

CS/EE 3700 — Digital System Design
Lab 5 — Finite State Machine Implementation

Circuit demo in your lab in the week of April 7th
Documentation due on Tuesday April 15th.

Laboratory Objectives

This lab will require you to design and implement a finite state machine on the XSA/XST board combination. In particular you will:

1. Design a state diagram based on a written description of the operation of the system.
2. Demonstrate your ability to turn your state diagram into a circuit, either by describing the circuit as a Verilog program and synthesizing with ISE, or by following the “hand design” procedure with schematics.
3. Whichever method you use, you should have a top-level schematic showing the entire circuit.
4. Simulate your state machine by simulating the clock signal and checking that the circuit goes through the same set of states as the state diagram.
5. Map your circuit to the XSA/XST board. You can use the built-in 100MHz clock on the XSA board for your system clock, and you can use the pushbuttons and LEDs that already exist on the board for inputs and outputs.
6. 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

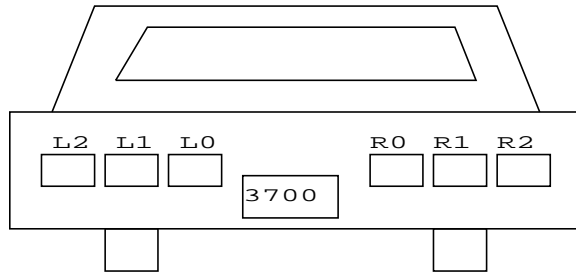


Figure 1: 1965 Ford Thunderbird Tail Lights

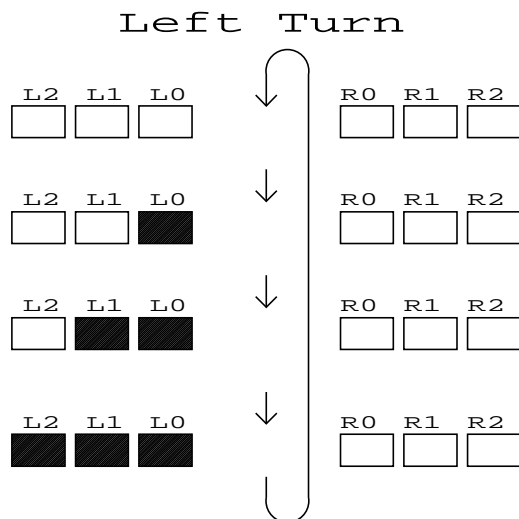


Figure 2: Tail Light Sequence for Left Turn

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, 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 XST board for LEFT, RIGHT and HAZ. I recommend PB5 for LEFT, PB4 for HAZ, and PB3 for RIGHT. Use the pushbutton on the XSA board for your master reset signal.

For outputs use the bar-LED on the XST board. Your right turn signal LEDs should show up on segments 3-2-1 and your left turn signal should show up on segments 10-9-8. The pin assignments for these buttons and LEDs can be found in the Pin document on the class web site.

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.

Whichever way you come up with a circuit, you need to test the circuit by simulating the operation of the state machine as the clock signal makes transitions. For this circuit the clock signal is simply another input to the circuit and you can control its value in your Verilog testbench the same way you would control any other input.

The Real Clock

The XSA board has a built-in clock oscillator whose default frequency is 100MHz. You can modify this clock rate using the GXSETCLK program and entering a divisor between 1 and 2052, but I don't recommend doing this. The lowest you can make the frequency is 48.7kHz so you'll still need to divide it down anyway. Instead, I would use the straight 100MHz frequency and build a circuit in your project to divide the 100MHz clock down to something in the range of 2Hz so that the clock ticks are at a good rate to be seen by a human looking at the tail lights. The 100MHz clock can be used in your circuit though the input on pin P88 on the Xilinx part. Simply assign the input to that pin and you'll get the 100MHz clock on that signal.

What to Turn In

1. 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 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.

2. A circuit simulation using ISE. The testbench file that applies inputs and checks them using "check" statements should be 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.
3. Demonstrate your working circuit in your lab.