임베디드 소프트웨어 및 실습

- Term Project(움직이는 알람시계) -

담당교수님	안병철 교수님
학과	컴퓨터 공학과
조원	21413503 이우빈(조장) 21413500 손준우 21413506 최수연 21413507 최인자
제출날짜	2017.12.20

윤 리 서 약 서

팀장 : 이우빈

팀원 : 손준우, 최수연, 최인자

우리팀은 임베디드소프트웨어 및 실습 프로젝트를 수행함에 아래사항을 지키고 성실히 수행할 것을 서약합니다.

1. 인터넷이나 타른팀 혹은 선배들의 자료를 그대로 복사하지 않겠습니다.
2. 참고시 참고한 내용을 보고서의 참고문헌에 기재하겠습니다.

2017년 12월 5일

	이름	사인
팀장 :	이우빈	이우빈
팀원 :	손준우	손준우
팀원 :	최수연	최수연
팀원 :	최인자	최인자

컴퓨터공학과장 귀하

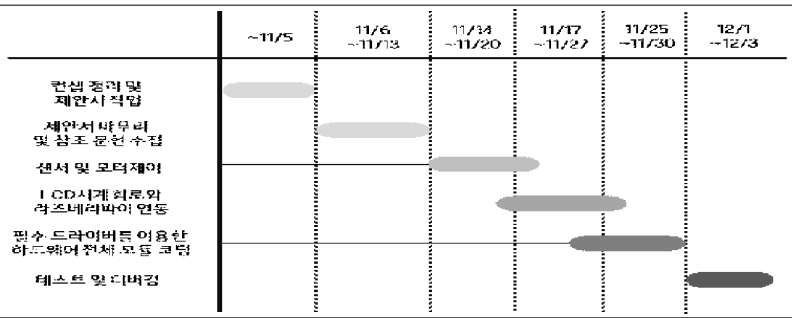
< INDEX >

1. 서론

1.1. 개발목적

이 프로젝트의 명칭은 움직이는 알람시계이다. 현대인들은 바쁜 일상생활 속에서 부족한 잠으로 피곤한 일상을 살고 있다. 일과를 마치고 늦은 시간에 잠이 깊게 들면 다음날 알람시계를 맞추어도 자신도 모르게 무의식 적으로 끄고 다시 잠드는 것 같은 경우들 때문에 알람이 무용지물인 상황이 생기기 마련이다. '움직이는 알람시계'는 이에 착안하여 개발을 하게 된 프로젝트이다. 이는 알람을 그냥 끄고 자는 것이 문제라면 알람을 끄는 것을 어렵게 하는 건 어떨까? 같은 생각에서 시작되었다. '움직이는 알람시계'는 울리자마자 사용자가 손에 바로 잡아서 끌 수 없게 랜덤 한 방향으로 빠르게 움직인다. 이렇게 되면 사용자는 알람을 듣고 바로 끄고 다시 자고 싶어도 움직이는 알람시계를 잡으려고 일어나게 되고 잡으러 다니는 과정에서 잠이 깨게 될 것이다. '알람시계를 끈다.'는 과정에서 어려움을 발생시켜 자의가 아닌 외부의 영향으로 잠을 쉽게 깰 수 있다는 점이 주요한 점이다.

1.2. Project Proposal

Team Members	이우빈, 손준우, 최수연, 최인자
Objectives	Moving alarm clock
Functions	- 알람 시간을 설정함. - 설정된 시간이 되면 노래가 나오고 바퀴를 이용해 랜덤한 방향으로 움직임. - 알람시계를 잡아서 버튼을 누르면 알람이 꺼짐,
Constraints (제한사항)	- 비용/기간 : 30,000원 내외 / 약 5주 - 윤리 : 서약서 - 디자인 : GUI 설계 제한적 - 하드웨어/소프트웨어적 요소: 메모리 크기, 프로세서기능, 언어 등
Background & Basic Knowledge	- 라즈베리 파이(마이크로 프로세서)에 대한 기본 지식 - 소프트웨어 공학 - 리눅스와 C프로그래밍
Resource & Reading Material	- 라즈베리 파이 매뉴얼 - 임베디드 프로세서 - 수업자료 등
Procedures & schedule	① Outline : 지정한 시간이 되면 알람이 울리는데, 이때 랜덤한 방향으로 알람 시계가 움직이고 사용자가 이를 잡아 정해진 버튼을 누르면 알람이 멈춘다. User cases : 기존의 알람 기능만으로는 기상하는데 어려움이 있는 모든 사람 ② System model - 시중의 일반적인 알람시계, 아이디어 요소가 있는 알람시계 - 아두이노 또는 라즈베리를 이용한 기존의 프로젝트. ③ User interface : LCD시계, 버튼 등 ④ Test plan : 2017.12.01~2017.12.11
Project Schedule	
Demonstration Guide	STEP 1: L298N 모터드라이브로 8883M Motor 10100(RPM) 속도로 움직이는 모터의 방향을 random 하게 움직인다. STEP 2: 4000 (+-) 500Hz 의 크기의 MH-FMD 부저가 지정 시간이 되면 울린다. STEP 3: EPX37XMU Dot LED가 지정시간이 되면 'UP' 이라는 단어를 보여준다. STEP 4: 스위치를 누르면 인터럽트가 발생하면서 모든 기능들이 종료 된다.
Notes	최종 문서 제출 할 때 프로젝트 회의 일지 제출

1.3 제안서 대비 스펙 표

구분	제안스펙	구현스펙	만족	사유
모터 속도	10100 RPM	10100 RPM	만족	.
모터 방향 제어	random 한 방향으로 움직임	random 한 방향으로 움직임	만족	.
부저 울림 크기	3500~4500Hz	3500~4500Hz	만족	.
dot led	2초를 간격으로 'UP'이라는 문자가 깜빡거리며 출력	2초를 간격으로 'UP'이라는 문자가 깜빡거리며 출력	만족	.
알람 울림	사용자가 설정한 시간에 부저가 울림	사용자가 설정한 시간에 부저가 울림	만족	.
스위치	스위치를 누르면 모든 기능을 종료.	스위치를 누르면 모든 기능을 종료.		

1.4 설계제한사항

현실적 제한조건	현실적 제한조건을 해결한 내용
1. 경제	설계 전체 비용을 3만원 이내로 함.
2. 기간	제작 기간은 5주 이내로 제작.
3. 윤리	윤리 서약서를 준수함.
4. 미적	GUI Design을 포함하지 않음.
5. 환경	host : Linux Ubuntu 16.04 target : raspbian backend : ethernet cable

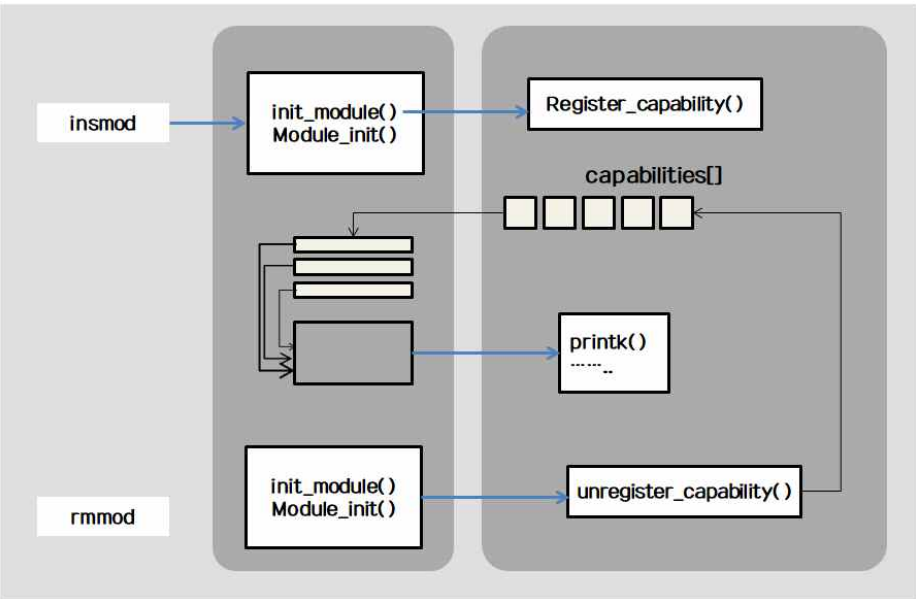
2. 개발방법

2.1. 개발방법

1) MicroSD 초기화
① 라즈비안 이미지 파일을 다운 받는다.
② 이미지파일의 압축을 푼다.
③ microSD에 이미지를 설치한다.
④ 라즈베리파이의 네트워크를 설정한다.
⑤ 호스트의 네트워크를 설정한다.
⑥ 라즈베리파이부팅을 한 후 로그인을 한 후 터미널에서 설정 한다.
⑦ 라즈베리파이터미널에서 NFS를 테스트한다.

2) 커널 설치
① 커널소스를 다운로드한다.
② 커널 컴파일을 한다.
③ 커널이미지와 모듈을 복사한다.

3) 커널모듈
① 커널모듈 작성
② 커널모듈 컴파일
③ 모듈적재(insmod) 및 제거(rmmod)



※모듈 명령어

- insmod : 모듈을 설치
- rmmod : 실행중인 모듈을 제거
- lsmod : 로드된 모듈들의 정보를 표시
- depmod : 커널 내부에 적재된 모듈간의 의존성을 검사한다.
- modprobe

: 모듈 간 의존성을 검사하여 그 결과 누락된 다른 모듈을 찾아서 적재한다.

- modinfo : 목적파일을 검사해서 관련된 정보를 표시

4) 디바이스드라이버

① 커널모듈의 형태로 디바이스 드라이버 함수 작성

-struct file_operations 정의 및 함수 구현

-init_module, cleanup_module 정의 및 함수 구현

② 컴파일

-makefile을 작성하여서 make로 컴파일 수행

-이후 make install명령으로 nfs폴더로 드라이버명.ko파일과 test파일을 이동시킨다.

③ ssh로 라즈베리파이 접속

④ cd nfs명령으로 nfs폴더로 이동

⑤ 노드생성

-mknod /dev/파일이름 드라이버타입 주번호 부번호

⑥ 디바이스 드라이버 적재

-insmod 드라이버명.ko

⑦ 드라이버의 적재 여부 확인

-lsmod

⑧ 디바이스 드라이버를 삭제하려면 밑과 같은 명령으로 삭제

-rmmod 드라이버명

2.2. 커널모듈구현방법 및 프로그램 설명

움직이는 알람시계에는 커널 버전 "4.4 50 –v7+"을 사용한다. 커널 모듈과 디바이스 드라이버 프로그램은 모두 리눅스 우분투에서 작성을 한다. 본 프로그램을 실행하기 위해서는 유저가 사용하는 디바이스(Raspberry Pi 3)에서 커널에 적재(loading) 및 삭제(unloading)가 이루어 져야 한다.

전체적인 순서는 리눅스(우분투) 에서 make 파일을 통해 컴파일을 한 후 디바이스와 mount, mknod를 통해 노드를 생성한다. 이후 insmod를 통해 커널에 적재를 시키고 마지막으로 실행파일을 실행시킨다.

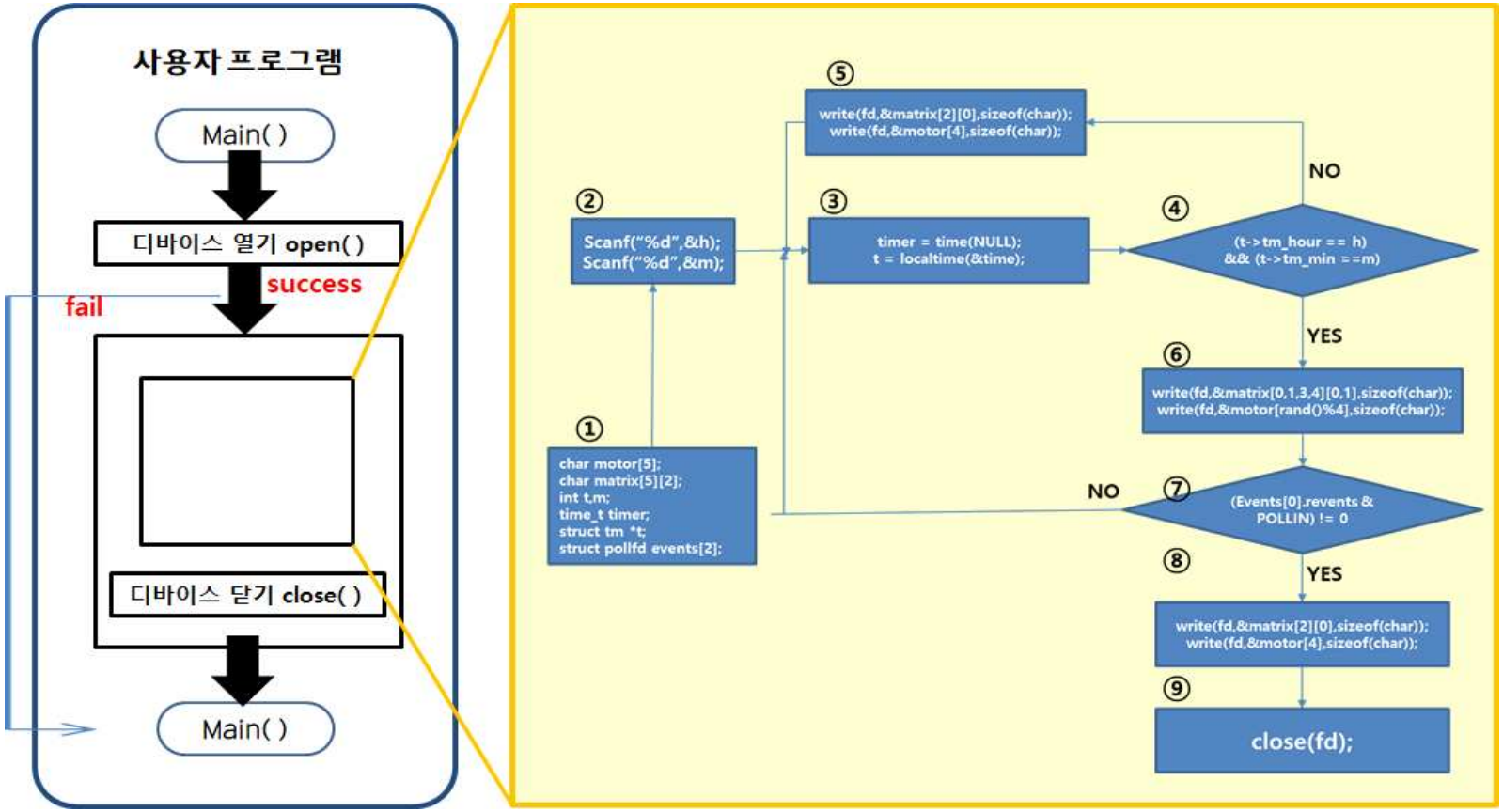
본 프로그램에서는 커널 모듈 함수로는 module_init, module_exit을 사용 한다.

이외 디바이스 드라이버로 사용된 함수는 irqreturn_t ind_interrupt_handler, poll, open, release, write를 사용한다. 아래 표는 사용된 함수들과 설명을 표로 정리한 것이다.

함수	기능 설명
• static irqreturn_t ind_interrupt_handler (int irq, void *pdata)	인터럽트 서비스 루틴 설정 함수이다. 스위치를 눌렀을 때 인터럽트 발생시키도록 하는 함수이며 Alarm을 끌 때 사용 한다.
• static unsigned key_poll (struct file *mfile, struct poll_table_struct *pt)	인터럽트 종료 함수이다. 인터럽트가 발생했었더라면 인터럽트 설정들을 모두 종료시킨다.
• static int _open (struct inode *minode, struct file *mfile)	디바이스 노드에 의해 첫 번째로 수행되는 함수 이다. open 함수에서 gpio select 와 mode 설정 및 인터럽트 요청을 한다.
• static int _release (struct inode *minode, struct file *mfile)	디바이스를 닫을 때 쓰는 함수 이다.
• static int _write (struct file *mfile, const char *gdata, size_t length, loff_t *off_what)	디바이스에 데이터를 쓰기 위해 쓰는 함수이다. write 함수에서 gpio를 set,clear를 설정 하며 프로그램 전체 동작이 이 함수에서 쓰여 진다.
• static int _init(void)	드라이버 적재 함수이다.
• static void _exit(void)	드라이버 제거 함수이다.
• int register_chrdev (unsigned int major,const * name,struct file_operations * fops)	커널에 지정되어 있는 chrdevs 구조에 새로운 char device 등록
• int unregister_chrdev (unsigned int major,const * name,struct file_operations * fops)	등록된 char device 제거

2.3. Flow Chart

아래의 FLOWCHART는 움직이는 알람시계의 전체적 알고리즘을 순서도로 나타낸 것이다. 아래페이지의 FLOWCHART에서 핵심 알고리즘에 대한 각 기능들의 설명이다.



① 변수선언
② 알람시간 입력(시간,분 단위로 입력)
③ OS의 현재 시간들에 대한 정보를 받는다.
④ OS의 시간과 user가 입력한 시간이 일치하면 YES 아니면 NO
⑤ 일반상태(gpio가 모두 clear 인 상태)
⑥ Alarm 활성화 및 모터 동작(Dot Matrix, 모터, BUZEER gpio 모두 set)
⑦ switch를 눌렀을 경우(interrupt 발생 했을 경우)면 YES 아니면 NO
⑧ 기능은 ⑤번과 동일, 이 경우는 스위치를 눌러서 모두 clear 됐다는 의미
⑨ 파일 close

3. 결론

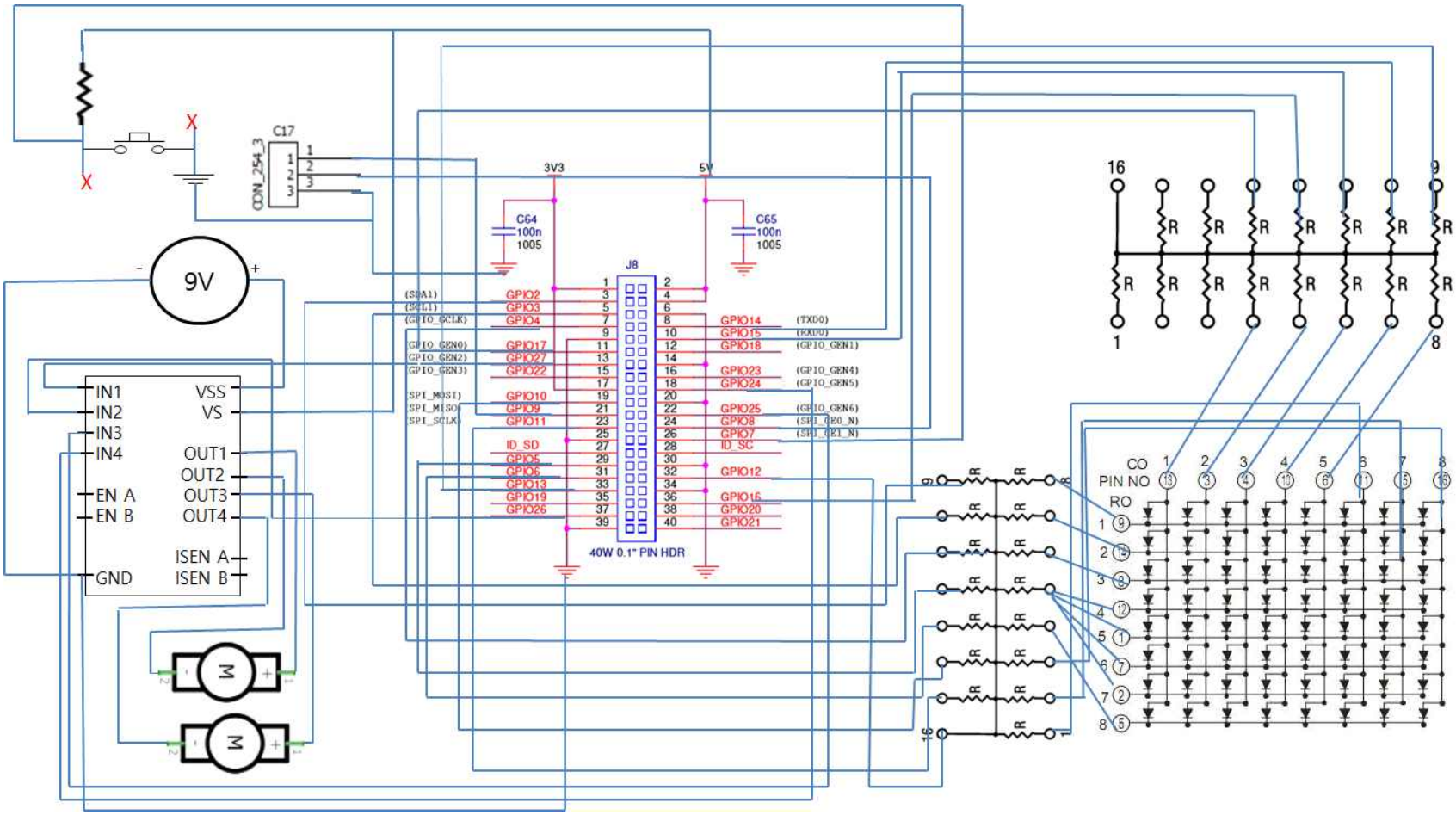
이번 프로젝트를 통해서 우리가 실현 하고 싶었던 것은 우리가 수업시간에 배운 지식을 최대한 활용하여 재미있고 실용적인 작품을 만드는 것이었다. 활용할 수 있을 것이라고 생각되는 디바이스는 dot led와 스위치, 모터 정도였는데 이것들을 기반으로 어떠한 제품을 만들 것인지 정하는 것부터 많은 우여곡절이 있었다.

평소 수업시간이나 실습시간을 통해 기본적인 이론과 실습을 하면서 어느 정도는 이해를 했다고 생각해왔다. 하지만 생활에서 사용할 만한 제품을 실제로 만드는 데는 또 다른 차원의 이해가 필요했다. 생각 한 것을 실제로 구현하는 과정에서 결론이 생각처럼 나오지 않은 적이 많았고 이를 고치고 해결하는 과정에서 좀 더 깊은 수준의 이해를 할 수 있었다.

서로 임베디드에 관한 이해도도 모두 다르고 프로젝트에 대해 생각하는 방향도 다른 조원들이 모여 역할을 분배하고 서로 모르는 것들을 물어보며 프로젝트를 완성하는 과정이 결코 순탄하거나 쉽진 않았다. 하지만 모르는 것은 도와주고 의견을 공유하며 프로젝트를 완성하다 보니 그 속에서 많은 것을 배울 수 있었다.

4. 부록

4.1. 회로도


C17
CON_254_3
3V3
5V
C64
100n
1005
C65
100n
1005
J8
40W 0.1" PIN HDR
9V
IN1
IN2
IN3
IN4
EN A
EN B
GND
VSS
VS
OUT1
OUT2
OUT3
OUT4
ISEN A
ISEN B
M
M
GPIO2
GPIO3
GPIO4
GPIO17
GPIO27
GPIO22
GPIO10
GPIO9
GPIO11
ID_SD
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
GPIO14
GPIO15
GPIO18
GPIO23
GPIO24
GPIO25
GPIO8
GPIO7
ID_SC
GPIO12
GPIO16
GPIO20
GPIO21
(TXD0)
(RXD0)
(GPIO_GEN1)
(GPIO_GEN4)
(GPIO_GEN5)
(GPIO_GEN6)
CO
PIN NO
RO
R

4.2. 코드

1)led_driver.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/slab.h>
#include <linux/gpio.h>
#include <mach/platform.h>
#include <linux/io.h>
#include <linux/poll.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/sched.h>
#include <linux/wait.h>


#define _MAJOR  221                                             //디바이스 드라이버 주번호
#define _NAME   "LED_DRIVER"                                    //디바이스 드라이버 이름

#define _SIZE   256                                             //gpio 물리적 주소

char _usage = 0;                                                //드라이버 사용 체크
static void *_map;                                              //물리적 매핑을 받기 위한 포인터
volatile unsigned *led;                                         //gpio 설정을 위한 포인터
static int event_flag = 0;                                      //이벤트 발생횟수 체크

DECLARE_WAIT_QUEUE_HEAD(waitqueue);                             //이벤트 체크할 대기큐 생성

static irqreturn_t ind_interrupt_handler(int irq, void *pdata)          //ISR 설정 함수
{
    *(led + 13) & (1 << 7)                                              //gpio7 (스위치 GPIO IN)을 읽는다

    wake_up_interruptible(&waitqueue);                                  //인터럽트 활성화
    ++event_flag;

    return IRQ_HANDLED;                                                 //인터럽트 handler 리턴
}

static unsigned key_poll(struct file *mfile, struct poll_table_struct *pt)  //인터럽트 종료 함수
{
    int mask =0;                                                            //인터럽트 상태 체크 변수

    poll_wait(mfile, &waitqueue, pt);                                       //큐에 대기 시킨다 (sleep로 변형)
    if(event_flag > 0)
        mask |= (POLLIN | POLLRDNORM);                      //읽을 일반 데이터가 존재(인터럽트가 발생했었다)

    event_flag = 0;                                                                 //flag 값을 다시 0

    return mask;                                                                    //상태 return
}

static int _open(struct inode *minode, struct file *mfile)                  //디바이스 open 함수
{
    int i,k;                                                                //gpio select를 위한 반복문 변수
    int result;                                                             //요청한 인터럽트 결과확인 변수
    if (_usage != 0)                                                    //드라이버가 0이 아니면 이미 open되었다
        return -EBUSY;                                                      //는 뜻이므로 return시켜 함수 종료
    _usage = 1;                                                             //드라이버가 open 되었음을 체크
```

```
    _map = ioremap(GPIO_BASE, _SIZE);                                   //물리적 매핑
    if (!_map)                                                          //매핑 오류처리
    {
        printk("error: mapping gpio memory");
        iounmap(_map);                                                  //매핑 되어 있을경우 매핑을 끊는다
        return -EBUSY;
    }

    led = (volatile unsigned int *)_map;                                //레지스터로 접근하기 위한 4바이트 포인터지정

for(k=0;k<=2;k++)                                                       //gpio 0~27까지 모두 OUT으로 설정
    for(i=0;i<=9;i++)
    {
       if(k == 0 && (3*i == 21)) continue;                              //단, gpio 7은 IN으로 설정을 위해 넘어간다
      *(led + k) &= ~(0x7 << (3 * i));                                  //gpio select
      *(led + k) |= (0x1 << (3 * i));                                   //set gpio out
    }

    *(led) &= ~(0x7 << (7*3));                                          //gpio 7 select
    *(led + 22) |= (0x1 << 7);                                          //Set GPFEN0 to 1 to detect Falling edge

    result = request_irq(gpio_to_irq(7), ind_interrupt_handler,IRQF_TRIGGER_FALLING, "gpio_irq_key", NULL);
                                                                        //gpio 7번에 대한 인터럽트 셋팅

    if(result < 0)                                                      //인터럽트 요청에 실패
    {
        printk("error: request_irq()");
        return result;
    }
    return 0;
static int _release(struct inode *minode, struct file *mfile)           //디바이스 release(close) 함수
{
    _usage = 0;                                                         //드라이브상태 0로 변경
    if (led)
        iounmap(led);                                                   //드라이브 매핑 해제
    free_irq(gpio_to_irq(7), NULL);                                     //ISR 해제
    return 0;
}


static int _write(struct file *mfile, const char *gdata, size_t length, loff_t *off_what)
{                                                                       //디바이스 데이터 write 함수
    char tmp_buf;                                                     //gpio set 할 번호를 선택하는 변수
    int result
    int left,right;                                                     //copy from user 함수 상태 확인 변수
                                                          //left : gpio address 설정, right : gpio shift 크기 설정

            /* 아래의 16진수 비교문들은 user가 보낸 16진수에 따라 gpio가 동적으로 write 된다.*/

    if(0xFF == *gdata)                                                  //Matrix OFF(gpio clear)
    {
        left = 10;
        right = 0;
    }

    if(0x80 == *gdata || 0x60 == *gdata || 0x18 == *gdata || 0x07 == *gdata || 0x01 == *gdata)
    {                                                                   //gpio 10~19번 write
        left = 7;
        right = 10;
    }
```

```
    if(0x6B == *gdata || 0x77 == *gdata || 0x3F == *gdata)                          //gpio 0~9번 write
    {
        left = 7;
        right = 0;
    }

    if(0xC0 == *gdata || 0x30 == *gdata || 0x20 == *gdata || 0x10 == *gdata)        //gpio 20~29번 write
    {
        left = 7;
        right = 20;
    }


    result = copy_from_user(&tmp_buf, gdata, length);                   //user 에서 데이터를 받아온다
    if (result < 0)                                                     //데이터를 정상적으로 못받아 왔으면 0
    {
        printk("Error: copy from user");
        return result;
    }

    printk("data from app : %d₩n", tmp_buf);

    if(0x00 == *gdata)          //Dot exit                              //Dot Matrix, BUZZER gpio clear
    {
        *(led+10) = (0xFF << 0);                                        //0~9 gpio clear
        *(led+10) = (0xFF << 10);                                       //10~19 gpio clear
        *(led + 10) = (0x01 << 8);                                      //gpio 8 clear
        *(led + 10) = (0x01 << 9);                                      //gpio 9 clear
    }

    else if(0x02 == *gdata)                                             //Motor gpio clear
    {
        *(led+10) = (0xFF << 20);                                       //20~39 gpio clear
    }

    else
    {
        *(led + left) = (tmp_buf <<  right);        //16진수 조건문에서 설정한 left,right 값에따라 gpio 설정
        *(led + 10) = (0x01 << 8);                                      //BUZZER gpio 8 활성화
        *(led + 7) = (0x01 << 9);                                       //BUZZER gpio 9 활성화
    }
    return length;
}

static struct file_operations _fops =                                   //파일연산 구조체
{
    .owner      = THIS_MODULE,                                          //모듈 소유자
    .open       = _open,                                                //open
    .release    = _release,                                             //release
    .write      = _write,                                               //write
    .poll       =key_poll,                                              //poll
};

static int _init(void)                                                  //드라이버 적재 함수
{
    int result;                                                         //드라이버 적재 상태 변수

    result = register_chrdev(_MAJOR, _NAME, &_fops);                    //드라이버를 적재 한다
    if (result < 0)                                                     //드라이버 적재에 실패
    {
```

```
    else if(0x02 == *gdata)                                     //Motor gpio clear
    {
        *(led+10) = (0xFF << 20);                               //20~39 gpio clear
    }

    else
    {
        *(led + left) = (tmp_buf <<  right);        //16진수 조건문에서 설정한 left,right 값에따라 gpio 설정
        *(led + 10) = (0x01 << 8);                              //BUZZER gpio 8 활성화
        *(led + 7) = (0x01 << 9);                               //BUZZER gpio 9 활성화
    }
    return length;
}

static struct file_operations _fops =                           //파일연산 구조체
{
    .owner      = THIS_MODULE,                                  //모듈 소유자
    .open       = _open,                                        //open
    .release    = _release,                                     //release
    .write      = _write,                                       //write
    .poll       =key_poll,                                      //poll
};

static int _init(void)                                          //드라이버 적재 함수
{
    int result;                                                 //드라이버 적재 상태 변수

    result = register_chrdev(_MAJOR, _NAME, &_fops);            //드라이버를 적재 한다
    if (result < 0)                                             //드라이버 적재에 실패
    {
        printk(KERN_WARNING "Can't get any major!₩n");
        return result;
    }
    return 0;
}

static void _exit(void)                                         //드라이버 제거 함수
{
    unregister_chrdev(_MAJOR, _NAME);                           //드라이버 제거
    printk("module removed.₩n");
}


module_init(_init);                                             //커널모듈 적재
module_exit(_exit);                                             //커널 적재모듈 제거

MODULE_LICENSE("GPL");                                          //모듈 라이센스
```

2)test_led.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <linux/kdev_t.h>
#include <time.h>
#include <linux/poll.h>
#include <signal.h>

#define _FILE_NAME  "/dev/led_driver"       //드라이브 위치
#define MAX_VALUE 100                       //8x8 Dot Matrix에서 불을 on,off 하기 위해
                                            //반복하는 loop의 최대값

int main(int argc, char **argv)
{
    time_t timer;                           //time.h 파일에서 운영체제 시간을 구하기위한 변수
    struct tm *t;                           //시간값을 거내는 구조체 포인터 변수
    struct pollfd events[2];                //체크할 이벤트 (switch 눌러짐) 정보를 저아할 변수
    int retval;                             //event 상태 체크 변수
    int fd;                                 //디바이스 상태 체크
    int i,j,k;                              //Dot Matrix, Motor를 수행하기 위한 loop의 변수들
    int h,m,r;                              //시간을 입력하기 위한 변수

    char matrix[5][2] = {                   //8x8 Dot LED를 켜기위한 gpio 16진수 배열
        {0xFF,0x80},    //matrix[0~4][0]는 Dot matrix row부분을 제어하고 matrix[0~4][1]은 Dot matrix col부분을 제어
        {0x3F,0x60},
        {0x00,0x18},
        {0x6B,0x07},
        {0x77,0x01},
    };

    char motor[5] = {                       //모터  제어를 위한 gpio 16진수 배열
        0xC0,0x30,                          //후진,전진
        0x20,0x10,                          //좌회전,우회전
        0x02                                //exit
    };


    fd = open(_FILE_NAME, O_RDWR);                      //디바이스 오픈

    if (fd < 0)                                         //디바이스가 정상적으로 오픈되지 않으면 오류 처리후 종료
    {
        fprintf(stderr, "Can't open %s₩n", LED_FILE_NAME);
        return -1;
    }


    printf("Please input the hour ");
    scanf("%d",&h);                                     //알람 맞출 시간 입력

    printf("Please input the minute ");
    scanf("%d",&m);                                     //알람 맞출 분 입력


    while (1)
    {

        timer = time(NULL);                             //운영체제 현재시각을 초단위로 얻기위한 함수
```

```
        t = localtime(&timer);                                  //초 단위 시각을 분리하여 변수에 넣는다

        events[0].fd    =fd;                                    //체크할 이벤트 디스크립터
        events[0].events    = POLLIN;                           //데이터 읽기, 비트값으로 지정

        retval = poll(events, 1, 1000);                         //event waiting(swich가 눌러지길 기다림)
                                                                //events: 이벤트정보, 1: 이벤트 수,1000: 대기시간
        if (retval < 0)                                         //event 오류 처리
        {
            fprintf(stderr, "Poll error₩n");
            exit(0);
        }



       if((t->tm_hour == h) && (t->tm_min == m))        //운영체제 시간과 자신이 설정한 시간이 일치 -> Alarm 시작
       {

        r = rand() % 4;                                 //모터 방향을 랜덤으로 설정하기 위한 인덱스와 rand함수
        if(events[0].revents & POLLIN)                  //스위치를 눌렀을경우  -> 발생한 이벤트가 POLLIN인지 비교
        {
            write(fd,&motor[4],sizeof(char));       //모터종료 (motor[4]는 모터관련 gpio를 모두 clear시키겠다는 의미)
            write(fd,&matrix[2][0],sizeof(char));
            //Dot Matrix를 모두 끈다(matrix[2][0]은 Dot Matrix gpio선들을 모두 clear 시키겠다는 의미)
            break;
            //스위치는 알람을 끈다는 의미이므로 프로그램 종료(while(1) 에서 나온다)
        }


        for( k=0;k< MAX_VALUE ;k++)                             //Dot Matrix를 제어하기 위한 반복문
        {
            for(i=0;i<5;i++)
                //matrix 배열 크기는 matrix[5][2] 이다. 이것은 5까지 순회를 해야 단어가 출력이 된다는 의미
            {
                write(fd,&matrix[2][0],sizeof(char));
                                        //초기값 또는 다음 matrix를 켜기 위해서는 이전 matrix값을 clear 한다.
                write(fd,&motor[r],sizeof(char));               //rand 함수를 통해 입력받은 값을통해 모터방향 제어
                for(j=0;j<2;j++)                                //col 부분 제어
                    write(fd,&matrix[i][j],sizeof(char));       //Matrix을 반복문을 돌면서 set 시킨다.
                usleep(1000);                                   //1000 마이크로초 시간동안 Matrix[i][j] 값을 출력
            }

            if(events[0].revents & POLLIN)                      //스위치를 눌렀을경우
            {
                write(fd,&motor[4],sizeof(char));               //모터종료
                write(fd,&matrix[2][0],sizeof(char));           //Matrix값 모두 clear
                break;
            }

        }

        printf("Wake UP!!!!!!₩n");
        write(fd,&matrix[2][0],sizeof(char));   //Matrix값 clear  -> 이부분을 통해 Matrix는 On , off를 반복한다.
        write(fd,&motor[r],sizeof(char));       //모터방향 제어
```

```
        if(r == 2 || r==3)                          //좌회전 또는 우회전을 할경우 1초동안 전진 한다.
        {
            write(fd,&motor[1],sizeof(char));
            sleep(1);
        }

        write(fd,&motor[4],sizeof(char));           //모터값 모두 clear -> 모터 최종 종료

        if(events[0].revents & POLLIN)              //스위치를 눌렀을경우
        {
            write(fd,&motor[4],sizeof(char));       //모터종료
            write(fd,&matrix[2][0],sizeof(char));   //Matrix값 모두 clear
            break;
        }
        }

       else                                         //일반상태 (Alarm 상태가 아닌경우)
       {
            write(fd,&matrix[2][0],sizeof(char));   //Matrix gpio값 모두 clear
            write(fd,&motor[4],sizeof(char));       //모터 gpio값 모두 clear
            printf("%d : %d₩n",t->tm_hour,t->tm_min);   //현재 운영체제 시간,분을 출력
       }

    }
    close(fd);                                      //파일 close
    return 0;
}
```

4.3. 회의일지

번호	날짜	시간	지난주 회의내용	이번 주 회의내용 및 이행사항
1	11/2	18:00 ~ 21:00	1, 개인별 주제 조사	1. 주제 결정 2. 제안서 작성 3. 참고 문헌 조사
2	11/10	12:00 ~ 14:00	1. 주제 결정 2. 제안서 작성 3. 참고 문헌 조사	1. 재료 조사 2. 설계 계획 3. 역할 분배
3	11/17	12:30 ~ 15:00	1. 재료 조사 2. 설계 계획 3. 역할 분배	1. 제안서 수정 2. 재료 결정
4	11/24	12:00 ~ 15:00	1. 제안서 수정 2. 재료 결정	1. 모터 제어 2. 발표 준비 3. Dot Led 제어
5	12/1	12:10 ~ 16:00	1. 모터 제어 2. 발표 준비 3. Dot Led 제어	1. 발표 준비 2. 스위치 제어 3. 추가 재료 결정
6	12/8	12:00 ~ 15:00	1. 발표 준비 2. 스위치 제어 3. 추가 재료 결정	1. 하드웨어 디자인 2. 테스트 및 디버깅 3. 시연연습