# Digital Electronics & Computer Engineering (E85)

## Lab 3: Adventure Game

## Introduction

If you owned a personal computer in the early 1980's, 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 Xilinx Schematic Editor, and finally you will be able to use the Simulator to play the game.

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 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.

## 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 (i.e. "E" to go east). Objects can be found is certain rooms, and are manipulated or interacted with using a very limited set of simple commands. Figure 1 gives 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.
```

**Figure 1: Sample from the "Dunnet" Adventure Game**[1]

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 your book for more information about FSM's.

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 Steps**

[1] Dunnet is built into the Emacs text editor (M-x dunnet).

## 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.

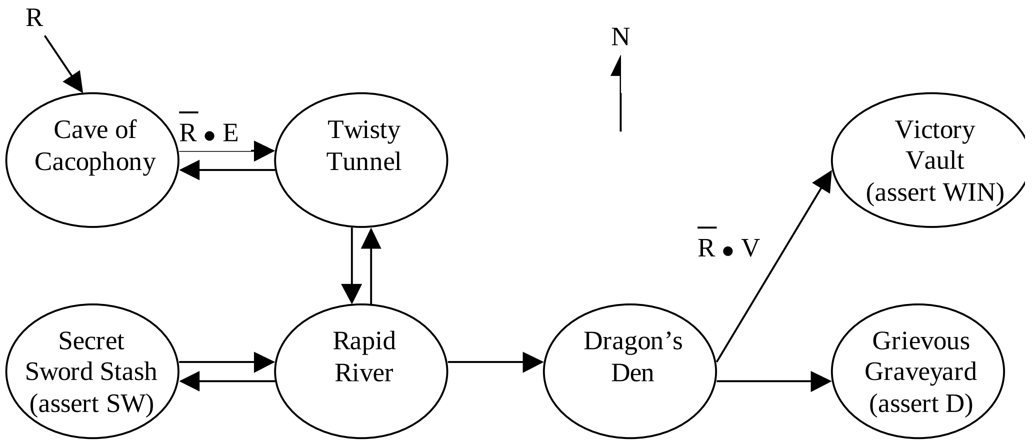


**Figure 3: Partially Completed State Transition Diagram for Room FSM**

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 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.

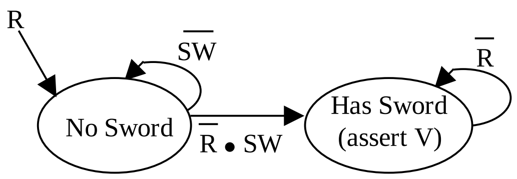


**Figure 4: State Transition Diagram for Sword FSM**

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-type flip-flops. Since flip-flops have a clock input, this means that there is 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 S1-S7, indicating which of the seven rooms our hero is in. This is the last step of the design that will be given to you.

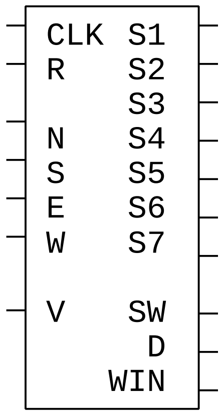


**Figure 5: Symbol for Room FSM, showing its Inputs and Outputs**

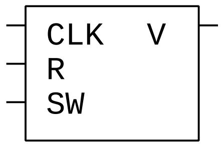


**Figure 6: Symbol for Sword FSM, showing its Inputs and Outputs**

Next, draw a table for each FSM showing how the current state and inputs determine 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 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.[2] 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 the 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 S1…S7, 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 different 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 FSM's directly in hardware. That is what you will do in the next section.

[2] 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 if the machine catches on fire when given bad inputs.

## 2. Schematics

For this lab, open the Xilinx Project Manager with a new project named "lab3_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.
2. With the completed schematic opened in the Schematic Editor, create a symbol from it by choosing 'Hierarchy•Create Macro Symbol from Current Sheet' from the menu. When you create symbols in this way, they are entered into a special symbol library that is associated with the current project, and are then available for use in other schematics.
3. Once you have drawn schematics and created the symbols for both FSM's, you can edit the symbols themselves so that their pin placement makes sense (as in Figure 5 and Figure 6). This step described in detail below.
4. Finally, draw a third schematic to connect the FSM's to each other to form the completed adventure game. You access the symbols that you created for the FSM's in the same way that you would draw the symbol for a logic gate (you symbol names should appear when you open the Symbols Window in the Schematic Editor). 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 as an output. Label the V and SW wires by double-clicking on them and typing in the name.

If you realize that you need to edit the schematic for a symbol that you have already created, you can choose File•Open Macro in the Schematic Editor to make the appropriate changes. However, if you change a symbol after using it in a higher level schematic, you will need to update the higher level schematic by opening it in the Schematic Editor and choosing File•Update Libraries.

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

- Work with a B-sized landscape orientation schematic page so you have enough room. Don't place your gates too close together or you'll have difficulty finding room for the wires.
- First add the input and output terminals
- Next, draw the flip-flops that will store the state. The symbol name for a D flip-flop is "FD." 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.)

- Finally, add and connect the 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).

When drawing the schematics for your FSM's you may end up with two different outputs that are always equal, and indeed are connected to the same place. However, the Schematic Editor will not let you connect two output terminals with different names to the same wire (it insists that everything connected to the same wire have the same name, for fear that otherwise you are accidentally shorting two different signals together). Instead, you can connect one of the output terminals to the wire, and connect the other output terminal to the wire through a buffer, as shown in Figure 7.

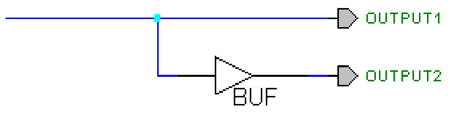
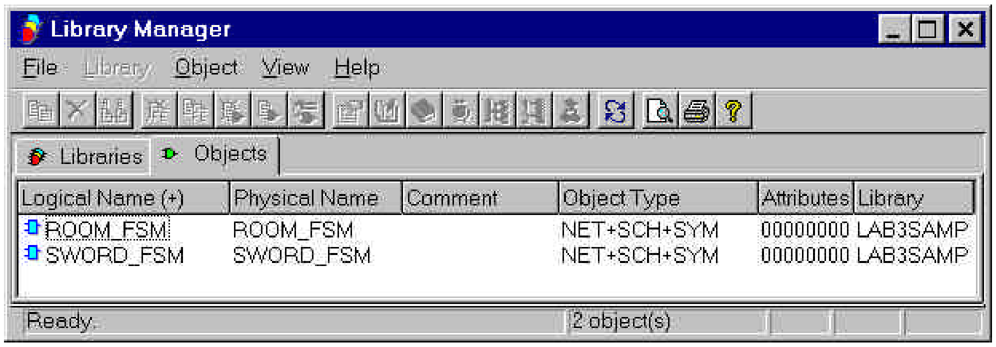


**Figure 7: Generating two differently-named outputs from the same source requires a buffer.**

As mentioned above (in step 3), you will need to edit the pin placements of your symbols to get them to look like the ones in Figure 5 and Figure 6. By default, the inputs and outputs may appear somewhat randomly placed, which is not very convenient when drawing wires in your higher level schematic that uses the symbols.

To edit a symbol itself, you first need to open the Library Manager. You can do so by double clicking on the project symbol library (which has the same name as your project – something like "lab3_xx") in the upper left panel of the Project Manager. This should open the Library Manager with your project's symbol library. You should see the names of the two symbols that you created for your FSM's, as in Figure 8.



| Logical Name (+) | Physical Name | Comment | Object Type | Attributes | Library |
|---|---|---|---|---|---|
| ROOM_FSM | ROOM_FSM | | NET+SCH+SYM | 00000000 | LAB3SAMP |
| SWORD_FSM | SWORD_FSM | | NET+SCH+SYM | 00000000 | LAB3SAMP |

Ready. 2 object(s)

**Figure 8: Library Manager Window**

By double-clicking again on the name of a symbol, you can edit it in the Symbol Editor. You will want to do this for every symbol that you create. All you need to do is click and drag the input and output pins to move them where you want them. Once you have the layout of the symbol the way you like it, save it and close the Symbol Editor. After editing all your symbols, you will be ready to use them in your high level schematic.

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 play the game in the simulator!

## 3. Simulation

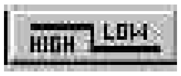
Once you have completed all your schematics for the adventure game, open the Simulator (from the Project Manager, as you have done before). Choose Signal•Add Signals from the Simulator menu and place check marks on all the inputs and outputs of your game (again, just like you have done before). Note that you click and drag the signal names in the simulator window to rearrange the order that they appear. Be sure to watch V and SW in your simulation. If they do not show up in the list while adding signals, you probably forgot to label the wires in your top-level (third) schematic.

In this lab, your simulation can make use of more complex stimulus than we have used before to specify the values of input signals. You will learn how to bind an input signal to a key on the keyboard, so that pressing the key in the simulator window will cause the value of the corresponding input signal to toggle. This will come in handy for the directional inputs when you are playing the game.

To bind a keyboard key to an input signal, choose Signal•Add Stimulators from the menu and click and drag the key onto the signal name. You should see the letter of the key in red text to the right of the signal name now. Do this for the N, S, E, W, and R inputs, so that you can use the keyboard to reset and play the game.

While you have the Stimulator Selection window open, drag the least significant bit of the counter onto the clock input. This will supply your FSM's with a clock cycle of 10ns, so you will also want to set the simulation step time to 10ns, so that you can make one move at a time while playing the game.

Note that if you want to specify a single constant value for an input signal, you can do so by pressing the "Logical States" button.



This will open the window shown in Figure 9, where you can click and drag a value such as "low" or "high" onto a signal name.

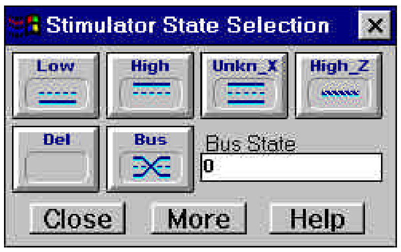


**Figure 9: Stimulator State Selection Window**

You are now ready to play your adventure game! Press your "R" key until the R input is high, and then proceed forward one simulation step. You should notice the state of your room FSM entering the state corresponding to the Cave of Cacophony. Press R, N, S, E, and W until all inputs are low. Now press E and proceed forward one simulation step to go east into the Twisty Tunnel. Continue playing the game, and remember that at every step, only one of the directional inputs should be high. Otherwise, the behavior of your game is unspecified, as it is when you attempt to go in a direction that is not listed in the transition table in Figure 3.

Make sure that your game always enters the correct state while you are playing. For example, check that the V output of the Sword FSM is high after entering the Secret Sword Stash. Also, make sure that you have tested every valid transition between states in the diagrams from Figure 3 and Figure 4. Fix any bugs that you find in your game.

If you find an error and need to change your schematics, you can save your input waveforms by using the File•Save Waveform command in the simulator. Quit the simulator, edit your schematic, restart the simulator, and use File•Load Waveform to reload the inputs and stimulators you had chosen. Use Device•Power On/Reset if you ever need to restart the simulation time at 0.

When you are confident that your game is working, play it twice (once winning, once losing) to generate printouts of your simulation waveforms to turn in. Note that you can clear the waveforms (i.e. when you want to restart the game) by selecting Waveform•Delete•All Waveforms from the menu.

## What to Turn In

Please provide a hard copy of each of the following:

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. A completed State Transition Diagram for the "Room" FSM.
3. Your tables listing next state and outputs in terms of current state and inputs (one 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"). 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 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:

1. If there's a dark blue 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 blue 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.
2. 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.
3. When you see warning messages in the Project Manager window, pay heed to them, especially when your circuit isn't working. Understand what warnings are normal and what ones indicate a problem.
4. Don't label nodes with the same name as an input or output. The tools will short the wire to the input or output and get horribly confused.
5. 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.
6. If everything seems right and the tools are still acting up, try quitting and restarting Foundation.
7. "People who read through the whole lab do better than those who don't." – former Computer Engineering lab assistant.