**The Big Walk-Through**

This is a textual walk-through for the verilog infrastructure for CSE 148. It doesn’t take you through a bunch of options, just a single example path. I’ve tested it with **Quartus II 12.1 Web Edition**, and **Modelsim-Altera 10.1b Starter Edition**. You should have Quartus 13.0SP1. I expect this Walk-Through will work exactly the same. Please let me know right away if something doesn’t work as stated. Note that more recent versions of Quartus II will not support our FPGA model.

**Loading the verilog code:**

1. Decompress the baseline zip file into a directory. There is a README file (somewhat obsolete), two other files, and three directories at the highest level. *ref* has some MIPS ISA references and some info about the benchmarks. *src* has the verilog and related files. Inside the src directory is the *hex* directory that has our benchmarks. For each benchmark, we have the C file, a disassembly of the compiled MIPS code, and the compiled code in the form of a HEX file which allows it to be downloaded onto the DE2 board RAM. We’ll only be using the hex file here, but you may find the others useful at some point. You’ll learn about the Control Panel directory later in the walkthrough.

2. Launch Quartus II. In the first window/dialog, “create a new project”.

3. “New Project Wizard” dialog. First entry item – browse to the *src* directory within the directory you just created. Second – name the project (eg, mips\_cpu). Click next.

4. “New Project Wizard: Add Files”. Click “Add All”, then “next”.

5. “Family and Device Settings”. Choose “Cyclone II” and EP2C35F672C6. Click next, next, finish.

**Compiling the verilog code:**

1. In Project Manager subwindow (upper left), choose “Files” tab, then double-click mips\_cpu.v. You should see the file opened in a larger subwindow. Scroll down and make sure that the main module (the first non-comment code you see) has the same name as the project name you chose in step 3 above.

2. Menu item Processing->Start Compilation compiles the whole thing. Go on the web and read a good verilog tutorial while it’s compiling – it’s slow.

3. Hit OK when it’s done.

4. Assignments->Import\_Assignments opens up the pin assignment window. Select the file “DE2\_pin\_assignments” (it’s up one directory level). This does all kinds of cool magic – basically, as long as you use the right names for things, it ensures that all the inputs and outputs to the board I/O get automatically connected to the right fpga pins.

5. Repeat step 2.

**Modeling the CPU in ModelSim:**

0. Go into test\_mips\_cpu.v (eg, in the Quartus II editor). Find the readmemh call. Modify the file name to give the full path of the nqueens.hex file.

1. Run the ModelSim program. Close the welcome window. In the main window, File->Change Directory and select the folder containing your project.

2. Menu Compile->Compile. In the dialog that pops up, go to the src directory and select all files (shift click) and click “compile”. This includes the test\_mips\_cpu.v file, which is the testbench waveform to make this all go. When the dialog asks about the “work” library, say “yes”. Click done.

3. The main (library) window now has a library named work. Click + to open it. Find test\_mips\_cpu.v. Double click it. That opens several windows.

4. Find the “sim” subwindow. Click on “MIPS\_CPU” which opens the main module, showing all signals visible for the highest-level module, in the Objects window. Go to the Objects window. Select (control-click) the following (somewhat arbitrary) signals: InstructionsExecuted, CycleCount, IMEM\_i\_Address, DEC\_i\_PC, ALU\_i\_PC, ALU\_o\_Result, WB\_i\_Write\_data. Once they are all selected, right-click one and choose Add To-> Wave->Selected Signals.

5. This opens the “Wave” window. You may need to enlarge it. Find the box at the top that says “100 ps” and change 100 ps to 60 ms. Select all the signals, then right click and select Radix->Hexadecimal. Hit Run (the icon to the right of “60 ms” with a little document and down arrow). This will take a little while. Eventually, you should see some outputs in the “transcript” window: PASS test 1, … and the last output DONE test 00000160. When it is done, go back to the Wave window and look around. Because the flashloader (see below) takes a long time, nothing interesting happens until about 4 ms (you could trac o\_FlashLoader\_Done, which signals the start of real instructions executing. Simulation after that will get faster once you add optimizations to make execution faster. When you’re doing a lot of simulation (e.g., debugging), you probably want to find a way to bypass the flashloader, since it is not essential for simulation. For now, look at the output in the Wave window and see if you can make some sense of it.

6. Alternatively (to step 4), in the Wave window, you can menu->file->Macro File… file “wavesetup.do” which will preload the Wave window with interesting signals to look at.

**Downloading files to onboard flash: [haven’t re-tested the rest of this yet this quarter.]**

In order to run out of memory (on-board DRAM), we need to actually load the program into onboard memory. We’re actually going to load the data into the flash memory, and every time we start up a program, we’re going to dump the flash into DRAM.

0. Make sure the DE2 board is plugged in, and connected to the computer with the USB cord. Turn it on with the big red button.

1. Compile the design in Quartus II.

2. Menu Tools->Programmer

3. In the new window, choose “Hardware Setup…” and select USB-Blaster in the “Currently Selected Hardware” dialog, and close.

4. Select the .sof file.

5. Click “delete”.

5. Click “add file”.

6. Go to the control panel directory and select the control panel .sof file.

7. Back in the programmer window, select the .sof file and click “start”.

8. The control panel is now loaded into the fpga, but we need to run the software that interacts with the control panel. Close the programmer window.

9. Outside of Quartus II, find the control panel directory and open it up. Double click the DE2\_Control\_Panel program.

10. Open->Open USB Port 0. Now you can communicate directly with the board. Try turning on some LEDs.

11. Select FLASH. Click “Chip Erase”. Wait. Click File Length box. Click “Write a File to Flash”

12. Select the nqueens.hex file (make sure it’s not one of the other files named nqueens.\*). It should take just a second or two.

13. Open -> Close USB Port.

14. Close the Control Panel application.

15. Now you can run the nqueens program on the FPGA:

**Running on the FPGA:**

1. Make sure the DE2 board is plugged in, and connected to the computer with the USB cord. Turn it on with the big red button.

2. Go back to Quartus II. Menu Tools->Programmer. If you are still seeing the Control Panel .sof file, close it and select the menu item again. Now you should see the mips\_cpu file

3. Select “Start”. You should see a blue led go on briefly then off.

4. The board is now programmed and will automatically run the program. If you want to run it again, hit button KEY0. It should quickly display P01‐>P60 (wrapping around once), and finish with D60 (okay, everything but the last is too fast to see unless you slow down the clock) in the LED 7‐segment display. (this is the equivalent of the Done test 160 result from the simulation).

5. The memory image is stored in flash memory, which is non-volatile. As long as you don’t want to switch benchmarks, you don’t need to program the flash again, even if you’ve turned the board on and off. But you do need to program the FPGA again after losing power.

**Debugging the CPU with SignalTap II:**

If you want to debug the hardware (and you will have to debug hardware – I find that simulation and the hardware often do not match up), sometimes you can do it by outputting signals to the LEDs or LCD. But often you will need to do something more sophisticated. But you don’t need to go get a logic analyzer, you can program a logic analyzer right onto the FPGA (if there is sufficient extra logic).

1. In Quartus II, Tools->SignalTap II Logic Analyzer.

2. Upper right window, where it says “Hardware:”, select USB-Blaster.

3. Double click the big window, underneath the empty table “Type|Alias|Name…”

4. In Node Finder window. Filter: SignalTap II: pre-synthesis.

5. Click “List”. This gives the available signals from the main file. In “Nodes Found:” Select CycleCount, InstructionsExecuted, flashreader:flashloader2:o\_Done, fetch\_unit:IFETCH:o\_PC. Select “>”.

6. Hit OK.

7. Back in main SignalTap window, right click Trigger conditions box for for the flashloader Done signal. Select “rising edge”. This will trigger the data collection when the flashloader finishes.

8. Also need to add the clock. Click “…” to the right of “Clock:”. In this window, select “Look in: mips\_cpu”, again select SingalTap II pre‐synthesis, click “List”.

9. Select CLOCK\_50. Select “>”. Click “OK”.

10. In main window, select “sample depth: 1K”.

11. Switch back to the Quartus II window, and hit Control‐L to recompile. This now compiles a version of your CPU plus the logic analyzer.

12. When it is done, use programmer to load the program into the fpga again.

13. In the signaltap window, select Processing‐>Run Analysis.

14. Hit Key0 to start the nqueens program on your CPU.

15. SignalTap II configures a logic analyzer to actually record data on the fpga, then uploads it via the USB. You should see the resulting waveforms now.

16. Pretty cool, huh?

17. The logic analyzer wastes space, so make sure it is not being compiled in when you’re not

using it.