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

Today, this channel will try to start into a new area: The area of the replacement of our beloved ESP8266 by the new ESP32. I invite you to join this journey. At the end of this video, you should be able to judge, if you want to order your first board (or take it out of your drawer).

In video #143 I got a lot of different ESP32 boards. So, we are able to check the first, but important feature of the ESP32: Its price point. Last year, when the ESP32s came to the market, a board costed more than 20$, which was at least 5 times more expensive than a comparable ESP8266 board. Today, the cheapest board I found was 6.80$, which is similar to what we had to pay when the ESP8266 hit the road. So, the price is no more an obstacle nor an excuse.

But what else do we have to check before we put a lot of energy and money into this new technology?

* We have to check, if the new features of the ESP32 are worthwhile to do the upgrade
* Then, we have to check out if we can use the ESP32 boards with our Arduino IDE
* Next, we have to check the different features of the ESP32, if they are already implemented in the Arduino IDE, how to use them, and if they work properly
* Features like digital in- and output
* Analog in-and output
* PWM output
* Libraries
* Deep-sleep
* Power consumption in general
* Bluetooth
* Encryption
* The role of the two processors
* And we will find many other things we have to understand if we want to use its full capabilities.

You might agree, that this cannot be done in one video. But we can do a first step.

If you are not familiar with the ESP32 and its advantages over the ESP8266, I encourage you to watch my video #103.

For the other questions: Let’s start today with the obvious: The installation of the new environment for the ESP32 into our Arduino IDE. You find a link in the description on how to do that. I started with the upgrade to the newest Arduino IDE, installed GIT on my PC, and entered the source location as described in the “how-to”. The target directory was not exactly as in the description, but you can easily get it: Just go to preferences 🡪 Sketchbook location, copy this directory, and add hardware/espressif/esp32. Then, the installation continues. At the end, you double click the get.exe file, and the compiler is downloaded from the internet. Now, you have already many ESP32 boards to select from.

Today, we will use the Wemos Lolin board and start with the blink sketch. And really, the sketch compiles and uploads. Quite fast, I have to admit. And the built-in LED starts to blink. What is the LED pin on this board? We start Serial and print the pin number: It is GPIO5. And really, if we connect a second LED to pin 5, it blinks. BTW, the Built-in LED is connected to VCC, and not to GND. This is, why, it blinks inverse…

In addition, we also checked Serial, and it works as expected. With this many available pins, does the ESP32 also have a Serial2?

No, it does not have one.

Can we still use our D0 to D8 GPIO addresses? No, we cannot. We have to use the GPIO numbers. What a relief! Much easier. But we have to change all our NodeMCU pin definitions. Usually, this should be easily done with the text editor.

But how many IO ports do we have on this Lolin board? There are all 22 GPIO pins of the WROOM-32 available. This is already something compared D0..D8 of the ESP8266

I suggest we use the available examples to discover the new features, and test, if they work. But, what kind of examples come with the ESP32?

Here, we have the list. Of course, other examples made for the Arduino or for the ESP8266 might also work. And the Wi-Fi examples are also ported to the ESP32. But let’s stick with the dedicated examples for the moment.

The first is LEDCSoftwareFade. It demonstrates the PWM capabilities of the ESP32. With the Arduino or the ESP32, we had the analogWrite() function. This function is not implemented in the ESP32 yet. But we have a similar function, the ledcWrite(channel, duty). To use this function, we have to initialize it first:

ledcSetup(LEDC\_CHANNEL, LEDC\_BASE\_FREQ, LEDC\_TIMER\_BIT);

ledcAttachPin(LED\_PIN, LEDC\_CHANNEL);

What do we do here? The first statement sets one of 16 available timers with a particular frequency and a resolution. For our purpose, we can set the resolution to at least 4 bits more than the resolution of our PWM signal. In this example, we use a maximum of 255 different values for our PWM signal, which is 8 bits. And, the LEDC\_TIMER\_BIT had to be at least 12. We can set it higher, if we want. But the higher we set the value, the lower the maximum PWM frequency. With a value of 10 I was able to reach 78 kHz, which is pretty fast! Each bit more halves this maximum frequency.

The second step attaches a pin to this timer. In our case, we use the built-in LED pin 5. And really the LED nicely fades on and off. If we look at the oscilloscope, we see the corresponding signal. It has a frequency of 5kHz and the duty cycle changes over time. Where did we see a similar signal in the past? Yes, you are right: When we used servos. The servo signal is much slower, and the duty cycle must not go over about 50%. So, let’s try. At 166 Hz, our mini servo moves exactly as expected. Great. And another cool application results, if we add a 1k resistor and a 10uF capacitor (low-pass filter): We created a primitive Digital to analog converter (also called DAC). And, because we count it up and down, it creates a saw tooth curve consisting of many distinct small points. But I hope, we will later be able to use the built-in DAC of the ESP32 which should be faster and more accurate.

Summarized, we do not have our well known analogWrite command, but we get more than that, if we go the extra mile: Much faster PWM at all available pins, if we need it.

The ESP32, as the ESP8266, is a connected device. That is, why we want to connect it to the internet. There are many different examples available. Let’s try the simpleWiFiServer first. We add the Wi-Fi credentials, and fire the ESP32 up. And really, we can connect to its IP address, and we can switch the built-in LED on- and off. Also, here, we have an inverse signal, because the LED is connected to VCC.

Let’s quickly walk through the code: The first difference we see is, that the library is no more called ESP8266WiFi.h, it is now called WiFi.h, which is also a better thing for future compatibility. But, again, we have to change also this in all our ESP8266 sketches if we want to use them with the ESP32. The rest of the sketch is the same as for the ESP8266. Good to know.

The next challenge is to check if I still can use my ESP8266 Libraries. I test the NTPtime library. And because it uses the ESP8266Wifi library, I have to create a new one, just with this change. After this small change, it works fine and prints the actual time to Serial.

Here, I discovered the first not so nice behavior of this new environment: The default setting of the upload speed for the Lolin board is 921000 baud, which is very fast. And at the beginning it worked fine with this speed. But suddenly, it started to create error messages during upload. Fortunately, it still worked with 115200 baud. Much slower, but as a work-around it is ok. With a different board, it still worked with 921000 baud. And later on, even with the same Lolit board. A typical instability…

Let’s continue with another new feature of the ESP32: The built-in touch sensors. 10 pins can be used as touch sensors. They are called T0..T9. The command to read these sensors is touchRead(T0). The value is influenced by the capacity of the wire which is connected to the particular GPIO pin. But where are these pins? Most of the ESP32 pins are multi-purpose. GPIO4, for example, can take 8 different functions:

* GPIO4, the normal digital I/O function
* ADC2\_CH0, is one of the 16 different analog pins
* TOUCH0, is one of the 10 touch sensors
* RTC\_GPIO10, one of the many GPIO pins which are connected to the second, ultra-low power CPU. They can be used to wake the ESP32 from deep sleep
* HSPIHD, which is part of the high-speed SPI bus
* HS2\_DATA1 is part of the SD card interface. Because very fast SD cards operate at 1.8 volt, the ESP32 even has a built-in programmable voltage regulator to supply these cards. Incredible!
* SD\_DATA1, also part of the SD card interface
* EMAC\_TX\_ER, part of the 10 and 100mbps Ethernet interface

This table gives us a glimpse in the HW possibilities of this chip. It is mindboggling! We just have to be aware, that we cannot use all of these functions together, we have to select one function for each pin. And in many cases, there are more than one pin involved in these decisions.

And the second important fact is, that we need the software to support all of these functions.

But for sure, we will see many different combinations of ESP32 features on one board for a special need, or a shield system, were specialized HW is combined with a base ESP board. For me, the biggest question mark is, if Espressif is willing to integrate all the possibilities into the Arduino IDE or if enough Enthusiasts will write libraries to use these functions. But for sure, this chip has lots of features to be discovered.

To conclude, I combine some touch sensors with the PWM signal generator from before. And with a little googling about the right frequencies, we can create a simple instrument!

For today, I think, it was enough and we can summarize:

* We were able to integrate the new architecture into our well-known Arduino IDE and everything works exactly as we know: The editor, the boards and port selection, the compilation, and the upload via USB.
* We used a Lolin32 board and the upload speed was default 921000 baud. Fast enough!
* We checked the GPIOs out, and discovered, that we have many more of them, and that they abandoned the Dx numbering scheme. So, we do not have to write our sketches different for the different boards.
* Then, we looked at the different examples provided with the infrastructure. This will be stuff for many videos to come. Today, we tired PWM and it worked. We were able to generate a fading LED, steer a servo, and even create a primitive DAC with this function.
* Next, we tried a simple example to check, if the Wi-Fi really works. And it does. We learned, that we will have to change all our ESP8266 sketches, if we want to port them to the ESP32.
* Then, I ported my NTP library to the ESP. With one small change, it worked and delivered accurate time also on the ESP32.
* At the end, we checked also the new touch pins and, together with the PWM functionality from the beginning of the video, were able to create a small instrument.

As I said: This is the beginning, not the end of a story. And if you are subscribed, you will automatically be part of this journey.

How do you judge the question in the title: Is the entry into the ESP32 not as difficult as You Thought? Please leave a comment about your plans.

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

<https://github.com/espressif/arduino-esp32>

<https://www.esp32.com/>

<http://hackaday.com/2016/09/15/esp32-hands-on-awesome-promise/>

https://en.wikipedia.org/wiki/ESP32