CSCI 345 Laboratory II

September 17, 2003

Revised 9/19/03

Objective

The goal of this laboratory session is to familiarize you with the DK development environment by exploring what actually happens when you set up a workspace and a project, and to see what commands are actually executed when you build a project.

Lab Activities

1. Create a Workspace and Add a Project/File to It
2. Use Command Line Tools to Compile a Standalone Program and to Generate a Downloadable Programming File From It
3. Simulate the Standalone Design
4. Compile and simulate the standalone program using the PAL.
5. Submit a Report of Your Lab Activities

Create a Workspace and Add a Project/File to It

You are to work in groups of two or three during the laboratory session. Do the work for the session using one person's account. The amount of code written will be so little that there is no need to make the files available to the others in the group, but you may do so if you wish.

As you know, your home directory is on the H: drive, which is a networked drive located on the domain server for the "tree" network. When you log into your account, your files from H: are copied to the computer you log into, and when you log off, they are copied back to H: Although you can do your work on either the H: drive or on the C: drive, you can save a lot of time when working on your projects if you work on the local copy, on the C: drive. The instructions that follow assume the model in which you work on the C: drive and count on the system to copy your work back to the H: drive for you automatically. If you don't trust this model, you should back up your work, either to a floppy or over the network to another computer, such as forbin before you log off. But this is Windows, so there is really no need to be concerned about backups.

Create a New Workspace for This Laboratory

If you haven't done so already, click "My Documents" on the desktop, and create a new folder named "My Projects." It will be on the same level as 'My Pictures," My Music," etc. *Note the path to your "My Projects" directory in the Address bar of Windows Explorer.*

Start DK, select File→New, and choose the Workspace tab on the dialog box that comes up. Use "Laboratory II" as the *Workspace name*. For the *Location*, browse to your "My Projects" directory at the address you noted in the previous paragraph.

Create a Project And Add a Handel-C Source File to It.

Use File→New again, but this time, select the Project tab. Name the new project "Simple," select Xilinx Virtex II in the left-hand pane, and be sure the project is part of the Workspace you just created. Now use File→New again to add a Handel-C source file named "*simple.hcc*" to the project. If you haven't done so already, select Tools→Options and go to the Tabs tab. Set it so the editor substitutes spaces for tabs. (A Vickery pet peeve.) You might also want to set the tab stop width to 2 instead of 4, and be sure auto-indent is checked. Put a comment line containing the file name at the beginning of *simple.cc*, and put a comment block at the beginning of the file that describes the program briefly and that lists the names of the people in your lab group as authors. Save the file (Ctrl-S).

Use Command Line Tools to Compile a Standalone Program and to Generate a Downloadable Programming File From It

Use either your favorite text editor (C:\Utils\Vim\Vim6.2\gvim.exe) or the DK editor to make *simple.hcc* a Handel-C program that has a three-bit unsigned register which is incremented by one for every clock pulse. There is to be a simple *main()* function preceded by *set* statements for the clock, family, and part.

- Look in Section 3.7 of the RC200 Hardware Manual and choose one of the three external clock pins for your design, B11, C11, or E12. Look up the set clock statement in Chapter 9 of the Handel-C Language Reference Manual, and use the external_divide keyword so your design will use a 5Hz clock. (This is just an exercise; you won't actually get to work with this clock speed on the RC200.)
- For the set family statement, you may look at the Chip settings for your project and use the name in parentheses there, or you can look up the valid names on page 159 of your Handel-C Language Reference Manual, or you can use the name given in class on 9/15. Be sure you spell and capitalize this name exactly as given in the manual or on the Chip tab.
- For the set part string, you should use the string given in class on 9/15 ("XC2V1000-4FG456"). Be sure you get this string exactly right.
- Save *simple.cc* to disk.
- Start a Cygwin bash shell. Cygwin gives you a Unix environment under Windows. Use a cd command to change directories to your project directory. (*cd "/cygdrive/c/Documents and Settings/<account>/My Projects/Laboratory II/Simple")* You need the quotes because of the spaces inside the string.
- Compile your program with the *–verilog, -vhdl,* and *-edif* options (three invocations of the handelc command). Ignore the warnings about "No HDL output style specified …" for the Verilog and VHDL versions.
    - Look at *simple.v* and tell how the clock pin is implemented in Verilog.

- Look at *simple.vhd* and tell how the clock pin is implemented in VHDL.
- Look at *simple.edf* and tell how the family, part, and clock are used in the netlist.
- What did you find in the *simple.ncf* file that was generated when you used the *–edif* option?

- Compile your program again using the *–edif* option, and append the full pathname to the RC200 support library file, *rc200.hcl*, to the command line. This is needed for the next step to work.

The *handelc* command, like *gcc*or *g++* is actually a "compiler driver." That means it is a program that invokes other programs (preprocessor, compiler, linker) depending on what options and files are given on the command line. By adding a library file to the command line, the compiler driver not only compiles the program, but also links it to a Celoxica-supplied library that includes standard information used by the Xilinx tools invoked from the *edifmake* batch file (see next step.)

- Run the *edifmake* batch file on the simple module. You will have to type the command "edifmake.bat simple" from the Cygwin prompt.
  - How large is the resulting file, *simple.bit?*
  - Look at the output from *map* and from *par*, and tell how many flip-flops, how many slices, and how many CLBs are used by this design. (*Hint: I mentioned how many slices there are in each CLB in lecture.)*

Simulate the Standalone Design

- Type exit or Control-D to exit the bash shell. Go back to the Simple project in DK. If the Active Build Configuration is not already "Debug" select that from the drop down list, and click the build toolbar button (or press F7) to build a simulation of the program.
  - What files/directories were built under the Simple directory as a result of building the simulation?
  - What file contains the simulation code for the simple program?
- Press the Step-Into toolbar button (or press F11) to start simulating the program. There should be at least two windows at the bottom of DK, a Clock/Thread window, and a Watchpoint window. If they aren't there use the View→Debug Windows menus to make them visible. (You may use the Variables window instead of the Watchpoint window if you prefer.) Type the variable name you used for your 3 bit register into the watchpoint window, and resize that window and the clock/thread window so you can see all the information in both of them.

- Single step through the program (F11) repeatedly and observe the behavior of the two lower windows.
    - What sequence of values does your register take on? Explain.
    - What is the sequence of values you see if you change the type of your register from *unsigned* to *int*?
    - What happens if you change the register size to 4?
    - What happens if you omit the size of the register when you declare it?
- Exit DK, saving all your work.

Compile and Simulate a Simple Program Using the PAL

- Create a new project named PAL_Example in your Laboratory_II workspace, and add a new Handel-C source file named *pal_example.hcc* to it. Make sure this new project is the active project.
- Write a version of *pal_example.hcc* that does the same thing as *simple.hcc*, but instead of the three *set* statements for device and family, enter them in the Chip tab of the Build Configuration. To set the clock, define the rate you want (in Hz) by defining a value for the preprocessor symbol PAL_TARGET_CLOCK_RATE, and then #include the standard header file, *pal_master.hch* in your code before the main function. Omit all three of the set statements from *simple.hcc.*
    - What other symbol do you have to define when compiling a program that includes *pal_master.hch*? (Hint: You'll get an error message when you try to build the simulation; you'll have to ask, or look at one of the examples to determine the symbol's name.)
- Add the needed #define statement to your code, or define the missing symbol on the Preprocessor tab of the Project→Settings panel. Compile and simulate your program, and verify that it works the same as the simple version.

Submit a Report of Your Lab Activities

Write answers to the questions asked in this handout, and submit it to me by Monday the 22nd. Write your report so that it can be understood without referencing the handout. Include program listings of *simple.hcc* and *pal_example.hcc* in your report.