



FPGA TOOLS; HELLO WORLD

NOTE: This a DRAFT!

1 Objective

The goal of this lab is to introduce you to some of the top-level tools used to create an FPGA bitstream and the process of configuring an FPGA with that bitstream. More details will be unveiled in future labs. This time, the goal is to come away with:

- working knowledge of how to set up your environment
- start Xilinx Platform Studio
- synthesize an existing hardware design
- compile and run a simple C application

2 Tutorial

Step.1 Login and Setup

From any X-Windows machine on campus (i.e., the Mosaic Solaris labs), open a Terminal window and log in to homer:

```
bash-2.05$ ssh -X rsass-homer.uncc.edu
```

(The capital -X tells ssh to send X-Window output back to your local computer.)

- Create a directory build directory:

```
bash-2.05$ mkdir lab2
bash-2.05$ mkdir lab2/build
bash-2.05$ cd lab2/build
```

- Add Xilinx settings to your environment:

```
bash-2.05$ . /opt/xilinx/ise/8.2.03i/settings.sh
bash-2.05$ . /opt/xilinx/edk/8.2.01i/settings.sh
```

- Finally, add the cross-compiler tools to your path:

```
bash-2.05$ . /opt/crosstool/settings-2.4.sh
```

(We are using the cross-compiler built for Linux 2.4.)

You can verify that your settings are correct by checking that certain executables are in your path. The `which` command will tell you what directory a command is in. Test these programs:

```
bash-2.05$ which xst
bash-2.05$ which xps
bash-2.05$ which powerpc-405-linux-gnu-gcc
```

If `which` says “no xst in (...)” then there is a problem.

Step.2 Download Existing Design

Next, we are going to download a working Xilinx Platform Studio project to give you some familiarity with the user interface.

- Download the package from the web:

```
wget http://ronsass.net/esd/hello.tar.bz2
```

- Uncompress and unpack this project:

```
tar xvj hello.tar.bz2
```

- Change directories and open the project with Xilinx Platform Studio:

```
cd Hello
xps hello.xmp
```

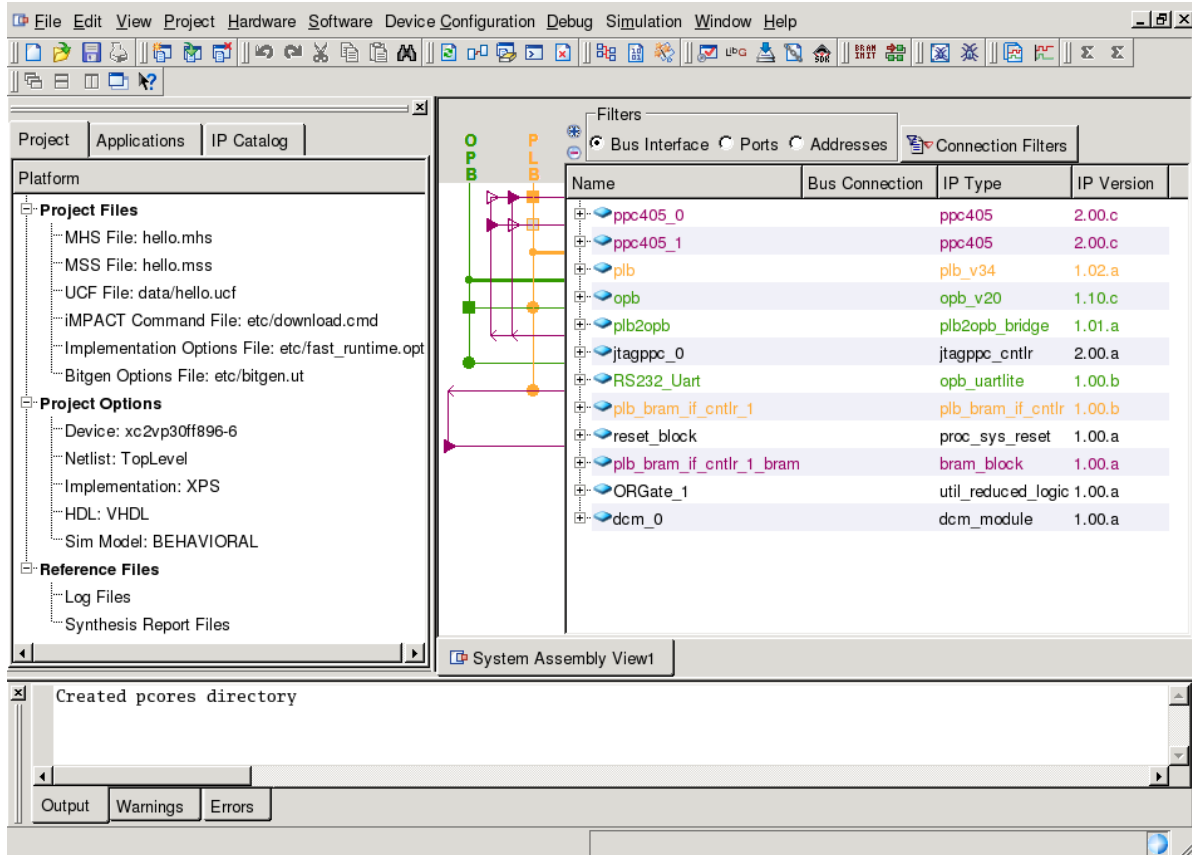


Figure 1: the main XPS window

At this point, the main XPS window should appear and look something like the window in Figure 1.

This is a simple “Hello, world.” project. The hardware consists of one PowerPC core (the second one is idle), an RS-232 serial port on a peripheral bus (OPB), and some main memory (built from Block RAMs) on the processor bus (PLB). There are also two cores that bridge the OPB and PLB buses. This is a little bit of extra logic to handle the clock, reset, and JTAG interfacing to the PowerPC.

These components and their connections are listed in the system view area (on the right side).

On the bottom is a console window (which will log the output from various commands that XPS invokes).

On the left side is the project information area.

Of course, across the top are menus and a row of buttons that provide a short cut to specific menu items.

Take some time to look at the various menu options. Pay close attention to the selections under “Hardware,” “Software,” and “Device Configuration.” Also, explore the system information window (the tree of components is compressed by default, but you can expand some items).

Step.3 Generating A System

Now we will use the tool to start generate the hardware and software part of our system.

Hardware The normal flow for hardware generation is synthesis to a netlist, mapping, place-and-route, and then bitstream generation. This is accomplished with two buttons in XPS. Locate the “Generate Netlist”



button (). Pressing this button will do several things. First, the top-level synthesis files, necessary wrappers, and signals needed build the system shown in the system information window are generated. Next, these and any peripheral cores are synthesized. This takes about 5 minutes on homer, if nothing else is running. The second button, “Generate Bitstream,” is to the right of Generate Netlist button and will finish the rest of the steps. This also takes about 5 minutes and you will notice a `hello.bit` file in the `implementation` subdirectory created by XPS.

Software The Embedded Development Kit (EDK), of which XPS is part of, includes a simple cross-compiler and is the default. We’ll use it for simplicity. In the Project Information area, click on the Application tab. If you open the sources tree, you’ll see a `hello.c` file listed. Double click on that file to view it in a window. To compile this program also takes two buttons. Locate the “Generate Libraries and Drivers”



button () and press it. This compiles software libraries and peripheral device drivers needed for this application. The button to the right cross-compiles “`hello.c`” and links it to the libraries. The final executable is in `HelloApp/executable.elf`.

Now we need to combine the hardware and software. Since our main memory is in BRAMs, we modify the bitstream (`hello.bit`) so that when the FPGA is configured, the BRAM’s initial contents is our application (`executable.elf`). This is called BRAM initialization; press the button that says “BRAM INIT” and XPS will create a new bitstream in the implementation directory called `download.bit`.

At this point, one could power-on an ML-310 board and use a JTAG programmer to configure the FPGA with the `download.bit` file. However, we will use a spare slot (7) on the compact flash so that we can configure over the Internet.

To use the compact flash, we need to create an ACE file. This is done with the `xmd` command in a terminal window (not through XPS).

- From the Hello project directory, go the implementation directory:

```
cd implementation
```

- Run the xmd with a genace script:

```
xmd -tcl $XILINX_EDK/data/xmd/genace.tcl -jprog \  
-hw download.bit -ace hello.ace -board ml310
```

The message

```
SystemACE file 'hello.ace' created successfully.
```

indicates that you have built a base system.

Step.4 Boot Base System

There is one ML-310 board directly connected to rsass-homer. For this assignment, everyone will share this board. To maintain order and prevent two people from simultaneously accessing the board, you must use two commands for controlling the board. (If someone else is using the board, the first command will exit immediately and tell you the username of the person using the board.)

Once you have an ACE, the steps to download that ACE file to the board are:

One. Start a new session with the ML-310 board by running the `ml310-session` command on the machine `rsass-homer`.

```
rsass-homer% ml310-session
```

This will bring up a new window and, with a two second delay, power on the ML-310 board. It takes about 20 seconds to boot to a menu that will give you details about what ACE files are currently installed on the compact flash. Figure 2 shows a typical boot menu.

From the menu, one can select the next slot to boot (type a number between 0 and 7 and hit *enter*), boot a new ACE file, or shutdown the board.

These commands can also be executed remotely. Back on `rsass-homer`, you can use the command `bootctl` to do the same. Also, `bootctl` can be used to transfer an ACE file to a slot of the CompactFlash. By convention, we always use slot 7 for testing. If you have an open session on the ML-310 board, you are allowed to overwrite any files currently in slot 7. To transfer your `hello.ace` file, type the following:

```
rsass-homer% bootctl upload 7 hello.ace "ron's test"
```

The first argument to `bootctl` is the command (upload, select, boot). The upload and select commands take a second argument which is a slot number. The upload command has third argument which is the ACE file to upload. There is an optional (but recommended) fourth argument to upload which is a short description. Since everyone in the class is creating a `hello.ace` file, this is a chance to add a little bit of identifying information to help ensure that you are booting your ACE file (not one that was just left on the compactflash).

The upload command will take several seconds and when it is finished, the screen on the ML-310 session will refresh. In this case, slot 7 should say,

```
(7) hello.ace (ron's test; 106af7203aae9ba2ed568d9338ebfbab)
```

```

      _/_/_/      _/_/_/      _/_/_/      _/      _/
      _/      _/      _/      _/      _/      _/_/_/      _/_/_/
      _/_/_/      _/      _/_/_/      _/      _/      _/      _/      _/
      _/      _/      _/      _/      _/      _/      _/      _/
      _/      _/      _/_/_/      _/_/_/      _/      _/_/_/      _/_/_/

slot ace file (description; md5sum)
-----

==> (0) bootctl.ace (RCS Lab Remote Boot; e73146104f23e3b6c72c37c089d2051f)
      (1) ml310_pci_linux.ace (Linux w/PCI; af289410060b99c1dc901e61c9beae3f)
      (2) ml310_diags.ace (Original POST; 6886e8a3929c4f0ffdf1f4011e3d692c)
      (3) <no ace file>
      (4) linux_boot_hda1.ace (Boot Hard Disk; c5ff10b074314cd783e0af401bb5d15a)
      (5) <no ace file>
      (6) <no ace file>
      (7) PLBMemTest.ace (Andy Test 3; 106af7203aae9ba2ed568d9338ebfbab)

Enter slot number, 'b' to boot, 'h' to halt:

```

Figure 2: the ML-310 boot menu

The string of 40-hex characters is an md5sum checksum. If you type (on Homer)

```
rsass-homer% md5sum hello.ace
```

you will get a checksum. When ACE was uploaded, the ML-310 also computed a checksum. If these two numbers match, then it highly probable that there were no errors in transmission.

Now it is time for you to boot your ACE file. In the ML-310 session window, type 7 (and hit *enter*) and then b (and hit *enter*). The boot menu program will do a soft reset on the ML-310 and everything after the `Restarting system.` line is the new ACE file.

After verifying that your program worked, you want to turn off the board. This is accomplished by exiting your ml310-session. Type *control-A* and then hit *X*. It will ask you if you want to Exit without Reset. Hit *enter* to confirm Yes.

The window will disappear, the ML-310 board will be turned off, and the board will be free for someone else to use it.

NOTES:

- Whenever you have the option, always halt the board before leaving your ml310 session. This is not possible for stand-alone C programs (like today's assignment) but this is the only case.
- Always use slot 7!
- Do not use the bootctl command unless you currently have an open ML-310 session.

Also note that you have to have permission to use the ml310-session command. You will need to demo a working ACE file to a TA or the Instructor your first time. After that, you'll be given permission to access it freely.

3 Assignment

Now that you know the steps to synthesize, let's create this system from scratch using the Base System Builder wizard. Start by creating a new, empty subdirectory. In that directory, start `xps`. (Note: last time you started XPS with the file name of a project; this time just start the application.)

The XPS window will appear and a window will pop up asking if you want create a new project (with Base System Builder) or open an existing project. Choose the default which is the wizard.

- type the name of your project “hello” (click next)
- choose new design (click next)
- select Xilinx as the vendor, ML-310 evaluation board as the platform (click next)
- leave processor set to PowerPC (click next)
- click no debug, leave other defaults (click next)
- leave UART checked, but uncheck DDR_SDRAM and SPIEEPROM (click next)
- uncheck LEDs, LCD, pci_arbiter, pci_bridge (click next)
- uncheck SysACE, IIC_Bus (click next)
- select 64 KB of BRAM (click next)
- uncheck Memory Test, leave Peripheral Self Test (click next)
- leave defaults (locates program to BRAMs) (click next)
- click ‘Generate’ to create project files
- click ‘Finish’ to return to XPS

At this point, you can recreate the hardware of Step 3. Before compiling the software, go to the “Applications” tab of project area, go to source code (under Test Peripheral), edit main to insert a `printf` with a Hello *your name* message.

Now finish building an ACE file and test it on the ML-310 board.

4 Grading

This lab assignment is due Friday 11/10 by 5pm (EST). To receive credit, you must meet with either the T.A. or the instructor and demonstrate that you've completed the lab. By default, this can be done in Woodward Hall room 237 (the Unix lab). Alternatives possible but need to be arranged in advance. Be prepared to answer in person any questions in the lab or recompile your program as part of demonstrating that you completed the lab. Do not wait until the last minute to begin the project; extensions will not be granted.