Name:	Muhammad Ibtisam Butt
Reg No:	23-NTU-CS-1269
Section:	BSAI^{4TH}
Subject	Embedded IOT
Submitted To:	Sir. Nasir Mehmood

Assignment-01

Question 1

1. Why is volatile used for variables shared with ISRs?

- When a variable is changed inside an **Interrupt Service Routine (ISR**), the compiler might try to optimize the code by storing its value in a register. This can cause the main program to read an outdated value.
- By declaring such a variable as volatile, we tell the compiler not to optimize it and always read its current value directly from memory. This ensures proper synchronization between the ISR and the main code.

2. Compare hardware-timer ISR debouncing vs delay()-based debouncing.

- **Hardware-timer ISR debouncing** uses internal timers to handle switch bouncing with precise timing, so it's accurate and doesn't stop other code from running. It's ideal for real-time or multitasking programs.
- **delay()-based debouncing**, on the other hand, stops all code execution during the delay period, which makes the system slower and less responsive.
  Overall, timer-based debouncing is more efficient and reliable.

3. What does IRAM_ATTR do, and why is it needed?

- **The IRAM_ATTR keyword** tells the compiler to store the ISR function in the ESP32's internal RAM (IRAM) instead of flash memory.
- This is important because during certain operations (like flash memory writing), the flash cannot be accessed. Keeping the ISR in RAM ensures it can still run instantly and safely without causing system crashes or delays.

4. Define LEDC channels, timers, and duty cycle.

- **Channels:** These are PWM outputs linked to specific GPIO pins. Each channel can control a device such as an LED or motor.
- **Timers:** Control how quickly the PWM signal turns ON and OFF, defining the signal's frequency.

- **Duty cycle:** Represents how long the signal stays ON within one full cycle, expressed as a percentage. Higher duty cycle means brighter LEDs or faster motor speed.

5. Why should you avoid Serial prints or long code paths inside ISRs?

- **Interrupt Service Routines** must execute very quickly. Using Serial.print() or running long code inside an ISR can delay other interrupts and make the system unstable.
- It's better to set a flag inside the ISR and handle the actual work in the main loop, where delays won't cause problems.

6. What are the advantages of timer-based task scheduling?

- **Timer-based scheduling automatically triggers** tasks at fixed intervals without using delays. This improves timing accuracy and makes the code cleaner.
- It also allows multiple tasks (like blinking LEDs or reading sensors) to run in parallel without interfering with each other, improving multitasking and system efficiency.
- multitasking easier.

7. Describe I²C signals SDA and SCL.

- **SDA (Serial Data Line):** Carries the data being transmitted between the master and slave devices.
- **SCL (Serial Clock Line):** Sends timing pulses generated by the master to control data flow.
  Both lines are open-drain and require pull-up resistors. Data moves in sync with clock pulses on SCL.

8. What is the difference between polling and interrupt-driven input?

- **Polling:** The microcontroller continuously checks an input in a loop to see if it has changed. This wastes CPU time if nothing happens.
- **Interrupt-driven**: The microcontroller waits until a signal change triggers an interrupt, then responds immediately.
  Interrupt-driven input is more efficient and allows the processor to do other tasks while waiting for events.

9. What is contact bounce, and why must it be handled?

- When a button or switch is pressed, the contacts don't close cleanly they rapidly make and break connection several times in a few milliseconds. This causes multiple unwanted signals (bounces).
- If not filtered, the microcontroller may think the button was pressed several times. Debouncing (in hardware or software) smooths out these false signals and ensures only one valid press is detected.

10. How does the LEDC peripheral improve PWM precision?

- **The ESP32's LEDC (LED Controller**) module uses dedicated hardware timers and high-frequency clocks to create very fine **PWM** signals.
- It supports high resolution (up to 20 bits) and stable timing, resulting in smoother LED brightness transitions and better motor speed control, without using much CPU power.

11. How many hardware timers are available on the ESP32?

- The ESP32 has 4 general-purpose hardware timers, and each one has two 64-bit counters (one per group).
- These timers can be used for counting, measuring intervals, or generating periodic events independent of the main CPU tasks.

12. What is a timer prescaler, and why is it used?

- A **prescaler divides** the main clock frequency before it reaches the timer. For example, if the main clock is 80 MHz and the prescaler is 80, the timer runs at 1 MHz.
- This allows you to create longer delays or slower PWM signals while maintaining high precision in timing control.

13. Define duty cycle and frequency in PWM.

- **Duty cycle**: The ratio of the ON time to the total period of a signal, usually shown as a percentage.
- **Frequency:** How many complete ON/OFF cycles occur in one second. Together, these control how bright an LED glows or how fast a motor spins.

14. How do you compute duty for a given brightness level?

- To calculate the duty value:

$$Duty = (Brightness\%/100) \times MaxDuty$$

- **For example**, if maximum duty is 1023 and brightness is 50%, then duty = 512.

A higher duty means the LED stays ON longer, producing more brightness.

15. Contrast non-blocking vs. blocking timing.

- **Blocking timing:** Uses delay() which stops the program from doing anything else during the delay.
- **Non-blocking timing:** Uses functions like millis() or hardware timers to track time without freezing the program.

Non-blocking timing is preferred for multitasking or responsive systems.

16. What resolution (bits) does LEDC support?

The **LEDC** module supports resolutions from 1 bit up to 20 bits.
Higher resolution means more steps in duty cycle control, allowing very fine adjustments in LED brightness or motor power **(e.g., 20 bits = 1,048,576 steps).**

17. Compare general-purpose hardware timers and LEDC (PWM) timers.

- **General-purpose timers**: Can be used for delays, counting, or generating interrupts. They're flexible for many timing tasks.

- **LEDC timers**: Are specialized hardware timers built specifically for generating PWM signals with precise frequency and duty control.
  LEDC timers are **better for smooth LED dimming or motor control.**

18. What is the difference between Adafruit_SSD1306 and Adafruit_GFX?

- **Adafruit_SSD1306:** A driver library for OLED displays that use the SSD1306 controller chip. It handles communication with the display.

- **Adafruit_GFX:** A core graphics library that provides drawing functions like text, lines, and shapes.

The SSD1306 library depends **on GFX to render graphics on the screen.**

19. How can you optimize text rendering performance on an OLED?

To make text updates faster:

- Use smaller fonts or simpler graphics.
- Only redraw the areas that change instead of refreshing the full screen.
- Avoid clearing the entire display every frame.
- Store fonts in program memory (PROGMEM) if possible.
  These steps reduce flicker and improve performance.

20. Give short specifications of your selected ESP32 board (NodeMCU-32S).

- Dual-core 32-bit processor (up to 240 MHz)
- Built-in WiFi and Bluetooth
- 520 KB SRAM
- Multiple GPIO pins
- 4 hardware timers, 2 ADCs, and PWM support
- USB-to-UART for easy programming

Question 2

1. A 10 kHz signal has an ON time of 10 ms. What is the duty cycle? Justify with the formula.

- The frequency is 10 kHz, so the time period is:

$$T=1/f=1/10000=0.0001s=0.1ms$$

- But the question says ON time = 10 ms, which is longer than the total period.
  which is much longer than the total period (100 µs). That is not possible for a 10 kHz square wave.
- Now using the formula:

$$\textbf{Duty Cycle}=T_{ON}/\text{Total} \times 100=10\mu s/100\mu s \times 100$$

$$\textbf{Duty Cycle} = 10\%$$

2. How many hardware interrupts and timers can be used concurrently? Justify.

- **Timers:** ESP32 provides **4 general-purpose hardware timers** (usable concurrently).
- **Interrupts (hardware sources):** There are **many interrupt sources** — each GPIO can be an interrupt, plus UART, SPI, I²C, timers, ADC, RTC, etc.
- **Concurrency limits:** You can enable many interrupts at once; the CPU services them by priority. Practically you can use multiple interrupt sources concurrently, but only **4 independent hardware timers** at the same time.
- **Note:** Efficient ISR code and proper priority/handler design affect how many interrupts you can handle reliably.

3. How many PWM-driven devices can run at distinct frequencies at the same time on ESP32? Explain constraints.

- **LEDC architecture:** The ESP32 LEDC has **16 PWM channels** and **4 LEDC timers**.
- **Distinct frequencies:** Each LEDC timer can be configured to a different frequency. Therefore you can have up to **4 distinct PWM frequencies** simultaneously.
- **Channel mapping:** The 16 channels are grouped onto those 4 timers. Channels sharing a timer must use the same frequency.
- **Constraints:**
    - If you need more independent frequencies, you must reassign timers (limited to 4) or use software PWM (less precise).
    - Resolution and prescaler choices per timer affect achievable frequencies and duty resolution.

4. Compare a 30% duty cycle at 8-bit resolution and 1 kHz to a 30% duty cycle at 10-bit resolution (all else equal).

- **8-bit:** 256 levels (0–255). Step size ≈ 1 / 255 ≈ **0.392% per step**.
    - 30% corresponds to duty value ≈ 0.30 × 255 ≈ **77** (rounded). Actual = 77/255 ≈ 30.2%.
- **10-bit:** 1024 levels (0–1023). Step size ≈ 1 / 1023 ≈ **0.0977% per step**.
    - 30% corresponds to duty value ≈ 0.30 × 1023 ≈ **307** (rounded). Actual = 307/1023 ≈ 30.0%.
- **Comparison / result:** Both produce roughly the same average brightness (30%), but **10-bit** gives **much finer steps** and smoother dimming — fewer visible jumps and better precision.

5. How many characters can be displayed on a 128×64 OLED at once with the minimum font size vs. maximum font size?

Assume the **smallest font size** is 6×8 pixels (width x height).

- Columns=128/6=21
- Rows=64/8=8
- So, **21 × 8 = 168 characters** max.

For the **largest font size**, say 12×16 pixels:

- Columns=128/12=10
- Rows=64/16=4
- So, 10 × **4 = 40 characters** max.

So, **10 × 4 = 40 characters** can fit.

Answer: About **168 small characters** or **40 large characters** can fit on the 128×64 OLED.