LABORATORY VIII – ESP32 AS A CLIENT

1. Introduction

The previous laboratory work introduced you to the ESP32 development board's most widely used mode of operation, the Wi-Fi access point. Other devices could connect to ESP32 as a client, to request and receive content by means of the HTTP protocol.

This laboratory work will cover another operating mode of the ESP32 board, the Client mode, and we'll study how two boards can exchange data.

2. Different modes of Wi-Fi operation

The development board supports three technical standards of wireless communication, namely IEEE 802.11 *b* (Wi-Fi 1 - 11 Mbps) / *g* (Wi-Fi 3 - 54 Mbps) / *n* (Wi-Fi 4 - 600 Mbps). Please note that these are theoretical data throughput values, and in practice you may get different results. It is possible to configure the development board to support only a certain standard. By default all three standards are in use, and, depending on the client, ESP32 will automatically switch to the newest supported version, similarly to a modern Wi-Fi router.

An ESP32 board can use Wi-Fi in two ways:

AP - Access Point mode, or in our case **SoftAP**, meaning **software enabled Access Point**, because the underlying mechanism is implemented via a piece of code running on a microcontroller, instead of it being hardwired. This was the mode used in the last laboratory work, where other devices such as your phones / laptops were able to connect to the development board as clients.

STA - Station mode, where the development board operates as a client, meaning it will connect to other access points.

Working in pairs / groups, you will create an access point using the example code from last week, and also program another development board to work as a client.

Don't forget to change the name of the SSID (and perhaps the password) to something unique before uploading your code to the development board. You can also add a password to the server, be sure that both the SSID and the password match on the server and the client.

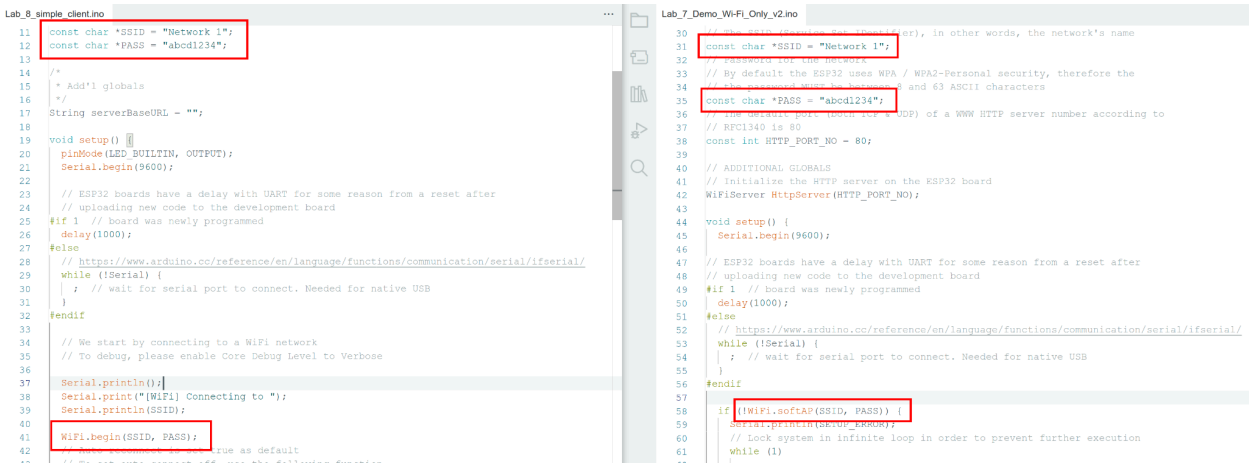

Figure 1. Make sure the SSID and password match on the client and the server

The example code for the client can be found below.

You can find the code as well at

https://github.com/UTCN-AC-CS-DMP/Lab-8-ESP32-Part-2/blob/main/Lab_8_simple_client.ino

```
#include <WiFi.h>

/*
* WiFi related globals
*/
const char *SSID = "Network 1";
const char *PASS = "abcd1234";

/*
* Additional globals
*/
String serverBaseURL = "";

void setup() {
 pinMode(LED_BUILTIN, OUTPUT);
 Serial.begin(9600);

 // ESP32 boards have a delay with UART for some reason from a reset after
 // uploading new code to the development board
#if 1  // board was newly programmed
 delay(1000);
#else
 // https://www.arduino.cc/reference/en/language/functions/communication/serial/ifserial/
 while (!Serial) {
   ;  // wait for the serial port to connect. Needed for native USB
 }
#endif

// We start by connecting to a WiFi network
```

```
  Serial.println();
  Serial.print("[WiFi] Connecting to ");
  Serial.println(SSID);

  WiFi.begin(SSID, PASS);
  // Auto reconnect is set true as default
  // To set auto connect off, use the following function
  //    WiFi.setAutoReconnect(false);

  // Will try for about 10 seconds (20x 500ms)
  const int tryDelay = 500;
  int numberOfTries = 20;

  // Wait for the WiFi event
  while (true) {

    switch (WiFi.status()) {
      case WL_NO_SSID_AVAIL:
        Serial.println("[WiFi] SSID not found");
        break;
      case WL_CONNECT_FAILED:
        Serial.print("[WiFi] Failed - WiFi not connected! Reason: ");
        return;
        break;
      case WL_CONNECTION_LOST:
        Serial.println("[WiFi] Connection was lost");
        break;
      case WL_SCAN_COMPLETED:
        Serial.println("[WiFi] Scan is completed");
        break;
      case WL_DISCONNECTED:
        Serial.println("[WiFi] WiFi is disconnected");
        break;
      case WL_CONNECTED:
        Serial.println("[WiFi] WiFi is connected!");
        Serial.print("[WiFi] IP address: ");
        Serial.println(WiFi.localIP());
        serverBaseURL = String("http://") + WiFi.gatewayIP().toString();
        Serial.println(String("[WiFi] Current Server IP: ") + WiFi.gatewayIP().toString());
        digitalWrite(LED_BUILTIN, HIGH);
        return;
      default:
        Serial.print("[WiFi] WiFi Status: ");
        Serial.println(WiFi.status());
        break;
    }
    delay(tryDelay);

    if (numberOfTries <= 0) {
      Serial.print("[WiFi] Failed to connect to WiFi!");
      // Use disconnect function to force stop trying to connect
      WiFi.disconnect();
      return;
    } else {
      numberOfTries--;
    }
  }
}

void loop() {}
```

For the code above, observe the debug messages from the serial monitor on both the client (station) and server (access point) side.

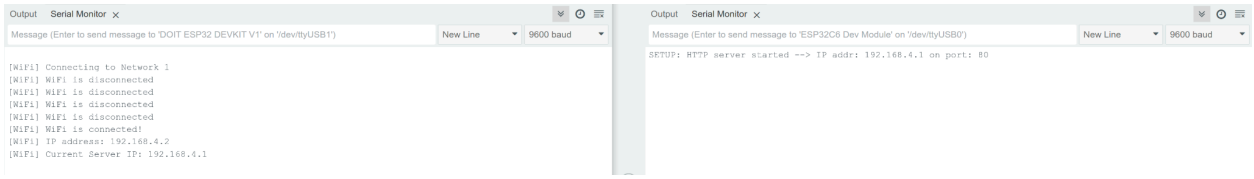

Figure 2. Output on the serial monitor from the client (left) and the server (right)

The LED on the client should also light up in case it has successfully connected to the server

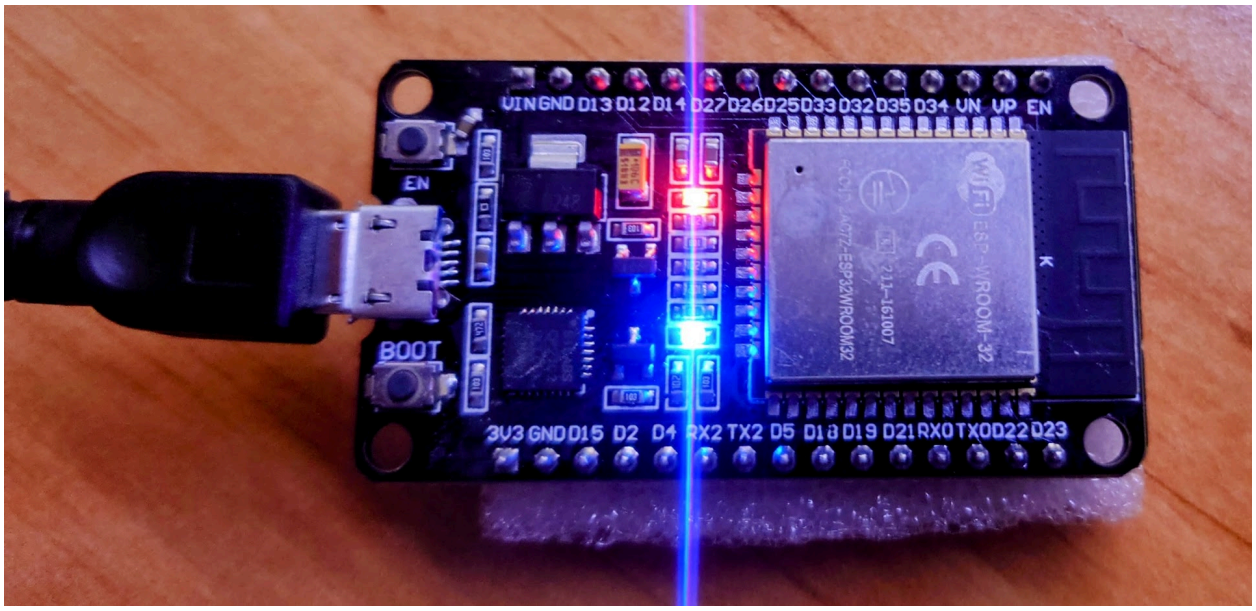

Figure 3. The blue LED on the client should light up in case it has successfully connected to the server

3. Wi-Fi communication between two ESP32 boards

The Wi-Fi access point provides the MAC (Medium Access Control) layer for the communication between two devices. On top of this layer, the network layer provides the **IP (Internet Protocol)** addressing. In the realm of TCP/IP networking, communication between processes is facilitated by the use of both an IP address and a port number. The IP address serves as a unique

identifier for a device or host on the network. It enables the routing of data packets from a source to a destination across the network. In the broader context of the internet, each device is assigned a distinct IP address.

Meanwhile, a **port number** designates a specific process or service running on a host. It permits multiple processes (programs, services) on the same host to utilize network resources concurrently. Port numbers range from 0 to 65535. Standard services, such as HTTP (web), are associated with well-known port numbers (e.g., 80), while other applications might use other port numbers.

When a client seeks to communicate with a server, it utilizes the server's IP address along with the port number linked to the particular service it intends to access on that server. This combination of IP address and port number creates a distinct endpoint commonly referred to as a **socket**.

While most Wi-Fi applications on ESP32 use the HTTP protocol, this application layer protocol is not needed for exchanging basic data between boards. The WiFiClient and the WiFiServer classes operate at the transport layer (TCP), and can be used to send and receive any type of data.

Below you can find a sample code for sending messages between two ESP32 boards, one being a server and another being a client.

Server code

The code is also available at:
https://github.com/UTCN-AC-CS-DMP/Lab-8-ESP32-Part-2/blob/main/Lab_8_tcp_server.ino

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

// MESSAGE STRINGS
const String SETUP_INIT = "SETUP: Initializing ESP32 dev board";
const String SETUP_ERROR = "!!ERROR!! SETUP: Unable to start SoftAP mode";
const String INFO_NEW_CLIENT = "New client connected";
const String INFO_DISCONNECT_CLIENT = "Client disconnected";

// BASIC WIFI CONFIGURATION CONSTANTS
// The SSID (Service Set IDentifier), in other words, the network's name
// Make sure to CHANGE THIS to something unique before uploading the code to
// the development board
const char *SSID = "DMP Server";
// Password for the network
// By default the ESP32 uses WPA / WPA2-Personal security, therefore the
// the password MUST be between 8 and 63 ASCII characters
const char *PASS = "123456789";
```

```
// The default port (both TCP & UDP) of a WWW HTTP server number according to
// RFC1340 is 80
// You can change the port to any value, but make sure the client knows it!
const int port = 80;

// Initialize the server on the ESP32 board
WiFiServer server(port);

void setup() {
 Serial.begin(9600);
 // ESP32 boards have a delay with UART for some reason from a reset after
 // uploading new code to the development board
#if 1  // board was newly programmed
 delay(1000);
#else
 while (!Serial) {
   ;  // wait for serial port to connect. Needed for native USB
 }
#endif

 if (!WiFi.softAP(SSID, PASS)) {
   Serial.println(SETUP_ERROR);
   // Lock system in infinite loop in order to prevent further execution
   while (1)
     ;
 }

 // Get AP's IP address for info message
 const IPAddress accessPointIP = WiFi.softAPIP();
 const String serverInfoMessage = "Server started " + accessPointIP.toString()
                                  + " on port " + port;

 // Start the server
 server.begin();
 Serial.println(serverInfoMessage);
}

void loop() {
 WiFiClient client = server.available();

 if (client) {  // Check if a device has connected to the AP
   Serial.println(INFO_NEW_CLIENT);
   String currentLine = "";
   while (client.connected()) {  // Loop until a client is connected to the TCP server
     if (client.available()) {  // If there's data in the buffer (received from the client)
       const char c = client.read(); // read it
       currentLine += c;             // add it to a string
       if (c == '\n') { // read characters until NEWLINE is received
         // No need for println, because it already has a new line at the end
         Serial.print(currentLine);
         // send a message to the client, including the received message
         client.print("Hello client, I received the following message: " + currentLine);
         currentLine = "";   // clear the reception string
         client.stop();  // disconnect the client
         Serial.println(INFO_DISCONNECT_CLIENT);
         Serial.println();
       }
     }
   }
 }
}
```

Client code

The code is also available at:
https://github.com/UTCN-AC-CS-DMP/Lab-8-ESP32-Part-2/blob/main/Lab_8_tcp_client.ino

```
#include <WiFi.h>

const char* ssid = "DMP Server";      // Change this to your WiFi SSID
const char* password = "123456789";  // Change this to your WiFi password

const char* host = "192.168.4.1";  // Default address of the server
const int port = 80;               // Default port. You can put any value, as long as
                                   // the client and server are the same
void setup() {
 Serial.begin(9600);
 // ESP32 boards have a delay with UART for some reason from a reset after
 // uploading new code to the development board
#if 1  // board was newly programmed
 delay(1000);
#else
 while (!Serial) {
   ;  // wait for serial port to connect. Needed for native USB
 }
#endif

 // We start by connecting to a WiFi network
 Serial.println("*****************************************************");
 Serial.print("Connecting to ");
 Serial.println(ssid);

 WiFi.begin(ssid, password);

 while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
 }

 Serial.println("");
 Serial.println("WiFi connected");
 Serial.print("IP address: ");
 Serial.println(WiFi.localIP());
}

void loop() {
 WiFiClient client;  // TCP client

 // Try to connect to the server at address "host" and port "port"
 if (!client.connect(host, port)) {
   Serial.println("Cannot connect");
   delay(2000);
   return;
 }

 String message = "Hello Server!\n";
```

```
  client.print(message);  // Trimite mesaj la server


  // wait for response
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }

  String line = "";
  while (client.available()) {
    // read server response, char by char
    char c = client.read();
    line = line + c;

    if (c == '\n')
      Serial.print(line);  // print the response to Serial
  }

  delay(2000);  // wait 2 seconds and start again
}
```

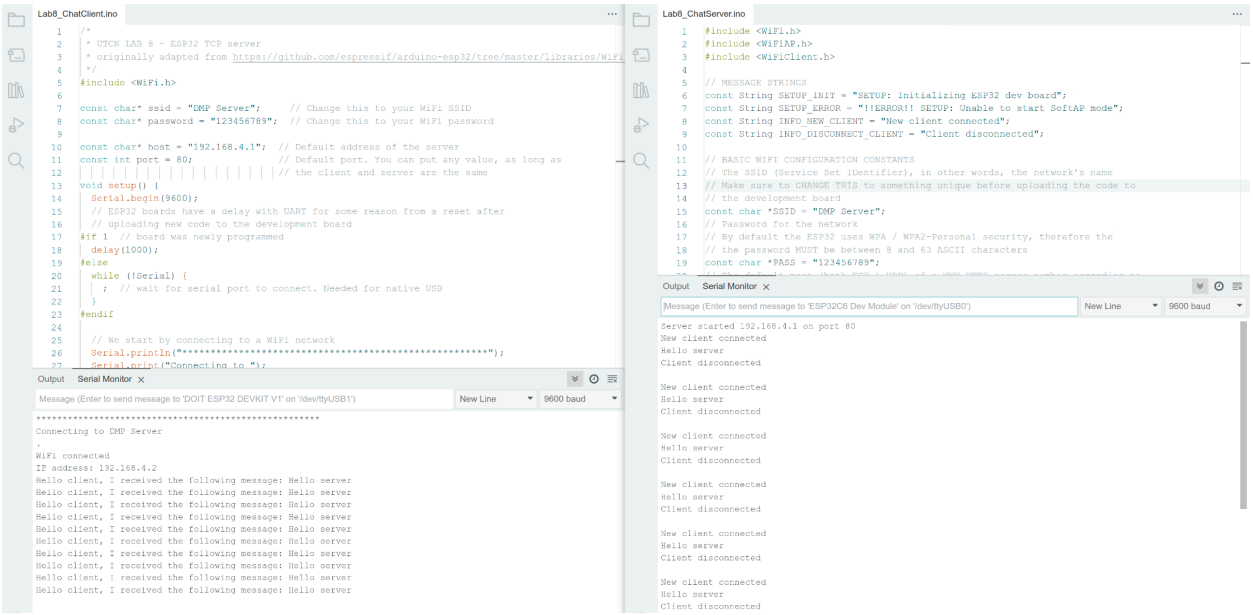

Figure 4. Messages sent between the client and the server

4. Individual work

1. Run the examples, analyze the code and the explanations.

2. Create a mechanism on a server (AP) that enables toggling the built-in LED of the ESP32 board ON or OFF, based on a character received from the client. On the client board (STA), read the BOOT button (digital pin 0 used as input) and use it to command the toggling of the Server's LED state.
3. Implement a chat program between two ESP32s. Each board is connected to a PC, and when the user types a message in its Serial Monitor, the message will appear in the Serial Monitor of the other PC, connected to the other board. The chat should be bi-directional.

Bibliography

1. Description of the ESP32 microcontroller https://www.espressif.com/en/products/socs/esp32

2. The ESP32 WiFi API https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html

3. TCP/IP addresses and ports
https://www.ibm.com/docs/en/cics-ts/5.4?topic=protocols-tcpip-internet-addresses-ports