UNIVERSITY OF CALIFORNIA AT BERKELEY
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Lab 1
FPGA CAD Tools

1.0 Motivation

In this lab you will take a simple design through the FPGA Computer Aided Design (CAD) tool-flow, starting from design entry all the way to programming the hardware. This lab will give you experience with the software that you'll be using for the rest of the semester.

2.0 Introduction to the CAD Flow

Figure 1 below shows the general CAD tool flow to which you have access in this lab. Highlighted in **bold** is the flow which we will be using.

A design written in Verilog using Notepad is fed through Synplify Pro, the Xilinx PAR tools and finally iMPACT which will program it into the FPGA. The whole process is described in detail below.

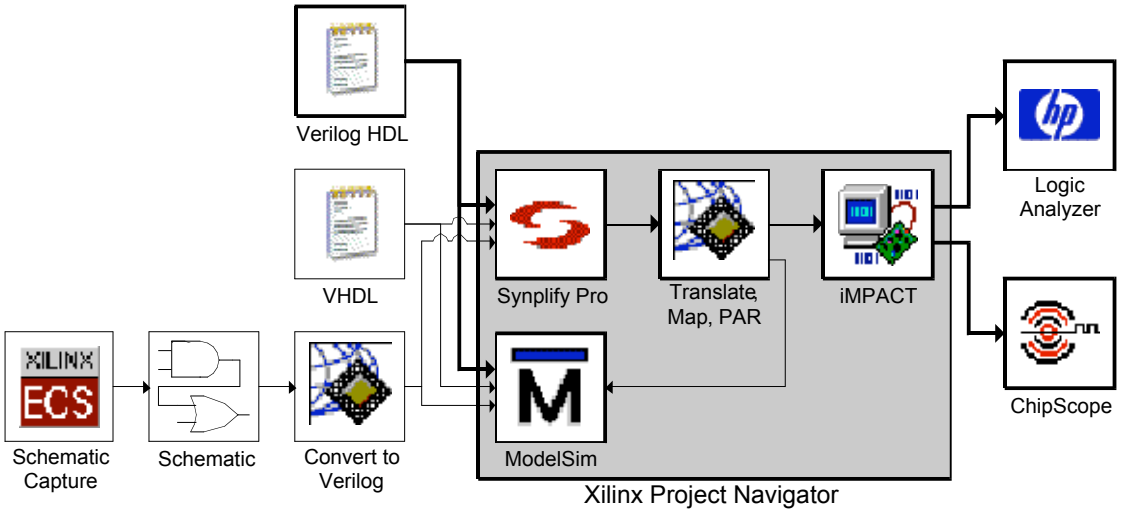

Figure 1: General CAD Tool Flow

2.1 Design Entry

The first step in logic design is to conceptualize your design. Once you have a good idea about the function and structure of your circuit and maybe a few block diagram sketches, you can start the implementation process by specifying your circuit in a more formal manner.

In this class we will use a Hardware Description Language (HDL) called Verilog. HDLs have several advantages over other methods of circuit specification:

1. Ease of editing, since files can be written using any text editor
2. Ease of management when dealing with large designs
3. The ability to use a high-level behavioral description of a circuit.

In this class we will default to using Notepad to edit Verilog. Fancier editors are available, and in fact are included with the CAD tools such as Project Navigator and ModelSim; however these tools are slow and will often hinder you. For this lab, we will provide you with a complete and working project in Verilog.

2.2 Simulation

With a design in hand, the first step is always to test it using an HDL simulator. Because actually implementing any large design can take upwards of half an hour, it is much too time consuming to simply synthesize a design, check to see if it works and then tweak it. In fact, because a fully implemented design in hardware runs so quickly (at megahertz clock frequencies) and involves so many signals, even if implementation took a mere 30 sec, it is highly impractical to debug the final hardware implementation directly.

To speed up the design cycle and also to provide the designer, you, with more detailed information about the functioning of a running circuit, we use HDL simulators such as ModelSim. Simulation allows you to provide specific test inputs to your circuit and observe both the outputs and the internal operation of your circuit giving you very detailed feedback as to what is happening and when.

Because simulation is software rather than hardware-based, it is relatively slow taking perhaps 5 min to simulate 5 msec of real time. But it allows you to create very specific input conditions, using special Verilog modules called “testbenches” to exercise your circuit. You can even to print out text messages in the event of problems, rather than forcing you to look at the binary output of your circuit.

2.3 Synthesis

Once a design is entered, simulated and debugged, the next step in the CAD Tool Flow is synthesis. It is the job of the synthesis program to translate the Verilog description of the circuit into an equivalent circuit made of primitive circuit components that can be directly implemented in an FPGA.

In a way, the synthesis tool is almost like a compiler. Where a compiler translates a high level language, such as C, into a sequence of primitive commands that can be directly executed on a processor, synthesis translates a high level language, in this case Verilog, into primitive circuit components that can be directly implemented on an 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 connected.

2.4 Place and Route

From the netlist produced by the synthesis tool, we must somehow create a file containing the bits needed to configure the LUTs, Switchboxes, Flip-Flops, and other resources that make up the FPGA. This is the job of the Place and Route (PAR) tools.

2.4.1 Placement

To properly connect the various FPGA resources our design will use, the tools must first take each LUT, Flip-Flop or other resource called for in the netlist and decide which physical piece of the FPGA will play that role. For example, a 4LUT implementing the function of a 4-input NAND gate in a netlist could be placed in any of the 38,400 4LUTs in a Xilinx Virtex XCV2000E FPGA chip. Clever choice of placement will make the subsequent routing easier and result a circuit with less delay.

2.4.2 Routing

Once the components are placed, the proper connections must be made. This step is called routing, because the tools must choose, for each signal, one of the millions of paths to get that signal from its source to its destination.

Because the number of possible paths for a given signal is very large, and there are many signals, this is typically the most time consuming part of implementing a design, aside from specification. Planning your design well and making it compact and efficient will significantly reduce how long this step takes. **Designing your circuit well can cut the time it takes to route from 30 min to 30 sec.**

2.4.3 Place and Route Tools

Unlike synthesis, which need only know a set of primitive components to express its result, placement and routing are dependent upon the specific size and structure of the target FPGA. Because of this the FPGA vendor, Xilinx in our case, usually provides the placement and routing programs, whereas a third party, Synplicity, can often provide more powerful and more general synthesis tools.

The end product after placement and routing is a *.bit file containing the stream of bits used to configure the FPGA.

Note: placement and routing are NP hard optimization problems and the provided software uses heuristics to solve them. There are cases where humans can do a better job by hand. However as the tools improve these cases are becoming quite rare.

2.5 Program Hardware

This is perhaps the simplest step in the entire tool flow, transferring the synthesized, placed and routed, and fully implemented design into the actual FPGA. Of course since this requires detailed knowledge of the programming cable and the FPGA, we must use a specialized tool for this step too.

In this class we'll be using a Xilinx Parallel Cable IV, which is really nothing more than a rather fancy, expensive wire to connect the program, iMPACT, to the FPGA. iMPACT will then download the *.bit file into the FPGA, thereby completing the implementation process.

2.6 Verification

As with any major design process, the most important step is testing and verification. While most software designs are relatively easy to test, at least to some degree, hardware presents several interesting challenges. For example when a piece of software crashes or discovers a problem it will often report some kind of error giving you, the designer, some kind of clue as to what has gone wrong. However when a circuit

in an FPGA doesn't operate the way you would like, there are no error messages or clues as to why it is not operating "correctly." In fact 99.9% of the time, the circuit is operating perfectly, but it is not the circuit you intended to build.

Verification is the rather complicated process of ensuring that under every conceivable set of valid input conditions, your circuit behaves as desired, giving the correct output at the correct time. In contrast to functional simulation described in Section 2.2 Simulation, verification is the much more thorough process of testing the final design as a whole. This often involves simulations using accurate timing models as well as complicated hardware set-ups.

In this class, we will limit the verification step to making sure that the circuit works at a human visible level. Since humans can't generally perceive anything under 1 ms to 10 ms, this will make your job significantly easier.

3.0 Prelab

Please make sure to complete the prelab before you attend your lab section. This week's lab will be very long and frustrating if you do not do the prelab ahead of time.

1. **Read Section 2.0 Introduction to the CAD Flow** above, please make sure you understand it.
2. **Examine the Verilog** provided for this weeks lab. It's okay if you don't understand it, but try to decipher what you can of it.
    a. Confine your attention to FPGA_TOP2.v, Lab1Circuit.v and Lab1Testbench.v
    b. **Do not** try to understand ButtonParse.v, Debouncer.v or Edgedetect.v, as they are rather complicated.
3. **Read the online tool tutorials**
    a. http://www-inst.eecs.berkeley.edu/~cs150/fa05/Documents.htm#Tutorials
    b. **Project Navigator** Tutorial
    c. **ModelSim** Tutorial
    d. These tutorials will guide you through the complex CAD tools.

4.0 Lab Procedure

4.1 Project Setup

1. Unzip all of the **Lab1** files into **C:\Users\cs150-xxx\Lab1Verilog**
2. **Double-Click** the **Project Navigator icon** on the desktop to start Xilinx Project Navigator
3. Click **File -> New Project** and type in a project name (i.e. **Lab1**)
    a. Set the project location to **C:\Users\cs150-xxx** (Project Navigator will create a subdirectory for the project)
    b. The **Top-Level Module Type** should be set to **HDL**
    c. Click **Next**
4. A new dialog will appear with configuration settings
    a. Device Family: **VirtexE**
    b. Device: **xcv2000e**
    c. Package: **fg680**

   d. Speed Grade: **-6**
   e. Synthesis Tool: **Synplify Pro**
   f. Simulator: **ModelSim**
   g. Generated Simulation Language: **Verilog**
   h. Click **Next**
5. Skip the **Add New Sources** dialog by clicking **Next**
6. In the **Add Existing Sources** dialog you will want to add the **Const.V** file to your project.
   a. Click **Add Source**
   b. Navigate to the **C:\Users\cs150-xxx\Lab1Verilog** folder and select **Const.V**, then click **Open**
   c. Select **Verilog Design File** and click **OK**
   d. Notice that the **Copy to Project** box should be **Checked**
   e. Click **Next**, you will add the other Verilog files in a minute
7. **Take a moment to review the project settings.** Then click **Finish**
8. **Right-Click** in the **Sources in Project** box in the upper left corner of Project Navigator, and select **Add Source** (not Add Copy of Source)
   a. Navigate to the **C:\Users\cs150-xxx\Lab1Verilog** folder and select **everything except Const.V**, then click **Open**
      i. Use shift-click and control-click to select multiple files
   b. For **Lab1Testbench.v** you will need to select **Verilog Test Fixture File.**
   c. For everything else, select **Verilog Design File**.
9. You should now have a Project Navigator project, which looks like Figure 2, below.

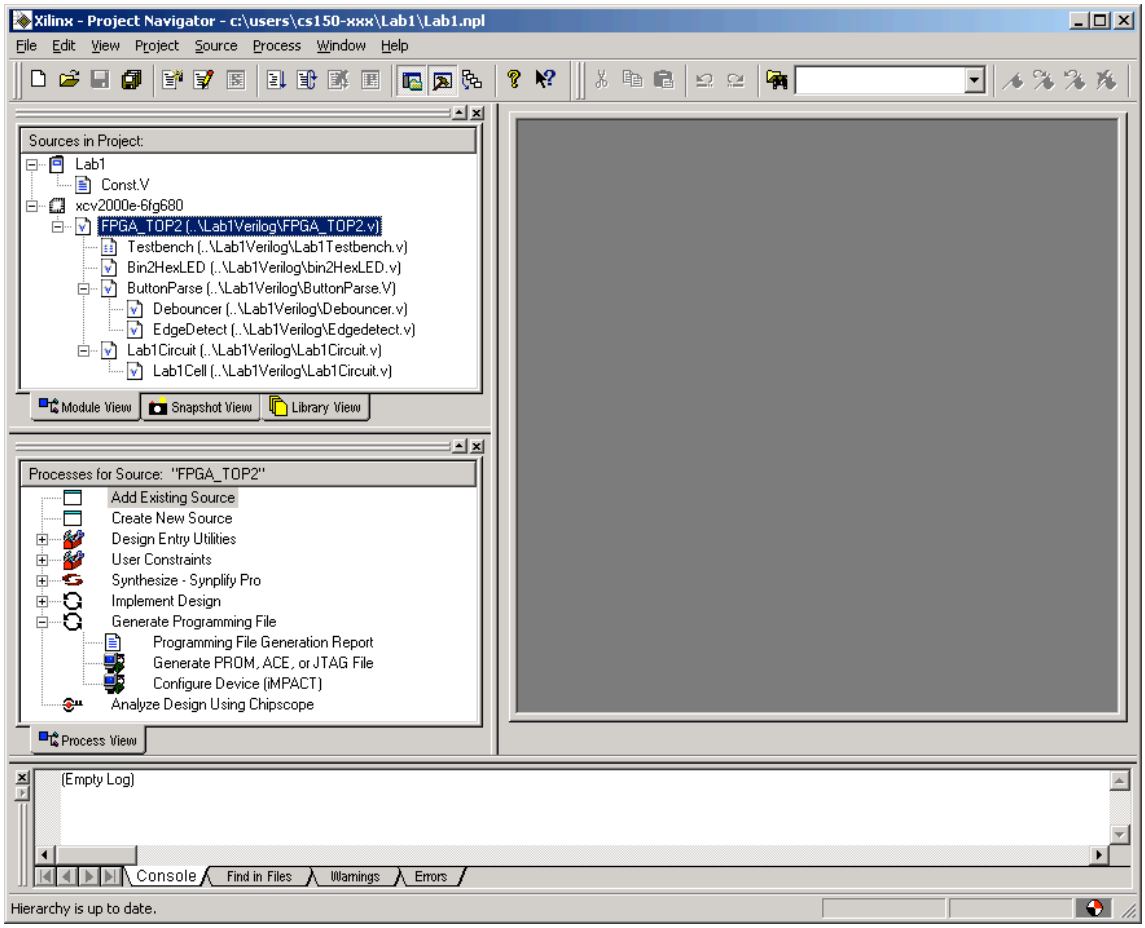

Figure 2: A Complete Project

The Project Navigator Window as should in Figure 2, above, will allow you to manage your files and invoke the various CAD tools from a central location. Do not depend on Project Navigator for everything; complex testbenches may require manual creation of ModelSim projects, just as you should manage your files from Windows Explorer.

In the upper left is the "Sources in Project" box, where you can see all the modules and testbenches that are part of your project, as well as which modules they depend on (or test) and which files that are in.

In the middle left, you can see the "Processes for Source" box, which will show all of the tools which can be applied to the currently selected source file. Notice that if you select the testbench, the Processes for Source box will change to show you ModelSim rather than Synplify Pro and the Implement Design tools.

4.2 Functional Simulation

This part of the lab is design to acquaint you with **ModelSim** by using it to **simulate the Lab1Circuit** with the Lab1Testbench. **Pay careful attention to this section, you will spend most of your time in EECS150 running simulations.**

1. Go to the **Edit -> Preferences** menu in Project Navigator.
    a. Navigate to the **Processes** tab.

b. Set the **Property Display Level** to **Advanced**

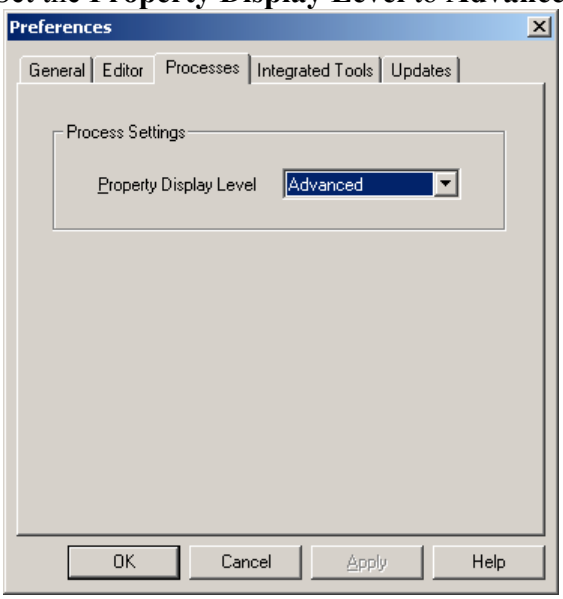

Figure 3: Setting Property Display Level to Advanced

c. Click **OK**

2. Select the **Lab1Testbench** in the **Sources in Project** box
   a. This will change the **Processes for Source** box to show a number of steps involving **ModelSim**
3. **Right-Click** on **ModelSim Simulator -> Simulate Behavioral Model** process in the **Processes for Source** box
   a. Select **Properties** from the popup menu
   b. In the **Other VLOG Command Line Options** box type **+define+MODELSIM**, taking care to keep the capitalization and the plusses.
   c. In the **Simulation Runtime** box type **10us** to run the simulation for 10 microseconds.

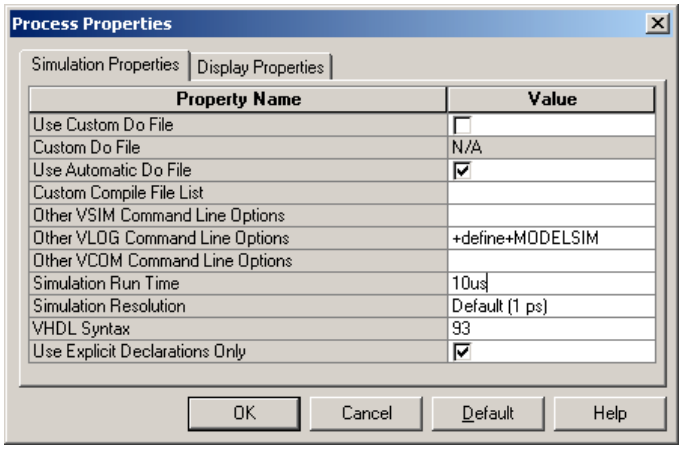

Figure 4: Defining the MODELSIM Simulation Flag

d. Click **OK**

4. Having set all the simulation options you can now simply **Double-Click** on the **Simulate Behavioral Model** process
    a. It may take ModelSim a minute to start.
    b. A number of windows will appear including: the **ModelSim Workspace** showing text messages both from the ModelSim tools and anything printed by the circuit you are simulating, **Wave Window** showing the waveforms from your testbench and any other signals you choose, **Signals Window** which lists the signals in the currently selected module allowing you to drag them to the **Wave Window** and **Structure Window** which will let you navigate the tree of modules in your project to change the contents of the **Signals Window**. Note that by **right-clicking** on a module in the **Structure Window** you can add **all of the signals** from that module to the **Wave Window**.
5. Examine the **Wave Window** with care; attempt to discover the function of this circuit.
    a. You can use the **magnifying glass buttons** to zoom in or out.
        i. The **darkened magnifying glass** is especially useful.
    b. You can **drag the vertical dividers** to show more or less of the signal names, value and waveforms.
    c. The signal values listed in the second column are those at the **vertical yellow cursor**.
        i. Move the cursor by simply clicking in the wave window.
        ii. To see the signal values in **hexadecimal** select one or more signals, **right-click** and select **Radix -> Hexadecimal**.
    d. You will probably want to **maximize** the **wave window**.
    e. If you know what the circuit does, feel free to answer Question 2 on the Checkoff Sheet.
6. **Adding more signals** to the **Wave Window**
    a. Go to the **ModelSim Workspace** or **Structure Window**.
    b. Navigate the **module tree** to the module you wish to examine.
    c. **Right-Click** on that module and select **Add -> Add to Wave**.
    d. The signals will initially have **No Data**. To get waves you will need to **restart the simulation**. Type **restart –f; run 10us** at the **VSIM 2>** prompt in the **ModelSim Workspace**.
        i. This will restart the simulation and then run it for 10_s.
7. Look at the **Wave Window** again and measure the **Clock-to-Output** delay in this simulation. That's the time from when a **rising edge** of the clock happens until when the **output** changes. Notice that this only applies to cycles where the **output actually changes**, so look for the cycles where **Enable** (Not EnablePin) is 1'b1.
    a. Answer Question 1 on the Checkoff Sheet.
8. **Close ModelSim** and return to **Project Navigator**.

4.3 Synthesis

Having simulated the circuit, you will now synthesize and implement it so that you can play with the actual circuit on a CaLinx2 board with a Xilinx XCV2000E FPGA. This step will acquaint you with **Synplify Pro**, including its ability to generate **schematics from your Verilog**.

1. In **Project Navigator** select **FPGA_TOP2** from the **Sources in Project** box
   a. This will cause a long list of implementation steps to appear in the **Processes for Source** box.
2. **Double-Click** the **Synthesize – Synplify Pro** step to start the synthesis
   a. If there is an **X** or a **!** next to the **Synthesize – Synplify Pro** step, this means that there has been an error or warning.
   b. To see the errors and warnings from Synplify Pro, **double-click** the **Synthesize – Synplify Pro -> View Synthesis Report** step.
3. To view a schematic of the circuit **double-click** on the **Synthesize – Synplify Pro -> View RTL Schematic** step
   a. This will launch **Synplify Pro** and automatically open the **RTL Schematic**
   b. Navigate through the schematic, **look inside** the **Lab1Circuit** and attempt to figure out what it does. You may find it easier if you first figure out the **Lab1Cell**.

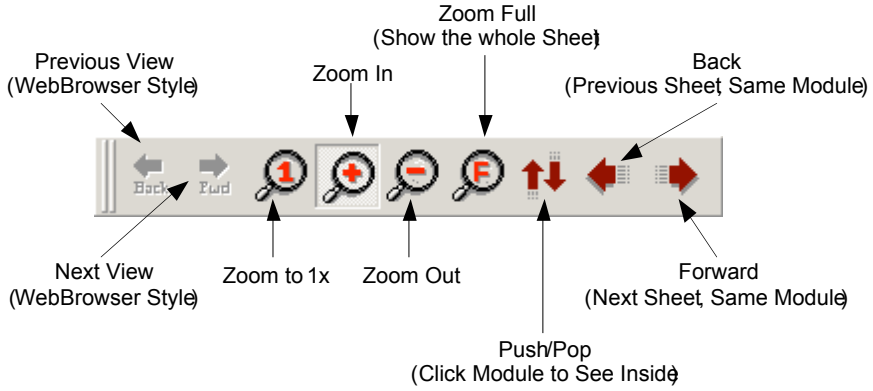

Figure 5: The Synplify RTL Navigation Toolbar

   c. Answer Question 2 on the Checkoff Sheet
4. Close **Synplify Pro** and return to **Project Navigator**.

4.4 Place and Route

While the Place and Route tools are fairly automatic, their output can be **extremely important**. This step is designed to show you some of the most basic information that can be extracted from the Xilinx Place and Route tools.

1. In **Project Navigator** make sure **FPGA_TOP2** is still selected in the **Sources in Project** box.
2. To invoke the Xilinx **Place and Route** tools, **double-click** on the **Implement Design** step in the **Processes for Source** box.
   a. This will run three sub tools: Translate, Map and PAR.

   b. Ignore any warnings from these steps in this lab. They will often give warnings that can be safely ignored.
3. Learning about the size and speed of a design can help optimize it, therefore Question 3 on the Checkoff Sheet asks you to find four separate design metrics.
   a. Navigate to the **Implement Design -> Map -> Map Report** and **Double-Click** to open it.
   b. This report should provide the information for **Questions 3a-c** on the Checkoff Sheet. When you have answered those, **Close the Map Report**.
   c. Navigate to **Implement Design -> Place & Route -> Generate Post Place & Route Static Timing -> Analyze Post Place & Route Static Timing (Timing Analyzer)** and **Double-Click** to open it.
      i. This will start the **Timing Analyzer** tool.
   d. Click the **Analyze against Auto Generated Design Constraints** button on the toolbar ()
   e. Find the **Default Period Analysis for net Clock** and write down the **Minimum Period** as the answer to **Question 3d**.
   f. From that information, compute the maximum clock frequency and write that down in answer to **Question 3e**.
   g. When you are done, **Close** the **Timing Analyzer**
4. Return to **Project Navigator**

4.5 Post PAR Simulation

Section 4.2 Functional Simulation was designed to introduce you to simulation. However a **functional simulation** does not take **timing** into account. It shows **what the circuit does** but it does not give an accurate picture of **when** certain things happen.

To run a circuit at **high-speed** we must get a better picture of **when everything will happen**. For this we will perform a **simulation** using a **Post Place and Route Verilog Model**, which will use accurate timing numbers as generated by the place and route tools, which know the **exact physical structure of the circuit**.

1. Select the **Lab1Testbench** in the **Sources in Project** box.
   a. This will change the **Processes for Source** box to show a number of steps involving **ModelSim**.
2. If you have not done a functional simulation for some reason you will need to perform steps 1-3 of section 4.2 Functional Simulation above.
3. **Double-Click** on the **Simulate Post Place and Route Verilog Model** process.
   a. First Project Navigator will generate a **Post Place and Route Verilog Model** of the Lab1 circuit.
   b. Then it will launch ModelSim to simulate the Lab1Testbench and the newly generate verilog model.
4. Upon examining the **Wave Window** you will find that the circuit is not working the way it did before. This is because the **ButtonParse** module is

designed to behave differently in simulation and synthesis so that it is faster to simulate, but more robust when you synthesize it.

a. You will need to **edit the testbench**.
b. **Close ModelSim**.
c. **Double-Click** the **Lab1Testbench** file in the **Sources in Project** box to open up that file in the Xilinx editor.
d. Change **line 35** to **`define ActiveCycles 65536** to hold the buttons down for longer during this simulation.
e. You will now need to change the simulation to run for **40ms**. For more information on how to do this, see Step 3 of Section 4.2 Functional Simulation.

5. **Double-Click** on the **Simulate Post Place and Route Verilog Model** process.
   a. **If your simulation does not run** you will need to type **run 40ms** at the **VSIM(paused)>** prompt.
   b. Notice that this simulation takes quite a while to run? That is why we make ButtonParse behave differently when we do a functional simulation.
6. You can now examine the fully functional, timing accurate simulation.
   a. Make sure to answer **Question 4** on the Checkoff Sheet.
      i. Be sure to use the **Enable** and **Out** signals as with **Question 1**.
   b. You should also be prepared to **talk with your TA** about **what causes this delay**, where the critical path is in this circuit and so forth. If you are not sure of your answer, that's okay, you only need to think about it.
7. We're now going to have you modify the testbench again, this time to try and break it.
   a. **Close ModelSim**.
   b. **Double-Click** the **Lab1Testbench** file in the **Sources in Project** box to open up that file in the Xilinx editor.
   c. Change **line 33** to **`define HalfCycle XXX** so that the clock cycle is 2 ps shorter than the minimum cycle you wrote down for **Question 3d**.
      i. HalfCycle represents _ of the minimum period. The clock will be high for one HalfCycle and then low for one HalfCycle.
   d. Rerun the simulation and answer **Question 5** on the Checkoff Sheet.
8. **Close ModelSim** and return to **Project Navigator**.

4.6 Hardware Verification

After having navigated the complicated and exceedingly **quirky** CAD tools, this is the real payoff, programming the FPGA and seeing if the circuit works.

1. In Project Navigator **Double-Click** the **Generate Programming File -> Configure Device (iMPACT)** step.

   a. This will first generate the FPGA_TOP2.bit programming file which contains the 1.2MB worth of information needed to configure the XCV2000E FPGA.
   b. It will then launch the iMPACT programming tool.
2. Make sure that the **Parallel Cable IV** is connected to the **Slave Serial** port on the CaLinx board and that the CaLinx board is on.
   a. The little light on the Parallel Cable IV will turn **green** when the cable detects that it is connected to a **powered Xilinx chip**.
   b. The power switch for the CaLinx board is in the upper right of the board.
3. When **iMPACT** launches:
   a. Select **Slave Serial Mode**.
   b. When it asks for a file select **FPGA_TOP2.bit**.
   c. **Right click** on the picture of the FPGA and select **program**

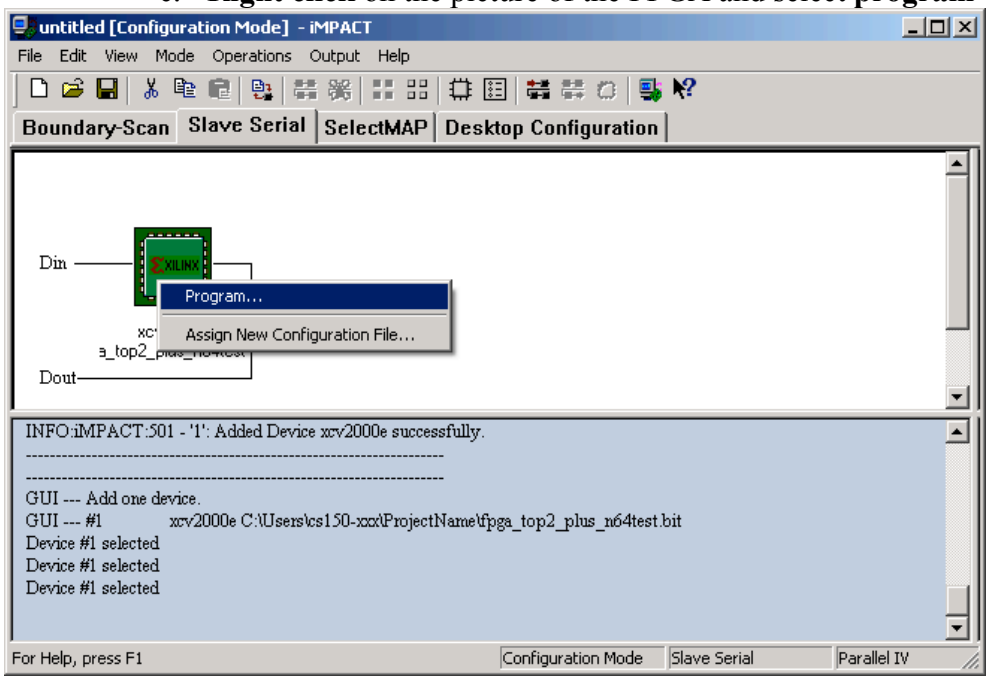

Figure 6: iMPACT

4. Once the device is configured you can play with the circuit!
   a. Select the **input on SW9**, the upper dipswitch on the right hand edge of the board.
   b. Press **SW1** to reset the circuit
   c. Press **SW2** to enable it.
   d. You will see the input on the **leftmost pair** of **7Segment LEDs**.
   e. You will see the output on the **rightmost** pair of **7Segment LEDs**.

5.0 Lab 1 Checkoff

Name: _____
Section: _____

I Functional Simulation _____ (20%)
II Timing Simulation _____ (20%)
III Hardware Demonstration _____ (20%)
IV Questions _____ (40%)

1 What is the Clock-to-Output Delay in the functional simulation?
_____ps

2 What is the function of this circuit?

3 Please fill out the following information:
a Number of Slice Flip Flops: _____
b Number of occupied Slices: _____
c Total Number of 4 input LUTs: _____
d Minimum Period: _____ns
e Maximum Clock Frequency: _____MHz

4 What is the Clock-to-Output Delay in the timing accurate simulation?
_____ps

5 Does the circuit still function with the shortened clock period? Why or why not?

V Hours Spent: _____
VI Total: _____
VII TA: _____

RevD – 8/14/05	Randy Katz	Fixed typos and general editing
RevC – 1/13/2005	Greg Gibeling	Updated to incorporate errata from Fall 2004 semester
RevB – 7/2/2004	Greg Gibeling	Complete Rewrite of Lab1 Based on the old Lab2
RevA	Multiple	Original Lab2 from Sp03-Sp04 Spring 2004: Greg Gibeling & Eric Chung Fall 2004: Aaron Hurst Spring 2003: Sandro Pintz