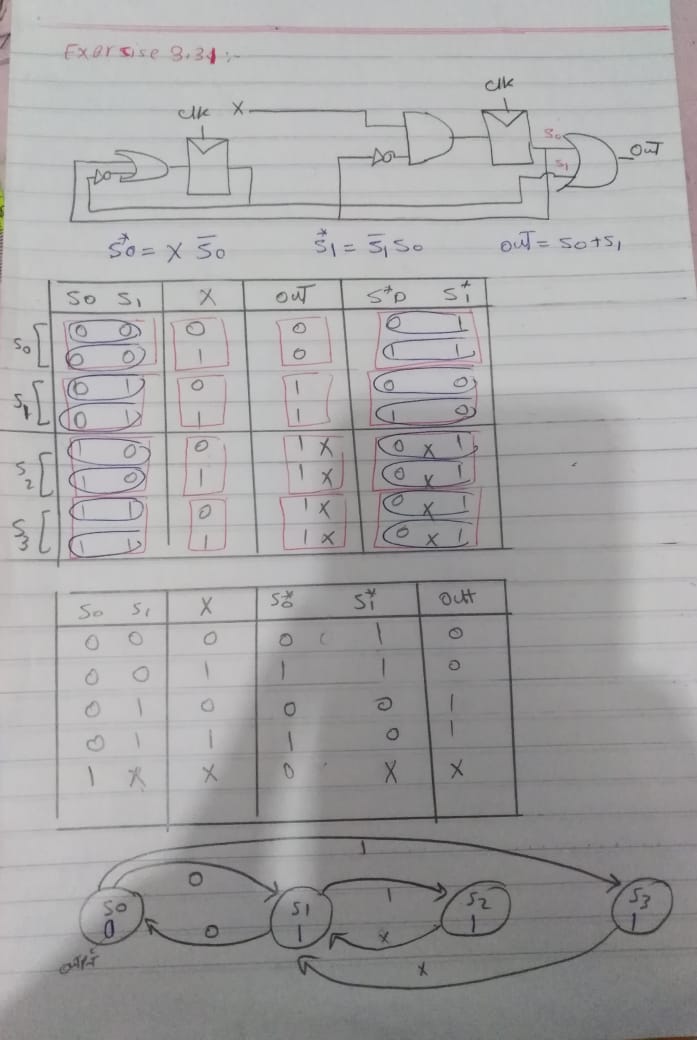
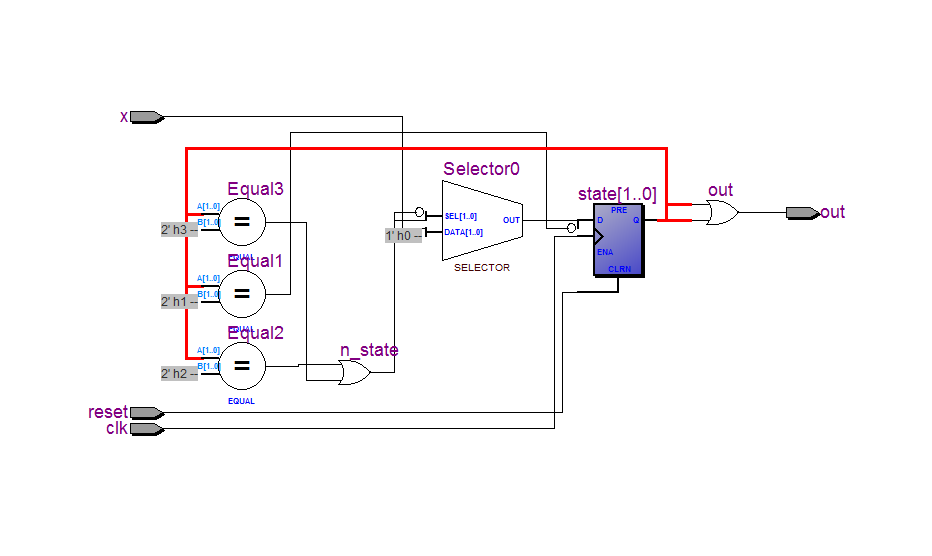
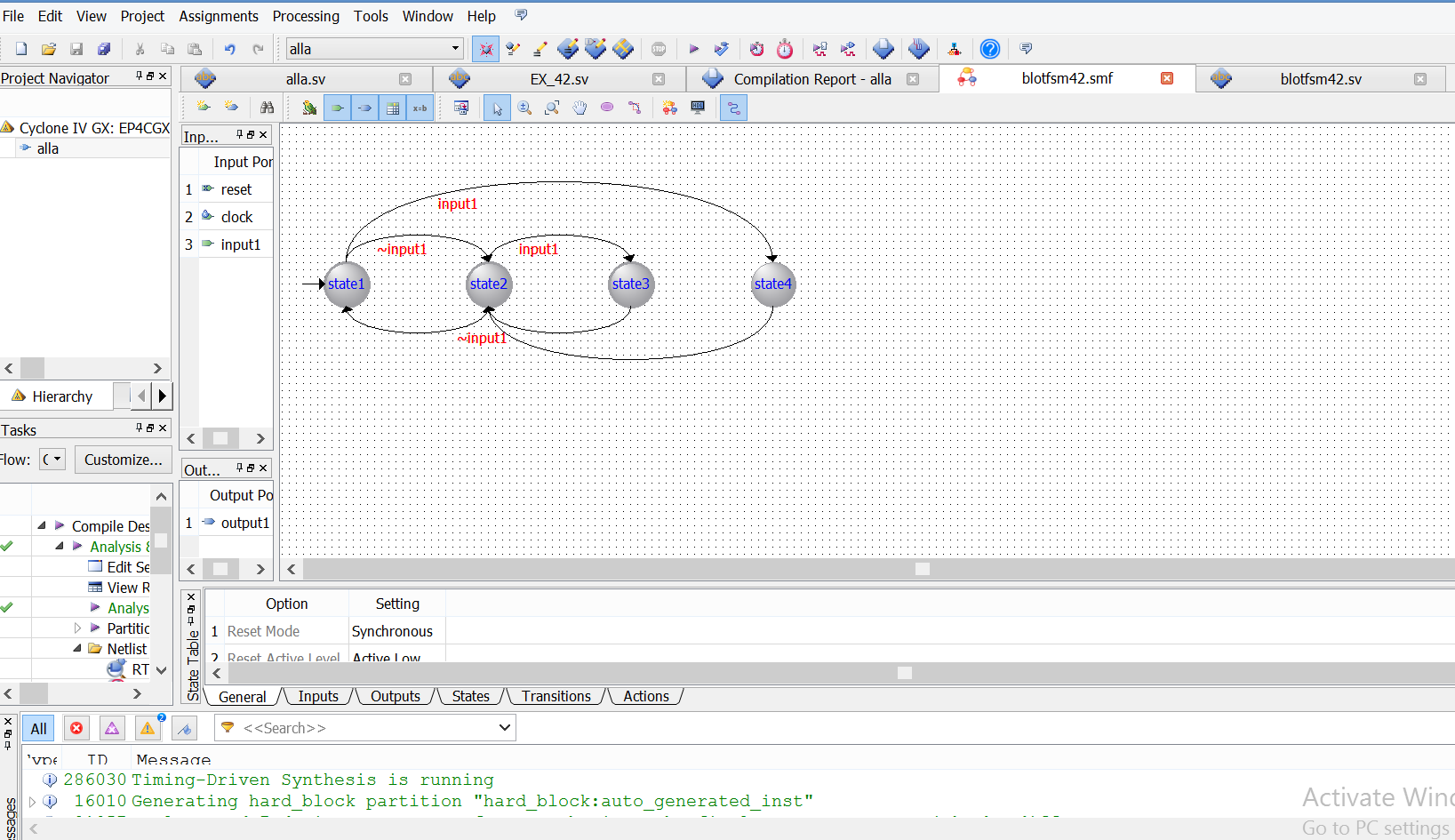
Exercise 3.31 Analyze the FSM shown in Figure 3.72. Write the state transition and output tables and sketch the state transition diagram. Describe in words what the FSM does.



Exercise 4.42 Write an HDL module for the circuit in Exercise 3.31.



// Copyright (C) 1991-2013 Altera Corporation

// Your use of Altera Corporation's design tools, logic functions

// and other software and tools, and its AMPP partner logic

// functions, and any output files from any of the foregoing

// (including device programming or simulation files), and any

// associated documentation or information are expressly subject

// to the terms and conditions of the Altera Program License

// Subscription Agreement, Altera MegaCore Function License

// Agreement, or other applicable license agreement, including,

// without limitation, that your use is for the sole purpose of

// programming logic devices manufactured by Altera and sold by

// Altera or its authorized distributors. Please refer to the

// applicable agreement for further details.

// Generated by Quartus II Version 13.0.0 Build 156 04/24/2013 SJ Web Edition

// Created on Sat Dec 05 20:04:54 2020

// synthesis message\_off 10175

`timescale 1ns/1ns

module blotfsm42 (

input reset, input clock, input input1,

output output1);

enum int unsigned { state2=0, state3=1, state4=2, state1=3 } fstate, reg\_fstate;

always\_ff @(posedge clock)

begin

if (clock) begin

fstate <= reg\_fstate;

end

end

always\_comb begin

if (~reset) begin

reg\_fstate <= state1;

// output1 <= 1'b0;

end

else begin

//output1 <= 1'b0;

case (fstate)

state2: begin

if (input1)

reg\_fstate <= state3;

else if (~(input1))

reg\_fstate <= state1;

// Inserting 'else' block to prevent latch inference

else

reg\_fstate <= state2;

// output1 <= 1'b1;

end

state3: begin

reg\_fstate <= state2;

// output1 <= 1'b1;

//output1 <= 1'b1;

end

state4: begin

reg\_fstate <= state2;

end

state1: begin

if (~(input1))

reg\_fstate <= state2;

else if (input1)

reg\_fstate <= state4;

// Inserting 'else' block to prevent latch inference

else

reg\_fstate <= state1;

// output1 <= 1'b0;

end

default: begin

//output1 <= 1'bx;

$display ("Reach undefined state");

end

endcase

end

end

endmodule // blotfsm42

----------------------------------------------------------------------------------------------------------------------

module alla(input logic reset , clk , x ,

output logic out);

enum logic [1:0] {s0 , s1 , s2 , s3}

state , n\_state;

always\_ff @(posedge clk , posedge reset )

if (reset) state <=s0;

else state <=n\_state;

always\_comb

case (state)

s0: if (x) n\_state =s3;

else n\_state =s1;

s1: if (x) n\_state =s2;

else n\_state =s0;

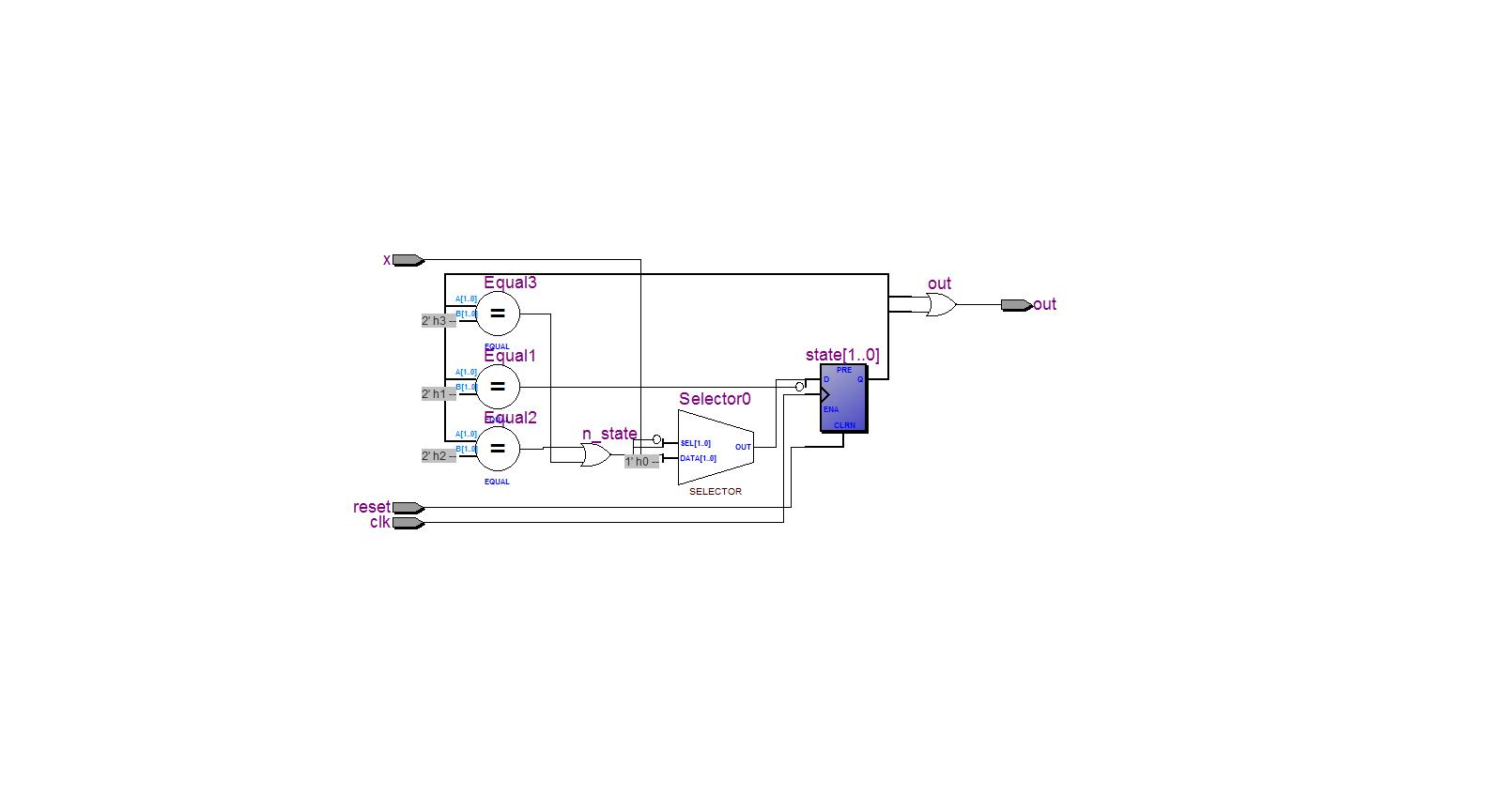
s2: n\_state =s1;

s3: n\_state =s1;

endcase

assign out = state[0] | state[1] ;

endmodule



Exercise 4.46 What does it mean for a signal to be declared tri in SystemVerilog?

Ans: tri can have multiple drivers , Two types of nets in SystemVerilog are called tri and trireg, that means the signal can have multiple drivers.

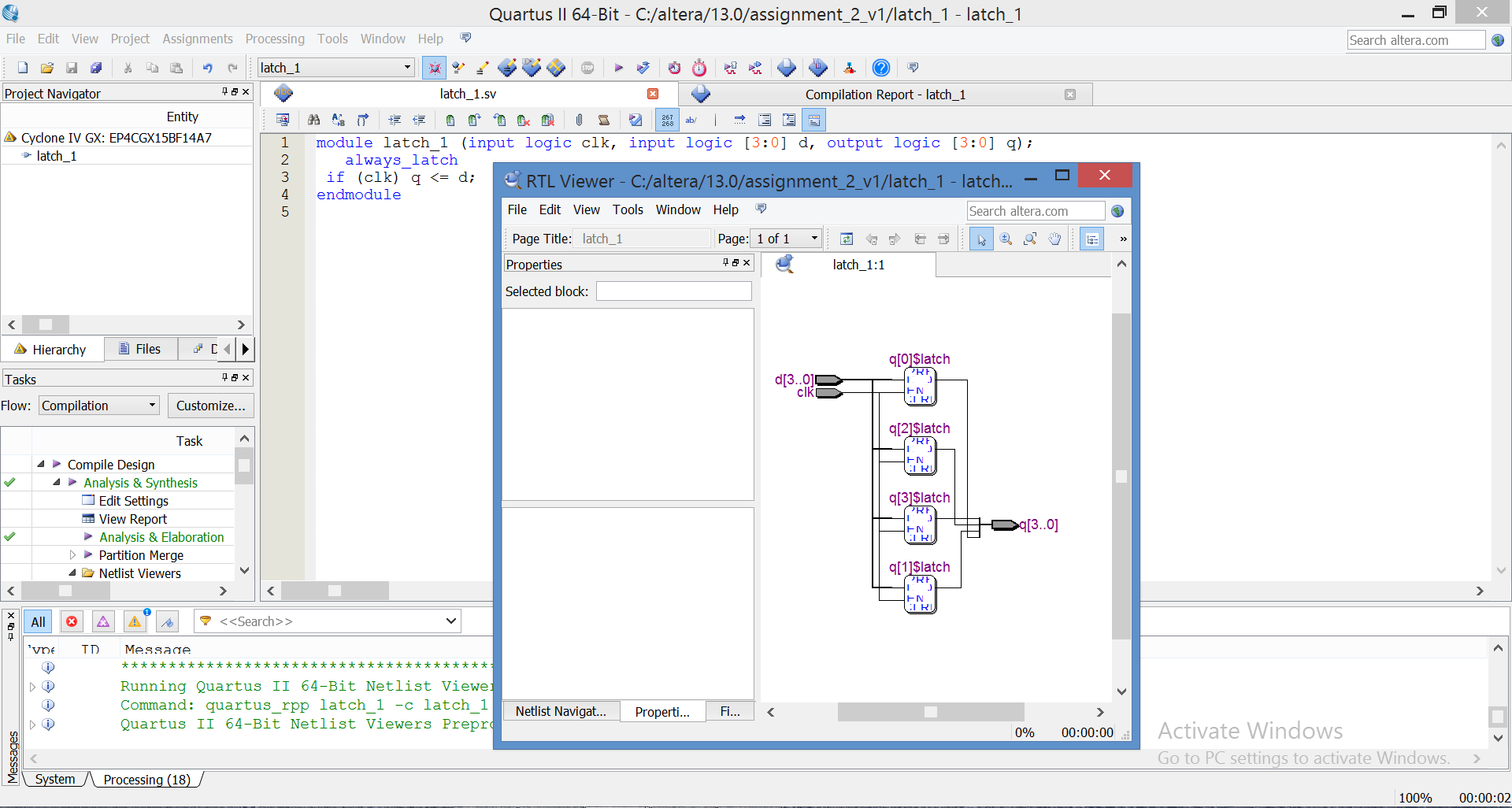
Exercise 4.50 The following SystemVerilog modules show errors that the authors have seen students make in the laboratory. Explain the error in each module and show how to fix it.

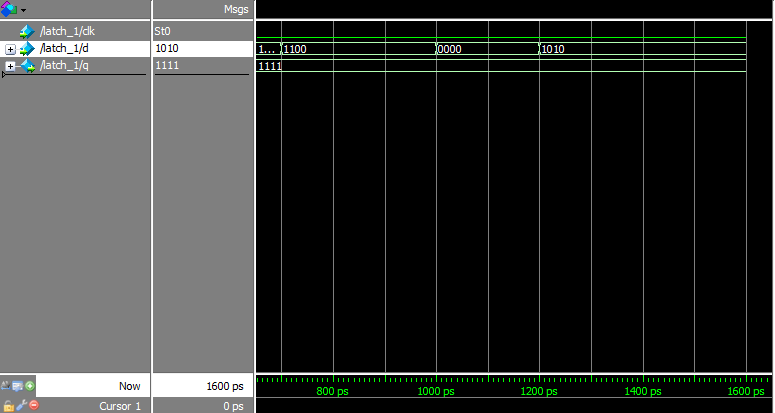
a) module latch\_1 (input logic clk, input logic [3:0] d, output logic [3:0] q);

always\_latch

if (clk) q <= d;

endmodule





---------------------------------------------------------------------------------------------------------------------------

b) module gates(input logic [3:0] a, b, output logic [3:0] y1, y2, y3, y4, y5);

always \_comb

begin

y1=a & b;

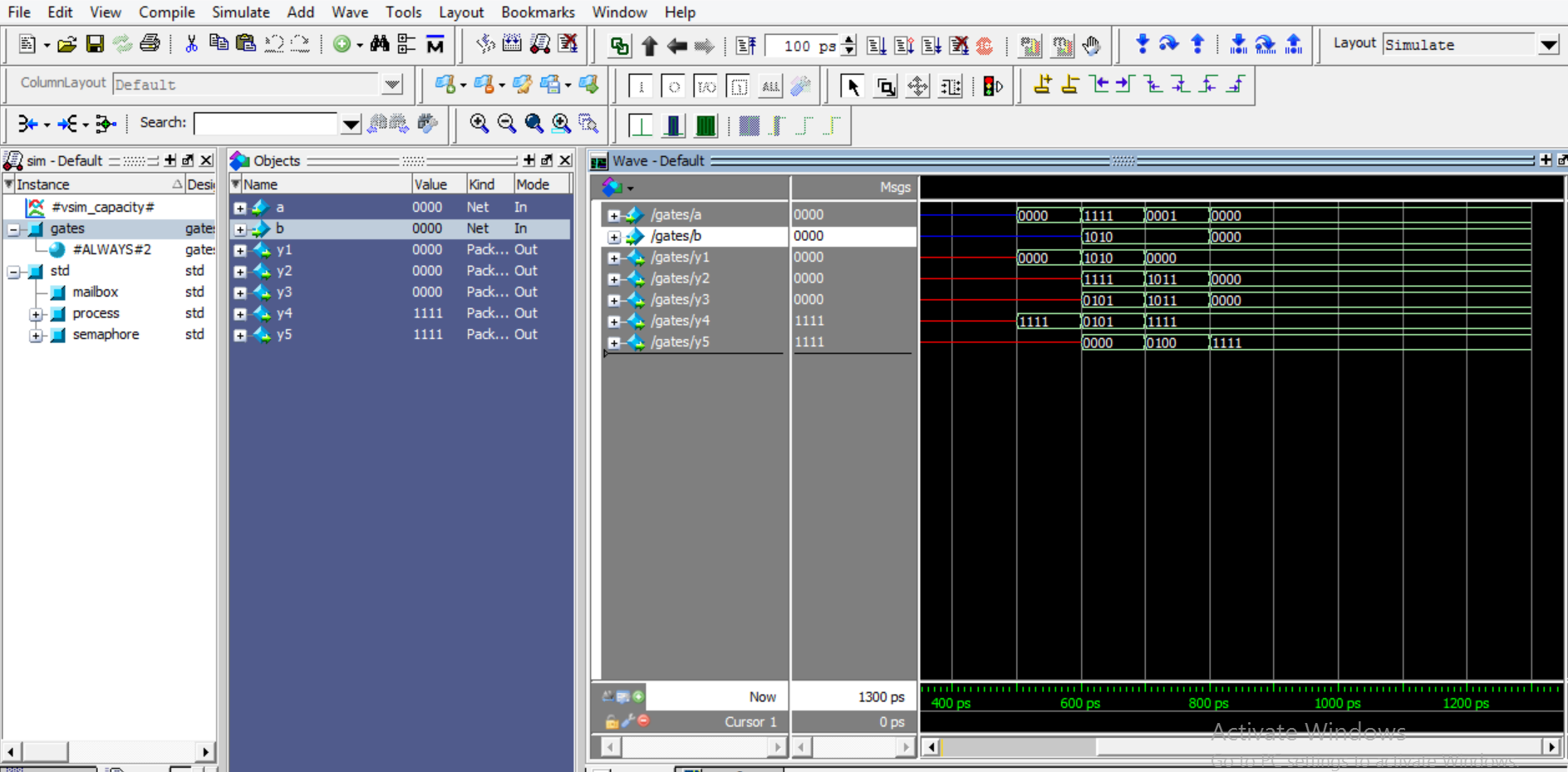
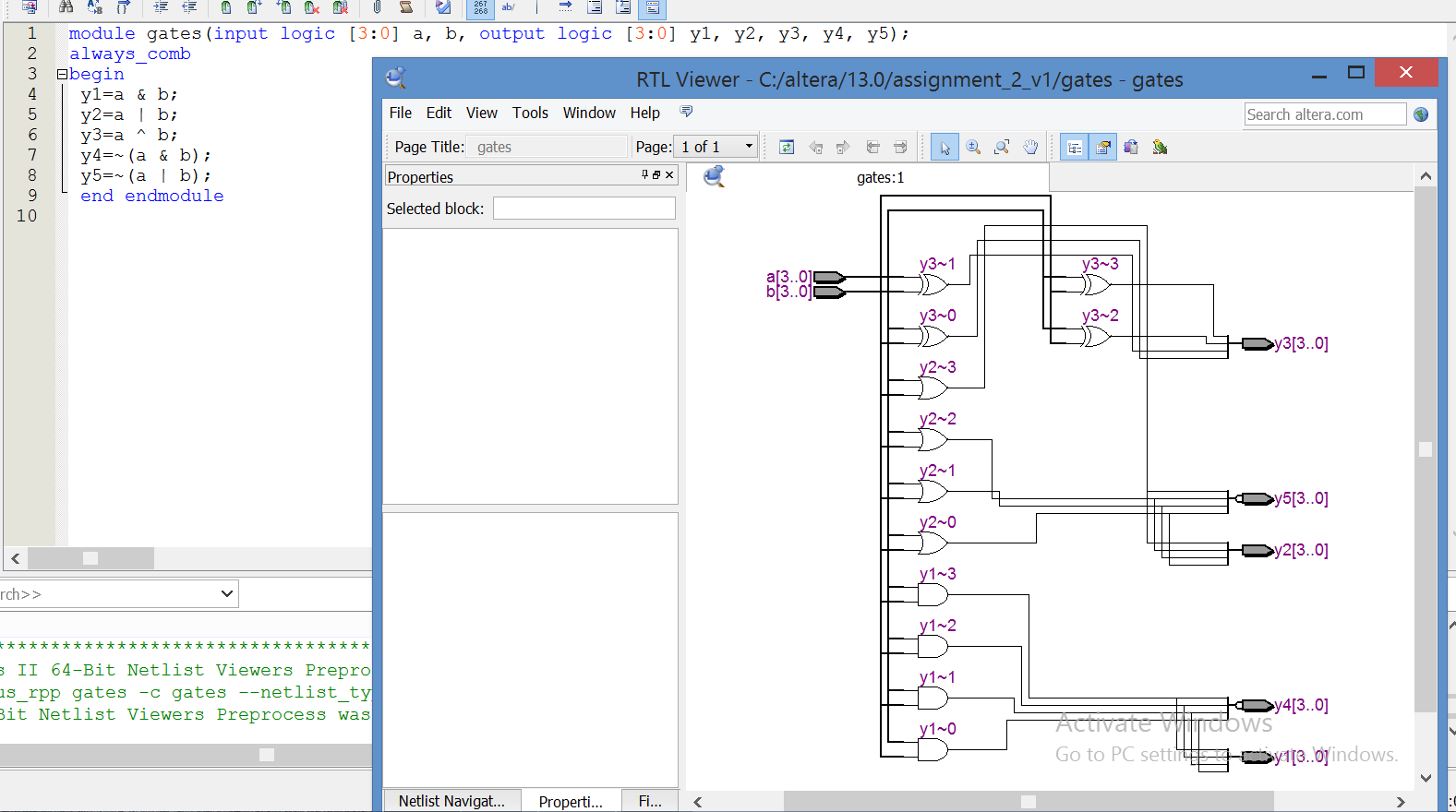
y2=a | b;

y3=a ^ b;

y4=~(a & b);

y5=~(a | b);

end endmodule



---------------------------------------------------------------------------------------------------------------------------

c) module mux\_2(input logic [3:0] d0, d1, input logic s, output logic [3:0] y);

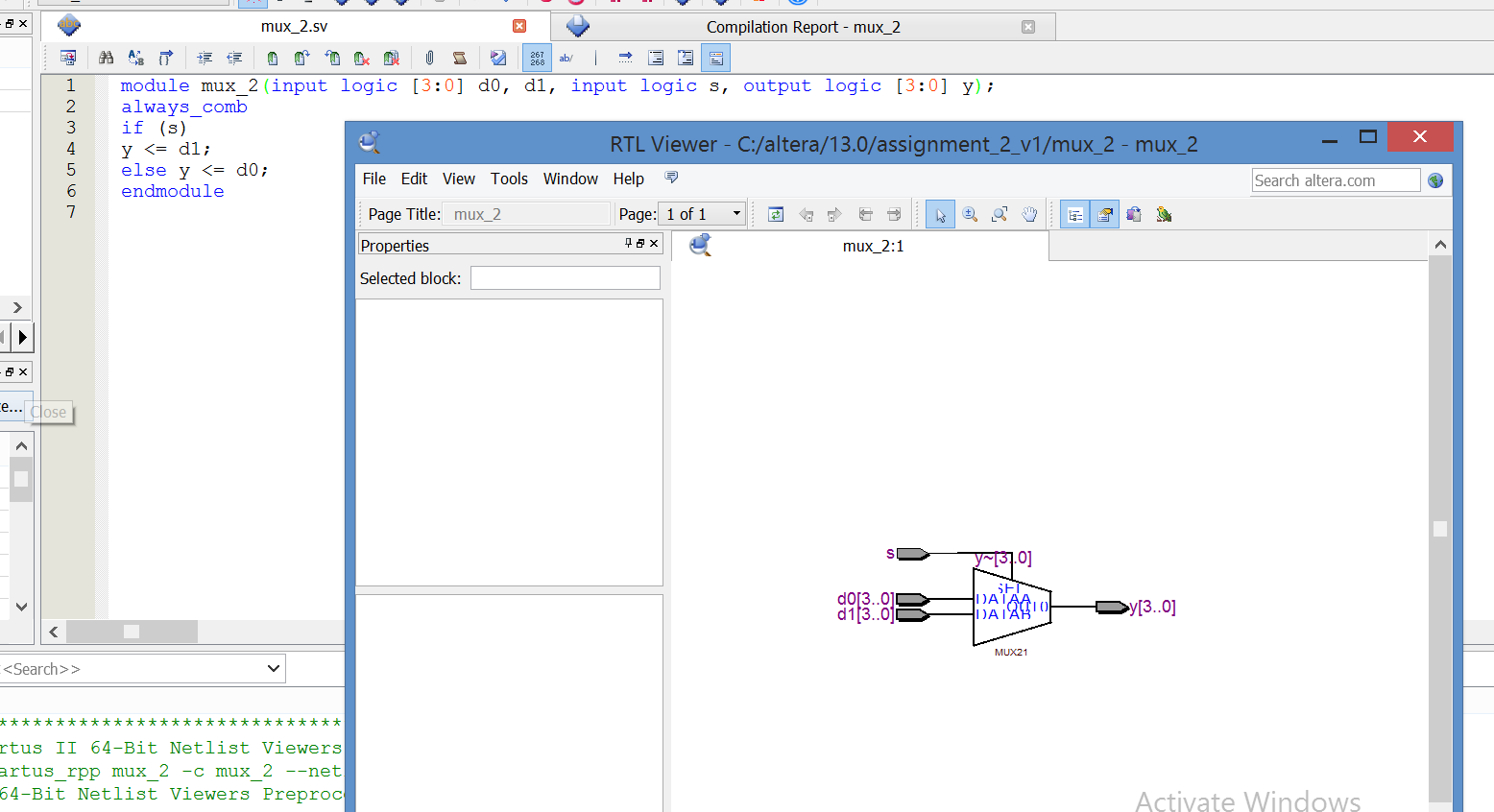
always\_comb

if (s)

y <= d1;

else y <= d0;

endmodule



---------------------------------------------------------------------------------------------------------------------------



d)dmodule twoflops(input logic clk, input logic d0, d1, output logic q0, q1);

always\_ff @(posedge clk)

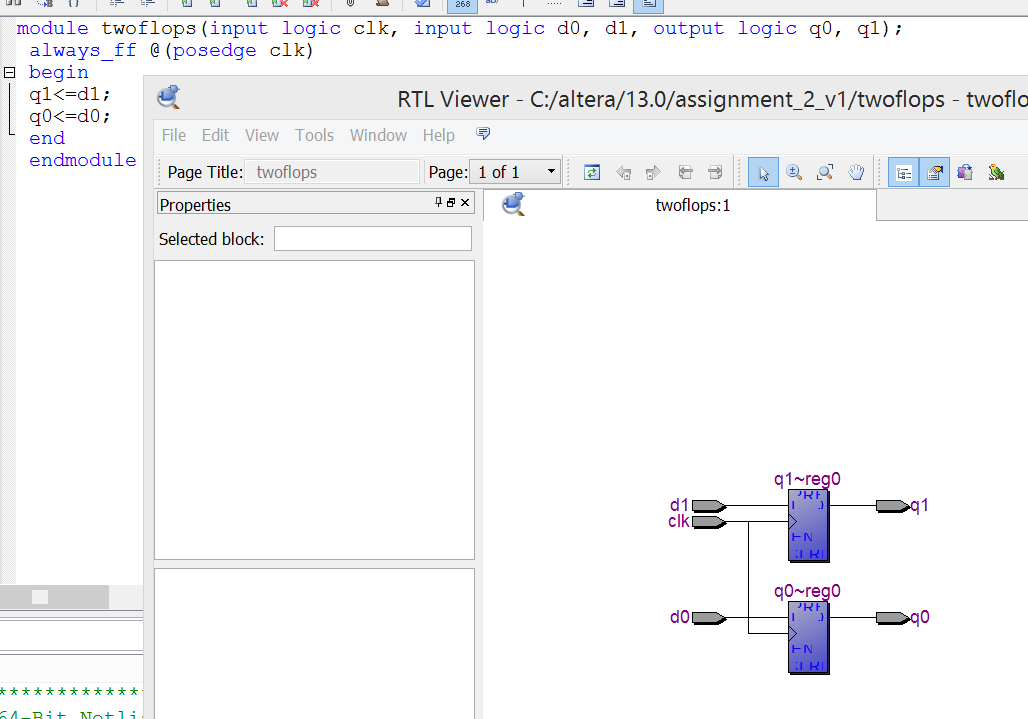
begin

q1<=d1;

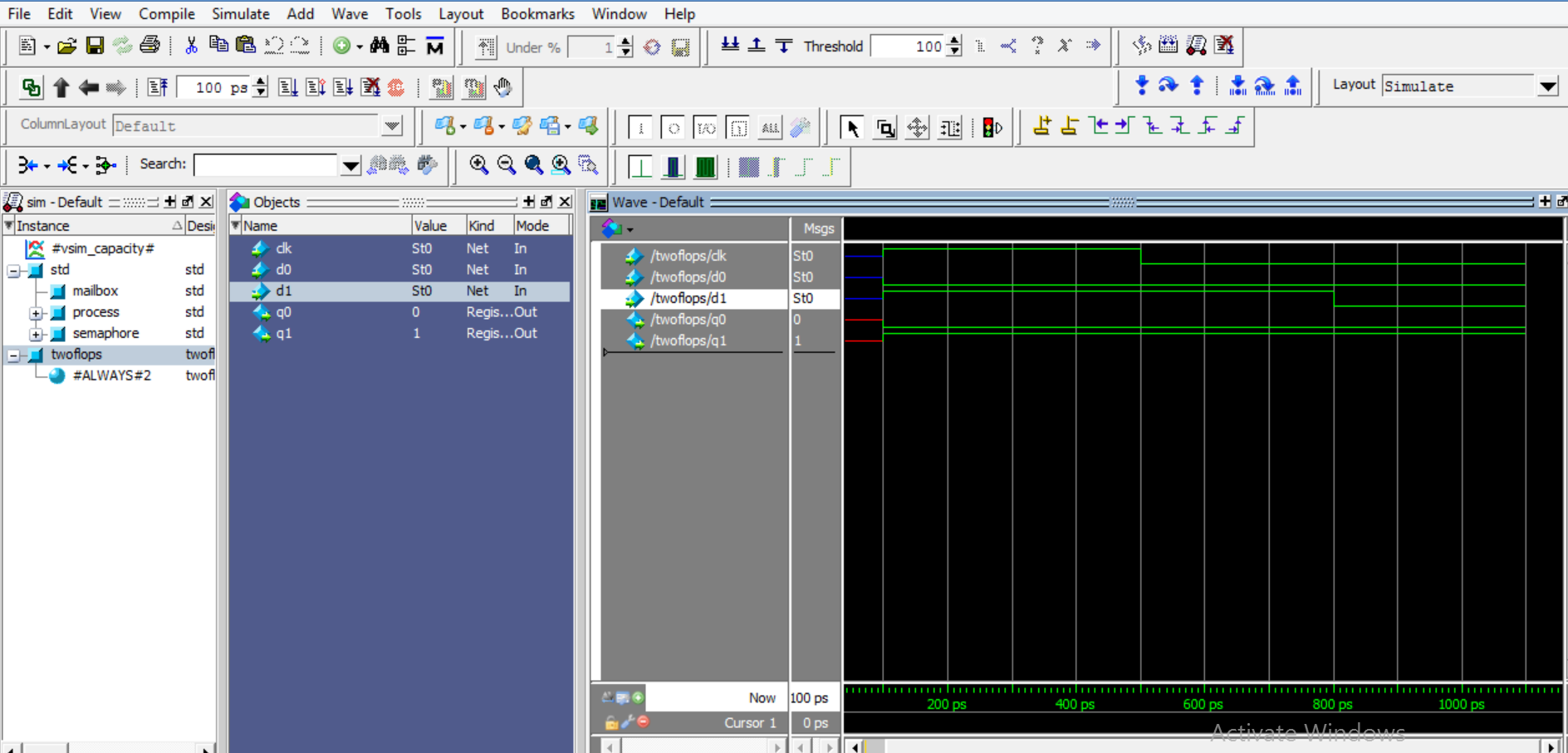
q0<=d0;

end

endmodule



---------------------------------------------------------------------------------------------------------------------------



e)module FSM\_1(input logic clk,

input logic a,

output logic out1, out2 ,

input logic reset);

logic state , n\_state;

// sequential

// register

always\_ff @(posedge clk , posedge reset )

if (reset)

state<=1'b0;

else

state <=n\_state;

// next state logic

always\_comb

case (state)

1'b0: if(a) n\_state = 1'b1;

else n\_state = 1'b0;

1'b1: if(~a) n\_state = 1'b0;

else n\_state = 1'b1;

endcase

//to output

always\_comb

begin

if(state == 0)

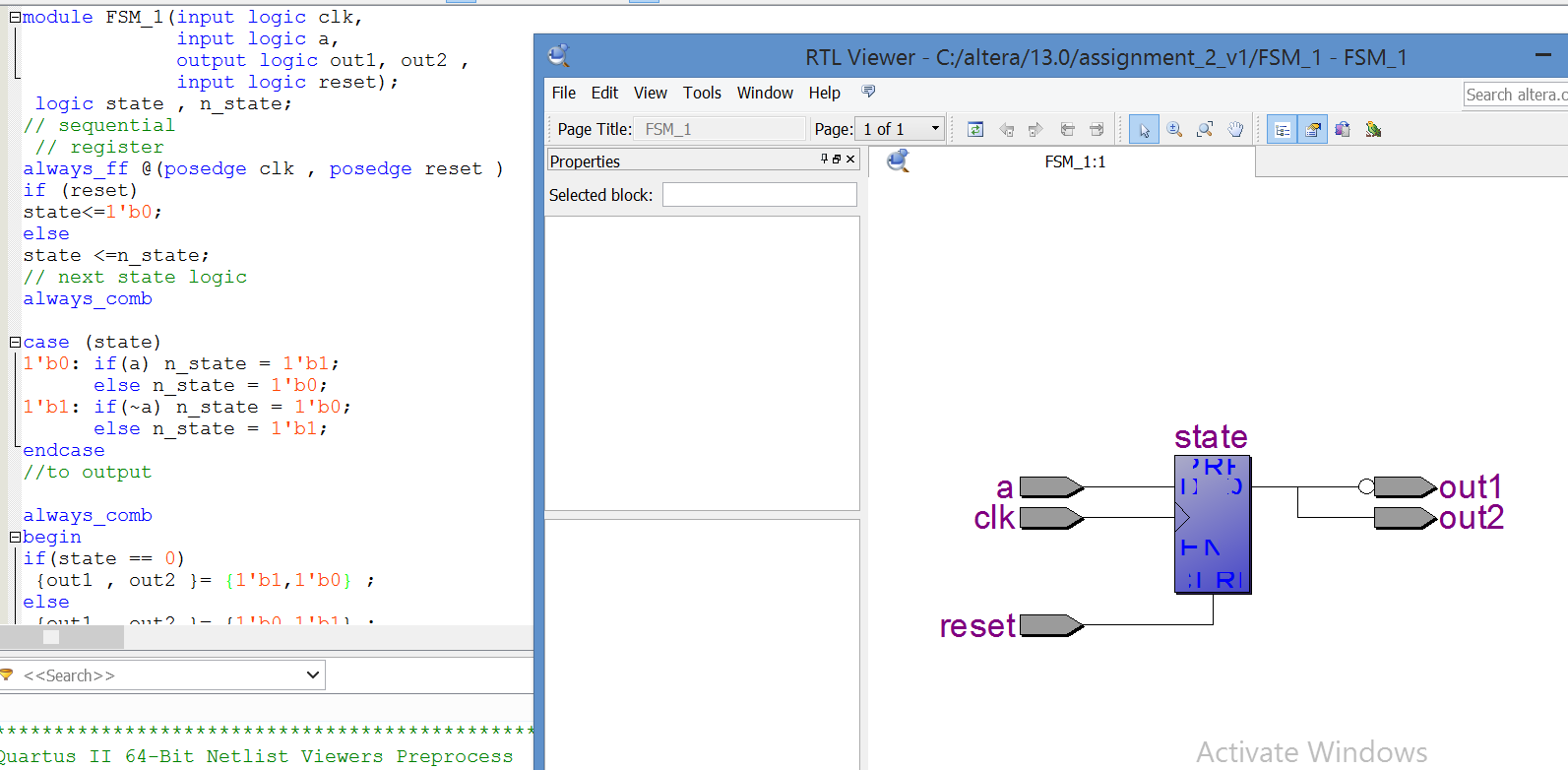
{out1 , out2 }= {1'b1,1'b0} ;

else

{out1 , out2 }= {1'b0,1'b1} ;

end

endmodule



---------------------------------------------------------------------------------------------------------------------------

