Fundamentals of Reconfigurable Computing ECGR6090/8090

CRC IMPLEMENTED IN SOFTWARE

1 Objective

The goal of this lab is to become familiar with a software implementation of a Cyclic Redundancy Check (CRC). The overall objective of the next few labs is to compare a software implementation of a CRC to that of a hardware implementation of the CRC. To begin we must first understand what a CRC is and how to develop it in software. Since software development is much quicker than hardware development we will begin with the software implementation.

This lab builds on the previous lab by adding an additional software application that will then be included into the RAM Disk image. You may need to refer back to Lab 3's handout. *Read this lab handout completely before beginning!*
The goal is to come away with:

- learning what a Cyclic redundancy check is.
- learning how to implement a CRC is software.
- combining your CRC with your previous Linux base system.
- preparing you to take a software CRC and implement a hardware CRC core.

2 Lab Setup

This lab builds on Lab 3 so before beginning you will need to complete Lab 3. Once you have a working Lab 3 you will only be modifying software and reuse your previously created base system (hardware). This is why it is critical your lab 3 design is fully functional.

1. Getting Started

To begin, start by making a copy of your *lab3* and call it *lab4*.

```
cp -r ~/lab3 ~/lab4
```

Change Directories to your *lab4* folder. Create a folder called **sw_crc** which will contain your software CRC implementation. In the **sw_crc** folder copy a template C file which will contain some initial data you will need to get started. Do not modify the data, just add your code within the Main Function as you would with any other program. The file can be found on Homer:

```
/build/ecgr6090/lab4/crc.c
```

In **crc.c** you will see a Table which contains lookup values used by the CRC you will write. When you read about CRCs (from Wikipedia or whatever website / books you use during your research about CRC) you will see some implementations use tables and others do not. We will be implementing a Table based CRC since it is the current IEEE standard. For more information read about **cksum**. Changing the table will change your CRC calculation (which is bad).

The **Assignment Section** outlines the application you will need to implement along with references that can be used to help guide you while creating your application. When compiling your application, first compile it to run on Homer and call the program **sw_crc_homer**. Once you can verify its functionality on Homer, cross-compile the application to run on the PowerPC405 on the ML310. Name this application **sw_crc_ppc**.

Writing this software by yourself will greatly improve your understanding of CRC and significantly help your design when you have to implement the CRC in hardware in the next lab.

2. Recompiling Linux

Since you have already configured Linux in Lab 3 you will only need to modify the RAM Disk and recompile Linux. This subsection assumes you have already written your software CRC implementation and cross-compiled it for the PowerPC405. Begin by copying your **sw_crc_ppc** application into your rootfs/root. You will also need to copy some small test file so that you can run your CRC on that file.

```
cp ~/lab4/sw_crc/sw_crc_ppc ~/lab4/mkrootfs/rootfs/root/.
cp <some test file> ~/lab4/mkrootfs/rootfs/root/.
```

Following lab 3, recreate the RAM Disk Image by running **mkext2.sh**. This will write over your previous *ramdisk.image.gz*. Since you copied your linux-base folder from your lab3 folder you will need to reset the symbolic links. You will also need to clean your Linux Compilation. Run the following commands (you need to run each command individually, don't try and group them together or you will get an error about missing /include/asm files):

```
make clean
make symlinks
make dep
make zImage.initrd
```

This will recompile Linux with your new RAM Disk resulting in a new *zImage.initrd.elf*.

Change Directories to **images** and copy over your new *zImage.initrd.elf* (overwriting your Lab 3 elf). Using XMD create **lab4.ace** then use *ml310-session* to download and run on an ML310. Once you have booted into Linux run your *sw_crc_ppc* on your test file.

Helpful Hints: Before running anything on an ML310 it is strongly encouraged that you can first run your Software CRC on Homer. If it doesn't work on Homer, it isn't going to work on the PowerPC and you will just waste your time creating a new RAM Disk, new ELF, new ACE, and transferring / running your ACE file.

Homer (and most Linux distributions) have a program called: **cksum** which takes one parameter (a file name) and returns the CRC checksum. Compare your output with this **cksum**.

3 Assignment

Your assignment is to develop a C program that implements the Cyclic Redundancy Check (cksum) algorithm. You will write the program on Homer, compile it on Homer, and test it on Homer. Once you have a working program you will then cross-compile it for the ML310 and run it just like you ran lab3_ppc.

This will require you to recreate the RAM Disk to include you new program and any test file(s). Keep in mind the larger the test file or the more test files you include, the longer it will take to uncompress the Linux Kernel so try and keep them somewhat small.

There are many implementations of CRC, we are implementing **cksum**. From the *man* page of **cksum**:

```
The cksum utility shall calculate and write to standard output a
cyclic redundancy check (CRC) for each input file, and also write to
standard output the number of octets in each file. The CRC used is
based on the polynomial used for CRC error checking in the ISO/IEC
8802-3:1996 standard (Ethernet).
```

The program **cksum** is already on Homer and available for you to use to compare your results. You will know you have a working implementation when you can run your CRC cksum program on the same file as **cksum** and get the same checksum. **It is recommended that you read the manual page(s) for *cksum***. Try running cksum on some file you have and see what is output.

Since there are many different algorithms available online we will give you the basic algorithm. It is up to you to fully understand how the CRC works. You may (will) be tested on the functionality and might even be asked to write it on an exam. The algorithm is given in the **crc.c** file described earlier in the handout.

The test data can come from anywhere – a old ACE file, a list of words (`/usr/share/dict/words`, a C executable. However, you will want to compile the program with a native compiler first and test it on the host so that you know what to expect as output from the FPGA-implementation. The TA or instructor will give you another data file to test when you are ready to demo. **So you will want to make sure you can test different files easily.**

Sample Output:

```
------------ Beginning Program -------------
Name: Bond, James Bond
ID: 007
Enter File: test.data
Opening: test.data
Calculated CRC: 918749774
------------- Exiting Program --------------
```

The aim of learning this particular algorithm is three-fold. First, it is intended to help you exercise your C skills. Second, it is an algorithm with an obvious hardware implementation. It also highlights one of the common problems that crop up in reconfigurable systems when the host architecture and target architecture are different.

(**Hint:** Think about (draw block diagrams) how this would be implemented in hardware. For Lab 5 you will be required to only do the CheckSum (not opening files). The more you think about it the better you will understand it and the more prepared you will be for Lab 5 and Lab 6.

4 Grading

This lab assignment is due **Friday November 9th, 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.

References

For more information about CRC-32 (and CRC's in general), search google with the keywords `crc cyclic and cksum`. Wikipedia has a solid (but very mathematical) description. This is a often-used programming assignment, so there are lots of individual implementations. Be aware, though, that there are several *incorrect* solutions posing as correct solutions on the web. Many a student have been tripped up by this code!

To verify your programs functionality use **cksum** which is installed on Homer. It should be available on most standard Linux distributions. Once you have verified it works on Homer Cross-Compile it to run on the ML310. When running it on the ML310 you should get the same Checksum (assuming you are using the same data file).