Chapter 1: Configuring the ESP8266

204335 : Microcontroller and IoT

Part ESP8266

Suphakit Awiphan

Chiang Mai University

Chapter 1: Configuring the ESP8266

204335 Microcontroller and IoT

Chapter 1 – Configuring the ESP8266

Parts You’ll Need for This Chapter

- ESP8266 board
- DHT11
- Photocell
- Relay
- LED
- Resistor
- Breadboard
- Jumper wires
- Micro USB cable

Outline

- **Introducing the ESP8266**
- **Uploading Your First Sketch**
- **Connecting ESP8266 to WLAN**
- **Connecting ESP8266 to Cloud Server**

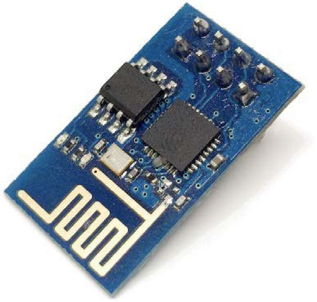
1.1 Introducing the ESP8266

Introducing the ESP8266

- ESP8266 module is a self-contained System On Chip (SOC)
- ESP8266 features an integrated **TCP/IP** that allows you to add **Wi-Fi capability** to projects
- ESP8266 boards come in different forms, depending on the company that manufactures
- All the boards use **Espressif's ESP8266** chip as the main controller
  - but different additional components and different pin configurations

Introducing the ESP8266

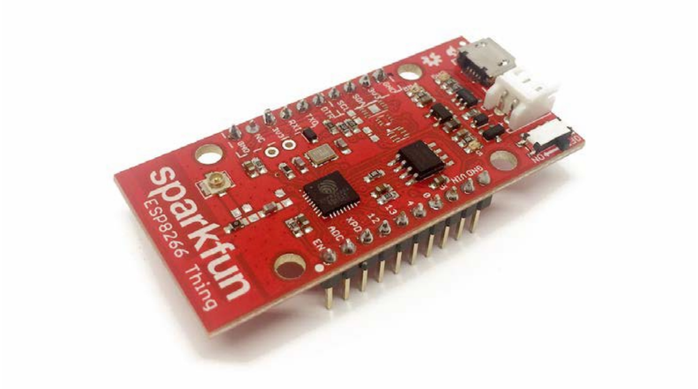
- **ESP8266-01** module is the most basic ESP8266 board
- eight pins, which include four GPIO pins, serial communication TX and RX pins, an enable pin and power pins, and VCC and GND
- 8-pin header on the ESP8266-01 module has a 2.0 mm spacing that is *not* compatible with breadboards.



- **ESP8266-07** is an improved version of the ESP8266-01 module.
- has 16 pins, including nine GPIO pins
- comes with a UFL connector that used to plug an external antenna

Introducing the ESP8266

- **Sparkfun ESP8266 Thing** is a development board for the ESP8266 Wi-Fi SOC
- It has 20 pins that are breadboard-friendly.
- features SPI, I2C, serial UART, and GPIO interface pins, enabling it to be interfaced with many input and output devices.



- 3.3V voltage regulator.
- powered using a micro USB cable or Li-Po battery

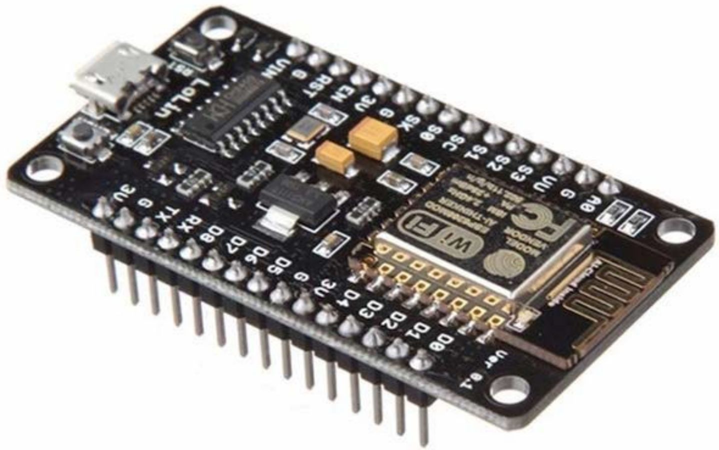
Introducing the ESP8266

- **Adafruit ESP8266** is a fully standalone ESP8266 board
- has a built-in USB to serial interface that eliminates the need for using an external FTDI breakout board to program it
- has an integrated battery charging circuit
- has a 3.3V voltage regulator.

- 28 breadboard friendly pins
  - Only 22 pins are useable
  - Ten of those pins are GPIO pins
  - One of the GPIO pins is an analog pin

Introducing the ESP8266

- **NodeMCU** is an open source firmware and development kit
- This board has a USB-to-serial chip, but lacks a Li-Po battery charging circuit and a Li-Po battery connector
- NodeMCU development kit comes with 4 MB of flash memory, so it can support MicroPython, in addition to LUA and the Arduino core

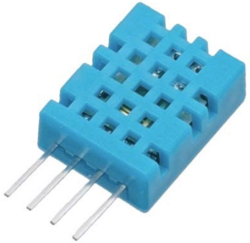


Introducing the ESP8266

Note

- Wi-Fi is not the only technology that we can use to connect our projects to the Internet
- There are other options such as Ethernet and 3G/LTE
- There are shields and breakout boards that can be used to add these features to open source projects

Sensors – DHT11 / DHT22



- DHT11 / DHT22 is a digital temperature and humidity sensor
- It uses a thermistor and capacitive humidity sensor to monitor the humidity and temperature of the surrounding air, and produces a digital signal on the data pin
- A digital pin on the ESP8266 can be used to read the data from the sensor data pin

DHT11

- Ultra low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion
- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50 C temperature readings ±2 C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

DHT22

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion
- Good for 0-100% humidity readings with 2-5% accuracy
- Good for -40 to 80 C temperature readings ±0.5 C accuracy
- No more than 0.5 Hz sampling rate (once every 2 seconds)
- Body size 15.1mm x 25mm x 7.7mm
- 4 pins with 0.1" spacing

Sensors – Photocell

- A photocell is a light sensor that changes its resistance depending on the amount of light
- can be used in a voltage divider setup to detect the amount of light
  - output of the voltage divider goes high when the light is bright and low when the light is dim
- output of the voltage divider is connected to an analog input pin and the voltage readings can be read

Sensors – Soil Humidity Sensor

- Soil humidity sensor is used for measuring the amount of moisture in soil and other similar materials
- has two large exposed pads that act as a variable resistor
- If there is more moisture in the soil, the resistance between the pads drops, leading to a higher output signal
- output signal is connected to an analog pin from where its value is read

Actuators – Relays

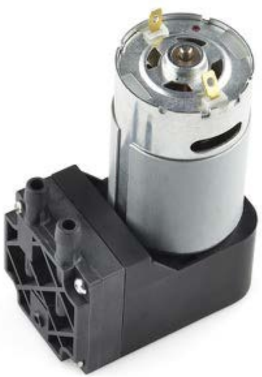
- **Relay** is a switch that is operated electrically. It uses electromagnetism to switch large loads using small voltages

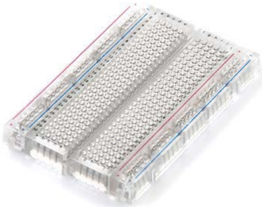
- When the coil is energized by a high signal from a digital pin on the ESP8266, it attracts the contacts, forcing them closed.
  - This completes the circuit and turns on the connected load

- When the signal on the digital pin goes low, the coil is no longer energized and the spring pulls the contacts apart.
  - This opens the circuit and turns off the connected load

Other Components

- Power switch tail kit
- Power switch tail kit
- Breadboard
- Jumper wires

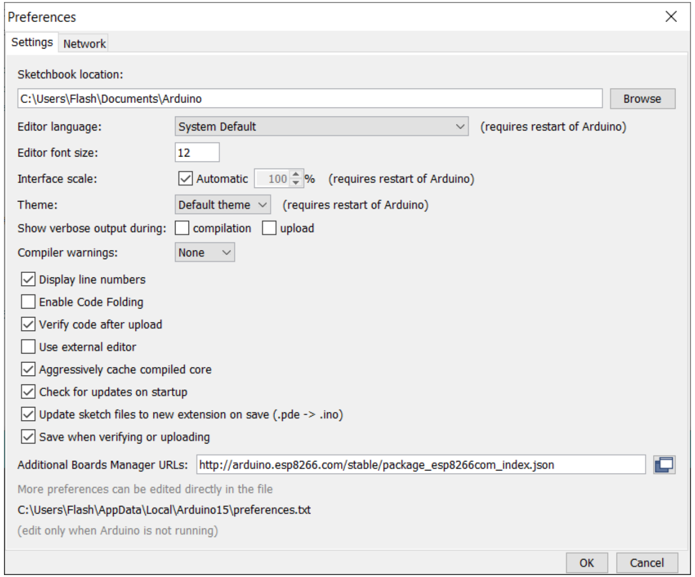




1.2 Uploading Your First Sketch

Getting Ready

- Open the preference window in the Arduino IDE from File|Preferences
- Copy this URL: http://arduino.esp8266.com/stable/package_esp8266com_ index.json
- Paste it in the file labeled Additional Board Manager URLs



Sketchbook location:
C:\Users\Flash\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: □ compilation □ upload

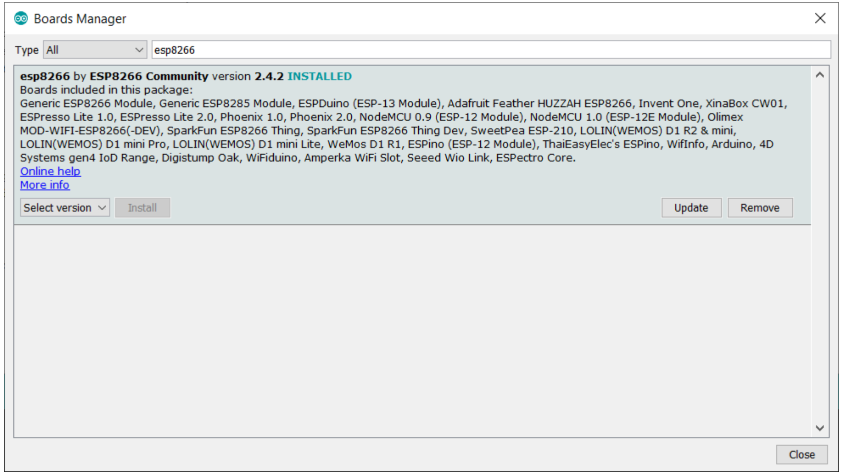
Compiler warnings: None

☑ Display line numbers
□ Enable Code Folding
☑ Verify code after upload
□ Use external editor
☑ Aggressively cache compiled core
☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading

Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json

More preferences can be edited directly in the file
C:\Users\Flash\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

- Open the board manager from the Tools|Board menu and install the ESP8266 platform



Type All esp8266

esp8266 by **ESP8266 Community** version **2.4.2** INSTALLED
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core.
Online help
More info

First Sketch – Blinking LED

- The program blinks an LED connected to pin 5 of the Adafruit ESP8266 board every 3 seconds
- This is done by turning off the LED for 1 second and turning on the LED for 2 seconds, continuously

```
// LED pin
int ledPin = 5;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // OFF
  digitalWrite(ledPin, LOW);
  delay(1000);
  // ON
  digitalWrite(ledPin, HIGH);
  delay(2000);
}
```

1.3 Connecting ESP8266 to WLAN

Connecting ESP8266 to WLAN

- We will include the ESP8266 library in the program and set the **Wi-Fi network ssid and password** so that the ESP8266 connects to the network when it gets powered on.
- We will print out a confirmation and the IP address of the ESP8266 when the connection is successful

```
// Libraries
#include <ESP8266WiFi.h>
// WiFi network
const char* ssid = "your-ssid";
const char* password = "your-password";
void setup() {
   // Start serial
   Serial.begin(115200);
   delay(10);
   // Connecting to a WiFi network
   Serial.println();
   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.println("IP address: ");
   Serial.println(WiFi.localIP());
}
void loop() {
}
```

Connecting ESP8266 to WLAN

- sets up the Wi-Fi SSID and password using the WiFi.begin() function of the ESP8266Wifi library
- executes the WiFi.status() function to try to connect to the Wi-Fi network.
- checks whether the WiFi.status() function returns a true value to indicate that the ESP8266 board has successfully connected to the Wi-Fi network.
- If the connection was not successful, the sketch tries to connect again using the WiFi. status() function and
- repeats that until the WiFi.status() function returns a true value to indicate that the ESP8266 has successfully connected to the Wi-Fi network.

1.4 Connecting ESP8266 to Cloud Server

Connecting ESP8266 to Cloud Server

1) We will connect the ESP8266 to a local Wi-Fi network that has an active Internet connection.

2) Once the connection is successful, we will send a GET request to the cloud server and then display the reply that the server sends back to the ESP8266 board:

```
// Libraries

#include <ESP8266WiFi.h>
```

3) Enter the SSID and password:

```
// SSID

const char* ssid = "your-ssid";

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

4) Store the hostname of the cloud server:

```
// Host

const char* host = "dweet.io";
```

Connecting ESP8266 to Cloud Server

5) Configure the SSID and password and connect the ESP8266 to Wi-Fi network:

```
void setup() {
  // Serial
  Serial.begin(115200);
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  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.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

Connecting ESP8266 to Cloud Server

6) Delay for five seconds and then print the name of the host we are connecting to on the serial monitor:

```
void loop() {
        delay(5000);
        Serial.print("connecting to ");
        Serial.println(host);
```

7) Connect to the host server:

```
        // Use WiFiClient class to create TCP connections
        WiFiClient client;
        const int httpPort = 80;
        if (!client.connect(host, httpPort)) {
           Serial.println("connection failed");
           return;
}
```

Connecting ESP8266 to Cloud Server

8) Formulate the URI for the GET request we will send to the host server:

```
// We now create a URI for the request
String url = "/dweet/for/my-thing-name?value=test";
```

9) Send the GET request to the server and check whether the request has been received or if it has timed out:

```
// Send request
Serial.print("Requesting URL: ");
Serial.println(url);
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                "Host: " + host + "\r\n" +
                "Connection: close\r\n\r\n");
unsigned long timeout = millis();
while (client.available() == 0) {
        if (millis() - timeout > 5000) {
          Serial.println(">>> Client Timeout !");
          client.stop();
          return;
        }
}
```

Connecting ESP8266 to Cloud Server

10) Read incoming data from the host server line by line and display the data on the serial monitor.

11) Close the connection after all the data has been received from the server:

```
  // Read all the lines from the answer
  while(client.available()){
     String line = client.readStringUntil('\r’);
     Serial.print(line);
  }
  // Close connecting
  Serial.println();
  Serial.println("closing connection");
}
```

Connecting ESP8266 to Cloud Server

- The program connects to the Wi-Fi network
- connect to the provided cloud/host server using the **client.connect()**

  function, and sends the provided URI to the host server using the **client.print()** function
- Once the data has been successfully sent, the sketch waits for a reply from the server
  - It does this with the **client.available()** function, which checks whether there is incoming data from the server
- If there is data available, the sketch reads it and displays it on the serial monitor
- The process is repeated until the ESP8266 is turned off

Reference:
Marco Schwartz, “ESP8266 Internet of Things Cookbook”,
Packt Publishing, 2017.