

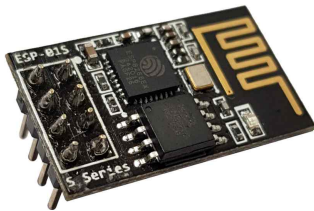
APPENDIX  
17

## 와이파이 연결

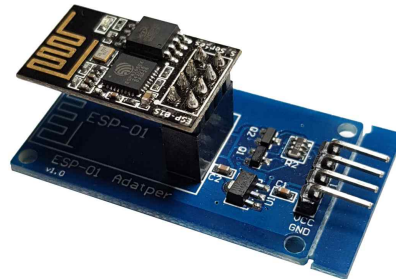
### 필요한 부품

- 아두이노 우노
- ESP-01
- RGB LED(공통 양극 방식)
- 220Ω 저항(4개)
- 피에조 버저
- 캐릭터 LCD(16×2 크기)
- 캐릭터 LCD I<sup>2</sup>C변환 모듈

이 장에서는 ESP8266 칩을 사용하여 만든 시리얼 모듈인 ESP-01 모듈을 사용한다. [그림 1(a)]는 ESP-01 모듈을 나타낸다.



(a) ESP-01 모듈

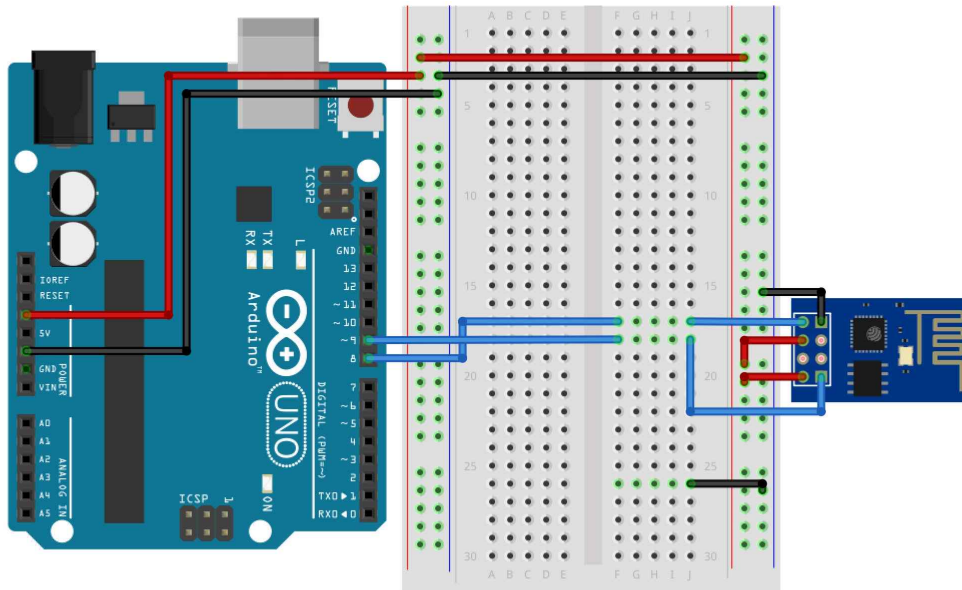


(b) ESP-01 모듈 어댑터 보드

[그림 1] ESP-01 모듈

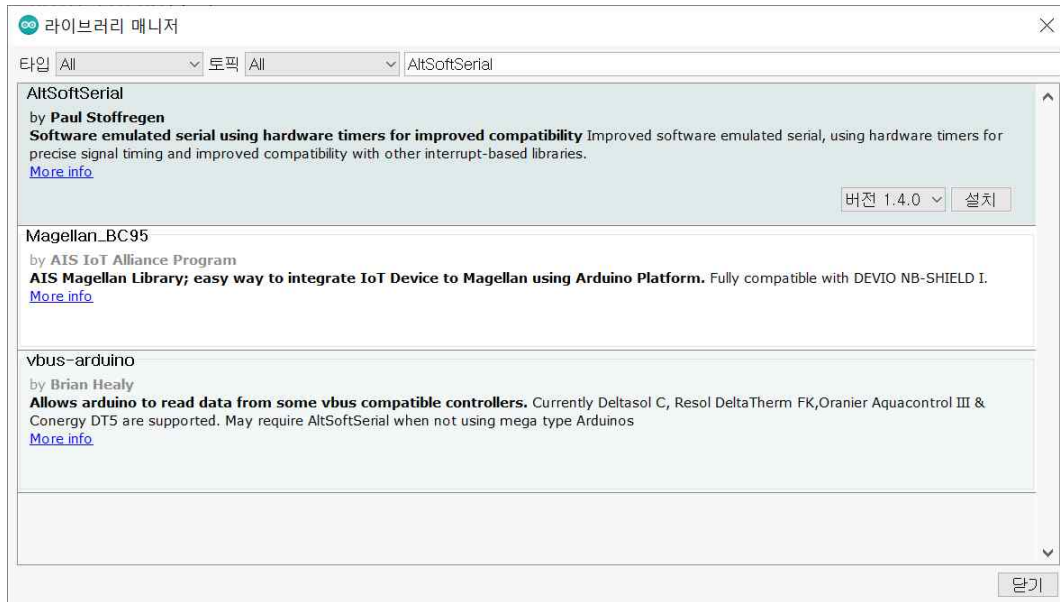
ESP 시리즈 모듈은 저렴한 가격으로 와이파이 통신을 가능하게 해주며 특히 ESP-01 모듈은 작고 저렴한 모듈로 와이파이 어댑터로 사용하기에 적합하다. 하지만 ESP-01 모듈은 2열의 핀 구성을 가지고 있어 브레드보드에 연결하여 사용할 수 없고 핀 연결이 복잡해질 수 있으므로 [그림 1(b)]의 어댑터 보드 사용을 추천한다. 어댑터 보드를 사용하면 UART 통신 장치와 마찬가지로 전원과 RX, TX의 4개 핀 연결만으로 간단하게 연결하여 사용할 수 있다.

ESP-01 모듈은 HM-10 블루투스 모듈과 마찬가지로 UART 시리얼 통신으로 AT 명령을 전송하여 설정하고 사용할 수 있다. 먼저 ESP-01 모듈을 그림 2와 같이 아두이노 우노에 연결한다. 이때 ESP-01 모듈에 연결해야 하는 전원은 3.3V임에 주의해야 한다.



[그림 2] ESP-01 모듈 연결

아두이노 우노에서는 AltSoftSerial 라이브러리를 사용한다. AltSoftSerial 라이브러리는 사용할 수 있는 핀이 정해져 있지만, 하드웨어 인터럽트를 사용하므로 SoftwareSerial 라이브러리보다 많은 데이터를 송수신할 때 안정적인 동작을 보여준다. WiFi 통신에서는 많은 데이터를 전송해야 하는 경우가 많으므로 이 장에서는 AltSoftSerial 라이브러리를 사용하였다. AltSoftSerial 라이브러리는 SoftwareSerial 라이브러리와 같은 방법으로 사용할 수 있다. 먼저 라이브러리 매니저에서 'AltSoftSerial'을 검색하여 AltSoftSerial 라이브러리를 설치한다.



[그림 3] AltSoftSerial 라이브러리 검색 및 설치

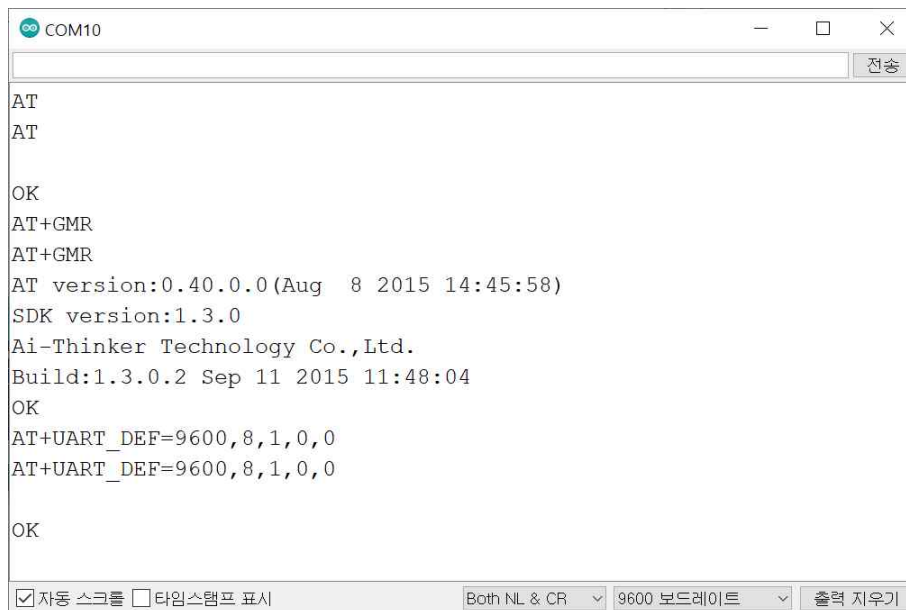
아두이노 우노에는 [코드 1]을 업로드한다. [코드 1]은 시리얼 모니터에 입력된 내용을 ESP-01 모듈로 전달하고, ESP-01 모듈에서 출력된 내용을 시리얼 모니터로 출력하는 스케치다. 시리얼 모니터는 9600 보율을 선택하고 추가 문자는 'Both NL & CR'을 선택한다.<sup>1</sup>

코드 1	ESP-01 모듈 설정	APPENDIX_17-1_ESP_setting.ino
<pre>#include &lt;AltSoftSerial.h&gt;  AltSoftSerial ESP;                                // 미리 정해진 핀 사용  void setup() {   Serial.begin(9600);                             // 컴퓨터와의 시리얼 통신 초기화   ESP.begin(9600);                                 // ESP 모듈과의 시리얼 통신 초기화 }  void loop() {   if (Serial.available()) {                        // 시리얼 모니터 → 아두이노 → ESP 모듈</pre>		

<sup>1</sup> ESP-01 모듈에 설치된 펌웨어 버전에 따라 ESP-01 모듈과의 통신 속도는 9,600 또는 115,200 보율일 수 있으므로 설정 과정에서 보율을 변경하면서 테스트해 보아야 한다. 이 장에서는 115,200 보율을 사용하는 것으로 가정한다.

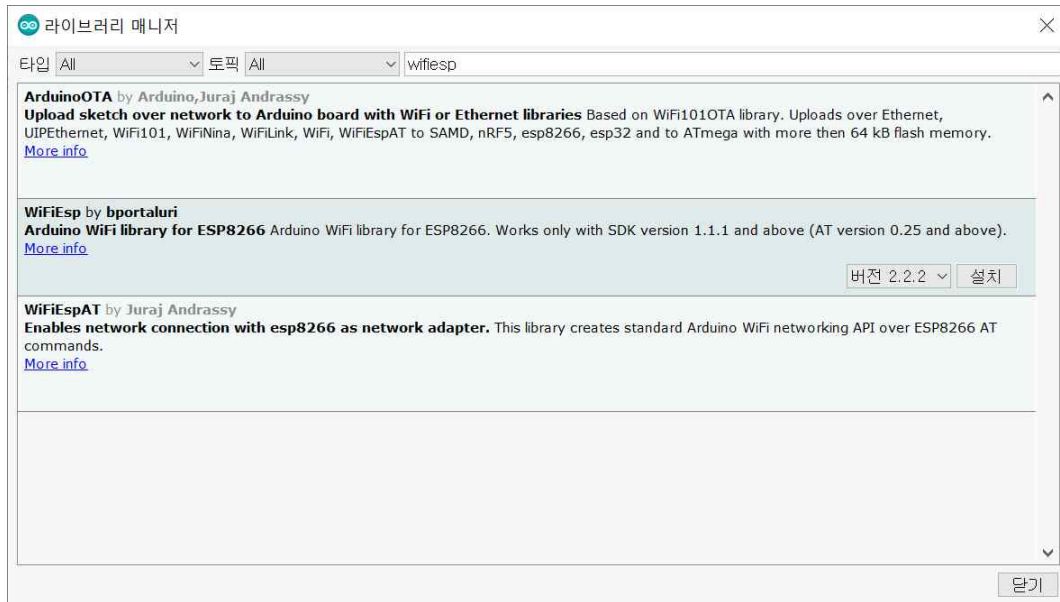
```
char ch = Serial.read();  
Serial.write(ch);  
ESP.write(ch);  
}  
  
if (ESP.available()) {           // ESP 모듈 → 아두이노 → 시리얼 모니터  
    char ch = ESP.read();  
    Serial.write(ch);  
}  
}
```

시리얼 모니터에 'AT' 명령을 입력하여 'OK'가 수신되면 ESP-01 모듈이 정상적으로 동작하는 것이다. AT+GMR은 설치된 펌웨어의 버전을 확인하는 명령이다. 보울을 9600으로 변경하기 위해서는 AT+UART\_DEF=9600,8,1,0,0 명령을 실행하면 된다. 보울 변경에 사용된 인자는 각각 보울, 데이터 비트 8 비트, 정지 비트 1비트, 패리티 없음, 흐름 제어 없음을 의미한다.



[그림 3] ESP-01 모듈의 동작 확인

AT 명령으로 ESP-01 모듈을 제어할 수는 있지만, 사용이 복잡하고 직관적이지 않은 단점이 있으므로 전용 라이브러리를 통해 사용하는 것이 일반적이다. 몇 가지 라이브러리가 있지만, 이 장에서는 WiFiEsp 라이브러리를 사용한다.



[그림 4] WiFiEsp 라이브러리 설치

라이브러리를 설치하였으면 스케치북의 라이브러리 디렉터리 아래 ‘\WiFiEsp\src\utility’ 디렉터리의 ‘debug.h’ 파일에서 `_ESPLOGLEVEL_` 상수를 0으로 설정하여 진단 메시지가 출력되지 않도록 한다.

[코드 2]는 WiFiEsp 라이브러리를 사용하여 와이파이 네트워크에 연결하고 DHCP에 의해 ESP-01 모듈에 할당된 IP 주소를 출력하는 스케치다.

코드 2	와이파이 네트워크 연결	APPENDIX_17-2_connect_to_wifi.ino
<pre>#include &lt;AltSoftSerial.h&gt; #include &lt;WiFiEsp.h&gt;  AltSoftSerial ESP;                                // 미리 정해진 핀 사용  // 와이파이 네트워크 정보 : 네트워크 이름(SSID)와 비밀번호 char WIFI_SSID[] = "네트워크 이름"; char WIFI_PASSWORD[] = "네트워크 비밀번호";  // 와이파이 액세스 포인트와의 연결 상태 관리 int wifi_status = WL_IDLE_STATUS;</pre>		

```
void setup() {  
    Serial.begin(9600);           // 컴퓨터와의 시리얼 통신 초기화  
    ESP.begin(9600);             // 와이파이 모듈과의 시리얼 통신 초기화  
  
    WiFi.init(&ESP);             // ESP-01 모듈 초기화  
  
    Serial.print("연결할 액세스 포인트 : ");  
    Serial.println(WIFI_SSID);  
  
    wifi_status = WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  
  
    if (wifi_status != WL_CONNECTED) {  
        Serial.println("** AP에 연결할 수 없습니다.");  
        while (1);  
    }  
    else {  
        Serial.println("* AP에 연결되었습니다.");  
        Serial.println();  
    }  
  
    // DHCP를 통해 할당받은 IP 주소 출력  
    IPAddress ip = WiFi.localIP();  
    Serial.print("아두이노의 IP 주소 : ");  
    Serial.println(ip);  
}  
  
void loop() {  
    // 아무런 작업도 하지 않음  
}
```



[그림 5] 스케치 2 실행 결과

ESP-01 모듈 역시 서버를 구축하기 위해 사용할 수 있다. [코드 3]은 ESP-01 모듈을 서버로 설정하고 클라이언트, 즉 브라우저가 ESP-01 모듈에 할당된 IP 주소로 접속하는 경우 클라이언트가 보낸 요청 정보를 출력하는 스케치다. 아두이노 우노는 서버에서 수신한 데이터를 모두 저장할 만큼 SRAM이 충분하지 않으므로 최대 200 바이트의 문자만을 저장하는 것으로 하였다.

### 코드 3

### 간단한 서버

APPENDIX\_17-3\_bare\_minimum\_server.ino

```
#include <AltSoftSerial.h>
#include <WiFiEsp.h>

AltSoftSerial ESP;                                // 미리 정해진 핀 사용

// 와이파이 네트워크 정보 : 네트워크 이름(SSID)와 비밀번호
char WIFI_SSID[] = "네트워크 이름";
char WIFI_PASSWORD[] = "네트워크 비밀번호";

// 클라이언트 데이터 수신 버퍼 제어
const int MAX_LENGTH = 200;
char incoming_line[MAX_LENGTH + 1], ch;
int index;
```

```
// 와이파이 액세스 포인트와의 연결 상태 관리
int wifi_status = WL_IDLE_STATUS;

WiFiEspServer server(80);

void setup() {
  Serial.begin(9600);           // 컴퓨터와의 시리얼 통신 초기화
  ESP.begin(9600);             // 와이파이 모듈과의 시리얼 통신 초기화

  WiFi.init(&ESP);             // ESP-01 모듈 초기화

  Serial.print("연결할 액세스 포인트 : ");
  Serial.println(WIFI_SSID);

  wifi_status = WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  if (wifi_status != WL_CONNECTED) {
    Serial.println("*** AP에 연결할 수 없습니다.");
    while (1);
  }
  else {
    Serial.println("* AP에 연결되었습니다.");
    Serial.println();
  }

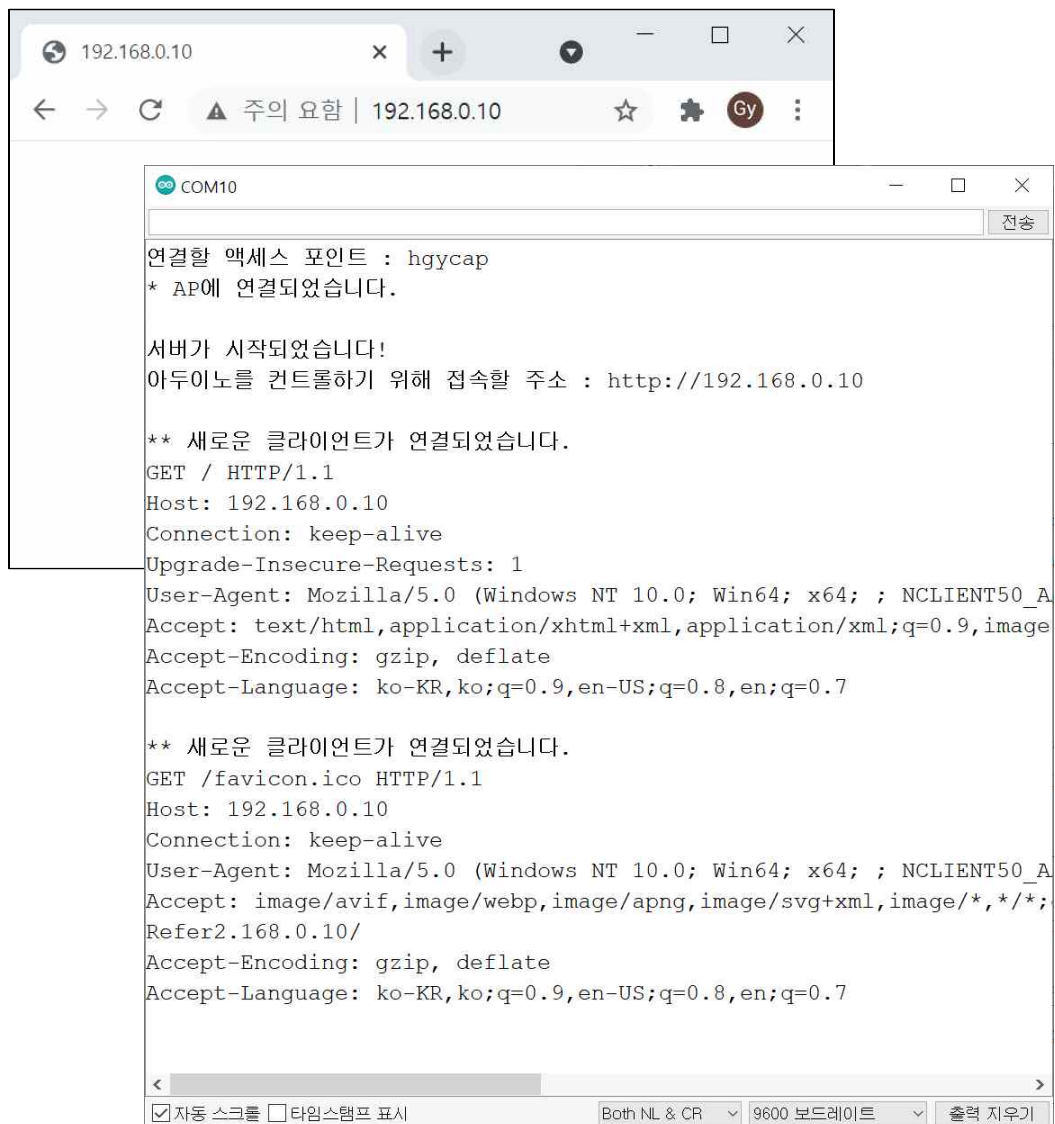
  // 서버 시작
  server.begin();
  Serial.println("서버가 시작되었습니다!");

  // DHCP를 통해 할당받은 IP 주소 출력
  IPAddress ip = WiFi.localIP();
  Serial.print("아두이노를 컨트롤하기 위해 접속할 주소 : http://");
  Serial.println(ip);
  Serial.println();
}
```



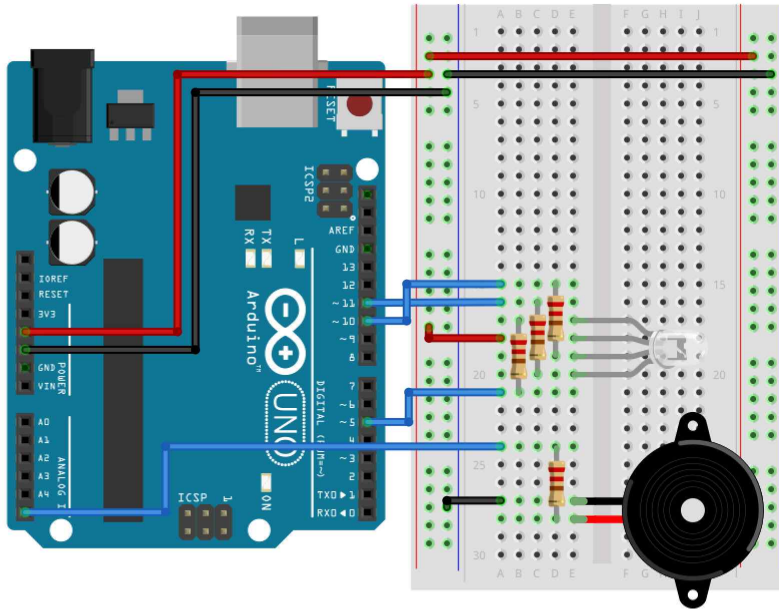
```
void loop() {  
    // 클라이언트 요청 검사  
    WiFiEspClient client = server.available();  
  
    // 클라이언트(브라우저)의 요청을 서버가 수신한 경우  
    if (client) {  
        Serial.println("*** 새로운 클라이언트가 연결되었습니다.");  
  
        // 클라이언트가 연결되어 있으면 줄 단위로 데이터를 수신  
        while (client.connected()) {  
            index = 0;  
  
            // do-while 루프를 사용하여 첫 번째 문자를 수신하여 저장하기 전에는  
            // 형식 검사가 이루어지지 않도록 함  
            do {  
                while (!client.available()); // 바이트 데이터 수신 대기  
                ch = client.read();          // 수신 데이터 읽기  
  
                incoming_line[index] = ch;  
                index = (index + 1) % MAX_LENGTH;  
            } while (ch != '\n');  
            incoming_line[index] = 0;  
  
            Serial.print(incoming_line);          // 줄 단위 데이터 출력  
  
            // 현재 수신한 줄이 '\r'과 '\n' 만으로 이루어진 빈 줄인 경우  
            // HTTP 요청 데이터 수신이 끝남  
            if (strlen(incoming_line) == 2 && incoming_line[1] == '\n') {  
                // HTTP 요청에 대해 코드 200을 사용하여 응답 전송  
                client.println("HTTP/1.1 200 OK");  
                client.println("Content-type:text/html");  
                client.println("Connection:close");  
                client.println();  
  
                // 현재 클라이언트와의 연결 종료  
                while(client.connected()) {
```

```
        client.flush();
        client.stop();
        delay(100);
    }
}
}
}
}
```



[그림 6] [코드 3] 실행 결과

[코드 3]에서 서버는 클라이언트로 아무런 데이터도 전송하지 않는다. [그림 7]과 같이 아두이노 우노에 RGB LED를 5, 10, 11번 핀에 연결하고 피에조 버저를 A5번 핀에 연결한 후 이를 웹 페이지를 통해 제어해 보자. [그림 7]에서 ESP-01 모듈 연결은 생략하였으며 [그림 2]와 같이 연결된 것으로 가정한다.



[그림 7] RGB LED와 피에조 버저 연결

웹 페이지는 [그림 8]과 같이 구성되며, HTML 파일은 [코드 4]와 같다.

#### 코드 4

#### HTML 페이지

```
<!DOCTYPE HTML>
<html>
<body>

<form action='' method='get'>
  <input type='hidden' name='L' value='5' />
  <input type='submit' value='Toggle Red' />
</form>

<form action='' method='get'>
  <input type='hidden' name='L' value='10' />
  <input type='submit' value='Toggle Green' />
</form>
```

```
</form>

<form action='' method='get'>
  <input type='hidden' name='L' value='11' />
  <input type='submit' value='Toggle Blue' />
</form>

<form action='' method='get'>
  <input type='range' name='S' min='0' max='1000' step='100' value='0' />
  <input type='submit' value='Set Frequency' />
</form>

</body>
</html>
```



[그림 8] 크롬에서 웹페이지 테스트

[코드 5]는 웹 페이지를 통해 RGB LED를 켜거나 끄고 피에조 버저로 재생하는 주파수를 설정하는 스케치다. 원격 제어를 위해 필요하지 않은 클라이언트 요청을 줄이기 위해 브라우저에서 favicon.ico을 요청하지 않도록 서버에서 전송하는 HTML 페이지의 헤더에 태그를 추가하였다.

#### 코드 5

#### 웹 서버에 의한 원격 제어

APPENDIX\_17-5\_web\_control\_server.ino

```
#include <AltSoftSerial.h>
#include <WiFiEsp.h>

AltSoftSerial ESP;                                     // 미리 정해진 핀 사용
```

```
// 와이파이 네트워크 정보 : 네트워크 이름(SSID)과 비밀번호
char WIFI_SSID[] = "네트워크 이름";
char WIFI_PASSWORD[] = "네트워크 비밀번호";

// 클라이언트 데이터 수신 버퍼 제어
const int MAX_LENGTH = 100;
char incoming_line[MAX_LENGTH + 1], ch;
int index;

// 와이파이 액세스 포인트와의 연결 상태 관리
int wifi_status = WL_IDLE_STATUS;

// HTML 폼을 통해 제어할 핀 번호
const int RED = 5;
const int GREEN = 10;
const int BLUE = 11;
const int SPEAKER = A5;

WiFiEspServer server(80);

void setup() {
    Serial.begin(9600);           // 컴퓨터와의 시리얼 통신 초기화
    ESP.begin(9600);             // 와이파이 모듈과의 시리얼 통신 초기화

    // LED 연결 핀 설정
    pinMode(RED, OUTPUT);
    digitalWrite(RED, HIGH);     // 공통 양극 방식 RGB LED로 HIGH를 출력하면 꺼짐
    pinMode(GREEN, OUTPUT);
    digitalWrite(GREEN, HIGH);   // 공통 양극 방식 RGB LED로 HIGH를 출력하면 꺼짐
    pinMode(BLUE, OUTPUT);
    digitalWrite(BLUE, HIGH);    // 공통 양극 방식 RGB LED로 HIGH를 출력하면 꺼짐

    WiFi.init(&ESP);            // ESP-01 모듈 초기화

    Serial.print("연결할 액세스 포인트 : ");
```

```
Serial.println(WIFI_SSID);

wifi_status = WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

if (wifi_status != WL_CONNECTED) {
    Serial.println("** AP에 연결할 수 없습니다.");
    while (1);
}
else {
    Serial.println("* AP에 연결되었습니다.");
    Serial.println();
}

// 서버 시작
server.begin();
Serial.println("서버가 시작되었습니다!");

// DHCP를 통해 할당받은 IP 주소 출력
IPAddress ip = WiFi.localIP();
Serial.print("아두이노를 컨트롤하기 위해 접속할 주소 : http://");
Serial.println(ip);
Serial.println();
}

void loop() {
    // 클라이언트 요청 검사
    WiFiEspClient client = server.available();

    // 클라이언트(브라우저)의 요청을 서버가 수신한 경우
    if (client) {
        String command = "";

        while (client.connected()) {
            index = 0;

            // do-while 루프를 사용하여 첫 번째 문자를 수신하여 저장하기 전에는
```

```
// 형식 검사가 이루어지지 않도록 함
do {
    while (!client.available()); // 바이트 데이터 수신 대기
    ch = client.read();          // 수신 데이터 읽기

    incoming_line[index] = ch;
    index = (index + 1) % MAX_LENGTH;
} while (ch != '\n');
incoming_line[index] = 0;

// GET에 의한 요청 수행
// "GET /?L=10 HTTP/1.1"과 같은 형식을 가지는 데이터 해석
if (strstr(incoming_line, "GET")) {
    Serial.print(incoming_line); // 'GET'으로 시작하는 문장만 출력

    // "L=10" 형식의 명령 분리
    for (int i = 6; ; i++) {
        if (incoming_line[i] == ' ' || incoming_line[i] == '\n') {
            break;
        }
        command += incoming_line[i];
    }
    Serial.println(String(" => 분리된 명령 : ") + command);
}

// 현재 수신한 줄이 'Wr'과 'Wn' 만으로 이루어진 빈 줄인 경우
// HTTP 요청 데이터 수신이 끝남
if (strlen(incoming_line) == 2 && incoming_line[1] == '\n') {
    // 완료된 요청에 대해 폼 페이지를 응답으로 전송
    // 응답 코드 200 : 요청을 정상적으로 수신하였음을 나타냄
    client.println(F("HTTP/1.1 200 OK"));
    client.println(F("Content-type:text/html"));
    client.println(F("Connection:close"));
    client.println();

    client.println(F("<!DOCTYPE HTML>")); // 데이터 시작
}
```

```
client.println(F("<html>"));

// favicon.ico 요청을 없애기 위해 추가
client.println(F("<head>"));
client.println(F("<link rel='icon' href='data:,'>"));
client.println(F("</head>"));

client.println(F("<body>"));

// 빨간색 LED 토글 버튼을 누른 경우
client.print(F("<form action='' method='get'>"));
client.print(F("<input type='hidden' name='L' value='5' />"));
client.print(F("<input type='submit' value='Toggle Red' />"));
client.print(F("</form>"));

// 초록색 LED 토글 버튼을 누른 경우
client.print(F("<form action='' method='get'>"));
client.print(F("<input type='hidden' name='L' value='10' />"));
client.print(F("<input type='submit' value='Toggle Green' />"));
client.print(F("</form>"));

// 파란색 LED 토글 버튼을 누른 경우
client.print(F("<form action='' method='get'>"));
client.print(F("<input type='hidden' name='L' value='11' />"));
client.print(F("<input type='submit' value='Toggle Blue' />"));
client.print(F("</form>"));

// 스피커를 위한 재생 주파수 조정 슬라이드바
client.print(F("<form action='' method='get'>"));
client.print(
  F("<input type='range' name='S' min='0' max='1000' step='100' val-
ue='0' />"));
client.print(F("<input type='submit' value='Set Frequency' />"));
client.print(F("</form>"));

// 다른 하드웨어 제어를 위해 다른 폼을 여기에 추가할 수 있음
```



```
client.println(F("</html>"));
client.println(F("</body>"));

// HTTP 응답의 끝을 나타내기 위한 빈 줄
client.println();

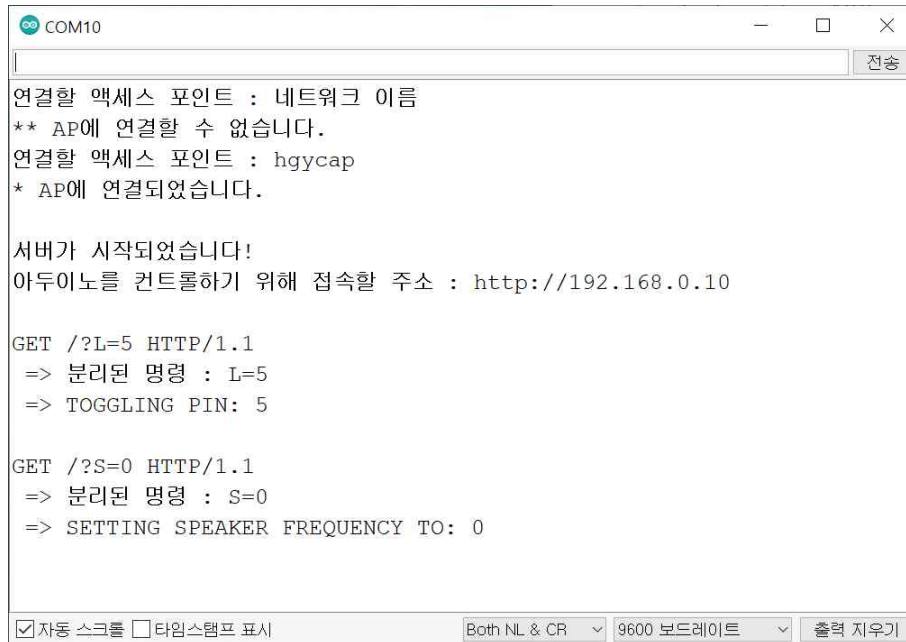
// 현재 클라이언트와의 연결 종료

while (client.connected()) {
    client.flush();                // 수신 버퍼 비우기
    client.stop();
    delay(100);
}

// 수신된 명령에 따라 하드웨어 제어
if (command.startsWith("L=")) {
    int led_pin = command.substring(2).toInt();
    Serial.print(" => TOGGING PIN: ");
    Serial.println(led_pin);
    Serial.println();
    digitalWrite(led_pin, !digitalRead(led_pin));
}
else if (command.startsWith("S=")) {
    int speaker_freq = command.substring(2).toInt();
    Serial.print(" => SETTING SPEAKER FREQUENCY TO: ");
    Serial.println(speaker_freq);
    Serial.println();
    if (speaker_freq == 0) noTone(SPEAKER);
    else tone(SPEAKER, speaker_freq);
}
else {
    Serial.println();
}

// 다른 명령 처리를 위해 'else if' 문을 추가할 수 있음
```

```
}  
}  
}  
}
```



```
COM10  
연결할 액세스 포인트 : 네트워크 이름  
** AP에 연결할 수 없습니다.  
연결할 액세스 포인트 : hgycap  
* AP에 연결되었습니다.  
  
서버가 시작되었습니다!  
아두이노를 컨트롤하기 위해 접속할 주소 : http://192.168.0.10  
  
GET /?L=5 HTTP/1.1  
=> 분리된 명령 : L=5  
=> TOGGING PIN: 5  
  
GET /?S=0 HTTP/1.1  
=> 분리된 명령 : S=0  
=> SETTING SPEAKER FREQUENCY TO: 0  
  
☒ 자동 스크롤 ☐ 타임스탬프 표시 Both NL & CR 9600 보드레이트 출력 지우기
```

[그림 9] [코드 5] 실행 결과

[코드 5]는 아두이노가 웹 서버로 동작하는 경우였다. 이제 아두이노가 웹 클라이언트로 동작하여 웹 API를 통해 날씨 서버에서 오늘의 날씨를 얻고 현재 기온을 출력하도록 해 보자.

날씨 서버에서 날씨를 얻어오는 형식은 다음과 같다. **units=metric**은 섭씨 온도를 의미하며, **q=Busan**은 날씨 정보를 요청할 도시의 이름을, **appid=\*\*\*\*\***는 날씨 정보 요청을 위해 사용한 API 키로 **openweathermap.org**에 가입하여 얻을 수 있다.

```
http://api.openweathermap.org/data/2.5/weather?units=metric&q=Busan&appid=*****
```

날씨 서버에서 얻을 수 있는 데이터는 JSON 형식이므로 필요한 정보를 얻어내기 위해 Arduino JSON 라이브러리를 사용할 수 있다. 하지만 날씨 서버에서 수신한 데이터를 모두 저장하고 이를 Arduino JSON 라이브러리로 처리하기에 아두이노 우노의 메모리는 충분하지 않다. 따라서 날씨 서버에서 수신하는 JSON 형식 데이터 중 일부만 저장하고 문자열 검색을 통해 온도 정보를 찾아낸다. 아두이노 메가2560을 사용하면 JSON 형식 데이터를 모두 저장하고 저장한 데이터를 Arduino JSON 라이브러리로 분석할 수 있다.

## 코드 6

## 웹에서 실시간 날씨 얻기

APPENDIX\_17-6\_web\_weather.ino

```
#include <AltSoftSerial.h>
#include <WiFiEsp.h>

AltSoftSerial ESP;                                // 미리 정해진 핀 사용

// 와이파이 네트워크 정보 : 네트워크 이름(SSID)와 비밀번호
char WIFI_SSID[] = "네트워크 이름";
char WIFI_PASSWORD[] = "네트워크 비밀번호";

// API 요청을 위한 상수
const char SERVER[] = "api.openweathermap.org";
const char HOST_STRING[] = "HOST: api.openweathermap.org";
const String API_KEY = "사용자의 API 키";
const String CITY = "Busan";                      // 원하는 도시 이름 지정

// 연결 상태를 나타내기 위해 내장 LED 사용
const int ONBOARD_LED = 13;

// 데이터 수신 버퍼 제어
const int MAX_LENGTH = 200;
char json[MAX_LENGTH + 1], ch;
int index;

// 아두이노는 클라이언트로 동작
WiFiEspClient client;

// 와이파이 액세스 포인트와의 연결 상태 관리
int wifi_status = WL_IDLE_STATUS;
```

```
void setup() {  
    Serial.begin(9600);           // 컴퓨터와의 시리얼 통신 초기화  
    ESP.begin(9600);             // 와이파이 모듈과의 시리얼 통신 초기화  
  
    WiFi.init(&ESP);             // ESP-01 모듈 초기화  
  
    Serial.print("연결할 액세스 포인트 : ");  
    Serial.println(WIFI_SSID);  
  
    wifi_status = WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  
  
    if (wifi_status != WL_CONNECTED) {  
        Serial.println(F("** AP에 연결할 수 없습니다."));  
        while (1);  
    }  
    else {  
        Serial.println(F("** AP에 연결되었습니다."));  
        Serial.println();  
    }  
  
    // API 요청 준비  
    String request = "GET /data/2.5/weather?units=metric&q=" +  
        CITY +  
        "&appid=" +  
        API_KEY +  
        " HTTP/1.1";  
  
    // 서버에 연결하고 API 요청 전송  
    if (client.connect(SERVER, 80)) {  
        Serial.println(F("Sending Request: "));  
        Serial.println(request);  
        Serial.println("");  
        client.println(request);  
        client.println(HOST_STRING);  
        client.println(F("Connection: close"));
```

```
        client.println();
    }

    // 서버 응답 대기
    while (!client.available());

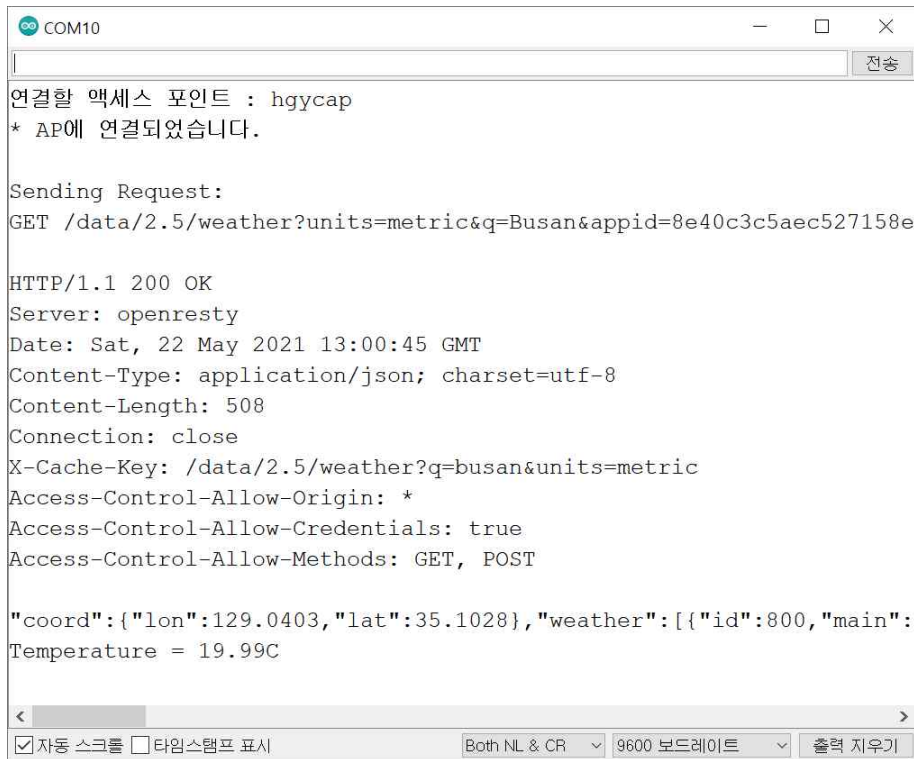
    // '{'로 시작되는 JSON 객체를 수신하기 전에 서버에서 수신한 헤더 데이터는
    // 사용하지는 않지만, 디버깅을 위해 헤더 데이터를 시리얼 모니터로 출력한다.
    while (true) {
        char h = client.read();
        if (h == '{') break;
        Serial.print(h);
    }

    // JSON 형식 데이터 수신이 시작되면 문자열로 저장
    index = 0;
    do {
        ch = client.read();
        if (index < MAX_LENGTH) {
            json[index] = ch;
            index++;
        }

        Serial.print(ch);
    } while (client.connected());
    json[index] = 0;

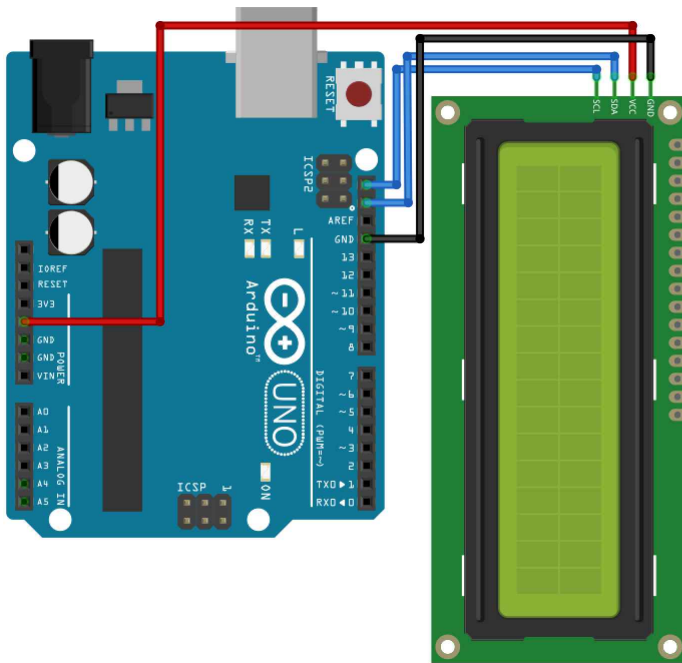
    char *ptr = strstr(json, "\"temp\":");
    Serial.print("\nTemperature = ");
    for (int i = 0; ; i++) {
        ch = ptr[7 + i];
        if (ch == ',') break;
        Serial.print(ch);
    }
    Serial.println("C");
}
```

```
void loop() {  
    // setup() 함수에서 한 번만 데이터를 얻어 표시하므로 loop() 함수는 비어 있다.  
}
```



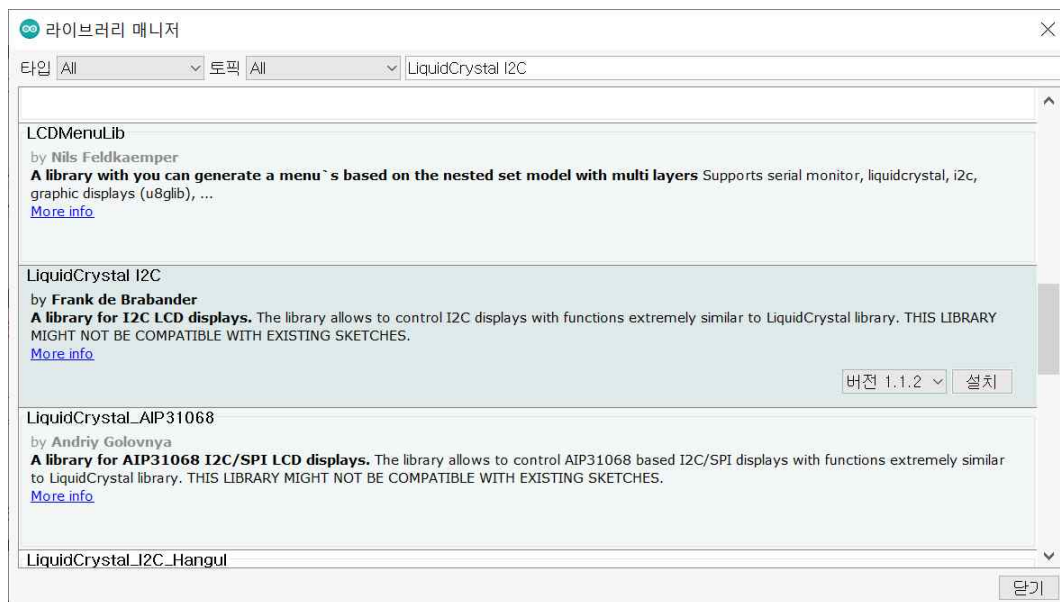
[그림 10] [코드 6] 실행 결과

[코드 6]은 날씨 서버에서 날씨 정보를 얻어와 시리얼 모니터로 출력하므로 항상 컴퓨터에 연결하여 사용해야 한다. 만약 아두이노에 별도의 출력 장치를 연결하여 사용한다면 자유롭게 온도 표시장치를 설치할 수 있으며 이를 위해 사용할 수 있는 것이 텍스트 LCD다. 이 장에서는 4개의 연결선만을 사용하는 I<sup>2</sup>C 방식 텍스트 LCD를 사용한다. 텍스트 LCD를 [그림 11]과 같이 연결한다. [그림 11]에서는 ESP-01 모듈을 표시하지는 않았지만 [그림 2]와 같이 연결된 것으로 가정한다.



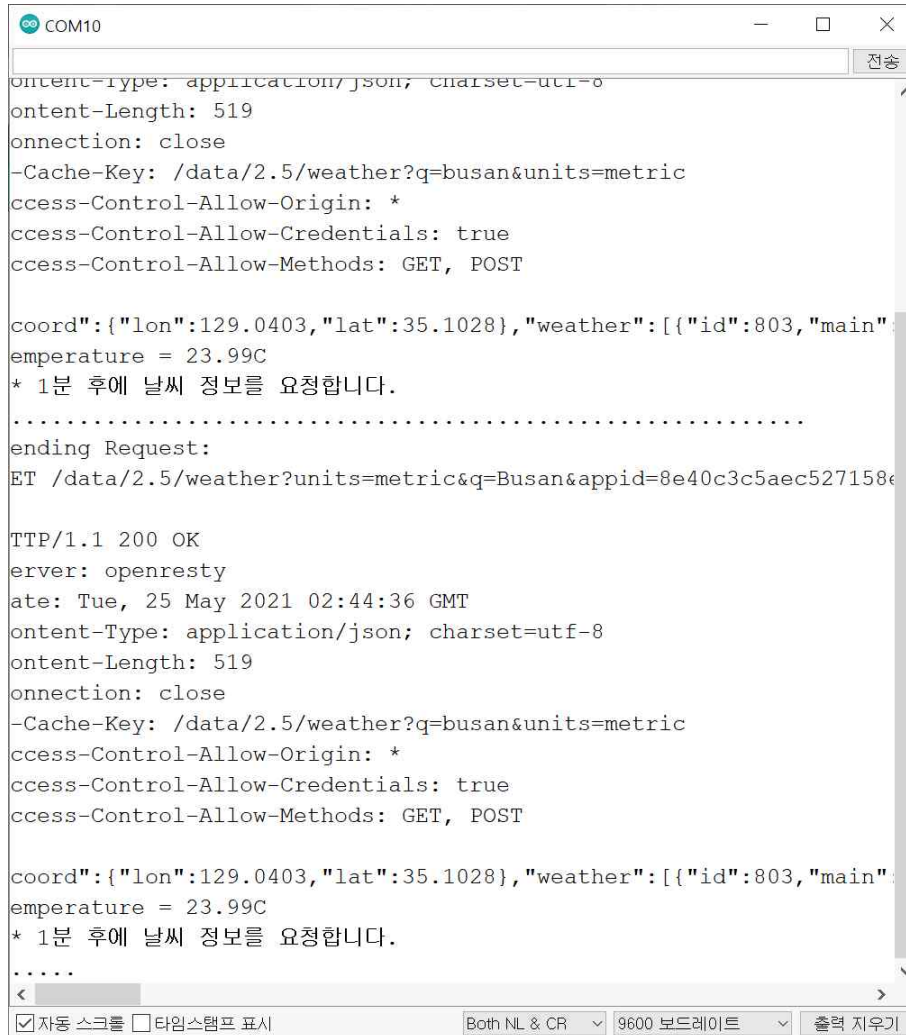
[그림 11] I<sup>2</sup>C 방식 텍스트 LCD 연결

아두이노의 LiquidCrystal 라이브러리는 I<sup>2</sup>C 통신을 지원하지 않으므로 별도로 라이브러리를 설치해야 한다. 라이브러리 매니저에서 ‘LiquidCrystal I2C’를 검색해서 LiquidCrystal I2C 라이브러리를 설치하자.



[그림 12] LiquidCrystal I2C 라이브러리 설치

LiquidCrystal I2C 라이브러리는 객체를 생성할 때 I<sup>2</sup>C 주소를 지정해야 하는 등의 차이를 제외하면 LiquidCrystal 라이브러리와 같은 방법으로 사용할 수 있다. [코드 7]은 I<sup>2</sup>C 방식 텍스트 LCD에 현재 기온을 출력하는 스케치로, [코드 6]에서 온도를 텍스트 LCD에 표시하는 부분과 1분 간격으로 반복해서 온도 정보를 얻어오도록 수정한 것이다.



```
COM10
Content-Type: application/json; charset=utf-8
Content-Length: 519
Connection: close
-Cache-Key: /data/2.5/weather?q=busan&units=metric
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST

coord":{"lon":129.0403,"lat":35.1028},"weather":[{"id":803,"main":
emperature = 23.99C
* 1분 후에 날씨 정보를 요청합니다.
.....
ending Request:
ET /data/2.5/weather?units=metric&q=Busan&appid=8e40c3c5aec527158e

HTTP/1.1 200 OK
Server: openresty
Date: Tue, 25 May 2021 02:44:36 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 519
Connection: close
-Cache-Key: /data/2.5/weather?q=busan&units=metric
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST

coord":{"lon":129.0403,"lat":35.1028},"weather":[{"id":803,"main":
emperature = 23.99C
* 1분 후에 날씨 정보를 요청합니다.
.....
```

[그림 13] [코드 7] 실행 결과 - 시리얼 모니터



[그림 14] [코드 7] 실행 결과 - 텍스트 LCD