ESP32 Timers, Interrupts, and Debounce: A Comprehensive Guide

Introduction

The ESP32 microcontroller offers robust capabilities for handling timers, interrupts, and debounce mechanisms. This document will explore these topics in detail, using MicroPython code examples that you can apply in practical projects.

---

1. Timers in ESP32

Timers are essential for executing tasks at specific time intervals. The ESP32 supports hardware timers that can operate independently of the CPU, making them useful for precise time-based operations.

1.1 Timer Basics

The ESP32 provides multiple hardware timers. Each timer can operate in different modes:

- **ONE_SHOT:** Executes the timer callback function once after a specified delay.
- **PERIODIC:** Executes the callback function at regular intervals.

1.2 Timer Example in MicroPython

Below is an example of how to use a timer in MicroPython:

```
from machine import Timer

def timer_callback(timer):
    print("Timer triggered!")

# Create a timer instance
my_timer = Timer(0)
my_timer.init(mode=Timer.PERIODIC, period=1000,
callback=timer_callback)  # Runs every 1000ms (1s)
```

This timer will execute timer_callback every second, printing "Timer triggered!" to the console.

---

2. Interrupts in ESP32

Interrupts allow the ESP32 to respond immediately to external events, such as button presses or sensor readings, without constantly checking their status (polling).

2.1 Interrupt Service Routines (ISR)

- An ISR is a function that gets executed when an interrupt occurs.
- It should be as short as possible to avoid blocking the main program.
- Avoid using print(), sleep(), or complex computations inside an ISR.

2.2 Button Interrupt Example

Below is a practical example of using an interrupt to toggle an OLED display on button press:

```
from machine import Pin, Timer
import ssd1306

i2c = machine.I2C(scl=machine.Pin(9), sda=machine.Pin(8))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

button = Pin(0, Pin.IN, Pin.PULL_UP)
pressed = False

def button_pressed(pin):
    global pressed
    pressed = not pressed
    if pressed:
        oled.poweroff()
    else:
   oled.poweron()

button.irq(trigger=Pin.IRQ_FALLING, handler=button_pressed)  #
Interrupt on button press
```

This code sets up a button that toggles the OLED display on and off when pressed.

---

3. Debouncing Mechanisms

Mechanical switches (buttons) tend to "bounce," causing multiple rapid signals instead of a single clean press. Debouncing helps filter out these unwanted signals.

3.1 Software Debouncing

A software debounce technique introduces a small delay after detecting an interrupt, ensuring that only one event is processed per press.

```
debounce_timer = None

def button_pressed(pin):
    global debounce_timer, pressed
```

```
    if debounce_timer is None:
        pressed = not pressed
        if pressed:
            oled.poweroff()
        else:
            oled.poweron()

        # Start a debounce timer
        debounce_timer = Timer(0)
        debounce_timer.init(mode=Timer.ONE_SHOT, period=200,
callback=debounce_callback)

def debounce_callback(timer):
    global debounce_timer
    debounce_timer = None
```

This ensures that a button press is only registered once every 200ms.

---

4. Practical Example: Reading Sensor Data and Displaying on OLED

Let's integrate everything and create a full program that:

- Reads temperature and humidity from a DHT11 sensor.
- Displays sensor data on an OLED screen.
- Uses a button interrupt with debounce to toggle the OLED display.

```
from machine import Pin, I2C, Timer
import machine
import ssd1306
import dht
import time

DHT_PIN = 4  # DHT11 data pin
button = Pin(0, Pin.IN, Pin.PULL_UP)
dht_sensor = dht.DHT11(machine.Pin(DHT_PIN))
i2c = machine.I2C(scl=machine.Pin(9), sda=machine.Pin(8))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

pressed = False
debounce_timer = None

def button_pressed(pin):
```

```
    global debounce_timer, pressed

    if debounce_timer is None:
        pressed = not pressed
        if pressed:
            oled.poweroff()
        else:
            oled.poweron()

        debounce_timer = Timer(0)
        debounce_timer.init(mode=Timer.ONE_SHOT, period=200,
callback=debounce_callback)

def debounce_callback(timer):
    global debounce_timer
    debounce_timer = None

button.irq(trigger=Pin.IRQ_FALLING, handler=button_pressed)

while True:
    try:
        dht_sensor.measure()
        time.sleep(0.2)
        temp = dht_sensor.temperature()
        humidity = dht_sensor.humidity()
        print(temp, humidity)
        oled.fill(0)
        oled.text("Temp: {} C".format(temp), 0, 0)
        oled.text("Humidity: {}%".format(humidity), 0, 16)
        oled.show()
    except Exception as e:
        print("Error reading DHT11 sensor:", e)
    time.sleep(1)  # Update every second
```

Conclusion

This guide covered essential concepts of timers, interrupts, and debouncing in ESP32 using MicroPython. By integrating these techniques, you can build responsive and efficient embedded systems.

Key Takeaways:

- **Timers** help execute code at defined intervals without blocking execution.

- **Interrupts** allow the ESP32 to respond quickly to external events.
- **Debouncing** prevents unintended multiple detections from buttons and mechanical switches.

These concepts are fundamental to real-world embedded system development and can be further expanded with additional peripherals and sensors.