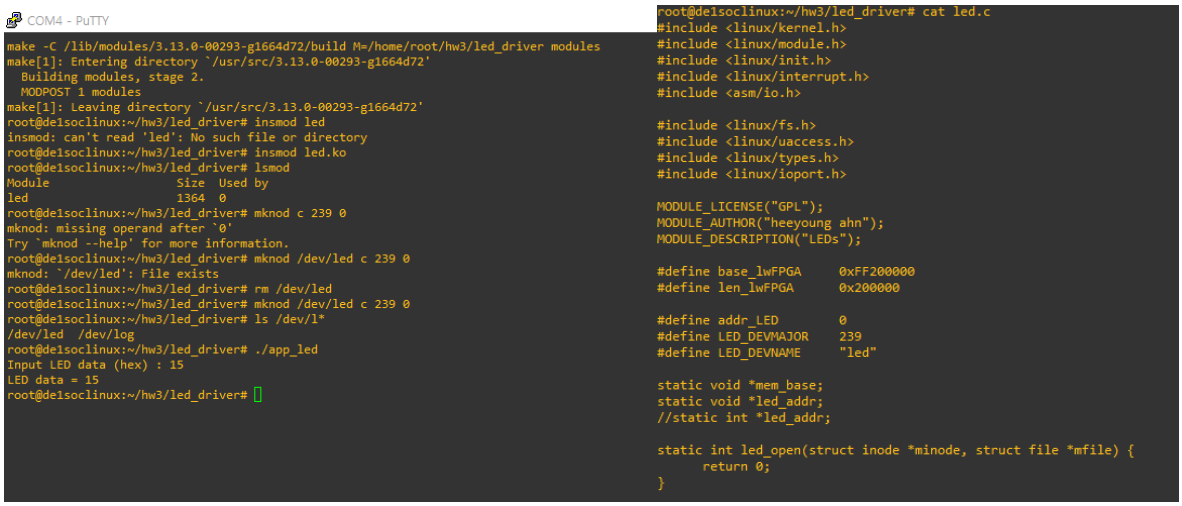
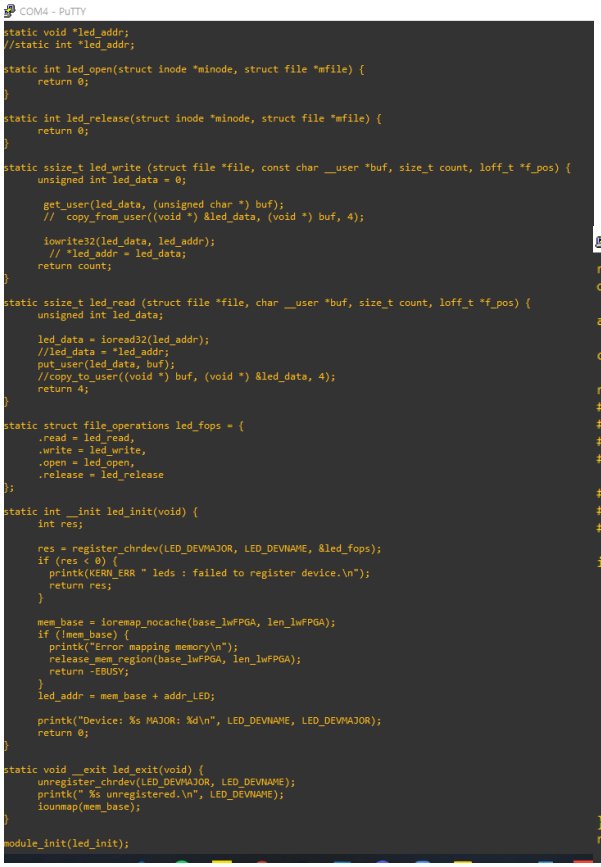
실습. 디바이스드라이버 프로그래밍(1)

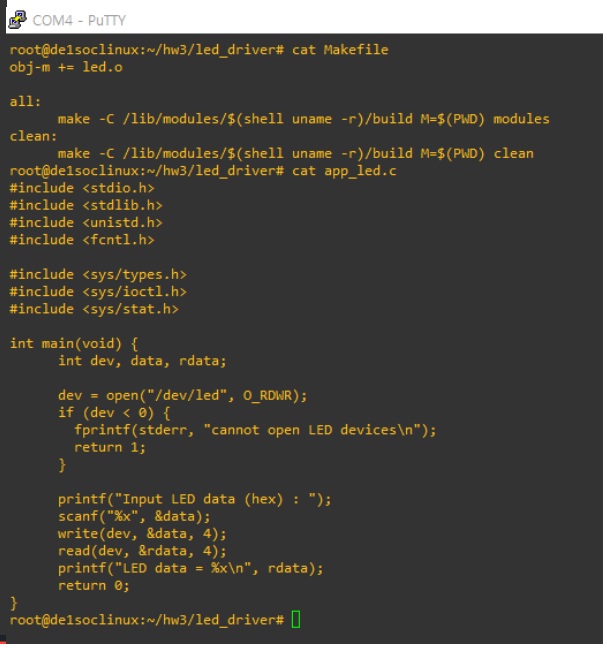
2017253019안희영

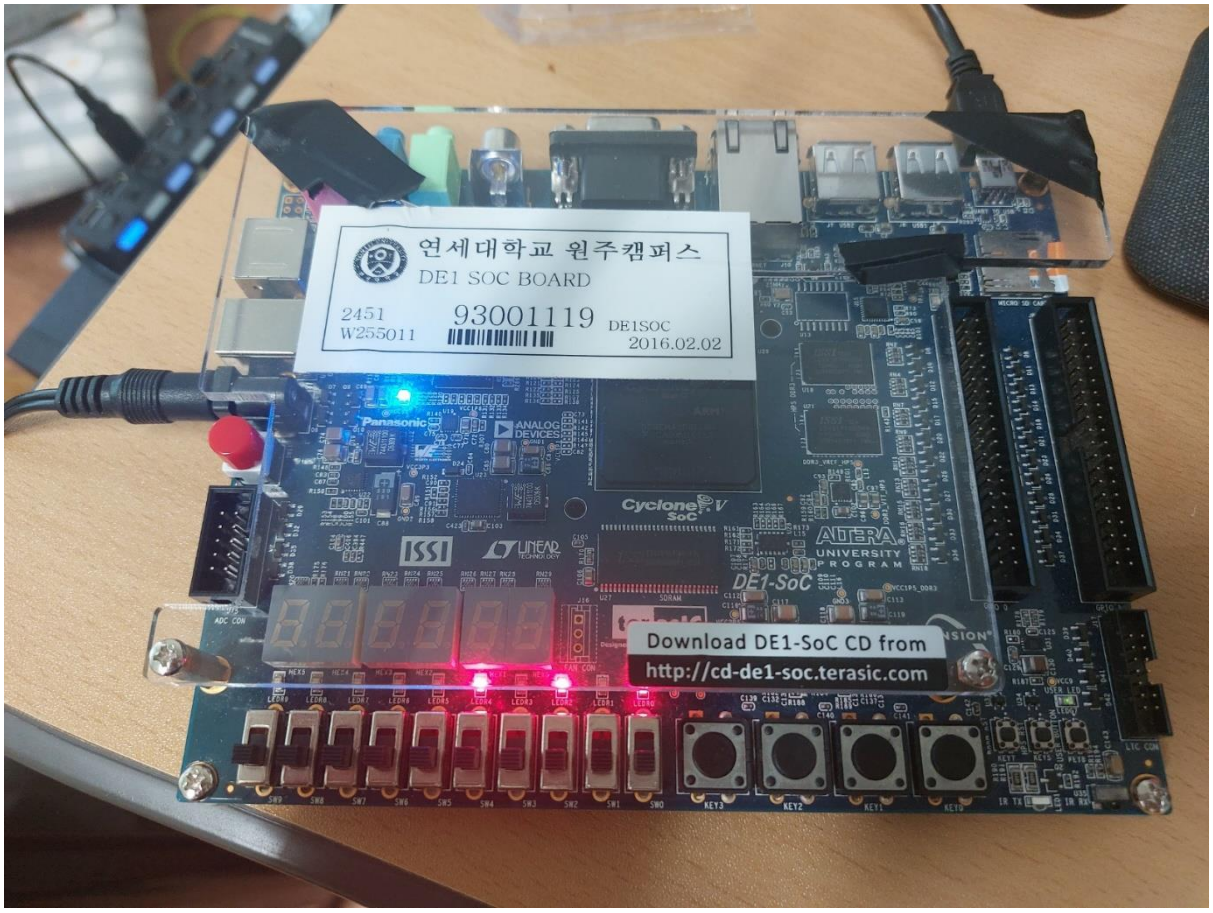
1. (LED 디바이스 드라이버)

(1) 강의자료에 있는 LED용 디바이스 드라이버(led.c)를 작성하고, 커널 모듈 컴파일을 위한 Makefile을 작성한 후 make를 사용하여 컴파일하시오. 그리고 컴파일 과정의 메시지를 참고하여 컴파일 과정이 어떻게 진행되는지 말하시오.





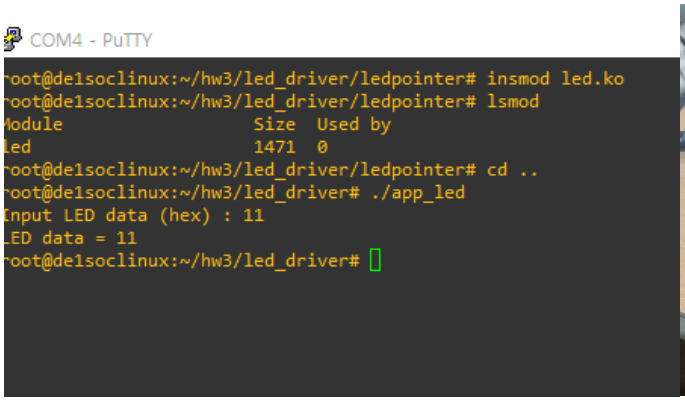




첫번째 이미지에 나오는 대로 입력된 0x15값을 led로 표현하고 있습니다.

모듈의 컴파일 과정은 모듈 소스 코드를 컴파일하여 오브젝트 파일로 출력한 다음 해당 파일을 커널 오브젝트로 만들기 위한 추가 정보를 입력한 mod.o 파일을 생성하여 두 파일을 링크하여 최종 커널 오브젝트 파일을 만듭니다.

(4) 작성한 디바이스 드라이버를 다음과 같이 수정하여 작성한 후 다시 앞에서와 같이 실행하여 동작을 확인하고, 둘의 차이점을 비교해보시오.



```
root@de1socLinux:~/hw3/led_driver# cat ./ledpointer/led.c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <asm/io.h>

#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/types.h>
#include <linux/ioport.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("heeyoung ahn");
MODULE_DESCRIPTION("LEDs");

#define base_lwFPGA     0xFF200000
#define len_lwFPGA      0x200000

#define addr_LED        0
#define LED_DEVMAJOR    239
#define LED_DEVNAME     "led"

static void *mem_base;
static int *led_addr;


static int led_open(struct inode *minode, struct file *mfile) {
      return 0;
}

static int led_release(struct inode *minode, struct file *mfile) {
      return 0;
}
```

첫번째 이미지에 나온 0x11의 값을 led로 표시하고 있습니다.

. get_user, put_user로 입출력하는 데이터와 다르게 copy from user 등을 이용한 경우 읽어올 크기를 지정해 줘야 합니다.

2. (7세그먼트 LED 디바이스 드라이버)

COM4 - PuTTY

```
root@de1soclinux:~/hw3/hex_driver# cat hex.c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <asm/io.h>

#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/types.h>
#include <linux/ioport.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Heeyoung Ahn");
MODULE_DESCRIPTION("seven segment LEDs");

#define base_lwFPGA     0xFF200000
#define len_lwFPGA      0x200000

#define addr_HEX30      0x20
#define addr_HEX54      0x30
#define HEX_DEVMAJOR    240
#define HEX_DEVNAME     "hex"

static void *mem_base;
static void *hex30_addr;
static void *hex54_addr;

static int hex_conversions[16] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x67, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
static unsigned int hex_data = 0;

static int hex_open(struct inode *minode, struct file *mfile) {
      return 0;
}

static int hex_release(struct inode *minode, struct file *mfile) {
      return 0;
}

static ssize_t hex_write (struct file *file, const char __user *buf, size_t count, loff_t *f_pos) {
      unsigned int tmp_hex_data = 0;
      unsigned int hex30_data = 0;
      unsigned int hex54_data = 0;

      get_user(tmp_hex_data, (unsigned int *) buf);

      tmp_hex_data &= 0xFFFFFF;

      hex_data = tmp_hex_data;
      hex30_data |= hex_conversions[tmp_hex_data & 0xF];
      tmp_hex_data >>= 4;
      hex30_data |= hex_conversions[tmp_hex_data & 0xF] << 8;
      tmp_hex_data >>= 4;
      hex30_data |= hex_conversions[tmp_hex_data & 0xF] << 16;
      tmp_hex_data >>= 4;
      hex30_data |= hex_conversions[tmp_hex_data & 0xF] << 24;
      tmp_hex_data >>= 4;
      hex54_data |= hex_conversions[tmp_hex_data & 0xF];
      tmp_hex_data >>= 4;
      hex54_data |= hex_conversions[tmp_hex_data & 0xF] << 8;

      iowrite32(hex30_data, hex30_addr);
      iowrite32(hex54_data, hex54_addr);
      return count;
}
```

```
static ssize_t hex_read (struct file *file, char __user *buf, size_t count, loff_t *f_pos) {
      put_user(hex_data, (unsigned int *) buf);
      return 4;
}

static struct file_operations hex_fops = {
      .read = hex_read,
      .write = hex_write,
      .open = hex_open,
      .release = hex_release
};

static int __init hex_init(void) {
      int res;

      res = register_chrdev(HEX_DEVMAJOR, HEX_DEVNAME, &hex_fops);
      if (res < 0) {
        printk(KERN_ERR " hexs : failed to register device.\n");
        return res;
      }

      mem_base = ioremap_nocache(base_lwFPGA, len_lwFPGA);
      if (!mem_base) {
        printk("Error mapping memory\n");
        release_mem_region(base_lwFPGA, len_lwFPGA);
        return -EBUSY;
      }
      hex30_addr = mem_base + addr_HEX30;
      hex54_addr = mem_base + addr_HEX54;

      printk("Device: %s MAJOR: %d\n", HEX_DEVNAME, HEX_DEVMAJOR);
      return 0;
}

static void __exit hex_exit(void) {
      unregister_chrdev(HEX_DEVMAJOR, HEX_DEVNAME);
      printk(" %s unregistered.\n", HEX_DEVNAME);
      iounmap(mem_base);
}

module_init(hex_init);
module_exit(hex_exit);
root@de1soclinux:~/hw3/hex_driver#
```

COM4 - PuTTY

```
root@de1soclinux:~/hw3/hex_driver# make
make -C /lib/modules/3.13.0-00293-g1664d72/build M=/home/root/hw3/hex_driver modules
make[1]: Entering directory `/usr/src/3.13.0-00293-g1664d72'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/3.13.0-00293-g1664d72'
root@de1soclinux:~/hw3/hex_driver# cat Makefile
obj-m += hex.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
root@de1soclinux:~/hw3/hex_driver# cat app_hex.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/stat.h>

int main(void) {
	int dev, data, rdata;

	dev = open("/dev/hex", O_RDWR);
	if (dev < 0) {
	  fprintf(stderr, "cannot open LED devices\n");
	  return 1;
	}

	printf("Input HEX7 data (hex) : ");
	scanf("%x", &data);
	write(dev, &data, 4);
	read(dev, &rdata, 4);
	printf("read data = %x\n", rdata);
	return 0;
}
root@de1soclinux:~/hw3/hex_driver# ./app_hex
Input HEX7 data (hex) : ff
read data = ff
root@de1soclinux:~/hw3/hex_driver#
```

입력받은 ff값을 hexled에서 출력하고 있는걸 볼 수 있습니다.