

EECE 144  
Fall 2011

Lab Report #13  
Section 4  
12/07/2011

Submitted by: Jeremiah Mahler

Signature

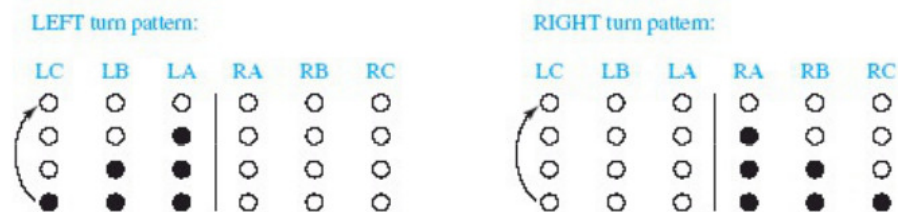
Printed Name

Date

	Jeremiah Mahler	Dec 7, 2011
	Marvane Johnson	Dec 7, 2011

## 1 Description/Objectives

The goal of this lab is to design a circuit in Verilog[1] which replicates the tail light operation of an older model Thunderbird.



Design a Moore sequential circuit to control these lights. The circuit has three inputs LEFT, RIGHT and HAZ. LEFT and RIGHT come from the driver's turn signal switch and cannot be 1 at the same time. As indicated above, when LEFT = 1 the lights flash in a pattern LA on; LA and LB on; LA, LB, and LC on; all off; and then the sequence repeats. When RIGHT = 1, the light sequence is similar. If a switch from LEFT to RIGHT (or vice versa) occurs in the middle of a flashing sequence, the circuit should immediately go to the IDLE (lights off) state and then, start the new sequence. HAZ comes from the hazard switch, and when HAZ = 1, all six lights flash on and off in unison. HAZ takes precedence if LEFT or RIGHT is also on. Assume that a clock signal is available with a frequency equal to the desired flashing rate.

[2, p. 547, prob. 16.27].

## 2 Procedure

There is more than one way to design this circuit. The method used here builds a state diagram and state table for one side and then to use this as a general guide in which to design the Verilog code.

From Figure 1 and Table 1, which describe the operation of the left turn signal, it can be discerned in general how the system will operate. The operation of the right turn signal is similar but with the states/values of right and left reversed.

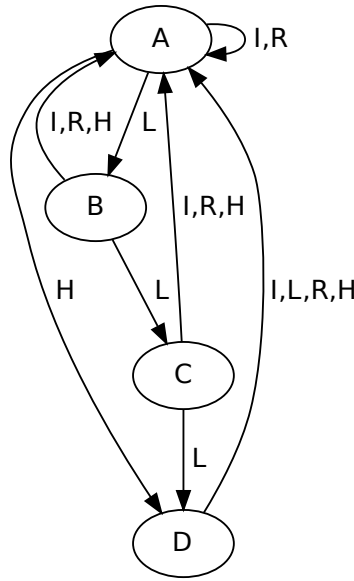


Figure 1: State diagram for left turn signal operation. State A has zero lights on (000), state B has one (001), state C has two (011) and state D has all three (111). Each transition is denoted with I as idle, R as right, L as left, and H as hazard.

$S$	$S^+$				$Z$
	$X = I$	$L$	$R$	$H$	
$A$	$A$	$B$	$A$	$D$	000
$B$	$A$	$C$	$A$	$A$	001
$C$	$A$	$D$	$A$	$A$	011
$D$	$A$	$A$	$A$	$A$	111

Table 1: State table for LEFT turn signal. The states of  $X$  are all mutually exclusive.

The source code for the Verilog definition of the tail lights is given in Listing 1. The module is only triggered on the positive edge of the clock (Line 13), similar to a flip-flop. The first decision level consists of a chain of **if else** statements which discriminate among the possible X input values (H, L, R, and the catch all state I). For each value the current state of the tail lights (TL, TR) is examined and a decision is made as to what state to assign next.

```

1  /*
2  *  Replicate the operation of tail lights on old Thunderbirds.
3  *
4  *  L: left signal
5  *  R: right signal
6  *  H: hazard signal
7  *
8  *  TL: left tail light
9  *  TR: right tail light
10 *
11 */
12 module taillights (input clk, L, R, H, output reg [2:0] TL, TR);
13     always @(posedge clk) begin
14         if (1 == H) begin
15             if (TL == 3'b000 && TR == 3'b000) begin
16                 TL <= 3'b111;
17                 TR <= 3'b111;
18             end
19             else begin
20                 TL <= 3'b000;
21                 TR <= 3'b000;
22             end
23         end
24         else if (1 == L) begin
25             TR <= 3'b000;
26
27             if (TL == 3'b000) begin
28                 TL <= 3'b001;
29             end
30             else if (TL == 3'b001) begin
31                 TL <= 3'b011;
32             end
33             else if (TL == 3'b011) begin
34                 TL <= 3'b111;
35             end
36             else begin
37                 TL <= 3'b000;
38             end
39         end
40         else if (1 == R) begin
41             TL <= 3'b000;
42
43             if (TR == 3'b000) begin
44                 TR <= 3'b001;

```

```

45             end
46             else if (TR == 3'b001) begin
47                 TR <= 3'b011;
48             end
49             else if (TR == 3'b011) begin
50                 TR <= 3'b111;
51             end
52             else begin
53                 TR <= 3'b000;
54             end
55         end
56     else begin
57         TL <= 3'b000;
58         TR <= 3'b000;
59     end
60 end
61 endmodule

```

Listing 1: Verilog source for the tail lights.

## 2.1 Compiling Verilog source and running GTKWave

Once all the code has been defined it can be compiled and run. The test bench is given in Appendix A.

To compile the source code using Icarus Verilog[1] under Linux the following command can be run.

```
iverilog test.v
```

This will produce a file named 'a.out'. Under Linux this can be executed directly.

```
./a.out
```

Alternatively the `vvp` command can be used. This method works under Linux or Windows.

```
vvp a.out
```

Because the `$dumpfile` and `$dumpvars` have been added to the test bench (Appendix A) it will produce an output file suitable for GTKWave[3]. The file should have the extension '.vcd' and can be run as shown below.

```
gtkwave output.vcd
```

And from within GTKWave the different variables can be selected and displayed.

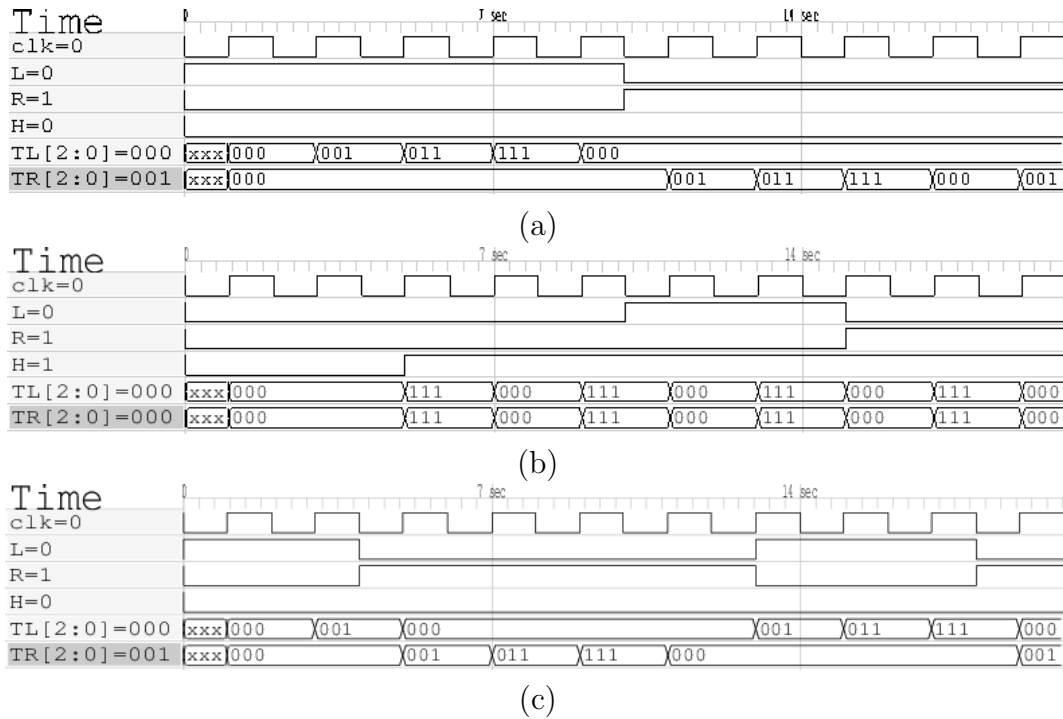


Figure 2: Wave forms for of the tail lights in various situations. In (a) it goes from from left and then switches to right. In (b) it starts at idle, then the hazards are turned on, then left turn signal is engaged, and then the right turn signal is engaged. Notice that the hazard has priority and that the left and right have no effect. In (c) it is switched back and forth from left to right to show the transitions when switch midway through a cycle.

### 3 Observations

The output for various operations of the turn signal are given in Figure 2. It can be seen that when it is in either the left or right position it progresses through the proper tail light sequence. It also handles transient situations where it is changed from left to right mid way through a sequence. And the hazard operation alternates from all on to all off as it is unaffected by either left or right being actuated at the same time.

### 4 Conclusion

This lab was a complete success in implementing a design for old Thunderbird tail lights in Verilog. All design requirements were met and the tail lights behaved as expected.

## 5 References

- [1] S. Williams, “Icarus verilog.” <http://iverilog.icarus.com/>, 2011.
- [2] C. Roth Jr., Fundamentals of Logic Design. Cengage Learning, 2009.
- [3] T. Bybell, “Gtkwave.” <http://gtkwave.sourceforge.net/>, 2011.

## A Verilog Test Bench

```
1  /*
2  *  Test bench tail lights.
3  *
4  *  To compile this file run:
5  *
6  *    iverilog test.v
7  *
8  *  Run the executable:
9  *
10 *    ./a.out
11 *  OR
12 *    vvp a.out
13 *
14 *  Because $dumpfile and $dumpvars have been
15 *  added it will generate data for gtkwave
16 *  in gtkwave-output.vcd.
17 *
18 *  This output file can then be shown with Gtkwave.
19 *
20 *    gtkwave gtkwave-output.vcd
21 *
22 *  And from within Gtkwave you can pick and choose
23 *  variables to see their waveforms over time.
24 *
25 *
26 *  This project was completed as part of lab 13 in the
27 *  class EECE-144 taught by Kurtis Kredo II at Chico State
28 *  during the Fall of 2011.
29 *
30 *  Author(s): Jeremiah Mahler <jmmahler@gmail.com>
31 *              Marvanee Johnson
32 */
33
34 'include "taillights.v"
35
36 module test;
37     reg clk, L, R, H;
38     wire [2:0] TL, TR;
39
40     taillights t11 (.clk(clk), .L(L), .R(R), .H(H), .TL(TL), .TR(TR));
41
42     initial begin
43         $dumpfile("gtkwave-output.vcd");
44         $dumpvars(0, test);
45
46         // initialize
47         L = 0;
48         R = 0;
49         H = 0;
```

```

50     clk = 0;
51
52         // configure how to run the test
53
54         /*
55         // Left
56         L = 1; R = 0;
57
58         // Right
59         #10 L = 0; R = 1;
60         */
61
62     /*
63     // Hazard
64     H = 0;
65     #5 H = 1;
66     // engaging the left or right should not change anything
67     #5 L = 1;
68     #5 L = 0; R=1;
69     */
70
71     // back and forth from left to right
72     H = 0;
73     L = 1; R = 0;
74     #4 L = 0; R = 1;
75     #9 L = 1; R = 0;
76     #5 L = 0; R = 1;
77
78     #2 $finish;
79 end
80
81 always begin
82     #1 clk = ~clk;
83 end
84 endmodule

```