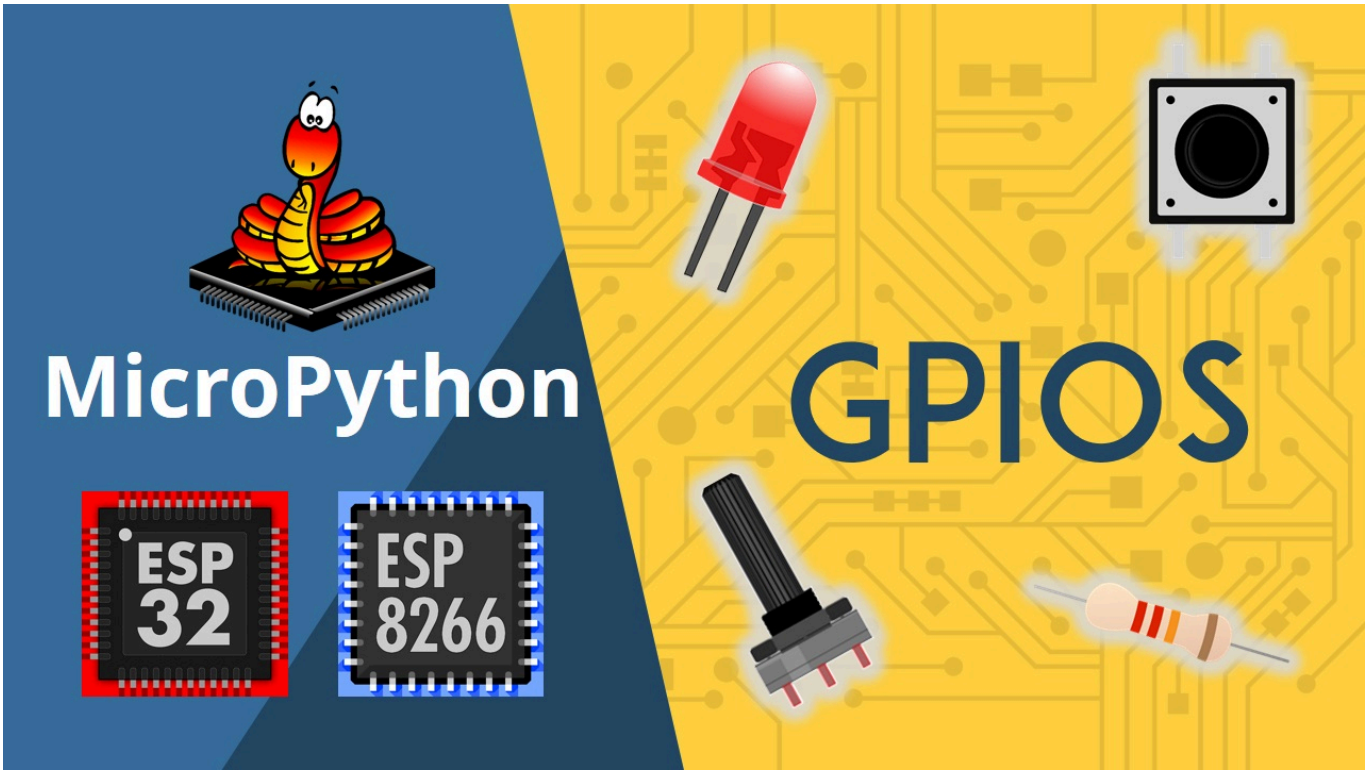
MicroPython with ESP32 and ESP8266: Interacting with GPIOs

In this article we're going to take a look on how to interact with the ESP32 and ESP8266 GPIOs using MicroPython. We'll show you how to read digital and analog inputs, how to control digital outputs and how to generate PWM signals.



Prerequisites

To program the ESP32 and ESP8266 with MicroPython, we use uPyCraft IDE as a programming environment. Follow the next tutorials to install uPyCraft IDE and flash MicroPython firmware on your board:

- Install uPyCraft IDE: Windows PC, MacOS X, or Linux Ubuntu
- Flash/Upload MicroPython Firmware to ESP32 and ESP8266

Alternatively, if you're having trouble using uPyCraftIDE, we recommend using Thonny IDE instead: Getting Started with Thonny MicroPython (Python) IDE for ESP32 and ESP8266

If this is your first time dealing with MicroPython you may find these next tutorials useful:

- Getting Started with MicroPython on ESP32 and ESP8266
- MicroPython Programming Basics with ESP32 and ESP8266

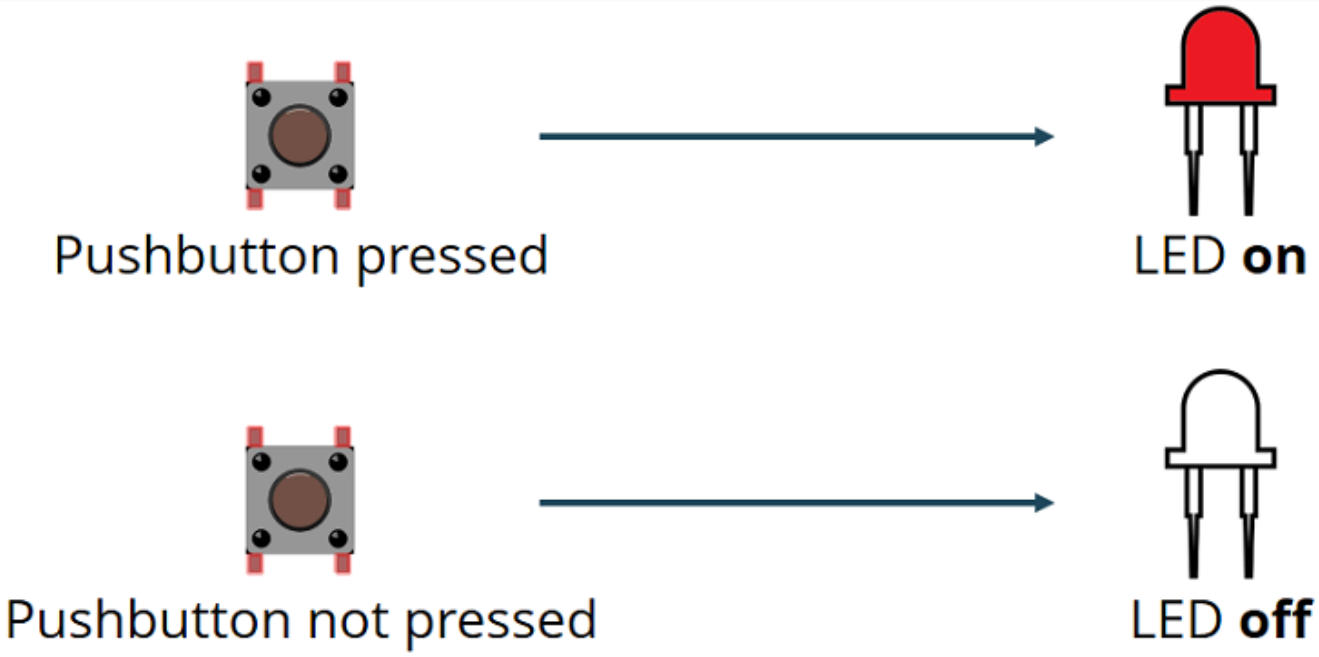
Project Overview

With this tutorial you'll learn how to use the ESP32 or ESP8266 GPIOs with MicroPython. You can read the separate guide for each topic:

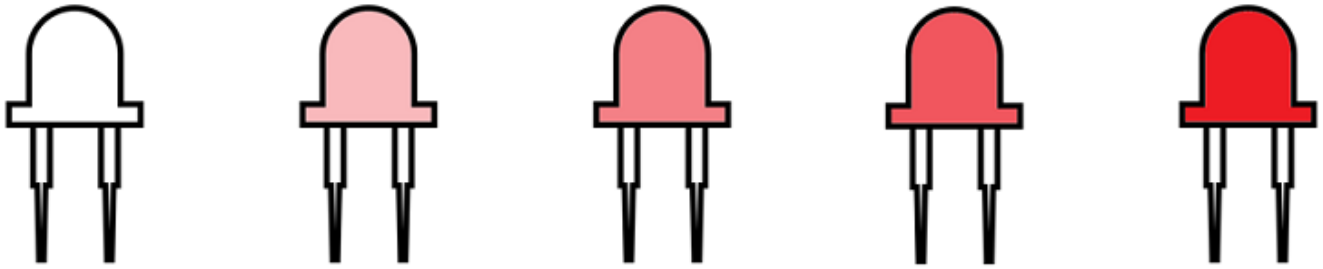
- Read digital inputs
- Control digital outputs
- Read analog inputs
- Generate PWM signals

We'll build a simple example that works as follows:

- Read the state of a pushbutton and set the LED state accordingly – when you press the pushbutton the LED lights up.



- Read the voltage from a potentiometer and dim an LED accordingly to the shaft's position of the potentiometer.



Schematic

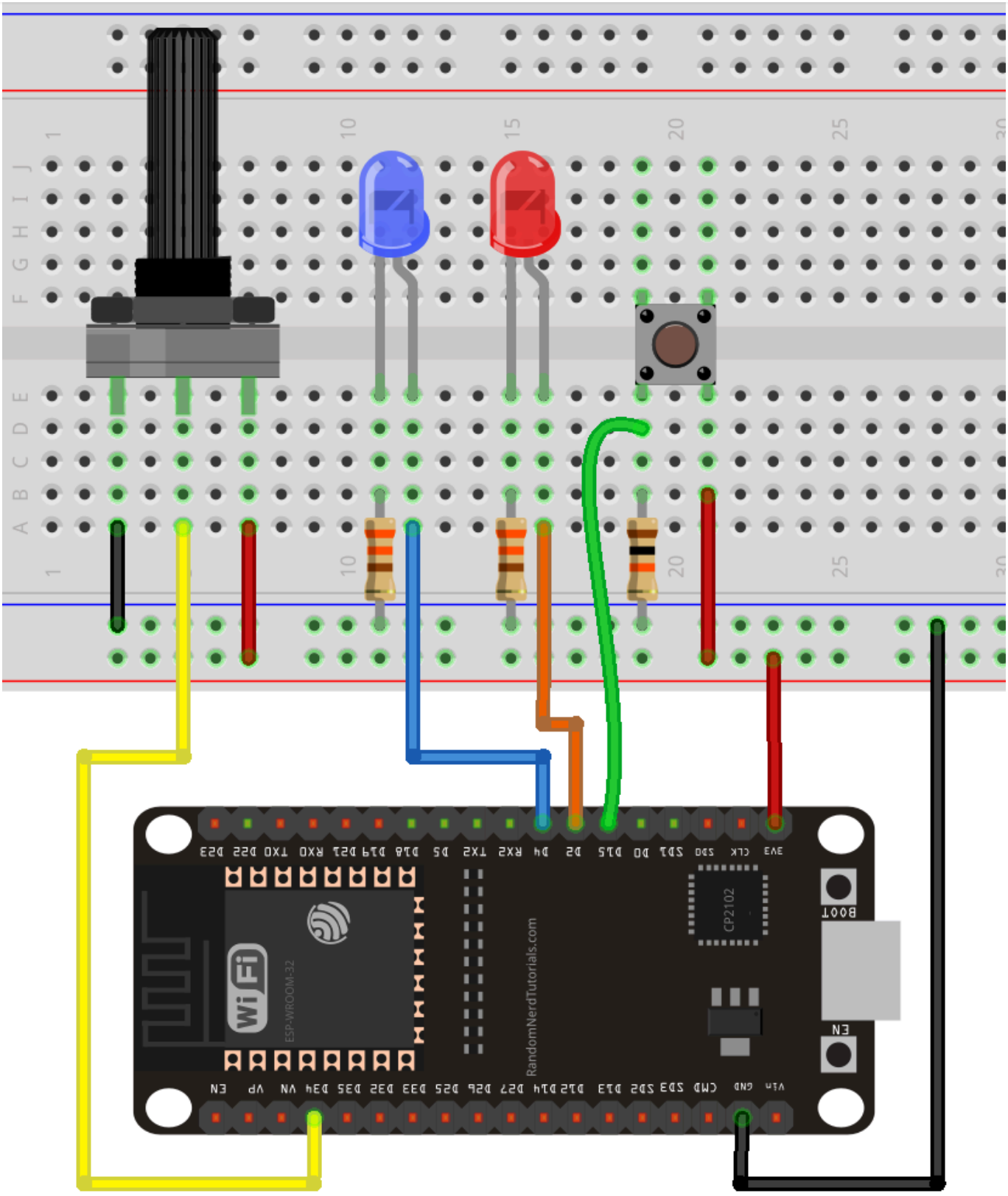
The circuit for this project involves wiring two LEDs, a pushbutton, and a potentiometer. Here's a list of all the parts needed to build the circuit:

- ESP32 or ESP8266 (read: ESP32 vs ESP8266)
- 2x LEDs
- 2x 330 Ohm resistor
- Pushbutton
- Potentiometer
- Breadboard
- Jumper wires

You can use the preceding links or go directly to MakerAdvisor.com/tools to find all the parts for your projects at the best price!

ESP32 – Schematic

Follow the next schematic diagram if you're using an ESP32:

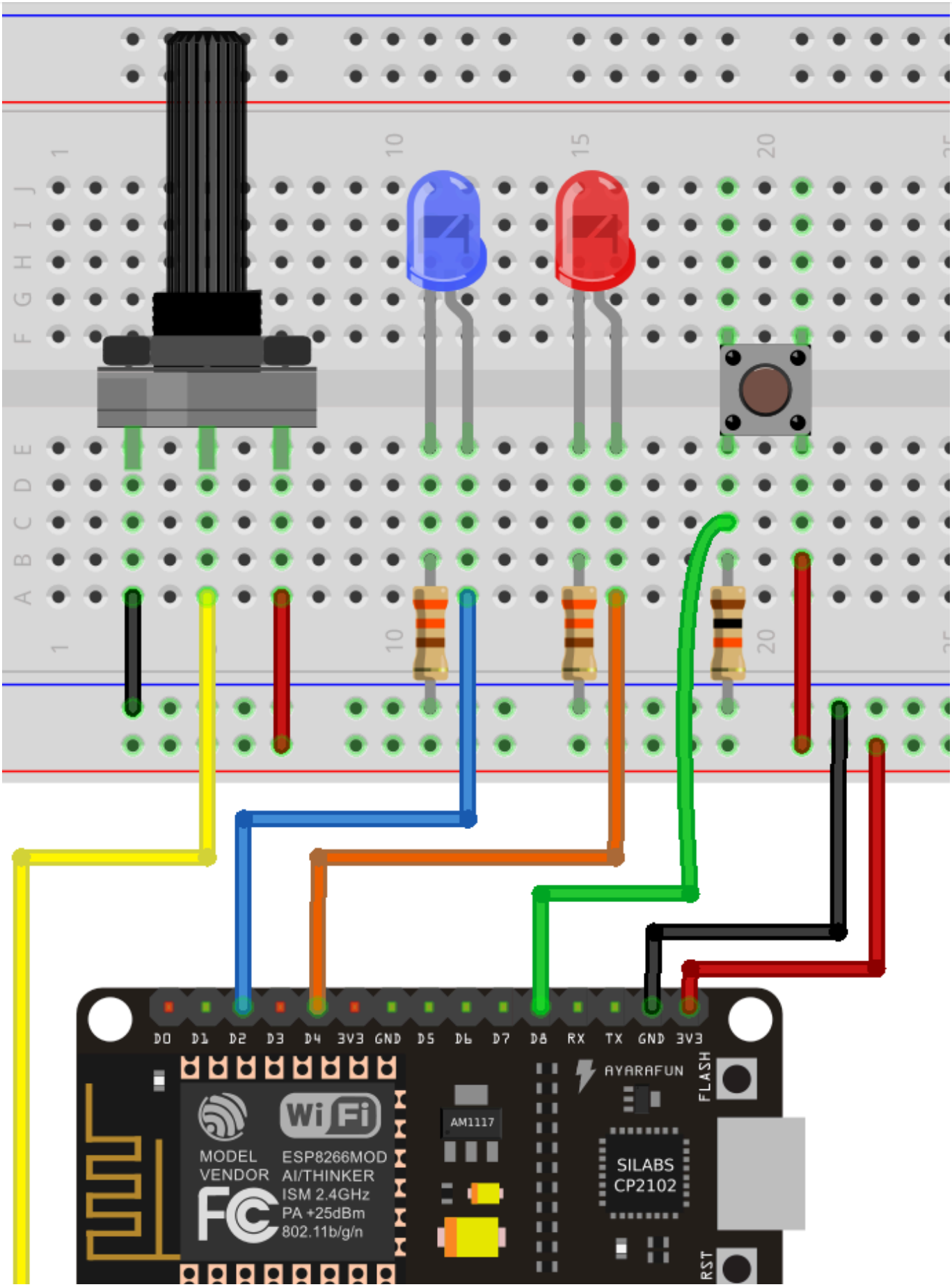


Note: the ESP32 supports analog reading in several GPIOs: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27 32, 33, 34, 35, 36, and 39.

Recommended reading: ESP32 Pinout Reference: Which GPIO pins should you use?

ESP8266 – Schematic

Follow the next schematic diagram if you're using an ESP8266:


D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
AYARAFUN
FLASH
RST
AM1117
SILABS
CP2102
MODEL ESP8266MOD
VENDOR AI/THINKER
ISM 2.4GHz
PA +25dBm
802.11b/g/n

Note: the ESP8266 only supports analog reading in pin ADC0 (A0).

Code

Copy the following code to the *main.py* file in the uPyCraft IDE.

Note: analog reading works differently in ESP32 and ESP8266. The code works right away in ESP32. To use with ESP8266, you have to uncomment and comment the lines described in the MicroPython script.

```
# Complete project details at https://RandomNerdTutorials.com
# Created by Rui Santos

from machine import Pin, ADC, PWM
from time import sleep

led = Pin(2, Pin.OUT)
button = Pin(15, Pin.IN)

#Configure ADC for ESP32
pot = ADC(Pin(34))
pot.width(ADC.WIDTH_10BIT)
pot.atten(ADC.ATTN_11DB)

#Configure ADC for ESP8266
#pot = ADC(0)

led_pwm = PWM(Pin(4),5000)

while True:
```

```
    pot_value = pot.read()
    led_pwm.duty(pot_value)

    sleep(0.1)
```

View raw code

How the code works

Continue reading to learn on how the code works.

Importing Libraries

To interact with the GPIOs you need to import the `machine` module that contains classes to interact with the GPIOs. Import the `Pin` class to interact with the pins, the `ADC` class to read analog value, and the `PWM` class to generate PWM signals.

```
from machine import Pin, ADC, PWM
```

Import the `sleep()` method from the `time` module. The `sleep()` method allows you to add delays to the code.

```
from time import sleep
```

Instantiating Pins

After importing all the necessary modules, instantiate a `Pin` object called `led` on GPIO 2 that is an OUTPUT.

```
led = Pin(2, Pin.OUT)
```

```
Pin(Pin number, pin mode, pull, value)
```

- **Pin number** refers to the GPIO we want to control;
- **Pin mode** can be input (`IN`), output (`OUT`) or open-drain (`OPEN_DRAIN`);
- The **pull** argument is used if we want to activate a pull up or pull down internal resistor (`PULL_UP` , or `PULL_DOWN`);
- The **value**corresponds to the GPIO state (if is is on or off): it can be `0` or `1` (True or False). Setting `1` means the GPIO is on. If we don't pass any parameter, its state is `0` by default (that's what we'll do in this example).

After instantiating the `led` object, you need another instance of the `Pin` class for the pushbutton. The pushbutton is connected to GPIO 15 and it's set as an input. So, it looks as follows:

```
button = Pin(15, Pin.IN)
```

Instantiating ADC

In the ESP32, to create an `ADC` object for the potentiometer on GPIO 34:

```
pot = ADC(Pin(34))
```

If you're using an ESP8266, it only supports ADC on ADC0 (A0) pin. To instantiate an ADC object with the ESP8266:

```
pot = ADC(0)
```

The following line applies just to the ESP32. It defines that we want to be able to read voltage in full range. This means we want to read voltage from 0 to 3.3 V.

```
pot.atten(ADC.ATTN_11DB)
```

The next line means we want readings with 10 bit resolution (from 0 to 1023)

```
pot.width(ADC.WIDTH_10BIT)
```

The `width()` method accepts other parameters to set other resolutions:

- **WIDTH_9BIT**: range 0 to 511
- **WIDTH_10BIT**: range 0 to 1023
- **WIDTH_11BIT**: range 0 to 2047
- **WIDTH_12BIT**: range 0 to 4095

If you don't specify the resolution, it will be 12-bit resolution by default on the ESP32.

Instantiating PWM

Then, create a `PWM` object called `led_pwm` on GPIO 4 with 5000 Hz.

```
led_pwm = PWM(Pin(4), 5000)
```

To create a `PWM` object, you need to pass as parameters: pin, signal's frequency, and duty cycle.

The **frequency** can be a value between 0 and 78125. A frequency of 5000 Hz for an LED works just fine.

The **duty cycle** can be a value between 0 and 1023. In which 1023 corresponds to 100% duty cycle (full brightness), and 0 corresponds to 0% duty cycle (unlit LED).

We'll just set the duty in the while loop, so we don't need to pass the duty cycle

Getting the GPIO state

Then, we have a while loop that is always True. This is similar to the `loop()` function in the Arduino IDE.

We start by getting the button state and save it in the `button_state` variable. To get the pin state use the `value()` method as follows:

```
button_state = button.value()
```

This returns 1 or 0 depending on whether the button is pressed or not.

Setting the GPIO state

To set the pin state, use the `value(state)` method in the `Pin` object. In this case we're setting the `button_state` variable as an argument. This way the LED turns on when we press the pushbutton:

```
led.value(button_state)
```

Reading analog inputs

To read an analog input, use the `read()` method on an `ADC` object (in this case the ADC object is called `pot`).

```
pot_value = pot.read()
```

Controlling duty cycle

To control the duty cycle, use the `duty()` method on the PWM object (`led_pwm`). The `duty()` method accepts a value between 0 and 1023 (in which 0 corresponds

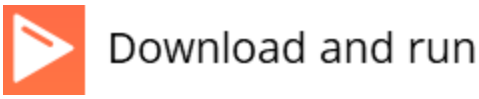
`pot_value` (that varies between 0 and 1023). This way you change the duty cycle by rotating the potentiometer.

```
led_pwm.duty(pot_value)
```

Testing the Code

Upload the *main.py* file to your ESP32 or ESP8266. For that, open uPyCraft IDE and copy the code provided to the *main.py* file. Go to **Tools** > **Serial** and select the serial port. Select your board in **Tools** > **Board**.

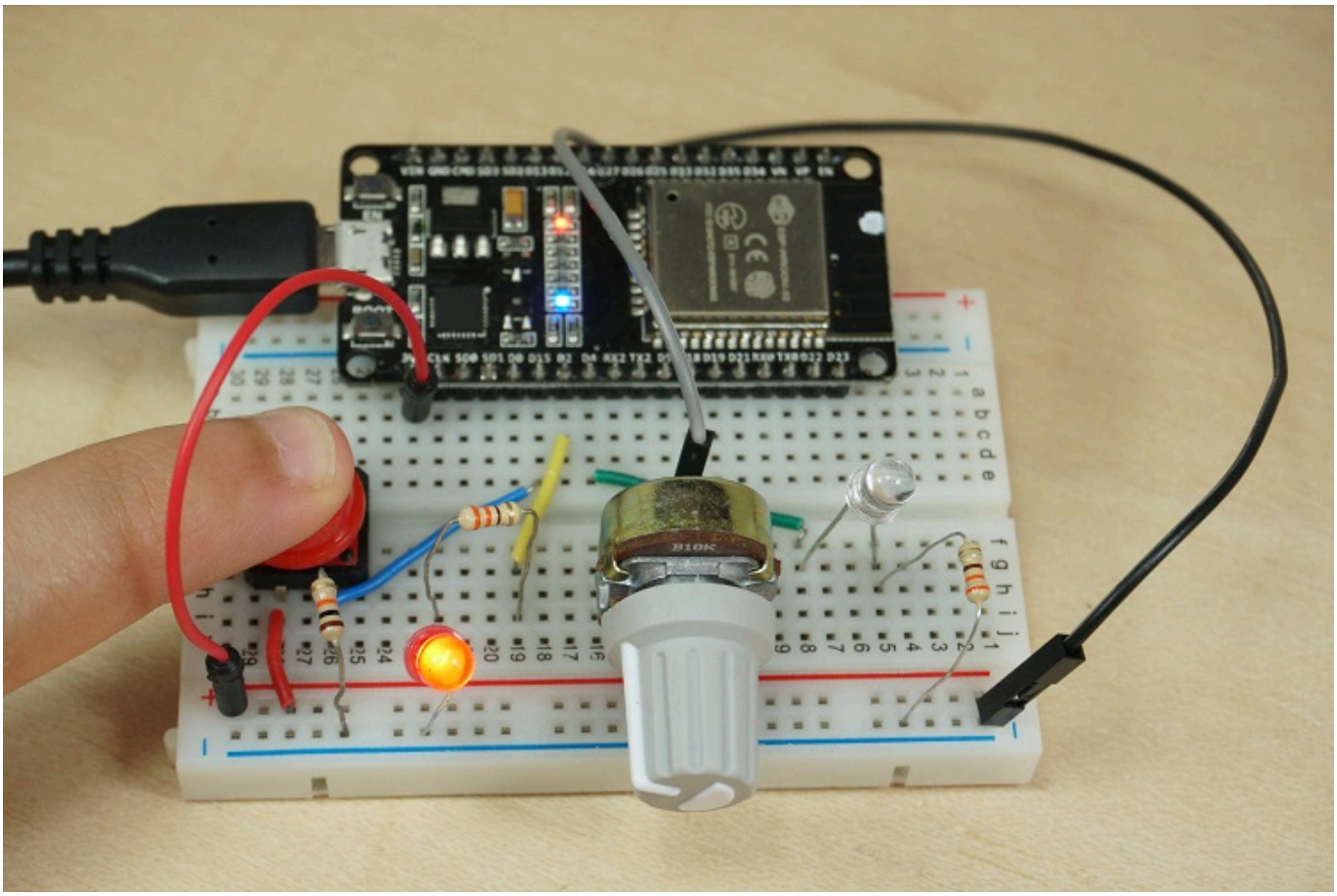
Then, upload the code to the ESP32 or ESP8266 by pressing the **Download and Run** button.



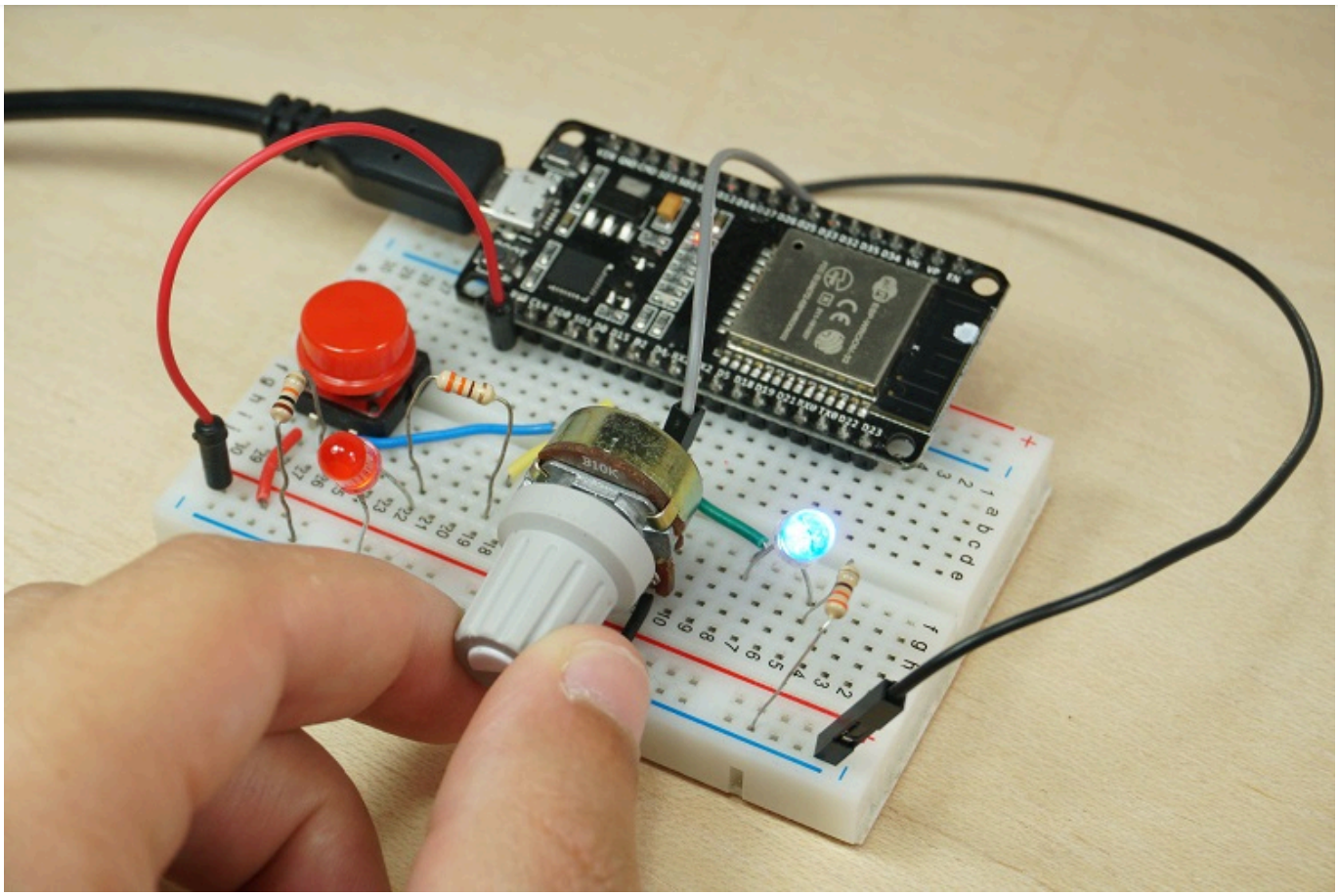
Note: to get you familiar with uPyCraft IDE youn can read the following tutorial – Getting Started with MicroPython on ESP32 and ESP8266

After uploading the code, press the ESP32/ESP8266 on-board EN/RST button to run the new script.

Now, test your setup. The LED should light up when you press the pushbutton.



The LED brightness changes when you rotate the potentiometer.



Wrapping Up

This simple example showed you how to read digital and analog inputs, control digital outputs and generate PWM signals with the ESP32 and ESP8266 boards using MicroPython.

If you like MicroPython, you may like the following projects:

- ESP32/ESP8266 MicroPython Interrrupts
- MicroPython: WS2812B Addressable RGB LEDs with ESP32/ESP8266
- Low Power Weather Station Datalogger using ESP8266 and BME280 with MicroPython
- ESP32/ESP8266 MicroPython Web Server

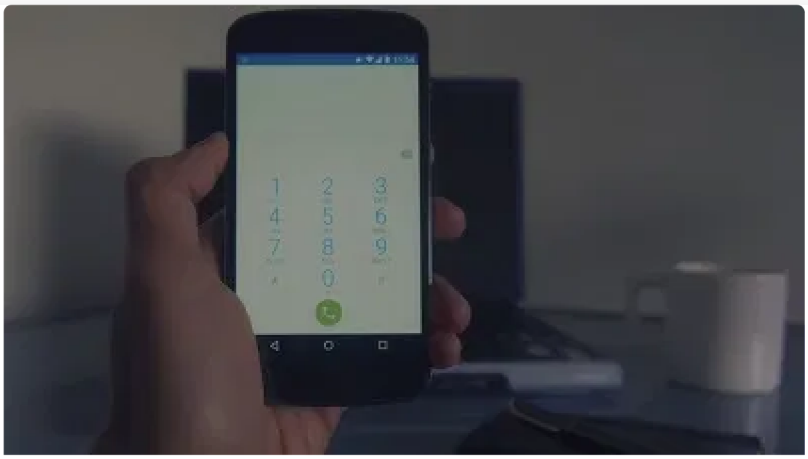
We hope you've found this article about how to control ESP32 and ESP8266 GPIOs with MicroPython useful. If you want to learn more about MicroPython, make sure you take a look at our eBook: **MicroPython Programming with ESP32 and**

Thanks for reading.

SMART HOME with Raspberry Pi, ESP32, ESP8266 [eBook]

Learn how to build a home automation system and we'll cover the following main subjects: Node-RED, Node-RED Dashboard, Raspberry Pi, ESP32, ESP8266, MQTT, and InfluxDB database **DOWNLOAD »**

Recommended Resources



What to Read Next…

ESP32 External Wake Up from Deep Sleep

ESP32: Upload Files to LittleFS using Arduino IDE

ESP32: ESP-NOW Encrypted Messages

14 thoughts on “MicroPython with ESP32 and ESP8266: Interacting with GPIOs”

Denis Brion

October 24, 2018 at 5:45 pm

Excuse me, but are

sure for the 2 others -else docs would be massively wrong – But it may need time to interface with uPython

What is the difference w/r resource greediness between uPython and Arduino (RAM/stack/Flash/other -I am very naive and ignorant- needs).

Reply

Dave W

October 24, 2018 at 7:04 pm

Denis,

at a REPL Prompt (>>>) type:
import machine
help(machine)

You'll get a long list of functions (and other stuff) which should include:

UART —
SPI —
I2C —
PWM —
ADC —
DAC —
SD —
Timer —
RTC —

All those classes handle the different protocols.

Also, http://docs.micropython.org/en/latest/esp8266/quickref.html is the

Dave

Reply

Denis Brion

October 25, 2018 at 10:53 am

Thanks dave, for linking to the official site (is interesting) and for explaining how to use python help (I very seldom use python: always forget how to use its help) and that micropython help is the same (I am still wondering whether I shall use uPython or C++ : C++ eat less resources, is almost as concise, but does not have comfortable help -Arduino classes have, however-

Reply

Mauro

October 24, 2018 at 10:28 pm

Yes, DHT, TWI and SPI protocols are supported.
I wrote several articles using MicroPython here: lemariva.com/micropython .
A post with sensors (some using with SPI/I2C) is here: lemariva.com/blog/2018/06/tutorial-getting-started-with-micropython-sensors available.

I prefer to use Atom and the PyMakr plugin to load the code and debug it: lemariva.com/blog/2017/10/micropython-getting-started

MicroPython uses several resources.

* The ESP8266 with only 160kB RAM is very limited for MicroPython, you get usually a memory allocation error, when you import 3/4 libraries. You can compile your files using mpy-cross (read the links above) to reduce the RAM usage.

* On the ESP32, it works really good, especially with the ESP32-wrover (4MB RAM or 8MB RAM version B).

* With the ESP32-WROOM (512Kb RAM) it works also good. Sometimes you need to compile your files too.

You can play with some code available in my GitHub: github.com/lemariva. If you have further questions, contact me.

Reply

Sara Santos

October 25, 2018 at 8:39 am

Hi Mauro.
Thank for sharing your projects about MicroPython.
Regards,
Sara 🙂

Reply

Mauro

October 26, 2018 at 9:59 am

Hi Sara.

Thanks You!, for publishing about MicroPython. It's a nice language.

Btw, really nice blog! My blog is really small in comparison with yours.

Let me know if I can help you with some tutorials.

Regards,

Mauro

Reply

Sara Santos

October 29, 2018 at 10:04 am

I see you have a lot of tutorials using ESP32 and micropython too.
I definitely have to take a closer look at your projects.
I'll tell Rui to take a look at your blog too.
Regards,
Sara 🙂

Denis Brion

October 25, 2018 at 9:13 am

Well, thanks a lot, Mauro, for useful information w/r resource greediness. I

volts peripherals…. – ; use wifi, I am not accustomed to) with ESP8266. There remain a ESP32 resource I do not understand : ESP32 is a dual core, and one can manage (Arduino + freeRTOS) to use both cores (this may be interesting: say, transmitting data, at an unpredictable rate, and samplig other data at regular rate seems vey easy with dual cores (no interrupts). Does uPython manage parallelism?
Remains another detail: I noticed esptool used a 4 Mbauds , and nanoPis/RPis use at most 115200 bauds. I bet (worked with my ESP8266; in the general case, do not not know) having a lower than default/recommanded upload speed does not harm.

Reply

Mauro
October 26, 2018 at 9:46 am

Yes, parallel processing is possible using the _thread class. It is only available on the ESP32, and WiPy2.0/3.0 (not on the ESP8266). It looks like this:

```
import _thread
import time

def ThreadWorker():
while True:
print(“Hello from thread worker”)
time.sleep(2)

_thread.start_new_thread(ThreadWorker, ())
```

This project includes multitasking: Captive Portal

problem.

Reply

Dave W

October 24, 2018 at 7:15 pm

Rui,

thanks for the Tutorial.
I notice you don't worry about debouncing the button, which is OK when you are just lighting an LED (and the eye can't see the quick ON/OFF sequences). I'm trying to send button presses to a WWW site and get multiple operations every time I press a button.
I'm trying a timer interrupt and it quickly gets messy (setting up the timer alarm, the function to handle the buttons and then the actual main code). Do you know of a function I can use instead?

Dave

Reply

Sara Santos

October 29, 2018 at 9:37 am

Hi Dave.
At the moment, I don't have a code for what you are looking for.
But there is an example here:

Regards,
Sara 🙂

Reply

martin
February 3, 2019 at 2:55 pm

many many thanks fpr the great tutorial – would love to see a port to mycropython . d o you thnk taht this is possible
love to hear from you
greetings

Reply

Sara Santos
February 5, 2019 at 9:35 am

Hi Martin.
What do you mean by “port to micropython”?
Regards,
Sara

Reply

babu

Great tutorials. I am a new bee and I have hard time building multiple input and outputs. would you mind if you could demonstrate an esp32 with micropython. that has 4 in put and 4 outputs. it could be 4 LED with individually 4 switch. I mean one switch for one LED.

Reply

Leave a Comment

Name *

Email *

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Post Comment

ESP32, and ESP8266 » learn how to build a complete home automation system.

Learn Raspberry Pi Pico/Pico W with MicroPython » The complete getting started guide to get the most out of the the Raspberry Pi Pico/Pico W (RP2040) microcontroller board using MicroPython programming language.

🔥 **Learn LVGL: Build GUIs for ESP32 Projects »** Learn how to build Graphical

using LVGL (Light Versatile Graphics Library) with the Arduino IDE.