In Episode #58 we learned how to use the deep sleep mode of the ESP8266. This is very valuable if you want to run it on batteries or solar power. It works fine and we could extend the operation time to weeks or even years. Unfortunately, the ESP loses all memory content during deep Sleep mode. This fact prevents us from using it for all applications where we need to keep some data during deep sleep. In this video, we will learn how to overcome this problem and use it to extend the operation time on a battery by another factor of 10.

We could try to store data which has to survive the deep sleep in the EEPROM. Unfortunately, the ESP stores the EEPROM data in flash memory and this memory technology wears out after a few thousand write cycles. While this is no problem for storage of programs, it can quickly become a problem for counters which have to be written every hour or even every minute.

But fortunately, another possibility exists. Even in deep sleep mode, the ESP has to keep one part alive. It is called real time clock or RTC. This RTC is used to clock the deep sleep duration and wake the ESP after the pre-defined time.

Fortunately, the engineers have given us a goody in addition to the RTC clock. They included some memory called “RTC memory” which our sketches can use. In the ESP8266 SDK API Guide we find the needed information. So, let’s start to find out how it works:

The RTC memory is 768 bytes long and it is arranged in 4 byte buckets. The first 256 bytes are used by the system. The other 512 bytes can be used by us.

Because this memory has to be written in buckets of 4 we cannot use it like normal memory. To overcome this problem, I propose to work with data structures for our variables we want to store in RTC memory.

But, what is a structure? It is a very simple principle. Let’s assume, we want to store 2 variables in RTC memory: battery and other. So, we define our variables as normal but “typedef” them as a “structure”. Just that you are not surprised: Integer values are stored as 4 bytes in the ESP8266. The compiler then places the variables one after the other in memory as junk of 8 bytes. So, we can read or write a whole data set with one command by just using the structure. So, we can, for example, place the next structure just behind the first one and so on. We have to stop if we reach our 512 bytes or 128 buckets. With our structure definition as an example, we can place 64 measuring results in RTC memory.

After the definition, we have to initialize a variable in our sketch. I call it rtcMem. Now we are ready to store data.

To use the RTC memory, we need to include the C file "user\_interface.h" right at the beginning of our sketch. This file includes the two important methods for reading from and writing to the RTC memory.

Now let’s try if the RTC memory works as described. To test it, I plan to write a number into each byte of the available memory, read it back later, and compare the result. If the numbers fit, the memory seems to work. I do this in two loops, so we can later add a deep sleep phase between the write and the read.

The program is straight forward. Because I only want to run it once I include everything in setup()

First, I define the start and the length of the memory in buckets. The memory starts at bucket 64 and ends after 128 buckets.

Then, we have to calculate how many buckets our structure uses. Sizeof(structure) delivers the size used by the structure in bytes. Divided by 4 equals the number of buckets needed for each structure element. In our case, it is 2.

Then, we loop through all available RTC memory and write numbers into the two fields. The two numbers do not matter; they just have to be different. Variables inside a structure have to be accessed a little different than normal variables. We always have to put the name of the structure in front of the variable name. That’s all.

Then, we calculate where we have to place the particular data set in memory and use the command system\_rtc\_mem\_write() to write the structure to the memory. We write the whole structure in one as requested by Espressif. Of course, we do not forget the yield() command to keep the watchdog happy.

In the following “for” loop, we read the memory with the function system\_rtc\_mem\_read().

Now, we can compare the two results and see, if our RTC memory delivers exactly the same numbers as stored before.

Now let’s test if the RTC memory really survives the deep sleep. For this experiment, we just add the command ESP.deepSleep(5000000, WAKE\_RFCAL). But wait: The ESP boots after every deep sleep. And just after booting, the ESP writes the numbers again into memory. So, we are not able to check if the memory values really survived the deep sleep. What to do?

We could introduce a toggle Flag and if the flag is false, we fill the memory and if it is true, we read it. So, we have to scarify one bucket for this flag. I place it just at the beginning at position 64. So, our measurements can only start at 65.

And the result is good. The ESP fills the RTC memory, goes to sleep for 5 seconds, and, after wake up, reads the values. And they are still the same.

So, we can now store and retrieve data in from and to the RTC memory.

Now let’s use what we learned to improve the sketch from video #58. In this video, we built a sensor which read the battery voltage and transmitted it to sparkfun.com. After that, it went to sleep for 2 minutes. We saw, that nearly all energy was used for the establishing of the Wi-Fi connection.

With our new knowledge we can measure 63 values and store them in the RTC memory without starting Wi-Fi. Only every 128 minutes we have to start Wi-Fi and transmit all in one dash. Sparkfun’s reaction time was quite slow. In average it took 3.5 seconds per line. The fastest postings were well below a second. In video #58, the transmission of one value took us about 8 seconds So, with a fast cloud service with response times below one second, we can reduce the power-intensive phase by a factor of 10. So, our sensor will run again much, much longer without losing any data. Cool!

As usual, you find a link to the code in the comment.

In one of the next episodes we will use the rtc timer to make the ESP sleep really long.

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

<https://github.com/HarringayMakerSpace/ESPDailyTask>

file:///C:/Users/Andreas/Downloads/2C-SDK-Espressif%20IoT%20SDK%20Programming%20Guide\_v1.0.0.pdf