**마이크로프로세서 설계 텀 프로젝트 최종 보고서**

<미세먼지 센서를 이용한 소형 공기청정기>

컴퓨터과학부 2014920044 이홍석

1. **서론**
   1. 배경

이 주제를 선택하게 된 이유는 최근 부쩍 늘어난 미세먼지 농도를 보고 과연 내가 평소에 생활하는 방과 같은 공간에 얼마나 많은 미세먼지가 있는지 궁금했고 또 간단하게 정화를 할 수 있는 기기가 있으면 좋겠다고 생각해서 이 주제를 선택하게 되었다. 집 전체 공기를 정화하는 용도로 사용하는 공기청정기가 있긴 하지만 매번 돌리려면 전기도 많이 사용하고 번거롭다. 간단하게 방안의 미세먼지를 측정하고 정화하는 기기가 있다면 전기도 조금 사용하면서 맑은 공기를 만끽할 수 있다.

* 1. 주제 : 미세먼지 센서를 이용한 소형 공기청정기

이 기기는 30초 간격으로 현재 공간의 미세먼지 농도를 측정하고 이에 따라 미세먼지가 많으면 팬의 속도를 빠르게, 미세먼지 농도가 낮으면 팬의 속도를 느리게 조절한다. 기기에 스위치를 하나 두어서 스위치를 끄면 미세먼지 농도를 측정하는 용도로만 사용할 수 있고, 스위치를 키면 자동 공기 정화기능도 사용할 수 있다. 외부 전원은 팬을 돌리기 위한 9V 건전지와 JKIT-128-1을 동작 시키기 위한 보조배터리로 이루어져 있다.

1. **본론**
   1. 기능
      1. 미세먼지 측정기능

30초 간격으로 현재 공간의 미세먼지 농도를 측정한다. LCD 디스플레이에 농도를 ug/m3단위로 표시한다. 그리고 현재 미세먼지 농도가 한국 미세먼지 농도 기준 중 어디에 속하는지도 표시한다.

* + 1. 자동 공기 정화기능

스위치를 키면 현재 농도에 따라 팬의 속도가 조절된다. 현재 미세먼지 농도가 낮으면 팬이 멈추고 높으면 팬이 빠른 속도로 회전한다.

* 1. 사용방법
     1. JKIT-128-1에 전원을 넣는다.
     2. 30초 정도 센서의 Initializing 시간을 기다린다.
     3. 자동 공기 정화기능을 사용하고 싶다면 스위치를 켠다.
     4. 미세먼지 농도 측정기능만 사용하고 싶다면 스위치를 끈다.
  2. 주요부품
     1. 미세먼지 센서(PPD42NS)

이 기기의 부품 중에서 가장 중요한 부품이다. 이 부품을 통해 미세먼지를 측정하여 팬의 속도를 조절한다. 공기중의 미세먼지 입자를 발견하면 High에서 Low로 신호를 바꾸고 발견되지 않으면 다시 Low에서 High로 바꾸는 센서이다. 기본 단위는 pcs/L이므로 보기 편한 단위인 ug/m3로 변환하는 과정이 필요하다.

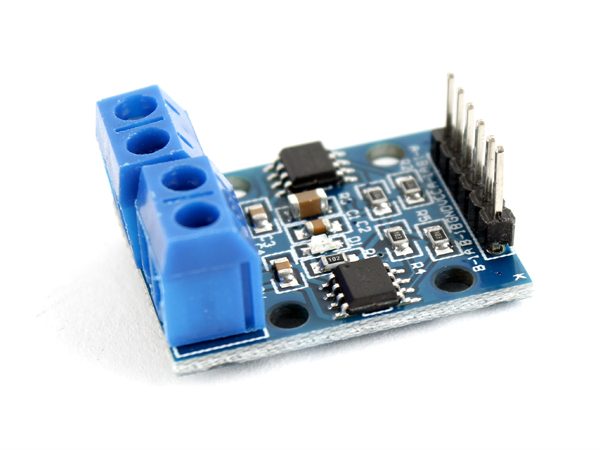
* + 1. LCD 패널(DLC01602B)

미세먼지 농도를 눈으로 확인하기 위해 사용한 부품이다. 처음에는 미세먼지 센서의 초기화가 필요하므로 Initializing…이라는 문구를 표시한다. 30초마다 측정된 미세먼지 농도를 첫번째 줄에 ug/m3단위로 표시하며 두번째 줄에 해당 농도가 어느정도 나쁜지 표시해준다. 나쁨의 정도는 기상청 미세먼지 기준을 사용하였으며 Good, Normal, Bad, Very bad 이렇게 4가지로 표시한다.

* + 1. 팬(BFL9025S24)

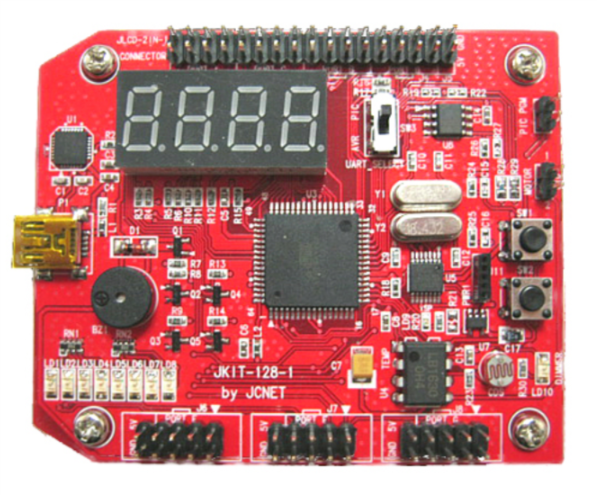
이번에 사용한 팬은 24V짜리 팬이다. 이 팬을 이용하여 공기를 빨아들이고 필터로 거른 후에 다시 내뿜는다. 기기에는 두개의 팬이 달려있고 하나는 기기내부로, 하나는 기기외부로 공기를 이동시킨다. 24V의 팬을 이용한 이유는 팬의 크기가 어느정도 커야 공기 정화기능이 높을 것 같다고 생각하였기 때문이다. 그런데 5V인 JKIT으로 24V짜리 팬 두개를 컨트롤하려고 해보니 잘 안되었다. 따라서 다른 수업시간에 사용한 모터드라이버를 활용하였다.

* + 1. 모터드라이버(L9110S)

팬의 설명대로 두개의 24V짜리 팬을 컨트롤 하기에는 5V는 턱없이 부족했다. 따라서 외부입력 전원으로 최대 12V까지 받는 모터드라이버와 9V 건전지를 이용하여 이를 해결하였다. 이 모터드라이버는 두개의 H-브릿지로 이루어져 있기 때문에 각 모터마다 두개의 핀이 있고 두개의 모터가 같은 속도로 움직이면 되므로 브래드보드를 이용하여 각 H-브릿지의 핀 하나씩을 JKIT의 모터2번핀에 연결하였다. 5V 입력을 통해 모터들을 제어할 수 있다.

* + 1. JKIT-128-1

JKIT-128-1 보드를 사용하였으며 다른 보드는 사용하지 않았다. Port는 PORTA, PORTC, 모터를 사용하였으며 5V, GND핀을 사용하였다.

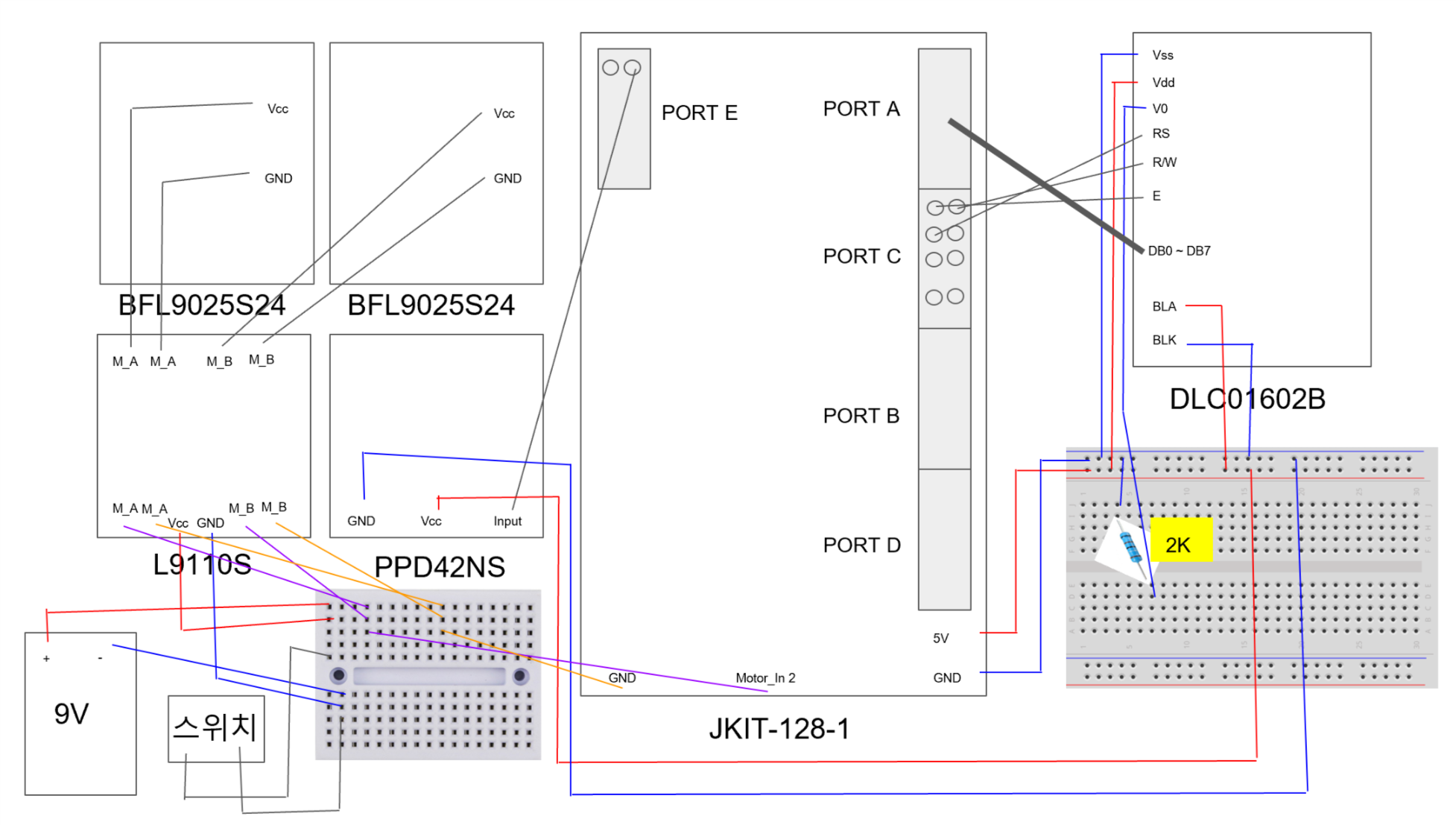


* 1. 부품견적

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **제품명** | **단가(원)** | **수량** | **합계(원)** | **상품코드** | **링크** |
| 1 | 1/4W 1% 저항 14종 키트 [NT-KIT-R015] | 1800 | 1 | 1800 | 1381629 | <http://www.devicemart.co.kr/1381629> |
| 2 | 방진필터-2091 (2EA/1봉지) | 2670 | 2 | 5340 | 1139195 | <http://www.devicemart.co.kr/1139195> |
| 3 | 테스트[CH254]소켓 점퍼 케이블 40P (칼라) (M/M) 10cm | 700 | 1 | 700 | 1328409 | <http://www.devicemart.co.kr/1328409> |
| 4 | 테스트[CH254]소켓 점퍼 케이블 40P (칼라) (F/F) 10cm | 700 | 1 | 700 | 1328410 | <http://www.devicemart.co.kr/1328410> |
| 5 | 브레드보드 400핀 Half Size Breadboard [SZH-BBAD-005] | 1000 | 1 | 1000 | 1328148 | <http://www.devicemart.co.kr/1328148> |
| 6 | BFL9025S24 2000RPM | 2850 | 2 | 5700 | 1360130 | <http://www.devicemart.co.kr/1360130> |
| 7 | 1602 16x2 캐릭터 LCD 디스플레이 모듈 [DLC01602B] | 6000 | 1 | 6000 | 1384302 | <http://www.devicemart.co.kr/1384302> |
| 8 | 먼지센서 Grove - Dust Sensor（PPD42NS）[NT101020012] | 16800 | 1 | 16800 | 1066618 | <http://www.devicemart.co.kr/1066618> |
| 9 | 미니 브레드보드 |  | 1 |  |  | 개인소유 |
| 10 | Dual H-Bridge L9110s 모터 드라이버 모듈 [SZH-MDBL-002] | 1500 | 1 | 1500 | 1324245 | <http://www.devicemart.co.kr/1324245> |
| 11 | 9V 건전지 홀더 |  | 1 |  |  | 개인소유 |
| 12 | 9V 건전지 |  | 1 |  |  | 개인소유 |
| 13 | 토글 스위치 |  | 1 |  |  | 개인소유 |
| 14 | 보조배터리 |  | 1 |  |  | 개인소유 |

* 1. 회로도

다음은 USB전원과 보조배터리를 제외한 회로도이다.



* 1. 소스코드
     1. 헤더파일 추가 및 정의 부분

|  |
| --- |
| 1. #include<avr/io.h> 2. #include<avr/interrupt.h> //인터럽트 헤더 파일 3. #define *F\_CPU* 16000000UL 4. #include<util/delay.h> //딜레이 헤더 파일 5. #define LCD\_WDATA PORTA //LCD데이터 포트 정의 6. #define LCD\_WINST PORTA // 7. #define LCD\_CTRL PORTC //LCD 제어 포트 정의 8. #define LCD\_EN 0 //LCD 제어(PING0~2)를 효과적으로 하기위한 정의 9. #define LCD\_RW 1 10. #define LCD\_RS 2 11. #define Byte unsigned char |

Interrupt와 delay를 사용하기 위해 4~6번 라인을 이용하였다. 8번부터 14번 라인은 LCD를 사용하기 위한 정의부분이다. PORTA를 데이터포트, PORTC의 0,1,2번을 각각 EN, RW, RS로 사용하였다. 14번은 많이 쓰는 자료형을 쉽게 사용하기 위해 추가하였다.

ii. LCD 함수들 및 나만의 딜레이 함수

|  |
| --- |
| 1. void delay\_us(int n){ 2. for(int i = 0 ; i < n ; i++); 3. } 4. void delay\_ms(int n){ 5. for(volatile int i = 0 ; i < n ; i++) 6. for(int j = 0 ; j < 1000 ; j++); 7. } 8. void LCD\_Data(Byte ch){ 9. delay\_ms(2); 10. LCD\_WDATA = ch; //데이터 출력 11. LCD\_CTRL |= (1 << LCD\_RS); //RS=1, R/W=0 으로 데이터 쓰기 싸이클 12. LCD\_CTRL &= ~(1 << LCD\_RW); 13. LCD\_CTRL |= (1 << LCD\_EN); //LCD 사용 14. LCD\_CTRL &= ~(1 << LCD\_EN); //LCD 사용안함 15. } 16. // 0 1 2 17. void LCD\_Comm(Byte ch){ //PG0-EN , PG1-RW, PG2-RS , PG4-TOSC1핀(사용안함) 18. //LCD\_CTRL = LCD제어부 포트(4핀인데 실질적인 사용3핀) 19. delay\_ms(2); 20. LCD\_WINST = ch; //명령어 쓰기 21. LCD\_CTRL &= ~(1 << LCD\_RS); // RS=0, RW=0 으로 정의함. 22. LCD\_CTRL &= ~(1 << LCD\_RW); 23. LCD\_CTRL |= (1 << LCD\_EN); //LCD 사용허가 24. LCD\_CTRL &= ~(1 << LCD\_EN); //LCD 사용안함 25. } 26. void LCD\_CHAR(Byte c){ //한문자 출력 함수 27. LCD\_Data(c); //CGROM 문자코드의 0x31 ~ 0xFF 는 아스키코드와 일치함! 28. delay\_ms(2); 29. } 30. void LCD\_STR(Byte \*str){ //↑문자열을 한문자씩 출력함수로 전달 31. while(\*str !=0){ 32. LCD\_CHAR(\*str); 33. str++; 34. } 35. } 36. void LCD\_pos(Byte col, Byte row){ //LCD 포지션 설정 37. LCD\_Comm(0x80 | (row+col\*0x40)); // row=행 / col=열 ,DDRAM주소설정 38. } 39. void LCD\_Clear(void){ // 화면 클리어 40. LCD\_Comm(0x01); 41. delay\_ms(2); //1.6ms이상의 실행시간소요로 딜레이필요 42. } 43. void LCD\_Init(void){ //LCD 초기화 44. DDRA = 0xff; //PORTA(LCD연결포트)를 출력으로 설정 45. DDRC = 0xff; //PORTG(LCD컨트롤)의 하위 4비트를 출력으로설정 46. LCD\_Comm(0x38); //데이터 8비트 사용, 5X7도트 , LCD2열로 사용(6) 47. delay\_ms(2); 48. LCD\_Comm(0x0f); //Display ON/OFF 49. delay\_ms(2); 50. LCD\_Comm(0x06); //주소 +1 , 커서를 우측으로 이동 (3) 51. delay\_ms(2); 52. LCD\_Clear(); 53. } |

LCD는 타이밍에 따라 state이 달라지는데 이것은 데이터 시트를 참조 해야 한다. 하지만 타이밍이 생각보다 잘 맞아서 적당한 딜레이만 주어도 잘 동작한다. 각 함수에 대한 설명은 주석에 적어 두었다.

iii. 필요한 전역변수

|  |
| --- |
| volatile long pcs = 0; // 측정된 시간의 합산  volatile Byte str1[30] = ""; // 출력할 문자열  volatile unsigned int cont = 0; // 정수부  volatile unsigned int flt = 0; // 소수부  const int time\_ms = 30000; // 30초간 측정  volatile unsigned int pulse\_state=0; // 0 : Up  // 1 : Down  volatile long particle\_counter=0; |

위의 전역변수들은 아래의 함수들과 main함수를 위한 변수들이다.

iv. 인터럽트 처리 함수 및 모터함수

|  |
| --- |
| ISR(INT6\_vect){  if(pulse\_state == 0){ // Down Event  pulse\_state = 1;  particle\_counter = 0; // 1당 100us단위  }  else{ // Up Event  pulse\_state = 0;  pcs += particle\_counter\*10; // ms단위의 시간을 합산  }  }  ISR(TIMER0\_OVF\_vect){ // 한 사이클당 100us=0.1ms  if(pulse\_state == 1) // 측정 상태이면 counter를 증가  particle\_counter++;  TCNT0 = 206;  }  volatile int motor\_count=0;  ISR(TIMER2\_COMP\_vect){  motor\_count++;  }  void motor(int sec, int speed\_val){ // 모터동작함수  OCR2 = speed\_val; // 속도 조절  TCNT2 = 0;  while(motor\_count != 1000\*sec); // 30초 동작  motor\_count = 0;  } |

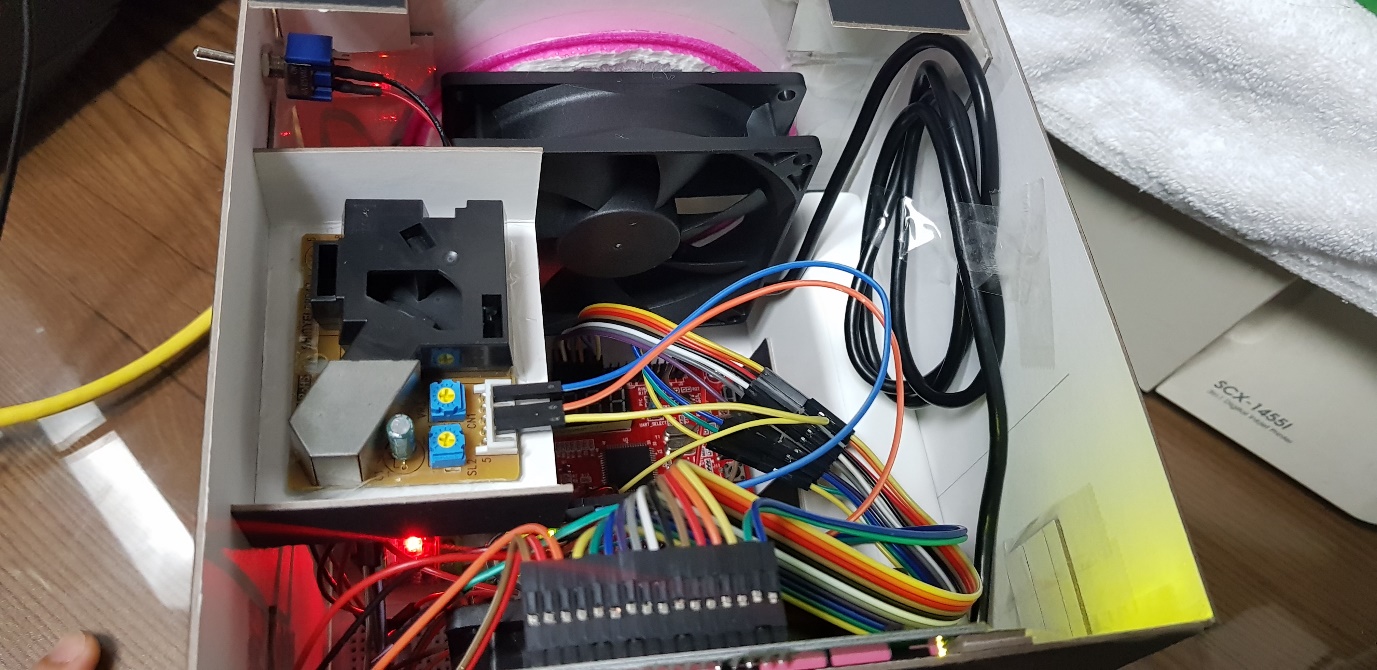
먼저 미세먼지 센서를 위한 인터럽트 함수가 있고 그 다음은 100us를 세는 타이머 인터럽트, 그리고 그 밑에는 모터를 위한 타이머 인터럽트가 있다. 각 함수의 설명은 주석에 써두었다.

v. main 함수

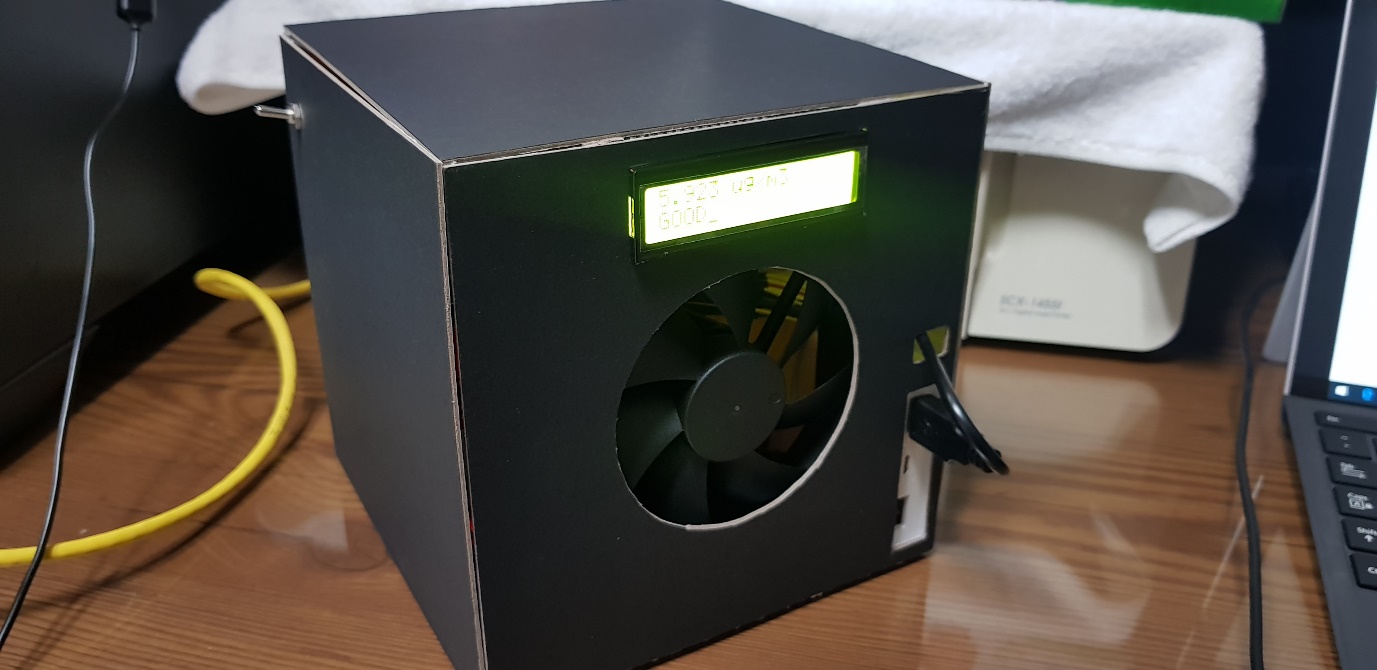
|  |
| --- |
| int main(){  DDRE = 0xbf; // E6번 Input  EIMSK = 0x40;  EICRB = 0x30;// 논리적 변화시 인터럽트  TCCR0 = 0x03;  TIMSK = 0x81;  TCNT0 = 206; // 100us  TCCR2 = 0x7B;  DDRB = 0xff;  sei();    int motor\_time=time\_ms/1000;  LCD\_Init(); //LCD 초기화  delay\_ms(20);  LCD\_pos(0,0); //LCD 포지션 0행 1열 지정  delay\_ms(20);  *sprintf*(str1,"Initialize...");  LCD\_STR(str1); //문자열 str을 LCD출력  delay\_ms(20);    while(1){  delay\_ms(time\_ms);  float ratio = pcs\*1.0/(time\_ms\*10.0);  pcs=0;  //단위변환(pcs/L->ug/m3)  float concentration = (1.1\*ratio\*ratio\*ratio  -3.8\*ratio\*ratio+520\*ratio+0.62)/130;  cont = (unsigned int)concentration;  flt = (unsigned int)((concentration-cont)\*1000);  *sprintf*(str1,"%d.%d ug/m3",cont,flt); //출력    LCD\_Init(); //LCD 초기화  delay\_ms(20);  LCD\_pos(0,0); //LCD 포지션 0행 1열 지정  delay\_ms(20);  LCD\_STR(str1); //문자열 str을 LCD출력  delay\_ms(20);  LCD\_pos(1,0); //LCD 포지션 1행 1열 지정  delay\_ms(20);    unsigned char str2[10]; //기상청 기준 상태 저장  if(cont<16){  *sprintf*(str2,"GOOD");  LCD\_STR(str2); //문자열 str을 LCD출력  delay\_ms(20);  motor(motor\_time,255);    }  else if(cont<36){  *sprintf*(str2,"NORMAL");  LCD\_STR(str2); //문자열 str을 LCD출력  delay\_ms(20);  motor(motor\_time,120);  }  else if(cont<76){  *sprintf*(str2,"BAD");  LCD\_STR(str2); //문자열 str을 LCD출력  delay\_ms(20);  motor(motor\_time,80);  }  else{  *sprintf*(str2,"VERY BAD");  LCD\_STR(str2); //문자열 str을 LCD출력  delay\_ms(20);  motor(motor\_time,0);  }  }  } |

메인 함수에는 각종 레지스터에 비트를 할당하였다. 위의 인터럽트들을 위한 레지스터에 비트를 할당하였다.

1. 결과물
   * 1. 내부



* + 1. 외부



1. **결론**
   1. 고찰

이번에 이 장치를 만들면서 이런 간단한 장치도 설계하고 만드는 것은 상당히 어렵다는 것을 알게 되었다. 익숙하지 않은 자료형 size와 센서들, 레지스터들, 함수들을 이용하면서 기존에 내가 쓰던 컴퓨터가 얼마나 편한 것인지를 알게 되었다. 마이크로프로세서는 간단한 장치도 설계하기 무척 어렵고 회로도 동시에 잘 설계해야 한다. 이번 장치를 만들면서 생겼던 어려운 점들과 아쉬운 점에 대해 고찰해본다. 이번에 이 장치를 설계하면서 처음에는 24v 팬이 5v 여러 개로 돌아갈 수 있을 것이라고 생각했다. 그러나 막상 해보니 그렇게 간단하게 전압이 승압되지 않았다. 다양한 방법을 고민하던 중 다른 과목시간에 쓰던 모터 드라이버가 생각났다. 이것을 이용하면 최대 12V를 사용할 수 있었다. 그래서 9V와 3V를 직렬로 연결해서 12V를 만든 후 저항을 넣어서 회로를 구성해보았다. 그런데 타는 냄새가 났고 성급히 저항을 뺐는데 저항이 매우 뜨거웠다. 이를 통해 높은 전압을 다루는 것에 대해 두려워졌고 9V만 사용하게 되었다. 9V만 사용해도 모터 드라이버를 통해 두개의 팬을 돌릴 수 있는 것을 확인하고 추가적인 것은 필요 없다고 생각하였다. 확실히 9V를 이용하니 팬의 속도가 좀 느렸다. 하지만 선풍기가 아니고 공기 정화 용도이기 때문에 바람이 셀 필요가 없었다. 또 어려운 점은 LCD를 사용할 때였다. 아무리 코드를 수정하고 타이밍을 맞추어도 LCD화면에는 아무 글자도 표시되지 않았다. 한참을 찾아보니 Contrast를 VCC에 연결하여 생긴 문제였다. GND로 바꿔 끼니 정상적으로 동작하였다. 이번에 이것을 만들면서 아쉬운 점은 팬의 속도가 낮은 점이다. 너무 낮아서 필터를 거치면 바람이 거의 안 느껴진다. 팬을 12V짜리로 사용하던가 아니면 24V 외부전력에 트랜지스터를 이용했으면 더 좋았을 것이다. 회로에 대해 아직 많이 미숙하므로 섣불리 더 큰 전압을 시험해 볼 수 없었다. 만약 적절한 전압과 전류를 조절할 수 있다면 더욱 완벽했을 것이다. 또 아쉬운 점은 케이스이다. 케이스를 좀더 계획적으로 만들었으면 더 이쁘고 효율적으로 만들 수 있었을 것인데 좀 급하게 만드느라 완성도가 그리 높지 않았다. 이번 기회에 아쉬운 점도 많았지만 다 만들고 나니 뿌듯했다. 나중에 좀더 보완하여 완벽하게 만들고 싶다.