Grüezi YouTubers. Here is the guy with the Swiss accent again.

In episode #58 we built a sensor which is capable to use the deepsleep functionality of the ESP8266, in episode #60 we were able to store values in the RTC memory, and in episode #67 we were able to get exact time from an NTP server. Today, we will combine these three features and create a sensor which samples a value every day exactly on the same minute.

But first a remark to the NTP service. One of my viewers pointed me to two functions which are natively implemented in the ESP SDK:

The implementation of the SNTP protocol, a simple version of the NTP protocol, and the time() function, which reads the internal ESP time based on the RTC timer.

These two functions have some advantages: First, they are already implemented in the SDK and second, the time() function gets the actual time without connecting to the internet every time. Because we know, that the ESP needs a lot of energy and also time for internet access, this is highly appreciated.

This does not mean, that the ESP has a realtime clock. It jus means, that, after getting the time once from an NTP server, we can use the internal clock. But keep in mind, the internal timer is not completely precise. I show you an example:

I read the time, wait for 10 seconds, and read the time again. We should see exactly the same number at the last place. This works for nearly an hour. But then, we start to see differences. This difference depends on the individual chip and probably also on ambient temperature etc. So, you can use this internal time for your projects if you can live with a slight drift. Keep in mind, that this time does not survive the deepsleep as we see here. After deepsleep, the time is always 0 UNIX time (which is 1st of January 1970).

But all-in-all, this is a very handy function if you just need the time.

To make it easy for you, again, I wrote a small library. This time, it has two different parts:

1. Set the internal time with the call setSNTPtime())
2. Read the time with the method getTime().

You need the settime method only at the beginning, probably in setup, and, if you want to run your sketch over a long time, also after a day or so. The timezone and the summertime adjustments are the same as in the NTPtimeESP library. I just changed the timezone variable to “double” format for my Indian viewers. India’s time zone differs 30 minutes and not a full hour.

But let’s now continue with our task to build a sensor which reads the value every day once.

I got the idea of creating this library from HarringayMakerSpace. You find a link to their library in the description.

The library is easy to use. I show you the usage with the help of the included example file:

As always, we include the library and create the object ESPDailyTaskNTP dailyTask().

The first two parameters define the hour and the minute of the planned task. The next two readings are the credentials of your Wi-Fi and the last parameter defines the reset pin. If you ground this pin, the timer is reset and starts from scratch. The reset pin parameter is optional. You do not need it if you can accept that the first daily task does not start at the exact time.

The setup is very simple: If you want to get the debug messages, you start the serial connection. Otherwise you just need three parts:

The first line, dailyTask.sleepOneDay(); checks if the time to execute the task has arrived. If not, it sends the ESP straight to deepsleep again. Normally, this function does even not switch the Wi-Fi on and takes just a few milliseconds.

If it is time to execute the daily task, the next lines are called. This is, where you place your coding. To show you the working of the sketch I read here the actual NTP time and display it in serial. At the end, I post it to data.sparkfun.com. If you run this sketch, it also will post it to sparkfun. Please fill in your YouTube name and I will see who used the sketch.

Like that, we can check every day if the task was executed at the right time.

After this code, we have to send the ESP back to sleep again. Done.

As usual, if you only want to use the library you can stop now and get a beer or so. If you want to know, how this “miracle” is done, you can stay on.

Again, we now go to the library. The function sleeponeday is the central part of it. As usual, I programmed it as a state machine: The states of this machine are: RESET, COUNTING, CHECK, and WORK. These states are all in one case statement and the status is stored in RTC memory to survive the deep sleep. Details about this function you find in episode #60.

Some other variables also survive in the RTC memory:

A marker flag which I set here to 85. This flag is used to make sure, that the other variables in RTC memory originate from this sketch and not another one. You can use whatever marker you want. Just make sure, that each of your sketches use a different one.

The second variable is a counter which counts the hours. Unfortunately, the ESP8266 only can deepsleep for about one hour. Then, it has to wake up. This counter counts, how often the ESP has to wake-up till the 24 hours are through.

So, let’s look what happens if we call the method sleeponeday: First, the RTC memory variables are read. And second, we check if a reset is necessary. A reset is necessary if the reset pin is low or if the marker is not found.

After that, the code switches to the right case, based on the status variable.

The reset just initializes the RTC variables and send the ESP to deep sleep for one microsecond. It also defines, that the ESP has to wake-up with the Wi-Fi on. After this microsecond the ESP wakes up with the status CHECK.

In the status CHECK we check the current time by calling the NTP server and compare it with the time defined for the recurring task. If this difference is shorter than one hour, the ESP can sleep for this time and wake up directly in the status WORK. This is possible, because we saw before, that the RTC is exact enough to count one hour.

In the status WORK the only thing we do is, not to go to sleep. If we do not go to sleep, we go on in our sketch and execute the daily task. After this task, we call the method backtosleep().

After our daily task, we know that we have to wait 24 hours. This is, why we can sleep for one hour and set the counter to 23. During the next wake-up we just need to go to the COUNTING status and do not need Wi-Fi.

The status COUNTING finally checks, if we are close to the time for the next daily task. If so, it sets the status CHECK to make sure we call the NTP time and set the needed sleep time to wake-up at exactly the right time. And then, the whole story starts over. Simple. The only thing we have to do is, to shorten the time for one hour a bit that we are for sure a little before the exact time. If you are familiar with the Swiss Railway clocks you know this principle. The seconds hands run a little faster and waits one second before the minute till the exact pulse arrives from a central master clock. This made it possible in early days to have exact time in all Swiss cities. And maybe, this is the basis for the famous Swiss precision…

I hope, this episode was useful or at least interesting for you. Bye