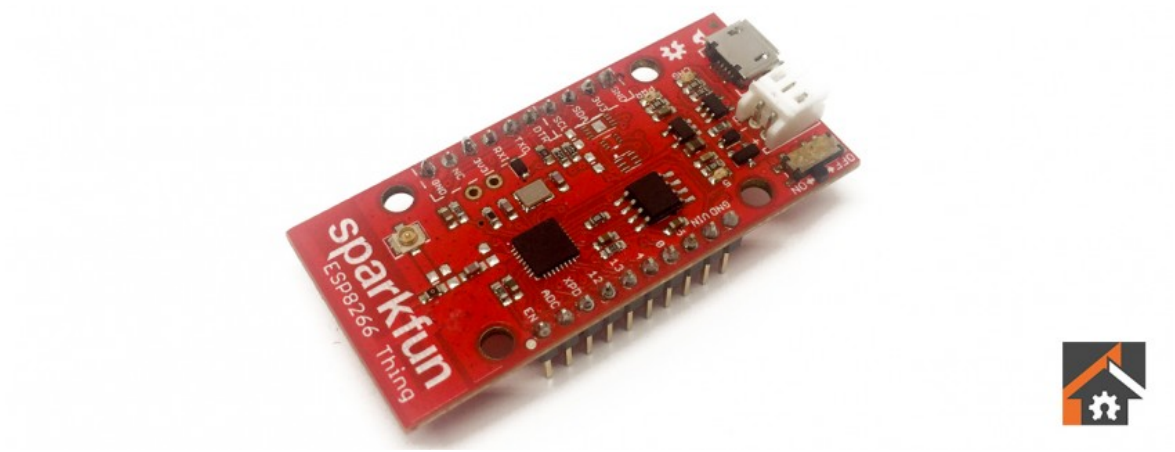


How to Run Your ESP8266 for Years on a Battery

April 25, 2016 / 0 Comments / in Energy, ESP8266 / by Marco Schwartz

For most of the projects I am building with the ESP8266 WiFi chip, I usually don't care too much about the power consumption aspect. I for example build data loggers that are constantly connected to the mains electricity, and appliances controller which also have an easy access to power. However, in some cases, we want to build projects that are only powered by batteries. This is for example the case for a motion sensor

that you will install in your home, or a data logger you would put in a remote location.

For those cases, you don't want to be changing the batteries constantly. For example, an ESP8266 chip with a standard 2500mAh LiPo battery would last for about 30 hours. Not good enough. That's why in this article, I will show you how to significantly reduce the power consumption of your ESP8266 boards using the deep sleep mode of the chip, so you can build projects that will last for years on a single battery. Let's dive in!

Hardware & Software Requirements

Let's first see what we need to build this project. The first thing you need is an ESP8266 board. Here, as we want the project to be low-power, the most important is to choose a board without a lot of features, so there are no extra components to reduce the battery life of your project. Here, I choose the SparkFun ESP8266 Thing [http://www.amazon.com/SparkFun-ESP8266-Thing/dp/B00YUU4AMK/ref=sr_1_1?ie=UTF8&qid=1460364142&sr=8-1&keywords=esp8266+thing] as it allows to work at very low powers.

You will also need a 3.3V FTDI USB adapter, as well as a breadboard and jumper wires. Optionally, to test the power consumption part, you will also need a breadboard power supply, a multimeter, and a LiPo battery.

This is the list of the required components for this project:

- SparkFun ESP8266 Thing [http://www.amazon.com/SparkFun-ESP8266-Thing/dp/B00YUU4AMK/ref=sr_1_1?ie=UTF8&qid=1460364142&sr=8-1&keywords=esp8266+thing]
- 3.3V FTDI USB adapter [http://www.amazon.com/XCSOURCE-FT232RL-Serial-Adapter-Arduino/dp/B00HSX3CXE/ref=sr_1_2?ie=UTF8&qid=1460364267&sr=8-2&keywords=ftdi+breakout]
- Breadboard [http://www.amazon.com/microtivity-IB400-400-point-Experiment-Breadboard/dp/B0084A7PI8/ref=sr_1_5?ie=UTF8&qid=1460364368&sr=8-5&keywords=breadboard]
- Jumper wires [http://www.amazon.com/Kalevel%C2%AE-120pcs-Multicolored-Female-Breadboard/dp/B00M5WLZDW/ref=sr_1_1?ie=UTF8&qid=1460364375&sr=8-1&keywords=jumper+wires]
- Breadboard power supply (optional) [http://www.amazon.com/Breadboard-Power-Supply-Stick-3-3V/dp/B0068QH7XS/ref=sr_1_14?ie=UTF8&qid=1460364207&sr=8-14&keywords=breadboard+power+supply]
- Multimeter (optional) [http://www.amazon.com/INNOVA-3320-Auto-Ranging-Digital-Multimeter/dp/B000EVYGZA/ref=sr_1_2?ie=UTF8&qid=1460364229&sr=8-2&keywords=multimeter]
- 2500 mAh LiPo battery (optional) [http://www.amazon.com/Ofeely-2500mAh-Lithium-Rechargeable-Accumulator/dp/B017CTEA3E/ref=sr_1_2?ie=UTF8&qid=1460369106&sr=8-2&keywords=lipo+2500+mah]

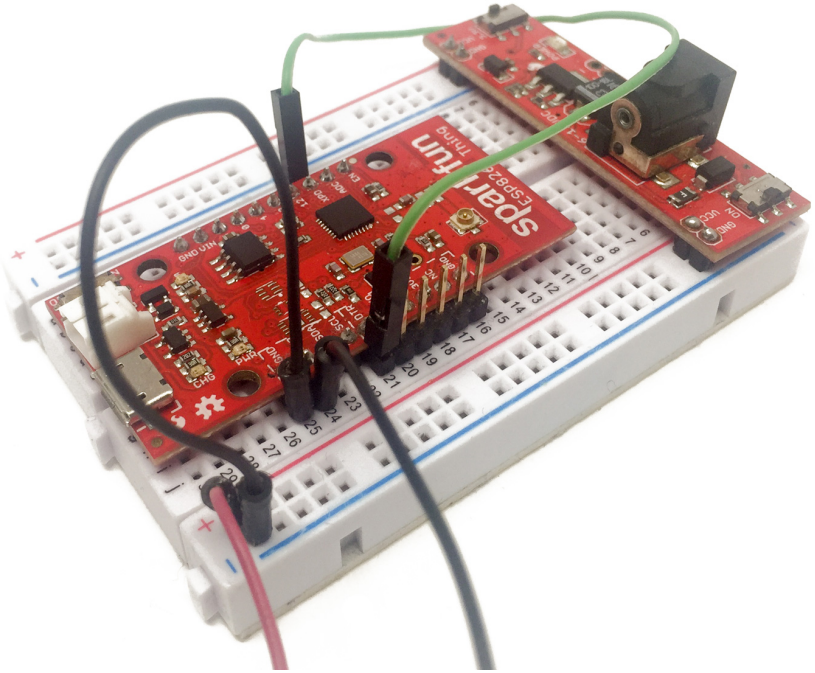
You will also need the latest version of the Arduino IDE, as well as the

ESP8266 board definitions.

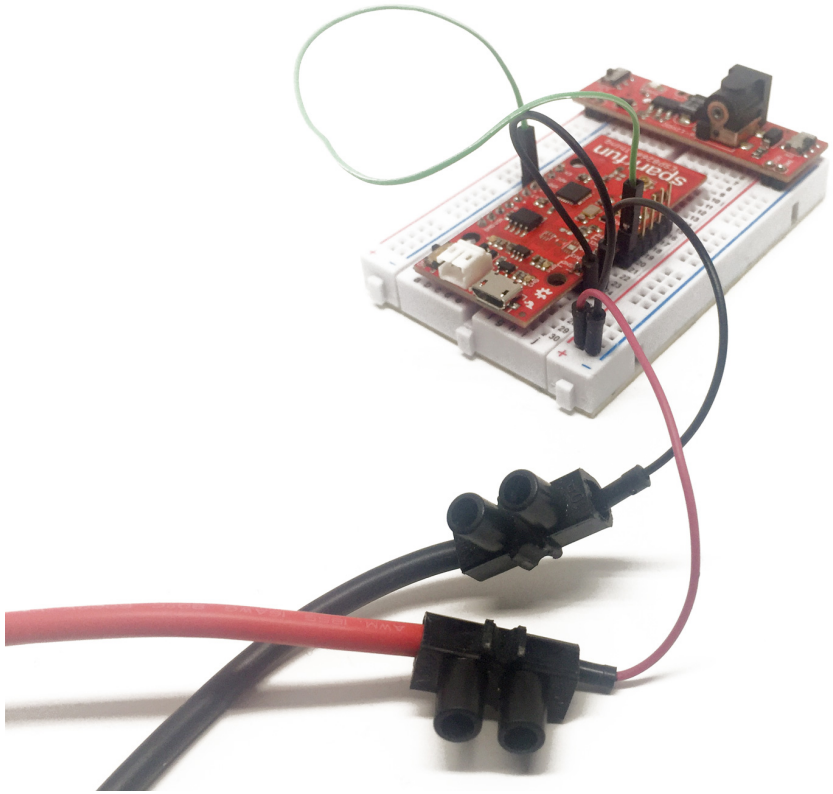
Hardware Configuration

Let's now assemble the project. As we just want to lower the power consumption of the board, the configuration will be quite simple here. If you just want to use the project with a low power consumption, you simply need to connect the DTR pin of the board to the XPD pin, which will make sure the chip can wake up from the deep sleep mode.

Here, I want to measure the power consumption as well, so I'll also be using a breadboard power supply, and connect the power to a multimeter so I can measure the current flowing through the chip. Here is a closeup picture of the project:



This is a picture from farther away, showing the connections to the multimeter:



Reducing the Power Consumption of Your ESP8266

We are now going to see how to lower the power consumption of your ESP8266 WiFi chip. To do that, we are going to use the deep sleep functions of the chip, that will simply sleep when no actions are required. As a simple example, we are going to log a simple dummy message to Dweet.io, which is a cloud service that is used to log data online. This will for example illustrate a data logger project that will only make measurements every 10 minutes for example, and sleep the rest of the

time.

This is the complete code for this part:

```
// Library
#include <ESP8266WiFi.h>

// WiFi settings
const char* ssid = "wifi-name";
const char* password = "wifi-password";

// Time to sleep (in seconds):
const int sleepTimeS = 10;

// Host
const char* host = "dweet.io";

void setup()
{

  // Serial
  Serial.begin(115200);
  Serial.println("ESP8266 in normal mode");

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
```

```
29.
30.     // Print the IP address
31.     Serial.println(WiFi.localIP());
32.
33.     // Logging data to cloud
34.     Serial.print("Connecting to ");
35.     Serial.println(host);
36.
37.     // Use WiFiClient class to create TCP connections
38.     WiFiClient client;
39.     const int httpPort = 80;
40.     if (!client.connect(host, httpPort)) {
41.       Serial.println("connection failed");
42.       return;
43.     }
44.
45.     // This will send the request to the server
46.     client.print(String("GET /dweet/for/myesp8266?message=lowpower") +
    " HTTP/1.1\r\n" +
47.                  "Host: " + host + "\r\n" +
48.                  "Connection: close\r\n\r\n");
49.     delay(10);
50.
51.     // Read all the lines of the reply from server and print them to
    Serial
52.     while(client.available()){
53.       String line = client.readStringUntil('\r');
54.       Serial.print(line);
55.     }
56.
57.     Serial.println();
58.     Serial.println("closing connection");
59.
```

```
    // Sleep
    Serial.println("ESP8266 in sleep mode");
    ESP.deepSleep(sleepTimeS * 1000000);

}

void loop()
{

}
```

This code is quite long, but let's now focus on what we need for the deep sleep functions. First, we define how long we want the chip to stay in deep sleep mode. For test purposes, I set it to 10 seconds here:

```
const int sleepTimeS = 10;
```

Then, inside the setup() function of the sketch, after sending the request to Dweet.io we put the chip in deep sleep mode:

```
Serial.println("ESP8266 in sleep mode");
ESP.deepSleep(sleepTimeS * 1000000);
```

Note that here we need to put the whole code inside the setup() function of the sketch, as whenever the chip goes out of deep sleep mode, it starts again at the start of the setup() function.

You can get the whole code from the GitHub repository of the project:

https://github.com/openhomeautomation/esp8266-battery
[https://github.com/openhomeautomation/esp8266-battery]

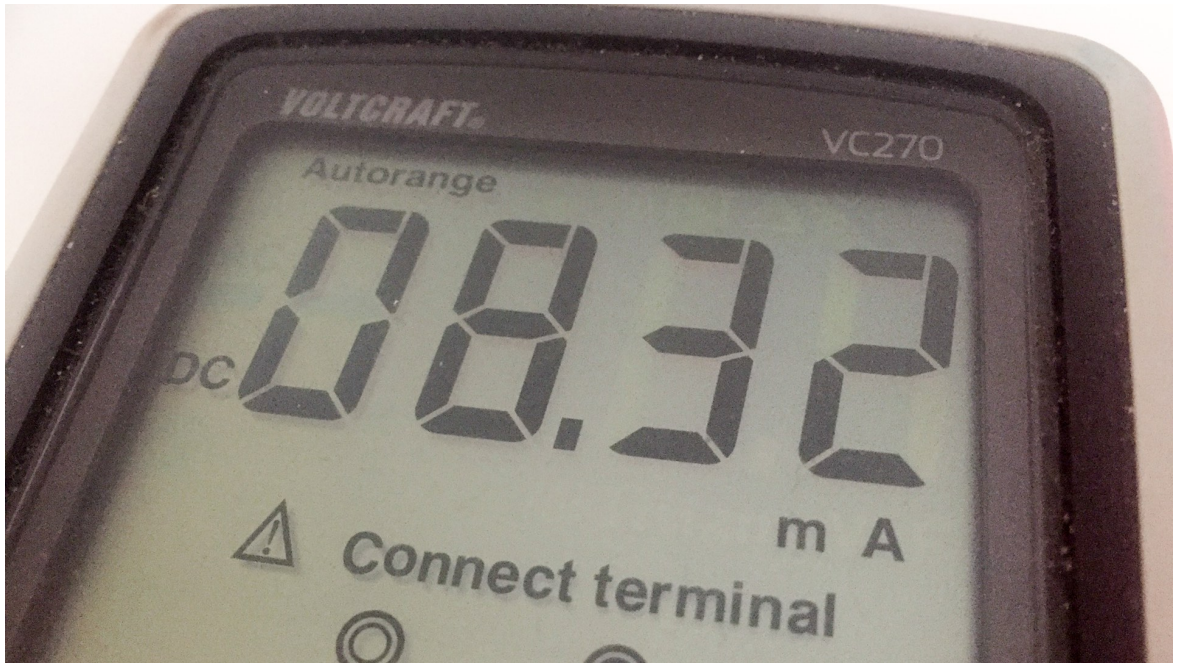
It's now time to test the project! First, remove the connection between DTR and XPD, so you can actually program the board. Also modify the WiFi credentials inside the code. Then, upload the code to the board, and connect the jumper cable again.

If you have a multimeter monitoring the current consumption of the ESP8266, this is what you first should see when the chip is booting:

This is the current that is used when the board is uploading data to Dweet.io, but it is also what the chip would use if we didn't do any kind of optimisation for power. In that case, a 2500 mAh battery would last

about 28.5 hours.

After a few seconds, the chip will enter deep sleep mode, and you should immediately see the power consumption going down:



As you can see, we already have a 10 times lower current consumption! At this rate, if we take the case of a data logger that just stays in sleep mode most of the time, the battery would now last 300 hours, or 12.5 days! It's already a great improvement, but we can do much more with the SparkFun Thing.

Actually, most of this power is now used by ... the power indicator LED on the board! This is great when you are developing applications on your desk, but not that useful when you are deploying your project in the field.

Therefore, we are going to get rid of this LED here.

For newest versions of the SparkFun thing, you can simply unsolder the “PWR” jumper at the back of the board. For older versions like the one I have, you can simply cut the trace between the PWR LED and the nearby resistor. After that, just power the project again. The reading on the multimeter immediately changed to 77 uA, or 0.077 mA. This means that the same project will now last on the same battery for … 3.7 years! Of course, this doesn’t take into account the characteristics of the battery, so in reality you will end up with 1-2 years battery life for your project.

How to Go Further

In this article, we learned how to reduce the power consumption of the ESP8266 WiFi chip, so you can build projects that last for years on a single battery. Of course, this is an ideal situation, and it can’t be applied to all projects, and in reality probably the battery will be dead before the time I calculated in the article. However, this is a great solution for anybody interested in data logging projects where the device is spending most of the time doing nothing.

You can now use what you learned in the project, and build your own projects with it. You can simply use the code I used in this article, and just add a few lines to make measurements from sensors, and send those measurements to Dweet.io or another cloud platform of your choice.

Note that this article is a continuation of the very popular article I wrote before about How to Run an Arduino for Years on a Battery

[https://www.openhomeautomation.net/arduino-battery/] .

Share this entry

0 Comments Open Home Automation 1 Login

Recommend Share Sort by Best

Start the discussion…

Be the first to comment.

ALSO ON **OPEN HOME AUTOMATION**

Wireless Security Camera with the Arduino Yun

15 comments • 2 years ago

Jonathan Oberreuter — What should I put on "sandbox"? uploadFileInputs.set_Root("sandbox")

How to Keep Hackers Out of Your Smart Home

1 comment • 3 months ago

Nicolas — You said that "Certain devices emphasize security as a top priority" ; do you have any exemple

Connect the ESP8266 WiFi Chip to your Raspberry Pi

18 comments • 6 months ago

Marco Schwartz — Hey! I'd check the resistor between pin 1 & 2, if 4.7k Ohm doesn't work try 10k

A REST API for Arduino & the CC3000 WiFi Chip

34 comments • 2 years ago

Chuan Mark Yang — Hello, I have a problem that I am using arduino mega 2560 and CC3000 wifi shield to upload

© 2015 Open Home Automation - Enfold Theme by Kriesi