

6. 아날로그 데이터 입출력

아두이노 우노에 사용되는 ATmega328 마이크로컨트롤러에는 아날로그 데이터를 위해 6 채널의 아날로그-디지털 변환기(Analog-to-Digital Converter, ADC)가 포함되어 있으며 각 채널은 A0에서 A5까지의 아날로그 핀에 대응한다. 아날로그 핀에 대응하는 상수는 디지털 핀을 나타내는 값인 0에서 13번 이후 A0가 14번, A1이 15번 등으로 정의되어 있으며 필요한 경우 아날로그 입력 핀은 디지털 입출력 핀으로도 사용이 가능하다.

```
static const uint8_t A0 = 14;
static const uint8_t A1 = 15;
static const uint8_t A2 = 16;
static const uint8_t A3 = 17;
static const uint8_t A4 = 18;
static const uint8_t A5 = 19;
```

ATmega328 마이크로컨트롤러에는 디지털-아날로그 변환기(Digital-to-Analog Converter, DAC)가 포함되어 있지 않으므로 아날로그 데이터를 출력할 수는 없다. 하지만 아날로그 데이터와 유사한 효과를 내는 펄스 폭 변조(Pulse Width Modulation, PWM) 디지털 신호 출력을 지원한다. PWM 신호 출력을 위한 함수의 이름도 analogWrite로 'analog'라는 단어가 포함되어 있지만 실제로 출력되는 신호는 디지털 신호라는 점에 유의하여야 한다. 표 1은 아두이노 우노에서 사용할 수 있는 아날로그 및 디지털 데이터 입출력 방법을 요약한 것으로 이 장에서는 아날로그 데이터 입력 방법과 PWM 신호 출력 방법에 대해 알아본다.

	입력	출력
디지털	<pre>int pinNo = 13; pinMode(pinNo, INPUT); int value = digitalRead(pinNo);</pre>	<pre>int pinNo = 13; int value = HIGH; pinMode(pinNo, OUTPUT); digitalWrite(pinNo, value);</pre>
아날로그	<pre>int pinNo = A0; pinMode(pinNo, INPUT); int value = analogRead(pinNo);</pre>	<pre>int pinNo = 3; int dutyCycle = 128; pinMode(pinNo, OUTPUT); analogWrite(pinNo, dutyCycle);</pre>

표 1. 디지털 및 아날로그 데이터 입출력 함수

1. 아날로그 데이터 입력

아두이노 우노에는 아날로그 데이터 입력을 위해 6개의 아날로그 입력 핀이 준비되어 있다. 아날로그 입력 핀에 아날로그 데이터를 출력하는 장치를 연결하면 ADC를 거쳐 디지털로 변환된 아날로그 값이 마이크로컨트롤러로 입력된다. 아두이노 우노에 사용되는 ATmega328 마이크로컨트롤러의 ADC는 10비트의 해상도를 가진다. 즉, 입력되는 아날로그 값을 10비트의 디지털 값으로 변환하여 반환하므로 0에서 1023 사이의 값을 반환한다. 디지털 데이터의 경우 1비트 단위로 데이터가 입출력되는 반면 아날로그 데이터의 경우에는 한 번의 함수 호출로 10비트의 데이터가 읽혀진다는 점에서 다르다. 아날로그 입력을 읽는데 걸리는 시간은 약 $0.0001\text{s} = 100\mu\text{s}$ (microsecond)이므로 초당 최대 10,000개 아날로그 입력을 받아들일 수 있다. 아날로그 데이터 입력을 위해 사용하는 함수는 analogRead 함수이다.

```
int analogRead(uint8_t pin)
    pin : 핀 번호
    반환값 : 0에서 1023 사이의 정수값
```

아날로그 데이터를 읽을 때 고려하여야 할 점은 기준 전압의 설정이다. 기준 전압은 analogRead 함수가 반환할 수 있는 최대값인 1023에 해당하는 전압으로 그 이상의 전압이 입력되면 보드가 손상될 수 있으므로 주의하여야 한다. 기본적으로 기준 전압은 아두이노 우

노 보드가 동작하는 5V를 사용한다. 따라서 아두이노 우노는 $\frac{5}{1024} \approx 4.9mV$ 의 전압 차이를 인식할 수 있으며 기준 전압은 analogReference 함수를 통해 변경이 가능하다.

```
void analogReference(uint8_t type)
    type : DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, EXTERNAL 중 한
           가지
    반환값 : 없음
```

기준 전압으로 사용할 수 있는 옵션에는 다음과 같은 것들이 있다.

- ◆ DEFAULT : 아두이노 보드의 동작 전압인 5V로 설정한다. 3.3V를 사용하는 아두이노 보드의 경우 3.3V로 설정한다.
- ◆ INTERNAL : 내부 기준 전압으로 설정한다. ATmega168이나 ATmega328의 경우 1.1V, ATmega8의 경우 2.56V로 설정된다. Arduino Mega의 경우에는 사용할 수 없다.
- ◆ INTERNAL1V1 : 내부 1.1V를 기준 전압으로 설정한다. Arduino Mega에서만 사용할 수 있다.
- ◆ INTERNAL2V56 : 내부 2.56V를 기준 전압으로 설정한다. Arduino Mega에서만 사용할 수 있다.
- ◆ EXTERNAL : AREF 핀에 인가된 0V ~ 5V 사이의 전압을 기준 전압으로 설정한다.

기준 전압으로 5V 이상을 직접 사용할 수는 없지만 전압 분배를 사용하면 5V 이상의 전압을 입력할 수도 있다. AREF 핀은 외부 기준 전압을 설정할 수 있도록 해주는 핀으로 AREF 핀에는 32kΩ의 내부 저항이 연결되어 있다. 따라서 외부에 32kΩ의 저항을 연결하면 10V까지의 외부 기준 전압을 사용할 수 있다. 하지만 이 경우 ATmega328 마이크로컨트롤러에서 인식할 수 있는 전압의 차이는 0.98mV로 증가하게 된다.

아날로그 데이터 입력을 확인하기 위해 그림 1과 같이 아날로그 3번 A3 핀에 가변 저항을 연결해 보자. DM-UStyle EX 보드에는 A3 핀에 연결된 가변 저항이 포함되어 있다.

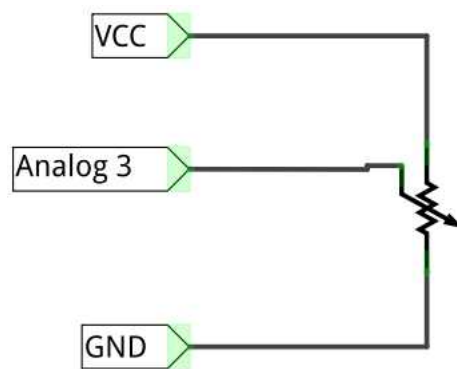


그림 1. 가변 저항 연결 회로도

스케치 1은 가변저항에 가해진 전압을 읽어 이를 시리얼 모니터로 출력하는 스케치의 예이다. 0V 전압이 가해진 경우 출력값은 0, 5V 전압이 가해진 경우 출력값은 1023이다.

스케치 1. 가변 저항

```
int analogPin = A3;    // 가변저항을 아날로그 3번 핀에 연결
                       // 5V와 0V (GND)를 기준 전압으로 사용
int val = 0;           // 입력값 저장을 위한 변수

void setup()
{
    Serial.begin(9600); // 직렬 포트 초기화
}

void loop()
{
    val = analogRead(analogPin); // 아날로그 입력값 읽기
    Serial.println(val);         // 입력값을 직렬 포트에 출력

    delay(500);                 // 0.5초 대기
}
```

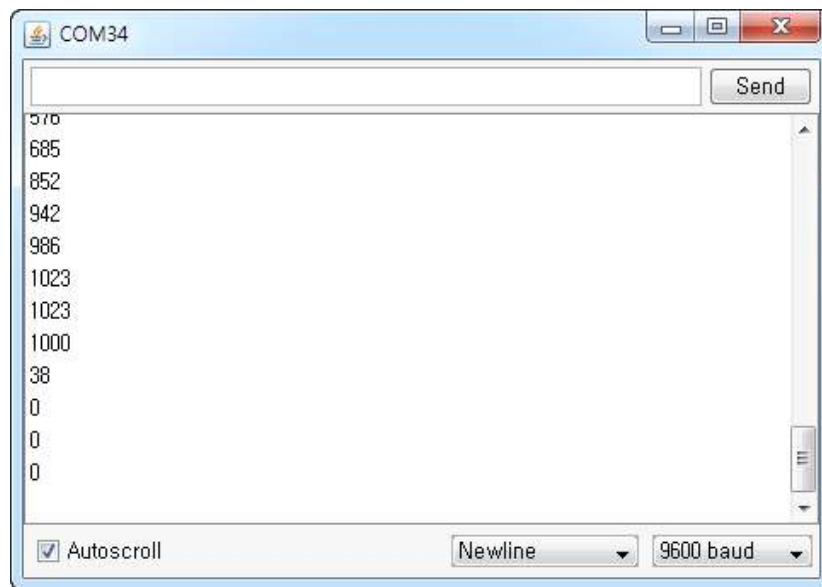


그림 2. 스케치 1 실행 결과

2. 난수 생성

아날로그 입력 핀에 외부 회로가 연결되어 있지 않은 경우 핀은 디지털 입출력 핀과 마찬가지로 플로팅 상태에 있어 `analogRead` 함수가 반환하는 값은 다른 입력 핀에 인가된 전압이나 정전기 등의 영향으로 의미 없는 무작위 값을 반환한다. 반환되는 무작위 값은 난수 생성 함수의 초기화 방법으로 사용될 수 있다. 아두이노에서 난수를 생성하는 함수는 `random`이다. 최소값이 지정되지 않은 `random` 함수에서 최소값은 0으로 설정된다.

```
long random(long max)
long random(long min, long max)
    min : 생성될 난수의 최소값
    max : max - 1이 생성될 난수의 최대값
    반환값 : [min, max - 1] 범위의 난수값
```

`random` 함수는 의사 난수를 생성한다. 의사 난수는 수학적 계산에 의해 생성되므로 시작점이 동일한 경우 항상 동일한 숫자가 순서대로 생성된다. 따라서 `random` 함수를 사용하기 전에는 `random` 함수의 시작점을 무작위로 설정해 주어야 한다. `random` 함수의 무작위 초기화를 위해 사용하는 함수가 `randomSeed` 함수이며 매개변수로 외부 회로가 연결되지 않은

아날로그 입력 핀의 값을 읽어 사용한다.

```
void randomSeed(unsigned int seed)
    seed : 의사 난수의 시작 위치 결정을 위한 값
    반환값 : 없음
```

스케치 2는 random 함수로 1에서 10사이의 정수값을 10개 무작위로 생성하고 시리얼 모니터로 출력하는 스케치이다. DM-UStyle EX 보드의 경우 모든 아날로그 입력 회로에 외부 회로가 연결되어 있지만 아날로그 4번이나 5번 핀의 경우 I2C를 사용하지 않는 경우 무작위 값을 반환하므로 random 함수의 초기화에 사용할 수 있다.

스케치 2. random 함수의 초기화

```
long randNumber;

void setup(){
    Serial.begin(9600);           // 직렬 포트 초기화

    // A4 또는 A5 핀을 random 함수 초기화를 위해 사용한다.
    randomSeed(analogRead(A5));
}

void loop() {
    for(int i = 0; i < 10; i++){
        randNumber = random(1, 11);    // 1~10 사이 난수
        Serial.println(randNumber);
    }

    while(true);
}
```

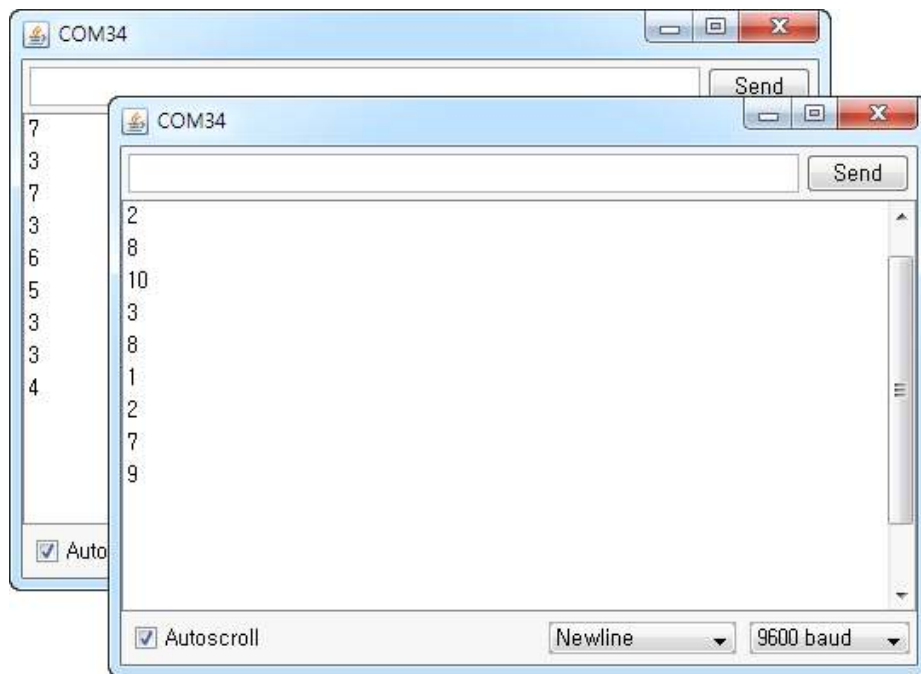


그림 3. 스케치 2 실행 결과

3. 가변저항으로 Blink 속도 조절

Blink 예제의 경우 점멸 속도는 고정되어 있다. 버튼으로 점멸 속도를 바꾸는 스케치를 디지털 데이터 출력 장에서 알아보았다. 여기서는 가변저항을 통해 점멸 속도를 바꾸는 스케치를 작성해 보자. 먼저 그림 4와 같이 디지털 10번 핀에 LED를 연결하고 아날로그 3번 A3 핀에 가변저항을 연결한다.

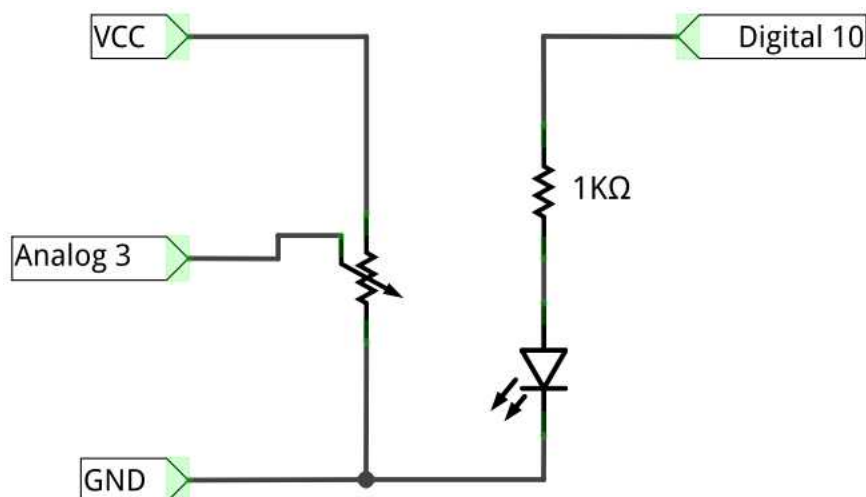


그림 4. 가변저항으로 Blink 속도 조절을 위한 회로도

가변저항을 통해 입력되는 값은 0에서 1023사이의 값이므로 이 값을 점멸 속도로 변환하여야 한다. 이처럼 주어진 범위의 값을 새로운 범위의 값으로 변환하기 위해서는 map 함수를 사용하면 된다.

```
long map(long value, long fromLow, long fromHigh, long toLow, long toHigh)
    value : 데이터
    fromLow : 현재 데이터가 가질 수 있는 최소값
    fromHigh : 현재 데이터가 가질 수 있는 최대값
    toLow : 변환하고자 하는 범위의 최소값
    toHigh : 변환하고자 하는 범위의 최대값
    반환값 : 지정한 범위로 사상된 값
```

map 함수는 주어진 데이터 값을 지정된 범위의 값으로 선형 사상(linear mapping)시킨다. fromLow 값은 toLow 값으로 사상되고 fromHigh 값은 toHigh 값으로 사상된다. 이 때 value는 fromLow에서 toLow 사이의 값이 아니어도 되며 Low와 High 값보다 커도 무방하다. 스케치 3은 가변저항의 값을 100ms에서 1000ms 사이의 점멸 속도로 변환하고 있다.

스케치 3. 가변저항으로 Blink 속도 조절

```
int led = 10; // 10번 핀에 LED 연결
int variableR = A3; // A3 핀에 가변저항 연결

unsigned long time_previous; // 이전 LED 상태 변경 시간
unsigned long time_current; // 현재 시간
boolean state; // LED 점멸 상태
long R; // 현재 가변저항의 값
long interval; // 현재 설정된 점멸 간격

void setup() {
    pinMode(led, OUTPUT); // 10번 핀을 출력 모드로 설정
    pinMode(variableR, INPUT); // A3번 핀을 입력 모드로 설정
```

```
time_previous = millis();           // 이전 LED 상태 변경 시간
state = false;                      // LED가 꺼진 상태에서 시작
interval = 1000;                    // 초기 점멸 시간은 1초
}

void loop() {
    time_current = millis();         // 현재까지의 시간
    R = analogRead(variableR);       // 가변저항 값 읽기
    // 0에서 1023사이의 가변저항 값을
    // 100ms에서 1000ms 사이의 점멸 시간으로 변경
    interval = map(R, 0, 1023, 100, 1000);

    // LED 상태를 변경한 시간이 1000밀리초 이상 경과한 경우
    if(time_current - time_previous > interval){
        time_previous = time_current; // LED 상태 변경 시간 저장
        state = !state;               // LED 상태 반전

        digitalWrite(led, state);     // LED 점멸
    }
}
```

4. PWM 출력

아두이노 우노에 사용된 ATmega328은 아날로그 데이터 입력은 가능하지만 아날로그 데이터 출력은 불가능하다. 대신 펄스 폭 변조 (Pulse Width Modulation, PWM) 신호를 이용하여 아날로그 신호와 유사한 효과를 얻을 수 있다. PWM 신호는 LED의 밝기 조절이나 모터 속도 조절을 위해 많이 사용된다. 아두이노 우노에는 14개의 디지털 입출력 핀이 제공되며 이 중 6개(핀 3, 5, 6, 9, 10, 11)은 PWM 신호 출력을 위해 사용할 수 있다. 디지털 신호는 구형파(square wave)를 기본으로 on(5V)과 off(0V) 상태를 가진다. PWM은 한 주기 내에서 on 상태인 시간의 비율로 나타내며 이를 듀티 사이클(duty cycle)이라 한다. 그림 5는 듀티 사이클에 따라 생성되는 구형파를 나타낸다.

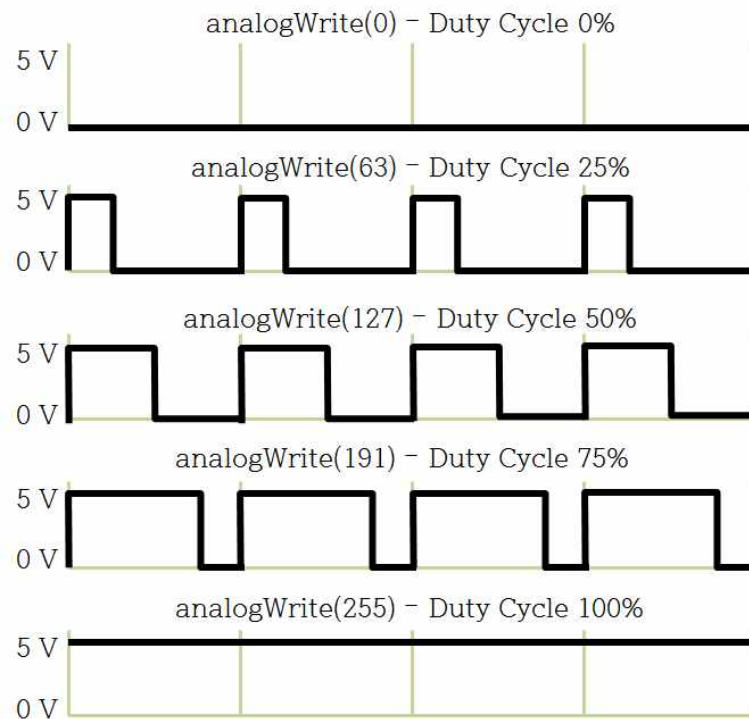


그림 5. 펄스 폭 변조 방식 (Pulse Width Modulation)

PWM 신호를 출력하기 위해서는 디지털 핀에 `analogWrite` 함수를 이용하여 0에서 255 사이의 값을 출력함으로써 서로 다른 듀티 사이클을 가지는 구형파를 출력할 수 있다. 255를 출력하면 듀티 사이클은 100%가 되고 0을 출력하면 듀티 사이클은 0%가 된다. PWM 신호는 디지털 신호이며 이 디지털 입출력 핀으로 출력되지만 PWM 출력을 위해 사용하는 함수는 `analogWrite`임에 유의하여야 한다.

```
void analogWrite(uint8_t pin, int value)
```

pin : 핀 번호

value : 듀티 사이클 (duty cycle). 0(항상 off)에서 255(항상 on) 사이의 값

반환값 : 없음

디지털 10번 핀에 LED를 연결해 보자. 디지털 10번 핀은 PWM 신호 출력이 가능한 핀으로 DM-UStyle EX 보드에는 디지털 10번 핀에 연결된 LED가 포함되어 있다.

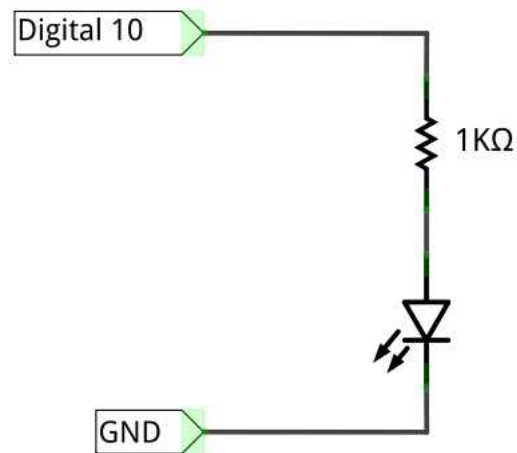


그림 6. PWM 테스트를 위한 회로도

스케치 4는 디지털 10번 핀에 연결된 LED를 완전히 꺼진 상태에서 완전히 켜진 상태로 서서히 변화시키는 스케치의 예이다. 디지털 10번 핀에 연결된 LED가 서서히 밝아지는가?

스케치 4. PWM 신호에 의한 LED 밝기 조절 I

```
int LED_pin = 10;           // LED 연결 핀

void setup()
{
  pinMode(LED_pin, OUTPUT); // 출력 모드 설정
}

void loop()
{
  for(int i = 0; i < 256; i++){
    analogWrite(LED_pin, i); // 0(꺼짐)에서 255(켜짐)로 서서히 변화
    delay(10);
  }
}
```

스케치 4의 경우 LED가 서서히 밝아지다가 완전히 밝아지면 다시 꺼진 상태로 돌아간다. loop 함수에 for 루프를 하나 추가하면 완전히 켜진 상태에서 서서히 완전히 꺼진 상태로 변화하도록 할 수 있다.

스케치 5. PWM 신호에 의한 LED 밝기 조절 II

```

int LED_pin = 10;           // LED 연결 핀

void setup()
{
  pinMode(LED_pin, OUTPUT); // 출력 모드 설정
}

void loop()
{
  for(int i = 0; i < 256; i++){
    analogWrite(LED_pin, i); // 0(꺼짐)에서 255(켜짐)로 서서히 변화
    delay(10);
  }

  for(int i = 255; i >= 0; i--){
    analogWrite(LED_pin, i); // 255(켜짐)에서 0(꺼짐)으로 서서히 변화
    delay(10);
  }
}

```

5. 가변저항으로 LED 밝기 조절

스케치 3에서 가변저항을 이용하여 Blink 속도를 조절하는 스케치를 살펴보았다. 이를 수정하여 가변저항의 값으로 LED의 밝기를 제어하는 스케치를 작성해 보자. 가변저항으로 Blink 속도를 조절하는 그림 4와 동일하게 가변 저항과 LED를 연결한다. 스케치 6은 0에서 1023 사이의 가변저항 값을 읽어 이를 0에서 255 사이의 값으로 map 함수를 이용하여 변환한 후 변환된 값으로 LED의 밝기를 조절하는 예이다.

스케치 6. 가변저항으로 LED 밝기 조절

```

int led = 10;                // 10번 핀에 LED 연결
int variableR = A3;          // A3 핀에 가변저항 연결

long R;                      // 현재 가변저항의 값
long duty_cycle;             // LED 밝기

void setup() {
  pinMode(led, OUTPUT);      // 10번 핀을 출력 모드로 설정
  pinMode(variableR, INPUT); // A3번 핀을 입력 모드로 설정
}

void loop() {
  R = analogRead(variableR); // 가변저항 값 읽기
  // 0에서 1023사이의 가변저항 값을 0에서 255 사이의 duty cycle로 변경
  duty_cycle = map(R, 0, 1023, 0, 255);

  analogWrite(led, duty_cycle);
}

```

6. LED 모듈

그림 7은 여러 개의 LED 출력을 테스트하기 위한 LED 모듈의 회로도로서 8개의 LED가 포함되어 있다.

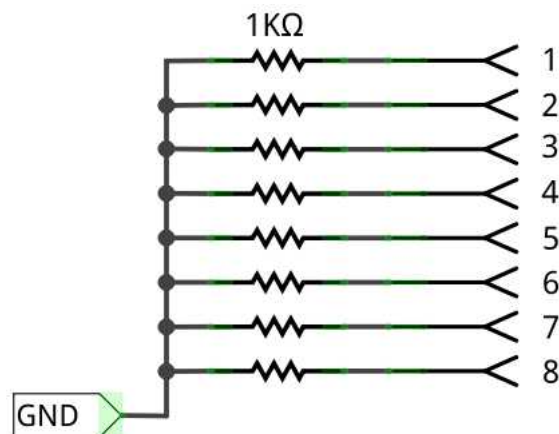


그림 7. LED 모듈 회로도

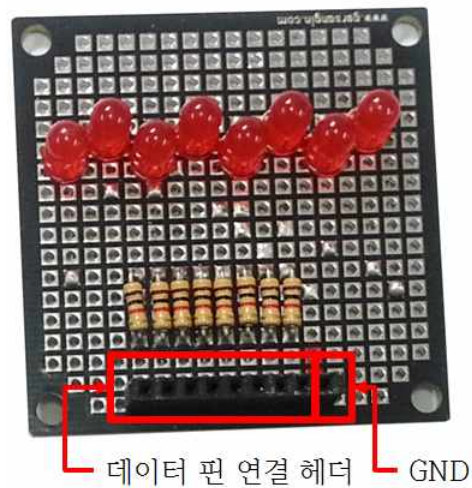


그림 8. LED 모듈

LED 모듈을 디지털 5번에서 12번 핀까지 연결해 보자. 디지털 4번 핀의 경우 스피커와 연결되어 있으므로 스케치를 실행하는 경우 잡음이 발생하므로 피한다.

LED 모듈과 DM-UStyle EX 보드 연결 사진

그림 9. LED 모듈 연결

스케치 7은 8개의 LED를 한 번에 하나씩 켜는 코드의 예이다.

스케치 7. LED 모듈 테스트

```

int pins[] = {5, 6, 7, 8, 9, 10, 11, 12};    // LED 모듈 연결 핀

void setup(){
    for(int i = 0; i < 8; i++)                // LED 모듈 연결 핀을 출력으로 설정
        pinMode(pins[i], OUTPUT);
}

void loop(){
    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 8; j++){
            if(i == j)
                digitalWrite(pins[j], HIGH);    // 한 번에 하나의 LED만 켜
            else
                digitalWrite(pins[j], LOW);       // 나머지 LED는 끄
        }
        delay(100);
    }
}

```

LED 모듈과 가변저항을 연결하고 가변저항의 값에 따라 켜지는 LED의 개수를 조절하는 스케치를 작성해 보자. LED 개수 조절을 위한 스케치는 기본적으로 스케치 3 및 스케치 6과 유사하다. 다만 LED의 개수는 최대 8개이므로 map 함수에서의 범위 조절에 유의하여야 한다. 스케치 8은 가변저항으로 LED 모듈에 켜지는 LED 개수를 조절하는 스케치의 예이다.

스케치 8. 가변저항으로 LED 모듈 제어

```

int pins[] = {5, 6, 7, 8, 9, 10, 11, 12};    // LED 모듈 연결 핀
int analog_pin = A3;                        // 가변저항 연결 핀

void setup(){
    for(int i = 0; i < 8; i++)                // LED 모듈 연결 핀을 출력으로 설정
        pinMode(pins[i], OUTPUT);
}

```

```
void loop(){
  int value = analogRead(analog_pin);           // 가변저항 값 읽기
  int LED_count = map(value, 0, 1023, 0, 8);     // LED 개수로 변환

  for(int i = 0; i < 8; i++){
    if(i < LED_count)                           // 범위 내의 값은 켜고
      digitalWrite(pins[i], HIGH);
    else                                         // 범위 밖의 값은 끈다.
      digitalWrite(pins[i], LOW);
  }
}
```