# Timer Interrupts in MicroPython (ESP32)

Timer interrupts allow your ESP32 to execute code at precise intervals without blocking the main program. In **MicroPython**, you can use the **machine.Timer** module to set up timer-based interrupts.

**1. Understanding ESP32 Timer Types**

ESP32 has two types of timers in MicroPython:

* **Software Timer (machine.Timer)**
  + Runs in software and is suitable for most applications.
* **Hardware Timer (machine.Timer with mode=Timer.PERIODIC)**
  + Uses ESP32's hardware timers for more precise timing.

**2. Basic Timer Example**

This example toggles an LED every **500 milliseconds** using a timer.

**Wiring:**

* **LED** → ESP32 **GPIO2** (or any other available pin)

**Code:**

from machine import Pin, Timer

# Define LED pin

led = Pin(2, Pin.OUT)

# Timer callback function

def toggle\_led(timer):

led.value(not led.value()) # Toggle LED

# Create a hardware timer

timer0 = Timer(0) # Timer ID 0

timer0.init(period=500, mode=Timer.PERIODIC, callback=toggle\_led)

✅ **What Happens?**

* The toggle\_led function runs **every 500ms**, toggling the LED state.
* The Timer.PERIODIC mode ensures it repeats continuously.

**3. One-Shot Timer Example**

A **one-shot** timer runs **only once** after a delay.

**Code:**

from machine import Timer

def one\_time\_action(timer):

print("Timer executed once!")

timer1 = Timer(1)

timer1.init(period=3000, mode=Timer.ONE\_SHOT, callback=one\_time\_action)

✅ **What Happens?**

* After **3 seconds**, "Timer executed once!" prints.
* The timer **does not repeat**.

**4. Timer with Multiple Interrupts**

You can run **multiple timers** independently.

**Code:**

from machine import Pin, Timer

led1 = Pin(2, Pin.OUT)

led2 = Pin(4, Pin.OUT)

def blink\_led1(timer):

led1.value(not led1.value())

def blink\_led2(timer):

led2.value(not led2.value())

timer0 = Timer(0)

timer1 = Timer(1)

timer0.init(period=500, mode=Timer.PERIODIC, callback=blink\_led1) # LED1 every 500ms

timer1.init(period=1000, mode=Timer.PERIODIC, callback=blink\_led2) # LED2 every 1s

✅ **What Happens?**

* **LED1 blinks every 500ms**.
* **LED2 blinks every 1 second**.

**5. Stopping a Timer**

To **stop** a timer, use:

timer0.deinit() # Stops Timer 0

This will **disable the interrupt**.

**6. Using Timer for Non-Blocking Code Execution**

Instead of using time.sleep(), you can use a timer to **schedule tasks** without blocking other operations.

**Example: Non-blocking Counter**

from machine import Timer

counter = 0

def increment\_counter(timer):

global counter

counter += 1

print("Counter:", counter)

timer2 = Timer(2)

timer2.init(period=1000, mode=Timer.PERIODIC, callback=increment\_counter)

✅ **What Happens?**

* The counter **increments every second**.
* The main program remains **responsive**.

**7. Handling Critical Sections (Disabling Interrupts Temporarily)**

If your interrupt modifies shared data, you may want to **disable interrupts temporarily** using **micropython.disable\_irq()**.

**Example: Prevent Interrupt Collision**

import micropython

from machine import Timer

counter = 0

lock = False

def safe\_increment(timer):

global counter, lock

irq\_state = micropython.disable\_irq() # Disable interrupts

counter += 1

micropython.enable\_irq(irq\_state) # Re-enable interrupts

print("Counter:", counter)

timer3 = Timer(3)

timer3.init(period=1000, mode=Timer.PERIODIC, callback=safe\_increment)

✅ **Why?**

* Prevents multiple interrupts from interfering with the counter update.

**Summary**

| **Feature** | **Usage** |
| --- | --- |
| Timer.PERIODIC | Runs continuously at intervals |
| Timer.ONE\_SHOT | Runs **once** after a delay |
| .deinit() | Stops the timer |
| disable\_irq() | Temporarily disables interrupts |

Would you like a **specific timer application** (e.g., debounce buttons, ADC sampling, etc.)? 🚀

In MicroPython, the **timer parameter** passed to the callback function is an instance of the **machine.Timer** class.

**Timer Callback Function Parameter Type**

The callback function for a timer **must accept a single argument**, which is the **timer instance that triggered the callback**.

**Example:**

from machine import Timer

def my\_callback(timer):

print("Timer triggered:", timer)

t0 = Timer(0)

t0.init(period=1000, mode=Timer.PERIODIC, callback=my\_callback)

✅ **What Happens?**

* Every **1 second**, my\_callback() runs.
* The timer parameter inside my\_callback(timer) is the **Timer(0) instance** that triggered the callback.

**Checking the Timer Parameter Type**

You can check the type inside the callback:

def my\_callback(timer):

print("Timer type:", type(timer))

t1 = Timer(1)

t1.init(period=500, mode=Timer.PERIODIC, callback=my\_callback)

📌 **Output:**

Timer type: <class 'Timer'>

**Key Takeaways**

1. The callback function **must accept one argument**.
2. This argument is a **Timer object**, which can be used inside the callback.
3. You can **pass the same function to multiple timers**, and check timer to identify which one triggered it.

Would you like more advanced timer usage examples? 🚀

The machine.Timer class in MicroPython provides several methods to configure and control hardware timers on the ESP32.

## ****List of**** machine.Timer ****Methods****

|  |  |
| --- | --- |
| **Method** | **Description** |
| Timer(id) | Creates a timer object with the given id (0-3 for ESP32). |
| .init(period, mode, callback) | Initializes the timer with a time period, mode, and function to call. |
| .deinit() | Stops the timer and releases resources. |
| .value() | (Some versions) Returns the current timer value. |

## ****📌 Timer Method Details & Examples****

### Timer(id) ****→ Create a Timer Instance****

Creates a new **Timer** object.

* ESP32 supports **hardware timers 0-3**.

from machine import Timer

t0 = Timer(0) # Create Timer 0

t1 = Timer(1) # Create Timer 1

### init(period, mode, callback) ****→ Initialize Timer****

This method starts the timer.

#### ****Parameters****:

* **period** (int) → Timer interval in **milliseconds**.
* **mode** (enum)
  + Timer.PERIODIC → Repeats continuously.
  + Timer.ONE\_SHOT → Runs once.
* **callback** (function) → Function to run when the timer triggers.

#### ****Example: Periodic Timer****

from machine import Timer

def my\_callback(timer):

print("Timer triggered!")

t0 = Timer(0)

t0.init(period=1000, mode=Timer.PERIODIC, callback=my\_callback) # Runs every 1s

#### ****Example: One-Shot Timer****

t1 = Timer(1)

t1.init(period=3000, mode=Timer.ONE\_SHOT, callback=lambda t: print("One-time trigger!"))

### deinit() ****→ Stop the Timer****

This **stops** the timer and **frees resources**.

t0.deinit() # Stops Timer 0

### value() ****→ Get Timer Value (Limited Support)****

On some MicroPython versions, .value() may return the current timer count.  
If it's **not supported**, you’ll get an **AttributeError**.

print(t0.value()) # Might return the current count

## ****Summary****

|  |  |
| --- | --- |
| **Method** | **Usage** |
| Timer(id) | Create a timer object |
| .init(period, mode, callback) | Start the timer |
| .deinit() | Stop the timer |
| .value() | (Optional) Get timer count |

Would you like an advanced example, like **PWM generation or ADC sampling** with a timer? 🚀