Lab #2: FPGA Design

ECE352F: Computer Organization

Fall 1999

In this lab you will be introduced to the use of some commercial CAD tools for FPGA design. The tools that you will use are *FPGA Express* from Synopsys and the Xilinx *Foundation Series* software. Synopsys and Xilinx now have an agreement that allows Xilinx to fully integrate the *FPGA Express* package into the Xilinx Foundation tools. The version you will use is **F1.5**. For this lab we will do a simple VHDL flow using Foundation.

1 Getting Started

In *~pc/352/lab2* you will find a sample vhdl program called **lc.vhd** and another file called **lc.ucf**. Make a new directory in your own account and copy the files there.

Examine **lc.vhd** and try to understand it. The circuit will use the seven-segment LED display on the board.

Signal names of the form **XCBUSnn** correspond to the pin names shown on the schematic for the board. You should be able to identify the signals that are connected to the display.

The board has a 12-MHz clock, so to make things visible the clock is first divided down to generate a slower clock. You can see this being done in the `CLOCK_DIVIDER_PROCESS` process.

The `COUNT_PROCESS` implements a BCD counter that is clocked with the slow clock.

The `LIGHT_DISPLAY_PROCESS` does the decoding of the counter output into the signals to turn on the display.

The interconnection of the three processes is done by variables that are declared as **signals**, which you should think of as wires.

Preparation: Draw a schematic of this circuit at the block diagram level. Each process should have a corresponding block in your diagram. All inputs, outputs and signals connecting processes should be clearly labeled.

2 Getting Started on the PCs

The PCs are setup with *Windows'95* on the local drive. When you boot the machine, you will be asked whether to **reinstall** (1) or **check** (2) the current installation. If you suspect that something has been corrupted, then you can do the reinstall and that will copy a complete new version of everything onto your local drive. Be warned that this takes many minutes. Also, if the system has been

upgraded, then you will need to do the reinstall to upgrade the version on your local disk. Normally, and if you do not answer quickly enough, it will just default to checking a bunch of files.

Most of the tools that we will be using are actually mounted on a network drive, **X:**, not the local drive.

To be able to save files, you can attach (or mount) your ugsparc directory to a drive on the PC. By convention our software assumes that user files are mounted on the **w:** drive. You can run an automatic script to do this.

Hit the Windows **Start** button and then **Run**. Enter `xilinx your_ugsparc_login`. You will then be prompted to enter the appropriate "year" digit. This will do two things. It will mount your ugsparc account on **w:** and it will start up the Xilinx **Project Manager**.

Note: For the rhythmically challenged, the middle button will do a double click.

3 Using The Xilinx Foundation Tools

Note that the *Xilinx Foundation Tools* were started automatically when you mounted your ugsparc account.

Project Manager

- **Create a New Project:** Select this option by ensuring that there is a black dot in the circle adjacent to it.
- **OK:** Push this button

Create New FPGA Express Project

- **Name:** Give a name for your project such as `ledcount`.
- **Flow:** Select the HDL option (ie black dot in the circle).
- **OK:** Push this button

Identify Source Files

- **Files:** Select the **Add HDL Source File(s)...** from the **Synthesis** menu
- **Look in:** Looks in the directory where your source file is.
- **File name:** Click on the name of the source file in the list (`lc.vhd`)
- **Open:** Push this.

At this point, your file is being **analyzed**, which is doing a first-pass compile of your program. This is the step where syntax errors will be flagged. The program you copied should work correctly so you should see no red errors displayed in the console and the question mark in the **Design Entry** box in the Flow window should be changed to a check mark.

If you happen to get errors at this point, then you will see an **X** instead of the check mark and errors in the *Error* window. To get more information on an error, you can double click on the error number.

To edit the file, you can right click on it (**lc.vhd**) in the source window and select the editor or you can edit it outside of the **Foundation** tool. After modifying the file, you should see a question mark (?) icon next to the name of the file. This means that the file needs to be analyzed again. To analyze the file, left click on **Synthesis** and select the **Analyze All HDL Source Files** option or find the shortcut button on the tool bar.

Next you must identify the top-level design. This is required in case your circuit is in several files. Click on the "+" sign beside **lc.vhd**. You can also double click the source file to expand it. Then select the **lc** icon, right click it, and select **Synthesize**.

In the **Synthesize** window, you will specify the target chip:

Family: XC4000XL

Device: 4010XLPC84

Speed Grade: You can ignore this.

Clock Frequency: Set to 12 MHz.

Effort: Low to get a faster compile

Run: Push this.

If you get errors, you can double click the error number in the error viewer for more information. There are some warnings that you can view, but we can ignore them.

This step has created a net list of your circuit. This net list describes the actual interconnection of the logic cells and the function in each cell. This net list will be taken to the next step. You should find some files with the suffix **.xnf** in your work directory. This is most likely called `xilinx/active/projects/LEDCOUNT` in your home directory unless you specified something else when you created a new project. These are the *Xilinx Netlist Format* files.

4 Implementation

Now that we have an **XNF** net list, we need to map it into the actual chip. You can think of the net list as only describing the interconnection of a bunch of gates. It does not specify where the gates should be placed on the chip.

The next major phase of this process is to do what is called **Place and Route**. Recall that you can think of an FPGA as having a number of fixed sites of programmable logic cells. The **Place** step tries to find a reasonable site for each of the gates that have been listed in the net list. Important constraints include the length of wires that are needed and critical path delays. The **Route** step tries to assign specific wires for each of the connections that are required once the placement of the cells has been done.

To do this phase of the design, select the **Implementation** box in the Flow window or from the top select **Implementation** and then **Implement Design**.

Options: Click this

- **User Constraints:** Browse and select the **lc.ucf** file that you also copied if it is not already in the form. Examine this file. This file shows the tool how to map your signal names to pads on the chip. A comment is started by a sharp sign (#). This particular file actually has most of the pins in there that you might use and comments out the ones not needed.

  Now click **Open**
- **Implementation:** Back in the **Options** window, click on **Edit Template** for **Implementation**. Go to the **Place and Route** tab. Set the **Place&Route Effort Level** to the fastest runtime. This tells the placer to just get a circuit placed. We are not overly concerned with speed at this time. Hit **OK**.
- **OK:** Hit this to get out of the **Options** window.

Run: Hit this in the implement window.

At this point, the **Flow Engine** starts up. This will show you what is happening to your design. You can also stop and back up the compilation, if for example, there is an error and you need to start over.

The final output of this process is a **.bit** file (ledcount.bit) that contains the data to program the FPGA.

5 Programming the FPGA on the Board

You will likely see two parallel port cables, one male and one female. You can connect your board to the PC using the male parallel port cable.

We have two versions of the FPGA board. Version 1.0 has two 25-pin DB-25 connectors. Version 1.2 has on 25-pin DB-25 for the parallel port and a VGA monitor connector. I guess if you want to make a graphical output, make sure that you get this board!

The schematics for the two boards are attached. They will also be made available via the web as a *pdf* file.

Now you need to download the bit file to the board to program the FPGA. Get a DOS window (**Start** → **Run** → **command**) or click the **MS DOS** shortcut icon on the desktop. Go to the directory with your bit file (eg. ledcount.bit), which is probably `xilinx/active/projects/LEDCOUNT`. Run the **xsload** command.

`w:\ledcount` **xsload ledcount.bit**

When this completes, you should see the display counting.

6 What Can Go Wrong

The whole process is very complicated, and the directions so far have only outlined a very simple procedure, where, hopefully, not much goes wrong. However, many, many things can go wrong.

Most of the manuals for modern tools are found online, so you can search the help directories. Often it is easier to find someone that has done it before.

7 What is missing

In every CAD flow, there should be a lot of simulation done. You should be able to simulate the original VHDL code before any of it is compiled. This is called *functional simulation* of the VHDL where you first test that the logic of your circuit works. On the student version, you will have access to the *Aldec* student simulator. It is not available in the lab. Hopefully it will be soon. We also have access to the *Synopsys VSS* simulator on the ugsparcs. If you want information about it, please let me know.

After you get the net list from compiling the VHDL code, you should also be able to simulate it. This is where you may find problems that the VHDL compiler caused when it translated your circuit. This can be done with the *Foundation* tools.

After you place and route your circuit, you can extract timing information. The actual placement and wires will introduce different delays than were used during functional simulation. This will give you a much more accurate picture of how fast your circuit will run. We will skip this. Our emphasis will be on getting functionality.

8 XSPORT

There is one other command, called **xsport**. This allows you to send bits to the parallel port of the PC to act as inputs instead of using switches. For example:

`w:\ledcount` **xsport 1**

will set a 1 on the least significant bit (XCBUS44) and zero on the rest. The pins that you can access are XCBUS49, XCBUS48, XCBUS47, XCBUS46, XCBUS45, and XCBUS44. The value **1011** will set a one on XCBUS47, XCBUS45, and XCBUS44. The rest will have zero.

9 What you should do

In this part, you can try to modify the example program in several ways.

Copy the program and try to add an asynchronous **Reset** signal. This signal should force the counter to zero as long as the **Reset** signal is high, independent of the clock. You should map **Reset** to pin 44 so that you can control it with **xsport**.

Once you have **Reset** working, add a synchronous preset. This means that on a clock edge you will want to load four bits into the counter. You will need a **Preset** signal. Map it to pin 45. The four bits to be loaded can be mapped to pins 46–49. Make pin 49 be the most significant bit. Since you only have six input bits, you should be able to manipulate your circuit using **xsport**.

If you still have time, try making your circuit count up or down, or even count by different increments. You can add other inputs using switches.

Preparation: Prepare modified versions of the counter program that will implement the above functions.