CS/EE 3710 — Computer Design Lab
Lab 1 — Finite State Machine Implementation
and Spartan-3E board mapping

Due Thursday, September 9th, 2010

Laboratory Objectives

This lab will be to implement a simple finte state controller to refresh your memory, and to learn how to synthesize and map designs to the Spartan-3E FPGA board. You may have already built this FSM in a previous class. That's fine. The point of the lab is to refresh your memory about finite state machines, and to learn how the new Xilinx boards work.

1. Design a state machine based on a written description of the operation of the system. You can design the circuit using Verilog, schematics, or a mixture of both.
2. Simulate your state machine by simulating the clock signal and checking that the circuit goes through the appropriate set of states.
3. Map your circuit to the Spartan-3E board. You can use the built-in 50MHz clock on the Spartan-3E board for your system clock, and you can use the pushbuttons and LEDs that already exist on the board for inputs and outputs (defined later in this document).
4. Demonstrate your working circuit.

Thunderbird Turn Signals

The Ford Thunderbird (or at least a simplified model of a Thunderbird) has three tail lights on each side of the rear of the car. When you engage the turn signal, the lights flash in sequence to indicate the direction you are going to turn. For example, a very crude drawing of the back of the car is shown in Figure 1. You can see three lights to indicate a left turn numbered L0, L1, and L2, and three lights to indicate a right turn numbered R0, R1, and R2. These lights operate in sequence to indicate a turn. An example sequence of lights for a left turn is shown in Figure 2. In this figure, the open boxes are lights that are off, and the filled-in boxes are lights that are on. Thus, the filled-in boxes imply asserted signals that turn on the lights. Basically, the lights light up in a sequence that points to the direction that you will be turning the car. The other use for these lights is a hazard indicator. If a hazard is being indicated, all six lights flash on and then off alternately.

Your job is to design a finite-state controller for these lights. There are three inputs, LEFT, RIGHT, and HAZ. There are six outputs, L[2:0] and R[2:0] one for each of the six tail lights. The operation of the machine is as follows: Assume the clock controlling the machine is running at the same frequency as the desired flash of the lights. Anytime the HAZ input is asserted, all six lights should flash on at the same time and then off at the same time on successive rising clock edges. If there is no hazard, and LEFT is asserted, the lights should sequence through the left-turn sequence. The lights should go through a complete left-turn sequence even if LEFT is de-asserted sometime in the middle of the sequence or if RIGHT is asserted in the middle of a sequence. If HAZ is asserted in the middle of a left-turn sequence,

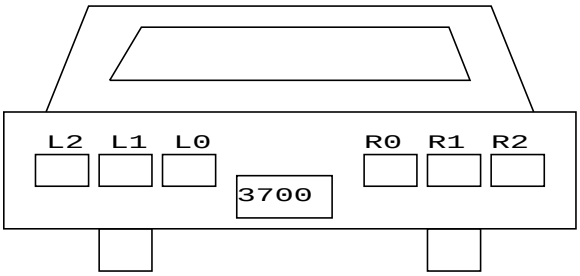

Figure 1: 1965 Ford Thunderbird Tail Lights

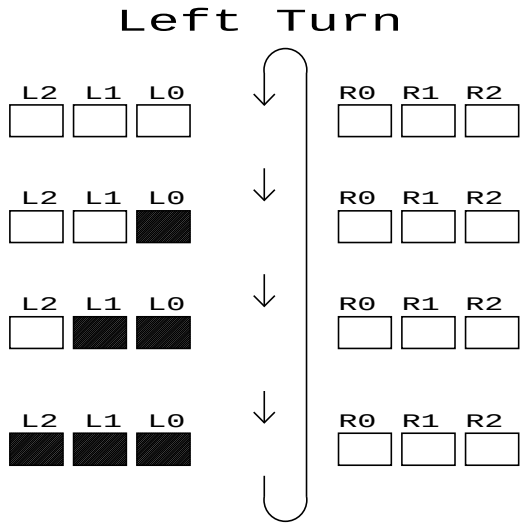

Figure 2: Tail Light Sequence for Left Turn

you should immediately start hazard-flashing. In other words, HAZ has precedence over LEFT or RIGHT. RIGHT causes a similar sequence of actions indicating a right turn. You may assume that LEFT and RIGHT will never be asserted simultaneously.

You should use pushbutton switches on the Spartan-3E board for LEFT, RIGHT and HAZ. These buttons surround the rotary knob on the Spartan-3E board. You should use BTN_WEST for LEFT, BTN_EAST for RIGHT, and BTN_NORTH for HAZ. The UCF information for these buttons is as follows. Use this information to fill out the required information in the PlanAhead tool, but use your own signal names (See the ISE 12.2 Tutorial on the class web site for details).

```
NET "BTN_EAST"  LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_NORTH" LOC = "V4"  | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_WEST"  LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
```

Note that all parts of this UCF information are important. In particular, if you don't specify a PULLDOWN resistor in the Xilinx pad, the buttons won't work. The buttons are logic low until pushed. When they're pushed they become logic high. See Chapter 2 in the **Spartan-3E FPGA Starter Kit Board User Guide** linked to the class web site for more details.

For reset, you should use SW3 which is the leftmost of the sliding switches on the Spartan-3E board. The UCF information for the sliding switches is:

```
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
```

For your outputs you can use the eight green LEDs on the Spartan-3E board which are right above the slider switches. Use LED(7:5) for the left turn lights, and LED(2:0) for the right turn lights. The UCF information for the LEDs is:

```
NET "LED<7>" LOC = "F9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

The Spartan-3E board has a built-in clock oscillator with a frequency of 50MHz. Your design will need to include a clock divider to get that down into the range where it's slow enough to be useful as a turn signal. Think about the frequency at which you see real turn signals and hazard lights blink. That's the frequency you're aiming at. The clock is available on the following pin on the Spartan-3E board:

```
NET "CLK_50MHZ" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
```

Again, this is all documented in greater detail in the **User Guide** linked to the class web site. A tutorial which describes, among other things, how to apply the UCF information in an ISE project in ISE 12.2 is also linked to the class web site.

Please use a strictly synchronous design style for this (and subsequent) assignments. That is, the ONLY thing that should go into a clock input is a clock signal. On this board that means that the 50MHz

clock on bin C9 is the only signal that should go into a clock input. If you want to operate with a slower clock, use a divider to generate an enable signal and condition the behavior of your device on the value of that enable signal. The slower signal should be used as an enable and NOT as a slow clock. That is, I'd like to see things like:

```
// An example of a register that only captures data when En is high.
// This particular En is assumed to be high for a single clock
// cycle when it's time to capture the data
always @ (negedge clr, posedge clk)
   if (clr == 0) state = idle;  // idle state if clr is asserted
   else if (En) state = NS;     // If En is high, capture new data
                                // otherwise, hold previous value (default)
```

Procedure

The design part of any finite state machine project involves producing a state diagram from the written description of the behavior. You need to understand what the sequence of events is that you want the machine to go through, which inputs it is using to direct those state transitions, and what the outputs will be in each state.

Once you have a state machine you can proceed in one of two ways: you can design the next state logic, output logic, and state register by hand and enter them as a schematic, or you can write a description of the state diagram in Verilog. Even if you use Verilog, make a schematic symbol for the Verilog code so that the top-level description of your system is a schematic.

If you decide to build your state machine as a schematic directly (instead of writing Verilog), you can use any of the gates in the Xilinx library. Note that the flip flops are under the Flip_Flop category in ISE. The components that start with "fd" are "Flip-Flops, D-type." The number is the width of the flop. That is, fd8 has 8 flip flops in the symbol with a common clock. A "c" in the name says that it has an asynchronous clear signal whereas an "r" means that the reset is synchronous. An "e" is a flip flop with an additional enable signal.

Once you have a state machine that is simulating correctly, follow the procedure in the ISE 12.2 Tutorial on the class web site and synthesize, map, and program the circuit onto the Spartan-3E board. The working circuit on the Spartan-3E board should be checked off by the instructor or TA.

What to Turn In

1. A README file that describes your results, problems you encountered, and what files you're including in the rest of the documentation.
2. A complete, documented, design of the state machine. This should include a neatly drawn and labeled state diagram.

   If you've designed the state machine circuit by hand then you need to include your state table, state encoding, next-state and output equations, and K-maps etc. that show how you implemented that logic. You should include all schematics for the circuit.

   If you've used Verilog for the state machine then you need to include the commented Verilog code, and a schematic showing how the Verilog is used in the top-level system. You also need to turn in all the other schematics/code that you used for other parts of the system.

3. A circuit simulation using ISE. The testbench file that applies inputs and checks them using "check" statements should submitted along with a log file of simulation results. Make sure you think about what it means to test a sequential circuit whose description is a state diagram. You need to run the machine through enough cycles to demonstrate that it really is doing the right thing. Please comment your command file, and remember that you don't want to have to simulate for 50,000,000 cycles before a single state change happens. Simulate the part of the circuit that runs at the human-speed clock. In this case you can turn in the waveforms that show the circuit moving through the states instead of explicit Verilog testbench checks of each of those states.

4. Demonstrate your working circuit to the TA or instructor.