Lab 5: Example State Machine: Traffic-Light Controller

OBJECTIVES

The objective of this lab is to design and implement a state machine using a PLD. You will use an ASM to design a traffic light controller and implement this state machine on your proto-board using your UF-3701 PLD board, switch circuits and LED circuits.

MATERIALS

- Your lab kit (including your NAD/DAD)
- Read the *Creating Graphical Components* document on both our Lab and Software/Docs web pages. Download this document onto your computer.

SPECIFICATIONS

Design a state machine to control the traffic lights at an intersection in Gainesville's midtown, the intersection of West University Avenue and NW 17th Street. To reduce the number of LEDs needed in lab, you only need to control the light outputs for West University Avenue, not NW 17th Street (Buckman Drive). The traffic light controller has three **active-high** outputs (`Red, Yellow, Green`), and two inputs: **c**ar **w**aiting at 17th Street [`CW(H)`] and **e**mergency **v**ehicle approaching [`(EV(L)`].

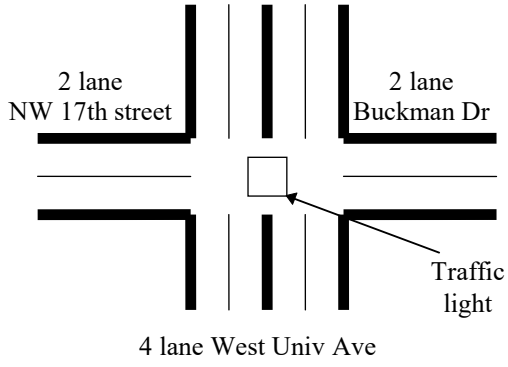

Figure 1. Traffic Light

TIMING:

1. The University Avenue green light normally stays on for six cycles. However, if an emergency vehicle comes up to the light (while traveling down the 17th Street), `EV` will go true and this can occur during any of the six green cycles. If this occurs, the light should **immediately** turn yellow and stay on for at least one clock cycle (but no more than two) and then proceed on to step #4. Otherwise, if `EV` is not true after the **six** cycles, move to step #2.
2. After the University Avenue green light has been on for six cycles as described in step 1, check if there is a car waiting (`CW` is true). If so, then continue to step #3; else go back up to step #1.
3. `Green` turns off and `Yellow` turns `on`. This lasts for two clock cycles.
4. `Yellow` turns off and `Red` turns on for four clock cycles.
5. If EV is not true, go back to step #1. Otherwise, as long as EV remains true, keep the light Red.

A few notes about our traffic lights and traffic light controller are listed below.

1. There should **never** be more than one traffic light on at a time.
2. There should **always** be a single traffic light on.
3. You can assume that an emergency vehicle will take at least one clock cycle to approach an intersection.

PRE-LAB REQUIREMENTS

1. Draw an ASM chart for your controller. Note: You should **never** have a situation when two lights are on at the same time, i.e., Green and Yellow should never be on at the same time.
   - The simplest way to create a multiple clock delay (e.g., step 1 in the Timing section and step 4) is to have successive states. This is precisely what you should do. (An alternative, **NOT** recommended here, but used in EEL 4712: *Digital Design*, is to design a counter to count the required number of states. In EEL 4712, VHDL is used to design several different types of counters, making this a relatively simple procedure.)
2. Create a next-state truth table showing the current state, input, next state, flip-flop inputs and system outputs (i.e., red, green or yellow) for each state in your ASM diagram.
3. During previous labs during the semester, you designed your circuits using logic gates (by drawing the circuit elements in Quartus). This is called *schematic entry.* You **could** create a Quartus schematic entry design using D flip-flops and SSI logic gates (as you have all semester, and recently, for implementation in the PLD); but you do **NOT** need to do this for this lab. (Do this only if you want to; there will be **no** credit for doing this part and this design should **NOT** be submitted.) In part 4 you will use VHDL to replace the combinatorial **part** (not the flip-flops)

of your schematic entry design, and this **is** required.

Note: Simplification of the equations is helpful, but not required in this lab. Quartus will simplify for you! After compiling your design, make a Quartus equation file by selecting Processing | Start | Start Equation Writer (Post-Synthesis). The generated file will have an ".eqn" extension. Open this file to observe the equations. Some of the operators in Quartus equation files are as follows: & = AND, ! = NOT, # = OR, and $ = XOR (exclusive OR).

- If you want to (**NOT** required), simulate this design in Quartus. There will be **no** credit for doing this part.

4. Design a solution for this problem using D flip-flop(s) for all of the state bits, but this time use a VHDL program instead of the combinatorial circuits that drive the flip-flops. The design should be named **`Lab5_DFF_Traf_Cont`**. Your design will consist of a VHDL and a bdf file; the bdf file will have an instantiation of the VHDL design and the required connections to the flip-flops. (See the *Creating graphical components* file on the website for information on how to incorporate VHDL code in a bdf file.) Output the state bits (the outputs of the flip-flops) so that you always know the present state of your system. Having these state bit outputs available greatly simplifies the debugging process.
5. Simulate this design in Quartus. As always, annotate your design simulation.
6. Program your PLD with your **`Lab5_DFF_Traf_Cont`** design. Use your NAD/DAD for the non-clock inputs and outputs. Use a debounced switch circuit for your clock input (as used in previous labs).

   Note: You could use your NAD/DAD for your debounced clock input too, but I want you to get some more experience building a debounced switch circuit, so do **NOT** do this for this lab. I suggest that you use your PLD and not a 74xxx chip for your debounce circuit.

   Warning: Always tri-state all unused PLD pins, as described in the Quartus Tutorial.

IN-LAB

Demonstrate your design from Pre-Lab part 4.