

Arduino 수업계획서

1. Arduino

소속	경기과학고등학교	교사	박종화
순서	강의 주제		비고
1	아두이노 개요	아두이노 시스템 개요 실습 환경 구축 LED Blink 실습	
2	디지털 입출력	브레드 보드 회로 구성 One & Multi LED Blink	
3		버튼 입력처리	
4	시리얼 통신	시리얼 통신 이해	
5	아날로그 입력	전압 분배	
6		가변 저항 입력	프로세싱
7		조도 센서 입력	프로세싱
8		진동 감지 및 소리 출력	LED 연결
9		온도센서	프로세싱
10	아날로그출력	PWM 출력(LED Fading/Servo Motor)	
11	시각적 출력	Character LCD	

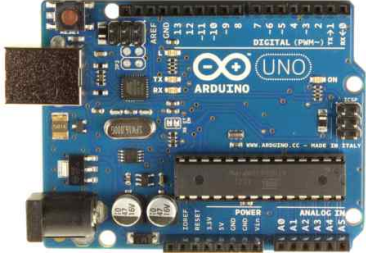
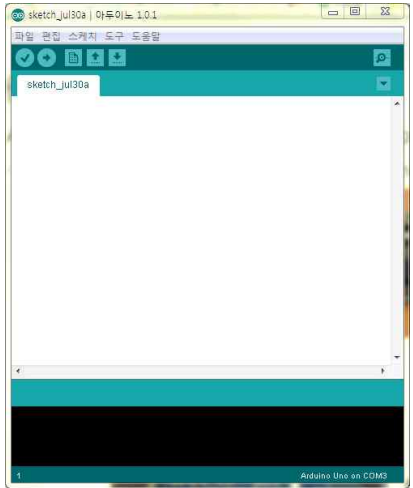
2. Arduino 개요

가. What is Arduino ?

아두이노란 오픈소스 하드웨어 플랫폼으로서 물리적인 장치를 구현할 수 있는 보드와, 이를 구동시키기 위한 프로그램을 수행할 수 있는 통합 개발 환경과, 오픈 소스의 철학이 들어있는 통합적인 의미를 가지고 있다. 아두이노의 배경으로는 피지컬 컴퓨팅이 있으며 이는 컴퓨터와 인간의 상호 작용의 폭을 더 넓히기 위해 보다 쉬운 개발 환경을 제공하여 전문적인 교육을 받지 않은 사람들도 쉽게 하드웨어와 프로그래밍을 통해 컴퓨터와 의사소통을 하고 자신이 구현 하고 싶은 것을 만들 수 있도록 도와주는 것이다.

아두이노를 구성하는 하드웨어는 사용자가 작성한 프로그램이 실제로 실행되는 곳으로서 AVR 계열의 마이크로 컨트롤러를 기반으로 작동을 한다. 아두이노 하드웨어는 다양한 변형이 존재하며, 오픈소스 하드웨어이니 만큼 원한다면 직접 하드웨어를 구성할 수 있다. 아두이노 보드 자체만으로는 외부 환경과의 상호작용을 할 수 없으며 다양한 센서나 부품들을 연결하여 원하는 작업을 수행하게 된다.

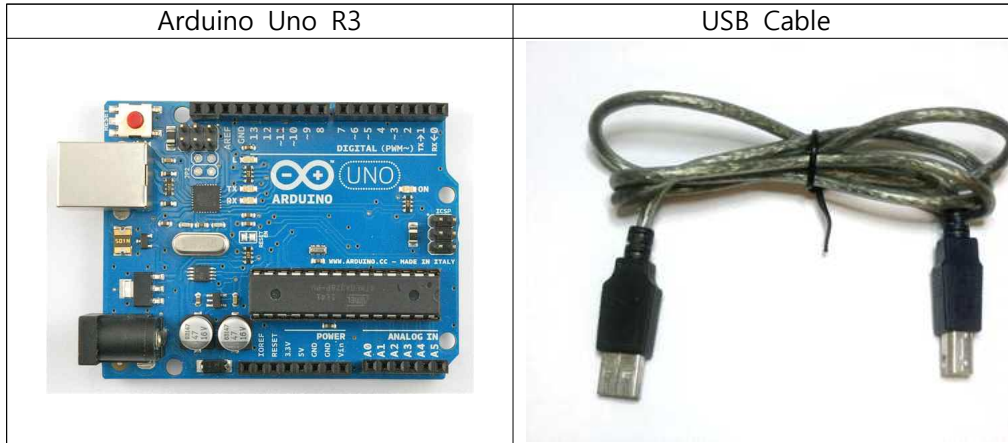
아두이노의 개발 환경은 프로세싱 언어의 IDE에 기반을 두고 있으며, C언어와 거의 유사한 문법을 통해서 하드웨어를 제어하게 된다. 아두이노 IDE는 윈도우 뿐만 아니라 다양한 운영체제에서 실행할 수 있으며, 프로그램을 작성하고 컴파일과 보드로의 업로드를 통해 원하는 실행 결과를 얻을 수 있게 된다.

Hardware	IDE-Integrated Development Environment
	

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

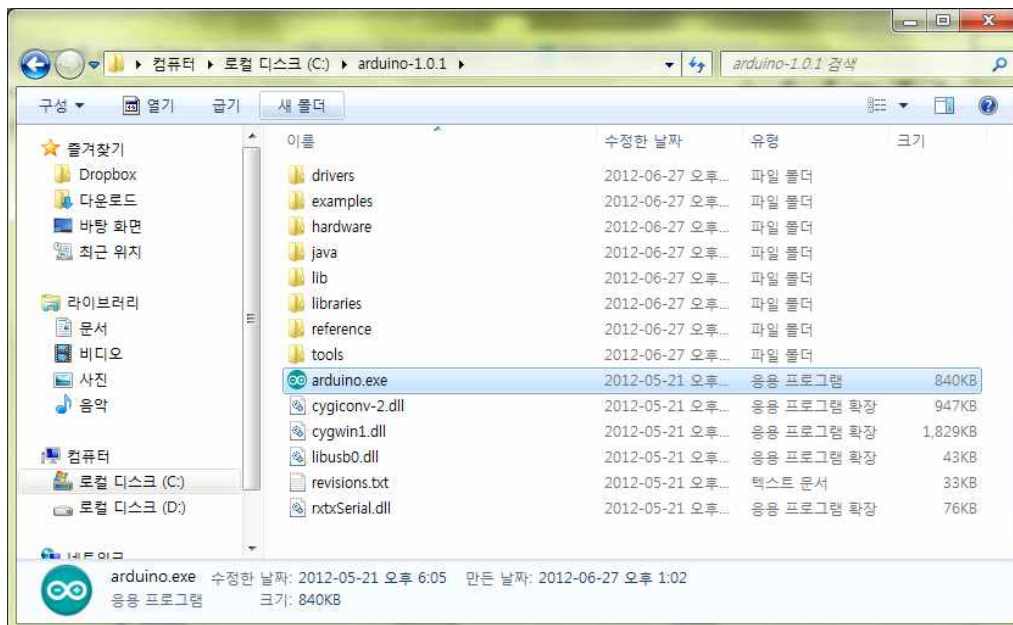
나. 실습환경 구축

1) 하드웨어



2) 소프트웨어

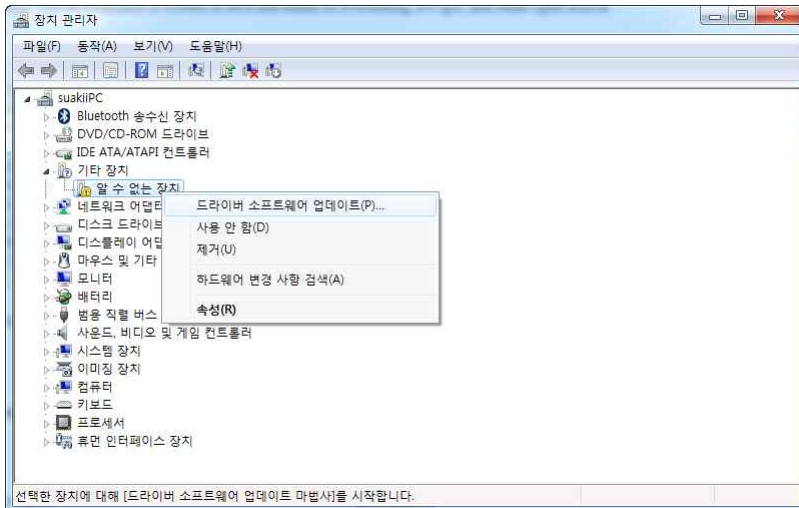
프로그램 다운로드는 아두이노 공식 사이트에서 받을 수 있으며 프로그램 설치하는 압축 파일을 풀어 놓는 것으로 끝난다. (<http://arduino.cc/en/Main/Software>)



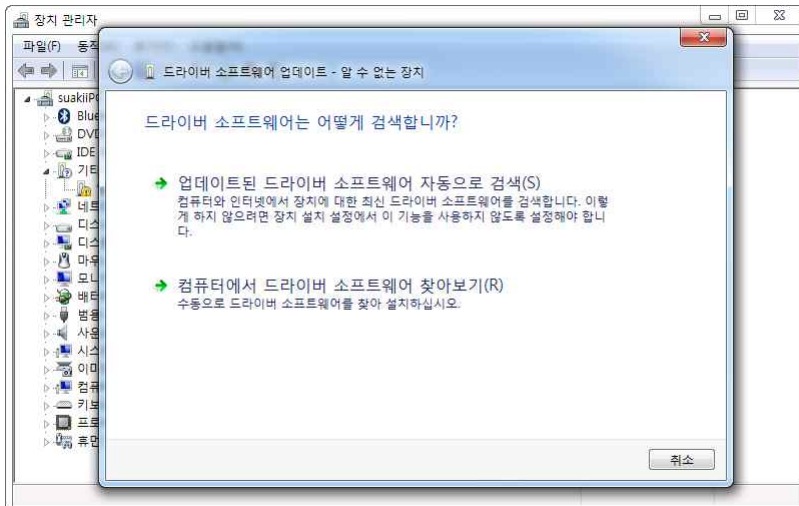
3) 드라이버 설치

USB를 통해 PC와 통신을 하기 위해서는 Driver를 설치 해주어야 한다. Driver 설치 과정은 다음과 같다.

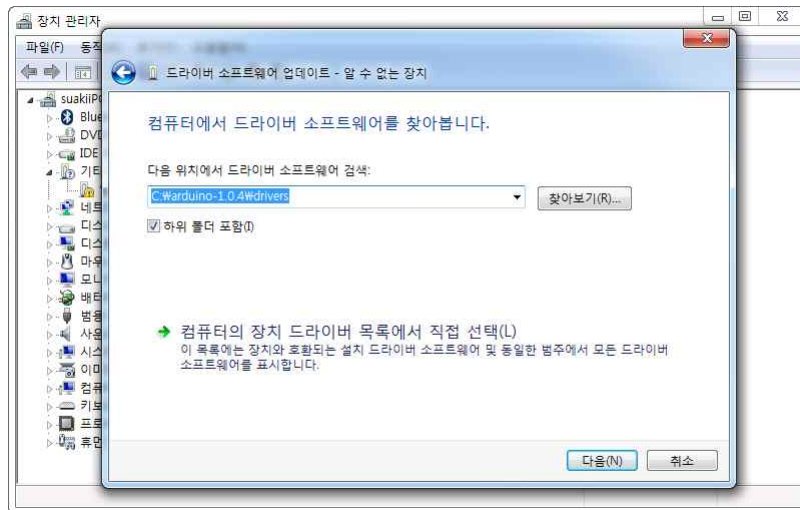
제어판의 장치 관리자에서 알 수 없는 장치로 아두이노 보드는 인식이 되며 드라이버 업데이트를 선택한다.



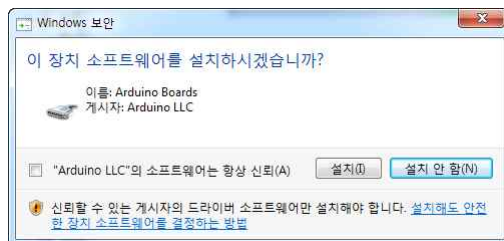
컴퓨터에서 드라이버 소프트웨어 찾아보기를 선택한다.



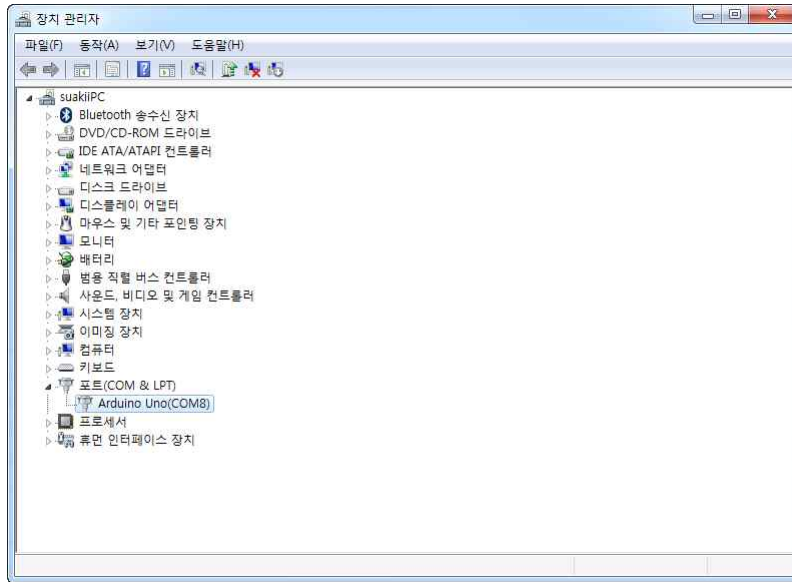
아두이노가 설치된 디렉토리의 drivers 폴더를 선택하여 준다.



다음을 클릭하여 드라이버의 설치 과정을 진행한다.

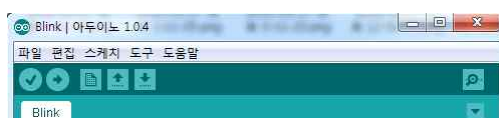
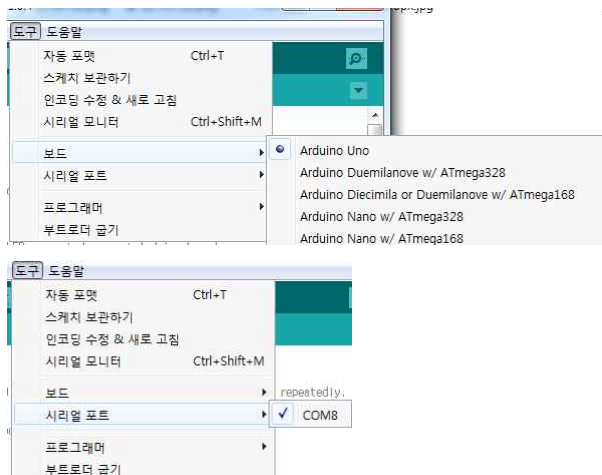


정상적으로 설치된 아두이노가 포트에 보이며 자신의 컴퓨터에 설치된 포트 번호와 아두이노 IDE에서 사용하는 포트 번호는 일치해야 성공적으로 프로그램을 보드로 업로드 할 수 있게 된다.



4) 소프트웨어 테스트

[파일]-[예제]-[Basics]-Blink 예제를 실행해서 보드가 정상적으로 동작하는지 확인해본다. [도구]에서 자신의 보드에 알맞은 것을 선택하고, 장치관리자에 인식되어 있는 시리얼 포트를 선택하고 프로그램 업로드 버튼을 누른다. 정상적으로 동작한다면 아두이노 보드의 작은 LED가 점멸할 것이다. 프로그램을 업로드 하지 않았음에도 LED가 점멸하는 경우 보드의 제조과정에서 기본적으로 해당 프로그램을 업로드 시켜 놓은 것이다.



3. 디지털 입출력(Digital Input & Output)

가. 브레드보드 사용법

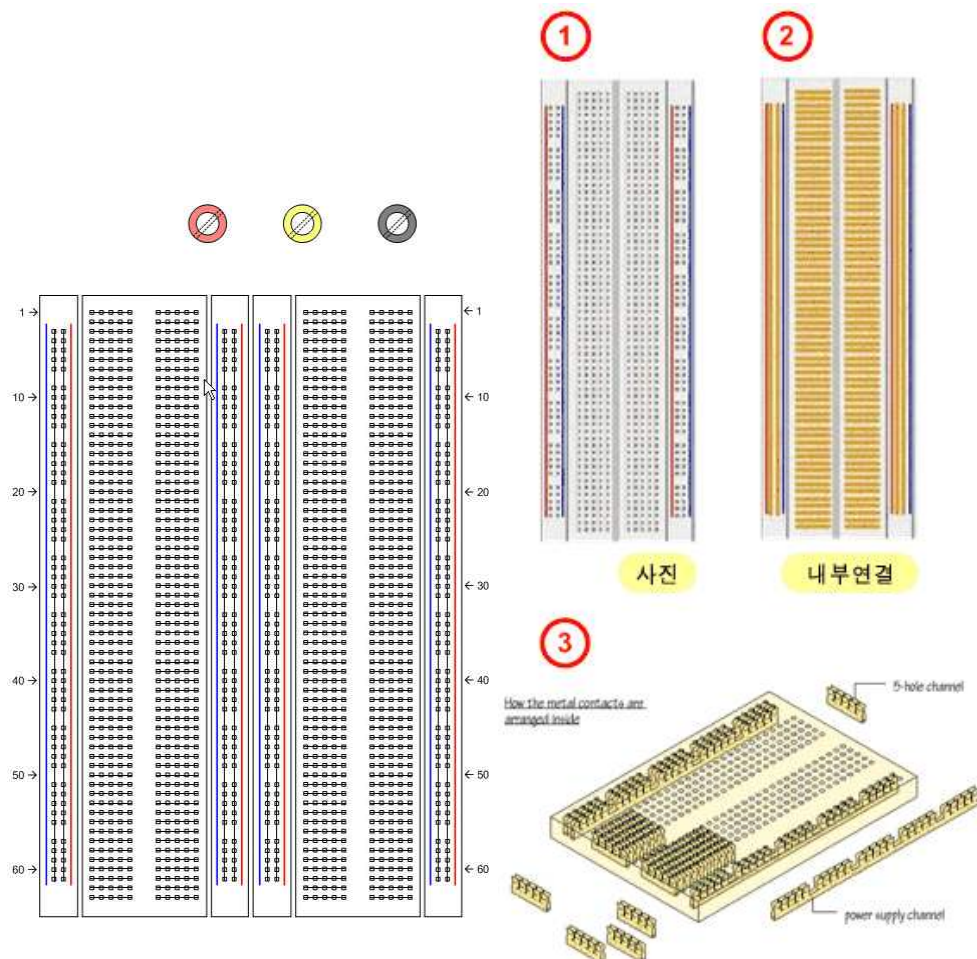
이번 예제는 아두이노 보드 상의 LED를 점멸시키는 것이 아니라 직접 브레드 보드에 하드웨어를 구성하고 프로그램을 업로드하여 LED를 점멸하는 작업을 해본다.

브레드보드(Breadboard)

브레드보드는 전자회로를 구성하는데 있어 납땜을 하지 않고 간단하게 회로를 구성할 수 있도록 도와주는 것이다. 브레드보드는 내부적으로 배선이 되어 있으며 외부에서 전원을 공급하고 회로를 구성하여 동작하게 한다. 일반적으로 기판에 회로를 만들기 전에 시험 및 디버깅을 하는 용도로 사용이 된다.

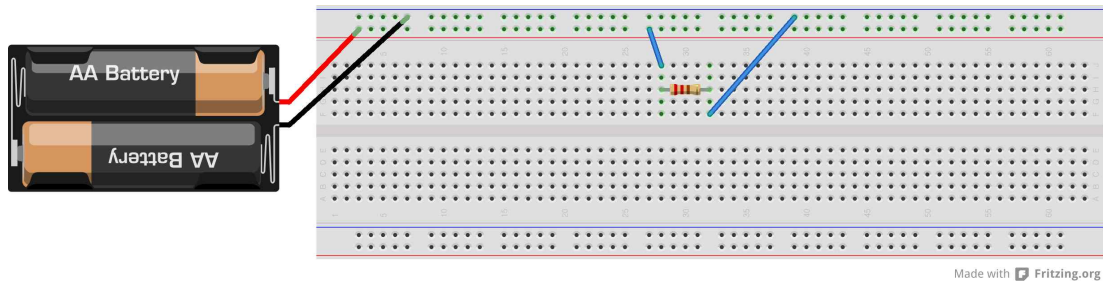
브레드보드의 사용법

일반적으로 빨간색으로 표시된 라인은 +전원, 파란색으로 표시된 라인은 GND에 연결하여 사용하며, 라인과 라인의 연결은 전선(케이블)을 이용하여 연결한다. 브레드보드의 윗쪽과 아래쪽에 있는 선들은 가로로 연결이 되어 있으며, 이 곳에 전원을 연결하여 모든 부품들이 전원을 공급 받거나, 공통 접지로 사용이 가능하게 된다. 중앙에 위치한 영역은 세로로 연결이 되어 있으며, 분리된 영역은 상하를 구분시키게 된다. 세로로 연결된 곳에 부품을 배치하는 경우에는 부품은 가로로 배치하여 회로를 구성하게 된다.



브레드보드 회로 구성 예

전원을 브레드 보드의 빨간색에 +, 파란색에 -를 연결한 후 부품들을 연결하여 준다. 폐 회로를 구성할 수 있도록 주의하여 연결한다.

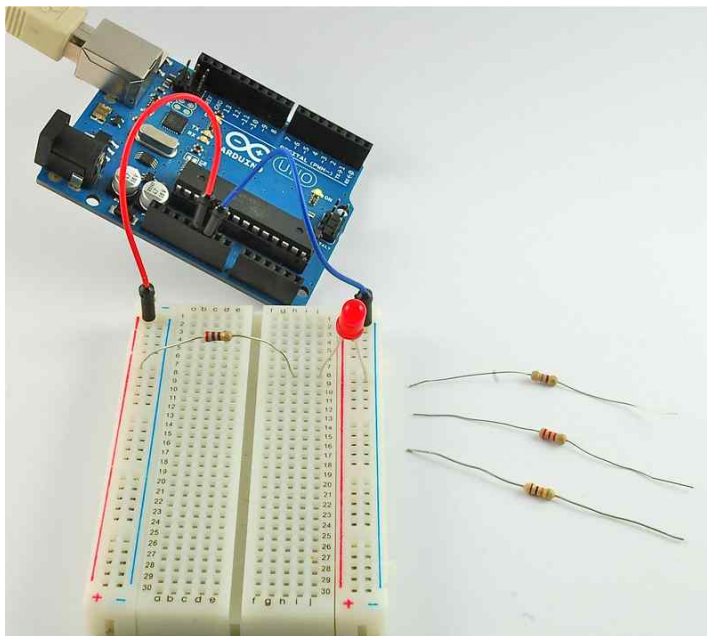


Made with Fritzing.org

나. LED Blink With Breadboard

위에서 LED를 점멸하는 기본 예제 코드를 아두이노 보드에 부착되어 있는 소형 LED에 적용시켜 보았다. 이번 작업은 외부에 회로를 구성하여 동일하게 작동하도록 하는 작업이다.

구성도



필요 부품

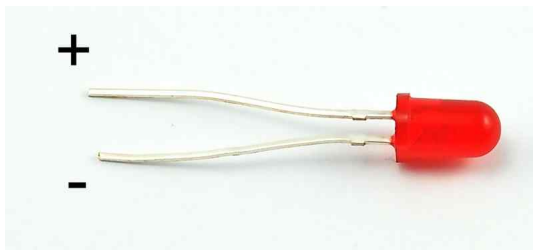
아두이노 보드, 브레드보드, LED, 저항, 전선, usb cable

LED의 특성

LED(Light Emitting Diode)는 한쪽 방향으로만 전류가 흐르는 특성을 가지고 있으며, 다양한 색상의 빛을 낼 수 있는 저렴한 소자이다. LED는 극성을 가지고 있는 소자로서 다리가 긴 쪽이 +이며, 브레드보드에 장착할 시 극성에 주의해서 연결을 해야 한다. 또한 LED는 최대 전류가 제한되어 있으므로 LED의 파손을 방지하기 위해서 전류의 양을 제한시켜주어야 한다. LED에 흐르는 전류를 제한시켜 주기 위해서는 옴의 법칙을 이용하여 저항값을 계산 한 후 회로에 적절한 저항을 추가시켜 주어야 한다. 사용하는 LED의 데이터시트를 참고하여 저항값을 계산해야 하며, 이번 작업에서는 2V의 순전압과 20mA의 전류량을 가지는 LED를 기준으로 하여 저항값을 계산한다. 이 값은 사용하는 LED 별로 다를 수 있으므로 데이터 시트를 참고하면 정확하게 저항을 계산할 수 있다. 저항의 계산은 간단한 옴의 법칙을 이용하여 다음과 같이 알 수 있다.

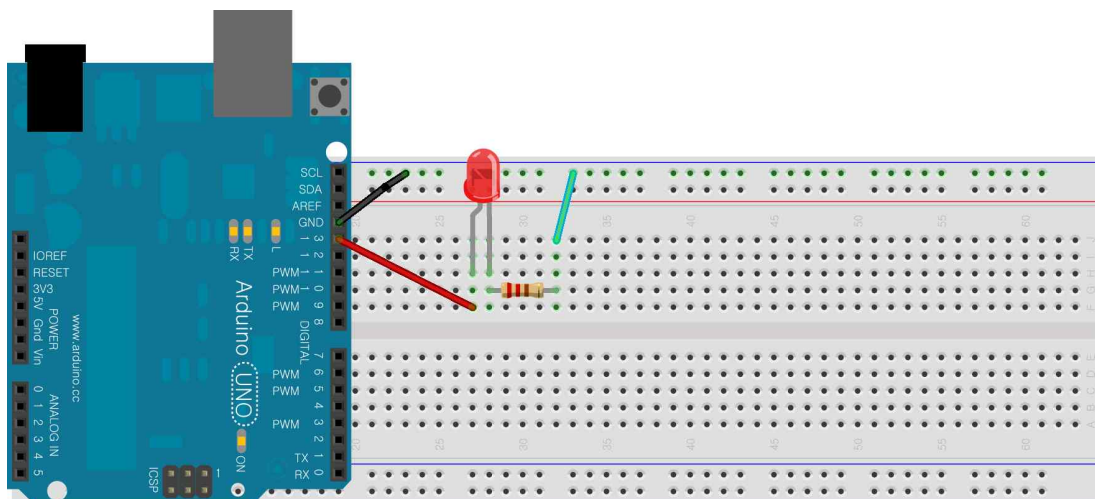
$$R = \frac{V_s - V_f}{I}$$

5V 아두이노를 사용하였을 경우 위 수식은 $\frac{5-2}{0.02} = 150\Omega$ 이 되며, 계산한 저항보다는 큰 저항을 사용하는 것이 안전하다. 다만 너무 높은 저항을 사용하게 되면, LED의 밝기는 매우 약하게 될 것이다.



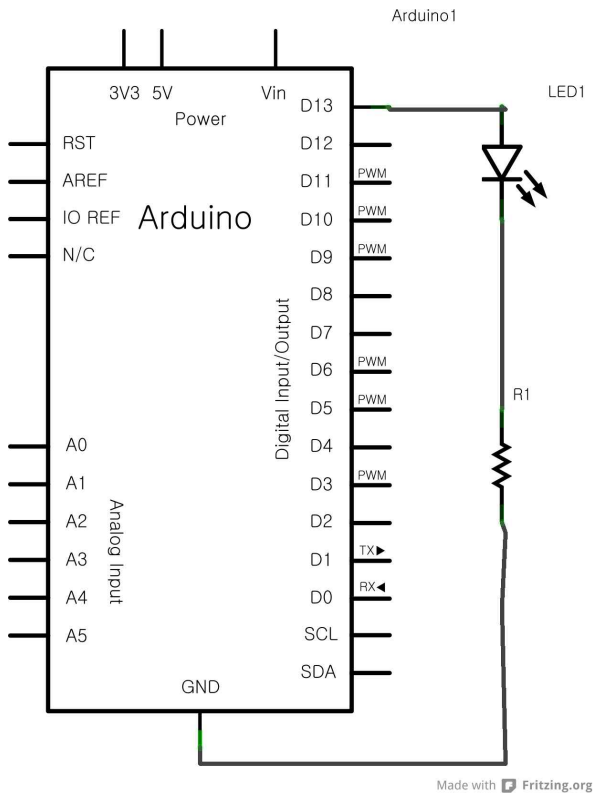
브레드 보드 연결

- 아두이노 13번 핀의 출력을 LED의 +에 연결하고 저항을 연결한 후 GND에 서로 연결을 시켜준다. 회로가 완전하게 구성되어 있는지 체크해보자.



Made with Fritzing.org

- 회로를 알기 쉽게 나타내는 방식은 다양하다. 스케메틱은 각 부품들을 표준 기호로 표시하며 모든 사람들에게 익숙한 방식으로 표현되므로 의사소통을 쉽게 만들어준다.



프로그래밍

- 앞서 작업했던 LED Blink와 동일한 프로그램을 사용한다.

```
int led = 13;

void setup() {

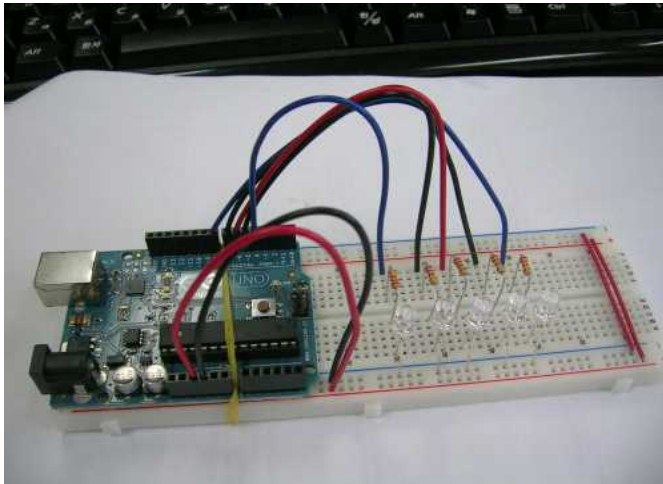
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

다. 다중 LED 출력

이번 예제에서는 하나의 LED가 아닌 5개의 LED를 연결하여 출력하고, 연속적으로 LED를 점멸시켜 본다. for문의 사용법을 익힌다.

구성도

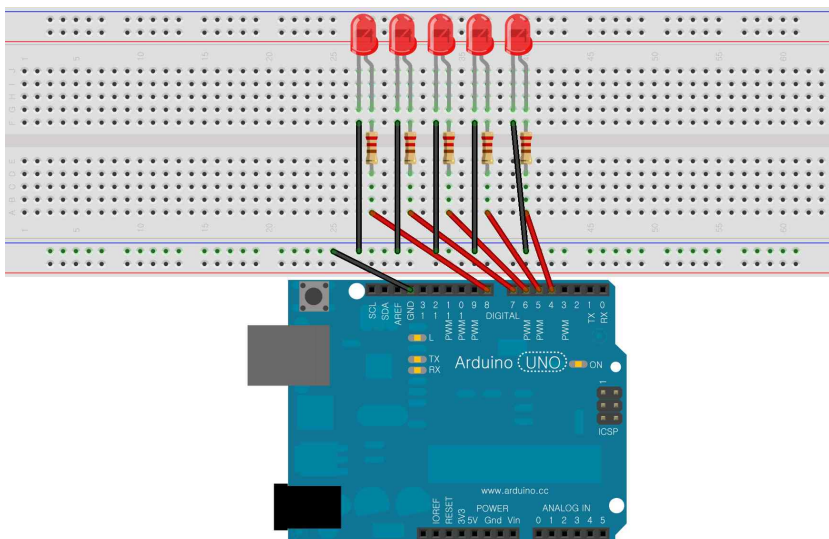


필요 부품

아두이노 보드, 브레드보드, LED, 저항, 전선, usb cable

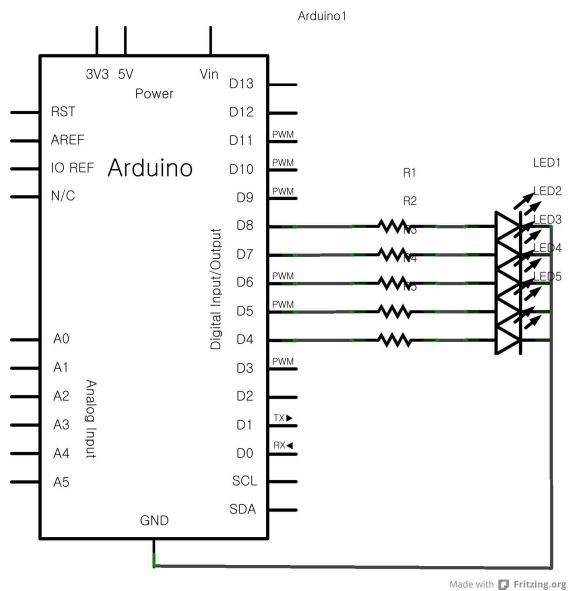
브레드 보드 연결

- 5개의 LED를 브레드보드에 연결하고, 알맞은 저항을 연결해준다. 아두이노의 출력포트 5개를 선택하여 LED에 연결시켜주고 공통의 접지를 시켜준다.



Made with Fritzing.org

스케메틱



프로그램 코드

```
int del = 100;
void setup()
{
  for (int i=4; i<=8; i++) {
    pinMode(i, OUTPUT);
  }
}

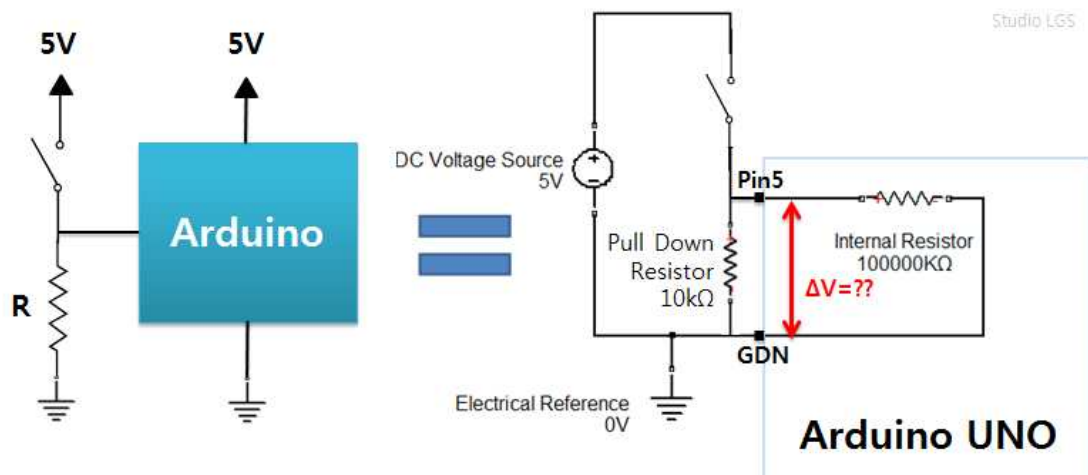
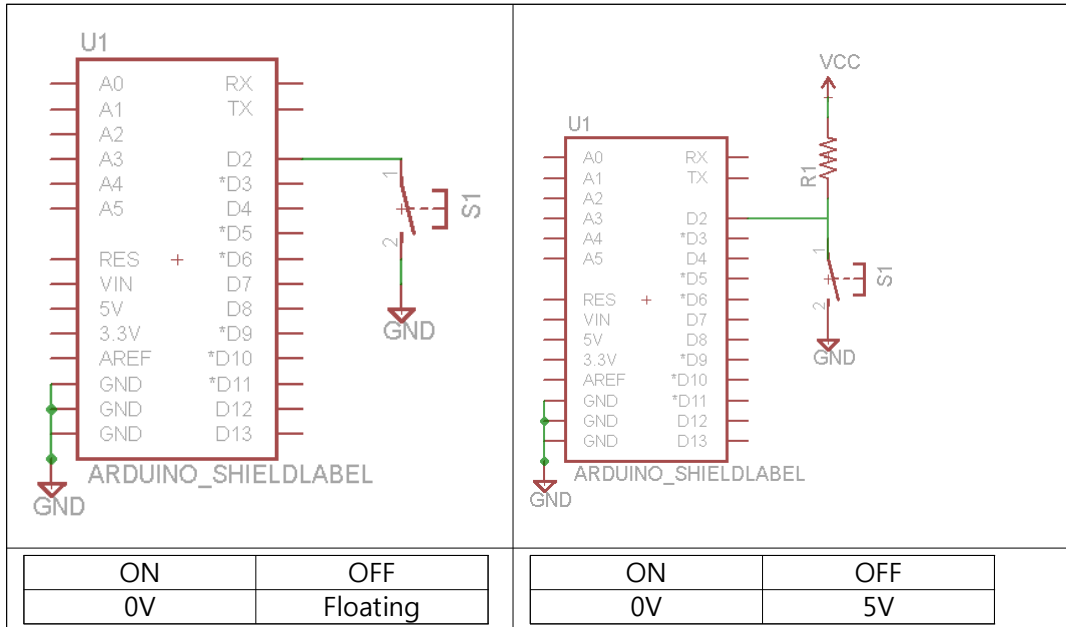
void loop()
{
  for (int i=4; i<=8; i++) {
    digitalWrite(i, HIGH); // turn on LED on pin i
    delay(del);           // wait (length determined by vaue of 'del')
    digitalWrite(i, LOW);  // turn it off
  }
  for (int i=7; i>=5; i--) {
    digitalWrite(i, HIGH); // turn on LED on pin i
    delay(del);           // wait (length determined by vaue of 'del')
    digitalWrite(i, LOW);  // turn it off
  }
}
```

라. 버튼 입력 받기

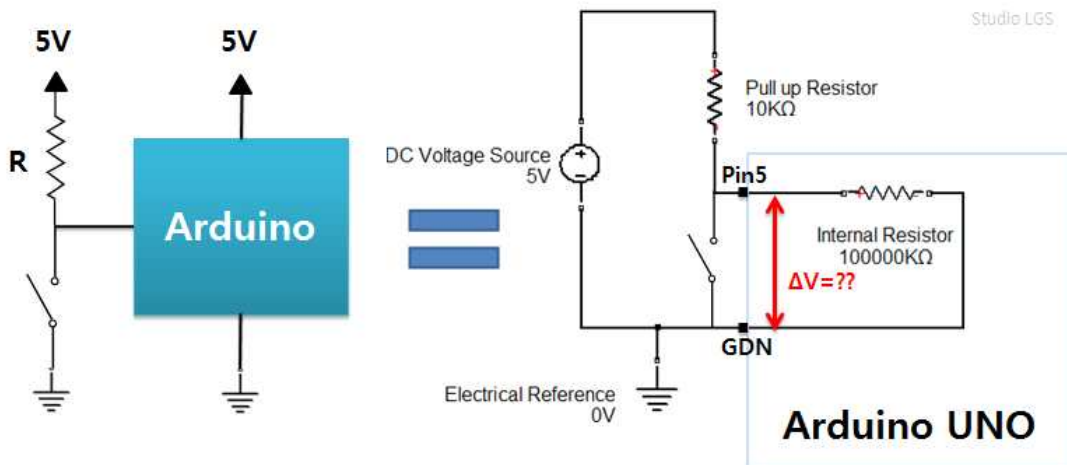
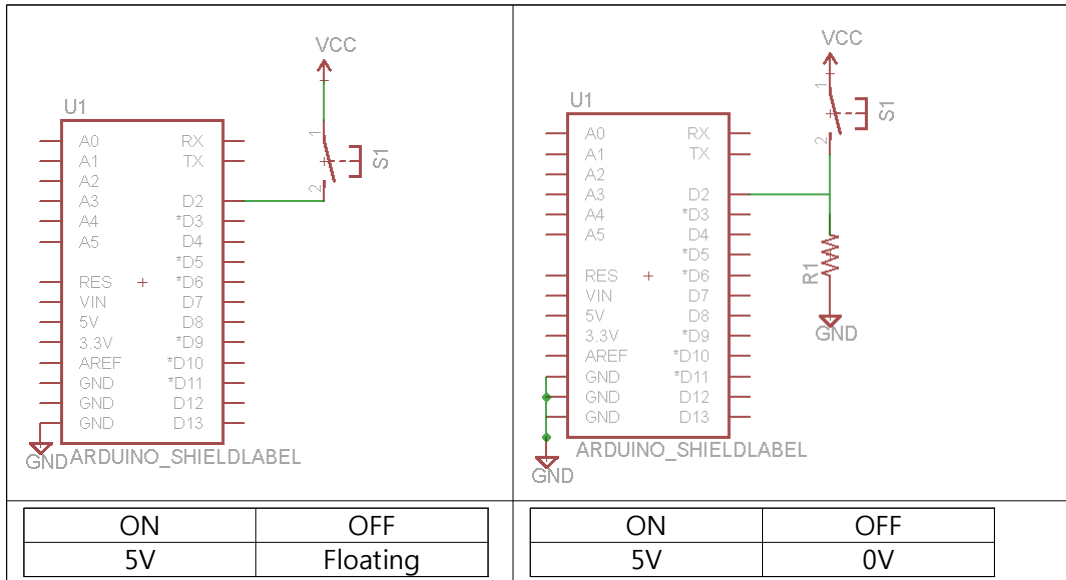
풀업 저항 / 풀다운 저항

버튼이나 스위치 등의 전자적인 접촉이 발생하는 장치의 외부 입력을 받기 위해서는 digitalRead 함수를 사용하여 그 상태를 알 수 있게 된다. 저항을 연결하지 않고 스위치의 입력을 그대로 받아서 처리하는 경우 그 값은 일정하지 않은 상태를 나타내게 되며 이는 회로의 불안정한 동작을 유발시키게 된다. 이러한 문제점을 방지하기 위하여 다음과 같은 풀업저항과 풀다운 저항을 연결하는 기법이 있다.

풀업저항



풀다운 저항

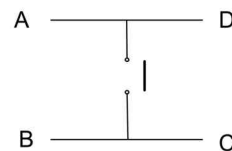
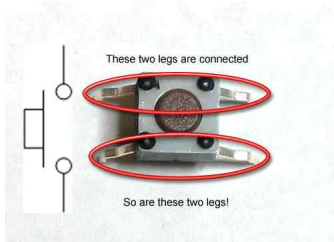


필요부품

아두이노보드, LED, 저항(220옴, 10K옴), 스위치, 케이블

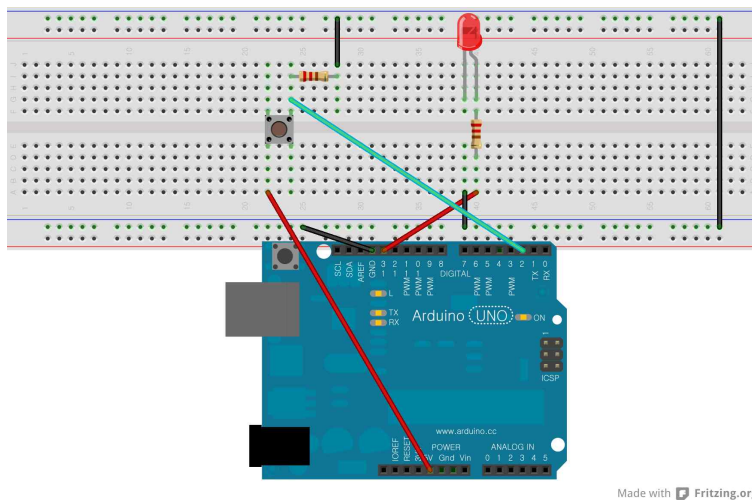
Push Switch

- 넓은 쪽의 다리는 서로 연결이 되어 있는 구조이다.

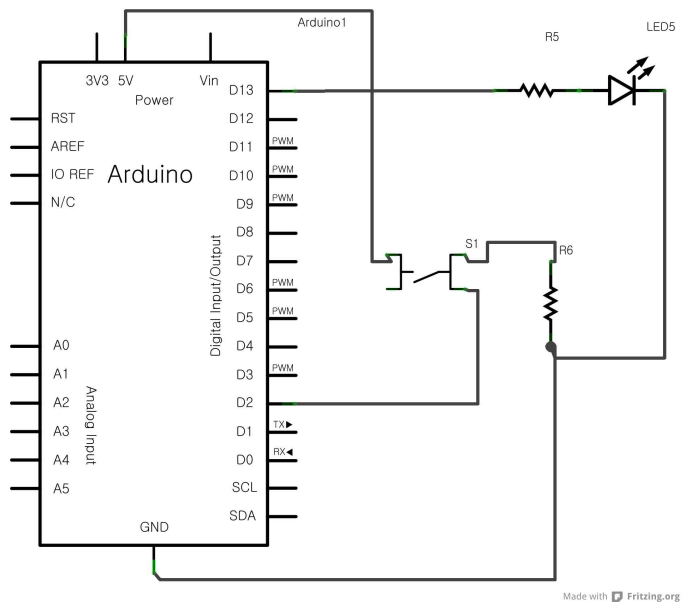


브레드보드연결

-풀 다운 저항을 연결하여 스위치가 오프일 때 입력이 0임을 확실히 보장해준다.



스케메틱



프로그램코드

- 스위치를 누를 때 만 13번 포트에 연결된 LED에 불이 들어오게 된다.

```
const int led = 13;
const int button = 2;
int val = 0;
void setup()
{
    pinMode(led, OUTPUT);
    pinMode(button, INPUT);
}
void loop ()
{
    val = digitalRead(button);

    if (val == HIGH)
        digitalWrite(led, HIGH);
    else
        digitalWrite(led, LOW);
}
```

도전과제1

위의 회로의 구조를 풀업 저항형태로 변경하고 동일하게 작동하도록 변경하시오.

도전과제2

스위치를 누를 때 마다 이전 상태를 변경하는 작업 - 이번 작업에서는 스위치를 누를 때 마다 LED의 상태가 변경되도록 해보자. 상태를 유지하는 변수 값을 보관하고 있다가 스위치의 상태 변화에 따라 이 값을 반영하도록 한다.

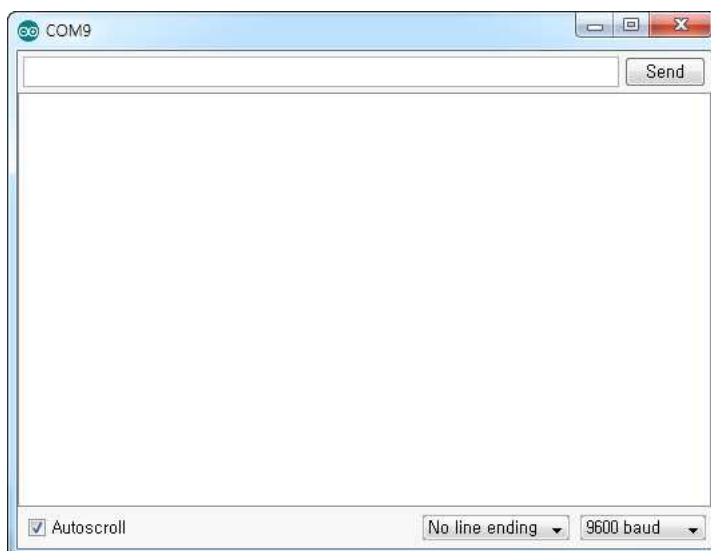
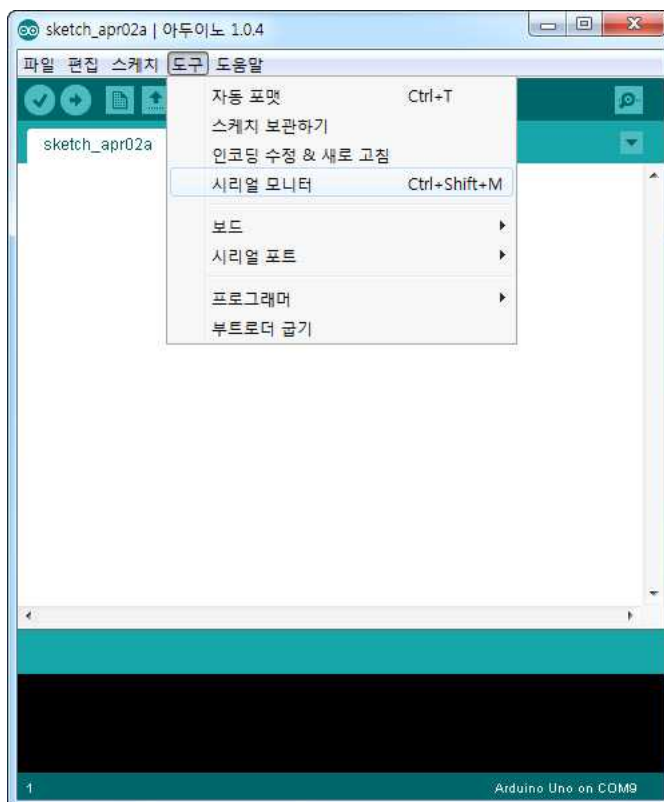
다음과 같은 스케치를 작성하고 업로드 후 예상대로 회로가 동작하는지 테스트 해보자.

<pre> const int led = 13; const int button = 2; int val = 0; int state = 0; void setup() { pinMode(led, OUTPUT); pinMode(button, INPUT); } void loop () { val = digitalRead(button); if (val == HIGH) state = 1- state; if (state == 1) digitalWrite(led, HIGH); else digitalWrite(led, LOW); } </pre>	<pre> const int led = 13; const int button = 2; int val = 0; int old_val = 0; int state = 0; void setup() { pinMode(led, OUTPUT); pinMode(button, INPUT); } void loop () { val = digitalRead(button); //이전 값을 저장해두고 이 값을 비교한다. if ((val == HIGH) && (old_val == LOW)) { state = 1- state; delay(10); } old_val = val; if (state == 1) digitalWrite(led, HIGH); else digitalWrite(led, LOW); } </pre>
--	--

아두이노의 MPU는 생각보다 빠르며, 위의 loop문의 digitalRead 함수는 그 상태가 변화를 것을 수 천번 읽게 되며, 이는 회로의 불안정한 상태를 가져오게 되는 것이다. 따라서, 회로의 동작은 의도한 대로 동작하지 않게 된다. 오른쪽의 개선된 코드는 이전의 값을 저장해 두고 스위치가 눌러졌고 또한 이전의 상태가 서로 다를 때만 상태를 변경시켜 LED의 변경이 제대로 이루어지게 하고 있다.

4. 시리얼 통신(Serial Communication)

시리얼 통신이란 하나의 선을 통하여 데이터를 순차적으로 보내고 이를 통해 두 기기의 의사소통이 이루어지는 것이다. 주위에서 흔히 볼 수 있는 USB 형태의 입력 장치들은 시리얼 통신 방식을 사용한다. 아두이노 보드는 시리얼 통신을 통해 컴퓨터와 다른 장치간에 연결을 할 수 있다. 이전의 작업에서 작성된 스케치 코드는 아두이노보드에 있는 USB-to-serial 칩을 통해 컴퓨터에 연결된 USB 포트를 시리얼 포트로서 사용하여 업로드가 수행된 것이다. 시리얼 통신은 프로그램의 디버깅에 효율적으로 사용될 수 있다. 업로드된 프로그램이 실행되고 있을 때 각종 변수의 값 등을 확인하기 위해서는 아두이노에서 시리얼 통신을 통해 컴퓨터로 값을 보내주고 이 값을 시리얼 모니터를 통해서 확인하게 되는 것이다.



아두이노가 보낸 데이터는 시리얼 모니터의 화면에 나타나게 되며, 사용자 컴퓨터에서 아두이노로 데이터를 보낼 때는 값을 입력 후 Send 버튼을 누르면 된다. 이때 시리얼 모니터와 아두이노 보드의 통신 속도를 동일하게 맞추어 주어야 한다.

No line Ending은 전송되는 메시지의 끝에 캐리지 리턴을 추가할 것인지를 결정 짓는 것이다. 시리얼 통신을 구현하기 위해서는 하드웨어와 소프트웨어가 필요하며, 하드웨어는 아두이노와 대상 장치 간에 전기적인 신호를 통해 정보를 주고 받는 역할을 하며, 소프트웨어는 하드웨어 위에서 인식할 수 있는 비트나 바이트를 전송하게 되는 것이다. 하드웨어와 관련한 복잡한 사항들은 아두이노의 시리얼 라이브러리가 대부분 처리하므로 크게 신경쓰지 않아도 된다. 아두이노와의 통신을 통해 주고 받는 정보를 시각적 혹은 의미있게 표현하기 위해서는 시리얼 모니터로는 한계가 있으며, 추가로 시리얼 통신 부분을 프로그래밍할 수 있는 프로그래밍 언어를 통해서 아두이노와 통신을 하게 된다. 아두이노가 보내는 온도 센서의 값을 그래프로 표현하기 위해서는 적합한 언어를 선택하고 해당 프로그램에서 센서의 값을 시각적으로 표현하는 작업을 추가로 해주어야 한다.

가. 기본 명령

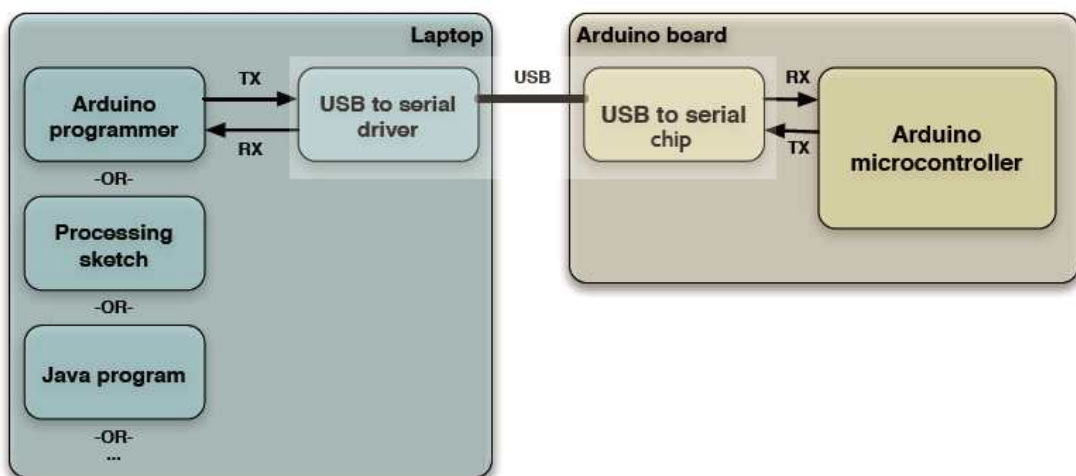
시리얼 통신을 사용하기 위해서 아두이노에 사용되는 기본 명령은 다음과 같다.

Serial.begin() - 시리얼 통신을 사용하기 위한 준비단계

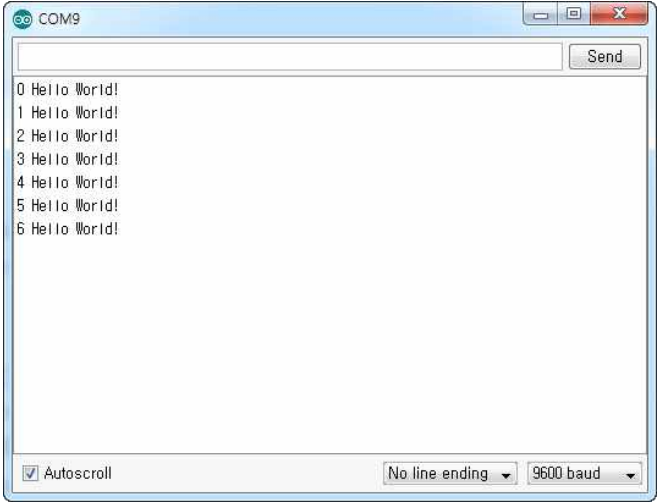
Serial.print() - 시리얼 통신을 통해 컴퓨터로 데이터를 보낸다.

Serial.read() - 시리얼 통신을 통해 컴퓨터로부터 데이터를 읽는다.

- TX – sending to PC
- RX – receiving from PC
- Used when programming or communicating



프로그램 - Serial 출력

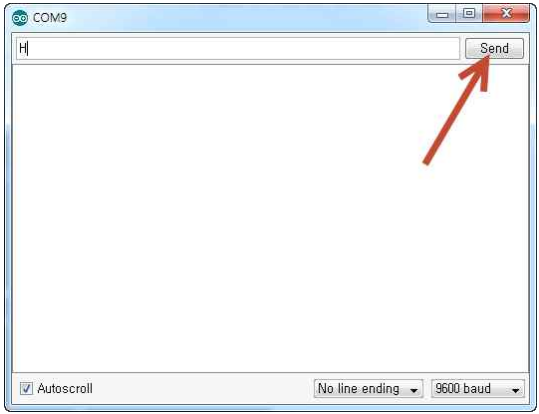
<pre> int led = 13; int i = 0; void setup() { pinMode(led, OUTPUT); Serial.begin(9600); } void loop() { Serial.println(i++); Serial.println(" Hello World!"); digitalWrite(led, HIGH); delay(500); digitalWrite(led, LOW); delay(500); } </pre>	
---	--

프로그램 분석

위 예제코드는 시리얼 포트에 1씩 증가하는 `i` 값과 문자열을 보내며, 13번 포트에 연결된 LED를 점멸하고 있다. 아두이노 보드를 보면 일정한 시간 별로 TX버튼이 점멸하는 것을 볼 수 있으며, 이는 시리얼 통신을 통해 아두이노 보드가 컴퓨터로 자료를 전송하고 있음을 의미한다.

프로그램 - Serial 입력

이번에는 컴퓨터에서 아두이노로 시리얼 포트를 통해 값을 보내고 아두이노에서 이 값을 받아 H이면 LED를 점멸시키는 코드를 작성해보자.

<pre> int led = 13; int val = 0; void setup() { pinMode(led, OUTPUT); Serial.begin(9600); } void loop() { if (Serial.available()) { val = Serial.read(); if (val == 'H') { digitalWrite(led, HIGH); delay(1000); digitalWrite(led, LOW); } } } </pre>	
--	--

프로그램 분석

시리얼 포트에서 데이터를 받아들이는 부분만 달라졌다. loop문 안에서 Serial에 이용할 수 있는 데이터가 있다면 이를 읽고 이 값을 H와 비교한 후에 LED를 일정 시간동안 켜주는 코드이다.

Serial.read()의 리턴 값은

the first byte of incoming serial data available(or -1 if no data is available) - int

나. Serial.print() & Serial.write()

Serial.print(val), Serial.print(val, format)

format 부분은 전송되는 값의 데이터 타입을 결정짓는다.

Serial.print(78, BIN) : "1001110"

Serial.print(78, OCT) : "116"

Serial.print(78, DEC) : "78"

Serial.print(78, HEX) : "4E"

Serial.println(1.23456, 0) : "1"

Serial.println(1.23456, 2) : "1.23"

Serial.println(1.23456, 4) : "1.2346"

Serial.write()는 시리얼 포트를 통해 바이너리 데이터를 보낸다는 차이가 있다.

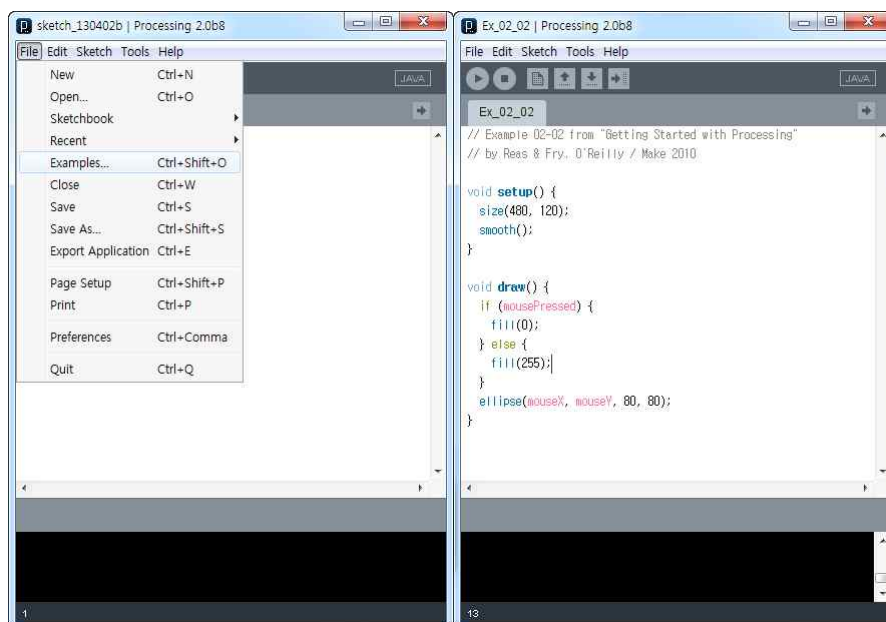
Serial.write(65); : A

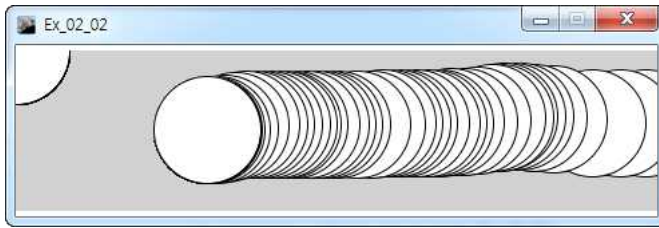
다. 데이터의 시각화

아두이노가 시리얼 포트에 보내온 데이터를 시각화해보자. 이는 다양한 센서의 값들을 시각적으로 표현해야 하는 작업이 많기 때문이다. 이번 작업에서는 편리하게 그래픽 영상 작업을 할 수 있는 프로세싱이라는 언어를 이용하여 작업을 한다. 프로세싱은 아두이노와 유사한 IDE 환경을 가지고 있으며, 자바를 기반으로 개발이 되었다. 프로세싱은 setup()과 draw()라는 두 가지 기본 함수를 가진다.

1) 프로세싱 다운로드

프로세싱을 processing.org에서 다운 받을 수 있으며 설치하는 아두이노와 마찬가지로 압축을 해제하는 과정으로 끝이 난다. 다양한 예제파일들이 포함되어 있어 스스로 학습을 할 수 있다.





2) 시리얼 데이터 표현하기

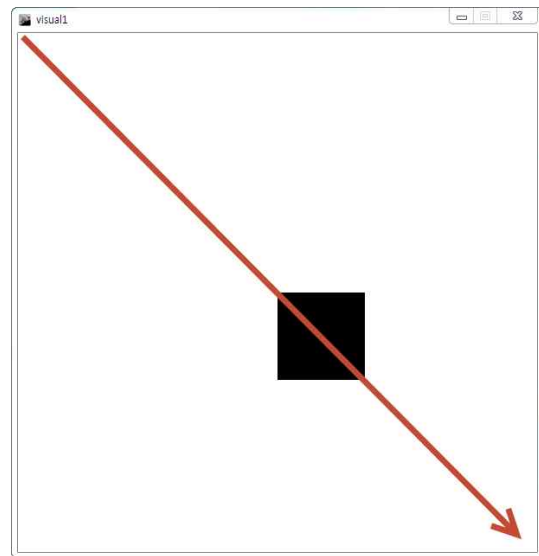
아두이노가 보내오는 시리얼 데이터를 읽기 위해 프로세싱은 시리얼 포트를 열고 해당 자료를 읽어 표시해야 한다. 이번 작업에서는 아두이노가 1씩 증가시키는 값을 입력으로 받아 화면상의 직사각형을 움직여 보는 작업이다.

```
import processing.serial.*;

int lf = 10;
Serial myPort;
String myString;
int value;

void setup() {
  size(600, 600);
  println(Serial.list());
  myPort = new Serial(this, Serial.list()[4], 9600);
}

void draw() {
  while (myPort.available() > 0) {
    myString = myPort.readStringUntil(lf);
    if (myString != null) {
      myString = trim(myString);
      value = int(myString);
      println(value);
      background(255);
      fill(0);
      rect(value*5, value*5, 100, 100);
    }
  }
}
```



프로그램 분석

setup 함수에서 화면의 크기를 선언하고, 0번 포트 번호를 사용한다. 이는 컴퓨터 별로 다를 수 있다. draw 함수에서 읽을 데이터가 있다면 해당 값을 출력하고 화면에 정사각형을 그린다. value 값이 증가함에 따라 정사각형은 화면의 대각선을 따라 움직이게 된다.

5. 아날로그 입력

가. 아날로그 값 읽기

연속적으로 변하는 형태의 값인 아날로그 값을 읽기 위해서는 아날로그 핀에 연결된 외부 입력 으로부터 값을 읽어 들여야 한다. 아두이노 Uno는 6채널의 아날로그 입력 포트를 가지고 있으며, 10비트의 해상도를 가진다. 아날로그 입력 역시 전압의 변화 값으로 읽혀지게 되며 0~5V 사이의 변화 값은 10비트 해상도에 따라 0~1023의 정수 값으로 변환되어 입력이 된다.

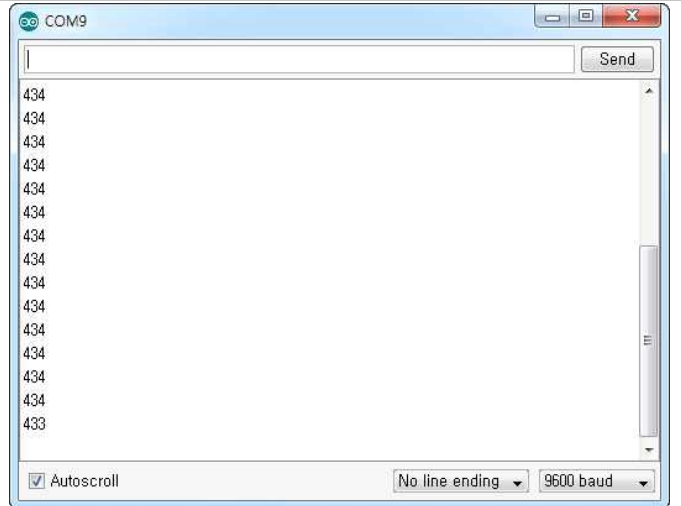
다음과 같은 프로그램을 통해 아두이노의 아날로그 3번 핀에 들어오는 입력을 시리얼 모니터에 출력해보자. 별도의 하드웨어를 구성할 필요는 없다.

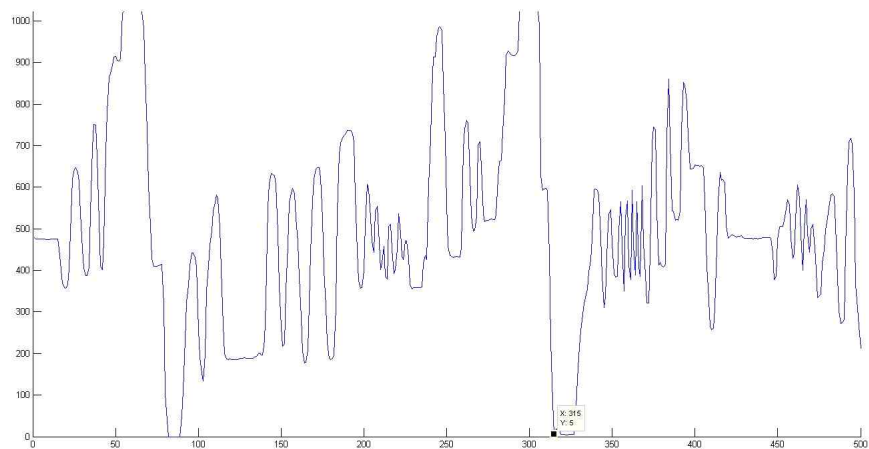
필요부품

아두이노, Usb 케이블

프로그램

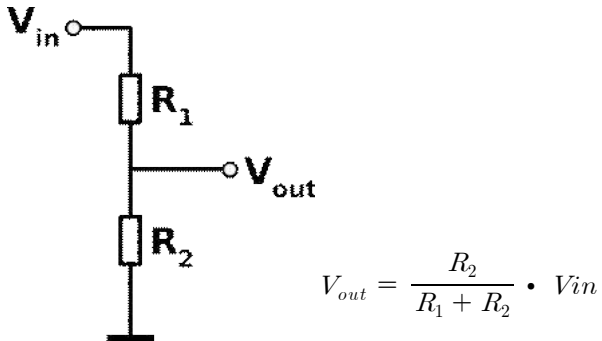
- 아날로그 3번 포트로부터 의미없는 임의의 값이 들어오고 있는 것을 알 수 있다.

<pre>int analogPin = 3; int val = 0; void setup() { Serial.begin(9600); } void loop() { val = analogRead(analogPin); Serial.println(val); }</pre>	
---	---



나. 전압측정하기

아두이노의 아날로그 입력은 0~5V 이며 해상도는 10bit 이므로 약 4.9mV 단위로 측정이 가능하게 된다.



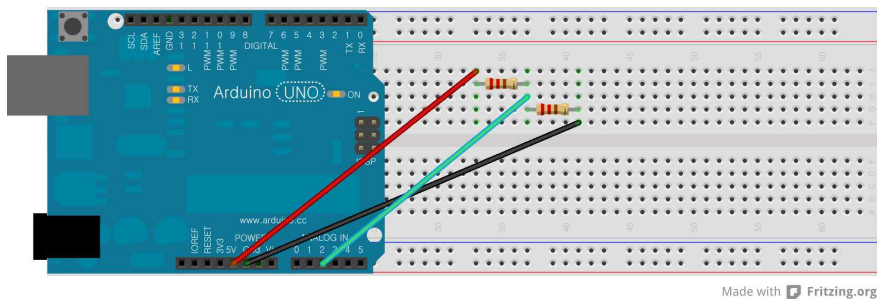
외부에서 입력되는 전압 V_{in} 을 측정은 아두이노가 받아들일 수 있는 5V를 초과하는 전압이 들어올 수 있으며 적절한 저항을 연결을 통해 외부 입력을 계산할 수 있다. 전압의 분배는 다음과 같이 계산할 수 있으며 아두이노 측정된 V_{out} 값을 통해 V_{in} 을 계산할 수 있게 된다.

필요부품

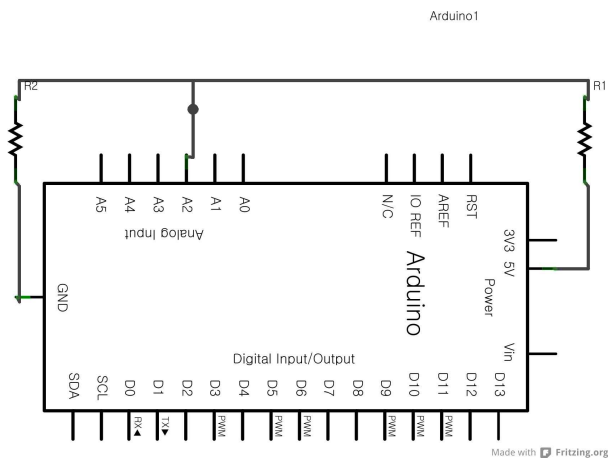
아두이노, 10K옴 저항 2개

브레드보드

간단하게 입력을 아두이노가 제공하는 5V의 전원을 사용하여 입력 전압을 계산하여 입력전압 V_{in} 을 측정해보자.



스케메틱



프로그램 및 결과

```
int analogPin = 0;
int val = 0;
int R1 = 10000;
int R2 = 10000;
float Vin = 0.0;
void setup() {
  Serial.begin(9600);
}

void loop() {

  val = analogRead(analogPin);
  Vin = (R1 + R2) / R2 * val * 5 / 1023.0;
  Serial.println(Vin);
  delay(500)
}
```

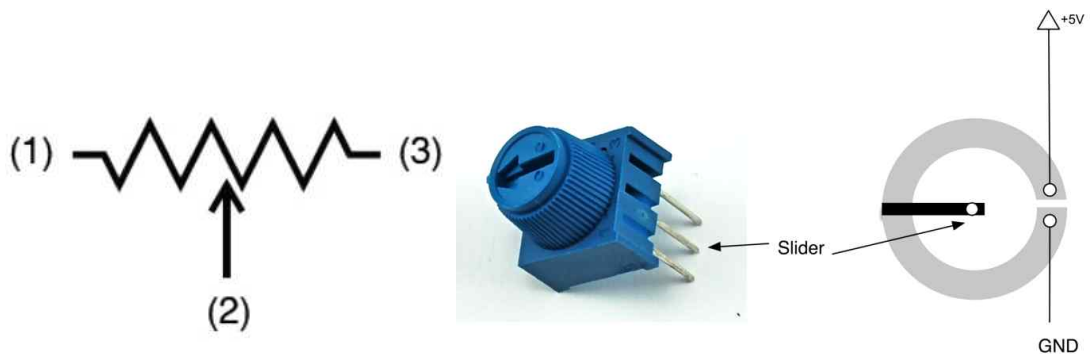


다. 가변저항

이번 작업에서는 가변저항을 사용하여 아날로그 입력에 따라 LED의 점멸 주기를 변경해 보자.

1) LED 점멸주기 변경하기

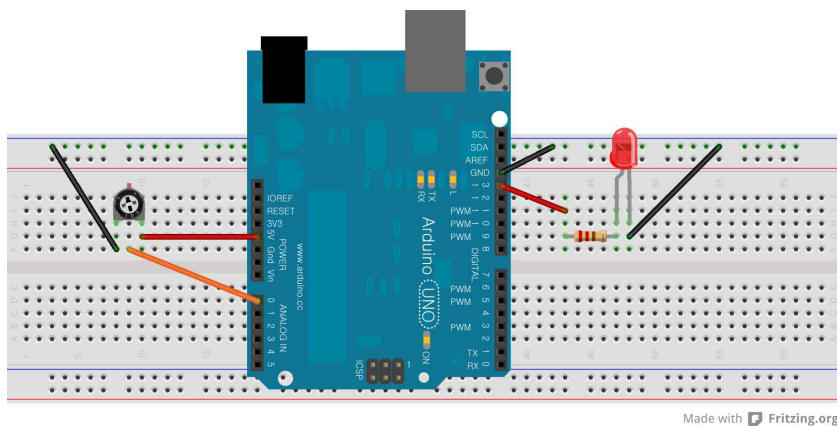
가변저항은 3개의 핀으로 구성되어 있으며, R1-R3의 저항은 동일하며 이는 가변 저항 전체의 저항 값이 된다. 보통 중앙에 위치한 2번은 저항을 회전시킴에 따라 일종의 와이퍼가 좌우로 움직여 저항의 변화를 발생시키게 된다. 즉, R1-R2와 R2-R3의 저항 값들이 변하며 전체 저항의 값은 일정하게 된다.



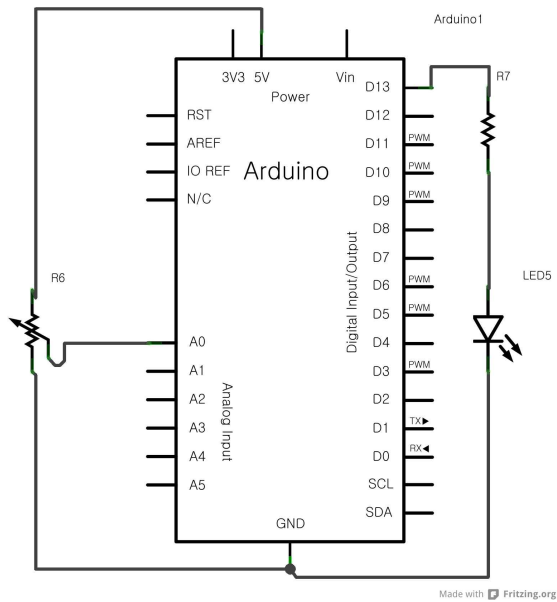
필요부품

아두이노, 가변저항(10Kohm), LED, 저항

브레드보드



스케메틱



프로그램

- 가변저항에서 읽은 저항 값에 따라서 LED의 점멸 주기를 변경 시킨다.

```
int analogPin = 0;
int led = 13;
int val = 0;

void setup() {
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}

void loop() {
  val = analogRead(analogPin);
  Serial.println(val);
  digitalWrite(led, HIGH);
  delay(val);
  digitalWrite(led, LOW);
  delay(val);
}
```



2) 프로세싱으로 시각화

프로세싱은 시리얼 포트를 통해 아두이노가 보내온 가변저항의 값을 이용하여 원을 그리고 해당 원의 크기를 가변 저항의 값으로 그린다. 따라서, 큰 저항 값이 전송되었을 때는 원의 지름이 커지고 반대인 경우에는 작아지게 된다.

가) 프로그램

- 가변저항에서 읽은 저항 값에 따라서 LED의 점멸 주기를 변경 시킨다.

```
import processing.serial.*;

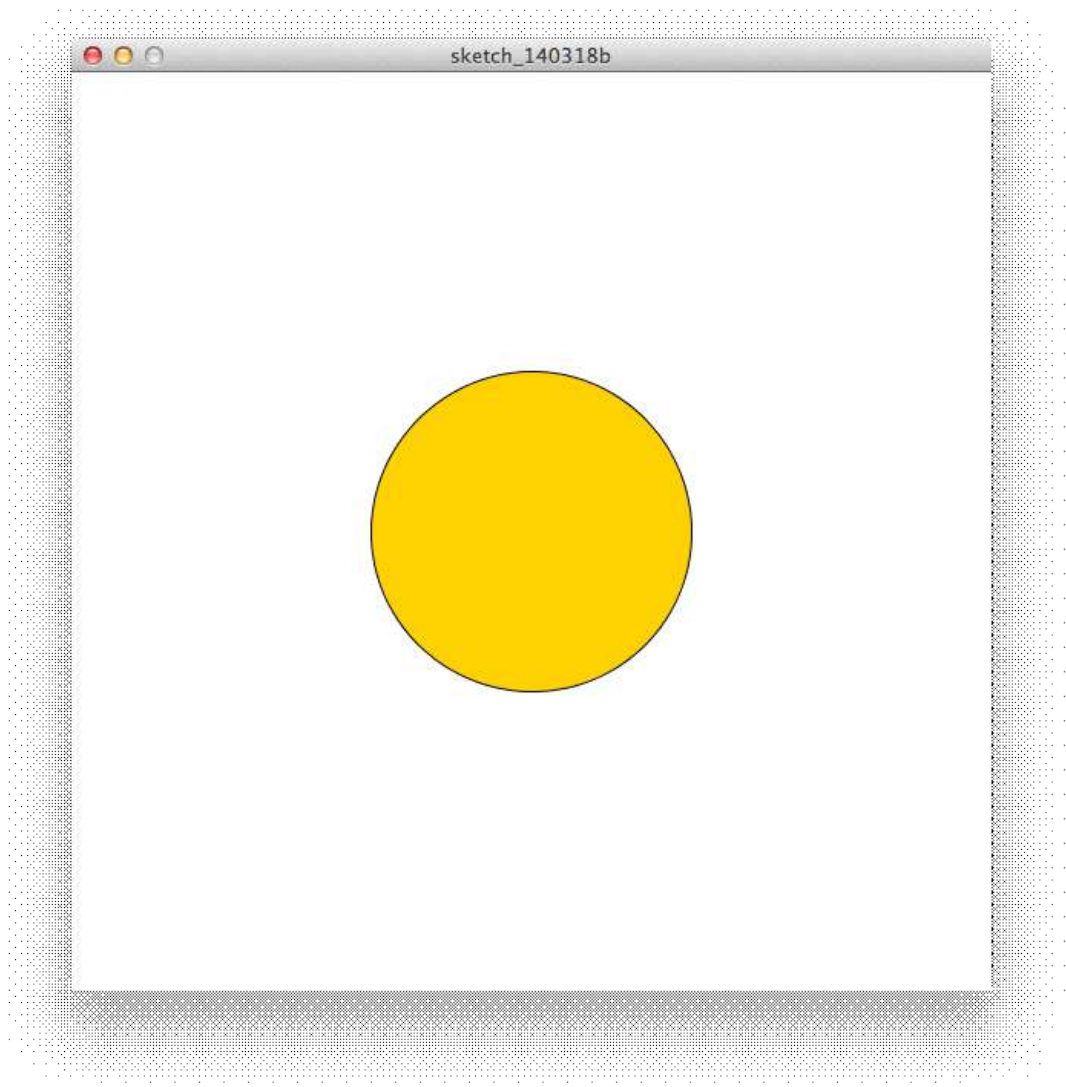
int lf = 10; // Linefeed in ASCII
String myString = null;
Serial myPort; // Serial port you are using
float num;

void setup() {
  background(255,255,255);
  size(600,600);
  println(Serial.list());
  myPort = new Serial(this, Serial.list()[4], 9600); //자신의 포트에 맞는 번호 선택
}

void draw() {
  while (myPort.available() > 0) {
    myString = myPort.readStringUntil(lf);
    if (myString != null) {
      num=float(myString); // Converts and prints float
      background(255,255,255);
      fill(#ffcb01);
      ellipse(width/2,height/2,num/2,num/2);
    }
  }
  myPort.clear();
}
```

나) 실행결과

- 아두이노의 시리얼 모니터 창을 닫아야 한다. 한 번에 하나의 프로그램만이 시리얼 모니터를 이용할 수 있다. 가변저항을 움직임에 따라 프로세싱은 서로 다른 크기의 원을 표시해주게 된다.




3) PWM 출력하기

지금까지 LED를 끄고 켜는 것은 단순히 연결된 포트에 HIGH, LOW 값을 주어서 제어를 한 것이다. 이러한 출력은 디지털 값을 통한 제어였으며, 각각의 LED의 밝기를 조절할 수는 없었다. 이번 프로젝트에서는 PWM(Pulse Width Modulation) 기법을 통해 각 LED의 밝기를 조절해보고, analogWrite함수의 사용법을 익힌다.

PWM(Pulse Width Modulation)은 아두이노가 아날로그 출력을 만들어 낼 수 없기에 펄스의 폭을 빠르게 변화시킴으로써 평균 전압의 양을 조절하여 아날로그 출력을 만들어내는 것이다. 아날로그 출력 함수의 인자로서 0을 넘겨주면 0V의 전압이 나가게 되는 것이며 255까지의 값을 이용하여 그 전압의 크기를 조절한다. 아두이노의 PWM 주파수는 500Hz, 2ms 이므로 아주 빠르며 LED 점등에 사용하면 우리 눈은 그 차이를 거의 구별하지 못할 만큼 서서히 밝기를 조절할 수 있다. 아두이노의 PWM 출력은 디지털 핀에 ~ 표시가 된 포트를 통하여 출력할 수 있다.(3, 5, 6, 9, 10, 11핀)

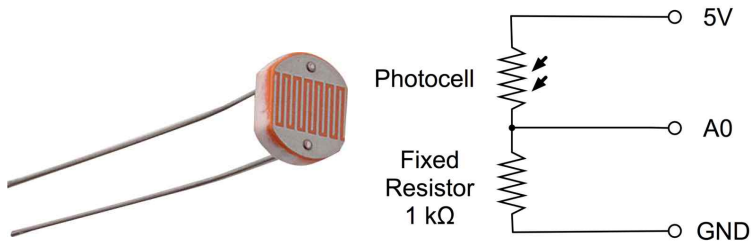
가) 프로그램

- 가변저항에서 읽은 저항 값에 따라서 LED의 밝기를 변화 시킨다.

<pre>int analogPin = 0; int led = 13; int val = 0; void setup() { Serial.begin(9600); pinMode(led, OUTPUT); } void loop() { val = analogRead(analogPin); Serial.println(val); analogWrite(11, val/4); }</pre>	
---	---

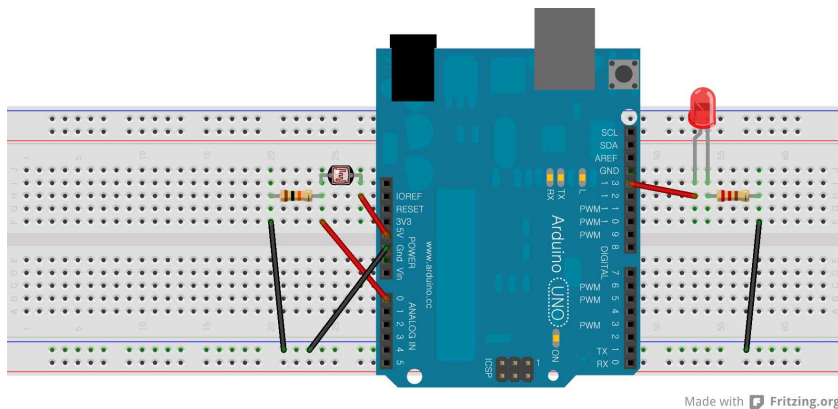
라. 조도 센서 입력

주위의 빛의 밝기를 재는 기본적인 방법은 포토레지스터를 이용한다. 빛의 세기가 증가하면 전기 저항이 낮아지는 특성으로 빛 의존성 저항(Light-Dependent Resistor)이라고 불린다. 조도 센서와 저항을 직렬로 연결하고 저항에 걸리는 전압의 변화량을 읽어 들여 빛의 밝기를 판단한다. 빛이 밝으면 LDR의 저항 값이 낮아지므로 A0의 입력은 높아지며, 어두워지면 저항 값이 증가하므로 A0의 입력은 낮아진다. 아래 그림에서 10킬로옴의 저항을 사용한다.

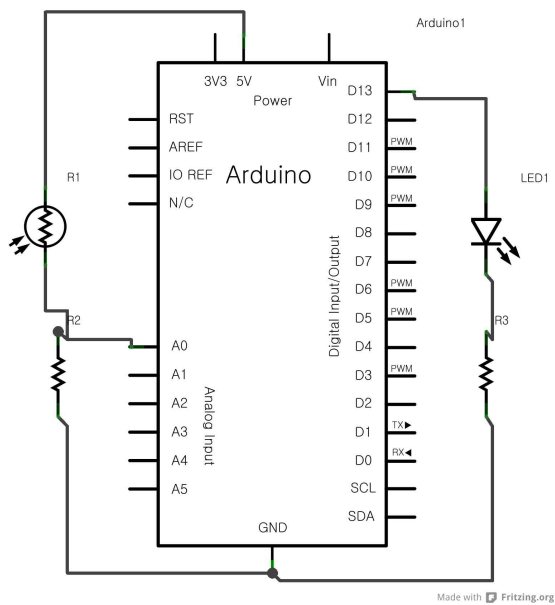


브레드보드

- LDR 센서 값을 읽어 들여 빛이 밝으면 LED를 어둡게, 어두우면 밝게 켜는 작업을 해보자.
또한 LDR 센서 값을 프로세싱에서 표현해보자.



스케메틱



프로그램

```
int sensorPin = 0;
int ledPin = 13;
int sensorValue = 0;

void setup() {

  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

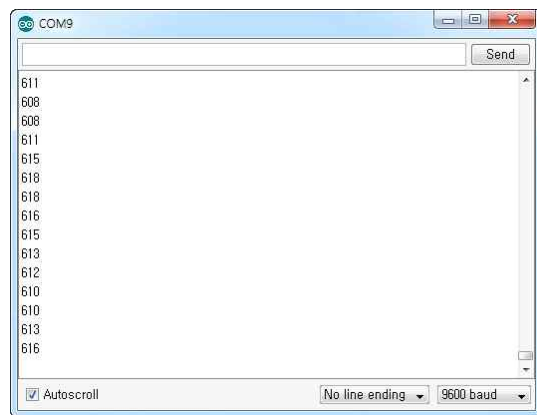
void loop() {

  sensorValue = analogRead(sensorPin);

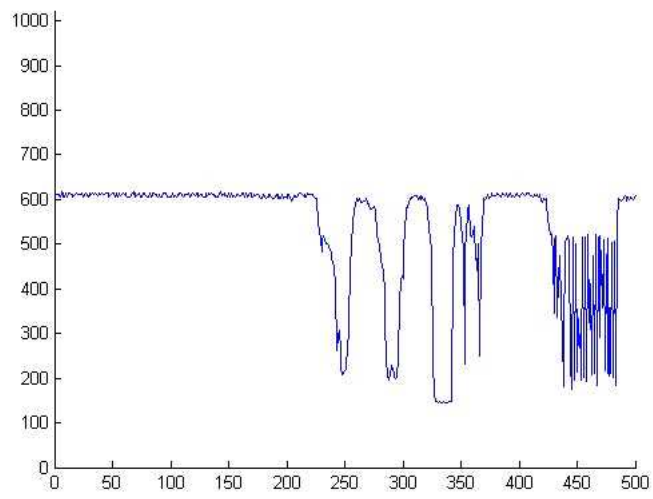
  Serial.println(sensorValue);
  int mapValue = map(sensorValue, 0,
1023, 255, 0);

  analogWrite(ledPin, mapValue);
  delay(500);

}
```



Graph using Matlab



1) 프로세싱으로 시각화

프로세싱은 시리얼 포트를 통해 아두이노가 보내온 가변저항의 값을 이용하여 원을 그리고 해당 원의 크기를 가변 저항의 값으로 그린다. 따라서, 큰 저항 값이 전송되었을 때는 원의 지름이 커지고 반대인 경우에는 작아지게 된다.

가) 프로그램

- 가변저항에서 읽은 저항 값에 따라서 LED의 점멸 주기를 변경 시킨다.

```
import processing.serial.*;

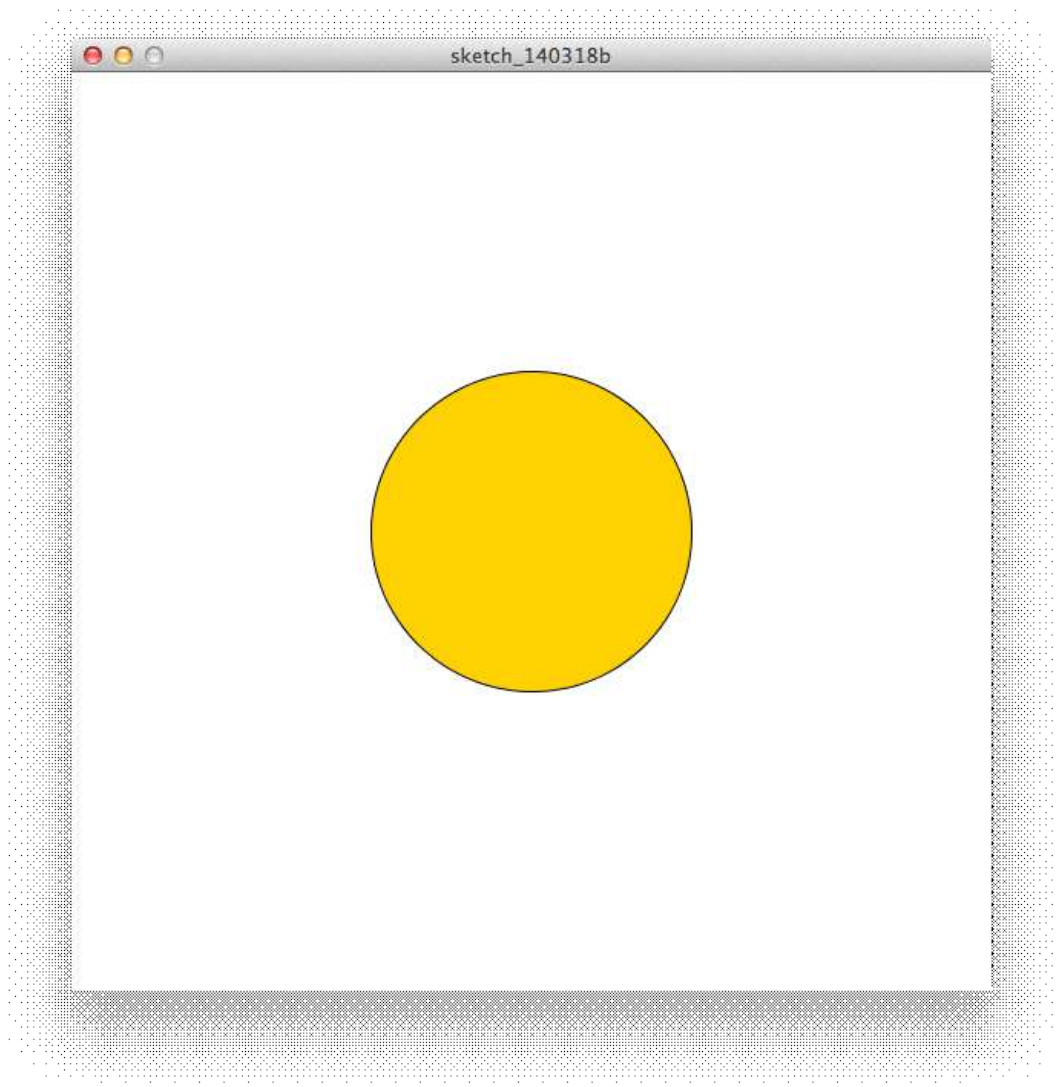
int lf = 10; // Linefeed in ASCII
String myString = null;
Serial myPort; // Serial port you are using
float num;

void setup() {
  background(255,255,255);
  size(600,600);
  println(Serial.list());
  myPort = new Serial(this, Serial.list()[4], 9600); //자신의 포트에 맞는 번호 선택
}

void draw() {
  while (myPort.available() > 0) {
    myString = myPort.readStringUntil(lf);
    if (myString != null) {
      num=float(myString); // Converts and prints float
      background(255,255,255);
      fill(#ffcb01);
      ellipse(width/2,height/2,num/2,num/2);
    }
  }
  myPort.clear();
}
```

나) 실행결과

- 아두이노의 시리얼 모니터 창을 닫아야 한다. 한 번에 하나의 프로그램만이 시리얼 모니터를 이용할 수 있다. 빛 센서의 밝기 변화에 따라 프로세싱은 서로 다른 크기의 원을 표시해주게 된다.



2) 엑셀을 이용한 데이터의 표현

아두이노가 보내오는 데이터를 처리하기 위해서는 컴퓨터 상에서 다양한 프로그램을 이용할 수 있다. 엑셀은 숫자 데이터의 처리에 적합하며 각종 함수를 이용하여 데이터 처리 및 표 작업등에 유용하게 사용할 수 있다. 이번 작업에서는 엑셀을 이용하여 아두이노가 보내온 조도 센서 데이터의 처리를 한다. 이 작업은 앞으로의 다양한 추가 작업에 공통적으로 이용될 수 있다.

가) 프로그램

parallax daq는 아두이노와 엑셀을 연결할 수 있는 프로그램으로 아두이노 코드상에서 엑셀에 어떠한 데이터를 출력할지 정하며 컴퓨터와 아두이노의 시리얼통신을 이용한다. 실행 파일을 클릭하여 parallax를 설치하며, 매크로가 포함된 엑셀파일을 이용하여 아두이노의 데이터를 처리한다.

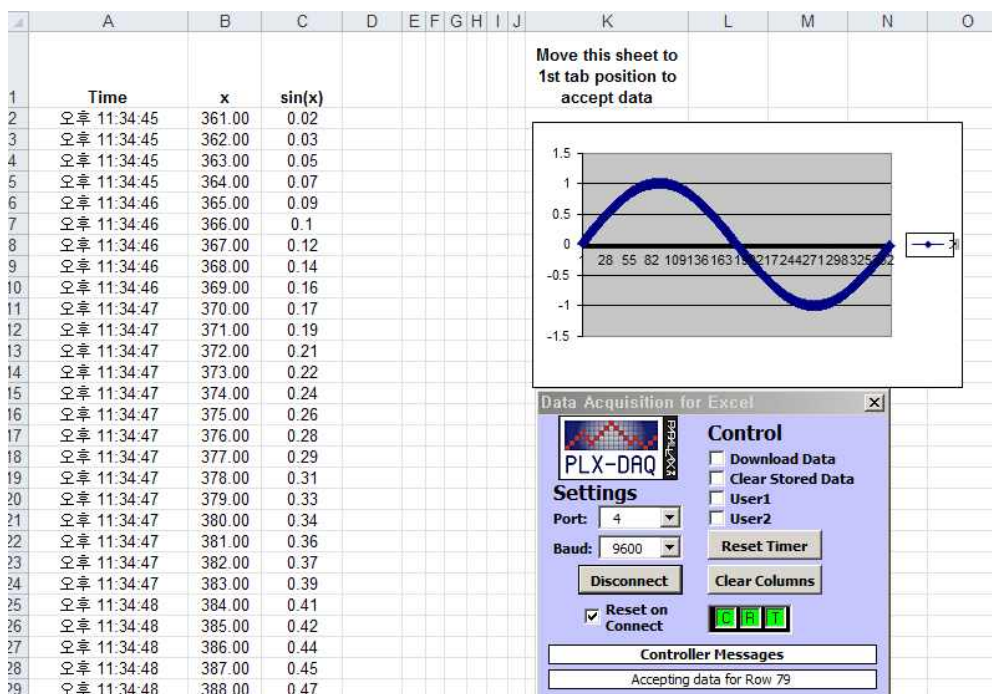


```

int x = 0;
int row = 0;
void setup() {
  Serial.begin(128000); // opens serial port, sets data rate to 9600 bps
  Serial.println("CLEARDATA");
  Serial.println("LABEL,Time,x,sin(x)");
}
void loop() {
  Serial.print("DATA,TIME,"); Serial.print(x); Serial.print(","); Serial.println(sin(x*PI/180));
  row++;
  x++;
  if (row > 360)
  {
    row=0;
    Serial.println("ROW,SET,2");
  }
  delay(100);
}

```

Serial.println("CLEARDATA");는 데이터를 보내기 전 엑셀에 있는 모든 데이터를 지우겠다는 것이며 Serial.println("LABEL,Time,x, sin(x));는 엑셀의 맨 윗줄에 라벨을 다는 것입니다. Loop 함수에서는 DATA, TIME 로 시작하는 다음 줄에 실제 데이터가 들어오게 된다. 데이터의 행의 수가 360이 넘으면 행의 값을 2로 하여 다시금 데이터가 입력되게 해주는 코드이다.



마. 진동 감지 및 소리 출력

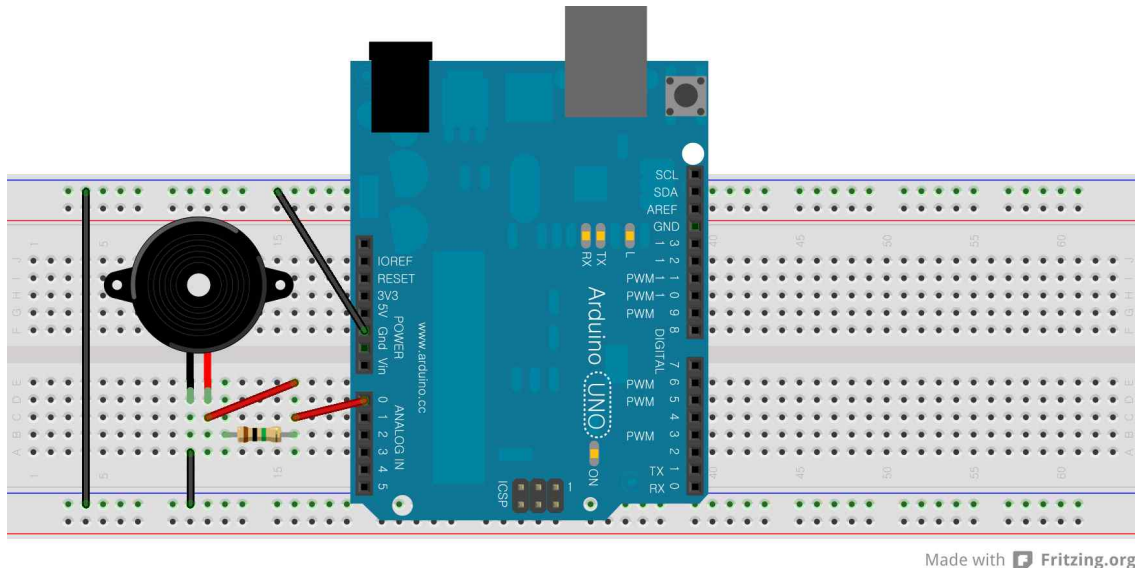
1) 진동 감지하기

피에조 센서는 소리를 내는 출력의 기능 이외에도 충격을 감지하는 센서로의 활용이 가능하다. 이번 프로젝트에서는 외부의 충격을 감지해서 LED를 점멸하는 코드를 작성해보고, 소리를 출력하는 기능은 추후에 작업하도록 한다. 피에조 센서는 압전소자로서, 2개의 면에 전압을 가하면 전압에 비례한 변형이 발생하여 소리가 나거나, 압력이나 비틀림이 주어지면 반대로 전압이 발생하는 소자이다.

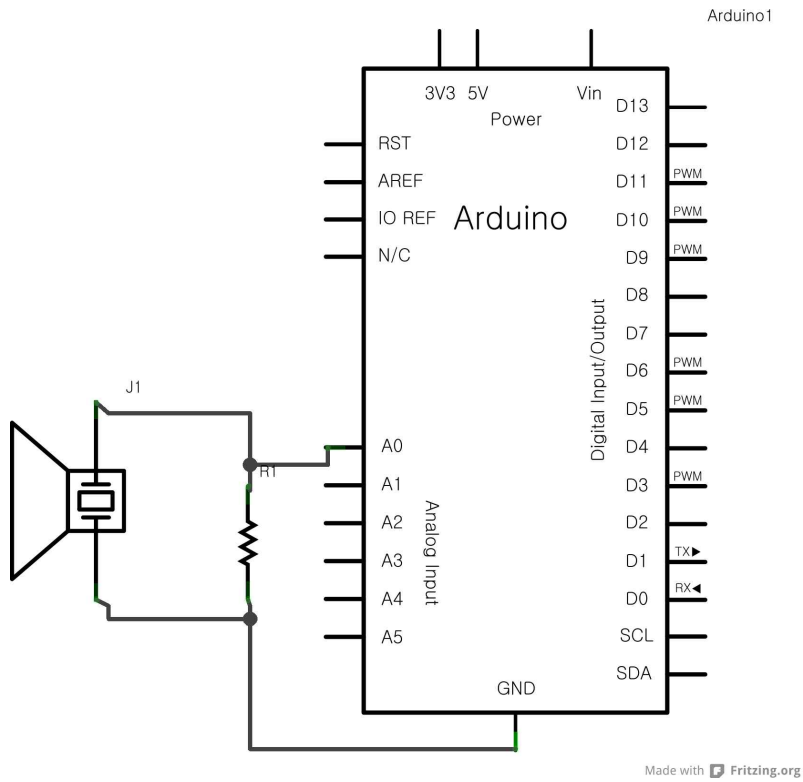


브레드보드

- 피에조는 극성을 띄고 있으므로, 극성에 주의해서 연결한다. 일반적으로 빨간색이 +임을 명시한다. 센서에는 1M옴의 높은 저항을 병렬로 연결해주고 아날로그 입력에 연결을 해준다. 아날로그 입력 역시 전압의 변화 값으로 읽혀지게 되며 0~5V 사이의 변화 값은 10비트 해상도에 따라 0~1023의 정수 값으로 변환되어 입력이 된다. 기준치를 넘어서는 압력이 가해지면 아두이노에 연결되어 있는 13번 LED를 On 시킨다. 13번 포트는 기본적으로 아두이노의 보드에 연결된 소형 LED에 연결이 되어 있다.



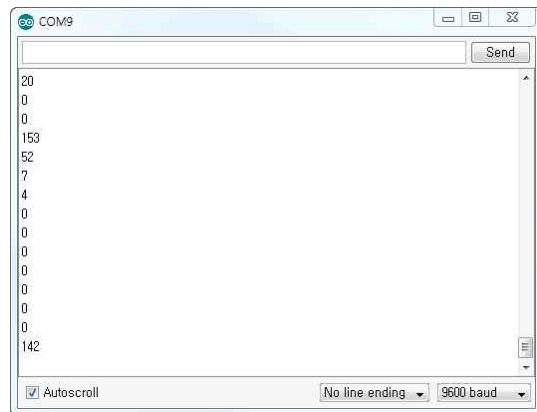
스케메틱



프로그램

```
int sensorPin = 0;
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int val = analogRead(sensorPin);
    Serial.println(val);
}
```



2) 도전과제

특정 수치를 넘어서면 LED를 연결하여 가시화 시켜보자.

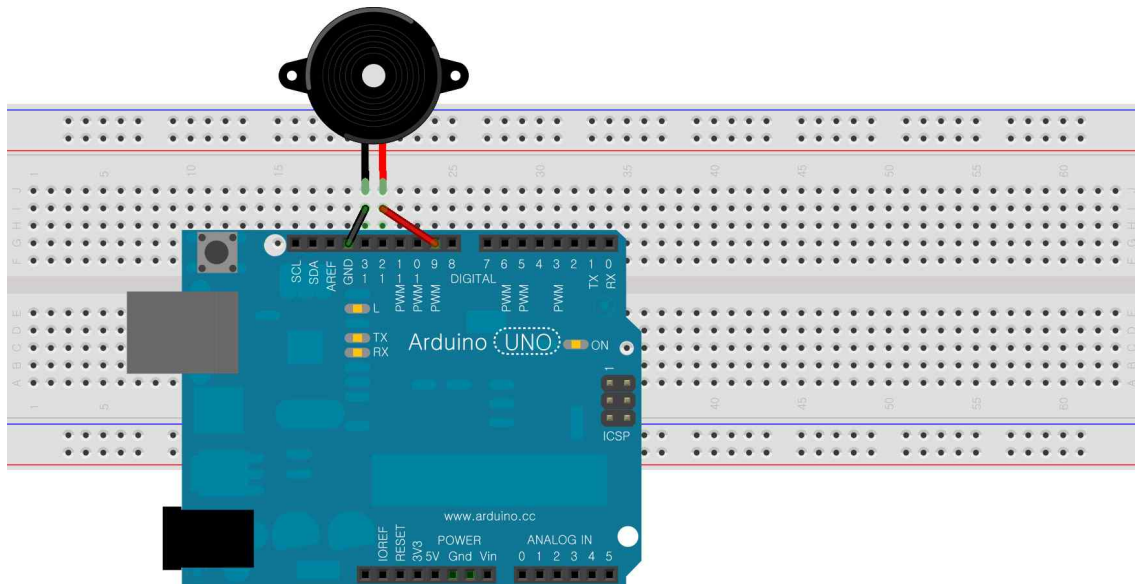
옆 사람과 2개 이상의 진동 감지 센서를 이용하여 다중의 LED를 가시화 시켜보자.

3) 소리 출력하기

이전 프로젝트에서는 피에조 센서를 입력으로 사용했다. 피에조 센서는 출력으로 소리를 낼 수 있으며 아두이노는 스피커와 같은 출력 장치를 통해 사운드를 출력할 수 있다. 소리는 공기의 진동을 통해서 만들어지게 되며, 아두이노는 펄스를 입력 받아 소리를 발생시키는 소형 세라믹 변환기로 이루어진 피에조 장치를 통해서 소리를 만들어 낼 수 있다. 소리의 주파수는 스피커에서 한 차례 진동이 발생하는 시간에 따라 결정되며, 시간이 짧을수록 주파수는 높아진다. 동일한 하드웨어를 변경하여 A0에 들어오는 임의의 값을 스피커로 출력해 보자. 아날로그 출력에 관한 것은 아날로그 출력 프로젝트에서 더 세부적으로 다룬다.

브레드보드

- 피에조는 극성을 띄고 있으므로, 극성에 주의해서 연결한다. 일반적으로 빨간색이 +임을 명시한다. 아두이노의 9번 포트에 피에조의 +극을 연결해준다. 아두이노의 라이브러리가 제공하는 함수를 사용하여 소리를 출력한다. 보다 안정된 동작을 위하여 피에조의 + 측에 100옴 정도의 저항을 연결해 준다. 아래 그림에서는 표시가 되어 있지 않다.



Made with  Fritzing.org


```
const int speakerPin = 9;
const int sensorPin = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(sensorPin);
  int frequency = map(sensorValue, 0, 1023, 100, 5000);
  int duration = 250;
  tone(speakerPin, frequency, duration); //스피커핀, 주파수, 재생시간(milli second)
  Serial.println(sensorValue);
  delay(100);
}
```

위 코드는 아날로그 입력에서 들어오는 의미 없는 값을 통해서 소리를 재생시킬 수 있는 주파수 성분을 선정하는 코드이다. 물론 듣기 좋은 음악은 아니다.

가) 간단한 멜로디 재생하기

```
const int speakerPin = 9; // connect speaker to pin 9

char noteNames[] = {'C','D','E','F','G','a','b'};
unsigned int frequencies[] = {262,294,330,349,392,440,494};
const byte noteCount = sizeof(noteNames); // the number of notes
                                         // (7 in this example)

//notes, a space represents a rest
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
const byte scoreLen = sizeof(score); // the number of notes in the score

void setup()
{
}



void loop()
{
    for (int i = 0; i < scoreLen; i++)
    {
        int duration = 333; // each note lasts for a third of a second
        playNote(score[i], duration); // play the note
    }

    delay(4000); // wait four seconds before repeating the song
}

void playNote(char note, int duration)
{
    // play the tone corresponding to the note name
    for (int i = 0; i < noteCount; i++)
    {
        // try and find a match for the noteName to get the index to the note
        if (noteNames[i] == note) // find a matching note name in the array
            // play the note using the frequency:
            tone(speakerPin, frequencies[i], duration);
    }
    // if there is no match then the note is a rest, so just do the delay
    delay(duration);
}
```

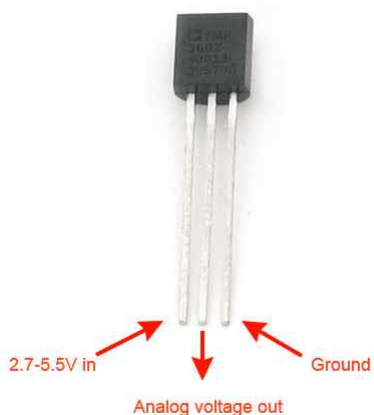
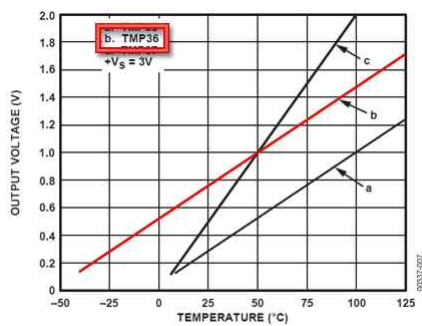
바. 온도 센서

일기예보에서는 내일의 날씨를 나타낼 때 대부분 온도 정보를 함께 보여준다. 온도는 물질의 뜨겁고 찬 정도를 나타내는 것으로서, 일상생활에서 많이 활용이 된다. 온도는 켈시우스 온도계나 디지털 온도계를 통해 손쉽게 측정이 가능하다. 아두이노에서 편리하게 사용할 수 있는 아날로그 온도 센서를 이용하여 외부의 온도를 측정하고 그 결과를 시리얼 모니터를 통해서 확인해 보자. 이를 통해 연속된 데이터인 아날로그 값을 아두이노가 처리 하는 방식과 온도를 나타내기 위하여 변환 과정을 거치는 것을 이해할 수 있다.

켈시우스 온도계	디지털 온도계
	

1) 온도 센서

온도 센서(TMP36)는 다음과 같은 구조를 가지며 전원 인가를 위한 핀과, 접지를 위한 핀 그리고 온도를 알 수 있도록 아날로그 전압을 출력해주는 3개의 핀으로 구성되어 있다.

TMP36	온도와 전압 특성
	 Figure 6. Output Voltage vs. Temperature

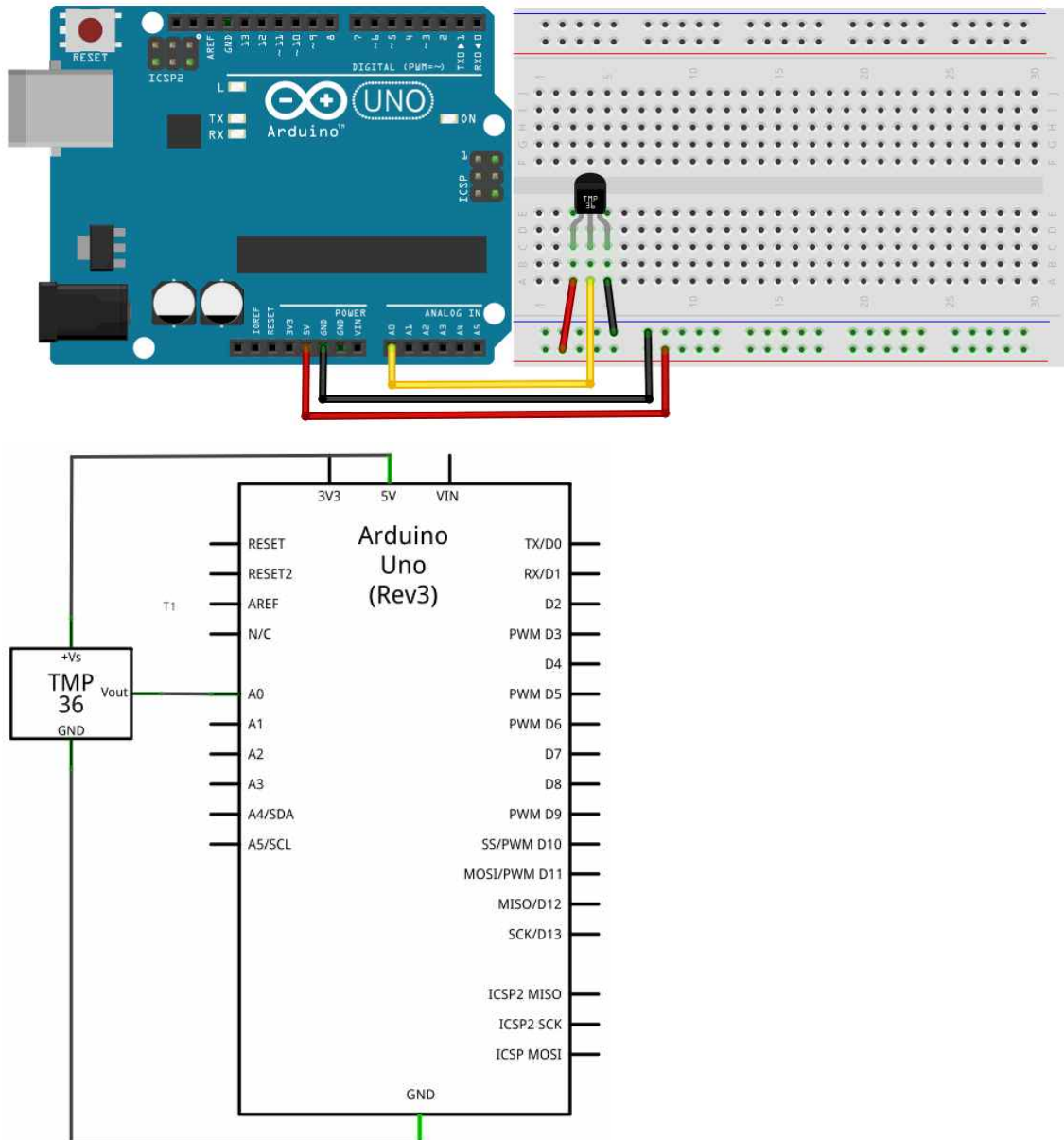
TMP36은 비교적 저렴한 아날로그 온도 센서로서 추가적인 회로를 구성하지 않고도 간편하게 온도를 측정할 수 있다. TMP36은 섭씨 -40°C ~ 150°C 까지의 온도를 측정할 수

있으며 분해능은 $10\text{mV}/^{\circ}\text{C}$ 이며 편리하게 온도 측정이 가능하다. 오차는($\pm 2^{\circ}\text{C}$) 정도가 발생한다. TMP36은 아래와 같은 특성을 지니며, 온도가 올라갈수록 전압이 증가하며, 섭씨 50도에서 1.0V 전압이 측정된다.

연속된 아날로그 신호를 아두이노 내부로 입력하는 과정에서 아날로그 데이터는 디지털로 변환이 일어나며, 아두이노의 아날로그 입력은 입력 값을 0~1023의 디지털 값으로 변환해서 보여주는 10bit 분해능을 갖추고 있다. 따라서, 입력 값을 기준 전압인 5V의 상대적인 위치값으로 변환을 한 다음 데이터시트상에서 온도를 측정하기 위해 Offset 값의 조정을 통해 원하는 온도 값을 알아낼 수 있게 된다.

2) 브레드 보드

VCC(왼쪽)는 아두이노의 5V에, GND(오른쪽)는 아두이노의 GND에 그리고 신호선(가운데))은 아두이노의 아날로그 0번 핀에 연결한다.



3) 프로그램 1

스케치를 아두이노 IDE에서 작성하고 Upload 한다.

```
01 int sensorPin = 0;
02 void setup()
03 {
04     Serial.begin(9600);
05 }
06 void loop()
07 {
08     int reading = analogRead(sensorPin);
09     float voltage = reading * 5.0;
10     voltage /= 1024.0;
11     float temperatureC = (voltage - 0.5) * 100 ;
12     Serial.println(temperatureC);
13     delay(1000);
14 }
```

[스케치 설명]

01 센서가 연결되어 있는 아날로그 포트인 0번 포트를 위한 변수 생성

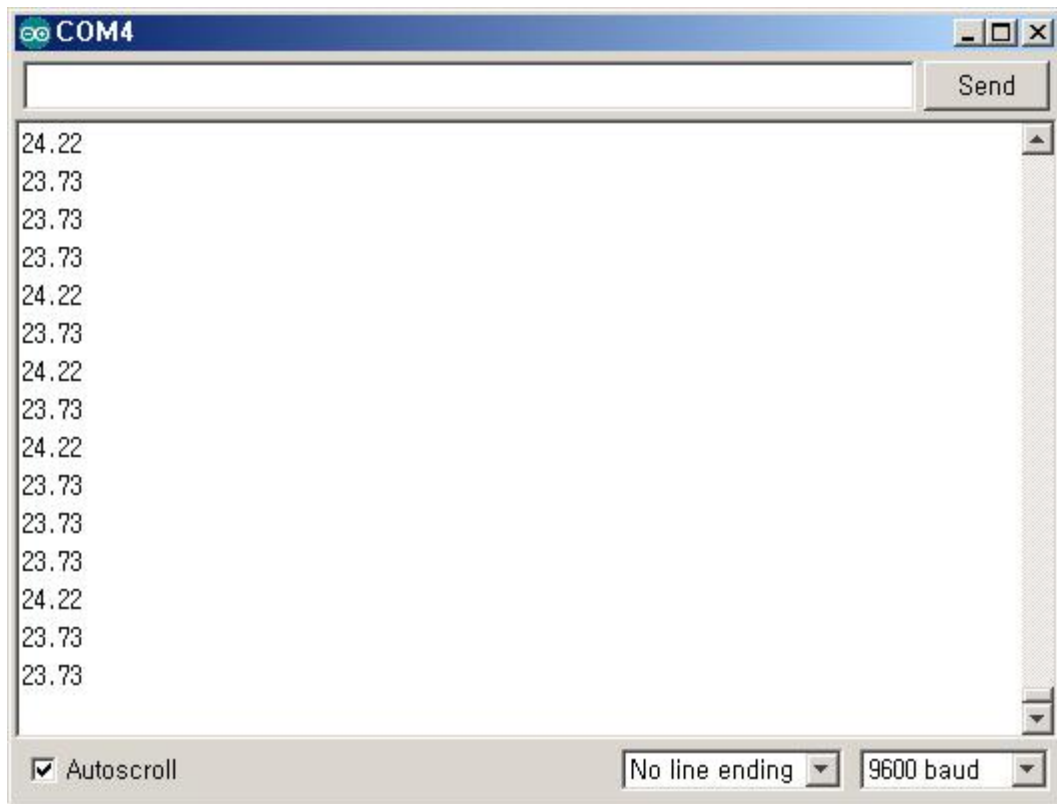
04 시리얼 통신을 위한 기본 속도를 설정해 준다.

08 A0에 들어오는 아날로그 값을 읽기 위한 함수

09-11 측정된 전압을 온도로 변환시키는 작업

12 측정된 온도를 시리얼을 통해 컴퓨터로 전송

13 1초간 지연 시킨 후에 다시 loop 함수가 시작되도록 해준다.



4) 프로그램 2

시리얼을 통한 온도 입력자료를 시각적으로 표현하기 위해 프로세싱을 이용하여 그래프로 표현한다.



```

import processing.serial.*;

Serial myPort;
int xPos = 1;
void setup () {
    size(400, 300);
    println(Serial.list());
    myPort = new Serial(this, Serial.list()[1], 9600);
    myPort.bufferUntil('\n');
    background(0);
}
void draw () {

void serialEvent (Serial myPort) {
    String inString = myPort.readStringUntil('\n');

    if (inString != null) {
        inString = trim(inString);
        float inByte = float(inString);
        fill(0);
        noStroke();
        rect(0, 0, 70, 50);
        fill(127, 34, 255);
        textFont(font12);
        textAlign(RIGHT);
        text(inString + " C" , 65, 25);
        stroke(127,34,255);
        line(xPos, height, xPos, height - inByte*7);
        if (xPos >= width) {
            xPos = 0;
            background(0);
        }
        else {
            xPos++;
        }
    }
}
}

```

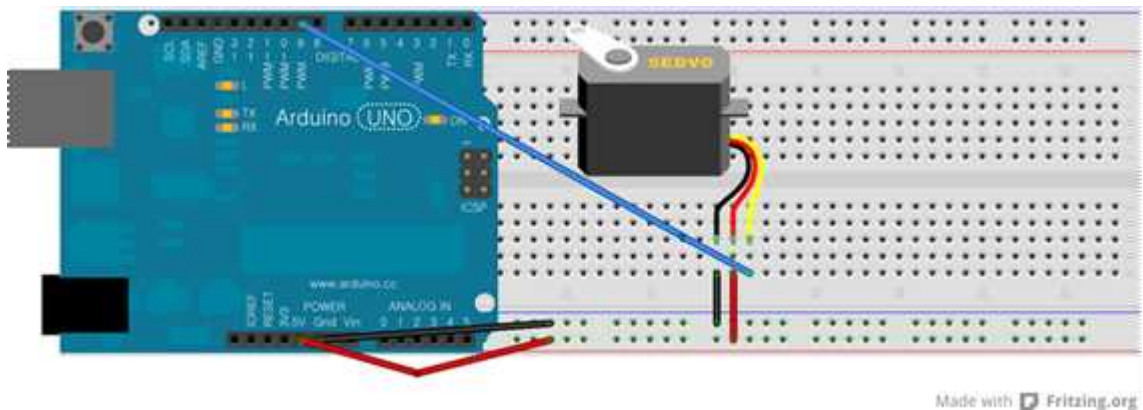
6. 아날로그 출력

가. 서보모터 제어

DC 모터는 단순히 전류를 흘려주면 전류가 공급되는 동안 모터는 회전을 하게 된다. 이에 반해 서보모터는 지속적으로 회전하는 것이 아니라, 자신이 원하는 위치로 모터를 이동시킬 수 있어 보다 정밀한 제어가 가능하게 된다. 서보 모터를 한쪽 끝에서 나머지 끝까지 이동시키고 다시 그 반대 동작을 지속적으로 하게 하는 작업이다.

1) 하드웨어 연결하기

서보모터의 전원(빨간색), GND(검은색)을 아두이노의 전원과 GND에 연결하고, 나머지 한 선을 아두이노의 9번 포트에 연결한다. 대부분의 서보모터는 일정한 각도 이상으로 회전을 할 수는 없다. 본 프로젝트에서 사용하는 모터는 180도까지만 회전을 하며, 아두이노의 PWM을 통해 각도를 조정할 수 있다. delay()함수를 같이 사용하여 회전 속도를 조절하는 것도 가능하다.



2) 프로그램 작성

```
#include <Servo.h>

int servoPin = 9;
Servo servo;//모터 객체 선언
int angle = 0;

void setup() {
    servo.attach(servoPin);
}

void loop() {
    for (angle = 0; angle < 180; angle++) {
        servo.write(angle);
        delay(15);
    }
    for (angle = 180; angle > 0; angle--) {
        servo.write(angle);
        delay(15);
    }
}
```

모터를 컨트롤 하기 위해 복잡한 코드를 작성하는 대신 Servo.h 라이브러리를 이용하여 보다 편리하게 모터를 제어할 수 있다.

`servo.attach(servoPin);`

모터가 연결된 핀을 설정하여 준다. 서보 모터는 PWM 제어를 사용하기에 보드에서 PWM을 지원하는 핀에 연결하여 준다.

`servo.write(angle);`

모터가 회전할 각을 입력해준다.

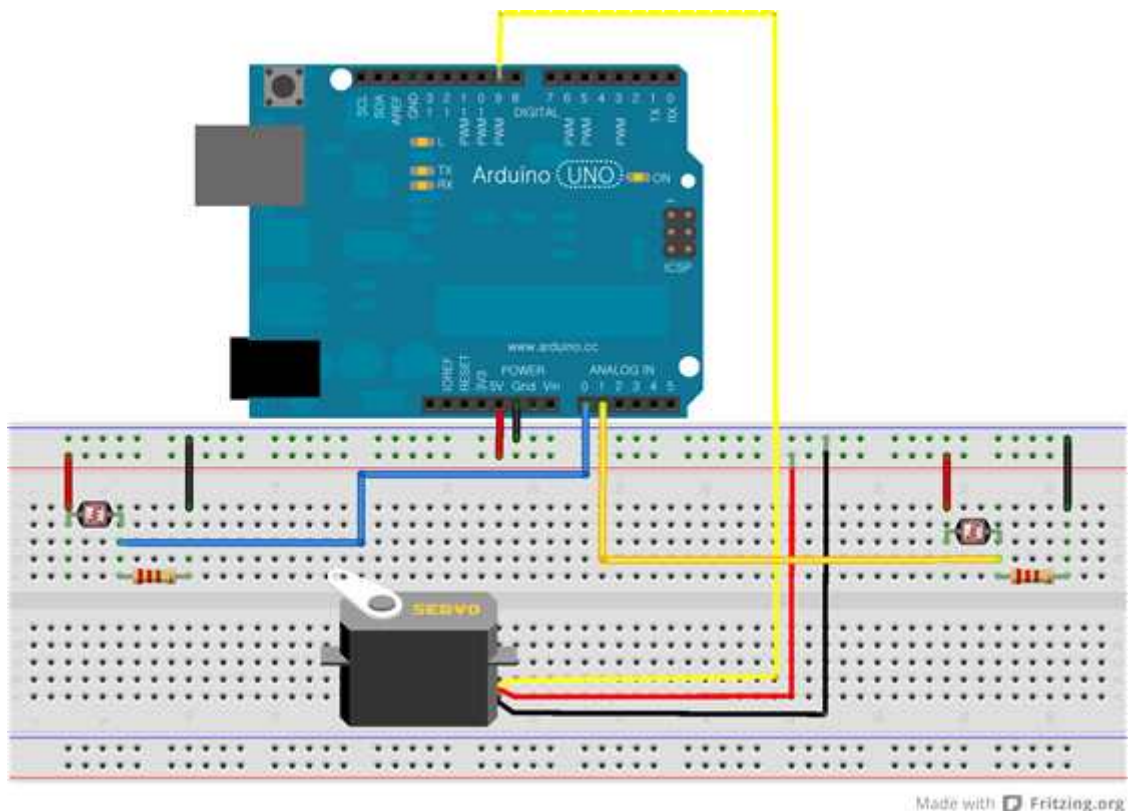
위 코드는 2개의 반복문을 통해 한쪽 0도에서 179도까지 그리고 180도에서 1도 까지 회전하는 코드이다.

3) 빛을 추적해보기

서보 모터의 기본적인 사용 방법을 이전 수업에서 학습하였다. 빛 센서 2개를 브레드 보드의 양단에 연결하고 서보모터를 이용하여 양쪽의 빛의 차이를 계산하여 해당 방향으로 모터를 회전 시키는 프로젝트이다.

가) 하드웨어 연결하기

서보모터의 전원(빨간색), GND(검은색)을 아두이노의 전원과 GND에 연결하고, 빛 센서 2개를 브레드 보드의 양단에 그림과 같이 설치하며, 10K옴 저항 2개를 직렬로 연결 시켜 준다. 양단의 저항에 걸리는 전압을 A0, A1에 연결하여 analogRead 함수를 통하여 그 값을 읽어준다.



나) 프로그램

```
#include <Servo.h>

int left = A0;
int right = A1;
int sensorValue = 0;
Servo myservo;
int value;
int center = 100;

void setup() {
    myservo.attach(9);
}

void loop() {
    int left = analogRead(A0); //왼쪽 빛 센서
    int right = analogRead(A1); //오른쪽 빛 센서
    value = (left - right) / 10;
    if (value == 0)
        myservo.detach();
    else
        myservo.attach(9);

    if (value > 10)
        value = -70;
    else
        value = 70;
    myservo.write(center+value);
    delay(10);
}
```

7. 시각적 출력

가. 16*2 Character LCD 출력

텍스트 LCD는 영문자와 숫자들을 간편하게 표시할 수 있는 출력 장치이다. 아두이노에서 값을 확인하기 위해 매번 시리얼 모니터를 이용했던 불편함을 해소할 수 있으며 보다 적극적인 표현이 가능하게 된다. 반면 그래픽이나 한글 문자등의 표현은 상당히 불편하며 이를 위해서는 직접 글꼴등을 디자인해야 한다. 아두이노에서는 텍스트 LCD를 제어하기 위한 라이브러리를 제공하며 이를 통해 보다 편리하게 문자를 출력할 수 있다. 우리가 사용할 LCD는 16글자를 2줄로 표현해주며 손쉽게 구할 수 있다는 장점이 있다.

LCD에는 총 16개의 핀이 있으며, 각각의 핀은 다음과 같은 의미를 가진다. 실제 데이터의 전송이 이루어지는 핀은 7~14번의 총 8핀이며, 나머지 핀은 전원 공급과 백라이트 조정 혹은 레지스터 선택 등에 사용이 된다.

RS: LCD를 제어하기 위해 사용되는 레지스터를 선택해주는 핀이다. 명령을 가지고 있는 레지스터와 데이터를 가지고 있는 레지스터 중 하나를 선택한다.

R/W: 읽기 및 쓰기 모드를 선택한다. 우리는 쓰기 모드만을 사용할 것이므로 이 핀을 GND에 연결하여 준다.

E : 데이터가 준비되었을 경우 전송을 시작하기 위해 사용이 된다.

6. TERMINAL PIN FUNCTION

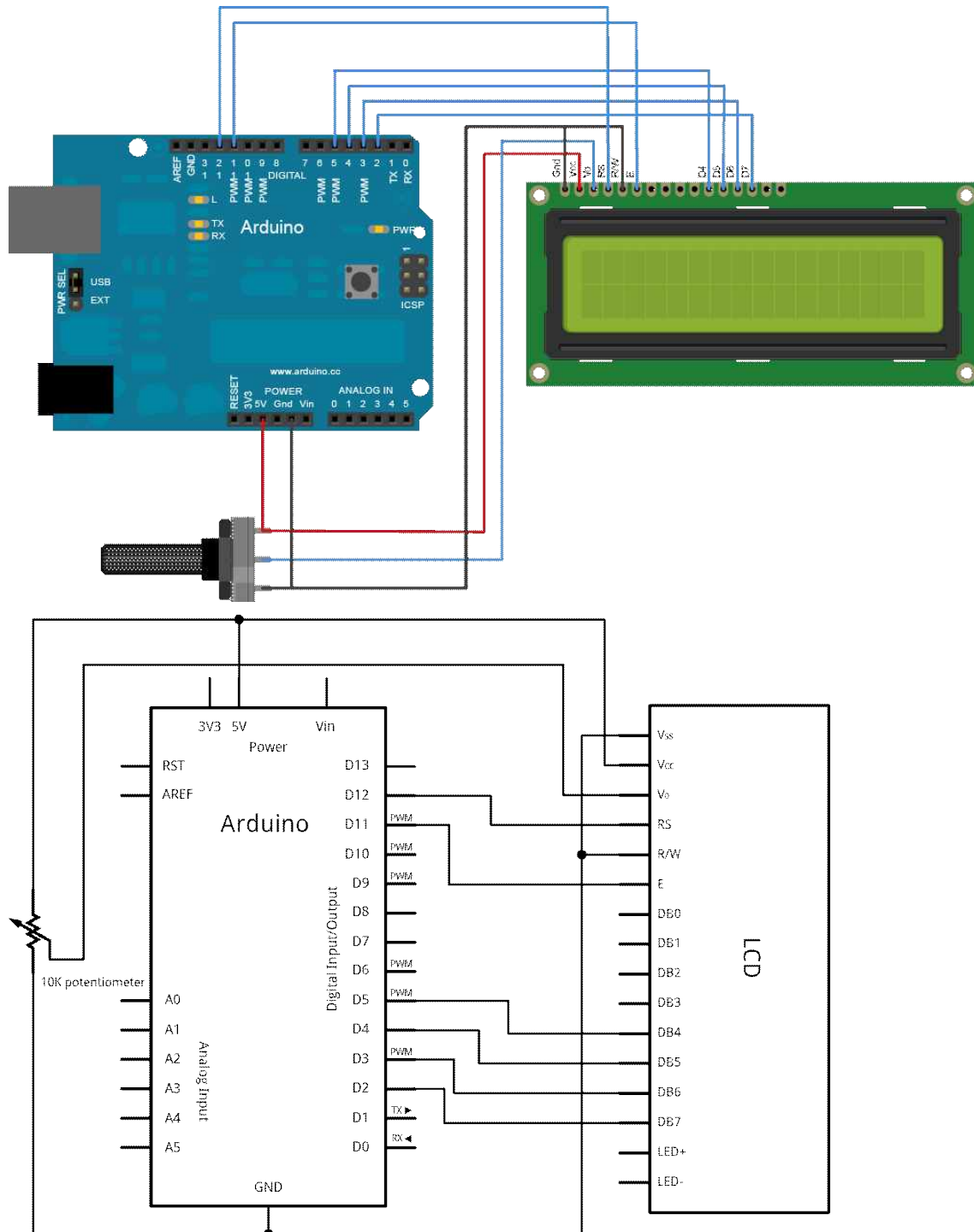
6-1. Interface Pin Function Description

Pin NO.	Symbol	I / O	Functions
1	Vss	Power	GND
2	Vdd	Power	Power supply for logic circuit
3	V0	Power	Contrast adjustment
4	RS	I	Register select signal
5	R/W	I	Used as read/write selection input when Rw="high" read operation Rw="Low", write operation
6	E	I	Enable signal
7	DB0	I/O	Data bus
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		
15	A	-	BACKLIGHT(+)
16	K	-	BACKLIGHT(-)

3번핀은 화면의 밝기를 조절하기 위해 사용이 되며, 15,16핀은 백라이트를 켜기 위해서 사용이 된다.

아두이노가 제공해주는 LCD 라이브러리는 4핀이 데이터 핀을 이용하여 화면에 글자를 표현할 수 있도록 해주어 필요한 데이터의 선의 수를 줄일 수 있는 장점이 있다.

1) 하드웨어 연결하기

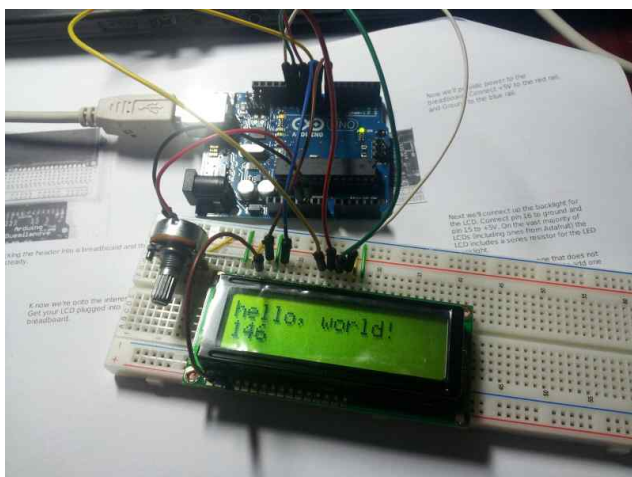


핀 번호를 정리하면 다음과 같다.

LCD 핀	기능	아두이노핀
1	GND	GND
2	+5V	5V
3	V0또는 (가변저항)	
4	RS	12
5	R/W	GND
6	E	11
7	D0	
8	D1	
9	D2	
10	D3	
11	D4	5
12	D5	4
13	D6	3
14	D7	2
15	+5V	
16	GND	

2) 프로그램

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  lcd.begin(16, 2);//16행 2열
  lcd.print("hello, world!");
}
void loop() {
  lcd.setCursor(0, 1);//0번째열 1번째 행
  lcd.print(millis()/1000);
}
```



3) 다양한 지원 함수들

<http://arduino.cc/en/Reference/LiquidCrystal?from=Tutorial.LCDLibrary>

에는 해당 라이브러리가 지원하는 다양한 함수들에 대한 자세한 설명을 볼 수 있다.

Function

LiquidCrystal()

begin()

clear()

home()

setCursor()

write()

print()

cursor()

noCursor()

blink()

noBlink()

display()

noDisplay()

scrollDisplayLeft()

scrollDisplayRight()

autoscroll()

noAutoscroll()

leftToRight()

rightToLeft()

createChar()

4) 프로그램2

Serial Output

```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup(){
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // initialize the serial communications:
    Serial.begin(9600);
}

void loop()
{
    // when characters arrive over the serial port...
    if (Serial.available()) {
        // wait a bit for the entire message to arrive
        delay(100);
        // clear the screen
        lcd.clear();
        // read all the available characters
        while (Serial.available() > 0) {
            // display each character to the LCD
            lcd.write(Serial.read());
        }
    }
}
```