SANTA CLARA UNIVERSITY	ELEN 127 Fall 2018	Jim Lewis

Laboratory #4: State Machine

For lab section on October 19, 2018

I. OBJECTIVES

Implement and test a state machine of modest complexity.

PROBLEM STATEMENT

A simple game, not unlike the dice game we saw in lecture. It's called the farmer game. A farmer has a fox, a chicken, and a bag of seeds on one side of a river and needs to get all three to the other side. He can only take one item across at a time. If he leaves the fox alone with the chicken, the fox will eat the chicken. If he leaves the chicken alone with the seeds, the chicken will eat the seeds.

At each step, the user/player decides how the farmer crosses the river: by himself, with the fox, with the chicken, or with the seeds.

PRE-LAB

(i) You're going to define the state machine for this, but first define what your inputs are going to look like. The inputs to this game come from the user/player, who decides how the farmer crosses the river at each step. The options are that he crosses by himself or that he cross with one of the three other entities (fox, chicken, seeds). So there are really four possible "moves". But we'll need another control signal to indicate that a selection has been made and the farmer should actually move. And we'll need a reset input to make sure the state machine is in a known starting state.

Clearly define how you will have these four selections, plus the reset and the indication of a valid move, provided as inputs to your state machine. In other words, if you were going to encapsulate the state machine in a Verilog module (which you will be), define what the input signals would be in the module definition, including widths if any of the inputs are to be buses.

(ii) Now let's clearly define the outputs.

As this is a game, there should be both a win and a lose indication. A win means the farmer successfully moved all the entities across. A lose means the famer either left the fox with the chicken, or the chicken with the seeds.

There also needs to be an "invalid move" indication. For example, if the farmer is on one side of the river and the fox is on the other, an input that says the farmer should take the fox doesn't make sense. So that should flag "invalid move".

Finally, to provide visibility for testing purposes, there should be some set of output signals to indicate the current position of each entity (the farmer, the fox, the chicken, and the seeds). You will use these outputs in your testbench to make sure you know where everything is at any given point in time.

Define exactly what you would call these outputs in your Verilog module. Again, if any of the outputs are buses, make sure the width is clear.

(iii) Consider that there are four entities here (the farmer, the fox, the chicken, and the seeds) and each entity can be in exactly one of two locations (the left bank or the right bank). So there are 16 possible combinations of where each entity is at any given time, i.e., there are 16 states. The starting state is that all four entities are on the left bank of the river.

Create a state diagram (or SM chart) that maps out the state transitions. Note that "invalid" moves should keep the machine in the same state. And once the machine reaches the winning state or any of the losing states, it should stay where it is until the reset signal is asserted.

Note that having the state machine coded in Verilog is not a requirement for pre-lab, but it is highly recommended that you have your code entered and ready to test before arriving at lab.

(iv) Outline a handful of important testcases that you will run against your design to make sure it's working properly. The TA will give you another template for your testbench.

LABORATORY PROCEDURE

You will again be paired. Decide between you and your partner who will work on step 1 (the game state machine) and who will work on step 2 (the testbench). When you're individually done with these two steps you can then move to step 3 to see if your design works, and debug if it doesn't.

1. Implement the state machine

   (i) Using the state diagram/SM chart you created in pre-lab, write the Verilog code to implement the behavior of the game.

2. Implement your testbench

   (i) Using the template provided by the TA, set up the code that will apply the testcases that you and your partner have identified.

3. Connect and simulate

   (i) Instantiate your DUT, set up a clock, and simulate the whole thing to make sure it appears to be working properly. Once you have it working, demonstrate to the TA.

REPORT

For your lab report, include the source code for your game and for the testbench you used to verify the behavior. In addition, include answers to the following questions.

- What problems did you encounter while testing your steps yourself?
- Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these testcases escaped your own testing.