

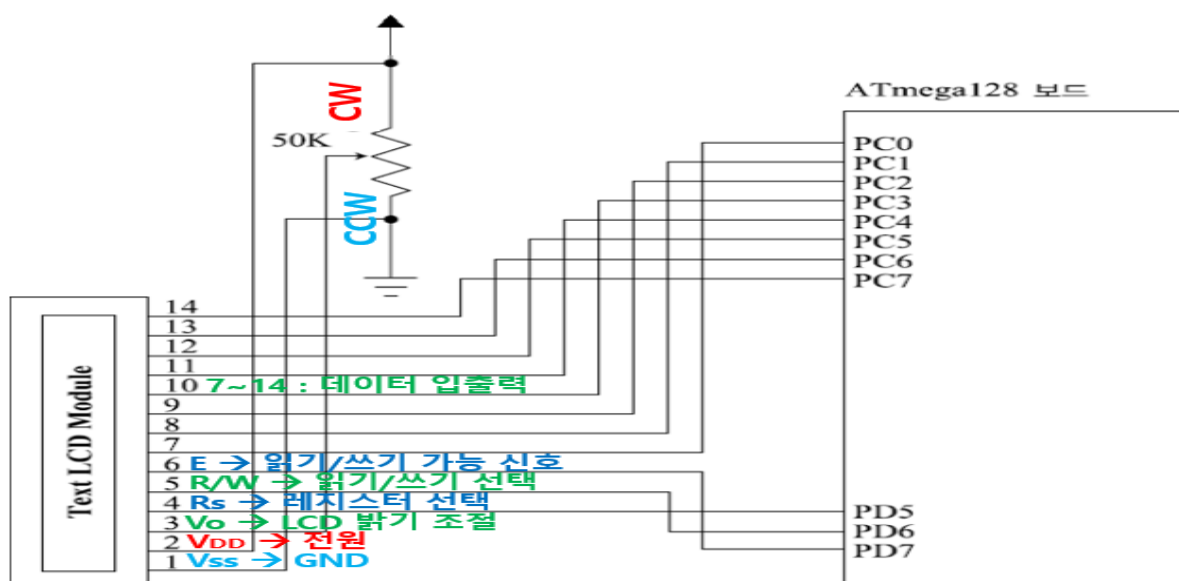
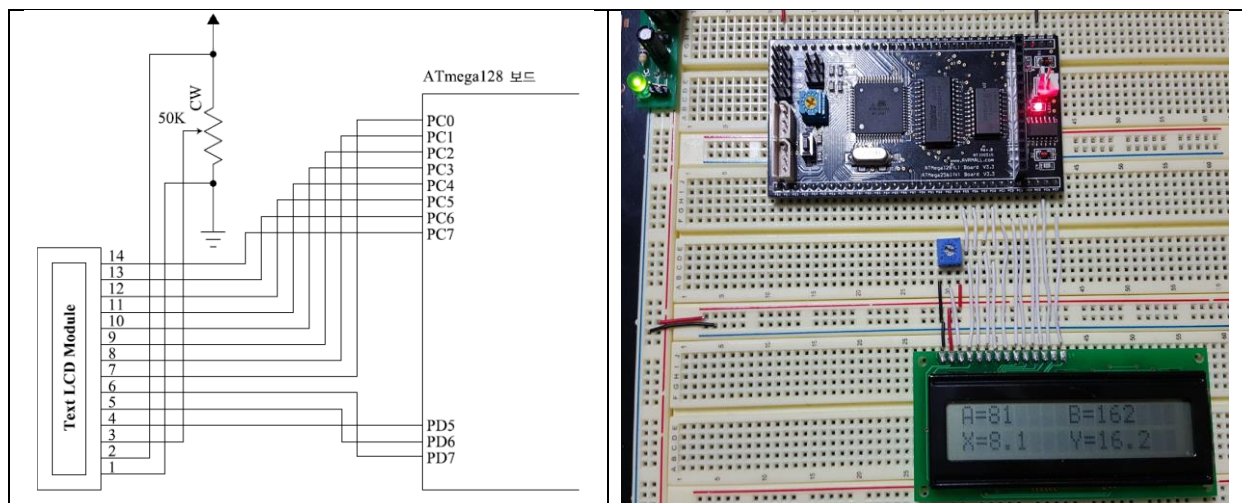
실험 5. LCD 디스플레이와 float 출력

전자공학과 21611591 김 난 희

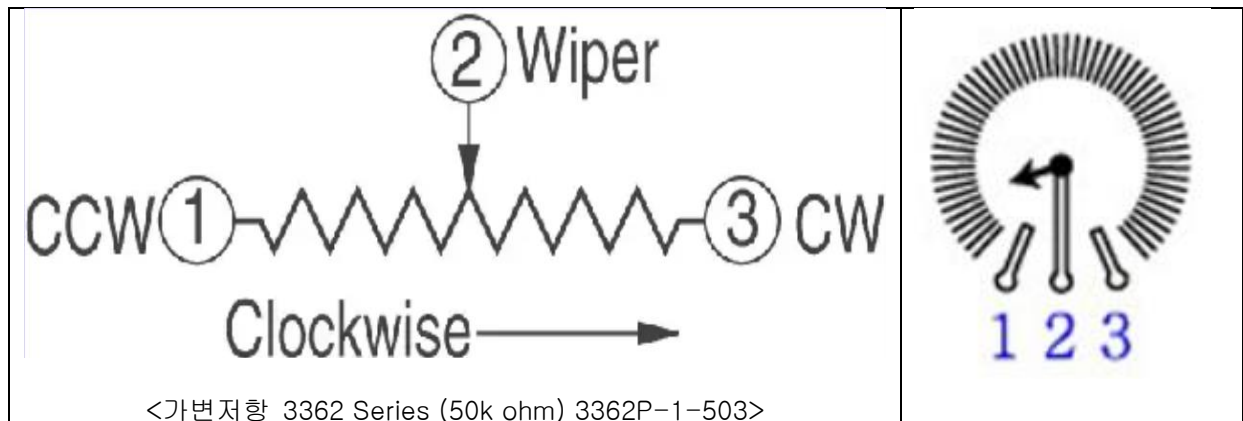
실험 목적

1. 헤더 파일과 소스 파일을 적절히 추가할 줄 안다.
2. 가변 저항의 역할을 이해하고 사용할 줄 안다.
3. 소수점 LCD 디스플레이를 위해 설정하는 링커 옵션을 이해한다.
4. 작성한 프로그램을 이해하고, 추가적으로 LCD 이해를 목적으로 다양한 실험을 해본다.

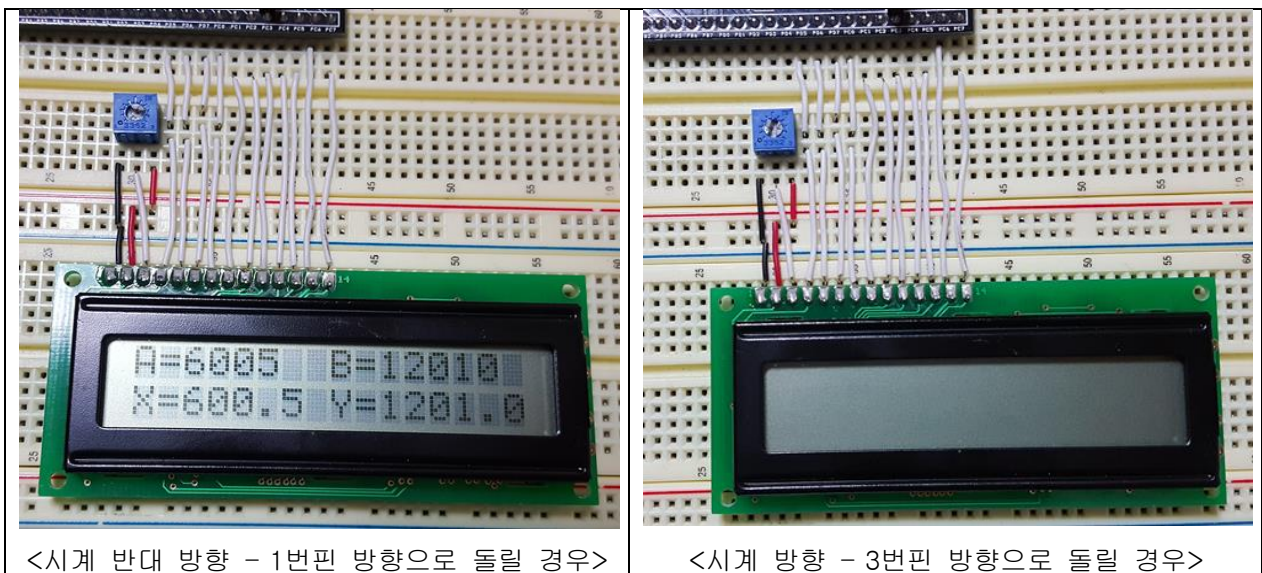
1. 회로 분석과 구동



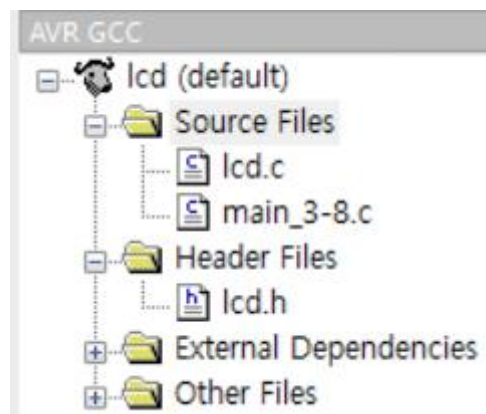
위 그림에서, LCD 옆에 적어놓은 것은 각각의 LCD 단자의 역할이다.



회로에서 사용된 가변 저항은 3362 시리즈의 50k옴이다. 가변저항의 다리는 3개가 있다. 1번핀과 3번핀의 저항 값을 측정해보면 50k옴 고정 저항치가 측정된다. 나사를 돌려 2번 핀이 가리키는 값을 조정하는데, 예를 들어 1번 핀과 2번 핀 사이에 저항을 측정하여 30k옴의 저항이 측정된다면, 2번 핀에서 3번 핀 사이에는 20k옴의 저항이 측정되는 것이다.



Clockwise는 시계방향임을 뜻하고, 실험 회로에서 3번핀 쪽으로 돌릴수록 LCD의 밝기가 점점 약해짐을 확인했다. 1번 핀으로 돌릴수록 저항의 크기가 작아져 LCD의 밝기가 진해졌다.



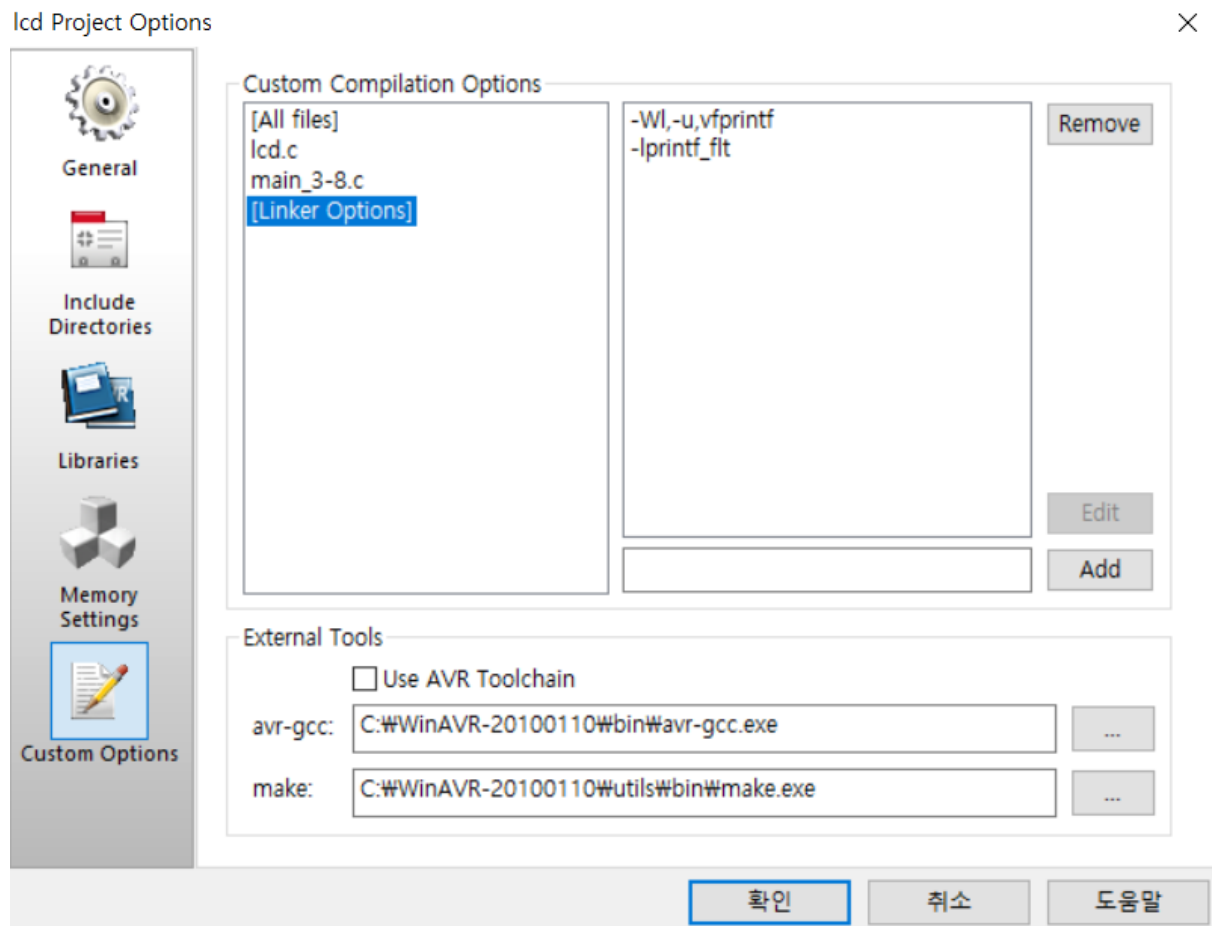
오른쪽 그림과 같이 lcd에 대한 c파일, main문에 대한 c파일, lcd에 대한 헤더파일을 추가해준다.

그리고 다음과 같이 링커 옵션을 추가해준다.
이는 lcd에서 소수점 출력을 위한 옵션이다.

`-Wl,-u,vfprintf -lprintf_flt`

띄워 쓰기를 하여서는 안되고,
따로 구글에서 찾아보고 알게 되었던
라이브러리를 추가해주는 것도 오류가 난다.

링커 옵션만 추가해주면 된다.



```

main.c

#include <stdio.h> // standard input output 헤더파일
#include <avr/io.h> // avr의 input output 헤더파일

// #define F_CPU 8000000 // avr의 clock을 8MHz 설정, 미리 옵션으로 16MHz 설정함
#include <util/delay.h> // delay 구문 사용하기 위함
#include "lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언

// 전역변수 선언 // main문 밖
long a = 0, b = 0; // long 형으로
double x = 0., y = 0.;

int main(void) //메인문 선언 //void로 인자가 없음을 표시
{
    // 지역 변수 선언 // 함수 내부 선언
    char lcd_string[2][MAX_LCD_STRING];

    LCD_init(); // LCD 초기화 함수 선언

    while(1){ // 무한 루프
        a = a + 1; // 1 더한 것을 자기 자신에다 넣어줌
        b = b + 2;
    }
}

```

```

        x += 0.1;
        y += 0.2;

        sprintf(lcd_string[0], "A=%-5ld B=%-5ld", a, b);
        // lcd_string[0]에 " "의 내용을 넣음
        LCD_str_write(0, 0, lcd_string[0]); // lcd에 출력, 첫번째 줄, 첫번째 칸이 0,0

        sprintf(lcd_string[1], "X=%-5.1f Y=%-5.1f ", x, y);
        LCD_str_write(1, 0, lcd_string[1]);
        _delay_ms(1000); // 1000ms = 1s 기다림 // 완전한 디스플레이를 위한 기다림
    }
    return 0; // 종료하며 마침
}

```

lcd.c	
#include	<avr/io.h> // avr의 input output 헤더파일
#include	"lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언
#define RS	PD5 // LCD 문자디스플레이에 연결된 포트D 의 핀번호 // RS는 명령인지 DATA 인지 결정
#define RW	PD6 // 포트D의 6번 핀에 RW 핀 연결
#define E	PD7 // Enable 연결
void	gen_E_strobe(void) // 사용할 수 있게 Enable 해주는 함수
{	volatile int i; // volatile 변수로, 사이즈 최적화를 피함.
	PORTD = 1<<E; // E 신호를 High로
	for(i=0; i<10; i++); // E 스트로브 신호를 일정기간 High로 유지
	PORTD &= ~(1<<E); // E 신호를 Low로
}	
void	wait_BusyFlag(void) // busy flag를 읽어 0이 될 때까지 기다림
{	volatile int i; // volatile 변수로, 사이즈 최적화를 피함.
	unsigned char bf; // buffer를 의미하는 unsigned 형 변수 선언
	DDRC = 0x0; // 포트C를 입력핀으로 설정
	PORTD = (PORTD & ~(1<<RS)) 1<<RW; // RS <- Low, RW <- High
	do{ // 먼저 실행 후 while()의 조건을 보고 후 판단
	PORTD = 1<<E; // E 신호를 High로
	for(i=0; i<10; i++); // E 스트로브 신호를 일정기간 High로 유지
	bf = PINC & 1<<PC7; // busy flag 읽어 냄
	PORTD &= ~(1<<E); // E 신호를 Low로
	}while(bf); // bf 값이 0이 아니면 busy, 0 일 때까지 반복
}	
void	LCD_command(unsigned char data) // lcd에 주는 명령 함수, 명령어 보냄
{	wait_BusyFlag(); // busy flag가 0될 때까지 대기
	DDRC = 0xFF; // 포트C를 출력핀으로 설정
	PORTC = data; // data 출력
	PORTD &= ~(1<<RS 1<<RW); // RS <- 0, RW <-0

```

        gen_E_strobe();           // E 스트로브 신호 만들기
    }

void LCD_data_write(unsigned char data) // 여기 data는 아스키 코드가 들어갈 것.
{
    wait_BusyFlag(); // busy 체크
    DDRC = 0xFF; // 출력으로 설정
    PORTC = data; // data의 값이 여기 들어옴
    PORTD = (PORTD | 1<<RS) & ~(1<<RW); // RS <- 1, RW <- 0
    // 데이터 내보냄 //명령어 x
    gen_E_strobe(); // E 스트로브 신호 만들기
}

void LCD_init(void) // lcd 초기화 함수
{
    DDRD |= 1<<RS | 1<<RW | 1<<E;           // RS, RW, E 핀을 출력핀으로 설정

    PORTD &= ~(1<<RS | 1<<E | 1<<RW); // 초기에 RS, E, RW <- 0

    LCD_command(0x3C); // 인터페이스, 디스플레이 설정
    LCD_command(0x02); // cursor 초기화
    LCD_command(0x01); // clear display
    LCD_command(0x06); // entry 모드
    LCD_command(0x0F); // display on/off
}

```

LCD 사용 순서는 다음과 같이 진행된다. 위의 LCD_init(void) 와도 매칭해보며 비교해보면 된다.

LCD 초기화 순서

1. 초기화 전 30msec 를 기다림
2. Function set 명령(0011xx00B)을 내보냄
3. Display On/Off control 명령(00001xxxB) 을 내보냄
4. Entry Mode set 명령(000001xxB)을 내보냄
5. DD RAM 어드레스를 내보냄
6. 표시할 문자데이터를 내보냄
7. 5,6과정을 반복

Function set에는 다음과 같은 것들이 초기에 설정된다.

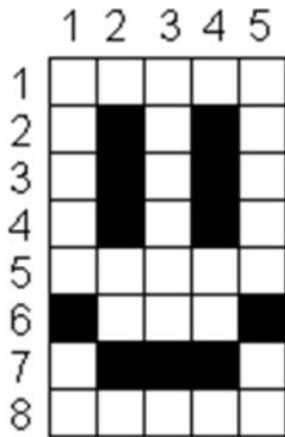
인터페이스/ 디스플레이 설정	DL	1 : 8비트 인터페이스 0 : 4비트 인터페이스
	N	1 : 두 줄 표시 0 : 한 줄 표시
	F	1 : 문자 5×10 도트 0 : 문자 5×7 도트

font를 보면 우리는 5x10 도트를 사용한다.

이런 것들이 어떻게 디스플레이되는 지 살펴보면, 다음 그림과 같다.

A Custom 5x8 Pixel Character:

Image Coding:



Binary Coding:

	0b000 00000
→	0b000 01010
	0b000 01010
→	0b000 01010
	0b000 00000
→	0b000 10001
	0b000 01110
	0b000 00000

1 = Black, 0 = White

위 그림은 5x8 도트를 예시로 나타낸 것이다. 검은색으로 디스플레이할 모양을 다음과 같이 1로 설정해주어 보이게 한다.

```
void set_cursor(unsigned int row, unsigned int col) // 커서 위치 설정
{
```

```
    LCD_command(0x80 + (row % 2) * 0x40 + (col % 0x40));
```

```
} // 0x80은 DDRAM 주소이고 row를 설정해주는 부분과 column을 설정하는 부분으로 되어있다.
```

만약 위의 함수에 set_cursor(0,0)을 넣어주면, 1000 0000이 들어가, 0x80 command를 실행한다. 만약 (1,2)를 넣어주면, 0x80 + 1x(0x40) + 2 이므로,

```
= 1000 0000
   0100 0000
   0000 0010
```

1100 0010 = C2 가 된다.

노란색으로 표시해 놓은 부분은 DDRAM 주소 부분이다.

결국 아래의 함수를 통해 위 set_cursor 내부의 Command 함수는 data 주소를 날리는 부분이라 이해할 수 있겠다.

// 함수 정의 : row, col 위치에서 문자열 str 을 LCD에 출력시킨다.

```
void LCD_str_write(unsigned int row, unsigned int col, char *str)
{
```

```
    int i;
```

```
    set_cursor(row, col); // 위 함수 인자로부터 받은 곳에 Cursor 위치 조정, DDRAM 주소
```

```
    for(i=0; (i+col < MAX_LCD_STRING) && (str[i] != '\0'); i++) // \0은 문자 끝을 말함
```

```
        LCD_data_write(str[i]); // data를 날림
```

```
}
```

lcd.h

```
#ifndef __LCD_H__
```

```

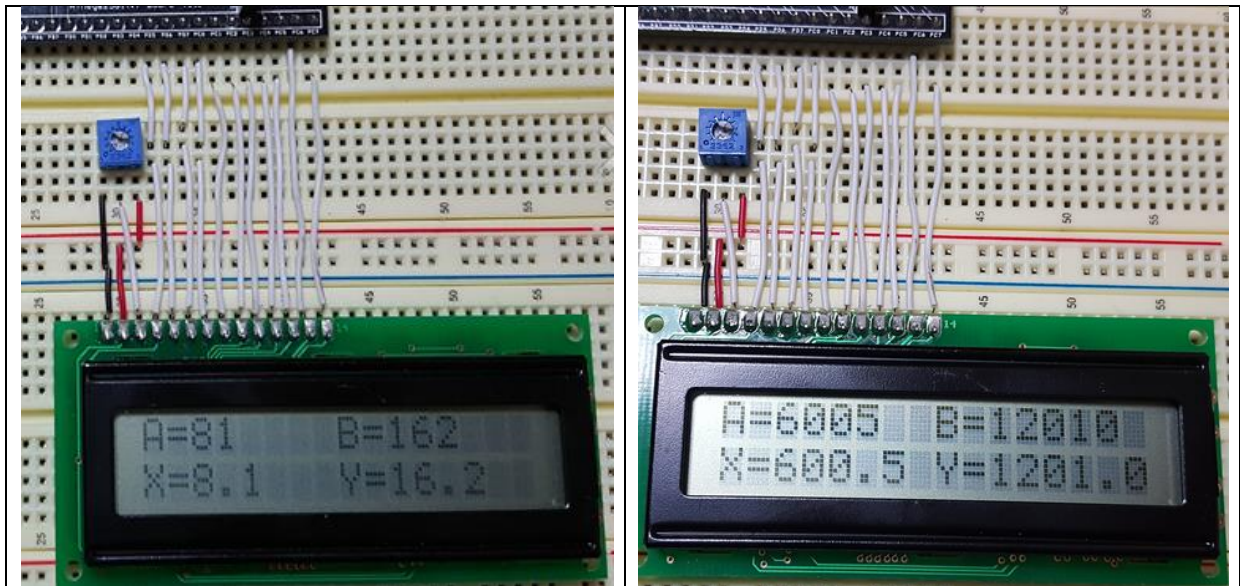
#define __LCD_H_

#define MAX_LCD_STRING      0x40 // 한 줄에 64글자 최대 디스플레이

// 아래는 외부에서도 함수를 쓸 수 있게 extern 으로 선언해 놓은 부분.
extern void      gen_E_strobe(void);
extern void      wait_BusyFlag(void);
extern void      LCD_command(unsigned char data);
extern void      LCD_data_write(unsigned char data);
extern void      LCD_init(void);
extern void      set_cursor(unsigned int row, unsigned int col);
extern void      LCD_str_write(unsigned int row, unsigned int col, char *str);
#endif

```

2. 실험 결과



처음에 x=?, y=? 로 떴으나, 링커 옵션을 통해 해결하였다. 링커 옵션은 위에 설명해 놓았다. 시간이 흐름에 따라 숫자가 점점 증가하는 것을 볼 수 있었다.

위의 lcd처럼 출력되는 이유

```

long    a = 0, b = 0;
double  x = 0., y = 0.;

```

```

a = a + 1;
b = b + 2;

```

전역으로 선언하여 a와 b는 1혹은 2씩 증가하는 형태,
x와 y는 0.1 혹은 0.2씩 증가하는 형태

```

x += 0.1;
y += 0.2;

```

다음의 x+= 0.1은 x=x+0.1과 같다.

2. 추가 실험

(1) 영문 이름 출력하기

2-1. 실험 회로 → 본 실험과 회로가 동일합니다.

2-2. 프로그램 분석 및 결과

소스 코드는 아래 main.c 파일의 main()에서 다음의 부분만 수정하였습니다.

```
int main(void)
{
    char lcd_string[2][MAX_LCD_STRING];

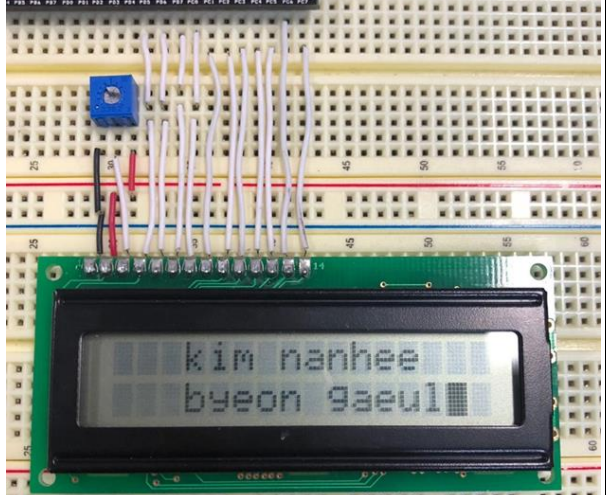
    LCD_init();

    while(1){
        a = a + 1;
        b = b + 2;

        x += 0.1;
        y += 0.2;

        sprintf(lcd_string[0], "A=%-5ld B=%-5ld", a, b);
        LCD_str_write(0, 0, lcd_string[0]);


        sprintf(lcd_string[1], "X=%-5.1f Y=%-5.1f  ", x, y);
        LCD_str_write(1, 0, lcd_string[1]);
        _delay_ms(1000);
    }
    return 0;
}
```

수정된 소스 코드	실험 결과
<pre>sprintf(lcd_string[0], "kim nanhee"); LCD_str_write(0, 3, lcd_string[0]); sprintf(lcd_string[1], "byeon gaeul"); LCD_str_write(1, 3, lcd_string[1]);</pre>	

lcd에 출력할 것을 결정하는 부분을 이름으로 바꾸어 주었고,
LCD_str_write(,3,)으로 바꾸어서 3칸 띄어 출력하였다.

다음과 같이 소스 코드를 설정해주어도 같은 결과가 나오는 것을 확인할 수 있

었다.

수정된 소스 코드	실험 결과
<pre>sprintf(lcd_string[0], " kim nanhee"); LCD_str_write(0, 0, lcd_string[0]); sprintf(lcd_string[1], " byeon gaeul"); LCD_str_write(1, 0, lcd_string[1]);</pre>	

첫번째 칸에 출력하는 대신에, 실제 출력할 글자를 3칸 띄워 lcd_string[0]에 넣어주었다.

2. 추가 실험

(2) 커서 이동하기

2-1. 실험 회로 → 본 실험과 회로가 동일합니다.

2-2. 프로그램 분석 및 결과

소스 코드는 아래 main.c 파일의 main()에서 다음의 부분만 수정하였습니다.

```
int main(void)  
{  
    char lcd_string[2][MAX_LCD_STRING];  
    LCD_init();  
  
    while(1){  
        a = a + 1;  
        b = b + 2;  
  
        x += 0.1;  
        y += 0.2;  
  
        sprintf(lcd_string[0], "A=%-5ld B=%-5ld", a, b);  
        LCD_str_write(0, 0, lcd_string[0]);  
  
        sprintf(lcd_string[1], "X=%-5.1f Y=%-5.1f  ", x, y);  
        LCD_str_write(1, 0, lcd_string[1]);  
        _delay_ms(1000);  
    }  
    return 0;  
}
```

수정된 소스 코드

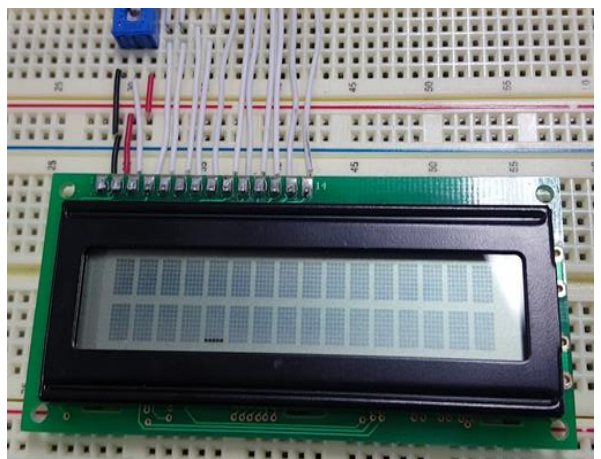
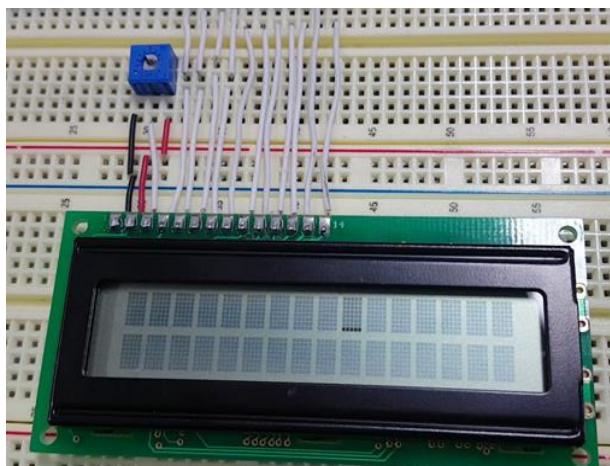
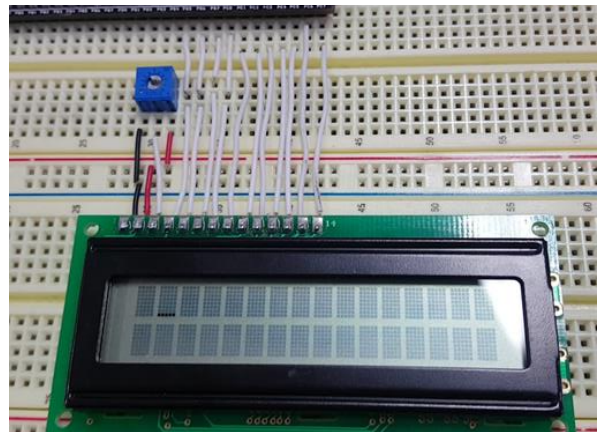
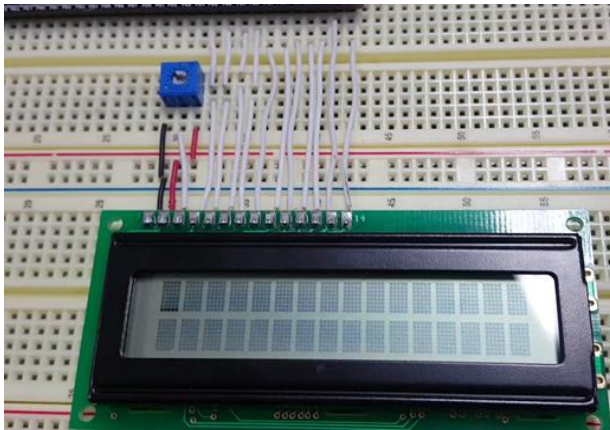
```

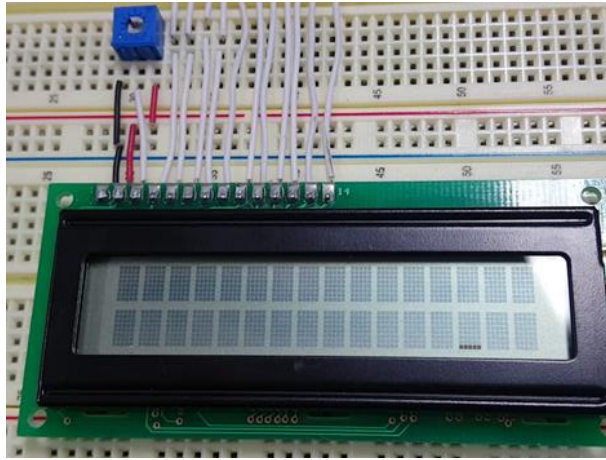
while(1){
    for(int cursor=0; cursor<32;cursor++)
    {
        sprintf(lcd_string[0], "");
        if(cursor>15) LCD_str_write(1, cursor-16, lcd_string[0]);
        else LCD_str_write(0, cursor, lcd_string[0]);

        _delay_ms(1000);
    }
}

```

시간이 흐름에 따라 변화하는 결과 사진(자세한 결과는 데모 영상에 있습니다.)





cursor가 증가함에 따라 첫번째 줄, 첫번째 칸에서 출발하여 두번째 줄의 마지막 칸까지 이동하는 것을 관찰할 수 있었다.

lcd_string[0]에는 “”를 넣어주었다. 커서의 이동을 관찰하기 위함이다.

if문에는 cursor 변수가 16 이상이 되면 아래줄에 출력하도록 하였다. cursor-16을 넣어준 것은 16이상부터 뺄셈을 해주면 0부터 시작하기 때문이다. 0부터 시작하면, LCD_str_write(unsigned int row, unsigned int col, char *str)에는 0부터 15까지 들어가게 된다.

<p>cursor = 16 → cursor-16=16-16= 0 : 시작 칸 cursor = 31 → cursor-16=31-16= 15 : 마지막 칸</p>
--

else문에는 16미만일 경우로, LCD_str_write(unsigned int row, unsigned int col, char *str)에 첫번째 인자가 0이 들어갔으므로, lcd의 첫번째 줄부터 출력하기 시작한다. cursor도 마찬가지로 첫번째 줄 첫번째 칸인 0부터 마지막 칸인15까지 이동한다.

다음과 같이 소스 코드를 설정해주면 커서의 이동이 두번째 줄 마지막 칸에서 첫번째 줄 첫번째 칸으로 이동하는 것을 확인할 수 있었다.

수정된 소스 코드

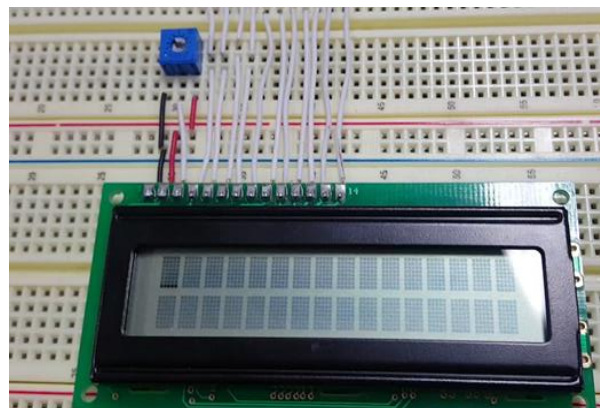
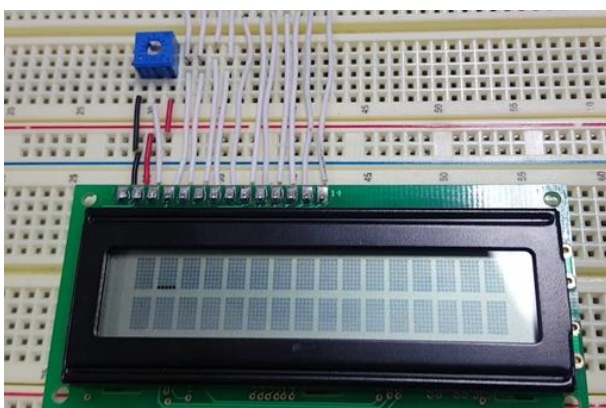
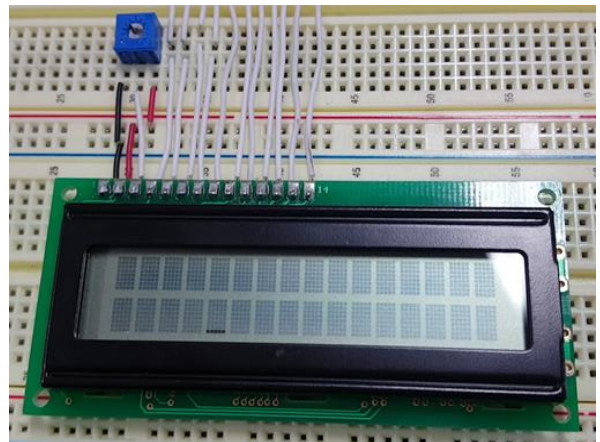
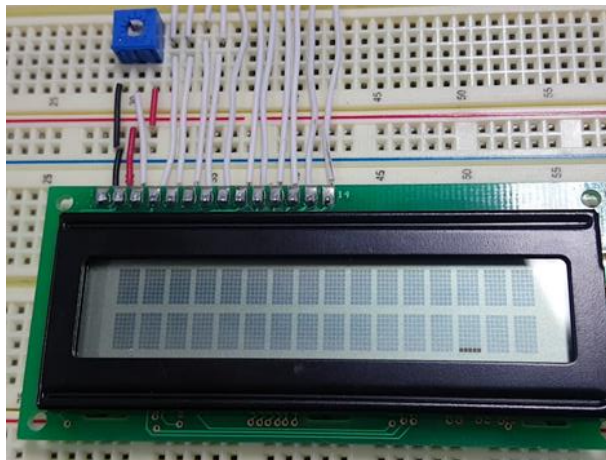
```

while(1){
    for(int cursor=31; cursor>=0;cursor--)
    {
        sprintf(lcd_string[0], "");
        if(cursor>15) LCD_str_write(1, cursor-16, lcd_string[0]);
        else LCD_str_write(0, cursor, lcd_string[0]);

        _delay_ms(1000);
    }
}

```

시간이 흐름에 따라 변화하는 결과 사진



위에서 실험한 커서 이동의 반대로 향하는 경우이다.

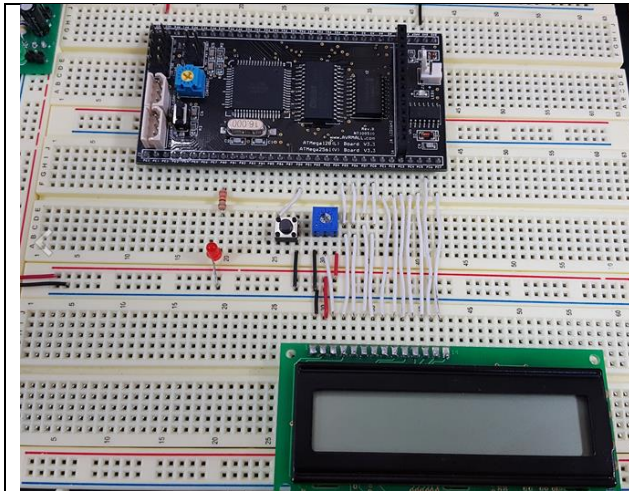
아래의 실험에서는 LCD.C 파일에 있는 함수, LCD_data_write(unsigned char

data)를 이용하여 글자를 출력해볼 것이다.

2. 추가 실험

(3) 스위치를 누르면 한 글자씩 출력하기

2-1. 실험 회로



PORT D의 PD0핀에 내부 풀업이 설정된 스위치를 달아주었다.

PORT B의 PB0핀에는 330옴 저항과 직렬 연결된 다이오드는 달아주었다.

내부 풀업이 설정된 스위치를 누르면 LOW로 인식한다. 스위치가 눌려지면 인터럽트가 작동한다. 인터럽트에서 state변수를 이용하여 LED와 LCD를 다르게 구동한다. LED는 스위치를 누를 때마다 불이 켜지며, LCD는 스위치를 누르면 커서를 한 칸 이동하여 그 자리에 출력될 글자를 한 글자씩 출력한다. <실험 1 LED ON/OFF>의 추가실험의 회로를 추가하였다.

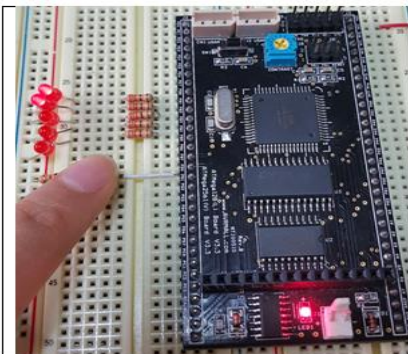
2-2. 프로그램 분석 및 결과

소스 코드는 첫번째 했던 <실험 1 LED ON/OFF>의 추가 실험을 본 실험의 회로에서 추가한 코드이다.

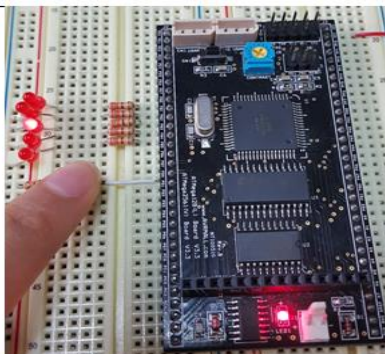
2. 추가실험

인터럽트를 사용한 LED 켜기

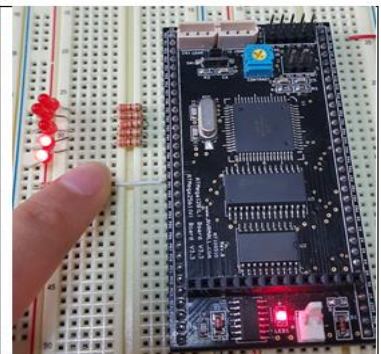
<실험 1 LED ON/OFF>



처음 스위치를 누르면,
1, 2번째 LED ON



다시 스위치를 누르면,
3번째 LED ON



다시 스위치를 누르면,
4, 5번째 LED ON

AVR 소스 코드 사진

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h> // 인터럽트 관련 헤더파일

#define F_CPU 8000000
#include <util/delay.h>
#include "lcd.h"

#define SD PBO // S는 Switch, PBO 핀을 풀업 저항이 달린 스위치로 사용
#define DO PBO // D는 Diode, PBO 핀을 330옴 저항이 달린 LED로 사용
#define string_num 4
volatile int state = 0; // 인터럽트 사용을 위한 Volatile 변수

ISR(INT0_vect) //Interrupt Service Routine: 인터럽트가 호출되었을 시 실행되는 루틴
{
    state++; //state로 어떤 LED를 켤지 결정
}

void INIT_PORT(void) //PORT 초기화 함수
{
    DDRB = 1<<DO; //PORT B의 0번 PIN을 다이오드로 사용하기 위한 출력으로 설정
    DDRD = 0<<SD; //PORT D를 인터럽트를 사용한 스위치로, PBO PIN을 입력으로 설정
    PORTD = 1<<SD; // PBO PIN의 풀업 저항 사용
}

void INIT_INT0(void) //인터럽트 초기화 함수
{
    EIMSK |= (1<<INT0); //INT0 사용, 인터럽트 마스크 레지스터 : 인터럽트 활성화
    EICRA |= (1<<ISC01); //인터럽트 컨트롤 레지스터 : 하강 엣지에서 인터럽트 발생
    sei(); // 전역적으로 인터럽트 허용, SREG |= 0x80;
}

int main(void)
{
    INIT_PORT(); //포트 설정
    INIT_INT0(); //INT0 인터럽트 설정

    LCD_init();

    char *string[] = {"line 1", "line 2", "hello!", "nice!"};
    int r=0, c=0, save=0;

    while(1){
        if(state!=save){
            PORTB |= (1<<DO);
            while(r<string_num){
                while(c<16){
                    if(string[r][c]=='\0'){
                        if(r%2==0) {LCD_command(0xc0);}
                        else {LCD_command(0x01); LCD_command(0x80);}
                        r++; c=0;
                        break;
                    }
                    else {
                        LCD_data_write(string[r][c]); _delay_ms(1000);
                        c++; save=state;
                        break;
                    }
                }
            }
            break;
        }
        else PORTB &= ~(1<<DO);
    }
    return 0;
}
```

프로그램 분석

다음의 소스 코드는 강의 자료에 있는 소스코드를 참고하여 만들었다. <실험 1 LED ON/OFF>의 추가 실험에서 인터럽트 발생에 관하여 자세히 언급해 놓았다. 이번 실험에서는 인터럽트를 이용하여 어떠한 동작이 발생하도록 하였는지를 더 자세히 분석할 것이다.

- 한 글자씩 문자를 디스플레이하는 동작

```
for(r=0; r<2; r++){
    for(c=0; c<16; c++)
        if( string[r][c] == '\\0' ){
            if( r == 0 )          LCD_command( 0xc0);
            break;
        }else
            LCD_data_write( string[r][c]);
}
```

- » r=0인 마지막 문자열에서는 두번째 줄에 디스플레이 해야 함
- » LCD_command(0xc0) 명령으로 DDRAM 커서 위치(AC6~AC0)에 0x40 값을 기록하여 커서 위치를 아랫줄 첫 번째 칸으로 이동시킴

[illegible]

```
0xc0 :      1      1      0      0      0      0      0      0
```

스위치를 누를 때마다 인터럽트가 발생하고 그에 따라 State가 증가한다.

```
ISR(INT0_vect) //Interrupt Service Routine: 인터럽트가 호출되었을 시 실행되는 루틴
{
    state++; //state로 어떤 LED를 켤지 결정
}
```

포인터 배열로 다음의 글자들을 가리키도록 하였다.

- 포인팅의 개요

```
char *string[] = {"ATmega128", "Microcontroller"};
```

<강 의 자 료>

```
char *string[] = {"line 1", "line 2", "hello!", "nice!"};
int r=0, c=0, save=0;
```

<소 스 코 드>

첫번째 줄에 출력할 문자열은 “line 1”, 두번째 줄에 출력할 문자열은 “line 2”,

다음 다음은 순서대로 hello! 와 nice! 이다. 포인터 배열이 어떻게 동작하는 지는 다음에서 자세히 설명할 것이다. r, c, save 변수로 한 글자씩 출력하는 데 이용할 것이다.

```
while(1){
    if(state!=save){
        PORTB |= (1<<D0);
        while(r<string_num){
            while(c<16){
                if(string[r][c]=='#0'){
                    if(r%2==0) {LCD_command(0xc0);}
                    else {LCD_command(0x01); LCD_command(0x80);}
                    r++; c=0;
                    break;
                }
                else {
                    LCD_data_write(string[r][c]); _delay_ms(1000);
                    c++; save=state;
                    break;
                }
            }
            break;
        }
    }
    else PORTB &= ~(1<<D0);
}
```

while(1)을 반복하며 스위치가 눌러지면 인터럽트가 작동한다.

save 변수는 이전의 state를 저장하여, 현재 state와 비교하여 다르면 if문을 실행한다. 이는 스위치가 다시 한번 눌리면서 state가 증가하기 때문에, 스위치가 눌러진 것을 알 수 있다. 그렇게 PORT B의 PB0를 HIGH시켜 LED를 켜다. 스위치가 눌러지지 않으면 else 문을 실행하며, PORT B의 PB0를 LOW시켜 다이오드를 끄는 역할을 한다.

```
while(r<string_num){
    while(c<16){
        if(string[r][c]=='#0'){
            if(r%2==0) {LCD_command(0xc0);}
            else {LCD_command(0x01); LCD_command(0x80);}
            r++; c=0;
            break;
        }
        else {
            LCD_data_write(string[r][c]); _delay_ms(1000);
            c++; save=state;
            break;
        }
    }
    break;
}
```

if 문 내부에서는 글자를 한 글자씩 출력하거나, 한 줄의 글이 다 출력되면 다음 줄로 넘어가도록 하는 소스 코드가 작성되어 있다. 이는 while() 안에서 break를 실행하도록 해, 한 글자씩 출력하도록 했다. string num은 전처리기로 4를 set해 주었다. string의 멤버로 4문장을 선언했기 때문이다.

```
char *string[] = {"line 1", "line 2", "hello!", "nice!"};
```

string[r][c]에서 r은 4문장을 선언하였기에 최대 4로, c는 각 문장 당 최대 16 글자 디스플레이가 가능하므로 최대 15까지 while()문을 실행한다.

```

if(string[r][c]=='#0'){
    if(r%2==0) {LCD_command(0xc0);}
    else {LCD_command(0x01); LCD_command(0x80);}
    r++; c=0;
    break;
}

```

이 부분은 문장이 끝이 났을 때 (W0를 만났을 때) 실행한다.

r%2==0이 참일 경우 if문을 실행한다. % 연산자는 나눈 후 나머지를 결과로 나타낸다. 즉, r이 짝수일 경우 if문을 실행한다. r은 0, 1, 2, 이렇게 증가하므로, 첫 번째 디스플레이 할 경우는 r=0이다. 두 번째 디스플레이 할 문장은 r=1이다. r이 0 이나 짝수인 것은 lcd의 첫 번째 줄에 출력할 문장인 것이다. 이 문장이 끝이 나면, lcd의 두 번째 줄에 출력하기 위해 LCD_command(0xc0)을 실행한다. 커서를 두 번째 줄로 이동하게 된다.

else일 경우는 lcd의 두 번째 줄에 출력할 문장이 끝이 나고, 다시 첫 번째 줄로 이동해야하는 경우이다. LCD_command(0x01)을 통해 lcd를 클리어해주고 첫 번째 줄로 이동한다. 그리고 문장이 끝이 났으므로 c=0을 통해 다시 첫 글자부터 출력하도록 설정하고, r은 증가시켜 다음 문장을 출력하도록 한다.

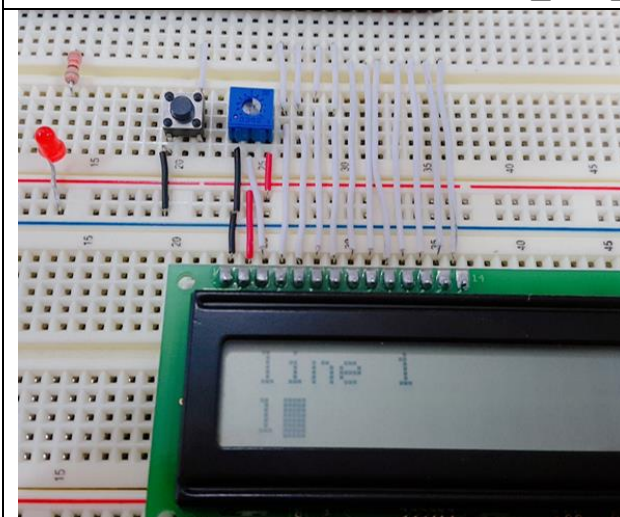
```

else {
    LCD_data_write(string[r][c]); _delay_ms(1000);
    c++; save=state;
    break;
}

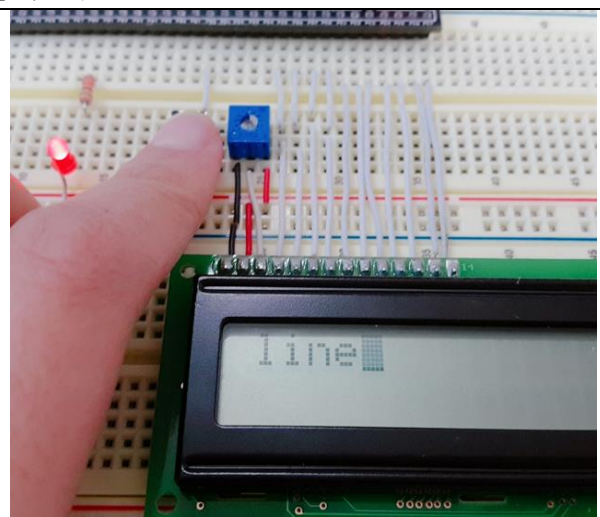
```

이 else문은 문장의 끝이 아닐 경우 실행하는 소스 코드이다. c를 증가시키며 한 글자씩 출력하도록 한다. 그리고 1초를 딜레이 시켜 lcd에 완전히 디스플레이 하도록 한다. 한 글자를 출력한 후 c를 증가시켜 다음 글자를 출력하도록 설정한다. 현재의 스위치 상태를 save에 저장하여 다음에 다시 스위치가 눌러지면 가장 겹의 if문을 실행하도록 한다.

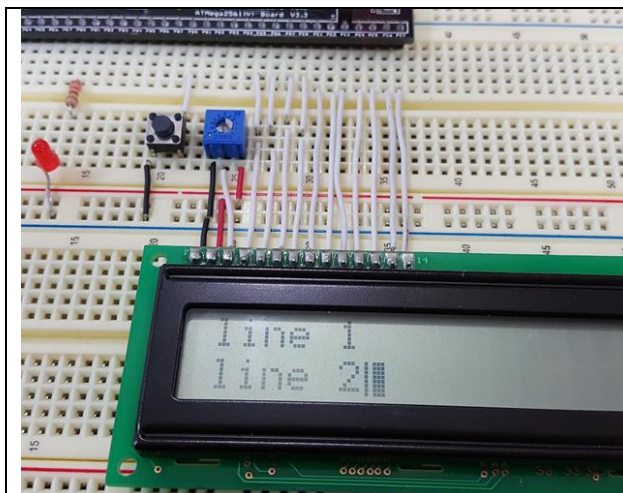
실험 결과 사진



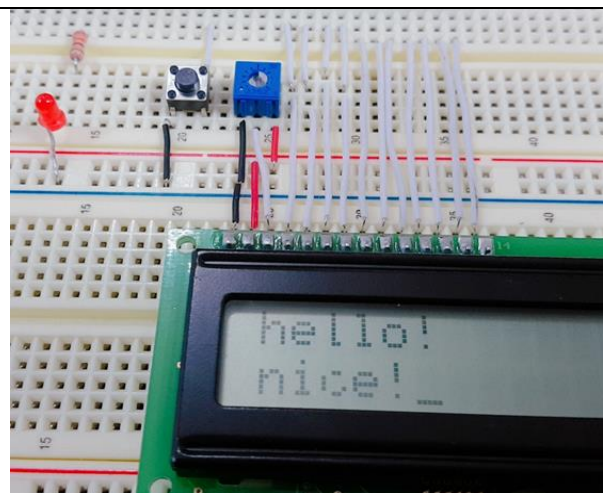
스위치를 계속 누르며 한 글자씩 출력하는 모습이다.



스위치를 누를 때 LED에 불이 들어온다.



스위치를 계속 눌러 첫 번째 줄과 두 번째 줄을 모두 디스플레이한 모습이다.



첫 번째 문장과 두 번째 문장을 다 디스플레이한 후 다시 스위치를 눌러 남은 두 문장을 출력한 모습이다.

실험을 할 때 동영상상을 보면 알 수 있다시피, 글자 중간에 픽셀이 안 나오는 부분이 있다. 제대로 lcd 초기화를 해주어도 그렇다. lcd를 옮겨주어 다시 회로를 구성하여 해결된 적이 있다. 선이 제대로 접촉이 안되어 있어 그럴 가능성이 크다. 전원부도 전원 공급 부분에 제대로 접촉이 되지 않는데 그것 때문일 수도 있다. 뒤의 실험 6을 테스트하면서 lcd도 함께 테스트해볼 것이다.

LCD 실험을 오랫동안 진행한 이유는, 공부가 부족해서이다. 본 실험만으로는 LCD.C에 사용된 모든 함수를 사용할 수 없었고, 다른 추가 실험을 하기 전에 이해할 수 없는 부분이 있었다. 시간이 오래 걸렸지만, 추가 실험을 통해 LCD를 어느 정도 잘 구동할 수 있을 것 같다.

스위치 부분도 이전에 실험했다시피 채터링 현상이 있었다. 소프트웨어적으로 채터링 현상을 방지하기엔 글자를 출력하는 부분을 더욱 강조하고 싶어 하드웨어적으로 채터링 현상을 해결하려 했다. 하지만, 인터럽트 사용 때문인지 오히려 스위치를 눌러도 잘 인식되지 않아 이 부분에 관해 더욱 공부가 필요할 것 같다.