ECE 3544: Digital Design I

Project 1 (Part A): Introduction to ModelSim Environment

**Objectives**

After completing this project, you should be more familiar with the ModelSim tools. You will gain experience with timing models and with the use of test benches for simulating digital circuits in Verilog.

**Requirements**

You must have the current version of ModelSim-Altera Starter Edition installed on your computer. The instructions in this assignment are consistent with that version of ModelSim. Even if these instructions are consistent with other versions of ModelSim, you should use the version indicated here.

*The DE1-SoC Board is not required for this project. This project involves only simulation.*

**Project Description**

Intel FPGA’s ModelSim is one of the most common simulation tools for digital logic design. In this project, you will learn how to use ModelSim to run simulations. First, you will simulate two pre-designed circuits:

* The 74138 3-to-8 decoder, which was written using a *structural* model using *primitive* gates, and
* A 4-to-16 decoder that uses two of the 3-to-8 decoders as *structural* blocks, along with an inverter as an additional *primitive* gate.

You will use a test bench to simulate each decoder and produce output waveforms.

After you gain proficiency with using ModelSim to simulate existing designs, you will perform the design of a simple digital logic circuit that you can model and simulate using the ModelSim tools.

**What is a test bench?**

The first thing you should understand is the difference between design files and test bench files.

Imagine you have real hardware that you want to verify. You might use probes to supply inputs to your hardware and to observe its outputs. In this way, you could verify that the hardware is behaving as expected. In the case of simulation, you have the Verilog description of the hardware instead of the hardware itself, but this need not stop you from simulating the verification process.

The module that the designer uses for applying test inputs is called a *test bench*. The test bench normally consists of an instance of your Verilog model (the “device under test”) and a procedure containing a sequence of inputs with specified timing. You can use simulation tools such as ModelSim to run the test bench and observe the output. For more complex models, the test bench can both apply inputs and check the outputs. We will use waveforms to see the generated output.

It is important to remember that test bench files *cannot* be synthesized to generate real hardware; they are only used for simulation.

**Instructions**

*Part 1: Simulate the decoder modules.*

1. Follow the instructions in Step 1 of the document “Creating and Simulating Verilog Source Files in ModelSim” to make a project for the starter files in the Part 1 Folder. (You’ll find the aforementioned document and other useful documents on the Pages page of the Canvas site.)

*Make sure that you follow the recommendations in that document for locating and naming the folder where you place your starter files – namely, create a working folder having a short, space-free name in a folder that is close to the root directory, and not in one having a long pathname that contains spaces*.

1. Study the example files to make sure that you understand how each model works in Verilog. First, look at sn74138.v. This is a Verilog structural model that describes a 74138 chip (a 3-to-8 decoder) using primitive logic gates. **Based on the structure of the 74138 decoder module, produce a neat gate-level schematic of this module and include it in your report.** Make sure that your schematic is well-labeled.

Next, look at decoder4to16.v. This is the Verilog model for a 4-to-16 decoder made from two 3-to-8 decoders and an inverter. Take note of the manner in which this module instantiates two of the sn74138 modules. **Based on the structure of the 4-to-16 decoder module, produce a neat block diagram of this module and include it in your report.** Make sure that your block diagram is well-labeled.

Instead of drawing another gate-level logic circuit, use a block symbol to represent the 74138. Here is an example of what a 74138 block might look like:



Now open the test-bench files for each decoder and study the details to understand how they were written. Compare the two test benches and address these questions in your report: **What qualities do the test benches have in common? (Focus on the fact that a test bench is a Verilog procedure.) In what significant ways do the test benches differ?**

1. Follow the instructions in Step 2 of the document “Creating and Simulating Verilog Source Files in ModelSim” to compile the files in the project. Follow the instructions in Step 3 of the document to simulate the 74138 chip. Remember to perform the simulation on the test bench, and not on the source file. Repeat the instructions in Step 3 of the document to simulate the behavior of the 4-to-16 decoder.

**Include screenshots of your results showing proper operation of the 3-to-8 decoder and 4-to-16 decoder in your final report.** If you have the ability to print to PDF, the waveforms can be printed to a PDF file in black-and-white. (**File** > **Print**, then choose the appropriate options.)

*Part 2: Create your own models and test bench*

Design and implement a digital logic circuit that acts as a “judge” for the game “Rock, Paper, Scissors, Lizard, Spock,” as created by Sam Kass and Karen Bryla and “popularized” by “The Big Bang Theory.”**1** Each of two players chooses one of five moves, where a winner is determined based on the directed graph linked below.

(For each comparison of two distinct moves connected by an edge, the arrow points to the winning move. If both players make the same move, the game is tied.)

<https://en.wikipedia.org/wiki/Rock_paper_scissors#/media/File:Rock_Paper_Scissors_Lizard_Spock_en.svg>

The circuit has the following module and port declaration. You may declare and use wires as needed. You **must** create the circuit described below using *only Verilog’s built-in gate primitives.* It should be organized in a general manner similar to the model of the 3-to-8 decoder.

module rpsJudge\_structural(player1, player2, p1wins, p2wins, tied);

input [2:0] player1, player2;

output p1wins, p2wins, tied;

The circuit accepts two 3-bit inputs representing each player’s move and provides three outputs that represent the game’s outcome. Each of a player’s possible moves should correspond to a three-bit value. **Your design should include an assignment of each three-bit value to a player move. Your report should discuss the decision-making process, and the impact that you believe your choice had on your implementation.** **Include a table similar to the one showed below in your report.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Player’s Move | Rock | Paper | Scissors | Lizard | Spock | *Not Used* |
| Input Value(s) |  |  |  |  |  |  |

The value of outputs p1wins and p2wins should equal 1 if the associated player wins. They should equal 0 otherwise. The output tied should equal 1 if the players tie. It should equal 0 otherwise. The outputs are mutually exclusive–only one of the three will equal one for any valid input combination. For example:

* If Player 1 chooses Rock and Player 2 chooses Scissors, then p1wins = 1, p2wins = 0, and tied = 0, since rock crushes scissors.
* If Player 1 chooses Paper and Player 2 chooses Spock, then p1wins = 0, p2wins = 1, and tied = 0, since paper disproves Spock.
* If Player 1 chooses Lizard and Player 2 chooses Lizard, then p1wins = 0, p2wins = 0, and tied = 1, since both players played lizard.

Use the starter files in the Part 2 folder, which include module declarations for your logic circuit and test bench. Name your module and re-name your files according to the directions in the starter files.

Create a new project for this step and add your files to it. You should also create a test bench that instantiates your game module and applies all valid combinations of the inputs to it. You should include waveforms from the test-bench in your report that show the proper operation of your design.

Your modules and test bench should include proper header information, contain reasonable comments, and be neatly formatted. The starter files provide examples of header information; make sure that you fill in the header comments.

**Other Simulation Tips**

* The Wave window allows the user to place and add cursors. A cursor allows the user to view signal values at the simulation time value where the cursor is placed. The user can place multiple cursors and switch between them by clicking on one. When the user places multiple cursors, the time interval between two cursors is visible.
* Drag the Signal Name and Value windows to size them appropriately. Signal names default to appearing as full path names. To compact the signal names, choose **Format > Toggle Leaf Names**. Values default to appearing in binary. To change the radix, right-click the signal name, then choose Radix and choose your desired radix.
* To organize your waveform, re-arrange signals by dragging the signal names up or down in the Signal Name window. Add dividers (**Add > New Divider**) to create sections in your waveform as needed.
* The Zoom options are useful for focusing your waveform as needed.



From left to right, the options are to Zoom In, Zoom Out, Zoom Full (expand the displayed time range to fill the window), Zoom in on Active Cursor, Zoom Between Cursor, Zoom Other Windows.

**Project Submission**

Your submission should include the following files:

* A project report. Your report should be in PDF format. Include your PID in your report filename, *e.g.*, project1Areport\_jthweatt.pdf.

Your report should contain the following elements:

* + The cover sheet included in the project materials. **I have included the cover sheet as a Word document, so that you can place it in your report document before you convert it to PDF**.
  + A restatement of the assignment’s purpose and objectives.
  + A section that addresses the questions posed in the project specification.
  + A section that describes the process by which you derived the design of your circuit and the manner in which you implemented it in Verilog. **In particular, you should comment on how applicable you believe your design process and your method for implementing it using structural Verilog are to performing more general and larger-scale designs and implementations.**
  + A section that shows the simulation of the decoders and the results of your game circuit testing. **Discuss how you formulated the tests contained in the test bench that you had to generate, and how you verified the correctness of your implementation.**
  + A section where you discuss your conclusions and the lessons you learned from the assignment.
* The Verilog files for your game module and test-bench. Name your files according to the convention described in the starter files. **Your code should be documented appropriately.**

Submit your report, your Verilog source file, and Verilog test bench file on Canvas by the deadline.