ESP32 Arduino: Soft AP and Station modes

ESP32 / Leave a Comment

Contents [hide]

Introduction

In this post we are going to learn how to setup the ESP32 to work in Soft AP and Station modes simultaneously, using the Arduino core.

When working in station mode, the ESP32 is acting as a WiFi enabled device connected to an existing WiFi network. When working in Soft AP mode, the ESP32 is acting as host of a WiFi network, to which other devices can connect.

In the sections below, we will learn how to make the ESP32 act as both at the same time. We will start by a minimal example where we will check how to configure the ESP32 to behave this way, then we will check how to confirm this behavior with WiFi events and, to finalize, how to make a HTTP webserver operate over both interfaces and distinguish from which the requests come.

Working as Soft AP and Station simultaneously

We will start our code with the **WiFi.h** library include. This library will expose to us the **WiFi** extern variable, which we will use for both connecting the ESP32 to a WiFi network and also set it as a soft Access Point.

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

After this, we will define two pairs of credentials: one containing the SSID and password of the WiFi network to which the ESP32 will connect (acting as station), and another to hold the SSID and the password of the soft AP (defined by us).

Note that, in the first pair, I'm using placeholders for the network credentials. When testing the code, don't forget to replace those placeholders with the credentials of your WiFi network.

```
const char* wifi_network_ssid = "yourNetworkName";
const char* wifi_network_password =  "yourNetworkPassword";

const char *soft_ap_ssid = "MyESP32AP";
const char *soft_ap_password = "testpassword";
```

Moving on to the Arduino setup function, we will start by opening a serial connection, to later output some results from our program.

```
Serial.begin(115200);
```

Then, we will call the mode method on the **WiFi** extern variable, passing as input the enumerated value WIFI_MODE_APSTA (you can check the enum definition here). By doing this, we will set the ESP32 to work in soft AP and station modes simultaneously.

You can read more details about the different WiFi modes in IDF's documentation.

```
WiFi.mode(WIFI_MODE_APSTA);
```

Now that we have setup the ESP32 to work in both modes, we will start by setting its soft AP interface. For a detailed tutorial on how to setup the ESP32 to work on this mode, please check this previous tutorial.

In short, we only need to call the **softAP** method on the **WiFi** extern variable, passing as first input the name (SSID) we want to assign to the network and as second input its password. We will use the soft AP credentials we have defined before.

```
WiFi.softAP(soft_ap_ssid, soft_ap_password);
```

We will then connect the ESP32 to an existing WiFi network. This was already covered in detail on this post.

To do this, we will call the **begin** method, also on the **WiFi** variable, passing as input the network name and password (also defined before as global variables). After this method call, we will poll the status the connection, until it is established.

```
WiFi.begin(wifi_network_ssid, wifi_network_password);


while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
}
```

After this, we will print the IP address of the ESP32 soft AP interface, with a call to the **softAPIP** method.

```
Serial.print("ESP32 IP as soft AP: ");
Serial.println(WiFi.softAPIP());
```

To finalize, we will print the IP address assigned to the ESP32 after connected to the WiFi network, with a call to the **localIP** method on the WiFi extern variable.

```
Serial.print("ESP32 IP on the WiFi network: ");
Serial.println(WiFi.localIP());
```

The complete code can be seen below.

```
#include <WiFi.h>

const char* wifi_network_ssid = "yourNetworkName";
const char* wifi_network_password =  "yourNetworkPassword";

const char *soft_ap_ssid = "MyESP32AP";
const char *soft_ap_password = "testpassword";

void setup() {

  Serial.begin(115200);

  WiFi.mode(WIFI_MODE_APSTA);

  WiFi.softAP(soft_ap_ssid, soft_ap_password);
  WiFi.begin(wifi_network_ssid, wifi_network_password);


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

  Serial.print("ESP32 IP as soft AP: ");
  Serial.println(WiFi.softAPIP());

  Serial.print("ESP32 IP on the WiFi network: ");
  Serial.println(WiFi.localIP());
```

```
29
30  }
31
32  void loop() {}
```

As usual, to test this code, compile it and upload it using the Arduino IDE. Once the procedure is finished, open the Arduino IDE serial monitor.

You should see a result similar to figure 1, which shows the IP address of the ESP32 operating as soft AP and station. Note that the addresses printed in your case will most likely differ from the ones in the image.

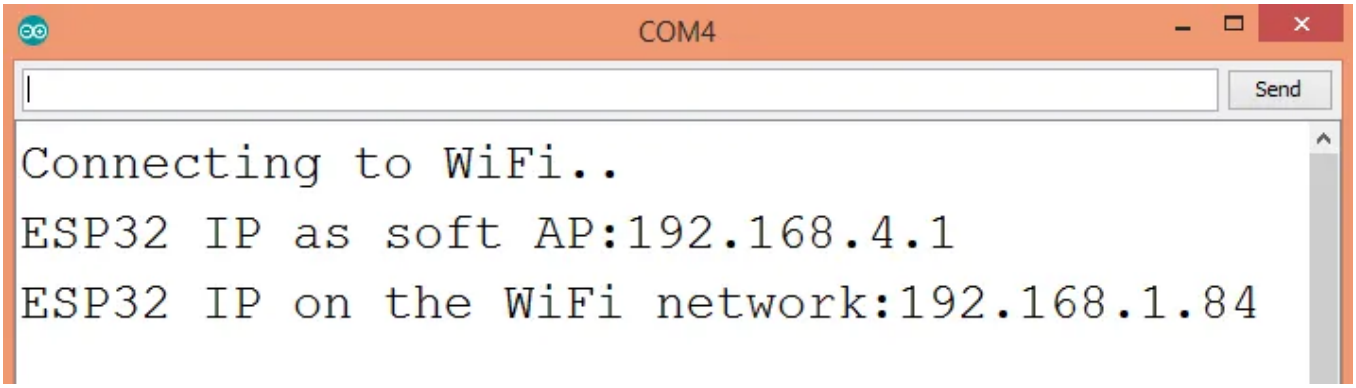

Figure 1 – Output of the program on the Arduino IDE serial monitor, showing the IP addresses of both the soft AP and station interfaces.

To confirm that everything is working as expected, we can ping the ESP32 on both the soft AP and station interfaces.

So, we will start by pinging the ESP32 from a machine connected to the same WiFi network as the ESP32 (in this case, we should use the second IP address printed to the monitor). To do so, just open the command line on that machine and send the following command:

```
1  ping your_ESP32_Station_IP
```

Figure 2 illustrates the result I got from pinging my ESP32 from a computer on the same network. As can be seen, the ESP32 was successfully reached, as expected.

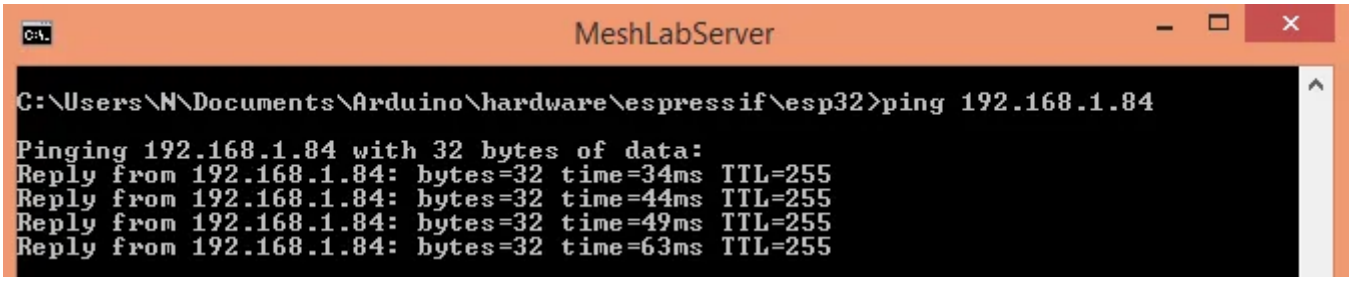

Figure 2 – Pinging the ESP32 station interface.

If you open the list of WiFi networks in your machine, you should see the ESP32 network we created, as shown in figure 3.

Figure 3 – ESP32 WiFi Network.

To further test our example, connect your machine to this network and then repeat the ping command, this time using the IP address of the ESP32 soft AP interface. You should also obtain a successful response, like shown in figure 4.

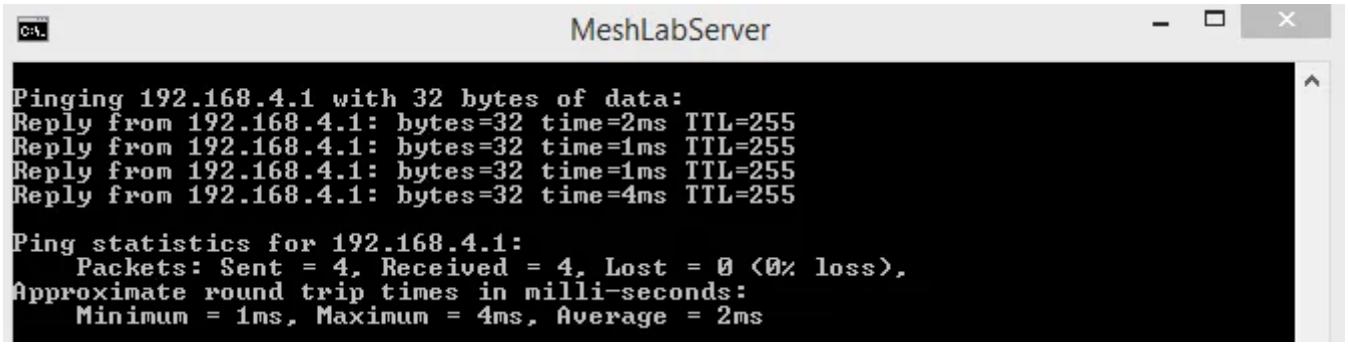

Figure 4 – Pinging the ESP32 soft AP interface.

An example with WiFi events

Now that we checked the basic example and how to do a simple test, we will further enhance it by using WiFi events (already covered here). Most of the code will be similar, so we will focus mostly on the event handling function.

But before we analyze the event handling function, note that we need to register it on the Arduino setup, right before we start any of the connections. We do so with a call to the **onEvent** method on the **WiFi** extern variable. As input, this method receives the callback function, which we will call **OnWiFiEvent**.

```
1 WiFi.onEvent(OnWiFiEvent);
```

This function should have a predefined signature: returning **void** and receiving as input the identifier of the event (you can check the enum with all the WiFi events here).

```
1 void OnWiFiEvent(WiFiEvent_t event)
2 {
3     // Implementation of the callback function
4 }
```

Since the list of possible events is quite extensive, we will just handle 4 events (in a switch case):

- **SYSTEM_EVENT_STA_CONNECTED**: ESP32 working as station connected to a WiFi network;
- **SYSTEM_EVENT_AP_START**: ESP32 soft AP started;
- **SYSTEM_EVENT_AP_STACONNECTED**: station connected to the ESP32 soft AP;
- **SYSTEM_EVENT_AP_STADISCONNECTED**: station disconnected to the ESP32 soft AP.

The complete callback can be seen below.

```
void OnWiFiEvent(WiFiEvent_t event)
{
  switch (event) {

    case SYSTEM_EVENT_STA_CONNECTED:
      Serial.println("ESP32 Connected to WiFi Network");
      break;
    case SYSTEM_EVENT_AP_START:
      Serial.println("ESP32 soft AP started");
      break;
    case SYSTEM_EVENT_AP_STACONNECTED:
      Serial.println("Station connected to ESP32 soft AP");
      break;
    case SYSTEM_EVENT_AP_STADISCONNECTED:
      Serial.println("Station disconnected from ESP32 soft AP");
      break;
    default: break;
  }
}
```

The complete code is available on the snippet below.

```
#include <WiFi.h>

const char* wifi_network_ssid = "yourNetworkName";
const char* wifi_network_password =  "yourNetworkPassword";

const char *soft_ap_ssid = "MyESP32AP";
const char *soft_ap_password = "testpassword";

void OnWiFiEvent(WiFiEvent_t event)
{
  switch (event) {

    case SYSTEM_EVENT_STA_CONNECTED:
      Serial.println("ESP32 Connected to WiFi Network");
      break;
    case SYSTEM_EVENT_AP_START:
      Serial.println("ESP32 soft AP started");
      break;
    case SYSTEM_EVENT_AP_STACONNECTED:
      Serial.println("Station connected to ESP32 soft AP");
      break;
    case SYSTEM_EVENT_AP_STADISCONNECTED:
      Serial.println("Station disconnected from ESP32 soft AP");
      break;
    default: break;
  }
}


void setup() {
```

```
31
32      Serial.begin(115200);
33      WiFi.onEvent(OnWiFiEvent);
34
35      WiFi.mode(WIFI_MODE_APSTA);
36
37      WiFi.softAP(soft_ap_ssid, soft_ap_password);
38      WiFi.begin(wifi_network_ssid, wifi_network_password);
39  }
40
41  void loop() {}
```

Once again, simply compile and upload the code to your ESP32 and open the IDE serial monitor. After both soft AP and station interfaces are connected, you should see the messages shown in figure 5 (at the time of writing, there's an issue that makes the Station connected event to fire twice, which might be related with this issue).

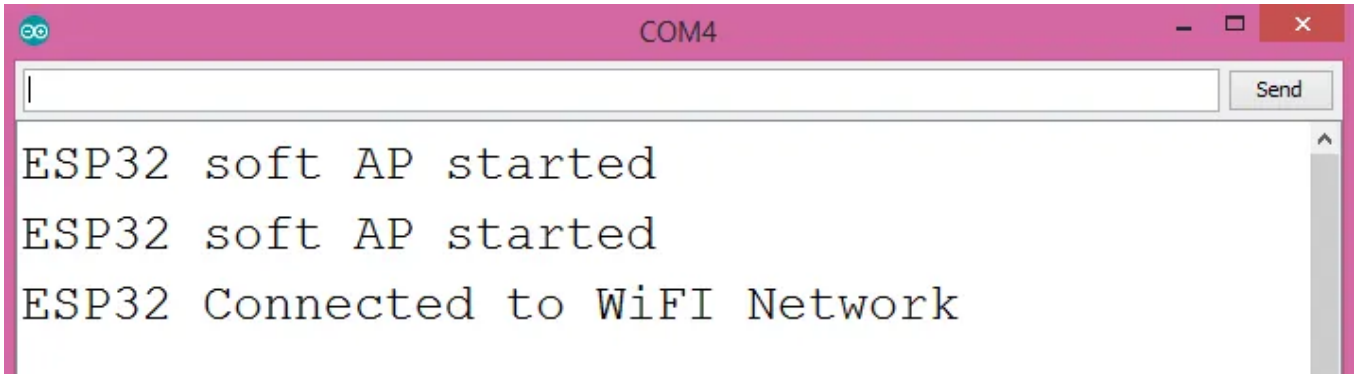

Figure 5 – Output of the two connection events in the Arduino IDE serial monitor.

If you then connect and disconnect a computer from the ESP32 soft AP, you should get the remaining messages, as shown in figure 6.

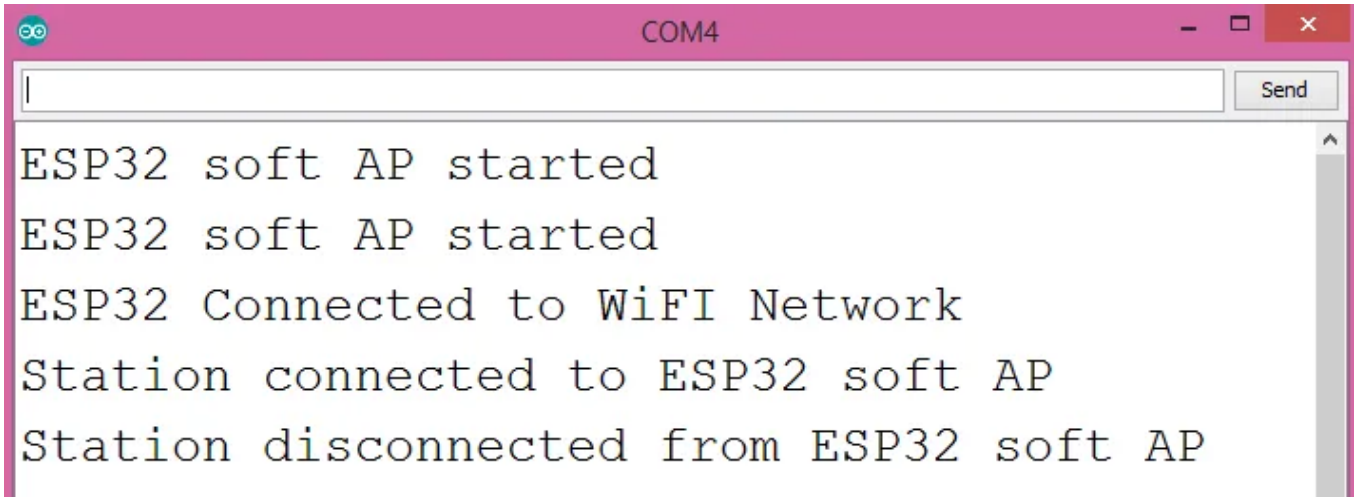

Figure 6 – Station connected / disconnected from the ESP32 soft AP.

A HTTP server over both interfaces

As a final example, we will check how to setup a HTTP webserver over both the soft AP and station interfaces. We will be using the Async HTTP web server library for the ESP32, which you can check in

detail how to get started on this tutorial. Note that the code to setup both the soft AP and station WiFi interfaces is exactly the same as before.

This time, we will need an additional library include: the **ESPAsyncWebServer.h**, which exposes to us the functions we need to setup the HTTP web server. Like before, we also need the **WiFi.h** lib.

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

We will still define the credentials of both WiFi networks as global variables.

```
const char* wifi_network_ssid = "yourNetworkName";
const char* wifi_network_password =  "yourNetworkPass";

const char *soft_ap_ssid = "MyESP32AP";
const char *soft_ap_password = "testpassword";
```

We will also need to create an object of class AsyncWebServer, which we will use to set up our HTTP server on the ESP32. As input of the constructor, this class receives the port where the server will be listening for incoming requests. We will use port **80**, which corresponds to the default HTTP port.

```
AsyncWebServer server(80);
```

Moving on to the Arduino setup function, the first part will be the same we have already done in the previous sections: initializing both the soft AP and station interfaces. Note that we will need the IP address of both, to be able to reach the HTTP server from a web browser.

```
Serial.begin(115200);
WiFi.mode(WIFI_MODE_APSTA);

WiFi.softAP(soft_ap_ssid, soft_ap_password);

WiFi.begin(wifi_network_ssid, wifi_network_password);


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

Serial.print("ESP32 IP as soft AP: ");
Serial.println(WiFi.softAPIP());

Serial.print("ESP32 IP on the WiFi network: ");
Serial.println(WiFi.localIP());
```

Then we will register a route called "**/hello**" in our server, which answers to HTTP GET requests. We register a route with a call to the **on** method on our **server** object. As inputs we pass:

- The name of the route;
- The method it supports;

- A callback function that will be executed every time a request is received on that route. Recall from previous tutorials that the callback function receives as input a pointer to an object of type **AsyncWebServerRequest**.

We will define our callback function using the C++ lambda function syntax.

```
server.on("/hello", HTTP_GET, [](AsyncWebServerRequest * request) {
   // Callback function implementation
});
```

Inside our callback function, we will determine if the request came from the soft AP or station interface, like shown here, and print a different message depending on the interface.

We can understand what is the interface by calling either the ON_STA_FILTER or the ON_AP_FILTER functions, passing as input the request object pointer. These functions return true if the request came from the station or soft AP interfaces, respectively.

```
if (ON_STA_FILTER(request)) {
      request->send(200, "text/plain", "Hello from STA");
      return;

} else if (ON_AP_FILTER(request)) {
      request->send(200, "text/plain", "Hello from AP");
      return;
}

request->send(200, "text/plain", "Hello from undefined");
```

The complete callback register can be seen below.

```
server.on("/hello", HTTP_GET, [](AsyncWebServerRequest * request) {

   if (ON_STA_FILTER(request)) {
      request->send(200, "text/plain", "Hello from STA");
      return;

   } else if (ON_AP_FILTER(request)) {
      request->send(200, "text/plain", "Hello from AP");
      return;
   }

   request->send(200, "text/plain", "Hello from undefined");
});
```

To finish the Arduino setup, we need to call the **begin** method on our server object, so it starts listening to incoming requests.

```
server.begin();
```

The whole setup function can be seen below.

```
void setup() {

```

```
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_APSTA);

  WiFi.softAP(soft_ap_ssid, soft_ap_password);

  WiFi.begin(wifi_network_ssid, wifi_network_password);


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

  Serial.print("ESP32 IP as soft AP: ");
  Serial.println(WiFi.softAPIP());

  Serial.print("ESP32 IP on the WiFi network: ");
  Serial.println(WiFi.localIP());


  server.on("/hello", HTTP_GET, [](AsyncWebServerRequest * request) {

    if (ON_STA_FILTER(request)) {
      request->send(200, "text/plain", "Hello from STA");
      return;

    } else if (ON_AP_FILTER(request)) {
      request->send(200, "text/plain", "Hello from AP");
      return;
    }

    request->send(200, "text/plain", "Hello from undefined");
  });

  server.begin();

}
```

The final code can be seen below.

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

const char* wifi_network_ssid = "yourNetworkName";
const char* wifi_network_password =  "yourNetworkPass";

const char *soft_ap_ssid = "MyESP32AP";
const char *soft_ap_password = "testpassword";

AsyncWebServer server(80);


void setup() {

  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_APSTA);

  WiFi.softAP(soft_ap_ssid, soft_ap_password);

  WiFi.begin(wifi_network_ssid, wifi_network_password);


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

```

```
    Serial.print("ESP32 IP as soft AP: ");
    Serial.println(WiFi.softAPIP());

    Serial.print("ESP32 IP on the WiFi network: ");
    Serial.println(WiFi.localIP());


    server.on("/hello", HTTP_GET, [](AsyncWebServerRequest * request) {

      if (ON_STA_FILTER(request)) {
        request->send(200, "text/plain", "Hello from STA");
        return;

      } else if (ON_AP_FILTER(request)) {
        request->send(200, "text/plain", "Hello from AP");
        return;
      }

      request->send(200, "text/plain", "Hello from undefined");
    });

    server.begin();

}

void loop() {}
```

To test the code, once again compile it and upload it to your ESP32 and open the Arduino IDE serial monitor when the procedure finishes.

Like on the first code sample, you should get the IP addresses of both the soft AP and station interfaces printed to the console. Copy the second one, which corresponds to the local address of the ESP32 acting as station, connected to a WiFi network.

On a computer connected to the same WiFi network as the ESP32, open a web browser and type the following URL:

```
http://your_ESP_IP/hello
```

You should get a result similar to figure 7. As can be seen, the message indicates that the interface was STA, which corresponds to the ESP32 acting as station, as expected.

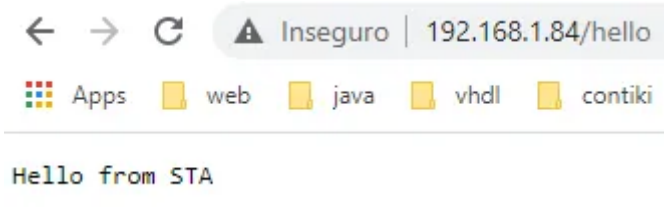

Figure 7 – Answer from the HTTP server, when the request comes from the ESP32 station interface.

To finalize the test, now copy the soft AP IP address of the ESP32. Then, connect the computer to the ESP32 hosted network and access the same URL as before, but this time using the soft AP IP address. You should now get a difference message, like figure 8.

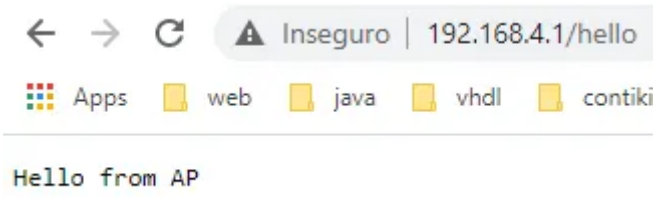

Figure 8 – Answer from the HTTP server, when the request comes from the ESP32 soft AP interface.

As could be seen, the ESP32 HTTP server acted as expected, allowing us to distinguish requests coming from the soft AP and station interfaces.

Suggested ESP32 readings

- How to connect to a WiFi network
- How to setup a soft AP
- Getting started with WiFi events
- How to setup a HTTP server
- Station got IP address event
- How to get WiFi station interface MAC address
- How to get WiFi Soft AP interface MAC address

← Previous Post

Next Post →

Leave a Reply

C# (9)

Electronics (8)

ESP32 (372)

ESP8266 (99)

IoT (3)

Javascript (23)

LinkIt Smart (14)

Micro:bit (30)

Microcontrollers (7)

Misc (24)

OBLOQ (15)

Python (67)

Raspberry Pi (15)

Sipeed M1 (4)

SQL (5)

Type your email...

Subscribe