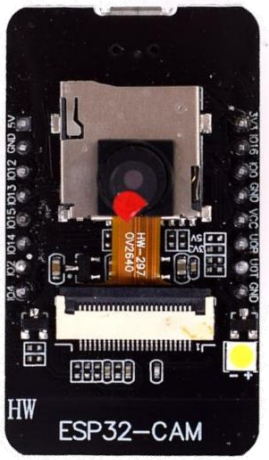
사물지능 과정

[Chapter 30]
ESP32

WeMos ESP32 Uno D1 R32

개요

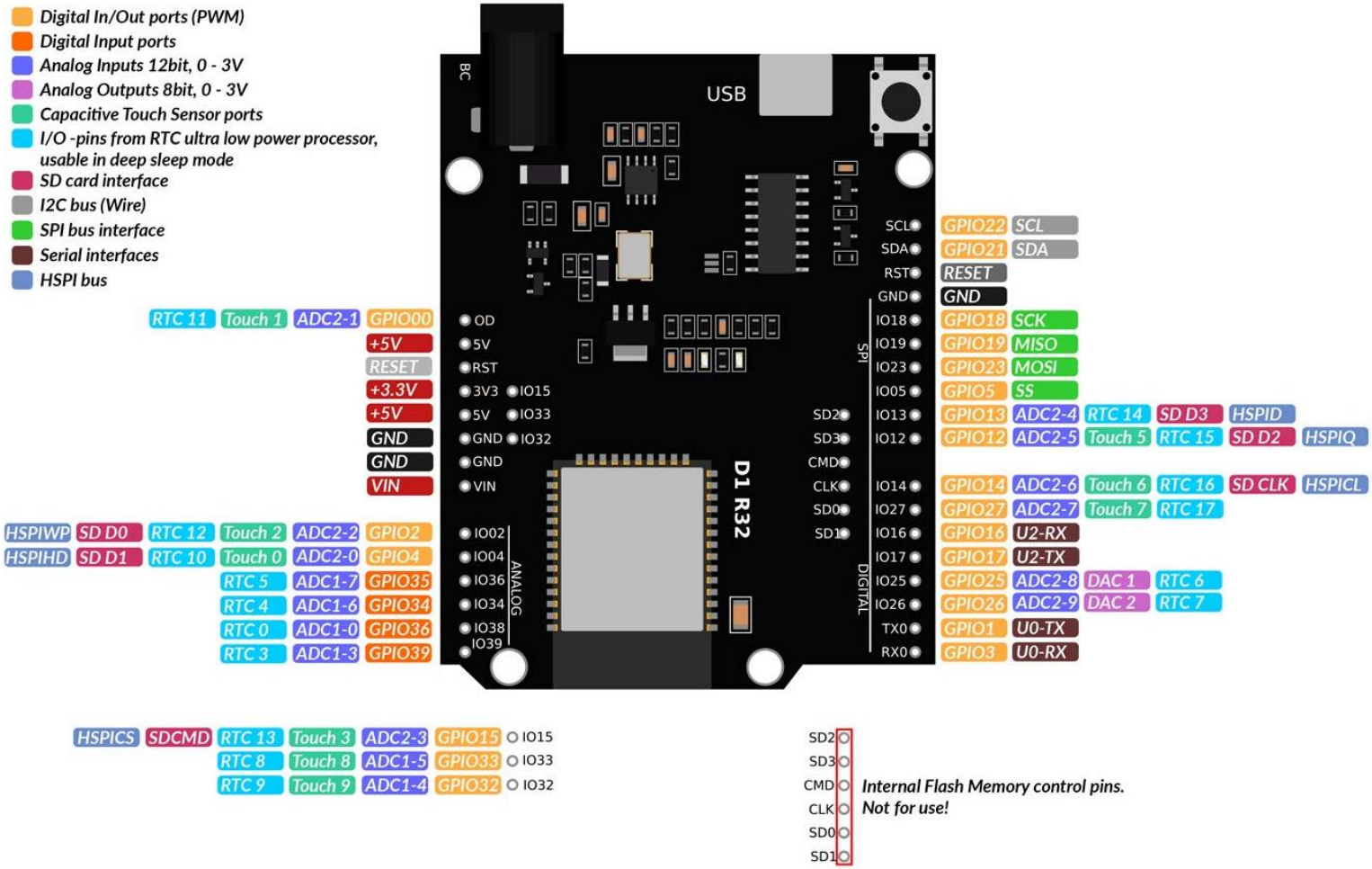
- SPI, I2C, CAN, UART, WiFi, Bluetooth 통신 지원
- CH340 드라이버 사용
- 듀얼 프로세스 지원 – ESP8266의 단점 보완
- 다양한 형태의 보드 지원 (OLED, CAM 등)
- 관련 사이트
  - https://www.espressif.com/en/products/socs/esp32
  - https://www.esp32.com/



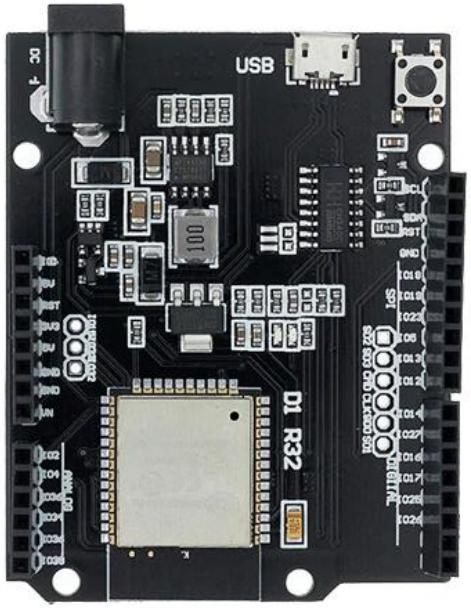
핀 구조

- 입력 전용핀 : GPIO34에서 GPIO39까지 6개의 핀
- 플래시 메모리 전용핀 : GPIO6 (SCK/CLK) 또는 GPIO6-GPIO11 (임으로 사용 불가)
- 터치센서용 핀 : GPIO 0, 2, 4, 12, 13, 14, 15, 27, 32, 33
- ADC 핀 (아날로그 입력) : 18개의 채널 지원
- DAC 핀 (아날로그 출력) : 2개의 채널 지원 (GPIO25, 26)
- PWM 핀 : GPIO25, 26 (GPIO34-39를 제외하고 사용 가능)
- I2C 통신 핀 : GPIO21(SDA), GPIO22(SCL)
- SPI 통신 핀 : GPIO 13, 23 (MOSI), GPIO12, 19 (MISO), GPIO14, 18 (CLK), GPIO5, 15 (CS)

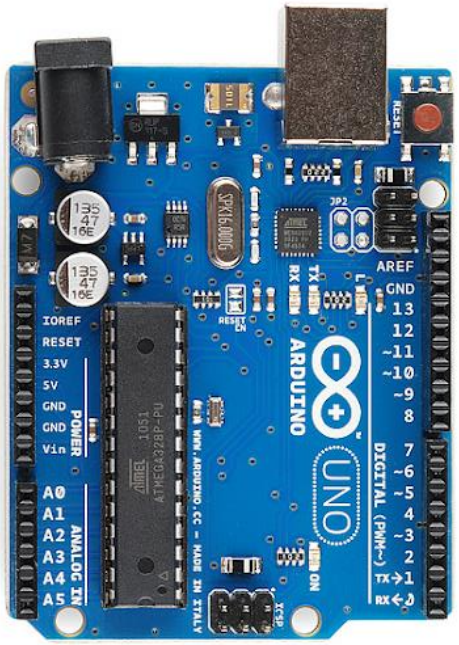
보드 구조


Digital In/Out ports (PWM)
Digital Input ports
Analog Inputs 12bit, 0 - 3V
Analog Outputs 8bit, 0 - 3V
Capacitive Touch Sensor ports
I/O -pins from RTC ultra low power processor, usable in deep sleep mode
SD card interface
I2C bus (Wire)
SPI bus interface
Serial interfaces
HSPI bus
USB
D1 R32
SPI
DIGITAL
ANALOG
RTC 11 Touch 1 ADC2-1 GPIO00
+5V
RESET
+3.3V
+5V
GND
GND
VIN
HSPIWP SD D0 RTC 12 Touch 2 ADC2-2 GPIO2
HSPIHD SD D1 RTC 10 Touch 0 ADC2-0 GPIO4
RTC 5 ADC1-7 GPIO35
RTC 4 ADC1-6 GPIO34
RTC 0 ADC1-0 GPIO36
RTC 3 ADC1-3 GPIO39
HSPICS SDCMD RTC 13 Touch 3 ADC2-3 GPIO15
RTC 8 Touch 8 ADC1-5 GPIO33
RTC 9 Touch 9 ADC1-4 GPIO32
GPIO22 SCL
GPIO21 SDA
RESET
GND
GPIO18 SCK
GPIO19 MISO
GPIO23 MOSI
GPIO5 SS
GPIO13 ADC2-4 RTC 14 SD D3 HSPID
GPIO12 ADC2-5 Touch 5 RTC 15 SD D2 HSPIQ
GPIO14 ADC2-6 Touch 6 RTC 16 SD CLK HSPICL
GPIO27 ADC2-7 Touch 7 RTC 17
GPIO16 U2-RX
GPIO17 U2-TX
GPIO25 ADC2-8 DAC 1 RTC 6
GPIO26 ADC2-9 DAC 2 RTC 7
GPIO1 U0-TX
GPIO3 U0-RX
Internal Flash Memory control pins.
Not for use!

핀 구조

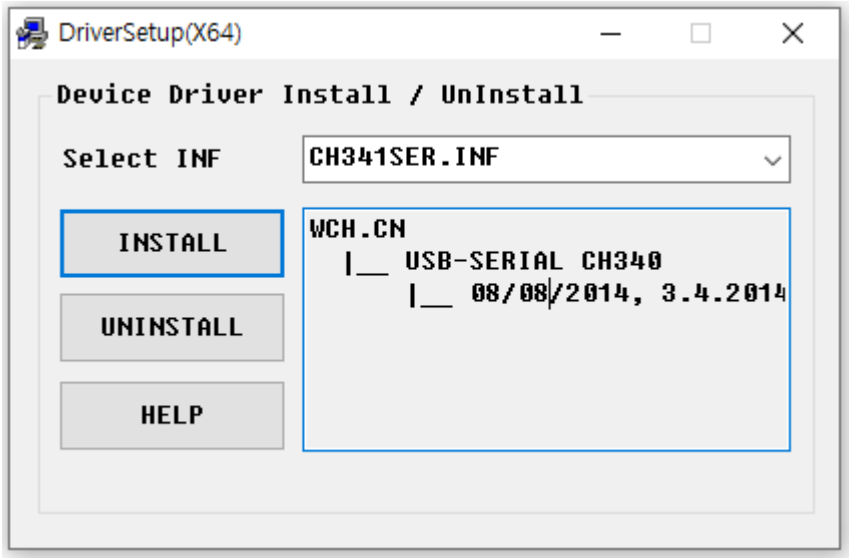


WeMos D1 R32	Arduino Uno
IO26	D2
IO25	D3
IO17	D4
IO16	D5
IO27	D6
IO14	D7
IO12	D8
IO13	D9
IO5	D10
IO23	D11
IO9	D12
IO8	D13
IO2	A0
IO4	A1
IO35	A2
IO34	A3
IO36	A4
IO39	A5



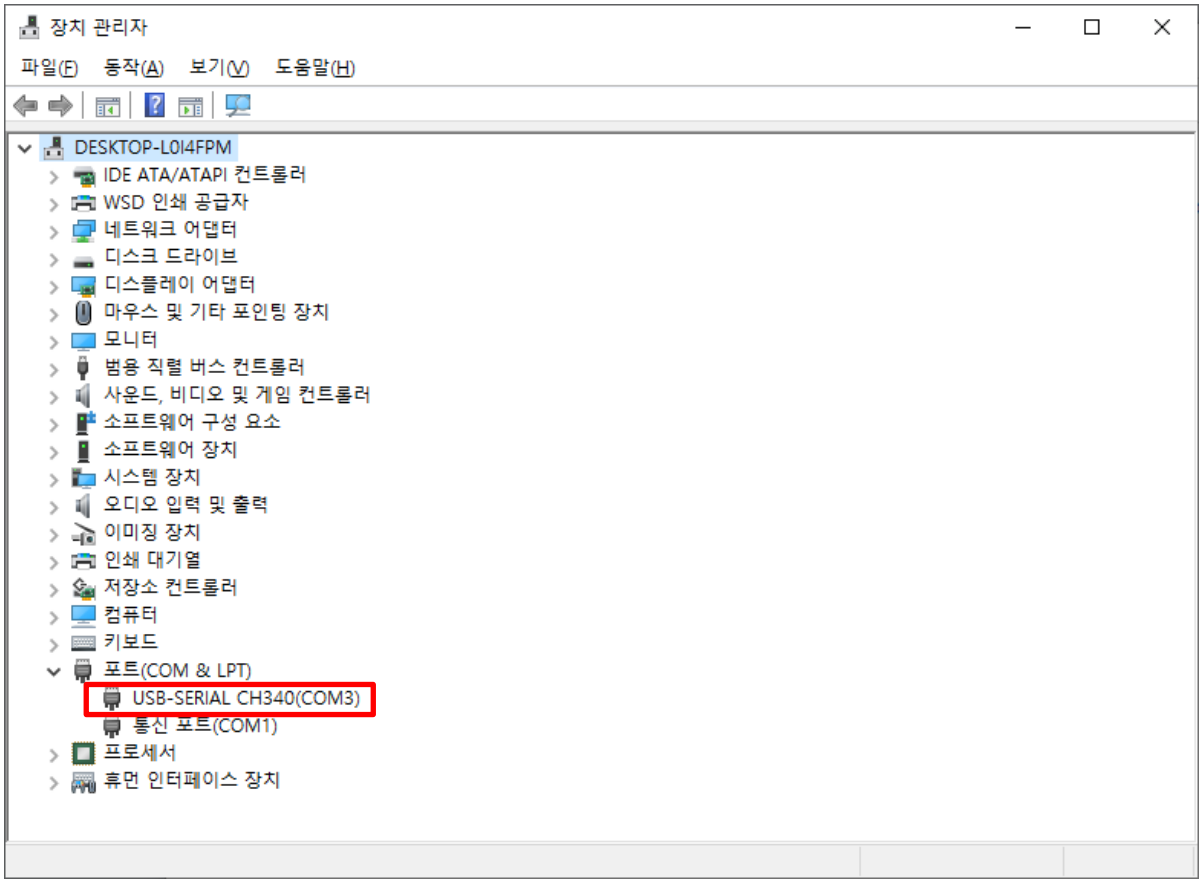
사용방법

- CH340 드라이버 설치 (http://www.wch.cn/download/CH341SER_ZIP.html)



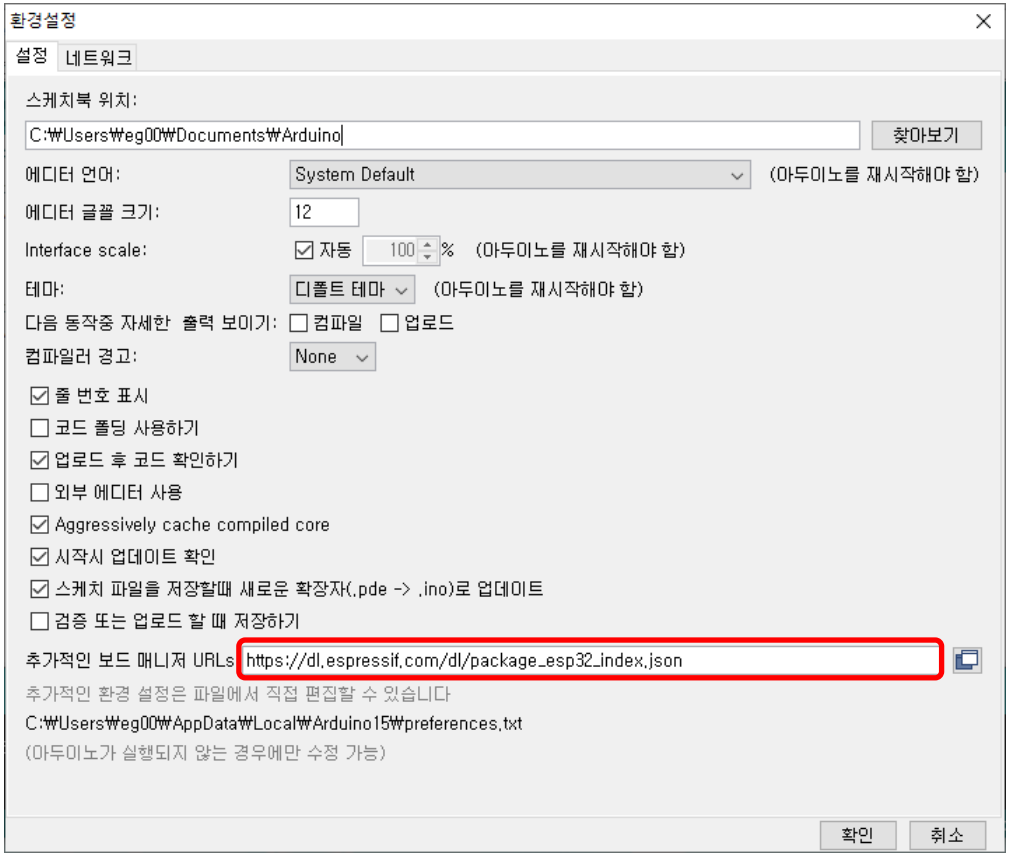
사용방법

- 제어판 > 장치관리자에서 CH340 드라이버 연동 확인



사용방법

- 파일 > 환경설정 > 추가적인 보드 메니저 URLs에 링크 입력
- https://dl.espressif.com/dl/package_esp32_index.json



사용방법

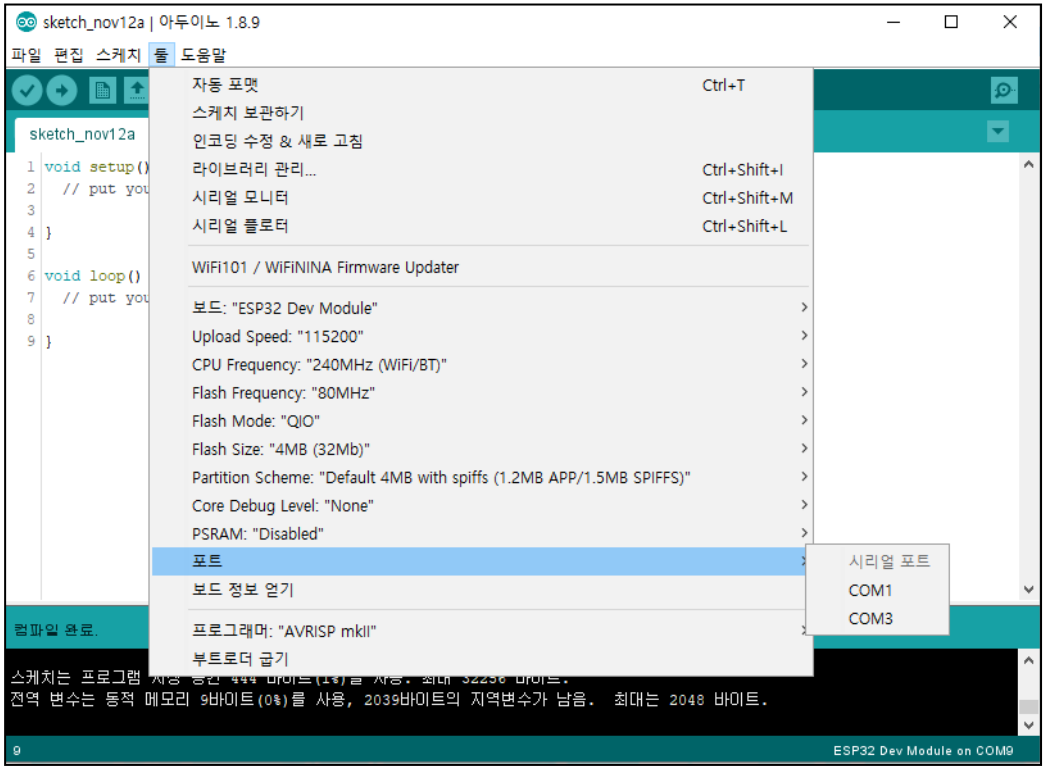
- 툴 > 보드 >보드 메니저에서 ESP32를 입력해서 해당 보드를 설치

사용방법

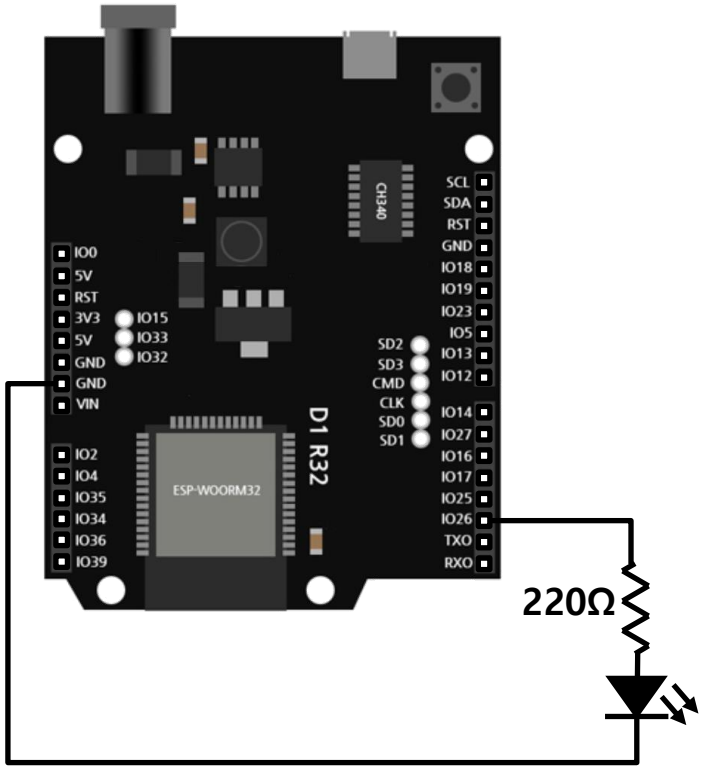
- 툴 > 보드 메뉴에 해당 보드가 설치되었지 확인
- ESP32 Dev Module 보드를 선택

사용방법

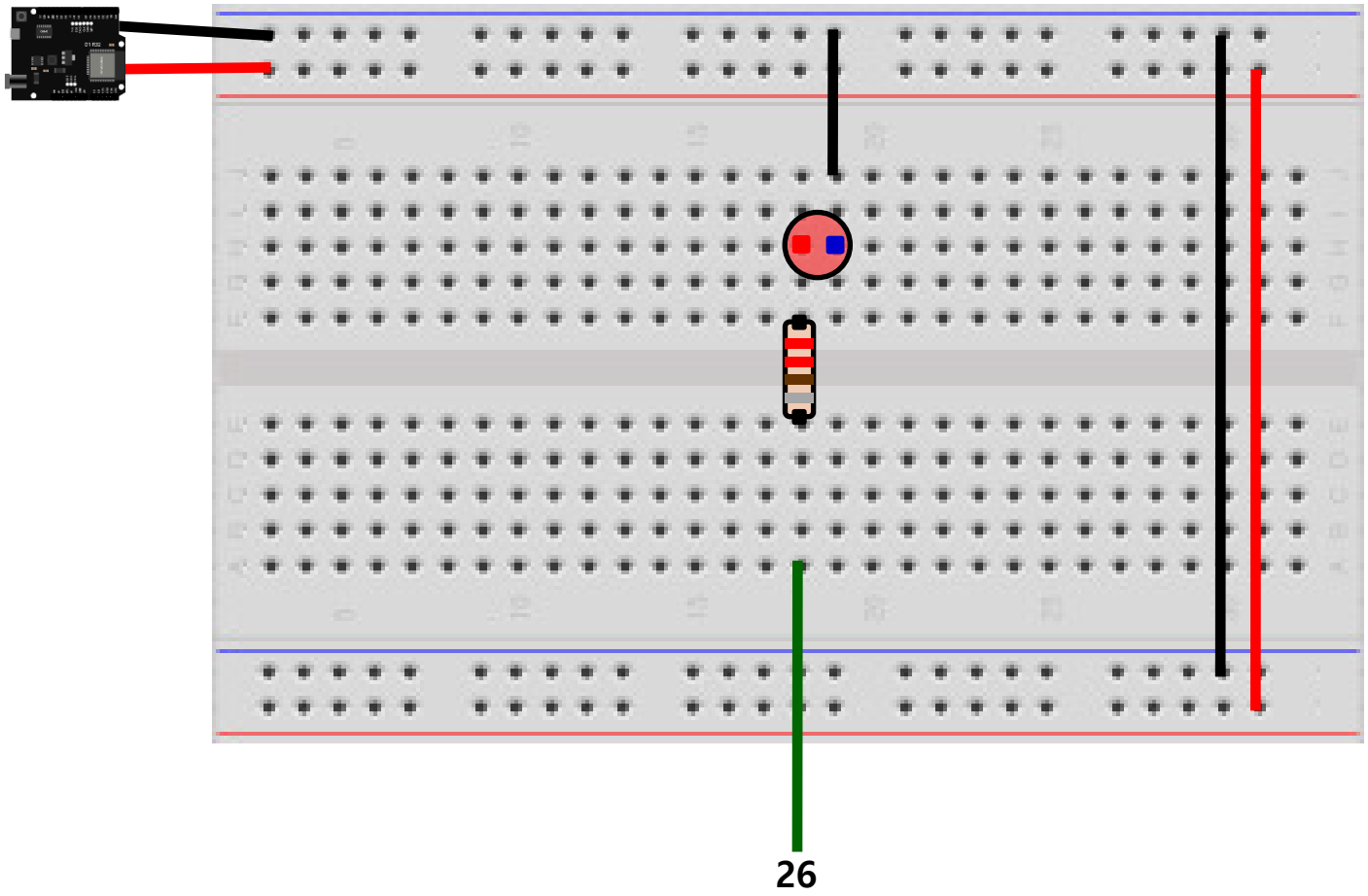
- Upload Speed : 115200
- 연결한 포트 선택



디지털 출력 - LED 제어


IO0
5V
RST
3V3
5V
GND
GND
VIN
IO15
IO33
IO32
IO2
IO4
IO35
IO34
IO36
IO39
ESP-WOORM32
D1 R32
CH340
SCL
SDA
RST
GND
IO18
IO19
IO23
IO5
IO13
IO12
IO14
IO27
IO16
IO17
IO25
IO26
TXO
RXO
SD2
SD3
CMD
CLK
SD0
SD1
220Ω

디지털 출력 - LED 제어



디지털 출력 - LED 제어

```
// 보드 상의 LED가 연결된 핀 (GPIO26)
int ledPin = 26;

void setup() {
  pinMode(ledPin, OUTPUT);
}

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

아날로그 출력 (PWM) 채널 설정

- ESP32에는 16개의 독립적인 PWM채널 지원
- 사용하고자 하는 채널의 PWM 주파수 설정 필요 (1초에 몇 번 ON/OFF를 반복할 것인지)
  - 통상 LED밝기 조절의 경우 5000Hz 정도면 충분 (아두이노 우노의 경우 490 Hz 또는 980 Hz)
- PWM 해상도 설정 : 1bit에서 16 비트까지 임의로 지정 (아누이노 우노는 8 bit 해상도 이었기 때문에 0~255까지의 숫자로 세기를 지정)

- ESP32에서 채널 설정
  - ch : PWM channel (0~15)
  - freq : PWM 주파수
  - resolution : PWM 해상도

```
ledcSetup(ch, freq, resolution);
```

PWM 신호 출력

- PWM채널을 설정했으면 setup함수에서 PWM신호를 출력할 ESP32 핀번호를 지정
  - gpio : GPIO 핀 번호
  - ch : PWM channel

```
ledcAttachPin(gpio, ch);
```

- 실제 PWM신호를 출력하고자 할 때에 ledcWrite()를 사용
  - ch : PWM channel
  - duty : Duty cycle

```
ledcWrite(ch, duty);
```

아날로그 출력 (PWM) – LED 밝기 제어

```
int ledPin = 26;

void setup() {
   // PWM 제어 설정
   ledcSetup(0, 5000, 8);

   // PWM 핀번호 설정 (핀번호, PWM channel)
   ledcAttachPin(ledPin, 0);
}

void loop() {
  for (int i=0; i<255; i++) {
    // PWM 출력
    ledcWrite(0, i);
    delay(15);
  }
  delay(1000);
}
```

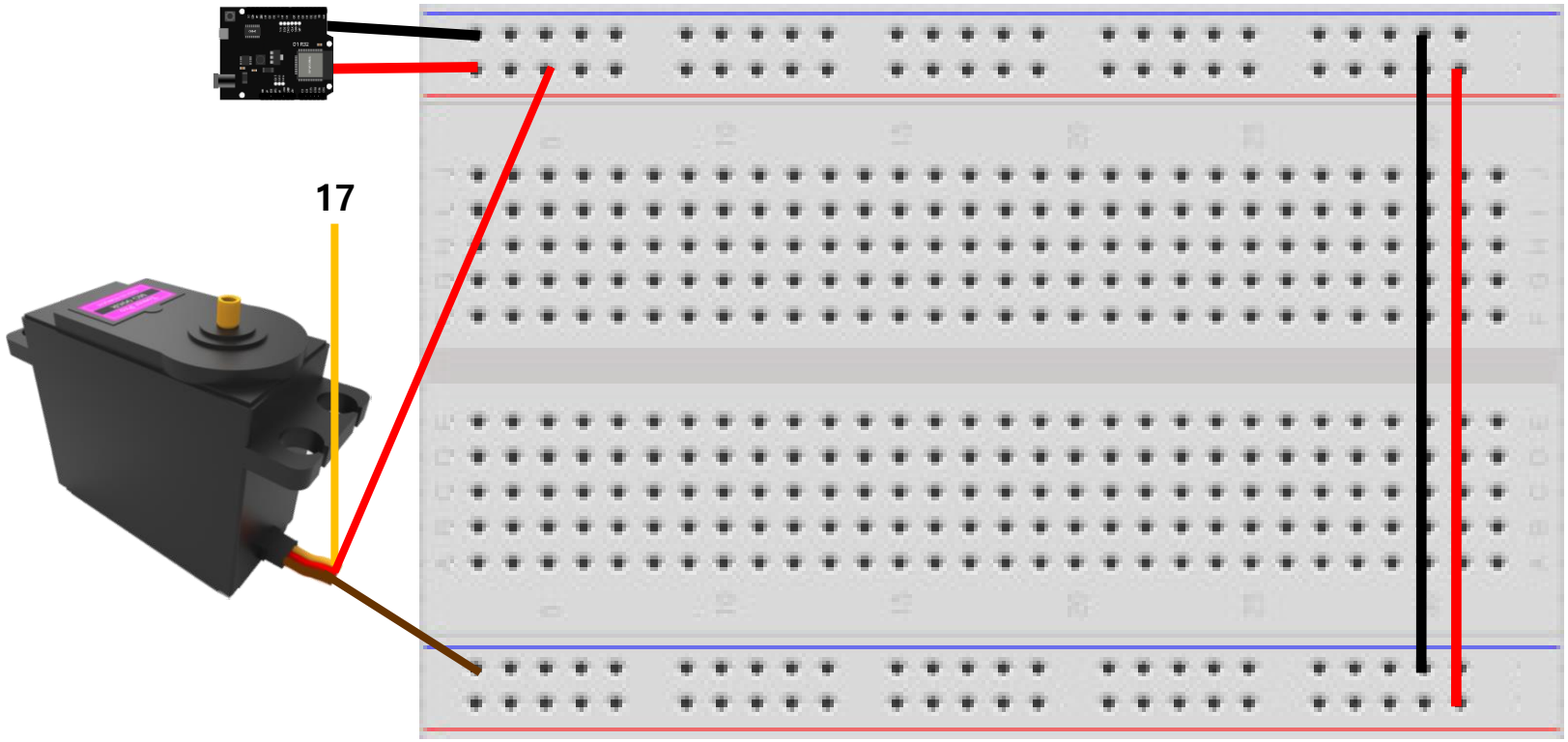
아날로그 입력 - 조도센서 값 읽기

```
int pirPin = 35;

void setup() {
  Serial.begin(115200);
}

void loop() {
  int value = analogRead(pirPin);

  // 0-4095까지 범위의 값으로 출력 (12비트)
  Serial.println(value);
  delay(1000);
}
```

서보모터 제어 (MG945)

- 최대토크 :10kg/cm (4.8v);12kg/cm/(6v)
- 동작 속도 : 0.25sec / 60도(4.8V)
- 동작 전압 : 4.8 ~ 6.6V
- 동작 온도 : 0 ~ 55도
- 크기 : 40.7 x 19.7 x 42.9mm
- 무게 : 55g
- VCC (적색), GND (갈색), 신호 (오렌지)

서보모터 제어 (MG945)



서보모터 제어 (MG945)

- ESP32용 서보모터 라이브러리 :
  https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=roboholic84&logNo=221838773803

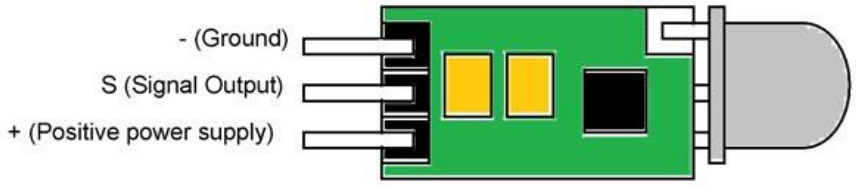
- 스케치 > 라이브러리 포함하기 > .zip 라이브러리 추가에서 다운 받은 zip파일을 등록

서보모터 제어 (MG945)

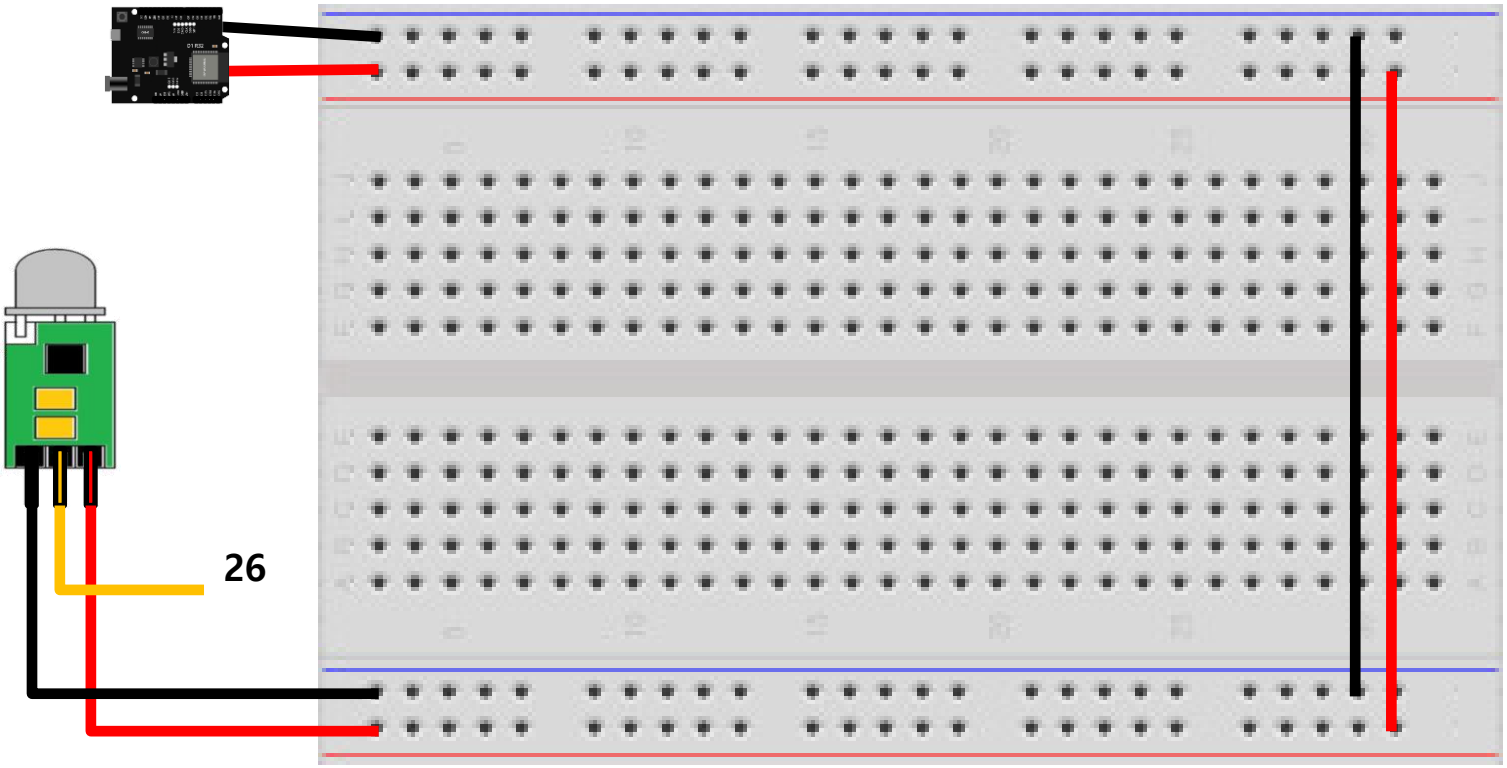
```
#include <ESP32_Servo.h>

Servo myServo;
int servoPin = 17;

void setup(void){
  myServo.attach(servoPin);
  myServo.write(0);
  delay(1000);
  myServo.write(180);
}

void loop(void){
}
```

디지털 입력 - 적외선 센서 (HC-SR505)

- 작동 전압 범위 : DC4.5-20V
- 출력 레벨 : 높이 3.3V/낮음 0V
- 지연 시간 : 기본 8 초 + -30% (몇 분의 10 분 범위 사용자 정의 가능)
- 유도 각도 : <100 도 콘 각도
- 유도 거리 : 3 미터

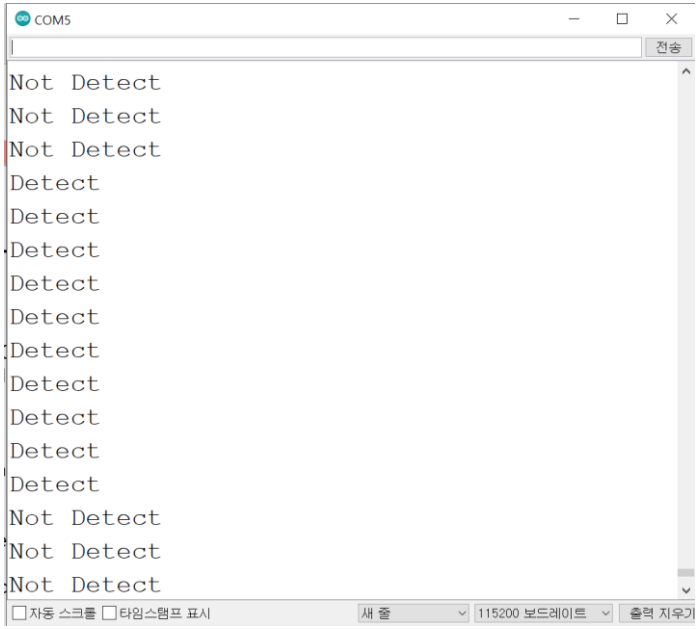


디지털 입력 - 적외선 센서 (HC-SR505)



디지털 입력 - 적외선 센서 (HC-SR505)

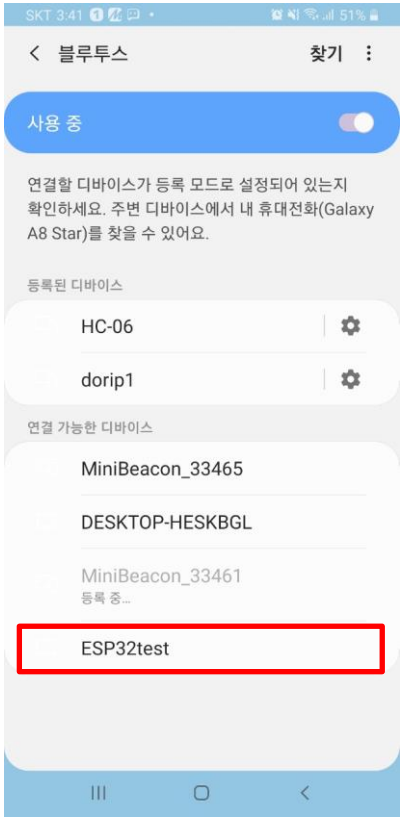
```
int pirPin = 26;

void setup() {
   Serial.begin(115200);
   pinMode(pirPin, INPUT);
}

void loop() {
   int val = digitalRead(pirPin);

   if (val == HIGH)
      Serial.println("Detect");
   else
      Serial.println("Not Detect");

   delay(1000);
}
```

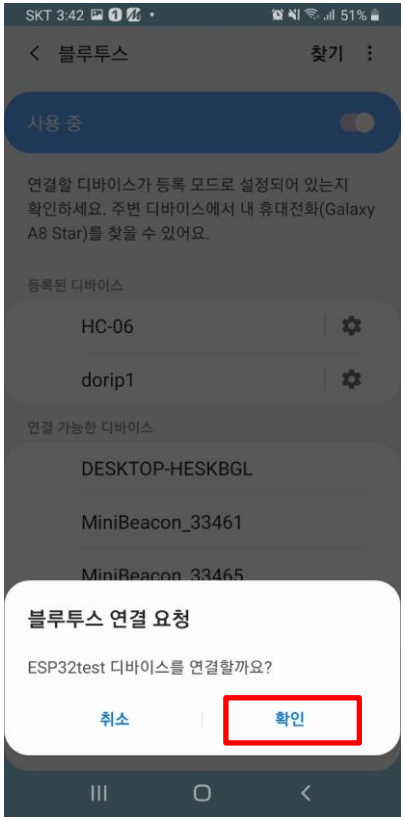


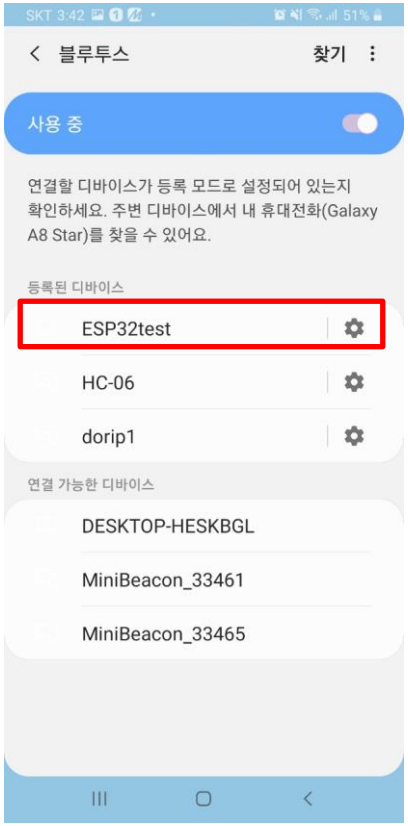
8초 정도의 지연시간

블루투스 통신

- 스마트폰에서 블루투스 장치 검색 및 페어링

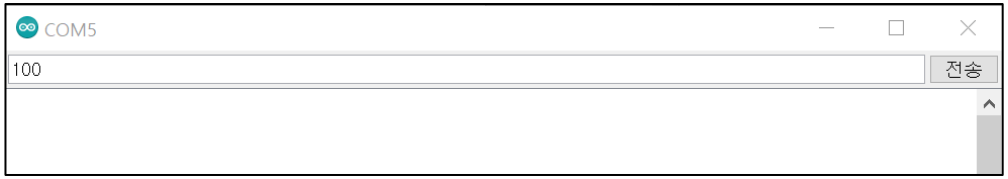






블루투스 통신

- 스마트폰 어플에서 블루투스 장치 연결
- 시리얼 모니터에서 값을 전송하면 스마트폰 어플에 표시되는지 확인



블루투스 통신

- 스마트폰 어플에서 값을 전송하면 시리얼 모니터에 표시되는지 확인

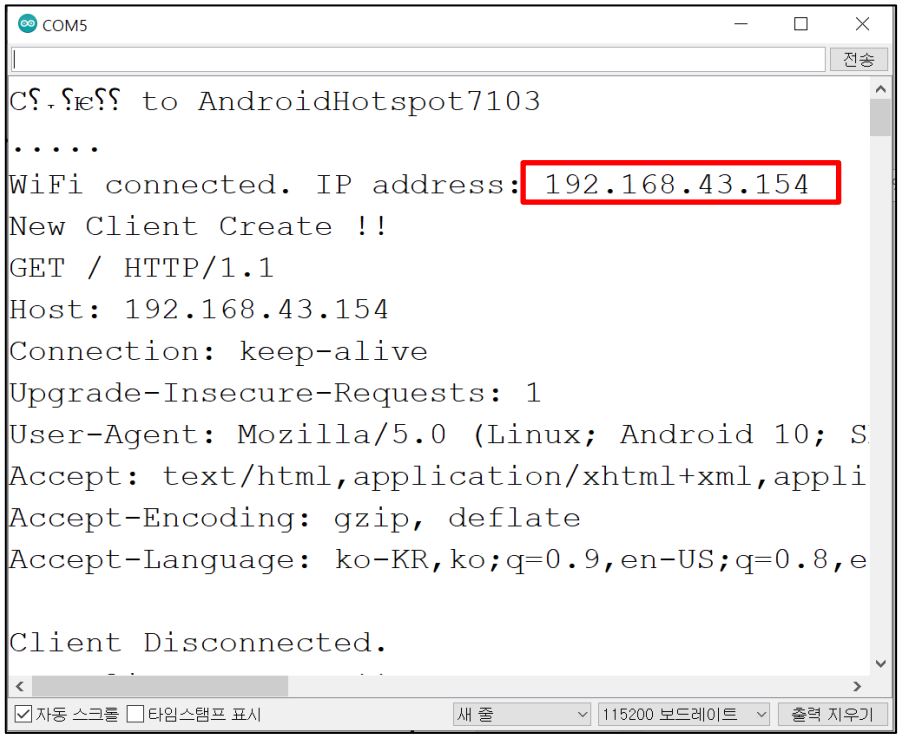
블루투스 통신

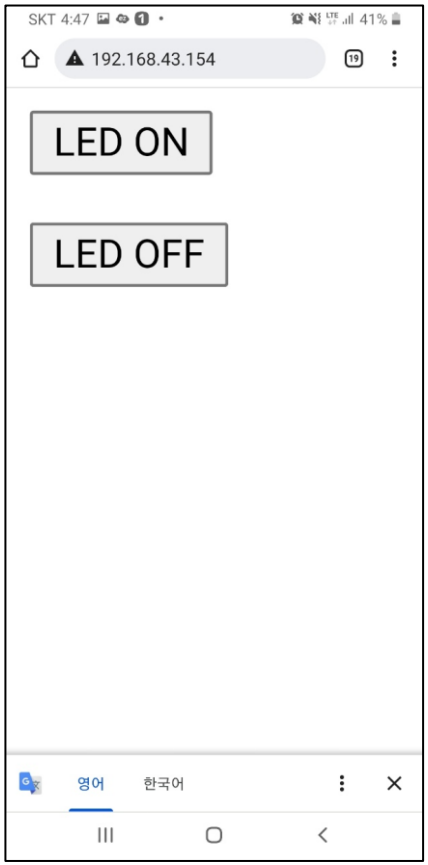
```
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);

  // 블루투스 장치 이름을 ESP32test로 설정하고 시작
  SerialBT.begin("ESP32test");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }

  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }

  delay(20);
}
```

블루투스 통신 – ESP32 간 통신

WiFi 통신 – 서버의 LED 제어하기

- 시리얼 모니터에 출력되는 IP로 동일 네트워크 내의 기기에서 접속
- 버튼을 클릭하여 LED 제어가 되는지 확인





WiFi 통신 – 서버의 LED 제어하기

```
#include <WiFi.h>

int ledPin = 25;

// 공유기 아이디와 패스워드 설정
const char* ssid = "AndroidHotspot7103";
const char* pass = "123456789";

WiFiServer server(80);

void setup(){
    Serial.begin(115200);
    pinMode(ledPin, OUTPUT);

    delay(10);

    // WiFi network에 접속
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, pass);
```

WiFi 통신 – 서버의 LED 제어하기

```
    // WiFi 네트워크에 접속되지 않았다면
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    // 접속 IP 출력
    Serial.println("");
    Serial.print("WiFi connected. IP address: ");
    Serial.println(WiFi.localIP());

    // 서버 시작
    server.begin();
}

int value = 0;

void loop(){
  // 새로운 클라이언트 생성
  WiFiClient client = server.available();
```

WiFi 통신 – 서버의 LED 제어하기

```
// 클라이언트가 생성되었다면
if (client) {
  Serial.println("New Client Create !!");
  String currentLine = "";

  // 클라이언트가 접속되었다면
  while (client.connected()) {
    // 클라이언트에서 문자를 받으면
    if (client.available()) {
      // 문자를 byte 단위로 읽고, 문자로 저장
      char c = client.read();
      Serial.write(c);

      // 문자가 개행문자이면
      if (c == '₩n') {
        // 현재 라인에 아무것도 없으면, 행 당 2개라인을 가짐
        // 이는 client HTTP 응답의 마지막이라는 것이므로 응답을 전송할 수 있다:
        if (currentLine.length() == 0) {
          // HTTP 헤더는 항상 응답코드 (HTTP/1.1 200 OK)로 시작하고
          // content-type이 나오고 빈칸이 나온다:
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:text/html");
          client.println("Content-type:text/html");
          client.println("<meta name='viewport' content='width=device-width, initial-scale=2.0'>");
          client.println();
```

WiFi 통신 – 서버의 LED 제어하기

```
            // HTTP 응답은 다음과 같은 헤더를 가진다:
            client.print("<font size=16><a href=₩"/H₩"><button>LED
ON</button></a><p></font>");
            client.print("<font size=16><a href=₩"/L₩"><button>LED OFF</button></a></font>");

            // HTTP 응답은 빈칸으로 끝난다.
            client.println();
            // while loop를 종료
            break;
          // 새로운 라인을 가지면, 현재라인 지움:
          } else {
            currentLine = "";
          }
        // carriage return 문자라면.
        } else if (c != '₩r') {
          // 현재라인 뒤에 붙임
          currentLine += c;
        }
```

WiFi 통신 – 서버의 LED 제어하기

```
        // 클라이언트 응답이 "GET /H" 또는 "GET /L" 였는지 확인하여 LED 제어
        if (currentLine.endsWith("GET /H")) {
          digitalWrite(ledPin, HIGH);
        }
        if (currentLine.endsWith("GET /L")) {
          digitalWrite(ledPin, LOW);
        }
      }
    }
    // 연결 종료:
    client.stop();
    Serial.println("Client Disconnected.");
  }
}
```