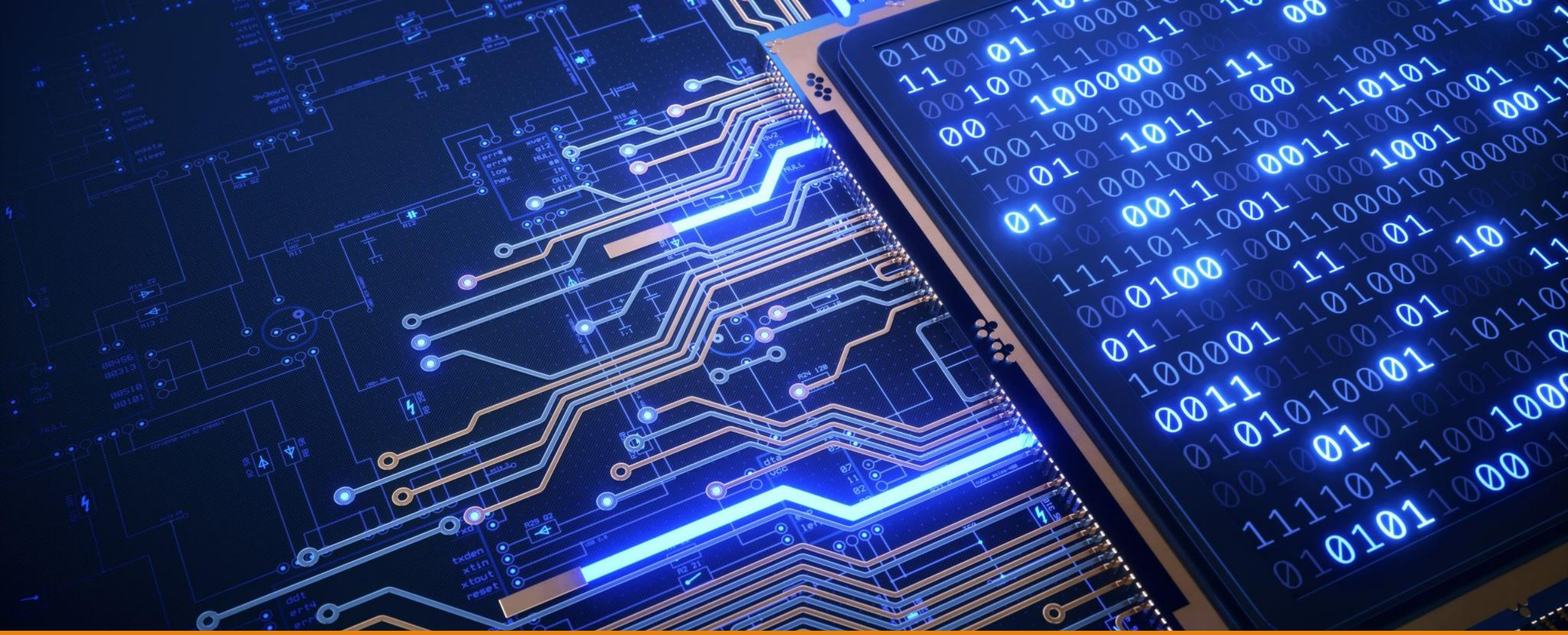


Embedded Systems

Dr. Abdelhay Ali

Lecture 8

ESP8266

Outlines

1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

Outlines

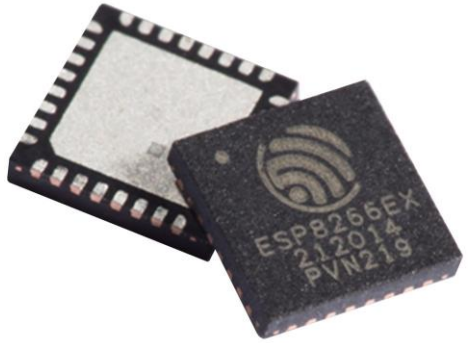
1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

Introduction

- The ESP8266 is a System on a Chip (SoC), manufactured by the Chinese company Espressif.
- It consists of a Tensilica L106 32-bit **micro controller** unit (MCU) and a **Wi-Fi transceiver**.
- It has **11 GPIO pins*** (General Purpose Input/Output pins), and an **analog input** as well.
- This means that you can program it like any normal Arduino or other microcontroller.
- And on top of that, you get Wi-Fi communication, so you can use it to connect to your Wi-Fi network, connect to the Internet, host a web server with real web pages, let your smartphone connect to it, etc

(*) The ESP8266 chip itself has 17 GPIO pins, but 6 of these pins (6-11) are used for communication with the on-board flash memory chip.

Introduction



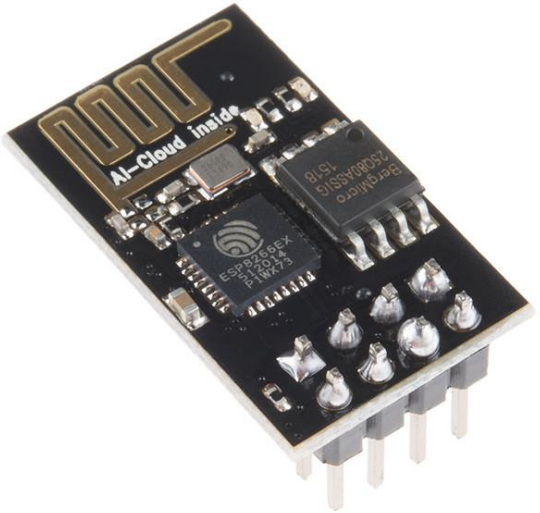
Introduction

There are many different modules available, standalone modules like the ESP-## series by AI Thinker, or complete development boards like the NodeMCU DevKit or the WeMos D1.

Different boards may have different pins broken out, have **different Wi-Fi antennas**, or a different amount of **flash memory** on board

Introduction

Ai-Thinker ESP-01:



The ESP-01 is one of the biggest selling IoT Wi-Fi modules on the market.

It's widely used in smart home and networking projects.

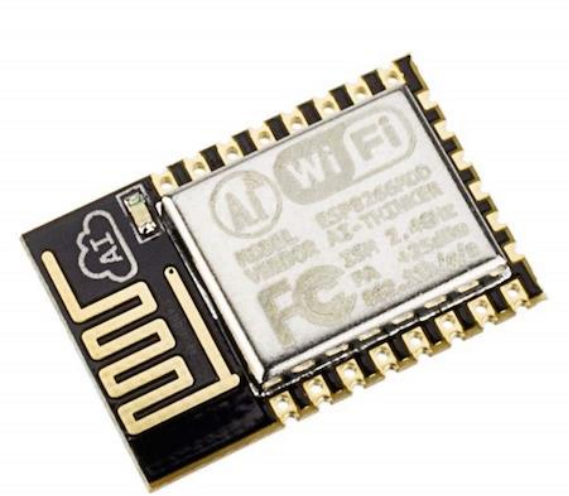
The default AT firmware enables it to be used in combination with an Arduino. However, you can easily update the firmware with a USB-to-ESP-01 adaptor module.

A common complaint with this board is that the pin posts make it difficult to plug it directly into a breadboard, but this can be easily overcome by building or buying an adaptor module.

There are two versions available, one with 500kb of flash and the other with 1Mbit of flash.

Introduction

Ai-Thinker ESP-12



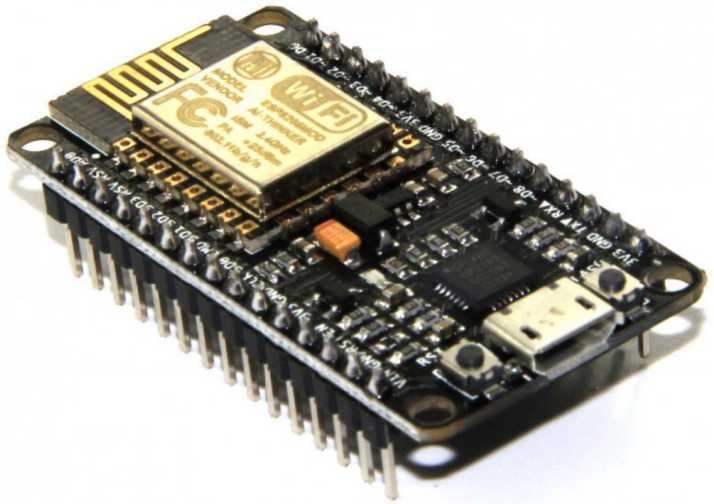
This is a more fully featured module with 11 GPIO pins, an ADC, 4Mbits of flash, and 10-bit resolution.

However, the module is not breadboard friendly, meaning you'll need to use an adaptor.

There are two versions available, ESP-12F, which has 20 GPIOS and ESP-12S, which has 14.

Introduction

Espressif NodeMCU module V1.0



This board has the ESP-12E module and comes with 4 Mbits of flash and features a row of pins on each side of the breadboard.

The board comes with four communication interfaces: SPI, I2C, UART, and I2S, with 16 GPIO and one ADC.

The RAM is 160KB, divided into 64KB for instruction and 96KB for data.

Outlines

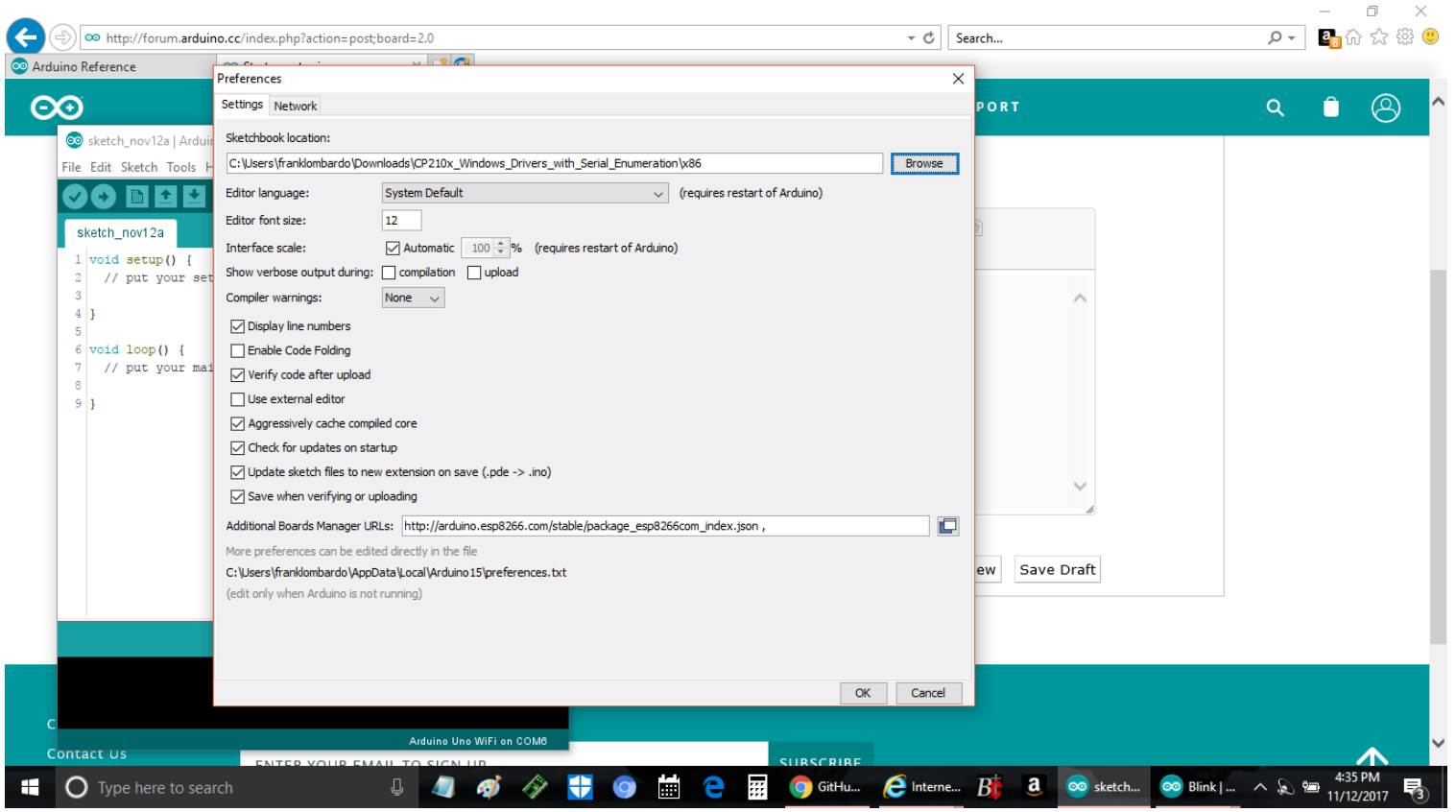
1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

Programming

- There are different ways to program the ESP8266, but I'll only cover the method using the Arduino IDE. This is really easy for beginners, and it's a very familiar environment if you've used Arduino boards before.
- The first step is to download and install the Arduino IDE

1. Open the Arduino IDE.
2. Go to File > Preferences.
3. Paste the URL http://arduino.esp8266.com/stable/package_esp8266com_index.json into the *Additional Board Manager URLs* field.
   (You can add multiple URLs, separating them with commas.)
4. Go to Tools > Board > Board Manager and search for 'esp8266'. Select the newest version, and click install. (As of February 7th 2017, the latest stable version is 2.3.0.)

Programming



Outlines

1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

The ESP8266 as a microcontroller - Hardware

Digital I/O

Just like a normal Arduino, the ESP8266 has digital input/output pins (I/O or GPIO, General Purpose Input/Output pins). As the name implies, they can be used as digital inputs to read a digital voltage, or as digital outputs to output either 0V (sink current) or 3.3V (source current).

Voltage and current restrictions

The ESP8266 is a 3.3V microcontroller, so its I/O operates at 3.3V as well. The pins are **not 5V tolerant, applying more than 3.6V on any pin will kill the chip.**

The maximum current that can be drawn from a single GPIO pin is **12mA**.

The ESP8266 as a microcontroller - Hardware

Digital I/O

Usable pins

The ESP8266 has 17 GPIO pins (0-16), however, you can only use 11 of them, because 6 pins (GPIO 6 - 11) are used to connect the flash memory chip. This is the small 8-legged chip right next to the ESP8266. If you try to use one of these pins, you might crash your program.

GPIO 1 and 3 are used as TX and RX of the hardware Serial port (UART), so in most cases, you can’t use them as normal I/O while sending/receiving serial data.

The ESP8266 as a microcontroller - Hardware

Internal pull-up/-down resistors

GPIO 0-15 all have a built-in pull-up resistor, just like in an Arduino. GPIO16 has a built-in pull-down resistor.

PWM

Unlike most Atmel chips (Arduino), the ESP8266 doesn't support hardware PWM, however, software PWM is supported on all digital pins. The default PWM range is 10-bits @ 1kHz, but this can be changed (up to >14-bit@1kHz).

Analog input

The ESP8266 has a single analog input, with an input range of **0 - 1.0V**. If you supply 3.3V, for example, you will damage the chip. Some boards like the NodeMCU have an on-board resistive voltage divider, to get an easier 0 - 3.3V range.

The ADC (analog to digital converter) has a resolution of 10 bits.

The ESP8266 as a microcontroller - Hardware

Communication

Serial

The ESP8266 has two hardware UARTS (Serial ports):
UART0 on pins 1 and 3 (TX0 and RX0 resp.), and UART1 on pins 2 and 8 (TX1 and RX1 resp.), however, GPIO8 is used to connect the flash chip. This means that UART1 can only transmit data.

The ESP8266 as a microcontroller - Hardware

GPIO overview

GPIO	Function	State	Restrictions
0	Boot mode select	3.3V	No Hi-Z
1	TX0	-	Not usable during Serial transmission
2	Boot mode select TX1	3.3V (boot only)	Don't connect to ground at boot time Sends debug data at boot time
3	RX0	-	Not usable during Serial transmission
4	SDA (I²C)	-	-
5	SCL (I²C)	-	-
6 - 11	Flash connection	x	Not usable, and not broken out
12	MISO (SPI)	-	-
13	MOSI (SPI)	-	-
14	SCK (SPI)	-	-
15	SS (SPI)	0V	Pull-up resistor not usable
16	Wake up from sleep	-	No pull-up resistor, but pull-down instead Should be connected to RST to wake up

Outlines

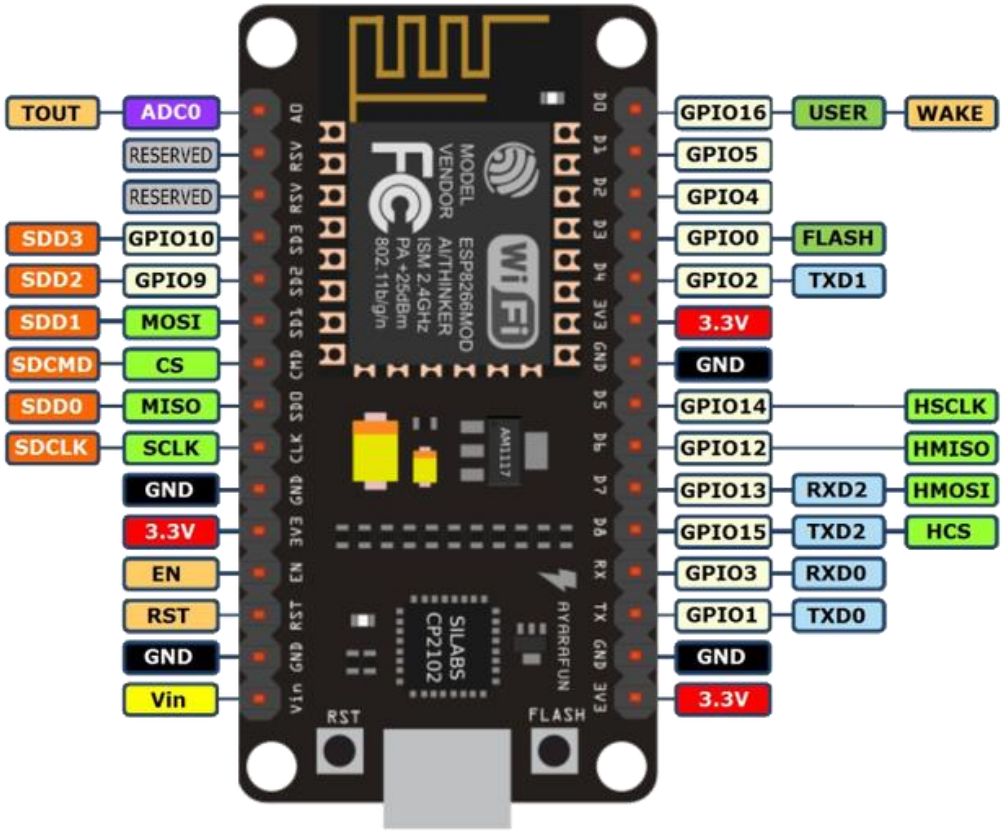
1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

The ESP8266 as a microcontroller - Software

Most of the microcontroller functionality of the ESP uses exactly the same syntax as a normal Arduino, making it really easy to get started.

Digital I/O

Just like with a regular Arduino, you can set the function of a pin using **pinMode(pin, mode);** where pin is the GPIO number*, and mode can be either **INPUT**, which is the default, **OUTPUT**, or **INPUT_PULLUP** to enable the built-in pull-up resistors for GPIO 0-15

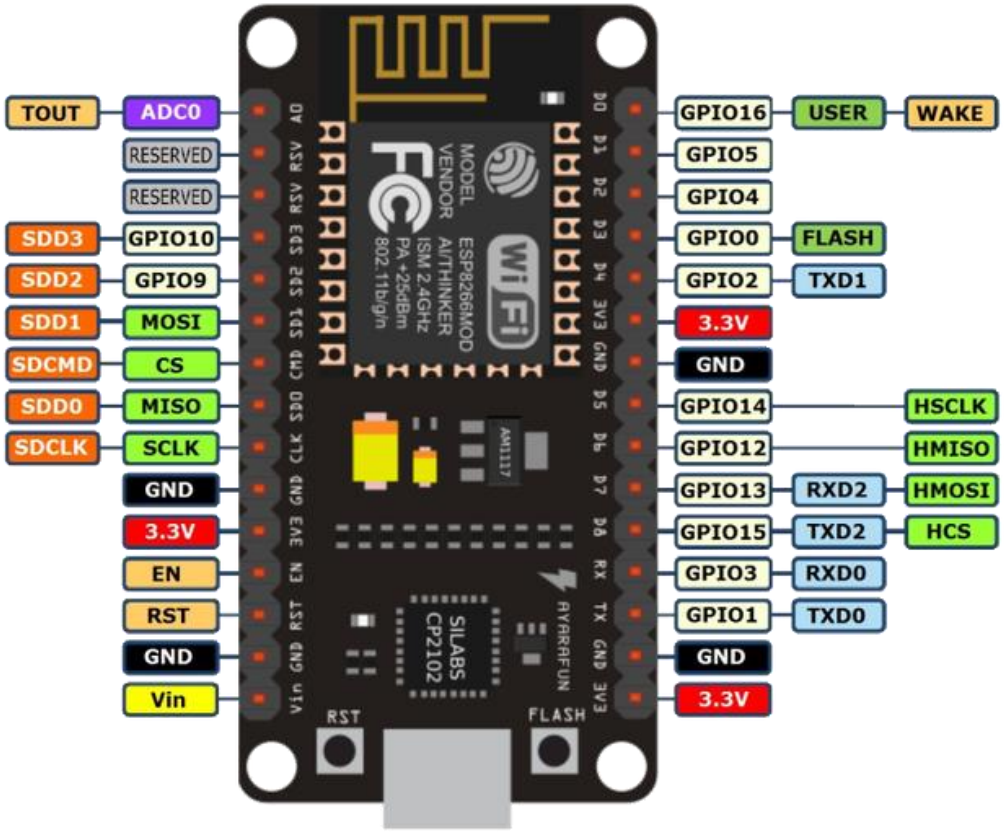


The ESP8266 as a microcontroller - Software

Digital I/O

To set an output pin high (3.3V) or low (0V), use **digitalWrite(pin, value);** where pin is the digital pin, and value either 1 or 0 (or **HIGH** and **LOW**).

To read an input, use **digitalRead(pin);**

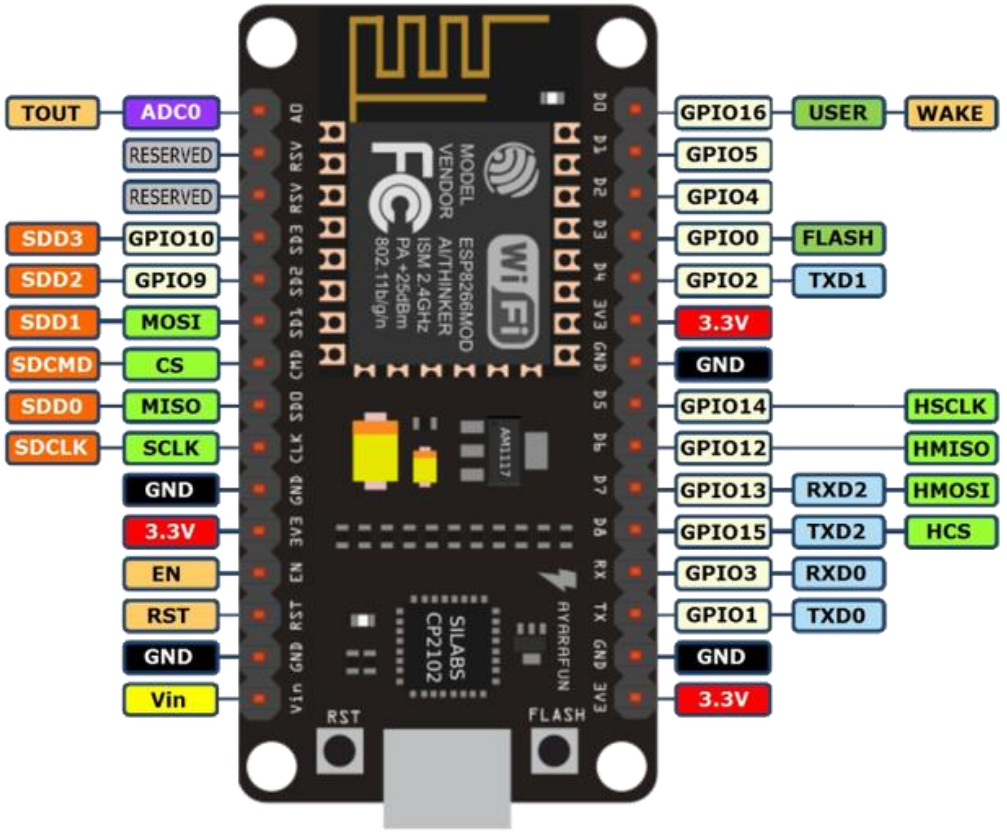


The ESP8266 as a microcontroller - Software

PWM

To enable PWM on a certain pin, use **analogWrite(pin, value);** where pin is the digital pin, and value a number between 0 and 1023.

The frequency can be changed by using **analogWriteFreq(new_frequency);.**
new_frequency should be between 100 and 1000Hz.

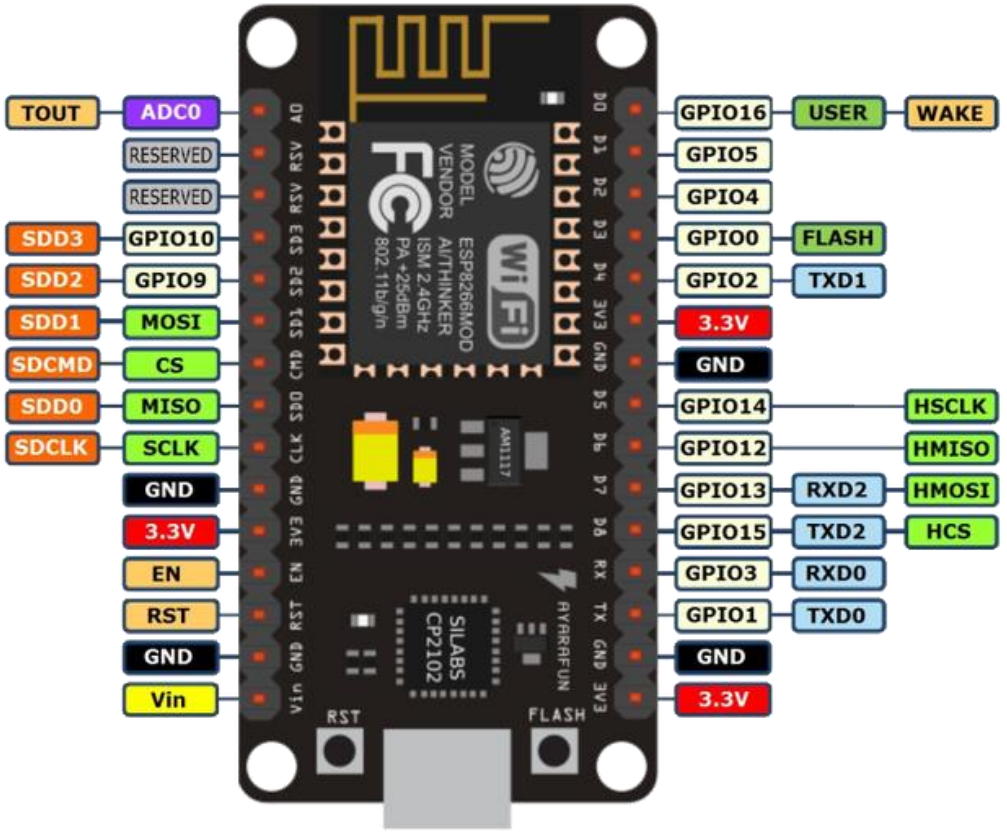


The ESP8266 as a microcontroller - Software

Analog input

Just like on an Arduino, you can use **analogRead(A0)** to get the analog voltage on the analog input.

(0 = 0V, 1023 = 1.0V).

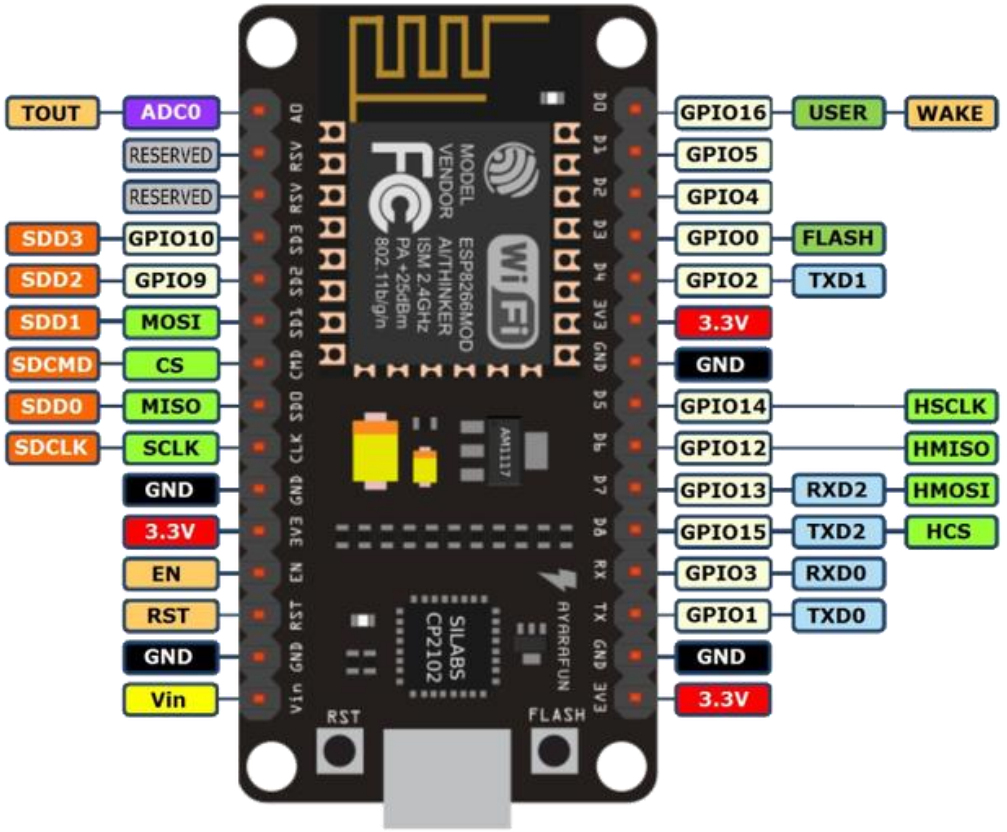


The ESP8266 as a microcontroller - Software

Serial communication

To use UART0 (TX = GPIO1, RX = GPIO3), you can use the Serial object, just like on an Arduino: **Serial.begin(baud).**

All Arduino Stream functions, like read, write, print, println, ... are supported as well.



Outlines

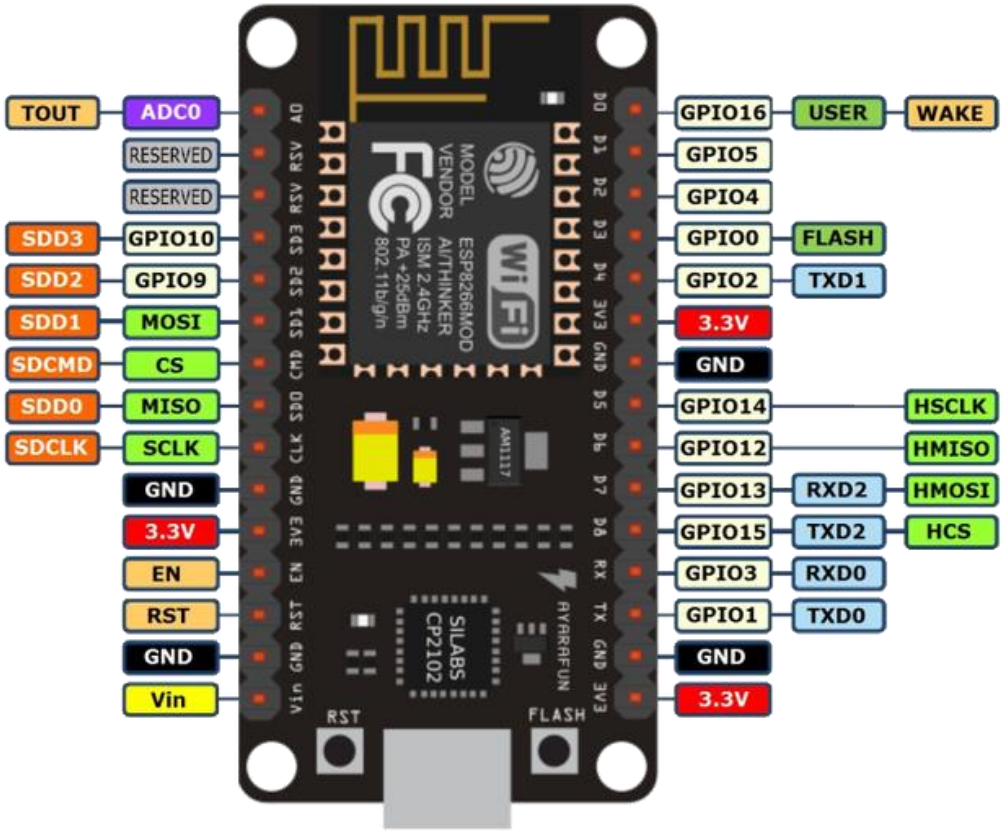
1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

Establishing a Wi-Fi connection

the ESP8266 can operate in three different modes:

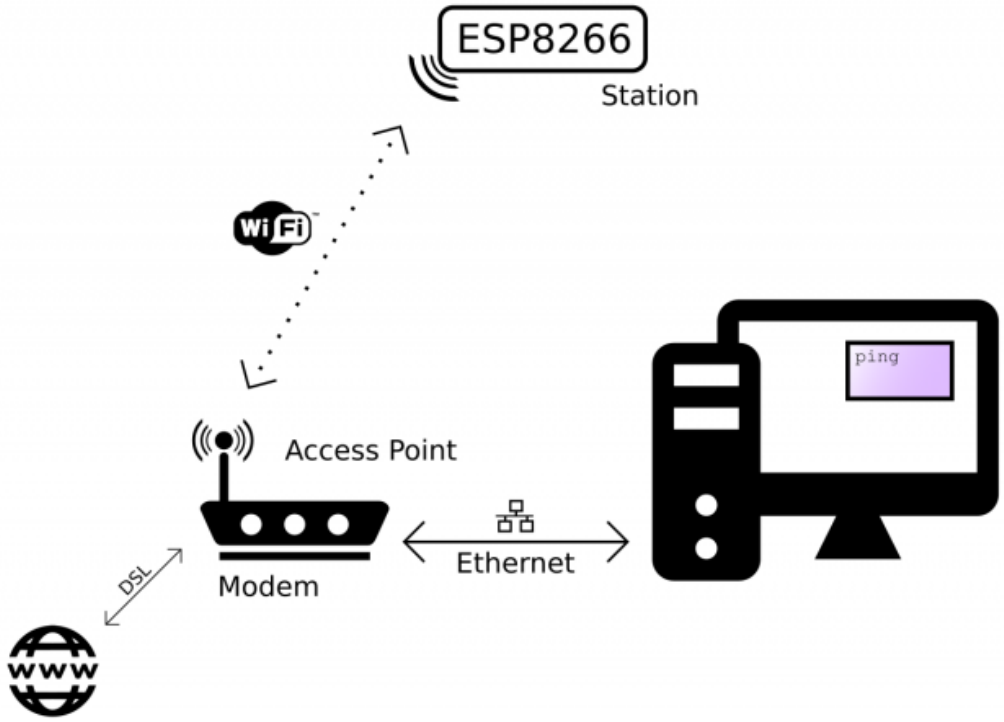
1. Wi-Fi station,
2. Wi-Fi access point,
3. and both at the same time.

We'll start by looking at the configuration of a Wi-Fi station.



Establishing a Wi-Fi connection

1. Wi-Fi station,



```
#include <ESP8266WiFi.h> // Include the Wi-Fi library
const char* ssid = "SSID"; // The SSID (name) of the Wi-Fi network you want to connect
const char* password = "PASSWORD"; // The password of the Wi-Fi network

void setup() {
Serial.begin(115200); // Start the Serial communication to send messages to the PC

WiFi.begin(ssid, password); // Connect to the network

Serial.print("Connecting to ");
Serial.print(ssid);

int i = 0;
while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
Serial.print(++i);  delay(1000);}

Serial.println("Connection established!");
Serial.print("IP address:\t");
Serial.println(WiFi.localIP()); // Send the IP address of the ESP8266 to the computer
}

 void loop() { }
```

Establishing a Wi-Fi connection

Serial output

```
Connecting to SSID ...
1 2 3 4 5 6 ...

Connection established!
IP address: 192.168.1.3
```

Establishing a Wi-Fi connection

Access Point mode

```
#include <ESP8266WiFi.h> // Include the Wi-Fi library

const char *ssid = "ESP8266 Access Point"; // The name of the Wi-Fi network that will be created
const char *password = "thereisnospoon"; // The password required to connect to it

void setup() {
WiFi.softAP(ssid, password); // Start the access point of the ESP8266

}


 void loop() { }
```

Outlines

1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

ESP8266 First Web Server

Access Point mode

```
#include <ESP8266WiFi.h> // Include the Wi-Fi library

const char *ssid = "ESP8266 Access Point"; // The name of the Wi-Fi network that will be created
const char *password = "thereisnospoon"; // The password required to connect to it

void setup() {
WiFi.softAP(ssid, password); // Start the access point of the ESP8266

}


 void loop() { }
```

ESP8266 First Web Server

The actual implementation of a web server is much easier than it sounds, because the ESP8266 Arduino Core includes some great libraries that handle pretty much everything for you.

Let's look at a basic Hello World! example.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

ESP8266WebServer server(80); /* Create a webserver object that listens for HTTP
request on port 80 */
void handleRoot(); /* function prototypes for HTTP handlers */
void handleNotFound();

const char *ssid = "ESP8266 Access Point"; /* The name of the Wi-Fi network that will be created
const char *password = "thereisnospoon"; /* The password required to connect to it */

void setup() {
WiFi.softAP(ssid, password); /* Start the access point of the ESP8266 */
server.on("/", handleRoot); /* Call the 'handleRoot' function when a client requests
URI "/" */

server.onNotFound(handleNotFound); /* When a client requests an unknown URI (i.e.
something other than "/"), call function "handleNotFound"  */

server.begin(); // Actually start the server
}
```

```
void loop(void){
server.handleClient(); // Listen for HTTP requests from clients
}

void handleRoot() {
server.send(200, "text/plain", "Hello world!"); /* Send HTTP status 200 (Ok) and send
some text to the browser/client */
}


void handleNotFound(){
server.send(404, "text/plain", "404: Not found"); /* Send HTTP status 404 (Not Found)
when there's no handler for the URI in the request */
}
```

Outlines

1. Introduction
2. Programming
3. The ESP8266 as a microcontroller - Hardware
4. The ESP8266 as a microcontroller - Software
5. Establishing a Wi-Fi connection
6. ESP8266 First Web Server
7. Turning on and off an LED over Wi-Fi

Turning on and off an LED over Wi-Fi

We can use the web server to serve interactive pages, and to react to certain POST request.

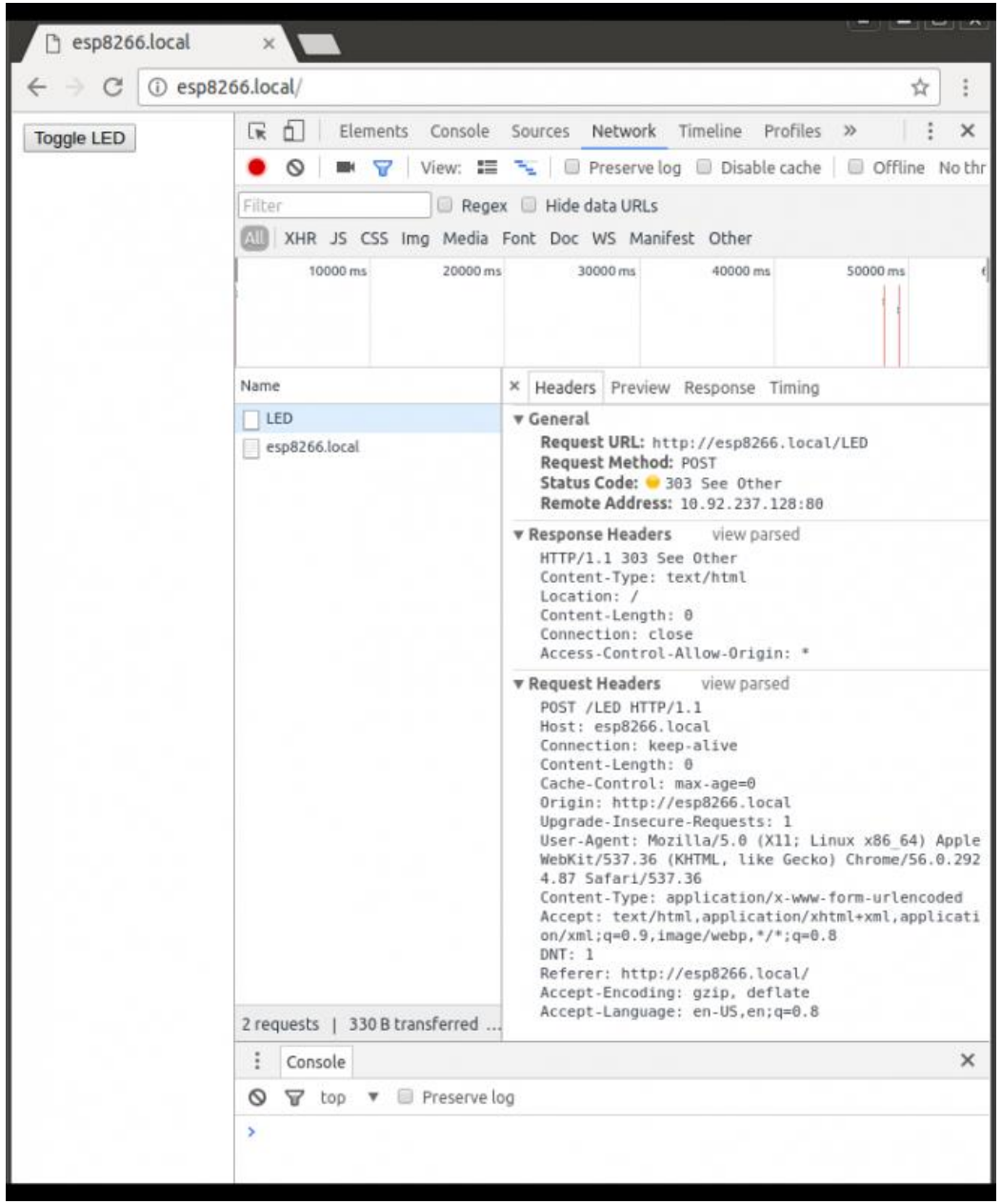
In the following example, the ESP8266 hosts a web page with a button.

When the button is pressed, the browser sends a POST request to /LED.

When the ESP receives such a POST request on the /LED URI, it will turn on or off the LED, and then redirect the browser back to the home page with the button.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
ESP8266WebServer server(80); /* Create a webserver object that listens for HTTP
request on port 80 */
const int led = 2;
void handleRoot(); // function prototypes for HTTP handlers
void handleLED();
void handleNotFound();
const char *ssid = "ESP8266 Access Point"; /* The name of the Wi-Fi network that will be created
const char *password = "thereisnospoon"; /* The password required to connect to it */

void setup() {
pinMode(led, OUTPUT);
WiFi.softAP(ssid, password); /* Start the access point of the ESP8266 */
server.on("/", handleRoot); /* Call the 'handleRoot' function when a client requests
URI "/" */
server.on("/LED", HTTP_POST, handleLED); /* Call the 'handleLED' function when a POST
request is made to URI "/LED" */
server.onNotFound(handleNotFound); /* When a client requests an unknown URI (i.e.
something other than "/"), call function "handleNotFound"  */
server.begin(); // Actually start the server
}
```

```
void loop(void){
server.handleClient(); // Listen for HTTP requests from clients
}

void handleRoot() {
server.send(200, "text/html", "<form action=\"/LED\" method=\"POST\"><input
type=\"submit\" value=\"Toggle LED\"></form>"); /* Send a web page with a button to
toggle the LED to the browser/client */
}


void handleLED() { // If a POST request is made to URI /LED
digitalWrite(led,!digitalRead(led)); // Change the state of the LED
server.sendHeader("Location","/"); /* Add a header to respond with a new location for
the browser to go to the home page again */
server.send(303); /* Send it back to the browser with an HTTP status 303 (See Other)
to redirect */
}

void handleNotFound(){
server.send(404, "text/plain", "404: Not found"); /* Send HTTP status 404 (Not Found)
when there's no handler for the URI in the request */
}
```


esp8266.local
esp8266.local/
Toggle LED
Elements Console Sources Network Timeline Profiles
View: Preserve log Disable cache Offline No thr
Filter Regex Hide data URLs
All XHR JS CSS Img Media Font Doc WS Manifest Other
10000 ms 20000 ms 30000 ms 40000 ms 50000 ms
Name
LED
esp8266.local
2 requests | 330 B transferred ...
Headers Preview Response Timing
General
Request URL: http://esp8266.local/LED
Request Method: POST
Status Code: 303 See Other
Remote Address: 10.92.237.128:80
Response Headers view parsed
HTTP/1.1 303 See Other
Content-Type: text/html
Location: /
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Request Headers view parsed
POST /LED HTTP/1.1
Host: esp8266.local
Connection: keep-alive
Content-Length: 0
Cache-Control: max-age=0
Origin: http://esp8266.local
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) Apple
WebKit/537.36 (KHTML, like Gecko) Chrome/56.0.292
4.87 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,applicati
on/xml;q=0.9,image/webp,*/*;q=0.8
DNT: 1
Referer: http://esp8266.local/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Console
top Preserve log

Thanks