ECEN 749 Lab 5 Report

Tong Lu
UIN 621007982

October 11, 2018

TEXAS A&M
UNIVERSITY®

Introduction

In this lab, we cross-compiled a ”Hello World!” kernel module for the ZYBO ARM processor, and then we also compiled an multiply module.

Procedure

1. Plug in the SD card and boot Linux on the ZYBO board.

2. mount SD card under /mnt/.

```
mount /dev/mmcb1k0p1 /mnt/
```

3. Test the mount operation:

```
cd /mnt/
ls -la
```

4. Create a directory on SD card directory and demo the date stamp to TA:

```
mkdir test
```

5. Un-mount the SD card:

```
cd /
unmount /mnt
```

6. Based on lab manual, create hello.c and makefile, and then cross-compile for ZYBO ARM platform:

```
make ARCH=arm CROSS COMPILE=arm-xilinx-linux-gnueabi-
```

7. Copy the generated hello.ko file to the SD card and mount the SD card on FPGA:

```
mount /dev/mmcb1k0p1 /mnt/
```

8. load the module into Linux kernel on ZYBO board:

```
insmod hello.ko
```

9. Check the output *printk* statement using *dmesg* and *tail*, then demo to TA:

```
dmesg | tail
```

10. Create a module directory:

```
mkdir -p /lib/modules/`uname -r`
```

11. Remove the module and ensure module was removed:

```
rmmod hello
lsmod
```

12. Following the same step above, create a multipy.c (see Appendix), modify makefile (see Appendix) and cross-compile:

```
make ARCH=arm CROSS COMPILE=arm-xilinx-linux-gnueabi-
```

13. Copy the generated module multiply.ko on SD card, and load the module on Linux kernel on ZYBO:

```
mount /dev/mmcblk0p1 /mnt/
```

Result

All the programs was finished and demonstrated to TA. The programs are working well and meet all the requirement on lab manual.

```
[lvtongtom305@lin15-424cvlb ~]$ picocom -b 115200  /dev/ttyUSB1
picocom v2.3a

port is        : /dev/ttyUSB1
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none

Type [C-a] [C-h] to see available commands

Terminal ready

zynq> mount /dev/mmcblk0
mmcblk0    mmcblk0p1
```

```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
    Please run fsck.
zynq> insmod /mnt/multiply.ko
Mapping virtual address...
Physical address: 608e0000; Virtual address: 43c00000
.
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
```

src/console_printout_short

.

Conclusion

In this lab, I learned to cross-compile an linux kernel for ARM processor using make and vivado, and successfully boot it up on the FPGA. This lab will be very helpful for the future projects.

Answer to Questions

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

If we press the reset button before plugging back the SD card, then the Linux kernel in the memory of the ZYBO board will be wiped out. Therefore **we need to turn off and turn on the ZYBO board in step 2.g** before executing the mount command.

(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.

The SD card is mounted under /dev/sd1/.

(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?**

We need to change *hello.o* to *<new_file_name>.o*
If we specified the kernel directory from lab 4, the source code still compiles, and we get the same kernel module file.

Appendix

```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 IO reads and writes
#include <linux/ioport.h>   // Used for io memory allocation

// From xparameters.h, physical address of multiplier
#define PHY_ADDR XPAR_MULTIPLY_0_S00_AXI_BASEADDR
// Size of physical address range for multiply
#define MEMSIZE XPAR_MULTIPLY_0_S00_AXI_HIGHADDR - XPAR_MULTIPLY_0_S00_AXI_BASEADDR + 1

// virtual address pointing to multiplier
void* virt_addr;

/* 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, print the physical and virtual address
    virt_addr = (void*)ioremap(PHY_ADDR, MEMSIZE);
    printk("Physical address: %x; Virtual address: %x\n.", (unsigned int)virt_addr, (unsigned int)PHY_ADDR);
    // 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);

    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));

    // 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 Student (and others)");
```

```c
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);
```

src/multiply.c

```makefile
obj-m += multiply.o

all:
    make -C /home/grads/l/lvtongtom305/ecen749/lab5/linux-3.14 M=$(PWD) modules

clean:
    make -C /home/grads/l/lvtongtom305/ecen749/lab5/linux-3.14 M=$(PWD) clean
```

src/Makefile