# Lab 4: Adventure Game

## Introduction

If you owned a personal computer in the early 1980s, you probably have a nostalgic fondness for text adventure games. If not, this lab may cause you to acquire one, because 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 4 Project Navigator, and finally you will be able to use the ModelSim to play the game.

Please read and follow the steps of this lab closely. 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. There is also a set of hints of common tool mistakes on the last page of the lab.

## Background

Before computers were capable of displaying graphics, text-based adventure games were popular. Each game consists of several rooms, each with a short description. The player uses directional commands to move between rooms (e.g., "E" to go east). Objects can be found in certain rooms and are manipulated or interacted with using a very limited set of simple commands. Here is a short example:

```
Dead end
You are at a dead end of a dirt road. The road goes to the east. In the distance you
can see that it will eventually fork off. The trees here are very tall royal palms,
and they are spaced equidistant from each other.
There is a shovel here.
> get shovel
Taken.
> E
E/W Dirt road
You are on the continuation of a dirt road. There are more trees on both sides of
you. The road continues to the east and west.
There is a large boulder here.
> E
Fork
You are at a fork of two passages, one to the northeast, and one to the southeast.
The ground here seems very soft. You can also go back west.
> W
E/W Dirt road
There is a large boulder here.
> look trees
They are palm trees with a bountiful supply of coconuts in them.
> shake trees
You begin to shake a tree, and notice a coconut begin to fall from the air. As you
try to get your hand up to block it, you feel the impact as it lands on your head.
You are dead.
```

In this lab you will be implementing an adventure game using an FSM. You should be familiar with finite state machines from class. Also, you should take a look at Section 6 in Appendix B of *Computer Organization and Design* for more information about FSMs.

You will design your FSM using the systematic design steps listed below. 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.

## 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.

The Room FSM is shown in Figure 1. 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" and "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.

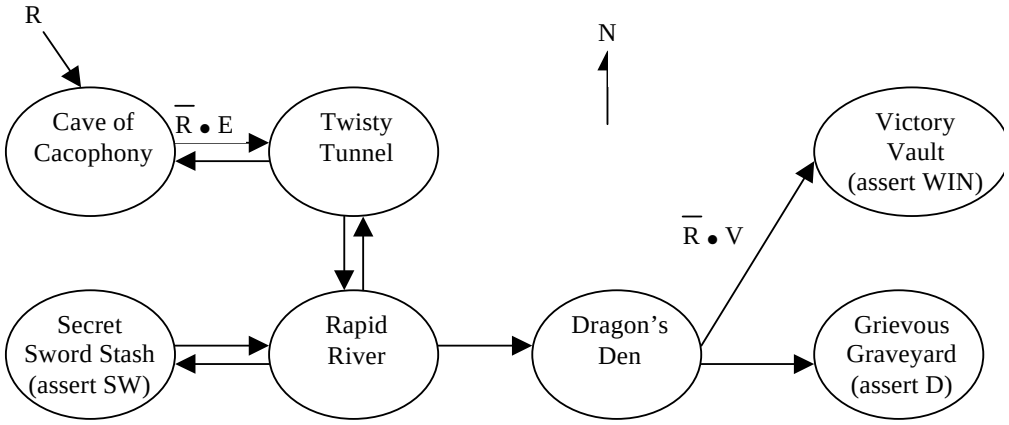


FIGURE 1 Partially Completed State Transition Diagram for Room FSM.

In the Sword FSM (Figure 2), 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 FSMs is stored using D-type flip-flops. Since flip-flops have a clock input, there must be a CLOCK input to each FSM that determines when the state transitions will occur.

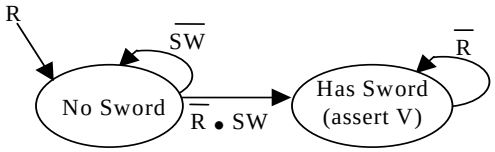


FIGURE 2 State Transition Diagram for Sword FSM.

So far, we have given an English description and a State Transition Diagram for each of the two FSMs. This corresponds to the first and second steps, respectively, in the systematic design process. You may have noticed, however, that the diagram in Figure 1 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 3 shows the inputs (on the left) and outputs (on the right) of the Room FSM, and Figure 4 does this for the Sword FSM. Note that, for navigational purposes, the Room FSM should output *S*1–*S*7, indicating which of the seven rooms our hero is in. This is the last step of the design that will be given to you.

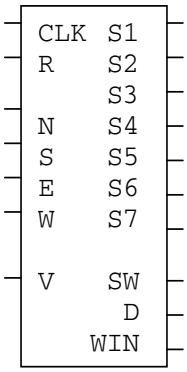


FIGURE 3 Symbol for Room FSM, showing its Inputs and Outputs.

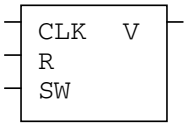


FIGURE 4 Symbol for Sword FSM, showing its Inputs and Outputs.

Next, draw a table for each FSM showing how the current state and inputs determine the next state and outputs. 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 and separate columns for each of the outputs. These tables are a way of representing the FSMs that is an alternative to the diagrams in Figure 1 and Figure 2.

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. 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 if the machine catches on fire when given bad inputs. 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 (e.g., Cave of Cacophony=`0000001`) and makes it trivial to output your current state, *S*1–*S*7, 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 the 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.

## Schematics

For this lab, open the Xilinx Project Navigator with a new project named "lab4_*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 each of the Sword and Room FSMs as a schematic. Check and correct for schematic errors before going to the next step.

2. Once the schematics are finished, you will see them listed as sources of the project. Select one of the schematic files, and then in the processes window click the "+" to expand the Design Entry Utilities. Double click Create Schematic Symbol. When a green check mark is shown, 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 Sword FSM and Room FSM schematics.

3. Once you 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 5. 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, you can choose Edit→Symbol, or click the right button of the mouse and choose Symbol and Edit. Move both the connections and names so that your symbols look like those in Figure 3 and Figure 4. *Important: you must move the name and the corresponding connection together during the process of editing the symbol.* When finish, make sure to save the symbol.

4. Finally, draw a third schematic to connect the FSMs to each other to form the complete 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, S1–S7, as an output. Label the V and SW wires by clicking the Name the Net icon in the toolbar and add output ports if you wish to monitor the values during the simulation. This is very similar to using 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 will need to be updated as well. If you changed anything in the lower level schematic, you have to save the new schematics and re-run the Create schematic Symbol procedure so that symbols can be updated accordingly. Keeping this in mind is very important, especially if you have many hierarchies in the schematic files.

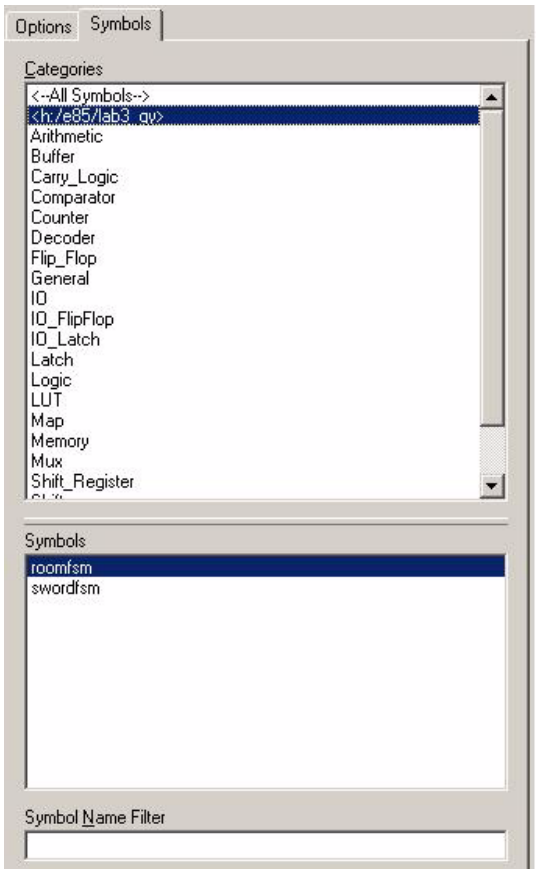


FIGURE 5 User-created symbol in the list.

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

1. In the Schematic Editor, double click on an empty part of the sheet, and in the schematic property you can change the Sheet Sizes. Choose a 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.
2. First draw the flip-flops that will store the state. The symbol name for a D flip-flop is "FD" under the category "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.)
3. Then add and connect logic gates that implement the Boolean 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).
4. 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 and high level connections). The only thing left to do is to test the game in the simulator!

## 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. In the New dialog window, select the Test Bench Waveform and type the name "testwin". Set the clock high for 5ns and low for 5ns. The input setup time is 2ns, and the output valid delay is 2ns. These timing constraint options can also be changed in the HDL Bencher window by clicking the Timing

Constraints button. Next, choose the top level schematic as the associated source. 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 in which they appear. Be sure to watch V and SW in your simulation to help you debug when things don't work correctly. If these signals do not show up in the list, you should go back to label the wires 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 1 and Figure 2.

Now you are ready to run the simulation, as before. Using Simulation Behavioral Verilog Model, you should be able to see 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 have chosen.

If simulation is successful, you can enter the input waveform for a losing case,and check the results. Generate print-outs for both cases of your simulation waveforms to turn in.

You can also play your game by setting the input states 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 Simulation behavioral Verilog Model similarly. Now, in the command window of the ModelSim, type in "force R 1" and Enter to set the initial state of the R input to be 1. Then type in "run 10ns" (if your timing constraints keep the clock cycle of 10ns). This will allow R to be high for the first clock cycle. In the wave window, you can see S1 becomes 1, which means you are in the state S1, 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 making sure reset R is 0. So type in "force E 1", "force R 0" and "run 10ns" in the command line. You should be able to see that you are entering Twisty Tunnel by checking that S2 is 1. Play the game for both a winning case and a losing 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.

## Extra Credit

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

## 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 brown 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.
- If you place two gates nearby so that the output of one touches the input of another, the gates will not be connected even though they look connected. 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 which warnings are normal and which indicate problems.
- If everything seems right and the tools are still acting up, try quitting and restarting the 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 with corrections you have done.
- If you make a change in your schematic, the simulator will not know about it. The best thing to do is quit the simulator and restart it. You can save your waveforms if you are sick of constantly adding them again.
- In creating the symbol from your schematic, if you want to rearrange the pin layout of the symbol, always

move the name and the end connection together so that you don't scramble them (*this is very important*).

- Before doing simulation, always check your schematic files. Correct all the errors.
- "People who read through the whole lab do better than those who don't."
  — former Computer Engineering lab assistant.