;
   ylatch = ylatch;
   end
end

always @ (negedge in_clkb)
begin
if (in_restart == 1'b1) begin
   out_al  = 4'b0000;
   end else begin
   out_al  = tempres;
   end
end

endmodule //End Of Module dp  datapath
```

- Top Module

```
module top_module (in_clka, in_clkb, in_restart, in_anxy, in_anxa, out_state,
in_newx, in_newy, out_al);
// Input Ports
input  wire in_clka, in_clkb, in_restart, in_anxy, in_anxa;
input  wire [3:0] in_newx, in_newy;
// Output Ports
output wire [1:0] out_state;
output wire [3:0] out_al;
// Internout_al Variables
wire loadx, loady, ac;
// Code startes Here
main_FSM main_FSM_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_anxy ( in_anxy ),
  .in_anxa ( in_anxa ),
  .out_state (out_state ),
  .out_loadx (loadx ),
  .out_loady (loady ),
  .out_ac  ( ac)
);

dp dp_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
```

```
  .in_restart ( in_restart ),
  .in_loadx ( loadx ),
  .in_loady ( loady ),
  .in_ac ( ac ),
  .in_newx (in_newx ),
  .in_newy (in_newy ),
  .out_al  (out_al)
);
endmodule // End of Module top_module
```

- Verilog testbench file covering the test sequence described in words in the problem statement

```
module top_module_tb;
// Ports
reg  in_clka = 0;
reg  in_clkb = 0;
reg  in_restart = 0;
reg  in_anxy = 0;
reg  in_anxa = 0;
reg [3:0] in_newx;
reg [3:0]  in_newy;
wire [1:0] out_state;
wire [3:0]  out_al;

top_module
top_module_dut (
    .in_clka (in_clka ),
    .in_clkb ( in_clkb ),
    .in_restart ( in_restart ),
    .in_anxy ( in_anxy ),
    .in_anxa ( in_anxa ),
    .in_newx (in_newx ),
    .in_newy ( in_newy ),
    .out_state (out_state ),
    .out_al  (  out_al)
);

initial
begin

// Cycle 1
in_restart = 1;
in_anxy = 0;
in_anxa = 0;
in_newx = 0;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 2
in_restart = 0;
in_anxy = 1;
```

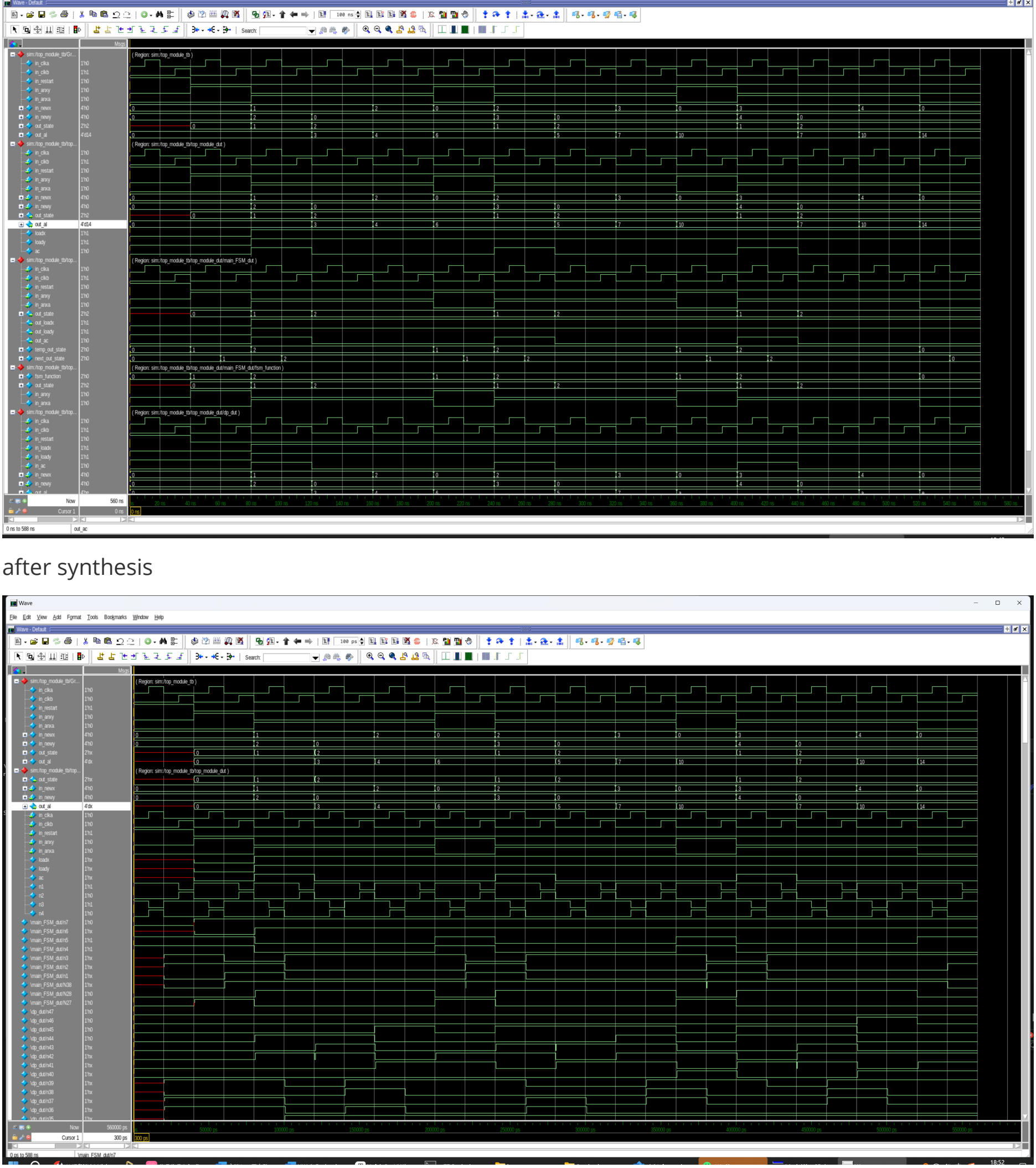
```
in_anxa = 0;
in_newx = 0;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 3
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 1;
in_newy = 2;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 4
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 1;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 5
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 2;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 6
in_restart = 0;
in_anxy = 1;
in_anxa = 0;
in_newx = 0;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 7
```

```
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 2;
in_newy = 3;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 8
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 2;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 9
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 3;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 10
in_restart = 0;
in_anxy = 1;
in_anxa = 0;
in_newx = 0;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 11
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 3;
in_newy = 4;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;
```

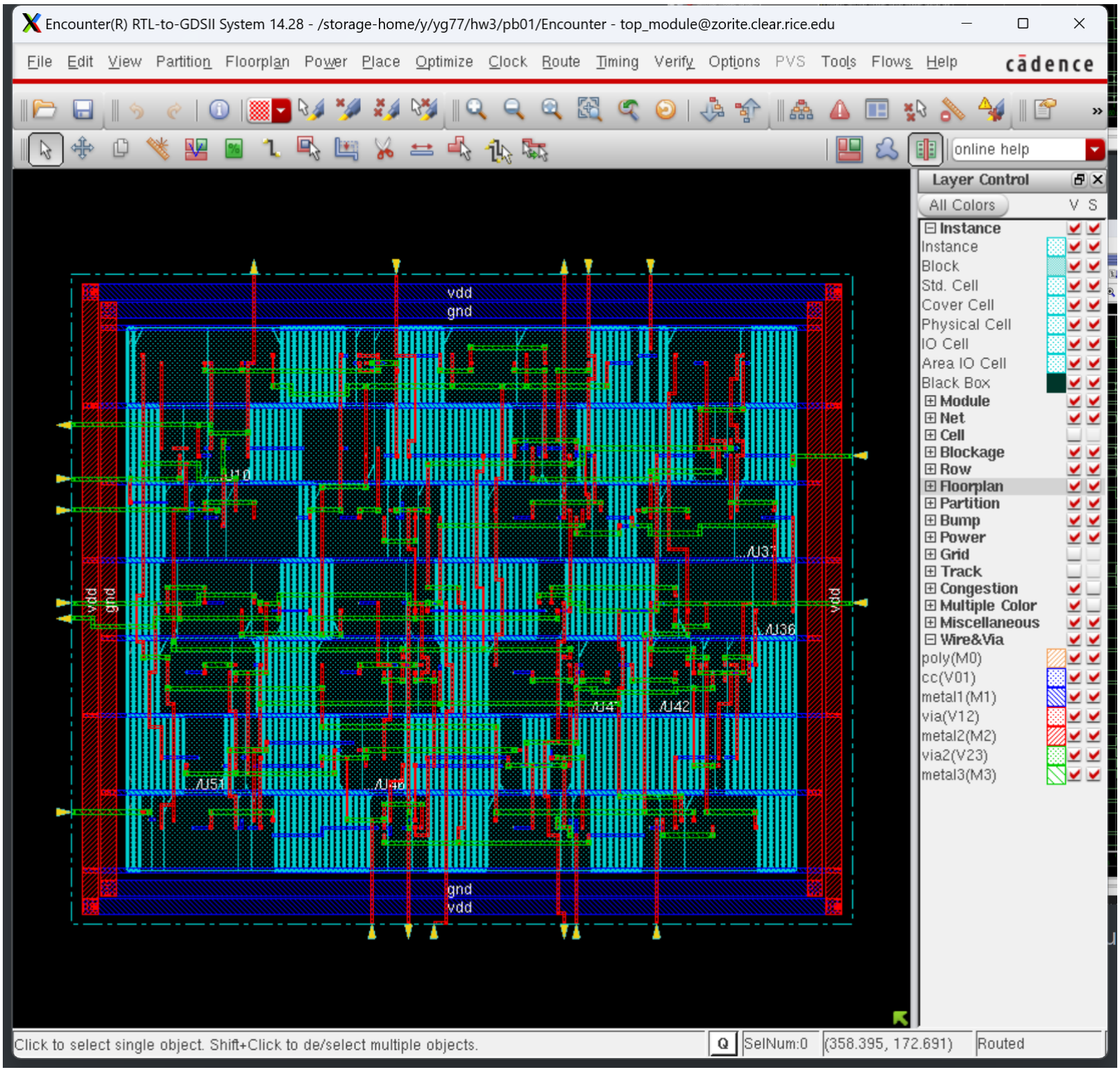
```
// Cycle 12
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 3;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 13
in_restart = 0;
in_anxy = 0;
in_anxa = 1;
in_newx = 4;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 14
in_restart = 0;
in_anxy = 0;
in_anxa = 0;
in_newx = 0;
in_newy = 0;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

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

$stop;
end
endmodule
```

- Modelsim output plot showing correct function both before and after synthesis
  - before synthesis

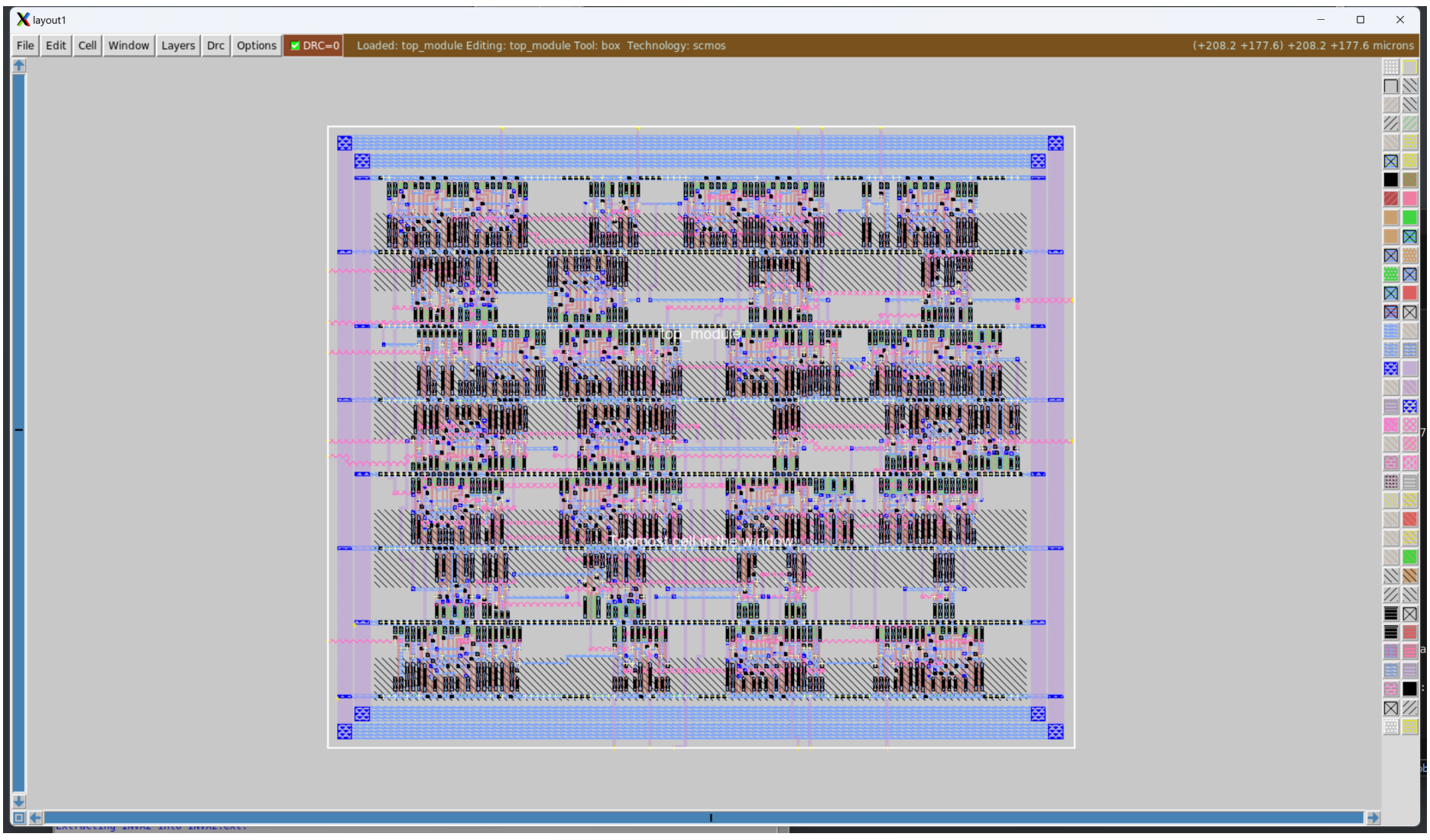
- after synthesis



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



- Sceenshot from Magic

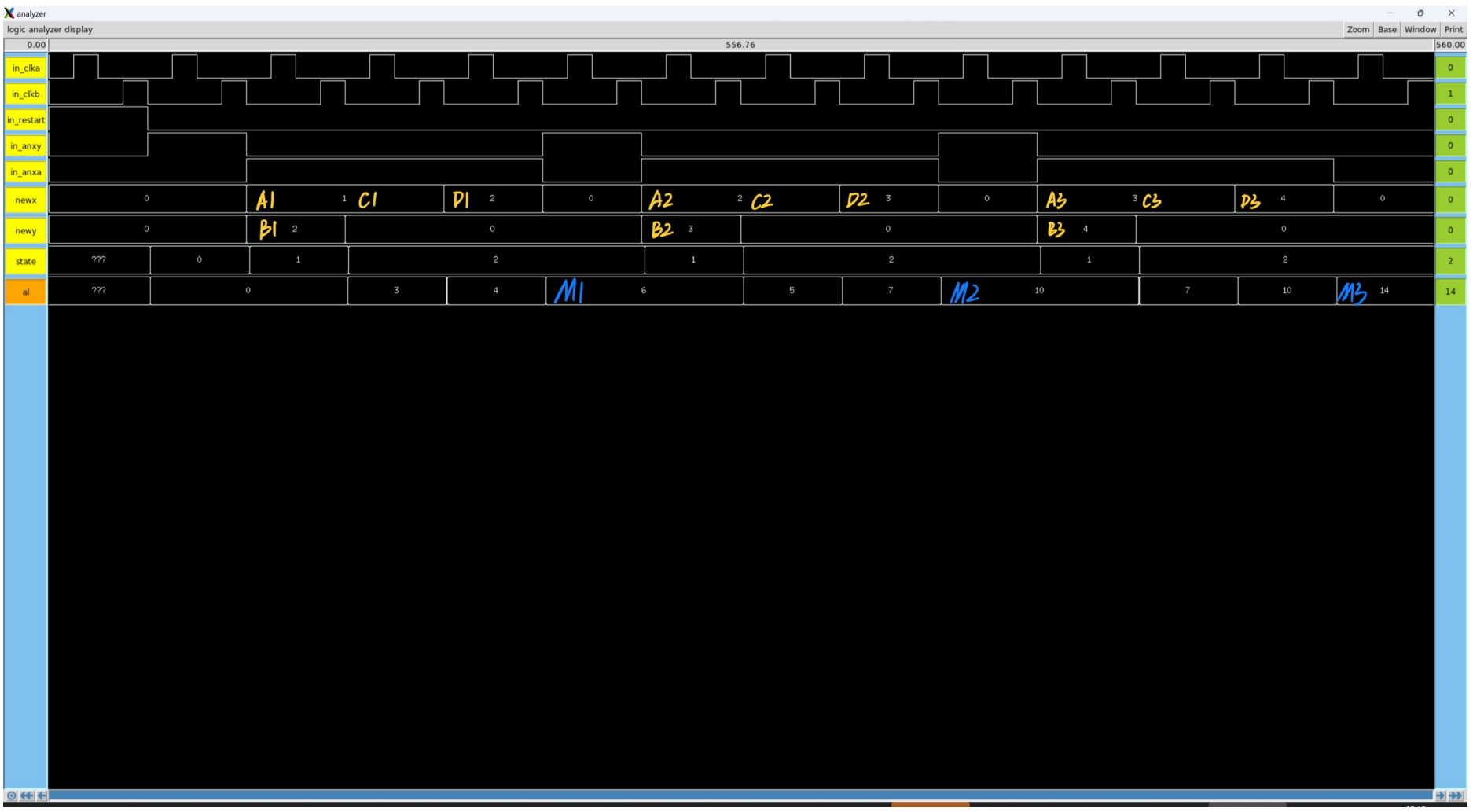


No any label or DRC errors.

- 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

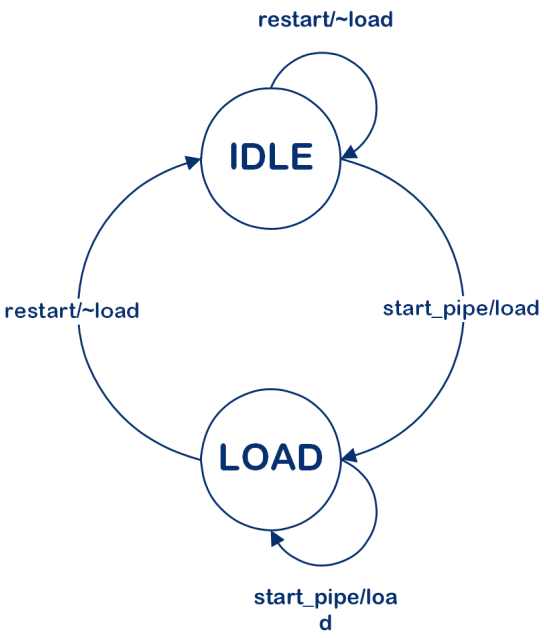
```
| define vectors for easier display
vector newx in_newx\[3\] in_newx\[2\] in_newx\[1\] in_newx\[0\]
vector newy in_newy\[3\] in_newy\[2\] in_newy\[1\] in_newy\[0\]
vector state out_state\[1\] out_state\[0\]
vector al out_al\[3\] out_al\[2\] out_al\[1\] out_al\[0\]
|
logfile main_FSM.log
ana in_clka in_clkb in_restart in_anxy in_anxa
ana newx newy state al
|
v   in_restart      1   0   0   0   0   0   0   0   0   0   0   0   0   0
v   in_anxy         0   1   0   0   0   1   0   0   0   1   0   0   0   0
v   in_anxa         0   0   1   1   1   0   1   1   1   0   1   1   1   0
v   in_newx\[3\]    0   0   0   0   0   0   0   0   0   0   0   0   0   0
v   in_newx\[2\]    0   0   0   0   0   0   0   0   0   0   0   0   1   0
v   in_newx\[1\]    0   0   0   0   1   0   1   1   1   0   1   1   0   0
v   in_newx\[0\]    0   0   1   1   0   0   0   0   1   0   1   1   0   0
v   in_newy\[3\]    0   0   0   0   0   0   0   0   0   0   0   0   0   0
v   in_newy\[2\]    0   0   0   0   0   0   0   0   0   0   1   0   0   0
v   in_newy\[1\]    0   0   1   0   0   0   1   0   0   0   0   0   0   0
v   in_newy\[0\]    0   0   0   0   0   0   1   0   0   0   0   0   0   0


| Two phase clock with non-overlap period - same as Modelsim testbench
clock in_clka 0 1 0 0
clock in_clkb 0 0 0 1
R
```

- output plot

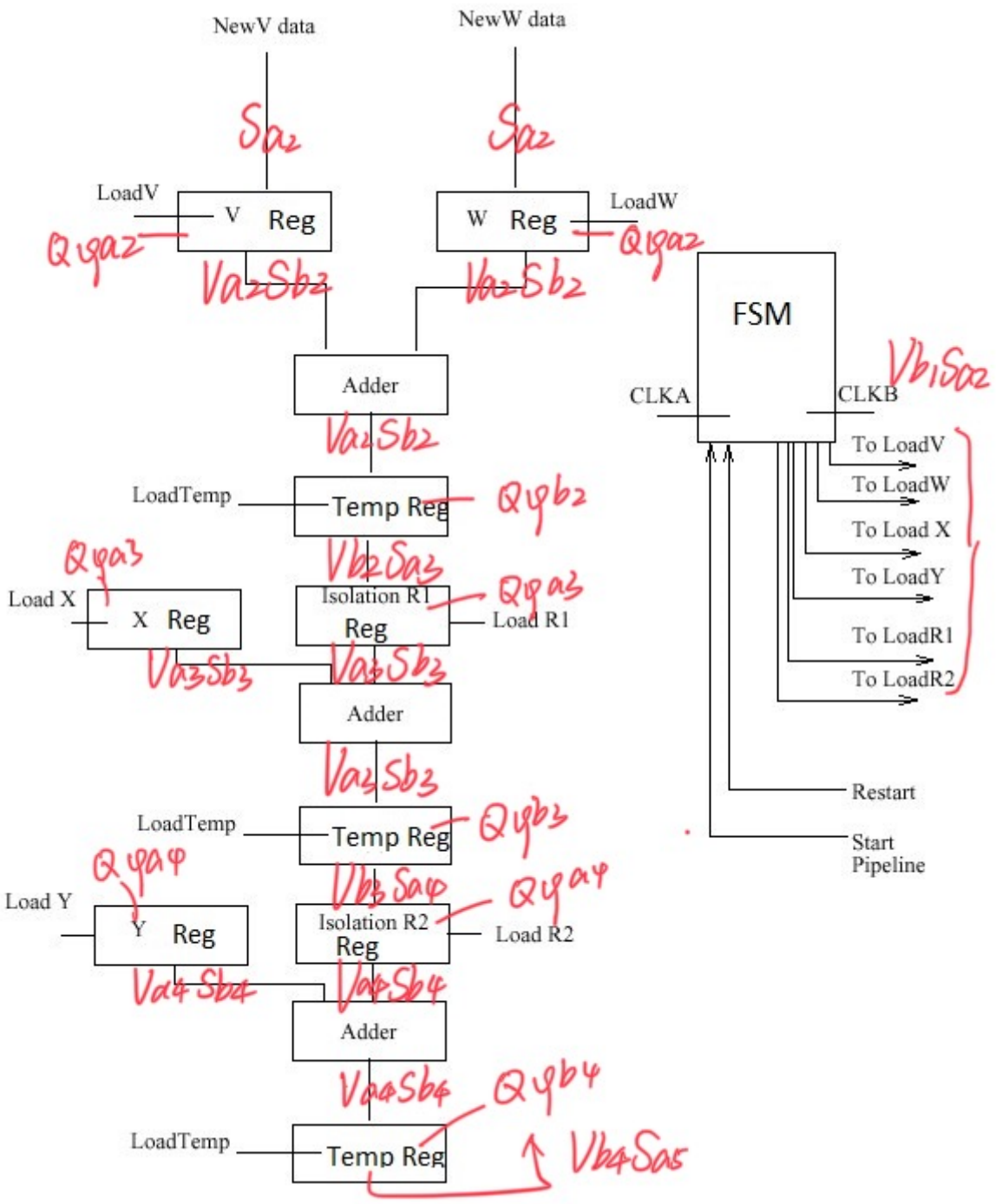


2. **Problem 2: Multiple Cycle System (Pipelined)**

- Timing analysis V,S labels on diagram
  - FSM diagram



- DP diagram



  - Verilog input module file using two phase clocking and datapath
    - FSM

```
module main_FSM (in_clka, in_clkb, in_restart, in_start_pipe, out_load, out_state);
  // Internal Constants
  parameter SIZE = 2;
  parameter IDLE = 2'b00, LOAD = 2'b01;
  // Input Ports
  input wire in_clka, in_clkb, in_restart, in_start_pipe;
  // Output Ports
  output reg [SIZE-1:0] out_state;
  output reg out_load;
  // Internal Variables
  wire [SIZE-1:0] temp_out_state;
  reg [SIZE-1:0] next_out_state;
```

```
// Code startes Here
assign temp_out_state = fsm_function(out_state, in_start_pipe);
// Function for Combinational Logic to read inputs
function [SIZE-1:0] fsm_function;
  input [SIZE-1:0] out_state;
  input in_start_pipe;

case(out_state)
  IDLE: begin
    if (in_start_pipe) begin
      fsm_function = LOAD;
    end else begin
      fsm_function = IDLE;
    end
  end
  LOAD: begin
    fsm_function = LOAD;
  end
  default: fsm_function = IDLE;
  endcase
endfunction
// Seq Logic
always @ (negedge in_clka)
begin : FSM_SEQ
  if (in_restart) begin
    next_out_state <= IDLE;
  end else begin
    next_out_state <= temp_out_state;
  end
end
// Output Logic
always @ (negedge in_clkb)
begin : OUTPUT_LOGIC
  case(next_out_state)
  IDLE: begin
          out_state <= next_out_state;
          out_load <= 0;
        end
    LOAD: begin
          out_state <= next_out_state;
          out_load <= 1;
        end
    default: begin
          out_state <= next_out_state;
          out_load <= 0;
         end
    endcase
  end // End Of Block OUTPUT_LOGIC

endmodule  // End of Module main_FSM
```

- DP

```
module dp (in_clka, in_clkb, in_restart, in_load, in_newv, in_neww, in_newx,
in_newy, out_al);
// Input Ports
input wire in_clka, in_clkb, in_restart, in_load;
input wire [3:0] in_newv, in_neww, in_newx, in_newy;
// Output Ports
output wire [3:0] out_al; //in_accum Latch
// Internout_al Variables
reg [3:0] xlatch, ylatch, vlatch, wlatch;
reg [3:0] templatch1, templatch2, templatch3;
reg [3:0] isolationr1, isolationr2;
// Code Starts Here

always @ (negedge in_clka) begin
   if (in_restart) begin
      vlatch <= 0;
      wlatch <= 0;
      xlatch <= 0;
      ylatch <= 0;
   end else if (in_load) begin
      vlatch <= in_newv;
      wlatch <= in_neww;
      xlatch <= in_newx;
      ylatch <= in_newy;
   end else begin
      vlatch <= vlatch;
      wlatch <= wlatch;
      xlatch <= xlatch;
      ylatch <= ylatch;
   end
end

always @ (negedge in_clka) begin
   if (in_restart) begin
      isolationr1 <= 0;
      isolationr2 <= 0;
   end else begin
      isolationr1 <= templatch1;
      isolationr2 <= templatch2;
   end
end

always @ (negedge in_clkb) begin
   if (in_restart) begin
      templatch1 <= 0;
      templatch2 <= 0;
      templatch3 <= 0;
   end else begin
      templatch1 <= vlatch + wlatch;
      templatch2 <= xlatch + isolationr1;
      templatch3 <= ylatch + isolationr2;
   end
end
```

```
assign out_al = templatch3;

endmodule //End Of Module dp  datapath
```

- Top Module

```
module top_module (in_clka, in_clkb, in_restart, in_start_pipe, in_newx, in_newy,
in_neww, in_newv, out_state, out_al);
// Input Ports
input  wire in_clka, in_clkb, in_restart, in_start_pipe;
input  wire [3:0] in_newx, in_newy, in_neww, in_newv;
// Output Ports
output wire [1:0] out_state;
output wire [3:0] out_al;
// Internout_al Variables
wire load;
// Code startes Here
main_FSM main_FSM_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_start_pipe ( in_start_pipe ),
  .out_state ( out_state ),
  .out_load  ( load)
);

dp dp_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_load ( load ),
  .in_newv (in_newv ),
  .in_neww ( in_neww ),
  .in_newx ( in_newx ),
  .in_newy ( in_newy ),
  .out_al  ( out_al)
);

endmodule // End of Module top_module
```

- Verilog testbench file covering the test sequence described in words in the problem statement

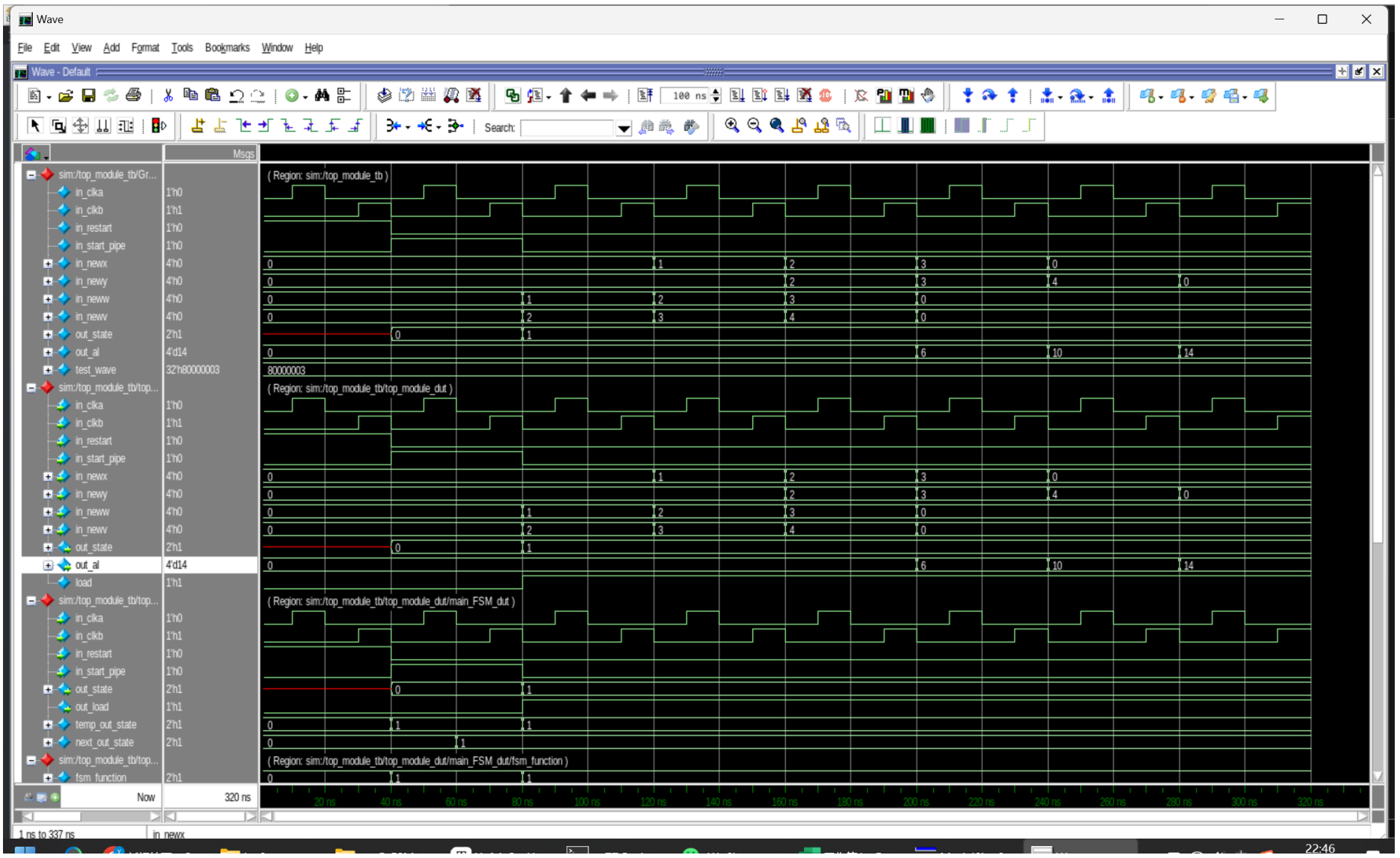
```
module top_module_tb;
// Ports
reg   in_clka;
reg   in_clkb;
reg   in_restart;
reg   in_start_pipe;
reg [3:0] in_newx;
reg [3:0] in_newy;
reg [3:0] in_neww;
reg [3:0] in_newv;
wire [1:0] out_state;
```

```
wire [3:0] out_al;

top_module top_module_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_start_pipe ( in_start_pipe ),
  .in_newx ( in_newx ),
  .in_newy ( in_newy ),
  .in_neww ( in_neww ),
  .in_newv ( in_newv ),
  .out_state (out_state ),
  .out_al  ( out_al)
);

initial
begin
// Cycle 1
in_restart              = 1 ;
in_start_pipe           = 0 ;
in_newx                 = 0 ;
in_newy                 = 0 ;
in_neww                 = 0 ;
in_newv                 = 0 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10

// Cycle 2
in_restart              = 0 ;
in_start_pipe           = 1 ;
in_newx                 = 0 ;
in_newy                 = 0 ;
in_neww                 = 0 ;
in_newv                 = 0 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10

// Cycle 3
in_restart              = 0 ;
in_start_pipe           = 0 ;
in_newx                 = 0 ;
in_newy                 = 0 ;
in_neww                 = 1 ;
in_newv                 = 2 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10
```

```
// Cycle 4
in_restart              = 0 ;
in_start_pipe           = 0 ;
in_newx                 = 1 ;
in_newy                 = 0 ;
in_neww                 = 2 ;
in_newv                 = 3 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10

// Cycle 5
in_restart              = 0 ;
in_start_pipe           = 0 ;
in_newx                 = 2 ;
in_newy                 = 2 ;
in_neww                 = 3 ;
in_newv                 = 4 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10

// Cycle 6
in_restart              = 0 ;
in_start_pipe           = 0 ;
in_newx                 = 3 ;
in_newy                 = 3 ;
in_neww                 = 0 ;
in_newv                 = 0 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 7
in_restart              = 0 ;
in_start_pipe           = 0 ;
in_newx                 = 0 ;
in_newy                 = 4 ;
in_neww                 = 0 ;
in_newv                 = 0 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

// Cycle 8
in_restart              = 0 ;
in_start_pipe           = 0 ;
in_newx                 = 0 ;
in_newy                 = 0 ;
```

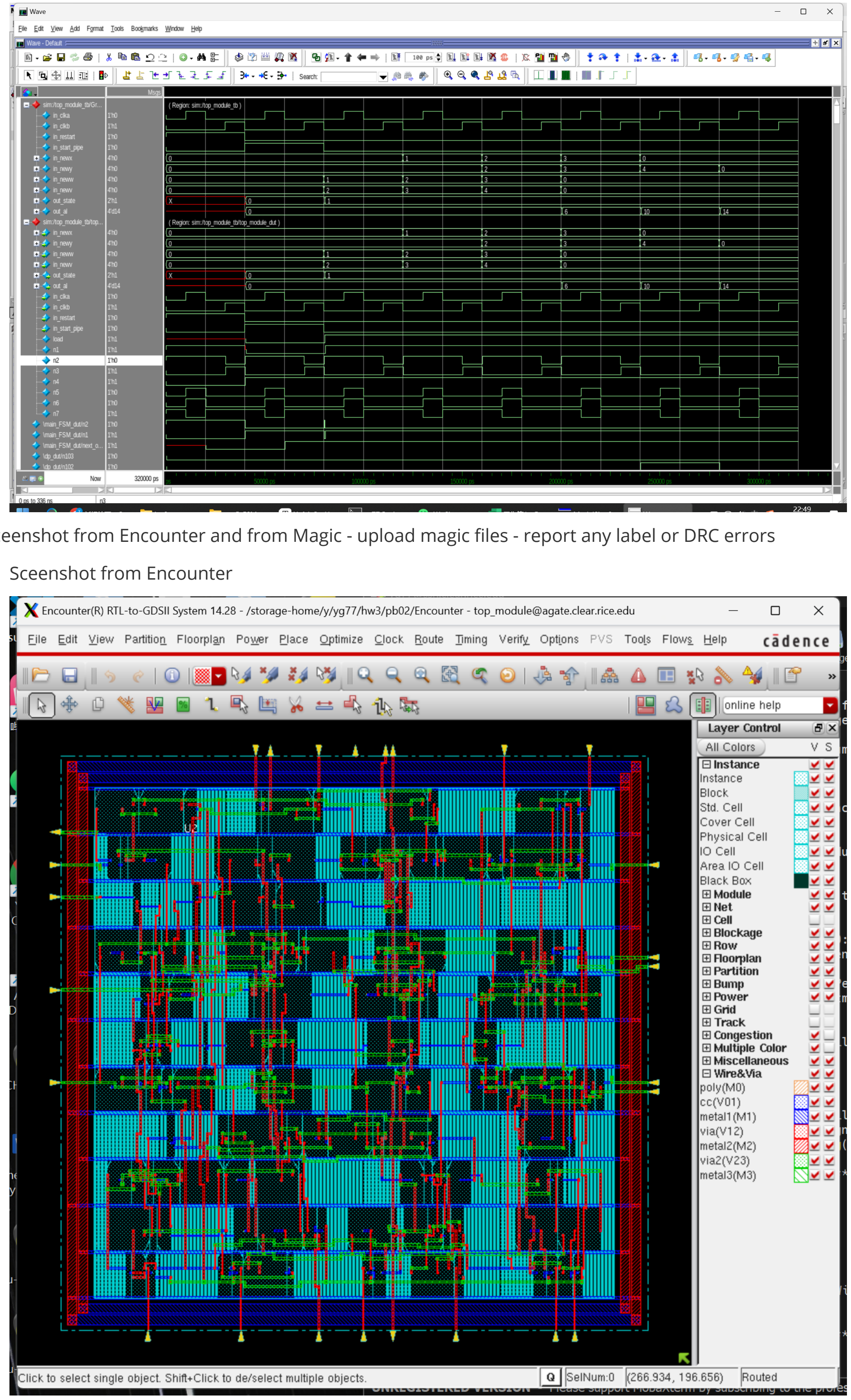
```
in_neww                 = 0 ;
in_newv                 = 0 ;
in_clka = 0; in_clkb = 0; #10;
in_clka = 1; in_clkb = 0; #10;
in_clka = 0; in_clkb = 0; #10;
in_clka = 0; in_clkb = 1; #10;

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

$stop;
end

endmodule
```

- Modelsim output plot showing correct function both before and after synthesis
  - before synthesis

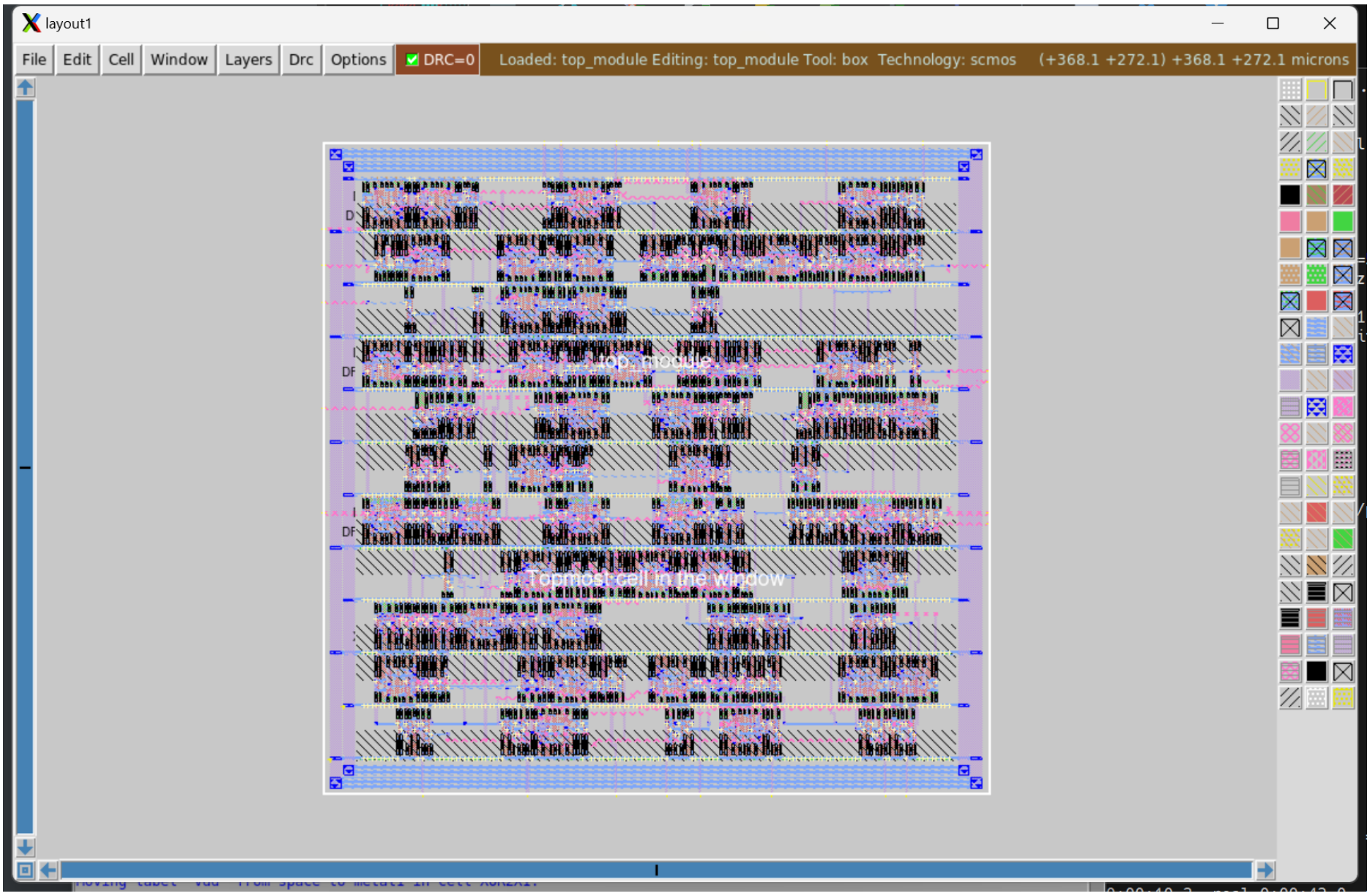


  - after synthesis

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



- Sceenshot from Magic



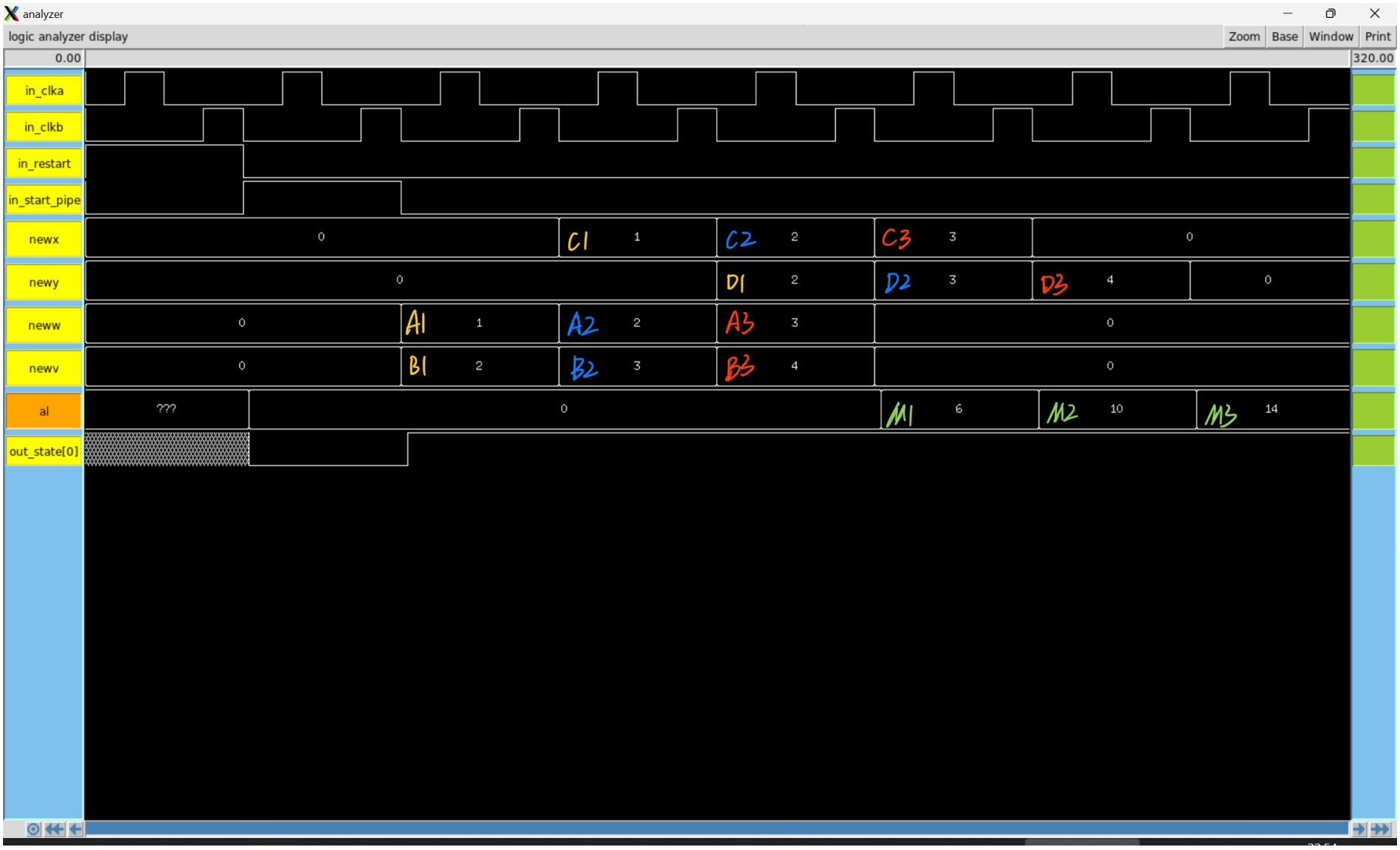
No any label or DRC errors.

- 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

```
| define vectors for easier display
vector newx in_newx\[3\] in_newx\[2\] in_newx\[1\] in_newx\[0\]
vector newy in_newy\[3\] in_newy\[2\] in_newy\[1\] in_newy\[0\]
vector neww in_neww\[3\] in_neww\[2\] in_neww\[1\] in_neww\[0\]
vector newv in_newv\[3\] in_newv\[2\] in_newv\[1\] in_newv\[0\]
vector al out_al\[3\] out_al\[2\] out_al\[1\] out_al\[0\]
|
logfile main_FSM.log
ana in_clka in_clkb in_restart in_start_pipe
ana newx newy neww newv al out_state\[0\]
|
v    in_restart     1   0   0   0   0   0   0   0
v    in_start_pipe  0   1   0   0   0   0   0   0
v    in_newx\[3\]   0   0   0   0   0   0   0   0
v    in_newx\[2\]   0   0   0   0   0   0   0   0
v    in_newx\[1\]   0   0   0   0   1   1   0   0
v    in_newx\[0\]   0   0   0   1   0   1   0   0
v    in_newy\[3\]   0   0   0   0   0   0   0   0
v    in_newy\[2\]   0   0   0   0   0   0   1   0
v    in_newy\[1\]   0   0   0   0   1   1   0   0
v    in_newy\[0\]   0   0   0   0   0   1   0   0
v    in_neww\[3\]   0   0   0   0   0   0   0   0
v    in_neww\[2\]   0   0   0   0   0   0   0   0
v    in_neww\[1\]   0   0   0   1   1   0   0   0
v    in_neww\[0\]   0   0   1   0   1   0   0   0
```

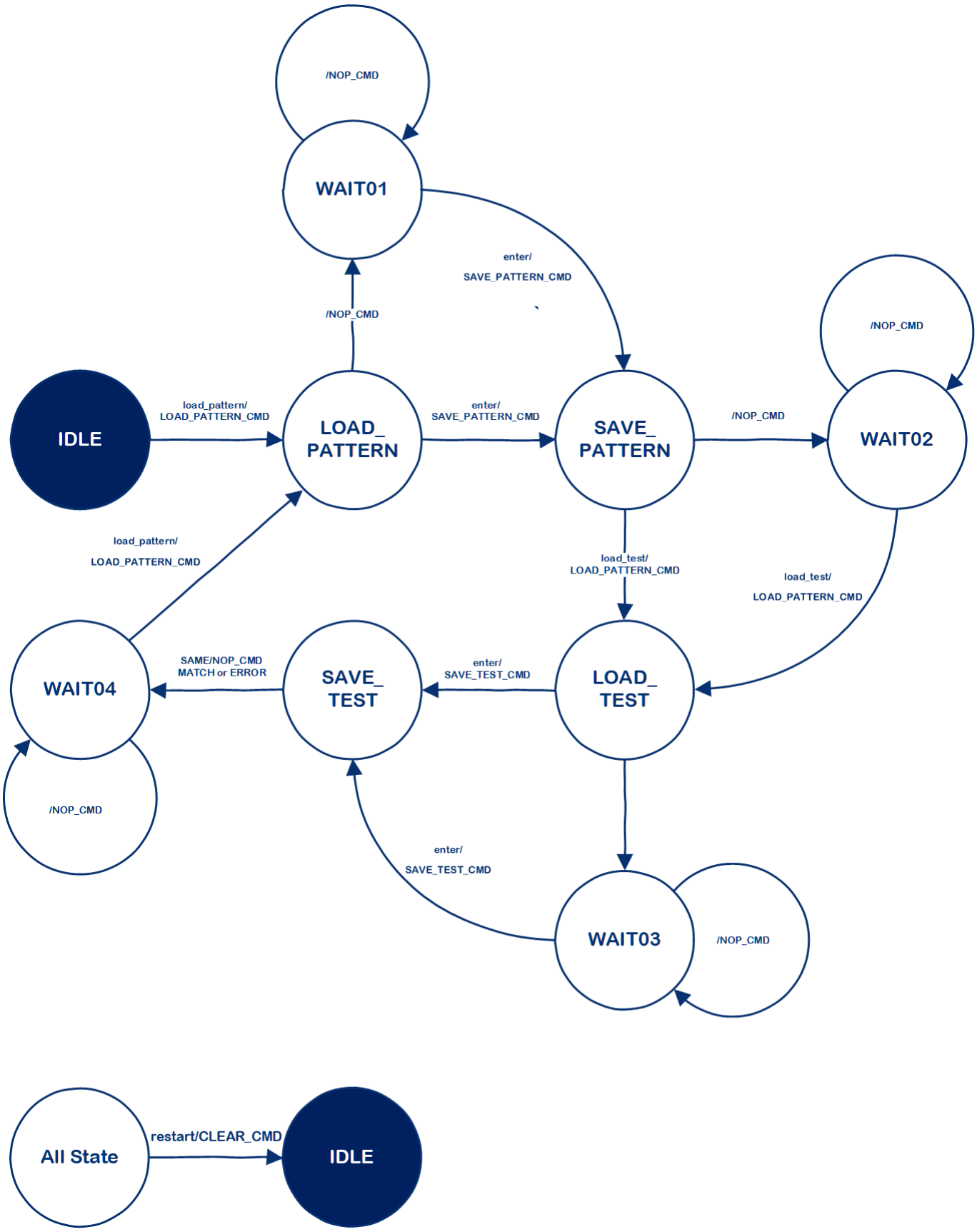
```
V    in_newv\[3\]   0   0   0   0   0   0   0   0
V    in_newv\[2\]   0   0   0   0   1   0   0   0
V    in_newv\[1\]   0   0   1   1   0   0   0   0
V    in_newv\[0\]   0   0   0   1   0   0   0   0

| Two phase clock with non-overlap period - same as Modelsim testbench
clock in_clka 0 1 0 0
clock in_clkb 0 0 0 1
R
```

- output plot

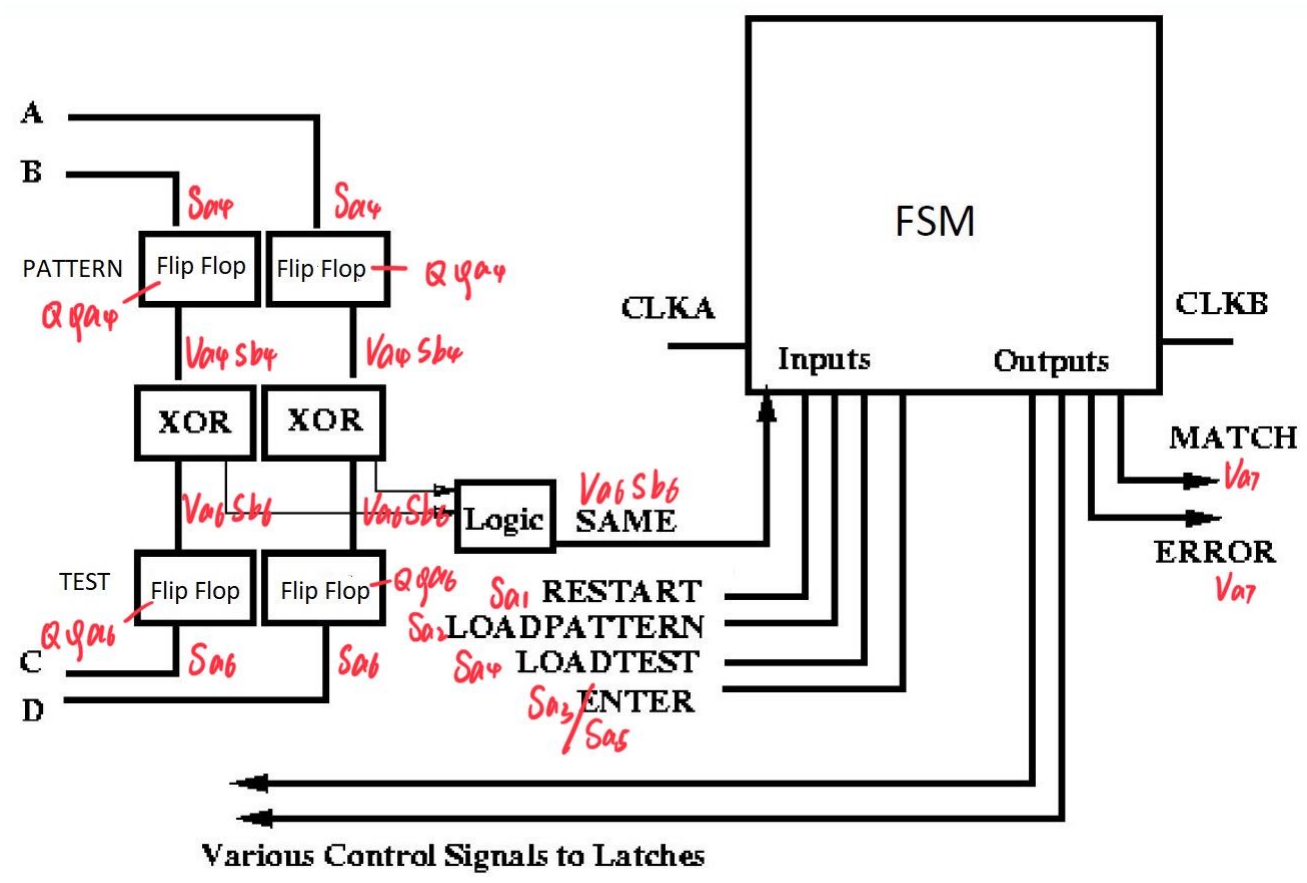


3. **Problem 3: Code Verifier (Simple Combination Lock)**

- State transition diagram and block diagram showing datapath interconnections and V,S timing labels
  - FSM



- DP



  - Verilog input module file for top, FSM, and datapath using two phase clocking
    - FSM

```
module main_FSM (in_clka, in_clkb, in_restart, in_loadpattern, in_loadtest,
in_enter, in_same, out_match, out_error, out_state, out_cmd);
  // Internal Constants
```

```
  parameter IDLE = 4'd0, LOAD_PATTERN = 4'd1, WAIT01 = 4'd2, SAVE_PATTERN = 4'd3,
WAIT02 = 4'd4,
            LOAD_TEST = 4'd5, WAIT03 = 4'd6, SAVE_TEST = 4'd7, WAIT04 = 4'd8;
  parameter LOAD_PATTERN_CMD = 3'd0,  SAVE_PATTERN_CMD = 3'd1,
            LOAD_TEST_CMD = 3'd2,     SAVE_TEST_CMD = 3'd3, CLEAR_CMD = 3'd4,
NOP_CMD = 3'd5;
  // Input Ports
  input wire in_clka, in_clkb, in_restart, in_loadpattern, in_loadtest, in_enter,
in_same;
  // Output Ports
  output reg [3:0] out_state;
  output reg [2:0] out_cmd;
  output reg out_match, out_error;
  // Internal Variables
  wire [3:0] temp_out_state;
  reg [3:0] next_out_state;
  // Code startes Here
  assign temp_out_state = fsm_function(out_state, in_loadpattern, in_loadtest,
in_enter);
  // Function for Combinational Logic to read inputs
  function [3:0] fsm_function;
    input [3:0] out_state;
    input in_loadpattern, in_loadtest, in_enter;

  case(out_state)
    IDLE: begin
      if (in_loadpattern) begin
        fsm_function = LOAD_PATTERN;
      end else begin
        fsm_function = IDLE;
      end
    end
    LOAD_PATTERN: begin
      if (in_enter) begin
        fsm_function = SAVE_PATTERN;
      end else begin
        fsm_function = WAIT01;
      end
    end
    WAIT01: begin
      if (in_enter) begin
        fsm_function = SAVE_PATTERN;
      end else begin
        fsm_function = WAIT01;
      end
    end
    SAVE_PATTERN: begin
      if (in_loadtest) begin
        fsm_function = LOAD_TEST;
      end else begin
        fsm_function = WAIT02;
      end
    end
```

```
    WAIT02: begin
      if (in_loadtest) begin
        fsm_function = LOAD_TEST;
      end else begin
        fsm_function = WAIT02;
      end
    end
    LOAD_TEST: begin
      if (in_enter) begin
        fsm_function = SAVE_TEST;
      end else begin
        fsm_function = WAIT03;
      end
    end
    WAIT03: begin
      if (in_enter) begin
        fsm_function = SAVE_TEST;
      end else begin
        fsm_function = WAIT03;
      end
    end
    SAVE_TEST: begin
      fsm_function = WAIT04;
    end
    WAIT04: begin
      if (in_loadpattern) begin
        fsm_function = LOAD_PATTERN;
      end else begin
        fsm_function = WAIT04;
      end
    end
    default: fsm_function = IDLE;
    endcase
  endfunction

  // Seq Logic
  always @ (negedge in_clka)
  begin : FSM_SEQ
    if (in_restart) begin
      next_out_state <= IDLE;
    end else begin
      next_out_state <= temp_out_state;
    end
  end

  // Output Logic
  always @ (negedge in_clkb)
  begin : OUTPUT_LOGIC
    case(next_out_state)
    IDLE: begin
            out_state <= next_out_state;
            out_cmd <= CLEAR_CMD;
            out_match <= 0;
```

```
            out_error <= 0;
          end
    LOAD_PATTERN: begin
            out_state <= next_out_state;
            out_cmd <= LOAD_PATTERN_CMD;
          end
    WAIT01: begin
            out_state <= next_out_state;
            out_cmd <= NOP_CMD;
          end
    SAVE_PATTERN: begin
            out_state <= next_out_state;
            out_cmd <= SAVE_PATTERN_CMD;
          end
    WAIT02: begin
            out_state <= next_out_state;
            out_cmd <= NOP_CMD;
          end
    LOAD_TEST: begin
            out_state <= next_out_state;
            out_cmd <= LOAD_TEST_CMD;
          end
    WAIT03: begin
            out_state <= next_out_state;
            out_cmd <= NOP_CMD;
          end
    SAVE_TEST: begin
            out_state <= next_out_state;
            out_cmd <= SAVE_TEST_CMD;
          end
    WAIT04: begin
            out_state <= next_out_state;
            out_cmd <= NOP_CMD;
            out_match <= in_same;
            out_error <= ~in_same;
          end
    default: begin
              out_state <= next_out_state;
              out_cmd <= NOP_CMD;
            end
    endcase
  end // End Of Block OUTPUT_LOGIC

endmodule  // End of Module main_FSM
```

- DP

```
module dp (in_clka, in_clkb, in_restart, in_cmd, in_a, in_b, in_c, in_d, out_same);
// Internal Constants
parameter    LOAD_PATTERN_CMD = 3'd0,   SAVE_PATTERN_CMD = 3'd1,
             LOAD_TEST_CMD = 3'd2,      SAVE_TEST_CMD = 3'd3, CLEAR_CMD = 3'd4,
NOP_CMD = 3'd5;
// Input Ports
input wire in_clka, in_clkb, in_restart;
```

```
input wire [2:0] in_cmd;
input wire in_a, in_b, in_c, in_d;
// Output Ports
output reg out_same;
// Internout_al Variables
reg alatch, blatch;
reg atemp, btemp, ctemp, dtemp;
// Code Starts Here
always @(negedge in_clka) begin
   if((in_restart) | (in_cmd == CLEAR_CMD))begin
      alatch <= 0;
      blatch <= 0;
      atemp  <= 0;
      btemp  <= 0;
      ctemp  <= 0;
      dtemp  <= 0;
      out_same <= 0;
   end else if(in_cmd == LOAD_PATTERN_CMD) begin
      atemp  <= in_a;
      btemp  <= in_b;
   end else if(in_cmd == SAVE_PATTERN_CMD) begin
      alatch <= atemp;
      blatch <= btemp;
   end else if(in_cmd == LOAD_TEST_CMD) begin
      ctemp <= in_c;
      dtemp <= in_d;
   end else if(in_cmd == SAVE_TEST_CMD) begin
      out_same <= (~(alatch^dtemp))&(~(blatch^ctemp));
   end else begin
      alatch <= alatch;
      blatch <= blatch;
      atemp  <= atemp;
      btemp  <= btemp;
      ctemp  <= ctemp;
      dtemp  <= dtemp;
   end

endmodule //End Of Module dp  datapath
```

- Top Module

```
module top_module (in_clka, in_clkb, in_restart, in_loadpattern, in_loadtest,
in_enter, in_a, in_b, in_c, in_d, out_match, out_error, out_state);
// Input Ports
input  wire in_clka, in_clkb, in_restart, in_loadpattern, in_loadtest, in_enter;
input  wire in_a, in_b, in_c, in_d;
// Output Ports
output wire [3:0] out_state;
output wire out_match, out_error;
// Internout_al Variables
wire [2:0] cmd;
wire same;
// Code startes Here
```

```
main_FSM main_FSM_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_loadpattern ( in_loadpattern ),
  .in_loadtest ( in_loadtest ),
  .in_enter ( in_enter ),
  .in_same ( same ),
  .out_state (out_state ),
  .out_cmd ( cmd ),
  .out_match (out_match ),
  .out_error  ( out_error)
);

dp dp_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_cmd ( cmd ),
  .in_a ( in_a ),
  .in_b ( in_b ),
  .in_c ( in_c ),
  .in_d ( in_d ),
  .out_same  ( same)
);

endmodule // End of Module top_module
```

- Verilog testbench file covering the test sequence in the problem statement

```
module top_module_tb;
// Ports
reg  in_clka = 0;
reg  in_clkb = 0;
reg  in_restart = 0;
reg  in_loadpattern = 0;
reg  in_loadtest = 0;
reg  in_enter = 0;
reg  in_a = 0;
reg  in_b = 0;
reg  in_c = 0;
reg  in_d = 0;
wire [3:0] out_state;
wire out_match;
wire out_error;

top_module top_module_dut (
  .in_clka (in_clka ),
  .in_clkb ( in_clkb ),
  .in_restart ( in_restart ),
  .in_loadpattern ( in_loadpattern ),
  .in_loadtest ( in_loadtest ),
  .in_enter ( in_enter ),
  .in_a (in_a ),
```

```
  .in_b ( in_b ),
  .in_c ( in_c ),
  .in_d ( in_d ),
  .out_state (out_state ),
  .out_match (out_match ),
  .out_error (out_error )
);

integer test_wave;

initial begin
    test_wave = $fopen("./test_wave.txt", "w");
    $fwrite(test_wave, "in_restart in_anxy in_anxa\n");
end

always @(negedge in_clkb) begin
    $fwrite(test_wave, "%d %d %d %d %d %d %d %d\n",
    in_restart,
     in_loadpattern,
  in_loadtest,
  in_enter,
  in_a,
  in_b,
  in_c,
  in_d
);
end

initial
begin
// Cycle 1
  in_restart              = 1   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 2
  in_restart              = 0   ;
  in_loadpattern        =   1   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
```

```
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 3
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   1   ;
  in_a                    = 1   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 4
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 1   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 5
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   1   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 1   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 6
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
```

```
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 7
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 8
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 9
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 10
  in_restart              = 0   ;
  in_loadpattern        =   1   ;
```

```
  in_loadtest              = 0   ;
  in_enter               =   0   ;
  in_a                     = 0   ;
  in_b                     = 0   ;
  in_c                     = 0   ;
  in_d                     = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 11
  in_restart               = 0   ;
  in_loadpattern         =   0   ;
  in_loadtest              = 0   ;
  in_enter               =   0   ;
  in_a                     = 1   ;
  in_b                     = 0   ;
  in_c                     = 0   ;
  in_d                     = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 12
  in_restart               = 0   ;
  in_loadpattern         =   0   ;
  in_loadtest              = 0   ;
  in_enter               =   1   ;
  in_a                     = 0   ;
  in_b                     = 0   ;
  in_c                     = 0   ;
  in_d                     = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 13
  in_restart               = 0   ;
  in_loadpattern         =   0   ;
  in_loadtest              = 1   ;
  in_enter               =   0   ;
  in_a                     = 0   ;
  in_b                     = 0   ;
  in_c                     = 0   ;
  in_d                     = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;
```

```
// Cycle 14
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 1   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 15
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   1   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 16
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 17
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
```

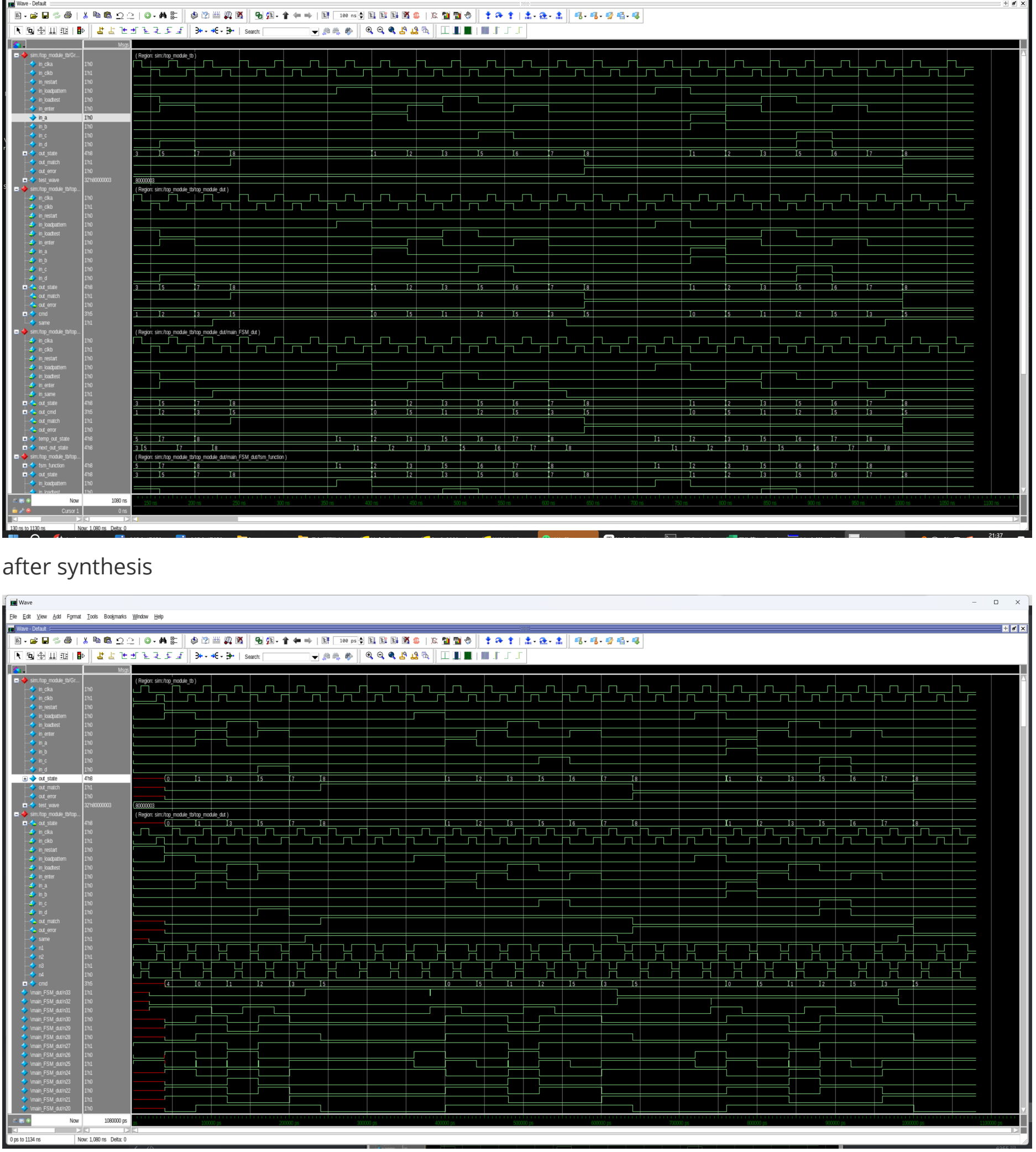
```
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 18
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 19
  in_restart              = 0   ;
  in_loadpattern        =   1   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 20
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 1   ;
  in_b                    = 1   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 21
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   1   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
```

```
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 22
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 1   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 23
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 1   ;
  in_d                    = 1   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 24
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   1   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 25
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
```

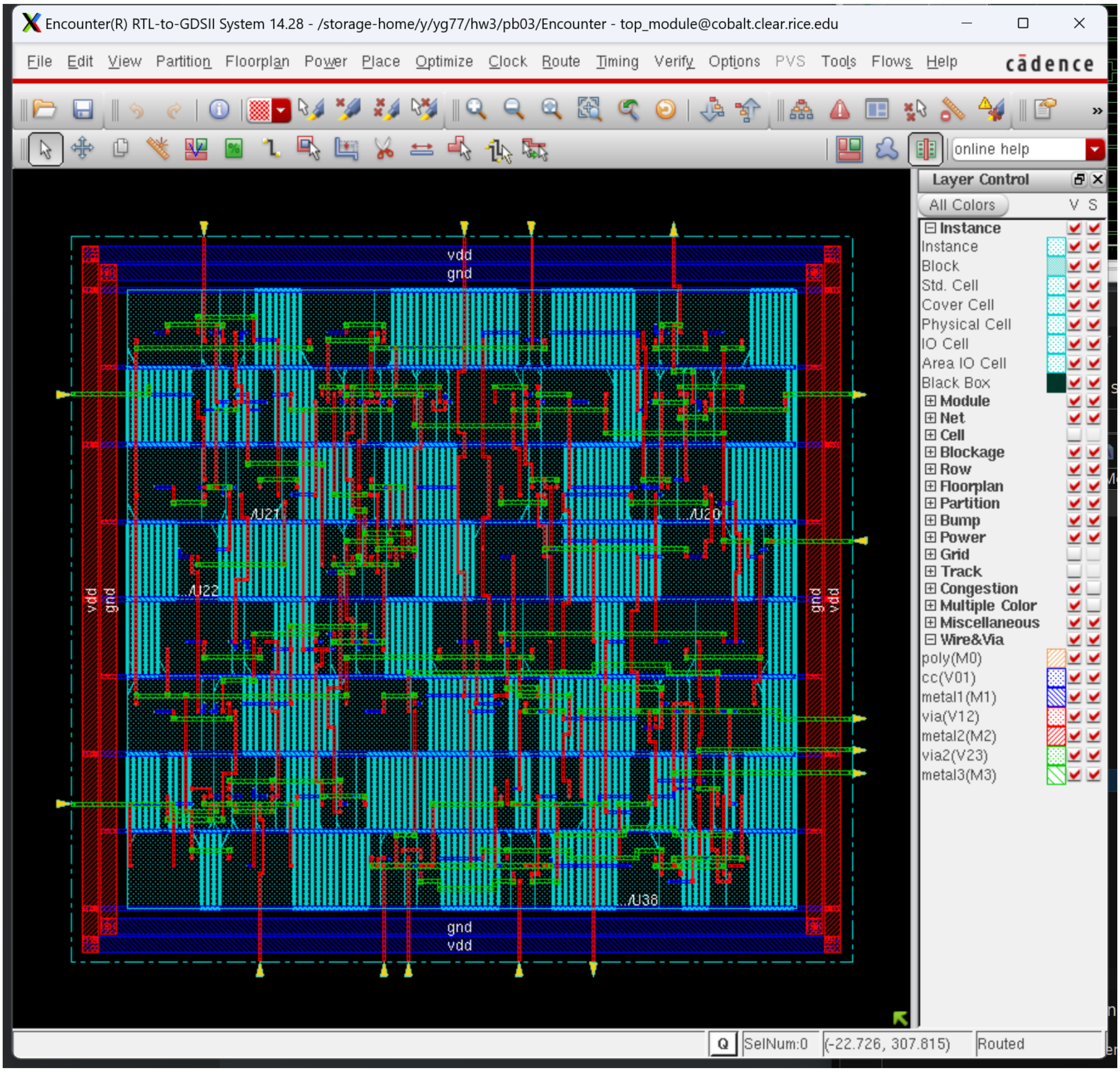
```
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 26
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

// Cycle 27
  in_restart              = 0   ;
  in_loadpattern        =   0   ;
  in_loadtest             = 0   ;
  in_enter              =   0   ;
  in_a                    = 0   ;
  in_b                    = 0   ;
  in_c                    = 0   ;
  in_d                    = 0   ;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 1; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 0; #10;
  in_clka = 0; in_clkb = 1; #10;

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

$stop;
end

endmodule
```

- Modelsim output plot showing correct function both before and after synthesis
  - before synthesis

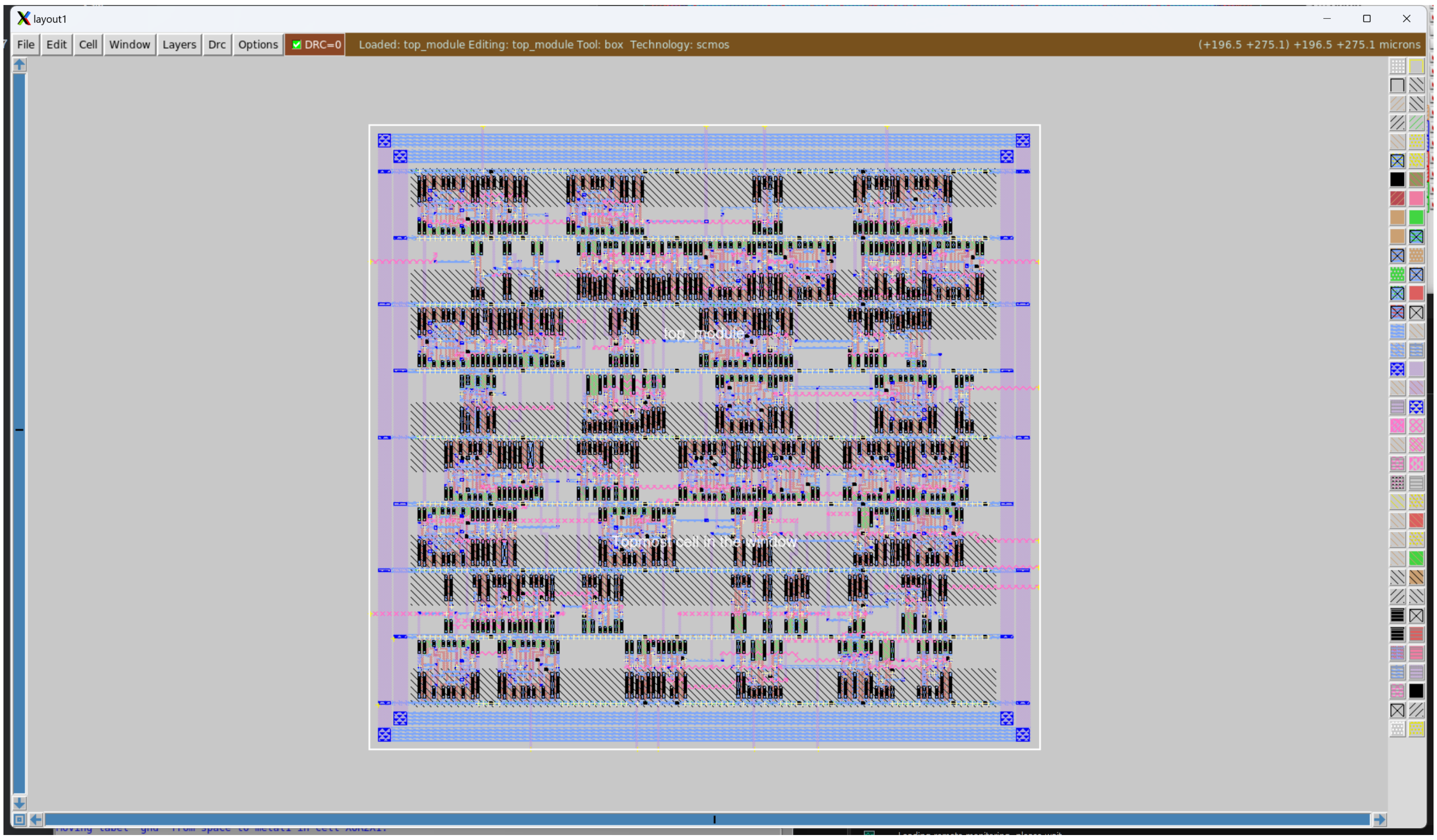
- after synthesis



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



- Sceenshot from Magic



No any label or DRC errors.

- 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

```
| define vectors for easier display
vector state out_state\[3\] out_state\[2\] out_state\[1\] out_state\[0\]
|
logfile main_FSM.log
ana in_clka in_clkb in_restart in_loadpattern in_loadtest in_enter
ana in_a in_b in_c in_d out_match out_error state
|
v    in_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
v    in_loadpattern      0  1   0   0   0   0   0   0   0   1   0   0   0   0   0
0   0   0   1   0   0   0   0   0   0   0   0
v    in_loadtest         0  0   0   1   0   0   0   0   0   0   0   0   1   0   0
0   0   0   0   0   0   1   0   0   0   0   0
v    in_enter            0  0   1   0   1   0   0   0   0   0   0   1   0   0   1
0   0   0   0   0   1   0   0   1   0   0   0
v    in_a                0  0   1   0   0   0   0   0   0   0   1   0   0   0   0
0   0   0   0   1   0   0   0   0   0   0   0
v    in_b                0  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   1   0   0   0   0   0   0   0
v    in_c                0  0   0   0   0   0   0   0   0   0   0   0   0   1   0
0   0   0   0   0   0   0   1   0   0   0   0
v    in_d                0  0   0   0   1   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   1   0   0   0   0

| Two phase clock with non-overlap period - same as Modelsim testbench
clock in_clka 0 1 0 0
clock in_clkb 0 0 0 1
R
```

- output plot