**FPGA Introduction Lab**

**A. La Rosa, B. Comnes**

**Physics 315**

**Portland State University**

# Abstract

In this lab we will be briefly exploring Field Programmable Gate Arrays (FPGA). This will be achieved by a brief background discussion of FPGA characteristics, followed up by a tutorial illustrating a very basic workflow. The lab was designed around the Digilent Nexys 3 prototyping board, which provides access to a Xilinx Spartan6 XC6LX16-CS324 FPGA chip along with a programmer, program memory, and array of switches, buttons, LEDs and I/O ports.

# FPGA Introduction

## What is an FPGA?

FPGAs are used as tools in modern digital circuit design. FPGAs provide a large array of generic logic gates inside of a single integrated circuit (IC) that can be configured arbitrarily, so long as it does not exceed the capabilities of your FPGA chip.

The FPGA is configured using a Hardware Description Language (HDL), making FPGA designs portable between hardware, so long as the correct constraints are used to map the HDL design to the specific details of the FPGA that is being used. There are also other options to configure the FPGA such as schematic layout interpreters, as well as proprietary solutions provided by LabVIEW as an example.

Not only can FPGAs be configured into any arbitrary digital circuit, they can also be reconfigured at a later time, if for example, you need to fix a problem in the circuit after you have integrated it into a project. It can even reconfigure itself to act as a different circuit depending on external conditions.

## How is it different than other embedded systems

There is a large array of programmable logic (PL) technologies available, and at this point it might seem like FPGAs are a glorified microcontroller. This is not so. When you configure an FPGA, the logic gates inside the IC are configured correctly such that you are actually, physically creating the circuit that you have specified, all inside the FPGA.

Also, unlike other PL, the FPGA can be rewritten as many times as needed. The flash memory which stores the program to configure the FPGA on power up will be the limiting factor, with a re-write limit of about 100,000.

Many FPGAs have an array of IP (Intellectual Property) modules which are premade FPGA configurations that can perform a complicated task. For example there are IP modules to implement a soft CPU in your FPGA so that it can be used as a more general purpose computer. These modules typically cost money and include some form of DRM or obfuscation to keep you from seeing how they work. The whole purpose of not just simply sharing the HDL code is to prevent others from seeing how the developer implemented a particular design.

## Advantages

When implementing complex digital circuits, FPGAs provide numerous advantages. The design can be written, tested and simulated on the computer. The designs can be done in a way so that they are portable to other FPGA devices. Multiple people can work on the same HDL files an increase the speed of circuit development.

Once the design is ready for implementation, the entire production process is largely decreased in size, as the FPGA will configure the circuit, rather than having to build it with a machine or by hand. If there is ever an issue with the circuit that is found out at a later date, it is only a matter up updating the HDL code and re-uploading it to the FPGA.

Running the circuit in a single chip also allows for high speed circuitry that would simply be unattainable on a breadboard. When used correctly, FPGAs can be extremely fast.

## Drawbacks

FPGAs have been historically quite expensive. Additionally, they can be fairly difficult to work with, requiring a wide range of knowledge of HDLs and digital circuit design to use effectively.

The software tools used to work with many FPGAs are also quite hefty in price, and in size. The IDE we are using is more than 8Gb in download size, and as you will learn, has a user interface that will take some time getting used too.

# Working with the FPGA

## Workflow overview

For working with our Nexys 3 FPGA, we will be using the **ISE Project Navigator**. This integrated development environment (IDE) will allow us to set up a project and add different design, test and constraint files, as well as act as a debugger and handle the compiling of our designs.

Once we have design, tested, and compiled our FPGA configuration, we will need to upload it to the board. For this, we must use a tool called **Adept** which is made by Digilent.

## Development options

There are a number of different options for configuring an FPGA, such as HDLs, schematics or proprietary solutions. We will focus on schematics and an HDL called **VHDL**.

## Hardware Description Languages

FPGA designs are usually made with an HDL such as **VHDL** or **Verilog**. It is easier to maintain code rather than a 2-D schematic, which is why it HDLs are used more often.

These languages are similar to a programming language, but are rather geared towards digital circuit design, so they have the same limitations has digital circuits. For this lab, these differences will not be of focus. Understanding these differences would be important if you were actually to use an FPGA in a more complex project.

# Configuring the FPGA using schematics

# Configuring the FPGA using VHDL

Open the ISE project Navigator

File - > New Project

Give your project a name with no spaces or special characters. DON’T USE hyphens! (-)

Under top-level source choose HDL

Click next

Under evaluation development board choose Spartan 6 sp601 evaluation platform

Product category family speed package will all be chosen for you

Under Preferred language choose VHDL

Click next

You will get a project summary window (new project 4.png)

This closes the new project wizard, and dropes you into your ISE project navigator Integrated Development enviroment. It is a very non-intuitive interface, that you will only become accustom to with experience.

We will be building a simple And circuit using VHDL and manual schematic layouts. Both are achieve the same thing, although VHDL will eventually get you more mileage.

Add a new source schematic to your project by selecting Project -> New Source which oopens the new source wizard (new source wizard 1)

Give it a name, avoiding special characters and click next to get to the summary of the wizzarrd ( new source wizard 2)

Click finished

It will create the schematic and open it. (new schematic 1)

Click the add symbol toolbar button in the vertical toolbar strip

This opens a menu to the left of the toolbar with digital electronic components. They are sorted into categories for somewhat quick access, but the name filter is the quickest way to find what you are looking for if you know the name

Click the logic category, and notice the symbols list shortened to only show the logic componants in alphabetical oder

Choose the and2 symbole from the symbol menu.

Now when you roll over the schematic view, you get something that looks like that compotant following around a cross. This lets you know what symbol you are holding and ready to place on your schematic. (schematic placement)

Go ahead and place two of these and gates, by hovering to where you want to place them, and clicking once . Press escape to clear your placement selection and go back to mouse mode. (schematic placed).

To see more clearly, lets zoom in. Use the zoom tools to zoom in on your parts. (zoom.png) or the zoom select tool from the verticle menu.

Use the add wire tool in the verticle toolbar to add short wires to the three leads on your and gate. Clicking on the terminal on the and gate, then clicking where you want the wire to go drops a wire segment. Double clicking will end the wireplacement. Press escape to end wireplacement all together and clear your tool. (add wire tool)

Add IO markers. These markers wire your circuit to the inputs and outputs of your fpga.

Click the add IO button in verticle tool bar. Click all 3 terminal leads to drop a marker. (lead marker dropped).

Right click to rename each port. Name the inputs A and B and the output OUT1 (io rename)

Check your schematic for errors in Tools -> Check shematic (check schematic png)

Errors show up in the conosole at the bottom. Fix any errors.

File -> Save all

Now we test our schematic to see if it works.

Project -> Add Source

Select VHDL Test Bench

Give it a name and press next (addtb)

The next window will ask you to select wich vhdl or schematic to test

You only have one (AND\_schematic) so select that (tbconfirm)

Click next to view a summary (tbsummary)

The wizard will close and you will be presented with a file called [name of file].vhd which is your vhdl code you will use to specify how to test your schematic you slected in the wizard.. (tbvhdl)

This VHDL file will be where we describe how we want to test our circuit during simulation. Make the following changes to the code. (tb changes)

File-> Save all

Click design tab on left

Click simulation (simtab)

Select your test bench

Expand ISIM smulator

Double click simulate behaviorl model

This compiles the schematic, and test bench.

This launches the ISim program and simulates the inputs to your circuit.

Make sure to press the zoom out button to see the full signal.

Close ISIm

Now we need to create a constraints file. A UFC file. This file assigns io pins of the fpga to the io ports of our code.

Switch back to the implementation view

We need to know the switch names and LED names (switches)

Right click the hierarchy payne and click new source

Select implementation constraints file and give it a name \*(new const)

You will see a constaint summary click finish. (const summary)

You should now see it under your schematic. open it (bare constraint)

Enter the code from the screenshot (Const Code)

Plug in the FPGA board

Turn it on. Make sure it says pass 128

Highlight you .sch in your hierarchy view and double click Generate programming file in your lower Process menu (generate programming file)

Get coffee. This takes a few minutes.

Once it stopped, you will have 3 green checks, If there were errors, try to fix them and try again. (Generate complete)

Now upload the program to your board. Open Digilent Adept, and switch to the Test tab and click run RAM/Flash test. IF it passes you are talking to your board. If not, something went wrong. (adept test)

Go to the config tab and click brows. (adept config)

Navigate to your project directory folder and open the .bit file. (bit file open)

Click program and your schematic is now being configured on the FPGA .

Test your and gate, confirming that the led comes on when they are both switched on.

Note, that powering down your FPGA will reset it. FPGAs will lose their configuration when they lose power. Luckily, our board has non volatile memory which can store our configuration such that it is reconfigured every time it is powered on.

Now we will do the same thing but using VHDL, a Hardware Description language HDL. This is a more viable alternative than the schematic layout for complex projects for many reasons. If you were actually to do a project with the FPGA, you would likely be using VHDL or verilog.

Click the design tab and then the Implementation view (VHDL start)

Right click the top folder and click New source.

Select VHDL module and give it a name (vhdl naming) and click next

Type in the names of the ports (they can be the same or different than the ones used in the schematic. We aare starting from scaratch), (VHDL io) and click next

You will be presented with a summary (VHDL summary)

Contiune on and you should now be presented with your new VHDL file. (bare vhdl file)

To do the same thing as our schemtic all you have to do is add one line of code between begin and end behavioral

Save the document,

Go to the simulation view in the right, right click the new vhdl file and create a new test bench file like you did before. (vhdl testbench)

Click next and then select the VHDL file we just created (vhdl tb select)

Click next and and confirm at the summary screen

After it is created, the new test bench file is opened.

Modify the code to look like vhdl\_test\_bench

Double click simulate behavioral model the same way we did for the schematic, and confirm that our vhdl model works the same way.

See if you can create a constraint file for the vhdl code and upload it to your fpga!