

**Digital Systems**  
**EE 245L**  
**Experiment 3**  
**Decoders, Multiplexers, and Design with Structural Verilog**

## 1 Objectives

1. Learn how to construct a small binary decoder and multiplexer using basic TTL logic gates,
2. Learn how to construct a binary decoders and multiplexers in our FPGA using basic structural Verilog,
3. Verify that the technique of combining small MSI components to form larger components carries over to our FPGA by using multiple instances of our basic module to build a larger modules.
4. Become able to translate a (possibly ambiguous) word description into a working engineering design,
5. Learn to use “active high” and “active low” logic signals, and use signal names to indicate the active level
6. Successfully implement your own Verilog design using the Altera boards

## 2 Procedure

This lab, and most of the remaining labs, will consist of two basic portions. In one portion of the experiment, you will implement your designs as actual discrete circuits, and test their functionality. You will also implement similar designs using the FPGAs on the Altera DE2 boards. Due to the nature of the two methods and our limited time in lab, slightly larger designs will sometimes be possible on the DE2 boards. We will follow a trend throughout the semester of building up a system out of our basic building blocks. For this lab your basic building blocks are logic gates (namely NAND gates and Inverters), and you will be building decoder and multiplexer circuits. For later labs, those same circuits will be used to build even more complex circuits.

### 2.1 Breadboard Procedure

1. Build the 2-to-4 decoder circuit shown Figure 6.16(c) of your textbook.

2. Verify the Truth Table shown in Figure 6.16(a) (of your textbook).
3. Use an entire 74x139 and whatever other components you need to construct a 3-to-8 decoder. The 74x139 chip contains two 2-to-4 decoders with active low outputs and an active low enable for each decoder. A data sheet on a product similar to that which you will use in lab can be found here:  
`\tthttp://www.labmaster.com/surplus/parts/html/ic200039-mot/sn741s139d.pdf`  
 The circuit will be similar to that shown in Figure 6.17 of your text. Active low outputs are acceptable, and your decoder does not need an enable input, so you should have a simpler circuit than that shown in Figure 6.17.
4. Verify the correct operation of your 3-to-8 decoder. Include a truth table in your report.
5. **Step away from the bench and allow the other partner to complete the remainder of this section.** You will be at a disadvantage during the lab practical(s) if you do not have experience with all facets of the labs.
6. Build the 4-to-1 multiplexer circuit shown Figure 6.2(c) of your textbook.
7. Verify the Truth Table shown in Figure 6.2(b) (of your textbook).

## 2.2 FPGA Procedure

1. Use a text editor (at home, prior to lab) to create a structural Verilog module for a 2-to-4 decoder. You can write your own module from scratch, or you can base your code on the lecture notes from Wednesday, but you should only use the built in Verilog gates listed in Table A.2 on page 785 of your text. You are allowed to make the outputs of the above decoder either active-high or active-low, the choice is up to you. You must provide at least one enable input (again, it is your choice whether this enable is active-high or active-low).
2. You are not required to do so at this time, but I will ask you very soon to download Altera's "Quartus II" web edition software from:  
`\tthttp://www.altera.com/products/software/products/quartus2web/sof-quarwebmain.html`  
 This software includes an editor that highlights Verilog keywords and does other helpful stuff. I am currently using Version 7.1 with Service Pack 1 of this software at home and hope to use that version in lab this semester. If there are not any major problems with this version, please do not move on to a newer version or you may risk becoming incompatible with the software in the lab.

3. Once in lab, your TA will show you the steps that you must take in order to compile, download, and run your Verilog module.
4. Verify the correct operation of your circuit by taking all of the steps necessary to download it to the Altera board and verifying all input combinations.
5. Use a text editor (at home, before lab) to create a Verilog version of the 74x138 (3-to-8 decoder). This is another MSI chip which consists of a single 3-to-8 decoder with active low outputs. This decoder has three enable inputs, two active low and one active high. All three must be asserted for the decoder to assert any outputs. If this doesn't make sense, a data sheet for a similar chip is given here:

`\tthttp://www.fairchildsemi.com/ds/MM/MM74HCT138.pdf`

You may not make this decoder from individual gates, you must make use of the decoder module you implemented above. You therefore **may not** have the same exact gate structure shown in Figure 6.17 Your decoder must *exactly* replicate the functionality (truth table) for the 74x138 decoder. Note that this includes **active-low** outputs and **three** enables (one active high, two active low). You must name your signals to indicate if they are active high or active low (by appending an underscore followed by a capital "L" to the active low signal names.

6. Compile, build, and download your circuit to the FPGA and demonstrate it to your TA.
7. **Optional:**If you have time after completing the design portion of this lab, the TA may ask you to repeat the above starting with a 2-to-1 multiplexer (Figure 6.1(c) of your text) and using two instances of that module to build a 4-to-1 multiplexer (Figure 6.3 of your text). Come to lab prepared to complete this portion also.

## 3 A Simple Design Problem

### 3.1 Pre-Lab Procedure

1. Adhering to the active-level and naming conventions discussed in class, design your circuit. You should have your design broken down into hierarchical modules along the natural lines indicated by the problem. The design problem for this week is extremely simple, but we will break it into several modules in order to demonstrate the concept.
2. Be prepared to present your circuit drawing *prior* to the start of lab.
3. Use structural Verilog to implement your circuit in Verilog.

4. Although this is a very small project, you are required to have a separate Verilog module for each of the main functions (e.g. for each of the four outputs). The **only logic devices allowed in your main module are “not” gates**, all other logic must be performed via separate modules which you will provide. Of course, your other Verilog modules can use all of the structures discussed in class. These other modules should have signal levels that make sense to the designer of the module, so you might need to invert some signals to use these modules. Use nice long identifier (“variable”) names, and **comment your code**. While Verilog purists would insist on a separate file for each module, for *this week* you are free to ignore that advice and generate a single Verilog file.

## 3.2 Problem Description

Your friend Norbert runs a nursery in a small town. Recently a new industry moved into town and drilled several high-volume wells. This caused Norbert’s well to become inadequate for watering his plants. The well will now deliver only one gallon per minute, but it is capable of maintaining this rate for as long as needed. The well casing holds 60 gallons of water, and his well pump is capable of pumping 5 gallons a minute. Norbert only needs 1000 gallons of water each day, but he needs it all in the late afternoon. He has access to a large cistern. Since all of the well-drillers are extremely busy (Norbert isn’t the only one having this problem), for the time being Norbert is pumping all of the water from his well into the cistern every hour so that he can water his plants every day. Aside from the wear-and-tear on the well pump (from pumping the well dry too often), he is becoming grumpy from lack of sleep. He therefore asks you (the bright young engineer) to help him out.

The cistern has three float switches (all active high, in other words, if there is water at the level of the switch the output is a “1”, otherwise it is a “0”). One of these switches is close to the bottom, indicating that the cistern is just about empty. Another is at the very top, indicating that it is just about full. The third is (conveniently enough) 50 gallons below the top of the cistern. There is also an identical switch in the bottom of the well to indicate that the well itself is about to go dry. The pump motor has it’s own dedicated power supply, but the controller requires a logic “1” on a line to start the pump, and a logic “1” on another line to make it stop. It will not hurt anything to assert either of these lines after the pump controller has started or stopped. Norbert also has some timers left over from his automatic sprinkler system.

You decide to be a little conservative, and pump 50 gallons out of the well every hour. You want to design a circuit which has the outputs from the four switches and the output from two timers as inputs. Your circuit will output four signals: the signal to start the motor, the signal to stop the motor, an alarm signal indicating that the cistern is dry, and another alarm signal indicating that

it has been an hour since you last pumped and the well is still dry. The timers are configured so that one will output a 1 value ten minutes after the start signal is sent (e.g. after about 50 gallons have been pumped), and the other will output a 1 one hour after the start is sent. You'll use the first timer output to (help) stop the pump, and the second to start it again. The timers are outside of the scope of this lab, and you expect that Norbert might need to tweak them a little to fine-tune your system.

Your logic should be such that the motor is told to start if there is room for 50 more gallons in the cistern, it's been an hour since you last pumped, **and** the well is not dry (it should also be smart enough to not start the motor if another part of your circuit is telling the motor to stop, but telling it to start after you just told it to start will not hurt anything). Your circuit should tell the motor to stop once it has been pumping for 10 minutes, if the cistern is about to overflow, **or** if the well starts to run dry. You should set off the alarm to indicate a dry well if the level in the well is low and it has been an hour since you last pumped. The other alarm should be asserted (active high) if the cistern is below the level of the bottom switch (Norbert will use this to shut down the automatic waterers).

In Lab, you will not have timers, float switches, motor controllers or alarm klaxons, you will instead have switches that can be used to simulate each of the six inputs (two timers and four float switches) and you will have LEDs to indicate the four outputs (start motor, stop motor, alarm for dry well, alarm for dry cistern). Be classy and use a green LED for "start motor", a yellow LED (if available) for "stop motor" and red LEDs for the alarms when using the DE2 boards. Also, group your switches in a way that makes sense.

## 4 Report

You are to write a report in the usual style. This report will include original copies of our signed circuit diagrams, neatly drawn circuit diagrams, and a copy of your Verilog source file(s).