Grüezi YouTubers. Here is the guy with the Swiss accent. With a new episode around sensors and microcontrollers.

Last year we worked with the deep-sleep function of the ESP8266. In video #147 I started to look into the new ESP32. And of course, we are interested if and how the deep sleep works, especially because the ESP32 has a so-called ultra-low power coprocessor. This fact promises new possibilities.

So, lets dig into the matter to find out:

* If we can deep-sleep the ESP32
* If we are able to wake it up
* What kind of possibilities we have to trigger the wake-up

But first, let’s get an overview: The ESP32 chip has 2 processors, the main, and the ultra-low power, or ULP processor. And it has plenty of pins, which can be used by these two processors. As we saw in video #147, most of the ESP32 pins have many different functions, which can be quite confusing if you look at the data sheet. This is, why I tried to sort them and put them in an Excel sheet. But I did not use the pins of the ESP32 chip itself for this chart, I used the pinout of the WROOM-32 module, which is on most of our development boards. Using this sheet, you can sort the pins along different lines. If you are for example interested in the GPIOs, you sort it according the GPIOs. Or according the ADC pins. You find a link to the sheet below in the comments.

Some pins can also be used by the ULP processor, they are called RTC\_GPIOs. Lets’ sort our table along these pins: We discover 17 pins, RTC\_GPIO1 and 2 are not available on the WROOM-32 module. Next, we look at the touch pins. We find 10 touch sensors. And they are also connected to the ULP area. These pins will be used later on.

Now, let’s move on to the power modes of the ESP32. It supports five different modes from active where everything is “ON” to hibernation, where everything is “OFF”. Today, we concentrate on two modes: Active and deep-sleep. And we see, that even during deep-sleep, the ULP processor can be kept on. Cool. But how much current does the module consume in this state? According the Data sheet, it is 0.15 mA or 150 µA. But how can we wake the ESP32 up from its deep-sleep? With the ESP8266 we basically had two possibilities: Either reset the chip from an external source or connect GPIO16 to the reset pin for a timed wake-up. And, with the timer, we were only able to sleep for about one hour.

The ESP32 has four different wake-up sources:

* The timer
* Two possibilities of external Wake-ups
* The touch pads
* ULP coprocessor wake-up

Today, we will only cover the three first possibilities, because I do not know how to program the ULP processor with the Arduino IDE.

The timer wake-up is similar to the same mode in the ESP8266. We define how long the ESP32 should sleep and next, we start the sleep mode.

In the definition statement, we define the time to sleep in microseconds, not in milliseconds as with the ESP8266. And the format of this number is uint64\_t, which means, that we could define a sleep duration of 584’942 years, not only 1 hour. And the second important difference is, that we do not need an external connection between GPIO16 and the reset pin as with our ESP8266. It works just out of the box.

Next, we start the deep-sleep. Now, the ESP32, as the ESP8266, sleeps and, when the wake-up call comes, starts with the execution of the setup() part of our sketch. Our sketch never reaches the loop statement and we have to write our programs completely different if we use deep-sleep than a normal sketch.

Here, we see another difference to the ESP8266: We get good information on the boot procedure. For example, we see, that the reset was caused by deep-sleep. And the wake-up was caused by the timer. We will see, that we even get more information later on.

You might have overseen another very cool new feature: We can use the RTC memory without a library. We just prefix the variable definition with RTC\_DATA\_ATTR and we do not lose data during deep sleep. Normal variables are lost, because, as I said, our sketch starts each time with setup(). This was much more complicated with the ESP8266 and it was even worth a whole video, video #60…

The ESP32 has 8 KB SRAM in the RTC part, which is called RTC FAST Memory. It survives deep-sleep, but not pressing the reset button, as we see here: The number of boots counts up till I press the reset button. Then, it is set to zero.

The first wake-up source had already many advantages over the ESP8266. Will this continue with the other sources?

The next is wake-up by an external source. This was also possible with the ESP8266 by resetting the chip. Here, we get much more possibilities: We can wake-up the chip by one particular pin only. Let’s assume, we have our IFTTT dash button from video #108 where we have just one button and we want to wake our ESP32 by pressing this button. To achieve this, we have to select the ext0 mode and tell the chip, which pin is connected to the button. And in addition, we can decide, if we want to trigger the wake-up by a low or a high state of the pin.

To find-out the pin number is a little tricky: We have to use the GPIO number of the pin, not the RTC\_GPIO pin number. Fortunately, we still have our Excel sheet and we can see, that RTC\_GPIO00 and GPIO36 are connected to the same pin on the WROOM-32 module. But on our LoLin board, we do not find a GPIO36 pin. Did they forget it? Fortunately, not. But it has a different name on the silk screen: It’s called VP. The secret behind this is here: Another function of this pin is also SENSOR\_VP… By the way, the same applies for pin GPIO3 which is named VN.

One thing is important: I had, depending on the interrupt polarity, to connect an external pull-up or down resistor to the pin. I used a 100k resistor and it worked.

But what, if we would like to have several buttons on our IFTTT device? Also here, our new toy offers an elegant solution: we use the ext1 wake-up source. In this mode, we can use several pins for wake-up. Because we can declare more than one pin, we have to define a so called “mask” instead of only a pin. The easiest way to understand this “mask” is to write it in binary format: A lot of zeros and a “1” for each GPIO you want to enable as a wake-up source. The numbering, again, is based on the normal GPIO numbering. To my knowledge, 8 pins, from GPIO32 to GPIO39 could be used, but GPIO37 and 38 are not available on the WROOM-32 module. So, we still have 6 pins left. Because the first pin available is GPIO32, the mask contains 32 times “0” on the right. Then, for each enabled pin we write a “1”. If you do not want to use a particular GPIO for wake-up, you have to write a “0” at this place. And in hexadecimal, this number is 0xFF00000000.

The definition call for this mode is like that: esp\_deep\_sleep\_enable\_ext1\_wakeup(BUTTON\_PIN\_BITMASK, ESP\_EXT1\_WAKEUP\_ANY\_HIGH);

After wake-up, you see the source of the wake-up and you can check, which pin was used. The corresponding call is:

esp\_deep\_sleep\_get\_wakeup\_cause(); Pay attention: The return value is a 64 bit integer and cannot be printed directly in the Arduino IDE. You have to “cast” it into a 32 bit integer for printing. The result is shown here:

And with this information, your ESP32 can decide if you pressed the button to order toilet paper or a bottle of wine…

So far , we covered two of the three wake-up sources. The last one is the touch wake-up. Also this wake-up works somehow similar. The deep sleep is started with the same command, and after deep-sleep, the ESP32 also starts with the execution of setup().

The definition of the touch pin, however is different: This source uses touchInterrupts() and you attach the interrupt of each touch sensor you want to use to a “callback function”. If you do not attach the interrupt, this sensor is inactive. In addition, you have to enable touch pad wakeups.

Now, you can use the touch sensors of my last video to wake-up the ESP32 with a touch of a finger. Cool.

Summarized,

* We were able to deep-sleep the ESP32 to save lots of power for our battery operated devices
* We were able to use three out of four sources for wake-up:
  + The timer
  + Two possibilities of external wake-ups
  + Touch pads
* We looked at the datasheet and saw, that the deep-sleep current should be around 150 µA
* We discovered, that not all ESP-32 pins are available on the WROOM-32 module, and some are named strangely
* By chance, we discovered how simple it is to use RTC memory which survives deep-sleep
* And we discovered, that we can deep-sleep the ESP32 for years, not only for an hour like the ESP8266

In a later video, we will then measure the real current used by the WROOM-32 module including the flash chip and all other parts.

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

In this video, we look into the deep-sleep functionality if the ESP32, and we are interested if and how the deep sleep works, especially because the ESP32 has a so-called ultra-low power coprocessor. This fact promises new possibilities.

* We will deep-sleep the ESP32 to save lots of power for our battery-operated devices
* We will use three out of four sources for wake-up:
  + The timer
  + Two possibilities of external wake-ups
  + Touch pads
* We have a look at the available (and sometimes strangely named) ESP-32 pins of the WROOM-32 module
* We use RTC memory which survives deep-sleep
* And we calculate, how long the ESP32 can deep-sleep compared to the ESP8266

Links:

Sketches: <https://github.com/SensorsIot/ESP32-Deep-Sleep>

<http://esp32.schaeffer.tk/singlehtml/index.html>

Details:

<http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep.html>

<https://media.readthedocs.org/pdf/esp-idf-fork/latest/esp-idf-fork.pdf>

<https://github.com/espressif/esp-idf/blob/master/docs/api-reference/system/deep_sleep.rst>

Supporting Material and Blog Page: <http://www.sensorsiot.org>

Github: https://www.github.com/sensorsiot

If you want to support the channel and buy from Banggood use this link to start your shopping: <https://bit.ly/2jAQEf4> (no additional charges for you)

Official Wemos Store: http://s.click.aliexpress.com/e/jUzBiIq

https://www.facebook.com/SensorsIOT/

<https://twitter.com/spiessa>