CPE/EEE 64

Lab One: Introduction to Verilog

Instructor:
Dennis DHALQUIST

Abstract

Two input logic gates are synthesized for the Altera Cyclone IV FPGA using the Quartus IDE. The logic gates are verified using a System Verilog testbench and Mentor's Modelsim HDL simulator

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

CONTENTS

LIST OF FIGURES

I. Introduction

VERILOG is a powerful way to describe circuits. Logic diagrams like those being used in lecture can become cumbersome in large designs. “Text based design entry” can be less prone to error because it is easier to track differences in large designs. Verilog is a text based hardware descriptive language the begun being used in ASIC(Application Specific Integrated Circuit) and now is the language of choice for FPGAs(Field Programmable Gate Array). We will be using the Terasic DE0-Nano development board with an Altera Cyclone IV FPGA on board. Altera provides a comprehensive solution for programming and debugging their FPGAs called Quartus. These labs will explore Quartus and use it to program the FPGA on the DE0-Nano development board. In this lab we will:

- Instantiate a System Verilog module
- Use a System Verilog Testbench
- Interact with external switches and indicators
- Synthesize Verilog code for a FPGA

A. *Verilog Modularity*

One of the most important features of Verilog is it’s ability to reuse a design. Reusing code allows you to rapidly assemble and test new designs. The ability to rapidly prototype a design is one the biggest advantages to the FPGA. This section will involve provided logic gate Verilog modules which will be reused in later labs to build more complex structures. Reusing these modules is very similar to how you would reuse code in the workplace to be more productive. The lab documentation comes with Verilog implementations of the four logic gates in **Source.zip** The demo for the lab will be implementing these modules with the DE0-Nano development board and testing the design on a breadboard. You could think of this as the source libraries that would be available at the company that you might work for.

B. *Test Bench*

Verilog roughly breaks into two halves synthesizable and non-synthesizable. FPGAs synthesis can take a very long time, using a simulator to verify individual modules can be much faster than resynthesizing the entire design. The Testbench also offers a unique ability to check expected outputs generate test stimulus. We will use a test bench to check the provided Verilog modules are providing the desired operation in part C of the procedure. This simulation should be verified against the known truth table for the logic gate to ensure the module is accurate.

II. Lab Procedure

THIS section is a guide for what must be demonstrated to the laboratory instructor and documented in your lab report. For this section you will need:

- DE0-Nano Development board
- A breadboard
- Four LEDs
- Two dip switches
- Windows or Linux based computer
- Internets

A. *Install Quartus*

Download most recent version of Quartus and Cyclone device drivers from Altera. Screencast 1 is a walk through for windows, but Quartus is also available for Linux.

B. *Expand Source.Zip*

The Quartus project file, logic gate modules, and test bench are contained within this archive. Expand it wherever is convenient for you, it will be accessed frequently. The Quartus project file included with the source code is generated with the Terasic DE0-Nano System Builder that is included with the Terasic System CD. You can use this if you want to generate a clean project for yourself.

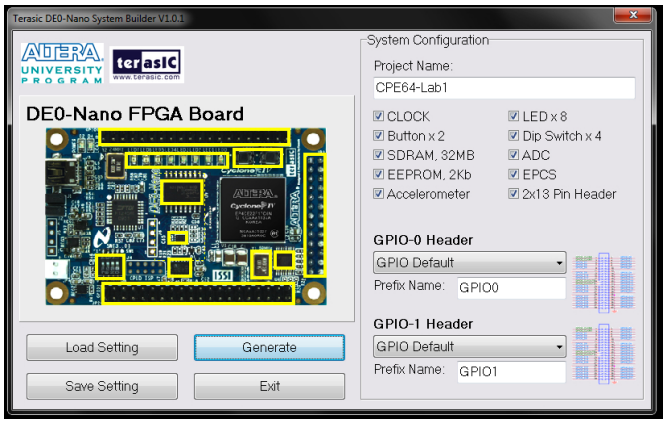

Fig. 1. TerasIC System Builder GUI

C. *Run testbench*

Now we want to verify our design. Using a simulator for a single logic gate is a bit asinine but the experience gained with the example test bench will help you greatly in the future. Start a new simulation and add the waveforms as shown in screencast 2. You'll want to have have truth tables laid out for the gates your testing so you can be sure they behave the way you expect. Have an extra column ready to record the behavior of the FPGA once it's programed. Figure II-C shows an example of what the wave section should look like.

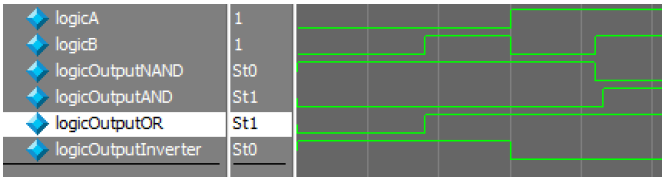

Fig. 2. Example output of testbench

Take a moment to look at the simulation transcript, it provides the states of the logic elements being tested. I prefer having the simulator give a test listing instead of reading the waveforms. This is from the $display() lines in the testbench. listing the outputs can be a very powerful debugging tool. I typically use the $assert() statement which can pause the simulation and alert you when an unexpected result is produced.

```
A:0 B:0 - Inverter:1 AND:0 OR:0 NAND:1
A:0 B:1 - Inverter:1 AND:0 OR:1 NAND:1
A:1 B:0 - Inverter:0 AND:0 OR:1 NAND:1
A:1 B:1 - Inverter:0 AND:1 OR:1 NAND:0

```

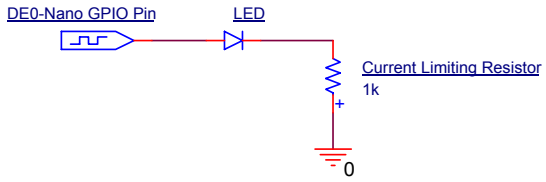

Fig. 3. LED with current limiting resistor

The LED will allow you to see the the output but we'll also need to be able to generate some input for the FPGA. We will do this with a dip switch and pulldown resistor. The pulldown resistor is needed to literally pull the charge off the wire so the input will read a solid zero. Otherwise the pin would "float" between 1 and 0 arbitrarily.

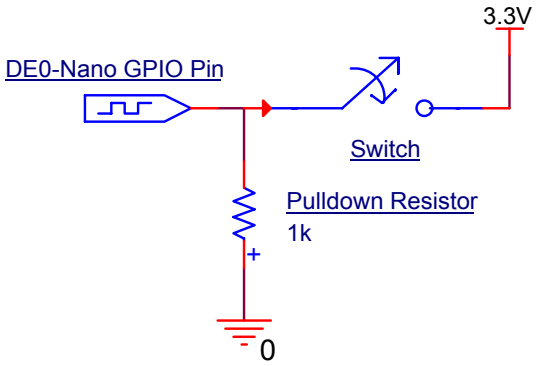

Fig. 4. Switch with pulldown resistor

D. *Prepare circuit to test Verilog gate modules with DE0-Nano*

A switch and LED are going to be used to test the FPGA while it's operating. This will allow you to generate inputs for the FPGA with the switches. The LED circuit needs a current limiting resistor as shown in the Figure below.

*1) **GPIO headers on the DE0-Nano:*** Be careful when referencing the pin diagrams in the DE0-Nano user manual. It is easy to read it backwards and that can be a mistake that will cost you hours. It is easiest to match the Nano's orientation with the schematic and count from the nearest edge. Always check VCC_SYS, VCC3P3, and GND with a multimeter before attaching a circuit you have built.

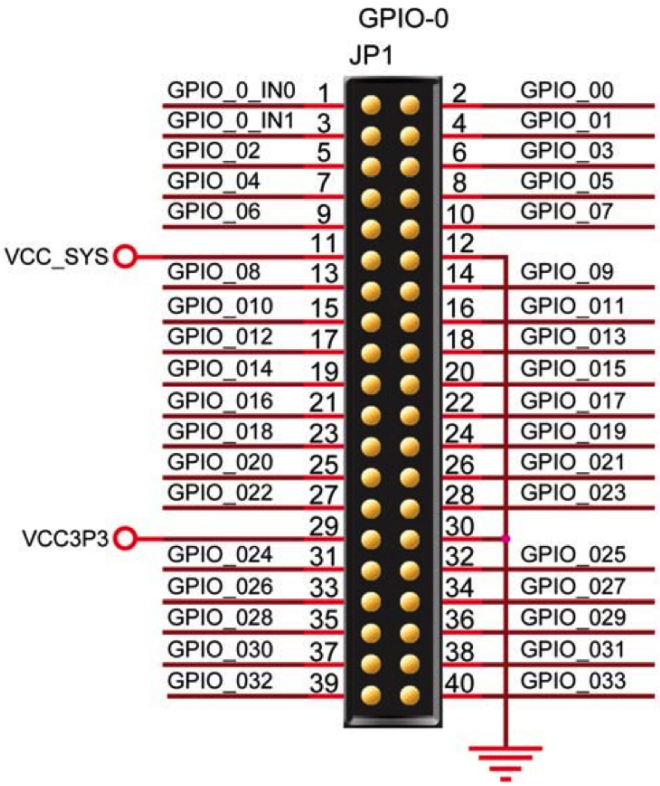

Fig. 5. Schematic of GPIO-0 header [1]

This is the header pin schematic from the DE0-Nano user manual. Inside the verilog code these pins follow a little different nomenclature. What is labeled as GPIO_0_IN0 in figure 5 is GPIO0_IN[0] and GPIO_00 is GPIO0[0]. Refer to the DE0-Nano user guide for details on the orientation of the headers.

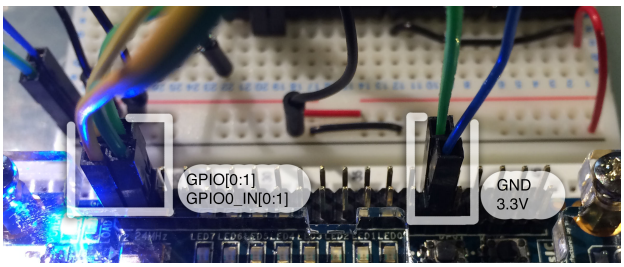

Fig. 6. Picture of loaded GPIO-0 header

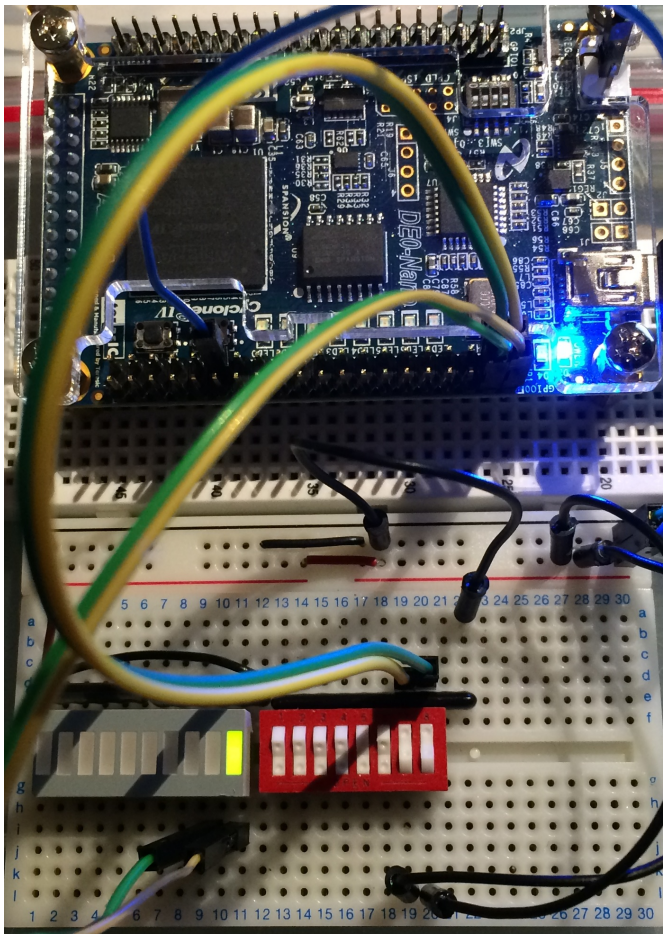

Fig. 7. Example switch and LED configuration with SIP resistors

*2) **Compile example code with Quartus:*** Once you've created and tested your switch circuit we'll need to compile the example code. This can be done with the Quartus development environment. A walk through is provided in screencast X.

*3) **Use Quartus to program the Nano:*** refer to screencast 2 for a walk through. After synthesis Quartus will generate a .SOF[1] file that can be used to program the FPGA using Quartus' programmer. Follow the procedure outlined in screencast X.

E. Test behavior against expected truth table

Use your table from the simulation

F. Write XOR module

Now that you've had the opportunity to experiment with the provided modules the time had come to write your own module. When starting a module

[1]the .SOF stands for SRAM object file. This is an Altera standard for of their FPGAs.

it's usually best to copy something you know works and modify it to suit your need. In this case I would make a copy of AND.sv and change the logic to XOR. Verilog does have a built in operator for this. Your best friend is google, that's always the best place to start.

III. Lab Report

The lab report must be typed and submitted in a PDF format. Look to IEEE's guidelines for formatting guidelines on format. The document should include

A. *Figures to include*

- Waveform captures from Modelsim
- Logic tables from theoretical prediction and experimental outcome
- Explanation and listing of your XOR Verilog module.

B. *Questions to answer*

- There are multiple ways to instantiate a module what are three different ways that you could instantiate the AND module included with this lab?
- Compiling a programming language like C and synthesizing Verilog are very different even though they appear to be the same in the IDE. How might
- Notice the report that pops up when you compile your project. There are a number of statistics given by Quartus, the logic element usage ratio is your designs use of the total device capacity. More Verilog roughly translates into more LC usage. What was your designs Logic Cell utilization?.

IV. Conclusion

Verilog is an IEEE standard(1364) [2], it is pervasive in industry and can be used to develop specialized hardware in the form of ASICs or reconfigurable FPGAs. It is important to underscore the differences between Verilog and a programming language like C, Java, even Assembly. Verilog offers the ability to take parallel action. Two numbers can be multiplied at once, multiple registers can be set and cleared. Entire microprocessors can be implemented on the Nano, one of the later labs will explore Altera's softprocessor the NIOS II. The TerASIC documentation included with the DE0-Nano kit is pretty good and worth the read. It will help you get the most out of the FPGA. The CD included with the Nano will also include circuit schematics that can provide a great reference when it comes time to make one of your own.

*a) **More Information:*** just like anything else the Internet has an amazing amount of information available on the Internet to the interested student.

- Altera Traning Curriculum for FPGA designers
- EEV Blog: What is a FPGA?

References

[1] TerasIC, *DE0-Nano User Manual*, 1st ed., 2012.
[2] W. Foundation. (2013) Verilog. [Online]. Available: https://en.wikipedia.org/wiki/Verilog