In episode #47 I looked into the sleep mode of the ESP8266. There, a viewer asked, if it is possible to run an ESP from a small button cell and how long it would run. In this episode, I try to answer this question as well as build a sensor which sends mails and post the measured values to several IOT platforms like data.sparkfun.com and ubidots.com. And I will use the sleep mode of the ESP to extend the battery live. In this episode the sensor will measure and send values every 2 minutes. In the next episode it will measure and send values every day at a defined time. So, let’s start with the simple scenario:

First, I tried to run an ESP-12 with a normal CR2032 3V button cell. We remember, that the ESP8266 uses about 80mA when the Wi-Fi is on. Let’s look at the data sheet of these CR cells: Even if the voltage of 3V would be ok, they are made for very small currents and cannot be used for our purpose.

Fortunately, other button cells exist: Lithium Ion batteries LIR2450. Watchers of my mailbag #3 know already, that I ordered and tested such batteries. The have a capacity of about 100mAh and a nominal voltage of 3.7 Volt. They are able to deliver 80 mA for a time of ca 10 seconds without any problems.

So, let’s try to connect an ESP-07 or ESP-12 module to such a battery. I use an ESP-07 because I had it available on a small PCB. I desoldered the red LED because it is also on when the module is in deep sleep. It would use way too much energy. The ESP-12 does not have such an LED. I keep the blue LED because it is only on during very short periods.

If we look at the datasheet of the ESP8266 we see, that it works from 3.0 – 3.6 Volts. Now we have a problem: Fully charged Lipo batteries have a voltage of 4.1V. Our ESP chip is only rated up to 3.6V operating voltage. So, we have to regulate the voltage to 3.3 volt. If the Lipo has 4.1 volts, the voltage regulator is necessary because it has to drop 4.1 – 3.3V = 1.2V. If the battery is at its end, the regulator should not drop any voltage because we want to run the ESP as long as possible. And, of course, during deep sleep the regulator should use “no” current.

All linear voltage regulators need a difference between the input and the output voltage. This voltage is called “drop-out” voltage. Normal linear regulators like the LM1117 with 1.2V or the LM78L33 with 1.7V drop-out voltage cannot be used because this “drop-out” voltage is way too high for our purpose.

In addition, all regulators need a small current for their stabilization, even if no load is connected. This current is called “quiescent current” This current is 5-6mA for the LM1117 or the LM78L33. Also way too high for battery operation.

Fortunately, better regulators exist for this purpose. For my ESP modules I use a small PCB which has a space for a HT7333 linear voltage regulator. Let’s check its values: Its drop-out voltage is only 0.15V and its quiescent current is typical 4 uA. This seems to be a much better choice for a device with a small battery. So, let’s go with this one.

As we saw in episode #47 the ESP draws peaks of current up to 350mA. This is definitively too much for such a battery and the voltage drops during these current spikes. I recently bought some 10’000uf SMD tantalum capacitors. They were the biggest I was able to get. For SMD parts they are huge, but they fit nicely on my module. With this capacitor right across the supply pins of the ESP module I have no spikes anymore!

Now, we have a stable supply voltage and an ESP8266 module ready for the test. For this episode, I will not attach an external sensor because I want to get the behavior of the module itself. In order to have something to transfer, I measure the supply voltage and transfer its value to the cloud. The ESP has a mode to measure Vcc with the internal ADC. This can also be used to log the voltage over time or to send a mail where the sensor asks for battery replacement.

Next, we need a sketch to measure the voltage, send it to the cloud, and sleep afterwards for a pre-defined time. For this episode, I want to build a sensor which measures quite frequently. This is, why I can use the internal clock to time the deep sleep periods. Literature says, that this is not very precise, but if we just sleep for a few minutes or hours, this is not so important.

In the next episode I will build a sensor which only measures once every day. Then, the internal timer is no more precise enough and we have to use an “atomic clock” to get the needed precision. Stay tuned…

I found a project by Rainer Ochs which I was able to use for my purpose. He wrote this sketch to monitor water levels. You find a link to his project in the comments.

I used this sketch as a base and expanded it. He uses mail as his communication channel to the internet. This has two disadvantages for a sensor which should run on batteries: First, it takes about 30 seconds to send a mail and second, it is not easy to do anything with the measured values if they are distributed in mails.

To send mails, however, is good for some purposes. This is why I keep the mails to announce the start of a sensor, its daily “I am still ok” message, and the request for a new battery if voltage is too low. For the rest, I want to use IOT cloud services.

In episode #48 I used MQTT and io.adafruit.com as a service. Today I will use data.sparkfun.com and ubidots.com as services. Like that, you get a “library” of different possibilities for your projects.

Sparkfun, as Adafruit recently added an IOT service. It can store data of several fields. To create a stream, you go to data.sparkfun.com and create a stream. Give it a name and a description and add as many fields as you want.

Now, you can save it and get various keys and urls. I strongly suggest to download the json file because you need this data later and, if you do not have it, you cannot come back to this page.

Now you are ready to communicate with this service. To store a value, you only need one HTTP request. You find it in my code. Make sure, that every character matches. Even a forgotten space can make a difference and the whole thing does not work. It took me some time to get it right, but now you should be able to copy-paste.

The same apples with ubidots.com. This is also a nice service and also free of charge if you just want to try with one sensor.

If you want to send mails from your device, you can use the SMTP service of your own mail account. But this is not recommended, because currently, the ESP does not use secure connections. So, I use a free mail service called smtp2go.com. This is a nice service and you find the sketch for the ESP in my code. The only trick you need: The service expects your credentials in so called “base64” format. Fortunately, a web service exists to convert your username and password to this format.

In this episode I will focus on the current consumption and will not cover the software aspects. However, you already find a link in the description to the code. In future episodes I will cover the coding part.

For now, we have a sensor which sends the actual Vcc voltage to two IOT services and deep sleeps afterwards for two minutes. Do not forget to connect GPIO16 (or D0 on the NodeMCU boards) to the RST pin. Otherwise, the ESP will not wake-up and the deep sleep will last forever…

After wake-up the ESP-07 connects to the Wi-Fi network and calls two HTTP requests. This takes about 8 seconds. The biggest part of the time is needed to establish the connection to the Access point. During these 8 seconds the ESP draws about 80 mA. During the deep sleep, the ESP chip itself should only use 15uA. In my configuration with the voltage regulator it consumes about 70 uA at 4.1 volt and 40 uA at 3.5 volt battery voltage. Let’s assume it is 60uA in average.

We know, that the battery has about 100mAh which is 100 mA\* 3600seconds = 360’000 mAs. One cycle uses 80\*8 + 120\*0.06 mAs = 647 mAs. The battery therefore should last about 360’000 / 647 = 556 cycles. There are 30 cycles per hour. So, the battery should last about 18.5 hours.

Let’s check in reality: To visualize, you can export the values of data.sparkfun.com to a service called analog.io. And here, we see, that the sensor transmitted data for about 26h, more than expected.

If we would send a message every hour, the formula would be: 80 \* 8 + 3600\*0.06 mAs = 856 mAs. The battery should then last only 360’000/856 = 420 cycles. But this is 420 hours or 17 days.

Just for fun I replaced the Lipo cell with two AA batteries. Their capacity is about 2200 mAh and because their voltage does not exceed 3.6V we do not need a voltage regulator. So, they should last at least 25 times longer than the small button cell which means 26\*25h = 650h or 27 days. With the AA batteries it should last 17x25 days = 425 days which is way more than a year. Great!

<https://www.base64encode.org/>

http://www.esp8266.com/viewtopic.php?p=41638

https://github.com/SensorsIot/Longterm-Sensor-with-ESP8266--Minutes-