

# E85: Digital Design and Computer Architecture

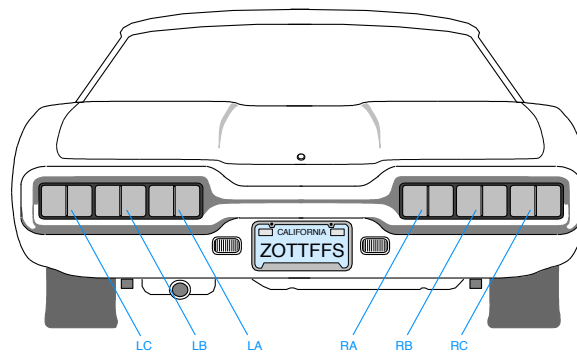
## Lab 3: Finite State Machine Design

### Objective

The purpose of this lab is to learn to design a finite state machine using structural SystemVerilog, debug it in simulation with a self-checking testbench, and download it onto an FPGA board.

### 1. Thunderbird Turn Signal

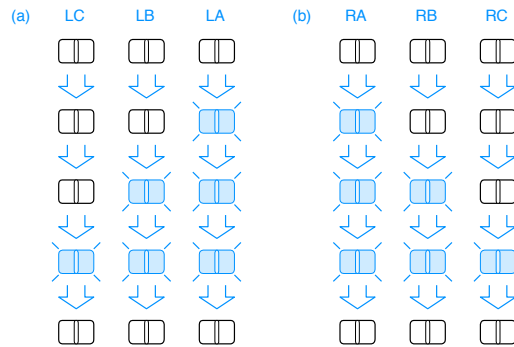
Your goal for this lab is to design a finite state machine in SystemVerilog to control the taillights of a 1965 Ford Thunderbird<sup>1</sup>. There are three lights on each side that operate in sequence to indicate the direction of a turn. Figure 1 shows the tail lights and Figure 2 shows the flashing sequence for (a) left turns and (b) right turns.



Copyright © 2000 by Prentice Hall, Inc.  
Digital Design Principles and Practices, 3/e

**Figure 1. Thunderbird Tail Lights**

<sup>1</sup> This lab is derived from an example by John Wakerly from the 3<sup>rd</sup> Edition of Digital Design.



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

The FSM should have two inputs, left and right, that trigger the flashing sequence on the cycle after they are asserted. At most one of the inputs will be asserted at any time. The FSM should have six outputs, LA, LB, LC, RA, RB, and RC. Once the sequence starts, it should continue even if the input is deasserted. When the sequence completes, it should go back to all lights off for a cycle before another sequence may begin. See the Thunderbird test vector file later in this lab for an example of the expected behavior.

Draw a state transition diagram for your FSM. Be careful to exactly follow the specification above because you will have to repeat all of the subsequent steps if your diagram is incorrect. Choose state encodings and write Boolean equations for the next state and output logic. The difficulty of this task is strongly influenced by the state encoding you choose. Sketch the circuit.

Write structural Verilog code for your FSM. Your FSM should have the following module declaration:

```
module lab3_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).

Simulate your FSM with the following self-checking testbench and vectors. Study the testbench and observe how it applies inputs and checks the outputs. Debug any discrepancies. You'll probably have errors at first. Get used to interpreting the messages from ModelSim and correct any mistakes. In fact, it's good if you have bugs in this lab because it's easier to learn debugging now than later when you are working with a larger system!

```
module testbench();
    logic      clk, reset;
    logic      left, right, la, lb, lc, ra, rb, rc;
    logic [5:0] expected;
    logic [31:0] vectornum, errors;
    logic [4:0] testvectors[10000:0];

    // instantiate device under test
    lab3_xx dut(clk, reset, left, right, la, lb, lc, ra, rb, rc);

    // generate clock
    always
```

```

begin
    clk=1; #5; clk=0; #5;
end

// at start of test, load vectors
// and pulse reset
initial
begin
    $readmemb("thunderbird.tv", testvectors);
    vectornum = 0; errors = 0; reset = 1; #27; reset = 0;
end

// apply test vectors on rising edge of clk
always @(posedge clk)
begin
    #1; {left, right, expected} = testvectors[vectornum];
end

// check results on falling edge of clk
always @(negedge clk)
if (~reset) begin // skip during reset
    if ({la, lb, lc, ra, rb, rc} != expected) begin // check result
        $display("Error: inputs = %b", {left, right});
        $display(" outputs = %b %b %b %b %b %b (%b expected)",
            la, lb, lc, ra, rb, rc, expected);
        errors = errors + 1;
    end
    vectornum = vectornum + 1;
    if (testvectors[vectornum] === 8'bx) begin
        $display("%d tests completed with %d errors", vectornum, errors);
        $stop;
    end
end
end
endmodule

```

```

// thunderbird.tv
// left right _ la lb lc ra rb rc
00_000000
10_000000
10_100000
10_110000
10_111000
10_000000
10_100000
00_110000
01_111000
01_000000
01_000100
00_000110
00_000111
00_000000

```

Synthesize your FSM. Look at the RTL Viewer. Does it match your expectations?

Assign pins. Use DIP switches<sup>2</sup> for clk, reset, left, and right, and LEDs for the six outputs. Resynthesize with the pin assignments.

Look at the compilation report in the Quartus Flow Summary. Find the number of registers and pins the design uses. Does that match expectations?

Under Analysis & Synthesis, look at the resource utilization summary. Check that the number of register and I/O pins matches your expectations.

---

<sup>2</sup> It is also possible to use the KEY<sub>3:0</sub> pushbutton, but beware that these are 1 when not pressed and 0 when pressed, and that at least the KEY0 on E85 board #7 appears to be damaged.

Test your design. Note that the switches sometimes experience a phenomenon called *bounce*, in which the mechanical contacts bounce as the switch is opening or closing, creating multiple rapid rising and falling pulses rather than a single clock edge. If your lights seem to skip through multiple states at a time, it is probably because of switch bounce on the clock switch. With a bit of practice, you can learn to push the switch in a way that bounces less. It is also possible to build a circuit to “debounce” a switch, but that is beyond the scope of this lab. Fix any other problems that you observe.

### **What to Turn In**

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. State transition diagram
3. State encoding and Boolean equations for next state and output
4. Sketch of circuit
5. Structural Verilog code
6. Simulation waveforms showing the FSM inputs and outputs. Did it pass the self-checking testbench?
7. RTL Viewer schematics. Do they match your expectations?
8. Did the tail lights function correctly on the DE2 Board?

If you have suggestions for further improvements of this lab, you’re welcome to include them at the end of your lab.