CS 61C (Clancy) **Lab assignment for week 12**
Spring 2003

Goals

In this lab exercise you will gain experience designing and simulating a simple sequential logic circuit. It will give you practice using Verilog for logic simulation and (we hope) an appreciation for the issues involved in designing, testing, and debugging sequential logic circuits.

Reading

Sections 6–9 of "CS 61C Verilog Tutorial"; P&H section B.6.

Working with partners

If you work with a partner on these exercises, make sure that you both understand all aspects of your solutions. Where you split work among partnership members, expect your lab t.a. or lab assistant to ask you to explain the work of your partner.

Background

Appendix B in P&H present a description of a circuit used to control a traffic light. Your job in this lab is to implement and test the traffic light controller in Verilog.

The details of the traffic light controller finite state machine are explained in P&H. You should follow that specification with one modification. Add an input signal called RST. Asserting this signal should "reset" the controller to the NSgreen state (on the next positive clock edge) regardless of its current state and the value of the other inputs. The easiest way to achieve this behavior is to wire the RST signal to the RST input on the flip-flop used to store the current state. Otherwise your circuit should be identical to the one described in the book.

Your circuit will need to include a flip-flop to hold the current state. Below is the Verilog source, provided in the file ~cs61c/labs/lab12/light.v, for the flip-flop module to use in your design.

```
//Behavioral model of D-type Flip-flop:
// positive edge-triggered,
// synchrounous active-high reset.

module DFF (CLK,Q,D,RST);
   input D;
   input CLK, RST;
   output Q;
   reg Q;
   always @ (posedge CLK)
     if (RST) Q = 0; else Q = D;
endmodule // DFF
```

Exercise 1 (1 checkoff point)

Copy ~cs61c/labs/lab12/light.v to your directory. Then add a new module to light.v that implements the next-state combinational logic for the traffic light controller. Run it through the Verilog compiler. Test it with one or two input cases and show the result to your t.a. for checkoff.

Exercise 2 (2 checkoff points)

Add a new module that implements the traffic light controller. This module must include instances of your next-state combinational logic and a flip-flop to hold the current state. Test it with one or two input cases and show the result to your t.a. for checkoff.

Use the following module and port names:

```
module light (NSlite, EWlite, NScar, EWcar, CLK, RST);
   output NSlite, EWlite;
   input  NScar, EWcar;
   input  CLK, RST;
```

Exercise 3 (1 checkoff point)

Now write a test bench and use it to test and debug your controller. Follow the procedures for testing finite state machines, presented in the CS61C Verilog Tutorial; in particular, make sure you test every possible transition from every state. Display your output to your t.a. for checkoff.

Having done this lab ...

We expect that, having done this lab assignment, you will understand the use of a flip-flop to store state and will be familiar with the pattern of splitting a finite state machine into the controller, which maintains the current state, and the next-state and output combinational functions.