마이크로프로세서설계및실습

프로젝트 최종 보고서

미세먼지센서를 이용한
자동 창문 개폐기

서울시립대학교
컴퓨터과학부
11조
2016920020 서예찬

1. 팀명단 및 역할 소개
구상 및 개발 서예찬

2. 텀프로젝트 주제 및 기능 소개
 저희 조에서 이번에 정한 텀프로젝트의 주제는 미세먼지센서를 이용한 자동 창문 개폐기 입니다.
 간단한 기능을 설명드리면,
1. 미세먼지센서를 이용해 측정한 미세먼지 데이터를 기반으로 일정한 값을 넘게되면, 미세먼지가 많은 상태를 가정하여 FND에 Bad를 출력합니다. 반대로 일정 값을 넘지 못하면, 미세먼지가 적은 상태를 가정하여 FND에 Good을 출력합니다.
2. 스위치를 통해 Mode를 변경할 수 있습니다. 스위치를 통해 신호 입력시 FND에 잠깐동안 Open -> Calc -> Shut -> Open의 순서로 변화하는 모드의 현재값이 출력됩니다.
3. 스위치를 통해 Mode를 변경할 경우, 각 모드에 맞게 Servo motor가 동작하게 됩니다.
- Open : 미세먼지의 양과 관계 없이 창문을 연 상태로 고정합니다.
- Calc : 미세먼지의 상태가 좋지 않으면 창문을 닫고, 미세먼지의 상태가 좋으면 창문을 엽니다.
- Shut : 미세먼지의 양과 관계 없이 창문을 닫은 상태로 고정합니다.

3. 텀프로젝트 진행 내용 요약
미세먼지센서와 Jkit-128-1 보드의 통신을 위해 USART1 통신을 사용하였으며 미세먼지센서의 데이터처리 구조를 파악하기 위해 다음 게시글의 코드를 참조하였습니다.

PMS7003 먼지센서 데이터 처리 코드 참고
https://blog.naver.com/dhtpals32123/221453434840

해당 센서의 신호와 스위치 신호를 처리하기 위해 인터럽트를 사용하였습니다.

상기한 두 신호를 받아 보드에서 정보를 처리하여 각각의 상황에 맞는 FND 출력을 내는 함수 display_fnd() 함수를 구현하였고
servo motor를 동작하는 출력을 내는 함수 setMotorPosition()을 구현하였습니다.

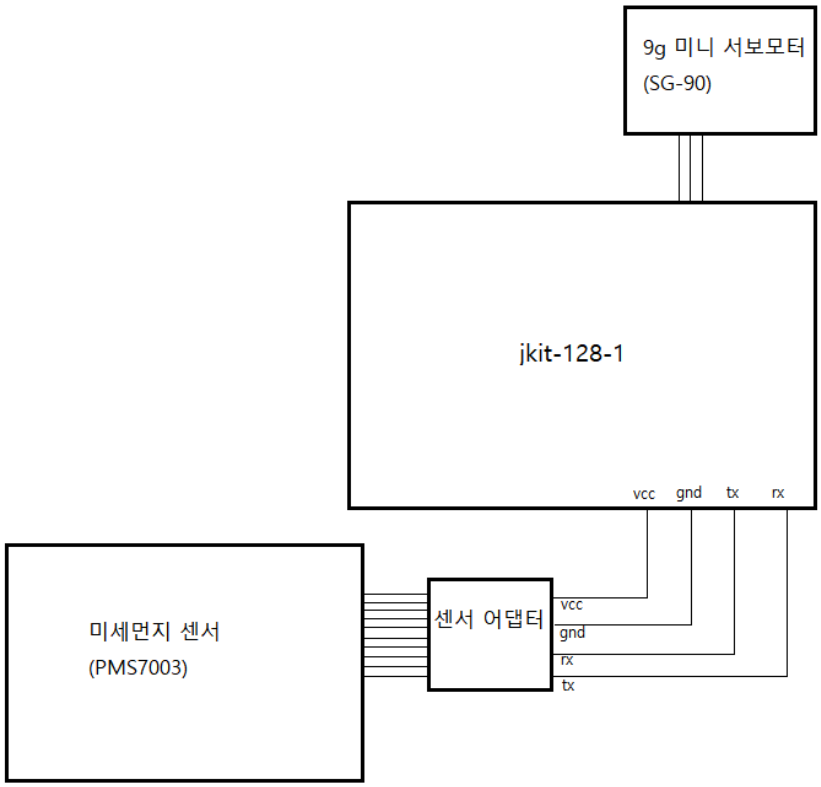
4. 추가 사용 부품 및 회로도

창문 모형을 만들기 위해 사용한 재료를 제외하고 추가적으로 사용한 것은 없습니다.

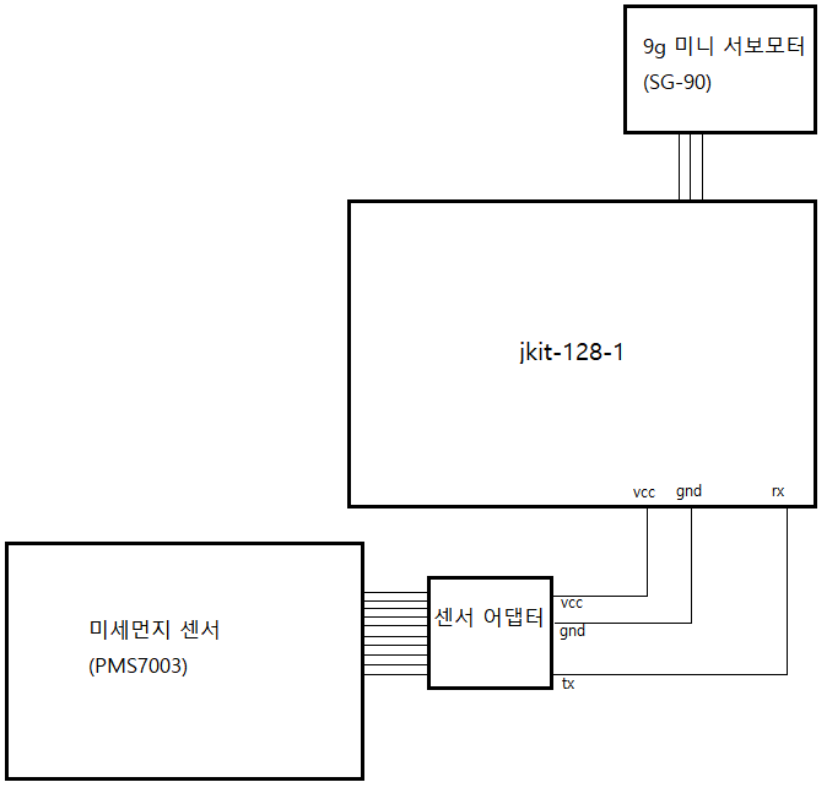
기존에 제출한 제안서에 있던 부품 리스트는 다음과 같습니다.

NO.	제품명	상품코드	단가	수량	합계
1	먼지 검출 센서 모듈 SY-PMS7003	1328954	20790	1	20790
2	PMS7003 어댑터 보드 SY-PMS7003ab	12234058	4400	1	4400
3	9g 미니 서보모터 SG-90	1128421	1650	1	1650
4	소켓 점퍼 케이블 40P (칼라) (M/F) 20cm	1321195	935	1	935
5	소켓 점퍼 케이블 40P (칼라) (F/F) 20cm	1321192	935	1	935

기존에 제출한 제안서에 있던 회로도는 다음과 같습니다.



변경된 회로도는 다음과 같습니다.
(센서와의 통신에서 보드->센서 방향의 통신은 불필요하여 제거)



5. 소스코드

```
/*
 * TermProject.c
 *
 * Created: 2021-12-20 오후 4:34:25
 * Author : 서예찬
 */
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include <stdio.h>
#include<avr/interrupt.h>

#define ON 1
#define OFF 0
#define OPEN 0
#define SHUT 2
#define CALC 1
#define GOOD 0
#define BAD 1

#define CHECK_START 1
#define START 2
#define FRAME_RATE 1
#define PMS_25 11 // PM2.5의 먼지에 대한 데이터가 담길 index
#define PMS_10 13 // PM10의 먼지에 대한 데이터가 담길 index

#define THRESHOLD 20 // 임계값. 미세먼지가 이 이상의 수치를 가지면 미세먼지가 많은것으로 판단.

char RX;
unsigned buttonmode = OPEN;
int timemarker = 0; // 버튼을 누를 시 잠시동안 button mode를 표기하기 위한 시간 변수
unsigned aircondition = GOOD;

unsigned char alpha[12] = {0x3f,0x73,0x79,0x37,0x39,0x77,0x38,0x39,0x6D,0x76,0x3E,0x31}; // O,P,E,N,C,A,L,C,S,H,U,T
unsigned char beta[8] = {0x7D,0x5C,0x5C,0x5E,0x00,0x7C,0x77,0x5E}; //
```

```
G,O,O,D,NULL,B,A,D
unsigned char fnd_sel[4] = {0x01,0x02,0x04,0x08};

struct air { //공기 중 미세먼지 상태 정보가 담긴 구조체
        unsigned mode;
        int data[50];
        int checksum;
        int count;

};

struct air PMS;


ISR(USART1_RX_vect){
        RX=UDR1;
        if(PMS.mode==START){ //3. 입력받기 시작
                PMS.data[PMS.count]=RX; //값 넣기

                if(PMS.count==28){ //4. 데이터 26바이트 + 체크썸 2바이트
                        PMS.mode=OFF;
                        if((int)PMS.data[PMS_10]>THRESHOLD){ // 미세먼지 농도가
높음
                                aircondition = BAD;
                        }
                        else{ // 미세먼지 농도가 낮음
                                aircondition = GOOD;
                        }
                }

                PMS.count++;
        }

        else{
                if(PMS.mode==CHECK_START){ //첫 데이터가 0x42였다면 다음 데이
터를 체크
                        if(RX==0X4D){ //2. 2번째 바이트가 4D면 입력받기 시작
                                PMS.mode=START; // 입력시작
                                PMS.count=0;
                        }
```

```
                        else PMS.mode=OFF;
                }
                if(RX==0X42) //1. 스타트 데이터 체크. 다음 바이트가 4D면 데이터 입
력받기 시작
                {
                        PMS.mode=CHECK_START;
                }
        }
}

ISR(INT4_vect) //switch1
{
        buttonmode++;
        buttonmode = buttonmode % 3;
        timemarker = 100; // 버튼을 누를 시 잠시동안 버튼모드를 표시할 시간.
        _delay_ms(10); // debouncing
}

void display_fnd()
{
        int i;
        if(timemarker > 0){ // 버튼모드를 표기. OPEN, CALC, SHUT의 표기가 존재.
                for (i = 0; i<4; i++)
                {
                        PORTC = alpha[buttonmode*4 + 3 - i];
                        PORTG = fnd_sel[i];
                        _delay_ms(2);
                }
                timemarker--; // 버튼모드를 표기할때마다 감소하며 0이되면 아래
else문 실행
        }
        else{ // GOOD/BAD를 표기
                for (i = 0; i<4; i++)
                {
                        PORTC = beta[aircondition * 4 + 3 - i];
                        PORTG = fnd_sel[i];
                        _delay_ms(2);
                }
        }
}
```

```
void setMotorPosition(){
        if(buttonmode == OPEN){
                OCR1A=625;      //90도
        }
        else if(buttonmode == SHUT){
                OCR1A=375;      //0도
        }
        else if(buttonmode == CALC){
                if(aircondition == GOOD){
                        OCR1A=625;      //90도
                }
                else{
                        OCR1A=375;      //0도
                }
        }
}

int main(void){
        DDRG=0XFF;
        UCSR1B=0x98; UBRR1L=103; // 9600 rx interrupt

        DDRB|=0x20;
        DDRC = 0xff; // C포트는 FND 데이터 신호
        DDRG = 0x0f; // G포트는 FND 선택신호
        DDRE = 0xcf;
        EICRB = 0x0a;
        EIMSK = 0x30; //interrupt enable

        PORTB|=0x20;
        TCCR1A=0x82;
        TCCR1B=0x1b;
        ICR1=4999;      //TOP
        OCR1A=375;       //0도
        PMS.mode=0;
        SREG=0x80; // 인터럽트 활성화
        while (1){
                display_fnd();
                setMotorPosition();
        }
```

```
}
```

6. 결과 및 고찰

실내에서도 충분히 결과를 보이기 위해 코드 내의 threshold의 값을 매우 작게 설정하였습니다.. 다른 기기 혹은 환경에서 동작하지 않을 경우 해당 코드 내의 threshold 값의 조정이 필요할 것입니다.

제안서에 제시한 목표기능 전부를 구현하였습니다.

만약 제안서와 다르게 조금 더 기능을 추가한다면, 현재 코드 내의 서보 모터의 각도를 반대로 설정한 서보모터 1대를 추가하여 좌우 창문을 여닫는 여닫이 창문으로도 수정이 가능할 것으로 보입니다.

또한 기존에 제안서에 적지 않아 구현하지 않았지만, 사용한 스위치는 한 개입니다. 남은 하나의 스위치를 사용하여 미세먼지로 측정한 데이터를 직접 FND로 출력하는 상태 추가 등의 기능 수정도 가능해보입니다.

(구현한 FND 출력 상태는 미세먼지 Good/Bad 출력,
스위치를 입력하여 현재 Mode 출력 이 2가지 입니다.
2번째 스위치를 입력하여, 측정되는 구체적인 데이터 출력 또한 충분히 가능했을 것으로 보입니다.)