Lab 5: State Machine Example: Traffic Light Controller

OBJECTIVES

- Use an ASM diagram to describe the behavior of a complex state machine.
- Use VHDL to simplify combinational logic design.
- Understand the difference between asynchronous and synchronous state machine outputs (i.e., Mealy and Moore outputs).

INTRODUCTION

State machines are extremely common in large digital circuits. So far, we have only designed simple counter state machines (in Lab 3), where the outputs of the state machine were the current state bits of the machine. However, most state machines in the real world have additional logic that creates the outputs of the state machine from its current state and additional user inputs. To help design these more complex state machines, we will use an ASM diagram to provide a graphical description of our state machine's behavior. We will then translate the ASM diagram into a next-state truth table. We will then augment this truth table (after choosing the type of flip-flops) in order to help us implement the logic required for the state machine.

This lab will also introduce you to VHDL. Hardware description languages such as VHDL and Verilog are preferred by digital designers to increase the speed of development and increase the ease of debugging. Most electronic design automation (EDA) tools such as Quartus support these languages as a standard way of specifying the behaviors of digital circuits. In fact, Quartus translates your BDFs into AHDL code (an internal Altera hardware description language) when it is compiling them for upload to the DE10-Lite and for simulation in the Waveform Viewer. While VHDL can be used to design combinational and sequential logic, this lab (and course) will only focus on the implementation of combinational logic. Sequential logic can be very obtuse in VHDL and is outside the scope of this introductory course.

LAB STRUCTURE

In this lab, you will design a traffic light controller state machine. You will start by designing an ASM diagram that describes the behavior of the state machine. After creating the corresponding next-state truth table and adding columns based on your choice of flip-flops, you will then derive next-state and output logic equations for your state machine that can be implemented in VHDL. Finally, you will make a symbol for the VHDL combinatorial circuit to include in a BDF that connects your combinational logic with the flip flops and I/O pins needed for your design.

REQUIRED MATERIALS

- Creating Graphical Components in Quartus

SUPPLEMENTAL MATERIALS

- *DE10-Lite Manual*
- *DE10-Lite Schematic*
- *DE10-Lite Pins*
- *DE10-Lite Safe Program*

PRE-LAB PROCEDURE

1. STATE MACHINE DESIGN

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 the 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, one active-low and one active-high: car waiting at 17th Street [`CW(L)`] and emergency vehicle approaching [`EV(H)`].

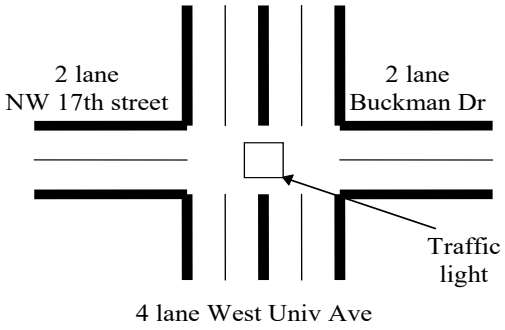

Figure 1. Traffic Light

TIMING SPECIFICATIONS:

1. The University Avenue green light normally stays on for **five** 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 **five** 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 **five** cycles, move to step #2.
2. After the University Avenue green light has been on for **five** cycles as described in step 1, in the **fifth** cycle, 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.

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 or allowed 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 combinational **part** (not the flip-flops) of your schematic entry design, and this **is** required. Since only combinational logic is required **(or allowed)** in VHDL for 3701 this semester, VHDL process blocks are unnecessary (**not** allowed), and should **not** be utilized in our course.

Note: Simplification of the equations is helpful, but not required in this lab. Quartus will automatically simplify for you (and **always has**) and can also generate an optimized circuit (even when the circuit is made in VHDL)!

To view the simplified equations made by Quartus, do the following: 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 in the output_files folder. 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).

To view the optimized circuit made by Quartus, do the following: You can view the generated circuit for your design by compiling your design and then running Tools | Netlist Viewers | RTL Viewer. The circuit may look very complex, but it will be optimized for use on our DE10-Lite board.

- 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 the state bits, but this time use a VHDL program instead of the combinational 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. ~~Make a debounced switch circuit for your clock input using the red SPDT switch included in your lab kit. Use your DE10-Lite for the NAND or NOR part of the debouncing circuit. You will add the necessary logic to the bdf file described above.~~ Use KEY0 or KEY1 on the DE10-Lite for your clock input.
7. Program your PLD with your **Lab5_DFF_Traf_Cont** design. You can use your DE10-Lite or your DAD for the non-clock inputs and outputs. Use a debounced switch circuit for your clock. If using your DAD as I/O for your DE10-Lite, it may be easier to use the 2x20 male GPIO headers on the DE10-Lite. See Chapter 3.5 of the *DE10-Lite Manual* for the associated pins.

   Warning: Make sure that you have programmed your DE10-Lite with the *DE10-Lite Safe Program* as a POF file. Afterwards, only program your DE-10 Lite with your design for this lab as a SOF file.

PRE-LAB PROCEDURE SUMMARY

1. Design a state machine that will control a traffic light according to strict timing specifications in § 1.

IN-LAB PROCEDURE

In lab, you will demo your state machine from Part 7 functioning on your DE-10 Lite.