In video #101 I showed a principle on how to save energy for extremely low power devices using a mechanical switch. This principle is necessary, if your devices uses any sensors or circuitry other than the ESP itself. I used an IFTTT dash button as a simple example to show the principle. Since then, many viewers asked about the Dash button itself. So, today, I will finish this button, using another low power principle of the Arduino.

Let’s start: In video #101 I used a FET to switch the power of the whole device more or less completely off during non-activity. This is sure method, but, if you only have an ESP and nothing else, like in the IFTTT Dash button, you do not need that extra FET: You can do it with an ESP alone. And in addition, I suggested to use a ESP8285 because of its small size. In the meantime, I remembered my few ESP-01 from the early days of ESP8266 collecting dust because they only have one or two usable I/O pins. For this scenario, they are ideal. And for me, free of charge.

But how can we conserve energy without an additional FET?

The German journal CT in its last issue makes a proposal: Use the deep sleep of the ESP. The ESP is always powered. This is different to the concept of video #101. To start the send, they switch the reset pin with a mechanical switch to low and back to high. So, the ESP is started and does its thing to send a message. They use GPIO2 with a LED to signal success or failure. At the end, they set the ESP in deep sleep with the command ESP.deepsleep(0). Let’s now quickly consult the datasheet: Really, during deep- sleep, the ESP8266 should only use 10 µA. So, I quickly set-up a scenario with an old ESP-01. And, of course, I used Dave Jones µcurrent gold to get an exact measurement of the current in deep sleep. You see, the module uses about 40 µA during deep sleep. Which is much more than the 10 µA promised in the data sheet. But looking at the board, we see the reason for that current: The red “power” LED. If you remove it using a solder iron, the deep-sleep current goes down to 15 µA. Which is close to specs. Now we can do a short calculation:

If we use this LiFePo4 battery with a capacity of 700 mAh, or 700’000 µA we see, that the dash button should run:

700’000/20 µA =35’000 h / 24 = 1458 days / 365 = 4 years without pressing the button and without considering the self-discharge of the battery.

Which is ok for me…

So, the power problem is solved. The diagram looks now like that: We connect the Chip enable to VCC and the reset pin via a 10k pullup resistor also to VCC. The button is connected between reset and ground. The LED in the button is connected to GPIO2. And of course, we add a current limiting resistor of a few hundred ohms.

A viewer sent me this link: Here, they try to build a low power ESP connected to aPIR ensor, and a little later, also with only a button. Instead of deep sleep and the reset pin it uses the chip enable pin to reduce power consumption of the ESP. Lets first check, how much current the ESP uses with chip enable low: After removing all unnecessary parts, the current with chip enable low is also around 10 µA. But with the proposed diagram, I was only able to measure 1.2 mA, which is 100 times more. I did not find out, why. Maybe a viewer knows more about this phenomenon. So, I stick with the first concept.

Now, let’s continue with the sketch. Fortunately, after the last video, one of my viewers created a neat little IFTTT library which uses https. So, we can directly call IFTT. Great. Now, we just have to enter the credentials of our channel, and also add up to three values, if you want. And the whole thing “chooces”, as AVE would say. Just a small remark: The channel description is case sensitive. At the end, the LED blinks if successful and the ESP goes to deepsleep.

But this thing is not yet finished. Of course, I would have like to put this app to IOTappstory.com. But, because the first ESP-01 only had 512k, they cannot be used for OTA. So, the sketch is written the conventional way with credentials hardcoded. Because I will use several of these buttons, for example also to switch my Camera LEDs in the lab, I milled a small PCB. Then, I soldered the ESP-01, the button, and the resistors on this PCB. Of course, I added also a decent capacitor.

As a last step, I printed a small box. In the base of the box, I hot-glued battery contacts from another battery holder. I use a LiFePo4 battery to power the thing. If you do not know, why this is ideal. Watch my videos about battery types for small devices.

The First version of the box did not work. Everything worked fine until I put the ESP in the box. One in the box, it did not work anymore. First, I thought, it loses contact in the small box, but finally, I discovered, that always out of the box, it worked, an in the box, not. This shows us another lesson about Radio frequencies: If the antenna somehow is close to metallic sufaces, it is influenced. What means close? The higher the frequencies, the shorter the wavelength and also the shorter the antenna. The ESP has already a pretty small antenna. But if metal, and batteries consist of metal, is closer than the size of the antenna, you can expect trouble. This is, why I had to enlargen the box a little. And now it works also in the box.

And here it is: The IFTTT Dash button. If I press the button, the LED shows with a small blink, that the sketch works. After a while, the LED flags “success” and a little later, the mail from IFTTT arrives in the mailbox.

I hope, this video was useful or at least interesting for you. If true, please like!

<https://github.com/pinae/ESP8266-Dash>

https://github.com/witnessmenow/arduino-ifttt-maker

<http://www.esp8266.com/viewtopic.php?f=11&t=4458&start=28>

<https://www.heise.de/ct/>

<https://github.com/SensorsIot/IFTTT-Dash-Button>