This is a very important episode. If you want to be part of the next wave of ESP microcontrollers, it might be a good thing to like this video and subscribe the channel.

The ESP32 is the new kid in the block. And I mean it as it is defined by the Cambridge dictionary. In this video, I first will compare the ESP8266 with the ESP32. Then, I will show you how to set-up the programming environment for the new ESP32 board, and will do some examples to show what works, and what not yet. Then, I will show you also how to write sketches which compile with the ESP8266 and the ESP32 without code change. And at the end, you will get my opinion, if you stay till there.

Let’s begin without further ado.

A few viewers asked about the difference between the ESP8266 and the ESP32 and whether the whole buzz has a base. So, lets quickly compare the two chips:

Let’s start with the block diagram of the ESP8266. It basically has two parts: The “radio Frequency”, or RF part and the logical part. On the left, we see the RF part which basically makes sure, that the chip can connect to Wi-Fi. The logical part on the right side consists, as every micro processing unit of the processor itself, of RAM memory, and some interfaces.

This architecture, together with the 32-pin package, is the reason for some limitations we know when using the ESP8266:

The first is, that the only processor has to deal with Wi-Fi and our sketch in parallel. This is, why we have to pay attention, that we do not block the only processor with our sketch. Otherwise it cannot serve the Wi-Fi stuff and crashes. This is the reason for the famous wdt resets and the “yield” command.

The second limitation is, that we only have a few Pins left for our usage. And many of them have special usage during the boot process like GPOI0, GPIO2, or GPIO16. If we do not pay attention on the logic levels of these pins, our device will not start properly and stay in blocked states. So, we fast run out of pins in projects.

The third limitation is, that we, in order to save power, can switch off the RF part or in addition to that, also set the logic part into deep sleep. The only way to wake the processing unit is by a reset signal, either created by the RTC of the ESP on pin16, or by an external signal. The processor itself is completely switched off during this time and we completely loose connectivity. We have to reconnect to Wi-Fi, which always takes a few seconds.

Fortunately, we already have some RTC memory which survives this deep sleep phase.

The speed of the processor is already fast, much faster than anything we were used in the old Arduino technology, and also the RAM and ROM capacity is usually no limitation.

So, lets now look at the block diagram of our new darling. The whole RF part is just three small rectangles. This RF part still works on the 2.4 GHz band as with the old ESP and supports all relevant transmission modes. In addition, it has a built-in encryption module which supports all kind of secure communications, which is very important for future IOT devices.

If we look at the processor, we see, that it got now two processors instead of one. I will check in a future video how fast they are in comparison with the old ones. But this is less important that the fact, that we now have one processor dedicated to communication. So, we can use the second one all the time without blocking communication. I hope, the yield command will be a thing of the past…

The new chip has 48 pins. At first sight, these 16 pins do not seem much more. But if we consider, that all processors have to have a number of basic pins like ground, Vcc, Antenna, crystal, etc., we see, that we maybe have three times the number of pins for our usage. This shows also the block with the different interfaces supported by the new chip. There are up to 18 12-bit ADC channels, two 8 bit Digital to analog converters, and so on. The list is very impressive and I foresee, that we will not have many limitations in this area.

The chip also got a so called ULP, an ultra-low power processor. This part enables the possibility to “deep sleep” the main two cores, but still keep Wi-Fi communication alive, and check wake-up events on other pins. In this mode, the chip only uses 150 µA, which should enable long lasting connected devices.

And the last, completely new section is Bluetooth. The ESP32 also supports the 4.2 BLE standard, which enables communication with many sensors on a short distance. Now, we not only have Wi-Fi, but we also have Bluetooth communication.

If you think, that such a device will cost a fortune, you are wrong. After a while, its price will be in the same sub 10$ range as the ESP8266. What a wonderful world!

But now I have to hurry up. Otherwise, the video gets too long. Let’s continue with the installation.

First, we have to find the new toolchain to program tis new kid. With the ESP8266 it was quite easy. We just had to enter an address in the preferences of the Arduino IDE and the rest was done automatically. We even got a pop-up reminding us if a new version was available. With the ESP32, we are not there yet. The good news, however, is: Ivan Grokhotkov, the mastermind behind the success of the ESP8266 on the Arduino IDE, joined Espressif, the designer of the ESP chips. So, the new toolchain is now provided by Espressif directly. This is, why we find it on github in the repository of Espressif, not under the name “ESP32”. In the repository, we find a description of what we have to do. Unfortunately, this description has an error, and I will show you how to overcome it.

I will only show the windows installation. But I think, the MAC installation will be similar.

So, we branch into the instructions for Windows:

First, it is recommended to upgrade to the newest Arduino IDE. You find this download on Arduino.cc and the installation is straight forward. If you never donated something for this platform, it might be a good time to do so, because the developers of the Arduino IDE will get nothing if we use their tools to program an ESP32.

After the installation or upgrade of the Arduino IDE, we have to install Python 2.7. The link is provided here. Make sure, you use the 2.7 version and not the 3.5 version.

Python is used to upload your sketch to the ESP32. If you already used ESP8266, the chance that you have Python installed is very high and you can skip this step.

The next step, however, is mandatory. We have to install a tool called “GIT”. Its name does not by change resemble the name Github. It is actually a client on your PC which downloads whole repositories from github (and many more things which are not interesting for our today’s task).

After installing this tool, we have to type in “Git bash” in the search field of your Windows 10 and, with a right click, start it as administrator. It will not work if you start it as a user.

It starts a command console where we can copy and paste the next commands to install pyserial.

After we did this successfully, we come to the next step. Keep the console window open in administrator mode, not as written, in normal mode and copy and paste each line separately. It will not work, if you copy and paste the block as a whole. Nothing will happen till you entered the last line.

But then, the whole toolchain to program the ESP32 will be downloaded onto your PC. Unfortunately, in a location which does not work. This becomes visible, if you start your Arduino IDE and search for the new ESP boards. At least in my case, they were not there. So, I had to copy the whole Espressif directory to the right place in the programs folder. You see here, how I did this. If I start the IDE now, we see the new boards. Hallelujah! Now, you can delete the files in the original directory.

Next, you have to have an ESP32 board. And this was a problem for me till yesterday evening. I had ordered some boards at different places, but so far, did not receiver one. This is also the reason, amongst others, why I did no video on the ESP32 in the Arduino world. Yesterday evening, I was at a meeting organized by the IOT guys from Zurich, where Thomas gave me his only board for this video! Thank you very much, Thomas. For the Swiss guys, I post the address of theses Meetups in the description below. Maybe we see each other in one of the next meetings, either in Zurich or in Basel…

After plugging the board from Adafruit into the USB connector, it is immediately recognized, because it uses a Silab USB to serial converter.

So, the fun can start. Because I have made some changes on the Sonoff Receiver script of video #97 it opened up when I started the IDE. So, let’s use this example to check, if it runs also on the ESP32 and if we have some compatibility issues between the two ESP chips. The Webpage said, that we do not have to expect a lot of problems. 99% should be the same for the WiFi functionality. And this sketch consists mainly of WiFi and some logic.

So, lets compile the thing. Unfortunately, nothing goes. More error messages than the code itself. So, I think, the fun has to wait a little and we need hard work before it starts. So, I try now exactly the contrary: I use the simplest example sketch provided by the toolchain. It is called “WiFi Client”. So, let’s start this one. After I created a new channel on data.Sparkfun.com, I entered all the needed credentials in the sketch and pressed “upload”. The first thing we see is, that the messages of the compiler are completely different and our well-known row of point during uploading is gone. But, the compilation time is fast and, this time, without any errors. Great! Let’s now fire up Serial and look, what the thing does. Overwhelming, the rich context of the messages, without typing one serial.print statement. So, the device connects to my WLAN and calls Sparkfun. Unfortunately, that’s all. The sketch is now blocked and Sparkfun did not get the message. So, the fun factor starts to decrease more and more. Some investigations show, that this problem is known and is a result of fast publishing new versions of code. I am sure, it will be solved shortly with a new version. And, as usual, this new version will contain other bugs. And so on, and so on. This is what we call the “bleeding edge” of software development, where we have to expect some pain and suffering. This in contryry to the next phase, which is usually called the “Leading edge”.

Do not misunderstand me: This has to be expected in this stage of development. But it shows also, if you are not interested in the technology alone, and you want to build something, you stick with the ESP8266 for another few months and probably join for the “leading edge” phase.

But because we all know, that we will use the ESP32 in the future, I show you now a small trick on how you can start to prepare your ESP8266 code that it will compile without changes also for the ESP32.

In the “boards.txt” file of the ESP32 environment, you find the name of a variable. For the Adafruit board, we had to choose the “ESP Dev Module”. If we search for this string, we find the ESP32.name constant. And little below, we find the “esp32.build.board” constant which has the value “ESP32\_DEV”. If we add the string “Arduino\_” in front of it, we can advise the compiler to test it, and, according its value, use the instructions between the if-then-else statements.

So, we advise the compiler to load the “WiFi.h” library for the ESP32 and the “ESP8266WiFi.h” file for all other boards.

We can use this also for pin definitions, for example. And If you start now to include these things, you can, from time to time, compile your files for the ESP32 and check, how many error messages you get. And, if it compiles completely, upload it to your board. By the way, I use this function also to use the same code for my NodeMCU boards and the Sonoff devices. I start to develop new functionality on a nodeMCU board and, if properly tested, upload it to a Sonoff device. You will find this trick in all my Sonoff files on Gihub.

This is, what I want to do now for an example which runs: The WiFiScan example. I first compile it for the ESP32, and then the exact same sketch also for the ESP8266. Unfortunately, we get another error. The constant “#define WIFI\_AUTH\_OPEN” seems not defined in the ESP8266 toolchain.

So, we have to define it for the ESP8266, and now, it compiles and we can compare both boards in parallel.

So, this episode has a found a happy end. Two brothers or sisters, running the same code, side by side. Or maybe one is the father and the other one is the daughter…

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

https://www.meetup.com/de-DE/IoT-Zurich/