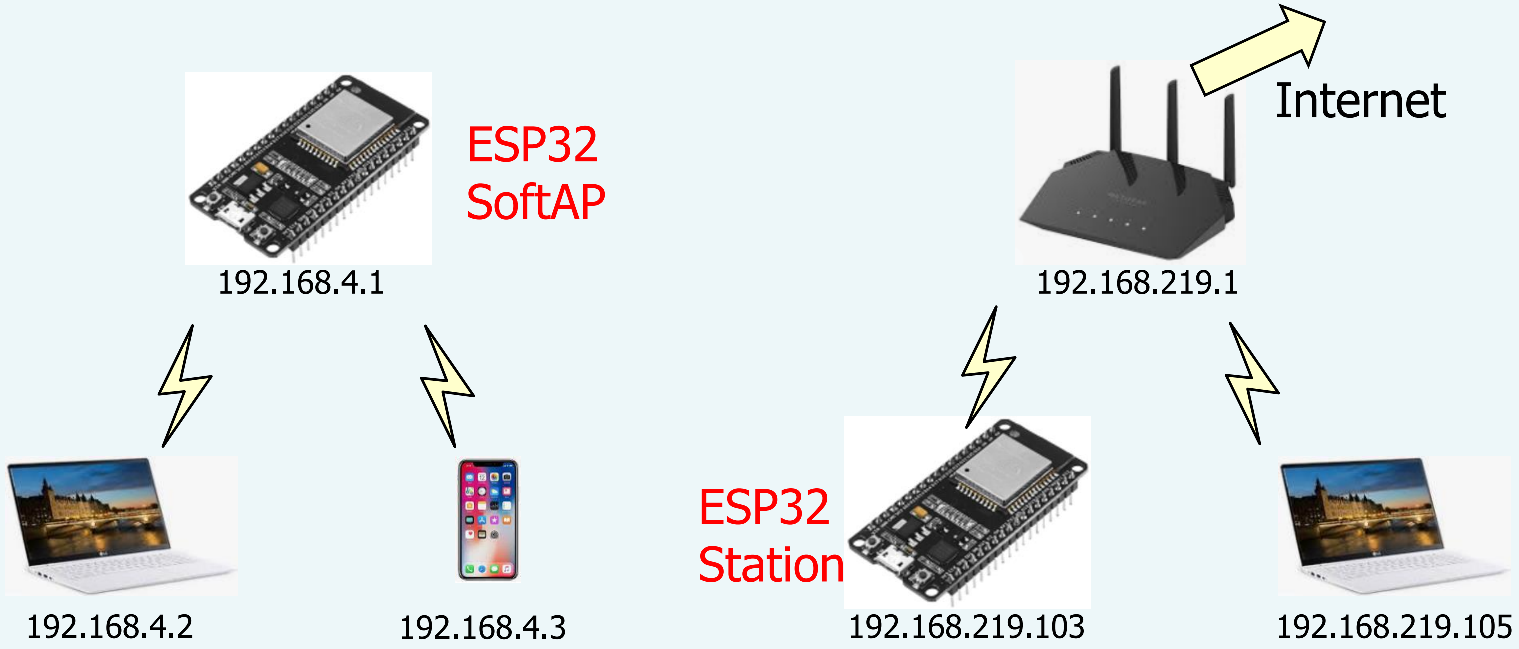
# Internet of Things
class 7

WiFi, Web Server, NTP

ESP32 WiFi

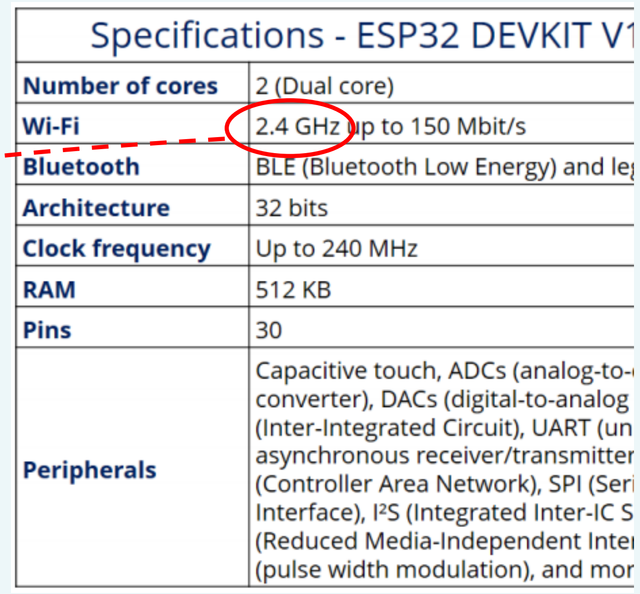
- SoftAP Mode, Station Mode



C.B.Choi gen1223@kau.ac.kr

ESP32 Web Server

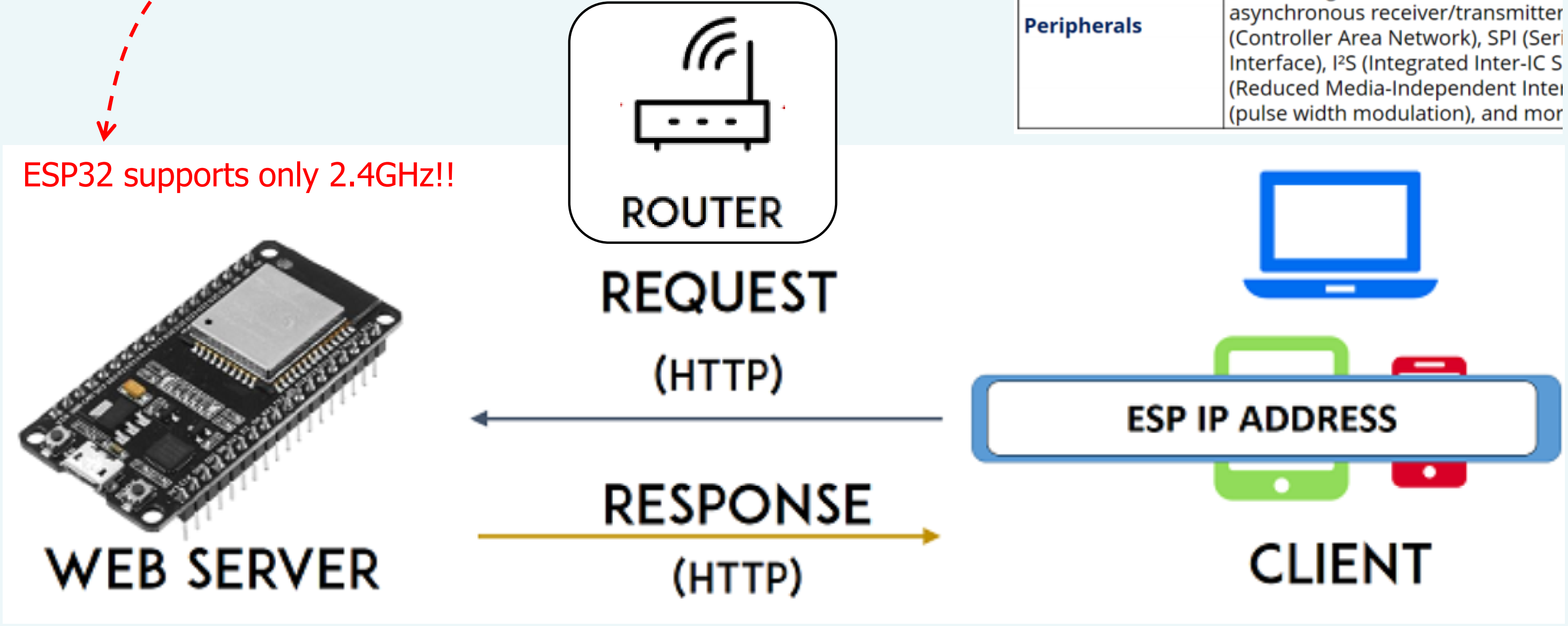
- Type ESP IP address in browser,
  - Sending an HTTP request to ESP32
  - ESP32 responds back with:
    - HTML text to display a web page
    - or Any data in ESP



Specifications - ESP32 DEVKIT V1

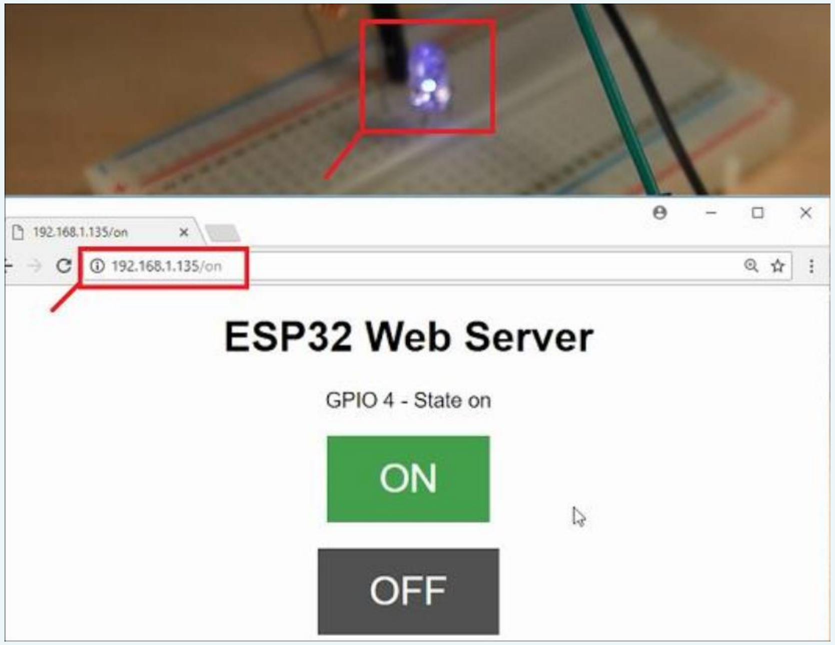
Number of cores	2 (Dual core)
Wi-Fi	2.4 GHz up to 150 Mbit/s
Bluetooth	BLE (Bluetooth Low Energy) and leg
Architecture	32 bits
Clock frequency	Up to 240 MHz
RAM	512 KB
Pins	30
Peripherals	Capacitive touch, ADCs (analog-to- converter), DACs (digital-to-analog (Inter-Integrated Circuit), UART (un asynchronous receiver/transmitter (Controller Area Network), SPI (Seri Interface), I²S (Integrated Inter-IC S (Reduced Media-Independent Inter (pulse width modulation), and mor

ESP32 supports only 2.4GHz!!



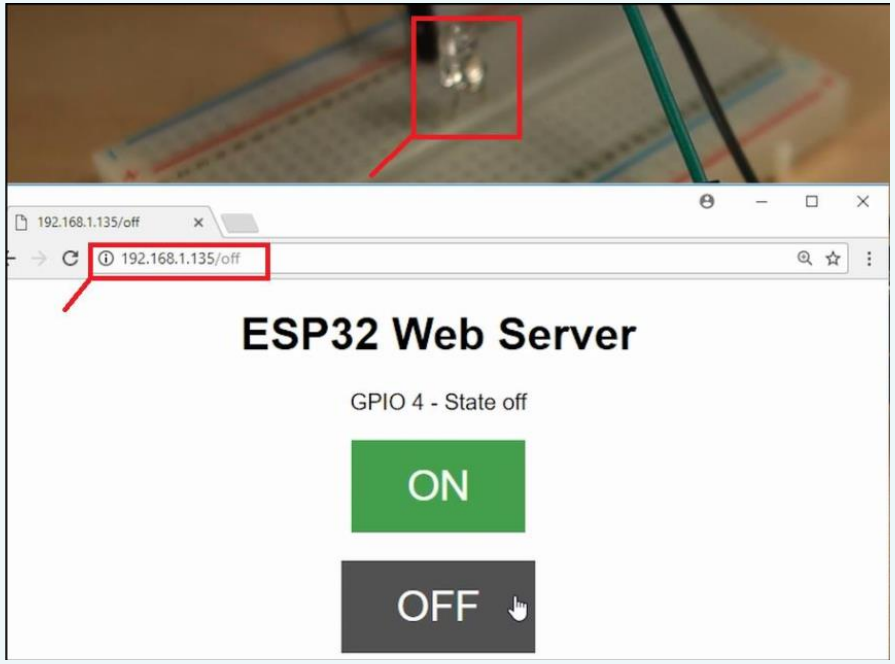
ESP32 Web Server

- Connect devices, and control them through the web



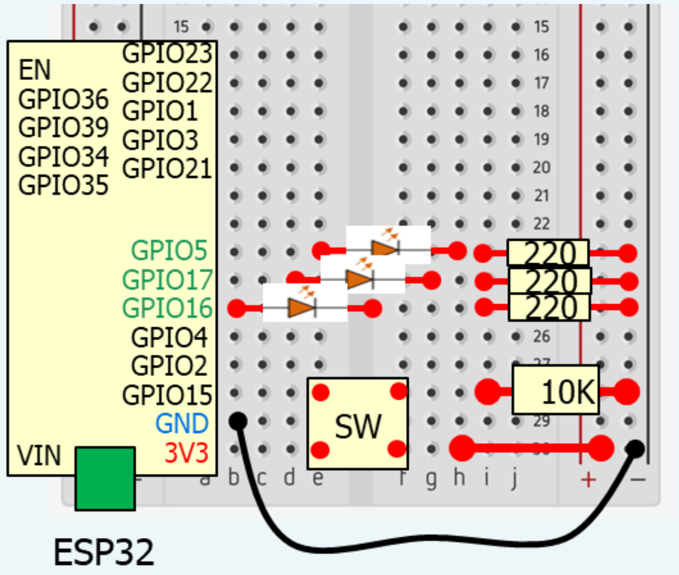
192.168.1.135/on

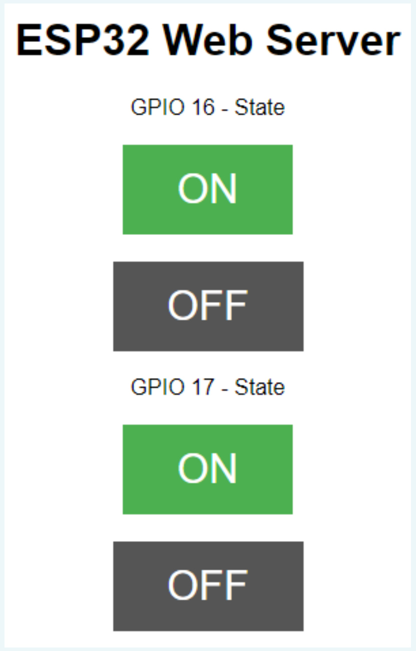
192.168.1.135/off



ESP32 Web Server

- <Task07-1> Control LEDs: GPIOs 16, 17
  - Can access Web server by typing the ESP32 IP address on a browser in the local network
  - Change LED state on Web server by clicking the buttons in web page




ESP32 Web Server

GPIO 16 - State

ON

OFF

GPIO 17 - State

ON

OFF

Web page

```
<!DOCTYPE html>
<html>
<head>
<title>ESP32 Web Server</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" href="data:,">
<style>
html {
font-family: Helvetica;
display: inline-block;
margin: 0px auto;
text-align: center;
}
.button {
background-color: #4CAF50;
border: none;
color: white;
padding: 16px 40px;
text-decoration: none;
font-size: 30px;
margin: 2px;
cursor: pointer;
}
```

Web page

```
.button2 {
background-color: #555555;
}
</style>
</head>
<body>
<h1>ESP32 Web Server</h1>
<p>GPIO 16 - State</p>
<p><a href="/16/on"><button class="button">ON</button></a></p>
<p><a href="/16/off"><button class="button button2">OFF</button></a></p>
<p>GPIO 17 - State</p>
<p><a href="/17/on"><button class="button">ON</button></a></p>
<p><a href="/17/off"><button class="button button2">OFF</button></a></p>
</body>
</html>
```

ESP32 Web Server

<Task07-1> Web Server controlling LEDs

```
// IoT07-1 ESP32 WebServer

#define SWAP 0      // sw access point

// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
#if SWAP
const char* ssid = "ESP32-AP";
const char* password = "123456789";     // password should be long!!
#else
const char* ssid = "KAU-Guest";
const char* password = "";
#endif

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;
// Auxiliar variables to store the current output state
String output16State = "off";
String output17State = "off";
// Assign output variables to GPIO pins
const int output16 = 16;
const int output17 = 17;
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
```

ESP32 Web Server

```
void setup() {
    Serial.begin(115200);
// Initialize the output variables as outputs
    pinMode(output16, OUTPUT);
    pinMode(output17, OUTPUT);
// Set outputs to LOW
    digitalWrite(output16, LOW);
    digitalWrite(output17, LOW);

#if SWAP
    WiFi.softAP(ssid, password);
    IPAddress IP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(IP);
#else
// Connect to Wi-Fi network with SSID and password
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

// Print local IP address and start web server
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
#endif

    server.begin();
}
```

ESP32 Web Server

```
void loop(){
  WiFiClient client = server.available(); // Listen for incoming clients
  if (client) {                    // If a new client connects,
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");  // print a message out in the serial port
    String currentLine = "";        // make a String to hold incoming data from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while the client's connected
      currentTime = millis();
      if (client.available()) {   // if there's bytes to read from the client,
        char c = client.read(); // read a byte, then
        Serial.write(c);        // print it out the serial monitor
        header += c;
        if (c == '\n') {        // if the byte is a newline character
        // if the current line is blank, you got two newline characters in a row.
        // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
          // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
          // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
          // turns the GPIOs on and off
            if (header.indexOf("GET /16/on") >= 0) {
              Serial.println("GPIO 16 on");
              output16State = "on";
              digitalWrite(output16, HIGH);
            } else if (header.indexOf("GET /16/off") >= 0) {
              Serial.println("GPIO 16 off");
              output16State = "off";
              digitalWrite(output16, LOW);
            } else if (header.indexOf("GET /17/on") >= 0) {
              Serial.println("GPIO 17 on");
              output17State = "on";
              digitalWrite(output17, HIGH);
```

ESP32 Web Server

```
} else if (header.indexOf("GET /17/off") >= 0) {
    Serial.println("GPIO 17 off");
    output17State = "off";
    digitalWrite(output17, LOW);
}

// Display the HTML web page
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");
// CSS to style the on/off buttons
// Feel free to change the background-color and font-size attributes to fit your preferences
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
client.println(".button { background-color: #4CAF50;border: none; color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
client.println(".button2 {background-color: #555555;}</style></head>");
// Web Page Heading
client.println("<body><h1>ESP32 Web Server</h1>");
// Display current state, and ON/OFF buttons for GPIO 16
client.println("<p>GPIO 16 - State " + output16State + "</p>");
// If the output16State is off, it displays the ON button
if (output16State=="off") {
    client.println("<p><a href=\"/16/on\"><button class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/16/off\"><button class=\"button button2\">OFF</button></a></p>");
}
// Display current state, and ON/OFF buttons for GPIO 17
client.println("<p>GPIO 17 - State " + output17State + "</p>");
// If the output17State is off, it displays the ON button
if (output17State=="off") {
    client.println("<p><a href=\"/17/on\"><button class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/17/off\"><button class=\"button button2\">OFF</button></a></p>");
}
client.println("</body></html>");
```

ESP32 Web Server

```
                // The HTTP response ends with another blank line
                client.println();
                // Break out of the while loop
                break;
            }  //** if (currentLine.length() == 0) {
            else { // if you got a newline, then clear currentLine
                currentLine = "";
            }
        }  //** if (c == '\n') {
        else if (c != '\r') { // if you got anything else but a carriage return character,
            currentLine += c;   // add it to the end of the currentLine
        }
      } //* if (client.available()) {
    } //** while

    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  } //** if (client) {
} //** loop() {
```

ESP32 Web Server

- 시리얼모니터 출력내용:

```
Connecting to myLGNet_Eagle_2
..
WiFi connected.
IP address:
192.168.123.108
```

클라이언트 주소창에 입력할 IP주소를 알려준다

```
New Client.
GET / HTTP/1.1
Host: 192.168.123.108
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_5 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/87.0.4280.77 Mobile/15E148 Safari/604.1
Accept-Language: ko-kr
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

클라이언트로부터 오는 처음 HTTP Request

```

Client disconnected.
```

연속한 두개의 Newline으로 Request 마지막임을 안다

```
New Client.
GET /16/on HTTP/1.1
Host: 192.168.123.108
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_5 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/87.0.4280.77 Mobile/15E148 Safari/604.1
Referer: http://192.168.123.108/
Accept-Language: ko-kr
Accept-Encoding: gzip, deflate
```

GPIO16 ON을 눌렀을때 오는 HTTP Request

```

GPIO 16 on
```

GPIO16 LED ON 명령을 수행한다

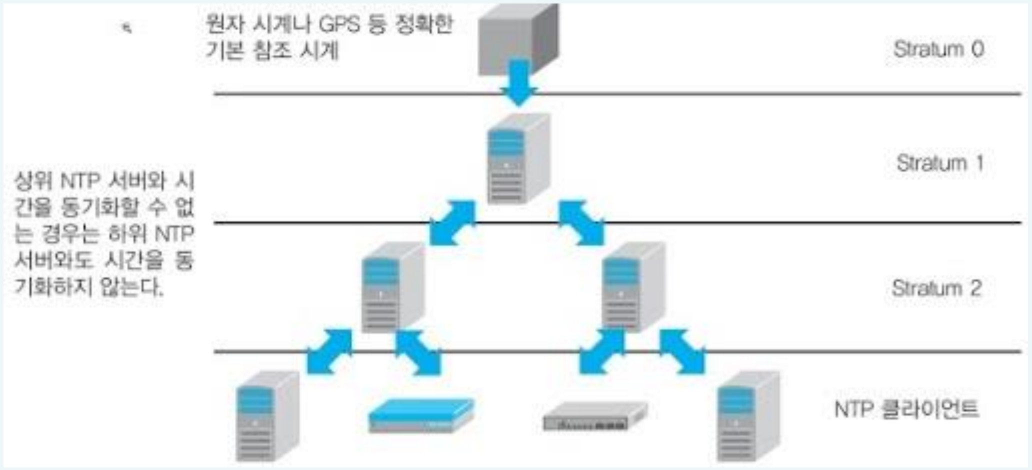
```
Client disconnected.
```

ESP32 Web Server- SW AP

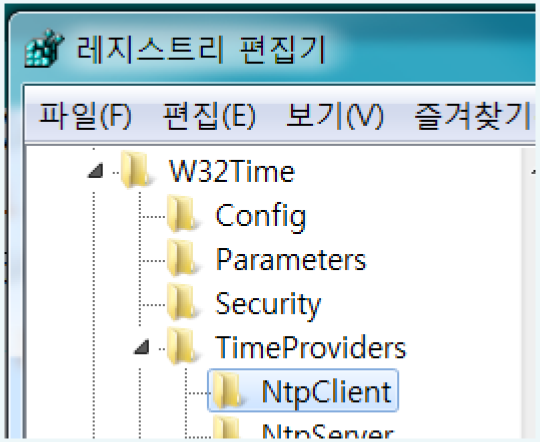
```
<Task07-2> Software Access Point
…
const char* ssid = "ESP32-AP";
const char* password = "123456789";     // password should be long!!
…
 WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);
  server.begin();
…
```

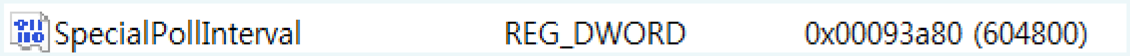
Network Time Protocol

- Clock synchronization between computer systems over packet-switched, variable-latency data networks
  - Intended to synchronize all participating computers within a few milliseconds of Coordinated Universal Time (UTC)
  - User Datagram Protocol (UDP) on port number 123


원자 시계나 GPS 등 정확한 기본 참조 시계
Stratum 0
Stratum 1
상위 NTP 서버와 시간을 동기화할 수 없는 경우는 하위 NTP 서버와도 시간을 동기화하지 않는다.
Stratum 2
NTP 클라이언트

[출처]m.blog.naver.com

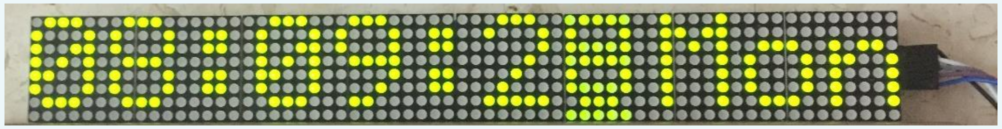


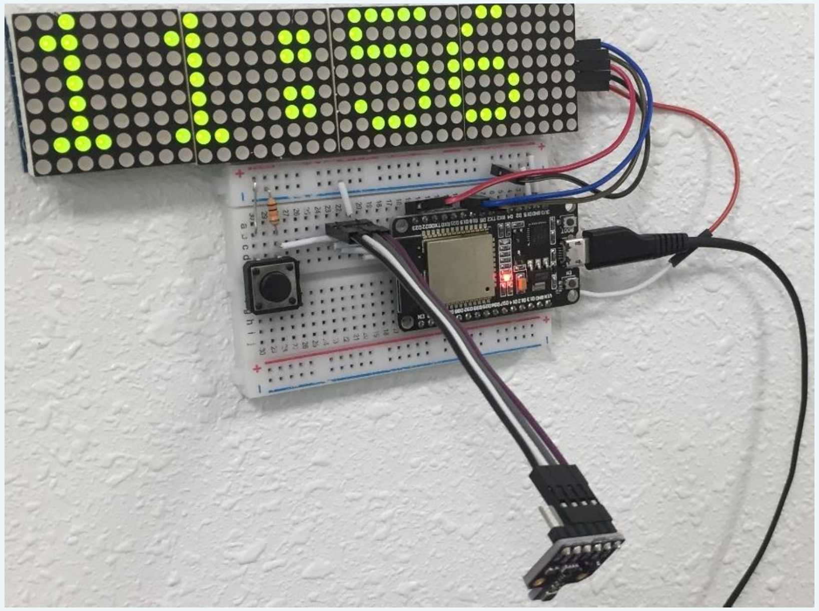


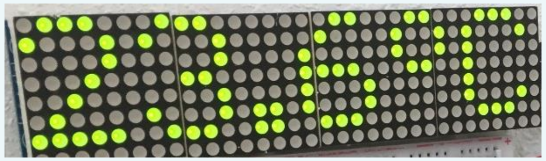
SpecialPollInterval REG_DWORD 0x00093a80 (604800)

Network Time Protocol- Example

- Dot Matirix Display NTP-Clock
  - Max7219, DHT11







ESP32 Web Server- SW AP

```
// IoT07-3 Network Time Protocol

#include <WiFi.h>
#include "time.h"
/* tm_year = s_yy - 1900;
 * tm_mon = s_MM - 1;
 * tm_mday = s_dd;
 * tm_hour = s_hh;
 * tm_min = s_mm;
 * tm_sec = s_ss;
 */

const char* ssid       = "KAU-Guest";
const char* password   = "";

const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600*9;   // 3600
const int   daylightOffset_sec = 0;   // 3600

void printLocalTime()
{
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
    Serial.println("Failed to obtain time");
    return;
  }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
  Serial.println("Year: " + String(timeinfo.tm_year+1900) + ", Month: " + String(timeinfo.tm_mon+1));
}
```

ESP32 Web Server- SW AP

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

  //connect to WiFi
  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }
  Serial.println(" CONNECTED");

  //init and get the time
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  printLocalTime();

  //disconnect WiFi as it's no longer needed
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
}

void loop()
{
  delay(1000);
  printLocalTime();
}
```