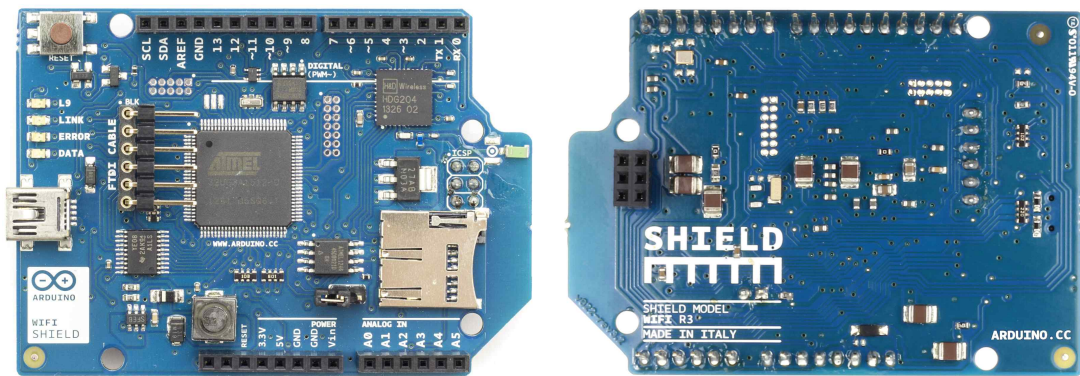


Chapter 280. 와이파이

1. 와이파이 쉴드

와이파이는 무선 근거리 네트워크(WLAN, Wireless Local Area Network)를 구성하기 위해 필요한 일련의 기술에 관한 표준으로, 인터넷에 데이터를 전달하는 기능을 하는 액세스 포인트(AP, Access Point)와 무선기기 사이의 통신을 가리키는 용어로 흔히 접하게 된다. 와이파이는 블루투스나 지그비와 마찬가지로 ISM (Industrial, Scientific, Medical) 대역 중 2.4GHz 대역을 사용한다. 하지만 최근 와이파어나 블루투스를 사용하는 무선기기의 증가로 인해 무선기기들 사이의 간섭이 증가하게 되었고 이는 전송속도의 저하로 이어졌다. 따라서 ISM 대역 중 사용 빈도가 적은 5GHz 대역을 와이파이에서 사용하게 되었다. 5GHz 대역은 2.4GHz 대역에 비해 간섭이 적은 것은 사실이지만, 높은 주파수는 직진성이 강해 장애물을 피하기 어려우므로 두 대역을 동시에 사용하고 있다.

아두이노에서 와이파이를 사용하기 위해서는 와이파이 쉴드가 필요하다. 여러 종류의 와이파이 쉴드가 판매되고 있으며 이 책에서는 아두이노의 표준 와이파이 쉴드를 사용한다.



(a) 전면

(b) 후면

그림 1. 와이파이 쉴드¹⁾

와이파이 쉴드는 무선 데이터 송수신을 위해 HDG204 칩을, TCP/IP 스택을 지원하기 위해 아트멜의 AT32UC3 칩을 사용하고 있다. 아두이노에는 와이파이 쉴드 사용을 위한 와이파이

1) <http://www.arduino.cc/en/Main/ArduinoWiFiShield>

라이브러리가 기본으로 포함되어 있으므로 추가적인 소프트웨어 설치 없이 와이파이 설드를 사용하여 무선 인터넷을 활용하는 스케치를 작성할 수 있다. 와이파이 설드에는 SD 카드 슬롯 역시 포함되어 있으며, 기본 라이브러리 중 하나인 SD 라이브러리를 통해 읽고 쓸 수 있지만 SD 카드에 대한 내용은 다루지 않는다.

와이파이 설드는 아두이노와의 통신을 위해 SPI 버스를 사용한다. 아두이노 우노는 디지털 11번, 12번, 13번 핀을 SPI 통신을 위한 공통 핀으로 사용하고 있다. SS (Slave Select) 핀은 임의의 핀을 사용할 수 있지만 와이파이 설드의 경우 디지털 10번 핀을 HDG204 칩을 위해, 디지털 4번 핀을 SD 카드를 위해 사용하고 있다. 또한 디지털 7번 핀은 와이파이 설드와 아두이노 사이의 핸드셰이크(handshake)를 위해 사용되므로 와이파이 설드를 사용하는 경우 7번 핀은 입출력 용도로는 사용할 수 없다.

와이파이 설드는 802.11b 및 802.11g 표준과 호환된다. 802.11 뒤에 붙는 알파벳에 따라 동일한 대역에서도 채널 폭, 사용 가능한 주파수 범위, 전송 속도 등에서 차이가 있다. 여러 가지 표준 중 b와 g가 가장 많이 사용되고 있으며 최근 듀얼밴드를 지원하는 n 역시 사용이 증가하고 있다.

암호화 방식으로는 WPA2 Personal이나 WEP 암호화 방식을 지원한다. WEP(Wired Equivalent Privacy)는 가장 널리 사용되는 방식으로 1999년 표준으로 제정되었다. 하지만 WEP는 보안상 취약점이 있어 해킹에 취약한 것으로 알려져 있어 WEP의 사용은 추천되지 않는다. 와이파이 연합(Wi-Fi Alliance)에서도 2004년 이후 WEP은 다루지 않는다. WPA(Wi-Fi Protected Access)는 WEP의 대안으로 2003년 발표된 암호화 방식이다. WPA는 2006년 WPA2로 대체되었으며 현재 가장 우수한 암호화 방식으로 알려져 있다. WPA2 Personal은 단일 패스워드를 사용하는 방식으로, 인증 서버를 필요로 하는 WPA2 Enterprise와 달리 별도의 서버를 필요로 하지 않으므로 대부분의 소규모 네트워크에서 사용되는 방식이다.

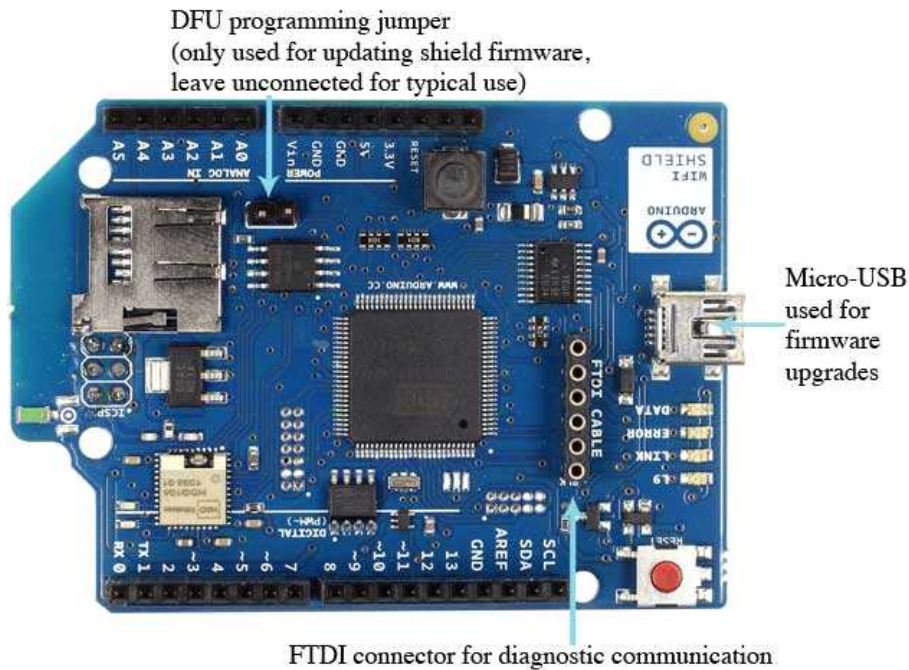


그림 2. 와이파이 쉴드 펌웨어 업그레이드

와이파이 쉴드에는 마이크로 USB 커넥터가 포함되어 있다. 이 커넥터는 아두이노 프로그래밍을 위해 사용되는 커넥터가 아니라 AT32UC3 칩의 펌웨어 업그레이드를 위해 사용된다. 펌웨어 업그레이드를 위해서는 SD 카드 슬롯 옆에 위치한 점퍼를 연결하여야 하며 와이파이 쉴드로 사용하는 경우에는 연결하지 않고 사용하여야 한다. AT32UC3 칩 옆에는 FTDI 커넥터가 마련되어 있으며 펌웨어 업그레이드 시 디버깅 용도로 사용할 수 있다.

2. WiFi 클래스

와이파이 쉴드를 사용하기 위해서는 아두이노의 기본 라이브러리로 포함되어 있는 WiFi 라이브러리를 사용하면 된다. WiFi 라이브러리 사용법은 Ethernet 라이브러리의 사용 방법과 기본적으로 동일하다. WiFi 라이브러리에는 WiFiClass, WiFiClient, WiFiServer 등의 클래스가 정의되어 있으며 이는 이더넷 라이브러리에서 EthernetClass, EthernetClient, EthernetServer 등의 클래스에 해당한다. WiFiClass의 유일한 객체인 WiFi가 헤더 파일에 정의되어 있어 별도의 객체 생성 없이도 와이파이를 사용할 수 있는 것은 EthernetClass의 유일한 객체인 Ethernet이 헤더 파일에 정의되어 있는 것과 동일하다.

와이파이가 이더넷과 다른 점 중 하나는 와이파이로 네트워크에 연결하기 위해서는 액세스 포

인트(Access Point, AP)에 연결하여야 한다는 점이다. AP는 인터넷과 연결하기 위해 무선기
기에서 거쳐야 하는 관문 역할을 하며, AP에 연결하기 위해서는 AP의 SSID가 필요하다.
SSID는 Service Set Identifier의 약어로 AP의 이름에 해당한다. 연결하고자 하는 AP가 공
개된 AP가 아니라면 패스워드 역시 필요하다.

WiFi 클래스에는 다음과 같은 멤버 함수들이 정의되어 있다.

- begin

```
int begin(char* ssid)
int begin(char* ssid, uint8_t key_idx, const char* key)
int begin(char* ssid, const char *passphrase)
- 매개변수
  ssid : 연결하고자 하는 와이파이 네트워크의 이름
  key_idx : WEP 암호화를 사용하는 네트워크에서 사용하고자 하는 키 번호 (0~3)
  key : WEP 암호화를 사용하는 네트워크에서의 16진수 문자열 키
  passphrase : WPA 암호화를 사용하는 네트워크에서의 패스워드
- 반환값 : 와이파이 연결 상태
```

WiFi 라이브러리의 네트워크 설정을 초기화하고 현재 상태를 반환한다. SSID 만을 매개변수
로 사용하는 경우는 공개된 무선 네트워크에 접속하는 경우이며, 사용하는 암호화 방식에 따
라 초기화 방법이 달라지므로 해당하는 함수를 사용하면 된다. WEP 암호화를 사용하는 경우
4개까지의 서로 다른 키를 사용할 수 있으므로 사용하고자 하는 키 번호를 지정해 주어야 한
다. 반환되는 값은 현재 와이파이의 상태로 표 1의 값 중 하나를 반환한다.

상수	값	비고
WL_NO_SHIELD	255	
WL_IDLE_STATUS	0	
WL_NO_SSID_AVAIL	1	
WL_SCAN_COMPLETED	2	
WL_CONNECTED	3	
WL_CONNECT_FAILED	4	
WL_CONNECTION_LOST	5	
WL_DISCONNECTED	6	

표 1. 와이파이 상태

- status

```
uint8_t status(void)
- 매개변수 : 없음
- 반환값 : 와이파이 연결 상태
```

현재 네트워크의 상태로 표 1의 값 중 하나를 반환한다.

- disconnect

```
int disconnect(void)
- 매개변수 : 없음
- 반환값 : 와이파이 연결 상태
```

현재 접속된 무선 네트워크에서 와이파이 설드의 연결을 종료한다. 현재 와이파이의 상태로 표 1의 값 중 하나를 반환한다.

- localIP, subnetMask, gatewayIP

```
IPAddress localIP(void)
IPAddress subnetMask(void)
IPAddress gatewayIP(void)
- 매개변수 : 없음
- 반환값 : IP 주소, 서브넷 마스크, 게이트웨이 주소
```

와이파이 설드의 IP 주소, 서브넷 마스크, 게이트웨이 주소를 반환한다.

- config

```
void config(IPAddress local_ip)
void config(IPAddress local_ip, IPAddress dns_server)
void config(IPAddress local_ip, IPAddress dns_server, IPAddress gateway)
void config(IPAddress local_ip, IPAddress dns_server, IPAddress gateway,
            IPAddress subnet)
    - 매개변수
        local_ip : 로컬 IP 주소
        dns_server : DNS 서버 주소
        gateway : 게이트웨이 주소. 디폴트값은 IP와 동일하고 마지막 바이트만 1인 값
        subnet : 서브넷 마스크. 디폴트값은 255.255.255.0
    - 반환값 : 없음
```

begin 함수는 DHCP를 통해 자동으로 IP 주소를 할당해 주는 반면, config 함수는 정적 IP 주소를 사용할 수 있도록 해준다. IP 주소 이외에도 DNS 서버 주소, 게이트웨이 주소, 서브넷 마스크 등도 설정할 수 있다. begin 함수 이전에 config 함수를 호출하면 begin 함수에서는 config 함수에 지정한 정보를 이용하여 와이파이 설드를 구성한다.

- setDNS

```
void setDNS(IPAddress dns_server1)
void setDNS(IPAddress dns_server1, IPAddress dns_server2)
    - 매개변수
        dns_server1: DNS 서버 주소 1 (primary DNS server)
        dns_server2: DNS 서버 주소 2 (secondary DNS server)
    - 반환값 : 없음
```

DNS 서버의 IP 주소를 설정한다. DNS 서버의 주소는 2개 까지 등록할 수 있다.

- SSID

```
char* SSID(void)
char* SSID(uint8_t networkItem)
    - 매개변수
        networkItem : 스캔 과정에서 발견된 무선 네트워크 번호
    - 반환값 : 네트워크의 SSID
```

매개변수가 없는 경우에는 현재 연결된 무선 네트워크의 SSID를 반환한다. 매개변수가 있는 경우에는 스캔 과정에서 발견된 네트워크의 번호에 해당하는 SSID를 반환한다.

- BSSID

```
uint8_t* BSSID(uint8_t* bssid)
- 매개변수
  bssid : BSSID 값 저장을 위한 버퍼
- 반환값 : BSSID 값 저장 버퍼의 포인터
```

연결되어 있는 네트워크의 BSSID 값인 액세스 포인트의 MAC 주소를 얻어온다.

BSSID 값은 매개변수로 지정된 버퍼에 저장되며, 반환되는 값은 버퍼의 포인터이다. 즉, 반환되는 값과 매개변수는 동일한 값을 가리킨다.

- RSSI

```
int32_t RSSI(void)
int32_t RSSI(uint8_t networkItem)
- 매개변수
  networkItem : 스캔 과정에서 발견된 무선 네트워크 번호
- 반환값 : 신호 강도
```

연결되어 있는 네트워크의 신호 강도를 반환한다. 매개변수가 있는 경우에는 스캔 과정에서 발견된 네트워크의 번호에 해당하는 RSSI 값을 반환한다. 반환되는 값의 단위는 dBm(decibel-milliwatts)으로 1mW를 0dB로 표현한 것이다.

- encryptionType

```
uint8_t encryptionType(void)
uint8_t encryptionType(uint8_t networkItem)
- 매개변수
  networkItem : 스캔 과정에서 발견된 무선 네트워크 번호
- 반환값 : 암호화 기법의 종류
```

연결되어 있는 네트워크에서 사용하는 암호화 기법 종류를 반환한다. 매개변수가 있는 경우에는 스캔 과정에서 발견된 네트워크의 번호에 해당하는 암호화 기법 종류를 반환한다. 반환되

는 값의 종류는 표 2 중 하나의 값을 갖는다.

상수	값	비고
ENC_TYPE_WEP	5	
ENC_TYPE_TKIP	2	WPA
ENC_TYPE_CCMP	4	WPA
ENC_TYPE_NONE	7	
ENC_TYPE_AUTO	8	

표 2. 암호화 기법

- `macAddress`

```
uint8_t* macAddress(uint8_t* mac)
- 매개변수
  mac : MAC 주소 저장을 위한 버퍼
- 반환값 : MAC 주소 저장 버퍼의 포인터
```

와이파이 칩의 MAC 주소를 얻어온다. MAC 주소는 매개변수로 지정된 버퍼에 저장되며, 반환되는 값은 버퍼의 포인터이다. 즉, 반환되는 값과 매개변수는 동일한 값을 가리킨다.

- `getSocket`

```
static uint8_t getSocket(void)
- 매개변수 : 없음
- 반환값 : 사용 가능한 첫 번째 소켓 번호 또는 NO_SOCKET_AVAIL (255)
```

사용 가능한 첫 번째 소켓의 번호를 반환한다. 와이파이 칩에서 사용 가능한 최대 소켓 수는 4개이다.

- `firmwareVersion`

```
static char* firmwareVersion(void)
- 매개변수 : 없음
- 반환값 : 와이파이 칩의 펌웨어 버전
```

와이파이 쉘드의 펌웨어 버전을 문자열 형식으로 반환한다.

- hostByName

```
int hostByName(const char* aHostname, IPAddress& aResult)
```

- 매개변수

aHostname : 호스트 이름

aResult : 호스트의 IP 주소

- 반환값 : 변환에 성공하면 1, 실패하면 오류 코드 반환

문자열로 주어지는 호스트 이름을 IP 주소로 변환한다.

- scanNetworks

```
int8_t scanNetworks()
```

- 매개변수 : 없음

- 반환값 : 발견된 네트워크의 수

사용 가능한 와이파이 네트워크를 스캔하고 발견된 네트워크의 수를 반환한다.

와이파이 쉘드를 사용하여 주변의 AP를 검색해 보자. 스케치 1은 주변의 AP를 검색하여 목록을 출력하는 스케치의 예이다

스케치 1. 무선 네트워크 검색

```
#include <WiFi.h>
```

```
#include <WiFiClient.h>
```

```
#include <WiFiServer.h>
```

```
#include <WiFiUdp.h>
```

```
#include <SPI.h>
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  listNetworks();
```

```
  // 액세스 포인트 검색
```

```
}

void loop() { }

void listNetworks() {
    Serial.println("** Scan Networks **");
    byte numSsid = WiFi.scanNetworks();    // 액세스 포인트 검색

    Serial.print("Number of available networks:");
    Serial.println(numSsid);                // 검색된 액세스 포인트 개수

    for (int thisNet = 0; thisNet < numSsid; thisNet++) {
        Serial.print(thisNet);
        Serial.print(" ");
        Serial.print(WiFi.SSID(thisNet));    // SSID
        Serial.print("\tSignal: ");
        Serial.print(WiFi.RSSI(thisNet));    // 신호 강도
        Serial.print(" dBm");
        Serial.print("\tEncryption: ");
        Serial.println(WiFi.encryptionType(thisNet));    // 암호화 종류
    }
}
```

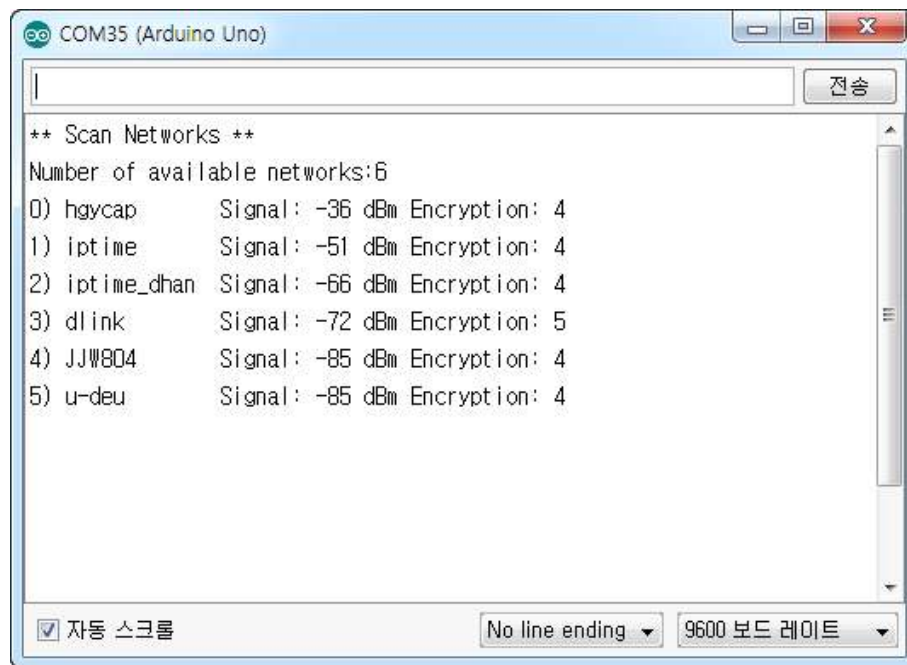


그림 3. 스케치 1 실행 결과

이제 무선 네트워크에 접속해 보자. 네트워크에 접속하기 위해서는 begin 함수를 사용할 수 있으며, WPA 암호화를 사용하는 네트워크의 경우 SSID와 패스워드를 지정하면 된다. 공개된 네트워크의 경우 패스워드는 생략할 수 있다.

스케치 2. 무선 네트워크 접속²⁾

```

#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>
#include <WiFiUdp.h>
#include <SPI.h>

char ssid[] = "hgycap";           // 네트워크의 SSID
char pass[] = "your_password";    // AP의 패스워드
int status = WL_IDLE_STATUS;

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

```

```
if (WiFi.status() == WL_NO_SHIELD) {    // 와이파이 쉴드 검사
    Serial.println("** WiFi shield NOT present.");
    while (true);
}

String fv = WiFi.firmwareVersion();    // 펌웨어 버전 검사
Serial.println("* Firmware : v." + fv);

while(status != WL_CONNECTED) {
    Serial.print("Attempting to connect to \");
    Serial.print(ssid);
    Serial.println("\");

    status = WiFi.begin(ssid, pass);    // WPA/WPA2 네트워크에 연결

    delay(10000);                        // 접속 대기
}

Serial.print("You're connected to the network \");
Serial.print(ssid);
Serial.println("\");

printCurrentNet();                        // 접속된 네트워크 정보
printWifiData();                          // 와이파이 쉴드 정보
}

void loop() { }

void printWifiData() {
    IPAddress ip = WiFi.localIP();
    Serial.print(" >> IP Address: ");
    Serial.println(ip);

    byte mac[6];
    WiFi.macAddress(mac);
```

```
Serial.print(" >> MAC address: ");
for(int i = 5; i >= 0; i--){
    Serial.print(mac[i], HEX);
    if(i != 0) Serial.print(":");
}
Serial.println();
}

void printCurrentNet() {
    Serial.print(" >> SSID : ");
    Serial.println(WiFi.SSID());

    byte bssid[6];
    WiFi.BSSID(bssid);
    Serial.print(" >> BSSID : ");
    for(int i = 5; i >= 0; i--){
        Serial.print(bssid[i], HEX);
        if(i != 0) Serial.print(":");
    }
    Serial.println();

    long rssi = WiFi.RSSI();
    Serial.print(" >> Signal strength (RSSI) : ");
    Serial.println(rssi);

    byte encryption = WiFi.encryptionType();
    Serial.print(" >> Encryption Type : ");
    Serial.println(encryption, HEX);
}
```

2) SSID와 패스워드는 접속하고자 하는 AP에 맞게 변경하여 사용하여야 한다.

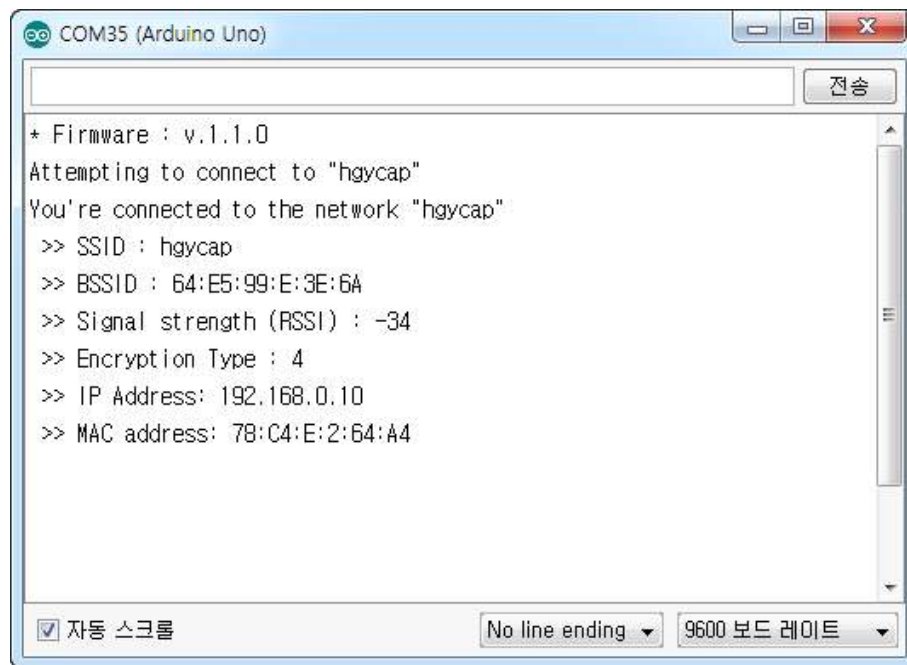


그림 4. 스케치 2 실행 결과

3. WiFiServer 클래스

WiFiServer 클래스는 다른 컴퓨터나 무선기기에서 실행 중인 클라이언트의 요청을 받아 이를 처리하기 위한 클래스로 다음과 같은 멤버 함수가 정의되어 있다.

- WiFiServer

WiFiServer(uint16_t port)

- 매개변수
port : 포트 번호
- 반환값 : 없음

지정한 포트에 들어오는 클라이언트의 요청을 감시하는 서버 객체를 생성한다.

- begin

```
void begin(void)
- 매개변수 : 없음
- 반환값 : 없음
```

서버를 시작한다. 서버가 시작되면 서버는 서버 객체 생성 시에 사용된 포트로 클라이언트의 접속을 감시하고 클라이언트의 요청이 들어오면 소켓을 생성하고 이를 클라이언트와 연결하여 클라이언트의 요청을 처리한다. 와이파이 쉘드는 최대 4개의 소켓을 사용할 수 있다.

- available

```
WiFiClient available(uint8_t* status = NULL)
- 매개변수 : 없음
- 반환값 : 클라이언트 객체
```

서버에 접속된 클라이언트 중 서버에서 읽어 들일 수 있는 데이터를 보낸 클라이언트 객체를 반환한다. 접속된 클라이언트는 클라이언트 클래스의 stop 함수를 통해 명시적으로 연결을 해제할 때까지 접속이 유지된다.

- write

```
size_t write(uint8_t data)
size_t write(const uint8_t *buf, size_t size)
- 매개변수
  data : 1 바이트 크기의 데이터
  buf : 바이트 배열 데이터
  size : buf 내 데이터 크기
- 반환값 : 클라이언트로 전송한 바이트 수
```

서버에 접속된 모든 클라이언트로 데이터를 전송한다. 반환되는 값은 개별 클라이언트에게 전송된 바이트 수를 모두 합한 값이다.

- print, println

서버에 접속된 모든 클라이언트에게 데이터를 전송한다. print와 println 함수의 사용법은 Serial 클래스의 print 및 println 함수와 동일하다.

4. WiFiClient 클래스

WiFiClient 클래스는 서버에 접속하여 통신할 수 있는 와이파이 클라이언트를 위한 클래스로 다음과 같은 멤버 함수들이 정의되어 있다.

- WiFiClient

WiFiClient()

- 매개변수 : 없음
- 반환값 : 없음

와이파이 클라이언트 객체를 생성하는 생성자로, 생성된 클라이언트는 connect 함수를 사용하여 지정된 IP 주소와 포트를 통해 서버에 접속할 수 있다.

- connect

int connect(IPAddress ip, uint16_t port)

virtual int connect(const char *host, uint16_t port)

- 매개변수
 - ip : IPAddress 형식의 서버 IP 주소
 - host : 문자열로 표현되는 서버 주소
 - port : 클라이언트가 접속할 서버의 포트 번호
- 반환값 : 접속에 성공하면 true, 실패하면 false를 반환

지정된 서버의 주소와 포트를 통해 서버로 접속을 시도한다.

- connected

uint8_t connected(void)

- 매개변수 : 없음
- 반환값 : 서버 접속 여부

클라이언트가 서버에 접속되어 있는지의 여부를 반환한다. 접속이 종료된 이후에도 읽지 않은 데이터가 존재하는 경우, 즉, 수신 버퍼에 데이터가 남아있는 경우에는 접속된 것으로 간

주된다.

- write

```
size_t write(uint8_t data)
size_t write(const uint8_t *buf, size_t size)
- 매개변수
  data : 1 바이트 크기의 데이터
  buf : 바이트 배열 데이터
  size : buf 내 데이터 크기
- 반환값 : 서버로 전송한 바이트 수
```

클라이언트가 접속되어 있는 서버로 데이터를 전송하고 전송된 바이트 수를 반환한다.

- print, println

클라이언트가 접속되어 있는 서버로 데이터를 전송한다. print와 println 함수의 사용법은 Serial 클래스의 print 및 println 함수와 동일하다.

- available

```
int available(void)
- 매개변수 : 없음
- 반환값 : 서버로부터 전송된 바이트 수
```

서버로부터 전송되어 클라이언트가 읽을 수 있는 데이터의 바이트 수를 반환한다.

- read

```
int read(void)
int read(uint8_t *buf, size_t size)
- 매개변수
  buf : 읽어 들인 데이터를 저장할 버퍼
  size : 버퍼의 바이트 단위 크기
- 반환값 : 읽어 들인 1 바이트 데이터, 0 또는 -1
```

매개변수가 없는 read 함수는 버퍼에 있는 첫 번째 바이트 데이터를 읽어 반환한다. 매개변수

가 있는 read 함수는 최대 size 바이트의 데이터를 읽어 buf에 저장하고 0을 반환한다. 읽어 들일 데이터가 존재하지 않는 경우에는 -1을 반환한다.

- flush

```
void flush(void)
- 매개변수 : 없음
- 반환값 : 없음
```

서버로부터 전송되어 아직 읽지 않은 데이터를 읽어서 수신 버퍼를 비운다.

- stop

```
void stop(void)
- 매개변수 : 없음
- 반환값 : 없음
```

서버와의 접속을 종료한다.

5. 웹 클라이언트

이더넷으로 할 수 있는 것들은 와이파이로도 할 수 있다. 다만 연결을 위한 절차와 방법에서 차이가 있을 뿐이다. 먼저 공개 API를 통해 현재 날씨 정보를 얻어오는 스케치를 와이파이를 이용해 작성해 보자. 날씨 정보 제공 서버에 관한 내용은 ‘이더넷 - 오늘의 날씨’ 장을 참고하면 되고, 와이파이로 무선 네트워크에 접속하는 방법은 스케치 2를 참고하면 된다. 스케치 3은 와이파이를 통해 날씨 정보를 얻어와 출력하는 스케치의 예이다.

스케치 3. 날씨 정보 출력³⁾

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>
#include <WiFiUdp.h>
#include <SPI.h>
```

```
char ssid[] = "hgycap";           // 네트워크의 SSID
char pass[] = "your_password";    // AP의 패스워드
int status = WL_IDLE_STATUS;

char server_address[] = "api.openweathermap.org";
char server_page[] = "/data/2.5/weather?q=\"busan\"&mode=xml";

String buffer = "";               // 수신 데이터 저장 버퍼
boolean process_it = false;

WiFiClient client;

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

  if (WiFi.status() == WL_NO_SHIELD) { // 와이파이 쉴드 검사
    Serial.println("** WiFi shield NOT present.");
    while (true);
  }

  while(status != WL_CONNECTED) {
    Serial.print("* Attempting to connect to \");
    Serial.print(ssid);
    Serial.println("\");

    status = WiFi.begin(ssid, pass); // WPA/WPA2 네트워크에 연결

    delay(10000); // 접속 대기
  }

  Serial.print("* You're connected to the network \");
  Serial.print(ssid);
  Serial.println("\");

  Serial.println("* Connecting to server...");
```

```
// 날씨 정보 제공 서버에 접속
if(client.connect(server_address, 80)){
    Serial.println("* Connected to server...");

    // 부산의 현재 날씨 정보 요청
    client.println(String("GET ") + server_page);
    client.println("Connection: close");    // 데이터 수신 종료 후 접속 종료
    client.println();                      // 요청 종료 알림
}
else{
    Serial.println("* Connection to server failed...");
}
}

void loop()
{
    // 웹 서버로부터 수신한 데이터가 존재하는 경우
    while(client.available()) {
        char c = client.read();            // 수신 데이터를 읽기
        if(c == '\n')
            process_it = true;             // 한 줄 수신 종료
        else
            buffer = buffer + c;

        if(process_it)
            process_data();                 // 수신 데이터 처리
    }

    // 서버와의 접속이 종료된 경우 클라이언트 종료
    if (!client.connected()) {
        Serial.println();
        Serial.println("* Disconnecting...");
        client.stop();                    // 클라이언트 종료

        while(true);
    }
}
```

```
}

void process_data(void)
{
    String key[] = { "temperature", "humidity", "speed" };
    String value_string = "";
    int index, position, position_end;

    // 추출하고자 하는 정보의 키워드 검색
    for(index = 0; index < 3; index++){
        position = buffer.indexOf(key[index]);
        if(position != -1) break;
    }

    // 온도, 습도, 풍속 정보가 포함되어 있는 경우
    if(index < 3){
        position = buffer.indexOf("value") + 7; // 값 시작 위치
        position_end = -1;
        for(int i = position; i < buffer.length(); i++){
            if(buffer[i] == '\\'){
                position_end = i;           // 값 종료 위치
                break;
            }
        }

        if(position_end != -1){
            value_string = buffer.substring(position, position_end);
            Serial.println(">> " + buffer);           // 수신된 문자열
            Serial.println(key[index] + " : " + value_string); // 추출된 내용
        }
    }

    buffer = "";
    process_it = false;
}
```

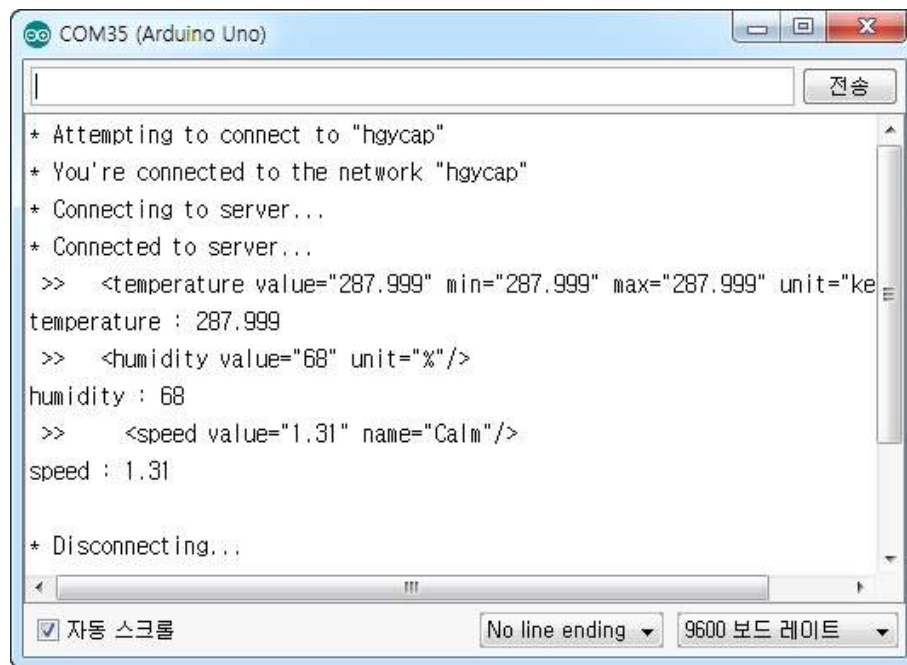


그림 5. 스케치 3 실행 결과

6. 웹 서버

와이파이 실드로 웹 서버를 구축하고 현재 가변저항의 값을 접속하는 클라이언트로 알려주는 스케치를 작성해 보자. 웹 서버 구축 및 클라이언트로 데이터 전송은 ‘이더넷 - 웹 클라이언트와 서버’ 장을 참고하면 되고, 와이파이로 무선 네트워크에 접속하는 방법은 스케치 2를 참고하면 된다. 가변저항은 아날로그 0번 A0 핀에 연결한다. 스케치 4은 와이파이를 통해 웹 서버를 구축하고 가변저항의 값을 접속한 클라이언트로 알려주는 스케치의 예이다.

스케치 4. 웹 서버⁴⁾

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>
#include <WiFiUdp.h>
#include <SPI.h>
```

3) SSID와 패스워드는 접속하고자 하는 AP에 맞게 변경하여 사용하여야 한다.

```
char ssid[] = "hgycap";           // 네트워크의 SSID
char pass[] = "your_password";    // AP의 패스워드
int status = WL_IDLE_STATUS;

WiFiServer server(80);

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

  if (WiFi.status() == WL_NO_SHIELD) { // 와이파이 쉴드 검사
    Serial.println("** WiFi shield NOT present.");
    while (true);
  }

  while(status != WL_CONNECTED) {
    Serial.print("* Attempting to connect to ");
    Serial.print(ssid);
    Serial.println("");

    status = WiFi.begin(ssid, pass); // WPA/WPA2 네트워크에 연결

    delay(3000); // 접속 대기
  }

  Serial.print("* You're connected to the network ");
  Serial.print(ssid);
  Serial.println("");

  server.begin(); // 웹 서버 시작
  Serial.print("* Web Server Address : "); // 웹 서버 주소출력
  Serial.println(WiFi.localIP());
}

void loop()
```

```
{
    // 아두이노의 웹 서버로 요청을 보낸 클라이언트 검사
    WiFiClient client = server.available();

    if(client){                                     // 클라이언트가 존재하는 경우
        while(client.available())
            client.read();                          // 클라이언트의 요청은 읽어서 버림

        // HTTP 메시지 전송
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connection: close");        // 응답 전송 후 연결 종료
        client.println("Refresh: 3");              // 3초 마다 웹 페이지 자동 갱신
        client.println();
        client.println("<!DOCTYPE HTML>");
        client.println("<html>");
        client.print("Analog Input (0 ~ 1023) : ");
        client.println(analogRead(A0));            // 가변 저항의 값을 읽어 전송
        client.println("</html>");
        client.println();

        client.stop();
    }
}
```

4) SSID와 패스워드는 접속하고자 하는 AP에 맞게 변경하여 사용하여야 한다.



그림 6. 스케치 4 실행 결과 - 시리얼 모니터

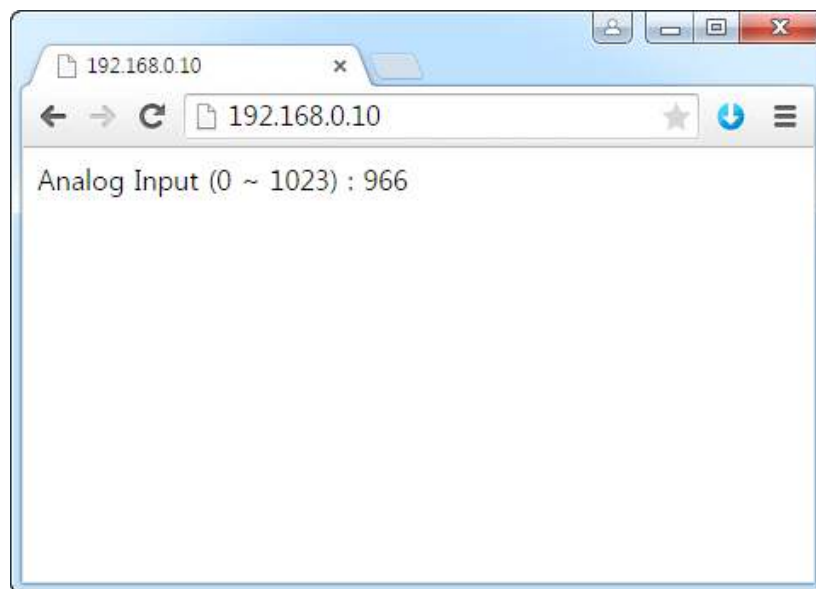


그림 7. 스케치 4 실행 결과 - 웹 브라우저