**Verilog-HW-3: FSM Design — Adventure Game**

This lab is derived from lab 3, Digital Design and Computer Architecture, Harris and Harris, © Elsevier, 2007

# Introduction

In this lab you will design a **Finite State Machine** (FSM) that implements an adventure game! You will use Verilog to implement your FS, then simulate the game using an HDL simulator, and finally you can play your game using the HDL simulator.

Please read and follow the steps of this lab closely. Don’t forget to read the entire lab and refer to the “What to Turn In” section at the end of this lab before you begin.

You will design your FSM using the systematic design steps listed in Figure 2. Parts of these steps will be given, while others will be entirely up to you.

**1.**      State the problem precisely (i.e. in English).

**2.**      Draw a State Transition Diagram.

**3.**      List all inputs and outputs.

**4.**      Construct a table showing how current state and inputs determine next state and outputs.

**5.**      Decide on a binary encoding for each of the inputs, states, and outputs.

**6.**      Rewrite the table using your binary encoding.

**7.**      Write Boolean logic equations using the information in your table.

**8.** Implement the equations using Verilog HDL.

Figure 2. Systematic FSM Design Steps

1.    Design

The adventure game that you will be designing has seven rooms and one object (a sword). The game begins in the Cave of Cacophony. To win the game, you must first proceed through the Twisty Tunnel and the Rapid River. From there, you will need to find a Vorpal Sword in the Secret Sword Stash. The sword will allow you to pass through the Dragon Den safely into Victory Vault (at which point you have won the game). If you enter the Dragon Den without the Vorpal Sword, you will be devoured by a dangerous dragon and pass into the Grievous Graveyard (where the game ends with you dead).

This game can be implemented using two separate state machines that communicate with each other. One state machine keeps track of which room you are in, while the other keeps track of whether you currently have the sword.



Figure 3. Partially Completed State Transition Diagram for Room FSM

The Room FSM is shown in Figure 3. In this state machine, each state corresponds to a different room. Upon reset (the input “R”) the machine’s state goes to the Cave of Cacophony. The player can move among the different rooms using the inputs N, S, E, or W. When in the Secret Sword Stash, the “SW” output from the Room FSM indicates to the Sword FSM that the player is finding the sword. When in the Dragon Den, signal “V,” asserted by the Sword FSM when the player has the Vorpal Sword, determines whether the next state will be Victory Vault or Grievous Graveyard; the player must not provide any directional inputs. When in Grievous Graveyard, the machine generates the “D” (dead) output, and on Victory Vault the machine asserts the “WIN” output.



Figure 4. State Transition Diagram for Sword FSM

In the Sword FSM (Figure 4), the states are “No Sword” and “Has Sword.” Upon reset (input “R” again), the machine enters the “No Sword” state. Entering the Secret Sword Room causes the player to pick up a sword, so the transition to the “Has Sword” state is made when the “SW” input (an output of the Room FSM that indicates the player is in the Secret Sword Stash) is asserted. Once the “Has Sword” state is reached, the “V” (vorpal sword) output is asserted and the machine stays in that state until reset.

The state of each of these FSM’s is stored using D flip-flops. Since flip-flops have a clock input, this means that there also must be a CLOCK input to each FSM, which determines when the state transitions will occur.

So far, we have given an English description and a State Transition Diagram for each of the two FSM’s. This corresponds to the first and second steps, respectively, in the systematic design process given in Figure 2.

You may have noticed, however, that the diagram in Figure 3 is incomplete. Some of the transition arcs are labeled, while others are left blank. Complete the State Transition Diagram for the Room FSM now by labeling all arcs so that the FSM operates as described.

The next step (step 3) in the design is to enumerate the inputs and outputs for each FSM. Figure 5 shows the inputs (on the left) and outputs (on the right) of the Room FSM and Figure 6 does this for the Sword FSM. Note that for navigational purposes the Room FSM should output S0-S6, indicating which of the seven rooms our hero is in. This is the last step of the design that will be given to you.



Figure 5. Symbol for Room FSM, showing its Inputs and Outputs



Figure 6. Symbol for Sword FSM, showing its Inputs and Outputs

Next, draw a state transition table for each FSM showing how the current state and inputs determine next state. The left side of the tables should have a column for the current state, and separate columns for each of the inputs. The right side should have a column for the next state. Also draw outputs tables, with the current state on the left, and the output(s) on the right. These tables are a way of representing the FSM’s that is an alternative to the diagrams in Figure 3 and Figure 4.

On the left side of the table for the Room FSM, you do not need to fill in every possible combination of values for all inputs (that would make for a rather large number of rows in your table!). Instead, for each state you only need to show the combinations of inputs for which there is an arc leaving that state in the state transition diagram. For example, when the input N is asserted and the current state is Twisty Tunnel, the behavior of the FSM is unspecified and thus does not need to be included in the table.[[1]](#footnote-1) Also, you do not need to show rows in the table for what happens when more than one of the directional inputs is specified at once. You can assume that it is illegal for more than one of the *N*, *S*, *E*, and *W* inputs to be asserted simultaneously. Therefore, you can simplify your logic by making all the other directional inputs of a row “don’t care” when one legal direction is asserted. By making careful use of “don’t cares,” your table need not contain more than a dozen rows.

The next step in FSM design is to determine how to encode the states. By this, we mean that each state needs to be assigned a unique combination of zeros and ones. Common choices include binary numeric encoding, one-hot encoding, or Gray encoding. A one-hot encoding is recommended for the Room FSM (i.e. Cave of Cacophony=0000001) and makes it trivial to output your current state S0…S6, but you are free to choose whichever encoding you think is best. Make a separate list of your state encodings for each FSM.

Now rewrite the table using the encoding that you chose. The only difference will be that the states will be listed as binary numbers instead of by name.

You are now approaching the heart of FSM design. Using your tables, you should be able to write down a separate Boolean logic equation for each output and for each bit of the next state (do this separately for each FSM). In your equations, you can represent the different bits of the state encoding using subscripts: S1, S2, etc. Depending on which state encoding you chose, a different number of bits will be required to represent the state of the FSM, and thus you will have a different number of equations. Simplify your equations where possible.

As you know, you can translate these equations into logic gates to implement your FSMs directly in hardware. That is what you will do in the next section.

**2.     write Verilog code for your FSM**

For this lab, open the modelsim（or any other HDL simulator） with a new project named “lab03\_xx” (where xx are your student ID number).

You need to implement two FSMs: the Sword or Room FSM, and connect them to each other. You can implement each FSM as a standalone module, then create a top level module with instances of the two FSM modules, which are connected to each other; or you can implement both FSMs and the connection in just one module.

The inputs and outputs of your top level module will determine which signals will be available in the simulator when you play the game, so you should make sure to include at least CLOCK, R, N, S, E, and W, as inputs and the current room S0-S6 as an output. Add output ports to monitor the values of V and SW during the simulation. (Remember, if you are adding multiple output ports to a wire, you will need to add a buffer between each additional port.) This is very similar to use a probe to debug circuit hardware. Check and correct any errors of the schematic.

3.     Simulation

Once you have completed the previous step, you can create a testbench, and simulate the design using ModelSim as you have done before. You should name your test bench as “lab03\_xx\_test.v”.

# What to Turn In

1. **Please indicate how many hours you spent on this lab.**  This will not affect your grade, but is critical for calibrating the workload for next semester’s labs.

2.      A completed State Transition Diagram for the “Room” FSM.

3.      Your tables listing (1) next state in terms of current state and inputs and (2) output in terms of current state and inputs. You need tables for each FSM.

4.      A list (one for each FSM) of your binary encoding for each state.

5.      The revised copy of your tables, using your binary encoding.

6.      Your Boolean logic equations for the outputs and each bit of the next state in terms of the previous state and inputs.

7.     Two printouts of your simulation waveforms: one that shows you playing the game and winning (entering “Victory Vault”), and another that shows an example of losing the game (entering the “Grievous Graveyard”). **Your signals must be printed in the following order: *CLK*, *R*, *N*, *S*, *E*, *W*, *S*6:0, *SW*, *D*, *V*, *WIN*.** (The notes on the next page describe how to format a testbench waveform output file if you’ve forgotten how to do so.) Please select “Landscape” under Print Setup before printing these so that they fit better on the page.

8. the Verilog code file of your design and testbench.

9.  EXTRA CREDIT: It is a little known fact that the Twisty Tunnel is located beneath the dining commons and that by heading north one can reach the dormitories. Extend your adventure game with more interesting rooms or objects. There will be a prize for the most interesting working enhancement!

1. 1 Since the behavior of the FSM is unspecified in cases like this, the actual behavior of the FSM that you build in these cases is up to you. In a real system, it would be wise to do something reasonable when the user gives illegal inputs. In this game, we don’t care what your game does when given bad inputs. [↑](#footnote-ref-1)