

EECS 151/251A FPGA Lab

Lab 1: Getting Set Up

Prof. Sophia Shao

TAs: Harrison Liew, Charles Hong, Jingyi Xu, Kareem Ahmad, Zhenghan Lin

Department of Electrical Engineering and Computer Sciences

College of Engineering, University of California, Berkeley

1 Setting Up Accounts

1.1 Course website and Piazza

The course webpage can be found at <http://inst.eecs.berkeley.edu/~eecs151/fa20/> and includes material for lectures, labs, and homework. You should register for a Piazza account and enroll in the EECS 151/251A class as soon as possible (<https://piazza.com/class/kdgqfut9t83k3>). We will be using Piazza to make announcements and as a discussion forum for this class and for the labs.

1.2 Submitting Pre-Semester Survey

First, make sure you have submitted the [introductory survey](#) so we can help accommodate your needs.

1.3 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.

1.3.1 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.

1.4 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.

1.5 How to Login to the Lab Workstations From Your Laptop

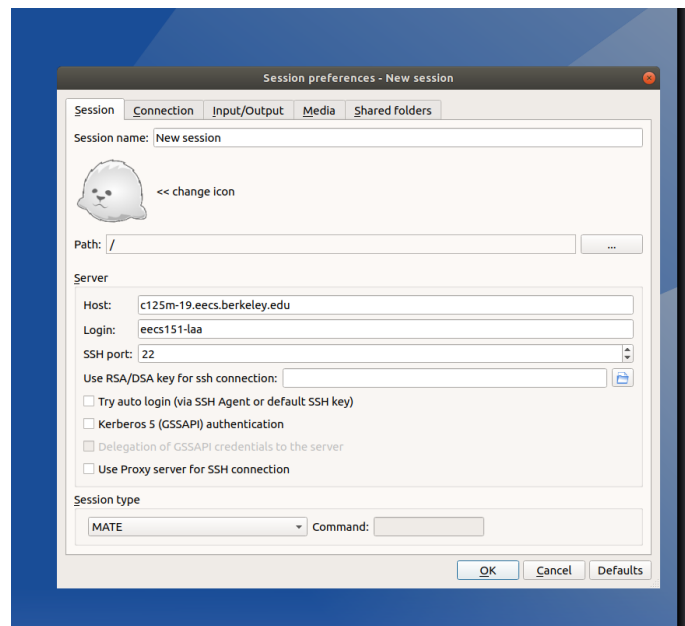
The workstations used for this class are `c125m-1.eecs.berkeley.edu` through `c125m-19.eecs.berkeley.edu`, and are physically located in Cory 125. You can access all of these machines remotely through SSH. Others such as `eda-1.eecs.berkeley.edu` through `eda-8.eecs.berkeley.edu` are also available for remote login. Not all lab workstations will necessarily be available at a given time, so try a different one if you're having trouble connecting.

Login to the lab machines by SSHing to them with your class account `eecs151-xxx`.

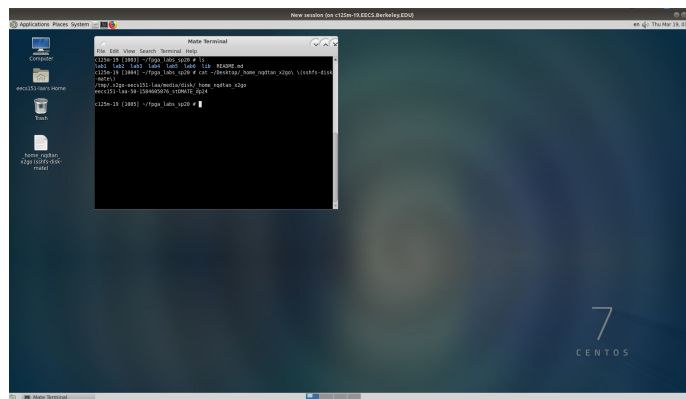
```
ssh eeecs151-xxx@c125m-1.eecs.berkeley.edu
```

For any software that requires a graphical user interface (GUI), you should login to the lab machines using x2go. We will be using it for this lab. Install it here: <https://wiki.x2go.org/doku.php>.

Once x2go is installed, create a connection with the following settings to get connected to the lab machines:



Once logged in using your class account and password, you should see a CentOS Linux desktop environment like the one below. This desktop is running on the lab machine of your choice and is being forwarded to you by x2go.



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. 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_fa20.git
```

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

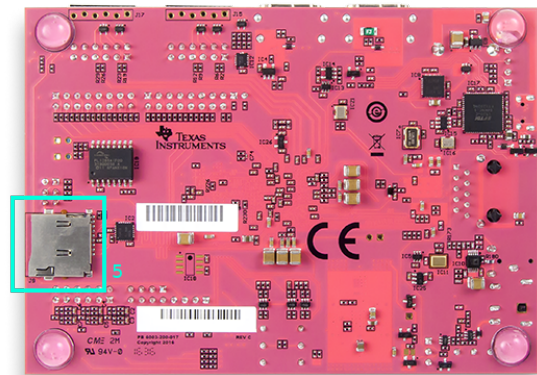
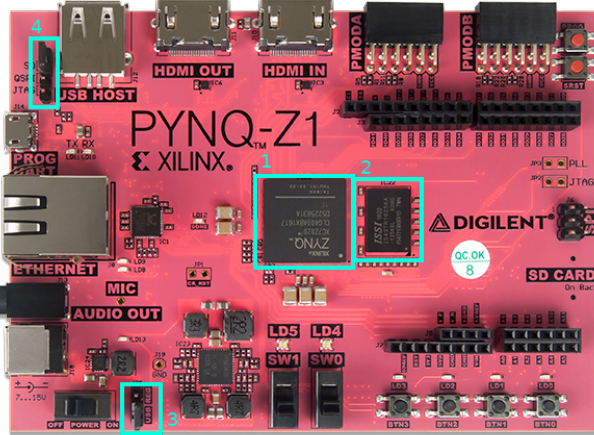
For now, you will only have pull access to this repository. If you create your own repository to store your code, make sure it's set to private to keep the code to yourself. Later on, each team will receive their own private repo for the final project, and you will be able to push and pull from that.

4 Our Development Platform - Xilinx Pynq-Z1

For the labs in this class, we will be using the Xilinx Pynq-Z1 development board which is built on the Zynq development platform. Our development board is a printed circuit board that contains a Zynq-7000 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 best [reference for this board](#) is provided by Digilent. Browse the documentation there to get a feel for both what features the board has and, more importantly, what information the documentation has, should you need it later.

Being a development board, the silkscreen print clearly identifies connectors of interest. You should be able to recognize the most basic IO features on the board: GPIO LEDs, slide switches, and push-buttons. You should also be familiar with other basic elements of the board: input power socket, power switch, and the USB programming port. The following image identifies important parts of the board that may not have been obvious:



1. Zynq 7000-series FPGA. It is connected to the peripheral ICs and I/O connectors via PCB traces.
2. ISSI DRAM chip
3. Power source jumper: shorting "REG" has the board use the external power adapter as a power source; shorting "USB" has it rely on the 5 V provided by USB. The latter will work unless your design needs to power a lot of external peripherals. To be safe, we prefer that you use the external power adapter.
4. Programming mode jumper
5. SD card slot

5 Overview of the FPGA Build Toolchain

To get started using our FPGA, 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. In previous years, older evaluation platforms (the ML505) used older FPGAs (a Xilinx Virtex-5 LX110T) and an older software suite (Xilinx ISE). Although there was a GUI, we had Makefiles to invoke each subsequent program in the toolchain to carry out the complete synthesis and perform analysis.

Our new boards use Xilinx's updated design software, the Vivado Design Suite. Vivado has integrated scripting capabilities (using the Tcl language) and integration with other high-level design tools (e.g. High-Level Synthesis). The GUI itself has the disadvantage of being very manual to work with. Repeatedly changing and running parameters quickly becomes tedious. We will automate the design process later. However, for learning, the GUI has the invaluable property of guiding us through each step of the process.

5.1 Verilog source file

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.

Open up the `lab1/lab1.srcs/sources_1/new/z1top.v` source file. This file contains a Verilog module description which specifies which signals are inputs into the module and which signals are outputs. Also check the constraints file in `lab1/lab1.srcs/constrs_1/z1top.xdc`.

HDL source files like `z1top.v` (where the HDL is Verilog) describe the circuit that you want to create on the FPGA. `z1top.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 `z1top.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.

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!

5.2 Set up your Pynq-Z1

1. Plug in the power adaptor to provide mains power.
2. Connect the USB interface to a spare USB port on your device.
3. Turn the board on.

5.3 Open the Lab 1 project in the Vivado Design Suite

Now that you have cloned the `fpga_labs_fa20` repository, you can `cd` to the `fpga_labs_fa20/lab1` directory to see this lab's skeleton files. You will note that there is a `lab1.srcs` (sources) directory and a `lab1.xpr` (project) file.

In the lab workstation 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 `lab1/lab1.xpr` project file. Look around the environment to try and get a feel for the GUI.

5.4 Synthesis

To run the synthesis step in the Vivado Design Suite (that is, turn your HDL into combinational and sequential logic), select *Run Synthesis* in the *Flow Navigator* pane to the left other interface. If this has been run before, the synthesized design can be inspected by selecting *Open Synthesized Design*. Select **Schematic** under “Open Synthesized Design” on the left toolbar to see a circuit schematic.

5.5 Implementation

The implementation step in the Vivado GUI is equivalent to the translation, mapping and place and route steps in the manual pipeline. Again, this takes the logical circuit synthesized previously and maps it to the physical logic devices our particular FPGA actually has. Select *Run Implementation* in the *Flow Navigator* to run it, then select *Open Implementation* to inspect its outputs.

5.6 Xilinx Design Constraints (XDC)

How do we connect one of our signals to a physical device? How do we specify special properties of the circuit that might matter for correctness and timing? The Xilinx Design Constraints file (with the `.xdc` extension) specifies timing information and pin placement. More information can be found on page 22 of the [Vivado migration guide](#).

Take a look at this snippet from the XDC inside `lab1/lab1.srscs/constrs_1/new/z1top.xdc`:

```
set_property -dict { PACKAGE_PIN L19    IOSTANDARD LVCMOS33 } [get_ports { BUTTONS[3] }];
```

This syntax assigns the properties `PACKAGE_PIN` and `IOSTANDARD` with the values `L19` and `LVCMOS33` (respectively) to the port `BUTTONS[3]`, a signal we defined in our Verilog source. Each of these properties has a separate consequence in the synthesis process:

- The pin to which the `BUTTONS[3]` signal should be connected to the physical pin `L19` on the FPGA package.
- The logic convention (maximum voltage, what ranges constitute low and high, etc) for that port will be `LVCMOS33`.

5.7 Bitstream generation

To generate the programming file our FPGA will understand, we invoke *Generate Bitstream* in the *Flow Navigator*.

5.8 Timing Analysis

A timing analysis report can be generated under *Synthesis* in *Flow Navigator*, by expanding *Open Synthesized Design* and selecting *Report Timing Summary*.

5.9 Design Reports

Reports are automatically generated at each step in the build flow. You should be able to discover them under each of the expanded stages in the *Flow Navigator*. The *Project Summary* window (under the *Window* menu) presents a nice summary of the reports generated through each step. You will see some examples later in the lab.

6 Setting Up the Vivado Hardware Server

If we were physically using the lab machines, we'd now be able to program the bitstream onto our FPGA with a couple clicks. However, since we're using the lab machines remotely, we'll set up our computers as a relay so that the lab machines can program our locally connected FPGAs through the Internet. We do this by running Vivado Hardware Server, a subcomponent of the Vivado software, locally, and setting up an SSH reverse tunnel. We recommend using the VM that we have set up for you, but if you have any difficulties it is also possible to install the Vivado Hardware Server directly onto your computer if you're using Linux or Windows (see Appendix A).

6.1 Installing the VM

For convenience, we've prepared a virtual machine image with Vivado Hardware Server already installed. There are two versions.

- GUI: https://drive.google.com/file/d/1S0duf_oHfXs9fImsqNypIvR-nmrZuDca/view?usp=sharing
- Command line-only: <https://drive.google.com/file/d/133YZzW4nnEgabDuhmfxPH3ncQPwQPVP/view?usp=sharing>

The command line-only version requires less computational resources, but is slightly less convenient. We'll be using the VM in a limited capacity (i.e. not for coding), so choose this version if performance is important for you.

To use the VM, install [VirtualBox](#). Import the downloaded VM image and start it. The VM runs CentOS 7, like the lab computers. The username is `eeecs151` and the password is `eeecs151`.

The VM sets up USB filtering to recognize the devices we use for this class. For VirtualBox on Linux, if the USB devices (FPGA and UART) don't map into the VM, then be sure your account is in the "vboxusers" group (and log-off and back-on). For example, Ubuntu users can use the following command:

```
sudo usermod -a -G vboxusers $USER
```

6.2 Running the Hardware Server

Run the executable at `/opt/Xilinx/HWSRVR/2019.1/bin/hw_server`.

In the VM, `/opt/Xilinx/HWSRVR/2019.1/bin` has been added to the PATH, so you can run it with just the terminal command `hw_server`.

6.3 Setting Up an SSH Reverse Tunnel

We now have to set up a connection through the Internet from our lab workstation to the machine running the hardware server. If you are running the Hardware Server in a VM, do this in the VM.

In order to prevent collisions between multiple students using the same lab server, each student has been assigned a unique port number to use. Check the following sheet for your port number: <https://docs.google.com/spreadsheets/d/1XFvLxtDk7QIIW2it2e4g4WEiXLvNRmDY4M9sCI3jGE/edit#gid=829006917>.

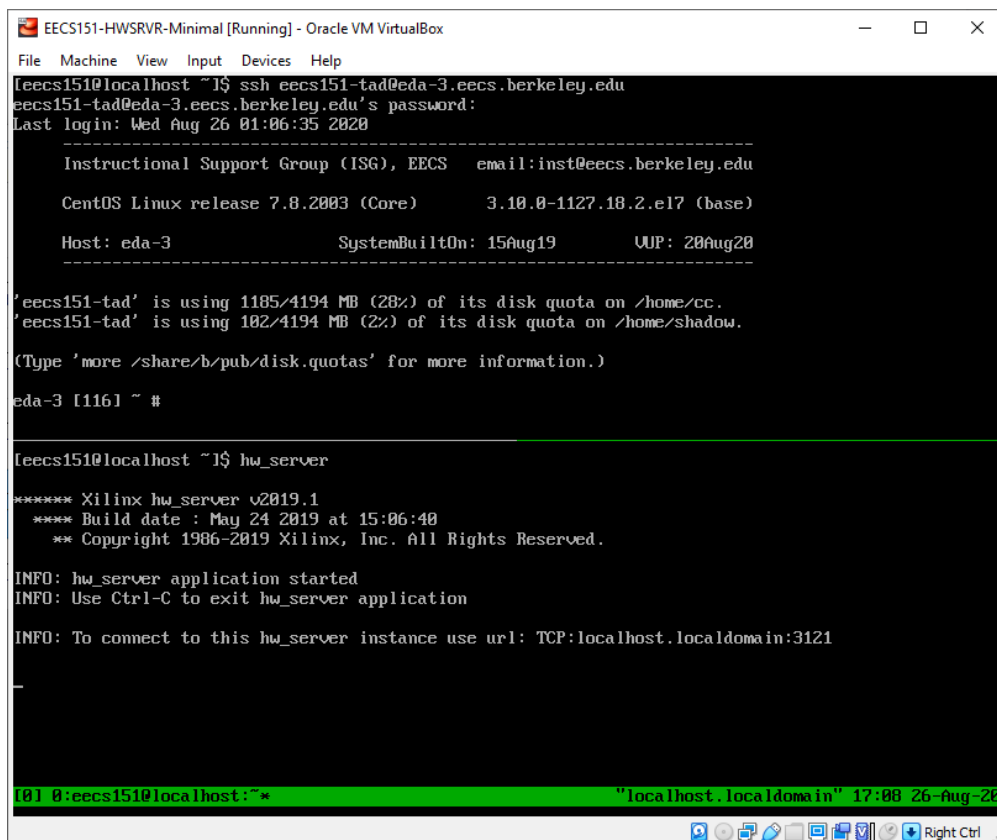
If you're not on this list, ask a TA.

In a new terminal, run the following, where `<your port>` is your port number from the sheet, `eeecs151-xxx` is your class account and `c125m-x` is the lab machine you're using:

```
ssh -R <your port>:localhost:3121 eeecs151-xxx@c125m-x.eecs.berkeley.edu
```

The `-R` flag indicates that this is a reverse tunnel connecting port `<your port>` of the lab machine to port 3121 of our local machine. If you only have a command line interface, `tmux` is a useful linux tool that can help you open multiple terminal sessions at once. The following tutorial explains how to use `tmux` to open two different terminals, in which you can run the above two commands: <https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/>

Here's an example of running the two commands side by side in `tmux`.



```
EECS151-HWSRVR-Minimal [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[eeecs151@localhost ~]$ ssh eeecs151-tad@eda-3.eecs.berkeley.edu
eeecs151-tad@eda-3.eecs.berkeley.edu's password:
Last login: Wed Aug 26 01:06:35 2020
-----
Instructional Support Group (ISG), EECS    email:inst@eecs.berkeley.edu
CentOS Linux release 7.8.2003 (Core)      3.10.0-1127.18.2.el7 (base)
Host: eda-3                               System built on: 15Aug19      UUP: 20Aug20
-----
'eeecs151-tad' is using 1185/4194 MB (28%) of its disk quota on /home/cc.
'eeecs151-tad' is using 102/4194 MB (2%) of its disk quota on /home/shadow.
(Type 'more /share/b/pub/disk.quotas' for more information.)
eda-3 [1161 ~ #

[eeecs151@localhost ~]$ hw_server
***** Xilinx hw_server v2019.1
***** Build date : May 24 2019 at 15:06:40
** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: hw_server application started
INFO: Use Ctrl-C to exit hw_server application

INFO: To connect to this hw_server instance use url: TCP:localhost.localdomain:3121

[0] 0:eeecs151@localhost:~* "localhost.localdomain" 17:08 26-Aug-20
```

6.4 Programming the FPGA

Now, in Vivado on the lab machine, *Open Hardware Manager* and connect to your FPGA. To do this, select *Open Target* → *Open New Target*. In the *Open New Hardware Target* dialog, choose to connect to a *Remote server* at Host name 127.0.0.1 and your assigned port number from before.

You should see `xilinx_tcf` listed under Hardware Targets in the top pane. In the bottom pane, two entries: `arm_dap_0` and `xc7z020_1`. You're connected! *Next* → *Finish*, then *Program Device*.

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

7 Checkoff

No checkoff this week! If you don't have your FPGA board yet, make sure you have all the setup from this lab done before next lab.

Appendices

A Installing Vivado Hardware Server Locally

We recommend that you use the provided VM, but it is also possible for Linux and Windows users to install Vivado Hardware Server natively to their computer.

A.1 Linux

First, create an install area:

```
sudo mkdir /opt/Xilinx
sudo chmod 775 /opt/Xilinx
sudo chown `whoami`:`whoami` /opt/Xilinx
```

Download the ‘Vivado 2019.1 64-bit Hardware Server for Linux’ (or 32-bit, if necessary) TAR file from [Xilinx](#). You will need to create a Xilinx account.

cd into the extracted directory and run the `xsetup` executable.

During the install process, specify `/opt/Xilinx` as the install directory.

After it’s done, install the Digilent drivers to program the Pynq over USB-JTAG.

```
cd /opt/Xilinx/HWSRVR/2019.1/data/xicom/cable_drivers/lin64/install_script/install_drivers
sudo ./install_drivers
```

A.2 Windows

Download the ‘Vivado 2019.1 64-bit Hardware Server for Windows’ (or 32-bit, if necessary) GZIP file from [Xilinx](#). You will need to create a Xilinx account. Extract using a file archiver like [7-Zip](#) and run the included `xsetup.exe`.

To run the HW server, run the batch file at `<install location>\HWSRVR\2019.1\bin\hw_server.bat`.

To set up the `ssh` reverse tunnel, use [PuTTY](#). Here’s a tutorial on how to set up a reverse tunnel using PuTTY: <https://vincetocco.com/how-to-setup-a-reverse-tunnel-with-putty/>

In our case, the connection is to `eeecs151-xxx@c125m-x.eecs.berkeley.edu`, the source port is your port number, and the destination port is `localhost:3121`.

B Installing Vivado Locally

We recommend that you use Vivado on the lab machines, but if this is difficult for you, it is also possible to install the full Vivado software locally, natively on Linux/Windows or using a VM on Mac. However, it will be harder to debug any issues you might have due to differences in setup between the lab machines and your computer. Vivado can take up to around 40 GB of disk space, so make sure you have enough disk space available before installing locally.

B.1 Linux

Create an install area for Vivado:

```
sudo mkdir /opt/Xilinx
sudo chmod 775 /opt/Xilinx
sudo chown `whoami`:`whoami` /opt/Xilinx
```

Download the ‘Vivado Design Suite - HLx Editions - 2019.1 Full Product Installation’ Linux bin from [Xilinx](#). You will need to create a Xilinx account. Execute the downloaded script.

```
chmod +x Xilinx_Vivado_SDK_Web_2019.1_0524_1430_Lin64.bin
./Xilinx_Vivado_SDK_Web_2019.1_0524_1430_Lin64.bin
```

During the install process, specify `/opt/Xilinx` as the install directory. Also, you should only select the install options we’re going to use in this class to save disk space:



After it’s done, install the Digilent drivers to program the Pynq over USB-JTAG.

```
cd /opt/Xilinx/Vivado/2019.1/data/xicom/cable_drivers/lin64/install_script/install_drivers
sudo ./install_drivers
```

You also need to add PYNQ-Z1 board files to Vivado since it does not have PYNQ support initially. This allows you to set PYNQ-Z1 as the target platform when you create a Vivado project.

Download the following file https://github.com/cathalmccabe/pynq-z1_board_files/raw/master/pynq-z1.zip.

Extract and copy the pynq-z1 folder to

```
<your Vivado installation directory>/Vivado/<version>/data/boards/board_files
```

B.2 Windows

Download the ‘Vivado Design Suite - HLx Editions - 2019.1 Full Product Installation’ Windows exe from [Xilinx](#). You will need to create a Xilinx account. Execute the downloaded exe. Also, you should only select the install options we’re going to use in this class to save disk space, as described in the Linux section.

B.3 Mac

We provide a VirtualBox VM with CentOS 7 and Vivado 2019.1 WebPACK, which provides the full Vivado toolchain, installed. The download is about 10 GB and the VM expands to about 24 GB when imported into VirtualBox. The environment is similar to the lab workstations. The username is eeecs151 and the password is eeecs151.

<https://berkeley.box.com/s/s4z0ykp0tudrm9hce8fsmitpgb2khhe>

Like the Hardware Server VM, the VM sets up USB filtering to recognize PYNQ and the Pmod USB-UART connection. For Linux, take the steps in Section 6.1 if the USB devices (FPGA and UART) don’t map into the VM.

Acknowledgement

This lab is the result of the work of many EECS151/251 GSIs over the years including:

- Sp12: James Parker, Daiwei Li, Shaoyi Cheng
- Sp13: Shaoyi Cheng, Vincent Lee
- Fa14: Simon Scott, Ian Juch
- Fa15: James Martin
- Fa16: Vighnesh Iyer
- Fa17: George Alexandrov, Vighnesh Iyer, Nathan Narevsky
- Sp18: Arya Reais-Parsi, Taehwan Kim
- Fa18: Ali Moin, George Alexandrov, Andy Zhou
- Sp19: Christopher Yarp, Arya Reais-Parsi
- Fa19: Vighnesh Iyer, Rebekah Zhao, Ryan Kaveh
- Sp20: Tan Nguyen