Fundamentals of Reconfigurable Computing ECGR6090/8090

COMPILING LINUX

1 Objective

The goal of this lab is to introduce you to the process of configuring and compiling the Linux Kernel for a Platform FPGA. This lab will be the foundation on which future labs will be built. Therefore, it is important to understand what each step is actually doing. This may require you to consult additional documentation (links provided in the Appendix at the end). Unlike previous labs, no tutorials accompany this lab. Read the lab through completely before beginning these steps. Missing a single step could result in hours of debugging, so **Read Each Step Carefully!**

At the end of the lab you should be familiar with:

- building a linux specific base system for the ML310
- compiling the linux kernel
- configuring linux kernel for use with the ML310
- using a cross-compiler

2 Lab Setup

To begin, ssh into Mosaic Linux server **lxs-sm1** and create a **lab3** directory. Then copy the following archive files into the directory:

1. **scp rcs.uncc.edu:/build/ecgr6090/lab3/linux-2.6-virtex.tar.bz2 .**

   The 2.6 Linux Kernel you will use for this lab

2. **scp rcs.uncc.edu:/build/ecgr6090/lab3/mkrootfs.tar.bz2 .**

   The Root File System used with Linux

3. **scp rcs.uncc.edu:/build/ecgr6090/lab3/busybox-1.8.2.tar.bz2 .**

   Tool to build additional Linux tools for our File System

Getting Started

After copying the three files to your **lab3** folder extract each of the files. Once you have extracted the three folders take a minute to look into each of the directories. Don't modify anything yet, just look around to see what each folder contains and start to think about what you might have to do within each folder.
In this lab we will be using Xilinx's **9.1i** tools just as you did in previous labs.

```
source ~rsass/opt/xilinx/9.1i/settings.sh
```

We will also be using a Cross-Compiler for this lab. There are multiple cross-compilers to use, but we will use one specific cross-compiler (cross tool's gcc-4.2.1). To use the cross-compiler, you will need source the following file. (**Note:** Just like the Xilinx tools you will need to source this file everytime you open a new terminal).

```
source ~rsass/opt/crosstools/settings.sh
```

To verify you have correctly sourced the cross-compiler correctly:

```
which powerpc-405-linux-gnu-gcc
```

3 Building the Base System (Hardware)

While the focus of this lab is the linux kernel, we will begin by building a base system which our compiled Linux system will run on. Within your **lab3** create a folder called **build**. Change into this directory and run XPS. Save the Base System into the **build** directory called **lab3.xmp**

Create a Base System with the following configurations:

- 300 MHz processor and 100 MHz Bus
- Use JTAG Debugging, but No Cache, and No On-Chip Memory
- Select OPB UartLite (check *use interrupt*)
- Select DDR on the PLB (check *use interrupt*)
- Uncheck SPI-EEPROM
- Uncheck LEDs, LCD, PCI Arbiter and PCI Bridge
- Uncheck SysAce_CompactFlash
- Uncheck I2C
- Select 64 KB of PLB BRAM
- Uncheck Memory and Peripheral Tests

Now that you have a base system we are only going to modify a few Software Configurations. In future labs we will create our own hardware IP cores and integrate it with this base hardware and with Linux.
In the past labs we created standalone applications to perform some specific function. With Linux we do not create any applications (like the previous labs), but rather rely on Linux to be our application. To do this we must configure the **Software Platform Settings**.

1. From the Menu Bar select **Software** and select **Software Platform Settings**
2. A Popup Menu opens, look towards the bottom and you will see a dropdown box called **OS**.

   Change this from *standalone* to *linux_2_6*
3. Look to the menu at the left and Select **OS and Libraries**
4. Set *connected_peripherals* Current Value to include the hardware you added during the Base System Builder. (RS232, OPB_Int)

   On the Row with Connected Peripherals click on "Edit" under the Current Value column

   This opens a new window, simply click OK to select all of the peripherals
5. Set *Memory Size* to **0x08000000** (be very careful - double check it)
6. Set *UART Frequency* to **100000000** (100 MHz)
7. Leave all other settings as defaults
8. Click OK

Next, Generate the Libraries and BSP (Board Support Package). From the Main Menu select **Software** and then select **Generate Libraries and BSPs**. This step creates the **xparameters_ml300.h** header file which contains specific information that Linux during compilation. This file is located in:

```
lab3/build/ppc405_0/libsrc/linux_2_6_v1_00_a/ \
linux/arch/ppc/platforms/4xx/xparameters/
```

Copy the **xparameters_ml300.h** file in this directory to:

```
lab3/linux-2.6-virtex /arch/ppc/platforms/4xx/xparameters/
```

We will then configure Linux to use these parameters when it compiles the kernel. Otherwise if we neglect this step and compile linux the Operating System will not know how to communicate with our hardware (specifically it won't know the address range of our hardware like UART and BRAMs).
Finally, we can now build our bitstream and initialize our BRAMs. But wait, initialize BRAMs? We need to create a download.bit, but what application are we going to use? This is where the bootloop is used. It is critical that we do not create a standalone application and keep PPC405_0's bootloop *Marked Initialize BRAMs*. All download.bit will do is run the PowerPC405 looping to 0xFFFFFFFC which will point to the Linux Kernel in RAM. So next we will have to configure the software to use this hardware. **The synthesis will take about 15 - 30 minutes**. While your design synthesizes continue on and configure the software system. **Note:** If you open a new terminal you will need to setup the Cross Compiler environments again.

4 Configuring the Software System

The following subsections will discuss how to configure the root filesystem, use a third-party software (BusyBox) to fill the root filesystem with executables (*ls*, *cd*, and *more*) and configure the kernel for our embedded system.

A. Configuring the Root Filesystem

To begin Change Directories into *mkrootfs*. Execute the following command:

```
sh mkfhs.sh
  - Notice this created a folder called rootfs
```

Next Change Directories to *busybox-1.8.2*. Execute the following commands:

1. **make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- defconfig**
2. **make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- menuconfig**

An ASCII Menu will appear - make the following changes:

- Select **Busybox Settings**, then Select **Installation Options**

  Set BusyBox Installation Prefix to **../mkrootfs/rootfs**

Now to build the executables which will be used within our system type:

1. **make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu-**
2. **make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- install**

If you are wondering how we are creating executables on the Mosaic Linux Server that will work on our PowerPC405, look into the *config.sh* script in *mkrootfs*. You do not need to modify anything in this script, it is already configured for use on Mosaic Linux Server.

B. Create a Simple Application

Change Directories back to *mkrootfs*. Create a file named **lab3.c** with the following functionality:

- Print your first name and last name
- Prompt the user for a number between 1 and 200
- Print the summation of all of the odd numbers from 1 through the number the user entered.

C. Using a Cross-Compiler

First, use GCC on the Mosaic Linux Server to simply compile and run your program and verify everything is fully functional (name the executable **lab3_mosaic**). Then use the *powerpc-405-linux-gnu-gcc* cross-compiler to compile lab3.c for the PowerPC on the ML310 (name the executable **lab3_ppc**). Copy **lab3.c** and **lab3_ppc** to **rootfs/root/** so that we can later execute our program on the ML310.

D. Creating the RAM Disk

Run the following Shell Script (in *lab3/mkrootfs*):

```
sh mkext2.sh
```

This creates an image which will be loaded into RAM after bitstream has been configured on the FPGA. It will compress the image and store it in **linux-2.6-virtex/arch/ppc/boot/images/ramdisk.image.gz**. When we compile the Kernel later on it will use this image along with our configuration settings to produce an executable (ELF) which we will run on the PowerPC405.

E. Configuring the Linux Kernel

Change Directories to *linux-2.6-virtex*. We will now configure some of the settings for the 2.4 Linux Kernel to run on the ML310. We have already configured some of the configurations, and now you will configure the rest. Future labs will expand on the complexity of the configuration you will preform, but for now just make these changes.
To begin type:

```
make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- ml300_defconfig

make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- menuconfig
```

to setup the Linux Kernel to the Default ML300 configurations. Next, the **menuconfig** opens the ASCII Configuration Menu to allow specific configuration modifications.

- Under General Setup:

  'y' Initial RAM filesystem and RAM disk support

- Under Platform Options:

  'y' Default bootloader kernel arguments

  Set Initial Kernel Command String to: **console=ttyUL0,9600 root=/dev/ram**

- Under Device Drivers:

  Under Character Devices, Serial Drivers:

  'n' 8250/16550 and compatible serial support

  'y' Xilinx uartlite serial port support

  'y' Support for console on Xilinx uartlite serial port

  'n' Network Device Support

  'n' HID Devices

  'n' USB support

5 Compiling and Running Your System

Now we must compile the kernel (approximately 5-10 minutes) so that it can run on our PowerPC405. Then we will create our executable to run on the ML310 so we can boot into Linux and impress all of your friends.

A. Compiling the Kernel

Change Directories to *linux-2.6-virtex*. To Compile Linux run the follwing two commands. Look to the notes to see if you can figure out what they are doing.

```
make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- clean
make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- zImage.initrd
```

Now you should have **zImage.initrd.elf** in **arch/ppc/boot/images**. Unlike the previous labs, we only have a bootloop running by default. In fact we do not have any other standalone application running. This program will run linux out of RAM (as specified by our configuration settings).

B. Creating an ACE File

Change back to your **lab3** directory and create a folder called **images**. Copy **zImage.initrd.elf** that you just created into this folder. Copy **download.bit** from your *build/implementation* folder.

Using XMD create **lab3.ace** which you will then download to the ML310 with *ml310-session* like in previous labs.

```
xmd -tcl ~rsass/opt/xilinx/9.1i/edk/data/xmd/genace.tcl -jprog \
  -hw download.bit -elf zImage.initrd.elf -board ml310 -ace lab3.ace
```

C. Downloading and Testing on the ML310

Finally, download *lab3.ace* to the ML310. Be patient, it will take approximately 2-3 minutes to synchronize the Compact Flash and then another 30-60 seconds to boot into your compiled linux. You will known it works if you see a login prompt. The username is **root** and there is no password.

Press Enter and you will see a # symbol for a prompt. You are now in the tiny linux you compiled. You can change to the */root* directory and run your *lab3_ppc* program. It is important to note that this is NOT a full version of Linux. This has only the most basic of functionality.

When you are ready to close the terminal you must first type **halt** to shutdown the system. It is very important to do so otherwise you run the risk of corrupting the File System. When you see **System Halted** it is safe to close the terminal and release the board.

6 Grading

This lab assignment is due **Tuesday October 30th, 2007 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). Alternative times are 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.

Appendix

These are some links to documentation you may find helpful while doing this lab. These are meant to provide you additional information as to what you are doing. If you come across something in the lab and cannot find a link below that explains what is happen, remember you are a grad student and all grad students know how to use **google**.

It is important that you read over everything first and then go slowly through the lab. You may have to restart the lab a few times if you make mistakes, it is perfectly natural, but **Start Early**. Do not wait until the day before the lab is due to get started.