Bangladesh University of Engineering and Technology

Department of Electrical and Electronics Engineering

Course Number: EEE 304

Course Title: Digital Electronics Laboratory

Experiment Number: 05

Name of the Experiment(s):

Design, Simulation, and Test of Finite State Machines Using Verilog and Implementation in FPGA

Prepared by: Tasmin Khan

Student ID: 1906044

Level:3 Term:2

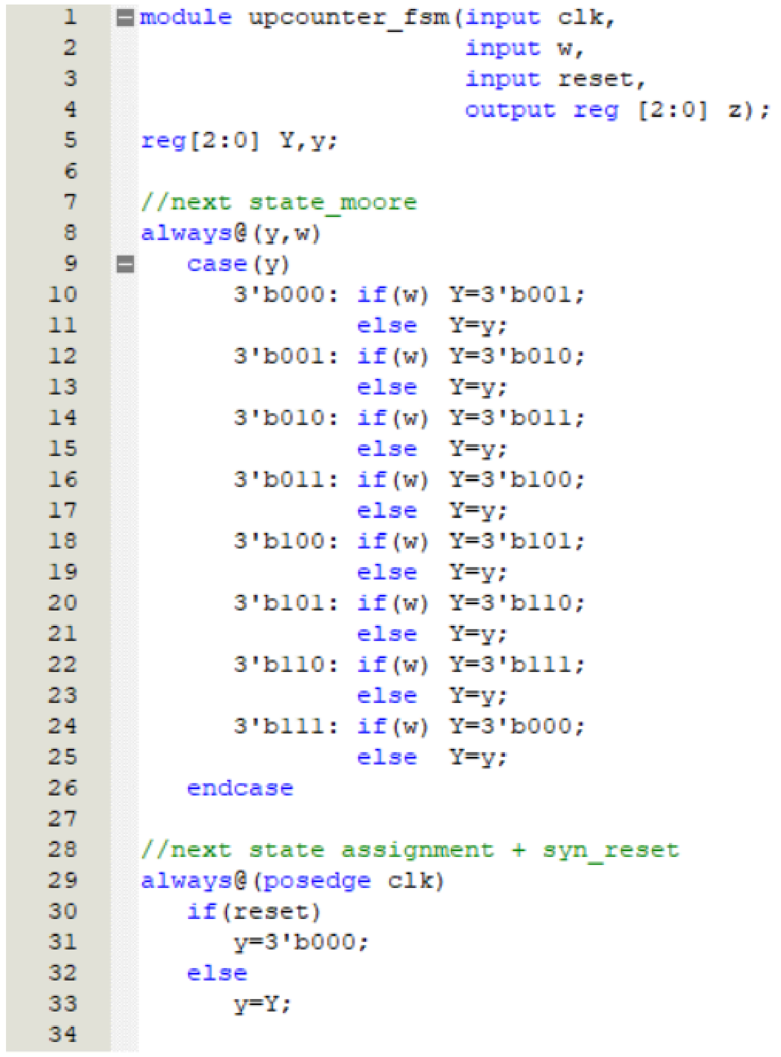
Lab Exercise 1

Question: Design a 3 bit counter using FSM according to the following specifications:

- The counting sequence is 0, 1, 2, . . . , 6, 7, 0, 1, . . .
- There exists an input signal w. The value of this signal is considered during each clock cycle. If w = 0, the present count remains the same; if w = 1, the count is incremented.

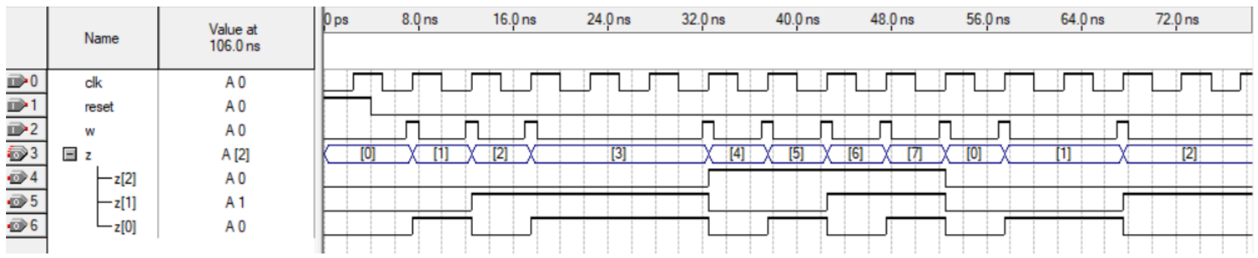
Write the Verilog program for this counter and verify your program using timing diagrams.

Verilog code:



```verilog
module upcounter_fsm(input clk,
                     input w,
                     input reset,
                     output reg [2:0] z);
reg[2:0] Y,y;

//next state_moore
always@(y,w)
   case(y)
      3'b000: if(w) Y=3'b001;
              else  Y=y;
      3'b001: if(w) Y=3'b010;
              else  Y=y;
      3'b010: if(w) Y=3'b011;
              else  Y=y;
      3'b011: if(w) Y=3'b100;
              else  Y=y;
      3'b100: if(w) Y=3'b101;
              else  Y=y;
      3'b101: if(w) Y=3'b110;
              else  Y=y;
      3'b110: if(w) Y=3'b111;
              else  Y=y;
      3'b111: if(w) Y=3'b000;
              else  Y=y;
   endcase

//next state assignment + syn_reset
always@(posedge clk)
   if(reset)
      y=3'b000;
   else
      y=Y;

```

Waveform:

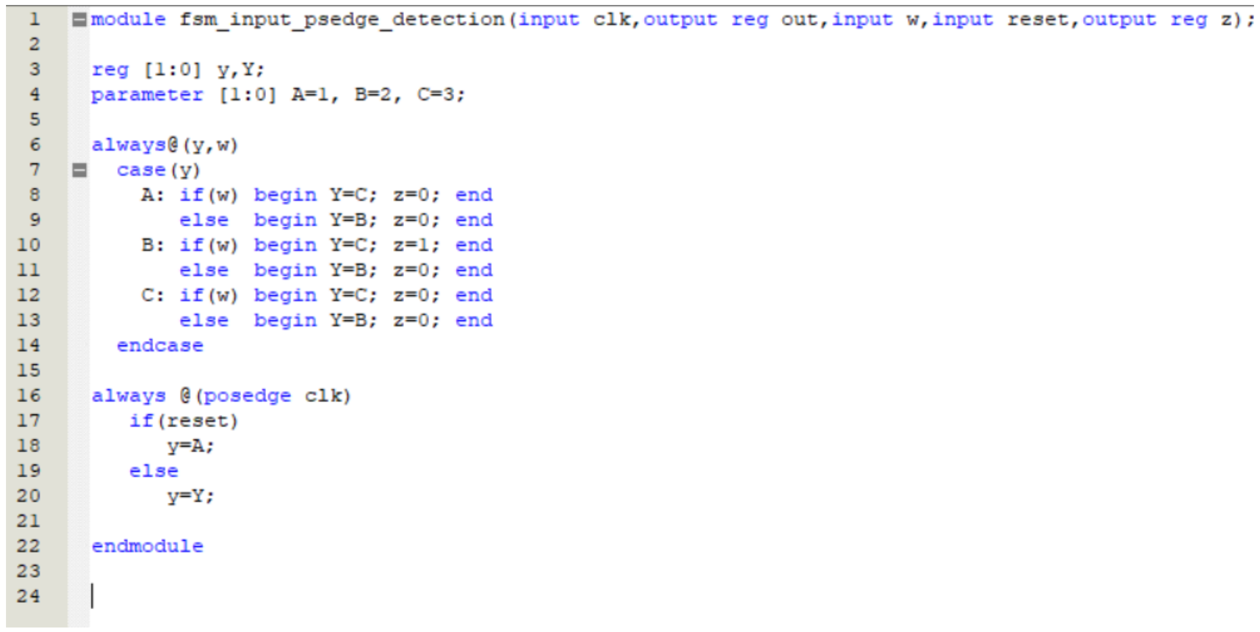


Here, we counted up to 7 with the input from w and repeated the cycle from zero once it reached the end.

Lab Exercise 2

Question: Design a FSM such that the output is asserted for one cycle when the input bit stream changes from 0 to 1. Write the Verilog program for this counter and verify the functionality by uploading your program to the CPLD board.

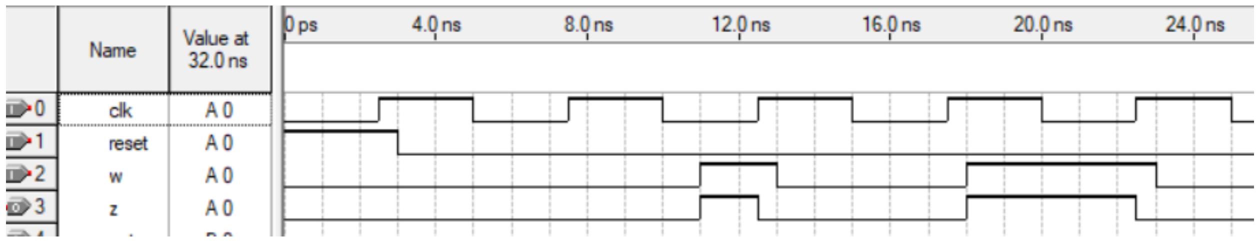
Verilog Code



```
module fsm_input_psedge_detection(input clk,output reg out,input w,input reset,output reg z);

reg [1:0] y,Y;
parameter [1:0] A=1, B=2, C=3;

always@(y,w)
  case(y)
    A: if(w) begin Y=C; z=0; end
       else  begin Y=B; z=0; end
    B: if(w) begin Y=C; z=1; end
       else  begin Y=B; z=0; end
    C: if(w) begin Y=C; z=0; end
       else  begin Y=B; z=0; end
  endcase

always @(posedge clk)
   if(reset)
      y=A;
   else
      y=Y;

endmodule

```

:

Vector Waveform

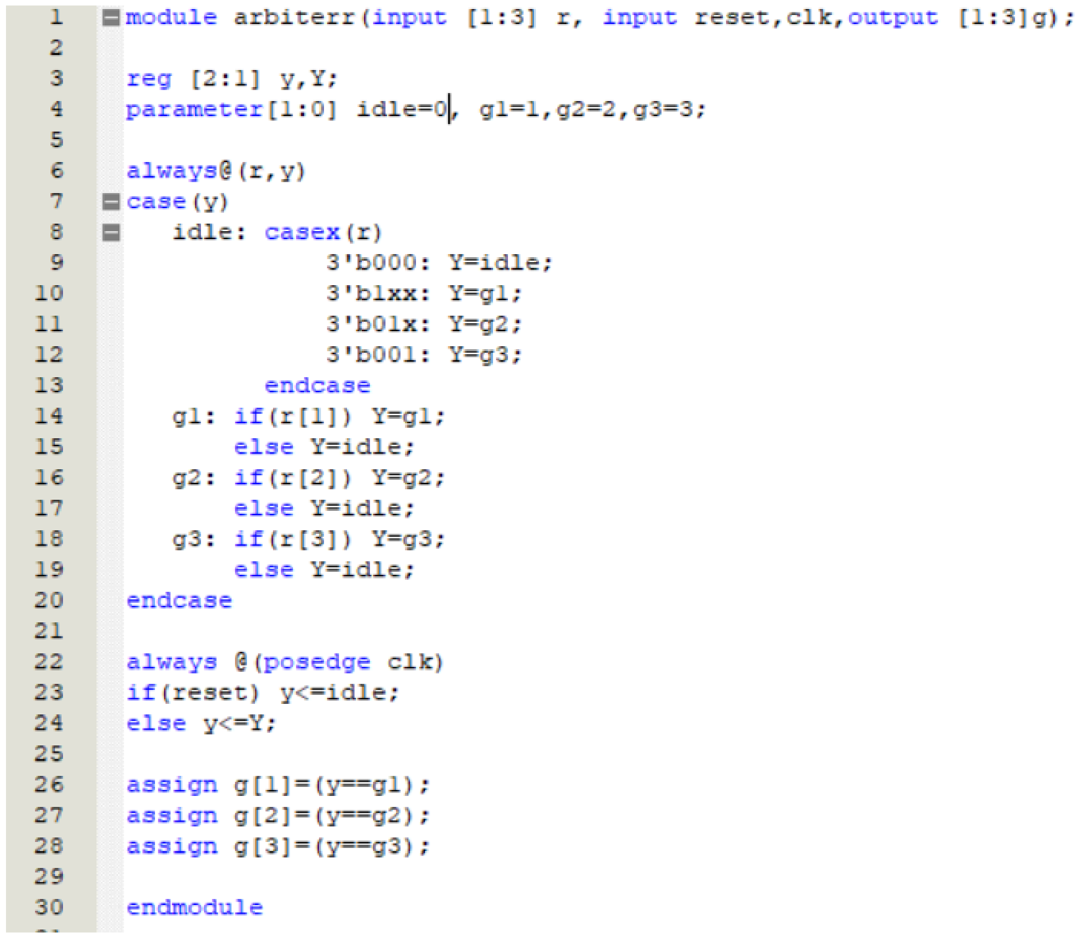


Here, as soon as input w went from 0 to 1, the output follows the same rule of a mealy machine. But if input w holds its value the same, output Z goes down to zero at the next clock edge.

Lab exercise 3

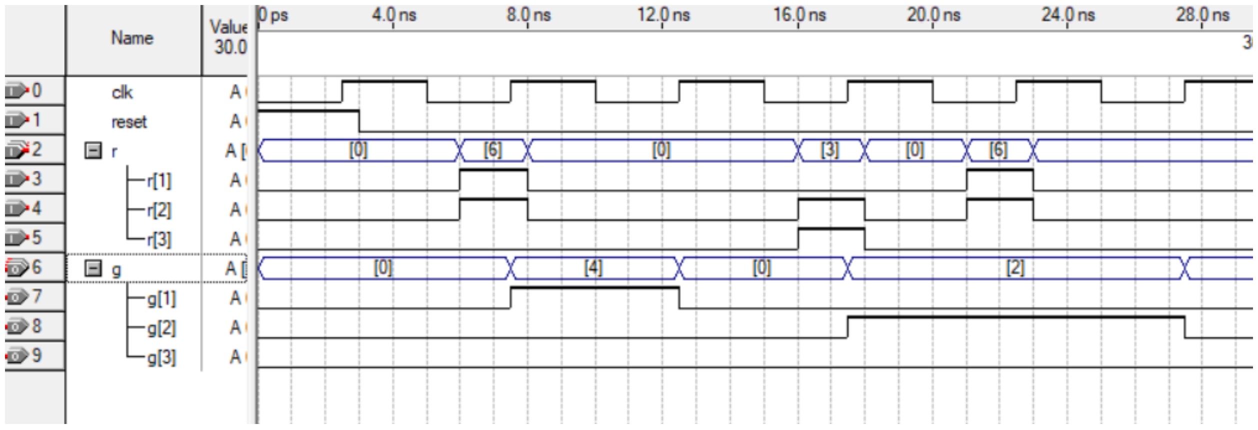
Question: Design a FSM as an arbiter circuit. The purpose of the arbiter circuit is to control access by various devices to a shared resource in a given system. Only one device can use the resource at a time. Assume that all signals in the system can change values only following the positive edge of the clock signal. Each device provides one input to the FSM, called a request, and the FSM produces a separate output for each device, called a grant. A device indicates its need to use the resource by asserting its request signal. Whenever the shared resource is not already in use, the FSM considers all requests that are active. Based on a priority scheme, it selects one of the requesting devices and asserts its grant signal. When the device is finished using the resource, it deasserts its request signal. Sketch the state diagram for the arbiter circuit. Write the Verilog code and verify the functionality by timing diagram simulation and uploading to CPLD board.

Verilog Code



```verilog
module arbiterr(input [1:3] r, input reset,clk,output [1:3]g);

reg [2:1] y,Y;
parameter[1:0] idle=0, g1=1,g2=2,g3=3;

always@(r,y)
case(y)
   idle: casex(r)
            3'b000: Y=idle;
            3'b1xx: Y=g1;
            3'b01x: Y=g2;
            3'b001: Y=g3;
         endcase
   g1: if(r[1]) Y=g1;
       else Y=idle;
   g2: if(r[2]) Y=g2;
       else Y=idle;
   g3: if(r[3]) Y=g3;
       else Y=idle;
endcase

always @(posedge clk)
if(reset) y<=idle;
else y<=Y;

assign g[1]=(y==g1);
assign g[2]=(y==g2);
assign g[3]=(y==g3);

endmodule
```

Vector Waveform



Here, the grant signal outputs follow the priority scheme as described. When 1 and 2 both request access, only g1 is high. Again, when 1 requests as 2 is using it, g2 remains high.

Report Task 1

Question: Your FSM should have the following module declaration:

```
module eee304_5_xx(input logic clk,
                   input logic reset,
                   input logic left, right,
                   output logic la, lb, lc, ra, rb, rc);
```

where xx are your initials.
You may assume that clk runs at the desired speed (e.g. about 1 Hz).

On reset, the FSM should enter a state with all lights off. When you press left, you should see LA, then LA and LB, then LA, LB, and LC, then finally all lights off again. This pattern should occur even if you release left during the sequence. If left is still down when you return to the lights off state, the pattern should repeat. right is similar. It is up to you to decide what to do if the user makes left and right simultaneously true; make a choice to keep your design easy.

Sketch a state transition diagram. Define your state encodings. Hint: with a careful choice of encoding, your output and next state logic can be quite simple. Choose a set of state encodings. At this point, you could write state transition and output tables and then a set of next state and output equations. Instead, we will design it in an HDL and let the synthesis tool choose the gate-level implementation. Create a new project named lab5_xx, where xx are your initials. Choose appropriate CPLD model as before. Create your Verilog entry. Compile the project. Simulate your Verilog design by using appropriate vector waveform files. Attach the code and the results.

Problem 1: FSM Design

Design a finite state machine in Verilog to control the taillights of a 1965 Ford Thunderbird[1]. There are three lights on each side that operate in sequence to indicate the direction of a turn. Figure 1 shows the tail lights and Figure 2 shows the flashing sequence for (a) left turns and (b) right turns.

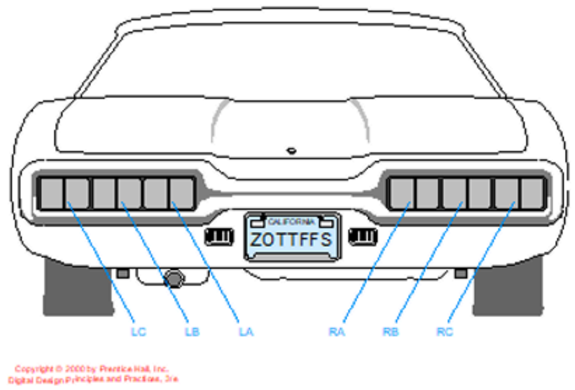

ZOTTFFS
LC
LB
LA
RA
RB
RC
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

Figure 1.1 **Thunderbird Tail Lights**

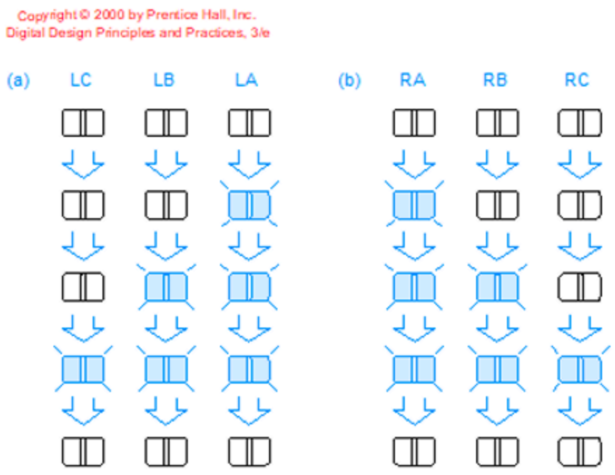
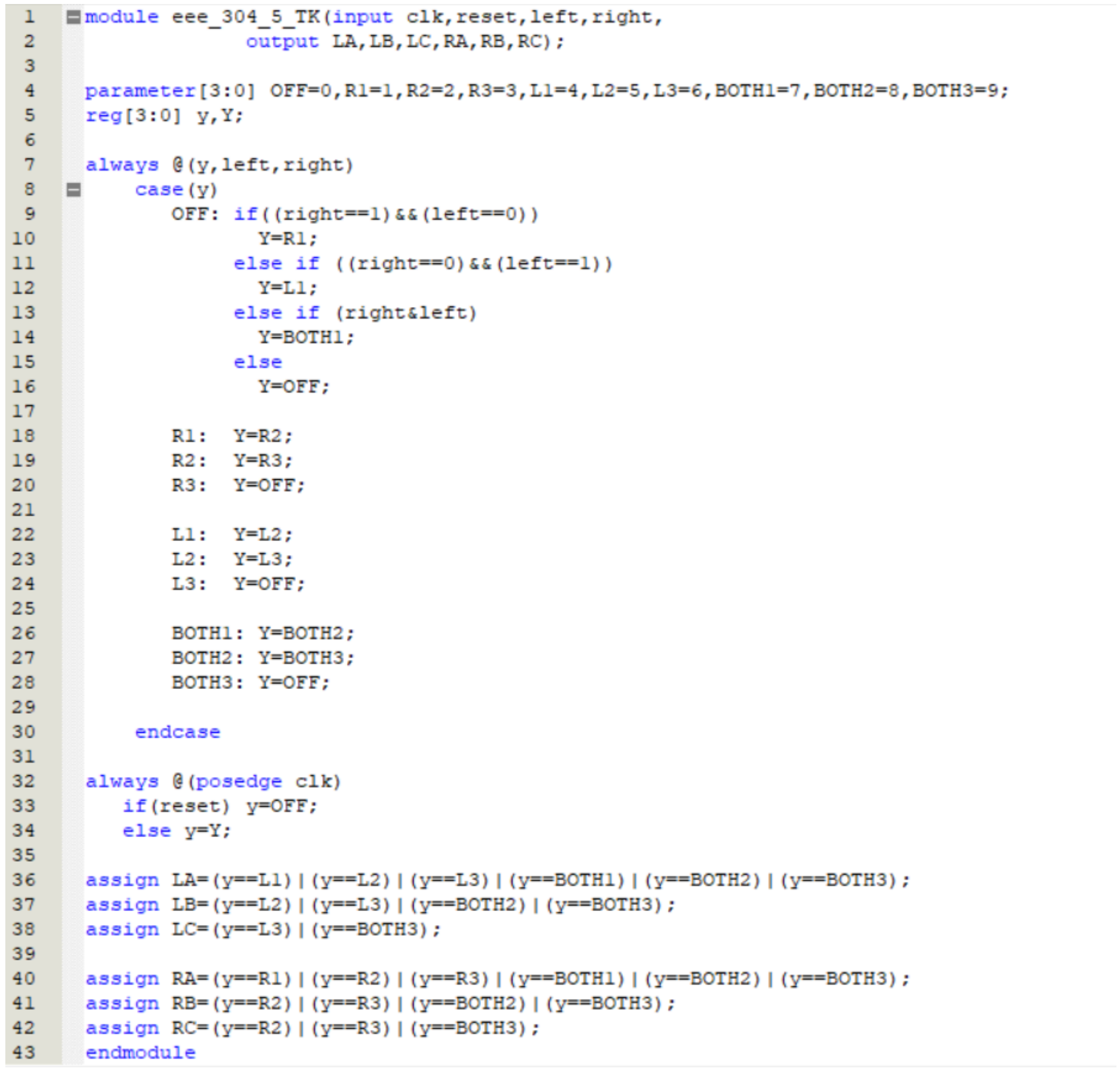

Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e
(a)
LC
LB
LA
(b)
RA
RB
RC

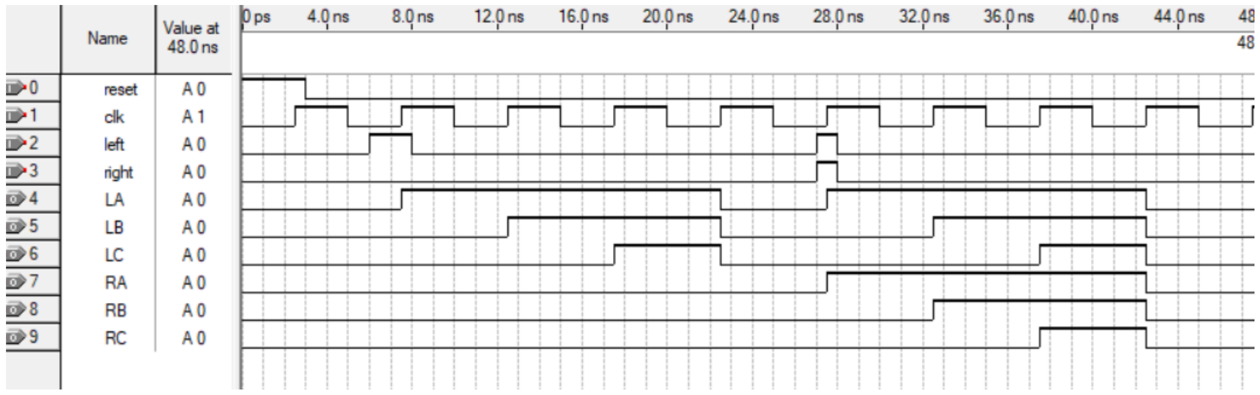
Fig: Shaded lights are illuminated

Verilog code



```verilog
module eee_304_5_TK(input clk,reset,left,right,
                    output LA,LB,LC,RA,RB,RC);

parameter[3:0] OFF=0,R1=1,R2=2,R3=3,L1=4,L2=5,L3=6,BOTH1=7,BOTH2=8,BOTH3=9;
reg[3:0] y,Y;

always @(y,left,right)
    case(y)
       OFF: if((right==1)&&(left==0))
              Y=R1;
            else if ((right==0)&&(left==1))
              Y=L1;
            else if (right&left)
              Y=BOTH1;
            else
              Y=OFF;

       R1:  Y=R2;
       R2:  Y=R3;
       R3:  Y=OFF;

       L1:  Y=L2;
       L2:  Y=L3;
       L3:  Y=OFF;

       BOTH1: Y=BOTH2;
       BOTH2: Y=BOTH3;
       BOTH3: Y=OFF;

    endcase

always @(posedge clk)
   if(reset) y=OFF;
   else y=Y;

assign LA=(y==L1)|(y==L2)|(y==L3)|(y==BOTH1)|(y==BOTH2)|(y==BOTH3);
assign LB=(y==L2)|(y==L3)|(y==BOTH2)|(y==BOTH3);
assign LC=(y==L3)|(y==BOTH3);

assign RA=(y==R1)|(y==R2)|(y==R3)|(y==BOTH1)|(y==BOTH2)|(y==BOTH3);
assign RB=(y==R2)|(y==R3)|(y==BOTH2)|(y==BOTH3);
assign RC=(y==R2)|(y==R3)|(y==BOTH3);
endmodule
```

Vector Waveform

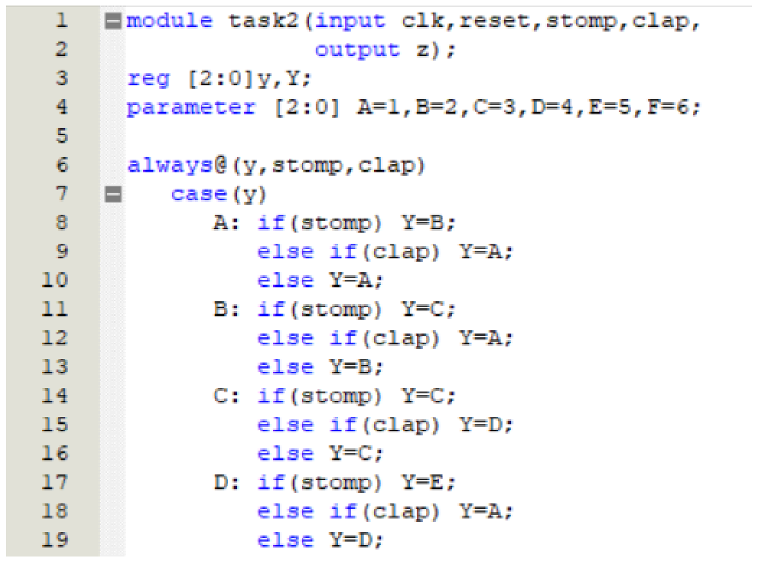


Here, we turned on one more light for left and/or right with each clock pulse.
When both lights are pressed at the same time, we enabled both the lights at the same time in a similar pattern.

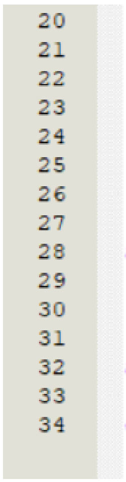
Repot Task 2

Question: Design a finite state machine in Verilog to detect the sequence “Stomp Stomp Clap” analogous to “We will rock you”. Your circuit will have 2 inputs “Stomp” and “Clap” through two pulse switches PS3 and PS4. Each time, a “Stomp Stomp Clap” sequence is input through the toggle switches, LED L1 will light up. For all other sequences of “Stomp” and “Clap” input, L1 should be off. For repeated “Stomp Stomp Clap” sequence, L1 will remain on. Attach the code and the results from timing diagram simulation.

Verilog code

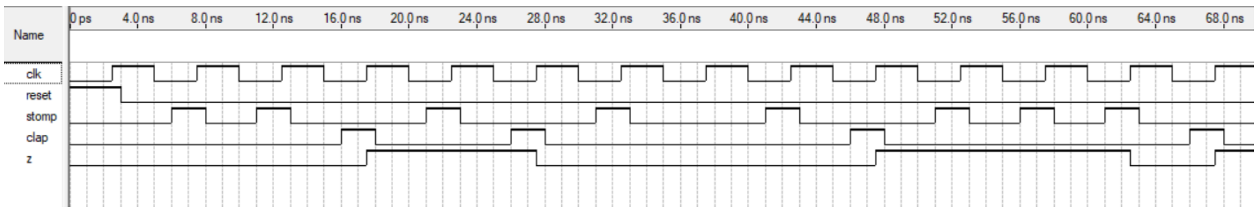


```
module task2(input clk,reset,stomp,clap,
             output z);
reg [2:0]y,Y;
parameter [2:0] A=1,B=2,C=3,D=4,E=5,F=6;

always@(y,stomp,clap)
   case(y)
      A: if(stomp) Y=B;
         else if(clap) Y=A;
         else Y=A;
      B: if(stomp) Y=C;
         else if(clap) Y=A;
         else Y=B;
      C: if(stomp) Y=C;
         else if(clap) Y=D;
         else Y=C;
      D: if(stomp) Y=E;
         else if(clap) Y=A;
         else Y=D;
```



```
      E: if(stomp) Y=F;
         else if(clap) Y=A;
         else Y=E;
      F: if(stomp) Y=C;
         else if(clap) Y=D;
         else Y=F;
    endcase

always@(posedge clk)
   if(reset) y=A;
   else y=Y;

assign z=((y==D)|(y==E)|(y==F));

endmodule
```

Vector Waveform



Here, output goes high for stomp stomp clap. The output remains high for as long as the pattern repeats. Considering this is a mechanical switch, we accounted for delays during two consecutive presses too and maintained the same state for no input.