**ECEN 449 - Lab Report**

**Lab Number:** 5

**Lab Title:** Intro to Kernel Modules on Zynq Linux System

**Section Number:** 508

**Student's Name:** Samuel Fafel

**Date Due:** 03-30-2023

**TA:** Prajwal Holla

#### Purpose/Introduction:

This lab builds upon the previous in that we will compile and run a "Hello World" program as well as a multiplication program which utilizes our "multiply" peripheral developed earlier. After compiling them on the CentOS computer, we will transfer them to the Zynq Linux system and run them there.

#### Procedure:

After making a backup of our Lab 4 directory, we created a "hello.c" file which printed a simple "Hello world!" message to terminal. After compiling and transferring the file to the Zybo board, we ran it to ensure that it and the linux system worked properly.

After this, we made another file "multiply.c" which utilized our "multiply" peripheral, writing a "7" and a "2" to registers 0 and 1 which resulted in a "14" being written to register 2. After writing these values, we read from the registers and printed their values. We compiled this "multiply.c" file, transferred it to the Zybo board, and verified that it worked as intended.

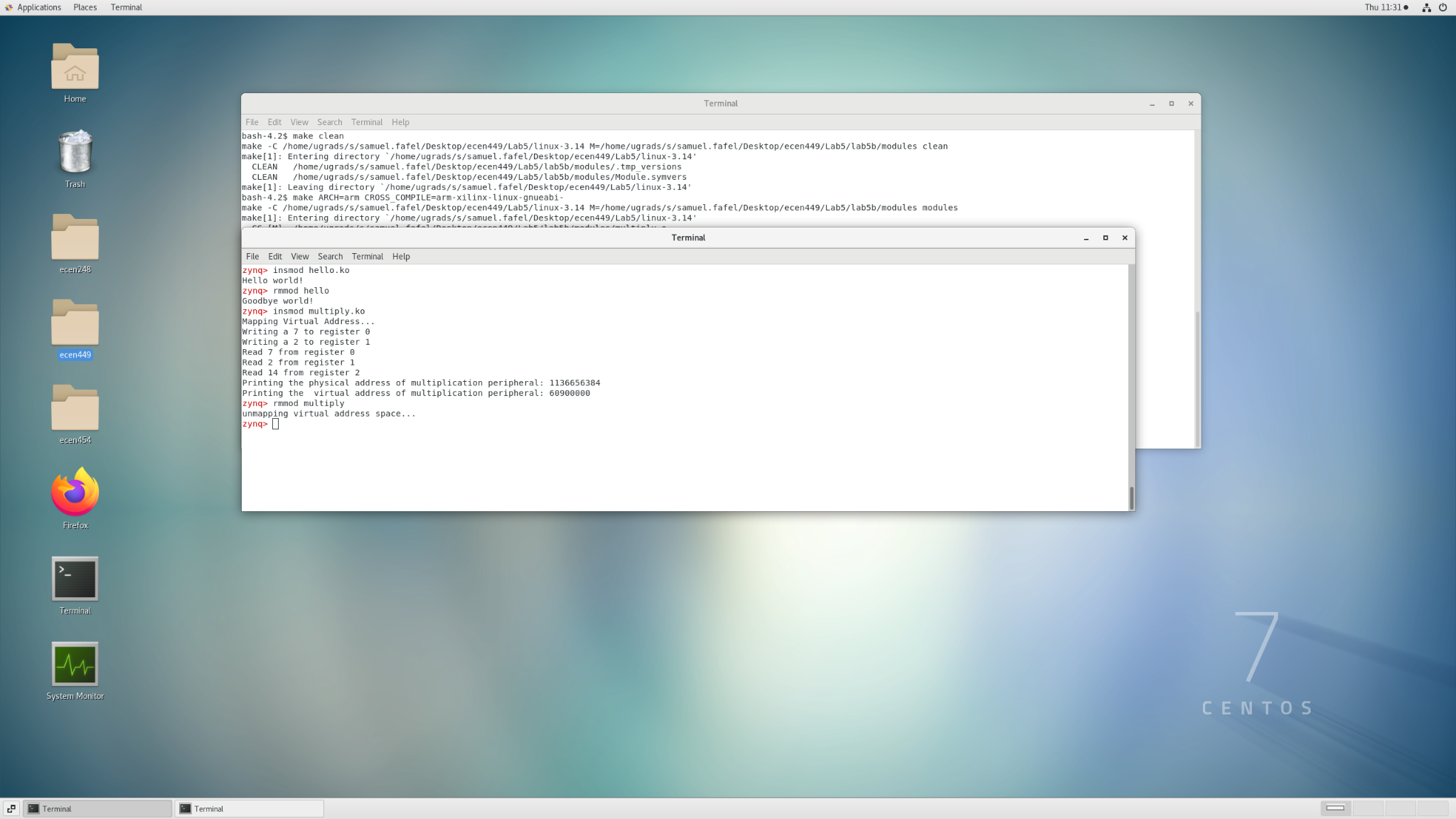
#### Results:

The "hello.c" file operates in a simple manner. On start (initialization), it prints "Hello world!" to the terminal, and on close (exit), it prints "Goodbye world!" to the terminal.

The "multiply.c" file operated in a more complex manner. Prior to initialization (during the compile stage), it defines the physical address and memory size using the "multiply" peripheral designed in previous labs. It does this by accessing the "xparameters" files made during that stage. Upon initialization, it maps the physical address along with the memory size to the virtual address pointing to the multiplier's register 0. Then, it writes a "7" to that address and prints out a statement confirming such. Then, it writes a "2" to the address next to that (virtual address + 4) which points to register 1 and again prints confirmation. Since we defined register 2 to be the multiplication of register 0 and register 1, we do not need to write to register 2. Instead, we read confirmation from registers 0, 1, and 2, and see that their values are equal to 7, 2, and 14, respectively. Upon exiting the program, the virtual address space is unmapped.

The primary challenge faced was writing the "multiply.c" file and getting through compilation without errors. Among other things, I had to look up how to print a void pointer type in order to print the virtual address. Other than that, the process was relatively smooth sailing.

See below for the screenshot showing output from both "hello" and "multiply".



This screenshot shows the proper output from the "hello.ko" and "multiply.ko" files on the Zynq Linux System.

#### Conclusion:

This lab taught us how to utilize kernel modules in the Zybo Linux System in addition to utilizing our "multiply" peripheral.

#### Questions:

1. In step 2.f, if we had power cycled the board, we would need to recreate the mounting directory.
2. The mount point for the SD Card is in the /media/ directory, under the user's name.
3. If we changed the name of our hello.c file, we would need to change the obj target in the Makefile to the new name (still with .o at the end). If we specified the lab 4 kernel directory instead of the lab 5 kernel directory, we would not be able to read the contents of the card when inserting it into the Zybo board.

#### Appendix:

All code was given in the lab manual.