마이크로컨트롤러 응용

2015년 2학기

동양미래대학교
로봇자동화공학부

실습 부품을 사용할 때의 주의점

❖ 실습 부품은 원래 사용했던 그대로 반납합니다.

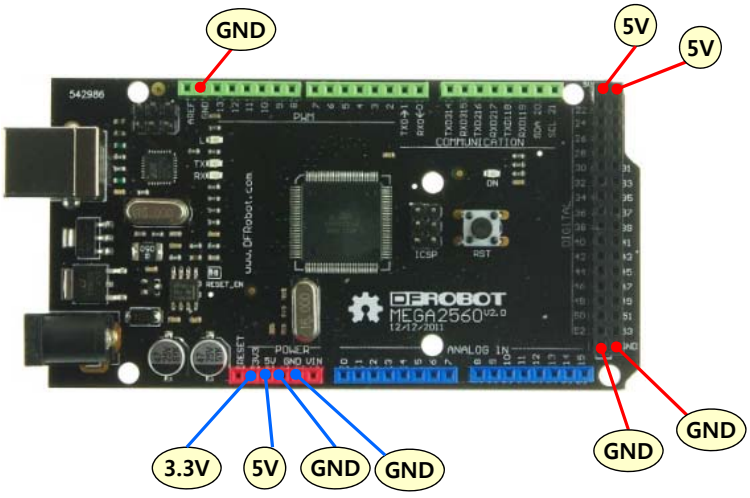
- 자기에게 할당된 번호의 실습 박스만 이용합니다. (출석부 번호, 모르면 교수님께 문의)
  - 실습 박스에 번호가 부여되어 있으므로 본인이 제대로 정리하지 않은 경우 추적이 가능합니다.
  - 실습 박스를 가져오자마자 분실된 부품이 있으면 교수님께 바로 보고 바랍니다.
    - 직전 강좌에서 이 실습 박스를 사용한 학생의 실습 점수에 반영할 수 있습니다.
- 박스에 들어 있던 부품은 수업 후 박스에 정상적인 상태로 넣어 반납합니다.
  - 사용한 케이블은 잘 정리해서 넣어 둡니다.
  - 비닐 팩에 들어 있던 부품은 비닐 팩에 다시 잘 넣어 둡니다.
- 프로젝트 실습실에 수업 시간에 실습 박스를 5개 정도 비치할 계획입니다.
  - 수업 외에는 프로젝트 실습실의 실습 박스를 이용하세요!
  - 실습 박스는 절대 실습실 밖으로 가져나가지 않습니다.
- 이를 지키지 않으면 실습 점수 0점을 부여하겠습니다.

1학기 복습

- ❖ Arduino로 디지털 출력을 할 수 있다.
- ❖ Arduino로 디지털 입력을 할 수 있다.
- ❖ Arduino로 아날로그 입력을 할 수 있다.
- ❖ Arduino로 아날로그 출력을 할 수 있다.

실습 시 Arduino Mega 2560 전원 연결 주의 사항

- ❖ 5V 출력: 세 군데
  - ■ 5V 출력이 네 개 이상 필요하면 브레드 보드로 확장해서 사용할 것!
- ❖ GND: 다섯 군데



Arduino Mega 2560 전원 공급

❖ 방법1: PWRIN(DC Jack)을 통한 전원 공급 (단독으로 동작시킬 때 권장)

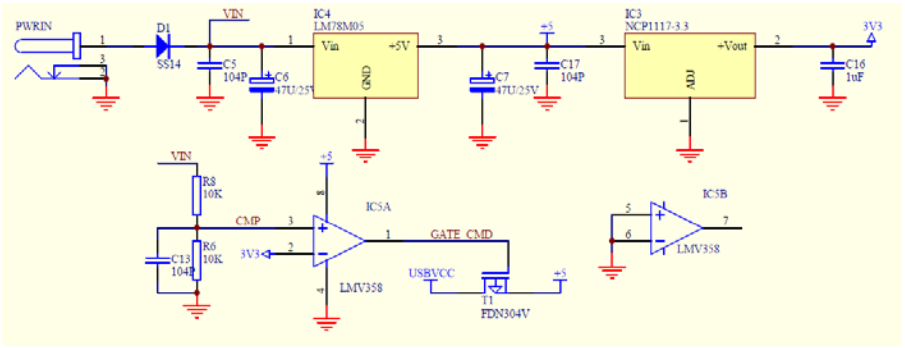
- 약 7.3V 이상의 전원 공급 필요 → LM78M05로 5V를 만들어 사용
  - ◆ D1 SS14의 전압강하 약 0.3V, IC4 LM78M05 Voltage Regulator의 전압강하 2V
  - ◆ 건전지를 사용한다면: 1.2V 충전지 6개로는 7.3V가 안 될 수 있음. 일반 건전지 1.5V 6개 정도 사용 권장
- 극성 뒤바뀜 보호 회로가 있음 (Schottky Diode SS14의 역할)
- 방법 1을 사용할 경우에 방법2도 사용하면 USB의 5V는 회로에 연결 안 됨 (아래 회로 참조)

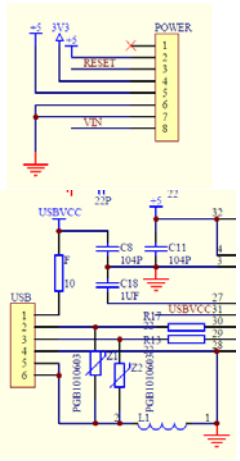
❖ 방법2: USB를 통한 전원 공급 (실습은 모두 이 방법으로 전원 공급)

- USB 5V 전원 공급 (IC5A와 T1에 의해 USB 전원 USBVCC가 +5에 연결됨)

❖ 방법3: VIN을 통한 전원 공급 (다른 보드로부터 전원 공급 받을 경우에 권장)

- 약 7.0V 이상의 전원 공급 필요
- 극성 뒤바뀜 보호회로가 없으므로 극성 주의

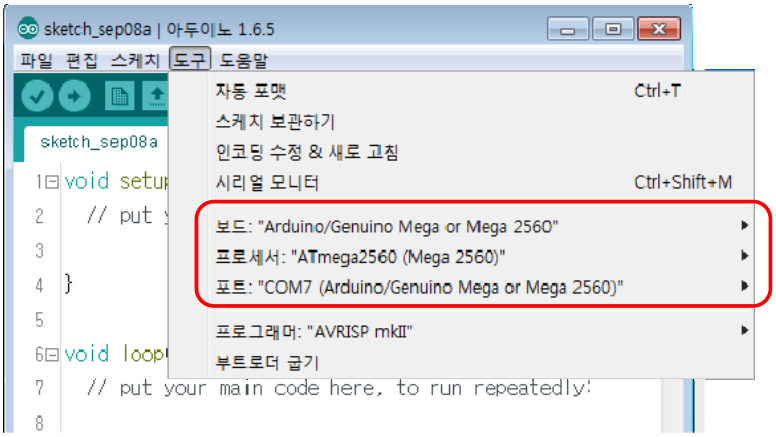




실습 시 Arduino 설정 주의 사항

❖ 우리가 사용하는 보드는 Arduino Mega 2560

- 보드 설정
- 프로세서 설정
- 포트 설정: 포트 번호는 보드에 따라 다를 수 있음

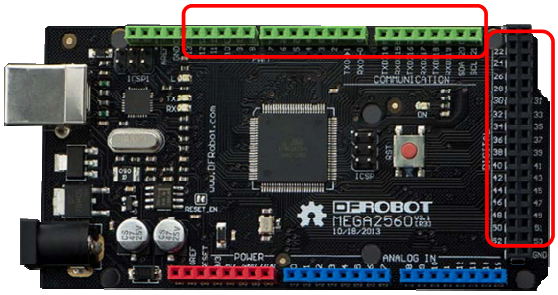


마이크로컨트롤러기초 복습 – 디지털 입출력 (1)

❖ Arduino 보드의 커넥터 중 숫자로 쓰여져 있는 핀은 모두 디지털 입출력이 가능하나... Arduino 기본 함수로 사용 가능한 부분은 아래 표시된 부분

❖ 어떻게 사용이 가능한가? 세 가지 모드 중 하나로 사용 가능

1. 디지털 출력 (디지털 전압 출력에 사용. High 전압 또는 Low 전압 출력)
   - ◆ AVR에 사용하는 전압이 5V이기 때문에 High 전압은 거의 5V, Low 전압은 거의 0V
   - ◆ 한 포트 당 최대 40mA까지 내보내거나 받을 수 있으나 전체 전류의 합 제한이 있으므로 수 mA 이하 권장
2. 내부 풀업 저항이 연결되어 있지 않은 디지털 입력 (디지털 전압 입력에 사용)
   - ◆ 전원 투입 직후 또는 Reset 후에는 이 상태임
3. 내부 풀업 저항이 연결되어 있는 디지털 입력 (디지털 전압 입력에 사용)



❖ 23, 25, 27, 29는 PCB 상에 표시만 안 되어 있을 뿐, 사용 가능합니다!

마이크로컨트롤러기초 복습 – 디지털 입출력 (2)

❖ 디지털 입출력 관련 함수

■ pinMode(pin, mode) 함수로 디지털 포트를 어떻게 사용할 것인지를 설정
- ◆ pin : 보드에 적혀 있는 핀 번호, Arduino Mega 2560은 1번부터 53번까지 존재
  - ■ 13번 핀은 보드 내부 LED에, 0번 핀과 1번 핀은 시리얼 통신에 사용하므로 사용하지 않을 것을 권장
- ◆ mode : OUTPUT(디지털 출력), INPUT(내부 저항이 없는 입력), INPUT_PULLUP(내부 저항이 있는 입력) 중 하나 (앞 페이지 설명 참조)
- ◆ 리턴 값 : 없음

■ digitalWrite(pin, value) 함수로 디지털 출력 설정
- ◆ pinMode 함수에서 mode가 OUTPUT 인 경우에 사용
- ◆ pin : 보드에 적혀 있는 핀 번호
- ◆ value : HIGH와 LOW 중 선택
  - ■ Arduino Mega 2560에서는 HIGH면 약 5V, LOW면 약 0V 출력
- ◆ 리턴 값 : 없음
- ◆ 참고: 특정 포트에 대해 이 함수를 한 번도 실행하지 않았다면 그 포트는 LOW가 출력되고 있음.

■ digitalRead(pin) 함수로 디지털 상태 입력
- ◆ pinMode 함수에서 mode가 INTPUT이나 INTPUT_PULLUP인 경우에 사용
- ◆ pin: 보드에 적혀 있는 핀 번호
- ◆ 리턴 값 : int 형으로 HIGH 아니면 LOW
  - ■ 리턴 값이 HIGH면 해당 핀이 5V 근방 전압, LOW면 0V 근방 전압이 연결되어 있는 것임
    - ▪ Arduino Mega 2560에서는 3V 이상이면 HIGH로 인식하고 1V 이하면 LOW로 인식

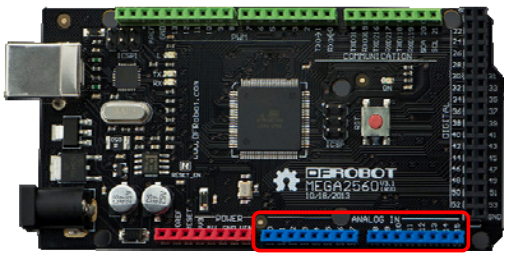
마이크로컨트롤러기초 복습 – 아날로그 입력 (ADC)

❖ 특징

- 분해능: 10 bit ADC → 디지털 값으로 0~1023 으로 변환
- 변환범위: 0V에서 analogReference() 함수로 정한 전압 범위까지
- MUX: Arduino Mega 2560의 경우 16채널 (0~15)

❖ analogReference(type) 함수

- type: DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, EXTERNAL 중 하나로 변환 범위 지정
  - ◆ DEFAULT: Arduino Mega 2560은 5V ← 기본 설정값
  - ◆ INTERNAL: Arduino Mega 2560은 사용 금지
  - ◆ INTERNAL1V1: Arduino Mega 2560은 1.1V
  - ◆ INTERNAL2V56: Arduino Mega 2560은 2.56V
  - ◆ EXTERNAL: AREF에 연결한 전압
- 리턴 값 없음



❖ analogRead(pin)

- pin: MUX 채널 지정, Arduino Mega 2560에서는 0~15 중 하나
- 리턴 값: 변환 값으로 0~1023 값 중 하나
  - ◆ 참고: analogReference(DEFAULT)인 경우, 0~5V가 0~1023으로 변환된다

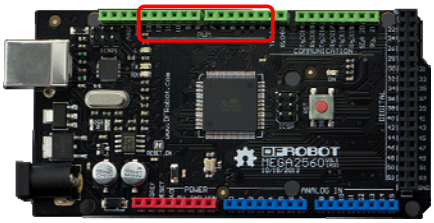
마이크로컨트롤러기초 복습 – 아날로그 출력 (PWM)

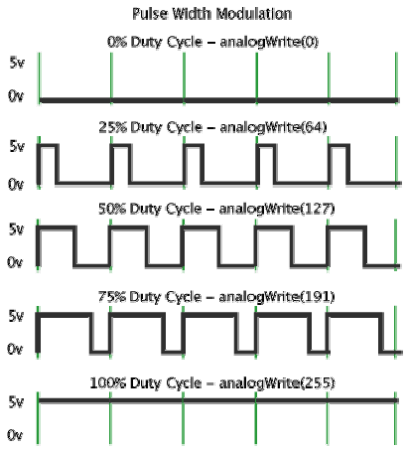
❖ analogWrite(pin, value)

- pin: Arduino 보드의 PWM으로 표시된 핀 번호
  - ◆ Arduino Mega 2560의 경우에는 2~13
  - ◆ Arduino Mega 2560에 LCD Keypad Shield가 바로 장착된 경우에는 2, 3, 11, 12, 13번 사용 가능
- value: duty cycle로 0~255 범위
  - ◆ 0은 0% duty cycle, 255는 100% duty cycle에 해당
  - ◆ 참고: 이 값의 범위는 analogWriteResolution() 함수로 변경 가능 → 2학기 때 다룰 예정
- 참고1: PWM 주기는 약 490Hz (5, 6번만 약 980Hz)
- 참고2: 5번과 6번은 시간 관련 함수와 연동되어 있어 사용을 권장하지 않음

❖ PWM(Pulse Width Modulation)

- 일정 주기 속에서 상승 펄스 폭을 조절
- 디지털로 아날로그 효과
  - ◆ 우측 그림에서의 PWM 평균 전압 = DutyCycle*5V
  - ◆ Arduino에서의 PWM 평균 전압 = value/255*5V
  - ◆ 모터의 속도 가변에 사용하면 효과적



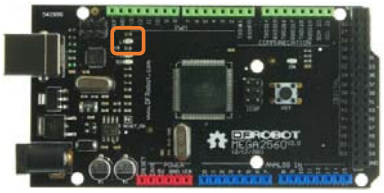


http://www.arduino.cc/en/Tutorial/PWM에서 발췌

마이크로컨트롤러기초 복습 – LED 점멸하기

❖ Arduino Mega 2560에 내장된 LED를 0.2초 주기로 점멸하기

- 이 LED는 13번 핀에 연결되어 있음



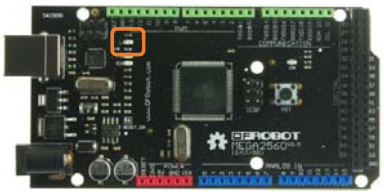
P01.Review01.Led

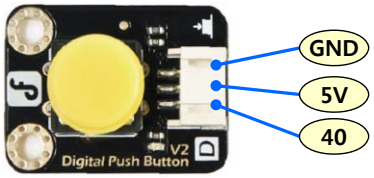
```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(100);
  digitalWrite(13, LOW);
  delay(100);
}
```


마이크로컨트롤러기초 복습 – 스위치 사용하기

❖ 스위치를 누르고 있을 때만 LED 켜기

- LED는 13번 핀에 연결된 내장 LED 사용
  - ◆ High 출력일 때 ON됨
- 스위치는 40번에 연결해서 사용
  - ◆ 누르면 High 입력, 안 누르면 Low 입력 → 내부 풀업 저항 불필요





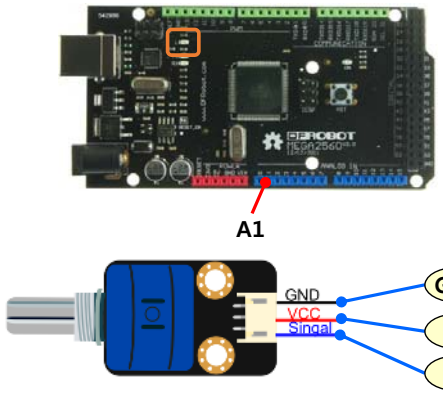
P01.Review02.Switch

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(40, INPUT);
}

void loop() {
  if (digitalRead(40) == HIGH)
    digitalWrite(13, HIGH);
  else
    digitalWrite(13, LOW);
}
```

마이크로컨트롤러기초 복습 – 아날로그 입출력

❖ Rotation Sensor (가변 저항)으로 LED 밝기 조절하기

- LED는 13번 핀에 연결된 내장 LED 사용
  - ◆ PWM Duty Cycle는 0~100%는 0~255값에 해당
- Rotation Sensor는 A1에 연결해 사용
  - ◆ Analog Input에서 Reference Voltage를 DEFAULT로 하면 0~5V가 0~1023으로 읽힘
- ➔ Rotation Sensor 측정값을 4로 나누어 Analog 출력값으로 사용!
- Rotation Sensor의 측정값을 전압 단위로 Serial로 출력
  - ◆ 측정값이 val이라면 5.0*double(val)/1023.0을 곱하면 됨!



P01.Review03.AnalogIO

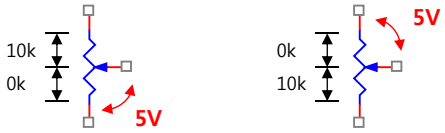
```
void setup() {
  analogReference(DEFAULT); // 생략 가능
  Serial.begin(9600);
}

void loop() {
  int val;

  val = analogRead(1);  // A1
  analogWrite(13, val/4); // 13번 LED

  Serial.println(5.0*double(val)/1023.0);
}
```


참고: Rotation Sensor는 왜 고장날 수 있을까?

❖ Rotation Sensor는 10kΩ 가변저항과 동일

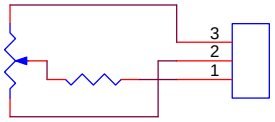
❖ 만약 다이얼을 끝까지 돌린 상태에서 선 연결을 잘못 한다면...

- 가변 저항 전체의 저항값은 10kΩ이지만 중간 단자와 다른 한쪽 끝까지의 저항값이 0kΩ이 되고
- 선 연결을 잘못해서 이 사이에 5V와 GND를 연결하면
- 저항이 작은 상태로 급격히 큰 전류가 흐르고 이에 따라 열이 발생해서 저항이 끊어진다!



❖ 어떻게 하면 해결할 수 있을까?

- 수 백 옴 정도의 저항을 직렬로 추가하면
- 배선을 잘못하더라도 이 저항에 의해 전류가 제한되므로 위와 같은 문제가 발생되지는 않는다.



마이크로컨트롤러기초 복습 – Lamp (1)

❖ P01.Review04.Lamp 문제

- 사용 재료
  - ◆ Switch: 40번 핀에 연결
  - ◆ LED: 13번 핀에 연결된 내장 LED
- 동작
  - ◆ Switch를 누를 때마다 LED의 밝기를 약간 밝음, 중간 밝음, 아주 밝음, 꺼짐을 반복하도록 하시오.
    - 약간 밝음은 10, 중간 밝음은 30, 아주 밝음은 90, 꺼짐은 0 사용
    - 최초에는 꺼진 상태
  - ◆ LED 상태에 변화가 생길 때마다 LCD와 시리얼 모니터에 해당 내용을 표시하시오.
    - 약간 밝음은 “ON 1”, 중간 밝음은 “ON 2”, 아주 밝음은 “ON 3”, 꺼짐은 “OFF!”

마이크로컨트롤러기초 복습 – Lamp (2)

❖ 소스 코드

P01.Review04.Lamp

```
int ledPin = 13;
int swPin = 40;

void setup() {
  pinMode(swPin, INPUT);
  analogReference(DEFAULT);
  Serial.begin(9600);
  analogWrite(ledPin, 0);
}
```

```
void loop() {
  static int swPrev = false, lampStep = 0;
  int swCurr;

  if (digitalRead(swPin)==HIGH)
    swCurr = true;
  else
    swCurr = false;

  if (swPrev == false && swCurr == true) {

    if (++lampStep>=4)  // 0, 1, 2, 3을 반복
      lampStep = 0;

    switch (lampStep)
    {
      case 0:
        analogWrite(ledPin, 0);
        Serial.println("OFF!");
        break;

      case 1:
        analogWrite(ledPin, 10);
        Serial.println("ON 1");
        break;
```

```
      case 2:
        analogWrite(ledPin, 30);
        Serial.println("ON 2");
        break;

      case 3:
        analogWrite(ledPin, 90);
        Serial.println("ON 3");
        break;
    }
  }
  swPrev = swCurr;

  delay(10);
}
```

마이크로컨트롤러기초 복습 – 복습하기

❖ 여기까지 마이크로컨트롤러 기초의 기본 내용을 복습했습니다.

- 마이크로컨트롤러기초 내용 중 회로 부분은 복습하지 않았습니다.
  - ◆ LED 구동 회로, 스위치 회로, 릴레이 회로 등은 반드시 익혀 두세요!
- 마이크로컨트롤러기초 내용 중 LCD 부분은 복습하지 않았습니다.
  - ◆ 사용 방법이 어렵지 않으니 관심 있는 학생만 복습하세요!

❖ 마이크로컨트롤러기초에서 다룬 예제 및 연습문제가 총 69개입니다.

- 남는 시간에 미진했던 부분에 대해 복습을 진행하기 바랍니다.

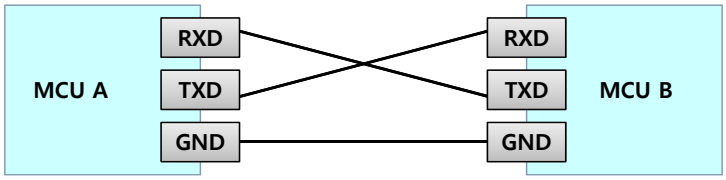
시리얼 통신

❖ Arduino를 이용해서 시리얼 통신을 할 수 있다.

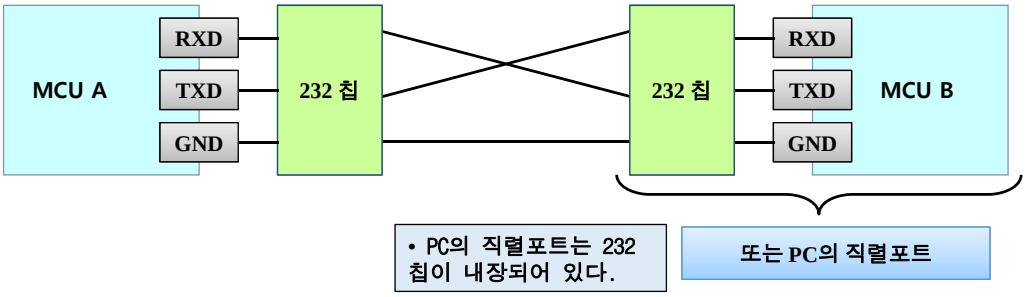
- ◆ 배선을 할 수 있다.
- ◆ 간단한 통신 프로그램을 작성하고 사용할 수 있다.
- ◆ Serial0에 해당하는 Serial 클래스뿐 아니라 Serial1 ~ Serial3 클래스도 사용할 수 있다.

UART 통신(=시리얼 통신) 배선 방법

❖ 통신 거리가 짧을 경우의 배선 방법



❖ 통신 거리가 긴 경우의 배선 방법



❖ UART↔ZigBee, UART↔Bluetooth, UART↔Ethernet 다 존재

- UART를 경유해서 ZigBee, Bluetooth(창의과제기초에서 사용한 방법), Ethernet 통신 사용 가능

시리얼 통신 (UART 통신) – 개요

❖ Arduino에서 UART 통신은 Serial 클래스로 제공

- 클래스에 포함된 함수(멤버 함수): 자세한 내용은 Reference 참조
  - if(Serial) – 시리얼 포트가 준비되었는지 판단
  - available() – 시리얼 포트로 전송된 데이터(바이트 단위)의 수를 리턴
  - begin() – 시리얼 포트 초기화 (Baudrate, 데이터 길이, 패리티를 초기화)
  - end() – 시리얼 포트를 더 이상 사용 안 할 때 사용
  - find() – 시리얼 버퍼에 원하는 문자가 있는지 판단
  - findUntil() – 시리얼 버퍼에 원하는 문자나 Terminator 문자가 있는지 판단
  - flush() – 시리얼 버퍼를 비움
  - parseFloat() – 시리얼 버퍼로부터 실수형 데이터를 얻어냄
  - parseInt() – 시리얼 버퍼로부터 정수형 데이터를 얻어냄
  - peek() – 시리얼 버퍼로부터 한 문자를 가져오지만 해당 문자를 버퍼에서 비우지는 않음
  - print() – 출력
  - println() – 출력하고 다음 줄로
  - read() – 시리얼 버퍼로부터 한 문자 읽음
  - readBytes() – 시리얼 버퍼로부터 일정 길이의 데이터를 읽음
  - readBytesUntil() – 시리얼 버퍼로부터 일정 길이의 데이터 또는 Terminator 문자까지 읽음
  - write() – 시리얼로 데이터를 전송함 (바이트 데이터, 문자열, 바이트 배열 가능)
  - serialEvent() – 시리얼 수신 인터럽트 서비스 루틴에 해당
- 이 중 마이크로컨트롤러기초에서 begin(), print(), println()의 의미를 알고 사용했고
- available(), read(), parseFloat(), parseInt()를 의미는 정확히 모르지만 예제에 사용했었음
- 멤버 함수의 사용법을 다 알면 좋겠지만 교과목 수준을 벗어나므로 이 중에서
  - available(), read(), write() 함수의 사용법을 소개하도록 하겠습니다.

Arduino의 시리얼 통신 구조

❖ 수신

- 다른 장치가 시리얼 통신으로 Arduino로 한 바이트를 전송하면
  - ◆ Arduino의 Serial 클래스 내부에 있는 인터럽트 서비스 루틴이 이 데이터를 받아 수신 버퍼에 저장해 둠
- 사용자는 버퍼에 전송된 데이터가 있는지를 available() 함수로 판단하고 다양한 함수를 이용해 이 데이터를 읽어감
  - ◆ 참고: available() 함수는 버퍼에 저장된 수신 데이터 길이를 리턴함

❖ 송신

- print(), println(), write() 함수를 이용하면 송신 버퍼에 송신할 데이터를 저장함
- Serial 클래스 내부의 송신 인터럽트 서비스 루틴이 송신 버퍼를 참조하여 알아서 데이터 송신

❖ 조언

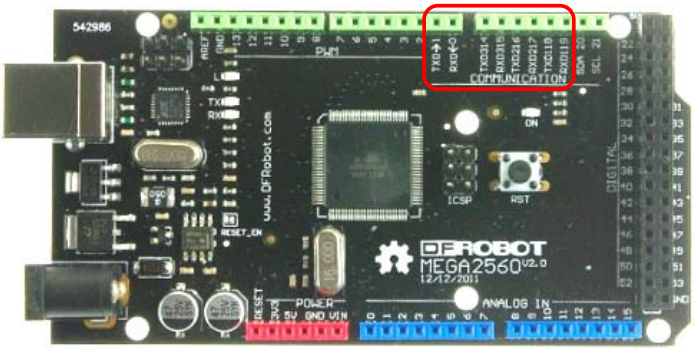
- 시리얼 통신 과정이 내부적으로는 다소 복잡할 수 있으나 이 과정은 모두 Serial 클래스 내부에서 일어나므로 시리얼 통신을 간단하게 사용할 경우에는 내부 내용은 모른 채 위 함수들의 사용방법만 정확히 알고 사용하면 됨!

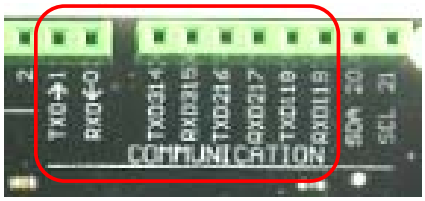
Arduino Mega 2560의 시리얼 통신

❖ 총 네 개의 시리얼 통신 채널 사용 가능

- Serial 클래스는 0번 핀(TX0)과 1번 핀(RX0)을 사용
  - ◆ 이 핀은 USB와도 연결되어 있으므로 USB에 의한 시리얼 통신을 할 경우에는 사용 못 함!
- Serial1 클래스는 19번 핀(TXD1)과 18번 핀(RXD1)을 사용
- Serial2 클래스는 17번 핀(TXD2)과 16번 핀(RXD2)을 사용
- Serial3 클래스는 15번 핀(TXD3)과 14번 핀(RXD3)을 사용

※ TX는 데이터를 보내는 핀, RX는 데이터를 받는 핀





시리얼 통신 함수 익히기

❖ available()

- 파라미터: 없음
- 리턴 값: 시리얼 수신 버퍼에 저장된 데이터의 개수

❖ read()

- 파라미터: 없음
- 리턴 값: int형으로 수신 버퍼에서 가장 먼저 전송된 데이터
  - ◆ 수신 버퍼가 비어 있으면 -1 리턴
- 주의: 이 함수를 사용해서 수신 버퍼의 데이터를 가져오면 해당 데이터는 수신 버퍼에서 제거된다!

❖ write(val), write(str), write(buf, len)

- 파라미터 val: val을 전송 (바이트)
- 파라미터 str: str 문자열을 전송
- 파라미터 buf와 len: 바이트 배열 buf에서 len 길이만큼 전송
- 리턴 값: 전송한 데이터 길이

시리얼 통신 예제 – LRUD (1)

❖ 문제: 컴퓨터에서 시리얼 통신으로 Arduino에

- 'l' 또는 'L'을 전송하면 Arduino는 컴퓨터로 "left" 전송
- 'r' 또는 'R'을 전송하면 Arduino는 컴퓨터로 "right" 전송
- 'u' 또는 'U'를 전송하면 Arduino는 컴퓨터로 "up" 전송
- 'd' 또는 'D'를 전송하면 Arduino는 컴퓨터로 "down" 전송

❖ 어떻게?

- Arduino에서는 available() 함수로 수신된 데이터가 있으면 read() 함수로 읽어 이 문자가 'l', 'r', 'u', 'd' 문자에 해당하면 println() 함수로 해당 문자열을 전송

❖ 참고

- Arduino를 이용해 작품 활동을 할 경우에 시리얼 통신으로 여러 가지 다양한 테스트를 할 경우가 많습니다.
- 여기 소개한 예제를 변형해서 테스트에 활용하면 큰 도움이 될 것입니다.
- 예를 들어
  - ◆ 'T'를 보내면 Tracking Sensor의 값을 출력
  - ◆ 'C'를 보내면 Color Sensor의 값을 출력
  - ◆ 'D'를 보내면 Distance Sensor의 값을 출력
  - ◆ 'M'을 보내면 Motor 동작

시리얼 통신 예제 – LRUD (2)

❖ 소스 코드

P02.Serial01.LRUD

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

```
void loop() {
  if (Serial.available()) { // 수신된 데이터가 있으면
    switch (Serial.read()) { // 수신된 데이터를 읽어
      case 'l':
      case 'L':
        Serial.println("left");
        break;
      case 'r':
      case 'R':
        Serial.println("Right");
        break;
      case 'u':
      case 'U':
        Serial.println("Up");
        break;
      case 'd':
      case 'D':
        Serial.println("Down");
        break;
    }
  }
}
```

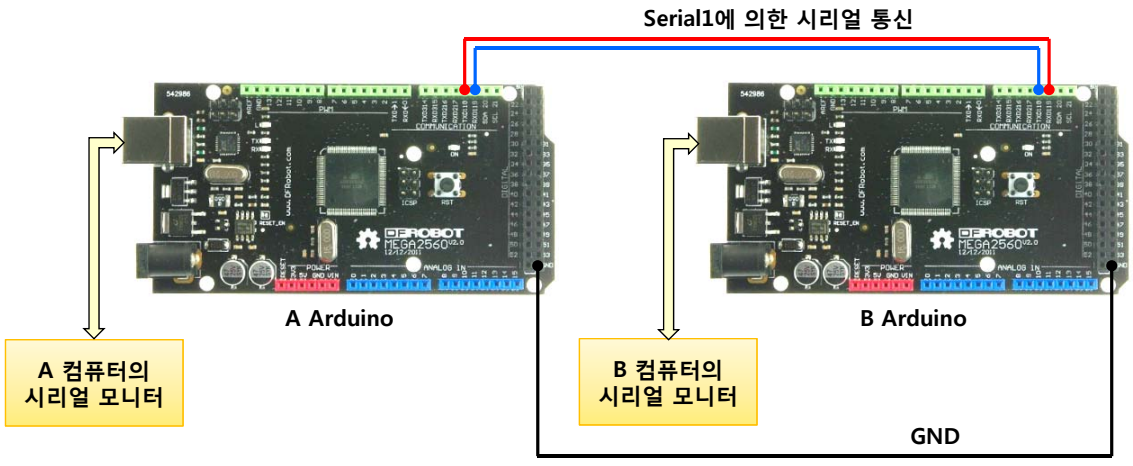

시리얼 통신 예제 – Chatting (1)

❖ 문제

- 두 사람이 두 개의 Arduino 보드를 Serial1로 연결해서 두 PC에서 시리얼 모니터로 채팅하기
- Serial 클래스는 9600 bps를 Serial1 클래스는 19200 bps를 사용할 것!

❖ 배선

- A Arduino의 18번 핀(TXD1)을 B Arduino의 19번 핀(RXD1)에 연결
- A Arduino의 19번 핀(RXD1)을 B Arduino의 18번 핀(TXD1)에 연결
- A Arduino의 GND와 B Arduino의 GND를 연결



시리얼 통신 예제 – Chatting (2)

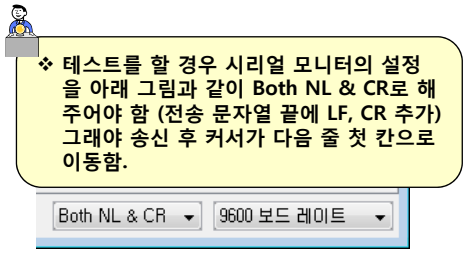
❖ 어떻게?

- Serial 클래스로 수신한 데이터가 있으면 이 데이터를 Serial1 클래스로 송신
  - ◆ 시리얼 모니터에서 데이터를 송신하면 Arduino의 Serial 클래스가 데이터를 수신함
  - ◆ 이 데이터를 Serial1 클래스로 송신하면 상대방 Arduino의 Serial1 클래스가 데이터를 수신함
- Serial1 클래스로 수신한 데이터가 있으면 이 데이터를 Serial 클래스로 송신
  - ◆ 위와 마찬가지

❖ 소스 코드

P02.Serial02.Chatting

```
void setup() {
  Serial.begin(9600);   // Serial 초기화
  Serial1.begin(19200); // Serial1 초기화
}

void loop() {
  // Serial로 수신한 데이터를 Serial1으로 송신
  if (Serial.available()) {
    Serial1.write(Serial.read());
  }

  // Serial1으로 수신한 데이터를 Serial로 송신
  if (Serial1.available()) {
    Serial.write(Serial1.read());
  }
}
```



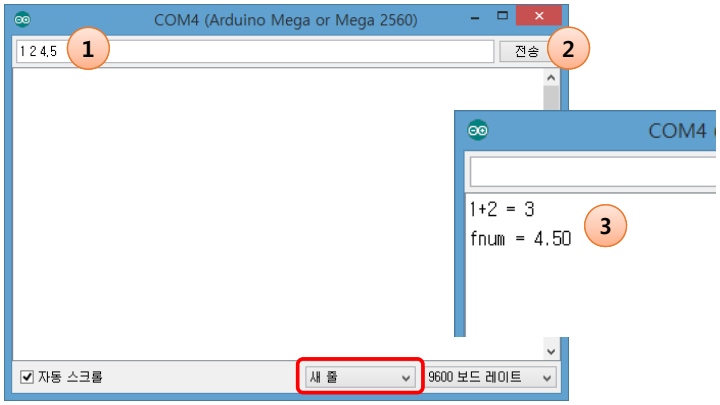
시리얼 통신 참고 – 기초 때 소개했던 방법 분석

❖ 분석

- 시리얼 모니터에서 ①과 같이 입력하고 ② 전송 버튼을 누르면 PC는 "1 2 4.5₩n" 문자열을 Arduino로 송신함
- Arduino에서는 시리얼 수신 버퍼에 수신된 데이터가 있으면
  - ◆ parseInt()으로 정수형 데이터가 올 때까지 기다렸다가 그 값을 리턴
  - ◆ parseInt(), parseFloat()로 이후 데이터 수신
  - ◆ 이후 바로 '₩n'(시리얼 모니터에서 '새 줄'에 의해 추가되는 문자)이면
    - print(), println()으로 결과를 출력해서
    - 시리얼 모니터에는 ③과 같은 결과가 출력된다

※ 반드시 '새 줄'을 선택해야 함!

※ 마지막 숫자 후 바로 Enter 키를 눌러야 함 (공백 있으면 안 됨)




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

void loop() {
  int inum0, inum1;
  float fnum;

  if (Serial.available() > 0) {
    inum0 = Serial.parseInt();
    inum1 = Serial.parseInt();
    fnum = Serial.parseFloat();

    if (Serial.read() == '₩n') {
      Serial.print(inum0);
      Serial.print('+');
      Serial.print(inum1);
      Serial.print(" = ");
      Serial.println(inum0+inum1);
      Serial.print("fnum = ");
      Serial.println(fnum);
    }
  }
}
```

시리얼 통신 – 연습 문제

❖ P02.Serial03.Exercise01

- 시리얼 모니터로 'a'를 송신하면 13번 핀에 연결된 내부 LED를 ON
- 시리얼 모니터로 'z'를 송신하면 13번 핀에 연결된 내부 LED를 OFF
- 시리얼 모니터로 's'를 송신하면 40번 핀에 연결된 스위치 상태를 시리얼로 출력

❖ P02.Serial04.Exercise02

- 시리얼 모니터로 문자열을 송신하면
  문자열 중 대문자는 소문자로 바꿔서 다시 시리얼 모니터로 전송

Limit Switch

❖ Limit Switch를 사용할 수 있다.

1. Limit Switch의 구조를 설명할 수 있다. (COM, NO, NC)
2. A접점 또는 B접점을 이용해서 Arduino와 연결하는 회로를 구성할 수 있다.
3. 풀업 저항의 역할을 설명할 수 있다.
4. Arduino의 내부 풀업 저항을 Limit Switch에 연결해서 사용할 수 있다.

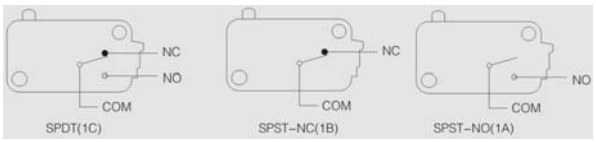
Limit Switch – 개요

❖ 리밋 스위치란?

- 자동화 장치에서 이송부의 감지에 사용하는 스위치
  - ◆ 예) 이송부의 원점 또는 이동 한계점 검출에 사용
- 디바이스마트에서는 마이크로스위치로 분류
  - ◆ http://www.devicemart.co.kr/goods/list.php?category=003005007
- 스위치가 눌리는 부분이 탄성체로 되어 있음 (이송부에 의해 눌리므로...)
  - ◆ 롤러가 달려 있는 경우도 있음

❖ 리밋 스위치의 사용 방법

- 일반 스위치와 사용법 동일!
  - ◆ COM, NO(A접점), NC(B접점)
- 스위치에는 풀업 저항 사용할 것!
  - ◆ 내부 풀업 저항 또는 외부 풀업 저항 사용!



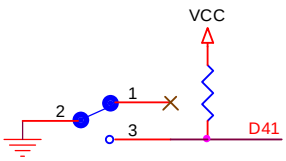
XURUI Electronic의 XV-15Micro switch2에서 발췌

XURUI Electronic의 XV-15Micro switch2에서 발췌

Limit Switch – 사용 방법 정리

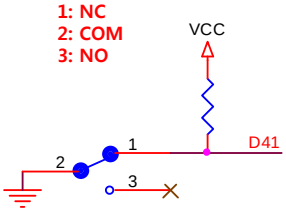
❖ 리밋 스위치의 상태를 41번 핀으로 입력 받는 경우를 예로 설명

- 외부 풀업 저항을 연결하고 사용하는 경우 A접점을 이용하는 경우
  - ◆ pinMode(41, INTPUT) 으로 설정하고
  - ◆ digitalRead(41)의 리턴 값이 HIGH면 OFF 상태, LOW면 ON 상태

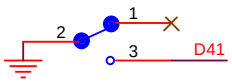


1: NC
2: COM
3: NO

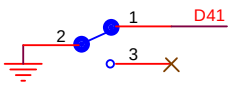
- 외부 풀업 저항을 연결하고 사용하는 경우 B접점을 이용하는 경우
  - ◆ pinMode(41, INTPUT) 으로 설정하고
  - ◆ digitalRead(41)의 리턴 값이 HIGH면 ON 상태, LOW면 OFF 상태



- 내부 풀업 저항을 연결하고 사용하는 경우 A접점을 이용하는 경우
  - ◆ pinMode(41, INTPUT_PULLUP) 으로 설정하고
  - ◆ digitalRead(41)의 리턴 값이 HIGH면 OFF 상태, LOW면 ON 상태



- 내부 풀업 저항을 연결하고 사용하는 경우 B접점을 이용하는 경우
  - ◆ pinMode(41, INTPUT_PULLUP) 으로 설정하고
  - ◆ digitalRead(41)의 리턴 값이 HIGH면 ON 상태, LOW면 OFF 상태



적외선 거리 측정 센서

❖ 적외선 거리 측정 센서를 사용할 수 있다.

- ❖ 적외선 거리 측정 센서를 Arduino에 연결할 수 있다.
- ❖ analogRead() 함수를 사용해서 적외선 거리 측정 센서의 전압을 측정할 수 있다.
- ❖ 거리-전압 그래프를 이해하고 사용할 수 있다.
- ❖ 여러 개의 거리 측정 센서를 이용해서 AND 조건이나 OR 조건으로 사용할 수 있다.

Distance Sensor – SHARP GP2Y0A21YK0F (1)

❖ 특징

- 적외선을 이용한 거리 측정 센서
- 이 센서의 측정 가능 거리는 10~80cm
  - ◆ 참고: 측정 범위가 다른 모델도 존재
- 주의: 아두이노에 연결할 때 케이블 순서 주의!

■Electro-optical Characteristics (Ta=25℃,Vcc=5V)

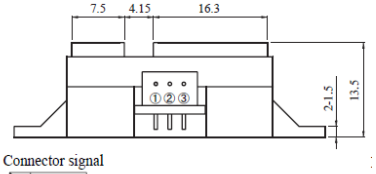
Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Average supply current	I_{CC}	L=80cm (Note 1)	—	30	40	mA
Distance measuring	ΔL	(Note 1)	10	—	80	cm
Output voltage	V_O	L=80cm (Note 1)	0.25	0.4	0.55	V
Output voltage differential	ΔV_O	Output voltage differece between L=10cm and L=80cm (Note 1)	1.65	1.9	2.15	V

* L : Distance to reflective object

Note 1 : Using reflective object : White paper (Made by Kodak Co., Ltd. gray cards R-27•white face, reflectance; 90%)

■Recommended operating conditions

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	4.5 to 5.5	V



Connector signal

	signal name
①	Vo
②	GND
③	Vcc

Connector :
J.S.T.TRADING COMPANY,LTD,
S3B-PH

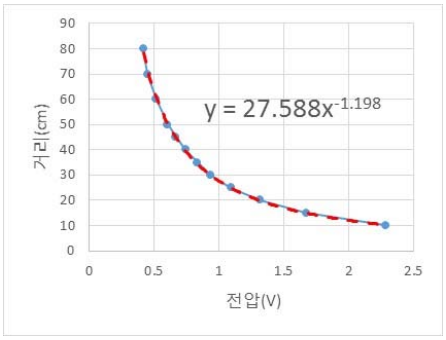
Distance Sensor – SHARP GP2Y0A21YK0F (2)

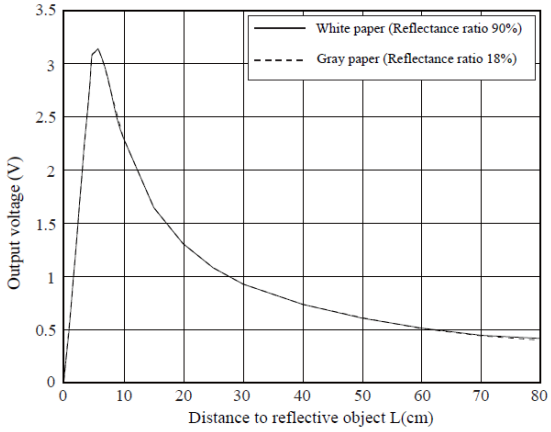
❖ 출력 특징

- 10 ~ 80cm 거리 측정에만 사용할 것
  - ◆ 거리가 멀수록 출력 전압이 낮아짐
  - ◆ 약 6cm 이하 구간에서는 거리가 멀수록 출력전압이 높아짐
- 참고: 이 센서의 전압으로 거리를 계산하려면
  - ◆ Excel 등에서 추세선 식을 구한 뒤 이 식으로부터 계산하는 방법도 있고
  - ◆ 여러 포인트에 대해 테이블을 만들어 보간법(interpolation)에 의해 계산하는 방법도 있음

❖ USB로 전원을 공급받을 때의 주의점

- 긴 케이블로 인해 전원이 불안정할 수 있음
- 이 경우에는 ADC의 reference voltage를 DEFAULT 대신 INTERNAL을 사용하는 것도 방법!
  - ◆ analogReference(INTERNAL2V56)
  - ◆ 이 경우는 0~2.56V가 0~1023으로 변환됨





Distance Sensor – 거리 측정

❖ 문제

- GP2Y0A21Y 센서 출력은 Arduino의 A2에 연결
- Arduino ADC의 Reference Voltage로 DEFAULT 사용
  - ◆ analogReference(DEFAULT);
- 측정 결과는 정확히 0.2초마다 LCD와 시리얼 모니터로 출력
- 거리 y와 측정 전압 x의 관계식은 앞 그래프의 y = 27.588*x^-1.198 식을 이용
  - ◆ math.h의 pow() 함수를 사용하면 됨, 예) distance = 27.588 * pow(voltage, -1.198);
  - ◆ Arduino에서는 #include <math.h>를 안 해도 됨

P03.DistSensor01.Measure

```
const int distSenPin = 2;

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

void measure(void) {
  int i;
  double voltage, distance;

  voltage = 5.0/1023.0*double(analogRead(distSenPin));
  distance = 27.588 * pow(voltage, -1.198);
  Serial.print(distance, 1);
  Serial.println(" cm");
}
```

```
void loop() {
  static unsigned long timePrev = 0;
  unsigned long time;

  // 정확히 0.2초마다 measure 함수 호출
  time = millis()/200;
  if (time != timePrev) {
    timePrev = time;
    measure();
  }
}
```

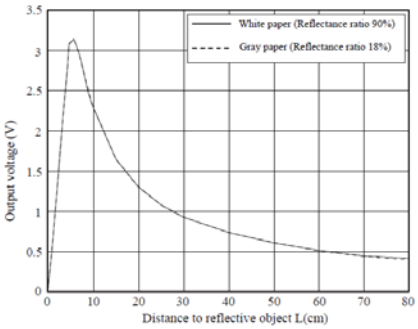
Distance Sensor – 거리 판단

❖ 문제

- A2에 연결된 GP2Y0A21Y 센서로 측정한 거리가 20cm 이내일 때만 13번 연결 LED를 켜시오.
- Analog Reference로는 DEFAULT(5V)를 이용하시오.

❖ 어떻게?

- 20cm일 때의 센서 출력 전압은 약 1.3V이다.
- 방법 1: 전압으로 비교
  - ◆ if (analogRead(2)*5.0/1023.0 >= 1.3) digitalWrite(13, HIGH);
- 방법2 : 디지털 변환 값으로 비교
  - ◆ if (analogRead(2) >= 266) digitalWrite(13, HIGH);
    - 1.3*1023.0/5.0 = 266



P03.DistSensor02.Threshold

```
void setup() {
  pinMode(13, OUTPUT);
  analogReference(DEFAULT);
  Serial.begin(9600);
}

void loop() {
  // if (analogRead(2)*5.0/1023.0 >= 1.3)
  if (analogRead(2) >= 266)
    digitalWrite(13, HIGH);
  else
    digitalWrite(13, LOW);
}
```


참고: ADC 여러 채널 사용 시 주의점

❖ 마이크로컨트롤로기초에서 Arduino의 analogRead() 함수의 문제점

- Arduino(AVR)에서 ADC의 MUX 채널을 변경한 직후의 값은 이전 채널의 영향을 받는다.
  - ◆ 예) val0 = analogRead(0); 후 val1 = analogRead(1); 을 하면, val0의 값이 val1에 영향을 끼친다.
    - val0이 val1 보다 작은 값이면 val1은 실제보다 작게 측정되고
    - val0이 val1 보다 큰 값이면 val1은 실제보다 크게 측정된다.
  - ◆ 몰라도 되는 이유: analogRead() 함수가 MUX의 Settling Time을 고려하지 않아서임.

❖ 해결책: ADC의 MUX 채널을 변경한 직후의 첫 번째 analogRead() 함수값은 사용하지 않고 버린다.

- 예) val0에 A0의 ADC 변환값을 val1에 A1의 ADC 변환값을 대입하고 싶으면

```
analogRead(0); // 채널 변경 후 첫 번째 읽은 값은 버림
val0 = analogRead(0);

analogRead(1); // 채널 변경 후 첫 번째 읽은 값은 버림
val1 = analogRead(1);
```

Distance Sensor – 팀 과제

❖ P03.DistSensor03.Exercise

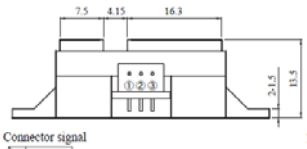
- 두 명 또는 세 명이 한 팀을 구성한다.
- 1개의 Arduino 보드, 2개의 거리 측정 센서 GP2Y0A21Y, 1개의 스위치 모듈, 1개의 LED를 사용한다.
  - ◆ GP2Y0A21Y는 A1부터 차례로 연결
  - ◆ 스위치는 40번에 연결
  - ◆ LED는 13번에 연결된 내부 LED 사용

- 동작
  - ◆ 스위치가 눌리지 않은 상태에서는
    - 측정한 거리가 하나라도 15cm 이내면 LED ON, 그렇지 않으면 LED OFF
  - ◆ 스위치가 눌린 상태에서는
    - 측정한 거리가 모두 15cm 이내면 LED ON, 그렇지 않으면 LED OFF
  - ◆ 센서로부터 측정한 모든 값을 시리얼 모니터로 출력할 것!
  - ◆ 참고: 15cm면 약 1.6V에 해당 → 디지털 변환 값으로는 1.6/5.0*1023.0 = 327에 해당

참고: GP2Y0A41SK0F

❖ 특징

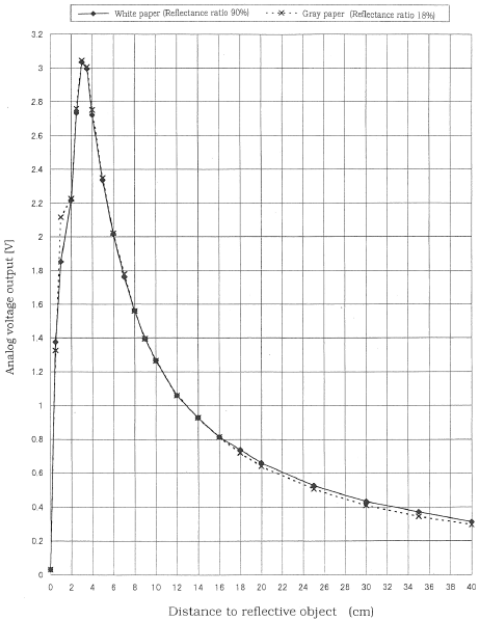
- GP2Y0A21YK0F와 형태 및 핀 배치는 동일.
- 거리 측정 범위만 4 ~ 30cm
- Datasheet에 측정 거리에 따른 출력 전압 등이 상세히 나와 있음!



Connector signal

	signal name
①	Vo
②	GND
③	Vcc

Connector :
J.S.T.TRADING COMPANY,LTD.
S3B-PH



라인 트래킹 센서 모듈

❖ 라인 트래킹 센서를 이용할 수 있다.

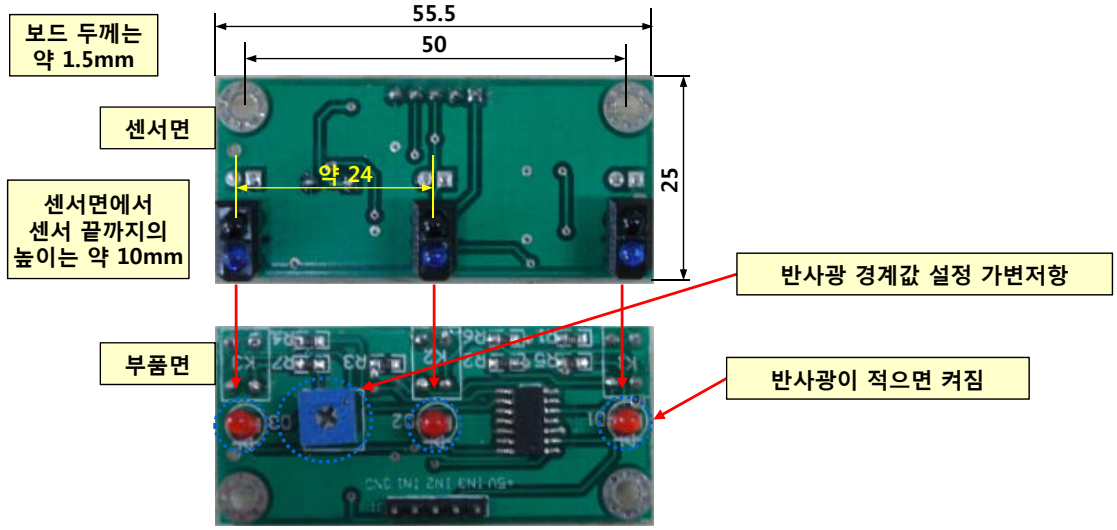
- ❖ 라인 트래킹 센서의 회로 구성을 설명할 수 있다.
- ❖ 라인 트래킹 센서의 특성을 설명할 수 있다.
- ❖ 라인 트래킹 센서를 Arduino에 연결해서 사용할 수 있다.

적외선 라인 트래킹 센서 – 특징

❖ 특징

- ■ 적외선 LED와 포토 TR이 한 몸체인 부품 사용
- ■ 가변 저항과 비교기(comparator)에 의한 출력
  - ◆ 가변 저항으로 H/L Threshold 조절 가능
    - ■ 가변 저항을 조절해서 감도를 조절해야 함!

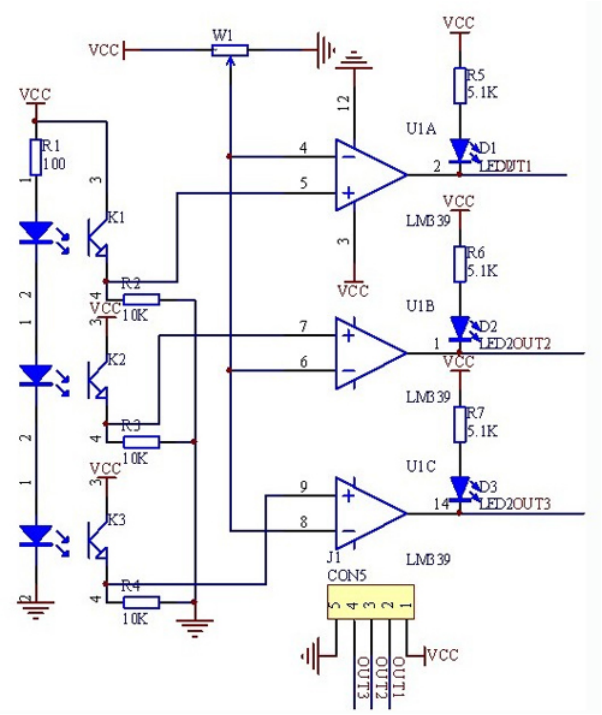
센서 끝과 라인 면까지의 거리가
15~20mm 정도 되어야
라인 인식이 잘 됨!
너무 가까워도, 너무 멀어도 안 됨!
각자 테스트 해보고 센서 설치 위치 결정할 것!



적외선 라인 트래킹 센서 – 회로도

❖ 동작 원리

- 적외선 LED는 항상 적외선을 쏘고 있음
- K1 Photo TR이 적외선을 받으면
  - ◆ K1의 C→E로 전류가 흘러 R2 전압 강하 발생
    - J1의 5번 핀 전압이 높아져 W1 가변 저항 전압보다 높게 됨
    - J1의 2번 핀이 High 전압이 되고 LED D1은 꺼짐
- K1 Photo TR이 적외선을 못 받으면
  - ◆ K1의 C→E로 전류가 안 흐름 R2 전압 강하 없음
    - J1의 5번 핀이 GND 전압이 되어 W1 가변 저항 전압보다 낮게 됨
    - J1의 2번 핀이 Low 전압이 되고 LED D1은 켜짐
- ➔ 흰 바탕은 H로 검은 라인은 L로 인식됨



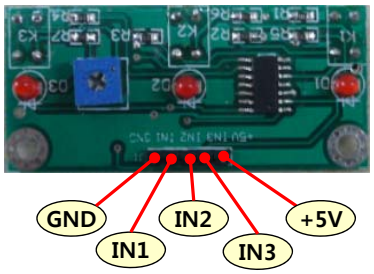
❖ 참고

- 회로에 사용된 Compator LM339는 출력 핀이 Open Collector 타입임
  - ◆ 부하가 반드시 필요함!
  - ◆ LED를 부하로 사용함!

적외선 라인 트래킹 센서 – 예제

❖ 배선

- 센서의 +5V는 Arduino의 5V에 연결
- 센서의 IN3(출력)는 Arduino의 46에 연결
- 센서의 IN2(출력)는 Arduino의 47에 연결
- 센서의 IN1(출력)는 Arduino의 48에 연결
- 센서의 GND는 Arduino의 GND에 연결



❖ 문제: IN1, IN2, IN3의 출력값을 반전시켜 변수에 저장하고 리턴하는 함수를 만들고 이 함수의 리턴값을 시리얼 모니터로 2진수로 출력한다.

- IN1은 B2 자리에, IN2는 B1 자리에 IN3는 B0 자리에 반전시켜 저장
  - ◆ Bn은 n번 비트를 의미하는 것으로 가장 오른쪽 비트가 B0임
- IN1, IN2, IN3가 각각 L, H, H면 2진수 011이 아니라 반전된 값인 2진수 100으로 저장하고 리턴

❖ 어떻게?

- val 값을 0으로 초기화
- IN1이 L면 B: val = val + 4; // 참고: bitSet(val, 2); 을 사용해도 됨
- IN2가 L면: val = val + 2; // 참고: bitSet(val, 1); 을 사용해도 됨
- IN3가 L면: val = val + 1; // 참고: bitSet(val, 0); 을 사용해도 됨
- val을 2진수로 출력: Serial.println(val, BIN);

적외선 라인 트래킹 센서 – 예제

❖ 소스 코드

P04.TrkSensor01.Combine

```
void setup() {
  Serial.begin(9600);
  pinMode(46, INPUT);
  pinMode(47, INPUT);
  pinMode(48, INPUT);
}

int getTrakingSensor(void) {
  int val = 0;

  if (digitalRead(48)==LOW)
    val += 4;
  if (digitalRead(47)==LOW)
    val += 2;
  if (digitalRead(46)==LOW)
    val += 1;

  return val;
}

void loop() {
  Serial.println(getTrakingSensor(), BIN);
  delay(100);
}
```


참고: Arduino의 비트 조작 함수

❖ void bitSet(x, n)
- x 변수의 n 비트를 1로 변경한다. n은 오른쪽 비트부터 0번부터 시작한다.

❖ void bitClear(x, n)
- x 변수의 n 비트를 0으로 변경한다. n은 오른쪽 비트부터 0번부터 시작한다.

❖ void bitWrite(x, n, b)
- x 변수의 n 비트를 b 값으로 변경한다. n은 오른쪽 비트부터 0번부터 시작한다. b는 0 또는 1

❖ bitRead(x, n)
- x 변수의 n 비트의 값을 리턴한다. n은 오른쪽 비트부터 0번부터 시작한다.
- 리턴 값은 0 또는 1

적외선 라인 트래킹 센서 – 연습 문제

❖ P04.TrkSensor02.Motion

❖ 배선

- 앞 예제와 동일하게 배선

❖ 문제

- 앞 예제에서 만든 getTrakingSensor() 함수의 리턴 값이
  - ◆ 0b111이면 "Stop" 출력
  - ◆ 0b100이거나 0b110이면 "Turn Right" 출력
  - ◆ 0b001이거나 0b011이면 "Turn Left" 출력
  - ◆ 0b000이거나 0b010이거나 0b101이면 "Go" 출력

초음파 센서 모듈

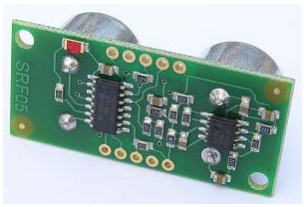
❖ 초음파 센서 모듈을 이용할 수 있다.

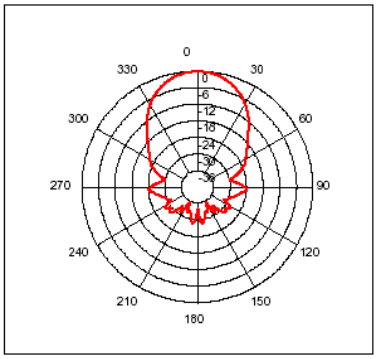
- ❖ 초음파 센서 모듈을 Arduino에 연결할 수 있다.
- ❖ 초음파 센서 모듈에 Trigger 펄스를 가하고 Echo 펄스의 길이를 측정할 수 있다.
- ❖ 초음파 센서 모듈을 이용해 물체까지의 거리를 측정할 수 있다.

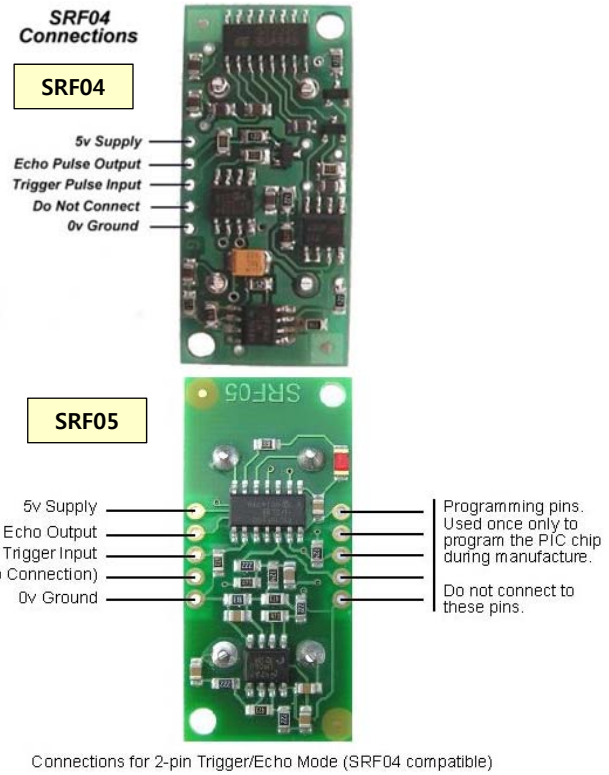
초음파 센서 모듈 – SRF04, SRF05 등

❖ 자료 출처

- SRF04 및 SRF05 Datasheet
- Range: 수 cm ~ 수 m
- Measuring Angle이 큼







Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)

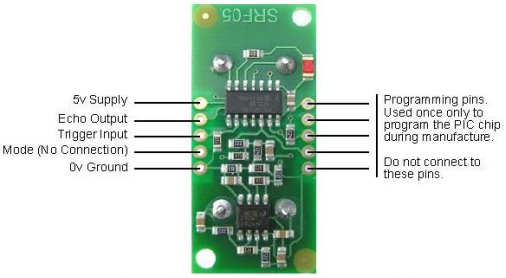
초음파 센서 모듈 – 사용 방법

❖ 배선

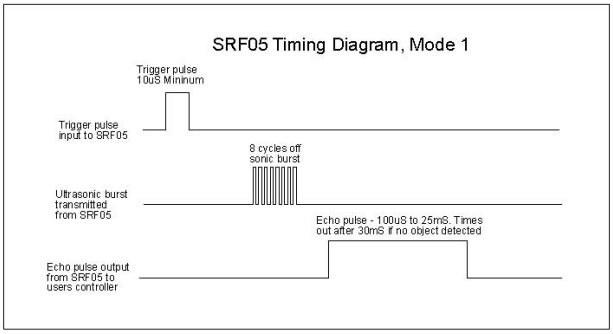
- 5V는 Arduino의 5V에 연결
- Echo는 Arduino의 디지털 입력 핀에 연결
- Trigger는 Arduino의 디지털 출력 핀에 연결
- 0V는 Arduino의 GND에 연결

❖ 사용 방법

- 초음파 센서 모듈 사용법은 대부분 비슷
- Trigger에 10 us 이상의 펄스 출력 후 Echo의 펄스 길이 측정
  - ◆ us 단위로 측정
- 이 길이에 소리의 속도/2를 곱해서 거리 계산
  - ◆ 소리의 속도: 340 m/s = 0.34 mm/us
  - ◆ 측정되는 시간은 소리가 왕복하는 시간이므로
  - ◆ 0.17을 곱하면 mm 단위의 거리로 측정됨



Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)



pulseIn()과 delayMicroseconds()

❖ pulseIn(pin, value) 또는 pulseIn(pin, value, timeout)

- pin에 해당하는 값이 value(HIGH 또는 LOW)를 유지하는 시간을 측정해서 us 단위로 리턴함
  - ◆ 리턴 값은 unsigned long 형
- timeout이 사용되는 경우, timeout 동안 펄스가 완성되지 않으면 리턴 값은 0임
  - ◆ timeout도 us 단위임
- 예1) 42번 핀이 HIGH인 시간을 측정하고 싶으면 pulseIn(42, HIGH);
- 예2) 43번 핀이 LOW인 시간을 측정하고 싶으면 pulseIn(43, LOW);
- 예3) 44번 핀이 HIGH인 시간을 30ms 안에 측정하고 싶으면 pulseIn(44, HIGH, 30000);
  - ◆ 이 때 리턴 값이 0이면 30ms 내에 양의 펄스가 완성되지 않은 것임
- 주의: 만약 value 값이 HIGH라면, 이 함수는 pin이 LOW 상태에서 HIGH 상태로 될 때까지 기다렸다가 HIGH 펄스 폭을 측정한다. 그리고 이 전 과정이 timeout 시간 내에 이루어지지 않으면 0을 리턴한다. value 값이 LOW인 경우도 마찬가지!

❖ delayMicroseconds(us)

- 마이크로초 단위의 us만큼 시간 지연을 하는 함수
- 참고: delay(ms) 함수는 밀리초 단위

초음파 센서 모듈 – 예제

❖ 배선

- 5V는 Arduino의 5V에 연결
- Echo는 Arduino의 50번 핀에 연결
- Trigger는 Arduino의 26번 핀에 연결
- 0V는 Arduino의 GND에 연결

❖ 문제

- 초음파 센서로 물체까지의 거리를 mm 단위로 측정해서 시리얼 모니터로 출력

❖ 어떻게?

- Trigger는 delayMicroseconds() 함수 이용
- Echo에는 pulseIn() 함수 이용

❖ 고찰

- 0mm으로 측정되는 것은 timout에 의한 오류임
- 초음파 센서인데 뚝뚝뚝뚝 소리가 나는 이유는?
- 측정 시간은 어떻게 될까?
- 측정각이 얼마나 될까?

P05.USonicSensor01.Measure

```
void setup() {
  Serial.begin(9600);
  pinMode(50, INPUT);
  pinMode(26, OUTPUT);
}

double getUltrasonicSensor(void) {
  unsigned long duration;

  //- Generate trigger pulse
  digitalWrite(26, HIGH);
  delayMicroseconds(10);
  digitalWrite(26, LOW);

  //- Measure echo pulse duration
  duration = pulseIn(50, HIGH, 30000UL);

  return duration * 0.17;
}

void loop() {
  Serial.print(getUltrasonicSensor(), 0);
  Serial.println("mm");
  delay(100);
}
```

초음파 센서 모듈 – 팀 과제

❖ 2인 이상이 팀을 이루어

- 두 명 또는 세 명이 한 팀을 구성한다.
- 1개의 Arduino 보드, 2개의 거리 측정 초음파 센서 모듈, 1개의 LED를 사용한다.

❖ 배선

- 초음파 센서 모듈의 Echo는 Arduino의 50번 핀과 51번 핀에 연결
  - ◆ 첫 번째 초음파 모듈의 Echo는 50번, 두 번째 초음파 모듈의 Echo는 51번에 연결
- 초음파 센서 모듈의 Trigger는 Arduino의 26번 핀과 27번 핀에 연결
  - ◆ 첫 번째 초음파 모듈의 Trigger는 26번, 두 번째 초음파 모듈의 Trigger는 27번에 연결
- LED는 Arduino 보드에 내장된 LED 사용
  - ◆ 13번 핀에 연결된 LED

❖ 문제1

- 초음파 센서로 측정한 거리가 하나라도 150~300mm 범위에 있을 때만 LED ON

❖ 문제2

- 초음파 센서로 측정한 거리가 모두 150~300mm 범위에 있을 때만 LED ON

Color Sensor 모듈

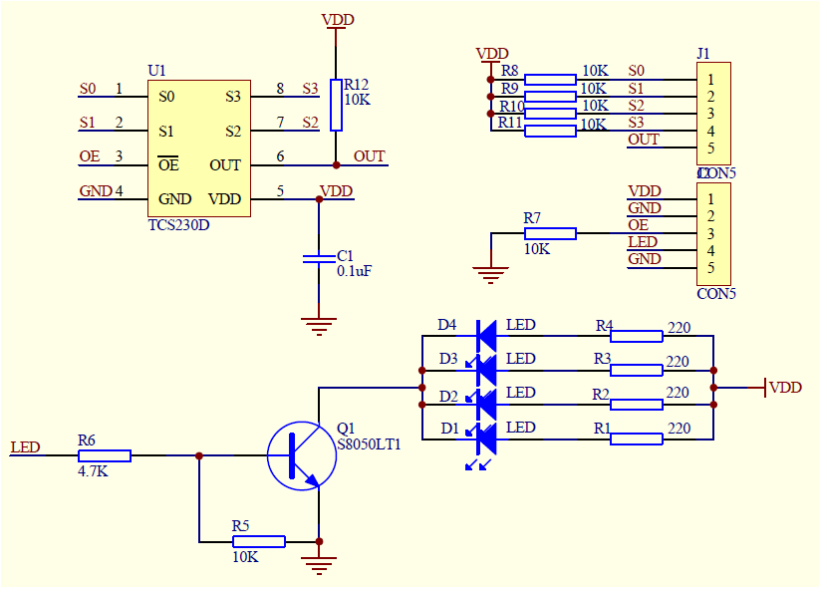
❖ 컬러 센서 모듈을 이용할 수 있다.

- ❖ 컬러 센서 모듈을 Arduino에 연결할 수 있다.
- ❖ 컬러 센서 모듈로 Red, Green, Blue 강도를 측정할 수 있다.
- ❖ Red, Green, Blue 강도로 원하는 색을 측정할 수 있다.

컬러 센서 모듈 – SEN0101

❖ DFRobot 사의 TCS3200 Color Sensor (SEN0101)

- 회로도에는 TCS230D로 되어 있으나
  실제로는 TCS3200을 사용하는 듯!
- 조명용 LED 네 개와 TCS3200 Color Sensor의 조합
  - ◆ LED로 물체를 비추고 반사된 빛을 TCS3200으로 감지하는 방식



컬러 센서 모듈 – TCS3200

❖ 데이터시트를 함께 훑어 봅시다.

- TCS3200은 R, G, B Photodiode로 측정한 결과를 주파수로 출력하는 센서
- 빛의 세기가 커질수록 ➔ 주파수가 높아진다! = 펄스 폭이 짧아진다!

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
GND	4		Power supply ground. All voltages are referenced to GND.
$\overline{OE}$	3	I	Enable for f_o (active low).
OUT	6	O	Output frequency (f_o).
S0, S1	1, 2	I	Output frequency scaling selection inputs.
S2, S3	7, 8	I	Photodiode type selection inputs.
V_{DD}	5		Supply voltage

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

컬러 센서 모듈 – 측정 방법

❖ 측정 방법

- 설정 단계
  - ◆ S0와 S1으로 Scaling을 정한다.
    - ■ 참고: 2%, 20%, 100% 가능, Scaling %가 낮을 수록 출력 주파수가 비례해서 낮아진다.
  - ◆ 조명용 LED를 꺼 둔다.
- 측정 단계
  - ◆ LED를 켠다
  - ◆ S2와 S3를 L과 L로 해서 Red 선택 후 OUT의 펄스 길이 측정
    - ■ 참고: Red의 광도가 클수록 주파수가 높아지고 펄스 길이가 짧아짐!
  - ◆ S2와 S3를 H과 H로 해서 Green 선택 후 OUT의 펄스 측정
  - ◆ S2와 S3를 L과 H로 해서 Blue 선택 후 OUT의 펄스 측정
  - ◆ LED를 끈다.
  - ◆ 측정한 Red, Green, Blue로부터 색상 판단

컬러 센서 모듈 – R, G, B 측정 예제

❖ 문제

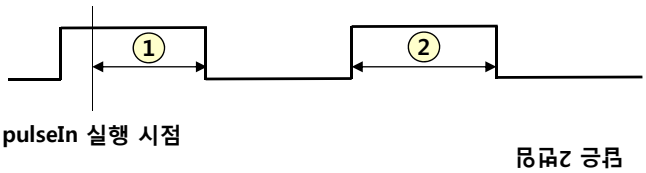
- 배선은 오른쪽 표 참조
- Scale을 20%로 한다. → 출력 주파수 가장 낮음
- 상승 펄스 폭만 측정하여 us 단위로 시리얼 모니터로 출력

Arduino	컬러 센서 모듈
29 (Output)	LED
30 (Output)	S0
31 (Output)	S1
32 (Output)	S2
33 (Output)	S3
50 (Input)	OUT
VCC	VDD
GND	GND

❖ 참고

- duration = pulseIn(50, HIGH, 10000); 을 실행할 때 맞는 것은?

1. pulseIn 함수 시작 시점에 50번 핀이 HIGH 전압이면 duration 값은 pulseIn 실행 시점부터 50번 핀이 LOW 전압이 될 때까지의 시간이다.
2. pulseIn 함수 시작 시점에 50번 핀이 HIGH 전압이면 duration 값은 다음 양의 펄스(positive pulse)를 측정한 시간이다.



컬러 센서 모듈 – R, G, B 측정 예제

❖ 소스 코드 및 고찰

- 참고: red에 1.051, green에 1.0157, blue 1.114를 곱한 값은 각 색에 대한 보정값임
- 고찰: 물체의 색상과 물체까지의 거리가 측정값에 어떤 영향을 미치는지 조사하고 빨간색, 녹색, 파란색을 어떻게 하면 구분해 낼 수 있는지 고찰하기!

P06.ColorSensor01.RGB

```
int colorLed = 29;
int colorS0 = 30;
int colorS1 = 31;
int colorS2 = 32;
int colorS3 = 33;
int colorOut = 50;

void setup() {
  Serial.begin(9600);
  pinMode(colorLed, OUTPUT);
  pinMode(colorS0, OUTPUT);
  pinMode(colorS1, OUTPUT);
  pinMode(colorS2, OUTPUT);
  pinMode(colorS3, OUTPUT);
  pinMode(colorOut, INPUT);

  //- 20% Scale
  digitalWrite(colorS0, HIGH);
  digitalWrite(colorS1, LOW);
}
```

```
void loop() {
  unsigned long red, green, blue;

  digitalWrite(colorLed, HIGH); // LED ON
  delayMicroseconds(10);

  //- red
  digitalWrite(colorS2, LOW);
  digitalWrite(colorS3, LOW);
  red = pulseIn(colorOut, LOW, 4000UL)*1.051;

  //- green
  digitalWrite(colorS2, HIGH);
  digitalWrite(colorS3, HIGH);
  green = pulseIn(colorOut, LOW, 4000UL)*1.0157;

  //- blue
  digitalWrite(colorS2, LOW);
  digitalWrite(colorS3, HIGH);
  blue = pulseIn(colorOut, LOW, 4000UL)*1.114;

  digitalWrite(colorLed, LOW); // LED OFF
```

```
  Serial.print("R=");
  Serial.print(red);
  Serial.print(", G=");
  Serial.print(green);
  Serial.print(", B=");
  Serial.print(blue);
  Serial.println("");

  delay(100);
}
```


컬러 센서 모듈 – 연습 문제

❖ P06.ColorSensor01.Measure

- 컬러 센서로 빨간색, 파란색, 녹색을 구분해서 리턴하는 getColorSensor 함수를 만드시오.
- 이 함수의 리턴 값은 다음 중 하나가 되도록 하시오.
  - #define CSEN_NONE 0 // R, G, B 구분이 어려울 때 리턴하는 값
  - #define CSEN_RED 1 // 빨간색
  - #define CSEN_GREEN 2 // 녹색
  - #define CSEN_BLUE 3 // 파란색

❖ 어떻게?

- 앞 예제의 측정 결과를 분석해서 빨간색, 파란색, 녹색을 구분하는 판단 기준을 정할 것!
- 예) 펄스 폭이 특정 경계값보다 작은 경우에만 색상을 인식한 것으로 판단하고
  - 이 때 red 펄스 폭이 가장 작으면 CSEN_RED 리턴
  - 이 때 green 펄스 폭이 가장 작으면 CSEN_GREEN 리턴
  - 이 때 blue 펄스 폭이 가장 작으면 CSEN_BLUE 리턴

컬러 센서 라이브러리 만들어 사용하기 (1)

❖ 일러 두기

- 만드는 방법은 잘 모르더라도 사용하는 방법만큼은 확실히 익혀둡시다!

❖ Arduino Libarary

- Arduino의 표준 라이브러리는 모두 C++ Class로 작성되어 제공된다.
- 일반적인 C++ 또는 C언어에서와 같이 .h와 .cpp가 한 쌍이 되어 제공된다.
- .h에는 클래스 선언이 있고, .cpp에는 클래스 구현이 있다.
- 사용자는 #include로 .h 헤더 파일을 포함시키고
  - ◆ LCD Library를 사용할 때 #include "LiquidCrystal.h" 한 것처럼...
- 필요에 따라 인스턴스를 만들고 (= 클래스 변수를 만들고)
  - ◆ LCD Library를 사용할 때 LiquidCrystal lcd(8, 9, 4, 5, 6, 7); 한 것처럼...
- 이후 필요한 멤버 함수를 사용하면 됨
  - ◆ LCD Library를 사용할 때 lcd.setCursor(0, 1); lcd.print("ABC"); 한 것처럼...

❖ 설계

- TCS3200 칩을 Arduino와 연결하고
  - ◆ 핀 연결 정보 등은 Tcs3200Class 생성자에서 초기화
- RGB를 측정하고
  - ◆ measure 함수로 구현
- RGB를 분석해서 색상을 결정하고
  - ◆ analysis 함수로 구현
- (RGB와) 색상값을 얻는다.
  - ◆ getColor 함수로 구현

컬러 센서 라이브러리 만들어 사용하기 (2)

❖ 소스 코드 – Tcs3200.h

Tcs3200.h

```
#ifndef _TCS3200_H_INCLUDE
#define _TCS3200_H_INCLUDE

#include "Arduino.h"

#define TCS3200_NONE   0
#define TCS3200_RED    1
#define TCS3200_GREEN  2
#define TCS3200_BLUE   3

class Tcs3200Class {
private:
  uint8_t _ledPin, _s0Pin, _s1Pin, _s2Pin, _s3Pin, _outPin;
  uint32_t _threshold;
  uint32_t _red, _green, _blue;
  int _color;
```

```
public:
  Tcs3200Class(uint8_t ledPin,
               uint8_t s0Pin,
               uint8_t s1Pin,
               uint8_t s2Pin,
               uint8_t s3Pin,
               uint8_t outPin,
               uint32_t threshold);
  void measure();
  void analysis();
  int getColor(uint32_t &red, uint32_t &green, uint32_t &blue);
  int getColor();
};

#endif
```

컬러 센서 라이브러리 만들어 사용하기 (3)

❖ Tcs3200Class 제공 함수 설명

- **Tcs3200Class(uint8_t ledPin, uint8_t s0Pin, uint8_t s1Pin, uint8_t s2Pin, uint8_t s3Pin, uint8_t outPin, uint32_t threshold)**
  - ◆ Tcs3200 생성자 함수로 파라미터의 의미는 순서대로 Arduino와 연결된 LED 핀 번호, S0 핀 번호, S1 핀 번호, S2 핀 번호, S3 핀 번호, OUT 핀 번호, 색상 판단 경계값

- **getColor() 또는 getColor(red, green, blue)**
  - ◆ 두 함수 리턴 값은 "Tcs3200.h"에 선언된 색 정보
    - #define TCS3200_NONE 0
    - #define TCS3200_RED 1
    - #define TCS3200_GREEN 2
    - #define TCS3200_BLUE 3
  - ◆ 파라미터 red, green, blue를 사용하면 각각에 대한 펄스 폭(us 단위)도 얻을 수 있다.

컬러 센서 라이브러리 만들어 사용하기 (4)

❖ 소스 코드 – Tcs3200.cpp

Tcs3200.cpp

```
#include "Tcs3200.h"

Tcs3200Class::Tcs3200Class(uint8_t ledPin,
                           uint8_t s0Pin,
                           uint8_t s1Pin,
                           uint8_t s2Pin,
                           uint8_t s3Pin,
                           uint8_t outPin,
                           uint32_t threshold)
{
  _ledPin = ledPin;
  _s0Pin = s0Pin;
  _s1Pin = s1Pin;
  _s2Pin = s2Pin;
  _s3Pin = s3Pin;
  _outPin = outPin;
  _threshold = threshold;

  pinMode(_ledPin, OUTPUT);
  pinMode(_s0Pin, OUTPUT);
  pinMode(_s1Pin, OUTPUT);
  pinMode(_s2Pin, OUTPUT);
  pinMode(_s3Pin, OUTPUT);
  pinMode(_outPin, INPUT);

  digitalWrite(_ledPin, LOW);
  digitalWrite(_s0Pin, HIGH);
  digitalWrite(_s1Pin, LOW);

}
```

```
void Tcs3200Class::measure()
{
  digitalWrite(_ledPin, HIGH); // LED ON
  delayMicroseconds(10);

  digitalWrite(_s2Pin, LOW);
  digitalWrite(_s3Pin, LOW);
  _red = pulseIn(_outPin, LOW, 4000UL)*1.051;

  digitalWrite(_s2Pin, HIGH);
  digitalWrite(_s3Pin, HIGH);
  _green = pulseIn(_outPin, LOW, 4000UL)*1.0157;

  digitalWrite(_s2Pin, LOW);
  digitalWrite(_s3Pin, HIGH);
  _blue = pulseIn(_outPin, LOW, 4000UL)*1.114;

  digitalWrite(_ledPin, LOW); // LED OFF
}
```

```
void Tcs3200Class::analysis()
{
  if (_red < _threshold || _green < _threshold || _blue
< _threshold) {
    if (_red < _green && _red < _blue)
      _color = TCS3200_RED;
    if (_green < _red && _green < _blue)
      _color = TCS3200_GREEN;
    if (_blue < _red && _blue < _green)
      _color = TCS3200_BLUE;
  }
  else {
    _color = TCS3200_NONE;
  }
}

int Tcs3200Class::getColor(uint32_t &red, uint32_t
&green, uint32_t &blue)
{
  measure();
  analysis();

  red = _red;
  green = _green;
  blue = _blue;

  return _color;
}

int Tcs3200Class::getColor()
{
  measure();
  analysis();

  return _color;
}
```

컬러 센서 라이브러리 만들어 사용하기 (5)

❖ 사용 예제

- Tcs3200.h와 Tcs3200.cpp를 .ino 파일과 같은 폴더에 넣어 두어야 한다!

P06.ColorSensor03.Library

```
#include "Tcs3200.h"

// 파라미터의 의미는 순서대로
// LED핀, S0핀, S1핀, S2핀, S3핀, OUT핀, 색상 판단 경계값
Tcs3200Class Tcs3200(29, 30, 31, 32, 33, 50, 70L);

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

void loop() {
  int color;
  unsigned long r, g, b;
```

```
#if 1 // 1이면 색상만 출력, 0이면 RGB값과 색상 출력
  color = Tcs3200.getColor();
#else
  color = Tcs3200.getColor(r, g, b);
  Serial.print(r);
  Serial.print(", ");
  Serial.print(g);
  Serial.print(", ");
  Serial.print(b);
  Serial.print(": ");
#endif

  switch (color) {
    case TCS3200_NONE:
      Serial.println("?");
      break;
    case TCS3200_RED:
      Serial.println("Red");
      break;
    case TCS3200_GREEN:
      Serial.println("Green");
      break;
    case TCS3200_BLUE:
      Serial.println("Blue");
      break;
  }
  delay(100);
}
```


DC 모터와 드라이버의 활용

❖ 모터 드라이버 MDRIVER 보드를 사용해서 DC 모터를 구동할 수 있다.

- ❖ L298 모터 드라이버의 역할을 설명할 수 있다.
- ❖ MDRIVER 보드에 전원은 연결할 수 있다.
- ❖ Arduino에 MDRIVER 보드를 연결할 수 있다.
- ❖ MDRIVER 보드를 이용해 4개의 모터를 각각 구동할 수 있다.

MCU로 모터를 구동하려면

❖ MCU로 모터를 구동하려면?

- MCU의 디지털이나 아날로그 포트로 출력할 수 있는 전력이 작으므로 전력을 키울 수 있는 회로를 추가해서 모터를 구동한다. 즉, 모터 드라이버를 추가해서 모터를 구동한다.

MCU → 모터 드라이버 → 모터

❖ 방법은?

- 한 방향으로 한 전압으로만 ON/OFF 하려면
  - ◆ 릴레이, TR, FET 등을 한 개 이용해서 모터를 ON/OFF
- 양 방향으로 한 전압으로만 ON/OFF 하려면
  - ◆ 릴레이 두 개 이상 이용해서 모터를 정/역으로 ON/OFF
  - ◆ TR이나 FET 4개로 브리지 회로를 구성해서 모터를 정/역으로 ON/OFF
- 임의의 전압이나 전류로 모터를 구동하려면
  - ◆ Linear Amplifier 회로를 적용해 구동
  - ◆ FET 브리지 회로와 PWM을 이용해 구동 ← 일반적인 모터 구동 방법

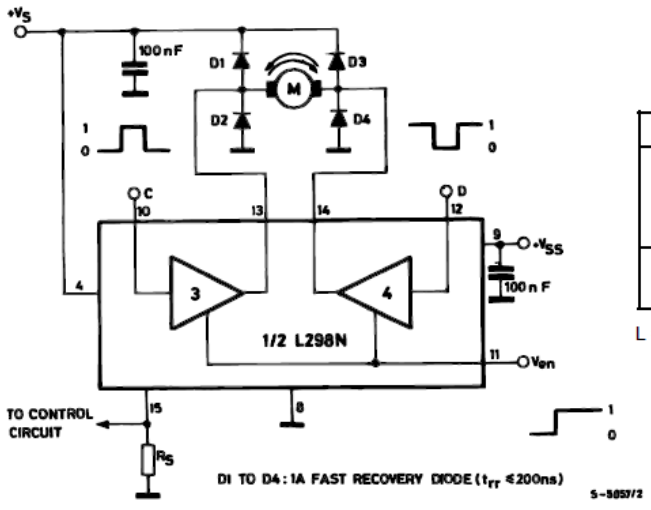
❖ 일반적인 모터 드리이버 칩이란?

- FET로 구성된 브리지 회로를 기본으로 해서 PWM 입력에 대한 모터 전력 공급 스위칭을 안정적으로 해주는 역할을 함
- 참고: 브리지 회로의 스위칭을 잘못하면 FET가 파괴되므로 직접 브리지 회로를 구성해서 MCU로 스위칭을 할 때는 매우 주의해야 한다. (학생들에게는 권장하지 않는 방법)

L298 모터 드라이버 칩

❖ 특징

- STMicroelectronics 사의 2채널 풀 브리지 모터 드라이버 칩
- Datasheet을 통해 패키지 타입, 핀 설명, 스펙, 구동 방법 등을 학습!



Inputs		Function
Ven = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
Ven = L	C = X ; D = X	Free Running Motor Stop

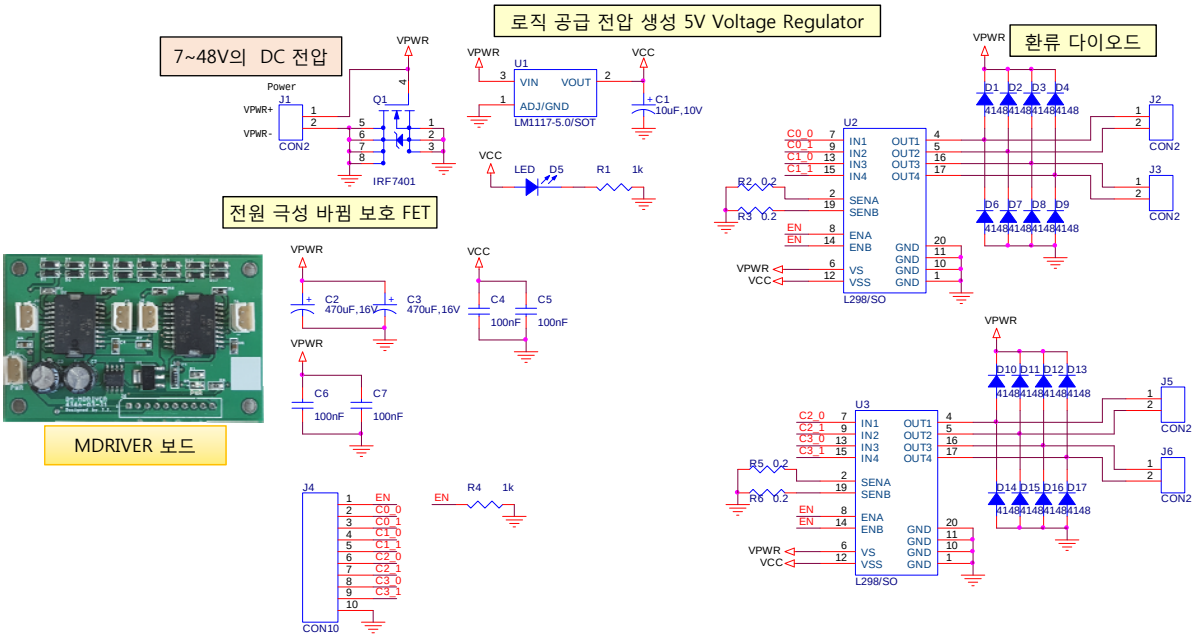
L = Low H = High X = Don't care

L298 데이터시트에서 발췌

MDRIVER는

❖ L298를 두 개 이용한 4채널 DC 모터 드라이버 보드

- PWM을 이용해서 모터에 공급하는 평균 전압을 양방향으로 조절할 수 있는 드라이버
- 입력은 0-5V의 PWM, 모터 출력은 2A까지의 전류를 보장하는 0-VPWR의 PWM 파형이 된다



MDRIVER를 사용할 때의 주의점

❖ 전원

- 7~48V 사이
- 모터에 인가할 최대 전압보다 전원 전압이 커야 한다.
- 모터를 구동할 전류를 충분히 공급할 수 있어야 한다.

❖ 모터 상식

- 모터를 다루는 교과목에서 자세히...
- 같은 전압을 인가하더라도 속도가 낮을 때 전류가 많이 흐른다
  - ◆ 같은 전압에 대해 부하가 커질수록 속도가 낮아져 전류가 많이 흐름
  - ◆ 모터에 전압을 인가한 상태에서 모터 회전이 방해를 받아 멈출 때 전류가 많이 흐름.
    - 이 때의 전류를 Stall Current라고 함

❖ 참고: 창의과제기초 및 창의과제응용에서 사용하는 모터와 공급 전압

- 로봇 이동용 DC 모터는 7.2V에서의 Stall Current는 1A 정도임.
  - ◆ MDRIVER에 7.2V 전원을 사용할 때 로봇 이동용 DC 모터에 최대 전압을 사용해도 됨
- 타미야 DC 모터는 7.2V에서의 Stall Current가 4A 이상임
  - ◆ MDRIVER가 채널 당 최대 사용 가능 전류가 2A이내이므로
  - ◆ MDRIVER에 7.2V 전원을 사용할 때 타미야 DC 모터에 7.2V를 계속 인가하면 드라이버나 모터가 탈 수 있음!
  - ◆ PWM으로 평균 전압을 약 3.6V 이내로 할 것!
    - 순간적으로는 높은 전압을 사용해도 되나 계속 높은 전압을 사용하면 과전류로 모터나 드라이버가 탈 수 있다.

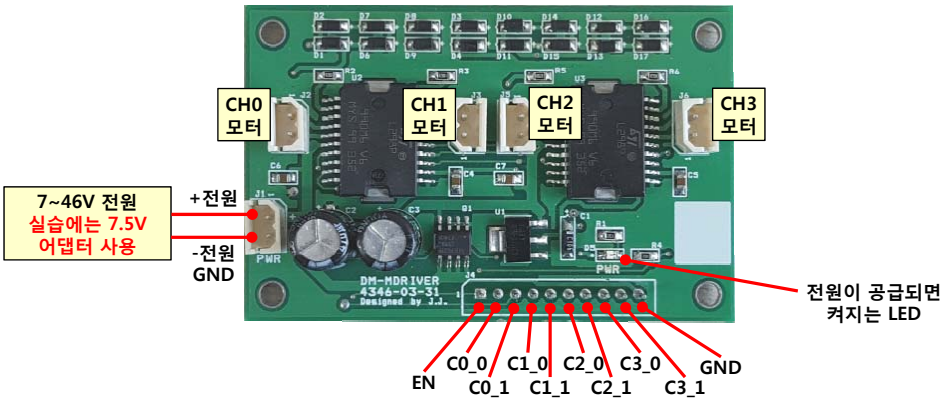
MDRIVER와 Arduino 연결 시 주의 사항

❖ Arduino의 전압 출력을 MDRIVER 전원으로 사용하지 말 것!

- Arduino에서 USB 전원을 사용한다면 USB 전원이 5V이고 0.5A가 최대이므로 사용 불가
- Arduino에서 별도 전원을 사용한다면 Arduino의 전원 극성 바뀜 보호 다이오드 SS14의 용량이 1A까지이므로 문제가 될 수 있음

❖ MDRIVER와 Arduino 연결 시 전원을 끈 상태로 연결

- 잘못된 배선으로 과전류가 흐를 수 있으므로 주의
- 전원 투입 후 GND 선을 절대 빼지 말 것! → 기준 전압 불일치로 회로에 문제가 발생할 수 있음



전원 및 모터 주의 사항

❖ 실습에 사용하는 MDRIVER 보드 전원

- DC 7.5V, 1.2A 어댑터를 사용
  - ◆ 출력 전압이 DC 7.5V이고, 최대 1.2A까지만 사용하라는 의미
  - ◆ MDRIVER에 이 어댑터를 사용한다면 모터에 부하를 가하지 말 것!
    - 부하를 가하면 과전류가 흘러 어댑터에 문제가 발생할 수 있음
  - ◆ MDRIVER에 이 어댑터를 사용한다면 모터를 세 개 이상 동시에 구동하지 말 것

❖ 실습에 사용하는 DC 모터

- 엔코더가 부착된 DC 기어드 모터
  - ◆ 엔코더는 추후 학습
- 모터 커넥터 핀별 내용
  - ◆ 1. Black : - MOTOR
  - ◆ 2. Red : + MOTOR
  - ◆ 3. Brown : HALL SENSOR Vcc
  - ◆ 4. Green : HALL SENSOR GND
  - ◆ 5. Blue : HALL SENSOR B Vout
  - ◆ 6. Purple : HALL SENSOR A Vout
- 주의 사항:
  - ◆ MDRIVER에는 1.Black과 2.Red만 사용할 것!

출처: 모터뱅크 홈페이지

MDRIVER EN에 PWM을 이용한 모터 전압 제어 (1)

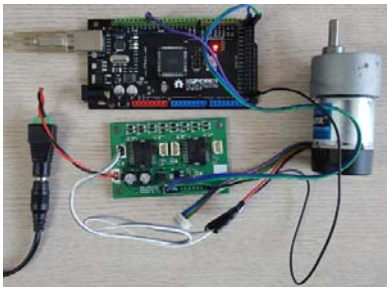
❖ CH0의 경우의 예

- C0_0과 C0_1로 모터의 방향을 설정하고
  - ◆ H, L이면 정방향, L, H면 역방향
- EN으로 모터의 평균 전압을 조절
  - ◆ PWM Duty Cycle %가 클수록 모터 양단의 평균 전압이 커짐

Arduino	MDRIVER	
PWMx	EN	
DO_a	C0_0	CH0 모터 (J2)
DO_b	C0_1	
DO_c	C1_0	CH1 모터 (J3)
DO_d	C1_1	
DO_e	C2_0	CH2 모터 (J5)
DO_f	C2_1	
DO_g	C3_0	CH3 모터 (J6)
DO_h	C3_1	
GND	GND	

❖ 단점

- MDRIVER 회로에서
  모든 채널의 EN이 묶여 있기 때문에
  두 채널을 서로 다른 PWM Duty Cycle로
  구동할 수 없다!



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

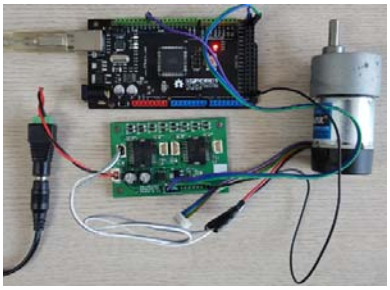
L = Low H = High X = Don't care

MDRIVER EN에 PWM을 이용한 모터 전압 제어 (2)

❖ 연결

- MDRIVER에 전원 연결 (그림 참조)
- Arduino와 MDRIVER 연결 (표 참조)
- 모터를 CH0에 연결 (그림 참조)

Arduino	MDRIVER
9	EN
22	C0_0
23	C0_1
GND	GND



❖ 연습 문제 (P07.MDriver01.Basic01)

- 모터의 방향과 PWM Duty Cycle을 변경해 가며
  모터의 방향과 평균 전압을 조절해 본다.
- 채널도 변경해 본다!
  - ◆ 모터 연결 커넥터를 CHn으로 변경
  - ◆ C0_0, C0_1 대신 Cn_0, Cn_1 이용
- 고찰
  - ◆ 낮은 PWM Duty Cycle에서
    모터가 회전을 안 하는 이유는?
    - 정지 마찰

P07.MDriver01.Basic01

```
void setup() {
  Serial.begin(9600);
  pinMode(22, OUTPUT);
  pinMode(23, OUTPUT);
  digitalWrite(22, LOW);  // 변경해 볼 것!
  digitalWrite(23, HIGH); // 변경해 볼 것!
  analogWrite(9, 120);   // 변경해 볼 것!
}

void loop() {
}
```

MDRIVER EN에 PWM을 이용한 모터 전압 제어 (3)

❖ 연습 문제 (P07.MDriver01.Basic02)

- 시리얼 모니터로 입력한 숫자에 따라 analogWrite 함수의 PWM Duty Cycle 값과 방향을 변경
  - ◆ 7을 전송하면 PWM 값을 200으로 모터 정방향 회전
  - ◆ 6을 전송하면 PWM 값을 120으로 모터 정방향 회전
  - ◆ 5를 전송하면 모터 멈춤
  - ◆ 4를 전송하면 PWM 값을 120으로 모터 역방향 회전
  - ◆ 3을 전송하면 PWM 값을 200으로 모터 정방향 회전

P07.MDriver01.Basic02

```
void setup() {
  Serial.begin(9600);
  pinMode(22, OUTPUT);
  pinMode(23, OUTPUT);
  analogWrite(9, 0);
}

void loop() {
  if (Serial.available()) {
    switch (Serial.read()) {
      case '7':
        digitalWrite(22, HIGH);
        digitalWrite(23, LOW);
        analogWrite(9, 200);
        break;
```

```
      case '6':
        digitalWrite(22, HIGH);
        digitalWrite(23, LOW);
        analogWrite(9, 120);
        break;
      case '5':
        digitalWrite(22, LOW);
        digitalWrite(23, LOW);
        analogWrite(9, 0);
        break;
```

```
      case '4':
        digitalWrite(22, LOW);
        digitalWrite(23, HIGH);
        analogWrite(9, 120);
        break;
      case '3':
        digitalWrite(22, LOW);
        digitalWrite(23, HIGH);
        analogWrite(9, 200);
        break;
    }
  }
}
```

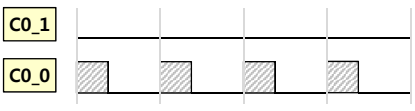

MDRIVER 다채널 동시 제어 (1)

❖ 어떻게? (CH0을 예로 설명)

- MDRIVER의 EN에 HIGH 전압을 인가한 상태에서

- 정방향으로 모터의 전압을 조절하고 싶으면
  - ◆ C0_1에 LOW 전압을 인가하고
  - ◆ C0_0에 PWM 신호로 조절 → PWM Duty Cycle 값이 클수록 평균전압이 커짐

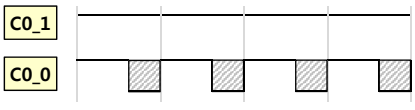
예) digitalWrite(C0_1Pin, LOW);
analogWrite(pwm0Pin, pwm);



빗금 친 부분에서만 정방향 나머지는 STOP

- 정방향으로 모터의 전압을 조절하고 싶으면
  - ◆ C0_1에 HIGH 전압을 인가하고
  - ◆ C0_0에 PWM 신호로 조절 → PWM Duty Cycle 값이 클수록 평균전압이 작아짐
    - 0~255의 PWM값을 적용하려면 255에서 뺀 값을 적용하면 이해하기 편함

예) digitalWrite(C0_1Pin, HIGH);
analogWrite(pwm0Pin, 255-pwm);



빗금 친 부분에서만 역방향 나머지는 STOP

MDRIVER 다채널 동시 제어 (2)

❖ 연결

- MDRIVER에 전원 연결 (앞 관련 그림 참조)
- Arduino와 MDRIVER 연결 (표 참조)
- 모터를 CHn에 연결 (그림 참조)

Arduino	MDRIVER
22	EN
9	C0_0
23	C0_1
10	C1_0
24	C1_1
11	C2_0
25	C2_1
12	C3_0
26	C3_1
GND	GND

❖ 연습 문제 (P07.MDriver01.Basic03)

- driveMotor(int channel, int pwm) 함수를 만들어
  - ◆ channel은 0~3의 값으로 MDRIVE의 채널 선택
  - ◆ pwm은 -255~255 범위로 음수는 역방향, 양수는 정방향, 절대값은 0~255로 pwm duty cycle 0~100% 해당
- CHn 모터를 정방향으로 pwm 0에서 200까지 5ms마다 1씩 증가시켰다가
  200 에서 0까지 5ms마다 1씩 감소켰다가
  0에서 -200까지 5ms마다 1씩 감소시켰다가
  -200에서 0까지 5ms마다 1씩 증가시키기는 것을
  전체 반복
- 참고: 동시에 두 개 이상 구동하는 예제는 이후 팀 과제에서...

MDRIVER 다채널 동시 제어 (3)

❖ 소스 코드 (P07.MDriver01.Basic03)

P07.MDriver01.Basic03

```
int enPin = 22;
int c00Pin = 9;
int c01Pin = 23;
int c10Pin = 10;
int c11Pin = 24;
int c20Pin = 11;
int c21Pin = 25;
int c30Pin = 12;
int c31Pin = 26;

void setup() {
  Serial.begin(9600);
  pinMode(enPin, OUTPUT);
  pinMode(c01Pin, OUTPUT);
  pinMode(c11Pin, OUTPUT);
  pinMode(c21Pin, OUTPUT);
  pinMode(c31Pin, OUTPUT);

  digitalWrite(enPin, HIGH);
}

void driveMotor(int channel, int pwm) {

  if (pwm < -255)
    pwm = -255;
  if (pwm > 255)
    pwm = 255;

  if (channel == 0) {
    if (pwm >= 0) {
      digitalWrite(c01Pin, LOW);
      analogWrite(c00Pin, pwm);
    }
    else {
      digitalWrite(c01Pin, HIGH);
      analogWrite(c00Pin, 255 + pwm);
    }
  }
  else if (channel == 1) {
    if (pwm >= 0) {
      digitalWrite(c11Pin, LOW);
      analogWrite(c10Pin, pwm);
    }
    else {
      digitalWrite(c11Pin, HIGH);
      analogWrite(c10Pin, 255 + pwm);
    }
  }
  else if (channel == 2) {
    if (pwm >= 0) {
      digitalWrite(c21Pin, LOW);
      analogWrite(c20Pin, pwm);
    }
    else {
      digitalWrite(c21Pin, HIGH);
      analogWrite(c20Pin, 255 + pwm);
    }
  }
  else if (channel == 3) {
    if (pwm >= 0) {
      digitalWrite(c31Pin, LOW);
      analogWrite(c30Pin, pwm);
    }
    else {
      digitalWrite(c31Pin, HIGH);
      analogWrite(c30Pin, 255 + pwm);
    }
  }
}

#define CHANNEL 3

void loop() {
  int i;

  for (i = 0; i <= 200; i++) {
    driveMotor(CHANNEL, i);
    delay(5);
  }
  for (i = 200; i >= 0; i--) {
    driveMotor(CHANNEL, i);
    delay(5);
  }
  for (i = 0; i >= -200; i--) {
    driveMotor(CHANNEL, i);
    delay(5);
  }
  for (i = -200; i <= 0; i++) {
    driveMotor(CHANNEL, i);
    delay(5);
  }
}
```

MDRIVER Library 만들어 사용하기 (1)

❖ 설계

- MDRIVER 보드를 Arduino와 연결하고
  - ◆ 핀 연결 정보 등은 MDriverClass 생성자에서 초기화
- n 채널 모터를 -255~255 범위의 값으로 회전(음수는 역회전, 양수는 정회전, 절대값은 평균 전압)
  - ◆ drive(channel, pwm)

❖ 소스 코드: MDriver.h

MDriver.h

```
#define MDRV_CH0 0
#define MDRV_CH1 1
#define MDRV_CH2 2
#define MDRV_CH3 3

class MDriverClass {
private:
  uint8_t _enPin;
  uint8_t _c00Pin, _c01Pin;
  uint8_t _c10Pin, _c11Pin;
  uint8_t _c20Pin, _c21Pin;
  uint8_t _c30Pin, _c31Pin;
```

```
public:
  MDriverClass(uint8_t enPin,
               uint8_t c00Pin,
               uint8_t c01Pin,
               uint8_t c10Pin,
               uint8_t c11Pin,
               uint8_t c20Pin,
               uint8_t c21Pin,
               uint8_t c30Pin,
               uint8_t c31Pin);


  void drive(int channel, int pwm);
};

#endif
```


MDRIVER Library 만들어 사용하기 (2)

❖ MDriverClass 제공 함수 설명

- MDriverClass(uint8_t enPin, uint8_t c00Pin, uint8_t c01Pin, uint8_t c10Pin, uint8_t c11Pin, uint8_t c20Pin, uint8_t c21Pin, uint8_t c30Pin, uint8_t c31Pin)
  - ◆ MDriverClass 생성자 함수로 파라미터의 의미는 순서대로 Arduino와 연결된 EN 핀 번호, C0_0 핀 번호, C0_1 핀 번호, ... 임
  - ◆ C00Pin, C10Pin, C20Pin, C30Pin은 Arduino의 PWM 핀으로 설정해야 한다. (Arduino Mega 2560의 경우 2~13 중에서 선택)

- drive(channel, pwm)
  - ◆ channel: MDRIVER 보드의 채널 CH0, CH1, CH2, CH3으로 Mdriver.h에 선언된 매크로 상수 이용 가능
    - #define MDRV_CH0 0
    - #define MDRV_CH1 1
    - #define MDRV_CH2 2
    - #define MDRV_CH3 3
  - ◆ pwm: -255~255 범위의 PWM Duty Cycle 값에 해당
    - 양수일 경우에는 이 값에 비례한 평균 전압으로 모터를 정방향으로 회전시킴
    - 음수일 경우에는 이 값의 절대값에 비례한 평균 전압으로 모터를 역방향으로 회전시킴

MDRIVER Library 만들어 사용하기 (3)

❖ 소스 코드: MDriver.cpp

MDriver.cpp

```
#include "MDriver.h"

MDriverClass::MDriverClass(uint8_t enPin,
                           uint8_t c00Pin,
                           uint8_t c01Pin,
                           uint8_t c10Pin,
                           uint8_t c11Pin,
                           uint8_t c20Pin,
                           uint8_t c21Pin,
                           uint8_t c30Pin,
                           uint8_t c31Pin)
{
  _enPin = enPin;
  _c00Pin = c00Pin;
  _c01Pin = c01Pin;
  _c10Pin = c10Pin;
  _c11Pin = c11Pin;
  _c20Pin = c20Pin;
  _c21Pin = c21Pin;
  _c30Pin = c30Pin;
  _c31Pin = c31Pin;

  pinMode(enPin, OUTPUT);
  pinMode(_c01Pin, OUTPUT);
  pinMode(_c11Pin, OUTPUT);
  pinMode(_c21Pin, OUTPUT);
  pinMode(_c31Pin, OUTPUT);

  digitalWrite(_enPin, HIGH);
}
```

```
void MDriverClass::drive(int channel, int pwm)
{
  if (pwm < -255)
    pwm = -255;
  if (pwm > 255)
    pwm = 255;

  if (channel == 0) {
    if (pwm >= 0) {
      digitalWrite(_c01Pin, LOW);
      analogWrite(_c00Pin, pwm);
    }
    else {
      digitalWrite(_c01Pin, HIGH);
      analogWrite(_c00Pin, 255 + pwm);
    }
  }
  else if (channel == 1) {
    if (pwm >= 0) {
      digitalWrite(_c11Pin, LOW);
      analogWrite(_c10Pin, pwm);
    }
    else {
      digitalWrite(_c11Pin, HIGH);
      analogWrite(_c10Pin, 255 + pwm);
    }
  }
```

```
  else if (channel == 2) {
    if (pwm >= 0) {
      digitalWrite(_c21Pin, LOW);
      analogWrite(_c20Pin, pwm);
    }
    else {
      digitalWrite(_c21Pin, HIGH);
      analogWrite(_c20Pin, 255 + pwm);
    }
  }
  else if (channel == 3) {
    if (pwm >= 0) {
      digitalWrite(_c31Pin, LOW);
      analogWrite(_c30Pin, pwm);
    }
    else {
      digitalWrite(_c31Pin, HIGH);
      analogWrite(_c30Pin, 255 + pwm);
    }
  }
}
```


MDRIVER Library 만들어 사용하기 (4)

❖ 사용 예제

- 주의: MDriver.h와 MDriver.cpp를 .ino 파일과 같은 폴더에 넣어 두어야 한다!
- 동작
  - ◆ CHn 모터를 정방향으로 pwm 0에서 200까지 5ms마다 1씩 증가시켰다가
    200 에서 0까지 5ms마다 1씩 감소켰다가
    0에서 -200까지 5ms마다 1씩 감소시켰다가
    -200에서 0까지 5ms마다 1씩 증가시키기는 것을
    전체 반복

P07.MDriver02.Library

```
#include "MDriver.h"

// 파라미터는 EN, C00, C01, C10, C11, C20, C21, C30, C31 순
// Cn0은 PWM 핀,EN과 Cn1은 디지털 핀 사용
MDriverClass MDriver(22, 9, 23, 10, 24, 11, 25, 12, 26);

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

```
void loop() {
  int i, channel = MDRV_CH0;

  for (i = 0; i <= 200; i++) {
    MDriver.drive(channel, i);
    delay(5);
  }
  for (i = 200; i >= 0; i--) {
    MDriver.drive(channel, i);
    delay(5);
  }
  for (i = 0; i >= -200; i--) {
    MDriver.drive(channel, i);
    delay(5);
  }
  for (i = -200; i <= 0; i++) {
    MDriver.drive(channel, i);
    delay(5);
  }
}
```

MDRIVER 팀 과제

❖ 팀 과제 1 – 로봇 이동시 리밋 스위치나 특정 물체 발견시 멈춤

- 2인 이상이 한 팀이 되어 Arduino 보드 1개, MDRIVER 보드 1개, 스위치 모듈 1개, 컬러 센서 1개, DC 모터 1개를 사용한다.
- 스위치를 안 누를 때는 모터를 120의 pwm 값으로 동작시키고 스위치를 누르거나 빨간색이 감지되면 모터를 즉시 멈춘다.

❖ 팀 과제 2 – 자동화 라인 속도 조절

- 2인 이상이 한 팀이 되어 Arduino 보드 1개, MDRIVER 보드 1개, 스위치 모듈 2개(1번, 2번), DC 모터 1개를 사용한다.
- 1번 스위치를 누를 때마다 pwm 값을 20씩 증가시킨다. 최대값은 200
- 2번 스위치를 누를 때마다 pwm 값을 -20씩 증가시킨다. 최소값은 -200

❖ 팀 과제 3 – 로봇 라인 트래킹 알고리즘

- 2인 이상이 한 팀이 되어 Arduino 보드 1개, MDRIVER 보드 1개, 트래킹 센서 1개, DC 모터 2개를 사용한다.
- 앞서 학습한 트래킹 센서의 getTrackingSensor() 함수를 이용해 이 리턴 값이
  - ◆ 0b111이면 CH0, CH1 모터 모두 정지
  - ◆ 0b100이거나 0b110이면 CH0 모터 정지, CH1 모터 120의 pwm 값으로 회전
  - ◆ 0b001이거나 0b011이면 CH0 모터 120 pwm 값으로 회전, CH1 모터 정지
  - ◆ 0b000이거나 0b010이거나 0b101이면 CH0, CH1 모터 120 pwm 값으로 회전

종합 예제

❖ 지금까지 배운 센서와 모터를 종합한 문제를 스스로 내고 해결해 보세요!

스테핑 모터의 구동

❖ MDRIVER 보드를 이용해 스테핑 모터를 구동할 수 있다.

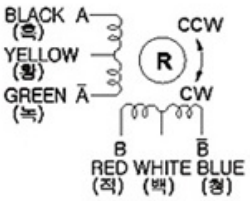
- ❖ 스테핑 모터의 구동 방법을 설명할 수 있다.
- ❖ MDRIVER를 이용해 스테핑 모터를 바이폴라 방식으로 구동할 수 있다.
- ❖ Arduino Stepper Library를 사용해서 스테핑 모터를 구동할 수 있다.
- ❖ MyStepper Library를 사용해서 스테핑 모터를 구동할 수 있다.

실습용 스테핑 모터

❖ NK243-02AT (출처: http://www.devicemart.co.kr/29707)

■ 2상 6선식 스테핑 모터로 한 스텝이 1.8도

■Circuit Diagram

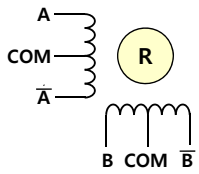


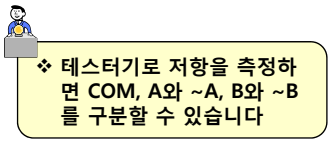
Model	Model	Voltage (V)	Current (A/Phase)	Resistance (Ω/Phase)	Inductance (mH/Phase)	Holding torque (gf-cm)	Detent torque (gf.cm)	Rotor inertia (g-㎠)	Length (mm)
STANDARD MOTOR	NK243-01AT	4.0	0.95	4.2	3.3	1.6	120	33	33
	NK243-02AT	6.0	0.6	10	9.5	1.6	120	33	33
	NK243-03AT	9.6	0.4	24	12	1.6	120	33	33
	NK243-04AT	12.0	0.31	39	23	1.8	120	33	33
SUPER TORQUE	NK243-01AT	4.56	0.95	4.8	3.1	2.1	120	33	33
HIGH TORQUE	NK243-01AT	3.7	0.95	3.9	3.6	1.9	120	33	31

스테핑 모터의 유니폴라 구동

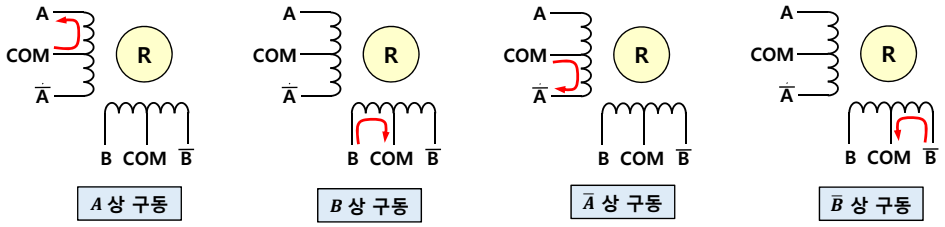
❖ 2상 6선식의 스테핑 모터의 유니폴라(Unipolar) 구동

- COM을 +전원 또는 GND에 연결하고 $A, \bar{A}, B, \bar{B}$를 스위칭하여 모터 회전자(rotor) 회전





- 다양한 구동 방법이 있으나 여기서는 1상 여자 방식만 소개 (다른 방식은 관련 교과목 참고)
  - ◆ 드라이버 칩 등을 이용해서 $A, \bar{A}, B, \bar{B}$를 스위칭하여 구동
  - ◆ 정회전: $A \rightarrow B \rightarrow \bar{A} \rightarrow \bar{B} \rightarrow A \cdots$ 순으로 구동하면 각 스텝마다 정방향으로 한 스텝씩 회전
  - ◆ 역회전: 정회전의 반대 순으로 구동하면 각 스텝마다 역방향으로 한 스텝씩 회전

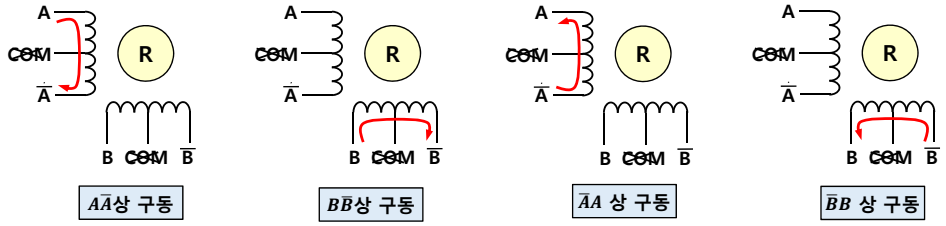


COM에 +전원을 연결한 경우의 $A, B, \bar{A}, \bar{B}$ 구동 전류 방향(빨간색 화살표)

스테핑 모터의 바이폴라 구동

❖ 2상 6선식의 스테핑 모터의 바이폴라(Unipolar) 1상 여자 구동

- COM을 사용하지 않고 $A, \bar{A}, B, \bar{B}$를 스위칭하여 모터 회전자(rotor) 회전
- 정회전: $A\bar{A} \rightarrow B\bar{B} \rightarrow \bar{A}A \rightarrow \bar{B}B \rightarrow A\bar{A}$ …순으로 구동하면 각 스텝마다 정방향으로 한 스텝씩 회전
- 역회전: 정회전의 반대 순으로 구동



$A\bar{A}, B\bar{B}, \bar{A}A, \bar{B}B$ 구동 전류 방향(빨간색 화살표)

❖ 여기서 잠깐! MDRIVER를 이용해서…

- MDRIVER의 한 채널은 한 코일의 구동 전압을 ON/OFF 할 수 있으므로
- ➔ MDRIVER의 두 채널을 이용하면 스텝 모터를 바이폴라 방식으로 구동할 수 있다.
- ➔ MDRIVER 보드 하나로 두 개의 스텝 모터를 바이폴라 방식으로 구동할 수 있다.

※ 스테핑 모터의 COM을 GND에 연결해서 유니폴러 방식으로도 구동할 수 있다.

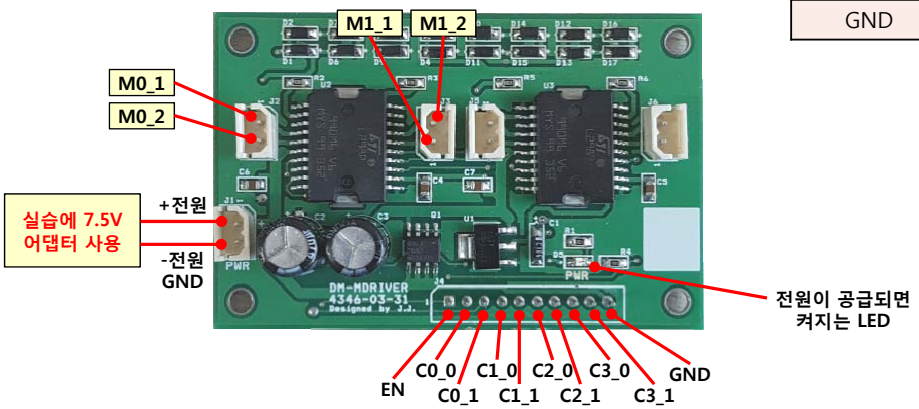
MDRIVER로 스테핑 모터를 구동하기 – 연결

❖ 연결 방법

- MDRIVER로 두 개의 스테핑 모터를 구동할 수 있다.
- 여기서는 M0, M1을 이용해서 스테핑 모터를 구동하는 예를 소개!

스테핑 모터	MDRIVER
A (BLACK)	M0_1
$\bar{A}$ (GREEN)	M0_2
B (RED)	M1_1
$\bar{B}$ (BLUE)	M1_2

Arduino	MDRIVER
23	EN (1)
24	C0_0 (2)
25	C0_1 (3)
26	C1_0 (4)
27	C1_1 (5)
GND	GND (10)



스테핑 모터 구동 기초

❖ P08.StepMotor01.Basic01

- CCW 방향으로 계속 회전

❖ P08.StepMotor01.Basic02

- CW 방향으로 계속 회전

❖ Q/A

- DELAY_TIME을 조절하면?
  - ◆ 회전 속도가 변경된다
- DELAY_TIME이 너무 짧으면?
  - ◆ 탈조가 발생해서 회전하지 못한다. (기구적으로 한 스텝이 완료되기 전에 다음 스텝으로 구동하기 때문)
- 모터가 회전하지 않는다면?
  - ◆ 배선 불량?
  - ◆ MDRIVER 불량?

P08.StepMotor01.Basic01

```
void setup() {
  pinMode(23, OUTPUT); // EN
  pinMode(24, OUTPUT); // A
  pinMode(25, OUTPUT); // ~A
  pinMode(26, OUTPUT); // B
  pinMode(27, OUTPUT); // ~B
  digitalWrite(23, HIGH);
}

#define DELAY_TIME 5

void loop() {
  // A상
  digitalWrite(24, HIGH);
  delay(DELAY_TIME);
  digitalWrite(24, LOW);

  // B상
  digitalWrite(26, HIGH);
  delay(DELAY_TIME);
  digitalWrite(26, LOW);

  // ~A상
  digitalWrite(25, HIGH);
  delay(DELAY_TIME);
  digitalWrite(25, LOW);

  // ~B상
  digitalWrite(27, HIGH);
  delay(DELAY_TIME);
  digitalWrite(27, LOW);
}
```

P08.StepMotor01.Basic02

```
void setup() {
  pinMode(23, OUTPUT); // EN
  pinMode(24, OUTPUT); // A
  pinMode(25, OUTPUT); // ~A
  pinMode(26, OUTPUT); // B
  pinMode(27, OUTPUT); // ~B
  digitalWrite(23, HIGH);
}

#define DELAY_TIME 5

void loop() {
 // ~B상
  digitalWrite(27, HIGH);
  delay(DELAY_TIME);
  digitalWrite(27, LOW);

  // ~A상
  digitalWrite(25, HIGH);
  delay(DELAY_TIME);
  digitalWrite(25, LOW);

  // B상
  digitalWrite(26, HIGH);
  delay(DELAY_TIME);
  digitalWrite(26, LOW);

  // A상
  digitalWrite(24, HIGH);
  delay(DELAY_TIME);
  digitalWrite(24, LOW);
}
```

Stepper Library의 활용 – 함수 소개

❖ Stepper Library는 Arduino에 기본적으로 내장된 Library

❖ Stepper Library의 함수

- Stepper(steps, pin1, pin2), Stepper(steps, pin1, pin2, pin3, pin4)
  - ◆ 초기화 함수로 스테핑 모터 구동 회로 구성에 따라 두 함수 중 선택
    - MDRIVER의 경우에는 네 개의 핀을 사용하므로 Stepper(steps, pin1, pin2, pin3, pin4) 이용
  - ◆ steps는 1회전당 스텝 수. 예를 들어 1.8도/스텝인 경우에는 200임

- setSpeed(rpm)
  - ◆ 회전 속도를 분당 회전수인 rpm으로 설정
    - 30으로 설정하면 초당 0.5회전이 된다.

- step(steps)
  - ◆ steps만큼 회전시킨다. 이때 회전속도는 setSpeed(rpm)로 설정한 속도!
  - ◆ 예를 들어 1.8도/스텝인 경우에 step(200)은 1바퀴 정회전, step(-200)은 1바퀴 역회전
  - ◆ 주의: 이 함수는 모터가 steps만큼 회전한 후에야 실행이 완료된다.
    - 다시 말해, 이 함수에 의해 모터를 구동하는 동안에는 다른 작업을 할 수 없다.

Stepper Library 소개

❖ 실습

- Stepper Library를 이용해서 스테핑 모터를 다음과 같이 회전 시키시오!
  - ◆ 30rpm으로 CCW 방향으로 1회전 후 1초 정지하고 60rpm으로 CW 방향으로 1회전 후 1초 정지하는 것을 무한 반복
- 배선은 앞 예제와 동일하게 함
- Stepper Library로 MDRIVER 보드를 사용하기 위해서는 MDRIVER의 EN 핀을 별도로 Enable 시켜주어야 한다.

❖ 주의

- Stepper Library는 2상 여자 방식을 사용하므로 1상 여자 방식에 비해 두 배의 전류를 사용한다 (2상 여자 방식은 한 스텝 당 두 상에 전류를 흘림)
- 이 실습은 6.0V 스테핑 모터에 7.5V를 사용함

➔ 전류가 많이 사용해 모터 드라이버와 어댑터 모두 열이 다소 많이 발생

➔ 실습처럼 모터에 부하가 없는 경우에는 문제가 없으나 부하가 있는 경우에는 문제가 있을 수 있으니 반드시 모터 전원을 제대로 사용하고 모터 드라이브 용량을 고려해서 사용할 것!

➔ 참고: 이 문제는 analogWrite의 PWM을 전압을 이용해서 해결할 수도 있음!

P08.StepMotor02.Stepper

```
#include <Stepper.h>

Stepper stepper(200, 24, 25, 26, 27);

void setup() {
  pinMode(23, OUTPUT); // EN
  digitalWrite(23, HIGH); // EN HIGH
}

void loop() {
  // 30 rpm CCW로 1회전 후 1초 정지
  stepper.setSpeed(30);
  stepper.step(200);
  delay(1000);

  // 60 rpm CW로 1회전 후 1초 정지
  stepper.setSpeed(60);
  stepper.step(-200);
  delay(1000);
}
```

MyStepper Library를 만들어보자! (1)

❖ MyStepper에 추가되는 기능 (Stepper Library 기능에 다음 기능 추가)

- stepTo(angle)
  - ◆ 기능: step(steps)는 현재 회전각에서 steps만큼 회전시키지만, stepTo(angle)는 초기 위치에 대해 angle에 해당하는 각으로 회전시킨다.
  - ◆ angle은 step 단위의 목표 회전각
    - 1.8도/스텝의 경우에 stepTo(200)하면 초기 위치에서 CCW로 1회전에 해당하는 각으로 회전, stepTo(-200)하면 초기 위치에서 CC로 1회전에 해당하는 각으로 회전
    - 초기 위치는 setOrg() 함수로 재설정 가능
- setOrg()
  - ◆ 현재 step을 stepTo()함수의 기준으로 설정

❖ MyStepper.h

MyStepper.h

```
#ifndef _MYSTEPPER_H_INCLUDE
#define _MYSTEPPER_H_INCLUDE

#include "Arduino.h"

class MyStepperClass {
private:
  uint8_t _enPin, _aPin, _anPin, _bPin, _bnPin;
  uint16_t _stepsPerRound;
  long _currentStepNum;
  long _targetStepNum;
  int _stepDelayMs;

  void stepMotor(void);
```

```
public:
  MyStepperClass(uint16_t stepsPerRound,
                 uint8_t enPin,
                 uint8_t aPin,
                 uint8_t anPin,
                 uint8_t bPin,
                 uint8_t bnPin);

  void setSpeed(long rpm);
  void step(long steps);
  void stepTo(long angle);
  void setOrg(void);
};

#endif
```


MyStepper Library를 만들어보자! (2)

❖ MyStepper.cpp

MyStepper.cpp

```
#include "MyStepper.h"

MyStepperClass::MyStepperClass(uint16_t
stepsPerRound,
                               uint8_t enPin,
                               uint8_t aPin,
                               uint8_t anPin,
                               uint8_t bPin,
                               uint8_t bnPin)
{
  _stepsPerRound = stepsPerRound;
  _enPin = enPin;
  _aPin = aPin;
  _anPin = anPin;
  _bPin = bPin;
  _bnPin = bnPin;

  pinMode(_enPin, OUTPUT);
  pinMode(_aPin, OUTPUT);
  pinMode(_anPin, OUTPUT);
  pinMode(_bPin, OUTPUT);
  pinMode(_bnPin, OUTPUT);
  digitalWrite(_enPin, HIGH);
  setSpeed(10);
  setOrg();
}
```

```
void MyStepperClass::stepMotor(void)
{
  while (_targetStepNum != _currentStepNum) {
    if (_targetStepNum > _currentStepNum)
      _currentStepNum++;
    else
      _currentStepNum--;

    switch (_currentStepNum&0x3) {
      case 0:
        digitalWrite(_aPin, HIGH);
        delayMicroseconds(_stepDelayMs);
        digitalWrite(_aPin, LOW);
        break;
      case 1:
        digitalWrite(_bPin, HIGH);
        delayMicroseconds(_stepDelayMs);
        digitalWrite(_bPin, LOW);
        break;
      case 2:
        digitalWrite(_anPin, HIGH);
        delayMicroseconds(_stepDelayMs);
        digitalWrite(_anPin, LOW);
        break;
      case 3:
        digitalWrite(_bnPin, HIGH);
        delayMicroseconds(_stepDelayMs);
        digitalWrite(_bnPin, LOW);
        break;
    }
  }
}
```

```
void MyStepperClass::setSpeed(long rpm)
{
  _stepDelayMs = 60000000L / (rpm *
_stepsPerRound);
}

void MyStepperClass::step(long steps)
{
  _targetStepNum = _currentStepNum + steps;
  stepMotor();
}

void MyStepperClass::stepTo(long angle)
{
  _targetStepNum = angle;
  stepMotor();
}

void MyStepperClass::setOrg(void)
{
  _targetStepNum = 0;
  _currentStepNum = 0;
}
```

MyStepper Library의 활용

❖ MyStepper Library의 활용 예제

- MyStepper Library의 구체적인 내용은 몰라도 사용법은 정확히 알고 사용합시다!
- 예제에 주석문이 적혀 있으므로 분석에 어려움이 없을 것입니다.

P08.StepMotor03.Library

```
#include "MyStepper.h"

// steps/round, EN, A, ~A, B, ~B 순
MyStepperClass stepper(200, 23, 24, 25, 26, 27);

void setup() {
}

void loop() {
  int i;

  // 30rpm으로 CCW방향으로 1바퀴 회전
  stepper.setSpeed(30);
  stepper.step(200);
  delay(1000);

  // 60rpm으로 CW방향으로 3바퀴 회전
  stepper.setSpeed(60);
  stepper.step(-600);
  delay(1000);

  // 현재 위치를 원점으로 해서
  // 50 rpm으로 45, -45, 90, -90, 135, -135, 180, -180 회전
  stepper.setOrg();
  stepper.setSpeed(50);
  for (i = 25; i<=100; i+= 25) {
    stepper.stepTo(i);
    delay(250);
    stepper.stepTo(-i);
    delay(250);
  }
  delay(1000);
}
```


팀 과제

❖ 팀 과제 1

- 버튼 모듈 두 개를 Arduino 40번과 41번 핀에 연결해서 스테핑 모터의 회전각을 다음과 같이 제어하시오
- 40번 버튼을 누르면 CW로 20 스텝씩 회전, 41번 버튼을 누르면 CCW로 20 스텝씩 회전
  - ◆ 회전 속도는 임의로 설정

❖ 팀 과제 2

- 시리얼로 1~9의 숫자를 전송하면 모터의 회전각을 다음과 같이 제어하시오
- 1은 -160, 2는 -120, 3은 -80, 4는 -40, 5는 0, 6은 40, 7은 80, 8은 120, 9는 160 스텝에 해당하는 각으로 회전
  - ◆ 회전 속도는 임의로 설정

❖ 팀 과제 3

- MDRIVER 1개로 스테핑 모터 두 개를 제어해서
- 첫 번째 스테핑 모터는 팀 과제 1처럼 제어하고 두 번째 스테핑 모터는 팀 과제 2처럼 제어하시오.

- 힌트: MyStepper 클래스 변수를 두 개 만들어 사용하면 된다!
  - ◆ MyStepper stepper0(200, 23, 24, 25, 26, 27), stepper1(200, 23, 28, 29, 30, 31);

RC 서보 모터의 구동

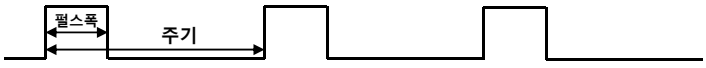
❖ RC 서보 모터를 구동할 수 있다.

- ❖ RC 서보 모터의 구동 원리를 설명할 수 있다.
- ❖ Servo Library를 이용해서 RC 서보 모터를 제어할 수 있다.

RC 서보 모터

❖ RC 서보 모터의 제어

- RC 자동차용 모터로 Pulse로 위치 신호 명령으로 구동하는 모터
  - ◆ 모터드라이버, 모터, 회전 센서(포텐셔미터), 제어 회로가 함께 내장되어 있음
  - ◆ 자동화 라인에서 사용되는 서보 모터와는 사용 방법에 차이가 크나 사용하기 편리하다는 장점 있음
- 외부 연결은 +전원선, -전원선, 제어선 3개 뿐
  - ◆ 선 색상은 제품마다 차이가 있을 수 있으므로 반드시 확인 후 사용!
- 제어선: 펄스에 의해 회전각 제어
  - ◆ 주기는 보통 10~20msec. 정도
  - ◆ 펄스 폭에 대한 일반적인 사양
    - Neutral이 보통 1.5msec.(0도에 해당)
    - 최소 0.7msec.(-90도에 해당), 최대 2.3msec.(90도에 해당) ← 제품마다 차이가 있음...
    - 펄스 폭에 비례해서 모터 회전각이 결정됨



실습용 RC 서보 모터

❖ RC 서보 모터

RC 서보 모터	의미
주황색 선	펄스 입력
빨간색 선	+전원
갈색 선	-전원(GND)

- 사양
  - Voltage: +4.8 ~ 6.0 V
  - Running current (at no load): 20mA@ 4.8V 25mA@ 6V
  - Stall current (at locked): 300mA@ 4.8V 350mA@ 6V
  - Rotation Degree: 180 Degree
  - Torque Size: 3.5kg · cm (4.8V); 4.2kg · cm (6.0V)
  - No load speed: 0.17 seconds / 60 degrees (4.8V); 0.12 sec / 60 degrees (6.0V)
  - Operating temperature: 0 ℃ ~ 60 ℃
  - Dead Set: 20us
  - Size: 40.2 X 20.2 X 43.2mm
  - Weight: 38±1g
- 참고 사이트
  - http://www.dfrobot.com/index.php?route=product/product&product_id=236#.ViNeI1jouUk

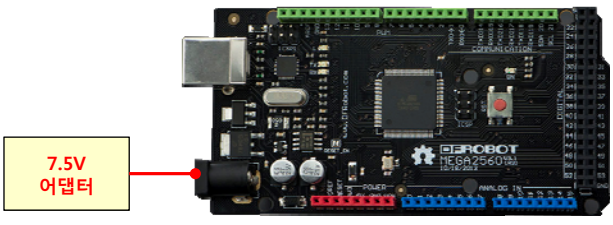
RC 서보 제어 기초 실습

❖ 전원은 4.8~6V를 사용해야 함

- Arduino의 5V도 사용이 가능하나 USB 전원을 사용하는 경우에는 USB 전원이 500mA 제한이 있으므로 작은 용량의 RC 서보 모터 1개 정도만 사용 가능하다.
- Arduino에 별도 어댑터를 사용하는 경우에도 Arduino 보드 내부의 5V를 만들어주는 칩이 500mA(순간 최대는 700mA)까지 전류를 공급할 수 있으므로 역시 작은 RC 서보 모터 1~2개만 사용 가능하다 (실습 중 절대 모터 회전 축에 힘을 가하지 말 것! → 과전류 방지)
- 용량이 큰 RC 모터를 사용할 때는 반드시 별도 전원으로 RC 서보 모터에 전원을 공급해야 한다!

❖ 방법

- 디지털 출력 핀으로 펄스를 지속적으로 내보냄!
  - 예) 10msec 주기로 1.5msec 펄스 폭을 내보냄!



RC 서보 모터	Arduino
주황색 선	22번
빨간색 선	5V
갈색 선	GND

P09.RCServo01.Basic

```
void setup() {
  pinMode(24, OUTPUT);
}

#define PULSE_T 1500L  // 700 ~ 2300

void loop() {
  // PULSE_T us 길이의 펄스를 10 ms마다 생성
  digitalWrite(24, HIGH);
  delayMicroseconds(PULSE_T);
  digitalWrite(24, LOW);
  delayMicroseconds(10000L-PULSE_T);
}
```

Servo Library의 사용

❖ Servo Library

- Arduino의 기본 Library로 RC 서보 모터 구동에 사용

❖ Servo Library 함수들

- attach(pin), attach(pin, min, max)
  - ◆ pin: 서보 모터의 제어선이 연결된 핀 번호
    - 예전에는 pin으로 9와 10만 가능한 경우도 있었으나 지금은 일반 디지털 핀도 사용 가능!
  - ◆ min, max: 마이크로초 단위의 펄스폭 최솟값 및 최댓값으로 기본값은 각각 544와 2400이다.
- write(angle)
  - ◆ angle: 0~180으로 도 단위임. attach의 min이 0도, max가 180도에 해당함
  - ◆ 이 함수는 즉시 리턴되며, 이후 Servo Class 내부의 인터럽트 루틴에서 지속적으로 펄스를 생성해서 RC 서보 모터로 보냄
- writeMicroseconds(uS)
  - ◆ uS: 마이크로초 단위의 서보 모터 제어 펄스 폭 설정
- read()
  - ◆ 리턴 값은 write에 의해 설정한 angle 값으로 0~180 범위
- attached()
  - ◆ attach()가 실행되어 있으면 true 리턴, detac가 실행되어 있으면 false 리턴
- detach()
  - ◆ attach에 의한 pin을 원래 상태로 되돌려 놓음

Servo Library의 예제 Sweep을 수정한 예제

❖ 문제

- 0~180도 사이를 반복적으로 회전

P09.RCServo02.Sweep2

```
#include <Servo.h>

Servo servo;

void setup()
{
  // 24번 핀으로 500~2500us가 0~180도에 해당하도록 함
  // 500, 2500은 RC 서보 모터 모델마다 다를 수 있음!
  servo.attach(24, 500, 2500);
}

void loop()
{
  int pos;

  for(pos = 0; pos <= 180; pos += 1) {
    servo.write(pos);
    delay(15);
  }

  for(pos = 180; pos>=0; pos-=1) {
    servo.write(pos);
    delay(15);
  }
}
```

팀 과제

❖ 팀 과제 1

- 스위치 모듈 두 개를 Arduino 40번과 41번 핀에 연결해서 24번 핀으로 제어하는 RC 서보 모터의 회전각을 다음과 같이 제어하시오
- 40번 스위치를 누르면 CW로 10도씩 회전, 41번 버튼을 누르면 CCW로 -10도씩 회전
  - ◆ 초기 위치는 90도 사용
  - ◆ 10~170도 범위만 사용할 것! 10도 미만이 되면 10도로 제한, 170 초과가 되면 170으로 제한

❖ 팀 과제 2

- 시리얼로 1~9의 숫자를 전송하면 모터의 회전각을 다음과 같이 제어하시오
- 1은 10도, 2는 30도, 3은 50도, 4는 70도, 5는 90도, 6은 110도, 7은 130도, 8은 150, 9는 170도로 회전

❖ 팀 과제 3

- RC 서보 모터 두 개를 제어해서
- 첫 번째 RC 서보 모터는 팀 과제 1처럼 제어하고 두 번째 RC 서보 모터는 팀 과제 2처럼 제어하시오.

- 힌트: Servo 클래스 변수를 두 개 만들어 사용하면 된다!
  - ◆ Servo servo0, servo1;

외부 인터럽트와 엔코더
엔코더를 이용한 DC 모터 구동

❖ 외부 인터럽트를 이용해서 엔코더의 회전각을 측정할 수 있다.

- ❖ 외부 인터럽트를 사용할 수 있다.
- ❖ 로타리 엔코더의 원리를 설명할 수 있다.
- ❖ 외부 인터럽트를 사용해서 로타리 엔코더의 펄스를 측정해서 모터의 회전각을 측정할 수 있다.
- ❖ 엔코더로부터 모터 회전각을 측정하여 모터 구동에 사용할 수 있다.

Arduino의 외부 인터럽트 구현

❖ Arduino의 외부 인터럽트(External Interrupt)

- 디지털 입력 상태에 변화가 발생되었을 때(= 인터럽트가 발생되었을 때) 실행 중인 부분을 잠시 보류하고 특정 함수를 실행시킬 수 있는 기능
  - ◆ 인터럽트가 발생되었을 때 실행되는 함수를 ISR(Interrupt Service Routine) 또는 인터럽트 함수라고 부릅니다. A 부분이 실행 중에 B 인터럽트가 발생하면 A 실행을 잠시 보류하고 B ISR이 실행된 후 다시 A를 실행합니다.
- attachInterrupt 함수로 구현

❖ 함수 attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)

- pin: 핀 번호로 Arduino Mega 2560의 경우에는 2, 3, 18, 19, 20, 21 중 하나만 가능
- ISR: 외부 인터럽트에 의해 실행될 함수 = 인터럽트 서비스 루틴 = 인터럽트 함수
- mode: 다음 중 하나로 다음 조건이 만족되는 경우 ISR 함수가 실행된다
  - ◆ LOW: pin의 입력이 LOW인 경우
  - ◆ RISING: pin의 입력이 LOW에서 HIGH로 되는 경우
  - ◆ FALLING: pin의 입력이 HIGH에서 LOW로 되는 경우
  - ◆ CHANGE: pin의 입력이 RISING이나 FALLING인 경우
- 참고: detachInterrupt(digitalPinToInterrupt(pin))
  - ◆ attachInterrupt에 의해 연결한 외부 인터럽트 기능을 끊어버리는 함수

외부 인터럽트 예제와 규칙

❖ 문제

- 19번 핀의 외부 인터럽트를 이용해서 스위치 모듈의 스위치를 누를 때마다 13번 LED를 토글
  - ◆ ON에서 HIGH를 출력하는 스위치 모듈 이용!

❖ 해설

- 프로그램을 보면 아무 것도 안 하는 loop 함수가 계속 실행되지만 19번 핀의 신호가 RISING 조건을 만족하면 blink 함수가 실행되어 LED가 토글된다.

❖ 지켜야 할 규칙

- ISR 함수의 형태는 void isrFunc() 형태
- ISR 함수에서 전역변수를 사용할 때는 전역변수를 선언할 때 반드시 앞에 volatile을 붙인다!

P10.Encoder01.ExtIsr01

```
volatile int state = LOW;

void setup() {
  pinMode(13, OUTPUT);
  pinMode(19, INPUT_PULLUP);
  // mode를 LOW, CHANGE, RISING, FALLING 중 하나로
  attachInterrupt(digitalPinToInterrupt(19), blink, RISING);
}

void blink() {
  state = !state;
  digitalWrite(13, state);
}

void loop() {
}
```

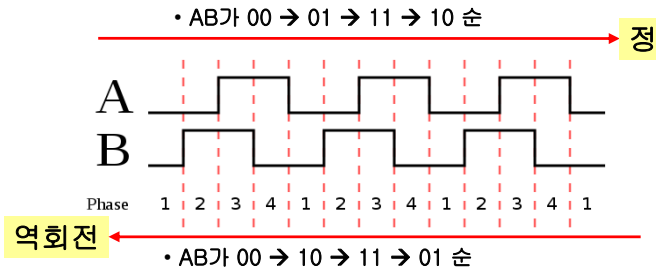
❖ 주의

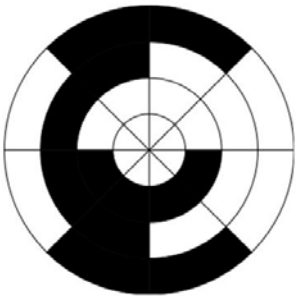
- 이 예제는 External Interrupt 사용 예를 보여주기 위해서 스위치를 사용했으나 실제 스위치에는 채터링이 있기 때문에 External Interrupt에 사용하지는 않는다.
- ISR 함수는 원래 실행되는 부분이 잠시 보류되고 실행되는 함수이므로 짧은 시간 내에 실행되도록 해야 한다. ISR 함수 내에서 delay나 print 함수를 사용하지 않는다!

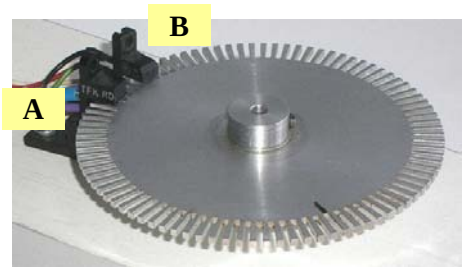
Rotary Encoder 원리

❖ 참고: Wikipedia(English)에서 Rotary Encoder 학습하기

- 제공하는 로타리 엔코더는 Incremental Type(좌측)이나 Absolute Type(우측)도 학습해보세요!
- 체배에 대해서 반드시 학습할 것!
- 출처: http://en.wikipedia.org/wiki/Rotary_encoder







출처: https://commons.wikimedia.org/wiki/File:Encoder.jpg?uselang=en, Rrudzik

Gray Coding				
Sector	Contact 1	Contact 2	Contact 3	Angle
0	off	off	off	0° to 45°
1	off	off	ON	45° to 90°
2	off	ON	ON	90° to 135°
3	off	ON	off	135° to 180°
4	ON	ON	off	180° to 225°
5	ON	ON	ON	225° to 270°
6	ON	off	ON	270° to 315°
7	ON	off	off	315° to 360°

출처: https://en.wikipedia.org/wiki/Rotary_encoder

Rotary Encoder 카운팅

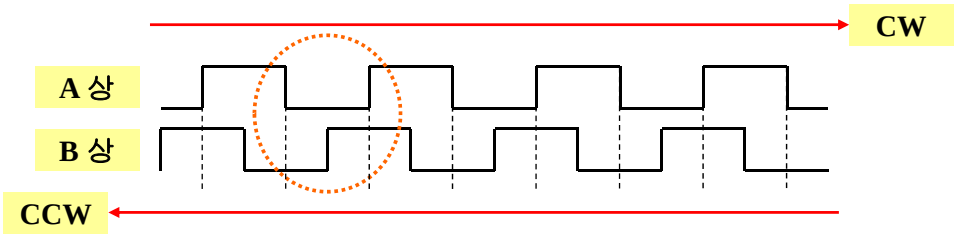
❖ Incremental Type의 경우!

❖ A의 Edge를 기준으로 한다면

- A Edge가 상승이고, B가 High면 CW 반 펄스 이동
- A Edge가 하강이고, B가 Low면 CW 반 펄스 이동
- A Edge가 상승이고, B가 Low면 CCW 반 펄스 이동
- A Edge가 하강이고, B가 High면 CCW 반 펄스 이동

※ 원래 엔코더 분해능의 두 배 가능 (2체배)

※ B상도 이용하면 원래 엔코더 분해능의 네 배 가능 (4체배)

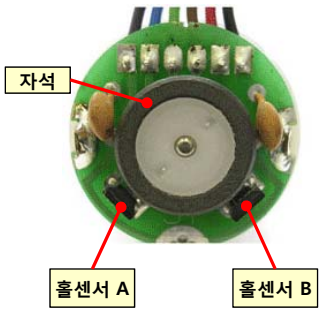


홀 센서를 이용한 엔코더

❖ 실습에 사용하는 DC 모터

- 엔코더가 부착된 DC 기어드 모터
  - ◆ 로터 축에 자석이 있고 이 자석의 회전을 두 개의 홀 센서로 검출해 회전각을 측정할 수 있다.
  - ◆ 로터 1회전 당 13 pulse
  - ◆ 기어가 30:1이고 2채배를 사용하면 구동 축은 13*30*2 = 780 ppr (pulse per round)

- 모터 커넥터 핀별 내용
  - ◆ 1. Black : - MOTOR
  - ◆ 2. Red : + MOTOR
  - ◆ 3. Brown : HALL SENSOR Vcc (5V)
  - ◆ 4. Green : HALL SENSOR GND
  - ◆ 5. Blue : HALL SENSOR B Vout (풀업 저항 반드시 필요)
  - ◆ 6. Purple : HALL SENSOR A Vout (풀업 저항 반드시 필요)



출처: 모터뱅크 홈페이지

로타리 엔코더 카운팅 예제 – 기초

❖ 엔코더로 회전각 검출

- CCW를 정방향으로 가정
- 연결

엔코더 모터	Arduino
BROWN	5V
GREEN	GND
BLUE	19
PURPLE	18

- 인터럽트 함수에서 사용하는 변수 규칙
  - ◆ 인터럽트 함수에서 값을 변경하는 변수값은 반드시 지역변수에 받아 사용
  - ◆ 이때 앞에는 noInterrupts() 함수를
  - ◆ 뒤에는 interrupts() 함수를 붙임
  - ◆ 자세한 이유는 상급레벨에 속하므로 생략

P10.Encoder02.Count01

```
volatile long count = 0;

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

  pinMode(18, INPUT_PULLUP);
  pinMode(19, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(18), countISR, CHANGE);
}

void countISR() {
  if (digitalRead(18) == HIGH) {
    if (digitalRead(19) == HIGH)
      count--;
    else
      count++;
  }
  else {
    if (digitalRead(19) == HIGH)
      count++;
    else
      count--;
  }
}
```

```
void loop() {
  long cval;

  // 규칙: 인터럽트 함수에서 변경하는 변수값은
  // 다음과 같은 형태로 지역변수에 받아서 사용한다.
  noInterrupts();
  cval = count;
  interrupts();

  Serial.println(cval);
  delay(100);
}
```

로타리 엔코더 카운팅 예제 – 응용 (1)

❖ 문제

- MDRIVER Library와 카운팅 함수를 이용해서 모터를 두 바퀴씩 다음과 같이 회전시킨다
  - ◆ PWM Duty 값 200으로 CCW 방향으로 회전시키다가 2바퀴가 회전되면 멈추고 1초 후 카운팅 값 출력
  - ◆ PWM Duty 값 200으로 CW 방향으로 회전시키다가 2바퀴가 회전되면 멈추고 1초 후 카운팅 값 출력
- 엔코더 카운팅 값을 리턴하는 getCount 함수와 카운팅 값을 임의로 설정하는 setCount 함수를 구현해서 사용한다

❖ 힌트

- 기어비가 30:1이고 2채배를 사용하면 한 바퀴는 30*13*2 = 780, 두 바퀴는 1560
- MDriver.drive 함수로 모터를 회전시키고 getCount 함수로 카운팅 값을 확인하여 2회전이 되면 모터를 정지시킨다.

❖ 연결

엔코더 모터	Arduino
BROWN	5V
GREEN	GND
BLUE	19
PURPLE	18

엔코더 모터	MDRIVER
RED	M0 1번 핀
BLACK	M0 2번 핀

Arduino	MDRIVER
22	EN (1)
9	C0_0 (2)
23	C0_1 (3)
GND	GND (10)

❖ 고찰해야 하는 것!

- 모터가 바로 멈추지 못하고 조금 더 지나쳐서 멈추는 현상 확인 (오버런 확인)

로타리 엔코더 카운팅 예제 – 응용 (2)

❖ 소스 파일

- MDriver.h 와 MDriver.cpp도 필요

P10.Encoder02.Count02

```
#include "MDriver.h"

// 파라미터는 EN, C00, C01, C10, C11, C20, C21, C30, C31 순
MDriverClass MDriver(22, 9, 23, 10, 24, 11, 25, 12, 26);

volatile long encCount = 0;

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

  pinMode(18, INPUT_PULLUP);
  pinMode(19, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(18), countISR, CHANGE);
}

void countISR() {
  if (digitalRead(18) == HIGH) {
    if (digitalRead(19) == HIGH)
      encCount--;
    else
      encCount++;
  }
  else {
    if (digitalRead(19) == HIGH)
      encCount++;
    else
      encCount--;
  }
}
```

```
long getEncCount(void) {
  long count;

  noInterrupts();
  count = encCount;
  interrupts();

  return count;
}

void setEncCount(long newCount) {
  noInterrupts();
  encCount = newCount;
  interrupts();
}

void loop() {
  // CCW로 2회전 후 멈추고 카운팅 값 출력
  MDriver.drive(MDRV_CH0, 200);
  while (getEncCount() <= 780*2) {
  }
  MDriver.drive(MDRV_CH0, 0);
  delay(1000);
  Serial.println(getEncCount());

  // CW로 원 위치로 온 후 멈추고 카운팅 값 출력
  MDriver.drive(MDRV_CH0, -200);
  while (getEncCount() >= 0) {
  }
  MDriver.drive(MDRV_CH0, 0);
  delay(1000);
  Serial.println(getEncCount());
}
```

팀 과제

❖ 팀 과제 1

- 다음과 같은 함수를 기능을 갖는 함수를 만들어 오버런을 최대한 줄인다
- 함수: moveTo(targetCount)
  - ◆ 처음에는 200의 pwm 값으로 회전하다가
  - ◆ targetCount 위치로부터 X 바퀴 안에 들어오면 A의 pwm 값으로 회전하고
  - ◆ targetCount 위치로부터 Y 바퀴 안에 들어오면 B의 pwm 값으로 회전하다가
  - ◆ targetCount 위치를 지나치면 바로 멈추게 해서 오버런을 줄인다.
  - ◆ moveTo 함수 내에서 사용하는 X, Y, A, B는 오버런을 줄이기 적절하게 정한다.
- 예제 동작으로 CCW 3회전, CW 3회전을 반복한다.

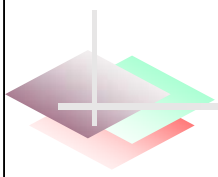
❖ 팀 과제 2

- 시작하면 모터를 CW 방향으로 회전시키다가 Arduino 40번에 연결한 스위치 모듈이 눌리면 멈추고 이후 CCW 방향으로 최대한 천천히 회전시키다가 스위치가 떨어지면 이 때의 엔코더 값을 0으로 설정한다. (스위치 모듈의 출력은 눌릴 때 HIGH이다.)
- 이후 시리얼로 1, 2, 3, 4, 5가 전송되면 앞서 설정한 엔코더 값을 기준으로 해서 전송된 숫자의 바퀴만큼 회전하고 멈추도록 한다.
  - ◆ 예) 2가 전송되면 스위치가 눌린 위치로부터 CCW 방향으로 2바퀴 위치까지 회전
- 모터의 회전에는 팀 과제 1에서 만든 moveTo 함수를 활용한다.

❖ 팀 과제 3

- 팀 과제 1에서 만든 moveTo 함수에 timeout을 도입하입해서 timout 내에 목적지까지 회전하지 못하면 false를 리턴하도록 하라.

DC 모터의 PI 속도 제어 소개

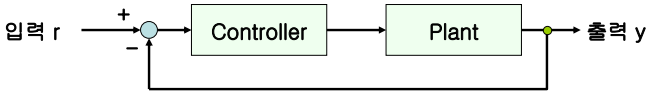


❖ PI 제어기를 이용한 DC 모터의 속도 제어를 설명할 수 있다!

피드백 제어와 제어기

❖ 피드백 제어란?

- 출력이 입력과 같아지도록 출력을 피드백해서 제어에 반영 (상대적 용어: 오픈루프 제어, 시퀀스 제어)
- 같은 말: 서보 제어, 폐루프 제어(closed loop control)



❖ 제어기(controller)란?

- 핵심 의미: 제어 로직을 의미함
- 광의의 의미: 제어 로직을 구현하는 H/W도 포함

❖ PID 제어란?

- 제어 로직 중 하나로 비례(proportional), 적분(interal), 미분(derivative)를 조합한 제어
- 제어 대상에 따라 PI, PD, PID 제어를 적용하기도 함

❖ 일반적으로 DC 모터의 속도 제어는

- 속도를 피드백해서 PI 제어를 하면 됨

DC 모터의 PI 속도 제어

❖ 엔코더를 이용한 속도 측정

- 단위 시간 동안 회전각의 변화로 속도 측정 가능
- 엔코더 분해능이 높을수록 속도를 더욱 세밀하게 측정할 수 있다
  - 실습에 사용하는 DC 모터의 엔코더는 로터를 기준으로 13 펄스/회전 밖에 안 되기 때문에 속도 분해능에 한계가 있다. 자동화 장치에 사용하는 모터에는 1회전 당 수 백 ~ 수 천 이상의 펄스를 갖는 엔코더를 사용한다!

❖ PI 제어기의 식: 자세한 내용은 제어공학에서 학습하세요!

- 연속계에서 $U(s) = (K_P + K_I)E(s)$

- 프로그램으로 구현하기 위한 이산 방정식으로 변환하면

$$u(k) = G_P \cdot \big(e(k) + e(k-1)\big) + G_I \cdot e(k-1) + u(k-1)$$

  - G_P : P Gain (비례 이득)
  - G_I : I Gain (적분 이득)
  - $u(k)$: 현재 시점에서 모터에 구동에 사용할 제어 입력 값
  - $u(k-1)$: 이전 시점에서 모터에 구동에 사용한 제어 입력 값
  - $e(k)$: 현재 시점에서의 속도 오차
  - $e(k-1)$: 이전 시점에서의 속도 오차

- 위 이산 방정식을 일정 시간 간격으로 계산하여 모터 구동에 사용하면 됨
  - 일정 시간의 결정은 사용 목적에 따라 다르며 실습 모터의 경우에는 10 밀리초 정도면 적당하다.

PI 속도 제어의 구현

❖ 제어기의 구현의 정석은 다음과 같으나...

- 제어기의 구현은 일정 시간 간격(= Sampling Time)으로 호출되는 ISR(Interrupt Service Routine)에서 처리하는 것이 일반적이나 Arduino의 경우 이러한 ISR을 구현하는 방법은 표준 라이브러리가 아닌 MsTimer2라고 하는 기부(contributed) 라이브러리로 구현해야 해서 여기에서는 소개하지 않음
  - ◆ 이와 같은 방법을 사용하면 PI 제어 중에도 다른 작업을 원활하게 할 수 있다는 장점이 있다!
- 여기에서는 다음과 같은 controlPI 함수를 만들어 일정 시간 간격으로 실행시키는 방법을 채택
  - ◆ 가장 간단한 형태로 구현한 PI 제어기 프로그램 소스!

```
void controlPI() {
  encCurr = getEncCount(); // 현재 회전각
  velocity = encCurr - encPrev; // 현재 속도 (SAMPLE_TIME 동안의 변화량)
  errCurr = targetCntPerSample - velocity; // 현재 속도 오차

  uOutCurr = GAIN_P * (errCurr - errPrev) + GAIN_I * errPrev + uOutPrev; // PI 제어값

  MDriver.drive(MDRV_CH0, uOutCurr); // PI 제어값 적용

  encPrev = encCurr; // 다음 시점을 위한 변수 처리
  errPrev = errCurr; // 다음 시점을 위한 변수 처리
  uOutPrev = uOutCurr; // 다음 시점을 위한 변수 처리
}
```


PI 속도 제어 예제 (1)

❖ 소스 코드

P11.PIControl01

```
#include "MDriver.h"

#define SAMPLE_TIME 10  // 밀리초 단위
#define GAIN_P   (8.0)
#define GAIN_I   (0.8)

// 파라미터는 EN, C00, C01, C10, C11, C20, C21, C30, C31 순
MDriverClass MDriver(22, 9, 23, 10, 24, 11, 25, 12, 26);

volatile long encCount = 0;

volatile double targetCntPerSample = 0, velocity;
volatile long encCurr = 0, encPrev = 0;
volatile double errCurr = 0, errPrev = 0;
volatile double uOutCurr = 0, uOutPrev = 0;


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

  pinMode(18, INPUT_PULLUP);
  pinMode(19, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(18), countISR, CHANGE);

  // SAMPLE_TIME 동안의 엔코더 목표 변화량 (-10~10 정도로 설정)
  targetCntPerSample = 3;
}
```

```
void countISR() {
  if (digitalRead(18) == HIGH) {
    if (digitalRead(19) == HIGH)
      encCount--;
    else
      encCount++;
  }
  else {
    if (digitalRead(19) == HIGH)
      encCount++;
    else
      encCount--;
  }
}

long getEncCount(void) {
  long count;

  noInterrupts();
  count = encCount;
  interrupts();

  return count;
}

void setEncCount(long newCount) {
  noInterrupts();
  encCount = newCount;
  interrupts();
}
```

PI 속도 제어 예제 (2)

❖ 소스 코드 계속

```
void controlPI() {
  encCurr = getEncCount(); // 현재 회전각
  velocity = encCurr - encPrev; // 현재 속도 (SAMPLE_TIME 동안의 변화량)
  errCurr = targetCntPerSample - velocity; // 현재 속도 오차

  uOutCurr = GAIN_P * (errCurr - errPrev) + GAIN_I * errPrev + uOutPrev; // PI 제어값

  MDriver.drive(MDRV_CH0, uOutCurr); // PI 제어값 적용

  encPrev = encCurr; // 다음 시점을 위한 변수 처리
  errPrev = errCurr; // 다음 시점을 위한 변수 처리
  uOutPrev = uOutCurr; // 다음 시점을 위한 변수 처리
}

void loop() {
  static unsigned long tCurr, tPrev = 0;

  tCurr = millis()/SAMPLE_TIME;

  // SAMPLE_TIME msec 마다 실행된다!
  if (tCurr != tPrev) {
    tPrev = tCurr;
    controlPI();
    // Serial 속도를 고려해서 제어 10번마다 1회 출력
    if (tCurr % 10 == 0) {
      Serial.print(uOutCurr);
      Serial.print(", ");
      Serial.println(velocity);
    }
  }
}
```


PI 속도 제어 예제 (3)

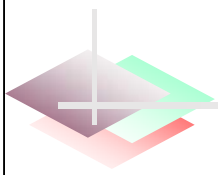
❖ 실습에서 해야 할 사항들

- 목표 속도의 변경 및 모니터링
  - setup() 함수의 끝 부분에 목표 속도 설정 부분이 있다. 이 값을 변경해서 테스트 할 것
    - // SAMPLE_TIME 동안의 엔코더 목표 변화량 (-10~10 정도로 설정)
    - targetCntPerSample = -3;
  - 시리얼 모니터로 모터 구동 PWM 값과 실제 속도값을 확인

- 외란에 대한 강인성 확인
  - PI 제어 중에 모터 축을 손으로 잡아 부하를 가해도 속도가 유지됨을 확인
    - 부하가 커져도 속도를 유지하기 위해 모터 구동 PWM 값이 커지는 것을 반드시 확인

- 게인의 변화에 따른 제어 특성 변화 확인
  - 소스 코드 앞 부분에 매크로 상수로 선언된 P게인과 I게인의 변화에 따른 제어 특성 확인
    - #define GAIN_P (8.0)
    - #define GAIN_I (0.8)
  - 제어 게인이 작으면 제어가 잘 안 되고, 제어 게인이 크면 불안정해지는 것을 확인해 본다

수고하셨습니다!



지금까지 배운 내용을 토대로
간단하게나마 학기말 작품을
제안하고 구현해 보세요!