EECS 151/251A FPGA Lab
Lab 1: Getting Set Up and Familiarizing Yourself with Tools

Prof. Jan Rabaey
TAs: George Alexandrov, Ali Moin
Department of Electrical Engineering and Computer Sciences
College of Engineering, University of California, Berkeley

1 Setting Up Accounts

1.1 bCourses and Piazza

If you are enrolled in this class, you should have access to the EECS 151/251A bCourses page (`https://bcourses.berkeley.edu/courses/1473222`) already. If you don't have access, please let a TA or the instructor know immediately so we can resolve the issue. We will be using bCourses to publish documents and make announcements for this course.

You should also register for a Piazza account and enroll in the EECS 151/251A class as soon as possible (`https://piazza.com/class/jkdsruyfq2x2xv`). We will be using Piazza as a discussion forum for this class and the labs. We will also send announcements through Piazza so it is vital that everyone is signed up.

1.2 Getting an EECS 151 Account

All students enrolled in the FPGA lab are required to get a EECS 151 class account to login to the workstations in lab. This semester, you can get a class account by using the webapp here: `https://inst.eecs.berkeley.edu/webacct`

Once you login using your CalNet ID, you can click on 'Get a new account' in the eecs151 row. Once the account has been created, you can email your class account form to yourself to have a record of your account information.

Now you should be able to login to the workstations we have available in the lab. Enter your login and initial password in the login screen. Let the lab TA know if you have any problems setting up your class account.

1.2.1 Changing your password

To change your default password, click on Applications on the top left toolbar on your workstation desktop, then hover over System, then click on Terminal. In the terminal type and execute the command: `ssh update.cs.berkeley.edu`

You can then follow the prompts to set up a new password. You can always use the same webapp that you used to create your account to reset your password if you forget it.

1.3 Getting a Github Account

If you haven't done so previously, sign up for a Github account at `https://github.com/` with your berkeley.edu email address.

If you already have a Github account that's registered with your personal email address, don't create a new account. Instead, login to Github, go here `https://github.com/settings/emails`, and add your berkeley.edu email address to your Github account.

Once you have an account, send the TAs an email with your Github account username and email and your class account login (eecs151-xxx). Our emails are gpalex@berkeley.edu and moin@berkeley.edu. Try to do this as soon as possible.

2 Getting Familiar with our Development Environment

2.1 Linux Basics

In this class, we will be using a Linux development environment. We will be using CentOS as our Linux distro, which is a free version of Red Hat Linux. If you are unfamiliar or uncomfortable with Linux, and in particular, using the bash terminal, you should definitely check out this tutorial:

`https://www.digitalocean.com/community/tutorial_series/getting-started-with-linux`

It is highly recommended to go through all four parts of the tutorial above, even if you already are familiar with the content. To complete the labs and projects for this course, you will find it helpful to have good command line skills.

One of the best ways to expand your working knowledge of bash is to watch others who are more experienced. Pay attention when you are watching someone else's screen and ask questions when you see something you don't understand. You will quickly learn many new commands and shortcuts.

2.2 Git Basics

Version control systems help track how files change over time and make it easier for collaborators to work on the same files and share their changes. For projects of any reasonable complexity, some sort of version control is an absolute necessity. There are tons of version control systems out there, each with some pros and cons. In this class, we will be using Git, one of the most popular version

control systems. It is highly recommended that you make the effort to really understand how Git works, as it will make understanding how to actually use it much easier. Please check out the following link, which provides a good high level overview:

`http://git-scm.com/book/en/Getting-Started-Git-Basics`

Once you think you understand the material above, please complete the following tutorial:

`http://try.github.com`

Git is a very powerful tool, but it can be a bit overwhelming at first. If you don't know what you are doing, you can really cause lots of headaches for yourself and those around you, so please be careful. If you are ever doubtful about how to do something with Git ask a TA or an experienced classmate.

For the purposes of this class you will probably only need to be proficient with the following commands:

- `git status`
- `git add`
- `git commit`
- `git pull`
- `git push`
- `git clone`

However, if you put in the effort to learn how to use some of the more powerful features (diff, blame, branch, log, mergetool, rebase, and many others), they can really increase your productivity.

Git has a huge feature set which is well documented on the internet. If there is something you think Git should be able to do, chances are the command already exists. We highly encourage you to explore and discuss with fellow classmates and TA's.

Optional: If you would like to explore further, check out the slightly more advanced tutorial written for CS250:

`http://inst.eecs.berkeley.edu/~cs250/fa13/handouts/tut1-git.pdf`

3 Setting Up Github Access

We will be using Github as our remote Git server for this class. Github is a popular Git hosting service which is home to many private and public (open-source) projects.

3.1 SSH Keys

Github authenticates you for access to your repository using ssh keys. Follow this tutorial to get SSH keys set up (this should be done on a lab workstation when you are logged in with your eecs151

class account).

First, create a new SSH key (do this on the lab computer):

```
ssh-keygen -t rsa -b 4096 -C "your_email@berkeley.edu"
```

Keep hitting enter to use the default settings.

Then, from your terminal run:

```
cat ~/.ssh/id_rsa.pub
```

Copy the public key that's printed out in its entirety. Go here: `https://github.com/settings/keys`, click on 'New SSH Key', paste your public key into the box, and click 'Add SSH key'.

Finally test your SSH connection: `https://help.github.com/articles/testing-your-ssh-connection/#platform-linux`.

If you have any issues, ask a TA for help.

3.2 Acquiring Lab Files

The lab files, and eventually the project files, will be made available through a git repository provided by the staff, so make sure you have emailed the TAs with your Github account username and email. Once this has been done, the suggested way to obtain these files is as follows. First, set up your ssh keys as described above. Then run the command below in your home directory,

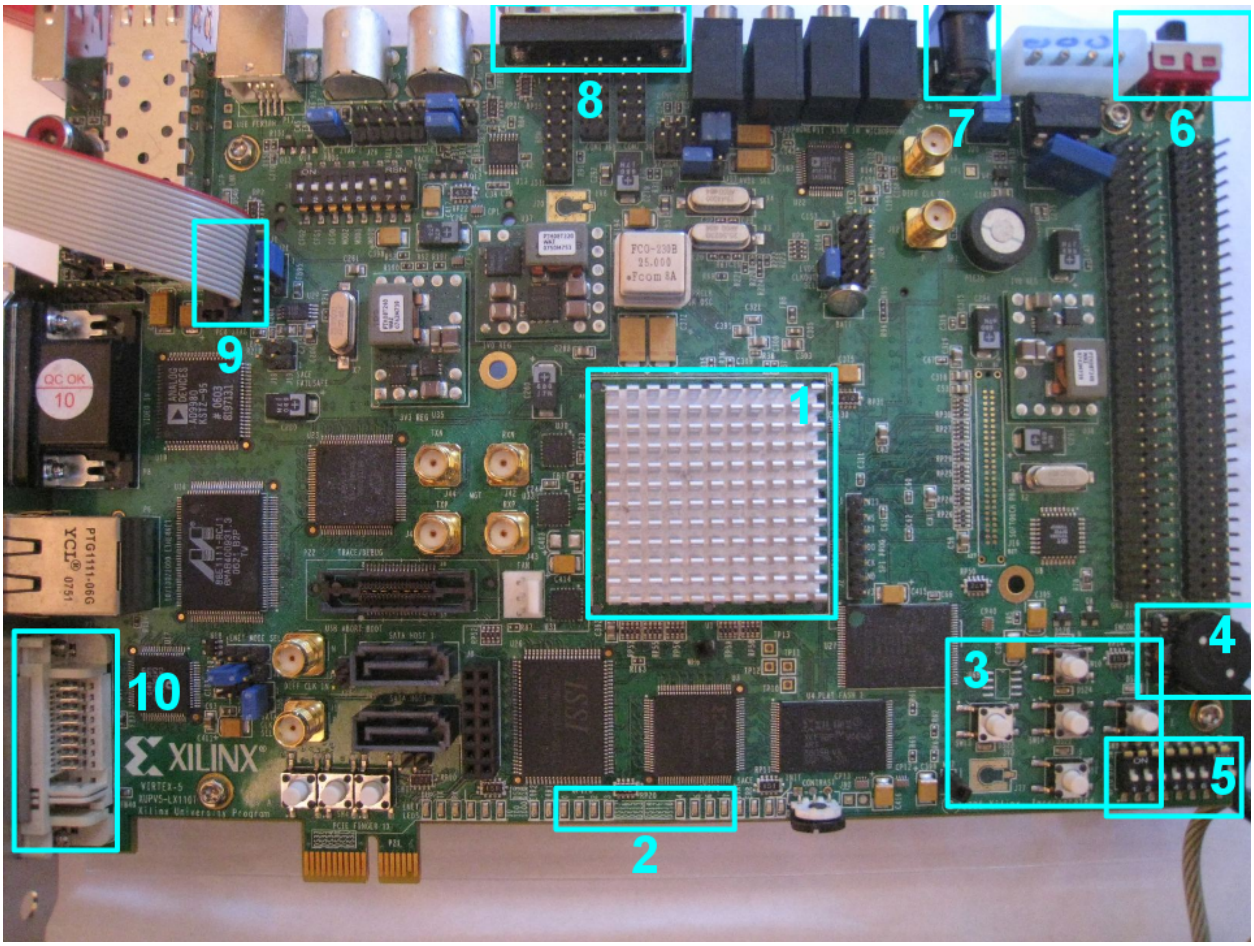
```
git clone git@github.com:EECS150/fa18_fpga_labs.git
```

Whenever a new lab is released, you should only need to do a `git pull` to retrieve the new files. Furthermore, if there are any updates to the labs, `git pull` will fetch the changes and merge them in.

For now, you will only have pull access to this repository. If you make any local commits, you will not be able to push them to the remote server. Later on, each team will receive their own private repo for the project, and you will be able to push and pull from that.

4 Our Development Platform - Xilinx ML505

For the labs in this class, we will be using the Xilinx XUPV5-LX110T development board which is built on the ML505 evaluation platform. Our development board is a printed circuit board that contains a Virtex-5 FPGA along with a host of peripheral ICs and connections. The development board makes it easy to program the FPGA and allows us to experiment with different peripherals. The following image identifies important parts of the board:



1. Virtex-5 FPGA (covered by heat sink). It is connected to the peripheral ICs and I/O connectors via PCB traces.
2. GPIO LEDs, numbered 0-7
3. North, East, South, West, Center user push buttons, each with a corresponding LED
4. Rotary encoder (a wheel we can rotate clockwise or counterclockwise and push inward towards the board)
5. GPIO DIP (dual-inline package) switches, numbered 1-8 (but referred to 0-7 in code)
6. Board power switch (toggle for a full board reset, after which you will have to reprogram the FPGA)
7. Power connector; the power cable on many of these boards is sensitive to movement and can come loose easily. Make sure you seat the power cable properly.
8. Serial port (for an on-FPGA UART)
9. JTAG header (used to program the FPGA, the Xilinx programmer is connected to this header)
10. DVI-I connector (for video output)

You will also notice a device sitting to the left of the development board that looks like this:

This is a Xilinx FPGA programmer that connects to your workstation (desktop computer) over USB to receive a compiled bitstream file which it then sends to the development board and the FPGA over the JTAG interface. Before you run `make impact` to send your design to the FPGA, make sure that the LED on the programmer is glowing green. If it is glowing yellow or not glowing at all, make sure that the wires are plugged in tightly and that the development board is powered on.

5 The FPGA - Xilinx Virtex-5 LX110T

To help you become familiar with the FPGA that you will be working with through the semester, please skim Chapter 5: Configurable Logic Blocks (pages 171-179 about SLICEs and pages 193-197 about multiplexing) of the Virtex-5 User Guide (also found under `fa18_fpga_labs/docs`) and answer the following questions (you should be able to discuss your answers for checkoff):

5.1 Checkoff Questions

1. How many SLICEs are in a single CLB?
2. How many inputs do each of the LUTs on a Virtex-5 LX110T FPGA have?
3. How many LUTs does the LX110T have?
4. How do you implement logic functions of 7 inputs in a single SLICEL? How about 8? Draw a high-level circuit diagram to show how the implementation would look. Be specific about the elements (LUTs, muxes) that are used.
5. What is the difference between a SLICEL and a SLICEM?

6 Overview of the FPGA Build Toolchain

Before we begin the lab, we should familiarize ourselves with the CAD (computer aided design) tools that translate HDL (Verilog) into a working circuit on the FPGA. These tools will pass your design through several stages, each one bringing it closer to a concrete implementation.

Looking at the directory structure of the `lab2` folder, you can see two folders, `src` and `cfg`. You will also find a `Makefile`. To execute the entire toolchain, run `make` in the `lab2` folder. The `Makefile` delegates to another `Makefile` that resides in the `cfg` folder. In the `cfg` folder you will

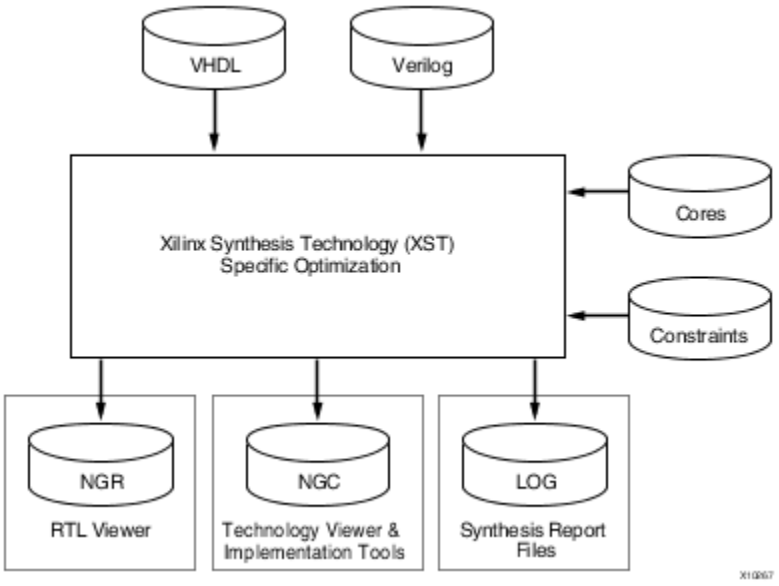
also find other files that are used by tools during the build process. We will discuss every part of the toolchain.

6.1 Synthesis with XST

The synthesis tool (in this case of this class, Xilinx Synthesis Tool (xst)) is the first program that processes your design. Among other tasks, it is responsible for the process of transforming the primitive gates and flip-flops that you wrote in Verilog into LUTs and other primitive FPGA elements.

For example, if you described a circuit composed of many gates, but ultimately of 6 inputs and 1 output, xst will map your circuit down to a single 6-LUT. Likewise, if you described a flip-flop it will be mapped to a specific type of flip-flop which actually exists on the FPGA.

The following figure shows the flow of files through XST.



XST takes in Verilog and/or VHDL files to be parsed and synthesized, "Cores" which are pre-built circuits designed by Xilinx, and "Constraints" (an XCF file). In this lab we don't supply any cores or constraints to xst.

XST produces log output, an NGR file which can be viewed with Xilinx ISE (as we will see soon), and a NGC netlist file. The NGC is a text file that contains a list of primitive components to be used in the FPGA circuit and a description of how they are connected.

6.2 Translation and Mapping with NGDBuild and Map

The tools that perform translation and mapping are NGDBuild and Map respectively. These tools take the output of the synthesis tool (a generic netlist) and translates each LUT and primitive FPGA element to an equivalent on the specific Xilinx vc5vlx110t FPGAs we have in the lab.

The translation tool merges all the input netlists and design constraint information, and outputs a Xilinx Native Generic Database (NGD) file. NGDBuild takes the UCF (User Constraints File) and the NGC (from XST) files as inputs.

The mapping tool maps the logic defined by an NGD file into FPGA elements such as CLBs (configurable logic blocks) and IOBs (input/output blocks). The Map tool takes in the NGD file produced by the translation tool and produces a Xilinx Native Circuit Description (NCD) file.

6.2.1 User Constraints File (UCF)

We will take a small detour here to cover what a UCF is and how to add top-level signal connections to it.

A user constraints file is passed to the translation tool (NGDBuild) and it describes the top-level pin assignments and timing constraints. When you opened `ml505top.v` in Lab 1, you noticed that your top-level Verilog module received several input and output signals. These signals were `input [7:0] GPIO_DIP` and `output [7:0] GPIO_LED`. You added some Verilog gate primitives to drive the outputs with some logic operations done to the inputs. But how do the tools know where those signals come from? That's what the UCF is for.

Open the UCF `lab1/ml505top.ucf` for Lab 1 and take a look. Here you can see where the `GPIO_DIP` and `GPIO_LED` nets come from. Here is the syntax used to declare a top-level signal that you can sense and/or drive from your top-level Verilog module.

```
NET (net name)<bit index> LOC="(FPGA pin number)"
NET (net name)<bit index> IOSTANDARD="(voltage level)"
```

The (net name) is the signal name that is presented to your top-level Verilog module. The bit index can be set optionally to create a multi-bit signal. The LOC defines what pin coming out of the FPGA contains that signal. All the pins coming out of the FPGA's package are labeled, and this is how we can tap or drive signals from a particular pin. The second line defines an IOSTANDARD for a given net; this is a statement of the signaling standard (FPGA IO driver type) to be used for a given pin. Here is an example for a `GPIO_LED`

```
NET GPIO_LED<0> LOC="H18";
NET GPIO_LED<0> IOSTANDARD="LVCMOS25";
```

These 2 lines give you access to a net called `GPIO_LED` in your top-level Verilog module. This net is connected to the `H18` pin coming out of the FPGA which is routed on the PCB to LED0 on the board using a trace.

Note that this declaration doesn't specify whether the net is an input or output; that is defined in your Verilog top-level module port declaration.

On the ML505 development board, we can utilize what is called a master UCF file to make adding signals to your design easy. This master UCF file is used in conjunction with the ML505 user guide and ML505 schematic to figure out what peripheral connections you want brought into your FPGA design. You can find these three files in the `fa18_fpga_labs/docs` directory. We will discuss how to use these files in the design exercise in this lab.

6.3 Place and Route with PAR

Now we resume where we left off after the mapping tool. The map tool's NCD output file is fed into the Place and Route tool which is called PAR. PAR places and routes the design that was generated by Map, and it outputs another NCD file with placement and routing information. This process is often the most time consuming of any of the steps in the toolchain; the algorithms used for placement and routing are sophisticated and have long run times.

6.4 Bitstream Generation with BitGen

The fully placed and routed design from PAR as a NCD file is now ready to be translated into another file that the Xilinx FPGA programmer can understand. We use a tool called BitGen to perform the generation of the bitstream that is sent to the FPGA. This is the last step in the FPGA build process, and it produces a .bit file which can be uploaded to the FPGA via the programmer.

6.5 Timing Analysis with TRCE

The `Makefile` we use in this class performs an additional step after running BitGen. To verify that our design met all timing requirements and to see a timing analysis, we use a tool called Trace (TRCE) which takes in output files generated by PAR and produces a timing report. It will let you know what is the maximum clock speed your design can operate at reliably.

6.6 Report Generation with ISE

A very important precaution to take after running each step of the toolchain is to verify that there are no errors or warnings that a given tool produced. We use a program called xreport which ships with Xilinx ISE to produce a report detailing the status of each build tool. To run this tool, execute `make report` in the `lab2` directory. The tool will give you all the warnings and errors emitted by each tool in a GUI. It will also report the resource usage of your design.

6.7 FPGA Programming with iMPACT

Finally, to send the bitstream file you generated with BitGen to the FPGA, we use a tool called iMPACT. To execute this tool run `make impact` in the `lab2` directory after the regular `make` process has completed and succeeded without any errors. iMPACT will send your bitstream file to the FPGA programmer over USB, and the FPGA programmer will send the bitstream to the FPGA over JTAG. After this tool runs, your design will be configured and active on the FPGA.

6.8 Toolchain Conclusion

This section was information dense. Don't worry about understanding the internals of each tool and the exact file formats they work with. Just understand what each step of the toolchain does

at a high level and you will be good for this class. We use all these tools regularly, but executing them is handled by the staff provided `Makefile`.

7 Your First FPGA Design

Throughout the semester, you will build increasingly complex designs using Verilog, a widely used hardware description language (HDL). For this lab, you will use basic Verilog to describe a simple digital circuit.

Now that you have cloned the `fa18_fpga_labs` repository, you can `cd` to the `fa18_fpga_labs/lab1` directory to see this lab's skeleton files. You will note that there is a `src` directory, a `cfg` directory, and a Makefile.

The `cfg` directory contains files that are used by the FPGA toolchain. We will learn about all the build tools in the toolchain and their configurations in the next lab, but for now you can ignore the `cfg` directory and its contents.

The `src` directory contains files you can edit that describe the circuit you want to create on the FPGA. There are two files in this directory. `ml505top.v` is a Verilog source file that represents the top-level of your circuit and it has access to the signals that come in and out of the FPGA chip. `ml505top.ucf` is known as a User Constraints File and it describes the physical FPGA pin-mappings to signals in your top-level Verilog source file. For now, don't concern yourself with the details of the UCF file, as that will be covered in detail in the next lab. Open the `ml505top.v` file in your preferred text editor (vim, emacs, nano, gedit, sublime, etc.) to get started.

This file contains a Verilog module description which specifies which signals are inputs into the module and what signals are outputs.

The `GPIO_DIP` input is a signal that is 8 bits wide (as indicated by the [7:0] width descriptor). This input signal represents the logic signals coming from the DIP switches on the bottom right side of the FPGA development board at your workstation. You should inspect your board to find these switches and confirm that there are 8 switches.

The `GPIO_LED` output is a signal that is also 8 bits wide (as indicated by the [7:0] width descriptor). This output signal represents the logic signals coming out of the FPGA and going into the bank of LEDs at the bottom center of the FPGA development board. You should inspect your board to find these LEDs and confirm that there are 8 LEDs.

In this file, we can describe how the DIP switches and the LEDs on the board are connected through the FPGA. There is one line of code that describes an AND gate that takes the values of the first 2 DIP switches, ANDs them together, and sends that signal out to the first LED. Let's put this digital circuit on the FPGA!

In the `lab1` directory, execute the command `make` in your terminal. This will execute the default build process as defined by the Makefile. This might take a few minutes to complete. Once the process has completed, execute the command `make impact` in your terminal to send the compiled digital circuit to the FPGA.

Now give it a try. Physically toggle the first two DIP switches on the board and watch as the first LED lights up only when both switches are toggled high. Go ahead and extend this example with more AND or other gates to see them in action!

There is no checkoff for this lab.