임베디드소프트웨어
프로젝트 보고서

제출일자 : 2019. 12. 09.

소속 : 컴퓨터공학과

학번&이름 : 21511784 이태원

21511727 김성준

21511762 여정동

21511791 정종현

21720827 배민체

목 차

1. 서　　론

1. 프로젝트의 목적 및 내용

본 과목에서 한 학기 동안 배운 임베디드 소프트웨어 개발 관련 지식을 응용하여 필수 드라이버와 응용 SW, 그 외 Embdded kit, 안드로이드 드라이버, Bootload 분석,microprocessor kit, 기타 임베디드 HW를 사용하여 실제로 구동하는 Device를 개발하는 것이다. (단순 게임, 디스플레이 SW 등 User 공간에서만의 SW프로그램 구현은 안됨)

2. 설계제한사항

2.1. 경제(부품포함)

총 제작단가는 40,000원 이하로 한다. 본 프로젝트에 사용된 물품들의 단가(기본적으로 제공된 Raspberry Pi, 서보모터 (3개), Dot Matrix를 제외)는 폼보드 2장 4,000원, 점퍼케이블 약 3,000원, TCS3200D 센서 4,400원, 그 외 기타 부품들의 가격 1,000원을 합한 약 12,400원이다.

2.2. 기간

1. 프로젝트 공지 2019년 10월 13일
2. 프로젝트 제안서 발표 2019년 10월 22일
3. 프로젝트 최종 발표, 데모 2019년 12월 2일
4. 프로젝트 최종 보고서 제출 2019년 12월 9일

2.3. 윤리

본 프로젝트의 결과물은 윤리적으로 문제가 될 소지가 없음.

2.4. 미적

본 프로젝트의 결과물은 미관상 일반적으로 타인에게 혐오감을 불러일으킬 소지가 없음.

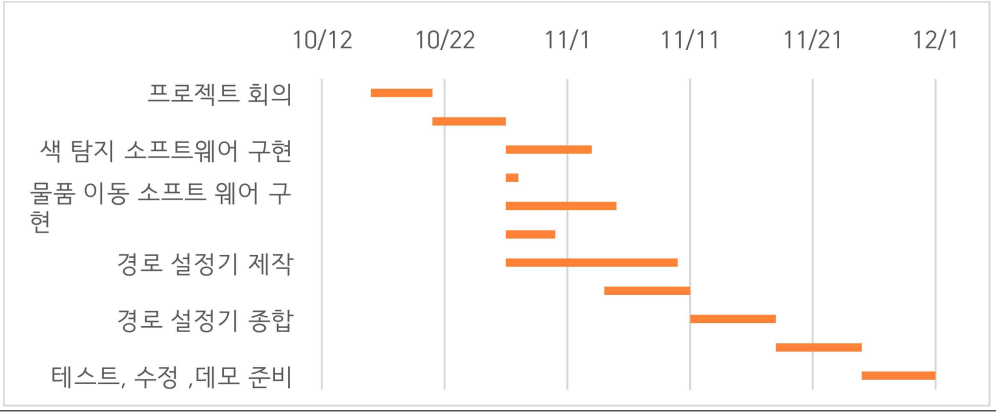
2.5. 산업표준

본 프로젝트의 결과물은 임의로 단기간 내 제작된 것으로 산업표준을 별도로 심도 있게 고려하지 않았으나, 제작 시 SI 단위계 기준으로 시간, 길이, 각을 측정하였음.

2. 본　　론

1. 제안서 spec과 실제 구현 비교

1.1. 기존 제안서

Team Members	이태원 21511784 김성준 21511727 여정동 21511762 정종현 21511791 배민체 21720827
Objectives	색깔 분류기는 색을 기준으로 물건들을 분류하는 시스템이다. 모드는 두 가지가 있으며 첫 번째는 사용자가 원하는 색만을 분류하는 모드이며 두 번째는 정해진 여러 가지의 색으로 물건들을 분류하는 모드이다.
Functions	1. 모드 선택 -사용자는 두 가지 모드 중 하나를 선택 할 수 있다. -첫 번째 모드는 사용자가 원하는 색만을 분류하는 모드이다. -두 번째 모드는 정해진 여러 가지의 색으로 물건들을 분류하는 모드이다. 2. 색 탐지 -카메라의 아웃풋을 바탕으로 어떤 색인지 판별한다. 3. 물건 분류 -탐지된 색을 기준으로 모터를 이용해 색에 맞는 경로를 설정한다. -첫 번째 모드에선 입력한 색 이외에는 default로 한다. -두 번째 모드에선 정해진 색 이외의 다른 색이 인풋으로 들어올 경우 default로 한다. 4. 분류한 색 카운트 -색깔 별로 감지된 물건의 개수가 몇 개인지 LED로 표시한다. 5. 물건 이동 -모터를 이용해서 색을 탐지할 물건을 색 탐지 영역으로 이동시킨다.
Constraints	1. 색을 판별하는 임계값을 적절히 설정해야한다. 2. 물건을 정확한 위치로 분류시켜야 한다. 3. 분류할 물건의 크기는 동일해야한다. 4. 원판의 구멍이 물건 크기보다 커야한다. 5. 조명이 너무 밝거나 너무 어두우면 안된다.
Project Schedule	
Demonstration Guide	1. 작동 : 시스템을 작동 시키고 터미널 창이 열린다. 2. 모드 설정 : 터미널 창에 원하는 모드를 입력하고 결정한다. 3. 모터에 의한 물건 이동 : 물건을 분류한 직후나 모드가 결정된 이후에 다음 물건을 색

	탐지 영역으로 이동시킨다. 4. 카메라에 의한 색 탐지 : 물건이 색 탐지 영역으로 이동된 이후 카메라는 물건의 RGB 값을 탐지하여 시스템에 저장한다. 5. 경로 설정 : 저장한 RGB 값을 로드하여 해당 모드의 조건에 부합한지 판별하고 모터에 의해 해당 색 영역으로 경로가 설정된다. 6. 개수 카운트 및 매트릭스에 표시 : 경로 설정에서 판별한 결과 값인 색 이니셜을 매트릭스에 표시 후 누적 개수를 표시한다. 또한 터미널에서도 누적 개수를 확인할 수 있다. 7. 물건 낙하 : 카운트 단계가 끝난 후 물건을 낙하 시키고 3번 단계로 돌아간다. 혹은 유저가 정지신호를 보낼 경우 시스템을 끝마치고 터미널에 물건의 개수를 출력한다.
Notes	

1.2. 기존 제안서 spec과 실제 구현 결과의 차이

	기존 제안서	실제 구현 결과
Objectives	색깔 분류기는 색을 기준으로 물건들을 분류하는 시스템이다. 모드는 두 가지가 있으며 첫 번째는 사용자가 원하는 색만을 분류하는 모드이며 두 번째는 정해진 여러 가지의 색으로 물건들을 분류하는 모드이다.	모드는 한 가지만 존재하며 미리 정해놓은 여러 색으로 물건들을 분류.
Functions	1. 모드 선택 -사용자는 두 가지 모드 중 하나를 선택할 수 있다. -첫 번째 모드는 사용자가 원하는 색만을 분류하는 모드이다. -두 번째 모드는 정해진 여러 가지의 색으로 물건들을 분류하는 모드이다. 3. 물건 분류 -탐지된 색을 기준으로 모터를 이용해 색에 맞는 경로를 설정한다. -첫 번째 모드에선 입력한 색 이외에는 default로 한다. -두 번째 모드에선 정해진 색 이외의 다른 색이 인풋으로 들어올 경우 default로 한다.	1. 사용자는 모드를 선택 할 수 없다. 3. 미리 정해진 색상에 따라서만 경로가 설정된다. 그리고 정해진 색 이외의 다른 색이 감지되면 프로그램을 종료한다.
Demonstration Guide	2 .모드 설정 : 터미널 창에 원하는 모드를 입력하고 결정한다. 4. 카메라에 의한 색 탐지 : 물건이 색 탐지 영역으로 이동된 이후 카메라는 물건의 RGB 값을 탐지하여 시스템에 저장한다.	2. 모드 설정은 하지 않는다. 4 .컬러 센서를 이용하여 RGB값을 구별한다.
그 외 나머지 부분은 동일		

2. 개발방법

2.1. 코드 작성, 컴파일

Device driver code와 Device 드라이버를 동작시키는 User code는 기본적으로 개인 PC의 리눅스(Ubuntu) 가상머신에서 작성하여 컴파일하였다. 컴파일하여 생성한 *.ko 파일과 실행 파일은 github에 업로드 한 후, Raspberry Pi에서 다운로드하는 방식으로 전달하였다.

2.2. Raspberry Pi

2.2.1. Raspberry Pi 3B

개인용 PC에서 리눅스 환경을 이용해 작성, 컴파일한 프로그램을 실행시키는 환경이다. port를 통해 5v, 3.3v의 전원을 부품에 공급할 수 있다. 또한, 27개의 GPIO port로 각 부품을 연결하여, 프로그래밍 된 동작을 수행하도록 한다.

- SoC : Broadcom BCM2837 SoC
- CPU : 1.2GHz ARM Cortex-A53 MP4
- GPU : Broadcom VideoCore IV MP2 400 MHz
- Memory : 1GB LPDDR2
- 영상 출력 : 컴포지트 HDMI(rev 1.3 & 1.4) DSI
- GPIO : 40pin

2.2.2. BCM2837

Raspberry Pi 3 모델에서 사용되는 Broadcom SoC(System-on-a-chip)칩의 명칭이다. BCM2837는 Memory mapped I/O 방식을 사용하므로, GPIO를 조작하기 위해 설정해야 할 Register의 주소가 memory에 mapping되어 있다. 따라서 우리는 해당 Register의 주소에 직접 접근하여 원하는 동작을 하도록 GPIO를 제어할 수 있다. Register에는 주소 값 0x7E200000부터 0x7E2000B0까지 각 GPIO가 입력, 출력, level 또는 edge 감지 등의 동작을 수행하도록 제어할 수 있는 필드가 존재한다.

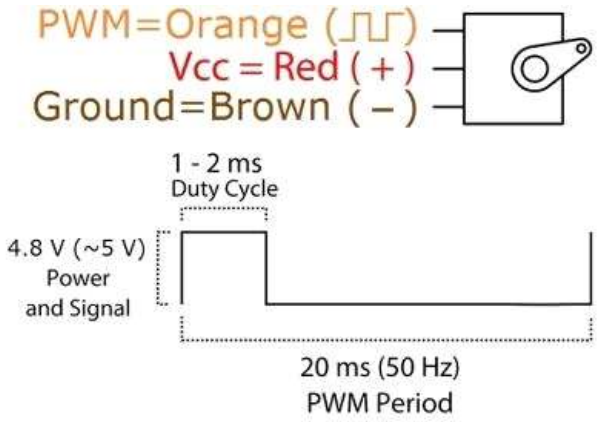
Adress	Field	Function
0x7E20 0000~0014	GPFSELn	GPIO pin의 작동을 정의하는 데 사용. Raspberry Pi 보드의 2번부터 27번까지의 GPIO pin이 각 3비트씩 할당되어 있다. 작동을 정의하고자 하는 pin의 위치에 000 또는 001을 mapping 함으로써 각각 Input용 또는 Output용으로 사용할 것인지를 정의할 수 있다.
0x7E20 001C~0020	GPSETx	GPIO pin의 출력을 1로 설정하는 데 사용된다. 해당 레지스터 주소 위치에서 설정하고자 하는 GPIO pin의 번호에 해당하는 위치의 비트에 1을 쓰면 해당 번호의 GPIO pin의 출력이 자동으로 1로 set 된다.
0x7E20 0028~002C	GPCLRx	GPIO pin의 출력을 0로 설정하는 데 사용된다. 해당 레지스터 주소 위치에서 설정하고자 하는 GPIO pin의 번호에 해당하는 위치의 비트에 1을 쓰면 해당 번호의 GPIO pin의 출력이 자동으로 0로 clear 된다.
0x7E20 0058~005C	GPFENx	GPIO pin이 Falling edge를 감지하도록 설정한다. 해당 event가 발생하면 GPEDSx 레지스터에 비트를 1로 설정한다. 인터럽트를 호출하기 위해서 사용한다.

위 표는 개발한 프로그램에서 사용한 Register의 주소 필드이다. 위 주소 필드들을 사용하여 원하는 번호의 GPIO 핀의 동작 상태를 정의할 수 있다. 모터를 동작시키는 Pulse를 조절할 때와 Dot-Matrix에 문자를 표시할 때, 그리고 센서의 색상 필터를 교체할 때 GPSET, GPCLR 필드를 사용하였고, 센서가 출력하는 주파수를 계산하기 위해 사용할 GPIO pin이 Falling edge를 감지하도록 GPFEN 필드에서 설정해 주었다.

2.3. 사용된 주요부품

2.3.1. 180도 Servo motor

- 모델명 : SG90
- 단가 : ₩ 1500
- 역할과 구조 : Servo motor(이하 모터)의 동작을 통해 해당 모터의 부착물로 카지노 칩을 이동시키는 역할을 한다. VCC 전원 port, Ground port, 모터 동작을 제어하기 위한 Pulse를 받는 port가 각각 Raspberry Pi에 연결된다.
- 동작 원리 :



Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

위 그림은 SG90 Data sheet의 일부를 발췌해 온 것이다. 위 사진에서 펄스의 주기는 20ms이고 High 펄스의 폭이 1~2ms인 pwm 파형을 보내 서보모터의 동작을 제어한다. 이 주기 안에서 High 신호의 시간이 얼마인가에 따라 모터의 회전 각도가 결정된다. 이 시간은 해당 제품의 Data sheet에 명시되어있다. High 신호가 1.0ms인 펄스의 경우 –90도에 위치, 1.5ms 펄스는 0도에 위치, 2ms 펄스는 +90도에 위치하도록 설정되어있다.

2.3.2. Color sensor

- 모델명 : TCS3200D
- 단가 : ₩ 4,400
- 역할과 구조 : 사물의 색상을 감지하여 주파수를 출력한다.

TERMINAL NAME	NO.	I/O	DESCRIPTION
GND	4		Power supply ground. All voltages are referenced to GND.
$\overline{OE}$	3	I	Enable for f_o (active low).
OUT	6	O	Output frequency (f_o).
S0, S1	1, 2	I	Output frequency scaling selection inputs.
S2, S3	7, 8	I	Photodiode type selection inputs.
V_{DD}	5		Supply voltage

VCC 전원 port, 두 개의 Ground(하나는 LED에 연결됨) port, 주파수 scale을 조절하는 S0, S1 port와 검출할 색상(Red, Green, Blue)을 지정하는 S1, S2 port, 그리고 발생한 주파수를 출력하는 OUT port, 그리고 가운데 색상을 감지하는 부분이 주요 구성 요소이다. OE port는 OUT port의 Enable 여부를 설정할 수 있고, 밝기가 충분하지 않은 곳에서도 색상을 잘 검출하기 위해 LED port를 통해 백색 LED를 점등할 수 있다.

- 동작 원리 :

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

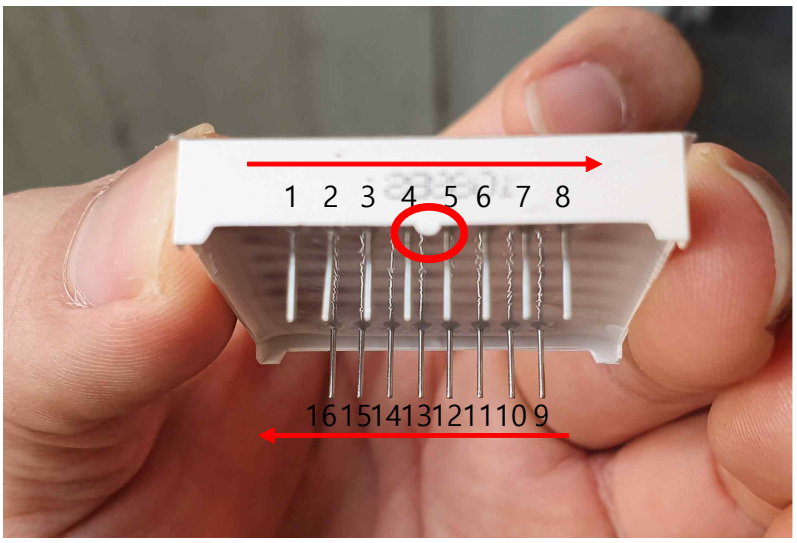
S0, S1의 값으로는 센서가 출력하는 주파수의 Scale을 조절할 수 있다. S0, S1가 모두 High일 경우, 센서는 최대(100%) scale의 주파수를 출력하며, 이때의 최대 크기는 약 500kh에서 600khz이다. 출력되는 주파수의 값이 크면 색상을 보다 세분화하여 구별할 수 있다는 장점이 있지만, 20% 정도의 주파수로도 원하는 색상을 충분히 구별할 수 있기에, 본 프로젝트에서는 S0 High, S1을 Low로 하는 20%의 주파수 scale을 사용하였다.

S2, S3의 값으로는 센서의 Photo Diode가 검출할 색상을 교체할 수 있다. 한 사물에 대

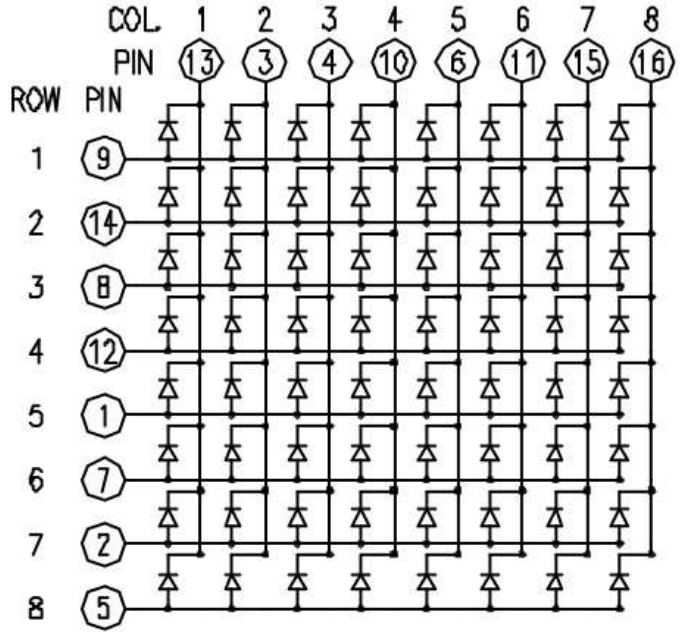
하여 Red, Green, Blue 각각의 주파수 값을 각각 구하여 적절히 범위를 설정해준다면 여러 가지 색상을 구별할 수 있게 해준다. 주파수는 검출하고자 하는 색상의 밝기 정도에 따라 결정된다. 예를 들어 S2, S3가 모두 Low인, 센서가 Red 색상을 검출하는 상태라고 할 때, 센서는 Red Color만 통과하고 나머지 색은 통과하지 않는 필터를 씌운다. 그때 해당 사물의 색상이 밝게 인식된다면 출력되는 주파수의 크기가 커지는 원리이다. 다시 말하자면, Red를 검출하는 상태일 경우에 측정하는 사물의 색상이 Red color에 가깝다고 출력되는 주파수의 크기가 무조건 증가하는 것이 아니라, 해당 사물의 색깔의 밝기가 얼마냐 밝게 인식되느냐에 따라 결정된다는 것이다. 예를 들어 센서가 Green을 측정하는 상태일 때 Yellow color가 Green color보다 주파수가 더 크게 출력된다. 그 이유는 색상이 Green color보다 더 밝은 편에 속하기 때문이다. 사물의 색이 가장 밝은색인 흰색에 가깝다면 Red, Green, Blue 상태일 때 각각 출력되는 주파수의 크기가 모두 증가한다. 반대로 가장 어두운색인 검은색에 가깝다면 Red, Green, Blue 상태일 때 각각 출력되는 주파수의 크기가 모두 감소한다.

2.3.3. 8X8 Dot Matrix

- 모델명 : 10888S
- 단가 : ₩ 2,200
- 역할과 구조 :

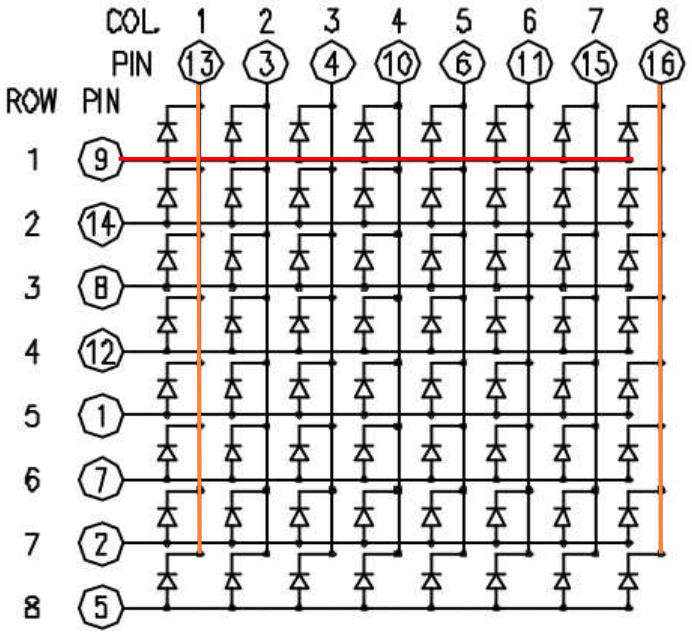


row 8열, col 8열 총 16개의 핀으로 구성되어 있다. 중앙에 반원 모양으로 튀어나온 면을 기준으로 왼쪽 핀부터 1번이며 시계방향 순으로 번호가 mapping된다.



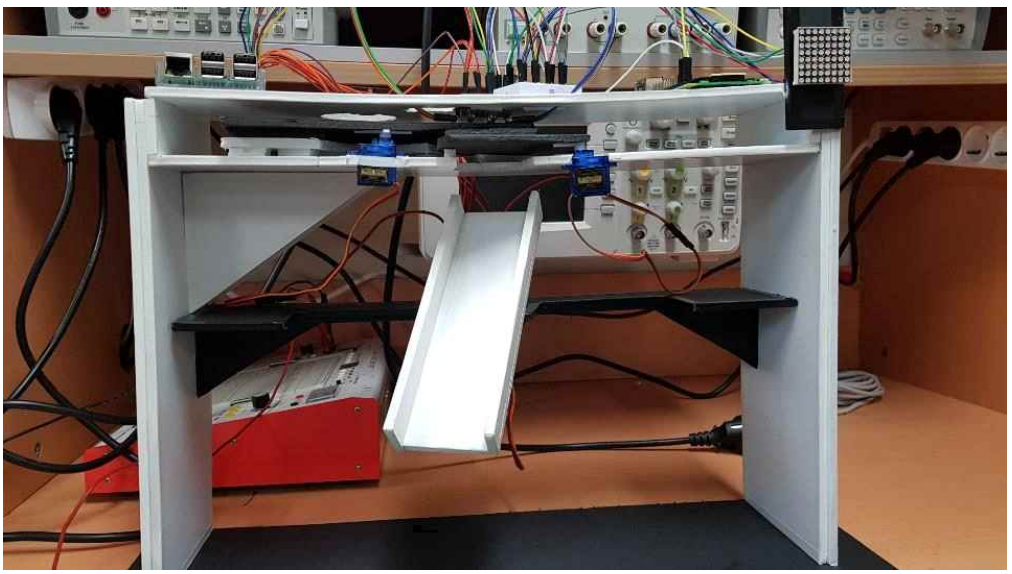
번호 순서와 row, col의 순서는 전혀 관계가 없으며 위의 회로도에 의하여 row, col에 해당하는 핀 번호가 결정된다. 예를 들면 첫 번째 열에 대하여 제어를 원할시, 매트릭스의 9번 핀에 High 또는 Low값을 주어 제어를 할 수 있다.

- 동작 원리 :



Anode형태의 회로이므로 row열의 High, Low제어를 이용하여 각각의 열의 led를 점등하였다. 현재 9번 핀에 High값을 준 상태라고 가정하였다(9pin : red line). 전류는 전압이 높은 곳에서 낮은 곳으로 흐른다. col에 가해준 전압이 없다면 전류의 흐름으로 인하여 모든 첫 번째 열의 led가 점등이 된다. 만약 1행, 8행을 제외한 나머지 행의 led를 점등하려고 한다면, 13번 핀 및 16번 핀에 High값을 준다면(9pin : red line, 13pin&16pin orange line) 같은 전압에 의해 전류가 흐르지 않게 되므로 해당 행의 led는 점등하지 않게 된다. 이와 같은 High, Low를 이용하여 한 열의(8 led) 제어가 가능하다. 이후 다음 열의 제어를 할 경우 이전 열에 High값을 Low로 전환하여야 한다. 만약 1열, 2열에 동시에 High값을 부여할 경우 1열에서 생성한 led의 모양과 같은 led의 모양이 2열에서도 생성이 된다. 즉, 각 열마다 다른 led의 모양을 표시하기 위해서는 이전 열의 High값을 Low로 전환 및 8개의 행 값 clear, 현재 열에 High값을 할당, 이후 8개의 행에 의한 원하는 모양을 출력을 실행, 일정 딜레이 부여, 현재 열의 High값을 Low로 전환 및 8개의 행 값 clear ~ 이 과정을 반복하여야지 원하는 모양의 매트릭스를 출력할 수 있다.

2.4. 구조물



위 그림 1은 실제 프로젝트 결과물 전체의 전면 부를 촬영한 사진이다. 외면을 구성하는 모든 부분은 폼 보드로 제작되었으며, 각각의 필요한 위치에 모터와 센서를 부착했다. 그림 기준 왼쪽 상단부에 데모에 사용될 사물이 투입되고, 각각의 모터 동작을 이용해 사물을 원하는 위치로 이동시키는 방식으로 설계하였다.

2.5. 테스트 대상 선정 (카지노 칩)

테스트할 물건은 카지노 칩으로 선정했다. 카지노 칩은 색상 구별이 또렷할 뿐만 아니라, 균일한 크기와 무게를 가지고 있어 프로젝트의 결과물이 성공적으로 작동하는지를 명확하게 확인할 수 있었다.

3. 커널 모듈 구현

3.1. Motor X

3.1.1. Device 정의

Motor X는 180도 서보모터이다. 센서에서 물건의 색상 주파수를 정해진 시간만큼 출력한 후, 그에 따라 물건이 떨어질 경로가 지정되고 나면 물건을 떨어뜨리는 역할을 하는 판을 움직인다.

3.1.2. Device driver code

```
#define motor180x_MAJOR 222
#define motor180x_NAME  "MOTOR180X_DRIVER"
#define GPIO_SIZE       256
#define GPIO            10
```

Device driver의 Major number는 222로 설정하였고, GPIO pin은 10번을 사용한다.

```
static int motor180x_open(struct inode *inode, struct file *mfile){
    if(motor180x_usage != 0)
        return -EBUSY;
    motor180x_usage=1;

    motor180x_map=ioremap(GPIO_BASE, GPIO_SIZE);

    if(!motor180x_map){
        printk("error:mapping gpio memory");
        iounmap(motor180x_map);
        return -EBUSY;
    }
    motor180x=(volatile unsigned int*)motor180x_map;

    *(motor180x+(GPIO/10))&=~(0x07<<(3*(GPIO%10)));
    *(motor180x+(GPIO/10))|=(0x01<<(3*(GPIO%10)));

    return 0;
}
```

open 함수에서 GPSEL1 필드의 10번 pin에 해당하는 위치에 001을 mapping하여 10번 GPIO pin을 Output mode로 설정하였다.

```
static int motor180x_write(struct file *minode, const char *gdata, size_t length, loff_t *off_what){
    char tmp_buf;
    int result;
    int frequency=20000;
    int minAngle=544;
    int maxAngle=2440;
    int x;
    unsigned int setClock;
    unsigned int clrClock;
    unsigned long stpDelay = jiffies + 2*HZ;

    result=copy_from_user(&tmp_buf, gdata, length);
    if(result<0){
        printk("Error:copy from user");
        return result;
    }
    printk("data from user: %d\n", tmp_buf);
```

write 함수에서 모터의 동작을 정의한다. 이 부분은 함수 내 상단의 변수 선언 부분이며 frequency, minAngle, maxAngle 변수들은 모터를 제어하기 위한 펄스를 생성하는데 최적화된 값을 설정하였다. stpDealy는 딜레이를 주기 위한 변수이고 jiffies의 값을 이용한다. jiffies이란 시스템 내의 전역적인 변수로써 초당 HZ (1초에 동일한 주기를 반복하는 횟수) 만큼 숫자가 증가한다. 조금 더 예를 들어 설명하면, HZ의 값이 100이라고 가정하고 현재 jiffies가 50이라 할 때, jiffies의 값이 150으로 변한다면 1초가 지났음을 알 수 있다는 것이다. 따라서 해당 stpDealy를 선언과 동시에 초기화하기 위한 코드의 의미는 stpDelay를 2초로 설정한다는 의미이다. copy_from_user 함수를 사용하여 User 영역에서 커널 영역으로 전달하고자 하는 값을 전달받을 수 있다.

```
    unsigned long delay = jiffies + (int)tmp_buf*HZ;
    while(time_before(jiffies, delay)){}
    for(x=90;x>=45;x--){
        setClock=(minAngle+((maxAngle-minAngle)/90*x));
        clrClock=frequency-setClock;
        *(motor180x+7)=(0x01<<GPIO);udelay(setClock);
        *(motor180x+10)=(0x01<<GPIO);
        udelay(clrClock);
    }
    while(time_before(jiffies, stpDelay)){}
    for(x=45;x<=90;x++){
        setClock=(minAngle+((maxAngle-minAngle)/90*x));
        clrClock=frequency-setClock;
        *(motor180x+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180x+10)=(0x01<<GPIO);
        udelay(clrClock);
    }
```

위의 코드는 write 함수에서 모터로 보낼 펄스 신호의 길이를 제어하는 부분이다.

'minAngle+((maxAngle-minAngle)/90*x))'는 x의 값에 따라 모터에 보낼 펄스 신호의 길이를 제어하기 위한 공식이다. '(maxAngle-minAngle)/90' 이 부분을 통해 전체 180도를 90으로 나눌 수 있으며, 이에 따라 x가 1 변화할 때마다 모터가 움직이는 각도는 2도씩 변화한다. 따라서 위의 코드에서 x가 90일 때 setclock의 크기가 최대가 되고, 이 크기만큼 GPSET1 필드를 사용해 해당 GPIO pin에 High level 펄스 신호를 모터에 보내면 2.3.1에서 설명한 동작 원리에 따라 모터가 처음 위치하는 각도는 +90도가 되고, x가 1씩 감소할 때마다 –2도씩 움직이게 된다. 반복문이 진행되면서 x가 점차 45까지 감소하게 되므로 모터는 총 –90도만큼 움직이게 된다. 그리고 위에서 설정한 딜레이 만큼 기다린 후, 다시 x를 증가시키며 High level 펄스 신호를 보내는 시간을 증가시켜 모터를 다시 원위치시키는 동작을 정의하였다.

3.2. Motor Y

3.2.1. Device 정의

Motor Y는 180도 서보모터이다. 투입구로 삽입된 칩을 센서가 있는 위치까지 운반하는 판을 움직인다.

3.2.2. Device driver

```
#define motor180y_MAJOR 223
#define motor180y_NAME  "MOTOR180Y_DRIVER"
#define GPIO_SIZE       256
#define GPIO        11
```

Device driver의 Major number는 223으로 설정하였고, GPIO pin은 11번을 사용한다.

```
static int motor180y_open(struct inode *inode, struct file *mfile){
    if(motor180y_usage != 0)
        return -EBUSY;
    motor180y_usage=1;

    motor180y_map=ioremap(GPIO_BASE, GPIO_SIZE);

    if(!motor180y_map){
        printk("error:mapping gpio memory");
        iounmap(motor180y_map);
        return -EBUSY;
    }
    motor180y=(volatile unsigned int*)motor180y_map;

    *(motor180y+(GPIO/10))&=~(0x07<<(3*(GPIO%10)));
    *(motor180y+(GPIO/10))|=(0x01<<(3*(GPIO%10)));

    return 0;
}
```

open 함수에서 GPSEL1 필드의 11번 pin에 해당하는 위치에 001을 mapping하여 11번 GPIO pin을 Output mode로 설정하였다.

```
static int motor180y_write(struct file *minode, const char *gdata, size_t length, loff_t *off_what){
    char tmp_buf;
    int result;
    int frequency=20000;
    int minAngle=544;
    int maxAngle=2440;
    int x;
    unsigned int setClock;
    unsigned int clrClock;
    unsigned long delay = jiffies + 1*HZ;

    result=copy_from_user(&tmp_buf, gdata, length);
    if(result<0){
        printk("Error:copy from user");
        return result;
    }
    printk("data from user: %d\n", tmp_buf);
```

write 함수 내 상단의 변수 선언 부분이며 3.1.2에서 선언한 것과 상당 부분 동일하나 여기서는 jiffies를 사용하여 delay값을 1초로 선언하였다.

```
if(tmp_buf==0){
    for(x=0;x<=24;x++){
        setClock=(minAngle+((maxAngle-minAngle)/40*x));
        clrClock=frequency-setClock;
        *(motor180y+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180y+10)=(0x01<<GPIO);
        udelay(clrClock);
    }

}

else{
        for(x=24;x>=0;x--){
        setClock=(minAngle+((maxAngle-minAngle)/40*x));
        clrClock=frequency-setClock;
        *(motor180y+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180y+10)=(0x01<<GPIO);
        udelay(clrClock);
    }

}
```

위의 코드는 write 함수에서 모터로 보낼 펄스 신호의 길이를 제어하는 부분이다. User

영역으로부터 전달받은 값인 tmp_buf의 값에 따라 모터가 다르게 움직이도록 설정하였다. 모터의 동작을 코드로 구현한 원리는 3.1.2에서 설명한 것과 동일하다. tmp_buf가 0일 경우 x가 1 증가할 때 마다 모터의 각도를 +7.5도씩 변화시켜 –90도에서 +18도까지 총 108도를 움직이도록 설정하였고, tmp_buf가 0이 아닐 경우, 반대로 +18도에서 –90도까지 다시 되돌아가도록 설정하였다.

3.3. Motor Z

3.3.1. Device 정의

Motor Z는 180도 서보모터이다. 색상 탐지가 끝나면 해당 색상에 맞게끔 경로를 바꿔주는 판을 움직인다.

3.3.2. Device driver

```
#define motor180z_MAJOR 224
#define motor180z_NAME  "MOTOR180Z_DRIVER"
#define GPIO_SIZE       256
#define GPIO        12
```

Device driver의 Major number는 224으로 설정하였고, GPIO pin은 12번을 사용한다.

```
static int motor180z_open(struct inode *inode, struct file *mfile){
    if(motor180z_usage != 0)
        return -EBUSY;
    motor180z_usage=1;

    motor180z_map=ioremap(GPIO_BASE, GPIO_SIZE);

    if(!motor180z_map){
        printk("error:mapping gpio memory");
        iounmap(motor180z_map);
        return -EBUSY;
    }
    motor180z=(volatile unsigned int*)motor180z_map;

    *(motor180z+(GPIO/10))&=~(0x07<<(3*(GPIO%10)));
    *(motor180z+(GPIO/10))|=(0x01<<(3*(GPIO%10)));

    return 0;
}
```

open 함수에서 GPSEL1 필드의 12번 pin에 해당하는 위치에 001을 mapping하여 11번 GPIO pin을 Output mode로 설정하였다.

```
static int motor180z_write(struct file *minode, const char *gdata, size_t length, loff_t *off_what){
    char tmp_buf;
    int result;
    int frequency=20000;
    int minAngle=544;
    int maxAngle=2440;
    int x;
    unsigned int setClock;
    unsigned int clrClock;
    unsigned long delay;


    result=copy_from_user(&tmp_buf, gdata, length);
    if(result<0){
        printk("Error:copy from user");
        return result;
    }
    printk("[motz]data from user: %d\n", result);
```

write 함수 내 상단의 변수 선언 부분이며 3.1.2에서 선언한 것과 상당 부분 동일하며, 설명 또한 동일함.

```
if(tmp_buf==1){
    printk("[motz] Active 1\n");
    for(x=0;x<=30;x++){
        setClock=(minAngle+((maxAngle-minAngle)/180*x));
        clrClock=frequency-setClock;
        *(motor180z+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180z+10)=(0x01<<GPIO);
        udelay(clrClock);
    }
    delay = jiffies + 3*HZ;
    while(time_before(jiffies, delay)){}
    for(x=30;x>=0;x--){
        setClock=(minAngle+((maxAngle-minAngle)/180*x));
        clrClock=frequency-setClock;
        *(motor180z+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180z+10)=(0x01<<GPIO);
        udelay(clrClock);
    }
}
```

```
else if(tmp_buf==6){
    printk("[motz] Active 6\n");
    for(x=0;x<=180;x++){
        setClock=(minAngle+((maxAngle-minAngle)/180*x));
        clrClock=frequency-setClock;
        *(motor180z+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180z+10)=(0x01<<GPIO);
        udelay(clrClock);
    }

    delay = jiffies + 3*HZ;
    while(time_before(jiffies, delay)){}
    for(x=180;x>=0;x--){
        setClock=(minAngle+((maxAngle-minAngle)/180*x));
        clrClock=frequency-setClock;
        *(motor180z+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180z+10)=(0x01<<GPIO);
        udelay(clrClock);
    }

}
```

위의 코드는 write 함수에서 모터로 보낼 펄스 신호의 길이를 제어하는 부분이다. 모터의 동작을 코드로 구현한 원리는 3.1.2에서 설명한 것과 동일하다. MotorZ는 센서가 색을 감지한 후에 넘겨주는 값인 tmp_buf에 따라 다르게 모터가 동작하도록 구현하였다. tmp_buf가 1일 경우 30도, 2일 경우 60도가 움직이는 방식으로 tmp_buf가 1에서 6일 경우까지 각각 30도씩 차이가 나게 움직이도록 설정하였다.

3.4. Color Sensor

3.4.1. Device 정의

Color Sensor는 사물의 색상을 감지하고 그에 따라 주파수를 발생시키는데 이를 인터럽트가 발생하였다는 신호를 보냄으로써 전달한다.

3.4.2. Device driver

```
#define s2 17
#define s3 18
#define out 27

#define GPIO_SIZE 256
#define SENSOR_MAJOR 241
#define SENSOR_NAME "Color_Sensor"
```

Device driver의 Major number는 223으로 설정하였고, GPIO pin은 17, 18, 27번을 사용한다. 각각 센서의 S2, S3, out 부분에 연결된다.

```
static irqreturn_t ind_interrupt_handler(int irq, void *pdata)
{
    value = 1;
    wake_up_interruptible(&waitqueue);
    ++event_flag;
    return IRQ_HANDLED;
}
static unsigned sensor_poll(struct file *inode, struct poll_table_struct *pt)
{
    int mask = 0;
    poll_wait(inode, &waitqueue, pt);
    if (event_flag > 0)
        mask |= (POLLIN | POLLRDNORM);
    event_flag = 0;
    return mask;
}
```

I/O의 비주기적인 특성과 I/O 장치와 프로세서 간의 속도 불일치로 인해 장치는 특정 하드웨어 신호를 비동기식으로 선언하여 프로세서에 신호를 보낸다. 이러한 하드웨어 신호를 인터럽트라고 한다. ind_interrupt_handler 함수는 인터럽트가 발생했을 때 호출되는 함수이다. wake_up_interruptible 함수는 waitqueue에 등록된 프로세스를 모두 차례대로 다음 스케줄러에서 수행될 대상으로 만드는 역할을 한다. 그리고 전역변수로 선언된 event _flag를 1 증가시키는데 이는 poll 함수가 호출 이전에 인터럽트가 발생하였는가를 판별하는 데 사용한다. poll 함수로 정의된 sensor_poll 함수는 호출되는 즉시 프로세스를 sleep 시키는 역할을 한다. 함수가 호출되면 poll_wait 함수를 통해 대기 큐를 poll_table에 등록한다. poll_table은 감시해야 하는 사건에 대한 대기 큐를 저장하는 역할을 한다. 이를 통해 주기적으로 감지해야 할 사건을 확인할 수 있다. 만약 poll 함수가 프로그램에서 호출될 때 ind_interrput_handler 함수가 이전에 호출되어 event_flag가 0보다 큰 상태일 경우, mask에 POLLIN(읽을 데이터가 존재 함, 해당 비트값은 0x001)과 POLLRDNORM(읽을 일반 자료가 존재 함, 해당 비트값은 0x040)을 or 연산하여 비트를 쓴다. 해당 mask를 반환하면 처리 가능한 데이터가 있음을 알려줄 수 있다. 다시 말해, 인터럽트가 발생하였다는 것을 poll 함수를 호출한 지점에 알려준다. 그리고 event_flag를 0으로 초기화 한다.

```
static int sensor_open(struct inode *minode, struct file *mfile)
{
    if (sensor_usage != 0)
        return -EBUSY;
    sensor_usage = 1;
    sensor_map = ioremap(GPIO_BASE, GPIO_SIZE);
    if (!sensor_map)
    {
        printk("error: mapping gpio memory");
        iounmap(sensor_map);
        return -EBUSY;
    }

    color_sensor = (volatile unsigned int *)sensor_map;

    *(color_sensor + (out/10)) &= ~(0x7 << (3 * (out%10)));
    *(color_sensor + (out/10)) |= (0x0 << (3 * (out%10)));  // (sensor out pin) Input mode [read]
    *(color_sensor + 22) |= (0x1 << out);

    *(color_sensor + (s3/10)) &= ~(0x7 << (3 * (s3%10)));
    *(color_sensor + (s3/10)) |= (0x1 << (3 * (s3%10))); // (sensor s3 pin)  output mode [write]

    *(color_sensor + (s2/10)) &= ~(0x7 << (3 * (s2%10)));
    *(color_sensor + (s2/10)) |= (0x1 << (3 * (s2%10))); // (sensor s2 pin) output mode [write]
    request_irq(gpio_to_irq(27), ind_interrupt_handler, IRQF_TRIGGER_FALLING, "irq_key", NULL);

    return 0;
}
```

open 함수에서 GPSEL2 필드의 27번 pin에 해당하는 위치를 000으로 초기화하여 Input mode로 설정한다. 그리고 GPFEN1 필드에서 27번 비트를 1로 set하여 GPIO 27번 핀이 Falling edge를 감지하는 기능을 하도록 설정한다. 그리고 request_irq 함수를 사용하여 27번 핀이 falling edge를 감지하면 ind_interrupt_handler 함수를 호출하도록 인터럽트 서비스 함수를 등록한다. 그리고 GPSEL1 필드의 17번, 18번 GPIO pin에 해당하는 위치에 001을 mapping하여 17번, 18번 GPIO pin을 Output mode로 설정하였다.

```
static int sensor_read(struct file *inode, char *gdata, size_t length, loff_t *off_what) {
    int result;
    //printk("sensor read\n");
    result = copy_to_user(gdata, &value, length);
    if (result < 0) {
        printk("error : copy_to_user()");
        return result;
    }
    return length;
}
```

read 함수는 커널 영역의 데이터를 User 영역으로 전달하고자 할 때 사용한다.

copy_to_user 함수를 사용하여 커널 영역의 데이터를 User 영역으로 데이터를 복사한다.

```
static int sensor_write(struct file* mfile, const char* gdata, size_t length, loff_t* off_what) {
    struct timeval start;
    struct timeval end;
    int Freq_UpDw = 1;
    int result;
    int RGB_buf;
    char tmp_buf;

    unsigned int red;
    unsigned int blue;
    unsigned int green;

    result = copy_from_user(&tmp_buf, gdata, length);
    if (result < 0) {
        printk("Error : copy from user");
        return result;
    }

    if (tmp_buf == 1) {
        *(color_sensor + 10) |= (0x1 << s2); // s2 L
        *(color_sensor + 10) |= (0x1 << s3); // s3 L mode red
    }
    else if (tmp_buf == 2) {
//        *(color_sensor + 10) |= (0x1 << s2); // s2 L
        *(color_sensor + 7) |= (0x1 << s3); // s3 H mode blue
    }
    else if (tmp_buf == 3) {
        *(color_sensor + 7) |= (0x1 << s2); // s2 H
//        *(color_sensor + 7) |= (0x1 << s3); // s3 H mode green
    }

    return length;
}
```

write함수에서는 S2, S3에 연결되는 17번 19번 GPIO핀의 출력을 제어하는 역할을 한다. 전달 받은 값인 tmp_buf에 따라 출력하는 상태가 변하게 되며, 1일 때 각각 0, 0 상태, 2일 때 1, 0 상태, 3일 때 1, 1 상태이다. 2.3.2에서 서술한 바와 같이 센서의 S2, S3 port로 어떤 level의 신호를 보내느냐에 따라 검출에 사용할 색상 필터를 바꿔 줄 수 있다.

3.5. Dot Matrix

3.5.1. Device 정의

8*8 도트 매트릭스이다. 색상 탐지가 끝나면 해당 색상에 맞는 알파벳과 해당 색상이 검출된 횟수를 출력한다. 본 프로젝트에서는 col열 3개 row열 5개를 매핑하여 총 8개의 핀을 사용하였다.

3.5.2. Device driver

```
#define MATRIX_MAJOR    221
#define MATRIX_NAME     "MATRIX_DRIVER"
#define GPIO_SIZE       256


static int matrix_open(struct inode *minode, struct file *mfile)
{
    unsigned char index;

    if (matrix_usage != 0)
        return -EBUSY;
    matrix_usage = 1;



    matrix_map = ioremap(GPIO_BASE, GPIO_SIZE);
    if (!matrix_map)
    {
        printk("error: mapping gpio memory");
        iounmap(matrix_map);
        return -EBUSY;
    }
    matrix = (volatile unsigned int *)matrix_map;
    for (index = 2; index <= 9; ++index)
    {
        *(matrix) &= ~(0x07 << (3 * index));
        *(matrix) |= (0x01 << (3 * index));
    }
    return 0;
}
```

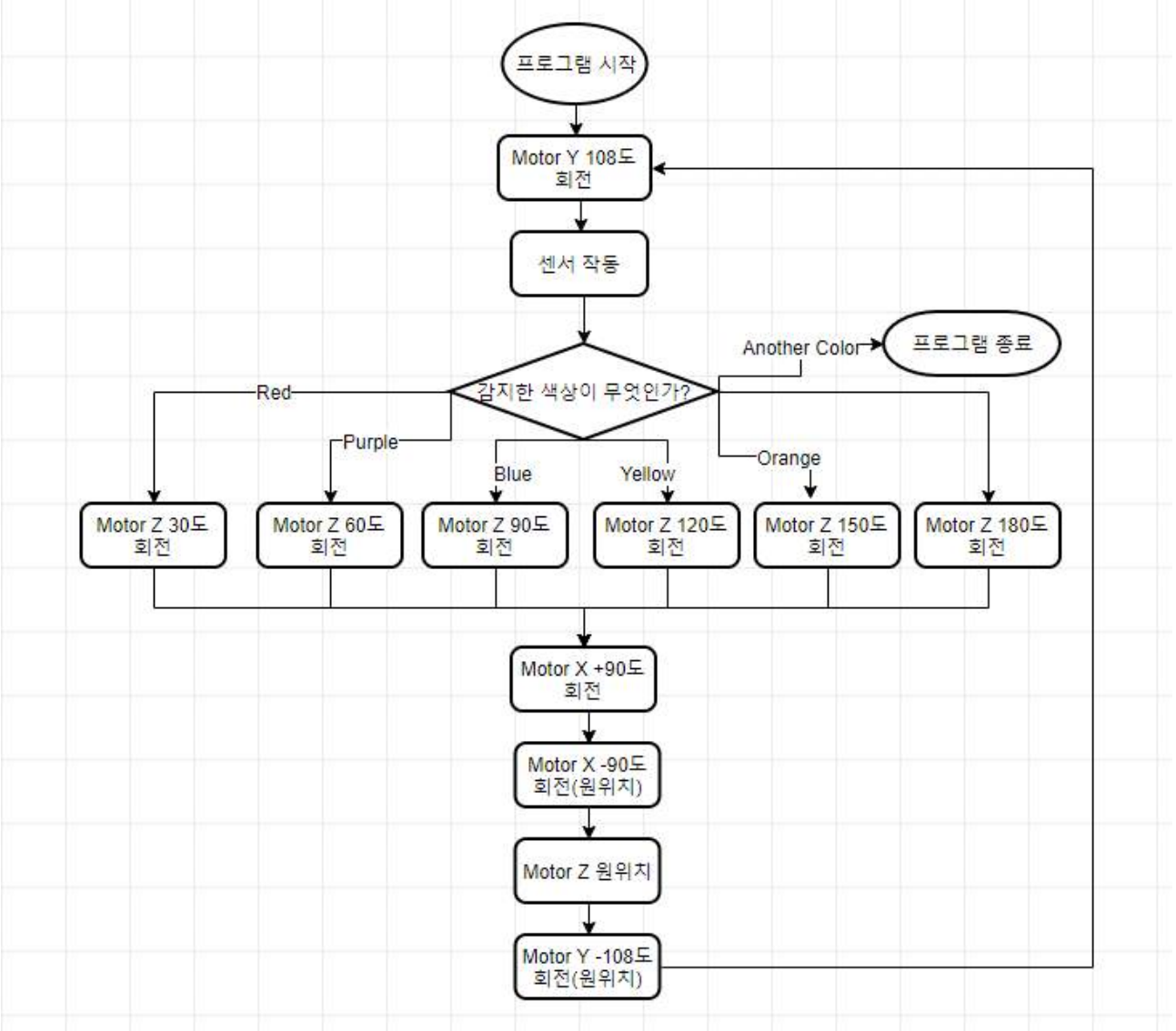
open 함수에서 반복문을 이용하여 GPSEL0 필드의 2번부터 9번까지, 총 8개의 pin에 해당하는 위치에 001을 mapping하여 2번~9번 GPIO pin을 Output mode로 설정하였다.

```
static int matrix_write(struct file *mfile, const char *gdata, size_t length, loff_t *off_what) {
    char tmp_buf[5];
    char val[5];
    int result, i;
    unsigned long delay = jiffies + 1*HZ;

    result = copy_from_user(&tmp_buf, gdata, length);
    if (result < 0) {
        printk("Error: copy from user");
        return result;
    }

    for (i = 0; i < 5; i++)
        val[i] = gdata[i];
    printk("matrix values = %s\n", tmp_buf);

    while(time_before(jiffies, delay)) {
        for (i = 0; i < 5; i++) {
            *(matrix + 7) = (val[i] << 2);
            *(matrix + 7) = (0x1 << (i + 5));
            udelay(1);
            *(matrix + 10) = (0xFF << 2);
        }
    }
    return length;
}
```

write 함수에서는 유저 영역에서 0x0~0x7 사이의 값(한 열에 대한 3개의 행 출력)을 5개(총 5개의 열에 led를 출력) 받아 왔다. 5열에 대한 led를 표시하기 때문에, 반복문을 5회 발생시킨다. GPIO 2, 3, 4번 핀을 col 3, 2, 1번 순으로 mapping하였기 때문에 shift를 2회 하여 vai[i](0x0~0x7 사이의 값) 을 넣음으로써 2, 3, 4번 핀에 한 번에 할당하였다. 해당하는 제어를 각각의 열에 표시하기 위하여 GPIO 5, 6, 7, 8, 9번 핀을 row 1, 2, 3, 4, 5로 mapping하였다. 처음에는 첫 번째 열에 col에 할당한 값을 표시해야 하므로 5번 핀에 High값을 할당한다. 이후 짧은 딜레이를 가진 후, shift를 2회 한 0xFF값(GPIO 2번~9번 / 총 8핀)을 이용하여 clear해주었다. 이후 col에 값이 할당되고 그 값을 두 번째 열에 표시하여야 하므로 6번 핀에 High값 할당 ~ 이 과정을 time_before()함수에 의하여 1초 동안 반복하게 된다.

4. Main User program

4.1. 플로우 차트



프로그램을 시작하면 Motor Y를 108도 회전시킨다. Motor Y는 삽입구로 투입된 물건을 센서가 색상을 측정할 수 있는 위치로 이동시킨다. Motor Y의 회전이 끝나고 나면, 센서는 물건의 색상을 판별한다. 만약 판별한 색상이 정해진 색상 중 포함되는 것이 없을 경우, 프로그램을 종료한다. 정해진 색상 중 포함되는 것이 있을 경우 Motor Z가 물건을 떨어뜨릴 경로를 선택하기 위하여 색상별로 정해진 각도만큼 회전한다. 그리고 Motor X를 회전시켜 판을 제거하여 물건을 떨어뜨리면 물건이 Motor Z에 의해 설정된 경로로 떨어지게 된다. 물건이 떨어지고 나면 다시 Motor X와 Motor Z, Motor Y를 모두 원위치 상태로 돌린다. 이후 프로그램이 종료될 때까지 동작을 반복한다.

4.2. 코드 구현

4.2.1. thread_motor180X

```
void *thread_motor180x(void *arg){

    int fdX;
    char data;

    fdX=open(MOTOR180X_FILE_NAME, O_RDWR);
    if(fdX<0){
        fprintf(stderr, "Can't open %s\n", MOTOR180X_FILE_NAME);
        return -1;
    }

    data = *(char *)arg;
    write(fdX, &data, sizeof(char));

    close(fdX);
    return 0;
}
```

위 코드는 메인 User program 인 test_test.c 중 MotorX를 동작시키기 위한 스레드 함수이다. 작성된 Device Driver 커널 실행 파일을 open하여 파일 디스크립터를 할당한다. 스레드가 생성될 때 main 함수로부터 넘겨받은 arg 값을 data 변수에 복사하고, write 함수를 호출하여 Device driver code중 write에 작성된 코드를 수행하게 한다.

4.2.2. thread_motor180Y

```
void *thread_motor180y(void *arg) {

    int fdY;
    char data;

    fdY=open(MOTOR180y_FILE_NAME, O_RDWR);
    if(fdY<0){
        fprintf(stderr, "Can't open %s\n", MOTOR180y_FILE_NAME);
        return -1;
    }

    data = *(char *)arg;
    write(fdY, &data, sizeof(char));

    sleep(2);
    close(fdY);
    return 0;
}
```

위 코드는 메인 User program 인 test_test.c 중 MotorY를 동작시키기 위한 스레드 함수이다. 4.1.1과 마찬가지로 작성된 Device Driver 커널 실행 파일을 open하여 파일 디스크립

터를 할당한다. 그리고 write 함수를 호출하여 Device driver code에 작성된 동작을 수행하도록 한다. write가 수행되고 난 후, 2초간 sleep 한다.

4.2.3. thread_motor180Z

```
void *thread_motor180z(void *arg) {

    int fdZ;
    char data;

    fdZ=open(MOTOR180Z_FILE_NAME, O_RDWR);
    if(fdZ<0){
        fprintf(stderr, "Can't open %s\n", MOTOR180z_FILE_NAME);
        return -1;
    }
    data = *(char *)arg;
    write(fdZ, &data, sizeof(char));

    close(fdZ);
    return 0;
}
```

위 코드는 메인 User program 인 test_test.c 중 MotorZ를 동작시키기 위한 스레드 함수이다. 4.1.1과 마찬가지로 작성된 Device Driver 커널 실행 파일을 open하여 파일 디스크립터를 할당한다. 그리고 write 함수를 호출하여 Device driver code에 작성된 동작을 수행하도록 한다.

4.2.4. thread_sensor

```
void *thread_sensor(void *arg) {

    struct timeval start_time;
    struct timeval end_time;


    int         sensor_fd[3];
    char        x;
    char        flag;
    struct pollfd events[1];
    int         retval;
    int         i = 0;
    int j = 0;


    double durationA = 0;
    double durationB = 0;
    double durationC = 0;


    double      r_value=0;
    double      b_value=0;
    double      g_value=0;
```

위 코드는 메인 User program 인 test_test.c 중 센서를 동작시키기 위한 스레드 함수 중 상단의 변수 선언 부분이다. timeval 구조체는 현재 시스템 시간을 저장하기 위한 구조체로, 초 단위의 값을 가지고 있는 tv_sec(long), 마이크로 초 단위의 값을 가지고 있는 tv_use(long)를 멤버로 가지고 있다. pollfd events 구조체는 파일 디스크립터인 fd(int), 요구된 이벤트를 나타내는 events(short), 반환된 이벤트를 나타내는 revents(short)를 멤버로 가지고 있다. events에서는 process에게 fd(파일 디스크립터)가 어떤 이벤트를 감시하고자 하는지를 커널에 전달하는 역할을 하고, revents는 poll함수가 반환될 때 커널로부터 각 fd에 발생한 이벤트에 해당하는 값을 전달받는 역할을 한다. duration A, B, C는 각각 센서가 Red, Blue, Green 필터 상태일 때 센서가 출력하는 주파수의 Falling edge가 10번 감지되는 시간을 측정하여 저장할 변수이다. 그리고 r_value, b_value, g_value는 센서가 Red, Blue, Green 필터 상태일 때 각각 계산된 주파수 값을 저장하기 위한 변수이다.

```
while(j < 10){
    for(x=1;x<=3;x++) {
        sensor_fd[x-1] = open(SENSOR_FILE_NAME, O_RDWR | O_NONBLOCK);
        if (sensor_fd[x-1] < 0)
        {
            fprintf(stderr, "Can't open %s\n", SENSOR_FILE_NAME);
            return -1;
        }

        puts("program start\n");
        write(sensor_fd[x-1], &x, 1);
        gettimeofday(&start_time,NULL);

        while (i < 10) {
            events[0].fd = sensor_fd[x-1];
            events[0].events = POLLIN;    // waiting read

            retval = poll(events, 1, 100);        // event waiting
            if (retval < 0) {
                fprintf(stderr, "Poll error\n");
                exit(0);
            }

            if (events[0].revents & POLLIN) {
                read(sensor_fd[x-1], &flag, 1);
                i++;
            }
        }
        if(x == 1){
            gettimeofday(&end_time,NULL);
            durationA = (double)(end_time.tv_usec) - (double)(start_time.tv_usec);
            r_value += 10 / (durationA/1000000);

        }

        if(x == 2){
            gettimeofday(&end_time,NULL);

            durationB = (double)(end_time.tv_usec) - (double)(start_time.tv_usec);
            b_value += 10 / (durationB/1000000);

        }

        if(x == 3){
            gettimeofday(&end_time,NULL);

            durationC = (double)(end_time.tv_usec) - (double)(start_time.tv_usec);
            g_value += 10 / (durationC/1000000);
        }
        i = 0;

        if(j != 9){
            close(sensor_fd[x-1]);
            printf("Close activate\n");
        }
        else{

            r_value=(r_value/10);
            b_value=(b_value/10);
            g_value=(g_value/10);
```

위 코드는 메인 User program 인 test_test.c 중 센서를 동작시키기 위한 스레드 함수에

서 실제 색을 구별하기 위한 부분이다. x는 감지할 현재 감지할 색상을 나타내는 파라미터이다 1일 때 Red, 2일 때 Blue, 3일 때 Green을 감지한다. 반복문 내에서 Device Driver 커널 실행 파일을 open하여 파일 디스크립터를 할당한다. write함수를 호출하여 센서의 색상 필터를 설정한다. 그리고 주파수를 계산하기 위하여 인터럽트를 통해 falling edge를 10번 감지한다. poll.events에 감지할 이벤트를 POLLIN(0x001)으로 설정해주면 인터럽트가 발생하였을 때 revents에 POLLIN(0x001)이 쓰여 진다. 만약 인터럽트가 발생하였으면, read를 함수를 한번 호출하고 파라미터를 증가시킨다. read 함수를 호출하는 이유는 원래 값을 전달받기 위함이지만, 여기서는 그저 정상적으로 falling edge가 감지되어 인터럽트가 발생하였다는 것을 알 수 있도록 임의로 사용하였다. 10번의 edge 감지가 끝나면 걸린 시간을 감지 횟수(10)으로 나눠서 주파수를 구한다. 그리고 이것을 각 색상 필터 별로 10번 반복한다. 마지막으로 반복한 10을 나눠주면 10번 동안 계산한 3개의 색상의 주파수 평균값을 계산할 수 있다.

```
if(9000 < r_value && 14000 > r_value && 5000 < b_value && 9000 > b_value && 4000 < g_value && 8000 > g_value){
    printf("Red\n");
    d_Main=1;
}
else if(7500 < r_value && 12500 > r_value && 7500 < b_value && 14000 > b_value && 5500 < g_value && 9500 > g_value){
    printf("Purple\n");
    d_Main=2;
}
else if(4000 < r_value && 7500 > r_value && 7000 < b_value && 12500 > b_value && 4000 < g_value && 8500 > g_value){
    printf("Blue\n");
    d_Main=3;
}
else if(25000 < r_value && 35000 > r_value && 13000 < b_value && 20000 > b_value && 16000 < g_value && 23000 > g_value){
    printf("Yellow\n");
    d_Main=4;
}
else if(16000 < r_value && 22000 > r_value && 7500 < b_value && 11500 > b_value && 7000 < g_value && 11000 > g_value){
    printf("Orange\n");
    d_Main=5;
}
else if(6000 < r_value && 10000 > r_value && 9000 < b_value && 14000 > b_value && 7000 < g_value && 12000 > g_value){
    printf("Green\n");
    d_Main=6;
}
```

위 코드는 메인 User program인 test_test.c 중 측정된 주파수 값에 따라서 색상의 범위를 나누어 놓은 것이다. 이 범위는 수많은 모의 테스트 결과를 거쳐 설정한 것이다. 측정된 각각 3 주파수 값이 6개의 조건문 중 모두 포함되는 범위가 있다면 그 범위의 색상으로 올바르게 감지된 것이다. 위에서부터 빨강, 보라, 파랑, 노랑, 주황, 초록색의 순이고, d_Main이라는 감지된 색상을 지정하는 변수에 값을 저장한다.

4.2.5. thread_matrixcolor

```
void *thread_matrixColor(void *arg) {
    int fdM_Color;
    char data;

    char color[6][5] = {
            0x1, 0x2, 0x1, 0x3, 0x2, // 빨
            0x1, 0x2, 0x1, 0x3, 0x3, // 보
            0x1, 0x2, 0x1, 0x2, 0x1, // 파
            0x2, 0x2, 0x5, 0x5, 0x5, // 노
            0x0, 0x2, 0x2, 0x2, 0x0, // 주
            0x4, 0x3, 0x2, 0x2, 0x4, // 초
    };


    fdM_Color = open(MATRIX_FILE_NAME, O_RDWR);
    if(fdM_Color < 0){
        fprintf(stderr, "Can't open %s\n",MATRIX_FILE_NAME);
        return -1;
    }


    data = *(char *)arg;
    printf("fdM_Color write = %d\n", data);
    write(fdM_Color, color[data - 1], sizeof(char));

    close(fdM_Color);
    return 0;

}
```

위 코드는 메인 User program 인 test_test.c 중 도트매트릭스 동작시키기 위한 스레드 함수 중 센서에 의해 측정된 색상의 알파벳을 표시하기 위한 스레드이다. 센서 스레드에 의하여 얻은 d_Main의 값을 *arg로 받아 2차원 배열인 color의 열을 지정하여 write로 배열 5개로 구성된 3비트의 값을 넘겨 원하는 알파벳을 출력하였다.

4.2.6. thread_matrixcount

```
void *thread_matrixCount(void *arg) {
    int fdM_Count;
    char data;

    char num[9][5] = {
            0x6, 0x6, 0x6, 0x6, 0x6, //1
            0x0, 0x6, 0x0, 0x3, 0x0, //2
            0x0, 0x6, 0x0, 0x6, 0x0, //3
            0x2, 0x2, 0x0, 0x6, 0x6, //4
            0x0, 0x3, 0x0, 0x6, 0x0, //5
            0x0, 0x3, 0x0, 0x2, 0x0, //6
            0x0, 0x6, 0x6, 0x6, 0x6, //7
            0x0, 0x2, 0x0, 0x2, 0x0, //8
            0x0, 0x2, 0x0, 0x6, 0x0, //9
    };


    fdM_Count = open(MATRIX_FILE_NAME, O_RDWR);
    if(fdM_Count < 0){
        fprintf(stderr, "Can't open %s\n",MATRIX_FILE_NAME);
        return -1;
    }


    data = *(char *)arg;
    printf("fdM_Count write = %d\n", data);
    write(fdM_Count, num[data - 1], sizeof(char));

    close(fdM_Count);
    return 0;

}
```

위 코드는 메인 User program 인 test_test.c 중 도트매트릭스 동작시키기 위한 스레드 함수 중 센서에 의해 측정된 색상의 누적 값을 표시하기 위한 스레드이다. 카지노 칩의 개수는 색상 당 5개를 보유하였기 때문에 10의 자리의 숫자는 존재하지 않는 2차원 배열을 생성하였고, 센서 스레드에 의하여 얻은 d_Main의 값을 *arg로 받아 2차원 배열인 num의 열을 지정하여 write로 배열 5개로 구성된 3비트의 값을 넘겨 원하는 숫자를 출력하였다.

4.2.7. main

```
int d_Main;
int count[6] = {0, };


int main(int argc, char **argv)
{
    int fd, data, motx, moty, motz, sensor, matColor, matCount;
    int check=1;
    int Xangle, Yangle;

    pthread_t motor180x_id;
    pthread_t motor180y_id;
    pthread_t motor180z_id;
    pthread_t sensor_id;
    pthread_t matrixColor_id;
    pthread_t matrixCount_id;
    void *t_return;
```

위 코드는 메인 User program인 test_test.c 중 main 함수의 가장 상단 부분이며 변수 선언 부분이다. 전역변수로 선언된 d_Main은 센서가 색을 판별하고 그에 해당되는 정수를 저장한다. 이 값에 따라 Dot matrix와 모터가 수행해야 할 동작이 달라진다. count배열은 각 색상별로 검출된 횟수를 나타낸다. 이 배열 내의 값에 따라서 Dot-Matrix에 출력되는 숫자가 달라진다. check 변수는 프로그램을 반복하거나 종료시키는 플래그이다. 그리고 그 외 필요한 변수들과 생성할 스레드들이 미리 선언되어 있다.

```
while(check) {


    sleep(1);
    printf("Sensor start..\n");
    Yangle = 1;
    moty=pthread_create(&motor180y_id, NULL, thread_motor180y, (void *)&Yangle);
    if(moty < 0){
        printf("setting motor degree create error");
        exit(1);
    }
    moty=pthread_join(motor180y_id, &t_return);
    sleep(1);
```

위 코드는 메인 User program인 test_test.c에서 반복문이 시작되는 부분이며, MotorY를 동작시키기 위해 스레드를 생성하는 부분이다. check 변수가 1일 때 반복문을 계속 수행한다. 1을 저장하고 있는 Yangle은 Motor Y의 Device Driver의 write 함수에 1을 전달하기 위한 변수이다. 이렇게 전달한 값은 Device driver 코드에 정의된 동작 중 원하는 동작을 실행하는 데 사용한다. 이 스레드는 join을 사용하여 스레드가 종료될 때까지 main 함수의

진행을 막는다.

```
sleep(1);
sensor=pthread_create(&sensor_id, NULL, thread_sensor, NULL);
if(sensor < 0){
    printf("sensor create error");
    exit(1);
}
sensor=pthread_join(sensor_id, &t_return);
sleep(1);
```

위 코드는 메인 User program인 test_test.c에서 센서 스레드를 실행하는 부분이다. 이 스레드는 join을 사용하여 센서가 색 판별을 모두 끝낼 때까지, 즉 스레드가 종료될 때까지 main함수의 진행을 막는다.

```
switch (d_Main) {
    case 1 :
        matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
        if(matColor < 0){
            printf("Matrix color create error");
            exit(1);
        }
        matColor=pthread_join(matrixColor_id, &t_return);
        count[d_Main-1]++;

        matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
        if(matColor < 0){
            printf("Matrix count create error");
            exit(1);
        }
        matCount=pthread_join(matrixCount_id, &t_return);
        break;
```

(생략)

```
     default:
         check = 0;
         close(fd);
         return 0;
}
```

위 코드는 메인 User program인 test_test.c에서 센서 스레드가 저장한 d_Main의 값에 따라 Dot-Matrix를 다르게 표시하기 위해서 switch문을 사용하였다. 각 색상별로 지정된 d_Main의 값에 따라 thread_matrixColor에 정의된 문자를 출력한다. 그리고 count 배열에서 색상별로 위치한 값을 1씩 증가시켜 검출된 색상을 카운트 한다. 그리고 배열 내 정수 값을 thread_matrixCount에 전달하여 해당 숫자를 Dot-Matrix에 출력한다. 각각의 스레드는 join을 사용하여 스레드가 끝날 때까지 main 함수의 진행을 막는다. 만약 d_Main이 case 이외의 값일 경우 default에 선언된 코드를 실행한다. 해당 부분에서 check를 0으로

만들어 프로그램을 종료한다.

```
motz=pthread_create(&motor180z_id, NULL, thread_motor180z, (void *)&d_Main);
if(motz < 0){
    printf("motorz degree create error");
    exit(1);
}

motx=pthread_create(&motor180x_id, NULL, thread_motor180x,  (void *)&d_Main);
if(motx < 0){
    printf("motorx degree create error");
    exit(1);
}

motz=pthread_join(motor180z_id, &t_return);
motx=pthread_join(motor180x_id, &t_return);

Yangle = 0;
moty=pthread_create(&motor180y_id, NULL, thread_motor180y, (void *)&Yangle);
if(moty < 0){
    printf("motory degree create error");
    exit(1);
}
moty=pthread_join(motor180y_id, &t_return);
d_Main = 0;
sleep(1);
```

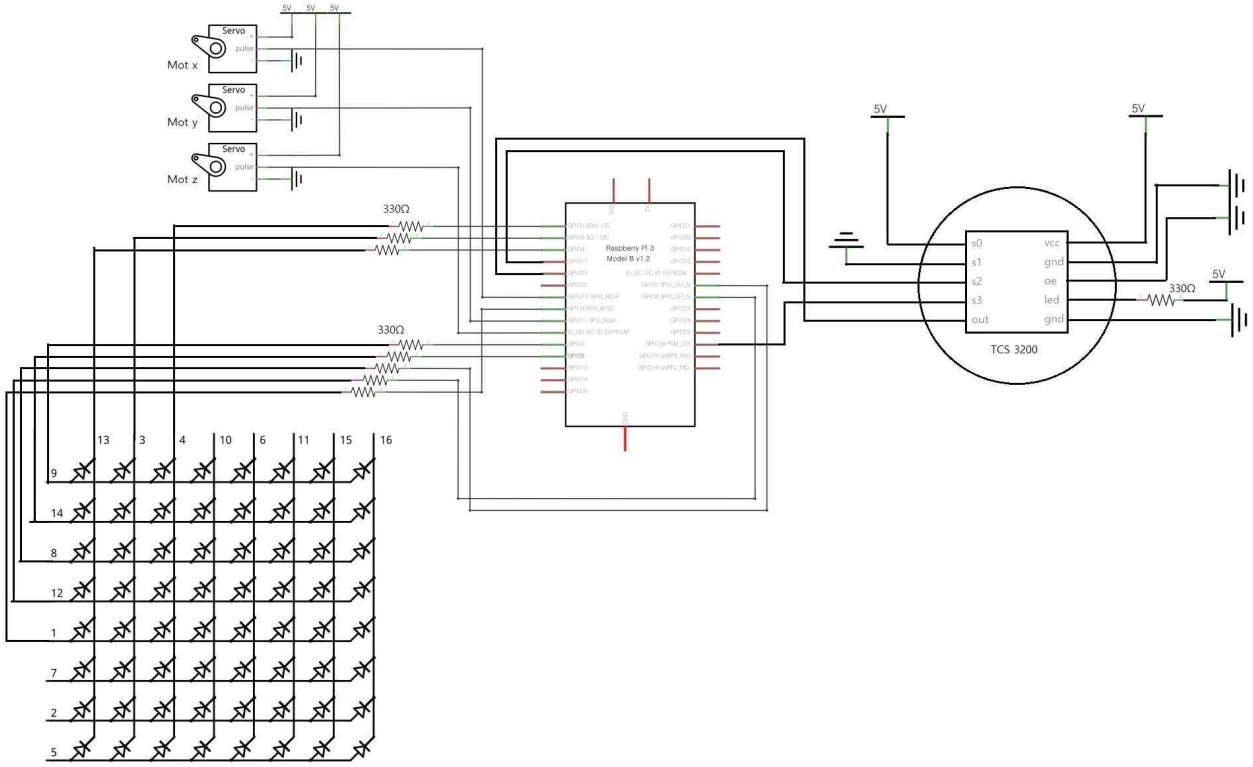
위 코드는 메인 User program인 test_test.c에서 MotorZ와 MotorX를 동작시키기 위해 스레드를 실행하는 부분이다. MotorZ가 먼저 색상에 따라 경로를 설정하고 MotorX가 물건 아래의 판을 제거한다. 물건을 떨어뜨린 후 각각 원위치로 돌아오는 동작을 하게 된다. 이 두 스레드는 동시에 실행되며 join을 사용해 두 스레드의 동작이 끝날 때까지 기다리게 만든다. 스레드가 종료되면 d_main을 초기화한다.

5. 결　　론

약 1달 반 정도의 기간 동안 이번 임베디드 소프트웨어 과목의 프로젝트를 진행하였다. 180도 서보모터의 동작을 완전히 제어하였고, Dot-Matrix에 원하는 문자를 출력하였으며, 인터럽트를 응용해 센서로 색을 감지하였다. 그리고 스레드를 사용하여 각각의 동작이 잘 수행될 수 있게끔 프로그래밍하였다. 비록 motor가 고장이나 정상적인 데모를 수행하지 못해 아쉬움이 남지만, 프로젝트의 진행에 있어서 성공적으로 목표를 완수하였다. 몇 개의 변경사항을 제외하고 프로젝트의 핵심적인 주제인 Device driver를 프로그래밍하여 실제로 구동되는 결과물을 만드는 데 성공하였다. Device driver에서 통해 원하는 동작을 구현하기 위해서는 부품의 동작 원리를 정확히 이해하고 응용하는 능력이 필요했다. 그래서 단순히 리눅스 시스템뿐만 아니라 하드웨어적인 요소를 포함하여 열심히 관련 정보와 지식을 습득하였고, 강의시간에 미처 깊게 다루지 못한 부분까지 세세하게 검토하며 프로젝트를 수행하였다. 다양한 아이디어를 바로 적용하여 결과를 육안으로 바로 확인할 수 있었다는 것이 이 프로젝트에서 재미를 느꼈던 점이다. 이 프로젝트를 통해 커널 영역은 어떻게 프로그래밍 되어있는지를 잘 알게 되었고, 하드웨어적인 부분과 소프트웨어적인 부분을 동시에 다루면서 임베디드 시스템을 개발하는 능력을 좀 더 착실히 키울 수 있는 계기가 된 것 같다.

6. 부 록

1. 회로도


5V
Servo
pulse
Mot x
Mot y
Mot z
330Ω
Raspberry Pi 3
Model B v1.2
s0
s1
s2
s3
out
vcc
gnd
oe
led
TCS 3200
13
3
4
10
6
11
15
16
9
14
8
12
1
7
2
5

2. 소스코드

❍ **test_test.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 <unistd.h>
#include <sys/time.h>
#include <linux/poll.h>
#include <signal.h>


#define MOTOR180x_FILE_NAME "/dev/motx_driver"
#define MOTOR180y_FILE_NAME "/dev/moty_driver"
#define MOTOR180z_FILE_NAME "/dev/motz_driver"
#define MATRIX_FILE_NAME "/dev/mat_driver"
#define SENSOR_FILE_NAME "/dev/sensor_driver"


pthread_mutex_t mutex;

void *thread_motor180x(void *arg);
void *thread_motor180y(void *arg);
void *thread_motor180z(void *arg);
void *thread_matrixColor(void *arg);
void *thread_matrixCount(void *arg);
void *thread_sensor(void *arg);



int d_Main;
int count[6] = {0, };


int main(int argc, char **argv)
{
    int fd, data, motx, moty, motz, sensor, matColor, matCount;
    int check=1;
    int Xangle, Yangle;

    pthread_t motor180x_id;
    pthread_t motor180y_id;
    pthread_t motor180z_id;
    pthread_t sensor_id;
    pthread_t matrixColor_id;
    pthread_t matrixCount_id;
    void *t_return;
```

```
pthread_mutex_init(&mutex, NULL);


while(check) {


    sleep(1);
    printf("Sensor start..₩n");
    Yangle = 1;
    moty=pthread_create(&motor180y_id, NULL, thread_motor180y, (void *)&Yangle);
    if(moty < 0){
        printf("setting motor degree create error");
        exit(1);
    }
    moty=pthread_join(motor180y_id, &t_return);
    sleep(1);



    sleep(1);
    sensor=pthread_create(&sensor_id, NULL, thread_sensor, NULL);
    if(sensor < 0){
        printf("sensor create error");
        exit(1);
    }
    sensor=pthread_join(sensor_id, &t_return);
    sleep(1);

    sleep(1);

    //color sensor code...
    switch (d_Main) {
        case 1 :
            matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
            if(matColor < 0){
                printf("Matrix color create error");
                exit(1);
            }
            matColor=pthread_join(matrixColor_id, &t_return);
            count[d_Main-1]++;

            matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
            if(matColor < 0){
                printf("Matrix count create error");
                exit(1);
            }
            matCount=pthread_join(matrixCount_id, &t_return);
            break;
        case 2 :
            matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
```

```
        if(matColor < 0){
            printf("Matrix color create error");
            exit(1);
        }
        matColor=pthread_join(matrixColor_id, &t_return);
        count[d_Main-1]++;

        matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
        if(matColor < 0){
            printf("Matrix count create error");
            exit(1);
        }
        matCount=pthread_join(matrixCount_id, &t_return);
        break;
    case 3 :
        matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
        if(matColor < 0){
            printf("Matrix color create error");
            exit(1);
        }
        matColor=pthread_join(matrixColor_id, &t_return);
        count[d_Main-1]++;

        matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
        if(matColor < 0){
            printf("Matrix count create error");
            exit(1);
        }
        matCount=pthread_join(matrixCount_id, &t_return);
        break;
    case 4 :
        matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
        if(matColor < 0){
            printf("Matrix color create error");
            exit(1);
        }
        matColor=pthread_join(matrixColor_id, &t_return);
        count[d_Main-1]++;

        matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
        if(matColor < 0){
            printf("Matrix count create error");
            exit(1);
        }
        matCount=pthread_join(matrixCount_id, &t_return);
        break;
    case 5 :
        matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
        if(matColor < 0){
            printf("Matrix color create error");
            exit(1);
        }
```

```
            matColor=pthread_join(matrixColor_id, &t_return);
            count[d_Main-1]++;

            matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
            if(matColor < 0){
                printf("Matrix count create error");
                exit(1);
            }
            matCount=pthread_join(matrixCount_id, &t_return);
            break;
        case 6 :
            matColor=pthread_create(&matrixColor_id, NULL, thread_matrixColor, (void *)&d_Main);
            if(matColor < 0){
                printf("Matrix color create error");
                exit(1);
            }
            matColor=pthread_join(matrixColor_id, &t_return);
            count[d_Main-1]++;

            matCount=pthread_create(&matrixCount_id, NULL, thread_matrixCount, (void *)&count[d_Main-1]);
            if(matColor < 0){
                printf("Matrix count create error");
                exit(1);
            }
            matCount=pthread_join(matrixCount_id, &t_return);
            break;
        default:
            check = 0;
            close(fd);
            return 0;
    }

    sleep(1);

    motz=pthread_create(&motor180z_id, NULL, thread_motor180z, (void *)&d_Main);
    if(motz < 0){
        printf("motorz degree create error");
        exit(1);
    }

    motx=pthread_create(&motor180x_id, NULL, thread_motor180x,  (void *)&d_Main);
    if(motx < 0){
        printf("motorx degree create error");
        exit(1);
    }

    motz=pthread_join(motor180z_id, &t_return);
    motx=pthread_join(motor180x_id, &t_return);

    Yangle = 0;
    moty=pthread_create(&motor180y_id, NULL, thread_motor180y, (void *)&Yangle);
    if(moty < 0){
```

```
                printf("motory degree create error");
                exit(1);
            }
            moty=pthread_join(motor180y_id, &t_return);
            d_Main = 0;
            sleep(1);

        }

        return 0;
}


void *thread_motor180x(void *arg){

        int fdX;
        char data;

        fdX=open(MOTOR180x_FILE_NAME, O_RDWR);
        if(fdX<0){
            fprintf(stderr, "Can't open %s₩n", MOTOR180x_FILE_NAME);
            return -1;
        }

        data = *(char *)arg;
        write(fdX, &data, sizeof(char));

        close(fdX);
        return 0;
}


void *thread_motor180y(void *arg) {

        int fdY;
        char data;

        fdY=open(MOTOR180y_FILE_NAME, O_RDWR);
        if(fdY<0){
            fprintf(stderr, "Can't open %s₩n", MOTOR180y_FILE_NAME);
            return -1;
        }

        data = *(char *)arg;
        write(fdY, &data, sizeof(char));

        sleep(2);
        close(fdY);
        return 0;
}

void *thread_motor180z(void *arg) {
```

```
    int fdZ;
    char data;

    fdZ=open(MOTOR180z_FILE_NAME, O_RDWR);
    if(fdZ<0){
        fprintf(stderr, "Can't open %s₩n", MOTOR180z_FILE_NAME);
        return -1;
    }
    data = *(char *)arg;
    write(fdZ, &data, sizeof(char));

    close(fdZ);
    return 0;
}


void *thread_sensor(void *arg) {

    struct timeval start_time;
    struct timeval end_time;


    int                 sensor_fd[3];
    char    x;
    char    flag;
    struct pollfd events[1];
    int                 retval;
    int     i = 0;
    int j = 0;


    double durationA = 0;
    double durationB = 0;
    double durationC = 0;


    double   r_value=0;
    double   b_value=0;
    double   g_value=0;



    while(j < 10){
        for(x=1;x<=3;x++) {
            sensor_fd[x-1] = open(SENSOR_FILE_NAME, O_RDWR | O_NONBLOCK);
            if (sensor_fd[x-1] < 0)
            {
                fprintf(stderr, "Can't open %s₩n", SENSOR_FILE_NAME);
                return -1;
            }
```

```
puts("program start₩n");
write(sensor_fd[x-1], &x, 1);
gettimeofday(&start_time,NULL);

while (i < 10) {
	events[0].fd = sensor_fd[x-1];
	events[0].events = POLLIN;	// waiting read

	retval = poll(events, 1, 100);		// event waiting
	if (retval < 0) {
		fprintf(stderr, "Poll error₩n");
		exit(0);
	}

	if (events[0].revents & POLLIN) {
		read(sensor_fd[x-1], &flag, 1);
		i++;
	}
}
if(x == 1){
	gettimeofday(&end_time,NULL);
	durationA = (double)(end_time.tv_usec) - (double)(start_time.tv_usec);
	r_value += 10 / (durationA/1000000);

}

if(x == 2){
	gettimeofday(&end_time,NULL);

	durationB = (double)(end_time.tv_usec) - (double)(start_time.tv_usec);
	b_value += 10 / (durationB/1000000);

}

if(x == 3){
	gettimeofday(&end_time,NULL);

	durationC = (double)(end_time.tv_usec) - (double)(start_time.tv_usec);
	g_value += 10 / (durationC/1000000);
}
i = 0;

if(j != 9){
	close(sensor_fd[x-1]);
	printf("Close activate₩n");
}
else{

	r_value=(r_value/10);
	b_value=(b_value/10);
	g_value=(g_value/10);
```

```
                    printf("Red..... %f₩t", r_value);
                    printf("Blue..... %f₩t", b_value);
                    printf("Green..... %f₩n", g_value);



                    sleep(1);

                    if(9000 < r_value && 14000 > r_value && 5000 < b_value && 9000 > b_value && 4000 < g_value
&& 8000 > g_value){
                        printf("Red₩n");
                        d_Main=1;
                    }
                    else if(7500 < r_value && 12500 > r_value && 7500 < b_value && 14000 > b_value && 5500 <
g_value && 9500 > g_value){
                        printf("Purple₩n");
                        d_Main=2;
                    }
                    else if(4000 < r_value && 7500 > r_value && 7000 < b_value && 12500 > b_value && 4000 <
g_value && 8500 > g_value){
                        printf("Blue₩n");
                        d_Main=3;
                    }
                    else if(25000 < r_value && 35000 > r_value && 13000 < b_value && 20000 > b_value && 16000
< g_value && 23000 > g_value){
                        printf("Yellow₩n");
                        d_Main=4;
                    }
                    else if(16000 < r_value && 22000 > r_value && 7500 < b_value && 11500 > b_value && 7000 <
g_value && 11000 > g_value){
                        printf("Orange₩n");
                        d_Main=5;
                    }
                    else if(6000 < r_value && 10000 > r_value && 9000 < b_value && 14000 > b_value && 7000 <
g_value && 12000 > g_value){
                        printf("Green₩n");
                        d_Main=6;
                    }


                    close(sensor_fd[0]);
                    close(sensor_fd[1]);
                    close(sensor_fd[2]);
                    pthread_exit(0);
                }
            }
            j++;
            sleep(0.05);


        }
    }
```

```
void *thread_matrixColor(void *arg) {
    int fdM_Color;
    char data;

    char color[6][5] = {
            0x1, 0x2, 0x1, 0x3, 0x2, // 빨
            0x1, 0x2, 0x1, 0x3, 0x3, // 보
            0x1, 0x2, 0x1, 0x2, 0x1, // 파
            0x2, 0x2, 0x5, 0x5, 0x5, // 노
            0x0, 0x2, 0x2, 0x2, 0x0, // 주
            0x4, 0x3, 0x2, 0x2, 0x4, // 초
    };


    fdM_Color = open(MATRIX_FILE_NAME, O_RDWR);
    if(fdM_Color < 0){
        fprintf(stderr, "Can't open %s₩n",MATRIX_FILE_NAME);
        return -1;
    }


    data = *(char *)arg;
    printf("fdM_Color write = %d₩n", data);
    write(fdM_Color, color[data - 1], sizeof(char));

    close(fdM_Color);
    return 0;

}


void *thread_matrixCount(void *arg) {
    int fdM_Count;
    char data;

    char num[9][5] = {
            0x6, 0x6, 0x6, 0x6, 0x6, //1
            0x0, 0x6, 0x0, 0x3, 0x0, //2
            0x0, 0x6, 0x0, 0x6, 0x0, //3
            0x2, 0x2, 0x0, 0x6, 0x6, //4
            0x0, 0x3, 0x0, 0x6, 0x0, //5
            0x0, 0x3, 0x0, 0x2, 0x0, //6
            0x0, 0x6, 0x6, 0x6, 0x6, //7
            0x0, 0x2, 0x0, 0x2, 0x0, //8
            0x0, 0x2, 0x0, 0x6, 0x0, //9
    };


    fdM_Count = open(MATRIX_FILE_NAME, O_RDWR);
    if(fdM_Count < 0){
```

```
        fprintf(stderr, "Can't open %s₩n",MATRIX_FILE_NAME);
        return -1;
    }


    data = *(char *)arg;
    printf("fdM_Count write = %d₩n", data);
    write(fdM_Count, num[data - 1], sizeof(char));

    close(fdM_Count);
    return 0;

}
```

❍ **mat_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/delay.h>
#include <linux/time.h>
#include <linux/jiffies.h>

#define     MATRIX_MAJOR    221
#define     MATRIX_NAME     "MATRIX_DRIVER"
#define     GPIO_SIZE 256

char matrix_usage = 0;
static void *matrix_map;
volatile unsigned *matrix;

static int matrix_open(struct inode *minode, struct file *mfile)
{
    unsigned char index;

    if (matrix_usage != 0)
        return -EBUSY;
    matrix_usage = 1;



    matrix_map = ioremap(GPIO_BASE, GPIO_SIZE);
    if (!matrix_map)
    {
        printk("error: mapping gpio memory");
        iounmap(matrix_map);
        return -EBUSY;
    }
```

```
    matrix = (volatile unsigned int *)matrix_map;
    for (index = 2; index <= 9; ++index)
    {
        *(matrix) &= ~(0x07 << (3 * index));
        *(matrix) |= (0x01 << (3 * index));
    }
    return 0;
}

static int matrix_release(struct inode *minode, struct file *mfile)
{
    matrix_usage = 0;
    if (matrix)
        iounmap(matrix);
    return 0;
}

static int matrix_write(struct file *mfile, const char *gdata, size_t length, loff_t *off_what) {
    char tmp_buf[5];
    char val[5];
    int result, i;
    unsigned long delay = jiffies + 1*HZ;

    result = copy_from_user(&tmp_buf, gdata, length);
    if (result < 0) {
        printk("Error: copy from user");
        return result;
    }

    for (i = 0; i < 5; i++)
        val[i] = gdata[i];

    printk("matrix values = %s₩n", tmp_buf);

    while(time_before(jiffies, delay)) {
        for (i = 0; i < 5; i++) {
            *(matrix + 7) = (val[i] << 2);
            *(matrix + 7) = (0x1 << (i + 5));
            udelay(1);
            *(matrix + 10) = (0xFF << 2);
        }
    }
    return length;
}

static struct file_operations matrix_fops =
        {
                .owner          = THIS_MODULE,
                .open           = matrix_open,
                .release        = matrix_release,
                .write          = matrix_write,
        };
```

```
static int matrix_init(void)
{
    int result;

    result = register_chrdev(MATRIX_MAJOR, MATRIX_NAME, &matrix_fops);
    if (result < 0)
    {
        printk(KERN_WARNING "Can't get any major number₩n");
        return result;
    }
    return 0;
}

static void matrix_exit(void)
{
    *(matrix + 10) = (0xFF << 2);// clear matrix
    unregister_chrdev(MATRIX_MAJOR, MATRIX_NAME);
    printk("MATRIX module removed.₩n");
}

module_init(matrix_init);
module_exit(matrix_exit);

MODULE_LICENSE("GPL");
```

❍ motX_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>
#include <linux/delay.h>
#include <linux/jiffies.h>

#define motor180x_MAJOR         222
#define motor180x_NAME          "MOTOR180X_DRIVER"
#define GPIO_SIZE       256
#define GPIO            10


char motor180x_usage=0;
static void *motor180x_map;
volatile unsigned *motor180x;
```

```
static int motor180x_init(void);
static void motor180x_exit(void);
static int motor180x_release(struct inode*, struct file*);
static int motor180x_open(struct inode*, struct file*);
static int motor180x_write(struct file*, const char*, size_t, loff_t*);



static struct file_operations motor180x_fops=
        {
                .owner =THIS_MODULE,
                .open   =motor180x_open,
                .release=motor180x_release,
                .write   =motor180x_write,
        };

static int motor180x_init(void){
    int result;
    result=register_chrdev(motor180x_MAJOR, motor180x_NAME, &motor180x_fops);
    if(result<0){
        printk(KERN_WARNING"Can't get any major!₩n");
        return result;
    }
}




static void motor180x_exit(void){
    unregister_chrdev(motor180x_MAJOR, motor180x_NAME);
    printk("motor180x module removed.₩n");
}



module_init(motor180x_init);
module_exit(motor180x_exit);
MODULE_LICENSE("GPL");


static int motor180x_open(struct inode *inode, struct file *mfile){
    if(motor180x_usage != 0)
        return -EBUSY;
    motor180x_usage=1;

    motor180x_map=ioremap(GPIO_BASE, GPIO_SIZE);

    if(!motor180x_map){
        printk("error:mapping gpio memory");
        iounmap(motor180x_map);
```

```
        return -EBUSY;
    }
    motor180x=(volatile unsigned int*)motor180x_map;

    *(motor180x+(GPIO/10))&=~(0x07<<(3*(GPIO%10)));
    *(motor180x+(GPIO/10))|=(0x01<<(3*(GPIO%10)));

    return 0;
}

static int motor180x_write(struct file *minode, const char *gdata, size_t length, loff_t *off_what){
    char tmp_buf;
    int result;
    int frequency=20000;
    int minAngle=544;
    int maxAngle=2440;
    int x;
    unsigned int setClock;
    unsigned int clrClock;
    unsigned long stpDelay = jiffies + 2*HZ;

    result=copy_from_user(&tmp_buf, gdata, length);
    if(result<0){
        printk("Error:copy from user");
        return result;
    }
    printk("data from user: %d₩n", tmp_buf);




    unsigned long delay = jiffies + (int)tmp_buf*HZ;
    while(time_before(jiffies, delay)){}
    for(x=90;x>=45;x--){
        setClock=(minAngle+((maxAngle-minAngle)/90*x));
        clrClock=frequency-setClock;
        *(motor180x+7)=(0x01<<GPIO);udelay(setClock);
        *(motor180x+10)=(0x01<<GPIO);
        udelay(clrClock);
    }
    while(time_before(jiffies, stpDelay)){}
    for(x=45;x<=90;x++){
        setClock=(minAngle+((maxAngle-minAngle)/90*x));
        clrClock=frequency-setClock;
        *(motor180x+7)=(0x01<<GPIO);
        udelay(setClock);
        *(motor180x+10)=(0x01<<GPIO);
        udelay(clrClock);
    }

    return length;
}
```

```
static int motor180x_release(struct inode *minode, struct file *mfile){
    motor180x_usage=0;
    if(motor180x)  iounmap(motor180x);
    return 0;
}
```

❍ **moty_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>
#include <linux/delay.h>



#define motor180y_MAJOR          223
#define motor180y_NAME           "MOTOR180Y_DRIVER"
#define GPIO_SIZE       256
#define GPIO            11

char motor180y_usage=0;
static void *motor180y_map;
volatile unsigned *motor180y;


static int motor180y_init(void);
static void motor180y_exit(void);
static int motor180y_release(struct inode*, struct file*);
static int motor180y_open(struct inode*, struct file*);
static int motor180y_write(struct file*, const char*, size_t, loff_t*);



static struct file_operations motor180y_fops=
        {
                .owner =THIS_MODULE,
                .open   =motor180y_open,
                .release  =motor180y_release,
                .write   =motor180y_write,
        };
```

```
static int motor180y_init(void){
	int result;
	result=register_chrdev(motor180y_MAJOR, motor180y_NAME, &motor180y_fops);
	if(result<0){
		printk(KERN_WARNING"Can't get any major!₩n");
		return result;
	}
}


static void motor180y_exit(void){
	unregister_chrdev(motor180y_MAJOR, motor180y_NAME);
	printk("motor180y module removed.₩n");
}



module_init(motor180y_init);
module_exit(motor180y_exit);
MODULE_LICENSE("GPL");


static int motor180y_open(struct inode *inode, struct file *mfile){
	if(motor180y_usage != 0)
		return -EBUSY;
	motor180y_usage=1;

	motor180y_map=ioremap(GPIO_BASE, GPIO_SIZE);

	if(!motor180y_map){
		printk("error:mapping gpio memory");
		iounmap(motor180y_map);
		return -EBUSY;
	}
	motor180y=(volatile unsigned int*)motor180y_map;

	*(motor180y+(GPIO/10))&=~(0x07<<(3*(GPIO%10)));
	*(motor180y+(GPIO/10))|=(0x01<<(3*(GPIO%10)));

	return 0;
}

static int motor180y_write(struct file *minode, const char *gdata, size_t length, loff_t *off_what){
	char tmp_buf;
	int result;
	int frequency=20000;
	int minAngle=544;
	int maxAngle=2440;
	int x;
	unsigned int setClock;
	unsigned int clrClock;
	unsigned long delay = jiffies + 1*HZ;
```

```
    result=copy_from_user(&tmp_buf, gdata, length);
    if(result<0){
        printk("Error:copy from user");
        return result;
    }
    printk("data from user: %d₩n", tmp_buf);


    if(tmp_buf==0){
  for(x=0;x<=24;x++){
            setClock=(minAngle+((maxAngle-minAngle)/40*x));
            clrClock=frequency-setClock;
            *(motor180y+7)=(0x01<<GPIO);
            udelay(setClock);
            *(motor180y+10)=(0x01<<GPIO);
            udelay(clrClock);
        }

    }

    else{
       for(x=24;x>=0;x--){
            setClock=(minAngle+((maxAngle-minAngle)/40*x));
            clrClock=frequency-setClock;
            *(motor180y+7)=(0x01<<GPIO);
            udelay(setClock);
            *(motor180y+10)=(0x01<<GPIO);
            udelay(clrClock);
        }

    }

    return length;
}


static int motor180y_release(struct inode *minode, struct file *mfile){
    motor180y_usage=0;
    if(motor180y)
       iounmap(motor180y);
    return 0;
}
```

❍ motz_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>
#include <linux/delay.h>



#define motor180y_MAJOR 223
#define motor180y_NAME "MOTOR180Y_DRIVER"
#define GPIO_SIZE 256
#define GPIO 11

char motor180y_usage=0;
static void *motor180y_map;
volatile unsigned *motor180y;


static int motor180y_init(void);
static void motor180y_exit(void);
static int motor180y_release(struct inode*, struct file*);
static int motor180y_open(struct inode*, struct file*);
static int motor180y_write(struct file*, const char*, size_t, loff_t*);



static struct file_operations motor180y_fops=
	{
		.owner =THIS_MODULE,
		.open =motor180y_open,
		.release =motor180y_release,
		.write =motor180y_write,
	};

static int motor180y_init(void){
    int result;
    result=register_chrdev(motor180y_MAJOR, motor180y_NAME, &motor180y_fops);
    if(result<0){
        printk(KERN_WARNING"Can't get any major!₩n");
        return result;
    }
}


static void motor180y_exit(void){
    unregister_chrdev(motor180y_MAJOR, motor180y_NAME);
    printk("motor180y module removed.₩n");
}
```

```
module_init(motor180y_init);
module_exit(motor180y_exit);
MODULE_LICENSE("GPL");


static int motor180y_open(struct inode *inode, struct file *mfile){
	if(motor180y_usage != 0)
		return -EBUSY;
	motor180y_usage=1;

	motor180y_map=ioremap(GPIO_BASE, GPIO_SIZE);

	if(!motor180y_map){
		printk("error:mapping gpio memory");
		iounmap(motor180y_map);
		return -EBUSY;
	}
	motor180y=(volatile unsigned int*)motor180y_map;

	*(motor180y+(GPIO/10))&=~(0x07<<(3*(GPIO%10)));
	*(motor180y+(GPIO/10))|=(0x01<<(3*(GPIO%10)));

	return 0;
}

static int motor180y_write(struct file *minode, const char *gdata, size_t length, loff_t *off_what){
	char tmp_buf;
	int result;
	int frequency=20000;
	int minAngle=544;
	int maxAngle=2440;
	int x;
	unsigned int setClock;
	unsigned int clrClock;
	unsigned long delay = jiffies + 1*HZ;

	result=copy_from_user(&tmp_buf, gdata, length);
	if(result<0){
		printk("Error:copy from user");
		return result;
	}
	printk("data from user: %d₩n", tmp_buf);


	if(tmp_buf==0){
   for(x=0;x<=24;x++){
			setClock=(minAngle+((maxAngle-minAngle)/40*x));
			clrClock=frequency-setClock;
			*(motor180y+7)=(0x01<<GPIO);
			udelay(setClock);
			*(motor180y+10)=(0x01<<GPIO);
```

```
                udelay(clrClock);
            }

        }

        else{
            for(x=24;x>=0;x--){
                setClock=(minAngle+((maxAngle-minAngle)/40*x));
                clrClock=frequency-setClock;
                *(motor180y+7)=(0x01<<GPIO);
                udelay(setClock);
                *(motor180y+10)=(0x01<<GPIO);
                udelay(clrClock);
            }

        }

        return length;
    }


    static int motor180y_release(struct inode *minode, struct file *mfile){
        motor180y_usage=0;
        if(motor180y)
            iounmap(motor180y);
        return 0;
    }
```

❍ sensor_driver.c

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

    #define s2 17
    #define s3 18
    #define out 27

    #define GPIO_SIZE 256
    #define SENSOR_MAJOR 241
    #define SENSOR_NAME "Color_Sensor"
```

```
char sensor_usage = 0;
static void *sensor_map;
char value;
volatile unsigned *color_sensor;
static int event_flag;
int d_flag;

DECLARE_WAIT_QUEUE_HEAD(waitqueue);


static irqreturn_t ind_interrupt_handler(int irq, void *pdata)
{
    value = 1;
    wake_up_interruptible(&waitqueue);
    ++event_flag;
    return IRQ_HANDLED;
}
static unsigned sensor_poll(struct file *inode, struct poll_table_struct *pt)
{
    int mask = 0;
    poll_wait(inode, &waitqueue, pt);
    if (event_flag > 0)
        mask |= (POLLIN | POLLRDNORM);
    event_flag = 0;
    return mask;
}


//change the map name an change the usage name
static int sensor_open(struct inode *minode, struct file *mfile)
{
    if (sensor_usage != 0)
        return -EBUSY;
    sensor_usage = 1;
    sensor_map = ioremap(GPIO_BASE, GPIO_SIZE);
    if (!sensor_map)
    {
        printk("error: mapping gpio memory");
        iounmap(sensor_map);
        return -EBUSY;
    }

    color_sensor = (volatile unsigned int *)sensor_map;

    *(color_sensor + (out/10)) &= ~(0x7 << (3 * (out%10)));
    *(color_sensor + (out/10)) |= (0x0 << (3 * (out%10)));   // (sensor out pin) Input mode [read]
    *(color_sensor + 22) |= (0x1 << out);

    *(color_sensor + (s3/10)) &= ~(0x7 << (3 * (s3%10)));
    *(color_sensor + (s3/10)) |= (0x1 << (3 * (s3%10))); // (sensor s3 pin)   output mode [write]
```

```
    *(color_sensor + (s2/10)) &= ~(0x7 << (3 * (s2%10)));
    *(color_sensor + (s2/10)) |= (0x1 << (3 * (s2%10))); // (sensor s2 pin) output mode [write]
    request_irq(gpio_to_irq(27), ind_interrupt_handler, IRQF_TRIGGER_FALLING, "irq_key", NULL);

    return 0;
}


static int sensor_read(struct file *inode, char *gdata, size_t length, loff_t *off_what) {
    int result;
    //printk("sensor read₩n");
    result = copy_to_user(gdata, &value, length);
    if (result < 0) {
        printk("error : copy_to_user()");
        return result;
    }
    return length;
}


static int sensor_write(struct file* mfile, const char* gdata, size_t length, loff_t* off_what) {
    struct timeval start;
    struct timeval end;
    int Freq_UpDw = 1;
    int result;
    int RGB_buf;
    char tmp_buf;

    unsigned int red;
    unsigned int blue;
    unsigned int green;

    result = copy_from_user(&tmp_buf, gdata, length);
    if (result < 0) {
        printk("Error : copy from user");
        return result;
    }

    if (tmp_buf == 1) {
        *(color_sensor + 10) |= (0x1 << s2); // s2 L
        *(color_sensor + 10) |= (0x1 << s3); // s3 L mode red
    }
    else if (tmp_buf == 2) {
//          *(color_sensor + 10) |= (0x1 << s2); // s2 L
        *(color_sensor + 7) |= (0x1 << s3); // s3 H mode blue
    }
    else if (tmp_buf == 3) {
        *(color_sensor + 7) |= (0x1 << s2); // s2 H
//          *(color_sensor + 7) |= (0x1 << s3); // s3 H mode green
    }
```

```
    return length;
}
static int sensor_release(struct inode *minode, struct file *mfile)
{
    //change the usage name
    sensor_usage = 0;
    if (color_sensor)
        iounmap(color_sensor);
    free_irq(gpio_to_irq(27), NULL);
    return 0;
}

static struct file_operations sensor_fops =
    {
        .owner      = THIS_MODULE,
        .open       = sensor_open,
        .release    = sensor_release,
        .write      = sensor_write,
        .read       = sensor_read,
        .poll       = sensor_poll,
    };



static int sensor_init(void)
{
    int result;
    //change ctrcut name address
    result = register_chrdev(SENSOR_MAJOR, SENSOR_NAME, &sensor_fops);
    if (result < 0)
    {
        printk(KERN_WARNING "Can't get any major!₩n");
        return result;
    }
    return 0;
}



static void sensor_exit(void)
{
    unregister_chrdev(SENSOR_MAJOR, SENSOR_NAME);
    printk("sensor module removed.₩n");
}



module_init(sensor_init);
module_exit(sensor_exit);
MODULE_LICENSE("GPL");
```

3. 회의록

번호	날짜	시간	지난주 회의 내용	이번주 회의내용(이행 사항 기록)
1	10.17	15:00 ~18:00	.	1. 진행할 프로젝트의 주제를 색깔 분류기로 정함. 2. 관련 자료 수집 필요 3. 구조도면 설계 4. 제안서 발표 준비
2	11.9	15:00 ~18:00	진행할 프로젝트의 주제를 색깔 분류기로 정함.	1. 프로젝트 계획 정리 2. 부품 수령 완료 3. Device Driver 프로그래밍에 대한 이해가 부족하므로 연구 필요 4. 각자 수집한 자료들을 종합
3	11.13	15:00 ~18:00	1. Device Driver 프로그래밍에 대한 이해가 부족하므로 연구 필요 2. 각자 수집한 자료들을 종합	1. 사용 부품 변경(카메라 -> 컬러 센서) 2. python3 프로그램을 통해 센서의 정상작동 확인필요 3. Device Driver 코드 및 User code 작성 시작
4	11.23	15:00 ~18:00	1. Device Driver 코드 및 User code 작성 시작	1. Device driver 코드 작성 확인 2. User code 수정 필요 확인 3. 구조물 제작 및 코드 세부 조정 필요
5	12.01	15:00 ~18:00	1. Device driver 코드 작성 확인 2. User code 수정 필요 확인 3. 구조물 제작 및 코드 세부 조정 필요	1. 결과물 최종 점검 2. 최종 발표 및 데모 준비

4. 참고 문헌, 사이트

❍ [외국. 저자 불명] "SG90 Data sheet"
http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

❍ [외국. Broadcom Corporation] "Broadcom BCM 2837 ARM Peripherals"
https://cs140e.sergio.bz/docs/BCM2837-ARM-Peripherals.pdf

❍ [외국. Raspberry Pi (Trading) Ltd] "Raspberry Pi Compute Module Data sheet"
https://www.alliedelec.com/m/d/1988f69b72864ea60d5867861de53e42.pdf