Chapter 3: More ESP8266 Functions

204335 : Microcontroller and IoT

Part: ESP8266

Suphakit Awiphan

Chiang Mai University

Chapter 3: More ESP8266 Functions

204335 Microcontroller and IoT

Chapter 3 – More ESP8266 Functions

Parts You'll Need for This Chapter

- ESP8266 board
- USB cable
- DHT22 temperature/humidity sensor
- 10 kΩ resistor
- Breadboard
- Jumper wires

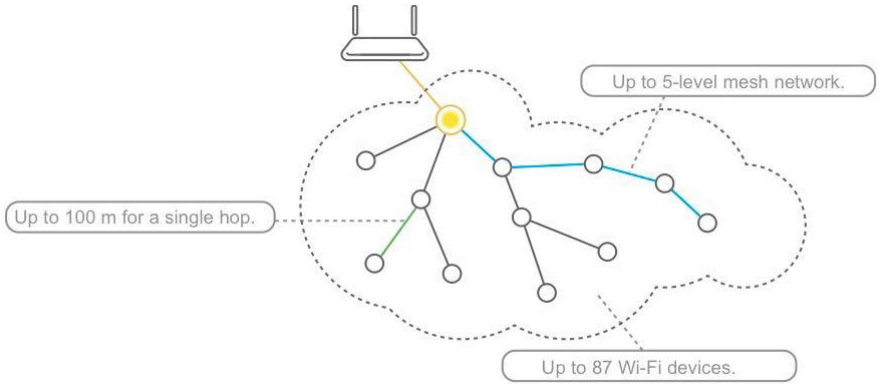
Outline

- **Discovering Advanced Functions of ESP8266**
- **Using Libraries on ESP8266**
- **Discovering Filesystem of ESP8266**
- **Storing Data in ESP8266**
- **Discovering OTA Update of ESP8266**
- **Programming Your ESP8266 OTA**

3.1 Discovering Advanced Functions of ESP8266

Wi-Fi Connectivity

- ESP8266 has a Radio Frequency (RF) transceiver
- RF transceiver supports 14 channels within the 2.4 GHz band in accordance to the IEEE802.11 b/g/n standard
- Wi-Fi connectivity allows ESP8266 to be used in IoT projects and to form mesh networks
- ESP8266 module can either be a web server or a Wi-Fi client



Real-Time Clock (RTC) & Over The Air (OTA) Update

RTC

- ESP8266 module has an RTC counter derived from the internal clock
- RTC returns a 32-bit counter value that you can use to track time
- Since it is a 32-bit counter, the RTC overflows after 7:45h

OTA

- Over the air update refers to delivering new configurations or software to a remote device via wireless
- You can upload programs onto your ESP8266 from a remote location via Wi-Fi

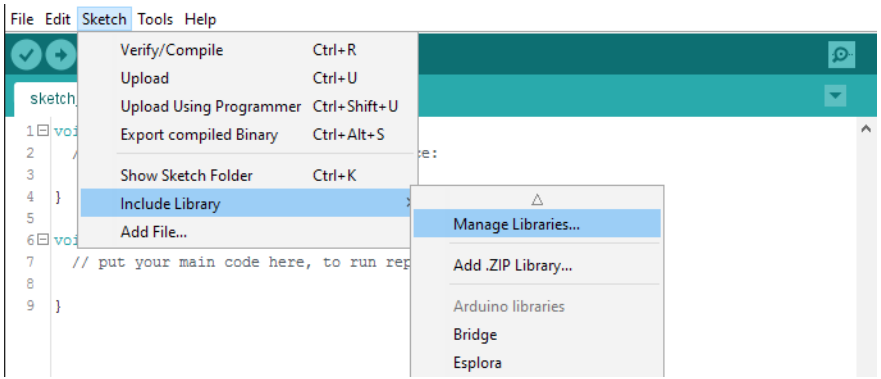
Low Power Management

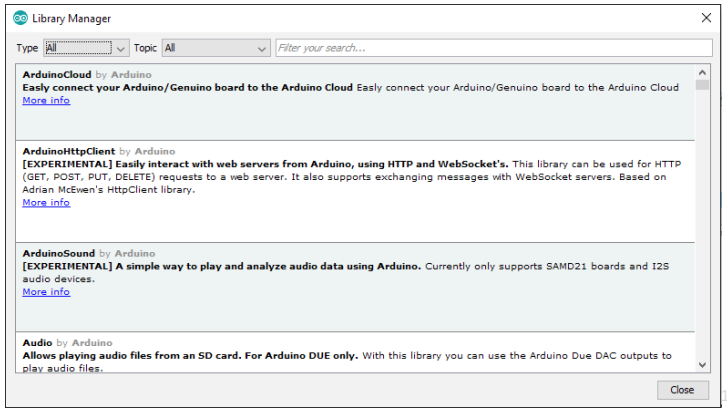
- ESP8266 module is designed for wearable electronics, IoT applications, and mobile devices
- **There are three modes of operation: deep sleep, sleep mode, and active mode**
- The **deep sleep** mode stops all other chip functions except for the RTC, which is left functional
  - This way you can continue tracking time even when in deep sleep mode.
  - The module draws only 60 µA when in deep sleep mode.
- ESP8266 has to be in active mode when transmitting or measuring data
- By alternating between deep sleep and active modes you can run the ESP8266 module on a battery for years

3.2 Using Libraries on ESP8266

Using Libraries

- The libraries will play a huge role in enabling us to access additional functions on our board
- There are some ESP8266 libraries that get installed on the Arduino IDE automatically when you install the ESP8266 core
- Some third-party ESP8266 libraries do not come with the ESP8266 core and you have to download them and install them on your Arduino
  - **navigate to Sketch | Include Library | Manage Libraries**





How to Do It

- For example, create a simple Wi-Fi web server using the ESP8266WiFi library
- **The web server will send a Hello World! reply whenever a client connects to it**

```
#include <ESP8266WiFi.h>
// Create an instance of the server
// specify the port to listen on as an argument
WiFiServer server(80);

const char* ssid = "your-ssid";
const char* password = "your-password";

void setup() {
   Serial.begin(115200);
   delay(10);
   // Connect to WiFi network
   Serial.print("Connecting to WiFi hotspot");
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
   }
   Serial.println("");
   Serial.println("WiFi connected");

   // Start the server
   server.begin();
   Serial.println("Server started");

   // Print the IP address
   Serial.println(WiFi.localIP());
}
```

How to Do It

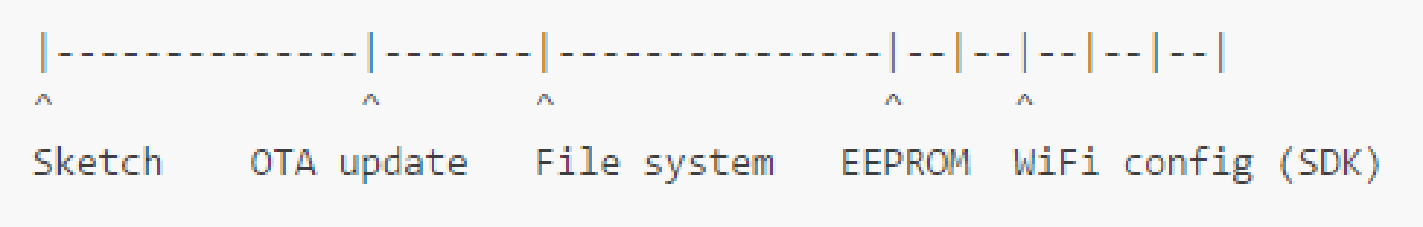
```
void loop() {
   // Check if a client has connected
   WiFiClient client = server.available();
   if (!client) {
      return;
   }
   // Wait until the client sends some data
   while(!client.available()){
      delay(1);
   }
   // Prepare the response
   String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
   <!DOCTYPE HTML>\r\n<html>\r\nHello World! </html>\n";

   // Send the response to the client
   client.print(s);
   delay(1);
   Serial.println("Client disconnected");
}
```

3.3 Discovering Filesystem of ESP8266

Flash Memory

- In most microcontroller architectures, the flash memory is used as the main storage media
- **Configuration files, firmware, and application data are all stored in it**
- The flash memory allows random writes and reads, and it can only be erased in blocks at a time
- In the Arduino environment, the flash memory is laid out in a specific way



- The size of the filesystem usually varies, depending on the size of the flash memory
- Different ESP8266 boards will not have the same flash memory size

Filesystem Memory Management

- SPIFFS is a module that enables access to and storage of data in the flash memory in file format
- SPIFFS partitions the available memory into pages to create a filesystem
- The pages are usually small enough to be cached in the RAM
- The default size is 256 bytes. The pages contain file contents and index information

- ESP8266FS is a tool that enables the Arduino IDE to upload files onto the ESP8266 flash filesystem

Setting Up ESP8266FS Tool

1. Download the ESP8266FS tool from this link:

   https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.5.0/ESP8266FS-0.5.0.zip

2. Create a Tools directory in your Arduino sketchbook directory

3. Unzip the ESP8266FS tool into the Tools directory. The path will look like this:

   MyDocuments/Arduino/tools/ESP8266FS/tool/esp8266fs.jar

4. Restart the Arduino IDE

5. Create a new sketch or open one that already exists

6. Click on the Sketch | Show sketch folder

7. Create a directory called data

8. Ensure that all the files you are intending to save in the filesystem are in the data directory

9. Select the board and port and close the serial monitor if it is open

10. Click on Tools | ESP8266 Sketch Data Upload.

11. The IDE will start uploading the files in the data directory to the ESP8266 flash filesystem

Setting Up ESP8266FS Tool

Once the upload is complete, the status bar on the IDE will display SPIFFS image Uploaded

The library that supports SPIFFS is called FS. Therefore, if you want to store some files in the ESP8266 flash filesystem, add the #include <FS.h>

`Filesystem` object:

- `begin()`: Mounts the SPIFFS filesystem
- `format()`: Formats the filesystem
- `end()`: Unmounts the SPIFFS filesystem
- `open(path, mode)`: Opens a file
- `exists(path)`: Returns true if there is a file with the provided path
- `openDir(path)`: Opens a directory whose path is provided
- `rename(pathfrom, pathTo)`: Renames files
- `remove(path)`: Deletes the provided path

`Directory` object:

- `next()`: Returns true if there are files to scroll over
- `fileName()`: Returns the filename of the file that is being pointed to
- `openfile(mode)`: Opens the file

`File` object:

- `seek(offset, mode)`: Returns true if the position was set successfully
- `position()`: Returns current position in the file in bytes
- `size()`: Returns the size of the file in bytes
- `name()`: Returns the filename
- `close()`: Closes the file

Example Code - Filesystem Basic

```
#include "FS.h"

void setup(){
  Serial.begin(115200);
  SPIFFS.begin();
  SPIFFS.format();
  if(SPIFFS.exists("/config.txt")){
    Serial.println("found");
  }
  else{
    Serial.println("not found");
  }

  File f = SPIFFS.open("/config.txt", "w");
  if(!f){
     Serial.println("file open failed");
  }
 Serial.println("--Write file--");
for(int i=1; i<=10; i++){
  f.println(i);
  Serial.println(i);
}
f.close();
Serial.println("--Read file--");
File fr = SPIFFS.open("/config.txt", "r");
if(!fr){
  Serial.println("file open failed");
}
while(fr.available()){
  String v = fr.readStringUntil('\n');
  int value = v.toInt();
  Serial.println(value);
}
fr.close();
```

Example Code - Filesystem Basic

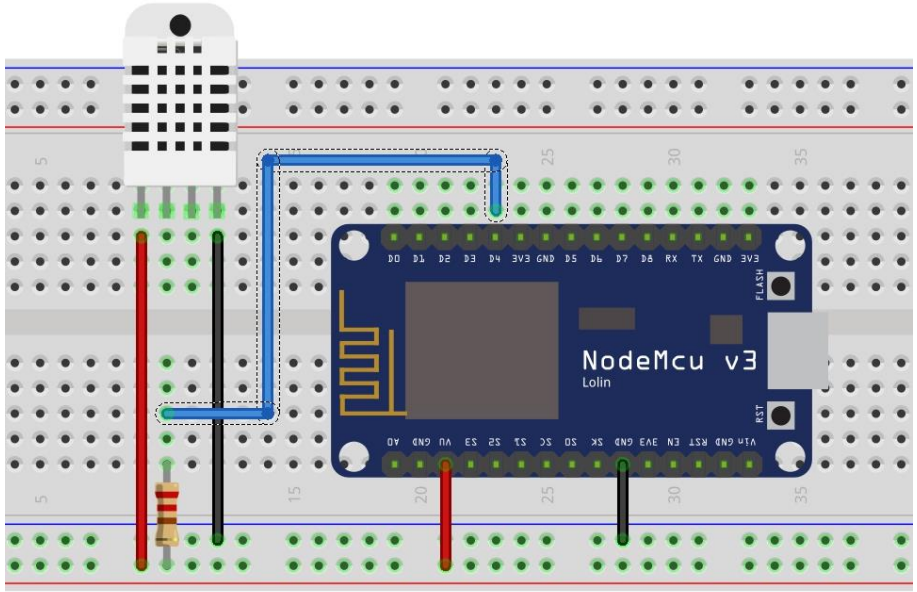
```
 if(SPIFFS.exists("/config.txt")){
    Serial.println("found");
  }
  else{
    Serial.println("not found");
  }

}
void loop()
{}
```

3.4 Storing Data in ESP866 Filesystem

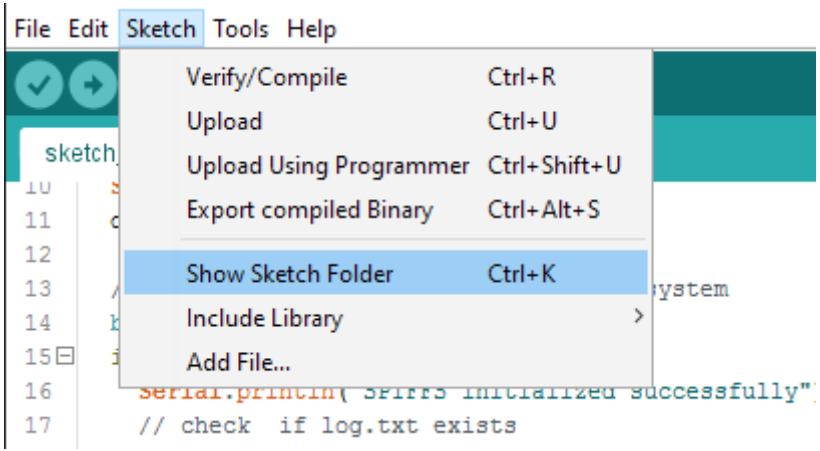
Getting Ready

- Connect the data pin of the DHT22 to GPIO 2 (D4) of the ESP8266 board



How to Do It

1) create a text file called log.txt that will hold the sensor data. There are two ways you can go about it. You can use the Arduino IDE to upload the text file to the ESP8266 filesystem or you can create it using our sketch
2) create a directory called data in the Sketch folder



How to Do It

3) create the log.txt file in the data folder

4) With the log.txt file successfully created, go back to the Arduino IDE.

Select the ESP8266 board you are using and the correct serial port from the tools menu,

And click on Tools | ESP8266 Sketch Data Upload

The log.txt file will be uploaded to the ESP8266 board filesystem.

How to Do It

```
#include "FS.h" #include "DHT.h"

#define DHTPIN 2         // the digital pin we are connected to
#define DHTTYPE DHT22    // define type of DHT sensor we are using

DHT dht(DHTPIN, DHTTYPE); // create DHT object

void setup() {
   Serial.begin(115200);
   dht.begin();          // intialize DHT object
   // always use this to "mount" the filesystem
   bool ok = SPIFFS.begin();
   if (ok) {
     Serial.println("SPIFFS initialized successfully");
   }
   else{
     Serial.println("SPIFFS intialization error");
   }
}
```

How to Do It

```
void loop() {
   // Wait a few seconds between measurements.
   delay(2000);
   //read temperature as Celcius
   float t = dht.readTemperature();
   // Check if any reads failed and exit early (to try again).
   if (isnan(t)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
   }
   //open log.txt file
   File f = SPIFFS.open("/log.txt", "a");
   if (!f) {
      Serial.println("file open failed");
   }
   // save temperature reading
   f.print(t);
   f.println("deg C");
   //close file
   f.close();
   delay(8000);
}
```

3.5 Discovering OTA Update of ESP8266

Over The Air

- OTA update involves loading firmware to an ESP8266 module via Wi-Fi, instead of using a serial port
- This is a very important feature that ensures the delivery of firmware updates to ESP8266 boards in remote areas

Note

- There are no security measures that prevent the OTA process from being hacked
- Ensure that only updates from legitimate sources are accepted
- OTA updates interrupt the normal running of the previous code, once the new code is executed

Security & Safety

- you can protect your updates with passwords or accept updates only from a specified OTA port

```
void setPort(uint16_t port);
void setHostname(const char* hostname);
void setPassword(const char* password);
```

- The best practice is putting the controlled processes or equipment in a safe state before beginning the update

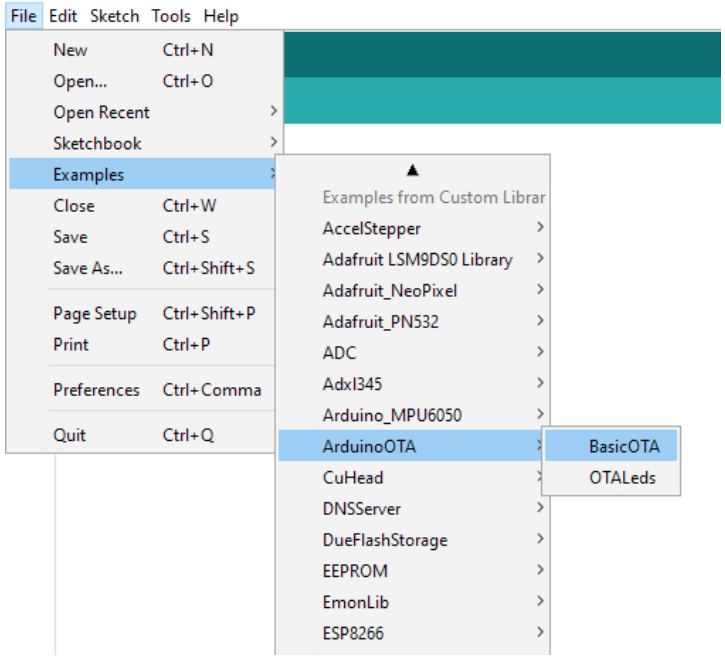
```
void onStart(OTA_CALLBACK(fn));
void onEnd(OTA_CALLBACK(fn));
void onProgress(OTA_CALLBACK_PROGRESS(fn));
void onError(OTA_CALLBACK_ERROR (fn));
```

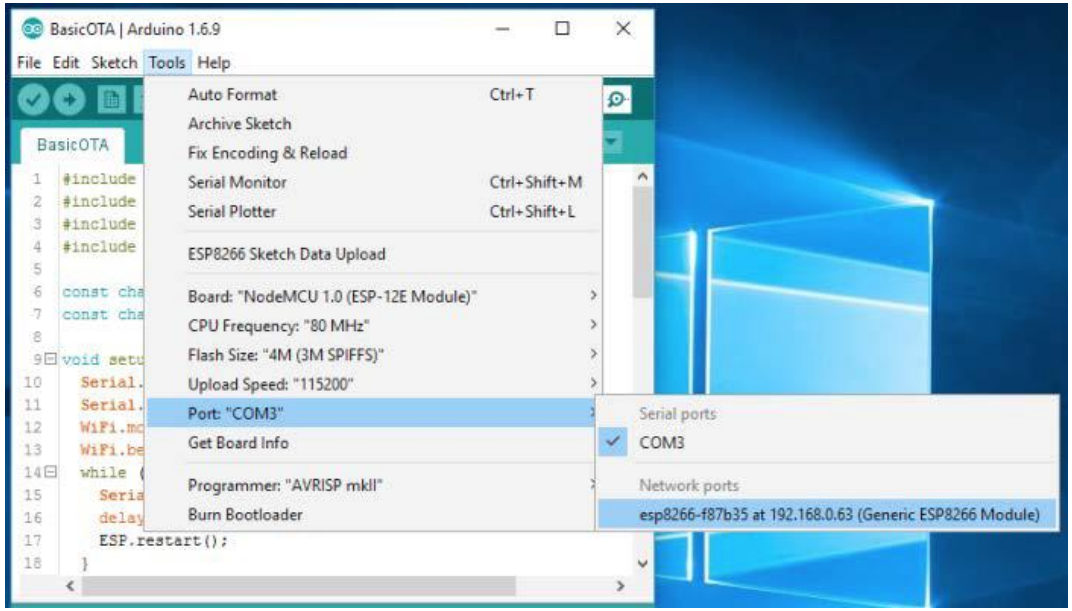
3.6 Programming Your ESP8266 OTA

How to Do It

- The first thing you need to do is set up your Arduino IDE for OTA updates
- You will require the following things:
  - Python 2.7 or 3.8 (https://www.python.org/)
  - Arduino core for ESP8266 (https://github.com/esp8266/Arduino#installing-with-boards-manager)
  - Arduino IDE
  - A Wi-Fi network

How to Do It


File Edit Sketch Tools Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
Examples from Custom Librar
AccelStepper
Adafruit LSM9DS0 Library
Adafruit_NeoPixel
Adafruit_PN532
ADC
Adxl345
Arduino_MPU6050
ArduinoOTA
CuHead
DNSServer
DueFlashStorage
EEPROM
EmonLib
ESP8266
BasicOTA
OTALeds


BasicOTA | Arduino 1.6.9
File Edit Sketch Tools Help
BasicOTA
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP8266 Sketch Data Upload
Board: "NodeMCU 1.0 (ESP-12E Module)"
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "115200"
Port: "COM3"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM3
Network ports
esp8266-f87b35 at 192.168.0.63 (Generic ESP8266 Module)
ESP.restart();

Basic OTA

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>

const char* ssid = "CSB310 AirStation";
const char* password = "netlab310";

void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
```

Basic OTA

```
 ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
      type = "sketch";
    } else { // U_SPIFFS
      type = "filesystem";
    }

    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using
SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
```

Basic OTA

```
ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) {
      Serial.println("Auth Failed");
    } else if (error == OTA_BEGIN_ERROR) {
      Serial.println("Begin Failed");
    } else if (error == OTA_CONNECT_ERROR) {
      Serial.println("Connect Failed");
    } else if (error == OTA_RECEIVE_ERROR) {
      Serial.println("Receive Failed");
    } else if (error == OTA_END_ERROR) {
      Serial.println("End Failed");
    }
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}
```

Try OTA Update with This Example