Fundamentals of Reconfigurable Computing ECGR6090/8090


# LINUX DEVICE DRIVERS

## 1 Objective

The primary goal of this lab is to combine what you have learned and implemented from the last three labs into a typical working system. The goal is for you to learn how to write a simple Linux device driver and application. The device driver, which is part of the operating system, will allow you to read and write to the hardware core that you have created. Specifically, the skills you should learn in this lab exercise are:

- How to compile a Linux Device Driver as a module
- Write simple char device driver
- How to pass data between hardware and an application

In short, the objective of this lab is to teach you how to integrate all of the skills that you have been learning piece-wise in the last three labs.

## 2 Overview

The general steps will be for you to create a hardware design with your CRC core. Next download and compile a linux-2.6 kernel. Then you will build a rootfile systems with a linux device driver and an application to use your CRC hardware core. The last step is to go back to the linux-2.6 kernel, create a ELF image with an initial ramdisk, and then create the ACE file. (NOTE: this lab only describes the new steps. If necessary, review the tutorials and previous labs if you have any questions on how to complete a step!) **Read these directions carefully and completely before you begin.**

## 3 Lab Setup

Unlike in previous labs, you will be using *Linux 2.6* with your hardware. There is a noticeable difference between Linux 2.4 and Linux 2.6 configuration menus. Make sure you follow the directions within this document carefully. It would be a good idea to look around at the configuration menu options to see how things have changed.

To get started you will need to create a new folder called **lab06**. It will probably be helpful to create two more sub-directories as well: (1) **build** for your hardware design and (2) **images** to build an ACE file image. You will need to extract three TAR files from **/build/ecgr6090/lab6** on Homer into your **lab06** folder:

- linux-2.6.tar.bz2
- mkrootfs.tar.bz2
- busybox-1.8.1.tar.bz2

## 1. Create Base Hardware

Create a Base System for the ML310 with the following hardware:

- 300 MHz Clock, FPGA JTAG, and No OCM
- RS232_Uart OPB UART16550 (**Not UARTLITE** (check Use Interrupt)
- PLB DDR
- 64KB of PLB BRAM
- Unselect all other options (including Test Applications)

Include the CRC Hardware Core you created in Lab 5 or copy it from **/build/ecgr6090/lab6/my_crc_core.tar.bz2**. Copy the entire folder into your **lab6/build/pcores** directory. Then from the IP Catalog add your core to the system. (**Hint**: You might need to Rescan User Repositories and then look under Project Local pcores). Connect your Core to the OPB and generate an address range for your core.

## 2. Software Platform Settings

Similar to Lab 3 and 4 you will need to modify the Software Platform Settings. Since you are using Linux 2.6 now you will need to make a few different changes:

- Set the **OS** to **Linux_2_6**
- Add RS232_UART and opb_intc_0 to the Connected Peripherals List
- Set the memory size to: 0x08000000
- Set Uart 16550 Frequency to: 100000000
- Set Target Directory to: ¡absolute path to lab6 directory¿/bspfiles

  I.E. /home/aschmidt/ecgr6090/lab6/bspfiles

**Generate Libraries and BSPs** followed by **synthesizing** the hardware to create a download.bit.

## 3. Configuring Linux-2.6 Kernel System

Copy the **xparameters_ml300.h** parameter file created when you generated the libraries and BSPs to the appropriate linux-2.6 sub-directory.

```
lab06/bspfile/arch/ppc/platform/4xx/xparameters/xparameters_ml300.h
```

**Note:** When configuring the 2.6 Kernel the commands used differ slightly from 2.4. Now you need to specify on the Command-Line two parameters: (1) ARCH - the hw architecture of the system you want to compile Linux for and (2) CROSS_COMPILE - specify the cross compiler prefix to use. Then you specify the command you want to run.

```
make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- <COMMAND>
```

In this lab we will begin with the command **menuconfig** to bring up the text-based configuration menu. Make the following changes to the Linux Configuration:

- 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=ttyS0,9600 root=/dev/ram**

- Under Device Drivers:

  'n' HID Devices

  'n' USB support

- Under Kernel Hacking:

  'n' Include xmon kernel debugger

The other two commands to use are **zImage** and **zIamge.initrd** which you will use at the appropriate time in the lab.

## 3. Create Root File System and RAM Disk Image

To begin, run the correct script to Generate your Root File System folder. Then change directories to busybox-1.8.1. In Lab 3 you used an older version of BusyBox; however, in version 1.8.1 there have been some updates which simplify the build process. As with Linux you need to specify the Architecture and Cross Compiler. Then you specify the command you want to run.

- make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- defconfig
- make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- menuconfig
- Under BusyBox Settings, Under Installation Options:

  Change BusyBox Installation Prefix to point to your **rootfs**

- make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu-
- make ARCH=ppc CROSS_COMPILE=powerpc-405-linux-gnu- install

Now copy your software-only crc binary you created in Lab 3 to your rootfs. You will be using your software crc implementation as one method to verify the functionality of your device driver and hardware.
Create directories under mkrootfs:

- **Device driver** - this folder contains all the necessary code for the device driver.
- **Application** - this folder contains all the necessary code for the user application that will use your device driver to communicate with your hardware.

Compile your application and copy it into the **rootfs/root** directory. Compile your device driver and copy it to the **rootfs/root** directory.
Finally, update your device file (in the mkext2.sh) to make a device with the right major/minor numbers, then create a ramdisk image.

### 4. Create Images

Create the RAM Disk Image and make sure it is copied over to the correct Linux 2.6 sub-directory. Create the software executable zImage.initrd.elf and copy it to your images folder. Copy your download.bit from your build/implementation folder into your images folder. Create an ACE file and test it on an ML310. Compare your output with both your software CRC implementation and the cksum program from BusyBox.

**Note:** Username is **root** and there is no password to login to Linux.

## 4 Assignment

In subsection three, you added a device driver and application to your RAM disk image. Your assignment is to create a device driver that supports your CRC core and an application to test the device driver.

The test application is what a user will interact with. You will need to be write the application such that it will use your device driver and hardware to compute the crc for an arbitrary file. It will:

- Accept the file name as an argument on the command line
- Open the file and the crc device driver
- Copy the contents of the file to device driver in 512-byte blocks
- Close the file
- Read the length and crc checksum from device driver

The device driver must support:

**initi/cleanup** – dynamically loadable kernel module

**open** – prevent two applications from using the same crc core at the same time; reset the state machine; initialize any variables

**write** – will accept variable-sized blocks of data and transfer them byte-by-byte to the crc core

note: if the core does not keep track of the length, the device driver will need to

**read** – ability for the application to read exactly two integers; the first integer is the length of the crc stream and the second is the crc value

if the application reads past offset 7 in the file, read should return "end of file" condition

The **shell.c** file from class is available under **/build/ecgr6090/lab6/** which you are free to use, but understand the code has been reduced to some of the most basic functions and you are required to add all missing code and error checking.

**NOTE:** Look into **mkrootfs/config.sh** to determine which version of the Cross Compiler you will need. Unlike Lab 4 you will need to set your PATH to a different version of the Cross Compiler tools.

**HINT:** Look into **/build/ecgr6090/lab6/shells** for templates for the Device Driver and Application. Also, there is a **README** which describes how to compile the Kernel Module. **Carefully read all of the comments in all of the files before beginning**. Make sure you add your own comments (including your name). Make sure you rename these files to be something more meaningful to you.

# 5 Grading

This lab assignment is due **TUESDAY DECEMBER 4, 2007 by 5:00pm (EST)**.

You will be asked to demonstrate your application/device driver. Specifically, we will want to see that you can:

- See your source application cross-compile on homer/marge
- insmod/rmmod your driver on a running system
- Run your application on an arbitrary file (i.e. checksum a file)
- Run cksum on the same file
- Run your software-only CRC on the same file

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.