Lab 6: Character Device Driver Development

Abhinav Subramanian

ECEN 449-502

Due date: 3/2/2020

Introduction:

The aim of this lab was to learn how to write a custom driver to interface with the Multiply IP we created in lab 3. This driver implements much the same functionality that the multiply software did in lab 3, but we are instead executing code directly on the ARM processor this time and using the driver we write to interface directly with the hardware. For this, I had to consider the interactions between the hardware, the kernel space, and the user space.

Procedure:

- Write a device driver multiplier.c that does the following:
    - Write an initialization function that maps the physical address to a virtual address with ioremap, and then register the driver. In the exit function, do the opposite.
    - In the open/close functions, simply print “open/close device”.
    - The read function should take a specified number of bytes from the kernel space and put them in the user buffer.
    - The write function should do the opposite (transfer from user to kernel).
- Now write a program called devtest.c that uses the driver we just wrote to multiply out numbers 0-16 together using the multiply IP. Also verify that the results are correct.
- Compile both files, and insert both multiplier.ko and the devtest executable onto the SD card.
- Boot up linux, mount the SD, insert the multiplier.ko kernel module and run the devtest executable. If successful, you should see numbers 0-16 being multiplied together to give an answer, along with “Result correct!”

Results:

Terminal

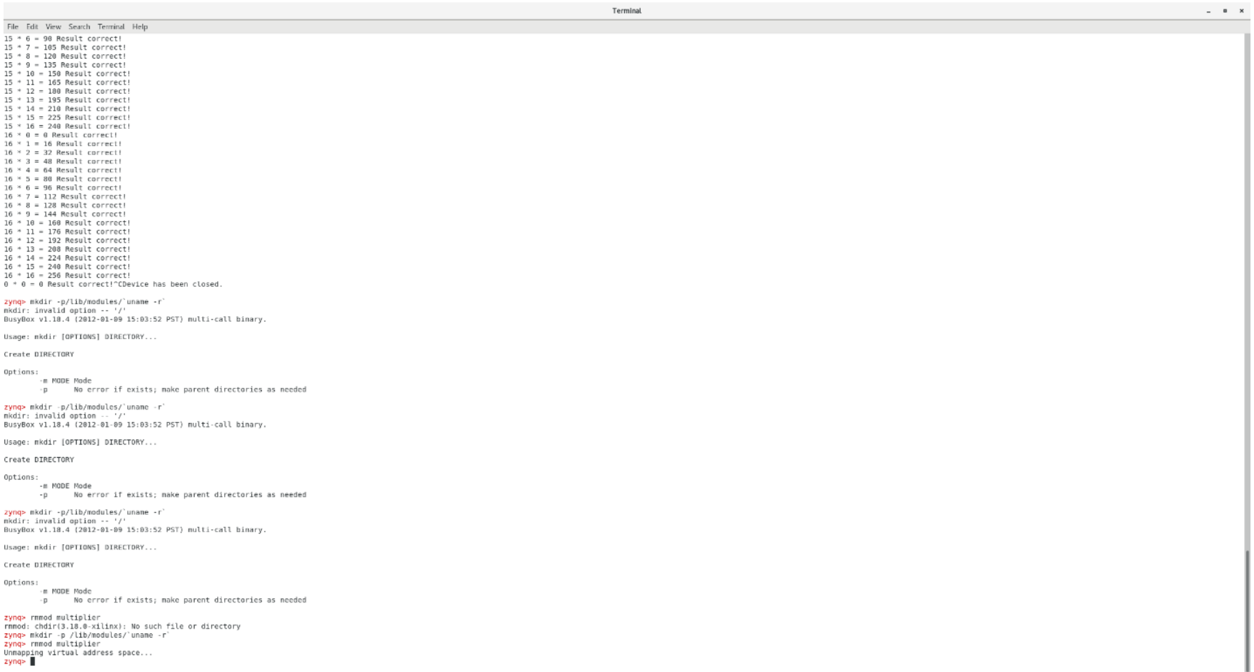
File Edit View Search Terminal Help

```
bash-4.2$ source /opt/coe/Xilinx/Vivado/2015.2/settings64.sh
bash-4.2$ picocom -b 115200 -r -l /dev/ttyUSB1
picocom v2.2

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     : yes
nolock is      : yes
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,

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

Terminal ready
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 multiplier.ko
Mapping virtual address..
Physical address: 43C00000
Virtual address: 608C1000
Registered a device with dynamic Major number of 245
Create a device file for this device with this command:
'mknod /dev/multiplier c 245 0'.
zynq>
```

Terminal

File Edit View Search Terminal Help

```
0 * 0 = 0 Result correct!
0 * 1 = 0 Result correct!
0 * 2 = 0 Result correct!
0 * 3 = 0 Result correct!
0 * 4 = 0 Result correct!
0 * 5 = 0 Result correct!
0 * 6 = 0 Result correct!
0 * 7 = 0 Result correct!
0 * 8 = 0 Result correct!
0 * 9 = 0 Result correct!
0 * 10 = 0 Result correct!
0 * 11 = 0 Result correct!
0 * 12 = 0 Result correct!
0 * 13 = 0 Result correct!
0 * 14 = 0 Result correct!
0 * 15 = 0 Result correct!
0 * 16 = 0 Result correct!
1 * 0 = 0 Result correct!
1 * 1 = 1 Result correct!
1 * 2 = 2 Result correct!
1 * 3 = 3 Result correct!
1 * 4 = 4 Result correct!
1 * 5 = 5 Result correct!
1 * 6 = 6 Result correct!
1 * 7 = 7 Result correct!
1 * 8 = 8 Result correct!
1 * 9 = 9 Result correct!
1 * 10 = 10 Result correct!
1 * 11 = 11 Result correct!
1 * 12 = 12 Result correct!
1 * 13 = 13 Result correct!
1 * 14 = 14 Result correct!
1 * 15 = 15 Result correct!
1 * 16 = 16 Result correct!
2 * 0 = 0 Result correct!
2 * 1 = 2 Result correct!
2 * 2 = 4 Result correct!
2 * 3 = 6 Result correct!
2 * 4 = 8 Result correct!
2 * 5 = 10 Result correct!
2 * 6 = 12 Result correct!
2 * 7 = 14 Result correct!
2 * 8 = 16 Result correct!
2 * 9 = 18 Result correct!
2 * 10 = 20 Result correct!
2 * 11 = 22 Result correct!
2 * 12 = 24 Result correct!
2 * 13 = 26 Result correct!
2 * 14 = 28 Result correct!
2 * 15 = 30 Result correct!
2 * 16 = 32 Result correct!
3 * 0 = 0 Result correct!
3 * 1 = 3 Result correct!
3 * 2 = 6 Result correct!
3 * 3 = 9 Result correct!
3 * 4 = 12 Result correct!
3 * 5 = 15 Result correct!
3 * 6 = 18 Result correct!
3 * 7 = 21 Result correct!
3 * 8 = 24 Result correct!
3 * 9 = 27 Result correct!
3 * 10 = 30 Result correct!
3 * 11 = 33 Result correct!
3 * 12 = 36 Result correct!
3 * 13 = 39 Result correct!
3 * 14 = 42 Result correct!
3 * 15 = 45 Result correct!
3 * 16 = 48 Result correct!
4 * 0 = 0 Result correct!
4 * 1 = 4 Result correct!
4 * 2 = 8 Result correct!
4 * 3 = 12 Result correct!
```



Conclusion:

In this lab, I learned how to write character device drivers for Linux systems. Drivers allow programmers to write high level programs for hardware and perform system calls without having to worry too much about how the hardware works. I realized this I was doing the lab, as devtest was not a problem at all to get working. However, it took hours for me to even understand how the driver was supposed to interact with the hardware, and this was a very simple program as it is.

Postlab questions:

1. When writing the read/write functions, I was unable to use straight pointer arithmetic to access the elements in virtual memory. This is because the virtual memory is mapped to the physical memory via a page table, and ioremap is responsible for initializing said page table.
2. This implementation is definitely faster than lab 3's, as you're executing the code directly on the ZYBO processor.
3. In this method, you're running the code directly on the processor via a character driver, which is both faster and more productive, as you can also interface with other IPs that might exist on the block more easily. You also don't need to be connected to a computer to execute the code. However, in lab 3, you're tethered to the computer and aren't running the code directly on the processor, which makes lab 6's implementation, although harder, better in the long term.

4. Registering the driver last makes sure that all the necessary settings have been configured first. Also, deregistering it last makes sure that all the necessary settings have been deconfigured.

Appendix:

multiplier.c

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

#include <linux/moduleparam.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <linux/sched.h>

#include "xparameters.h"

#include <linux/ioport.h>

#include <asm/io.h>

#include <asm/uaccess.h>


#define DEVICE_NAME "multiplier"

// 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


//function signatures

int init_module(void);

void cleanup_module(void);

int device_open(void);

int device_release(void);
```

```
ssize_t device_read(struct file *,char *, size_t, loff_t *);
ssize_t device_write(struct file *, const char *, size_t, loff_t *);

//file operations.
static struct file_operations fops = {
  .read = device_read,
  .write = device_write,
  .open = device_open,
  .release = device_release
};

int Major;
void* virt_addr;

int my_init(void){
  printk(KERN_INFO "Mapping virtual address..\n");

  //map virtual address to physical address
  virt_addr=ioremap(PHY_ADDR,12);
  printk(KERN_INFO "Physical address: %X \n",PHY_ADDR);
  printk(KERN_INFO "Virtual address: %X \n",virt_addr);

  Major=register_chrdev(0,DEVICE_NAME,&fops); //register character device driver.

  if(Major<0){ //failure
    printk(KERN_ALERT "Registering char device failed with %d\n", Major);
    return Major;
```

```
  }


  printk(KERN_INFO "Registered a device with dynamic Major number of %d\n", Major);
  printk(KERN_INFO "Create a device file for this device with this command:\n'mknod /dev/%s
c %d 0'.\n", DEVICE_NAME, Major); //registration success
  return 0;
}


void my_cleanup(void){ //deregister driver and unmap virtual memory
  unregister_chrdev(Major,DEVICE_NAME);
  printk(KERN_ALERT "Unmapping virtual address space...\n");
  iounmap((void*)virt_addr);
}
int device_open(void){
  printk(KERN_INFO "Device has been opened.\n");
  return 0;
}
int device_release(void){
  printk(KERN_INFO "Device has been closed.\n");
  return 0;
}
ssize_t device_read(struct file *filp, char *buffer, size_t length, loff_t * offset){
  int bytes_read=0;
  char kernel_buf[length*4]; //kernel buffer
  int i=0;
  for(i=0;i<12;i++){ //read bytes from memory into a buffer.
    kernel_buf[i]=ioread8(virt_addr+i);
```

```
 }
 char* tempKern=kernel_buf; //now move the bytes into the actual user buffer


 while (length!=0){
  put_user(*(tempKern++),buffer++);
  length--;
  bytes_read++;
 }
 return bytes_read;
}
ssize_t device_write(struct file *file, const char *buffer, size_t length, loff_t * off){
 char kernelBuf[length];
 int bytes_written=0;
 while(length>bytes_written){ //move bytes from buffer to kernel buffer
  get_user(kernelBuf[bytes_written],buffer++);
  bytes_written++;
 }
 int i=0;
 for(i=0;i<8;i++){ //now move into memory
  iowrite8(kernelBuf[i],virt_addr+i);
 }
 return bytes_written;
}
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abhinav Subramanian");
MODULE_DESCRIPTION("Module which creates a character device and allows user interaction
with it");
```

```
module_init(my_init);
module_exit(my_cleanup);
```

devtest.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
  unsigned int result;
  int fd;
  int i, j;
  int buf[3];
  unsigned int read_i;
  unsigned int read_j;

  char input=0;

  fd=open("/dev/multiplier",O_RDWR);

  if(fd==-1){
    printf("Failed to open device file!\n");
    return -1;
```

```
 }
 while(input!='q'){
  for(i=0;i<=16;i++){
    for(j=0;j<=16;j++){
      buf[0]=i;
      buf[1]=j;
      write(fd,buf,2*sizeof(int));//write to buffer
      read(fd,buf,3*sizeof(int));//read from buffer.
      //results of reading from the buffer.
      read_i=buf[0];
      read_j=buf[1];
      result=buf[2];

      printf("%u * %u = %u ",read_i,read_j,result);
      if(result==(i*j))
        printf("Result correct!");
      else
        printf("Result incorrect!");
      input=getchar();
    }
  }
 }
 close(fd);
 return 0;
}
```