Embedded System Software 과제 2

(과제 수행 결과 보고서)

과 목 명 : [CSE4116] 임베디드시스템소프트웨어

담당 교수 : 서강대학교 컴퓨터공학과 박성용

학번/이름 : 20151556 변홍수

개발 기간 : 2020. 05. 13. ~ 2020. 05. 20.

I. 개발 목표

타이머 디바이스 드라이버 및 관련 테스트 응용 프로그램을 개발한다.

II. 개발 범위 및 내용

가. 테스트 응용 프로그램

사용자로부터 타이머 디바이스 구동 옵션을 받아 **ioctl**을 통하여 타이머 디바이스 드라이버에 전달하고 **ioctl**을 통해 타이머 디바이스를 구동한다. 실행 예시는 다음과 같다.

./app TIMER_INTERVAL[1-100] TIMER_ CNT[1-100] TIMER_ INIT[0001-8000]

TIMER_INTERVAL : HZ 값 1~100 (0.1~10 초)

TIMER_CNT : 디바이스 출력 변경 횟수 (1~100)

TIMER_INIT : FND 에 출력되는 초기 문양과 위치 (0001~8000)

단, 4 자리 중 한 자리만 1~8 숫자가 입력

예를 들어, 0040 이라면 왼쪽에서 세번째 자리에 "4" 를 출력

나. 타이머 디바이스 드라이버

디바이스 드라이버(fpga_fnd, fpga_led, fpga_dot, fpga_text_led)와 타이머 기능을 포함한 하나의 Module을 구현한다.

사용자 프로그램에서 **ioctl**을 통하여 전달 된 옵션을 기준으로 4가지 디바이스(FND, LED, DOT, TEXT_LCD)의 요구 사항 에 따라 디바이스에 동시에 출력한다. TIMER_INTERVAL 에 따라 디바이스의 값들이 바뀐다.

다음 문양은 왼쪽부터 1번 ~ 8번이다.

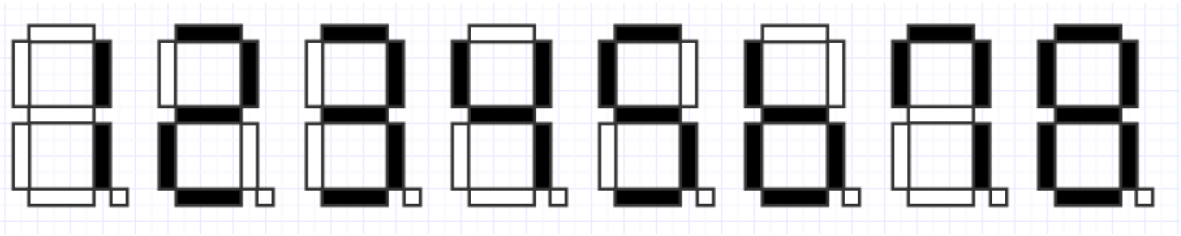

Figure 1

1) fpga_fnd(FND)

- 초기 상태 : TIMER_INIT에서 전달 된 문양과 위치에 따라 FND에 출력한다.
- 이후 상태 : TIMER_INTERVAL마다 다음 문양을 TIMER_CNT 횟수만큼 지정된 위치에 출력한다. TIMER_CNT 횟수만큼의 출력이 끝나면 FND의 불을 꺼준다. (0000으로 초기화).
  - ✓ 다음 문양이란 현재 수에서 1 증가된 수를 의미한다. 단, 8의 다음 문양은 1이다.
  - ✓ 문양이 출력되는 위치는 한 번의 로테이션이 끝날 때마다 우측으로 이동한다.
  - ✓ 만약 4번째 자리에서 로테이션이 끝났다면, 1번째 자리로 이동한다

2) fpga_led(LED)

현재 fpga_fnd에서 출력 중인 문양의 번호를 나타낸다. TIMER_CNT 횟수만큼의 출력이 끝나면 fpga_led의 불을 꺼준다.

3) fpga_dot(DOT)

현재 fpga_fnd에서 출력 중인 문양과 같은 모양의 문양을 출력한다. fpga_fnd의 문양이 바뀐다면, fpga_dot도 fpga_fnd와 같은 문양으로 함께 바뀐다. TIMER_CNT 횟수만큼의 출력이 끝나면 dot의 불을 꺼준다

4) fpga_text_lcd(TEXT_LCD)

- 초기 상태 : 첫 번째 줄에는 자신의 학번을 출력하고, 두 번째 줄에는 자신의 이름을 영문으로 출력한다. 이때 두 문장 모두 left align시킨다.
- 이후 상태 : TIMER_INTERVAL 마다 오른쪽으로 한 칸씩 shift 이동을 하고, 가장 우측 글자가 가장 오른쪽 칸에 도달한 경우, 다시 왼쪽으로 shift 이동을 시작한다. 왼쪽 shift 이동도 마찬가지로, 왼쪽 글자가 가장 왼쪽 칸에 도달한 경우, 다시 오른쪽으로 shift 이동을 시작한다. TIMER_CNT 횟수만큼의 출력이 끝나면 LCD의 불을 꺼준다.

III. 추진 일정 및 개발 방법

가. 추진 일정

2020. 05. 13.

- 응용 프로그램 및 모듈 기초 구현

2020. 05. 14. ~ 15.

- 모듈에 디바이스 드라이버 구현 및 테스트
  - ✓ fpga_fnd
  - ✓ fpga_led
  - ✓ fpga_dot
  - ✓ fpga_text_lcd
- 응용 프로그램과 모듈에서 ioctl 기초 작동 테스트

2020. 05. 16. ~ 18.

- 타이머 디바이스 구현
  - ✓ 타이머에 맞게 다른 디바이스들이 작동하도록 수정
  - ✓ 타이머 종료 시 모든 디바이스 초기화

2020. 05. 19.

- 응용 프로그램 Argument에러 예외처리 체계화
- 응용 프로그램과 모듈에서 ioctl 작동 최종 구현
- 코드 정리 및 최종 테스트

2020. 05. 20.

- Document 작성

나. 개발 방법

1) 응용 프로그램

실행할 때 TIMER_INTERVAL, TIMER_CNT, TIMER_INIT 3개의 Argument를 입력 받는다. Argument의 개수가 3개가 아닐 경우 프로그램을 종료한다.

입력 받은 3개의 Argument TIMER_INTERVAL, TIMER_CNT, TIMER_INIT가 각각 Format에 맞는지 확인한다. 맞지 않을 경우 프로그램을 종료한다.

모든 Argument들이 Valid할 경우 디바이스를 열어 두 번의 ioctl를 통해 각 디바이스를 초기화, 타이머를 실행 한 후 프로그램을 종료한다.

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#define DEVICE_MAJOR 242
#define DEVICE_NAME "dev_driver"
#define IOCTL_SET_OPTION _IOW(DEVICE_MAJOR, 0, char *)
#define IOCTL_COMMAND _IOW(DEVICE_MAJOR, 1, int *)

int main(int argc, char **argv){
	int i;
	int fd;
	int argv3 = 0;
	int command[2];

	/* Passed Argument number is not 3  */
	if(argc != 4) {
		printf("Usage : [TIMER_INTERVAL] [TIMER_CNT] [TIMER_ INIT]\n");
		return -1;
	}

	/* Timer interval is not valid */
	for(i=strlen(argv[1])-1; i>=0; i--){
		if(isdigit(argv[1][i]) == 0){
			printf("Usage : [TIMER_INTERVAL 1-100]
				[TIMER_CNT] [TIMER_ INIT]\n");
			return -1;
		}
	}

	/* Timer count is not valid */
	for(i=strlen(argv[2])-1; i>=0; i--){
		if(isdigit(argv[2][i]) == 0){
			printf("Usage : [TIMER_INTERVAL]
				[TIMER_CNT 1-100] [TIMER_ INIT]\n");
```

```
                        return -1;
                }
        }

        /* Timer init is not valid */
        if(strlen(argv[3]) != 4){
                printf("Usage : [TIMER_INTERVAL]
                        [TIMER_CNT] [TIMER_ INIT 0001-8000]\n");
                return -1;
        }
        for(i=strlen(argv[3])-1; i>=0; i--){
                if(isdigit(argv[3][i]) == 0 || argv[3][i] > '8'){
                        printf("Usage : [TIMER_INTERVAL]
                                [TIMER_CNT] [TIMER_ INIT 0001-8000]\n");
                        return -1;
                }
                if(argv[3][i] != '0') argv3++;
        }
        if(argv3 != 1){
                printf("Usage : [TIMER_INTERVAL]
                        [TIMER_CNT] [TIMER_ INIT 0001-8000]\n");
                return -1;
        }

        /*
        * Passed Argument is valid
        */
        fd = open("/dev/dev_driver", O_WRONLY);
        if (fd<0){
                printf("Open Failured!\n");
                return -1;
        }
        ioctl(fd, IOCTL_SET_OPTION, argv[3]);
        command[0] = atoi(argv[1]);
        command[1] = atoi(argv[2]);
        ioctl(fd, IOCTL_COMMAND, command);
        close(fd);

        return 0;
}
```

Code 1

2) 타이머 디바이스 드라이버

- **ioctl**

  long device_ioctl(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param)

응용 프로그램으로부터 넘겨받은 Parameter에서 *ioctl_num*을 사용하여 스위치 케이스를 구분한다.

*IOCTL_SET_OPTION*일 경우 넘겨받은 데이터에 맞게 모든 디바이스들을 초기화한다.

IOCTL_COMMAND인 경우 타이머 디바이스를 설정한 후 실행한다.

device_module.c (partial)

```
long device_ioctl(struct file *file, unsigned int ioctl_num,
                  unsigned long ioctl_param){
        char gdata[4];
        int i;
        int data_param[2];
        switch (ioctl_num) {

        /* Initialize Device */
        case IOCTL_SET_OPTION:

                /* Ignore exceed 4bytes */
                if (copy_from_user(&gdata, (char *)ioctl_param, 4))
                        return -EFAULT;
                for(i=3; i>=0; i--){
                        if(gdata[i] != '0') break;
                }

                fnd_index = i;
                number_index_rest = (int)gdata[i] - 0x30;
                number_index = number_index_rest;

                /* Error : Start Number > 8 */
                if(number_index > 8)
                        return EINVAL;

                lcd_shift.line1 = 0;
                lcd_shift.line2 = 0;
                line1_sign = 1;
                line2_sign = 1;

                device_write_fnd(fnd_index, (char)number_index);
                device_write_led(number_index);
                device_write_dot(number_index);
                device_write_text_lcd(lcd_shift);
                break;

        /* Run Command */
        case IOCTL_COMMAND:
                if (copy_from_user(&data_param, (int *)ioctl_param, 8))
                        return -EFAULT;

                counter = 0;
                timer_interval = data_param[0];
                timer_count = data_param[1];
                del_timer_sync(&timer);
                timer.expires = jiffies + (timer_interval * CUSTOM_HZ);
                timer.function = kernel_timer_write;
                /* Not use
                * timer.data =
                */
                add_timer(&timer);
                break;
```

```
        default:
                break;
        }
        return SUCCESS;
}
```

Code 2

- **Timer Device**

  void kernel_timer_write(unsigned long timeout)

전역 변수 *counter*를 이용해 다음의 연산을 수행한다.

- ✓ *TIMER_CNT*만큼 타이머를 반복 실행
- ✓ *fnd_index*값 계산
- ✓ *number_count, number_idnex*값 계산

위에서 계산된 값으로 각 디바이스 출력 함수를 호출한다.

*counter*값이 *TIMER_CNT*가 되었을 시 모든 디바이스를 초기화 하고 프로그램을 종료한다.

device_module.c (partial)

```
void kernel_timer_write(unsigned long timeout) {
        int number_count = counter%8;
        counter++;
        if(counter == timer_count){
                /* Print 0 All of Using Devic */
                int i;
                outw(0,(unsigned int)fnd_addr);
                outw(0, (unsigned int)led_addr);
                for(i=0; i<10; i++){
                        outw(0,(unsigned int)dot_addr+i*2);
                }
                for(i=0; i<TEXT_LCD_MAX_LENTH; i+=2) {
                        outw(0x2020, (unsigned int)text_lcd_addr+i);
        }
                return ;
        }
        if(counter%8 == 0){
                fnd_index++;
                fnd_index %= 4;
        }
        number_index = (number_count + number_index_rest)%8 + 1;
        calc_lcd_shift();
        device_write_fnd(fnd_index, (char)number_index);
        device_write_led(number_index);
        device_write_dot(number_index);
```

```
        device_write_text_lcd(lcd_shift);

        timer.expires = get_jiffies_64() + (timer_interval * CUSTOM_HZ);
        timer.function = kernel_timer_write;
        /* Not use
        * timer.data =
        */
        add_timer(&timer);
}
```

Code 3

- **FND**

  void device_write_fnd(int index, char number)

문양을 출력할 자리 *index*와 문양의 값 *number*를 Parameter로 넘겨받아 FND에 출력한다.

device_module.c (partial)

```
void device_write_fnd(int index, char number) {
        unsigned short int value_short = 0;
        unsigned char value[4];
        value[0] = value[1] = value[2] = value[3] = 0;
        value[index] = number;
        value_short = value[0] << 12 | value[1] << 8 | value[2] << 4 | value[3];
    outw(value_short,(unsigned int)fnd_addr);
}
```

Code 4

- **LED**

  void device_write_led(int index)

FND에서 출력하는 문양의 값을 Parameter *index*로 넘겨받아 전역 변수로 미리 선언된 *led_number* index에 해당하는 값을 LED로 출력한다.

device_module.c (partial)

```
/* Global Variable */
…
int led_number[8] = {128, 64, 32, 16, 8, 4, 2, 1};
…
void device_write_led(int index) {
        unsigned short _s_value;
    _s_value = (unsigned short)led_number[index-1];
    outw(_s_value, (unsigned int)led_addr);
}
```

Code 5

- **DOT**

  void device_write_dot(int index)

FND에서 출력하는 문양의 값을 Parameter *index*로 넘겨받아 전역 변수로 미리 선언된 *fpga_number* index에 해당하는 값을 DOT로 출력한다.

device_module.c (partial)

```
void device_write_dot(int index) {
	int i;
	unsigned short int _s_value;
	unsigned char value[10];

	for(i=9; i>=0; i--){
		value[i] = fpga_number[index][i];
	}

	for(i=9; i>=0; i--){
	_s_value = value[i] & 0x7F;
		outw(_s_value,(unsigned int)dot_addr+i*2);
    }
}
```

Code 6

- **TEXT_LCD**

  void device_write_text_lcd(struct line_shift lcd_shift)

  void calc_lcd_shift()

LCD에 첫번째 줄에는 학번을, 두번째 줄에는 영문 이름을 출력한다. 기본적으로 *value*는 모두 공백으로 초기화 한다. shift기능을 구현하기 위해 Parameter로 *calc_lcd_shift()*에서 각 라인에 shift해야 하는 값이 미리 계산된 *lcd_shift*를 넘겨받는다. *lcd_shift*를 사용하여 value에 알맞게 학번과 이름을 넣은 뒤 LCD에 출력한다.

타이머가 실행 될 때마다 *calc_lcd_shift()*가 실행된다. *calc_lcd_shift()*에서는 각 라인(*lcd_shift.line1, lcd_shift_line2*)에 라인 sign값(*line1_sign, line2_sign*)을 더한다. 기본적으로 sign값을 1로 초기화 되어 있지만 각 라인에서 더 이상 shift될 수 없는 경우 -1을 곱해주어 1과 -1이 Switching될 수 있게 한다.

device_module.c (partial)

```
void device_write_text_lcd(struct line_shift lcd_shift) {
	int i;
	unsigned short int _s_value = 0;
	unsigned char value[TEXT_LCD_MAX_LENTH + 1];
	int HALF_LENTH = TEXT_LCD_MAX_LENTH/2;
```

```
        value[TEXT_LCD_MAX_LENTH] = 0;
        for(i = TEXT_LCD_MAX_LENTH - 1; i >= 0; i--){
                value[i] = ' ';
        }

        for(i = STUDENT_NUMBER_LENTH - 1; i >= 0; i--){
                value[i+lcd_shift.line1] = STUDENT_NUMBER[i];
        }

        for(i = STUDENT_NAME_LENTH - 1; i >= 0; i--){
                value[i + HALF_LENTH + lcd_shift.line2] = STUDENT_NAME[i];
        }
        /*
        for(i=0; i<STUDENT_NAME_LENTH; i++){
                value[i+HALF_LENTH+lcd_shift.line2] = STUDENT_NAME[i];
        }
        */
        for(i=0; i < TEXT_LCD_MAX_LENTH; i+=2){
        _s_value = (value[i] & 0xFF) << 8 | (value[i + 1] & 0xFF);
                outw(_s_value,(unsigned int)text_lcd_addr+i);
    }
}

/* Calculate TEXT LCD line Shift vlaue */
void calc_lcd_shift(){
        lcd_shift.line1 += line1_sign;
        lcd_shift.line2 += line2_sign;
        if(lcd_shift.line1%8 == 0){
                line1_sign *= -1;
        }
        if(lcd_shift.line2%5 == 0){
                line2_sign *= -1;
        }
}
```

Code 7

IV. 연구 결과

- 2개의 명령어를 사용하는 ioctl을 통한 디바이스를 구동하는 응용 프로그램 구현
- 디바이스 드라이버와 타이머 기능을 포함한 하나의 Module 구현
- 프로그램 전체 흐름도

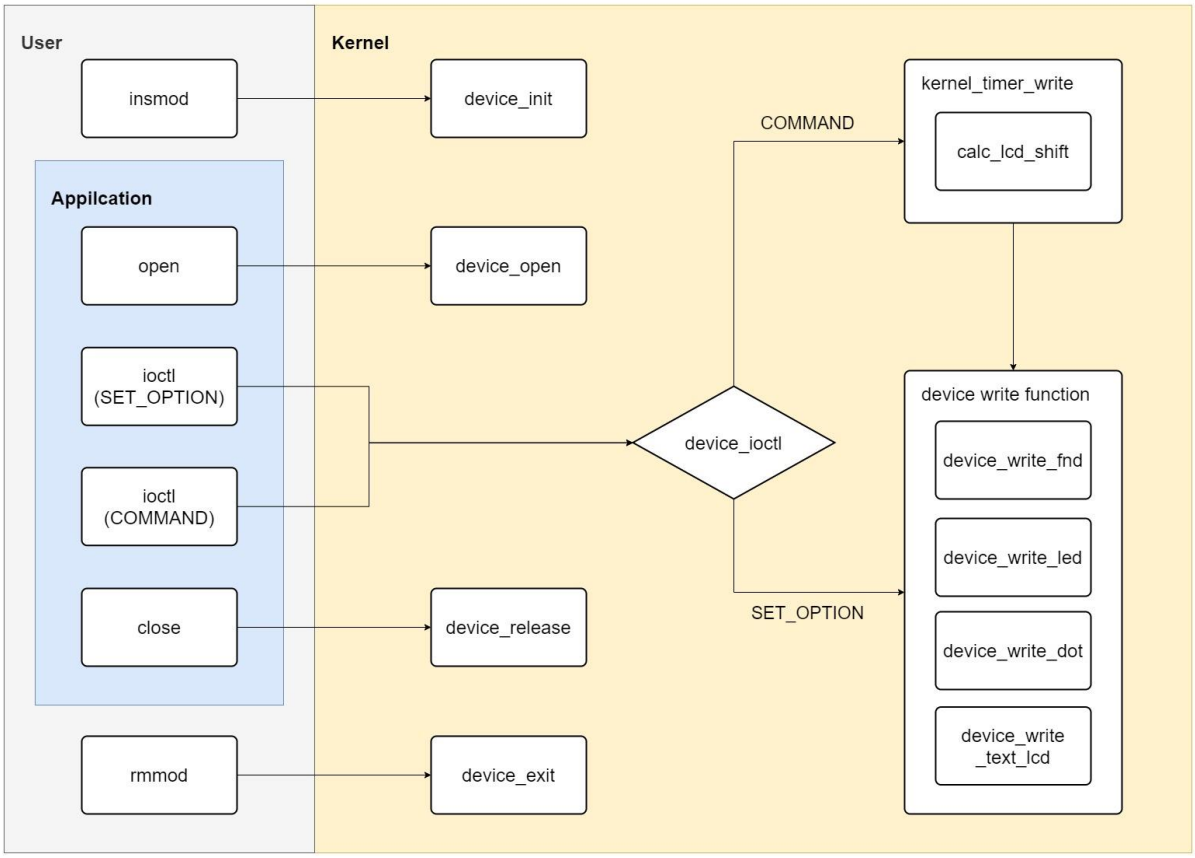

Figure 2

V. 기타

모듈을 잘못해서 작성할 경우 FPGA 시스템 자체가 다운 되는 경우를 접할 수 있었다. 시스템이 다운 되면 FPGA가 재부팅되며 초기상태로 되돌아가게 되는데 초기 설정부터 새로 해주어야 하게 된다. 이런 경우가 잦아지게 되면 적지 않은 시간을 소비하게 되기 때문에 모듈을 작성할 때 신중 해야할 필요가 있었다.

모듈을 작성한 후 업데이트를 할 때도 수동으로 rmmod/insmod를 해주어야 하기 때문에 단순한 Cross-Compile보다 더 많은 시간이 소비되었다. 이번 프로젝트를 진행할 때에는 자주 사용하는 명령어를 Ctrl-C,V를 통해 사용하였는데, 이러한 시간 소비가 더 오

래 지속되는 개발을 할 경우 자주 사용하는 명령어들을 스크립트를 작성해서 사용하면 좋을 것 같다는 생각이 들었다.

I. Appendix

device_module.c (entire)

```
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/io.h>
#include <asm/uaccess.h>

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

#include "./fpga_dot_font.h"

#define DEVICE_MAJOR 242            /* Device Driver Major Number */
#define DEVICE_NAME "dev_driver"    /* Device Driver Name */

#define FND_ADDRESS 0x08000004      /* FND Physical Address */
#define LED_ADDRESS 0x08000016      /* LED Physical Address */
#define DOT_ADDRESS 0x08000210      /* DOT Physical Address */
#define TEXT_LCD_ADDRESS 0x08000090 /* Text LCD Physical Address - 32 */

#define SUCCESS 0
#define STUDENT_NUMBER "20151556"
#define STUDENT_NAME "Byun HongSu"
#define STUDENT_NUMBER_LENTH 8
#define STUDENT_NAME_LENTH 11
#define TEXT_LCD_MAX_LENTH 32

#define IOCTL_SET_OPTION _IOW(DEVICE_MAJOR, 0, char *)
#define IOCTL_COMMAND _IOW(DEVICE_MAJOR, 1, int *)

#define CUSTOM_HZ HZ/10

/* Global Variable */
static int Device_Open = 0;
static unsigned char *fnd_addr;
static unsigned char *led_addr;
static unsigned char *dot_addr;
static unsigned char *text_lcd_addr;

int led_number[8] = {128, 64, 32, 16, 8, 4, 2, 1};
int fnd_index;
int number_index;
int number_index_rest;
int number_count;

struct line_shift{
       int line1;
       int line2;
```

```
} lcd_shift;
int line1_sign;
int line2_sign;

struct timer_list timer;
int timer_interval;
int timer_count;
int counter;

/* Function Prototype */
int __init device_init(void);
void __exit device_exit(void);
int device_open(struct inode *inode, struct file *file);
int device_release(struct inode *inode, struct file *file);
void device_write_fnd(int index, char number);
void device_write_led(int index);
void device_write_dot(int index);
void device_write_text_lcd(struct line_shift lcd_shift);
void calc_lcd_shift(void);
void kernel_timer_write(unsigned long timeout);
long  device_ioctl(struct  file  *file,  unsigned  int  ioctl_num,  unsigned  long
ioctl_param);

/*
* Define file_operations structure.
* This structure will hold the functions to be called when a process does something
to the device we created.
* Since a pointer to this structure is kept in the devices table, it can't be
local to init_module.
* NULL is for unimplemented functions.
*/
struct file_operations fops = {
       .owner                       =      THIS_MODULE,
       .open                        =      device_open,
       .unlocked_ioctl       =      device_ioctl,
       .release                     =      device_release
};

/*
* Module Declarations
*/

/* Initialize the module – Register the character device */
int __init device_init(void) {
       int result;
       result = register_chrdev(DEVICE_MAJOR, DEVICE_NAME, &fops);
       if(result < 0) {
              printk(KERN_WARNING "Can't get any major\n");
              return result;
       }
       led_addr = ioremap(LED_ADDRESS, 0x1);
       fnd_addr = ioremap(FND_ADDRESS, 0x4);
       dot_addr = ioremap(DOT_ADDRESS, 0x10);
       text_lcd_addr = ioremap(TEXT_LCD_ADDRESS, 0x32);

       init_timer(&(timer));
       printk("init module, %s major number : %d\n", DEVICE_NAME, DEVICE_MAJOR);
```

```
	return 0;
}

/* Exit the moudle - UnRegister the character device */
void __exit device_exit(void) {
	unregister_chrdev(DEVICE_MAJOR, DEVICE_NAME);
	iounmap(fnd_addr);
	iounmap(led_addr);
	iounmap(dot_addr);
	iounmap(text_lcd_addr);
	del_timer_sync(&timer);
	printk("Bye");
}

/*
* Function Declarations
*/

int device_open(struct inode *inode, struct file *file) {
	/* Don't want to run to two processes at the same time */
	printk("k test device open\n");
	if(Device_Open)
		return -EBUSY;
	Device_Open++;
	return SUCCESS;
}

int device_release(struct inode *inode, struct file *file) {
	/* Now Ready for our next caller */
	Device_Open--;
	return SUCCESS;
}

/* Wrtie FND Device */
void device_write_fnd(int index, char number) {
	unsigned short int value_short = 0;
	unsigned char value[4];
	value[0] = value[1] = value[2] = value[3] = 0;
	value[index] = number;
	value_short = value[0] << 12 | value[1] << 8 | value[2] << 4 | value[3];
    outw(value_short,(unsigned int)fnd_addr);
}

/* Write LED Device*/
void device_write_led(int index) {
	unsigned short _s_value;
    _s_value = (unsigned short)led_number[index-1];
    outw(_s_value, (unsigned int)led_addr);
}

/* Write Dot Device */
void device_write_dot(int index) {
	int i;
	unsigned short int _s_value;
	unsigned char value[10];
```

```
        for(i=9; i>=0; i--){
                value[i] = fpga_number[index][i];
        }

        for(i=9; i>=0; i--){
        _s_value = value[i] & 0x7F;
                outw(_s_value,(unsigned int)dot_addr+i*2);
    }
}

/* Write TEXT LCD Device */
void device_write_text_lcd(struct line_shift lcd_shift) {
        int i;
    unsigned short int _s_value = 0;
        unsigned char value[TEXT_LCD_MAX_LENTH + 1];
        int HALF_LENTH = TEXT_LCD_MAX_LENTH/2;

        value[TEXT_LCD_MAX_LENTH] = 0;
        for(i = TEXT_LCD_MAX_LENTH - 1; i >= 0; i--){
                value[i] = ' ';
        }

        for(i = STUDENT_NUMBER_LENTH - 1; i >= 0; i--){
                value[i+lcd_shift.line1] = STUDENT_NUMBER[i];
        }

        for(i = STUDENT_NAME_LENTH - 1; i >= 0; i--){
                value[i + HALF_LENTH + lcd_shift.line2] = STUDENT_NAME[i];
        }
        /*
        for(i=0; i<STUDENT_NAME_LENTH; i++){
                value[i+HALF_LENTH+lcd_shift.line2] = STUDENT_NAME[i];
        }
        */
        for(i=0; i < TEXT_LCD_MAX_LENTH; i+=2){
        _s_value = (value[i] & 0xFF) << 8 | (value[i + 1] & 0xFF);
                outw(_s_value,(unsigned int)text_lcd_addr+i);
    }
}

/* Calculate TEXT LCD line Shift vlaue */
void calc_lcd_shift(){
        lcd_shift.line1 += line1_sign;
        lcd_shift.line2 += line2_sign;
        if(lcd_shift.line1%8 == 0){
                line1_sign *= -1;
        }
        if(lcd_shift.line2%5 == 0){
                line2_sign *= -1;
        }
}

/* Write Timer */
void kernel_timer_write(unsigned long timeout) {
        int number_count = counter%8;
        counter++;
        if(counter == timer_count){
```

```
                /* Print 0 All of Using Devic */
                int i;
                outw(0,(unsigned int)fnd_addr);
                outw(0, (unsigned int)led_addr);
                for(i=0; i<10; i++){
                        outw(0,(unsigned int)dot_addr+i*2);
                }
                for(i=0; i<TEXT_LCD_MAX_LENTH; i+=2) {
                        outw(0x2020, (unsigned int)text_lcd_addr+i);
        }
                return ;
        }
        if(counter%8 == 0){
                fnd_index++;
                fnd_index %= 4;
        }
        number_index = (number_count + number_index_rest)%8 + 1;
        calc_lcd_shift();
        device_write_fnd(fnd_index, (char)number_index);
        device_write_led(number_index);
        device_write_dot(number_index);
        device_write_text_lcd(lcd_shift);

        timer.expires = get_jiffies_64() + (timer_interval * CUSTOM_HZ);
        timer.function = kernel_timer_write;
        /* Not use
        * timer.data =
        */
        add_timer(&timer);
}

long device_ioctl(struct file *file, unsigned int ioctl_num,
                  unsigned long ioctl_param){
        char gdata[4];
        int i;
        int data_param[2];
        switch (ioctl_num) {

        /* Initialize Device */
        case IOCTL_SET_OPTION:

                /* Ignore exceed 4bytes */
                if (copy_from_user(&gdata, (char *)ioctl_param, 4))
                        return -EFAULT;
                for(i=3; i>=0; i--){
                        if(gdata[i] != '0') break;
                }

                fnd_index = i;
                number_index_rest = (int)gdata[i] - 0x30;
                number_index = number_index_rest;

                /* Error : Start Number > 8 */
                if(number_index > 8)
                        return EINVAL;

                lcd_shift.line1 = 0;
```

```
                lcd_shift.line2 = 0;
                line1_sign = 1;
                line2_sign = 1;

                device_write_fnd(fnd_index, (char)number_index);
                device_write_led(number_index);
                device_write_dot(number_index);
                device_write_text_lcd(lcd_shift);
                break;

        /* Run Command */
        case IOCTL_COMMAND:
                if (copy_from_user(&data_param, (int *)ioctl_param, 8))
                        return -EFAULT;

                counter = 0;
                timer_interval = data_param[0];
                timer_count = data_param[1];
                del_timer_sync(&timer);
                timer.expires = jiffies + (timer_interval * CUSTOM_HZ);
                timer.function = kernel_timer_write;
                /* Not use
                * timer.data =
                */
                add_timer(&timer);
                break;

        default:
                break;
        }
        return SUCCESS;
}

module_init(device_init);
module_exit(device_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Huins");
```

Code 8