

ECEN 449 Lab Report 4

Philip Smith - 624002014 (Sec. 511)

February 27, 2020

Introduction

This lab focuses around the use of Linux kernel modules, which are modules that can be loaded into the running kernel to add functionality to it without needing to restart the system. The lab also covers basic file system operations such as mounting and unmounting block devices, in this case, a micro SD card. There is also the creation of a Makefile to assist us in the compilation process for the kernel modules.

Procedure

The first step in this lab is to mount, traverse, and unmount a block device for persistent storage. This will be used later on when compiling and inserting kernel modules.

After this we will compile two kernel modules. The first is a simple Hello World! program whereas the second will make use of the multiplication peripheral we created in lab 3.

Results

When compiled and loaded on the Zybo board both kernel modules print out the correct statements. The multiply peripheral seems to be working as expected based on this output.

Conclusions

Overall this lab was a success with no major incidents to report. The Linux kernel modules compiled successfully and were easy to copy over to the processor we created for this lab and future labs.

Questions

1. I do not see any reason the board shouldn't be power cycled. The mnt directory is the only directory touched and it will still exist after a reboot.
2. On the CentOS System the mount point should be under the root directory /media/.
3. The name of the file changes the structure of the Makefile. If we for instance renamed it to test, the first line would need to go from hello.c to test.c.

There would not be consequences as both Lab 4 and 5 used the same Linux kernel sources (3.14).

Appendix A - Part 2 Console Output

The following is the console output immediately inserting the second multiplication kernel module:

```
zynq> insmod multiply.ko
Mapping virtual address...
Physical Address:      43c00000
Virtual Address:      608e0000
Writing a 7 to register 0
Writing a 2 to register 1
Read 7 from register 0
Read 2 from register 1
Read 14 from register 2
zynq> █
```