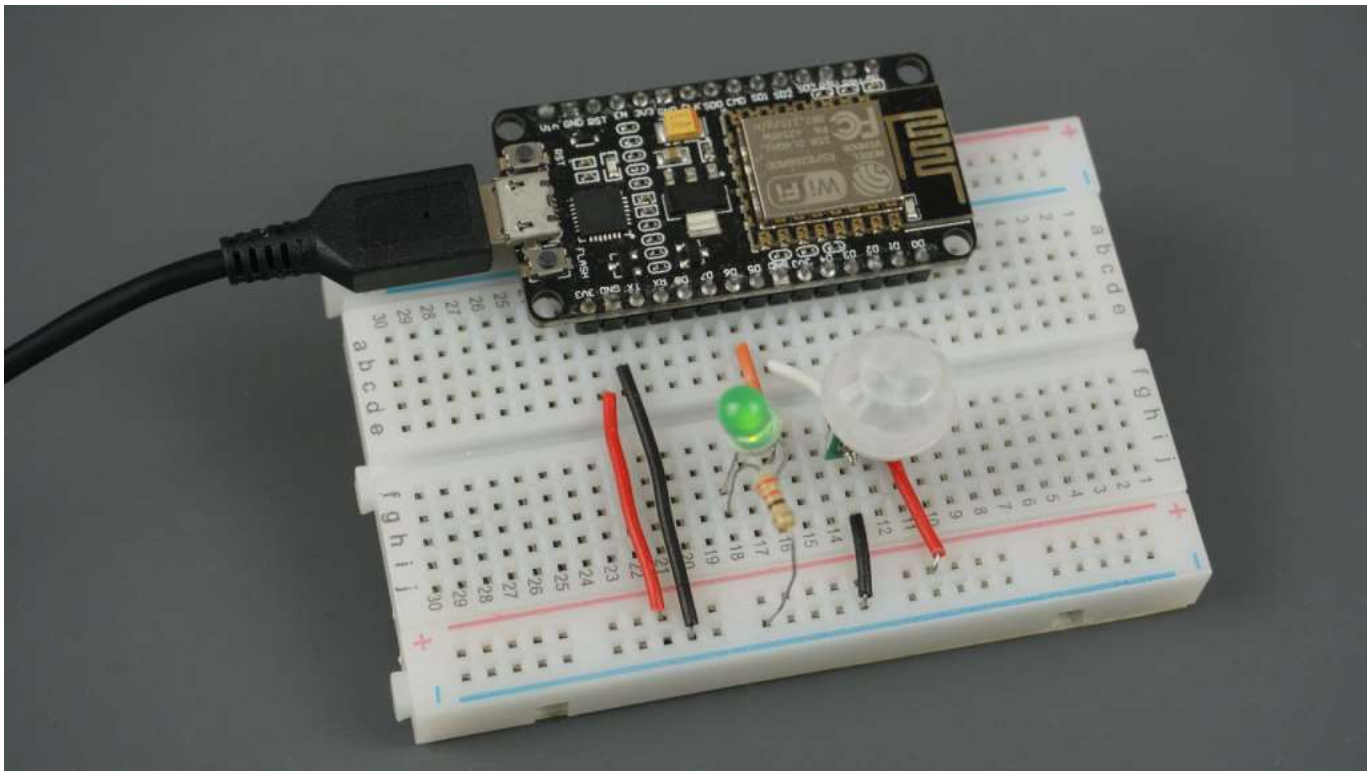
ESP8266 Interrupts and Timers using Arduino IDE (NodeMCU)

In this guide, you'll learn how to use interrupts and timers with the ESP8266 NodeMCU using Arduino IDE. Interrupts allow you to detect changes in the GPIO state without the need to constantly check its current value. With interrupts, when a change is detected, an event is triggered (a function is called).



As an example, we'll detect motion using a PIR motion sensor: when motion is detected, the ESP8266 starts a timer and turns an LED on for a predefined number of seconds. When the timer finishes counting down, the LED automatically turns off.

To create an interrupt, call `attachInterrupt()` and pass as arguments the GPIO interrupt pin, the ISR (funcion to be called) and mode. The ISR function must have the `ICACHE_RAM_ATTR` attribute declared. The mode can be `CHANGE`, `RISING` or `FALLING`.

```
attachInterrupt(digitalPinToInterrupt(GPIO), ISR, mode);
```

Before proceeding with this tutorial you should have the ESP8266 add-on installed in your Arduino IDE. Follow this tutorial to Install ESP8266 in Arduino IDE, if you haven't already.

Introducing ESP8266 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. This function is called interrupt service routine (ISR).

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 to constantly check its state.

attachInterrupt() Function

To set an interrupt in the Arduino IDE, you use the `attachInterrupt()` function, that accepts as arguments: the GPIO interrupt pin, the name of the function to be executed, and mode:

```
attachInterrupt(digitalPinToInterrupt(GPIO), ISR, mode);
```

GPIO interrupt pin

The first argument is a GPIO interrupt. You should use `digitalPinToInterrupt(GPIO)` to set the actual GPIO as an interrupt pin. For example, if you want to use GPIO 14 as an interrupt, use:

```
digitalPinToInterrupt(14)
```

The ESP8266 supports interrupts in any GPIO, except GPIO16.

ISR

The second argument of the `attachInterrupt()` function is the name of the function that will be called every time the interrupt is triggered – the interrupt service routine (ISR).

The ISR 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 and within the `loop()` check and clear that flag, and execute code.

ISRs need to have `ICACHE_RAM_ATTR` before the function definition to run the interrupt code in RAM.

Interrupt modes

The third argument is the mode and there are 3 different modes:

- **CHANGE:** to trigger the interrupt whenever the pin changes value – for example from HIGH to LOW or LOW to HIGH;
- **FALLING:** for when the pin goes from HIGH to LOW;

- **RISING:** to trigger when the pin goes from LOW to HIGH.

For our example, will be using the RISING mode, because when the PIR motion sensor detects motion, the GPIO it is connected to goes from LOW to HIGH.

Introducing ESP8266 Timers

For this tutorial, we'll use timers. We want the LED to stay on for a predetermined number of seconds after motion is detected. Instead of using a `delay()` function that blocks your code and doesn't allow you to do anything else for a determined number of seconds, we'll use a timer.

delay() vs millis()

The `delay()` function accepts a single int number as an argument. This number represents the time in milliseconds the program has to wait until moving on to the next line of code.

```
delay(time in milliseconds);
```

When you call `delay(1000)` your program stops on that line for 1 second. `delay()` is a blocking function. Blocking functions prevent a program from doing anything else until that particular task is completed. If you need multiple tasks to occur at the same time, you cannot use `delay()`. For most projects you should avoid using delays and use timers instead.

Using a function called `millis()` you can return the number of milliseconds that have passed since the program first started.

```
millis();
```

Why is that function useful? Because by using some math, you can easily verify how much time has passed without blocking your code.

Blinking an LED using millis() (without delay)

If you're not familiar with `millis()` function, we recommend reading this section. If you're already familiar with timers, you can skip to the PIR motion sensor project.

The following snippet of code shows how you can use the `millis()` function to create a blink project. It turns an LED on for 1000 milliseconds, and then turns it off.

```
pinMode(ledPin, OUTPUT);
```

```
    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
  }
```

View raw code

How the code works

Let's take a closer look at this blink sketch that works without the `delay()` function (it uses the `millis()` function instead).

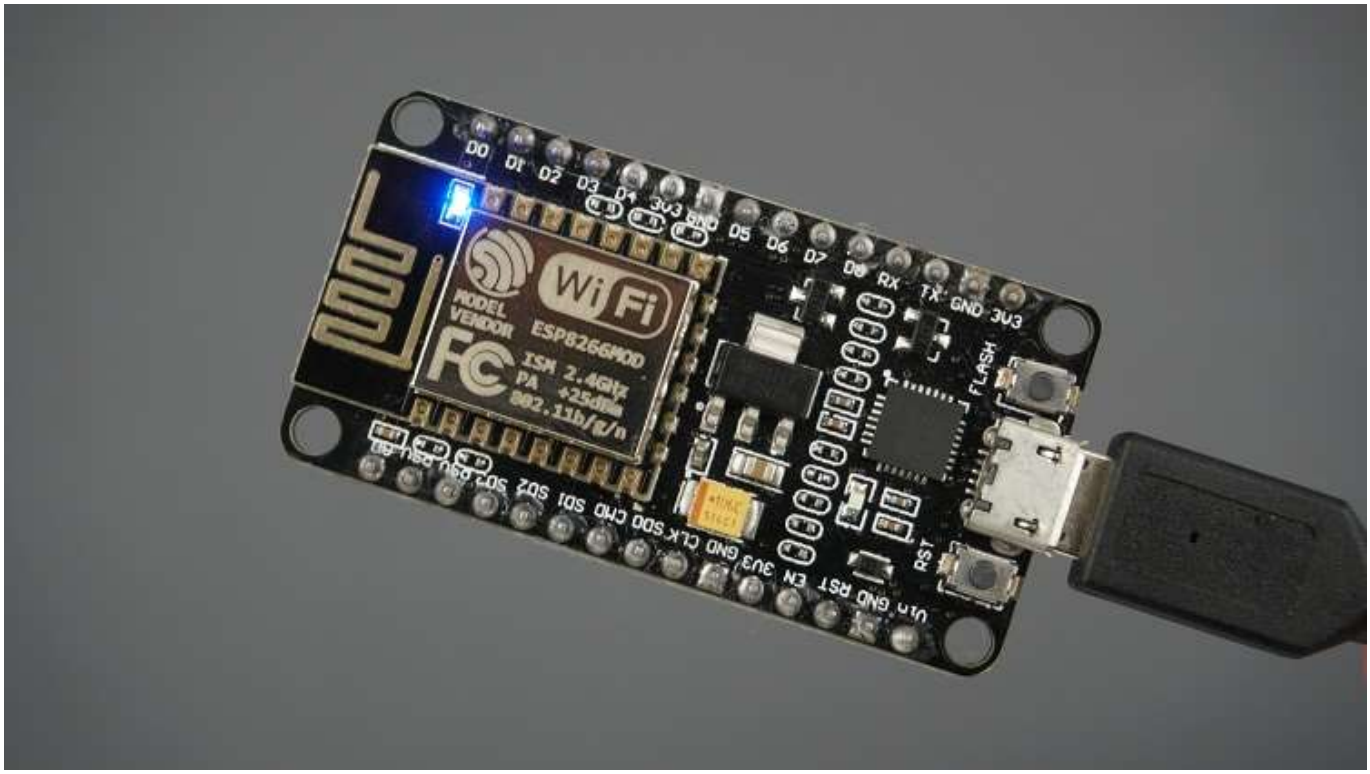
Basically, this code subtracts the previous recorded time (`previousMillis`) from the current time (`currentMillis`). If the remainder is greater than the interval (in this case, 1000 milliseconds), the program updates the `previousMillis` variable to the current time, and either turns the LED on or off.

```
if (currentMillis - previousMillis >= interval) {
  // save the last time you blinked the LED
  previousMillis = currentMillis;
  (...)
```

Because this snippet is non-blocking, any code that's located outside of that first if statement should work normally.

You should now be able to understand that you can add other tasks to your `loop()` function and your code will still be blinking the LED every one second.

You can upload this code to your ESP8266 to test it. The on-board LED should be blinking every second.



ESP8266 NodeMCU with PIR Motion Sensor

In this section, you'll learn how to detect motion with a PIR motion sensor using interrupts and timers in your code.

Parts Required

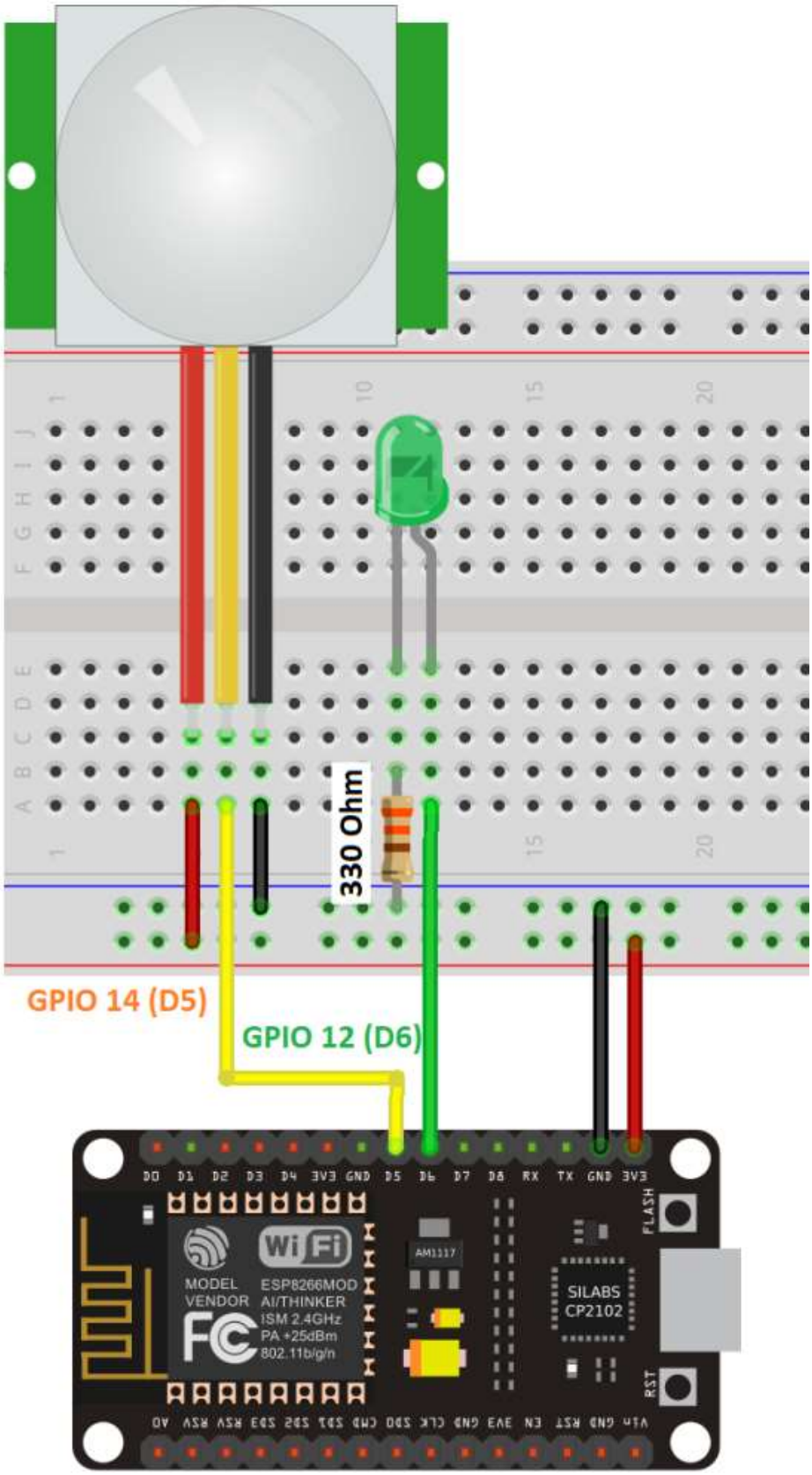
Here's a list of the parts required to complete this tutorial:

- ESP8266 (read Best ESP8266 development boards)
- PIR motion sensor (AM312)
- 5mm LED
- 330 Ohm resistor
- 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 Diagram

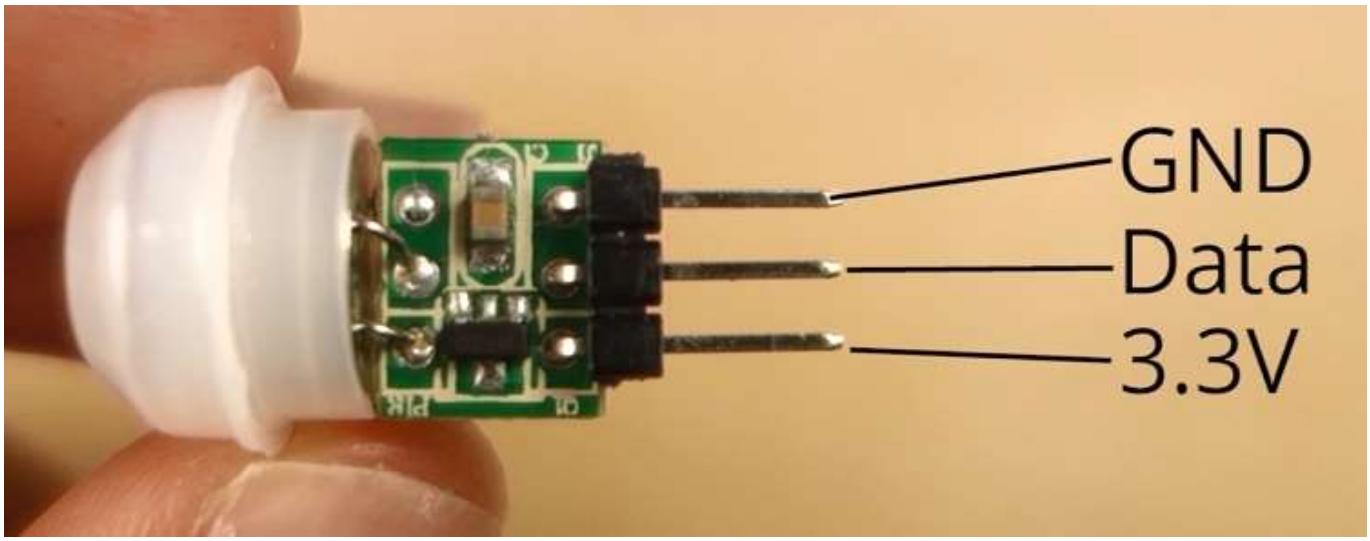
Assemble the PIR motion sensor and an LED to your ESP8266. We'll connect the LED to GPIO 12 (D6) and the PIR motion sensor data pin to GPIO 14 (D5).



Recommended reading: ESP8266 Pinout Reference Guide

Important: the Mini AM312 PIR Motion Sensor used 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.

The following figure shows the AM312 PIR motion sensor pinout.



Code

After wiring the circuit as shown in the schematic diagram, copy the code provided to your Arduino IDE.

You can upload the code as it is, or you can modify the number of seconds the LED is lit after detecting motion. Simply change the `timeSeconds` variable with the number of seconds you want.

```
  // Set LED to LOW
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
}

void loop() {
  // Current time
  now = millis();
  // Turn off the LED after the number of seconds defined in the
  if(startTimer && (now - lastTrigger > (timeSeconds*1000))) {
    Serial.println("Motion stopped...");
    digitalWrite(led, LOW);
    startTimer = false;
  }
```

View raw code

How the Code Works

Let's take a look at the code.

Start by assigning two GPIO pins to the `led` and `motionSensor` variables.

```
const int led = 12;
const int motionSensor = 14;
```

Then, create variables that will allow you set a timer to turn the LED off after motion is detected.

```
unsigned long now = millis();
unsigned long lastTrigger = 0;
boolean startTimer = false;
```

The `now` variable holds the current time. The `lastTrigger` variable holds the time when the PIR sensor detects motion. The `startTimer` is a boolean variable that starts the timer when motion is detected.

setup()

In the `setup()`, start by initializing the serial port at 115200 baud rate.

```
Serial.begin(115200);
```

Set the PIR Motion sensor as an `INPUT_PULLUP`.

```
pinMode(motionSensor, INPUT_PULLUP);
```

To set the PIR sensor pin as an interrupt, use the `attachInterrupt()` function as described earlier.

```
attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovemen
```

The pin that will detect motion is GPIO 14 and it will call the function `detectsMovement()` on `RISING` mode.

The LED is an `OUTPUT` whose state starts at `LOW`.

```
pinMode(led, OUTPUT);
digitalWrite(led, LOW);
```

loop()

The `loop()` function is constantly running over and over again. In every loop, the `now` variable is updated with the current time.

```
  now = millis();
```

Nothing else is done in the `loop()`. But, when motion is detected, the `detectsMovement()` function is called because we've set an interrupt previously in the `setup()`.

The `detectsMovement()` function prints a message in the Serial Monitor, turns the LED on, sets the `startTimer` boolean variable to true and updates the `lastTrigger` variable with the current time.

```
ICACHE_RAM_ATTR void detectsMovement() {
  Serial.println("MOTION DETECTED!!!");
  digitalWrite(led, HIGH);
  startTimer = true;
  lastTrigger = millis();
}
```

After this step, the code goes back to the `loop()`. This time, the `startTimer` variable is true. So, when the time defined in seconds has passed (since motion was detected), the following if statement will be true.

```
if(startTimer && (now - lastTrigger > (timeSeconds*1000))) {
  Serial.println("Motion stopped…");
  digitalWrite(led, LOW);
  startTimer = false;
}
```

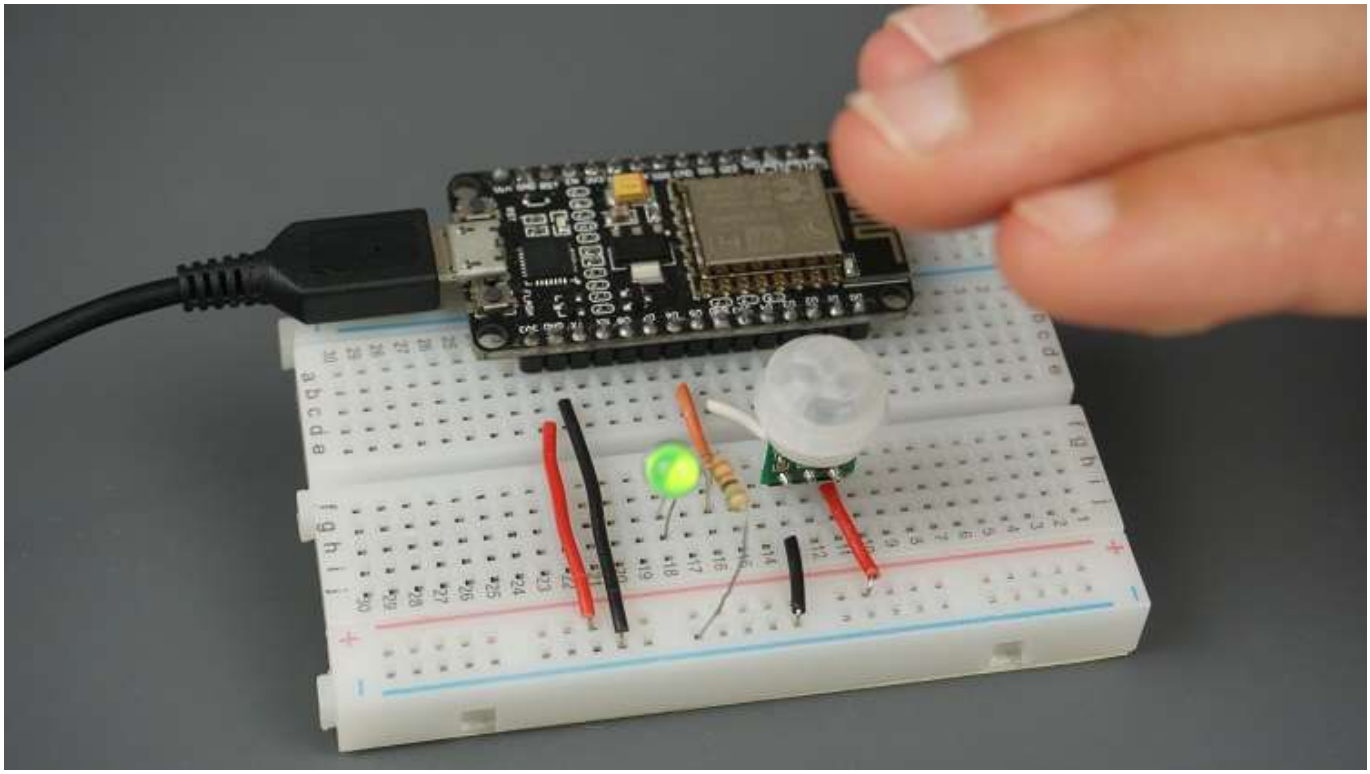
The "Motion stopped…" message will be printed in the Serial Monitor, the LED is turned off, and the `startTimer` variable is set to false.

Demonstration

Upload the code to your ESP8266. Make sure you have the right board and COM port selected.

Open the Serial Monitor at a baud rate of 115200.

Move your hand in front of the PIR sensor. The LED should turn on, and a message is printed in the Serial Monitor saying "**MOTION DETECTED!!!**". After 10 seconds the LED should turn off.



Wrapping Up

To sum up, interrupts are useful to detect a change in a GPIO state and instantly trigger a function. You've also learned that you should use timers to write non-blocking code.

We hope you've found this tutorial useful. We have other tutorials on how to handle interrupts using MicroPython and using ESP32:

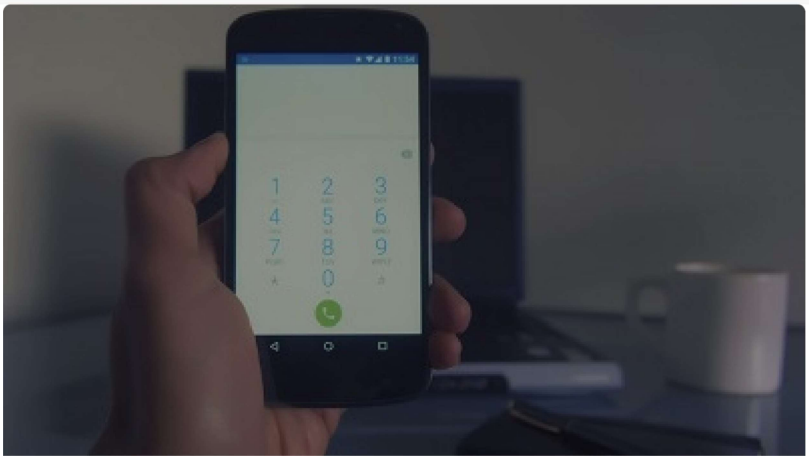
- MicroPython: Interrupts with ESP32 and ESP8266
- ESP32 with PIR Motion Sensor using Interrupts and Timers

Learn more about the ESP8266 board with our resources:

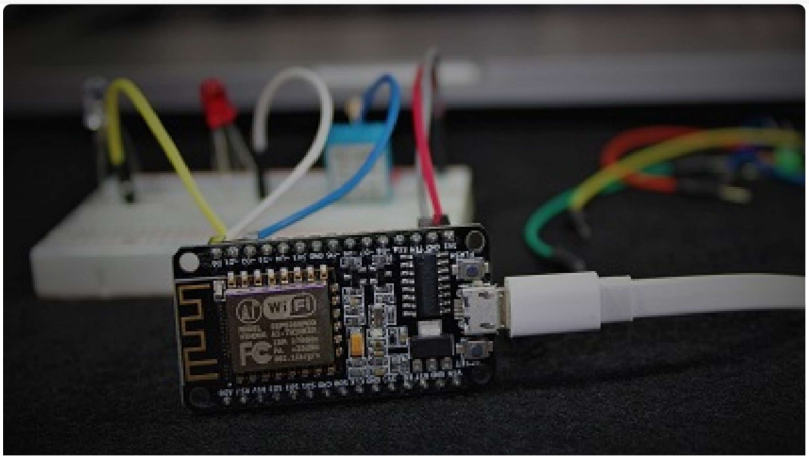
- Home Automation using EPS8266
- MicroPython Programming with ESP32 and ESP8266
- Free ESP8266 Projects and Tutorials

Thanks for reading.

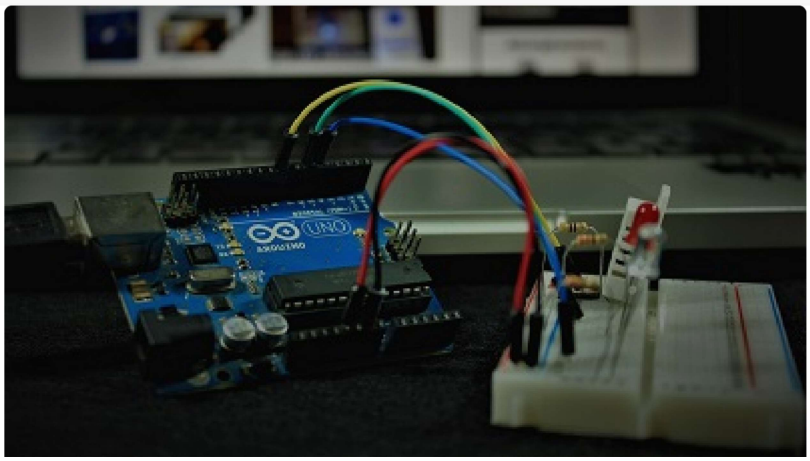
Recommended Resources



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.



What to Read Next…

ESP32: How to Log Data (9 Different Ways)

Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

32 thoughts on “ESP8266 Interrupts and Timers using Arduino IDE (NodeMCU)”

Oliver

August 6, 2019 at 4:36 pm

Hi!
A great tutorial again… as usual!

I have a question…
Can I define more than one interrupt (on different GPIOs) ?

If yes… how habe I to declare them?

Hope you can help me.

Greets fromme Germany, Oliver

Reply

Rui Santos
August 8, 2019 at 1:55 pm

Hello Oliver,
Yes, you should be able to do it just run declare another interrupt with a different GPIO number:
attachInterrupt(digitalPinToInterrupt(GPIO), ISR, mode);

Regards,
Rui

Reply

Laurent
August 17, 2019 at 11:46 am

Hi,
Great tutorial, thank you.
My question is regarding the blinking led.

Is it possible to set different intervals for the light on and the light off?
For example on for to 2 seconds every 10 seconds?

Reply

Rui Santos

September 13, 2019 at 9:25 am

Hello Laurent,
Yes, you need to create two different timers (one for 2 seconds and the other for 10 seconds)…

Reply

Mark from Hungary

October 19, 2019 at 5:49 pm

Dear Rui,

You are great!

I learnt almost everything about esp8266 from you, and now i created already several industrial data logging or controlling solutions to my company!

Thank you very much for your tutorials! Go ahead!

Best Regards!
Mark.

Reply

Sara Santos

October 20, 2019 at 10:35 am

Hi Mark.
That's great! We're glad to hear that.
Thank you for following our work.
Regards,
Sara

Reply

f f Poland

November 9, 2019 at 9:33 pm

great job!

Reply

Sara Santos

November 10, 2019 at 4:23 pm

Thanks 😀

Reply

Rajesh

December 19, 2019 at 4:04 pm

Problem with mills is that it gets reset over 50 days.
So need is to consider overflow of mills in case the device is to remain on more than 50 days.

Reply

Lennart

April 5, 2020 at 3:31 pm

That is no problem as long as you follow the code in the examples.
If you use unsigned long variables it is handled by the integer arithmetics.

```
now = millis();
if (now – lastTime >= DELAYTIME) {
do something;
}
```

Examples with 8-bit unsigned (value range 0 – 255):
DELAYTIME = 10
20 – 11 is 9 (not timeout yet)
255 – 246 is 9
4 – 251 is 9 (yes it is with 8-bit unsigned! : still no timeout)

Reply

Mark from Hungary

January 15, 2020 at 11:42 am

Hi there,

i need to read a data pin and a clock pin for a serial communication, data transmission starts with clock goes low. then the next clock pulldown also bring data line low or high.
I thought that i listening this action with interrupt, but altogether 64 times goes clock low, so means (i think) 64 times i go into the interrupt loop.

So whether is this a good solution if i detach the interupt with detachinterupt() at the first command in the interupt routine?
then i can wait for all clocks and that arrives… what do you think?

Reply

Jan Pekkeriet

May 20, 2020 at 2:19 pm

This reaction is probably too late for you but anyway:
When there is an Interrupt, on the Clock-line, then read the Data-line and save this reading, this is all your Interrupt should do. Only detach the interrupt, if you don’t want to receive data anymore.

Reply

Kamal

December 7, 2021 at 8:46 am

Hi, I know i'm following this too late, but encountered problem now, I wanted to do the same, but problem is that my clock that I am reading is around 1MHz, so I am not getting correct data. Can I know what is the interrupt latency of esp8266 on gpio for 80MHz clock or 160MHz clock? and if it is possible to read data at that clock using esp8266 gpio interrupts?Thanks

Reply

Alejandro

February 16, 2020 at 4:49 pm

Hi Rui and Sara.
I would like to use the RCSwitch library for data reception but to use in esp8266.
I tried it on ReceiveDemo advanced for arduino but I need to understand how to use it in a wifi module.
Can you give me a guide to solve it?

Reply

Svicar

August 4, 2020 at 6:03 am

Helo..I also have problems on ESP32 with rcswitch. When I short press remote, the ESP32 with receiver get nothing. When I long press I recieve the remote code. It seem that the interrupt dont triger. Do you have a solution?

Reply

Jorge

October 13, 2020 at 1:43 pm

Hello,
Everything fine and very clear, but I can't find a way to make your NodeMCU code work on Wemos d1 mini.
Any suggestion?
Thanks and regards

Reply

Sara Santos

October 16, 2020 at 9:04 am

Hi.
What's the error that you're getting?
Regards,
Sara

Reply

musicgowdam

January 23, 2022 at 1:29 pm

Instead of
const int ledPin = 26; as mentioned in the code,

the following declaration works for Wemos d1 mini.
const int ledPin = LED_BUILTIN;

Reply

Ray Constantine
November 18, 2020 at 8:59 am

Hi Rui, I just love your tutorials. I'm 70, and so a slow learner, but I'm getting there.
I have a problem trying to use an interrupt with your web server on an ESP8266. The interrupt routine counts pulses from a hall effect flow meter, and it resets the web server and I get a code exception, even when no pulses are coming in. This is the serial monitor error:

WiFi connected.
IP address:
10.1.1.52
ISR not in IRAM!

User exception (panic/abort/assert)
———————— CUT HERE FOR EXCEPTION DECODER ————————

Abort called

stack>>>

ctx: cont
sp: 3ffffee0 end: 3fffffc0 offset: 0000
3ffffee0: 60000314 00000001 3ffee8d0 3ffee6f8
3ffffef0: 000000fe 00000000 00000000 00000000
3fffff00: 00000000 00000000 00000000 00ff0000
3fffff10: 5ffffe00 5ffffe00 3ffee6b8 00000000
3fffff20: 00000001 00000002 3ffee6b8 40204b76
3fffff30: 401006ee 3ffee6b8 3ffee6f5 40204b88
3fffff40: 00000000 00000020 3ffee6b8 40205699
3fffff50: 00000000 3ffee6b8 3ffee6e4 402033cc
3fffff60: 3ffe8649 00001194 3ffee6b8 3ffee890
3fffff70: 3ffe8649 3ffe84cc 3ffee6b8 40205748
3fffff80: 3ffe84d4 3ffe84cc 3ffee6b8 40201166
3fffff90: 40206f48 3401010a feefeffe feefeffe
3fffffa0: 3fffdad0 00000000 3ffee850 40204558
3fffffb0: feefeffe feefeffe 3ffe84f8 40100fed
<<<stack<<<

———— CUT HERE FOR EXCEPTION DECODER ————

I would be so grateful if you could tell me what I’m doing wrong.

Reply

Sara Santos
November 18, 2020 at 11:11 am

Hi.
Your interrupt callback function must be in RAM.
Put the following word before the definition of your callback function.

ICACHE_RAM_ATTR
Regards,
Sara

Reply

Ray Constantine
November 19, 2020 at 5:58 am

Thank you Sara, that solved that issue. I'm still having trouble with the interrupt but I'll keep reading your tutes and maybe find a way.

Reply

Sara Santos
November 19, 2020 at 11:29 am

Great.
Please take into account that you should avoid using delay() inside the callback function.
Regards,
Sara

Reply

Mckenzie Gibson
February 15, 2021 at 5:37 pm

Another excellent article. I am trying to count pulses on 2 electric meters and one heat energy meter. the electric meters show a NO switch which I sispect closes on each 1/10 kWHhr used. one of the two terminals is marked +. I am getting a lot of noise/false counts even when there is no power consumption. I am also getting wha I take to be contact bounce.
Do you have any suggestions. eg I am using a 6kohm pull down resistor on the non plus pin. Should I use a bigger resistor?
Is is OK to add a de-bounce program line within the interrupt?

I look forward to your help

Reply

Leonardo
April 7, 2021 at 12:27 am

hi, i'm doing a project that interrupts the loop with the line signal (10ms) and then i have to fire an output to control a scr and the problem is that the sketch takes much longer than 10ms.
I believed that the 8266 had an internal timer that generates an interruption and from what I am investigating it seems that this is not the case
I await your comments, thank you very much

Reply

se
June 8, 2021 at 12:44 pm

Great tut, one question, startTimer and lastTrigger shouldn't be declared volatile?

Reply

Gerson

September 20, 2021 at 11:18 pm

Very good article, thank you. I have a question: A ´Serial.println´ was used when the interruption is active. Wouldn't that cause a code exception during the execution ? Thanks in advance .

Reply

Oleg

November 26, 2021 at 11:18 am

Hi Rui & Sara!
On one hand you pull the interrupt pin up, that is, it is normally High, on the other hand, you attach the interrupt to the Rising event. Does it mean that the device detects voltage overshoot over the normal High level?
Thanks for all you're doing,
Oleg

Reply

Tom

February 9, 2022 at 11:25 pm

Hi
Your tutorials are a great help to me starting ESP8266 arduino programming, I could not get the
1st Timer (millis) tutorial working until I traced to typing error below
“const int ledPin = 26; // the number of the LED pin ” this should read as below
“const int ledPin = 16; // the number of the LED pin”
Thank you

Reply

Gil

March 2, 2022 at 7:33 pm

Hi!
Thank you so much for this tutorial. It helped a lot. Everything not too much, not too little explained.

Just a hint about something that should be updated: `ICACHE_RAM_ATTR`is deprecated. One should use `IRAM_ATTR`these days.

best regards,
Gil

Reply

Youssef Dirani

May 23, 2022 at 2:35 pm

Hi,
super thanks for your effort.
In the ads, there is an image for a woman almost in her under****.
Please be cautious as this causes a major distraction to weak people like me.

Reply

Sara Santos

May 24, 2022 at 9:11 pm

Hi.
Our ads are filtered, and you're not supposed to get adds like that.
Please report any ads that might seem innapropriate.
Regards,
Sara

Reply

Cepi burhanudin

July 19, 2022 at 10:26 am

Halo tuan baga mana cara agar hal tersebut bisa di gunakan untuk pengaturan kurva pengapian kendaraan bermotor.harap bimbingan nya

Reply

Leave a Comment

Name *

Email *

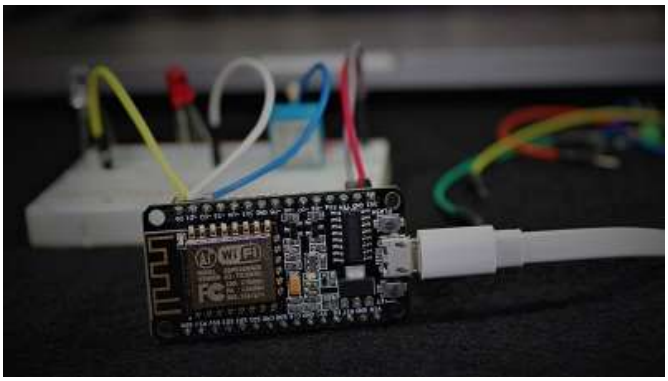
Website

☐ Notify me of follow-up comments by email.

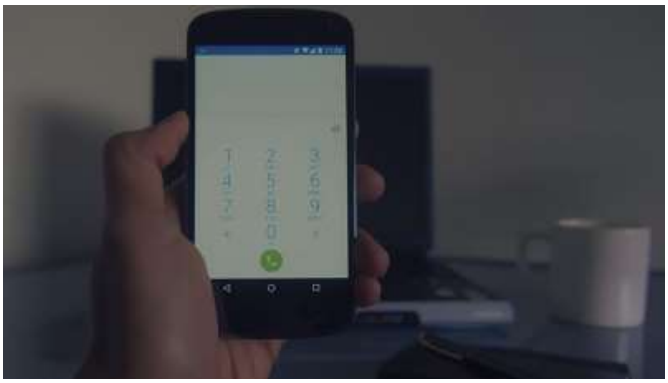
☐ Notify me of new posts by email.

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.