CS152 – Computer Architecture and Engineering
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

Compiled: 2004-09-16 for CS152, Prof. Dave Patterson, John Lazzaro
Sources: http://www.cs.berkeley.edu/~kubitron/courses/cs152-S04/handouts/homeworklab_2.html
http://www-inst.eecs.berkeley.edu/~cs150/fa04/ - various material with permission

Mini Lab 3: **Downloading to the Calinx board**

Name: _____ **Date** _____

Login: _____ **Prelab Checkoff** _____

Section: _____ **Lab Checkoff** _____

Introduction: Mini Lab 3 is the final part of a 3 part mini-lab series designed to orient you with the concepts of our CAD tool flow as well as let you get hands-on experience with industrial strength CAD tools. We will be going over how to implement your design on a Field Programmable Gate Array (FPGA) based board.

This lab is to be done individually and is due at the end of your section. You should be able to show us the steps you took, and explain what they mean.

IMPORTANT: Please read the lab report (and examine the references) and answer the following questions **before** coming to lab.

Prelab Questions:

1. **What does FPGA stand for? What is the FPGA that you will use this semester (specific part)? What is a LUT (theoretically and physically)? How many logic functions can a LUT on our part realize? How many LUTs in a Slice on our part? What other two *major* elements are in a Slice for our part? What is a CLB and how many slices does it have for our part? How many CLBs does the FPGA we use have and therefore how many total LUTs?**

2. **Give three advantages FPGAs have over ASICS. Give three weaknesses.**

3. **What is the synthesis tool we will be using? What is the product (result) of the synthesis tool fed a verilog description? Could different synthesis tools produce different results given the same input?**

4. **What is the program we use to download the design to the Xilinx chip? What type of cable is used for the programming? What is it programming physically? What interface does this cable use to program the device? Is this the only way to program the board? If not what is another way?**

Lab Questions:

1. **How many slice Flip Flops does your design use? How many occupied Slices? How many 4-input LUTs?**

2. **What is the Minimum Period for the clock in your design? Therefore what is the maximum clock frequency?**

CS152 – Computer Architecture and Engineering
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

Compiled: 2004-09-04 for CS152, Prof. Dave Patterson, John Lazzaro
Sources: http://www.cs.berkeley.edu/~kubitron/courses/cs152-S04/handouts/homeworklab_2.html
http://www-inst.eecs.berkeley.edu/~cs150/fa04/ - various material used with permission

Mini Lab 3: **Downloading to the Calinx board**

1 References

General Information

- Calinx Board - http://calinx.eecs.berkeley.edu/
- Xilinx Virtex-E Data Sheet - http://inst.eecs.berkeley.edu/~cs152/handouts/virtexE-datasheet.pdf
- CS150 FPGA Overview - http://www-inst.eecs.berkeley.edu/~cs150/fa04/Lecture/lec03.pdf
- Xilinx Web Site – http://www.xilinx.com

Computer Organization and Design: The HW/SW Interface (3rd Edition)

- Appendix B77-78 on Field Programmable Devices (FPDs)
- Appendix B77 on Field Programmable Gate Arrays (FPGAs)

Synplify Manual - http://www-inst.eecs.berkeley.edu/~cs152/handouts/local_only/synplify_ref.pdf

- Appendix H – Designing with Xilinx
- Pages 8-7 to 8-9 on Verilog Synthesis Guidelines

2 Overview

This section is meant to detail the process of actually creating and implementing your own hardware designs.

2.1 Platform

This semester you will be using the Xilinx based Calinx boards to realize your designs. This board has both the Xilinx FPGA as well as other peripherals such as an AC '97 Audio Codec, video encoder/decoder, ethernet ports, and other items such as LEDs and switches. The first stage of the process is *design entry*. You will create your designs in the Hardware Description Language (HDL) verilog (or verilog produced through schematic capture) and then use a set of CAD tools to transfer your design to this board. The next portion of the flow to the board encompasses a series of steps including *synthesis*, *place and route*, *programming*, and *verification*. The next sections give overviews into those areas.

2.2 Synthesis

Once your design is entered, the next step in the implementation path is synthesis. In our case, the function of the synthesis program is to translate the Verilog description of the circuit into an equivalent circuit comprising a set of primitive circuit components that can be directly implemented on an FPGA. In a way, the synthesis tool is almost like a compiler. Where a compiler translates to a sequence of primitive commands that can be directly executed on a processor, synthesis translates to primitive circuit components that can be directly implemented in FPGA. The final product of a synthesis tool is a netlist file, a text file that contains a list of all the instances of primitive components in the translated circuit and a description of how they are interconnected. The synthesis tool we will be using in this class is called Synplify. Note that Xilinx's software suite can also perform this synthesis procedure. The reason we use Synplify is because Synplify is an industrial strength CAD tool program. It's faster, produces better logic, and will give many more synthesis warnings—something very useful for students!

2.3 Placement, Routing

The next step in the implementation flow is to take the netlist of components generated by the synthesis tool and turn it into bits that are need to configure the LUTs, muxes, Flipflops, and other configurable resourses in the FPGA. To do that, first the primitive circuit components in the netlist need to be assigned to a specific *place* on the FPGA. For example, a 4LUT implementing the function of a 4 input NAND gate in a netlist could be implemented with any of the about 40,000 4LUTs in a Xilinx Virtex 2000E FPGA chip. Clever choice of placement will make the subsequent routing easier and result in a faster overall circuit. Once the components are placed, the proper connections must be made according to the netlist description obtained from the synthesis step. That step is called *routing*. Unlike synthesis, which only requires a set of primitive components to translate to; placement and routing are dependent upon the specific size and structure of the target FPGA chip. Due to this reliance, the FPGA vendor usually provides the placement and routing programs. Therefore, we will be using the Xilinx's Project Navigator to perform this step. The end product after placement and routing is a bit file containing the stream of bits used to configure the FPGA.

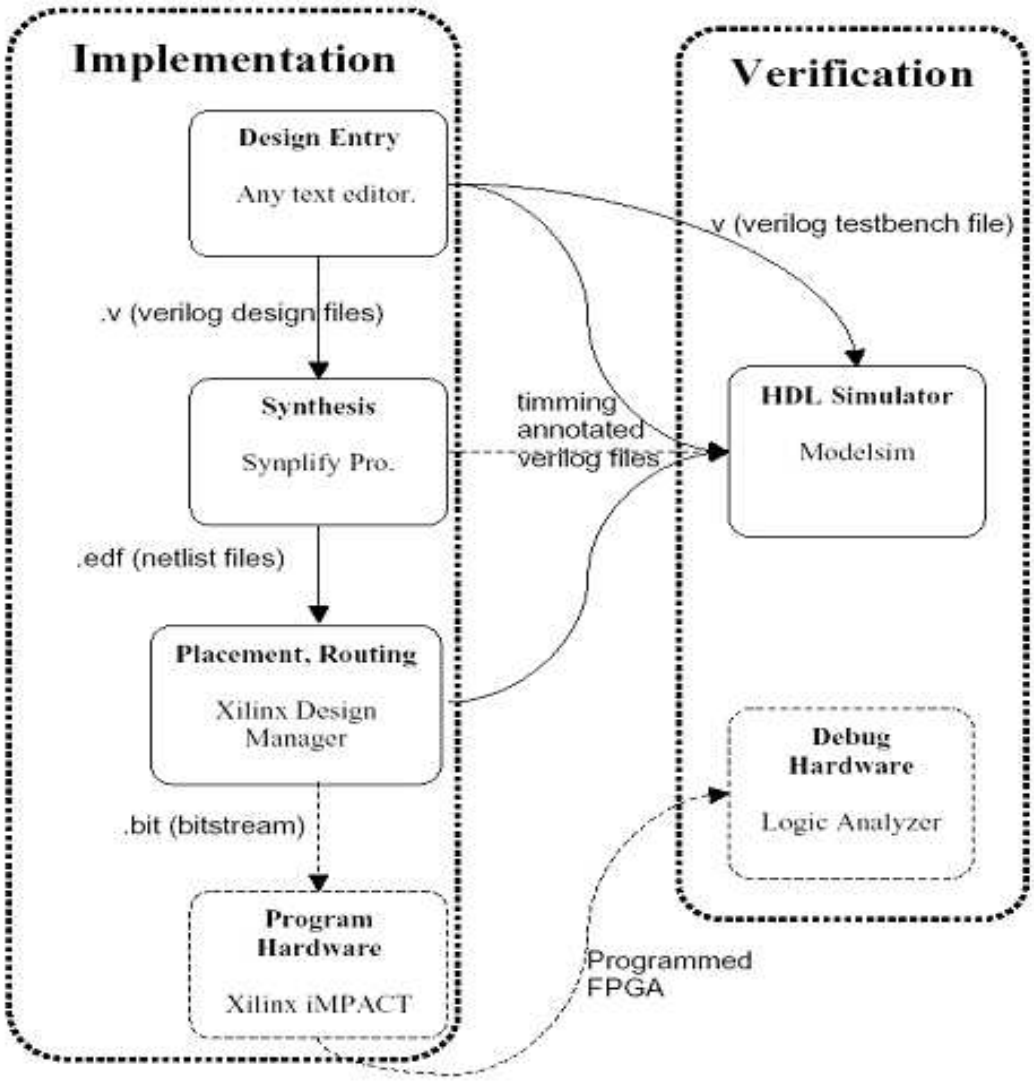
2.4 Program Hardware

The last step in the implementation flow is the simple act of transporting the configuration bits (in a *.bit file) to the FPGA. There are also many ways of doing this. For this class we will be mostly using the Parallel Cable IV along with the iMPACT software to program the board.

2.5 Verification

As you should have learned from experience, a significant part of the time and effort spent on any sizable project will be spent on debugging, and logic design is no exception. There are two ways to verify the correctness of a design: to program the FPGA with the design and check if the circuit is behaving correctly, or to run simulations of the design in software. While programming the FPGA and physically checking the functionality sounds simple, the whole tool flow requires a significant amount of time to complete, especially as your designs get larger and more complex (20 minutes towards the end of the semester!). Repeatedly tweaking the input design to fix errors would require running the flow repeatedly, a huge waste of time. In addition it can be difficult to physically observe the causes for an error on a

FPGA. For these reasons, software simulation is essential in the verification process. There are many places along the tool flow where you can use simulation to verify the correctness of your design. As you progress down the tool flow and more information about the physical implementation on the FPGA becomes available, more accurate timing simulations can be performed. **You should not attempt to verify that your design works on the board until it works in simulation!!!** To do so otherwise would just be a waste of time. In this class, we will be using ModelSim to do our verification.



CAD tool flow. Optional links are showed as dashed lines.

This figure demonstrates at a high level the design flow. Notice that while this flow relates to Xilinx hardware a similar flow is used for not only other FPGAs but also in the design of ASICs.

3 Implementing Your Multiplier Design

This is the portion of the lab where we will take the multiplier design **used in the previous Mini-lab 2** and simply take that design, augment it with interface Verilog files for the Calinx board, and program the

Calinx board using synthesized files. When that is complete you will observe some of the characteristics of the physical implementation. *In order to get the most out of this lab you should start with a working multiplier (i.e. fix the bugs from Mini-lab 2).* The multiplier design in Verilog constitutes the design entry portion of the flow. **You will need to convert the schematic version of the multiplier to Verilog as shown in Mini-lab 1.** This is because Synplify wants a pure Verilog description for synthesis. The next step is synthesize using Synplify Pro.

We have provided you with a top-level wrapper module, called FPGA_TOP2_Mult.v. This top level module will connect your multiplier to the input/output pins of the FPGA, which will allow you to take in actual inputs from the board, and display your results on the hex LEDs. We will be using the dipswitches on the board as input to the multiplier, one of the push buttons to be the reset button, one of the push buttons as the start button, and the hex LED as output. Feel free to look inside the FPGA_TOP2_Mult.v as there are a lot of comments inside that tell you what to do as well as show you want the peripherals are that the board supports. In particular look at how the multiplier interfaces with the LEDs. In addition there are support files bin2HexLED.v and debouncer.v which you will also have to add to your project.

You will need to demo your completed multiplier on the board to your TA during your section, so save all of your files.

3.1 Synthesis

Now we start to map the design to an FPGA:
1. Start the synthesis program *Synplify Pro*.
2. Start a new project from **File->New Project**.
3. Add your Verilog source files to the project by clicking the "**Add File**" button on the left side. Be sure to include the FPGA_TOP2_Mult.v and the other two support files. **Do not include your testbenches! Why?**
4. Change target device with the "**Implementation Option"** button to Xilinx Virtex-E XCV2000E FG680. Speed Grade is –6.
5. Under "**Implementation Results**" tab change the directory of rev_1 to your local rev_1 in your project and make sure the FPGA_TOP2_Mult.edf is the result file.
6. Indicate the FPGA_TOP2_Mult as the top level module under the "**Verilog Tab**".
7. Click **RUN.** There will be warnings but there should be no errors.
8. Synplify will create an EDIF file. The EDIF file is your netlist. This ultimately is the result of synthesis.

3.2 Place and Route

1. Start the Xilinx Design manager by choosing from Synplify Pro,
Options->Xilinx->Start ISE Project Navigator. Note: Synplify Pro will automatically create a project for you this way, using the EDIF file.
2. Under Processes for Current Source in the lower left corner, **right click** on **Generate Programming File** and bring up the properties window. On the "**Readback Options**" tab, make sure that the following checkboxes are checked: **Create Readback Datafiles** and **Create Mask File**. If these two options are not set then you will not be able to program using Boundary-Scan mode (and therefore you will not be able to use ChipScope; you will learn about this tool later).
3. Under Processes for Current Source, double click on **Generate Programming File.** This will generate the *.bit file which will be used to program the LUTs in the FPGA.

3.2.1 Alternative Method

1. If you do not want to go through the trouble of updating your project, you may simply use project navigator and create a new project, with the following settings:
Device Family: VirtexE
Device: xcv200e
Package: fg680
Speed Grae: -6
Design Flow: EDIF
2. Click **OK** to create the project.
3. Go to **Project -> Add Copy of Source**. Find the .edf file created by Synplify and add it to your project.
4. Under Processes for Current Source in the lower left corner, right click on **Generate Programming File** and bring up the properties window. On the "**Readback Options"** tab, make sure that the following checkboxes are checked: **Create Readback Datafiles** and **Create Mask File**. If these two options are not set then you will not be able to program using Boundary-Scan mode (and therefore you will not be able to use ChipScope; you will learn about this later as mentioned).
5. Under Processes for Current Source, double click on **Generate Programming File.**

3.3 Download Design to the Board

1. Make sure that the parallel cable is connected to the board and that it is connected to the **JTAG** port on the board **NOT** the **Slave Serial** port. **Turn the power on to the board!**
2. start iMPACT: In the **Processes for Current Source** window in the lower left, double click **Configure Device (iMPACT)** selection at the very bottom of the window under the **Generate Programming File** option. The previous step should have already created a .bit file to be programmed. This step should open up iMPACT, the tool actually used to program the board.
3. Connect cable: iMPACT should start up and ask you what operation mode to be in. Select **Configure Devices.** When it asks how you want to configure the device, select **Boundary-Scan Mode**. It will next ask you whether or not you want automatically connect or to enter a boundary scan chain, select **Automatically connect to cable and identify Boundary-Scan chain**. Depending upon how the computer is setup it will find one or two devices. One will be labeled **xccace** (depending upon the configuration of the computer this device may not be present) and the other will be labeled **xcv2000e**. If the connection fails, the most likely causes are that you have more than one iMPACTs open, the board is off, or you have not connected the parallel cable to the right port on the board.
4. Loading the bit file: next the program will prompt you to select a bit file for each of the devices that it found. If **xccace** was found then **click the bypass** button in the **Assign New Configuration File** dialog box. Once you are programming the **xcv2000e** device (the board) select your bitfile from the window. Make sure you select the one in the right folder, as iMPACT may not always open up in your expected folder! You can also double click the chip picture and manually load the bit file for the source module you want from your project directory. You can ignore the warning regarding the "Jtag_Clk".
5. Download to board: **Right click** on the chip picture labeled **xcv2000e**, and choose **Program** (do not select any options)**.** If the program was downloaded to the board, you should see a "Programming Successful" message. If it does not work, most likely the board is not connected correctly and you should make sure the parallel cable is connect to the JTAG slot and also that the board is turned on.

3.4 Testing the Multiplier

Now that the Multiplier is downloaded to the board you can test the operation. The dip switches on the right side of the board control the input. SW9 is the multiplicand while SW10 is the multiplier. The push button SW1 is the start button and SW3 is the reset. Play around with the multiplier and convince yourself that it is working.

3.5 Examining Reports

During this process many reports have been generated which give you information regarding the design. Here is some report information you should examine **in order to answer the lab questions**.

Learning about the size and speed of a design can help to optimize it. Here is how you can do so:
1. Navigate to the **Implement Design -> Map -> Map Report** and **Double Click** to open it. This will provide information regarding many things such as Removed Logic Summary, Removed Logic, Area Group Summary, Modular Design Summary, Timing Report, etc.

2. Navigate to **Implement Design -> Place & Route -> Generate Post Place & Route Static Timing -> Analyze Post Place and Route Static Timing** and **Double Click** to open it. This will start the Timing Analyzer tool. Click the **Analyze Period for Net Clock icon** (little clock on the toolbar) and write down the minimum period. This will help with the second post lab question.