ECE352F: Computer Organization

Lab #3: FPGA Design

Fall 1997

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.

1 Getting Started

In *~pc/352/lab3* 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.

Draw a schematic of this circuit at the block diagram level. Each block should be a process.

2 Getting Started on the PCs

The PCs are setup as *network PCs*. This means that they run diskless and all the files are found on the network. This makes for much easier system administration because each PC will be exactly the same. Security is also easier because you cannot introduce any viruses! The system software partitions are not writable by users.

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`. This will do two things. It will mount your ugsparc account on **w:** and it will start up the Xilinx **Design Manager**. Minimize the Design Manager for now, using the icon in the upper right that looks like an underscore. Do not kill it using the **X** icon.

3 Using FPGA Express

Double click on the *FPGA Express* (Synopsys) icon. For the rhythmically challenged, the middle button will do a double click.

Startup Window Create a new project

Create Project Folder

> **Save in:** Go to **w:** and find the directory with the VHDL file that you copied.
>
> **Name:** Give a name for your project such as `ledcount`.
>
> **Add files:** Make sure that **Add files after creating the project** is ticked off.
>
> **Create:** Push this button

Identify Source Files

> **Look in:** The directory where your source file is.
>
> **File name:** Click on the name of the source file in the list.
>
> **Automatically Analyze:** Check this off.
>
> **Add:** Push this.

At this point, your file is being **analyzed**, which is doing a first-pass compile of your program. If you are bored, you can watch the train zoom along in the logo. This is the step where syntax errors will be flagged. The program you copied should work correctly so you will see a window with **lc.vhd** in it and a check mark.

If you happen to get errors at this point, then you will see an **X** instead of the check mark. You can select the file with the errors and go to **View** $\Longrightarrow$ **Errors/Warnings**. To get more information on an error, you can double click on it.

To edit the file, you can right click on it (**lc.vhd**) in the source window. To analyze the file, use **Synthesis** $\Longrightarrow$ **Analyze All** or find the shortcut button on the tool bar.

Next you must identify the top-level design. This is in case your circuit is in several files. Pull down the list in the middle of the tool bar and select **lc–RTL**. The **RTL** comes from the architecture specification in the source code.

In the **Create Implementation** window, you will specify the target chip:

Vendor: Xilinx

Family: XC4000

Device: 4010EPC84

Speed Grade: You can ignore this.

Clock Frequency: Set to 12 MHz.

Keep User Pads: Make sure this is selected.

OK: Push this.

If you get errors, you can double click the message in the error viewer for more information. Here, you should see a design name show up called **lc–RTL–0**. Double click this and go to the **Ports** tab. Here we have to specify some information about pin 13 (XCBUS13). Observe on the schematic, that this pin is connected to the clock and it is also labeled as **PGCK1**. This means that this pin has a special internal buffer that is connected to an internally routed clock network. To use this buffer, go to the cell under **Global Buffer** for XCBUS13 and select **bufgp**. Use the **X** icon to close the window and save the file.

Now, you can do the final step. At this point, the circuit is in an intermediate form that must now be optimized and targetted for the actual chip. Select **Synthesize** $\Longrightarrow$ **Optimize** or find the shortcut button.

You will now see **lc–RTL–0–Optimized** show up in your list of designs. There are some warnings that you can view, but we can ignore them.

The final step is to create 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. Go to **File** $\Longrightarrow$ **Export Netlist** or find the short cut button. (Observe that the main steps you need to carry out are in the three buttons on the tool bar on the right.) You will need to specify the directory where to put the net list file. You can just keep it in the directory that you are working in. You should find some files with the suffix **.xnf**. 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, you must use a Xilinx implementation tool. Start up the **Design Manager** that you originally started at the beginning. It should be on the tool bar at the bottom.

Go to **File** $\Longrightarrow$ **New Project**. In the **New Project** window:

Input Design: Hit Browse.

Files of type: XNF Files

Select file: Click on **lc.xnf** to select it. It is the top-level file that will call the other one.

Open: Push this.

OK: Push this in the **New Project** window.

You should now select the design to implement. It should already be highlighted: **lc**. Now hit **Design** $\Longrightarrow$ **Implement**. This should get you the **Implement** window:

Part: It should show XC4010E-1-PC84

Options: Click this

User Constraints: Browse and select the **lc.ucf** file that you also copied. 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 **Placement Options** 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 that contains the data to program the FPGA.

5 Programming the FPGA on the Board

Now you need to download the bit file to the board to program the FPGA. Get a DOS window (**Start** $\Longrightarrow$ **Run** $\Longrightarrow$ **command**). Go to the directory with your bit file. Run the **xsload** command.

`w:\labs\ledcount` **xsload lc.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 really test that the logic of your circuit works. I am trying to arrange access to a new VHDL simulator for the PC that we should get very soon (within days I hope).

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. For our purposes, we will try to skip this step as your circuits should not be too complex.

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. Again, 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. For example:

`w:\labs\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 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 signal is high, independent of the clock. You should put **Reset** on 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. Put it on pin 45. The four bits to be loaded can be put on pins 46–49. Make pin 49 be the most significant bit. Since you only have six input bits, you should can 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.