

Digital Logic Design – ECEN 3233

Oklahoma State University

James E. Stine, Jr.

Lab 3: Adventure Game

Introduction

In this lab may you will design a **Finite State Machine** (FSM) that implements an adventure game! You will then enter the FSM into the Schematic Editor in Xilinx ISE Project Navigator, then simulate the game using ModelSim, and finally you can play your game using ModelSim.

Please read and follow the steps of this lab closely. Start early and ask questions if parts are confusing. It is much easier to get your design right the first time around than to make a mistake and spend large amounts of time hunting down the bug. As always, 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. There is also a set of hints of common tool mistakes on the last page of the lab.



Figure 1: Adventure Icon

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. Simplify and implement the equations using digital logic gates.

Figure 2. Systematic FSM Design Step

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 Vorpall 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 Vorpall 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.

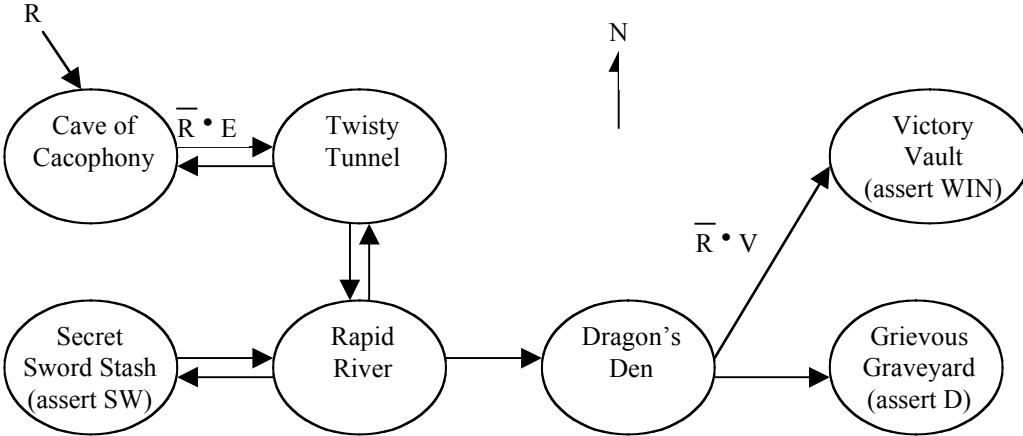


Figure 3. Partially Completed State Transition Diagram for Room FSM

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 Vorpall 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.

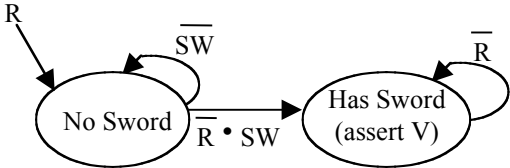


Figure 4. State Transition Diagram for Sword FSM

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” (vorpall 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.

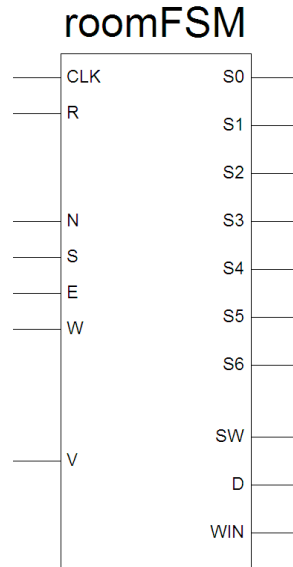


Figure 5. Symbol for Room FSM, showing its Inputs and Outputs

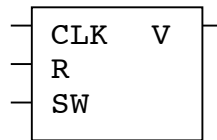


Figure 6. Symbol for Sword FSM, showing its Inputs and Outputs

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 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.

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.

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 $S_0 \dots S_6$, 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: S_1, S_2 , 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. Schematics

For this lab, open the Xilinx Project Navigator with a new project named “lab03_xx” (where xx are your initials).

By now, you are familiar with the Schematic Editor. In this lab, however, you will learn how to create **hierarchical** schematic designs. In the same way that you can add symbols such as AND and OR gates to your schematic, you can add sub-components that are themselves specified by schematics. This creates a hierarchy of schematics.

You will use a hierarchical design for your adventure game by doing the following:

1. First draw the Sword or Room FSM as a schematic. Check and correct the schematic for errors before going to the next step.
2. Once the schematic is finished, you will see it listed as a source of the project. Select the schematic file, and then in the processes window click the “+” to expand the Design Utilities. Double click “Create Schematic Symbol”. When the Xilinx Transcript console says “completed successfully”, a symbol with the same name as your schematic file is created. Now the symbol is available for use in other schematics. Create symbols for both the Sword and Room FSM schematics.
3. Once you have created the symbols for both FSMs, you will be able to access these symbols from the symbol list in the schematic editor as shown in Figure 7. You will see the project directory listed in the symbol library, where you can see the symbols you created. By default, the inputs and outputs of the symbol may appear somewhat randomly placed, which is not very convenient when drawing wires in your higher level schematic. If you wish to edit the pin placement of the symbol, first place the symbol in a schematic. Then right-click on the symbol, and choose **Symbol**→**Edit Symbol** in the pop-up menu or choose **Edit**→**Symbol** from the file menu. When editing the symbol, be sure to move around both the connections and names so that your symbols look like those in Figure 5

and Figure 6. **IMPORTANT:** you **must** move the name and the corresponding connection (the wire and the little box) together during the process of editing the symbol. When finished, make sure to save the symbol.

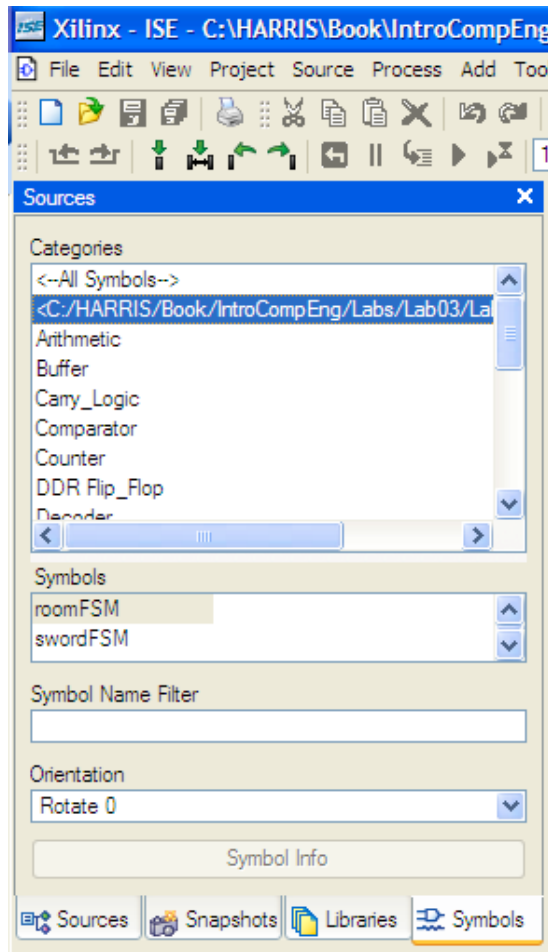


Figure 7. User created symbols in the symbols list

4. Finally, draw a third schematic to connect the FSM's to each other to form the completed adventure game. The inputs and outputs of your top level schematic 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.

If you realize that you need to edit the schematic for a symbol that you have already created, you can still click the symbol and do the editing as described above. However, you must save all the changes and the higher level schematic (where you placed your created symbol) will need to be updated as well. **Important:** If you change anything in the lower level schematic, you have to save the new schematics and re-run the "Create schematic Symbol" procedure. Then open the higher-level schematic (where you placed your created symbol). Xilinx will prompt you to

update the schematic symbol with a window that says “Open Schematic File Errors”. Highlight all of the symbols in the “Obsolete Symbols” and click on “Update” and OK.

Keep this in mind – it is very important especially if you have many hierarchies in the schematic files. If a schematic file is updated and saved, its symbol must be re-created and anything at a higher level (any schematic that contains the symbol) needs to be updated and saved. If there are even higher levels in your schematic, those symbols need to be re-created as well, and the relevant schematics need to be updated.

Here are some more guidelines for drawing the schematics for each of your FSM’s (do them one at a time):

- In the Schematic Editor, double click an empty spot on the sheet, and in the schematic property, you can change the Sheet Sizes. Choose D-size landscape orientation schematic page so that you have enough room. Don’t place your gates too close together or you’ll have difficulty finding room for the wires.
- First draw the flip-flops that will store the state. The symbol name for a D flip-flop is “FD” under category of “Flip_Flop”. If you used a one-hot encoding, there will be one flip-flop per state. (With other encodings there may be a different number of flip-flops.)
- Then add and connect logic gates that implement the Boolean equations (nextstate equations) from your design. Keep in mind that the “current state” corresponds to the values at the output of the flop-flops, while the “next state” corresponds to the values at the inputs of the flip-flops (generated by your combinational logic).
- Add logic gates that implement your output logic. The inputs to this logic block are only the current state bits.
- Finally add the input and output ports and name them properly.

After reading this part of the lab thoroughly, you should be ready to complete all three schematics needed for your adventure game (sword FSM, room FSM, high level connections). The only thing left to do is to test the game in the simulator and play it!

3. Simulation

Once you have completed all your schematics for the adventure game, you can create a testbench waveform using HDL Bencher, and simulate the design using ModelSim as you have done before. To create a testbench waveform, select **Project**→**New Source**, and in the dialog window, select the “Test Bench Waveform” and type the name “lab03_xx_test”. Next choose your top level schematic as the associated source. Set the clock high for 50 ns and low for 50 ns so that the clock period is 100 ns. Input setup time is 5 ns and Output valid delay is 5 ns. Offset is 0 ns.

Timing constraints for testbench waveforms can be changed in the HDL Bencher window by selecting **Test Bench** → **Rescale Timing** from the file menu. You can also change the simulation end time by double-clicking the upper left-hand text called “End Time” and entering a value.

Now all the inputs and outputs of your game are shown in the bencher. Note that you can click and drag the signal names in the window to rearrange the order that they appear. **Set the reset signal, R, high for two cycles (200 ns) before you begin playing.** Be sure to watch V and SW in your simulation to help debug when things don’t work correctly. If these signals do not show up in the list, you should go back and add output ports in your top-level (third) schematic. Enter the input waveform for a winning case, and make sure that you have tested every valid transition between states in the diagrams from Figure 3 and Figure 4. Now you are ready to run the simulation as before. The “Simulation Behavioral Verilog Model” should be able to show the output waveform. If the logic doesn’t seem to be right, fix any bugs that you can find in your schematic. If you find an error and need to change your schematics, quit the simulator but save your input waveforms. Once new editing is done with your schematic, you can restart the simulator for your waveform file to reload the inputs you had chosen.

If it is successful, enter the input waveform for a losing case, and check the results. Generate printouts for both cases of your simulation waveforms to turn in.

You can also play your game by setting the input state one at a time. This is very useful to check whether or not you are going to the correct state with different inputs. In order to do so, you can create a dummy testbench file without specifying any of the input states. You can launch the “Simulation behavioral Verilog Model” similarly. Now in the command window of ModelSim (i.e. the window labeled “Transcript”), type in “force R 1” to set the initial state of the reset (*R*) input to be “1” (Note that Xilinx is case sensitive.). Then enter, and type in “run 100ns” (if your timing constraints keep a clock period of 100 ns). This would allow *R* to be high for the first clock cycle. In the wave window, you can see *S0* becomes “1” which means you are in the state *S0* or in the Cave of Cacophony. You can do the same thing for the following clock cycles. For example if next you want to go to the Twisty Tunnel. You need to go “East” by setting *E* to be 1, and at the same time make sure reset *R* is 0. So type in “force E 1”, “force R 0”, “run 100ns” in the command lines. You should be able to see you are entering Twisty Tunnel by checking *S1* to be 1. Play the game for both win case and lose case. If the result is not what you expected, go back to the schematics, and review your design and schematics until you are comfortable you have a working game.


What to Turn In

Please provide a hard copy of each of the following (in the correct order, and each part clearly labelled).

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. A printout of your schematics for both the "Room" and "Sword" FSM's.
8. A printout of your schematic for the game (built by connecting both FSM's).
9. 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.
10. 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!
11. EXTRA CREDIT: Implement your adventure game and have the state output onto the seven segment display from Lab 2.

When Everything Else Doesn't Work...

If you've been pounding your head from some time and your design still doesn't work, here are some hints of common and subtle problems encountered by past students:

- If there's a red dot at the end of a wire, the wire is not connected to anything. Sometimes it may look like the wire is attached to the input of a logic gate, but the dark dot is the giveaway that there is no connection. Delete the wire and try redrawing it. Often this bug and the next one will manifest themselves as gray boxes somewhere in your simulation indicating floating outputs.:
- Remember, if you want to delete just part of a wire (not all of its connections – i.e. the “branch”), click on “Select the line segment” under the “When you click on a branch” section in the Select Options window on the left-hand side of your schematic window.
- If you place two gates nearby so that the output of one touches the input of another, make sure the gates connect. You can drag gates around to see if they are really connected.
- When you see warning messages in the Project Navigator window, pay heed to them, especially when your circuit isn't working. Understand what warnings are normal and what ones indicate a problem.
- If you make a change in your schematic, sometimes the simulator will not know about it. The best thing to do is quit the simulator and restart it.
- If everything seems right and the tools are still acting up, try quitting and restarting Project Navigator. If you constantly see the same error message when you check the errors of your schematic, try to exit the editor and re-enter and see if the error message is gone. Sometimes, the editor is not properly updated about the corrections you have done.
- In creating the symbol out of your schematic, if you want to rearrange the pin layout of the symbol, make sure always move the name with the end connection together so you don't scramble them logically. (**THIS IS VERY IMPORTANT**)
- Be careful, if you draw a wire across a pin, Xilinx may connect to the pin without your intending to.
- Before doing simulation, always check your schematic files. Correct all the errors.
- You can save the format of your testbench waveform outputs (i.e. the order and format of your inputs and outputs in Modelsim) by doing the following. After opening your testbench waveform in Modelsim and ordering the inputs and outputs as you would like, choose **File->Save**, or by clicking on the Save icon . Save the format file with a “.do” suffix. Make sure the “wave-default” pane is selected.

Next time you make changes to your schematic or waveform, you can open the saved format in the waveform viewer in Modelsim. First, make sure the “wave-default” pane is selected. Then (1) choose Edit->Select All and Edit->Delete to delete the current signals and (2) choose File->Load and open the file you previously saved “.do” file.

- If you make a change in a lower-level schematic, remember that you must create the symbol again for that schematic and then update it in the higher-level schematic. Once you have successfully created a schematic symbol for a schematic, you will see a green check mark by the Create Schematic Symbols option in the Processes window when you highlight the given schematic in the Sources window.
- “People who read through the whole lab do better than those who don’t.” – former Computer Engineering lab assistant.