EECS 151/251A FPGA Lab
Lab 2: Introduction to FPGA Development

Prof. Borivoje Nikolic and Prof. Sophia Shao
TAs: Cem Yalcin, Rebekah Zhao, Ryan Kaveh, Vighnesh Iyer
Department of Electrical Engineering and Computer Sciences
College of Engineering, University of California, Berkeley

1 Before You Start This Lab

Make sure that you have gone through and completed the steps involved in Lab 1. Let the TA know if you are not signed up for this class on Piazza or if you do not have a class account (eecs151-xxx), so we can get that sorted out.

To fetch the skeleton files for this lab, `cd` to the git repository (`fpga_labs_fa19`) that you had cloned in the first lab and execute the command `git pull`.

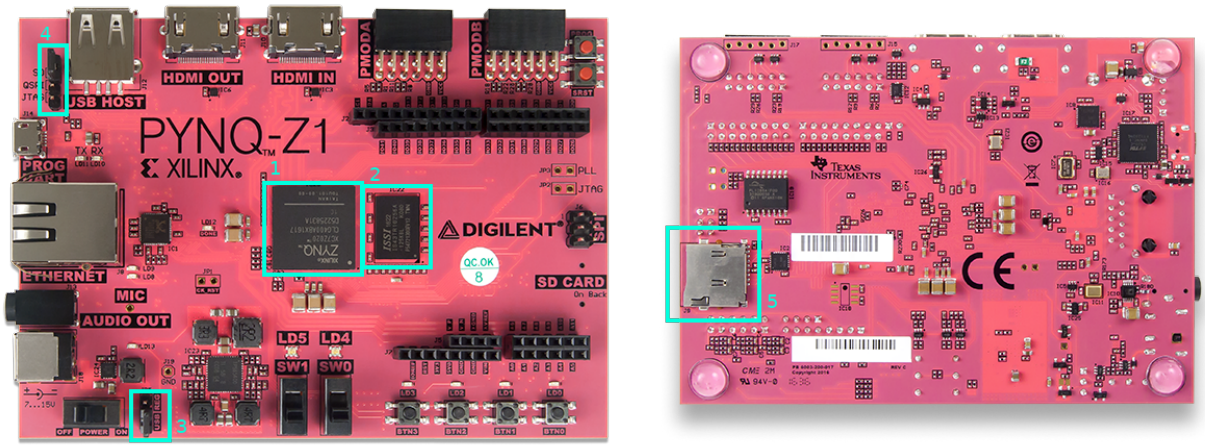
Go through the Verilog Primer Slides; you should feel somewhat comfortable with the basics of Verilog to complete this lab.

2 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. Since we have labs and power adaptors available, we avoid this.
4. Programming mode jumper
5. SD card slot

3 The FPGA - Xilinx Zynq-7000 7z020

To help you become familiar with the FPGA that you will be working with through the semester, please skim Chapter 21: Programmable Logic Description of the Technical Reference Manual and Chapter 2 of the Xilinx 7-series Configurable Logic Block User Guide. Pay particular attention to pages 15-25 on Slices and pages 40-42 on Multiplexers. You will need to answer the following questions in your lab report.

3.1 Lab Report Questions

1. How many SLICEs are in a single CLB?
2. How many inputs do each of the LUTs on a Zynq-7000 FPGA have?
3. How many LUTs does the 7z020 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?

4 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. 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 after this lab, however, for learning, the GUI has the invaluable property of guiding us through each step of the process.

4.1 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 `lab2/lab2.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 `lab2/lab2.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.

4.2 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 workstation.
3. Turn the board on.

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 Open the Lab 1 project in the Vivado Design Suite

Now that you have cloned the `fpga_labs_fa19` repository, you can `cd` to the `fpga_labs_fa19/lab2` directory to see this lab's skeleton files. You will note that there is a `lab2.srcs` (sources) directory and a `lab2.xpr` (project) file.

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 `lab2/lab2.xpr` project file. Look around the environment to try and get a feel for the GUI.

4.4 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 Synthsized Design" on the left toolbar to see a circuit schematic.

4.5 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.

4.6 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 `lab2/lab2.srcs/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.

4.7 Bitstream generation

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

4.8 Timing Analysis

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

4.9 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.

4.10 Programming the FPGA

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*.

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

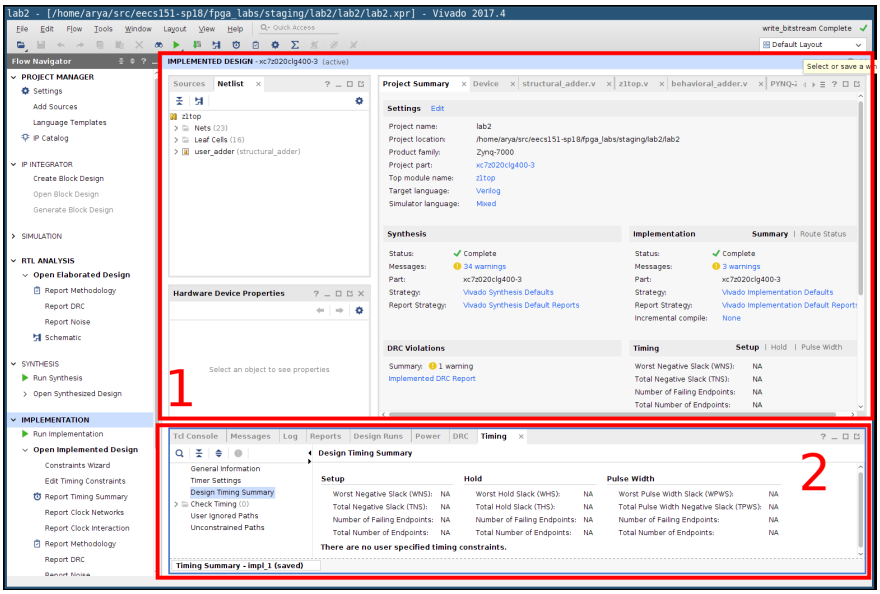
5 A Structural and Behavioral Adder Design and Inspecting the Schematic

5.1 Build a Structural 14-bit Adder

To help you with this task, please refer to the 'Code Generation with for-generate loops' slide in the Verilog Primer Slides (slide 35).

1. Open `lab2/lab2.srcs/sources_1/new/full_adder.v`; fill in the logic to produce the full adder outputs from the inputs
2. Open `structural_adder.v` and construct a ripple carry adder using the full adder cells you designed earlier and a 'for-generate loop'.
3. Finally, inspect the `z1top_adder.v` top-level module and see how your structural adder is instantiated and hooked up to the top-level signals. For now, just look at the `user_adder` instance of your structural adder. As we learned in previous lab, the basic I/O options on the Z1 board are limited. How are we managing to input two 3-bit integers?
4. Set `z1top_adder.v` as the top-level module by right-clicking `z1top_adder` in Sources → Design Sources and click 'Set as Top'.
5. Run bitstream generation as usual
6. Program the FPGA and test out the design; make sure you get correct results from your adder.

If there are any problems with your design, you can view the output report in Vivado from the *Project Summary* view. The Project Summary tab opens by default, but you can bring it up again from the *Window* menu. See box 1 in the figure for an example. At the bottom of the screen (box 2 in the same figure) you can also inspect the outputs of the individual tools that make up the pipeline; there lies bountiful debugging information should you ever need it. Unfortunately, not every log or warning message is useful, but it will serve you well to compare what outputs you do see with the relative success you have with your design.



5.2 Inspection of Structural Adder

5.2.1 Schematics and FPGA Layout

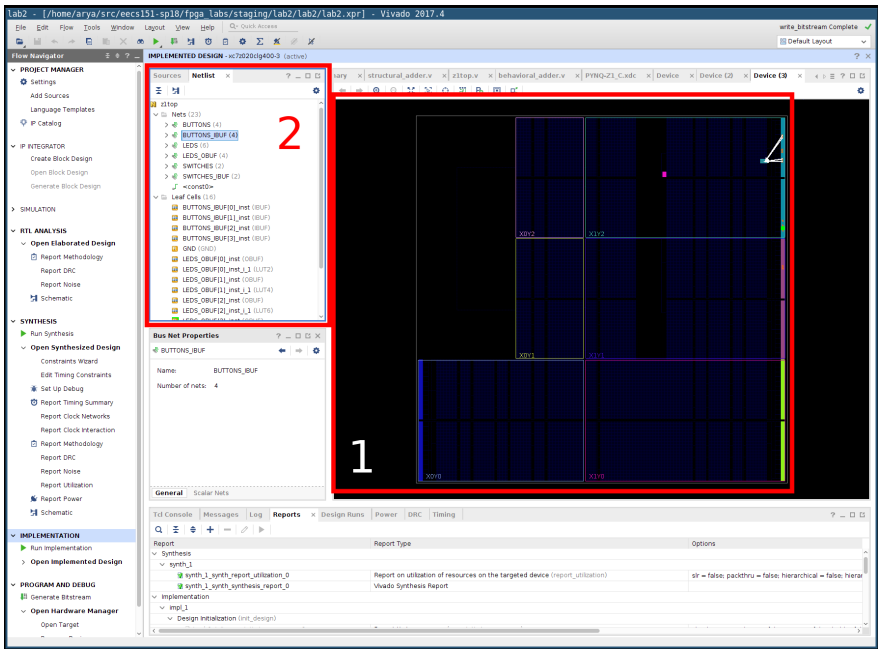
Take a look at how the Verilog you wrote mapped to the primitive components on the FPGA. Three levels of schematic are generated for you once you've run the pipeline. In the *Flow Navigator*, you can view *Schematic*s under

1. *RTL Analysis* → *Open Elaborated Design*
2. *Synthesis* → *Open Synthesized Design*
3. *Implementation* → *Open Implemented Design*

The first two will give you a fairly straightforward hierarchical block-level view of your design. You will find your circuit by drilling down into the `user_adder` module (that's the name you gave the instantiation of `structural_adder` in `z1top_adder.v`). Check to see that your structural adder module is hooked up properly and looks sane. It's ok if the wires don't appear to be connected, just hover your mouse over the endpoints on the schematic and ensure that the connections are as you expect. Take note of the primitive blocks used in your circuit.

In the RTL Analysis (1) you are viewing a visualisation of the topology your RTL describes. At this point, logic elaboration is very abstract: you'll notice that your logic is expressed in terms of the logic gates you described (XOR, AND, etc). Any logic you describe in RTL is included, even if it's disconnected. In the Synthesis schematic (2) this logic has been elaborated further into what look like FPGA elements, but still at higher layer of abstraction, and with some unused signals still present. In the final schematic of the three, Implementation (3), the schematic now shows which of the elements in your nominated chip are actually targeted. Superfluous logic has been elided from the design.

Finally, you also look at how your circuit was placed and laid out on the FPGA. Once you've run the pipeline, open *Implemented Design*, click on the *Window* menu, and select *Device*. You'll be presented with a layout of the FPGA package as in box 1 in the figure below. It'll be hard to see with a small design, but the logic elements you've ended up using with your design will be highlighted. You can highlight your own nets in the diagram to make it easier to find them by selecting a net or signal from the Netlist pane (*Window* → *Netlist*; see box 2).



Explore your design and look for the modules that you wrote. If you scroll down in the `Netlist Window` you should see various components of your logic. Some elements are mapped to LUTs: somewhere buried in their properties is the type of slice. See if you can find out which nets have been assigned to LUTs, and how they are connected. Go ahead and explore several SLICELs that implement the structural adder to see how they are connected to each other and the outputs of your circuit.

5.3 Build a Behavioral 14-bit Adder

Check out `behavioral_adder.v`. It has already been filled with the appropriate logic for you. Notice how behavioral Verilog allows you to describe the function of a circuit rather than the topology or implementation.

In `z1top_adder.v`, you can see that the `structural_adder` and the `behavioral_adder` are both instantiated in the self-test section. A module called `adder_tester` has been written for you that will check that the sums generated by both your adders are equal for all possible input combinations. If both your adders are operating identically, both RGB LEDs will light up. Verify this on your board.

5.4 Inspection of Behavioral Adder Schematics and FPGA Layout

Go through the same steps as you did for inspecting the structural adder. View the schematics at successive levels of logic elaboration and how FPGA components are connected. Record and note down any differences you see between both types of adders in the schematic and the FPGA layouts. You will be asked for some observations in the lab report.

6 Checkoff

To checkoff for this lab, have these things ready to show the TA:

1. Your programmed and functional board showing LED0 being controlled by BTN0 and SW0, as well as any modifications you made to the design.
2. Demonstrate your structural adder on the FPGA and show that the test passes.

6.1 Lab Report

Also please submit a short lab report to gradescope in which you answer/show the following things:

1. Answers for the questions in section 3.1
2. Explain the differences between the behavioral and structural adder as they are synthesized in both the high-level schematic and low-level SLICE views

In the next lab, we will simulate our digital designs in software, and create a `tone_generator` that can play square waves through the audio jack on the Pynq board.

A Installing Vivado Locally

A.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
```

A.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.

A.3 Mac

Create an Ubuntu 18.04.3 VM using Virtualbox. Use this tutorial. Allocate at least 4GB of RAM and 50GB of disk space for the VM.

Install Vivado inside the VM using the steps in the Linux section.

There's some more stuff to do like tunneling the USB device from OSX to the VM; ask a TA.

Ackowlegement

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