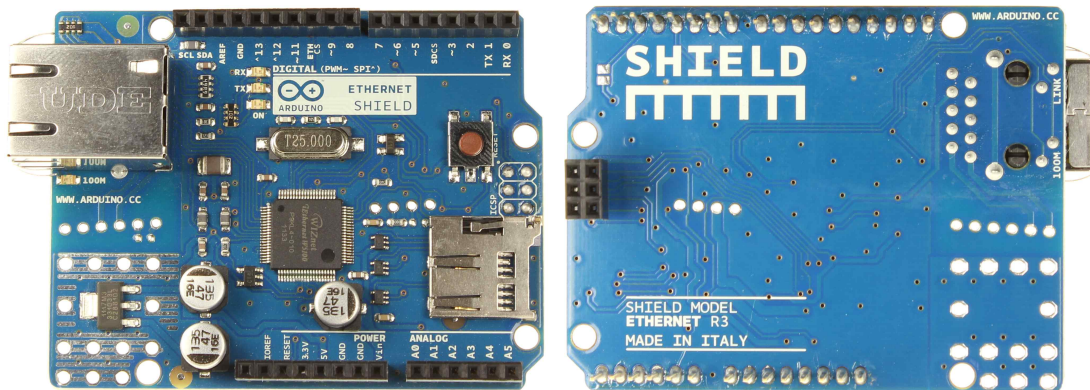


Chapter 30. 이더넷 - 웹 클라이언트와 서버

1. 이더넷 쉴드

이더넷은 근거리 네트워크(LAN, Local Area Network)를 구성하는 방식에 관한 표준으로 제록스에서 개발되어 1983년 IEEE 802.3으로 표준화되었다. 세계적으로 90% 이상이 네트워크 구성을 위해 이더넷을 사용하고 있는 것으로 추산되고 있는 등 이더넷은 인터넷 구성의 근간을 이루고 있다. 아두이노에서 이더넷을 통해 인터넷과 연결하기 위해서는 이더넷 쉴드가 필요하다. 여러 종류의 이더넷 쉴드가 판매되고 있으며 아두이노 공식 사이트에서도 이더넷 쉴드를 제작하여 판매하고 있다. 이 장에서는 아두이노의 표준 이더넷 쉴드를 사용한다.



(a) 전면

(b) 후면

그림 1. 이더넷 쉴드

이더넷 쉴드는 Wiznet W5100 이더넷 칩을 사용하고 있으며 4개까지 동시 연결을 지원한다. 아두이노 프로그램에는 이더넷 쉴드를 사용하기 위한 이더넷 라이브러리가 기본으로 포함되어 있으므로 프로그램의 추가 설치 없이도 이더넷을 이용하여 인터넷에 연결할 수 있다. 이더넷 쉴드에는 마이크로 SD (Secure Digital) 카드 슬롯이 포함되어 있으며 역시 기본 라이브러리인 SD 라이브러리를 통해 읽고 쓸 수 있지만 SD 카드에 대한 내용은 다루지 않는다.

이더넷 쉴드는 아두이노에 적층하여 사용할 수 있는 형태이며 아두이노 보드는 이더넷 쉴드와의 통신을 위해 SPI 버스를 사용한다. 아두이노 우노의 경우 11, 12, 13번 핀이 SPI를 위한 공통 핀으로 사용되며 이더넷을 위한 W5100 칩은 SS(Slave Select)로 10번 핀을, SD 카드

는 SS로 4번 핀을 사용하고 있다.

이더넷 쉘드를 사용하기 위해서는 이더넷 쉘드에 MAC 주소(Media Access Control address, 매체 접근 제어 주소)와 IP (Internet Protocol) 주소가 할당되어야 하며 Ethernet 클래스의 begin 함수를 이용하여 설정할 수 있다. MAC 주소는 하드웨어에 할당되는 고유 주소로 최신 이더넷 쉘드의 경우 MAC 주소가 표시되어 있으므로 이를 사용하면 된다. MAC 주소가 표시되어 있지 않은 쉘드의 경우에는 임의의 주소를 사용할 수 있지만 여러 장치에서 동일한 MAC 주소를 사용하여서는 안된다는 점은 기억하여야 한다. IP 주소의 경우 네트워크 설정에 따라 달라지며 고정 IP 주소는 물론 DHCP(Dynamic Host Configuration Protocol)를 통해 동적 IP 주소를 할당 받아 사용할 수 있다.

이더넷 연결을 위한 Ethernet 라이브러리는 이더넷 연결 설정을 위한 EthernetClass 클래스, 이더넷 클라이언트와 서버에 해당하는 EthernetClient 클래스와 EthernetServer 클래스 등으로 구성되어 있다. IP 주소 지정을 위한 IPAddress 클래스는 Ethernet 라이브러리가 아닌 아두이노의 기본 클래스에 포함되어 있다. Ethernet 라이브러리를 구성하는 클래스 중 EthernetClass는 아두이노가 이더넷을 통해 데이터를 주고받을 수 있도록 이더넷 쉘드 설정에 사용되는 클래스이므로 EthernetClient나 EthernetServer 클래스를 사용하기 위해서는 반드시 포함되어야 한다. EthernetClass 클래스는 유일한 객체인 Ethernet을 정의하고 있으므로 스케치에서는 객체를 생성할 필요가 없다.

1. IPAddress 클래스

IPAddress 클래스는 IP 주소 지정을 위한 클래스로 이더넷 쉘드가 지원하는 IP 주소 형식은 IPv4 주소인 32비트 형식이다. IP 주소는 4개의 부호 없는 1바이트 값(byte 또는 unsigned char) 배열이나 1개의 부호 없는 4바이트 값을 지정하여 초기화할 수 있다.

```
IPAddress( uint8_t first_octet, uint8_t second_octet,
           uint8_t third_octet, uint8_t fourth_octet)
IPAddress(uint32_t address)
IPAddress(const uint8_t *address)
- 매개변수
  first_octet, second_octet, third_octet, fourth_octet : 1 바이트 주소 데이터
  address : 4바이트 주소 데이터 또는 1바이트 주소 데이터 배열
- 반환값 :djqtdma
```

스케치 1은 IP 주소를 지정하는 여러 가지 방법을 보여준다. 포함된 헤더 파일은 ‘스케치 → 라이브러리 가져오기... → Ethernet’ 메뉴를 선택한 경우 포함되는 헤더 파일들이다. 모든 헤더가 필요한 것은 아니지만 그대로 사용하기로 하자. ‘SPI.h’는 이더넷 칩드가 SPI 방식을 사용하므로 포함되었다.

스케치 1. IP 주소 지정 방법

```
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <SPI.h>

byte address[] = {192, 168, 1, 1};

IPAddress ip1(192, 168, 1, 1);
IPAddress ip2(address);
IPAddress ip3(0xC0A80101);

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

void loop(){
    Serial.println(ip1);
    Serial.println(ip2);
    Serial.println(ip3);

    while(true);
}
```

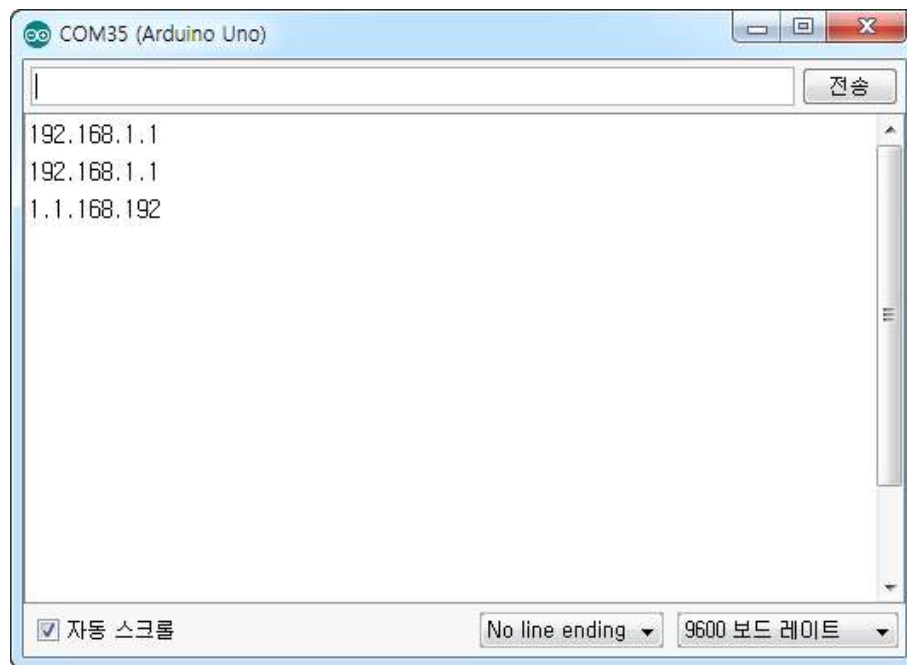


그림 2. 스케치 1 실행 결과

그림 2에서 알 수 있듯이 IP 주소를 4바이트 숫자로 지정하는 경우 다른 주소 지정 방법과 달리 주소가 역순으로 할당되는 점에 주의하여야 한다.

2. Ethernet 클래스

Ethernet 클래스는 이더넷 라이브러리와 네트워크 설정을 초기화하기 위해 사용하는 클래스이다. 버전 1.0 이후로는 IP 주소를 동적으로 할당받을 수 있는 DHCP를 지원하지만 DHCP를 사용하는 경우 스케치의 크기가 증가하는 단점이 있다. Ethernet 클래스에는 다음과 같은 멤버 함수들이 정의되어 있다.

- begin

```

int begin(uint8_t *mac_address)
void begin(uint8_t *mac_address, IPAddress local_ip)
void begin(uint8_t *mac_address, IPAddress local_ip, IPAddress dns_server)
void begin(uint8_t *mac_address, IPAddress local_ip, IPAddress dns_server,
            IPAddress gateway)
void begin(uint8_t *mac_address, IPAddress local_ip, IPAddress dns_server,
            IPAddress gateway, IPAddress subnet)

```

- 매개변수
 - mac_address : 6 바이트 배열의 MAC 주소
 - 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 : DHCP를 사용하는 경우 IP 주소 할당에 성공하면 1을 반환하고 실패하면 0을 반환한다. DHCP를 사용하지 않는 경우 반환값은 없다.

이더넷 라이브러리와 네트워크 설정을 초기화한다. DHCP를 통해 IP 주소를 할당받기 위해서는 MAC 주소만을 지정하고 begin 함수를 호출하면 된다. DHCP를 사용하는 경우 IP 주소 할당에 성공하면 1을, 실패하면 0을 반환한다. 고정 IP를 사용하는 경우 반환값은 없다.

- localIP, subnetMask, gatewayIP, dnsServerIP

```

IPAddress localIP()
IPAddress subnetMask()
IPAddress gatewayIP()
IPAddress dnsServerIP()

```

- 매개변수 : 없음
- 반환값 : IP 주소, 서브넷 마스크, 게이트웨이 주소, DNS 서버 주소

이더넷 설드의 IP 주소, 서브넷 마스크, 게이트웨이 주소, DNS 서버 주소를 반환한다. DHCP를 통해 주소가 자동으로 할당된 경우 주소 확인을 위해 사용할 수 있다. 스케치 2는 DHCP를 통해 자동으로 구성된 IP 정보를 확인하는 스케치의 예이다.

스케치 2. IP 주소 지정 방법

```
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <SPI.h>

byte MAC[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xD2, 0x16 };
IPAddress IP(192, 168, 0, 10);

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

  // Ethernet.begin(MAC, IP);           // 고정 IP를 사용하는 경우
  if(Ethernet.begin(MAC) == 0){        // DHCP로 유동 IP를 사용하는 경우
    Serial.println("* DHCP failed...");
    Ethernet.begin(MAC, IP);          // DHCP가 실패한 경우
  }

  delay(2000);                        // 이더넷 쉘드의 초기화 대기
}

void loop(){
  IPAddress address;

  Serial.print("IP Address  : ");
  address = Ethernet.localIP();        // IP 주소
  print_address(address);

  Serial.print("Subnet Mask : ");
  address = Ethernet.subnetMask();     // 서브넷 마스크
  print_address(address);

  Serial.print("Gateway    : ");
  address = Ethernet.gatewayIP();      // 게이트웨이 주소
```

```
print_address(address);

Serial.print("DNS Server  : ");
address = Ethernet.dnsServerIP();    // DNS 서버 주소
print_address(address);

while(true);
}

void print_address(IPAddress address)  // IP 주소 출력
{
    for(int i = 0; i < 4; i++){
        Serial.print(address[i], DEC);
        if(i != 3) Serial.print(".");
    }
    Serial.println();
}
```

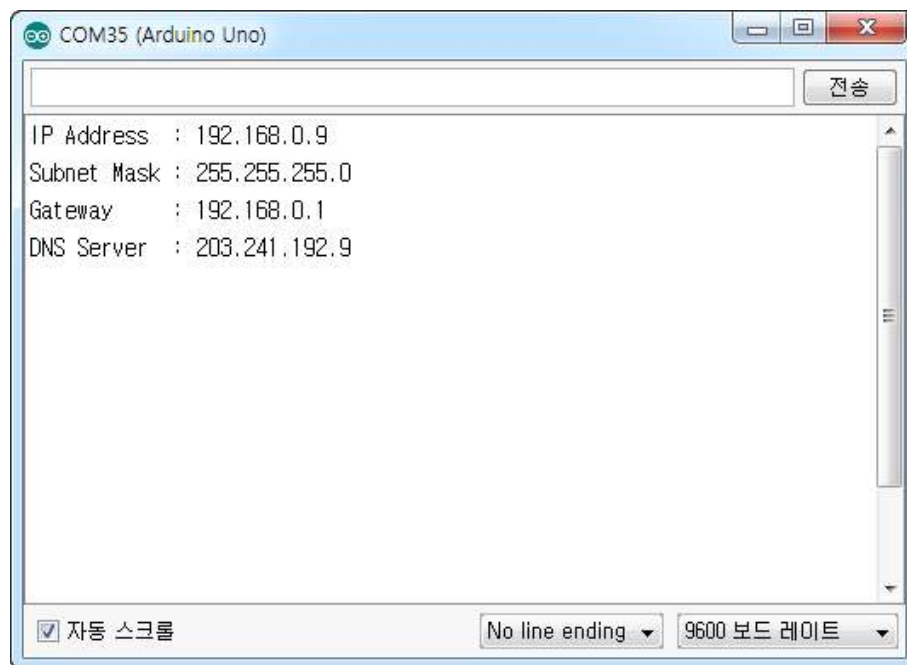


그림 3. 스케치 2 실행 결과

스케치 2에서는 DHCP를 통해 자동으로 IP 주소가 결정되도록 하였으나 이후로는 '192.168.0.10'의 고정된 주소를 사용하는 것으로 한다.

- maintain

```
int maintain()
- 매개변수 : 없음
- 반환값 : 주소 갱신 결과
```

DHCP를 통해 할당된 IP 주소를 갱신한다. DHCP를 통해 IP 주소가 할당된 경우 이더넷 장치는 일정 시간 동안만 할당된 주소를 사용할 수 있으며 maintain 함수를 통해 DHCP 서버에 주소 갱신을 요청하여야 한다. DHCP 서버 구성에 따라 갱신된 주소는 이전 주소와 동일한 주소이거나 다른 주소일 수 있으며 갱신에 실패할 수도 있다. 반환되는 값은 표 1의 값들 중 하나이다.

상수	값	의미
DHCP_CHECK_NONE	0	갱신 작업이 진행되지 않음
DHCP_CHECK_RENEW_FAIL	1	renew 실패
DHCP_CHECK_RENEW_OK	2	renew 성공
DHCP_CHECK_REBIND_FAIL	3	rebind 실패
DHCP_CHECK_REBIND_OK	4	rebind 성공

표 1. maintain 함수의 반환값

표 1에서 renew는 현재 사용 중인 IP 주소를 할당 받은 DHCP 서버로부터 주소를 갱신하는 것을 말하며 rebind는 기존 DHCP 서버로의 접속에 실패하여 새로운 DHCP 서버로부터 주소를 할당받는 것을 말한다.

3. EthernetServer 클래스

EthernetServer 클래스는 다른 컴퓨터나 기기에서 실행 중인 클라이언트의 요청을 받아들이고 클라이언트와의 통신을 관리하기 위한 클래스로 다음과 같은 멤버 함수들이 정의되어 있다.

- EthernetServer

```
EthernetServer(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 서버 생성 시에 지정한 포트에 들어오는 클라이언트의 요청을 감시하고 클라이언트의 요청이 들어오면 클라이언트와 통신하기 위한 소켓을 생성하여 클라이언트와 연결시킨다. 이더넷 실드는 최대 4개의 소켓을 생성할 수 있다.

- available

```
EthernetClient available()
```

- 매개변수 : 없음
- 반환값 : 클라이언트 객체

서버에 접속된 클라이언트 중 서버에서 읽어 들일 수 있는 데이터를 보낸 클라이언트 객체를 반환한다. 클라이언트의 접속 요청이 있는 경우 서버는 클라이언트의 접속을 허용해주는 절차가 필요하며 이 과정은 Ethernet 라이브러리에서 자동으로 처리된다. 접속이 이루어지면 서버는 클라이언트별로 연결을 관리하게 되며 available이 반환하는 EthernetClient 객체는 서버 측에서 접속을 요청한 클라이언트와 1:1로 연결된 서버 쪽 객체를 의미한다. 접속된 클라이언트는 클라이언트 클래스의 stop 함수를 통해 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 클라이언트에게 데이터를 전송한다. 반환되는 값은 개별 클라이언트에게 전송된 바이트 수를 모두 합한 값이다.

- print, println

서버에 접속된 모든 클라이언트에게 데이터를 전송한다. println 함수는 마지막에 개행 문자를 전송하는 것 이외에는 print 함수와 동일하다. print 함수는 문자열뿐만이 아니라 정수나 실수를 문자열로 변환하여 전송할 수 있다. print와 println 함수의 사용법은 Serial 클래스의 print 및 println 함수와 동일하다.

4. EthernetClient 클래스

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

- EthernetClient

```
EthernetClient()
- 매개변수 : 없음
- 반환값 : 없음
```

이더넷 클라이언트 생성자로 생성된 클라이언트는 connect 함수를 통해 지정한 IP 주소와 포트를 통해 서버에 접속을 시도할 수 있다. 한 가지 주의할 점은 클라이언트가 서버에 접속된 경우 서버에서 접속한 클라이언트와 통신하기 위해 생성되는 객체 역시 EthernetClient 클래스의 객체라는 점이다.

- connect

```
int connect(IPAddress ip, uint16_t port)
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 이용하여 서버로 접속을 시도한다. 서버의 주소는 IPAddress 형식의 주소로 지정할 수도 있고 문자열로 표현되는 URL (Uniform Resource Locator) 형식으로 지정할 수도 있다. URL로 주소가 지정되는 경우 DNS 서버로부터 URL에 해당하는 숫자 형식의 IP 주소를 얻어온 후 접속을 시도한다. 접속에 성공하면 true를, 실패하면 false를 반환한다.

- connected

```
uint8_t connected()
```

- 매개변수 : 없음
- 반환값 : 서버 접속 여부

클라이언트가 서버에 접속되어 있는지의 여부를 반환한다. 실제로 접속이 종료되었더라도 읽지 않은 데이터가 존재하는 경우, 즉, 수신 버퍼에 데이터가 남아있는 경우 접속되어 있는 것으로 간주된다.

- 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 전송하며 전송된 바이트 수를 반환한다.

- print, println

클라이언트가 접속되어 있는 서버로 데이터를 전송한다. println 함수는 마지막에 개행 문자를 전송하는 것 이외에는 print 함수와 동일하다. print 함수는 문자열뿐만이 아니라 정수나 실수를 문자열로 변환하여 전송해준다. print와 println 함수의 사용법은 Serial 클래스의 print 및 println 함수와 동일하다.

- available

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

서버로부터 전송되어 클라이언트가 읽을 수 있는 데이터의 바이트 수를 반환한다.

- read

```
int read()
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, 읽어 들인 데이터의 바이트 수 또는 -1
```

매개변수가 없는 read 함수는 버퍼에 있는 첫 번째 바이트 데이터를 읽어 반환한다. 매개변수가 있는 read 함수는 최대 size 바이트의 데이터를 읽어 buf에 저장하고 읽어 들인 데이터의 바이트 수를 반환한다. 읽어 들일 데이터가 존재하지 않는 경우에는 -1을 반환한다.

- flush

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

서버로부터 전송되어 아직 읽지 않은 데이터를 버린다.

- stop

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

서버와의 접속을 종료한다.

5. 웹 클라이언트

EthernetClient 클래스를 이용하여 웹 클라이언트를 만들어 보자. 웹 클라이언트는 웹 서버에 접속하여 웹 페이지를 다운받아 보여주는 역할을 한다. 웹 페이지는 HTML(Hyper Text Markup Language)로 만들어져서 내용 이외에도 페이지 레이아웃 및 속성 지정을 위한 다양한 태그들이 포함되어 있어 HTML 문서를 해석해서 보여줄 수 있는 기능이 없다면 그 내용을 알아보기 어렵다. 아두이노에는 웹 브라우저 기능을 구현하기 어려우므로 여기서는 수신된 내용을 시리얼 모니터로만 표시하도록 한다. 서버에 접속해서 데이터를 얻어오는 과정이 중요하며 수신된 내용 자체가 중요한 것은 아니다.

스케치 3. 웹 클라이언트

```
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <SPI.h>

byte MAC[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xD2, 0x16 };
IPAddress IP(192, 168, 0, 10);
EthernetClient client;
char server_address[] = "www.google.com";
```

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

  Ethernet.begin(MAC, IP);          // 고정 IP 사용

  delay(2000);                      // 이더넷 설드의 초기화 대기

  Serial.println("* Connecting to server...");
  // www.google.com의 80번 포트로 접속 시도
  if(client.connect(server_address, 80)){
    Serial.println("* Connected to server...");

    // 'Ethernet'에 대한 검색 결과 요청
    client.println("GET /search?q=Ethernet HTTP/1.0");
    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
  if (client.available()) {
    char c = client.read();           // 수신 데이터를 읽기
    Serial.print(c);                  // 시리얼 모니터로 수신 데이터 출력
  }

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

```
while(true);  
}  
}
```

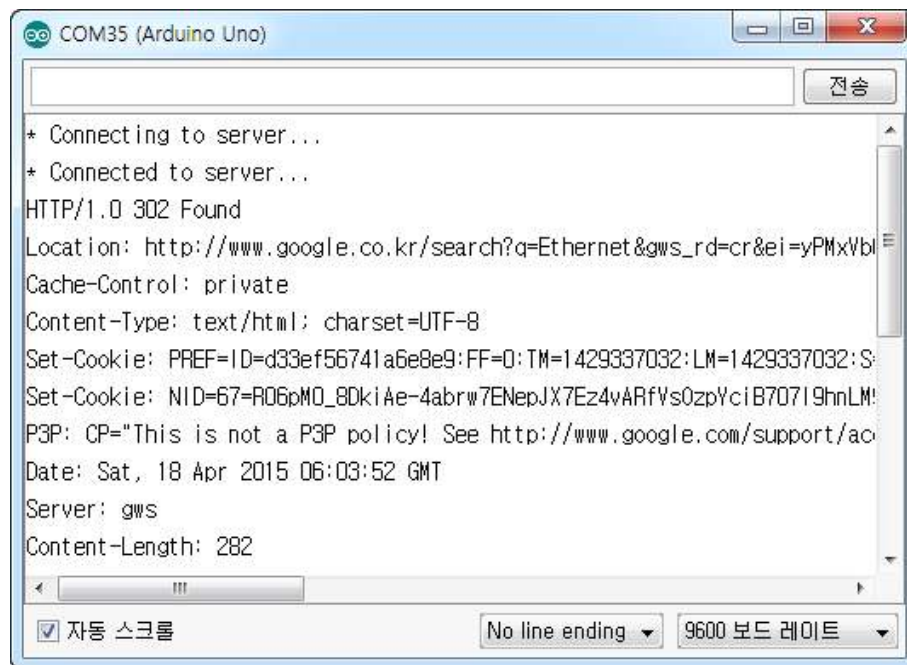


그림 4. 스케치 3 실행 결과

6. 웹 서버

아두이노로 일반 웹 서버에 접속하여 정보를 읽어오는 일보다는 아두이노에 웹 서버를 설치하고 웹 브라우저에서 아두이노로 접속하여 아두이노에 연결된 센서나 기타 장치들의 정보를 얻어오는 프로젝트는 흔히 볼 수 있다. 아두이노에 가변 저항을 연결하고 현재 가변 저항의 값을 아두이노로 접속하는 웹 클라이언트에 보여주는 프로그램을 작성해 보자. 이더넷 쉴드에 가변 저항을 아날로그 0번 핀 A0에 연결한다. 가변 저항 연결을 위한 핀은 이더넷 쉴드가 사용하는 핀과 겹치지 않으므로 아두이노 보드에 연결하는 것과 동일한 방법으로 연결하면 된다.

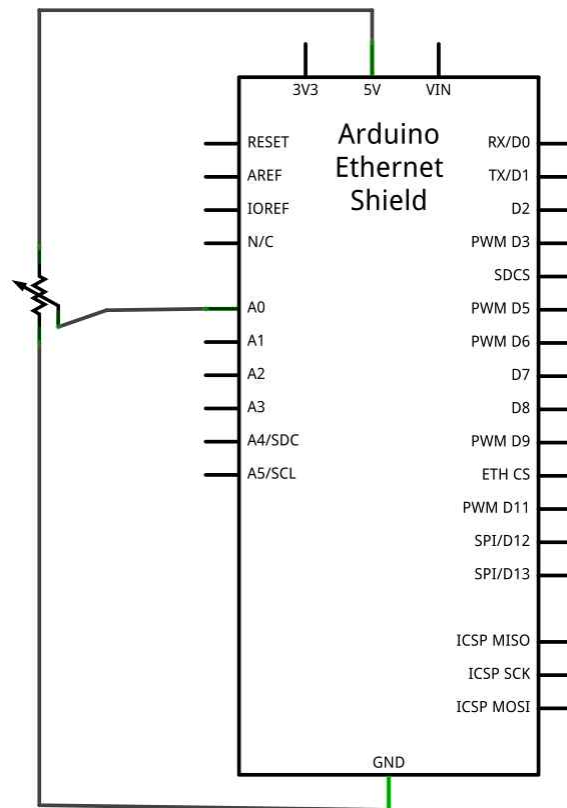


그림 5. 이더넷 실드에 가변 저항 연결 회로도

스케치 4는 아두이노에 웹 클라이언트가 접속한 경우 현재 가변저항의 값을 전송하는 스케치의 예이다. 웹 서버는 EthernetServer 클래스의 객체를 생성하고 begin 함수를 통해 시작할 수 있다. 클라이언트의 접속 여부는 EthernetServer 클래스의 available 함수를 통해 알아낼 수 있으며, 접속한 클라이언트가 있다면 available 함수는 EthernetClient 객체를 반환한다. 따라서 이후 클라이언트로의 데이터 전송은 EthernetClient 클래스의 객체를 통해 이루어진다.

스케치 4. 웹 서버

```
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
```

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

byte MAC[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xD2, 0x16 };
IPAddress IP(192, 168, 0, 10);
EthernetServer server(80);           // 80번 포트를 이용하는 웹 서버 생성

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

  Ethernet.begin(MAC, IP);           // 고정 IP 경우
  delay(2000);                       // 이더넷 설드의 초기화 대기

  server.begin();                    // 웹 서버 시작
  Serial.print("* Web Server Address : "); // 웹 서버 주소를 시리얼 모니터로 출력
  Serial.println(Ethernet.localIP());

}

void loop()
{
  // 아두이노의 웹 서버로 요청을 보낸 클라이언트 검사
  EthernetClient client = server.available();

  if(client){                        // 클라이언트가 존재하는 경우
    // 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();  
}
```

시리얼 모니터에는 현재 서버의 주소가 출력된다. 출력된 서버 주소를 웹 브라우저에 입력하면 3초 마다 자동으로 재접속하여 현재 가변 저항의 값을 읽어와 웹 브라우저에 나타낸다. 가변저항을 돌리면 그 값이 웹 페이지에 나타나는 것을 확인할 수 있다.



그림 6. 스케치 4 실행 결과 - 시리얼 모니터

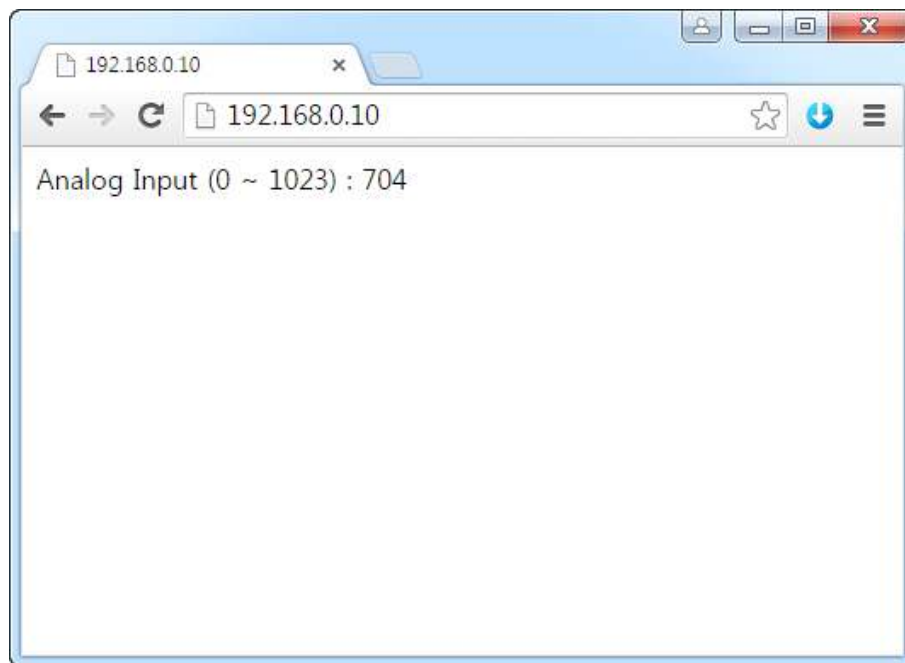


그림 7. 스케치 4 실행 결과 - 웹 브라우저

스마트폰에서도 아두이노에 접속하여 현재 가변저항의 값을 얻어오는 것이 가능하다. 다만, 이더넷 실드가 공유기를 통해 연결되어 있으므로 스마트폰 역시 동일한 공유기에 와이파이로 접속된 경우에만 가능하다.

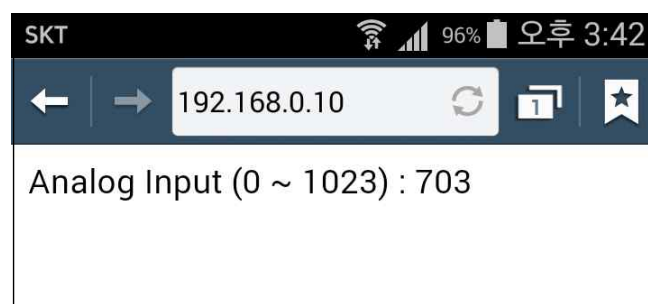


그림 8. 스케치 4 실행 결과 - 스마트폰