ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

ECEN 449: Microprocessor System Design

Lab 5: Introduction to Kernel Modules on Zynq Linux System

Kylan Lewis

UIN: 719001131

ECEN 449 -504

TA: Ashwin Ashokan

Date: 10/30/2020

Introduction:

The purpose of this lab is to learn and become familiar with the cross compiling C code on the FPGA. In this lab we created two modules and loaded them into the Linux kernel. The first module is a simple Hello-World test and the second prints the message to the kernel buffer.

Procedure:

Part 1:

1. Create a copy of the Lab 4 directory and name is Lab5.

2. Boot to linux on the Zybo and test read write access with mkdir in "/mnt/".

3. Unmount the SD Card from the Zybo Board and mount it onto your host machine

4. Create a simple hello.c file and compile this module with a modified Makefile.

5. Cross-compile the hello.c module & copy the generated hello.ko onto the SD Card.

6. Mount the SD Card back onto the Zybo and load the module into Linux kernel (insmod hello.ko)

7. Make a directory (mkdir -p /lib/modules/3.18.0-xilinx) & remove the module with (rmmod hello)

8. Copy Lab5 into an alternate Lab5b folder and create a new multiply.c in the Lab5b/modules

9. Modify the skeleton code for multiply.c then copy xparameters.h & x_parameters_ps.h from lab4.sdk/FSBL_bsp into the current directory.

10. Modify the Makefile to compile multiply.c, cross compile the file and copy the multiply.ko file to the SD Card (don't forget to unmount the SD Card, then remount it when unplugging)

11. Load the module multiply.ko into the Linux kernel like with hello.ko.

Results:

Testing the read/write in /mnt/

```
zynq> ls
BOOT.bin                  ramdisk8M.image
MISC                      test
System Volume Information uImage
devicetree.dtb            uramdisk.image.gz
modules
zynq> ls -lae
total 17857
drwxr-xr-x    6 root     0            32768 Thu Jan  1 00:00:00 1970 .
drwxr-xr-x   17 12319    300           1024 Thu Jan  1 00:00:53 1970 ..
-rwxr-xr-x    1 root     0          2506156 Wed Oct 14 15:46:52 2020 BOOT.bin
drwxr-xr-x    6 root     0            32768 Sat Oct  3 14:59:20 2020 MISC
drwxr-xr-x    2 root     0            32768 Wed Oct 14 15:45:16 2020 System Volume Information
-rwxr-xr-x    1 root     0             7490 Wed Oct 14 15:46:20 2020 devicetree.dtb
drwxr-xr-x    3 root     0            32768 Thu Oct 29 22:04:44 2020 modules
-rwxr-xr-x    1 root     0          8388608 Thu Jan 10 15:37:42 2013 ramdisk8M.image
drwxr-xr-x    2 root     0            32768 Tue Jan  1 00:00:00 1980 test
-rwxr-xr-x    1 root     0          3447904 Wed Oct 14 15:46:10 2020 uImage
-rwxr-xr-x    1 root     0          3693174 Wed Oct 14 15:46:02 2020 uramdisk.image.gz
zynq>
```

Output results for hello.ko and multiply.ko

```
zynq> dmesg | tail
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
Hello World!
Goodbye World!
random: nonblocking pool is initialized
unmapping virtual address space....
zynq> ls
Makefile          hello.mod.c       multiply.c        multiply.o
Module.symvers    hello.mod.o       multiply.ko       xparameters.h
hello.c           hello.o           multiply.mod.c    xparameters_ps.h
hello.ko          modules.order     multiply.mod.o
zynq>
```

Conclusions:

The modules were successfully loaded into the Linux kernel and I learned how to create and use the makefile to compile my C code. As a result, I also learned how to see the physical and virtual memory mapping and how to use some other Linux functions. Overall, I only encountered one error when unplugging my SD card without unmounting it first and fixed this with the “fsck” command.

Post-Lab 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?

If we were to accidentally reset the ZYBO Board, the additional steps needed in 2.g would be to recreate the mounting directory because the one we created was stored in RAM and volatile.

(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.

Didn't test on the lab CentOS machine, but if I was in lab this would be somewhere under: /media/kylanlewis/ " Storage Name"

(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 our hello.c file, we would then also have to change the matching naming convention in our makefile. If in our makefile we specified the kernel directory from lab4 instead of lab5, we lose functionality with our code because we are compiling modules in a folder that we did not make relevant changes to and the kernel would not be supported.

Code:

multiply.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 address 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 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 + 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);
    printk("Virtual Address", *(int)*virt_addr);

    // 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 Kylan Lewis");
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);
```

Hello.c

```c
/* hello.c - Hello World kernel module
*
* Demonstrates module initialization, module release and printk.
*
* */
#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 */

/* This function is run upon module load. This is where I set up data
    structures and reserve resources used by the module */

static int __init my_init(void) {
    //Linux kernel's version of printf
    printk(KERN_INFO "Hello world!\n");

    //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. I should release _ALL_resources used by my module
    here (otherwise I will have to reboot) */
static void __exit my_exit(void) {
    printk(KERN_ALERT "Goodbye world!\n");
}

//These define info that can be displayed by modinfo
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Kylan Lewis");
MODULE_DESCRIPTION("Simple Hello World Module");

/* Here we define which functions we want to use for initialization
    and cleanup */

module_init(my_init);
```

```
module_exit(my_exit);
```