ESP8266 Workshop by Volundr

Student manual

e.t.s.c. Volundr

January 12, 2023

Abstract

The ESP8266 is a chip that can be used to connect to WiFi networks. Originally this chip was used in combination with other microcontrollers and had to be controlled over serial with AT commands. Eventually the ESP-SDK was released which allowed the chip to be programmed itself. Development boards were produced which made it easy to get started with programming the ESP8266, an example of such a development board is the NodeMCU v3.0 which is shown in figure 1.

This guide will focus on programming this ESP8266 development board using the Arduino IDE.

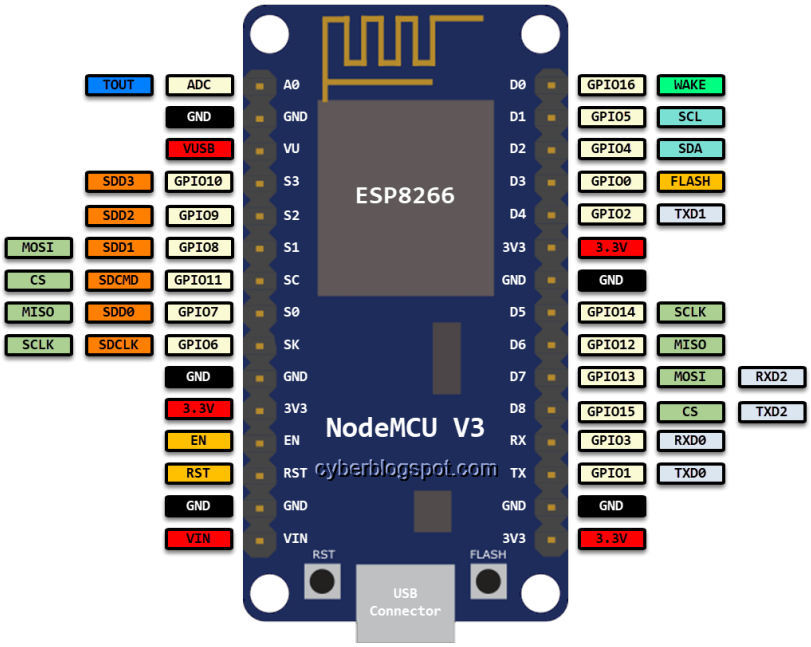

Figure 1: NodeMCU v3.0

1 Adding ESP8266 to the arduino IDE

Go to preferences and add `http://arduino.esp8266.com/stable/package_esp8266com_index.json` to additional boards manager urls as shown in Figure 2.
Then go to *Tools* → *Board* → *Boards manager*.
After updating, search for ESP8266 click on the single option and select version 3.1.0 and press install. After installing press close to close the window. Now in the *Tools* → *Board* menu there should be options available for the ESP8266.

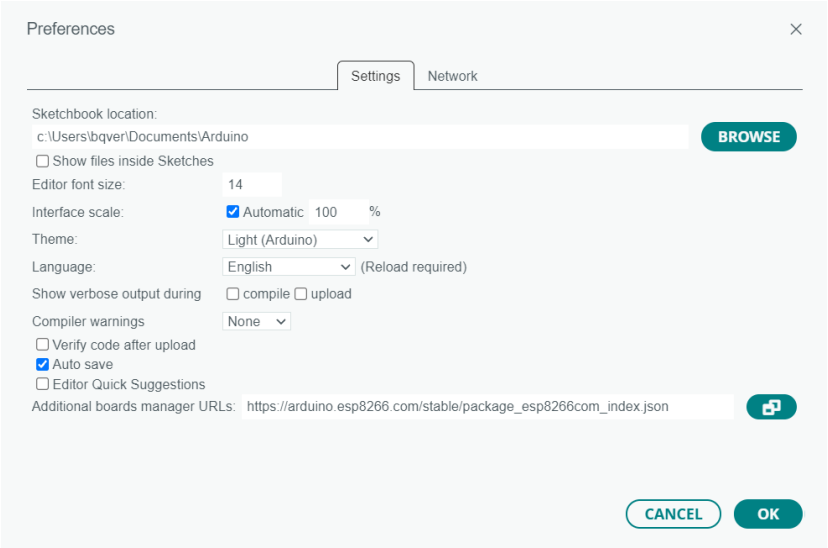

Figure 2: Preferences

For using the provided board, select "Generic ESP8266" in the boards menu and the right Serial port. After these steps the module is programmable by simply pressing the upload button in the arduino IDE.

2 Blinky

Just like programming an arduino, all arduino functions are available. All GPIO numbers shown in Figure 1 correspond with the numbers required for digitalread/digitalwrite.

2.1 Create a blinky program that blinks the led connected to GPIO2.

3 Wifi accesspoint

The ESP8266 can connect to a WiFi accesspoint or be an accesspoint. For this guide the ESP8266 will be an accesspoint. For this the ESP8266WiFi library contains the WiFi.softAP(const char *ssid, const char *password) function. After creating the accesspoint you can get the ip of the accesspoint with the function WiFi.softAPIP(). This function returns an object of the class IPAddress. Dont forget to include ESP8266WiFi.h to access the various WiFi related functions.

3.1 Create a program that creates a accesspoint with your own ssid and password. This password has to be at least 8 character. Check if you connect to this accesspoint with your phone.

3.2 Print ip address of the accesspoint over the serial port to the Arduino IDE.

4 Hosting a website

Connecting is all fine, but we want to display something on the phone. The easiest is to host a website and point the phones browser to the IP. The ESP8266 arduino library has a webserver called ESP8266WebServer that will handle most of the hard work. You still need to import it using *++#include <ESP8266WebServer.h>*.

When using this library you first create an object of this server that listens to port 80 as follows *ESP8266WebServer server(80);*. This should be global variable, so it can be accessed in the various parts of your program. During the setup of program you should register the callbacks that handle the web request made to the server and starting the server itself after all the callbacks have been registered. With the function *server.on(String requesturi, method, function name)*. You register callback that is called when a request is made with the specified method. For example *server.on("/", HTTP_GET, handleRoot)* is used to register a function called handleRoot when the main page is requested. To start the server you need to call the function *server.begin()*. After that you need to regularly call the function *server.handleClient()* for the webserver to work. To reply to web request you can use the function *server.send(int HTTPResponseCode, String ContentType, String content)* inside your callback function previously registered. For example *server.send(200, "text/html", F("<h1>You are connected</h1>"))* replies to a request with a simple webpage.

4.1 Reply to the root request with a simple webpage.

4.2 Display the webpage on your phone by connecting to it and use your browser to connect to the ip found in 3.2.

5 Controlling a LED

Two commonly used methods for HTTP are GET and POST. GET requests have been handled before. POST requests are made by forms on webpage. For instance textboxes,buttons,etc make a POST request to the webserver.
Now we want to toggle the led with a button on the webpage. For this we add a form with a button to the webpage. And add post handler to the program. This will handle all the post requests made by the button. Since html/css is outside of the scope of this tutorial sample websites can be found in Appendix A. By using the server.on function it is possible to register an POST function handler. Instead of using HTTP_GET you use HTTP_POST. Than the function will be called when a post is made on the page uri.
In the POST Handler you should place the code that with toggle the led. After toggling the led you should send the same webpage to the client.

5.1 Create a function handlePost that can toggle a led on the NodeMCU from the webpage.

Let say instead of toggling the LED we want to control it with an off and a on button. We expand the webpage with a second button and give these a different value. "1" When the on button is pressed and "0" when the off button is pressed. In handle post function we can use $server.arg("foo").toInt()$; command to find out what the value was that the post button had.

5.2 Create a function that has an on and off button for the led.

6 Reading a PIN

Previously we replied with a constant string as a webpage. However, we can make this string dynamic and build it up from multiple parts. The webpage can be changed depending on the state of pins, variables, etc. Tip: to auto refresh your page add this line at the start of your webpage: *"<meta http-equiv="refresh" content="3" >"*

6.1 Create a string that consists of Q4 and Q5.

6.2 Edit the webpage from question 5.2 so it also shows the state of the flash button that is connected to GPIO0.

7 Displaying serial data

Changing the current webpage to display the latest received serial data should now be quite easy.

7.1 Edit the webpage from question 6.1 so it shows its latest serial data.

A Sample websites

Single button webpage:

```
F("<meta name=\"viewport\" content=\"width=320\" />" //Zoom page
"<h1>You are connected</h1>"
"<form action=\"\" method=\"post\">"
"<button name=\"foo\" value=\"1\">Toggle</button>"
"</form>")
```

Dual button webpage:

```
 F("<meta name=\"viewport\" content=\"width=320\" />" //Zoom page
"<h1>You are connected</h1>" //Nice titel
"<form action=\"\" method=\"post\">" //Begin form
"<button name=\"foo\" value=\"1\">On</button>" //On button
"<br>" //New line
"<button name=\"foo\" value=\"0\">Off</button>" //Off button
"</form>")
```

The "F" before each string signifies that the uC does not need to load the string into SRAM first.

If you want to create your own website, but do not want to escape all characters, use a *const char html[] =R"rawliteral(<html></html>)rawliteral";*