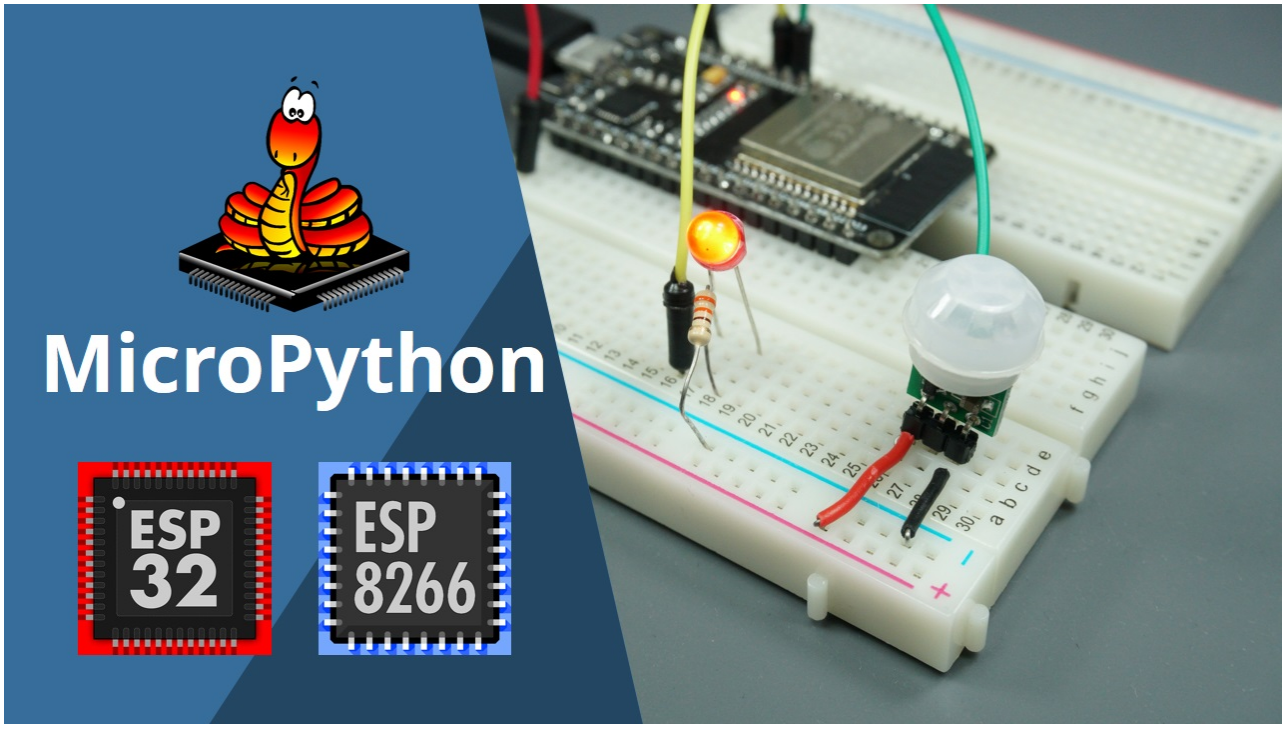
MicroPython: Interrupts with ESP32 and ESP8266

Learn how to configure and handle interrupts using MicroPython firmware with ESP32 and ESP8266 boards. You'll also build a project example with a PIR Motion Sensor.



Prerequisites

To follow this tutorial you need MicroPython firmware flashed in your ESP32 or ESP8266. You also need an IDE to write and upload the code to your board. We suggest using Thonny IDE or uPyCraft IDE:

- Thonny IDE:
  - Installing and getting started with Thonny IDE
  - Flashing MicroPython Firmware with esptool.py
- uPyCraft IDE:
  - Install uPyCraft IDE (Windows, Mac OS X, Linux)
  - Flash/Upload MicroPython Firmware to ESP32 and ESP8266

Introducing Interrupts

Interrupts are useful for making things happen automatically in microcontroller programs and can help solve timing problems. With interrupts you don't need to constantly check the current pin value. When a change is detected, an event is triggered (a function is called).

When an interrupt happens, the processor stops the execution of the main program to execute a task, and then gets back to the main program as shown in the figure below.

This is especially useful to trigger an action whenever motion is detected or whenever a pushbutton is pressed without the need for constantly checking its state.

> **ESP32 interrupt pins:** you can use all GPIOs as interrupts, except GPIO 6 to GPIO 11.

> **ESP8266 interrupt pins:** you can use all GPIOs, except GPIO 16.

Set Up an Interrupt in MicroPython

To setup an interrupt in MicroPython, you need to follow the next steps:

1. Define an interrupt handling function. The interrupt handling function should be as simple as possible, so the processor gets back to the execution of the main program quickly. The best approach is to signal the main code that the interrupt has happened by using a global variable, for example. The interrupt handling function should accept a parameter of type `Pin`. This parameter is

returned to the callback function and it refers to the GPIO that caused the interrupt.

```
def handle_interrupt(pin):
```

2. Setup the GPIO that will act as an interrupt pin as an input. For example:

```
pir = Pin(14, Pin.IN)
```

3. Attach an interrupt to that pin by calling the `irq()` method:

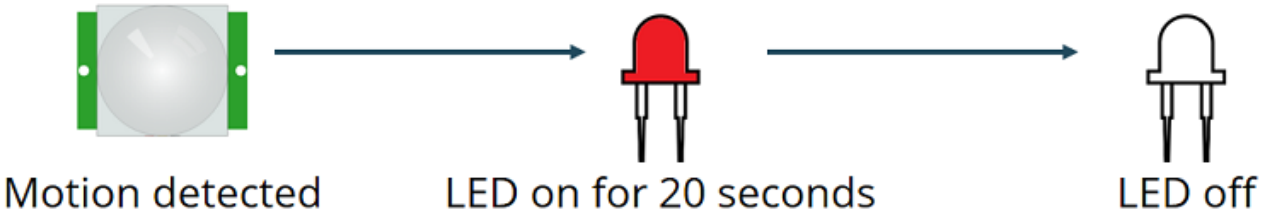
```
pir.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)
```

The `irq()` method accepts the following arguments:

- **trigger:** this defines the trigger mode. There are 3 different conditions:
  - **`Pin.IRQ_FALLING`:** to trigger the interrupt whenever the pin goes from HIGH to LOW;
  - **`Pin.IRQ_RISING`:** to trigger the interrupt whenever the pin goes from LOW to HIGH.
  - **`3`:** to trigger the interrupt in both edges (this means, when any change is detected)
- **handler:** this is a function that will be called when an interrupt is detected, in this case the `handle_interrupt()` function.

Project Example with PIR Motion Sensor

To demonstrate how to handle interrupts, we'll build a simple project with a PIR motion sensor. Whenever motion is detected we'll light up an LED for 20 seconds.



Parts required

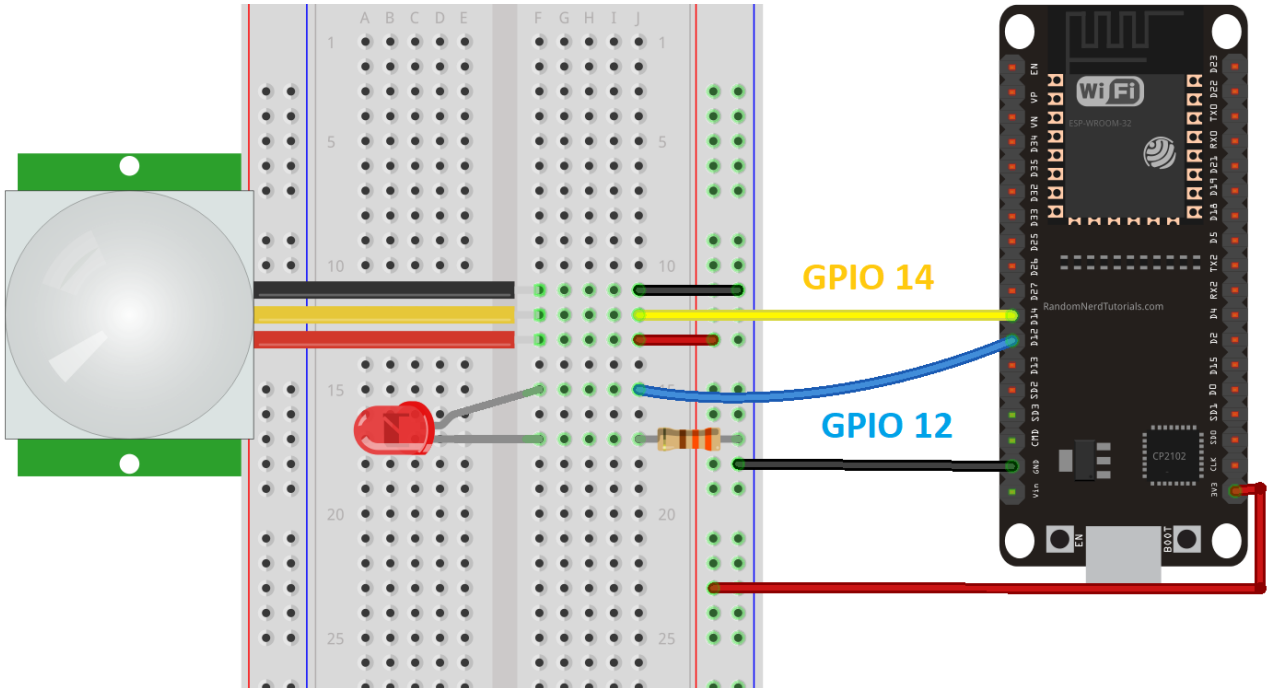
Here's a list of the parts you need to build the circuit:

- ESP32 (read Best ESP32 development boards) or ESP8266 (read Best ESP8266 development boards)
- 5mm LED
- 330 Ohm resistor
- Mini PIR motion sensor (AM312) or PIR motion sensor (HC-SR501)
- 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!

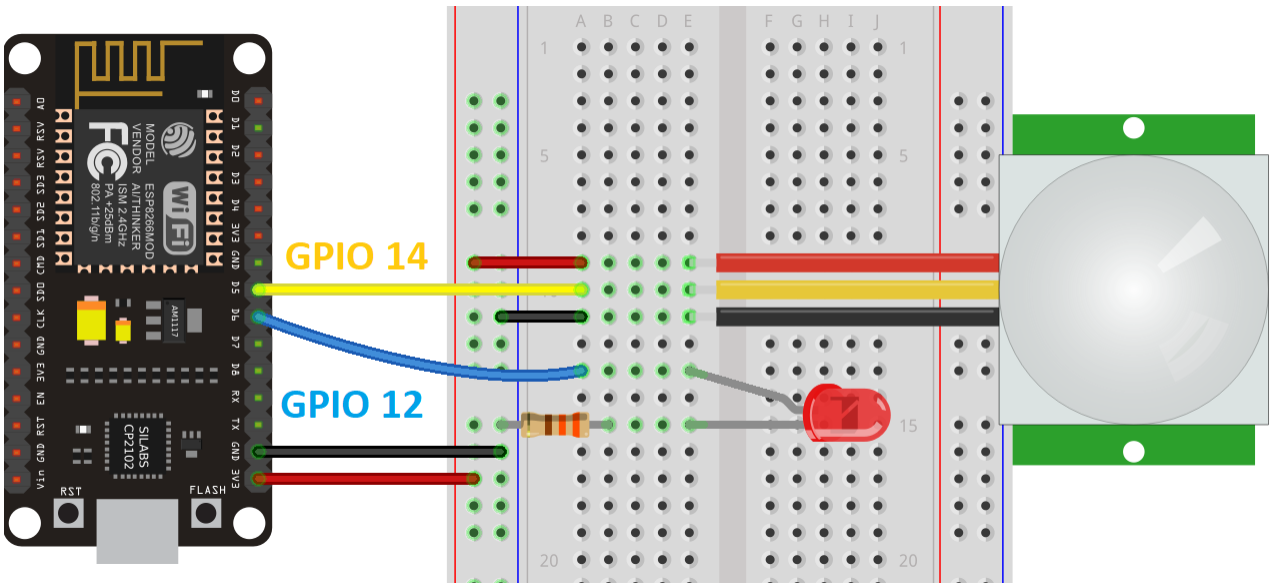
Schematic – ESP32

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



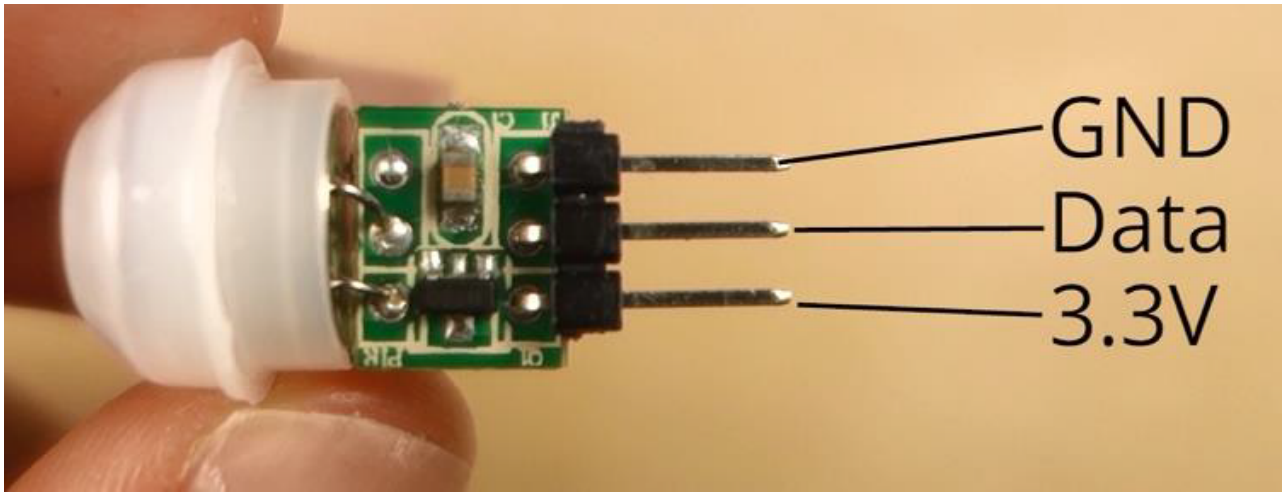
Schematic – ESP8266

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



Important: the Mini AM312 PIR Motion Sensor we're using in this project operates at 3.3V. However, if you're using another PIR motion sensor like the HC-SR501, it operates at 5V. You can either modify it to operate at 3.3V or simply power it using the Vin pin.

In the figure below, we provide the pinout for the Mini AM312 PIR motion sensor. If you're using another motion sensor, please check its pinout before assembling the circuit.



Code

Here's the script that detects motion and lights up an LED whenever motion is detected. This code is compatible with both the ESP32 and ESP8266.

```
# Complete project details at https://RandomNerdTutorials.com

from machine import Pin
from time import sleep

motion = False

def handle_interrupt(pin):
  global motion
  motion = True
  global interrupt_pin
  interrupt_pin = pin

led = Pin(12, Pin.OUT)
pir = Pin(14, Pin.IN)

pir.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)

while True:
  if motion:
    print('Motion detected! Interrupt caused by:', interrupt_pin)
    led.value(1)
```

```
    sleep(20)
    led.value(0)
    print('Motion stopped!')
    motion = False
```

View raw code

How the code Works

To use interrupts, import the `Pin` class from the `machine` module. We also import the `sleep` method from the `time` module to add a delay in our script.

```
from machine import Pin
from time import sleep
```

Create a variable called `motion` that can be either True of False. This variable will indicate whether motion was detected or not (this is the global variable that will be changed on the interrupt handling function).

```
motion = False
```

Then, create a function called `handle_interrupt` .

```
def handle_interrupt(pin):
  global motion
  motion = True
  global interrupt_pin
  interrupt_pin = pin
```

This function will be called every time motion is detected. The `handle_interrupt` function has an input parameter (`pin`) in which an object of class `Pin` will be passed when the interrupt happens (it indicates which pin caused the interrupt).

Here we're saving the pin that caused the interrupt in the `interrupt_pin` variable. In this case, it is not very useful because we only have one interrupt pin. However, this can be useful if we have several interrupts that trigger the same interrupt handling function and we want to know which GPIO caused the interrupt.

In our example, the `handle_interrupt` function simply changes the `motion` variable to `True` and saves the interrupt pin. You should keep your handling interrupt functions as short as possible and avoid using the `print()` function inside. Then, the main code should have all the things we want to happen when the interrupt happens.

Note: as you want `motion` to be usable both inside the function and throughout the code, it needs to be declared as global. Otherwise, when motion is detected nothing would happen, because the `motion` variable would be changing inside the function and not in the main body of the code.

Proceeding with the code, we need to create two Pin objects. One for the LED on GPIO 12 , and another for the PIR motion sensor on GPIO 14 .

```
led = Pin(12, Pin.OUT)
pir = Pin(14, Pin.IN)
```

Then, set an interrupt on the `pir` by calling the `irq()` method.

```
pir.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)
```

In the `loop()` , when the `motion` variable is True, we turn the LED on for 20 seconds and print a message that indicates that motion was detected and which pin caused the interrupt.

```
if motion:
  print('Motion detected! Interrupt caused by:', interrupt_pin)
```

```
    led.value(1)
    sleep(20)
```

After 20 seconds, turn the LED off, and print a message to indicate that motion stopped.

```
  led.value(0)
  print('Motion stopped!')
```

Finally, set the `motion` variable to False:

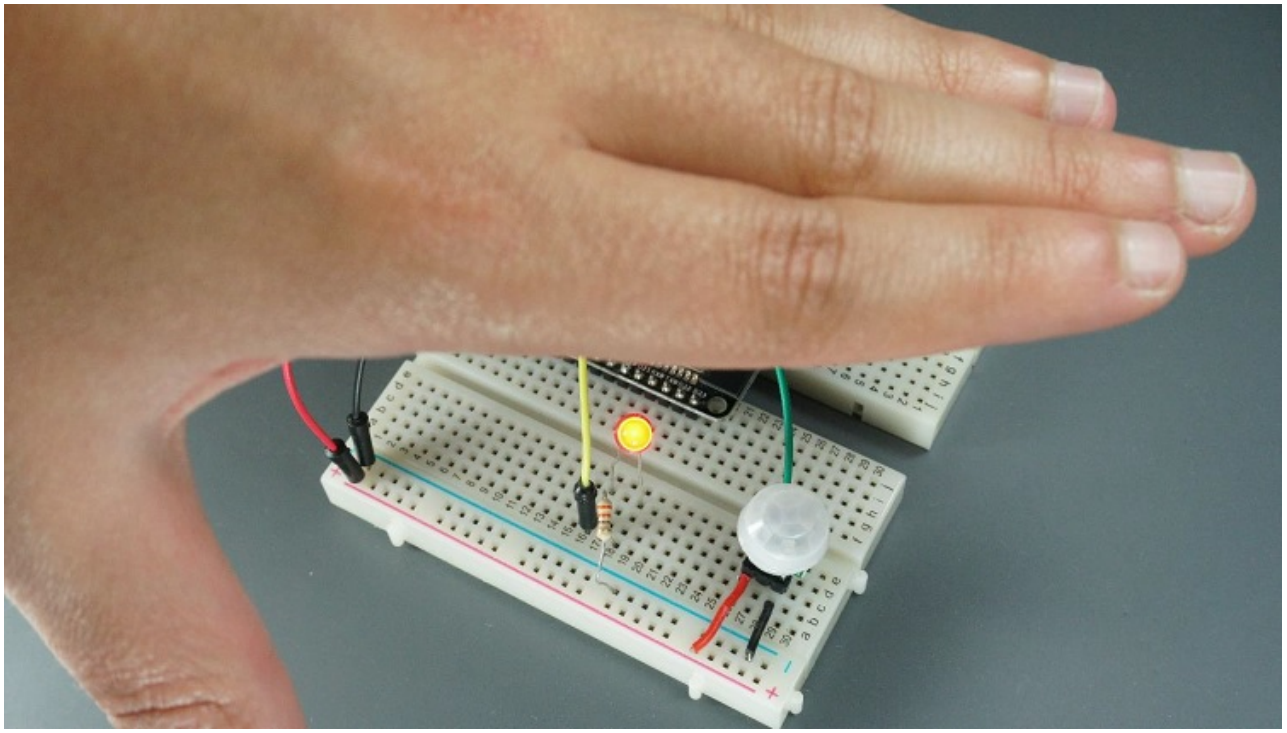
```
  motion = False
```

The `motion` variable can only become True again, if motion is detected and the `handle_interrupt` function is called.

For simplicity, in this example we use a delay to keep the LED on for 20 seconds. Ideally, you should use timers.

Demonstration

Upload the code to your ESP32/ESP8266 board. The LED should turn on for 20 seconds when motion is detected, and a message should be printed in the Shell.



After 20 seconds the LED turns off.

Note: the AM312 PIR motion sensor has a default delay time of 8 seconds. This means that it won't be triggered before 8 seconds have passed since the last trigger.

Wrapping Up

We hope you've found this article interesting. We've learned how to:

- setup a pin as an interrupt
- handle that interrupt in your code
- detect which GPIO pin caused the interrupt

In our example, we've used a PIR motion sensor to trigger the interrupt. But the example presented can also be used to detect a button press, for example.

If you like programming the ESP32 and ESP8266 boards with MicroPython, and you want to learn more, please take a look at the following resources:

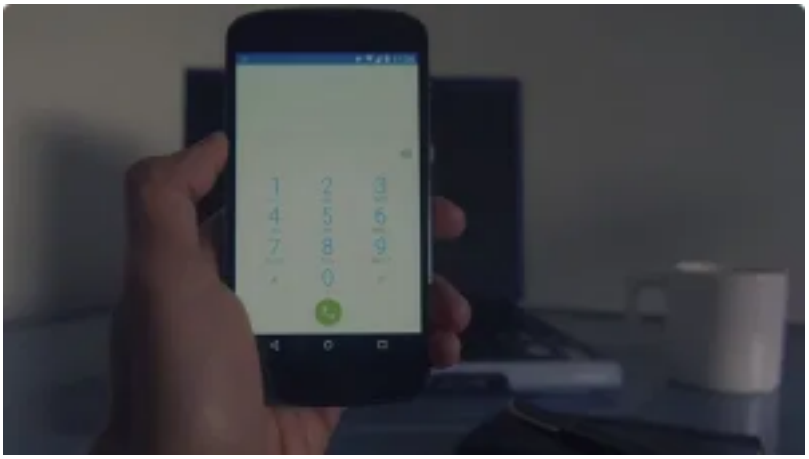
- **[eBook] MicroPython Programming with ESP32 and ESP8266**
- MicroPython: WS2812B Addressable RGB LEDs with ESP32 and ESP8266

- Low Power Weather Station Datalogger using ESP8266 and BME280 with MicroPython
- MicroPython – Getting Started with MQTT on ESP32/ESP8266

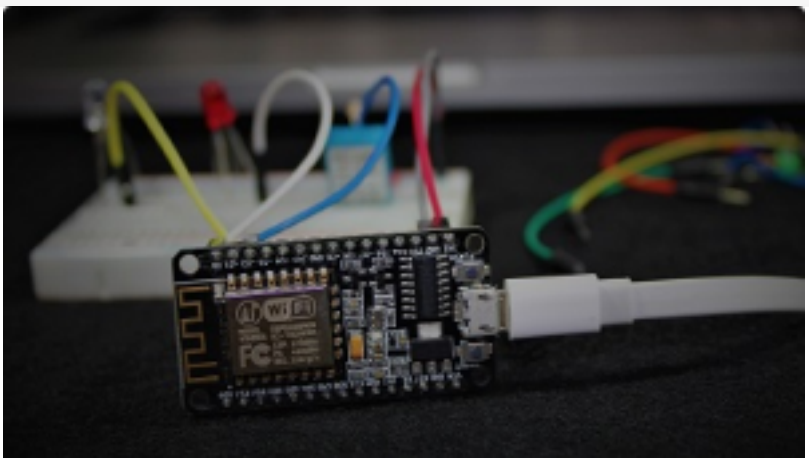
[eBook] Build Web Servers with ESP32 and ESP8266 (2nd Edition)

Build Web Server projects with the ESP32 and ESP8266 boards to control outputs and monitor sensors remotely. Learn HTML, CSS, JavaScript and client-server communication protocols **DOWNLOAD »**

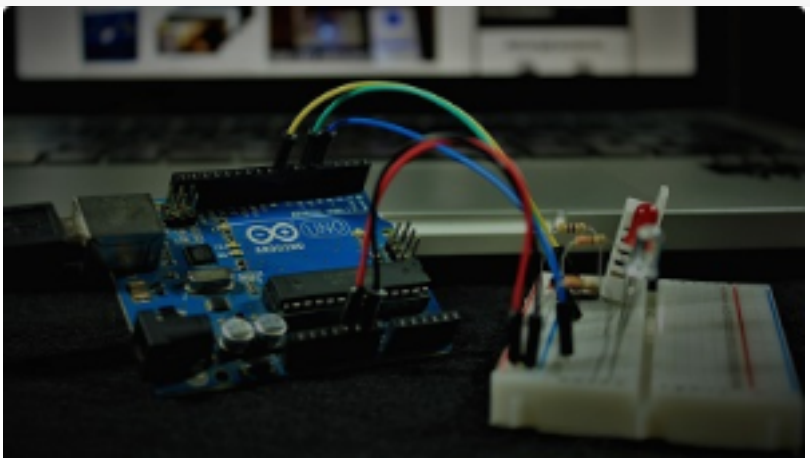
Recommended Resources



Build a Home Automation System from Scratch » With Raspberry Pi, ESP8266, Arduino, and Node-RED.



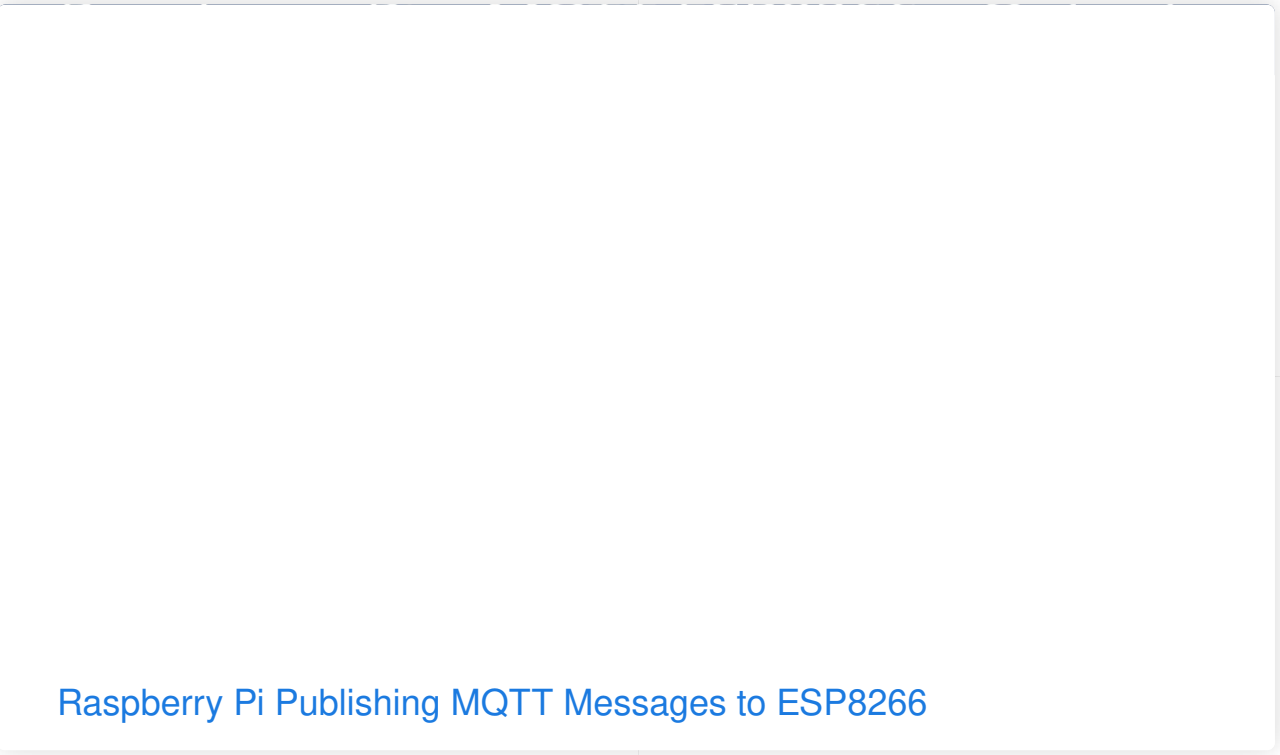
Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Arduino Step-by-Step Projects » Build 25 Arduino projects with our course, even with no prior experience!

What to Read Next…

Low Power Weather Station Datalogger using ESP8266 and BME280 with MicroPython



Raspberry Pi Publishing MQTT Messages to ESP8266

8 thoughts on “MicroPython: Interrupts with ESP32 and ESP8266”

Seth C Stenzel

March 11, 2019 at 7:50 pm

Would it be possible to wake the esp32 from deepsleep with these interrupts?

Reply

Sara Santos

March 12, 2019 at 4:05 pm

Hi Seth.

Yes, I think you can, but I haven't tried it yet.
It seems that you can pass a wake parameter when defining the interrupt and it can be:
– machine.IDLE
– machine.SLEEP
– machine.DEEPSLEEP

See more information here:
docs.micropython.org/en/v1.8.7/esp8266/library/machine.Pin.html?highlight=pin#machine.Pin.irq

Regards,
Sara

Reply

Mark
December 11, 2019 at 11:55 am

On a somewhat related topic, has anyone (else) had issues with debounce on the ESP32 with reed switches? I'm using an anemometer with a reed switch to fire an interrupt, and for some reason every time I get a falling edge interrupt (I've got the GPIO pin held high), it triggers between 1 and 4 interrupts. I'm using an adafruit HUZZAH32, and GPIO21.

Reply

Vardhan Batavia
November 26, 2020 at 6:49 am

you can add 200 ms of delay in ISR function before enabling using machine.enable_irq(state)
that will prevent interrupt to occur again and again.

Regards
Vardhan

Reply

Brian Hambleton
January 2, 2020 at 4:56 pm

Does Micro-Python support interrupt on ‘both’ rising and falling edge? I am sensing a damper in my HVAC system to monitor blower on off. Rising edge = timestamp start of blower cycle, falling edge = timestamp end of blower cycle. Can I do this with one input or do I need to use two inputs one for rising one for falling?

Reply

Sara Santos
January 3, 2020 at 11:47 am

Hi Brian.
There are 3 different conditions:
Pin.IRQ_FALLING: to trigger the interrupt whenever the pin goes from HIGH to LOW;
Pin.IRQ_RISING: to trigger the interrupt whenever the pin goes from LOW to HIGH.
3: to trigger the interrupt in both edges (this means, when any change

is detected)

So, yes. It supports both at the same time.
Regards,
Sara

Reply

Vardhan
November 26, 2020 at 6:48 am

you can add 200 ms of delay in ISR function before enabling using machine.enable_irq(state)
that will prevent interrupt to occur again and again.

Reply

Juan
March 13, 2021 at 8:44 pm

Hi people. Can NodeMcu 32 work with UART interrupts ??? May I use UART.IRQ (…)???
I can’t find how to do that..
Thanks .. !

Reply

Leave a Comment

Name *

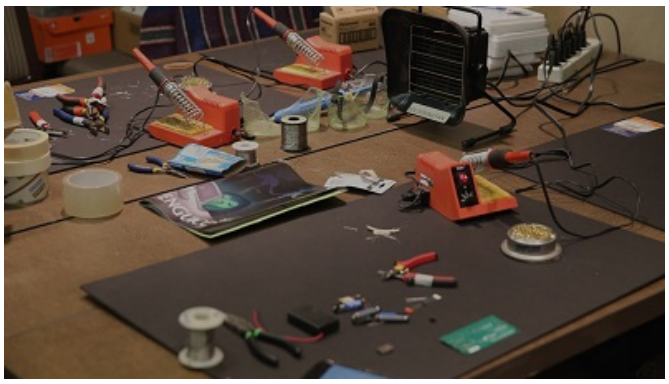
Email *

Website

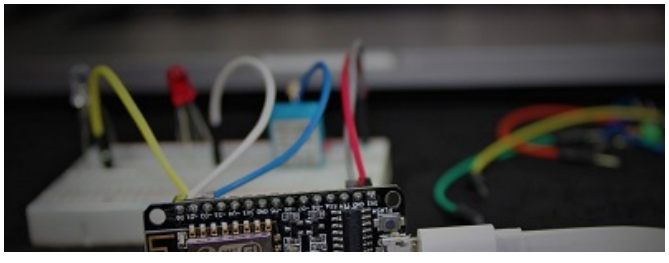
Notify me of follow-up comments by email.

Notify me of new posts by email.

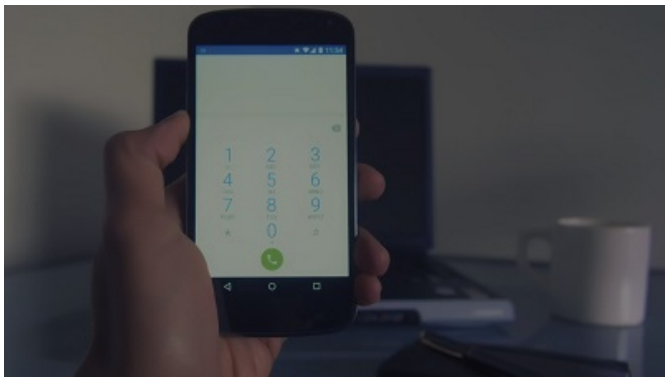
Post Comment



Visit Maker Advisor – Tools and Gear for makers, hobbyists and DIYers »



Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Build Web Servers with ESP32 and ESP8266 » boards to control outputs and monitor sensors remotely.