**LAB 6 – Raja Aadhithan**

Design – Sequence Detector:

Code:

module seq\_det(input seq\_in,clock,reset,output det\_o);

reg [1:0]state,next\_state;

parameter IDLE=2'b00, STATE1 = 2'b01, STATE2 = 2'b10, STATE3=2'b11;

always@(posedge clock)begin

    if(reset)begin

        state<=IDLE;

    end

    else begin

        state<=next\_state;

    end

end

always@(\*)begin

     case(state)

        IDLE   : next\_state = seq\_in ? STATE1 : IDLE ;

        STATE1 : next\_state = seq\_in ? STATE1 : STATE2 ;

        STATE2 : next\_state = seq\_in ? STATE3 : IDLE ;

        STATE3 : next\_state = seq\_in ? STATE1 : STATE2 ;

        default: next\_state=IDLE;

     endcase

   end

assign det\_o = (state==STATE3);

endmodule

Testbench:

module seq\_det\_tb();

   //Testbench global variables

   reg  din,clock,reset;

   wire dout;

   //Parameter constant for CYCLE

   parameter CYCLE = 10;

   //DUT Instantiation

   seq\_det SQD(.seq\_in(din),

          .clock(clock),

               .reset(reset),

          .det\_o(dout));

   //Step1 : Generate clock, using parameter "CYCLE"

   always

      begin

    #(CYCLE/2);

    clock = 1'b0;

    #(CYCLE/2);

    clock=~clock;

      end

   /\*Step2 : Write a task named "initialize" to initialize

   the input din of sequence detector\*/

   task initialize( );

      begin

         din = 0;

      end

   endtask

   //Delay task

   task delay(input integer i);

      begin

    #i;

      end

   endtask

   /\*Step3 : Write a task named "RESET" to reset the design,

   use delay task for adding delays\*/

   //Reset task

   task RESET();

      begin

    delay(5);

    reset=1'b1;

    delay(10);

    reset=1'b0;

      end

   endtask

   /\*Step4 : Write a task named "stimulus" which provides input to

   design on negedge of clock\*/

   task stimulus(input data);

      begin

         @(negedge clock);

         din = data;

      end

   endtask

   //Process to monitor the changes in the variables

   initial

      $monitor("Reset=%b, state=%b, Din=%b, Output Dout=%b",

      reset,SQD.state,din,dout);

   /\*Process to display a string after the sequence is detected and dout is asserted.

   SQD.state is used here as a path hierarchy where SQD is the instance name acting

   like a handle to access the internal register "state" \*/

   always@(SQD.state or dout)

      begin

    if(SQD.state==2'b11 && dout==1)

       $display("Correct output at state %b", SQD.state);

      end

   /\*Process to generate stimulus by calling the tasks and

   passing the sequence in an overlapping mode\*/

   initial

      begin

    initialize;

    RESET;

    stimulus(0);

    stimulus(1);

    stimulus(0);

    stimulus(1);

    stimulus(0);

    stimulus(1);

    stimulus(1);

    RESET;

    stimulus(1);

    stimulus(0);

    stimulus(1);

    stimulus(1);

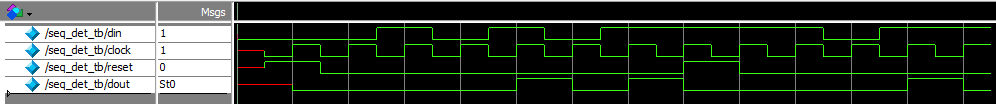
    delay(10);

    $finish;

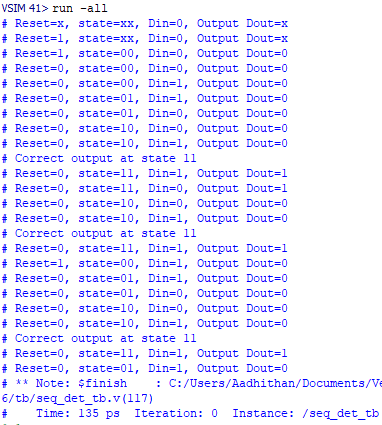
      end

endmodule

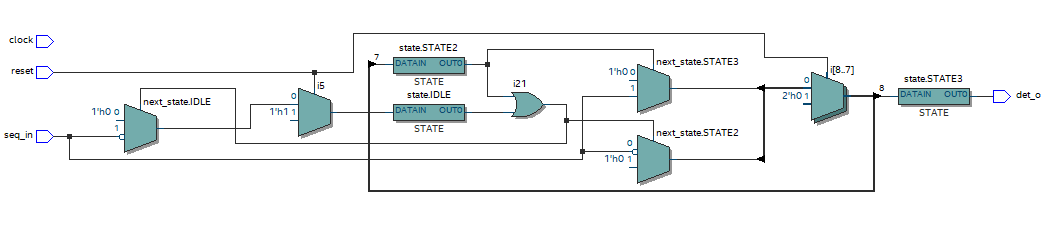
Wave:



Output:



RTL:



Design : Vending machine:

Code:

module coin(input clk,rst,i,j, output x,y);

parameter idle = 3'b000, st1 = 3'b001, st2 = 3'b010, st3 = 3'b011, st4 = 3'b100;

reg[2:0]next\_state,state;

always@(\*)begin

    case(state)

    idle : next\_state = i ? ( j ? st2 : st1) : idle ;

    st1  : next\_state = i ? ( j ? st3 : st2) : st1 ;

    st2  : next\_state = i ? ( j ? st4 : st3) : st2 ;

    st3  : next\_state = idle;

    st4  : next\_state = idle;

    default: next\_state = idle;

    endcase

end

always@(posedge clk or posedge rst)begin

    if(rst) state<=idle;

    else state<=next\_state;

end

assign x = (state==st3)||(state==st4);

assign y = (state==st4);

endmodule

Testbench:

module coin\_tb();

reg clk,reset,i,j;

wire x,y;

coin dut(clk,reset,i,j,x,y);

initial begin

    clk = 1;

    forever #5 clk = ~clk;

end

initial begin

    $monitor("@time:%3d,input (ij) is %b,%b and output (xy) is %b,%b",$time,i,j,x,y);

    reset = 1 ;

    #15;

    reset = 0;

    i = 0;

    j = 1;

    #10;

    i = 0;

    j = 0;

    #10;

    i = 1;

    j = 1;

    #10;

    i = 1;

    j = 0;

    #10;

    i = 0;

    j = 1;

    #10;

    i = 1;

    j = 1;

    #10;

    i = 1;

    j = 1;

    #10;

    i = 1;

    j = 0;

    #10;

    i = 1;

    j = 1;

    #10;

    i = 1;

    j = 0;

    #10;

    i = 0;

    j = 1;

    #10;

    i = 1;

    j = 1;

    #10;

    i = 1;

    j = 1;

    #10;

    i = 1;

    j = 0;

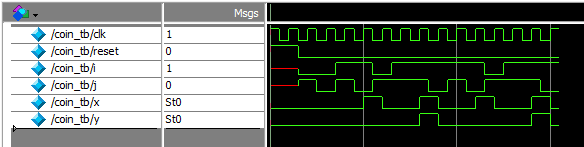
    #10;

    $finish;

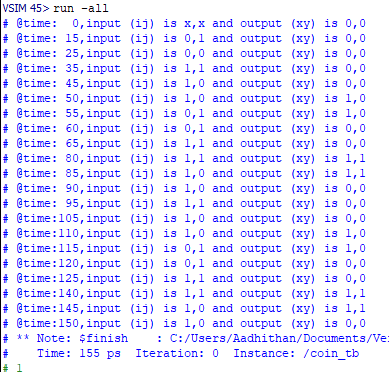
end

endmodule

Wave:



Output:



RTL:

