EECS 151/251A FPGA Lab
Lab 0: Getting Set Up and Familiarizing Yourself with Tools

Prof. John Wawrzynek, Nicholas Weaver
TAs: Arya Reais-Parsi, Taehwan Kim
Department of Electrical Engineering and Computer Sciences
College of Engineering, University of California, Berkeley

1 Setting Up Accounts

1.1 Course website and Piazza

If you are enrolled in this class, you should have access to the EECS 151/251A home page (`https://inst.eecs.berkeley.edu/~eecs151`) already. If you don't have access, please let a TA or the instructor know immediately so we can resolve the issue. You should also register for a Piazza account and enroll in the EECS 151/251A class as soon as possible (`http://piazza.com/berkeley/spring2018/eecs151251a`). We will be using Piazza to make announcements and as a discussion forum for this class and for the labs. The course website lists helpful resources, the course outline, and material for lectures, labs, and homework.

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 Arya (the TA) an email with your Github account username and email and your class account login (eecs151-xxx). Arya's email address is aryap@berkeley.edu.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. 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/fpga_labs_sp18.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 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 `fpga_labs_sp18` repository, you can `cd` to the `fpga_labs_sp18/lab0` directory to see this lab's skeleton files. You will note that there is a `lab0.srcs` directory and a `lab0.xpr` file.

Past versions of this course used an FPGA development toolchain from the Xilinx ISE Design Suite. The modern alternative, and that which we are now using for this course, is the Xilinx Vivado Design Suite ("Vivado" for short). We will initially use Vivado's Integrated Development Environment (IDE) instead of any command-line tools, though you will eventually see that the framework (especially with Pynq) is ripe for automation with TCL and Python scripting.

The lab skeleton files include the project meta data file, `.xpr`, a Verilog source file for a simple top-level module, `ml505top.v`, and a constraints file, `ml505top.xdc`.

HDL source files like `ml505top.v` (where the HDL is Verilog) describe the circuit that you want to create on the FPGA. `ml505top.v` describes a circuit that is the *top-level* of your circuit: it has access to the signals that come into and out of the FPGA chip. Constraints files, such as `ml505top.xdc`, allow the engineer (you!) to tailor specific properties of the synthesized design to how they wish to use their specific chip. This includes the crucial mapping between FPGA input/output pins and signal names used in circuit descriptions. We'll cover more on constraints files in the next lab, so don't worry about the details too much just yet.

4.1 Set up your Pynq-Z1

1. Plug in the power adaptor to provide mains power.

2. Insert the factory-imaged SD card to provide a boot OS for the onboard ARM (mostly for use later).

3. Connect the USB interface to a spare USB port on your workstation.

4. Turn the board on.

4.2 Open the Lab 0 project in the Vivado Design Suite

In our Centos environment, press Alt-F2 to bring up a command dialog. Type the full path to the `vivado` binary to execute it:

`/opt/Xilinx/Vivado/current/bin/vivado`

(You can also run this from a terminal or create a Desktop shortcut.)

Once in Vivado, open up the `lab0/lab0.xpr` project file. Look around the environment to try and get a feel for the GUI. Open up the `lab0/lab0.srcs/sources_1/new/ml505top.v` source file. Again, this file contains a Verilog module description which specifies which signals are inputs into the module and which signals are outputs.

The `BUTTONS` input is a signal that is 4 bits wide (as indicated by the [3:0] width descriptor). This input signal represents the logic signals coming from the momentary push-button switches on the bottom right side of your Pynq-Z1 board. You should inspect your board to find these switches and confirm that there are 4 of them. Another basic input signal is `SWITCHES`, which is 2 bits wide (as indicated by the [1:0] descriptor). Each of these two signals represents the slide switches on the Pynq-Z1, located just to the left of the momentary switches (look for SW0 and SW1).

The `LEDS` output is a signal that is 6 bits wide (as indicated by the [5:0] width descriptor). This output signal represents the logic signals coming out of the FPGA and going into the bank of LEDs at the bottom right of the Pynq-Z1, just above the buttons. Almost. There are only 4 LEDs there; 2 more are tri-color LEDs located just above the slide switches in the middle.

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

4.3 Synthesize and Program

1. In Vivado, locate the *Flow Navigator* pane to the left of the screen. Near the bottom, under *Program and Debug*, click *Generate Bitstream*. Accept the default settings and wait. This should invoke the dependent steps in the flow: *Synthesis* and *Implementation* (among other things).

   - Selecting *Project Manager* will give you a nice overview of the progress of various background steps while this happens.

   - So will watching the *Messages* and *Logs* output.

2. When the synthesis and bitstream generation is done, select *Open Hardware Manager* and connect to your FPGA.

   - If you haven't before, or the hardware manager says no devices are connected, select *Menu → Open New Target*

   - You should see `xilinx_tcf` listed under Harware Targets in the top pane. In the bottom pane, two entries: `arm_dap_0` and `xc7z020_1`. That's good. *Next → Finish*.

3. Back in the *Flow Navigator* on the left, under *Program and Debug*, select *Program Device*. The only option to program will be the FPGA, `xc7z020_1`. The default bitstream file path should work too.

4. See if it worked! What happens when you push the BTN0 button? What about when you change SW0? Both?

Go ahead and extend this example with more AND or other gates to see them in action!

There is no checkoff for this lab.

5 References and Resources

A very handy overview of what your Pynq-Z1 board can do is published by Digilent in the Pynq-Z1 Reference Manual: `https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/start`