

Laboratory II

Handel-C Development

Introduction

The purpose of this laboratory is to familiarize you with the Handel-C language and the DK development environment.

Activities

1. Create a Workspace.
2. Create a Project.
3. Simulate a Simple Project
4. Simulate and Run a PAL Project
5. Submit a Report of Your Laboratory Activities

Create a Workspace

Start the DK IDE, and use File → New to create a new blank workspace. Use the name “Laboratory II,” and be sure the workspace is in your “My Projects” directory. Save the workspace and exit DK. Use Explorer to verify that the Laboratory II directory has been created in your My Projects directory and that it contains a workspace information file named “Laboratory II.hw.” (You should also see a “Laboratory II.pref” file where your preferences are stored. You can double-click on the workspace file to re-launch DK.

Create a Project

Use File → New to create a new project. We’ll call it “Counters,” and will use it to practice using some of the features of the Handel-C language. Add the project to your workspace.

DK crates new projects with six different build configurations (Generic, Verilog, etc.) Use Build → Configurations to delete all but Debug. Copy the Debug configuration to another one named Simulate, and delete the Debug configuration too. Now the drop-down list for build configurations should list just the one that you have created without any others.

Simulate a Simple Project

Use File → New to add a Handel-C source file named *counters.hcc* to your project. Use the editor to enter the following code into your source file:

```
// counters.hcc
set clock = external "P1";
void main( void )
{
}
```

Handel-C *main()* functions must always have a clock associated with them. This code says the clock signal is coming from a pin named “P1” outside the FPGA. This will be sufficient for the simulation configuration, but not for downloading to an RC200E, where the clock will come from a different pin number.

With the Simulate configuration selected, build your project. There should be no errors or warnings. Use Windows Explorer to verify that your project directory has been created under your workspace directory, that the Debug directory has been created under the project directory, and that there is a dll file in the Debug directory. See if you can figure out how to get DK to use a different directory name, such as “Simulate” for generating the simulation dll.

You could try running your simulation now, but it wouldn’t do anything.

Add two global variables, an unsigned int with 3 bits and a signed int with 4 bits. Inside your *main()* code two parallel endless loops, one of which adds 1 to the unsigned variable, and the other of which adds one to the signed variable.

Build your simulation, and single step through it using the F11 key (or one of the blue “Step” toolbar buttons). Use the View → Debug Windows menu to be able to see the values of your two variables as you step through the code.

With two *while* loops, your code has two threads of execution, but you can step through only one thread using the F11 key. Use the breakpoint button (the hand icon three places to the right of the build icon on the toolbar) to set a breakpoint in the second while loop, and use F5 to set the simulation running. Use F11 to step from there.

Modify your code to implement each of the following changes, and simulate each one to make sure it works:

- Handel-C has a “delay” statement that does nothing except to use one clock cycle. Insert a delay statement so that the unsigned variable increments half as often as the signed one.
- Create an array of values, and write code that increments all of them in parallel.
- Now create a signed rom array containing an arbitrary list of signed 5-bit values, and write code that computes an 8-bit sum of the elements as fast as possible. Simulate the code to see how many clock cycles it takes.
 - You can use a standard library macro, *adjs()*, to convert a signed value of one width to another one. You need to put the statement `#include <stdlib.h>` near the beginning of your program to use macros from the standard library. You can examine *stdlib.h* with a text editor (vim and emacs are available; use Ctrl-Alt-V to start vim.)
 - To use the standard library, you must be sure *stdlib.hcl* is listed in

Project → Settings → Linker, “Object/library modules.”

- Create a four-stage “pipeline” array with a width of 8. When the sum has been computed, push the value into the pipeline, and add the rom array to the sum again. Pushing a value into the pipeline should take one clock cycle. The output of the pipeline should be connected to an 8-bit variable named “display.” Simulate the code to make sure it works.

Simulate and Run a PAL Project

This project builds on the concepts you practiced in the Counters project, but will be different enough to warrant creating a separate project for it. So create a new project in your Laboratory II workspace, and call it “Pipeline.” While you are creating it, you will have a chance to specify the type of project from a long list. Select “Xilinx Virtex II” (the type of FPGA on the RC200E). Create two build configurations, a copy of Debug configuration called Simulate, and a copy of the EDIF configuration called RC200E. Delete all the unused configurations from the project, leaving just the two new ones.

This project will operate as follows: The rom, sum, pipeline, and display variables will be the same as in the previous project. The contents of the display variable will be displayed (in hexadecimal) in two seven segment displays. One pushbutton (“Add”) will be used to cause a new sum to be computed and pushed into the pipeline, and the other pushbutton (“Reset”) will be used to reset the circuit by zeroing the sum and making the pipeline “empty.” Two LEDs will indicate in binary how many stages of the pipeline are currently in use (0, 1, 2, and 3 or more). The seven segment displays are to be dark until values are received from the pipeline.

- To use the PAL, you need to include the header file *pal_master.hch* in your Handel-C code. You will also need to link to different library files depending on whether you are building for simulation or for configuring the RC200E. Set these up on the Linker tab of the dialog you get from the Configuration → Settings menu.
 - For simulation, you need to link to *pal_sim.hcl* and *stdlib.hcl*. Also, you need to give the exact pathname of the PAL Virtual Console software in the “Additional C/C++ Modules” text field of that same tab. The pathname is
C:\Program Files\Celoxica\PDK\Software\Lib\PalSim.lib.
 - For RC200E configuration, include *stdlib.hcl*, *pal_rc200e.hcl*, and *rc200e.hcl*. This last library is the “Platform Support Layer” (PSL) for the RC200E, which the PAL library uses.
- Be sure to define the preprocessor symbol `PAL_TARGET_CLOCK_RATE` before you include *pal_master.hch* in your code.
- Be sure the preprocessor symbol `USE_SIM` is listed in the Preprocessor tab of the Project Settings for the Simulate configuration and the symbol `USE_RC200E` is listed in the Preprocessor tab of the Project Settings for the RC200E configuration.

- You need to add the following three commands to the Build commands tab of the Project Settings for the RC200E configuration:
 - `cd RC200E`
 - `call edifmake_rc200 Pipeline`
 - `beep`
- You enter a directory name (RC200E would be good) in three places in the Project Settings for the RC200E directory: twice on the General tab, and in the Outputs view of the Build commands tab.
- You need to enter the part number of the FPGA in the Chip tab of the Project Settings for the RC200E configuration. It's XC2V1000-4FG456.

You can see how to enable and disable the seven segment displays, how to read the pushbuttons, and how to turn the LEDs on and off by studying the PAL Reference Manual available from the course web site. (You can also look at sample code in the Celoxica directory, but do not try to cut and paste code from the examples into your program ... it's a bad habit.

Plan how to process the switches before you start writing your code. Pressing the Add switch must add a new sum just once, not repeatedly while the switch is held down. The reset switch, on the other hand can safely perform a reset on every clock cycle that it is pressed. What if both switches are pressed at once?

Submit a Report of Your Lab Activities

Use a word processor to write a report of your lab activities that follows the format of the [Lab Report Guidelines](#) for this course, and email it to me by midnight of the due date.