[System Programming]

# 19-1 시스템 프로그래밍
# 최종 결과 보고서

- Team 싶-


Professor : 고영배 교수님

3조


| | |
|---|---|
| 201420894 | 김성규 |
| 201720723 | 박수린 |
| 201720743 | 정소희 |

# Table of Contents

# 1. 프로젝트 개요

재난 상황 시 유독 가스 혹은 유해 물질의 호흡기로의 유입은 생명에 매우 치명적이다. 실제로, 화재 사고의 경우에 사망 원인의 60% 이상이 연기에 의한 질식사이다. 이런 위험성을 가지는 유독 가스의 유입을 막기 위해 방독면을 착용하는데, 현재 일반인에게 보급되는 방독면은 부족한 점이 많다. 보급형 방독면의 필터는 일회성으로 방호 가능 시간이 지나면 교체해야 하는데, 현재 보급 방독면은 방호 상태가 얼마나 더 지속될 지 예상하기 힘들다. 그래서 우리의 프로젝트는 현재 방독면 내의 이산화탄소 농도를 측정하여 수치에 따라 소리를 다르게 출력하여 사용자에게 방호 가능 시간이 얼마 남지 않았다는 신호를 출력한다.


사회 > 전체기사

화재 발생 시 가장 치명적인 것, 불 아닌 '연기와 가스'

입력 : 2018-01-26 15:51



26일 오전 7시 32분께 화재가 발생한 경남 밀양시 가곡동 세종요양병원 사고현장에서 소방대원이 수색작업을 하고 있다. 2018.01.26. (사진=경남도민일보 제공)

불보다 더 무서운 것은 연기와 유독가스였다.


실제로, 최근 '엑시트' 라는 영화에서 유독 가스 테러 상황의 재난을 다루었으며 방독면의 중요성이 매우 강조되었다. 그 중  주인공들이 방독면의 방호 가능 시간을 측정하기 위해 핸드폰의 타이머를 켜고 탈출하는 모습에 아이디어를 얻게 되었으며, 보급형 방독면의 기능을 보완하고 재난 상황에서 적절하게 대응할 수 있는 방독면을 주제로 선정하게 되었다.

이 프로젝트는 일반인에게 보급되는 방독면을 대상으로 진행하였다. 더 고수준의 센서와 기술을 활용한다면 구조 대원이 사용하는 전문적인 방독면에도 적용 가능할 것이다.

프로젝트의 방독면 기능은 크게 세가지로 나눌 수 있다.

| |
|---|
| 장애물을 감지하면 진동을 통해 알려주는 기능 |
| 방독면 내의 유독가스의 농도가 일정 수준을 초과하면 소리를 통해 알려주는 기능 |
| 방독면 내의 불쾌지수가 일정 수준을 초과하면 모터를 작동시켜 공기를 환기시켜주는 기능 |

# 2. 프로젝트 시나리오

## 가) 방호 가능 시간 알림

i. 가스 센서를 통해 방독면 내 CO2농도를 측정한다.
ii. 방독면 내에 입김을 불어 CO2 농도를 증가시킨다.
iii. CO2 수치가 일정 수준 이상이 되면 소리 모듈에서 정화통 교체 알림이 울린다.
(사람에게 불쾌감을 주는 이산화탄소 농도는 1000ppm이기 때문에 1000ppm, 1300ppm을 알림 기준으로 설정하였음)

## 나) 방독면 내부 환기

i. 온습도 센서를 통해 주기적으로 방독면 내 습도와 온도를 측정한다.
ii. 측정한 습도와 온도를 통해 불쾌지수를 계산한다.
iii. 방독면 내부의 불쾌지수가 임계 값을 초과하면 모터를 동작시킨다.
iv. 임계 값 이하의 불쾌지수가 갖게 될 때까지 방독면 내부 공기를 환기시킨다.

## 다) 전방의 장애물 감지

i. 방독면 전방에 장애물을 설치한다.
ii. 장애물을 방독면이 있는 쪽으로 이동시킨다.
iii. 방독면과 장애물 사이의 거리가 150cm 이하가 되면 진동 모듈이 작동한다.
iv. 장애물이 가까이 다가갈 수록 진동 모듈의 세기를 증가시킨다.
v. 전방에 장애물이 있음을 감지한다.
〈구현에 사용하는 센서의 측정 범위가 20~150cm이기 때문에 150cm를 기준으로 설정하였음〉

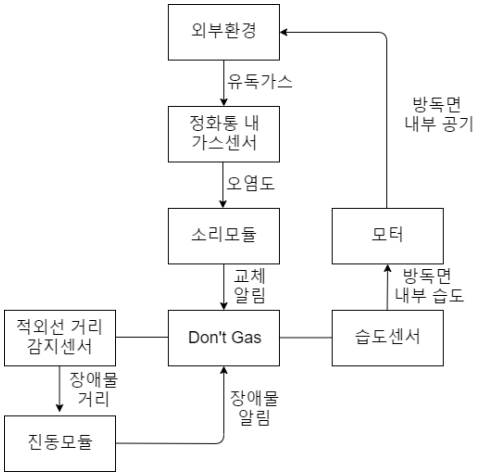



**Figure 1 시스템 구조도 및 구성 요소**

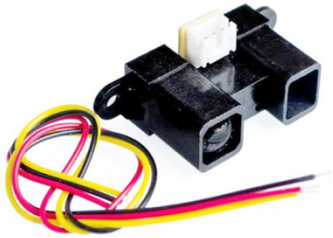
시나리오 구현을 위해서 여러 가지 제한 사항을 두었다. 실제 유독 가스는 매우 다양하지만 구현 시에는 구하기 쉬운 이산화탄소를 이용하였고, 시연을 위해, 방독면 모형의 구현물을 만들어 직사각형의 밀폐된 통에 구멍을 뚫어 원통형 종이를 연결한 후, 이 곳에 입김을 불어 이산화탄소가 가득 찬 상황을 만들었다. 또한 실제로 방독면은 이동 상황에서 사용되기 때문에 배터리가 필요하지만, 구현 시에는 전원으로 대체하였다.

# 3. 시스템 설계

## 가) 하드웨어 구조

### i. 적외선 거리 감지 센서 (GP2Y0A02YK0F)

① 하드웨어에 대한 설명 : 4.5V~5V 전압에서 작동하며 20~150cm 내의 거리에 있는 물체를 감지할 수 있다. 검출 전압은 15cm에서 2.8V에서 150cm에서 0.4V 까지 이다. 아두이노 및 라즈베리 파이에 적용 가능하다. 검출 주기는 38±10 ms이다.



② 선택 이유 : 가능한 범위 내에서 합리적인 가격의 가장 긴 거리의 광학 센서였기 때문에 이 센서를 선택하였다.

③ 작동 방식 : 감지된 거리에 상응하는 아날로그 전압을 출력한다. 데이터 시트를 참고하면, 센서의 출력 전압과 측정된 거리의 역수 관계가 선형을 이루는 것을 알 수 있다. 오른쪽 사진은 데이터 시트에서 참고한 전압과 거리 사이의 관계를 나타낸 그래프이다. 이 센서는 아날로그 출력 전압이기 때문에, 라즈베리파이에게 값을 전달하기 위해서는 ADC가 필요하다.

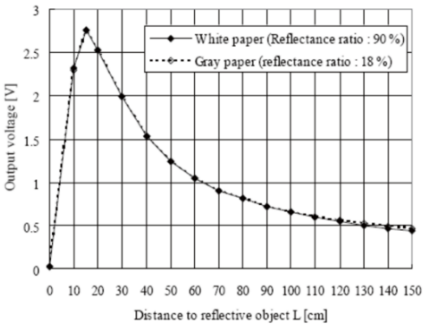
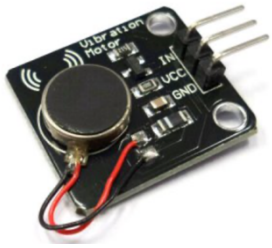



**Figure 2 측정된 거리와 출력 전압 사이의 관계**

### ii. 진동 모터 모듈

① 하드웨어에 대한 설명과 작동 방식 : 스마트폰이나 소형 진동기 등에 부착된 소형 진동모터 내장 모듈이다. 5V 전원으로 작동하며 디지털 신호 또는 PWM 신호에 따라 진동을 On/Off 하거나 진동의 세기를 제어해서 출력할 수 있다. IN, VCC, GND 3개의 핀으로 이루어져 있다.




② 선택 이유 : 가스 노출로 인한 위험 상황을 소리로 사용자에게 알려주기 때문에, 위험 상황을 알릴 수 있는 다른 수단이 필요하였고, 진동으로 알림을 주는 것이 적절하다고 생각하였다. 또한 모듈 자체가 소형이며, 가격에 비해 질이 좋기 때문에 이 전동 모터 모듈을 선택하였다.

### iii. CO2 Sensor(MH-Z19B)

① 하드웨어에 대한 설명 : CO2 농도 수준에 비례하여 UART, PWM, Analog의 형식으로 출력할 수 있는 센서이며 아날로그 출력 범위는 0~3V이다. 사용한 출력 TYPE은 Terminal connection version이다.

| | |
|---|---|
| Output signal | Serial Port (UART) (TTL level 3.3V) |
| | PWM |
| | Analog output(DAC) (default 0.4~2V)<br>(0~3V range could be customized) |

**Figure 3 MH-Z19B의 output signal의 종류**

오른쪽 표는 Terminal connection version의 PIN별 정보이다. 우리는 PIN 1에 연결하여 Analog 출ㅋ력을 사용했기 때문에 라즈베리파이에게 값을 전달하기 위해서는 ADC가 필요하여 MCP 3008과 연결하여 사용하였다.

| Pin | Terminal pin Definition |
|---|---|
| Pin 1 | Anolog Output Vo |
| Pin 2 | None |
| Pin 3 | Negtive Pole(GND) |
| Pin 4 | Positive Pole(Vin) |
| Pin 5 | UART(RXD)TTL Level data input |
| Pin 6 | UART(TXD)TTL Level data output |
| Pin 7 | None |

**Figure 4 MH-Z19B의 Terminal PIN의 종류**

② 선택 이유 : 화재 시 발생하는 유해 가스 종류 중 쉽게 얻을 수 있는 이산화탄소를 이용해 구현하기 위해 선택했다.

③ 작동 방식 : CO2의 농도에 따른 센서의 아날로그 출력 값을 입력 받아 Voltage로 전환 후 datasheet의 변환식을 통해 ppm을 구한다.

### iv. Active Buzzer(MH-FMD)

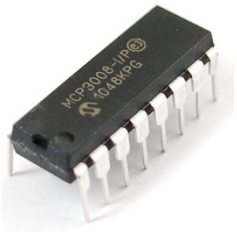
① 하드웨어에 대한 설명 : 미리 설계된 회로 때문에 음이 정해져 있는 부저이며 약 2KHz 대역의 소리를 출력한다.


② 선택 이유 : 화재 시 가려지는 시야 때문에 시각적인 알림보다 청각적인 알림이 더 효과적이다.

③ 작동 방식 : 음을 바꿀 수 없는 부저므로 on/off 반복적으로 실행시켜 위험을 나타낸다.

## v. MCP3008

① 하드웨어에 대한 설명 : 아날로그 값을 디지털로 변환해 주는 Analog to Digital Converter이다. 이 하드웨어는 아날로그 값을 0~1023개의 영역으로 나눈다. 즉 10개의 비트로 값을 읽어낼 수 있다.



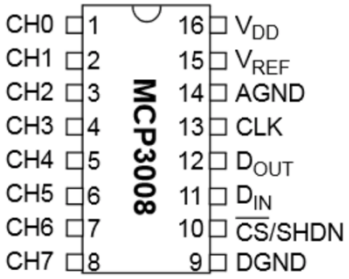



**Figure 5 MCP3008 PIN의 역할**

8개의 채널을 가지고 있어, 8개의 아날로그 입력을 처리할 수 있다. 라즈베리파이의 SPI버스가 3.3V에서 동작하기 때문에 더 큰 전력을 공급해줄 필요는 없다. 오히려 GPIO가 손상을 입을 수도 있다. 그래서, 3.3V의 전원을 Vdd, Vref에 공급해주고 AGND, DGND를 ground에 연결한 후, CLK, Dout, Din, CS를 라즈베리파이의 SPI핀에 연결해주면 사용할 수 있다. 아래 그림은 채널 0번에 해당 센서의 출력을 입력으로 했을 때, 라즈베리파이와의 연결 회로도 이다.

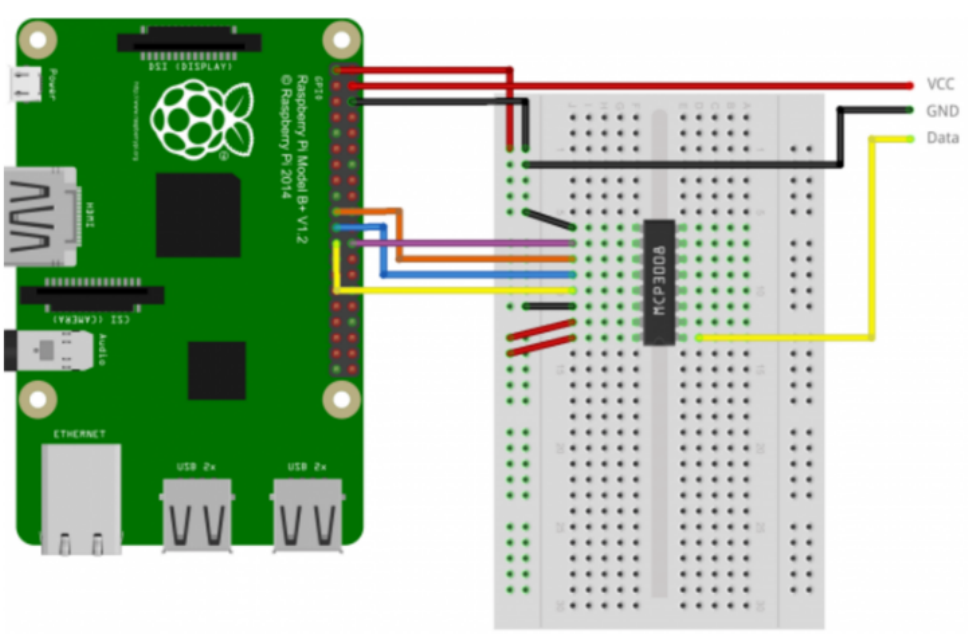



**Figure 6 MCP3008과의 SPI통신을 위한 회로도**

② 선택 이유 : 아두이노에 반해, 라즈베리 파이는 ADC가 내장되어 있지 않기 때문에 이 하드웨어가 필수적으로 필요하였다.

③ 작동 방식 : 라즈베리파이와 SPI를 사용해 칩과 통신하여 변환값을 전달해준다. 이 프로젝트에서는 아날로그 적외선 거리 감지 센서와 이산화탄소 센서가 아날로그 전압을 출력하기 때문에 각각 채널 0번과 1번에 연결하여 사용하였다. MCP3008을 이용해서 SPI 통신을 하기 위해서는 동작 방식을 알아야 한다. 데이터 시트를 보면, MCP3008은 (CPOL, CPHA)의 쌍이 (0,0) 이거나 (1,1)인 SPI 모드를 지원한다. (0,0) 모드의 신호 타이밍은 아래의 그림처럼 총 3byte(24bit)로 이루어져 있다.

| Control Bit Selections | | | | Input Configuration | Channel Selection |
|---|---|---|---|---|---|
| Single/ Diff | D2* | D1 | D0 | | |
| 1 | X | 0 | 0 | single-ended | CH0 |
| 1 | X | 0 | 1 | single-ended | CH1 |
| 1 | X | 1 | 0 | single-ended | CH2 |
| 1 | X | 1 | 1 | single-ended | CH3 |
| 0 | X | 0 | 0 | differential | CH0 = IN+ CH1 = IN- |
| 0 | X | 0 | 1 | differential | CH0 = IN- CH1 = IN+ |
| 0 | X | 1 | 0 | differential | CH2 = IN+ CH3 = IN- |
| 0 | X | 1 | 1 | differential | CH2 = IN- CH3 = IN+ |

**Figure 7 MCP3008 configuration bit**

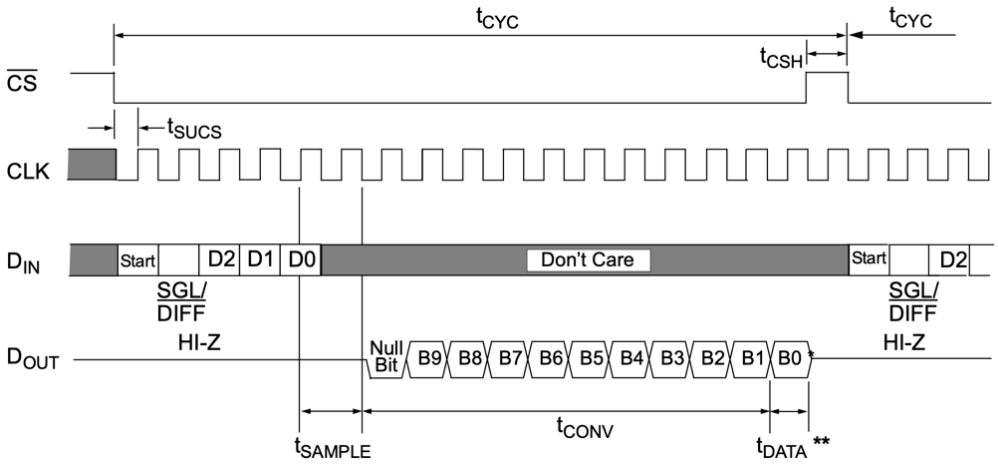
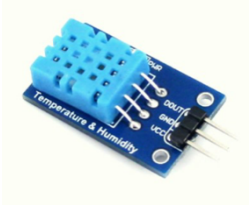



**Figure 8 MCP3008과의 communication**

첫 바이트에는 MCP3008의 DIN과 연결 되어 있는 라즈베리파이의 MOSI(Master out Slave In)을 통해 0x01을 전송하여 동작의 시작을 알리는 start 신호를 보낸다. 그 다음 두 번째 바이트에서는 상위 4비트로 configuration bits(SGL, D2, D1, D0)을 전송한다. 이 4비트의 구성으로 ADC값을 얻고자 하는 채널을 선택할 수 있다. 나머지 4비트는 아무런 영향을 미치지 않는 don't care 비트이다. MCP 3008이 D0까지 입력을 받으면, DOUT핀을 통해 null 비트로 시작하는 10 비트의 디지털 값을 라즈베리파이의 MISO(Master In Slave Out)으로 보내기 시작한다.

## vi. DHT11 온&습도 센서




① 하드웨어에 대한 설명 : DHT11은 0~50도의 온도 측정이 가능하며 +-2도의 오차범위를 가지고 있으며 20~95%의 습도 측정이 가능하고 +-5%의 오차범위를 가지고 있다.

② 선택 이유 : 방독면 내부의 불쾌지수를 측정하기 위해 온도와 습도를 측정하기위해 사용했다.

③ 작동 방식 : DHT11을 작동하기 위해서는 우선 DHT11을 OUTPUT MODE로 설정해 주어야 한다.
그리고 18ms이상의 low signal을 보낸 후 센서를 INPUT MODE로 변환하면 주변의 온도와 상대습도를 측정한다. DHT11은 Response signal을 전송하기 위해 pull up상태로 전환되고 low signal인 Response signal을 전송한다. 그 후 다시 pull up 상태로 전환되어 측정한 온도와 상대습도를 전송하며, 전송되는 데이터의 형태는 다음과 같다.

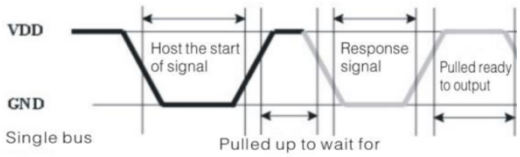



**Figure 9 data timing diagram(1)**

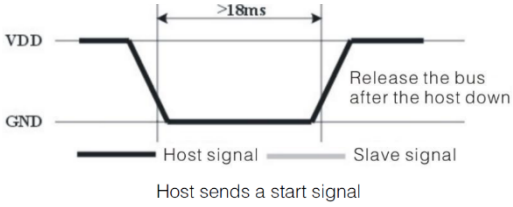



**Figure 10 data timing diagram(2)**

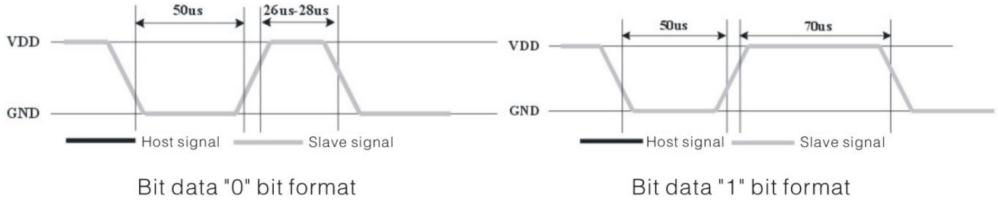



**Figure 11 bit data format**

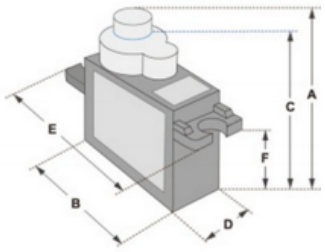
DHT11은 high signal을 전송하는 시간을 통해 전송하고자 하는 데이터의 값이 '0'인지 '1'인지 나타낸다. 데이터의 값이 '0'인 경우 DHT11은 26~28us의 high signal을 보내고 '1'인 경우 70us의 high signal을 보낸다. 이렇게 총 40비트의 데이터를 전송하게 되는데, 비트의 구성은 다음과 같다.

| 1~8 | 9~16 | 17~24 | 25~32 | 33~40 |
|---|---|---|---|---|
| 습도-정수 | 습도-소수점 | 온도-정수 | 온도-소수점 | Parity bits |

전송된 데이터의 마지막 8비트는 checksum을 통해 앞의 32비트 데이터가 올바른 값인지 확인하기 위한 parity bits로 구성되어 있다.

### vii. SG90 서보 모터

① 하드웨어에 대한 설명 : SG90은 모터이지만 한 방향으로 연속해서 360도를 회전하지 못한다. 중앙을 기준으로 최대 +-180도만큼 각도를 조절할 수 있는 모터이다. SG90은 PWM방식을 사용하여 동작하는데 pulse의 주기가 20ms로 설정되어 있다.



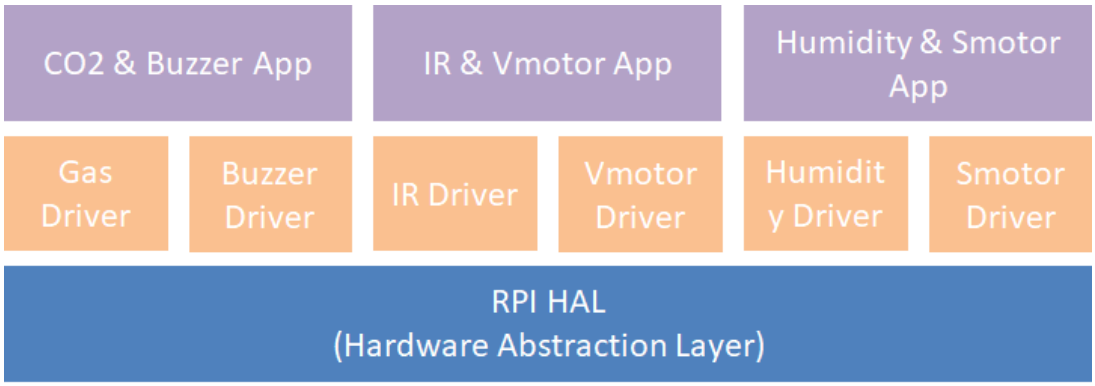
Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

② 선택 이유 : 방독면 내부의 공기를 순환시키기 위해서 사용했다.

③ 작동 방식 : 20ms 중 2.5ms를 high로 할당해 주면 중앙을 기준으로 180도에 위치하고 1.5ms를 high로 할당하면 0도에 위치하며 0.5ms를 high로 설정하면 -180도에 위치한다. 이렇게 0.5~2.5ms사이로 pulse의 폭을 조절하여 모터의 각도를 조절할 수 있다.

## 나) 소프트웨어 구조

센서와 액츄에이터 각각의 device driver를 만들고 커널에 올린 뒤, 기능 별로 app에서 이를 호출한 뒤 사용한다.



# 4. 시스템 구현

## 가) 디바이스 드라이버

### i. humidity_dev.c

커널 수준에서는 소수점 연산이 불가능하기 때문에 Info 구조체에 측정한 온도와 습도를 정수 부분과 실수 부분으로 나누어 저장하여 응용 프로그램 수준으로 전달한다. humidity_ioctl 함수는 DHT11을 통해 온도와 습도를 측정하는 함수이다. 라즈베리 파이의 GPIO 4번 핀에 DHT11을 연결했기 때문에 DHT11을 OUTPUT MODE로 변환하기 위해서 GPFSEL0 레지스터의 12번째 비트를 HIGH로 만들어 줘야 한다. 그 후에 18ms의 low signal을 보내주어야 하는데, 응용 프로그램에서 while문을 반복하다 보면 싱크가 맞지 않는 경우가 발생하기 때문에 이를 방지하기 위해 GPIO 4번핀을 HIGH로 변환하고 800ms동안의 delay를 주었다. 그리고 GPIO 4번핀을 18ms동안 LOW로 설정하여 DHT11에게 센싱을 요청하는 signal을 보내고 30us동안 HIGH로 설정하여 INPUT MODE로 변환했을 때 LOW signal이 입력되는 것을 방지하였다.

이제 GPIO 4번 핀을 LOW로 설정하여 INPUT MODE로 만들고 측정한 데이터를 전송하기 시작한다. 처음에 전달되는 3개의 비트는 무의미한 값이므로 while문을 통해 무시하고 4번째 비트부터 데이터를 저장하기 시작한다. DHT11은 HIGH signal의 시간을 통해 값을 표현하기 때문에 GPLEV0 레지스터를 통해 얻은 값이 LOW일 경우 HIGH가 될 때까지 이를 무시한다. GPLEV0 레지스터가 HIGH가 되면 HIGH signal의 시간을 측정하기 위해 1us단위로 counter의 값을 증가시킨다. DHT11는 '0'을 전송하기 위해 26~28us동안 HIGH를, '1'을 전송하기 위해 70us동안 HIGH를 전송하기 때문에 counter의 값이 70us를 넘게 되면 잘못된 값이 입력되었다고 판단하여 함수를 종료한다. counter값이 70이하라면 정상적인 값이 들어왔다고 판단하여 0으로 초기화된 data 배열의 값을 1만큼 left shift한다. 이 때, 총 40비트의 입력 값을 순서대로 습도의 정수, 습도의 소수, 온도의 정수, 온도의 소수, parity bits로 5개의 부분으로 나누어 주어야 하기 때문에 data배열의 index값을 [index/8]로 설정해 주었다. 그리고 만약 counter의 값이 28보다 크다면 이 값은 '1'을 의미하는 것이기 때문에 data배열의 값을 1로 설정하고 index값을 증가시켜 이 과정을 40번 반복한다. 저장된 data배열의 5번째 index와 1~4번째 index값의 합을 비교하여 둘이 같지 않다면 checksum이 제대로 이루어지지 않은 것이므로 함수를 종료한다. 만약 값이 제대로 저장이 되었다면 각각의 데이터를 Info 구조체의 변수에 할당하여 이를 응용 프로그램에게 전달한다.

### ii. motor_dev.c

motor_ioctl 함수는 SG90 서보 모터를 회전시키는 함수이다. PWM 채널2을 사용하기 위해서는 GPIO 13번 또는 19번 핀을 사용해야 하는데, 13번 핀에 꽂았기 때문에 GPFSEL1 레지스터의 9~11번째 비트를 ALT0로 설정하여 PWM1로 function을 선택한다. 현재 PWM_CTL 레지스터의 상태를 임시 변수에 저장 한 후, PWM clock을 설정하였다. 채널2을 사용하기 위해서 PWM_CTL 레지스터의 8번 비트(PWEN2)를 1로 설정하였고 M/S 모드로 PWM을 사용하기 위해서 PWM_CTL 레지스터의 15번 비트(MSEN2)를 1로 설정하였다. Pulse의 주기를 20ms로 맞추기 위해 div값을 기준으로 range를 320으로 설정하였다. SG90을 회전하기 위해 응용 프로그램으로부터 각도를 입력 받고 PWM_RNG1 레지스터 값인 320을 기준으로 SG90의 주기인 20ms에 대해 입력 받은 각도의 pulse 비율을 계산해준다. 계산된 각도를 PWM_DAT1 레지스터에 저장하게 되면 SG90이 응용 프로그램에서 입력 받은 각도만큼 회전하게 된다.

### iii. ir_dev.c & gas_dev.c

적외선 거리 감지 센서와 가스 센서를 제어하는 디바이스 드라이버는 내용이 동일하다. MCP3008과 SPI 통신을 하기 위해 먼저 GPFSEL0, GPFSEL1, SPI_CS, SPI_FIFO 레지스터를 사용하였다. ir_dev는 Channel 0, gas_dev는 Channel 1을 사용한다.

① ir_open() : gpio base와 spi base에 ioremap함수를 사용하여 메모리를 할당해 준 후, 사용할 레지스터들을 각각 volatile unsigned int*형으로 선언하였다. gpfsel0, 1을 사용하여 GPIO 8~11번 핀을 ALT0 function으로 select 해 주었다. SPI_CS레지스터를 clear해주어 초기화 시켜주고, spio CPOL과 CPHA 모드를 설정한 후, FIFO를 clear해주어 데이터를 주고 받을 준비를 해주었다.

| | Pull | ALT0 |
|---|---|---|
| GPIO7 | High | SPI0_CE1_N |
| GPIO8 | High | SPI0_CE0_N |
| GPIO9 | Low | SPI0_MISO |
| GPIO10 | Low | SPI0_MOSI |
| GPIO11 | Low | SPI0_SCLK |

**Figure 12 gpio PIN별 function**

② ir_release() : SPI 모드로 설정되었던 function들을 초기화 해준 후, iounmap 함수를 사용하여 gpio, spi base address에 할당해주었던 메모리를 해제한다.

③ ir_read() : DIN을 통해 보낼 데이터와 DOUT을 통해 받을 데이터를 unsigned char형으로 3바이트씩 배열로(spi_tData, spi_rData) 선언해 주었다. 그리고 tCount, rCount라는 변수를 선언하여 주고 받은 데이터의 바이트 수를 확인하는 용도로 사용하였다. 적외선 거리 감지 센서는 채널 0에 연결되어 있으므로 보낼 데이터 배열에 각각 1, (0x08)<<4, 0을 넣어준다. (이산화탄소 센서는 채널 1에 연결되어 있으므로 보낼 데이터 배열에 0, (0x09)<<4, 0을 넣어준다.) 데이터 전송을 시작하기 전 FIFO을 clear해준 후,

SPI_CS레지스터의 TA 비트를 1로 설정하여 transfer active 상태로 만든다.

```
while((tCount < 3) || (rCount < 3)){
    while((*spi_cs & (1<<18)) && (tCount < 3)){
        *spi_fifo = spi_tData[tCount];
        tCount++;
    }
    while((*spi_cs & (1<<17)) && (rCount < 3)){
        spi_rData[rCount] = *spi_fifo;
        rCount++;
    }
}
```

이제 전송이 시작되는데, 3바이트가 모두 보내지고 받아져야 하므로 tCount가 3보다 작거나 rCount가 3보다 작으면 계속 반복문을 돌린다. 반복문 안에서는 데이터를 보내는 반복문과 받는 반복문이 2개 있는데, 보내는 반복문은 먼저 SPI_CS 레지스터의 TXD 비트를 확인하여 TX FIFO에 1바이트라도 공간이 있어 전송할 수 있지는지와 tCount가 3보다 작아 아직 전송할 데이터가 남아있는지 확인하고 조건을 만족하면 spi_tData[tCount]의 값을 SPI_FIFO 레지스터에 보내고, tCount를 1 증가한다. 데이터를 받는 반복문도 마찬가지로, SPI_CS 레지스터의 RXD비트와 rCount를 통해 더 읽어들일 데이터가 있는지 확인한다. 조건을 만족하면 SPI_FIFO레지스터의 값을 spi_rData[rCount]에 저장하고 rCount값을 1 증가시킨다.

반복문이 종료되면, SPI_CS레지스터의 DONE을 확인하여 transfer가 완료된 상태인지 확인한다. 그 후, FIFO를 clear하고, TA비트를 0으로 설정한 후, copy_from_user 함수를 사용하여 사용자에게, spi_rData의 하위 10비트, 즉 ((spi_rData[1]&0x03)<<8)+spi_rData[2]를 int값으로 전달한다.

### iv. vmotor_dev.c

진동 모터 모듈을 제어하는 디바이스 드라이버이다. PWM 구현 내용은 서보모터와 동일하며, major number와 디바이스 이름을 통해서 구분해 주었으며, IOCTL함수 명령어로는 앱에서 받은 데이터만큼 모터를 진동시키는 IOW 형태의 IOCTL_CMD_ON, 진동 모터를 끄는 IO형태의 IOCTL_CMD_MOTOR_OFF cmd가 있다. PWM 채널 1번을 사용하기 때문에 PWM의 PWM_CTL 레지스터의 PWEN1, MSEN1 비트를 1로 설정해 주었다. 또한 적외선 거리 센서가 감지한 20~150cm에 상응하는 값을 출력하기 때문에 PWM_RNG1 레지스터에 130을 넣어주었고, 1cm 마다 출력이 달라지게 하였다. IOCTL에서 copy_from_user로 kbuf에 값을 받아와 PWM_DAT1 레지스터에 넣어주는 등의 다른 내용은 서보모터와 동일하다.

### v. buzzer_dev.c

부저를 제어하는 디바이스 드라이버이다.

1. int __init buzzer_init (void) : register_chrdev( ) 를 통해 디바이스 드라이버를 등록하고 초기화한다.

2. void __exit buzzer_exit(void) : unregister_chrdev( )를 통해 디바이스 드라이버를 종료한다.

3. long buzzer_ioctl : gpset0, gpclr0를 사용하여 buzzer를 on/off해준다. IOCTL_CMD_SET_BUZZER_ON과 IOCTL_CMD_SET_BUZZER_OFF cmd가 있다.

```
long buzzer_ioctl(struct file * filp, unsigned int cmd, unsigned long arg)
{
    int data;
    switch (cmd){
        case IOCTL_CMD_SET_BUZZER_ON:
            *gpset0 |= (1<<17);
            break;
        case IOCTL_CMD_SET_BUZZER_OFF :
            *gpclr0 |= (1<<17);
            break;
        default :
            printk(KERN_ALERT "ioctl : command error\n");
    }
    return 0;
}
```

## 나) 응용 프로그램

### i. 장애물을 감지하면 진동을 통해 알려주는 기능 (app1.c)

open함수를 사용하여 IR_DEV을 read only로 열고, VMOTOR_DEV을 write only로 열었다. 그 후 device driver가 잘 열렸는지 확인한다. 그 후, while()문을 계속 돌며 dev값을 읽어와 ir_data라는 변수에 저장하고 (ir_data/1024.0)*3.3의 전압으로의 변환식을 계산하여 volts 변수에 저장한다. 데이터 시트를 참고하고 여러 가지 변환식을 적용시켜 확인해 본 결과 volts값의 센티미터로의 변환값은 $cm = 61.681 * pow(volts, -1.133)$의 식이 제일 알맞은 거리 값을 산출해내는 것을 확인했다. 센서의 측정 범위를 넘어선 20 미만, 150 초과의 값이 나오면 아래 내용을 실행하지 않고 반복문을 continue한다. 20~150cm 거리에 가까울 수록 더 높은 전력을 진동 모터에 출력하기 위해서 값을 (int)(-1)*(cm-20)+130 식을 통해 변환하여 vmotor_data 변수에 저장한다. 진동 모터 디바이스 드라이버는 range가 130이므로 ioctl함수를 사용해 IOCTL_CMD_MOTOR_ON이라는 cmd를 주어 값을 전달한다. 종료되면 ioctl함수에 IOCTL_CMD_MOTOR_OFF cmd를 주어 진동모터를 OFF한다.

### ii. 방독면 내의 유독가스의 농도가 일정 수준을 초과하면 소리를 통해 알려주는 기능 (app2.c)

gas_dev.c와 buzzer_dev.c를 각각 open( ) 한 뒤, 반복적으로 read( ) 함수를 통해 SPI 통신의 결과를 읽어온다. 읽어 들인 결과 값을 voltage로 식 gas_volts = (gas_data/1024.0)*3.3을 이용해 변환한 후 datasheet의 공식을 참고해 (gas_volts*5000)/3의 식으로 계산된 값을 ppm에 저장한다. CO2 sensing은 여러가지 요인들에 의해 민감하게 반응하므로 오차 값을 줄이기 위해 여기서 구한 이 ppm을 바로 사용하지 않고 5회 단위로 ppm을 측정하여 평균 값을 구해 사용한다. CO2 농도에 따라 위험도가 다르고 농도가 높아질수록 방독면 방화 시간이 끝나 감을 뜻하므로 농도가 1000 ppm을 넘긴 경우 buzzer의 on/off사이에 sleep(2)를 주어 2초 간격의 반복 실행으로 경고 알림을 하고, 1300 ppm을 넘을 경우 sleep(1)을 주어 1초 간격의 반복 실행으로 경고 알림을 한다. 반복문이 종료되면 gas_dev.c와 buzzer_dev.c를 각각 디바이스를 close( ) 한다.

### iii. 방독면 내의 불쾌지수가 일정 수준을 초과하면 모터를 작동시켜 공기를 환기시켜주는 기능 (app3.c)

응용 프로그램에서는 polling방식을 사용하기 위해 while문을 반복한다. 가장 먼저 humidity_dev에 선언된 humidity_ioctl함수를 호출한다. IOCTL_CMD로 IOCTL_CMD_CHECK_HUMIDITY를 사용하였고 humidity_ioctl 함수를 통해 측정된 온도와 습도를 info 구조체에 저장한다. 온도와 습도의 소수점은 총 8비트이므로 0~127까지 표현이 가능하기 때문에 값이 100이상인 경우 소수점 3자리를 표현하기 위해 1000으로, 10이상 100미만인 경우 소수점 2자리를 표현하기 위해 100으로, 0~9인 경우 소수점 한자리를 표현하기 위해 10으로 나누어 저장하고 결과값을 정수 부분과 더하였다. 그리고 습도와 온도를 알 때 불쾌지수를 구하는 식을 통해 방독면 내부의 불쾌지수를 계산하였다. 임계 값 이상의 불쾌지수가 측정되는 경우 IOCTL_CMD로 IOCTL_CMD_ROTATE_MOTOR를 사용하여 motor_dev에 선언된 motor_ioctl함수를 호출하여 SG90 서보모터를 180도, -180도 회전시킨다. 불쾌지수가 75~80이면 50%의 사람이 불쾌감을 느끼고 80 이상일 경우 90%의 사람이 불쾌감을 느끼기 때문에 80% 이상의 불쾌지수일 경우에 대해 구현하려 하였지만 시연을 위한 측정 상황에 있어서 온도가 크게 변하지 않아 불쾌지수의 임계 값을 75로 설정하였다.

# 5. 시험평가

## 가) 프로그램 작동법

디바이스 드라이버를 구현하기 위해서 <linux/init.h>, <linux/kernel.h>, <linux/module.h>, <linux/uaccess.h>, <asm/uaccess.h>, <linux/fs.h>, <linux/slab.h>, <asm/mach/map.h>, <linux/delay.h> 라이브러리를 사용하였다.

응용프로그램을 구현하기 위해서는 <stdio.h>, <fcntl.h>, <unistd.h>, <stdlib.h>, <string.h>, <errno.h>, <sys/ioctl.h>, <sys/types.h>, <sys/sysmacros.h> 라이브러리를 사용하였고. App1에서는 pow함수를 사용하기 위해 <math.h> 라이브러리를 추가하였다.

디바이스 드라이버는 각 폴더마다 정의된 MakeFile에 따라 컴파일 된다. *_dev.ko 파일을 insmod를 통해 모듈을 커널에 올린다. mknod를 통해 디바이스 노드를 생성하고 응용 프로그램에 접근하게 된다. 응용 프로그램은 gcc를 통해 컴파일 하였고, app1.c의 경우 pow함수를 사용하기 위해 -lm option을 주어 컴파일 해야한다.

커널의 버전은 “4.19.66-v7+”이고, build을 위한 디렉토리의 path는 “/lib/modules/$(KERNEL_VER)/build이다.

모든 디바이스 드라이버를 컴파일 후 커널에 올리고 app을 모두 컴파일 하기 위해 source code 폴더 내에 proj_3.sh 파일을 정의해 두었다.

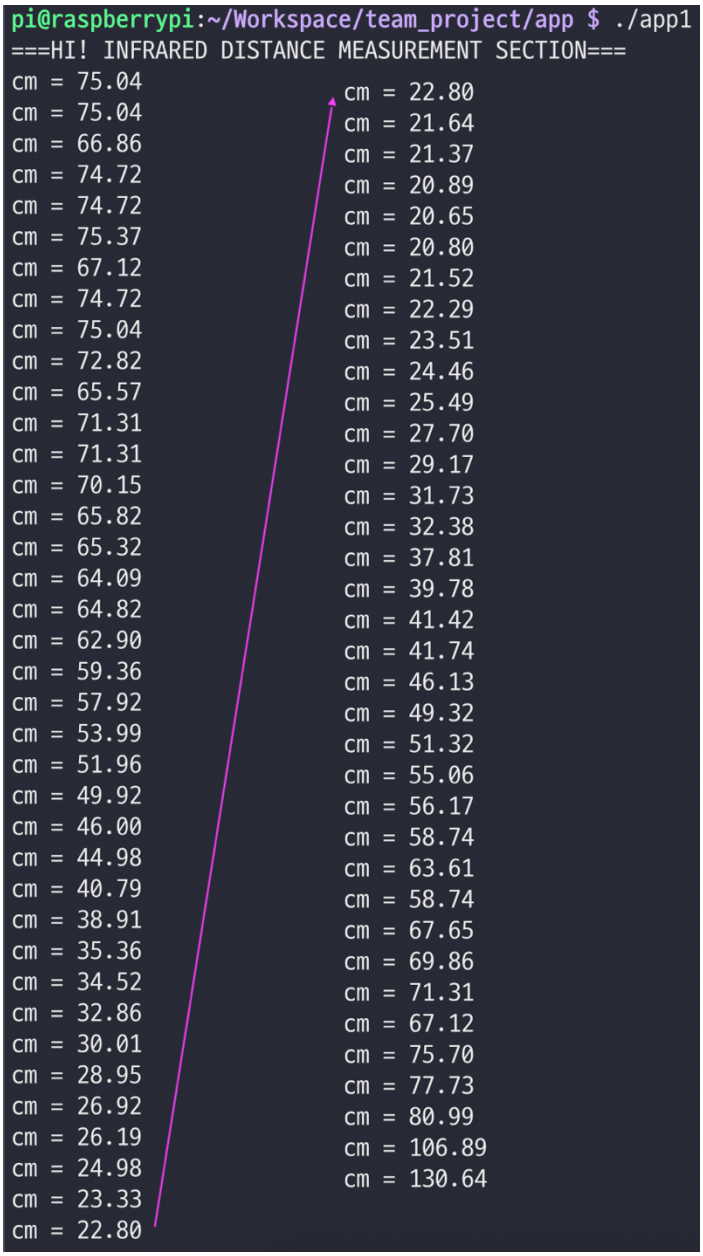
## 나) 테스트 결과

i. 방독면 내의 유독가스의 농도가 일정 수준을 초과하면 소리를 통해 알려주는 기능

```
pi@raspberrypi:~/Workspace/team_project/app $ ./app2
===HI! CO2 DETECTION SECTION===
ppm = 721
ppm = 720
ppm = 747
ppm = 741
ppm = 729
ppm = 729
ppm = 1172
ppm = 1354
ppm = 1353
ppm = 1413
```

보라색 선을 기준으로 이산화탄소에 입김을 불어 넣었을 때 평소에는 720~750ppm 사이의 값을 유지하다가 입김을 불어 넣으면 1100~1500ppm까지 값이 치솟는 결과를 관찰하였다. 1000ppm을 넘으면 2초 간격으로 부저가 울리고, 1300ppm을 넘자 1초 간격으로 5회 부저가 울림을 확인하였다.

ii. 장애물을 감지하면 진동을 통해 알려주는 기능




전방에 물체를 설치하고 센서를 장착한 구현물에 접근시켰다. 센서와 물체 사이의 거리가 20~150cm 사이가 되면 해당 거리 값을 출력하게 하였다. 거리에 반비례하여 진동 모터의 세기가 달라지는 것을 확인했다. 위의 사진은 물체를 75cm 떨어트린 상태에서 20cm 까지 가까이 했다가 다시 물체를 멀리 떨어트렸을 때 출력 값이다.

iii. 방독면 내의 불쾌지수가 일정 수준을 초과하면 모터를 작동시켜 공기를 환기시켜주는 기능

```
pi@raspberrypi:~/Workspace/team_project/app $ ./app3
humidity: 34.00% temperature: 28.20% discomfortIndex: 73.77
humidity: 32.00% temperature: 27.70% discomfortIndex: 72.94
humidity: 29.00% temperature: 27.90% discomfortIndex: 72.76
humidity: 28.00% temperature: 27.90% discomfortIndex: 72.63
humidity: 61.00% temperature: 27.90% discomfortIndex: 77.02
humidity: 95.00% temperature: 28.60% discomfortIndex: 82.78
humidity: 95.00% temperature: 28.40% discomfortIndex: 82.43
humidity: 77.00% temperature: 28.10% discomfortIndex: 79.47
humidity: 64.00% temperature: 28.30% discomfortIndex: 78.00
humidity: 57.00% temperature: 28.30% discomfortIndex: 77.04
humidity: 51.00% temperature: 28.10% discomfortIndex: 75.96
humidity: 46.00% temperature: 27.90% discomfortIndex: 75.03
humidity: 44.00% temperature: 28.40% discomfortIndex: 75.38
humidity: 41.00% temperature: 28.00% discomfortIndex: 74.48
humidity: 39.00% temperature: 28.00% discomfortIndex: 74.21
humidity: 35.00% temperature: 28.10% discomfortIndex: 73.79
humidity: 33.00% temperature: 27.90% discomfortIndex: 73.29
```

첫번째 보라색 선을 기준으로 위는 상온 상태이고 습도도 28~34%의 값을 유지하는 상태이다. 온습도 센서에 입김을 불어주면 습도가 95%까지 치솟고, 불쾌지수가 75이상이 되면 모터가 동작하는 것을 확인하였다. 센서 내의 습도가 다시 정상값으로 돌아오면 계산된 불쾌지수가 낮아지고 두 번째 보라색 선을 기준으로 아래부터 모터의 동작 또한 멈추는 것을 관찰하였다.

## 다) 미구현 사항 및 버그 사항

i. 제안서에 있는 모든 센서를 사용하여 구현하였고, 습도 센서 대신 온습도 센서를 사용하여 온+습도를 모두 측정하여 적절한 불쾌지수 값으로 변환해 모터의 세기를 조절하였다.

ii. 방독면 내의 이산화탄소 농도를 측정하여 수치에 따라 소리를 다르게 출력할 예정이었으나, 신청한 소리 모듈 대신 능동 부저를 지급받아서 이산화탄소 농도에 따라 소리의 간격을 다르게 출력하였다.

iii. 제안서에 적혀 있는 외부 공기 오염도 감지는 알고리즘이 방독면 내의 방호 가능 시간 알림과 동일하여 생략하였다.

iv. 방독면 내부 환기 기능은 지급받은 서보 모터가 연속으로 360° 회전이 되지 않기 때문에 불쾌지수가 떨어질 때 까지 -180°, 180° 회전하여 방독면 내의 공기를 순환시키도록 하였다.

# 6. Challenge Issue 및 소감

| | |
|---|---|
| 박수린 | 처음에는 LED와 버튼도 구현하기 어려워했고, 디바이스 드라이버를 만들어서 커널에 올리고 제거하는 방식도 이해하기 힘들었지만 프로젝트를 진행하면서 완벽하지는 않지만 점점 알게 되었다. 하지만, 아쉬우면서 힘들었던 점은 하드웨어를 제어하는 일이기 때문에 디버깅이 매우 어렵고, 실제로 알맞게 구동하고 있는 것인지 확인이 어렵다는 것이다. 물론, 조교님께 부탁해서 파형 분석기를 사용할 수도 있었지만 이유없이 센서가 값이 튀거나 작동 하지 않을 때는 속수무책으로 손 놓고 작동이 될 때까지 라즈베리파이를 재부팅 해보는 수 밖에 없어 매우 답답했다. 프로젝트 도중에 적외선 감지 센서가 말을 듣지 않아 포기하고 싶었던 순간도 있었지만, 다행히 잘 작동하였고 구현 난이도가 높은 SPI 통신을 성공해내어 매우 뿌듯 하였다. 하드웨어를 제어하는 프로젝트를 처음 접해보아 배경 지식이 매우 부족한 상태로 시작하였지만, 그만큼 배울 점이 많은 프로젝트 였다고 생각한다. |
| 정소희 | 우리가사용한 CO2 센서는 PWM 출력도 지원하여서 PWM 출력 값을 사용하여 농도를 계산하려 했으나 값을 읽어 들이는 과정에서 어려움을 느꼈다. 다행히도 SPI 통신을 잘 구현하여 analog 출력 값을 사용했지만, 프로젝트에 같이 사용된 적외선 센서와 다른 방식인 PWM 출력 값을 이용해 이산화탄소의 농도를 구했다면 더 다양한 방식의 디바이스 드라이버를 구현할 수 있었던 점에서 아쉬웠다.<br>개인적인 사유이지만 프로젝트 기간 중 입원을 하게 되어 팀원들과 만나는 시간이 제한적이었던 것이 아쉽다. 팀원 모두 프로젝트에 대한 적극성이 높았기 때문에 프로젝트 진행이 빨랐는데 제가 다치지않았다면 더 많이 모여 더 다양한 시도를 해볼 수 있었기 때문이다.<br>이번 프로젝트를 통해서 라즈베리파이를 처음 다뤄보고 디바이스 드라이버와 응용프로그램 사이의 관계, 그리고 센서들의 사용법 등을 이해하고 익숙해질 수 있어서 좋았다.<br>그리고 무엇보다도 팀 프로젝트의 순기능을 느낄 수 있었습니다. 팀원들의 배려와 응원이 큰 힘이 될 수 있고 서로 모여 고민하면서 다양한 구현 알고리즘이 탄생한다는 것을 느낄 수 있었던 프로젝트였다. |

| | |
|---|---|
| 김성규 | 이번 프로젝트를 진행하면서 가장 어려웠던 부분은 센서가 동작하는 원리를 이해하는 부분과 전송되는 데이터의 싱크를 맞추는 작업이었다. 센서가 동작하는 방법에 대한 자료를 검색해 보면 레지스터를 조작하기보단, 간편한 wiringPi 라이브러리를 사용하여 구현한 자료들 뿐이어서 datasheet을 보고 스스로 센서를 동작 시킬 방법을 터득해야 하는 부분에서 많은 어려움을 겪었다. 그리고 datasheet에 맞게 동작하도록 구현하더라도 실제 컴퓨터의 연산 속도가 빠르다 보니까 ms, us단위로 싱크를 조절해야 정확한 값을 입력 받을 수 있는데, 이 부분에서도 데이터의 싱크를 맞추는데 많은 어려움을 겪었다. 프로젝트를 진행함에 있어서 요청한 센서를 지급받지 못한 부분이 가장 아쉽지만, 그래도 kernel level에서 직접 레지스터를 조작하여 디바이스 드라이버를 구현해보고 디바이스의 작동 방법을 공부하면서 시스템적으로 더 깊이 알게 된 것 같아서 만족스럽다. |