

CPE 166 / EEE270 ADVANCED LOGIC DESIGN LAB 1

Lab Session: Wednesday 2PM - 4:40PM

Student Name: Andrew Robertson

TABLE OF CONTENTS

Introduction.....3

Demo 14

Demo 25

Part 16

Conclusion8

INTRODUCTION

In this lab we first use existing, simple code to make sure we get acquainted with the Xilinx software and the process of uploading code to our Spartan boards. After this we then use existing code again to implement and test a slightly more complex module both on the board and using a test bench. Finally, the training wheels are taken off and we design, implement, and test our own 2 – 4 decoder using Verilog.

DEMO 1

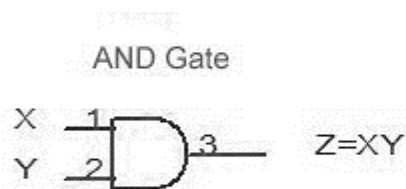
Design Purpose:

Demo 1 is to show us how to step through the process of designing in Xilinx using something simple that we know works. Both the implementation and test bench Verilog code have been provided.

Engineering Data:

The test1 module employs only one gate, an AND gate. Its properties are as follows:

2 Input AND Gate



TRUTH TABLE

INPUTS		OUTPUT
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Source code:

(Provided in lab manual)

Constraint file:

```
##Switches
set_property -dict { PACKAGE_PIN J15    IOSTANDARD LVCMOS33 } [get_ports
{ a }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 } [get_ports
{ b }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
## LEDs
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 } [get_ports
{ d0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
```

Simulation waveform(s):

(No waveform instructed to create)

Result discussion:

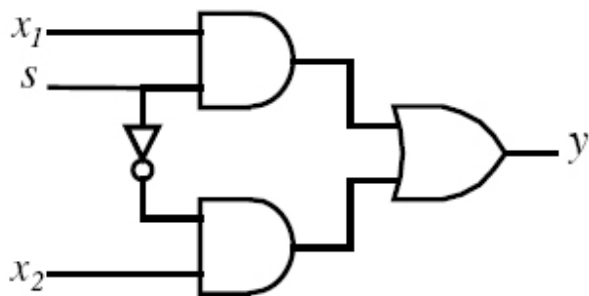
No problems encountered during this demo portion with everything being handed to us. Results were indeed as expected for an AND gate. Master constraints file really nice to have so only minor modifications are needed for pin association.

DEMO 2

Design Purpose:

Demo 2 is to show us how to simulate a 2 to 1 MUX circuit and produce a waveform using a test bench file. Both the implementation and test bench Verilog code have been provided.

Engineering Data:

<i>Truth Table</i>				<i>Logic Implementation</i>	
					
<i>s</i>	<i>x₂</i>	<i>x₁</i>	<i>y</i>		
0	0	0	0		
0	0	1	1		
0	1	0	0		
0	1	1	1		
1	0	0	0		
1	0	1	0		
1	1	0	1		
1	1	1	1		

Source code:

(Provided in lab manual)

Constraint file:

(Not to be uploaded to board)

Simulation waveform(s):

(Unfortunately, our group was unaware we needed a capture of the screen for this report, we were informed that it is okay this time but required for all labs from this point on.)

Result discussion:

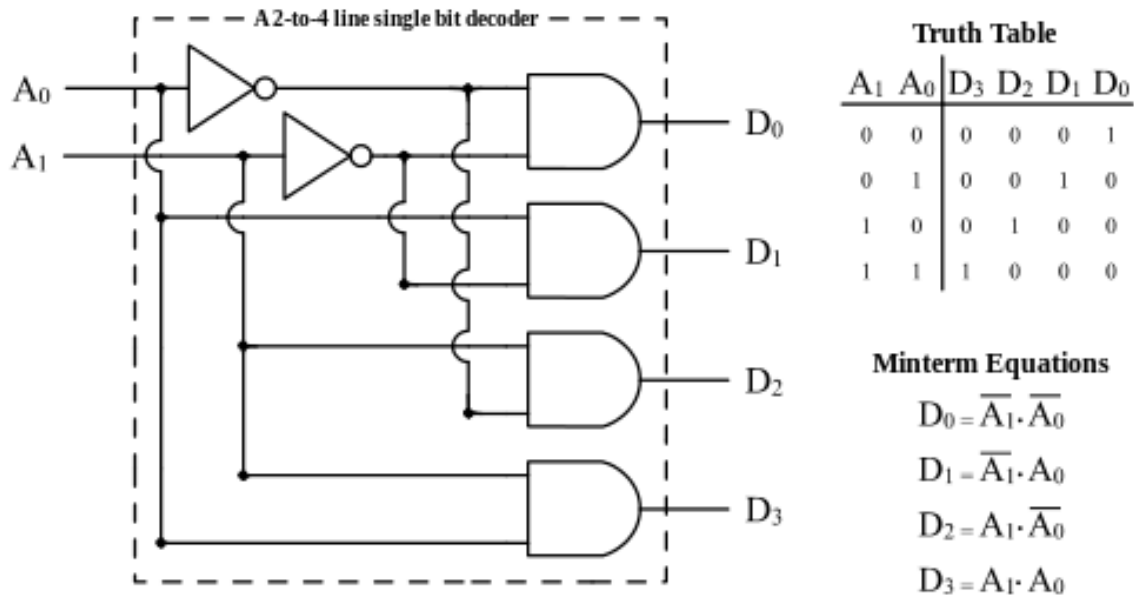
Simply adding the test bench file as a simulation file and having the module to be tested as a source file is all that was needed to get a simulation going. This is much easier than using Quartus and can manage enabling or disabling files in a project to allow multiple sources and test benches to exist without collision.

PART 1

Design Purpose:

Part1 is to allow us a chance to create a 2 to 4 decoder circuit using a given circuit diagram. We create a implementation file and upload to the board for testing and demo.

Engineering Data:



Source code:

```

/*
-----
--Author(s): Andrew R., Dylan, Angelica
--Date: 1/24/2018
--File Name: part3.v
--Purpose of code: Implement the design of a 2-4 decoder
--Project part Number: 1
--Hardware FPGA / CPLD Device: xc7a100tcsg324-1
-----
*/

module test1(a, b, d0, d1, d2, d3);

input a, b;
output d0, d1, d2, d3;
and g1(d0, ~a, ~b);
and g2(d1, a, ~b);
and g3(d2, ~a, b);
and g4(d3, a, b);

endmodule

```

Constraint file:

##Switches

```
set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports
{ a }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports
{ b }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
```

LEDs

```
set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports
{ d0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 } [get_ports
{ d1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 } [get_ports
{ d2 }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 } [get_ports
{ d3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
```

Simulation waveform(s):

(No waveform instructed to create)

Result discussion:

Here we were finally able to take a given diagram and create an implementation of our own. Although this design was still very straight forward, it was refreshing to be able to create something and see it come to life in the real world. We didn't run into any syntax errors or anything here either. After being shown the process twice via demo 1 and 2, this step flew by the fastest.

CONCLUSION

This lab was quick but did a great job refreshing me on some of the smaller details of Verilog. For example, for anyone else with a C background, for loops are very intuitive here, until spending a few minutes on Google only to find the ++ increment operator is not allowed. This lab also taught me some important features about the development environment that Quartus did not have. In Quartus, if you wanted to simulate something you had to modify the properties of the top-level file to point to a certain test bench file to run properly. Here, if the test bench file is added to the project as a simulation file and is the only one enabled it is automatically chosen to be the source of simulation.

Again, since this lab was more of an introductory and refresher session not too much thought needed to go into design. We didn't really exercise any muscles here over design style pros and cons or hierarchical design vs dataflow etc. but I'm sure this will be coming up very soon.