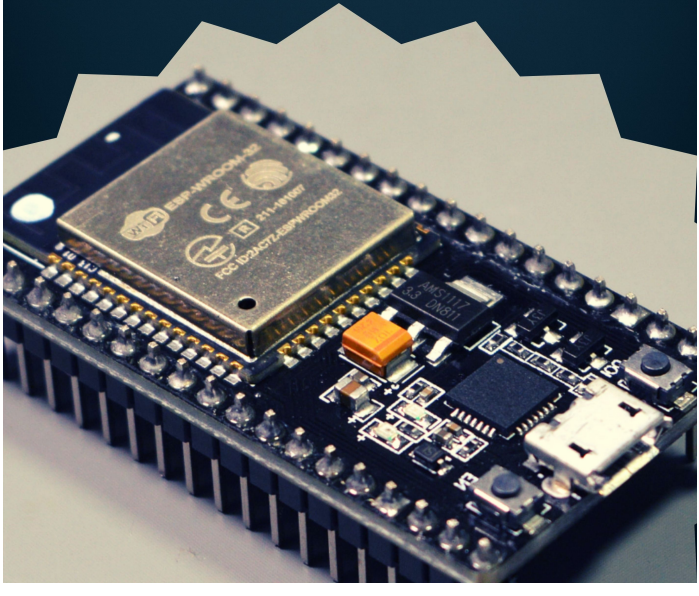
Tech Explorations

www.techexplorations.com

ESP32 INTRODUCTION

GET THE MOST OUT OF YOUR ESP32 WITH ARTICLES FROM THE TECH EXPLORATIONS BLOG



Peter Dalmaris, PhD

ESP32 An Introduction

Get the most out of your ESP32 with articles from the Tech Explorations Blog

Welcome to this special collection of articles, meticulously curated from the Tech Explorations blog and guides. As a token of appreciation for joining our email list, we offer these documents for you to download at no cost. Our aim is to provide you with valuable insights and knowledge in a convenient format. You can read these PDFs on your device, or print.

Please note that these PDFs are derived from our blog posts and articles with limited editing. We prioritize updating content and ensuring all links are functional, striving to enhance quality continually. However, the editing level does not match the comprehensive standards applied to our Tech Explorations books and courses.

We regularly update these documents to include the latest content from our website, ensuring you have access to fresh and relevant information.

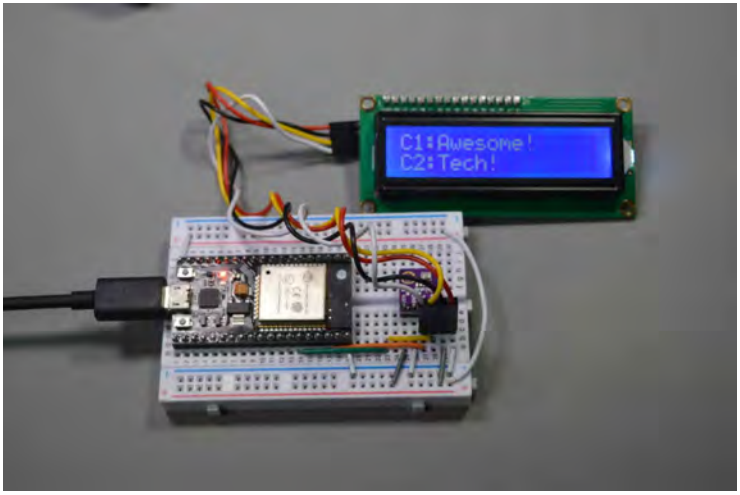
License statement for the PDF documents on this page

It feels and works like an Arduino, but... WOW

My New Favourite Microcontroller

Imagine an Arduino that fits snugly on a breadboard, has more memory, more pins, more speed, more communications (including WiFi, Bluetooth Classic, and BLE), but costs less than an Arduino Uno.

Imagine an Arduino that fits snugly on a breadboard, has more memory, more pins, more speed, more communications (including WiFi, Bluetooth Classic, and BLE), but costs less than an Arduino Uno. Because it is an Arduino, you can program like you would program an Arduino Uno. Cool?You can use the familiar and simple Arduino IDE. You old sketches will work, and so will your favourite libraries.Only, this is not an Arduino...



The ESP32 is my new favourite microcontroller. It features an excellent balance of performance and price.

It's the ESP32, from Espresif.

ESP32 is the successor to the ESP8266. The ESP8266 is a board that introduced the idea that a microcontroller with ample processing power and wireless connectivity at an insanely low cost is possible.

The ESP32 took all the awesome features of the ESP8266, improved on them, and added many more (like Bluetooth).

What I really like about the ESP32 though, is that its a mature product.

While the ESP8266 was, infamously, painful to work with for most people familiar with the Arduino, the ESP32 is a pleasure with which to work.

In addition, while the ESP8266 was plagued, in my experience, with reliability issues and felt experimental, the ESP32 is rock solid.

Espressif has done a remarkable job in implementing ESP32 support within the Arduino IDE.

And the hardware itself is super-reliable. I have one of my IoT prototypes working flawlessly for over two months now, without losing a beat.

What this means for you is simple: If you know how to use an Arduino Uno, you can use an ESP32.

Why would you want to use an ESP32? Because it can transform the scope of your projects while actually spending less money on hardware.

I have been using the ESP8266 for a few years now, but I never actually committed to it. I found it clunky, and I did not have the time to deal with its frequent problems.

The ESP32 changed this.

I started using it as my exclusive prototyping board in January 2019. I was blown away by how easy it was to transition to this board from the Arduino Uno.

Make no mistake: I still love the Arduino Uno and it is my go-to board for teaching electronics and programming.

But as a Maker, the ESP32 is my new favorite.

Since I focused on the ESP32, I have used it for several projects, including a heart rate monitor, and an Internet of Things gadget that can respond to my voice.

A couple of things impressed me during this time:

- How quick it was for me to start working with the ESP32.
- How quick prototyping was, once I got started.

- How I felt liberated from the Arduino Uno limitation; this liberation opened up my project horizons.

Let me show you what I mean with a quick example. Say that you want to connect your project to the Internet, and send out some sensor data. Here's how to do this in an ESP32, using the Arduino IDE.

Oh, and we'll use secure communications, because it is 2019.

You can see the full sketch here, but below I provide the highlights.

At the header, include the secure Wifi client library. This library ships with the ESP32-Arduino core for the Arduino IDE:

#include <WiFiClientSecure.h>

Provide the credentials to your Wifi network:

const char* ssid = "<your wifi network name>"; const char* password = "<your wifi network password>";

Provide the target site root certificate (I only show the first three lines here, and extracting this certificate from your target website is very easy):

const char* test_root_ca= "—–BEGIN CERTIFICATE—–n"
"MIIDdzCCAl+gAwIBAgIEAgAAuTANBgkqhkiG9w0BAQUFADBaM
QswCQYDVQQGEwJJn"
"RTESMBAGA1UEChMJQmFsdGltb3JlMRMwEQYDVQQLEwpDeWJ
lclRydXN0MSIwIAYDn"
"VQQDExlCYWx0aW1vcmUgQ3liZXJUcnVzdCBSb290MB4XDTA
wMDUxMjE4NDYwMFoXn"
"DTI1MDUxMjIzNTkwMFowWjELMAkGA1UEBhMCSUUxEjAQBgN
VBAoTCUJhbHRpbW9yn"
"ZTETMBEGA1UECxMKQ3liZXJUcnVzdDEiMCAGA1UEAxMZQmF
sdGltb3JlIEN5YmVyn"

“VHJ1c3QgUm9vdDCCASIwDQYJKoZIhvcNAQEBBQADggEPADC
CAQoCggEBAKMEuyKrn”
“mD1X6CZymrV51Cni4eiVgLGw41uOKymaZN+hXe2wCQVt2yg
uzmKiYv60iNoS6zjrn”
“IZ3AQSsBUnuId9Mcj8e6uYi1agnnc+gRQKfRzMpijS3ljwumUNK
oUMMo6vWrJYeKn”
“mpYcqWe4PwzV9/lSEy/CG9VwcPCPwBLKBsua4dnKM3p31vjsuf
FoREJIE9LAwqSun”
“XmD+tqYF/LTdB1kC1FkYmGP1pWPgkAx9XbIGevOF6uvUA65e
hD5f/xXtabz5OTZyn”
“dc93Uk3zyZAsuT3lySNTPx8kmCFcB5kpvcY67Oduhjprl3RjM71
oGDHweI12v/yen”
“jl0qhqdNkNwnGjkCAwEAAaNFMEMwHQYDVR0OBBYEFOWdWT
CCR1jMrPoIVDaGezq1n”
“BE3wMBIGA1UdEwEB/wQIMAYBAf8CAQMwDgYDVR0PAQH/BA
QDAgEGMA0GCSqGSIb3n”
“DQEBBQUAA4IBAQCFDF2O5G9RaEIFoN27TyclhAO992T9Ldcw
46QQF+vaKSm2eT92n”
“9hkTI7gQCvlYpNRhcL0EYWoSihfVCr3FvDB81ukMJY2GQE/szKN
+OMY3EU/t3Wgxn”
“jkzSswF07r51XgdIGn9w/xZchMB5hbgF/X++ZRGjD8ACtPhSNz
kE1akxehi/oCr0n”
“Epn3o0WC4zxe9Z2etciefC7IpJ5OCBRLbf1wbWsaY71k5h+3zv
Dyny67G7fyUIhzn”
“ksLi4xaNmjICq44Y3ekQEe5+NauQrz4wlHrQMz2nZQ/1/I6eYs9
HRCwBXbsdtTLSn”
“R9I4LtD+gdwyah617jzV/OeBHRnDJELqYzmpn” “—–END
CERTIFICATE—–n”;

Create a secure client object:

WiFiClientSecure client;

In setup() , connect to your Wifi network:

WiFi.begin(ssid, password);

And in loop(), first set the root certificate for the connection to the remote server:

client.setCACert(test_root_ca);

… and connect:

client.connect(server, 443)

You are now connected, and you can transmit data securely using client.println(), or receive using client.read().

Because the ESP32 dedicates a full core to running the radio components and has cryptographic hardware acceleration, the Wifi and secure web functions happen seamlessly for your sketch. Your sketch is running in its own full-speed core, at full speed, not losing a beat.

That is one less thing for me to worry about.

At this point, you might be thinking “That’s cool, I want to jump right in and use the ESP32 in my project NOW!”.

Well, that was my approach

I did learn that there are several similarities, but also differences between the ESP32 and my familiar Arduino Uno. I discovered that knowing what those are upfront would have made my work with the ESP32 easier, and my prototyping faster.

I am going to discuss those differences and similarities in the next article.

In the next article I also want to show you another cool thing that you can do with the ESP32, that is a stretch for the Arduino Uno.

In the meantime, do you have any questions about the ESP32 that you’d like to ask me? Just post it below.

INTRODUCTION TO THE ESP32 GUIDE SERIES

From the Arduino Uno to the ESP32: Maker transformation

In this article, I'll go over the differences and similarities between the Arduino and the ESP32, as well as show you an example of something cool you can do with the ESP32 and how simple it is to do it almost straight away.

In the previous article, I wrote about the ESP32 and how impressed I am with what I have been able to do with it since I started using it in my projects a few months ago.

I wrote about how the ESP32, the successor of the ground-breaking ESP8266, did everything right for Arduino Makers: processing power, communications options, form factor, compatibility with the Arduino ecosystem and the Arduino IDE, and, of course, the price.

In the previous article, I promised to discuss the differences and similarities between the Arduino and the ESP32. I will give a summary of this in this article (see below).

I also promised that I'd give you another example of something cool you can do with the ESP32, and how easy it is

to do that almost out of the box. And, yes, I'll do that in this article too (see below).

Before I get to the technical stuff, I want to say this: soon after I started working with the ESP32, I started feeling liberated from the shackles of the Arduino Uno. While the Uno is the perfect board for the beginner, its limitations are apparent as soon as you decide to build an IoT application or something that combines displays, sensors, communications, and a basic user interface.

Yes, there are many other boards that I could use instead of the ESP32. But none of those I played with seemed right. Their price was too high, or something important was missing from their hardware, or they were too different from the familiar Arduino sketching paradigm, or I had to use its manufacturer's infrastructure to make it work.

This ESP32 opened up my project horizons.

I believe that after your first steps with the ESP32, you will also feel liberated. Your project horizons will also open.

Now, I'd like to make a quick comparison between the Arduino and the ESP32. Since more Arduino Makers are familiar with the Arduino Uno, I will focus on Arduino Uno VS ESP32, but this comparison holds true for most Arduino boards.

These two don't have much in common.

Here's the thing: These two are totally different.

Not only they look different, but their architecture is also totally different, and they have a different hardware architecture.

Their built-in capabilities are very different.

Memory and storage, the processing capacity, the number of GPIOs that they expose, the communications features and much more; these are very different microcontrollers.

In terms of features, the closest Arduino boards that are comparable to the ESP32 are probably the Arduino MKR1010, the Arduino 101 or the Arduino Zero. At least, these Arduinos have integrated Wi-Fi and Bluetooth, and comparable computational capacity.

Because the Arduino Uno and the ESP32 are so different in terms of hardware, it doesn't make sense to compare them that way. It makes more sense to write about the use cases of each one.

Here's what I think:

Is the ESP32 right for you?

Well, it's definitely not for beginners. If you are not familiar with the Arduino, the ESP32 will be a complex board to learn. I don't recommend it.

The ESP32 provides a perfect opportunity for Arduino Makers to extend and expand on their skills. The hardware that the ESP32 contains means that you can work on more exciting projects and that alone is very desirable.

I find that the ESP32 is better value than comparable Arduino boards, like the MKR1000/1020 and the Arduino Zero.

Who is the Arduino Uno for?

I think that the Arduino is a much better choice for new makers.

It's simple.

It's more forgiving, as well, to problems in wiring and mistakes in wiring.

It's easier to set up.

In summary, if you are a new Maker, learn using the Arduino Uno. Then move on to the ESP32.

Now, I want to show you something really cool. As you probably know, the Arduino Uno doesn't have the ability to generate a true analog signal. It can create the effect of an analog signal via Pulse Width Modulation.

The ESP32 can generate PWM output, of course. Almost all of its pins are PWM-capable. And it can do that with far more programmable control of the PWM signal parameters than what is possible to do with the Arduino.

But the ESP32 can also generate true analog output because it

contains two 8-bit DACs: Digital to Analog Converter. You can access those two DACs (channels) via two of the GPIOs (25 and 26).

Let's see how easy it is to create a true analog signal.

Example 1: Let's create a flat waveform on GPIO 25 (this is DAC channel 1). The DAC resolution is 8 bits, so let's "write" decimal 127 to GPIO25.

It's this easy:

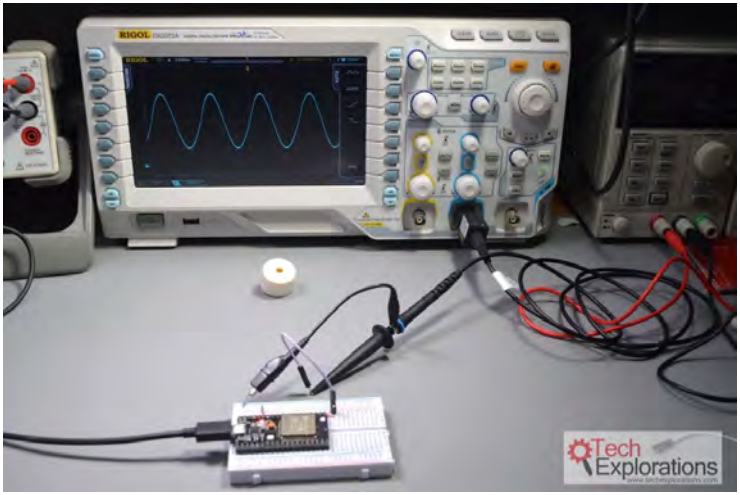
```
dacWrite(25,127);
```

Value 127 is in the middle of the range of possible values 0..255, so the result of "dacWrite(25,127);" is around 1.65V measured on GPIO25 (assuming that the supply voltage is 3.3V).

We can take this one step further and create an actual sine wave:

```
for (int deg = 0; deg < 360; deg++) dacWrite(25, int(128 + 80 * (sin(deg*PI/180)))); // Sine wave
```

This example will generate a sine wave on GPIO25. If you connect an oscilloscope on that pin, you will see this visualization of the signal:



A sine wave, calculated during run-time on the ESP32.

The ESP32 is fast enough to be able to calculate the waveform in runtime, so you don't have to create lookup tables in memory. And if you enjoy math, you can develop mathematical expressions to generate arbitrary waveforms.

In the next article in this series, I will write about my experience of using the ESP32 to make this gadget:

Something I made to help me learn the ESP32.

My motivation for working on this gadget was to learn the ESP32 in a real-world setting. I wanted something to help me explore as many elements of hardware as possible.

This project helped me learn how to build a system around the ESP32 that integrates environment sensing, an Internet clock, appliance voice control via IFTTT and Google Assist, a touch TFT display for the user interface, and the use of many of the ESP32 features such as the SPI File System, Wifi, timer, and more, with an efficient program design.

Checkout the next article in this series with more information about my pet project

Peter

PS. In the meantime, do you have any questions about the ESP32 that you'd like to ask me? Just post them below.

My first ESP32 project experience: What was it like?

In this article, I am writing about my experience in using the ESP32 in one of my pet projects.

I'm very excited

In the last two articles (first and second), I wrote about the ESP32 and how it opened up new learning and project possibilities for me.

In the first article, I wrote about why I am impressed by the ESP32 hardware and its integration with the Arduino ecosystem. I also showed you an example of how easy it is to do secure web communications with the ESP32, something impossible to do with the Arduino Uno without expensive external hardware.

And in the second article, I compared the Arduino with the ESP32 and gave you another example of ESP32 awesomeness: Digital to Analog Conversion, and how easy it is to generate an arbitrary true analog signal.

My work with the ESP32 begun in January 2019, after playing around with the ESP8266 for a few years before this. I never committed to the ESP8266 because I felt it wasn't ready. Software integration with the Arduino ecosystem was clunky, and the hardware was unreliable.

All this has changed with the ESP32. It works very well with the Arduino ecosystem, and in my long-running experiments, it has been rock-solid.

In this article, I am writing about my experience in using the ESP32 in one of my pet projects.

It's this one:

Something I made to help me learn the ESP32.

I started work on this gadget because I wanted to learn the ESP32. I believe in project-based learning, so there you go

This project helped me learn how to build a system around the ESP32 that integrates environment sensing, an Internet clock, appliance voice control via IFTTT and Google Assistant, a touch TFT display for the user interface, and the use of many of the ESP32 features such as the SPI File System, Wifi, timers, and more, with an efficient program design.

Right off the bat, I can say that this is something not possible to do on the Arduino Uno. Just the storage footprint of the sketch is more than what the Arduino can hold. I would need to expand the Arduino's flash memory with an external flash module.

Speaking of storage, in my ESP32 project, I used the integrated SPI File System it provides an efficient way to store files (flat text files, images, HTML files, or anything else you like) in the ESP32 flash memory. The basic file operations are

possible via simple functions. In the Arduino Uno, I would need to use an external SD card module, adding to the complexity of the hardware.

Supporting the large TFT screen is possible on the Arduino Uno via the SPI interface; however, the refresh rates would be low. Adding the touch capability would make the user interface slow and unresponsive.

You can read more details about this project on my blog (here, here, and here).

I want to focus on the experience.

Rapid

Prototyping

Fun

As a Maker, subconsciously, I operate in a fine balance between reaching a goal (getting my gadget to a state where it works reliably) and dealing with the constraints of my hardware, software, and my knowledge and skills.

With the Arduino, those constraints are particularly tight.

You know: memory, processing, communications, pins, etc.

I've been working with the Arduino for many years, and I know that when I run out of hardware resources, there are things I can do to make it go further. I can optimize memory use. I can multiplex pins and communications. I can remove libraries and replace them with optimized, custom code.

But as a hobbyist, this kind of work conflicts with my psychological need to achieve my goal. Unless these optimizations are the goal, they impact the enjoyment that comes out of making something new and achieving the goal.

Does this “sound” familiar?

Velocity in making, learning,and progress is important.

With the ESP32, I didn’t have to make such compromises.

Except for trying to use HTTPS REST communications (I learned first hand that MQTT is a much better protocol for IoT applications), I was able to reach every single milestone that I set in this project.

And it was fun.

The ESP32 is the perfect higher-end microcontroller for the Arduino Maker.

But, in my experience, it isn’t a zero-effort proposition. You will still need to do some learning.

I have good news and good news

First “good news”: As an Arduino Maker, you already know the bulk of what you need to start using the ESP32 in your projects.

Second “good news”: For the rest, I can help you. I have completed work on a new course, that I named “ESP32 for Busy People”. In this course, I show you how to use the ESP32 in your projects. I assume that you are already familiar with the Arduino, and help you reach a skill level where the ESP32 is the primary microcontroller for your projects.

If you are familiar with my course, Arduino Step by Step Getting Started, and Arduino Step by Step Getting Serious, then you know what “ESP32 For Busy People” is like; it is a comprehensive course and recipe resource for Makers.

If you want to learn more about ESP32 For Busy People, click here.

Peter

PS. Do you have any questions about the ESP32 or the course that you'd like to ask me? If you post your question in the next few hours, I will try to respond within 24h.

Just post your question below.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

It's packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

Learn more

The ESP32 module

In this lesson, you will learn about the ESP32 module that powers the development kit that I am using in this series of lessons, and the video course.

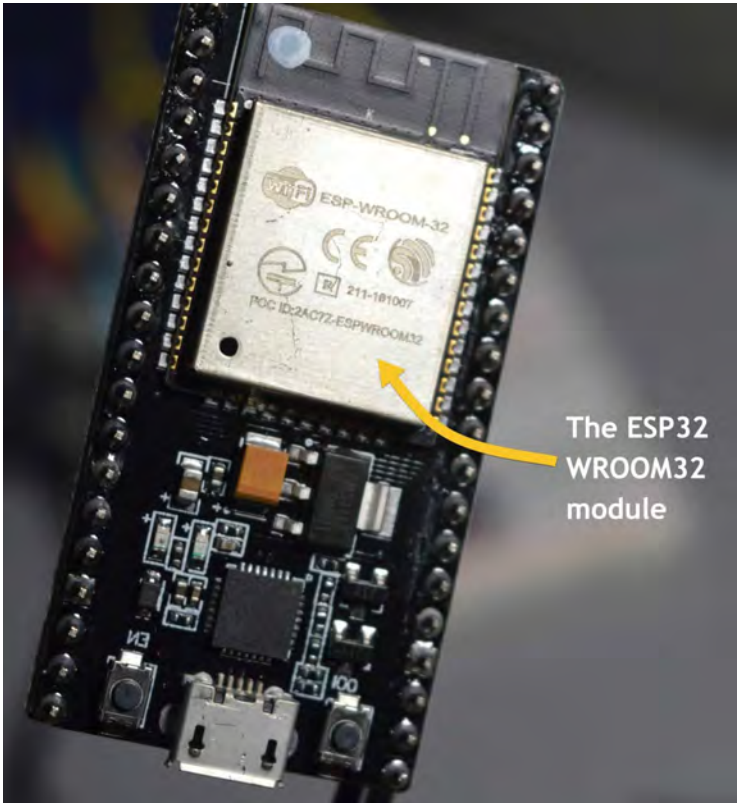
In this lesson, you will learn about the ESP32 module that powers the development kit that I am using in this series of lessons, and the video course.

You can watch the video, or if you are the “reading” type, you can read the text below.

The ESP32 WROOM32 module

In the photo below, the ESP32 module is indicated by the arrow, and it dominates the surface of the development kit. There are several variants of the ESP32 module (more about this below). The once used in the ESP32 DevKit v4 that I will be using in these lessons and course is the WROOM32.

In general, keep in mind that “ESP32” is a generic reference name. We use it to refer to all the ESP32 module models and even the dev kits. It’s a bit like saying “Arduino,” meaning the Arduino family of boards, not a specific board.



The ESP32 DevKit v4 contains the ESP32 WROOM32 module.

Drilling further into the module, we learn that the WROOM32 module contains the ESP32 D0WDQ6 microcontroller chip.

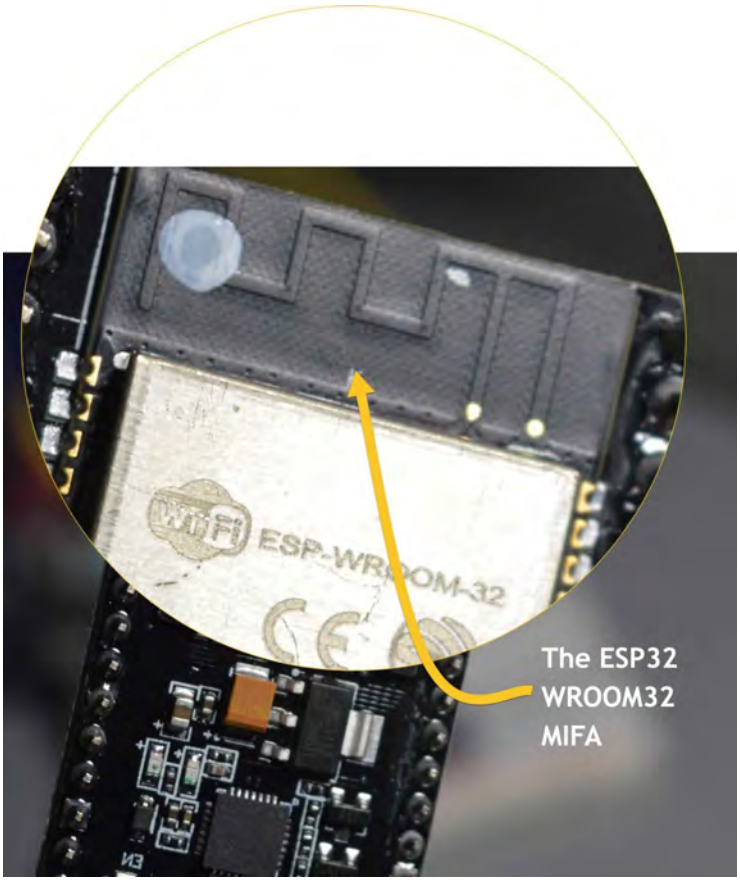
There's also another variant, the ESP32 WROOM32D, which contains a slightly different chip that powers it, and there's also the WROVER modules, such as the ESP32 WROVER-IB which also contains at ESP32 D0WD chip but with additional memory, and many many more.

Some of the ESP32 module models

Each of those modules and chip combinations has its own characteristics, and it's optimized for a particular purpose. Typically, the variation has to do with how much memory is available, whether or not there is pseudo-static RAM, the kind of antenna that they use, or whether they use a single or a dual-core processor, and so on.

The module in the development kit that I'll be using in this course has got a chip with a "**D**" in its name, which denotes "**dual core**."

My dev kit contains the ESP32 WROOM32 module, which, in turn, contains the ESP32 D0WDQ6 chip. This chip provides four megabytes of flash memory that we can use to store our programs and data. It contains no pseudo-static RAM, and it has a "MIFA" antenna. "MIFA" stands for "meandered inverted F" antenna.



The ESP32 WROOM32 MIFA antenna.

You can see the antenna at the top edge of the board with its zigzag pattern (see photo above). This antenna is a good fit for the ESP32 because of the small amount of available board space allocated to the antenna. We want this antenna to be etched on the PCB itself, instead of having to connect an external component for the antenna. And because there's not much space on the board, a MIFA antenna uses a meander shape, so that we end up with a full electrical length antenna that fits at this small available space.

The ESP32 WROOM32 D and U

The ESP32 WROOM32-D and ESP32 WROOM32-EU variants also contain the D0WDQ6.

These still have four-megabyte flash memory, and no pseudo-static RAM. The D model still has an integrated MIFA antenna.

But the “U” model has a connector that allows us to connect an external antenna.

In addition, both of those variants are smaller than the “regular” ESP32 WROOM32.

The WROOM-32D and WROOM-32U modules

The ESP32 WROVER

The WROVER variant is also more powerful compared to the WROOM models.

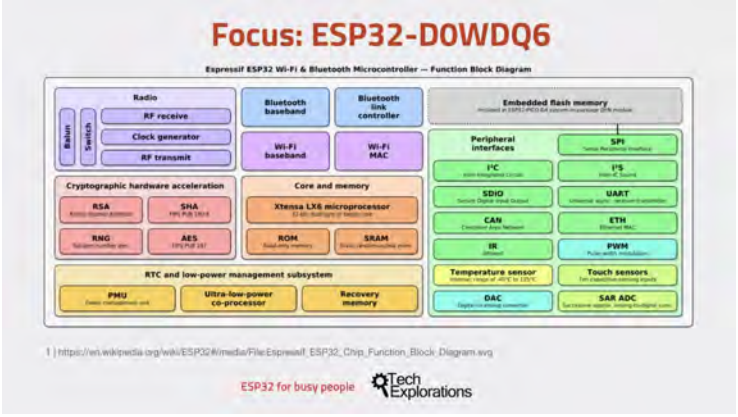
In summary, they have the same amount of flash as the WROOM modules. They contain SPI pseudo-static RAM (WROOM models have none).

They are available with an integrated MFA antenna or an external USFL antenna, and they have variants that can operate with as low as one point eight volts and up to one hundred and forty-four megahertz of clock speed.

The ESP32 WROVER modules

The ESP32 block diagram

Below, you can see the ESP32 D0WDQ6 module block diagram. It provides a map of all the hardware that is embedded in this microcontroller. In a sense, this diagram provides a summary of the module's capabilities.



The ESP32 D0WDQ6 module block diagram

At the center, you can see the two XTENSA LX6 microprocessors, with the ROM and static RAM.

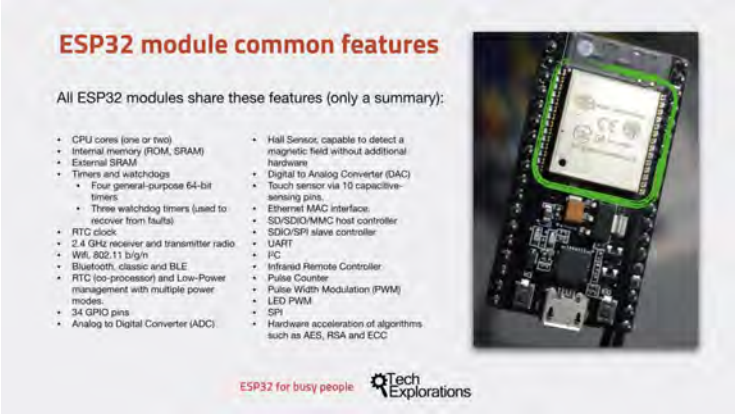
At the top left of this diagram, you can see the radio hardware, including the Wi-Fi and Bluetooth. On the right side is the embedded flash memory, where we store our programs and other data and files; the peripheral interfaces, I2C, and SPI, among many others.

There's also cryptographic hardware acceleration so that we can use TLS and SSL encryption when we are communicating with the Internet.

And, there is also a low power management system subsystem down the bottom left of this diagram.

Conclusion

To wrap this lesson, see below a summary of the hardware features of the ESP32 D0WDQ6.



A summary of the hardware features of the ESP32 D0WDQ6

Go on to the next lecture and have a look at the ESP32 development kit.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

It’s packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

Learn more

The ESP32 Development Kit

In this lesson, you will learn about the ESP32 “DevKit” or “developing kit.” At the time I am writing this, the latest development kit from Espressif is version four.

In this lesson, you will learn about the ESP32 “DevKit” or “developing kit.” At the time I am writing this, the latest development kit from Espressif is version four.

You can watch the video, or if you are the “reading” type, you can read the text below.

In the image here, you can see the ESP32 and the development kit on which the ESP32 module is situated.

The ESP32 DevKit version 4

The ESP32 dev kit exposes the ESP module's pins (at least some of those) to the outside world. Thanks to the dev kit, we can use the ESP32 by plugging it to a breadboard, or even attaching wires directly to its pins, without having to worry about implementing our own power, USB communications, reset circuit, etc.

Development kit core features and hardware

The core hardware that is present on the developing kit are things such as the USB-to-serial programming interface, the power subsystem that allows us to provide power within module push buttons for reset, and for setting it to the upload mode, indicator LCDs and a few other things.

So in this image below, I've marked with numbers the most important features of the ESP32 dev kit.

The ESP32 DevKit v4 core features

You can see:

1. The USB connector, which is also the way that we provide power to the ESB module and upload programs and also communicate with a serial monitor. It is a micro-USB port.
2. The reset pushbutton. We use this button to reset the module and restart the execution of a sketch that is already uploaded.
3. The "Enable" pushbutton. We use this button to place the module to upload mode. Variations of the development kit can place themselves into upload mode automatically, just like an Arduino can when connected to the Arduino IDE. The board I use in this guide does not, so I press button 3 immediately after I click on the "Upload" button in the Arduino IDE.
4. The power-on LED. As long as your ESP32 is

powered, that LCD will be on.

5. The two rows of headers marked five here in this photo. These pins break out the pins from the ESP32 module so that they become compatible with the breadboard. This is another really nice aspect of the ESP32 development kit as opposed to the Arduino: The fact that the development kit is compatible with the breadboard.
6. A programmable LED, which, in the case of my development board, is attached to GPIO 2. Be aware that not all development kits have this programmable LED. If yours doesn't have it, simply connect an external LED to GPIO 2, and you'll have the exact same functionality.

USB communication and power regulator

The square chip right above the USB connector is responsible for the UCB communications.

Above and to the right of the USB chip is another chip with four pins. This is the voltage regulator, which allows us to connect a voltage source from 5 to 12 V to the dev kit. You can learn more about the power options that you have with the ESP32 module in another lesson in this introductory series.

Other development kits

Before you move to the next lesson, I also wanted to mention that there is a variety of other development boards that host the ESP32 module.

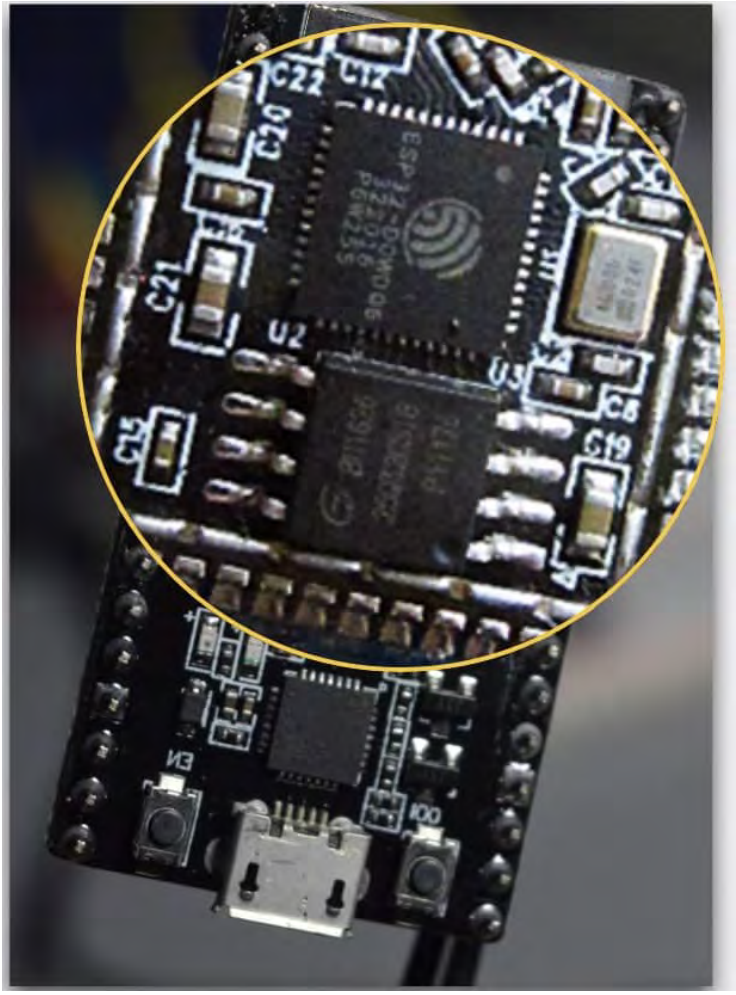
For example, there is the ESP32 Lyra and the ESP32 Pico. Go ahead to learn more about them.

There is a variety of ESP32 Dev Kits.

A “naked” ESP32 module

If you remove the cover, it reveals the bare module (don’t remove the cover unless you are prepared to break your ESP32 module).

Zoom in, and you can see the two integrated circuits in it.



The ESP32 WROOM32 without its cover

Photo of ESP32-D0WDQ6 by Brian Krent – Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=57745131

The larger one (this one here) is the ESP32 D0WDQ6 microcontroller, and the slightly smaller chip right here is the

SPI flash memory.

So now you know what is beneath the cover.

Continue to the next lecture where you will learn about the differences and similarities between the ESP32 and the Arduino.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

It’s packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

Learn more

The ESP32 compared to the Arduino

In this lesson, you will learn about the differences and similarities between the ESP32 and the Arduino and, in particular, the Arduino Uno.

In this lesson, you will learn about the differences and similarities between the ESP32 and the Arduino and, in particular, the Arduino Uno.

You can watch the video, or if you are the “reading” type, you can read the text below.

How similar (or different) are these two?

These two are totally different.

Not only they look different, but their architecture is also totally different. They have different hardware architecture. Their built-in capabilities are very different.

The ESP32 and the Arduino are very different creatures.

The amount of memory that they include, the processing capacity, the number of GPIOs that they expose, the communications features, and much more, are really different between these two microcontrollers.

The closest Arduino boards that are comparable to the ESP32 are probably the Arduino 101 or the Arduino Zero.

At least those share some of the features that the ESP32 has, such as integrated Wi-Fi and Bluetooth, and the computational capacity. But even there, the difference differences are more than the similarities.

Why is the ESP32 a good option for Arduino makers?

What makes the ESP32 an excellent choice for people that are familiar with the Arduino is the software.

Espressif, which is the company that designs and makes the ESP32, has made a huge effort in writing software that bridges the hardware gap between the ESP32 and the Arduino.

The ESP32-Arduino Core software bridges the hardware gap.

Thanks to the software, that we call as you'll see later the "ESP32-Arduino Core," we can use the ESP32 as if we are using the Arduino.

Thanks to the software, the ESP32 can be treated as being compatible with the Arduino:

- We can use the Arduino IDE as the development environment.
- We can use a programming language that matches almost one-on-one with the language that we have learned for the

Arduino.

- And, to a large extent, we can reuse almost 90 percent of the Arduino libraries in software that we write for the ESP32, which is pretty amazing.

The Arduino IDE

The ESP32 works with the Arduino IDE with the installation of the ESP32-Arduino Core and the integration between these two is remarkable.

Once you install the ESP32-Arduino Core, you get access to a large variety of development kits that are based on the ESP32, and you also get a lot of example sketches.

To program the ESP32 you can use the Arduino IDE.

You can start using it right away.

Even when it comes to the libraries, most of the Arduino libraries will just work with the ESP32, again because of the ESP32 Arduino core software that Espresif has developed.

Of course, the ESP32 contains unique features that are not present in the Arduino. To take advantage of those features, such as the SPI file system (SPIFFS), Espressif has to provide compatible libraries that we can use via the Arduino IDE. I show how to use the SPIFFS in the course.

Who is the ESP32 for?

So who is the ESP32 for?

Well, it's definitely not for beginners.

I've said this before: if you're not familiar with the Arduino, then the ESP32 will be complex. It will be a difficult thing to learn, and I don't recommend it.

I think that the Arduino is a much better choice for new makers.

It's a simpler device.

It's simpler to program.

It's more forgiving, as well, to problems and mistakes in the wiring.

It's more robust, so it's easier to set up.

The ESP32 is perfect for Makers familiar with the Arduino.

When you work with an Arduino Uno, you can just download the Arduino IDE, plug in your Arduino, and off you go. You don't have to make any modifications to it.

Once you've built up your knowledge and skill on the Arduino, in particular on the Arduino Uno, then the ESP32 provides a perfect opportunity to extend and expand on those skills.

The additional features that the ESP32 contains means that you can work on more interesting projects and that alone is very desirable.

You can start working with the ESP32 using your existing Arduino skills.

There is no or very small learning curve.

Then anything else that you want to do on top of what you already know means that you can improve your skills incrementally and gradually, gently without much stress.

You also get Wi-Fi and Bluetooth and lots of memory, essentially for free.

By “free,” I also mean the cost of the board.

The ESP32 dev kit is actually cheaper than Arduino Uno, which means that you get a more powerful board for a lower price.

At the level where you use your existing Arduino skills to work with the ESP32, you can treat the ESP32 as a supercharged Arduino Uno: faster, better in many respects.

And when you feel confident and ready, you can actually move away from the Arduino IDE to a completely integrated development environment.

Ready to learn about the ESP32 GPIO’s? Go on to the next lesson.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

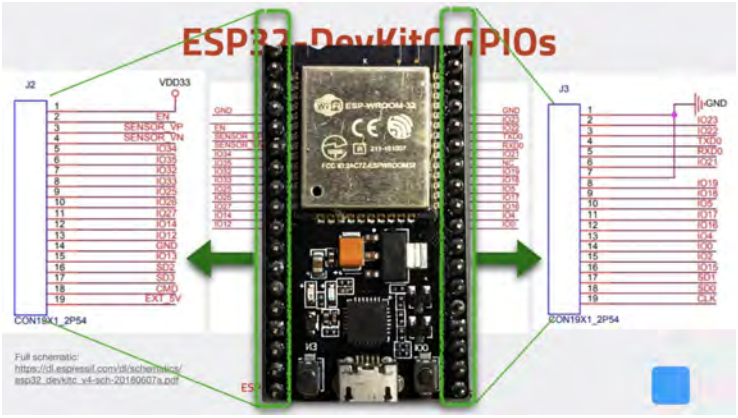
It’s packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

Learn more

The ESP32 GPIOs

In this lesson, you will learn the basics of the ESP32 GPIOs. You will learn how to recognize the various names we use to refer to them, and some of the functions that they expose.



In this lesson, you will learn the basics of the ESP32 GPIOs. You will learn how to recognize the various names we use to refer to them, and some of the functions that they expose.You can watch the video, or if you are the “reading” type, you can read the text below.

Which ESP32 pins are exposed via the headers?

Most of the 38 pins of the ESP32 module are broken out in two rows of pins in the ESP32 dev kit, but not all of them.

The ESP32 module and Dev Kit pins. Not all ESP32 pins are broken out to the headers.

In the figure shown above, you can see the ESP32 WROOM-32 module positioned at the top of the dev it. Notice that the module has three sides where pins are exposed. The fourth side is where the antenna is placed.Most of the module pins are broken out to the two headers of the dev kit, the left (J2) and the right (J3) headers. In the same figure, you can see the names of the broken out pins. For example, there are the GPIOs (“General Purpose Input Output” pins), such as “IO43” and “IO17.” Other names are used, too, such as “SENSOR_VP” and “RXD0.”

How do we refer to ESP32 pins?

Almost all pins on the ESP32 are multipurpose, and hence we can use several names that refer to the same physical pin. That depends on what we want to do with a particular pin. In this series and video course, I try to use the GPIO notation exclusively, so I will be using, for example, “GPIO21” to refer to physical pin 42 when I want to use it as the data pin of the I2C interface.

You can find all the information you need about the ESP32 pins

and their roles in the WROOM-32 datasheet. In the figure below, I have extracted part of the datasheet that contains the pins I use most frequently.

An extract of the ESP32 datasheet, showing the roles and names of several of its pins.

Notice how each of these pins exposes more than one functions? Apart from the power pins, all of them are multifunction.

Look at an example, such as GPIO34. This corresponds to physical PIN 6. It's an input by default, one of the analog to digital converter channels and it also provides access to the RTC GPIO4.

Try another example, like GPIO2, that is physical pin 24.This is an input/output pin and has got multiple functionalities: apart from just being GPIO2, it's also an analog to digital converter (ADC) pin channel 2, it exposes a touch sensor, and it's also part of the SPI hardware.

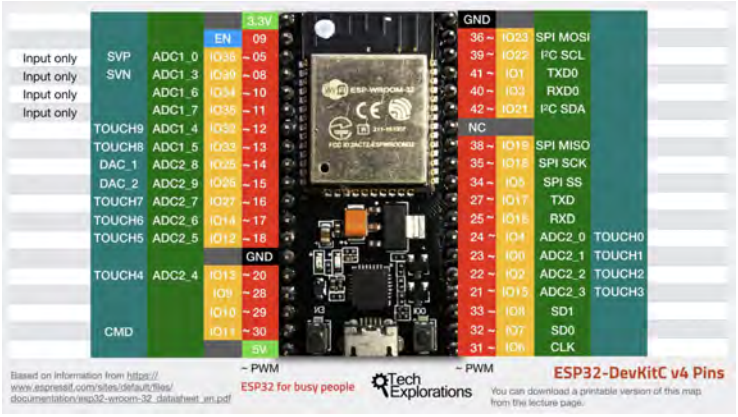
I keep a printout of this Figure handy so I can quickly check on pin roles.

You can download your copy of the high-res version of this

figure.

The ESP32 Dev Kit pin map

I have also developed the pin map that you can see below. Feel free to download your copy, print it out and keep it handy when you're working with the ESP32 throughout this course.

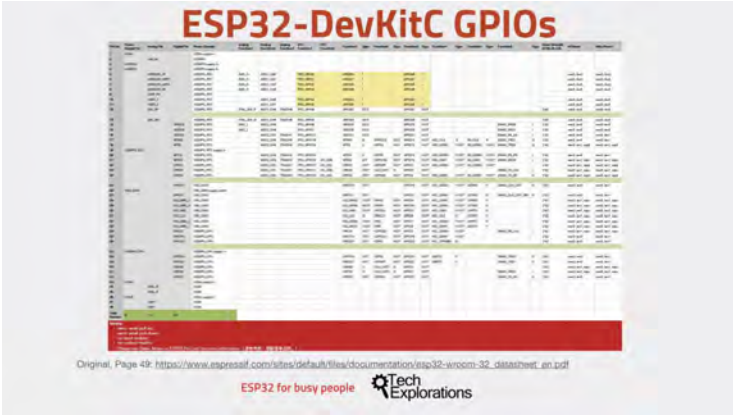


The Tech Explorations ESP32 Dev Kit v4 pin map, with my compliments

In this guide and in the video course, I make frequent reference to this map as I find it easier and faster compared to looking up this information in the datasheet. The map contains the pin functions that I use most often.A quick example. Find GPIO32 in the map. The map shows that this is physical pin 12 in the ESP module, and it give access to one of the touch sensors, and to the analog to digital converter (ADC).Also, I prefer to refer to pins using their GPIO numbers. In the map, these numbers are contained in the yellow columns.

Datasheet summary table of GPIOs

Another document that I use often is this one here:



The GPIO role and naming summary from the ESP32 WROOM32 datasheet.

It comes out of the datasheet and it's a summary of the various functionalities that are exposed at each pin.

Ready for more? Go on to the next lesson and have a look at the ESP32 communications capabilities.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

It's packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

Learn more

ESP32 Communications options

In this lesson, you will learn about the various communications capabilities of the ESP32, and specifically about communications between sensors and integrated circuits, or other devices such as mobile phones and the Internet.

In this lesson, you will learn about the various communications capabilities of the ESP32, and specifically about communications between sensors and integrated circuits, or other devices such as mobile phones and the Internet.You can watch the video, or if you are the “reading” type, you can read the text below.

How does the ESP32 communicate with its environment?

Most of the 38 pins of the ESP32 module are broken out in two rows of pins in the ESP32 dev kit, but not all of them.

A summary of the communications options of the ESP32.

On the left, you've got the main wireless capabilities, namely Wi-Fi and Bluetooth.

And on the right, you've got the wired capabilities, which allow the ESP32 to be connected to either other ESP32 and microcontrollers or smaller devices like sensors.

There are three **SPI** channels.

There are two **I2C** channels ("Inter-integrated circuit"), and two **I2S**, which is a lesser-known communications technology. I2S stands for "Inter IC for Sound," and it's an electrical serial bus, just like the I2C, but typically used for connecting digital audio devices.

We also have three serial interfaces, an **Ethernet MAC** interface, a **CAN bus** ("Controller Area Network bus"), which is

typically used in vehicle applications, automotive applications, and allow microcontrollers and devices to communicate with each other in cars.

And finally, there are hardware for infrared (“IR”) serial communications.

Functional block diagram

You can see how these capabilities are laid out in the functional block diagram below.

The ESP32 functional block diagram.

Let’s have a closer look at each one of these capabilities.

Wi-Fi is integrated into the module, and you find everything that you need to use to connect to a **Wi-Fi** network or to create a Wi-Fi hotspot.

There is the antenna circuit with an amplifier, various filters, and power management.

As far as **Wi-Fi** protocols are concerned, there are 802.11 b, g, and n. The 802.11n networking supposed up to one 150

Mbits/s bandwidth, with support for **Wi-Fi multimedia**.

You can find more details in the datasheet in Section 3.5.

For Bluetooth, the ESP32 is compliant with classic **Bluetooth** 4.2, and **BLE** (Bluetooth Low Energy) specifications.

It contains a Class 1 2 and 3 transmitters, and it can simultaneously advertise and scan.

Again there are more details in the datasheet.

There are three **SPI** channels, up to 80 MHz in frequency. All of them can go up to 80 MHz.

The ESP32 contains two full **I2C** bus interfaces, and they can be configured to operate as a master or a slave, standard, or fast mode.

We'll be using **I2C** in the video course to connect the ESP32 to things such as sensors, in particular, the BME280 and the LCD screen with the I2C backpack.

There are also two **I2S** interfaces, which are typically used in audio applications.

And of course, the ubiquitous UART (Universal asynchronous receiver transmitter). Three of those interfaces are present. We can use them with any serial device.

In addition to the above, there's also the infrared receive and transmit communication capability, and an **Ethernet MAC** adapter so we can connect to a local ethernet network.

As you can see, the ESP32 is quite a lot of capabilities to communicate with devices, near or far away via the Internet.

And we'll be demonstrating most of these capabilities in this course.

Now, let’s turn our attention to power. You will need to provide power to your ESP32 somehow, and in the next lesson, you will learn of three ways to do just that.

ESP32 dev kit power options

In this lesson, you will learn how to power your ESP32 dev kit.

In this lesson, you will learn how to power your ESP32 dev kit.You can watch the video, or if you are the “reading” type, you can read the text below.

Option 1: USB

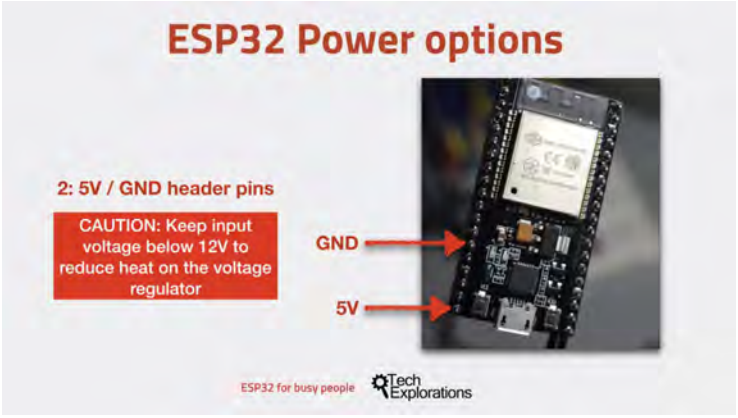
The easiest way to power your ESP32 dev kit is to use the USB port. The dev kit includes a micro USB port through which you can both supply power to the board, and implement serial communication with the host computer for uploading a sketch.

The easiest way to power your ESP32 dev kit is via the USB port.

Just plug one end of the cable into your computer's USB port or to a USB compatible power, the other end to the USB port of the ESP32 dev kit, and you're good to go.

Option 2: Unregulated power to GND and 5V pins

The second option is to connect an external unregulated power supply to the 5V pin and ground pins. Anything between around 5 and 12 Volts should work.But it is best to keep the input voltage to around 6 or 7 Volts to avoid losing too much power as heat on the voltage regulator.



You can connect external power via the 5V and GND pins. Beware of the voltage limits.

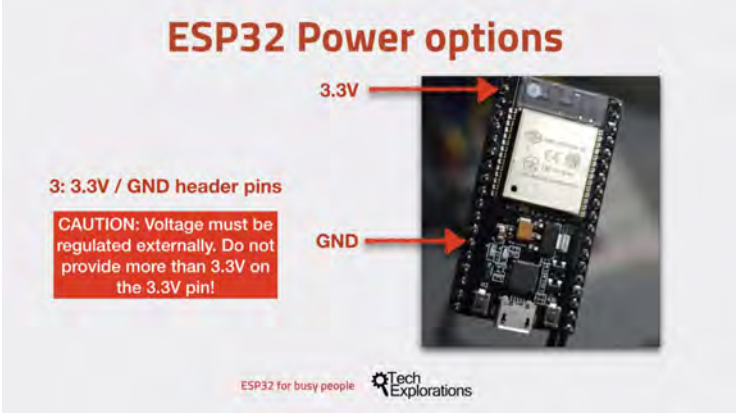
I did some experimentation using my bench power supply. I supplied voltage between 5V and 10V and observed the current draw. The ESP32 was running a sketch with an empty loop.At 10V input voltage, the current draw was 0.099 A (or 99.9mA). At 5V, the current draw was a little higher, at 0.128 A (or 128mA).

At 10V input voltage, the current draw was 99.9mA.

At 5V, the current draw was 128mA.

Option 3: Regulated power to GND and 3.3V pins

Another option that you have is to power your ESP32 is to use a 3.3V regulated power supply. For this, you will use the 3.3 Volt and GND pins.



You can connect a regulated 3.3V voltage supply to the 3.3V and GND pins.

The 3.3 volts pin is at the top left of the board right next to the antenna.

You have to be **very** careful when you do that. If you power your ESP32 this way, you're bypassing the on-board voltage regulator that is on board the dev kit, and therefore your module has no protection against over-voltage.

Again: Be very careful to make sure that your input voltage on the 3.3V pin is regulated and safe.

Power: conclusion

To power your ESP32 dev kit, you have three options:

1. Via the USB port.
2. Using unregulated voltage between 5V and 12V, connected to the 5V and GND pins. This voltage is regulated on-board.
3. Using regulated 3.3V voltage, connected to the 3.3V and GND pins. Be very careful with that: do not exceed the 3.3V limit, or your ESP32 module will be damaged.

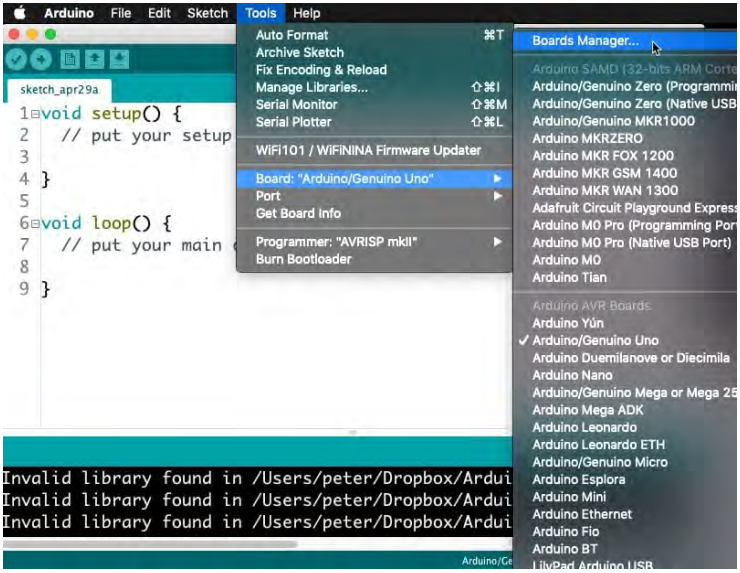
Attention: be very, very careful to **only use one of those options at the same time.**

For example, do not power your ESP32 dev kit via the 5V pin using a 10V input while at the same time you have the module connected to your computer via USB. This will surely damage your module, and perhaps even your computer.

With this, you should have a good understanding of what the ESP32 is, and you must be eager to get hands-on with it. I totally understand :-). Let's proceed with the next lesson, where I'll show you how to set up the ESP32-Arduino Core on the Arduino IDE.

Setup the ESP32 on the Arduino IDE (Mac OS)

In this lesson, you will learn how to set up your Arduino IDE on Mac OS, so that you can use it to program your ESP32.



In this lesson, you will learn how to set up your Arduino IDE so that you can use it to program your ESP32. This lesson contains instructions for Mac users. If you use Windows, please look at the Windows version of this lesson, also part of this series.

You can watch the video, or if you are the “reading” type, you

can read the text below.

One of the really nice things about the “ESP32” is that it works with the Arduino IDE.

Not only that, but because of the support that the manufacturer has implemented for the Arduino platform, we can use a lot of the existing Arduino libraries, infrastructure, and hardware.

That means that we can reuse what we already know from our work with the Arduino. We can build on that and make use of all of the additional hardware capabilities that the ESP32 brings along.

To be able to use the Arduino IDE to program the ESP32, you will need to install the ESP32-Arduino Core software. I will show you how to do this now.

I assume that the Arduino IDE is already installed on your Mac. If it isn’t, please install it now, and then continue (I’ll wait here, it’s ok).

Download ESP32-Arduino Core

With the Arduino IDE installed and operating on your computer, use your browser to download the ESP32 support software from Github.

You can download these files an manually copy them into the hardware folder of your Arduino IDE installation, but there is an easier and safer way. You should use the Arduino IDE Boards Manager. You can find instructions in Github (or just continue reading).

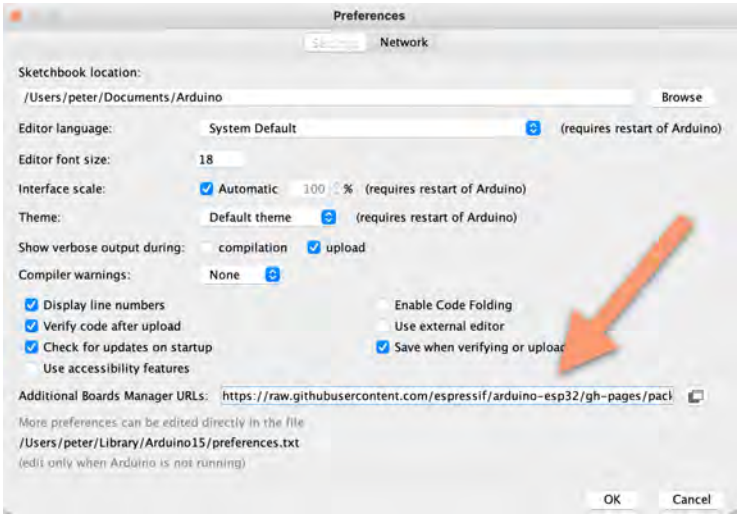
The Boards Manager utility in the Arduino IDE makes installation easy and safe.

The IDE Boards Manager utility works across platforms. You can use this method on Mac, Windows, or Linux.

Copy this URL:

> https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

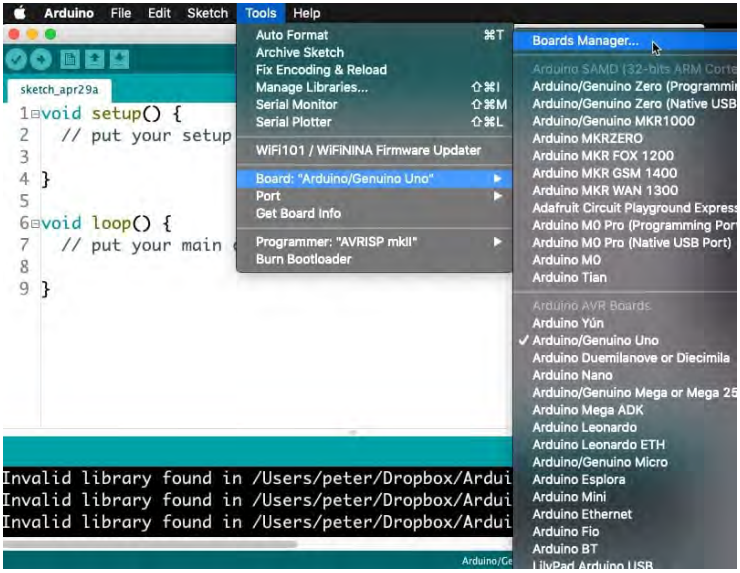
Open the Preferences window of the Arduino IDE, and paste the URL you just copied in the “Additional Board Manager URLs” field, as you can see in the figure below:



Copy the hardware definitions URL to the URLs field in the Preferences window.

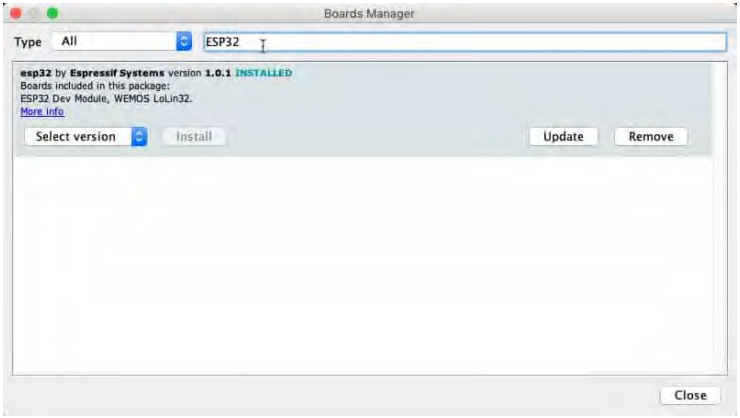
Click OK to dismiss the Preferences window.

Next, open the Board Manager utility by clicking Tools, Board, Boards Manager in the IDE menu.



Bring up the Boards Manager utility.

Search for “ESP32” in the text box. A single result should appear.Click on “Install” to do just that. In the Figure below, the ESP32 support is already installed in my Arduino IDE, so the “Install” button is inactive.



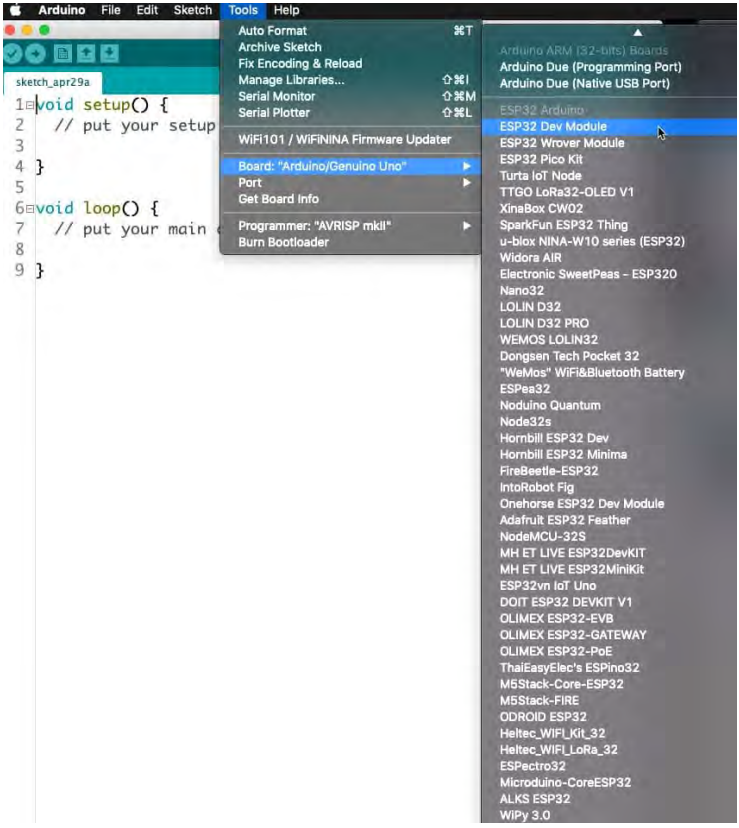
Search for the ESP32 module in the Boards Manager.

There are a few megabytes of file data to download and install, so be patient.

Test the ESP32 support

A while later, when the utility completes the download and installation of the software, confirm that the ESP32 support is there.

In the Arduino IDE menu, click on Tools, Board, and scroll down to see the new ESP32 section. There should be numerous boards there. The one we are interested mostly about, is the generic ESP32 Dev Module, like in the figure below. Click on “ESP32 Dev Module” to select it, and make it the active target in the IDE.

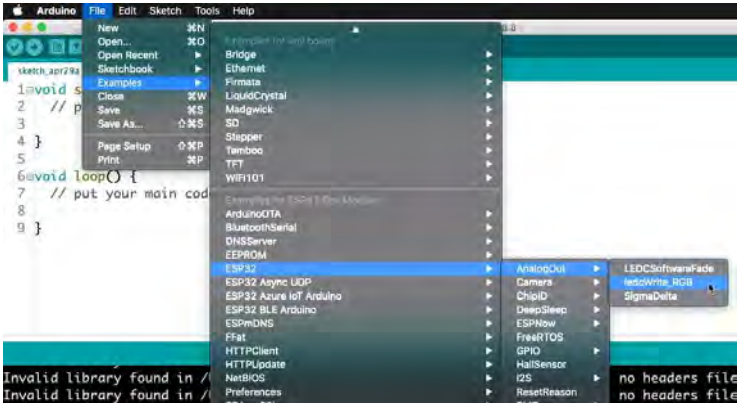


The ESP32 support software provides support for numerous ESP32 boards.

Now that you have selected the ESP32 Dev Module confirm the selection in the IDE window. The status line should contain the text “EXP32 Dev Module,” as well as its various parameters, like its frequency and flash memory size.

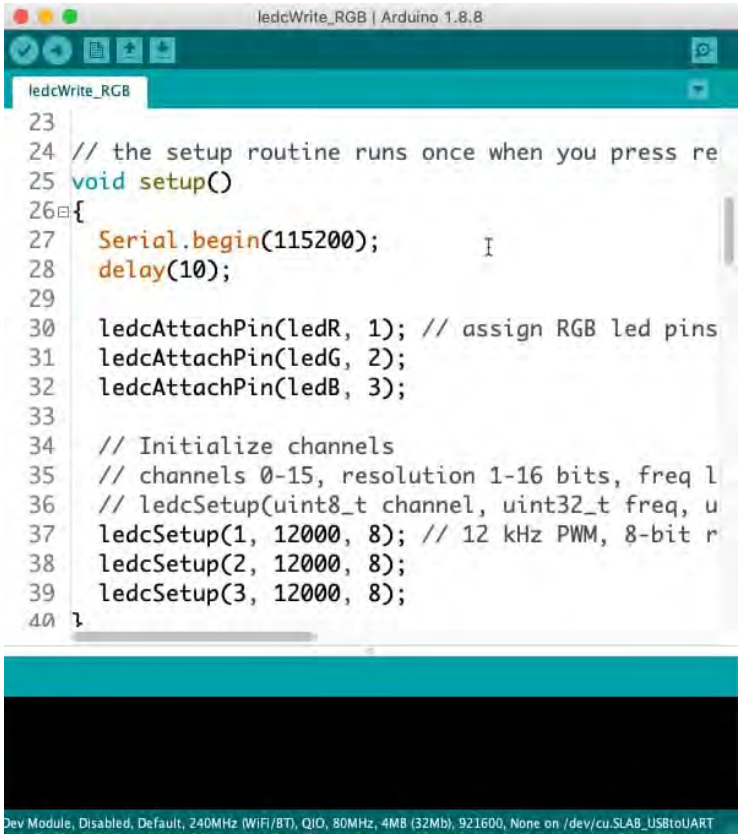
Yes, the ESP32 is the Arduino IDE programming target.

Let's do one last test: load one of the example ESP32 sketches. In the IDE menu, select File, Examples, ESP32, AnalogOut, ledcWrite_RGB. This is actually one of the sketches we discuss in the video lecture.



Load the ledcWrite_RGB example sketch.

This sketch demonstrates how to create PWM output, and uses functions that come with the ESP32-Arduino Core, and are not available to the Arduino Uno. You can learn more about these and other hardware-specific functions in the video course.



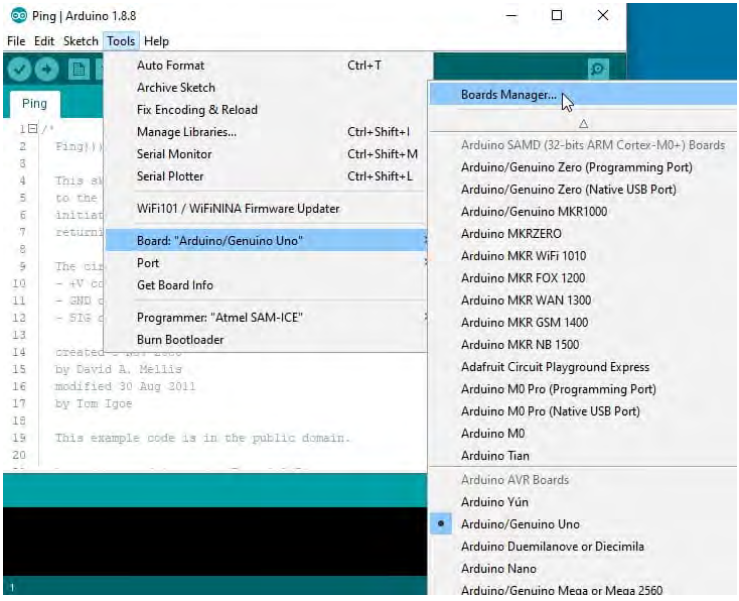
An ESP32 example sketch

This concludes the installation and verification of the Arduino IDE support for the ESP32 board. If you want to learn how to install the ESP32 support for Windows, check out the next lesson.

If your ESP32 Dev Kit contains the CP21012 USB driver chip (most lower-cost dev kits do), you will need to install the USB driver on your computer before you can upload your first sketch. You can learn how to do this in Lesson 9.

Setup the ESP32 on the Arduino IDE (Windows 10)

In this lesson, you will learn how to set up your Arduino IDE on Windows 10, so that you can use it to program your ESP32.



In this lesson, you will learn how to set up your Arduino IDE so that you can use it to program your ESP32. This lesson contains instructions for Windows users. If you use a Mac, please look at Lesson 7.You can watch the video, or if you are

the “reading” type, you can read the text below.

One of the really nice things about the ESP32 is that it works with the Arduino IDE.

Not only that, but because of the support that the manufacturer has implemented for the Arduino platform, we can use a lot of the existing Arduino libraries, infrastructure, and hardware.

That means that we can reuse what we already know from our work with the Arduino. We can build on that and make use of all of the additional hardware capabilities that the ESP32 brings along.

To be able to use the Arduino IDE to program the ESP32, you will need to install the ESP32-Arduino Core software. I will show you how to do this now.

I assume that the Arduino IDE is already installed on your PC. If it isn’t, please install it now, and then continue (I’ll wait here, it’s ok).

Download ESP32-Arduino Core

With the Arduino IDE installed and operating on your computer, use your browser to download the ESP32 support software from Github.

You can download these files an manually copy them into the hardware folder of your Arduino IDE installation, but there is an easier and safer way. You should use the Arduino IDE Boards Manager. You can find instructions in Github (or just continue reading).

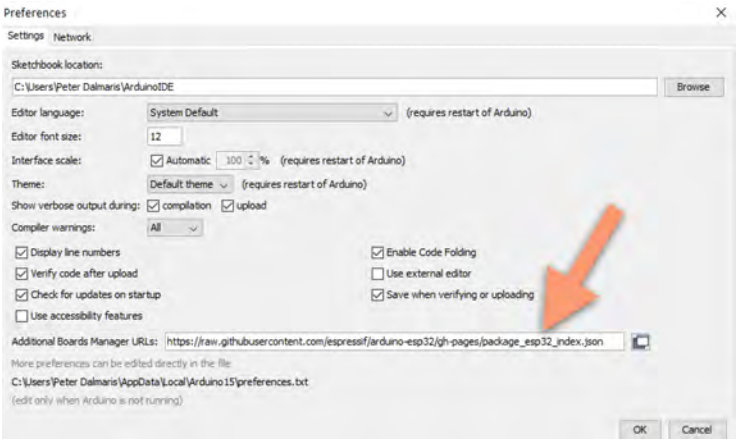
The Boards Manager utility in the Arduino IDE makes installation easy and safe.

The IDE Boards Manager utility works across platforms. You can use this method on Mac, Windows, or Linux.

Copy this URL:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

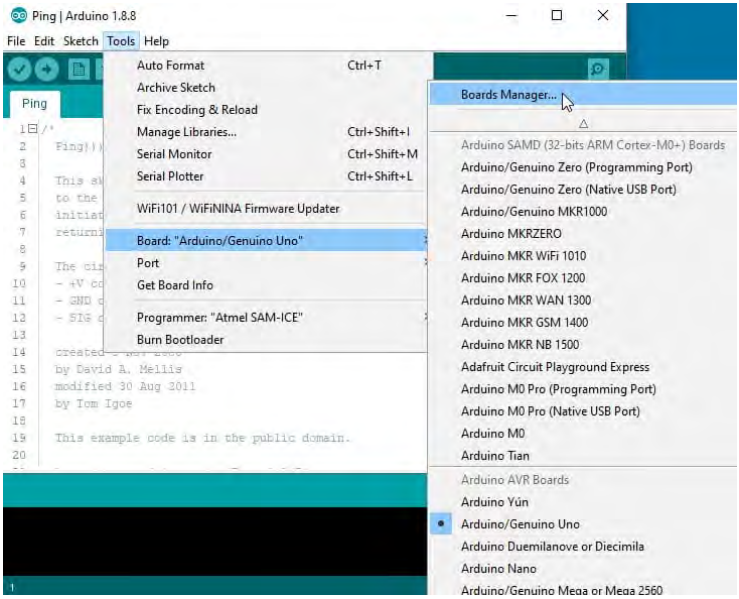
Open the Preferences window of the Arduino IDE, and paste the URL you just copied in the “Additional Board Manager URLs” field, as you can see in the figure below:



Copy the hardware definitions URL to the URLs field in the Preferences window.

Click OK to dismiss the Preferences window.

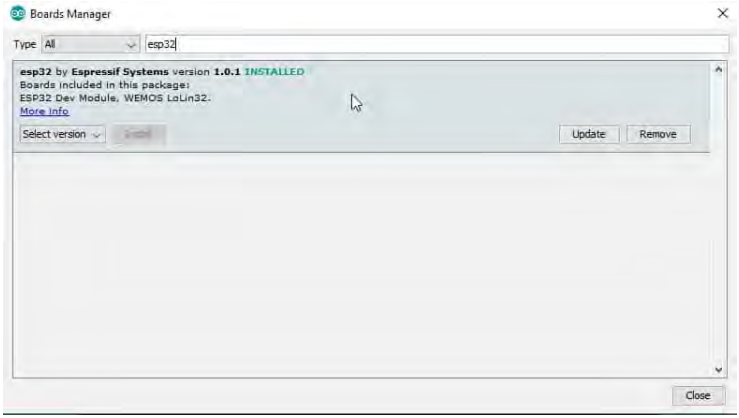
Next, open the Board Manager utility by clicking Tools, Board, Boards Manager in the IDE menu.



Bring up the Boards Manager utility.

Search for “ESP32” in the text box. A single result should appear.

Click on “Install” to do just that. In the Figure below, the ESP32 support is already installed in my Arduino IDE, so the “Install” button is inactive.



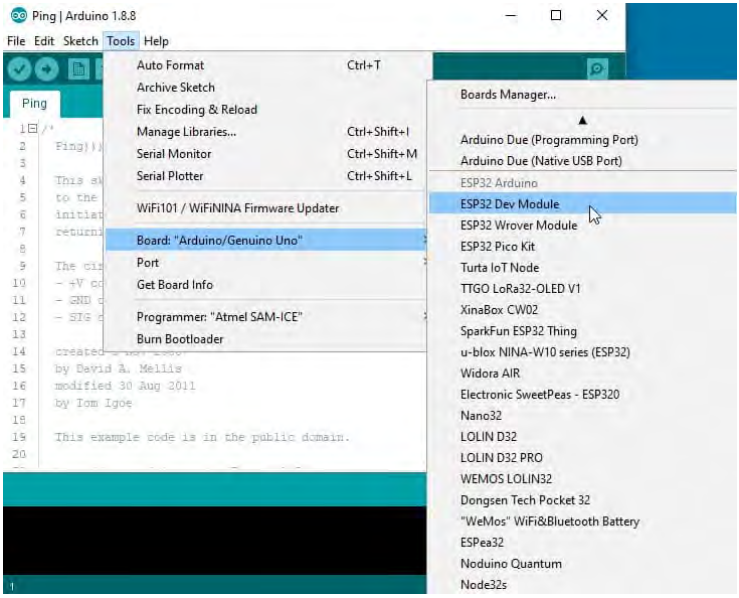
Search for the ESP32 module in the Boards Manager.

There are a few megabytes of file data to download and install, so be patient.

Test the ESP32 support

A while later, when the utility completes the download and installation of the software, confirm that the ESP32 support is there.

In the Arduino IDE menu, click on Tools, Board, and scroll down to see the new ESP32 section. There should be numerous boards there. The one we are interested mostly about, is the generic ESP32 Dev Module, like in the figure below. Click on “ESP32 Dev Module” to select it, and make it the active target in the IDE.



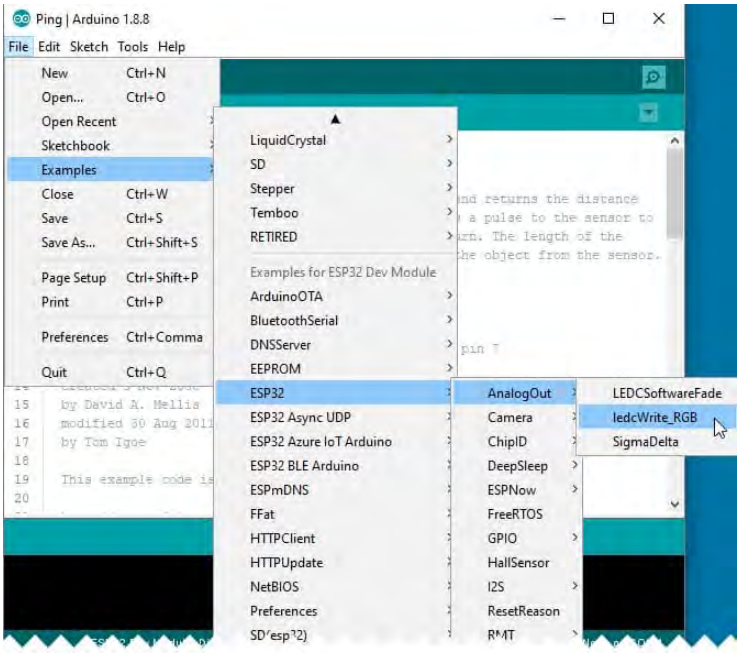
The ESP32 support software provides support for numerous ESP32 boards.

Now that you have selected the ESP32 Dev Module, confirm the selection in the IDE window. The status line should contain the text “EXP32 Dev Module,” as well as its various parameters, like its frequency and flash memory size.

Yes, the ESP32 is the Arduino IDE programming target.

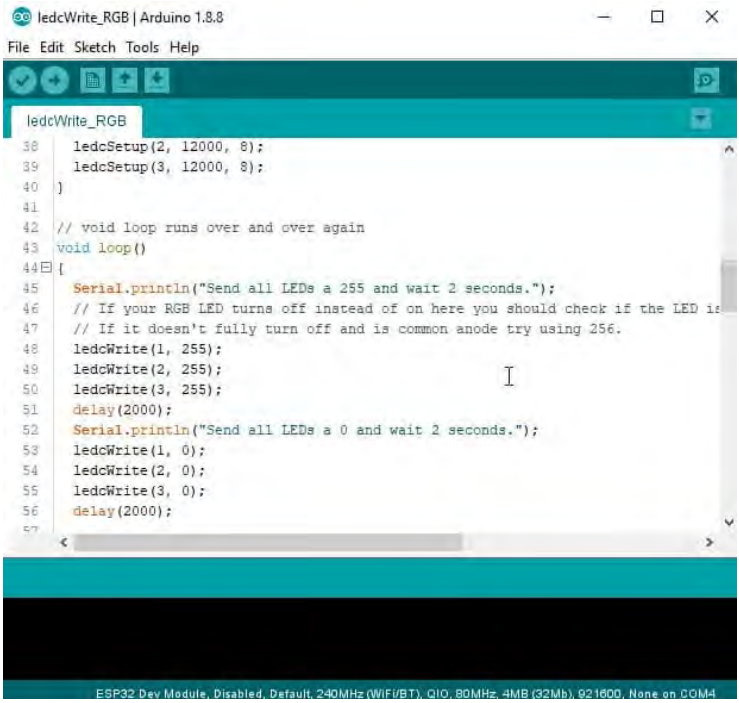
Let’s do one last test: Load one of the example ESP32 sketches. In the IDE menu, select File, Examples, ESP32,

AnalogOut, ledcWrite_RGB. This is actually one of the sketches we discuss in the video lecture.



Load the ledcWrite_RGB example sketch.

This sketch demonstrates how to create PWM output, and uses functions that come with the ESP32-Arduino Core, and are not available to the Arduino Uno. You can learn more about these and other hardware-specific functions in the video course.



An ESP32 example sketch.

This concludes the installation and verification of the Arduino IDE support for the ESP32 board. If you want to learn how to install the ESP32 support for the Mac, check out lesson 7.

If your ESP32 Dev Kit contains the CP21012 USB driver chip (most lower-cost dev kits do), you will need to install the USB driver on your computer before you can upload your first sketch. You can learn how to do this in Lesson 9.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

Install the drivers CP2102 for the USB bridge chip

In this lesson, I will show you how to install the driver for the CP210x family of USB to UART bridge chips. This chip is used on many ESP32 development boards to support USB communications.

In this lesson, I will show you how to install the driver for the CP210x family of USB to UART bridge chips. This chip is used on many ESP32 development boards to support USB

communications.

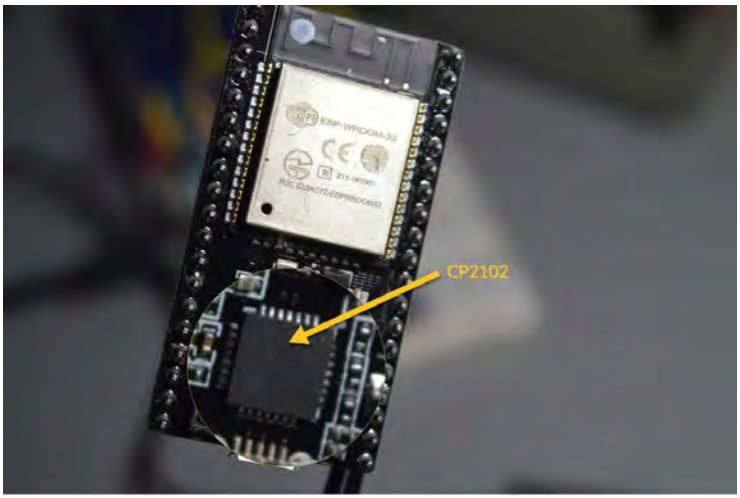
You can watch the video, or if you are the “reading” type, you can read the text below.

What is the CP210x USB to UART bridge?

Before we start experimenting with the ESP32 dev kit, I wanted to mention one issue that a lot of people come across, and that has to do with the sheer number of different development kits for the ESP32, and the small differences between them.

One of those differences has to do with the chip that is used to implement the USB to UART bridge which enables the USB programmability of the board.

My dev kit contains this chip, so I have to install the driver for the operating system, and it isn’t normally installed by default. So this chip here on my board, which is just stock standard as I’ve said before, requires a driver that typically is not installed by default on Windows or Mac OS computers and therefore has to be installed manually.



The CP2102 USB-UART bridge chip on my ESP32 dev board

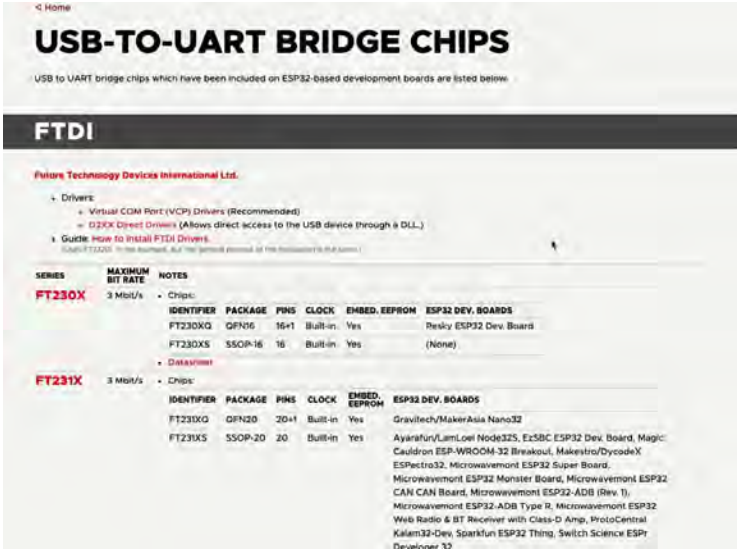
Without installing this particular driver, you will not be able to upload a sketch to the board.

How to identify your board's bridge chip

My particular board uses the CP2102 bridge chip. There's a good chance that your board uses the same one or at least a bridge from the same CP210x family as these are all popular low-cost USB to UART bridge chips.

To identify the USB to UART bridge chip on your board, first try to read the model number from the package of the chip itself. If at all possible, you will need a strong magnifying glass. In most of my boards, the UART bridge chips have no readable text on them, so I had to investigate further.

Go to esp32.net/usb-uart/, where you will find lists of bridge chips and the dev board on which they are used.



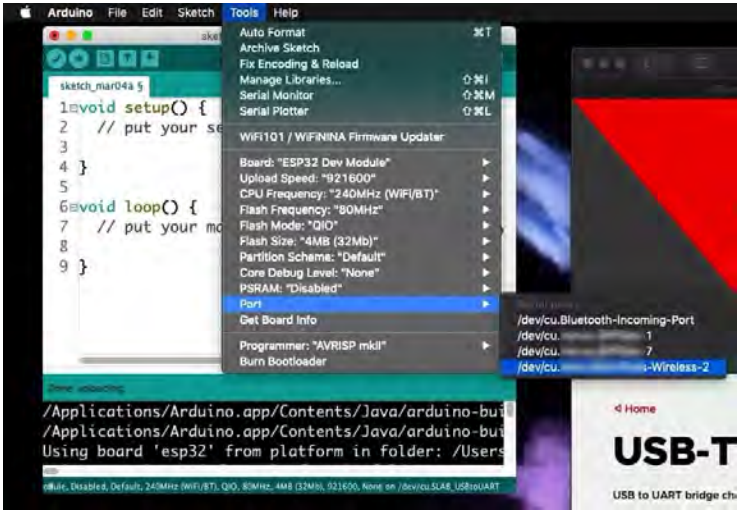
At esp32.net/usb-uart/, you will find information that will help you identify your board's UART bridge chip.

How do you know if you need to install the bridge driver?

You might be one of the lucky ones. Your ESP32 dev kit may be using a USB bridge chip, which is already supported by your operating system.

To determine that, connect your development kit to your computer via USB.

Then, start up your Arduino IDE, and look under Tools, Port.



If your dev kit is connected to your computer, but its port is not listed under "Port," you will need to install the USB-UART driver.

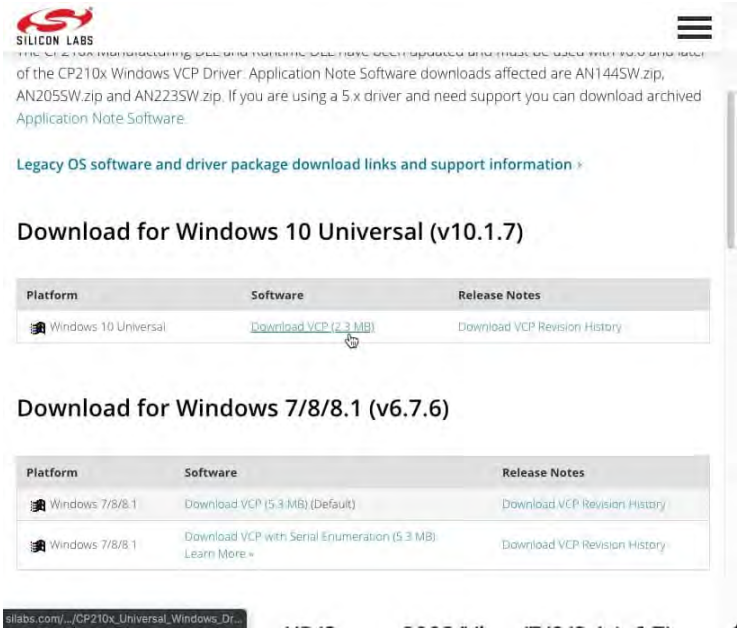
Can you see your dev kit port listed there? If not, you will need to install the third-party driver, so continue reading.

Install the USB-UART driver

Using the information available at esp32.net, determine the USB-UART bridge chip that is used in your ESP32 dev kit. I discovered that mine uses CP2102.

Once you have the model number, you need to find the driver that is appropriate for your operating system.

To do this, go to the Silicon Labs website. This link will take you directly to the driver download page.



The source of the USB-UART bridge drivers for the CP210x chips is Silicon Labs.

This driver is available for a variety of operating systems. Download the one for your operating system and install it.

Most likely, you will need to restart your operating system. If

you are not prompted to restart your operating system, be sure to restart the Arduino IDE.

Because of how many different drivers are out there and the differences between their operating systems and the exact installation procedure, I have not documented the driver installation process.

Verify the driver installation

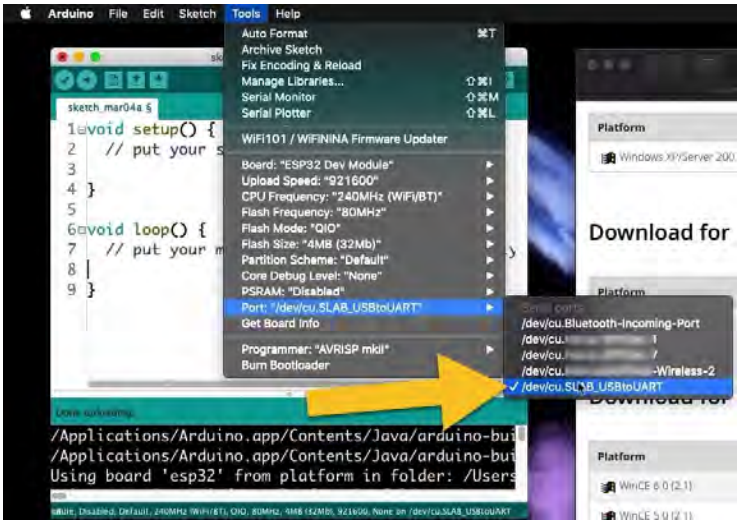
After the driver installation is complete, you should test it.

Start, or restart, your Arduino IDE.

Connect your ESP32 development kit to your computer.

Click on the Tools menu, then click on Port.

Can you see your development kit listed?

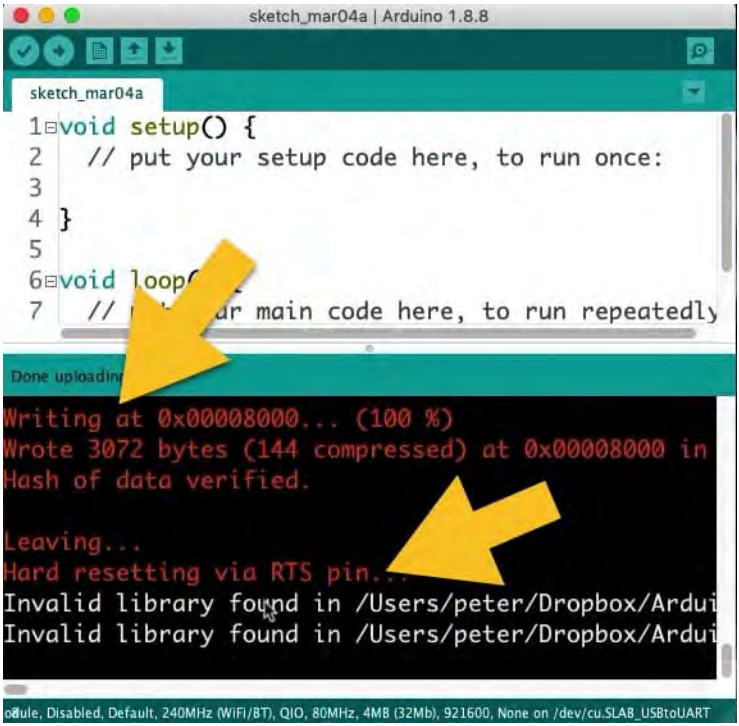


My dev kit is listed in the Port menu; the driver installation was successful.

Click on the dev kit port to make it the target for the Arduino IDE.

Let's try and upload the current sketch. It's ok that it is empty, we just want to verify that the Upload process works, nothing else.

Click on the Upload button (the IDE will probably ask you to save the sketch first; click Ok to that).



The test upload was successful.

Look at the log messages that appear in the Arduino IDE. You will be able to see the progress of the upload process until it reaches 100%, and the hash of the data is verified. Your board will then reset so that the newly uploaded sketch can start its execution.

And with this, your setup is complete, and you can start your experimentation with your ESP32. In the next lesson, you will learn how to make an LED blink, which is the obligatory first sketch for virtually any electronics platform.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

It's packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

Learn more

Jump to another article

It feels and works like an Arduino, but... WOW

From the Arduino Uno to the ESP32: Maker transformation

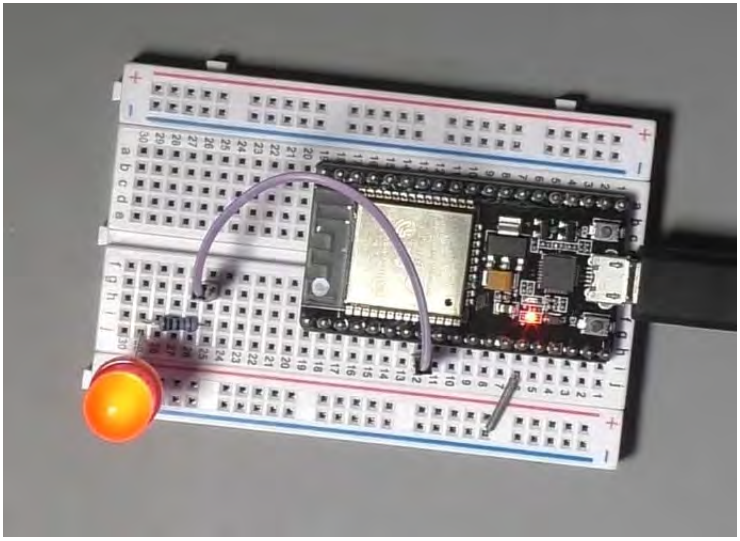
My first ESP32 project experience

Lessons

1: The ESP32 module2: The ESP32 Devkit3: The ESP32 vs Arduino4: The ESP32 GPIOs5: The ESP32 communications6: The ESP32 devkit power supply7: Setting up ESP32 in the Arduino IDE on Mac OS8: Setting up ESP32 in the Arduino IDE on Windows 109: Install the drivers CP2102 for the USB bridge chip10: Digital output LED11: Digital output PWM

Digital output experiment using an LED

In this lesson, you will start the practical exploration of the features of the ESP32 dev kit.Observing “tradition,” the first sketch and circuit I invite you to experiment with will make a red LED blink. By doing so, you will learn how to control the state of a GPIO on the ESP32.



In this lesson, you will start the practical exploration of the features of the ESP32 dev kit. Observing “tradition,” the first sketch and circuit I invite you to experiment with will make a red LED blink. By doing so, you will learn how to control the state of a GPIO on the ESP32.

You can watch the video, or if you are the “reading” type, you can read the text below.

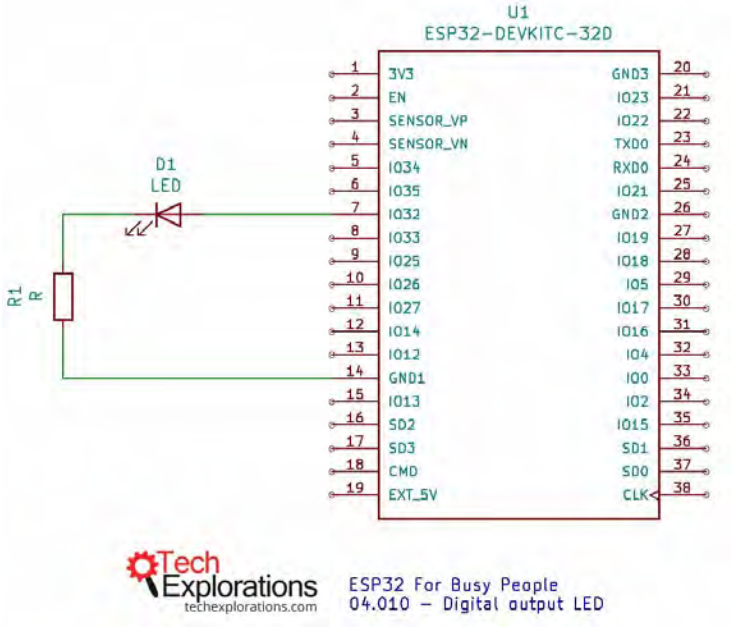
The wiring

This little example contains a single LED connected in series to a 230 resistor. You can use any resistor value between 230 and 500 , and the LED will be bright.

Careful, though, don’t go lower than 230 because then the LED will be drawing too much current from the ESP32, and this can cause damage.

Connect the anode of the LED to one of the GPIOs on the ESP32, and the cathode to the blue rail, which is connected to one of the GND pins of the ESP32 dev kit.

You can see the schematic below.

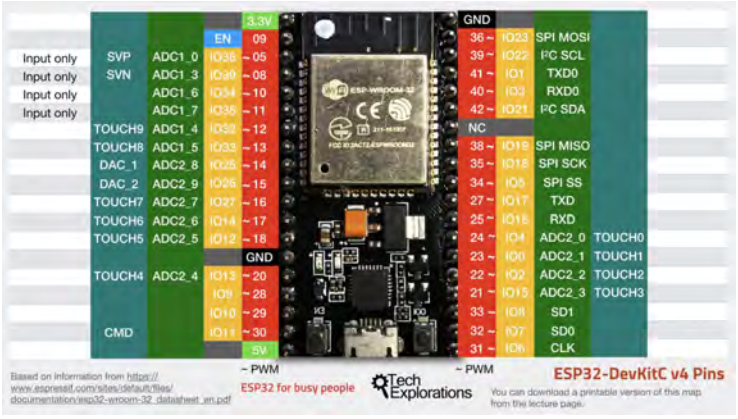


An LED is controlled by GPIO32 and is protected by a 230 resistor connected to GND1.

I have connected my LED to GPIO32, though I could have used any of the other pins (with some exceptions that I'm going to discuss later).

Pins

Throughout this course, I'll be using this pin map that I've put together. You have seen it already in lesson 4, but here it is again:



Peter's ESP32 pin map makes it easy to find the right pin for the job (click on the image for the hi-res version).

In this pin map, you can see the most frequently used functions of each pin.

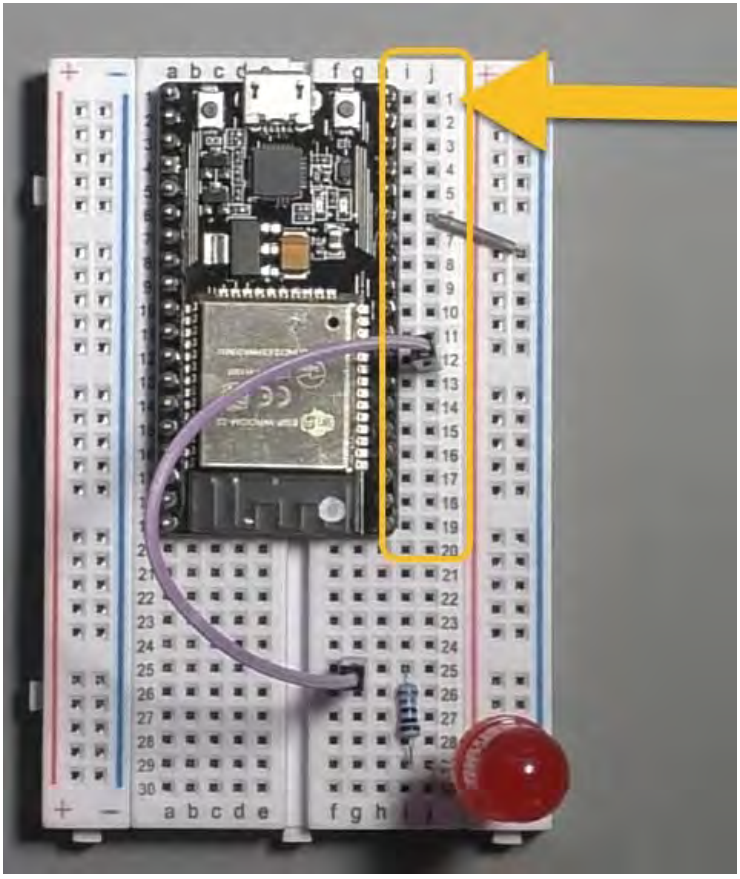
As you can see, GPIOs 34, 35, 36, 39 can only be used as inputs. Therefore, we can't use any of them to drive the LED. They can be used for other things, though.

Notice the little "~" marks? A pin that is marked with a "~" can be used for PWM (Pulse Width Modulation) output. I show

you how to generate PWM signals in the video course.

Working with the breadboard

For small circuits, you can plug your ESP32 dev kit to a mini breadboard, as I do in the photo below:



Place the dev kit on the mini breadboard so that holes from column 1 are exposed.

Attach the dev kit to the breadboard so that two rows of pins,

starting at column 1, are available for jumper wires. In the majority of the experiments that I demonstrate in the course, I use the pins on the right side of the dev kit, so this arrangement works well.

By placing the top-left pin of the dev kit on column 1 of the breadboard, it is easier to count pins and plug-in jumper wires as you can look at the column numbers on the breadboard.

The sketch

Let's have a look at the sketch now.

Instead of loading a pre-written sketch, you will load the blink LED example and then modify it slightly to get it to work with the ESP32.

Load the example sketch by clicking on File, Examples, Basics, Blink.

And this is the classic Arduino blink example. It should look like this:

// the setup function runs once when you press reset or power the boardvoid setup() { // initialize digital pin LED_BUILTIN as an output. pinMode(LED_BUILTIN, OUTPUT);}// the loop function runs over and over again forevervoid loop() { digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level) delay(1000); // wait for a second digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW delay(1000); // wait for a second}

This sketch will almost work out of the box.

There is a reference to the LED_BUILTIN constant, which does not work with the ESP32, and you must address it.

Create a new single-byte constant, and name it "led_gpio." The LED is connected to GPIO32, so store the value "32" in

“led_gpio.”

Then, replace all occurrences of the LED_BUILTIN reference with “led_gpio.”

The sketch should now look like this:

const byte led_gpio = 32;// the setup function runs once when you press reset or power the boardvoid setup() { // initialize digital pin LED_BUILTIN as an output. pinMode(led_gpio, OUTPUT);}// the loop function runs over and over again forevervoid loop() { digitalWrite(led_gpio, HIGH); // turn the LED on (HIGH is the voltage level) delay(1000); // wait for a second digitalWrite(led_gpio, LOW); // turn the LED off by making the voltage LOW delay(1000); // wait for a second}

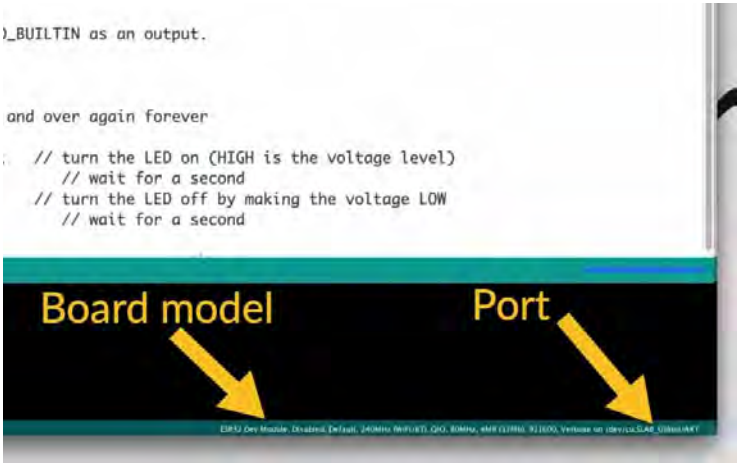
That’s all the modifications necessary. You can go ahead and upload this sketch to your board to see it working.

Upload to the board

Connect your ESP32 board to your computer.

Make sure the board model is selected (look at the status line of the Arduino IDE for the text “ESP32 Dev Module”). If the ESP32 module is not the current target, select it from the Tools menu, Board, “ESP32 Dev Module.”

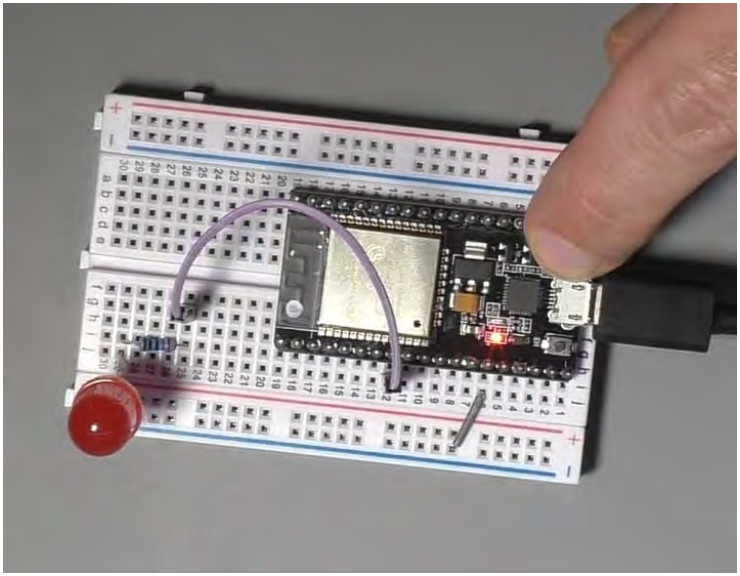
Then, check that the USB port for the ESP32 module is selected. Again, look at the status line for confirmation, and if the com/serial port does not look correct, select the correct one from the Tools menu, Port.



Check that your board is ready to receive the sketch.

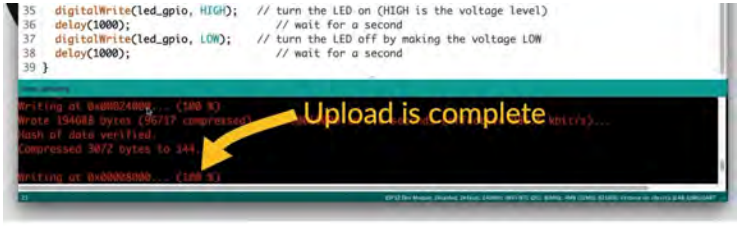
When the board model and port selections checkout (i.e., they are correct), it's time to upload the sketch.

Click on the Upload button. If your board is like mine, and can't automatically go into upload mode, press and hold the "BOOT" button immediately after you click on the Upload button. Then, observe the Arduino IDE output pane for upload progress information. When the upload starts, you can release the BOOT button.



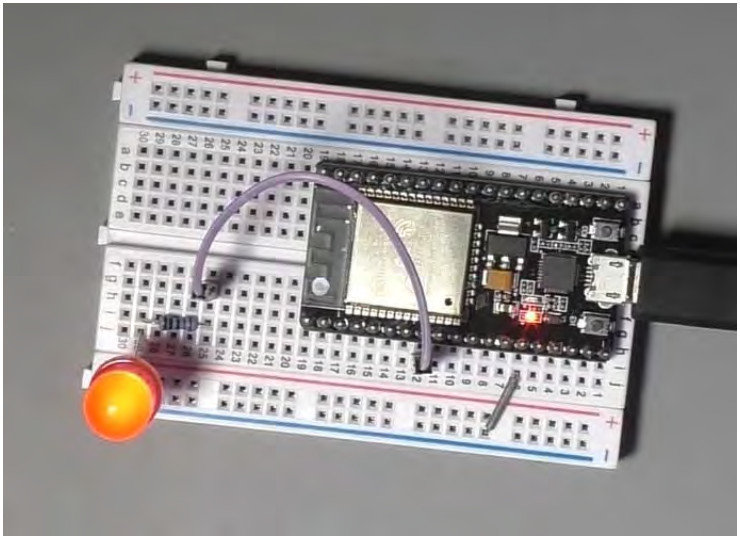
Press and hold the BOOT button until the upload begins.

The upload will complete when progress reaches 100%.



Writing progress is 100%.

When the upload is complete, the board will automatically reset, and the newly uploaded sketch will start running. The LED will blink at a rate of 1Hz.



The LED is blinking, so the sketch is running on the ESP32.

This wasn't too hard, right?

Let's take this one step further in the next lesson. You will use the same circuit to make the LED fade on and off, using the ESP32 PWM capability.

Ready for some serious learning?

Enrol to

ESP32 for Busy People

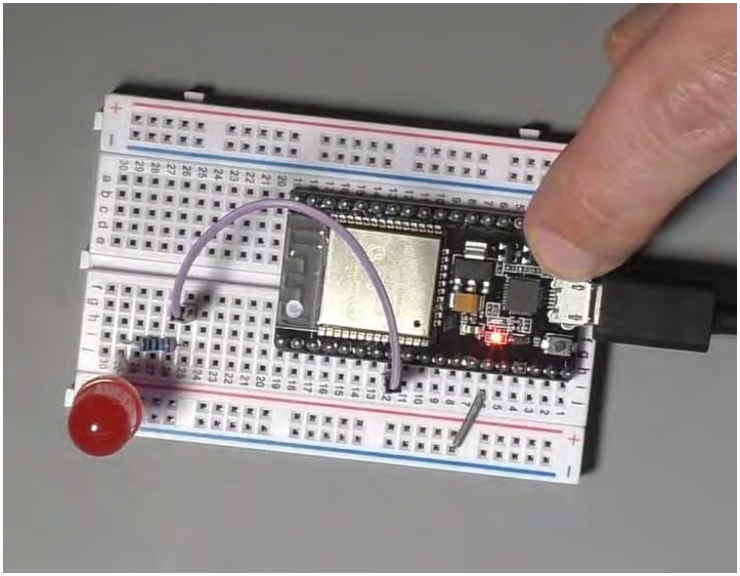
This is our comprehensive ESP32 course for Arduino Makers.

It's packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.

Just click on the big red button to learn more.

PWM output experiment using an LED

In this article, I'll show you how to make an LED fade using the PWM capability of the ESP32.



In the previous lesson, you learned how to get the LED connected an ESP32 to blink. In this article, I'll show you how to make it fade using the PWM capability of the ESP32.

If you are not familiar with PWM, please checkout Arduino Step by Step Getting Started.

The Arduino, of course, can also output PWM. But as you'll see, the ESP32 has got several additional capabilities in its hardware that the Arduino Uno with the Atmega328 cannot even imagine.

You can watch the video, or if you are the "reading" type, you can read the text below.

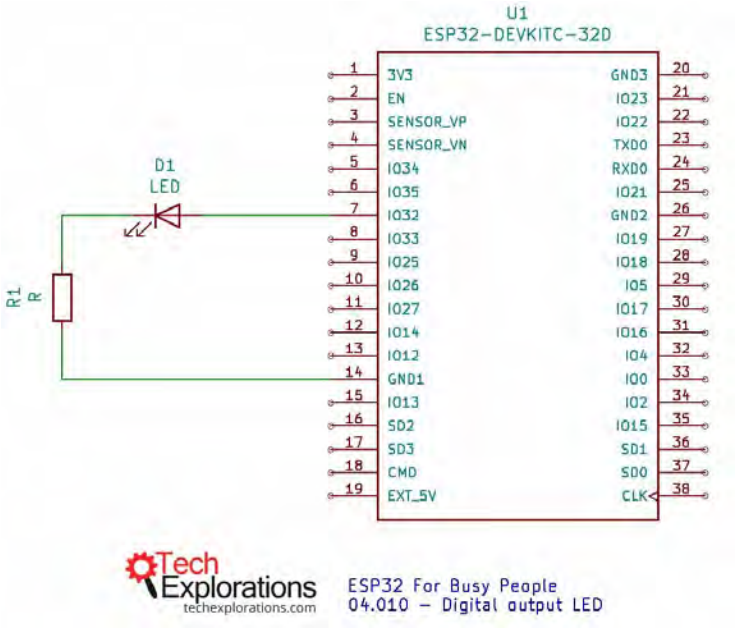
The wiring

For this experiment, you can reuse the circuit from the previous lesson.

The circuit contains a single LED connected in series to a 230 resistor. You can use any resistor value between 230 and 500 , and the LED will be bright.

Connect the anode of the LED to GPIO32 on the ESP32, and the cathode to the blue rail on the breadboard, which is connected to one of the GND pins of the ESP32 dev kit.

You can see the schematic below.



An LED is controlled by GPIO32 and is protected by a 230 resistor connected to GND1.

I have connected my LED to GPIO32, though I could have used any of the other pins (with some exceptions that I'm going to discuss later).

GPIO32, like most other GPIOs, are PWM-capable.

The sketch

Here, I think it is useful to compare the Arduino version of the fading LED sketch, with the ESP version. I have them side-by-side below:

Arduino version

int led = 9; // the PWM pin the LED is attached toint brightness

= 0; // how bright the LED isint fadeAmount = 5; // how many points to fade the LED by// the setup routine runs once when you press reset:void setup() { // declare pin 9 to be an output: pinMode(led, OUTPUT);}// the loop routine runs over and over again forever:void loop() { // set the brightness of pin 9: analogWrite(led, brightness); // change the brightness for next time through the loop: brightness = brightness + fadeAmount; // reverse the direction of the fading at the ends of the fade: if (brightness <= 0 || brightness >= 255) { fadeAmount = -fadeAmount; } // wait for 30 milliseconds to see the dimming effect delay(30);}

ESP32 version

const byte led_gpio = 32; // the PWM pin the LED is attached toint brightness = 0; // how bright the LED isint fadeAmount = 5; // how many points to fade the LED by// the setup routine runs once when you press reset:void setup() { ledcAttachPin(led_gpio, 0); // assign a led pins to a channel // Initialize channels // channels 0-15, resolution 1-16 bits, freq limits depend on resolution // ledcSetup(uint8_t channel, uint32_t freq, uint8_t resolution_bits); ledcSetup(0, 4000, 8); // 12 kHz PWM, 8-bit resolution}// the loop routine runs over and over again forever:void loop() { ledcWrite(0, brightness); // set the brightness of the LED // change the brightness for next time through the loop: brightness = brightness + fadeAmount; // reverse the direction of the fading at the ends of the fade: if (brightness <= 0 || brightness >= 255) { fadeAmount = -fadeAmount; } // wait for 30 milliseconds to see the dimming effect delay(30);}

In the Arduino, we use the *analogWrite()* function to “write” a value between 0 and 254 to the LED pin. Larger values will cause the LED to stay on for longer during a cycle, and thus the LED will be brighter.

The *analogWrite()* function accepts two parameters:

1. The number of the pin to which the LED is

connected
2. The PWM value from 0 to 254

Inside the loop function, each time it is called, the *brightness* value is increased or decreased by 5, depending on whether we are increasing or decreasing the LED brightness.

Simple.

The ESP32 does not support the *analogWrite()* function. But it does support a much better one, the *ledcWrite()* function.

The *ledcWrite()* is very similar to *analogWrite().* It also requires two parameters:

1. The PWM channel that we want to "write" a value to.
2. The PWM value we want to write to the selected channel.

This simple function shows off the power of the ESP32. Both the channel and the PWM values are configurable.

The PWM channel is not the same as a physical pin. The ESP32 contains 16 independent channels. Each of those can be assigned to any PWM-capable pin. This means that you can use any 16 GPIOs to generate PWM output.

The resolution of the PWM signal is also configurable. While in the Arduino, a PWM signal is set to 8 bit, in the ESP32, it can be whatever you choose, from 1 to 16 bits.

To assign a PWM channel to a pin, and configure the resolution of the signal, the ESP32-Arduino Core software provides two functions:

- *ledcAttachPin(gpio, channel)*
- *ledcSetup(channel, frequency, resolution)*

In *ledcAttachPin()*, pass the GPIO number and the channel number that you want to bind. In the example sketch, you can see:

ledcAttachPin(led_gpio, 0);

This binds the PWM channel “0” to GPIO32.

We use *ledcSetup(channel, frequency, resolution)* to configure the PWM signal. In the example sketch, you can see:

ledcSetup(0, 4000, 8)

The first parameter is “0”. This means that we are configuring the PWM channel “0”.

The second parameter is 4000, which means that we have chosen the PWM frequency to be 4KHz. The frequency range depends on the resolution you have chosen, but typical values for an 8-bit resolution are from 4KHz to 8KHz. If you are curious about the details, read this.

The third parameter is “8,” meaning 8 bits. You can set the resolution to any value between 1 and 16 bits.

Use *ledcAttachPin()* and *ledcSetup()* function in setup() to... setup the PWM channel 0, and then you are ready to start creating PWM output.

Inside the ESP32 version of the sketch, have a look at the loop(). Apart from the substitution of *analogWrite()* with *ledcWrite()*, the code should look very familiar.

PWM signal visualization with the oscilloscope

The ESP32 PWM output is very configurable, and this helps in creating ver precise output. I used my oscilloscope to see the PWM signal that was generated by my ESP32. It looks like this:

My oscilloscope visualizes the ESP32 8-bit, 4KHz PWM output.

Please watch the video at the top of this article for the details of my experimentation with my ESP32 and the oscilloscope.

This concludes this introductory series on the ESP32. There's a lot more to learn.

If you are keen to make the most of your ESP32, consider enrolling in my full video course, "ESP32 For Busy People." Unless you have time to waste

Ready for some serious learning?

Enrol to

ESP32 for Busy People

This is our comprehensive ESP32 course for Arduino Makers.

It's packed with high-quality video, mini-projects, and everything you need to learn Arduino from the ground up.