Jyresa Mae M. Amboang
Frances Nicole S. Gigataras
Crishelle A. Agopitac
Erica Dianne Q. Canillo
Laboratory Submitted: 05/16/23
Laboratory Hours: 288 hours

Laboratory Activity No. 4

Thunderbird Turn Signal

110 Points

Laboratory Activity Overview

In this laboratory activity, you will design a finite state machine in SystemVerilog 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.

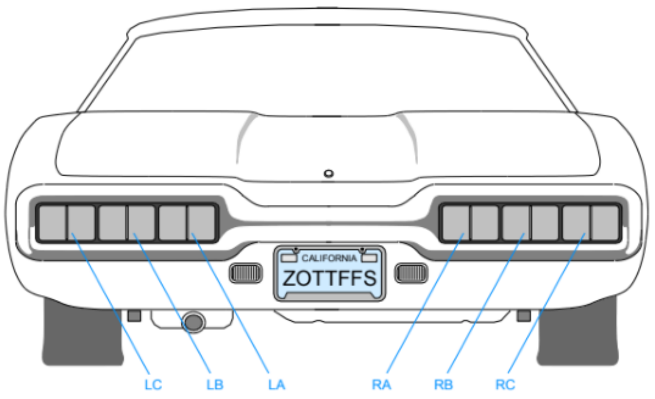


Figure 1. **Thunderbird Tail Lights**

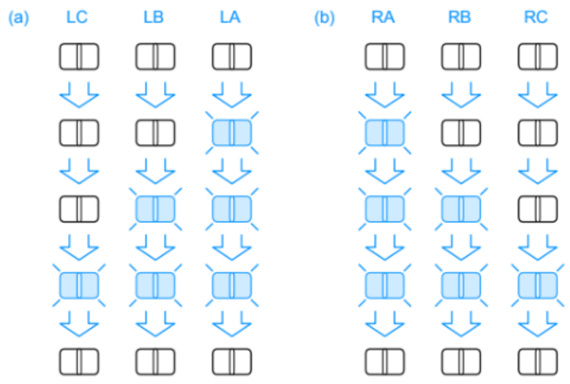


Figure 2. **Flashing Sequence (shaded lights are illuminated)**

This laboratory activity is divided into four parts: (A) Design, (B) SystemVerilog entry, (C) Simulation, and (D) Implementation. If you follow the steps of FSM design carefully and ask questions at the beginning if a part is confusing, you will save yourself a a great deal of time. As always, don't forget to refer to the "What to Submit" section at the end of this laboratory activity before you begin.

Design

The state machine has:

- 5 inputs: clock, left, and right turns, and hazard; if no signal the design waits in idle
- 6 outputs: LA, LB, LC, RA, RB, and RC
- free running clock with frequency equal to the flashing rate

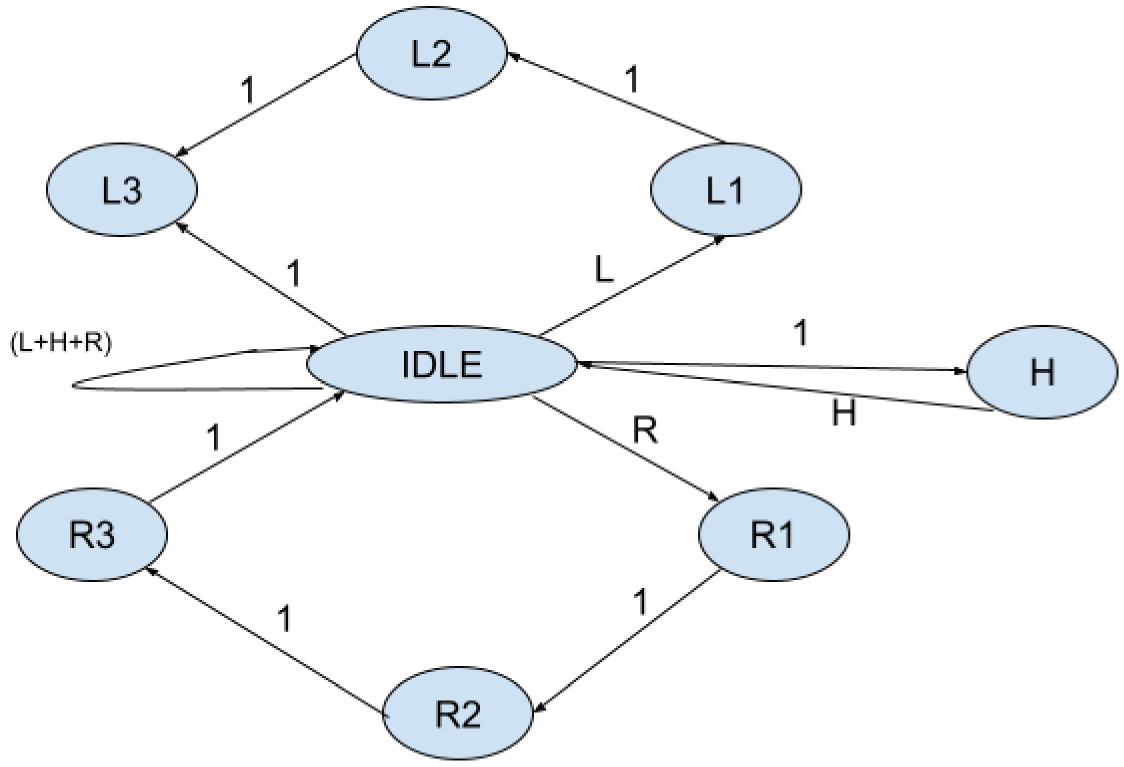
For a specified design, there could be four possible states namely: IDLE(No lights are on), Turning LEFT(left lights LA, LB, LC are blinking), RIGHT(right lights RA,RB,RC are blinking), HAZARD(all lights are blinking). The other input is the clock which determines the transition time from one state to another state. Given here it is assumed that the free running clock is equally to the desired flashing rate for the lights. So the transition occurs at every clock edge and it will be in the same state for one clock period.

Let us consider turning LEFT has 3 states namely L1(one light is on), L2(two lights are on), L3(three lights are on). Similarly for turning RIGHT has 3 states R1,R2,R3. The output table for state machine design be like:

STATE	LC	LB	LA	RA	RB	RC	Light Indicator
IDLE	0	0	0	0	0	0	NO lights on
L1	0	0	1	0	0	0	1 left light
L2	0	1	1	0	0	0	2 left lights
L3	1	1	1	0	0	0	3 left lights
R1	0	0	0	1	0	0	1 right light
R2	0	0	0	1	1	0	2 right lights
R3	0	0	0	1	1	1	3 right lights
HAZARD	1	1	1	1	1	1	ALL lights on

The state diagram could be as follows:

- L-left signal; R-right signal; H-hazard signal
- $(L + R + H)^1$- no signal is given
- 1- indicates after one clock pulse



Moore State Diagram

As the output is clearly depending upon state (NOT on transition) the state diagram is a ***moore state diagram.***

SystemVerilog Code

Lab4 testbench code:

```
module testbench();
        logic clk, reset, left, right;
        logic s0, l1, l2, l3, r1, r2, r3;
        logic s0_next, l1_next, l2_next, l3_next, r1_next, r2_next, r3_next;
        logic la, lb, lc, ra, rb, rc;

        Lab4 dut(.left, .right, .reset, .clk, .la, .lb, .lc, .ra, .rb, .rc);

        initial
                forever
                        begin
                                clk = 0; #50; clk = 1; #50;
                        end

        initial
                begin

                        @(negedge clk);  // advance time to first falling edge of clk

                        // cycle 1: reset to initial state
                        reset = 1;
                        left = 0;
                        right = 0;

                        @(negedge clk);
                        //shoul be at start state: s0

                        //+++++++++++++++++++++++++++++
                        //test self transitions for s0
                        //+++++++++++++++++++++++++++++

                        //cycle 2: don't press anything.
                        reset = 0;

                        @(negedge clk);
                        //Should stay at s0

                        //cycle 3: press both left and right.

                        left = 1;
                        right = 1;

                        @(negedge clk);
                        // Should stay at s0


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
                        // test right turn signal when user pressed right and then released the button
                        // right turn signal has to go to completion (cycle 4 - 7)

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++

                        //cycle 4: press right
                        left = 0;

                        @(negedge clk);
                        //Should be at r1

                        //cycle 5: release right button
                        right = 0;

                        @(negedge clk);
                        //should be at r2

                        //cycle 6: right button is still released
```

```
                        @(negedge clk);
                        //should be at r3

                        //cycle 7: everything is off

                        @(negedge clk);
                        //should be at s0


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
                        // test left turn signal when user pressed left and then released the button
                        // left turn signal has to go to completion (8-11)

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++

                        //cycle 8: press left
                        left = 1;

                        @(negedge clk);
                        //Should be at l1

                        //cycle 9: release right button
                        left = 0;

                        @(negedge clk);
                        //should be at l2

                        //cycle 10: left button is still released

                        @(negedge clk);
                        //should be at l3

                        //cycle 11: everything is off

                        @(negedge clk);
                        //should be at s0


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
                        // test right turn signal when user pressed right, then released the right button
                        // and presses the left button, then presses both of them
                        // right turn signal has to go to completion ignoring the button pressing (cycle 12
- 15)

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++

                        //cycle 12: press right
                        right = 1;
```

```
                @(negedge clk);
                //Should be at r1

                //cycle 13: release right button and press left button
                right = 0;
                left = 1;

                @(negedge clk);
                //should be at r2

                //cycle 14: right button pressed, left button pressed
                right = 1;

                @(negedge clk);
                //should be at r3

                //cycle 15: everything is off
                left = 0;
                right = 0;

                @(negedge clk);
                //should be at s0


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++
                // test left turn signal when user pressed left and then released the left
                // button and presses the right button, then presses both
                // left turn signal has to go to completion despite buttons being pressed (16-19)

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++

                //cycle 16: press left
                left = 1;

                @(negedge clk);
                //Should be at l1

                //cycle 17: release left button and press right
                left = 0;
                right = 1;

                @(negedge clk);
                //should be at l2

                //cycle 18: left button is pressed and right pressed
                left = 1;

                @(negedge clk);
                //should be at l3

                //cycle 19: everything is off
```

```
right = 0;
left = 0;

@(negedge clk);
//should be at s0

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++
// press right button and keep pressing
// so that the turn signal will go though 2 repetitions
// cycle 20 - 27
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++

//cycle 20: press right
right = 1;

@(negedge clk);
//should be at r1

//cycle 21: keep pressing right button

@(negedge clk);
//should be at r2

//cycle 22: keep pressing right button

@(negedge clk);
//should be at r3

//cycle 23: keep pressing right button

@(negedge clk);
//should return at s0 and go for another cycle

//cycle 24: keep pressing right button

@(negedge clk);
//should be at r1

//cycle 25: keep pressing right button

@(negedge clk);
//should be at r2

//cycle 26: keep pressing right button

@(negedge clk);
//should be at r3

//cycle 27: release  right button
right = 0;

@(negedge clk);
//should return at s0 and not go for a second run
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// press the left button and keep pressing
// so that the turn signal will go though 2 repetitions
// cycle 28 - 35
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//cycle 28: press left
left = 1;

@(negedge clk);
//should be at l1

//cycle 29: keep pressing left button

@(negedge clk);
//should be at l2

//cycle 30: keep pressing left button

@(negedge clk);
//should be at l3

//cycle 31: keep pressing left button

@(negedge clk);
//should return at s0 and go for another cycle

//cycle 32: keep pressing left button

@(negedge clk);
//should be at l1

//cycle 33: keep pressing left button

@(negedge clk);
//should be at l2

//cycle 34: keep pressing left button

@(negedge clk);
//should be at l3

//cycle 35: keep pressing left button

@(negedge clk);
//should return at s0

//+++++++++++++++++++++++++++++++++++++++++++++++++
// test resets from the middle of the pattern
//+++++++++++++++++++++++++++++++++++++++++++++++++

 //cycle 36: keep pressing left button
```

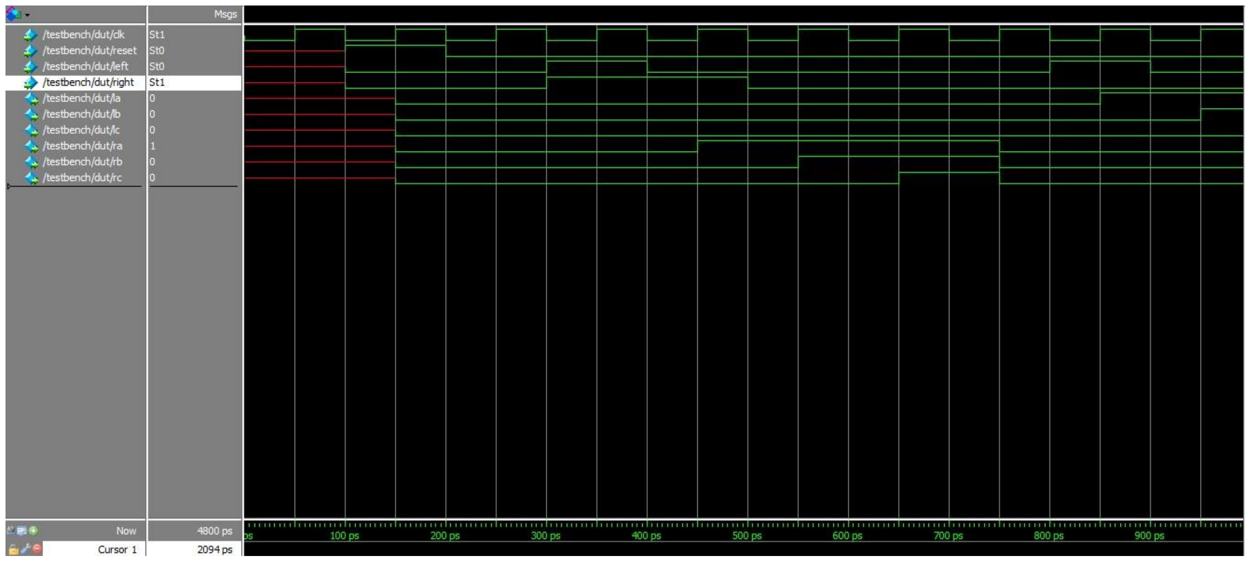
```
                        @(negedge clk);
                        //should be at l1

                        //cycle 37: release left button
                        left = 0;

                        @(negedge clk);
                        //should be at l2

                        //cycle 38: reset
                        reset = 1;

                        @(negedge clk);
                        //should be at s0

                        //cycle 39: release reset press right
                        reset = 0;
                        right = 1;

                        @(negedge clk);
                        // should be at r1

                        //cycle 40: reset while pressing right
                        reset = 1;

                        @(negedge clk);
                        //should be at s0

                        //cycle 41: keep pressing right while reset

                        @(negedge clk);
                        //should stay  at s0

                end
endmodule
```

Lab4_wrapper testbench code:

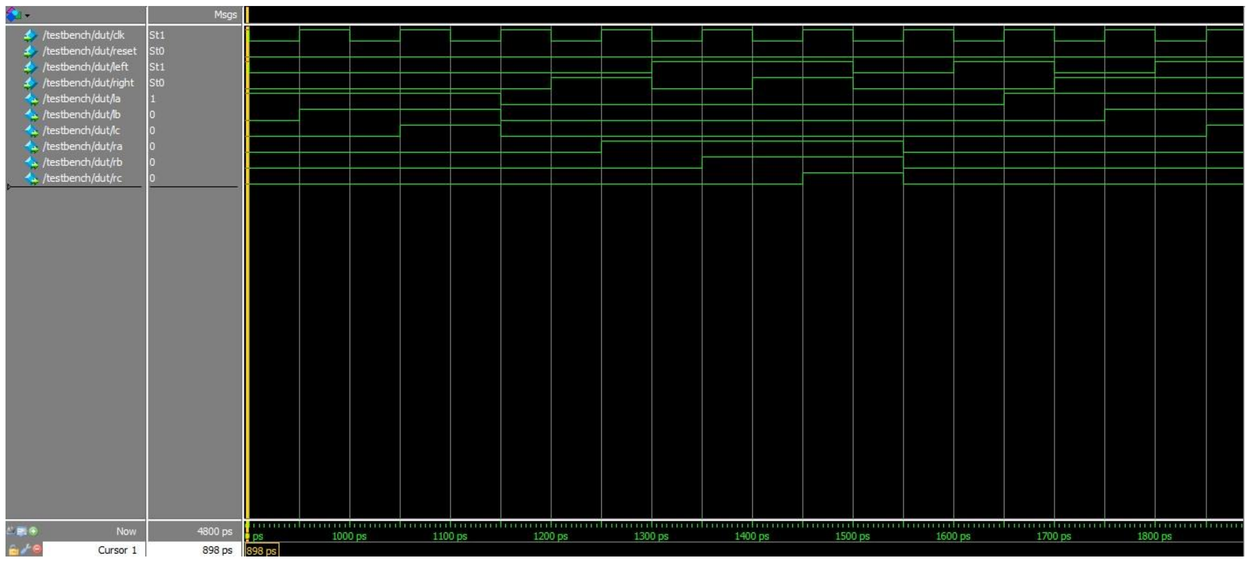
```
module project04_wrapper(input  logic [2:0] SW,
                input  logic [0:0] KEY,
                output logic [7:0] LED);

  // Use Key0 for clk
  // switches for inputs
  // red and green LEDs for output

  // Replace "xx" with your initials on the following line. Don't change
  // anything else.
  project04_al project04_al(.clk(KEY[0]), .reset(SW[0]), .left(SW[2]), .right(SW[1]),
                                                          .la(LED[5]), .lb(LED[6]), .lc(LED[7]),
                                                          .ra(LED[2]), .rb(LED[1]), .rc(LED[0]));
endmodule
```

ModelSim Simulation:

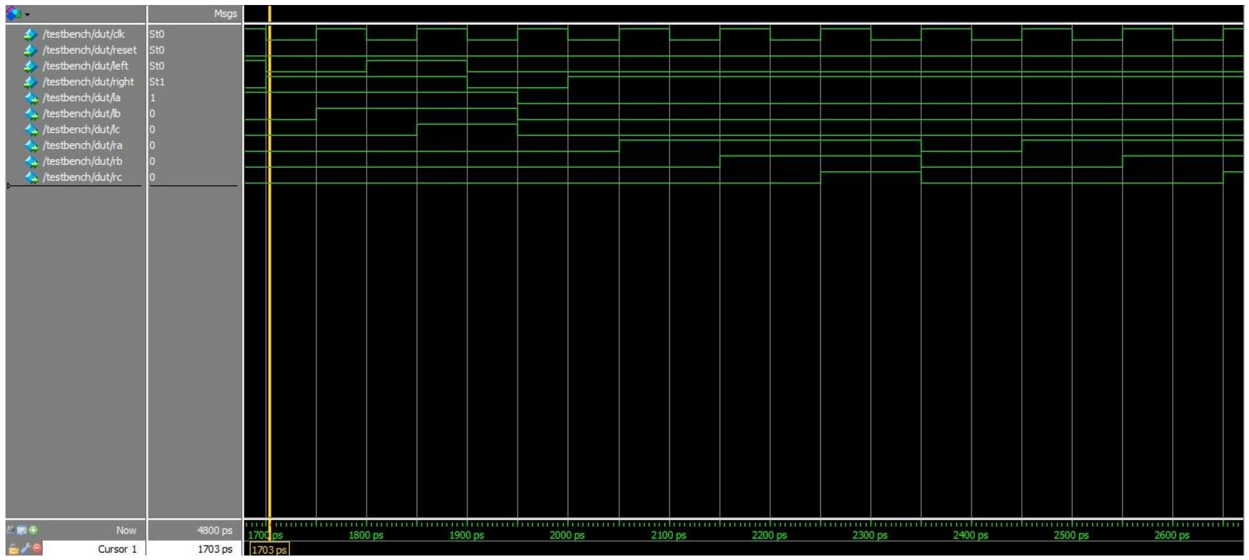
1st Simulation:



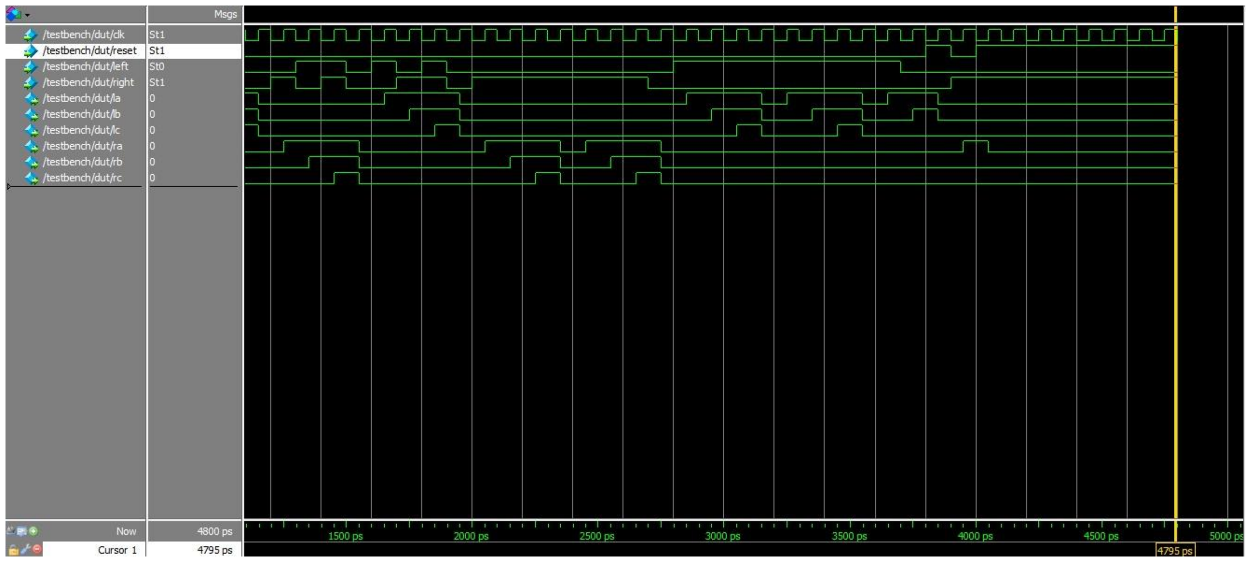
2nd Simulation:



3^{rd} Simulation:



4^{th} Simulation:



Hardware Implementation

The process for creating a wrapper module that will rename the inputs and outputs and connect them to switches and LEDs is the part of this lab that is most crucial. By adding \\bca181\Labs\lab4_wrapper.v to the directory and Quartus project and compile the design, correct any mistakes, review the compilation report, and review the resource usage summary. Check the wrapper to see which buttons are utilized for which inputs after downloading the design to the Altera DE2-115 FPGA board. Keep in mind that the clock was made using a pushbutton switch. We have to analyze the plan and observe the LEDs. Be aware that switches may undergo a phenomena known as bounce, in which the mechanical contacts bounce when the switch opens or closes, producing numerous rapidly rising and falling pulses as opposed to a single clock edge. Although it is feasible to design a circuit that will "debounce" a switch, it is outside of the scope of this lab.

Lab4 SystemVerilog Code:

```
module project04_al(input logic clk,
            input logic reset,
            input logic left, right,
            output logic la, lb, lc, ra, rb, rc);

        logic s0, l1, l2, l3, r1, r2, r3;
        logic s0_next, l1_next, l2_next, l3_next, r1_next, r2_next, r3_next;

        // next state equations
        assign s0_next = reset | (s0 & ((left & right) | (~left & ~right))) | l3 | r3;
        assign l1_next = ~reset & left & ~right & s0;
        assign l2_next = ~reset & l1;
        assign l3_next = ~reset & l2;
        assign r1_next = ~reset & ~left & right & s0;
        assign r2_next = ~reset & r1;
        assign r3_next = ~reset & r2;

        // state update
        always_ff@(posedge clk)
          begin
                s0 <= s0_next;
                l1 <= l1_next;
                l2 <= l2_next;
                l3 <= l3_next;
                r1 <= r1_next;
                r2 <= r2_next;
                r3 <= r3_next;
          end

        assign la = l1 | l2 | l3;
        assign lb = l2 | l3;
        assign lc = l3;
        assign ra = r1 | r2 | r3;
        assign rb = r2 | r3;
        assign rc = r3;

endmodule
```

Lab4_wrapper SystemVerilog code:

```
module project04_wrapper(input  logic [2:0] SW,
            input  logic [0:0] KEY,
            output logic [7:0] LED);

  // Use Key0 for clk
  // switches for inputs
  // red and green LEDs for output

  // Replace "xx" with your initials on the following line. Don't change
  // anything else.
  project04_al project04_al(.clk(KEY[0]), .reset(SW[0]), .left(SW[2]), .right(SW[1]),
                                                  .la(LED[5]), .lb(LED[6]), .lc(LED[7]),
                                                .ra(LED[2]), .rb(LED[1]), .rc(LED[0]));
endmodule
```

Conclusion:

Students gain a significant grasp of the significance and uses of finite state machines (FSMs) in digital design by working through a laboratory exercise in Thunderbird Turn Signal utilizing SystemVerilog code, the FPGA DE2-115, and FSM. Designing digital systems with complicated and sequential behavior requires the use of FSMs because they give the behavior of the system a clean, structured representation. Students may better grasp how to use FSMs in the design of systems used in the real world by using them in the Thunderbird Turn Signal project.