ECEN 449 - Lab Report

Lab Number: 5

Lab Title: Intro to kernel modules

Section Number: 511

Student's Name: Kris Gavvala

Student's UIN: 929000158

Date: 4/9/2025

TA: Gautham Nemani

Introduction:

This week's lab focuses on creating kernel modules on the Ubuntu machine and loading them into the Linux kernel on the ZYBO Z7-10 board. We developed two modules: *hello-world* and *multiply*. The *hello-world* module was a introduction to the process, while the *multiply* module shows how to interact with the multiplication peripheral built Lab 3.

Procedure:

1. Boot Linux on the ZYBO Z7-10 using files from the previous lab.

- On the Linux serial console, gain root access and mount the SD card:
- Navigate to /mnt and list contents to verify the mount. Create a test directory to confirm write access:
- Before removing the SD card, unmount it safely:

Creating the First Kernel Module (hello)

- Insert the SD card into the Ubuntu machine.
- Follow Lab 4 to set up a PetaLinux project (steps 2a–2h), but skip the build.
- Enter the project directory and source the PetaLinux environment:
- Create a module named hello:
- Replace the contents of hello.c with a simple "Hello World" kernel module that prints messages on load/unload.

Creating the Multiply Module

- Source the environment and navigate to the project directory:
- Create the module:
- In the multiply/files/ directory, add xparameters.h and xparameters_ps.h from the hardware BSP.
- Modify multiply.bb to include the headers (shown in appendix)
- Implement multiply.c (shown in appendix).
    - Map the AXI multiplier peripheral address.
    - Write input values to registers.
    - Read and print results using ioread32.
- Build the petalinux project and download the .ko files to the sd card.

- Insert the sd card into the fpga and run picocom to mount it and load the module onto the linux kernel with the insmod command

Results:

The results of lab were shown using the $ dmesg | tail command

Which showed the messages from the OS generated from our Multiply.ko module doing the operation 7 *2 = 14.

Conclusion:

We implemented two kernel modules and learned how to use the dynamic kernel module for the Linux OS to load the kernel modules on the OS. We then used this method to make a driver for our multiply peripheral and test it on the zybo fpga. This lab gave us a good introduction to creating drivers for linux and using the dynamic kernel module.

Post-lab Deliverables:

4. [6 points.] Answers to the following questions:

(a) What is the mount point for the SD card on the Ubuntu machine? Hint: Where does the SD card lie in the directory structure of the Ubuntu file system? This should be different from the Linux system on the Zybo board.

Answer: /run/media/kgavvala19/

(b) 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?

Answer: If we changed the name of the hello.c, we have to change the name hello.c to the new name in the makefile Using the lab 4 kernel directory could cause the module to crash.

Appendix:

Multiply.bb

multiply.bb

Z: > ecen449 > multiply > multiply.bb

```
SUMMARY = "Recipe for  build an external multiply Linux kernel module"
SECTION = "PETALINUX/modules"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"

inherit module

INHIBIT_PACKAGE_STRIP = "1"

SRC_URI = "file://Makefile \
           file://multiply.c \
       file://COPYING \
       file://xparameters.h \
       file://xparameters_ps.h \
          "

S = "${WORKDIR}"

# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.

```

Multiply makefile:

multiply.bb Makefile

Z: > ecen449 > multiply > files > Makefile

```
1   obj-m := multiply.o
2
3   MY_CFLAGS += -g -DDEBUG
4   ccflags-y += ${MY_CFLAGS}
5
6   SRC := $(shell pwd)
7
8   all:
9       $(MAKE) -C $(KERNEL_SRC) M=$(SRC)
10
11  modules_install:
12      $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
13
14  clean:
15      rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c
16      rm -f Module.markers Module.symvers modules.order
17      rm -rf .tmp_versions Modules.symvers
18
```

Multiply.c

```
#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/module.h>

#include <asm/io.h>


#include "xparameters.h"


#define PHYS_ADDR XPAR_MULTPLY_0_S00_AXI_BASEADDR


#define MEMSIZE XPAR_MULTPLY_0_S00_AXI_HIGHADDR -
XPAR_MULTPLY_0_S00_AXI_BASEADDR+1


void * virt_addr; //virtual address pointing to multiplier


static int __init my_init(void)

{

   printk(KERN_INFO "Mapping virtual address...\n");
```

```
    //print the physical and virtual adress of multiplication peripheral.

    printk("physical address:");

    printk(PHYS_ADDR); printk("\n");

    printk("virtual address:");

    printk(virt_addr); printk("\n");

    printk(KERN_INFO "Writing a 7 to register 0\n");

    iowrite32( 7, virt_addr+0);

    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));

    return 0;
}

/* run this function just prior to the modules 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 adress space...\n");
   iounmap((void*)virt_addr);
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Student: Kris Gavvala");
MODULE_DESCRIPTION("Simple Hello world module");

module_init(my_init);
module_exit(my_exit);
```

Hello.c

```
/*  hello.c - hello world kernel module.
  demonstrates  module instantiation , module release and printk.
*/

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>


/* 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)
{
   printk(KERN_INFO "Hello world!\n");
   //printk("<1>Module parameters were (0x%08x) and \"%s\"\n", myint,
    //      mystr);

   //return platform_driver_register(&hello_driver);
```

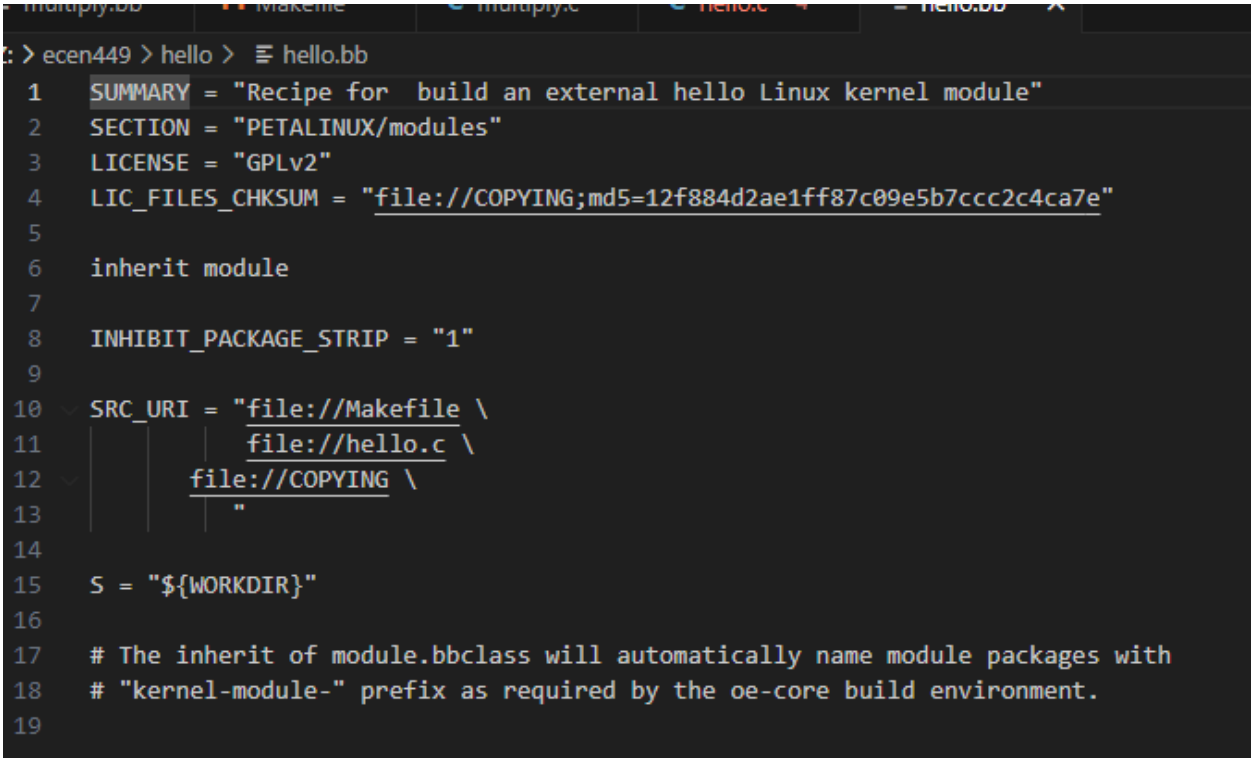
```
    return 0;
}

/* this function is run just prior to the modules removal from the system. here you should release all resources used by your module (otherwise be prepared for reboot).
*/
static void __exit my_exit(void)
{
    //platform_driver_unregister(&hello_driver);
    printk(KERN_ALERT "Goodbye world!\n");
}

/* Standard module information, edit as appropriate */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Student: Kris Gavvala");
MODULE_DESCRIPTION("Simple Hello world module");

module_init(my_init);
module_exit(my_exit);
```

Hello.bb


ecen449 > hello > hello.bb
SUMMARY = "Recipe for build an external hello Linux kernel module"
SECTION = "PETALINUX/modules"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"
inherit module
INHIBIT_PACKAGE_STRIP = "1"
SRC_URI = "file://Makefile \
file://hello.c \
file://COPYING \
"
S = "${WORKDIR}"
# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.