Laboratory Exercise #5
Introduction to Kernel Modules on Zynq Linux System

ECEN 449-511
Seth Pregler

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

Introduction

The purpose of this lab was to learn how to cross compile simple kernel modules for our multiply peripheral created in Lab 3.

Procedure

The following is an overview of the procedure of implementing and testing the design for this lab.

1. Under the lab 5 directory, create a subfolder called modules and add the necessary boilerplate hello.c code
2. Create a Makefile and provide the path to the Linux kernel source code
3. Within the 'modules' directory, cross compile the using 'arm-xilinx--linux-gnueabi-'
4. Copy the hello.ko file into the SD card and insert the card into the FPGA
5. Mount the /mnt/
6. Run 'insmod' to insert the module into the kernel
7. Verify module is loaded using 'lsmod'
8. Create a 'modules' directory and retrieve the name and information about the current kernel using 'mkdir -p /lib/modules/`uname -r`'
9. Remove the module using 'rmmod' and then run 'lsmod' to verify the module is no longer running.
10. Repeat steps 1-9 using multiply.c source code, add necessary function calls to code and print the physical and virtual addresses of the multiplication peripheral to the kernel message buffer.

Results

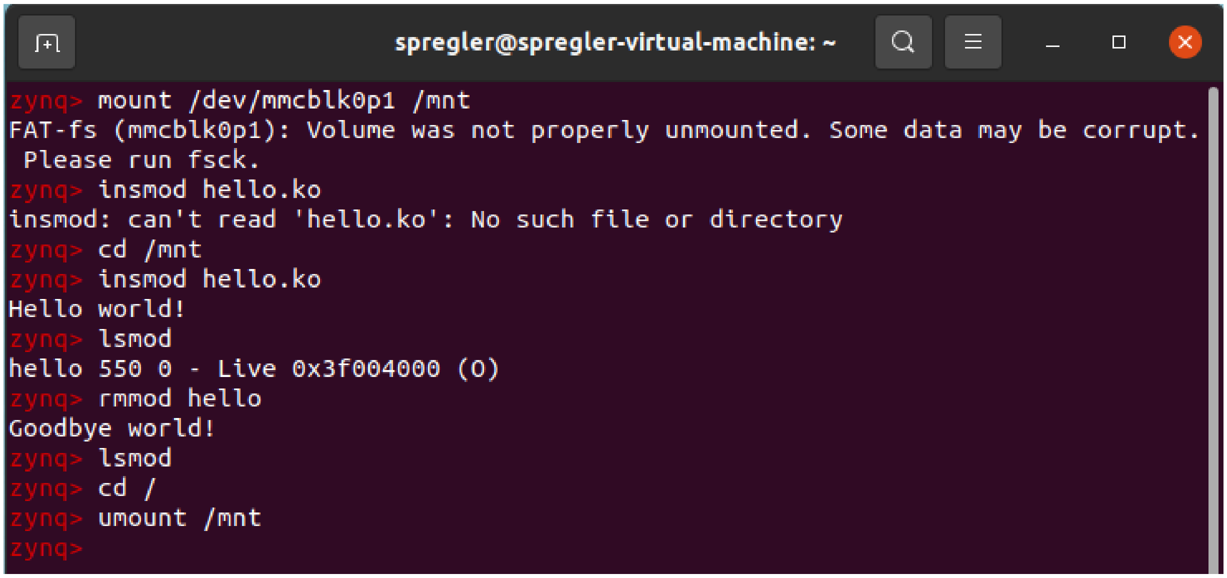
This lab was pretty straight forward and didn't require too much programming/debugging. Some challenges I ran into initially were long read/write times when copying/pasting the linux kernel source directory within the ECE filesystem. When connected to the ECE VPN, it would take 10-12 hours to do this. As a result, I did most of the work for this lab in person using the labspace. Outside of this, the lab was very straightforward and creating the kernel module went smoothly.

Conclusion

After completing this lab, I learned a bit more about cross compilation and how the linux kernel interacted with the multiplication peripheral developed in previous labs.

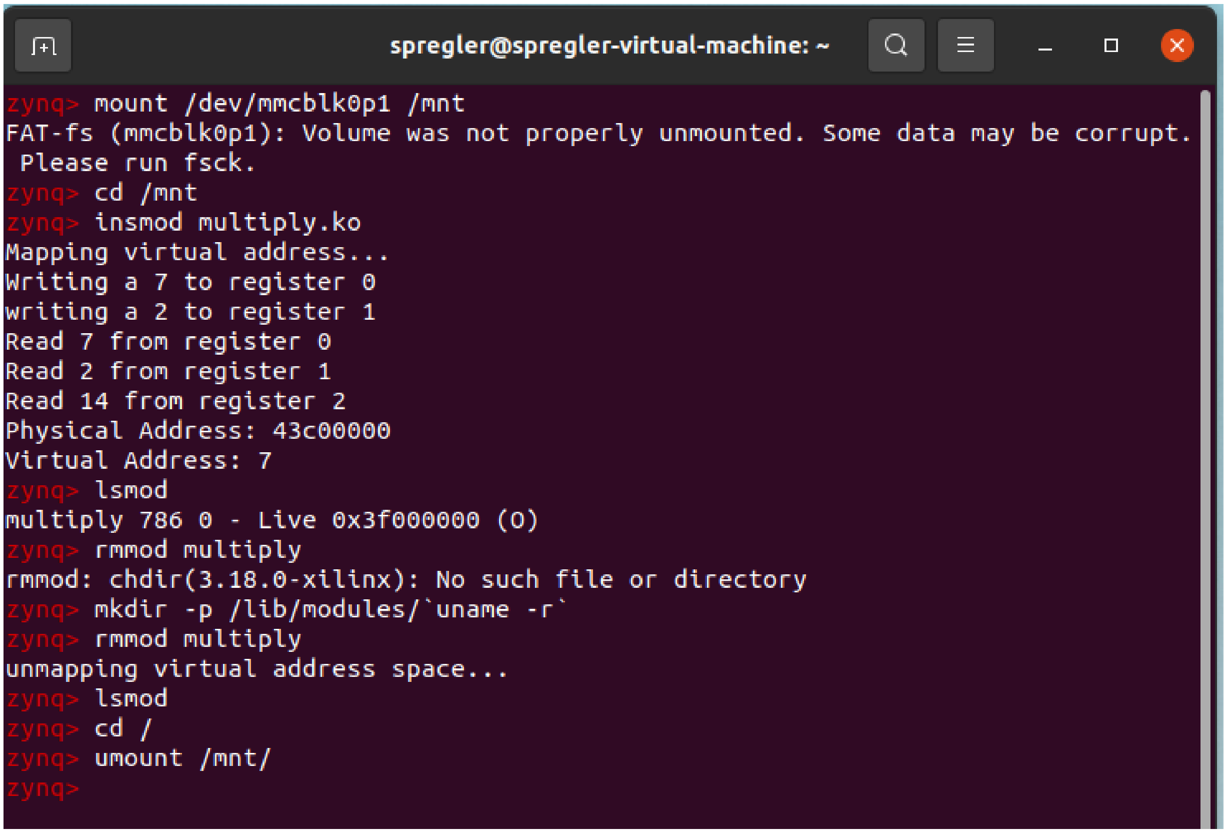
Output of the Terminal for Part II of Lab

hello.ko



```
zynq> mount /dev/mmcblk0p1 /mnt
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
 Please run fsck.
zynq> insmod hello.ko
insmod: can't read 'hello.ko': No such file or directory
zynq> cd /mnt
zynq> insmod hello.ko
Hello world!
zynq> lsmod
hello 550 0 - Live 0x3f004000 (O)
zynq> rmmod hello
Goodbye world!
zynq> lsmod
zynq> cd /
zynq> umount /mnt
zynq>
```

multiply.ko



```
zynq> mount /dev/mmcblk0p1 /mnt
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
 Please run fsck.
zynq> cd /mnt
zynq> insmod multiply.ko
Mapping virtual address...
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
Physical Address: 43c00000
Virtual Address: 7
zynq> lsmod
multiply 786 0 - Live 0x3f000000 (O)
zynq> rmmod multiply
rmmod: chdir(3.18.0-xilinx): No such file or directory
zynq> mkdir -p /lib/modules/`uname -r`
zynq> rmmod multiply
unmapping virtual address space...
zynq> lsmod
zynq> cd /
zynq> umount /mnt/
zynq>
```

Questions

(a) If prior to step 2.f, we accidentally reset the ZYBO Z7-10 board, what additional steps would be needed in step 2.g?

Resting the FPGA prior to mounting ‘/mnt/’ will reset the linux kernel on the board. Because we haven’t mounted anything yet, there are no additional steps, we just need to mount the SD.

(b) What is the mount point for the SD card on the CentOS machine? Hint: Where does the SD card lie in the directory structure of the CentOS file system.

/run/media/<student uin>/

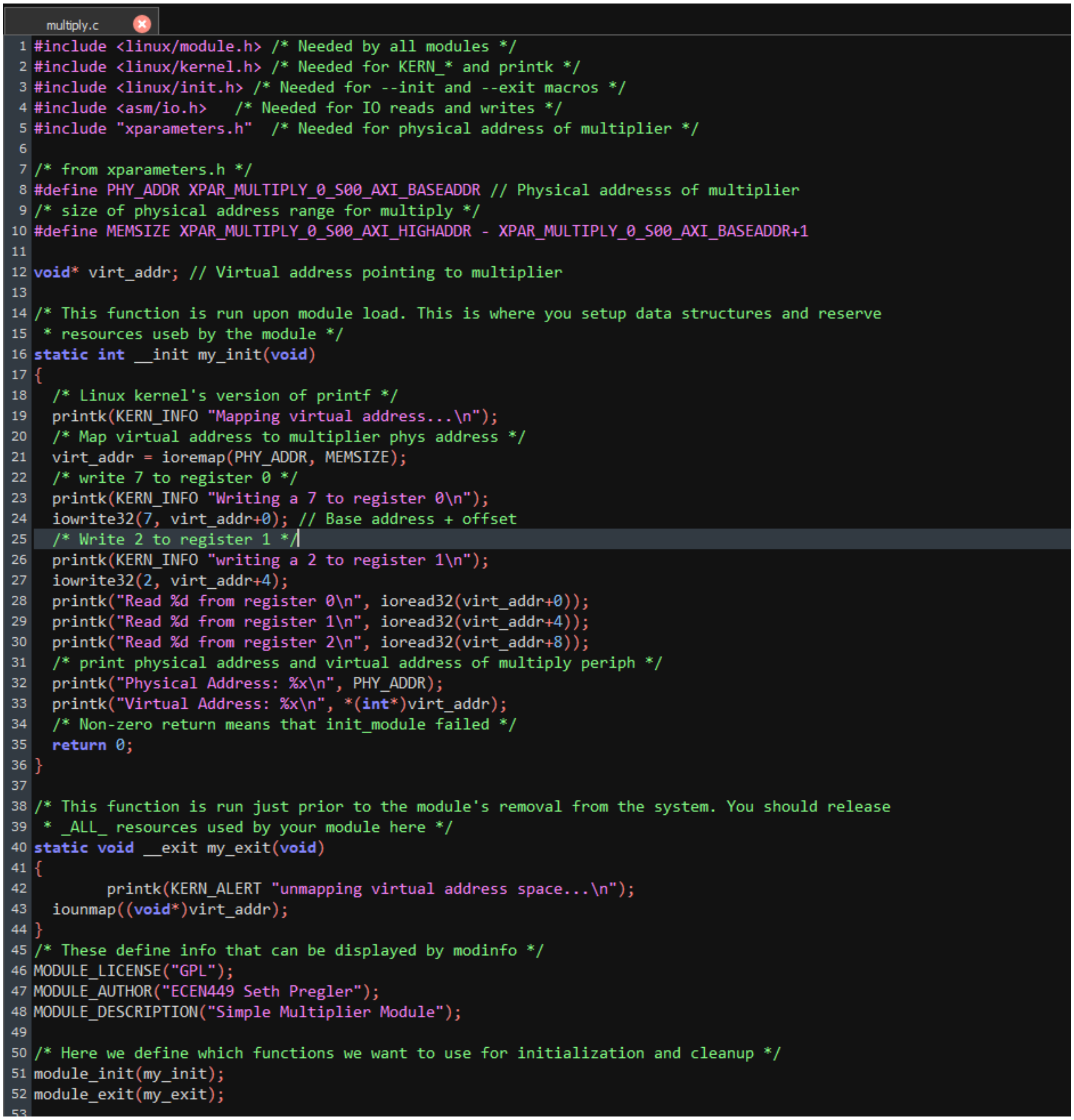
(c) If we changed the name of our hello.c file, what would we have to change in the Makefile? Likewise, if in our Makefile, we specified the kernel directory from lab 4 rather than lab 5, what might be the consequences?

If we change the name of the file, we need to change the name in the make file as well, namely the bit: “`obj m += <name>.o`” at the top of the makefile. If when we specify the kernel directory, we specify the path for lab 4, this may cause an error when copying from another directory.

Appendices

List of Contents

Appendix A
multiply.c



```c
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_* and printk */
#include <linux/init.h> /* Needed for --init and --exit macros */
#include <asm/io.h>    /* Needed for IO reads and writes */
#include "xparameters.h"  /* Needed for physical address of multiplier */

/* from xparameters.h */
#define PHY_ADDR XPAR_MULTIPLY_0_S00_AXI_BASEADDR // Physical addresss of multiplier
/* size of physical address range for multiply */
#define MEMSIZE XPAR_MULTIPLY_0_S00_AXI_HIGHADDR - XPAR_MULTIPLY_0_S00_AXI_BASEADDR+1

void* virt_addr; // Virtual address pointing to multiplier

/* This function is run upon module load. This is where you setup data structures and reserve
 * resources useb by the module */
static int __init my_init(void)
{
  /* Linux kernel's version of printf */
  printk(KERN_INFO "Mapping virtual address...\n");
  /* Map virtual address to multiplier phys address */
  virt_addr = ioremap(PHY_ADDR, MEMSIZE);
  /* write 7 to register 0 */
  printk(KERN_INFO "Writing a 7 to register 0\n");
  iowrite32(7, virt_addr+0); // Base address + offset
  /* Write 2 to register 1 */
  printk(KERN_INFO "writing a 2 to register 1\n");
  iowrite32(2, virt_addr+4);
  printk("Read %d from register 0\n", ioread32(virt_addr+0));
  printk("Read %d from register 1\n", ioread32(virt_addr+4));
  printk("Read %d from register 2\n", ioread32(virt_addr+8));
  /* print physical address and virtual address of multiply periph */
  printk("Physical Address: %x\n", PHY_ADDR);
  printk("Virtual Address: %x\n", *(int*)virt_addr);
  /* Non-zero return means that init_module failed */
  return 0;
}

/* This function is run just prior to the module's removal from the system. You should release
 * _ALL_ resources used by your module here */
static void __exit my_exit(void)
{
        printk(KERN_ALERT "unmapping virtual address space...\n");
  iounmap((void*)virt_addr);
}
/* These define info that can be displayed by modinfo */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Seth Pregler");
MODULE_DESCRIPTION("Simple Multiplier Module");

/* Here we define which functions we want to use for initialization and cleanup */
module_init(my_init);
module_exit(my_exit);
```