# Lab Two: Introduction to logic on the FPGA

Ben Smith

***Abstract*—This document is an introduction to the DE0-Nano development board, Altera's Cyclone FPGAs and the Quartus IDE. The block editor feature of Quartus is used to synthesize logic gate primitives and more complex logic functions from these primitives.**

## I. Introduction

THIS lab introduces the DE0-Nano development board and Altera's Quartus development environment. Quartus will be used to program the Altera Cyclone IV FPGA on the DE0-Nano.[1] We will start by using Quartus' schematic representation of logic modules and move into text based design capture. The schematic entry method his will be very similar to the logic designs that you have created in Multisim and implemented with discrete 7400 series logic ICs. The schematic editor in Quartus can be used to graphically represent Verilog modules as well, like you will do in Lab Two, or logic primitives like will be done in this lab. This lab is intended to introduce the student to the following concepts.

- Logic primitives on a FPGA
- Quartus development environment
- Synthesis of a block based design
- Assigning pins for a design
- Programming an Altera FPGA

### A. Included Screencasts

A number of screencasts are included with this set of labs. They are available on Youtube and as zipped MP4s on the course website. They are intended to be short and to the point so they cover individual topics.

1) TIME - Installing Quartus
2) TIME - Opening, Compiling, programming with Quartus
3) TIME - Block editor in Quartus
4) TIME - Using the generated pin assignments from the TerasIC system builder

### B. DE0-Nano Development board

The DE0-Nano has a number of devices built into the board to expand the capabilities of the FPGA. Most of these will be beyond the scope of the course but represent real world design challenges and are worth experimenting with after this basic course is completed. Altera offers a number of tutorials for anyone interested in learning more. These labs will use one of the 40-pin GPIO headers to interact with the outside world. The A/D converter and G-sensor will be great tools to interact with sensors in your later classes. If you choose to go down a path of controls engineering FPGAs will be a great way to implement "deterministic" control systems. Interfacing with these components will require the implementation of a SPI or I2C bus protocol. This is done in Verilog just like what we'll do in Lab Two. The De0-Nano system CD comes with some example code that uses these devices.

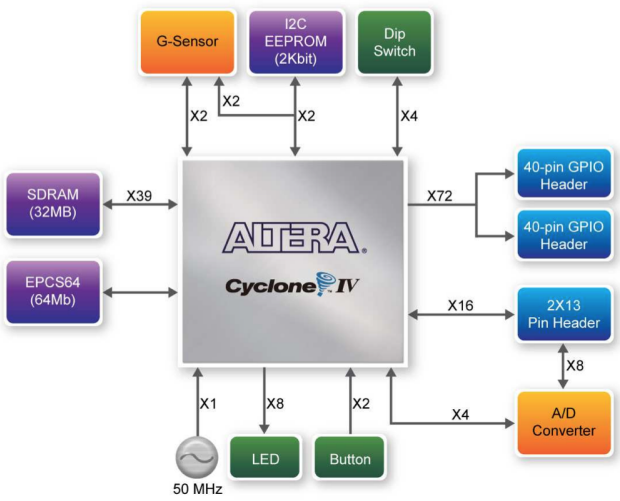



Fig. 1: Block Diagram of DE0-Nano [**?**]

## II. Lab Procedure

THIS section is a guide for what must be accomplished in the lab. Keep in mind the some of the following material will need to be documented in your lab report. It would behoove you to take a look at the lab report section before starting the procedure. Knowing what you need to record in the experiment will be very important to you. For this section you will need:

- De0-Nano Development board
- A breadboard
- Four LEDs
- Two dip switches
- Windows or Linux based computer
- Internets

### A. Optional: Install Quartus

You can download the 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. This is preinstalled on the computers in the Digital design laboratoy. The same version is available from Altera free of charge if you would like to use your own computer.

[1] EEV Blog: What is a FPGA? Take a look at this page for a great explanation of what FPGAs are all about, even if a kangaroo delivers the speech. So mount that wallaby and get some understanding

### B. Expand Source.Zip

The Quartus project file and top level schematic 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. The contents of the Zip are explained below, it is from the perspective of the root of the archive.

1) Folder: Quartus Project
   The quartus project file is contained in here with the myriad of required support files, generated reports, and logfiles.
2) Folder: Source Code
   The Quartus project references these files. I choose to store them outside of the quartus project file so they are easy to access.
   a) 64Lab1.bdf: top level schematic that will be used to capture logic designs
   b) .SV: System Verilog source file
   c) .V: Verilog source file

these files are used to save you some of the tedium of setup. It's worth attempting to create your own project, A tutorial has been included called ScreencastX.

### C. Open Quartus project

Navigate to where you expanded the Source.zip file. The Quartus project file you want to open is called "./Verilog/Quartus Project/64LabOne.qpf"[2] When you open the project file open the "Top Level" .BDF file Inside this file there will be a number of logic gates attached to pins of the FPGA. A screen cast explains the intricacies of the Schematic editor.

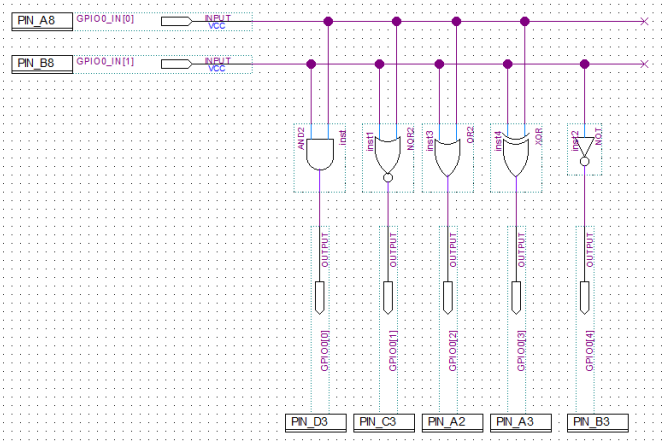



Fig. 2: Quartus logic schematic editor example

### 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 switch will allow you to generate inputs for the FPGA. The LED circuit needs a current limiting resistor as shown in the Figure below. 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.

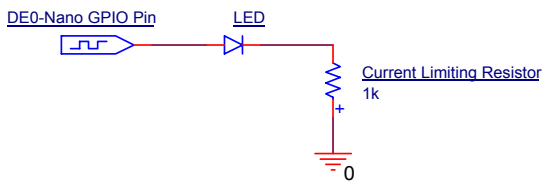



Fig. 3: LED with current limiting resistor

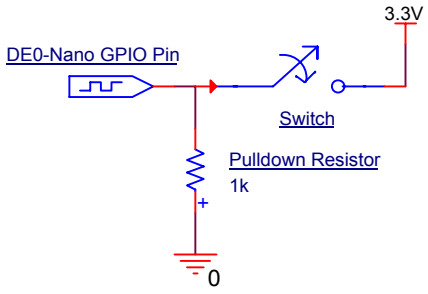



Fig. 4: Switch with pulldown resistor

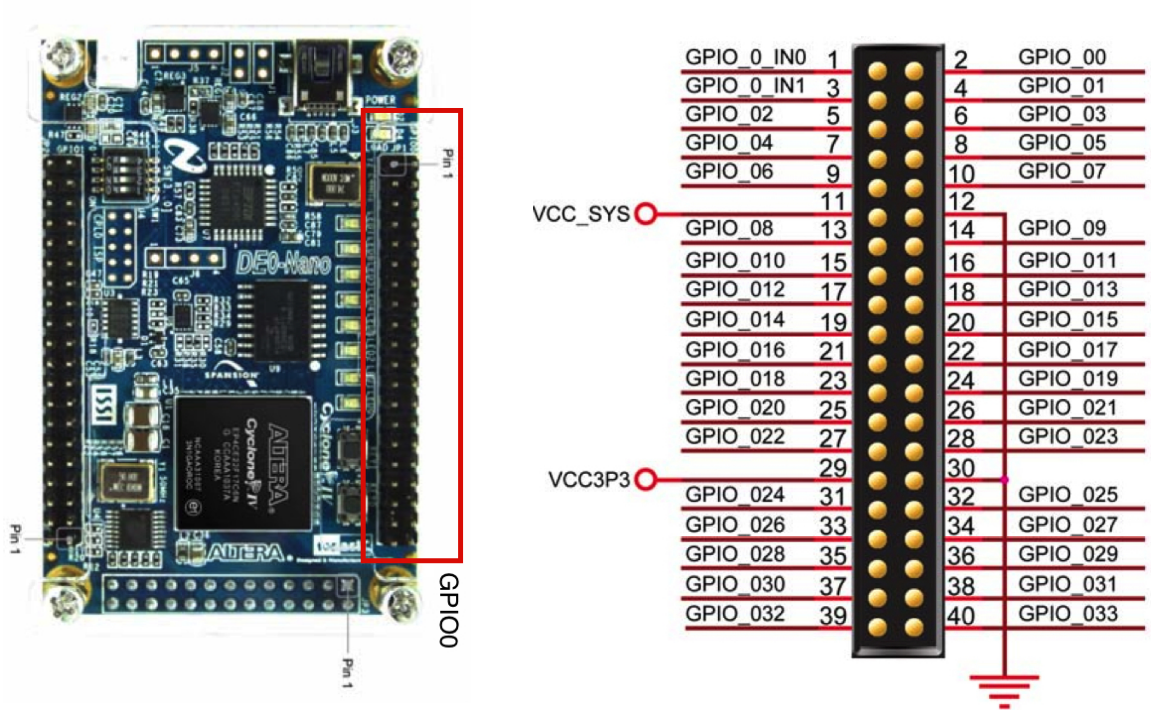
*1) GPIO headers on the DE0-Nano:* Be careful when referencing the pin diagrams in the DE0-Nano user manual. It is easy to read it backwards and that can be a mistake that will cost you hours. It is easiest to match the Nano's orientation with the schematic and count from the nearest edge. Figure 5 shows GPIO 0 next to a schematic of the header. Always check VCC_SYS, VCC3P3, and GND with a multimeter before attaching a circuit you have built. 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_IN[0] 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.

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

*3) Use Quartus to program the Nano:* refer to screencastX for a walk through. After synthesis Quartus will generate a .SOF[3] file that can be used to program the FPGA using Quartus' programmer.

[2]The ./ refers to the root of the source.zip package

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




(a) Orientation of GPIO Header [**?**]

(b) Schematic of GPIO-0 header [**?**]

Fig. 5: GPIO0 and it's orientation of the development board

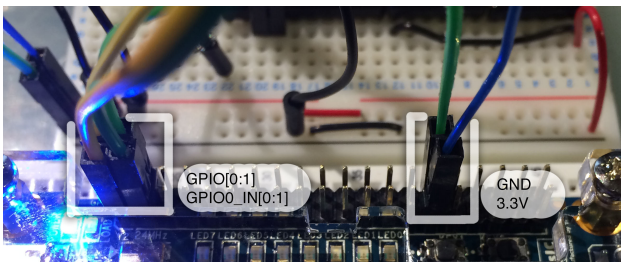



Fig. 6: Picture of loaded GPIO-0 header

### *E. Test behavior against expected truth table*

Use the known behavior of the logic gates to test your design on the FPGA. It is imperative that you always verify the operation of a design. This design can be verified by creating a Logic Table for each of the gates. If the LED and switch behaves as expected, record the experimental data and move on. If not the design will have to be debugged.

### *F. Design XOR module*

Now that you have had the opportunity to experiment with the logic primitives in the schematic editor the time has come to create your own design. Lets say that we need a four input XOR gate for some reason. We know the truth table of a XOR gate, it must be high when any number of the inputs is high but all of them as shown in Table I. Quartus does have an embedded XOR primitive which the use of is disallowed. Create the circuit from the basic 7400 series logic operators(AND, OR, NOT, NAND). Always remember, Google is your best friend. Be sure to include a boolean logic expression for this truth table in your report.

TABLE I: Truth table for XOR logic function

| A | B | C | D | Output Q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

### *G. Design AOI module from logical equation*

The "And or Invert" function is common in digital design, you will implement it with logic primitives in the schematic editor. There will be a wealth of information available on line about how to implement the function on the Internet but take a stab at it before heading to Google.

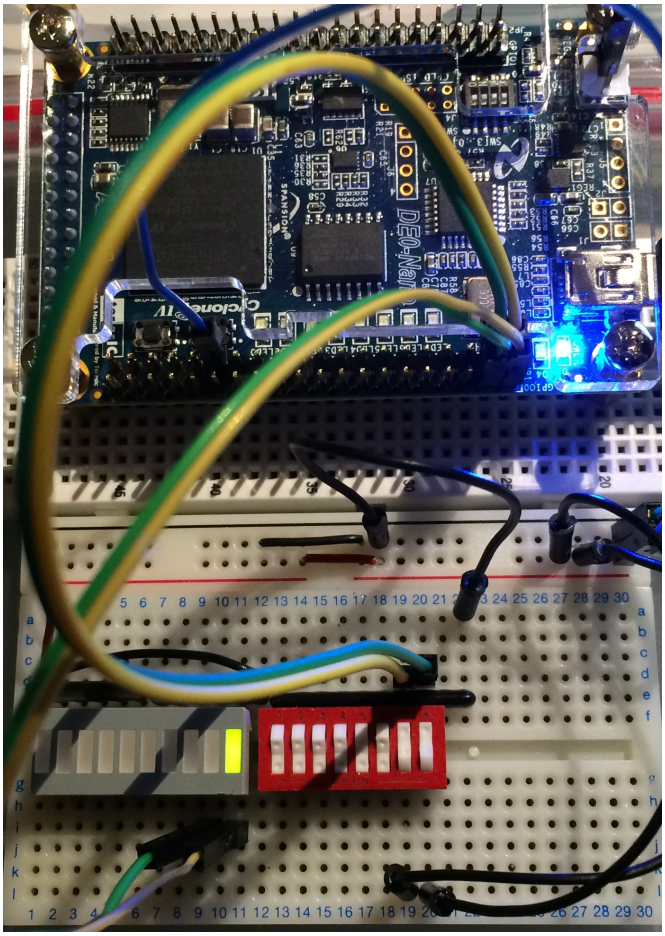


Fig. 7: Example switch and LED configuration with SIP resistors

$$F = \overline{(A \wedge B) \vee (C \wedge D)}$$

Be sure to include a truth table in your lab report.

## III. Lab Report

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

### A. Figures to include

- Schematic of your XOR Gate
- Logic tables from theoretical prediction and experimental outcome
- Explanation and schematic of your XOR module.
- Explanation and Schematic of your AOI module.

### B. Questions to answer

1) 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. What was the logic usage of your FPGA?
2) The synthesis engine in Quartus is very powerful and will optimize your design for the resources on the FPGA. Quartus generates the design using three logic gates and two inverters, which gate do you think it changed to inverters?

## IV. Conclusion

THIS lab introduced Quartus and some of it's basic functionality. Quartus is very similar to many industry standard tools. Xilinx, another manufacturer of programmable logic devices, offers a tool suite very similar to Quartus called ISE. Although we will not discuss ISE, a understanding of Quartus will allow ISE to be learned very quickly. These are tools that you can expect to see in industry as a CPE or EEE student. There is an extreme demand right now for engineers with an understanding of programmable logic.

## References