ECEN 449 - Lab Report

Lab Number: 5

Lab Title: Introduction to Kernel Modules on Zynq Linux System

Section Number: 501

Saira Khan

731005607

October 28, 2024

TA: Shao-Wei Chu

Introduction:

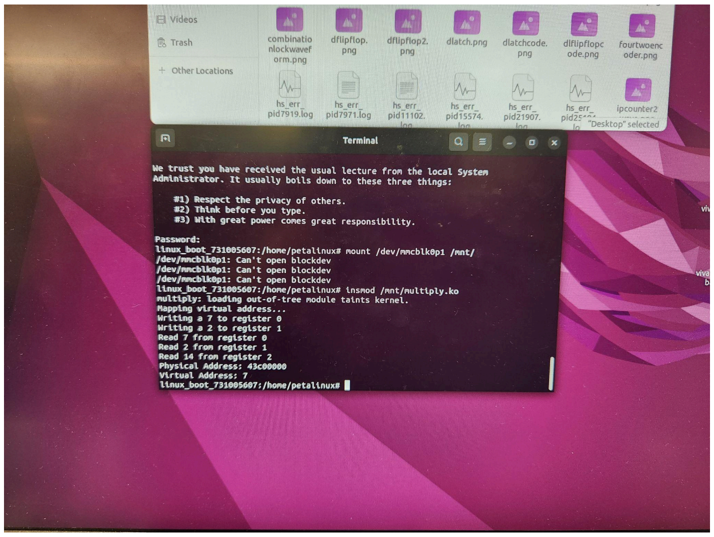
The purpose of this lab is to load a “Hello World!” and multiply module on the ZYBO Z7-10 board. PetaLinux was used to build and deploy these modulus onto Xilinx hardware.

Procedure:

The procedure of this lab was similar to that of the last lab. Linux was booted onto the ZYBO Z7-10 board and the SD card was mounted so that files could be read and written to it. After that, the SD card was used to transfer files between the CentOS workstation and the ZYBO Z7-10 board. The first module built was the “Hello World!” module. A PetaLinux project was built following the steps from the previous lab. Once a project was built and navigated into, a hello module was created. A hello.c, Makefile, and README file was created. The Makefile contains targets to build and install the module into the root file system. The README file is a file to introduces how to build the user module, and the C file is a simple kernel module in C. Code was copied into the C file, and the project was built. After the project was successfully built, a hello.ko file was created in the directory of the project. A .KO file is an object file that is used to extend the kernel of the Linux Distribution. This file was then put on the SD card and put in the ZYBO Z7-10 board. Picocom was used as the Linux terminal and the SD card was mounted so that the ZYBO board has access to the kernel. Then the module was loaded into the kernel on the ZYBO board. The same process was completed for the multiply module.

Results:

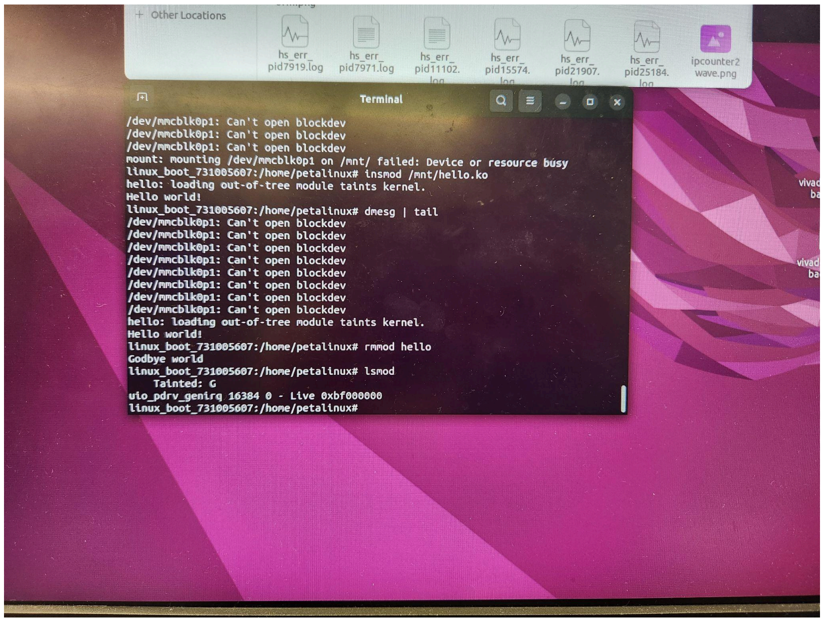
The Linux modules were successfully loaded onto the ZYBO board.



```
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

Password:
linux_boot_731005607:/home/petalinux# mount /dev/mmcblk0p1 /mnt/
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
linux_boot_731005607:/home/petalinux# insmod /mnt/multiply.ko
multiply: loading out-of-tree module taints kernel.
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
linux_boot_731005607:/home/petalinux#
```



```
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
mount: mounting /dev/mmcblk0p1 on /mnt/ failed: Device or resource busy
linux_boot_731005607:/home/petalinux# insmod /mnt/hello.ko
hello: loading out-of-tree module taints kernel.
Hello world!
linux_boot_731005607:/home/petalinux# dmesg | tail
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
/dev/mmcblk0p1: Can't open blockdev
hello: loading out-of-tree module taints kernel.
Hello world!
linux_boot_731005607:/home/petalinux# rmmod hello
Godbye world
linux_boot_731005607:/home/petalinux# lsmod
    Tainted: G
uio_pdrv_genirq 16384 0 - Live 0xbf000000
linux_boot_731005607:/home/petalinux#
```

Conclusion:

This lab taught me how to load modules into the Linux kernel on the ZYBO Z7-10 board. This lab made me comfortable with building PetaLinux projects despite the problems that arise with it. I learned how to use the command ioremap which makes bus memory CPU accessible via functions like readb.

Answers to Questions:

a) The SD card would be used to reboot Linux on the board and then it would be mounted.
b) /run/media/saira/

c) The hello.c in the makefile would have to be updated to the new name. Object files within the make file would have to be changed. If the kernel directory from lab 4 was used then it could lead to compatibility issues including mismatches in kernel headers which would prevent the module from compiling correctly.

Appendix:

```
#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 address of multiplier

/*size of physical address range for multiple */

#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 used 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 physical address*/
 //use ioremap
 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");
 //use iowrite32
 iowrite32(2, virt_addr+4); //base address + offset


 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));

 printk("Physical Address: %x\n", PHY_ADDR); //Print physical address

 printk("Virtual Address: %x\n",*(int*)virt_addr); //Print virtual address



 //a non 0 return means init_module failed; module can't be loaded.

 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 (otherwise be prepared for a reboot). */

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 saira khan");

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);
```