National Textile University, Faisalabad

Assignment.No.1

Name	Noor Fatima
Reg no	23-NTU-CS-1083
Section	BSCS 5^{th} -A
Subject	Embedded IOT Systems
Assignment	19-10-2025
Submitted to	Sir Nasir

Department of Computer Science

Question 1 — Short Questions

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

Volatile is used for variables shared with ISRs to prevent the compiler from optimizing them away, ensuring that updates made by the ISR are always read from main memory and not from a cached value.

- **With volatile:** Each time the loop checks done, it will force a read from the actual memory location, guaranteeing that it always gets the updated value, whether set by the main program or the ISR.
- **Without volatile**, the compiler might assume the variable's value is static and reuse a cached copy, leading to incorrect program logic when the main program expects the most up-to-date value set by the ISR.

- Volatile prevents incorrect assumptions in optimizations by the compiler in multi-threaded or interrupt-driven environments."

Volatile is necessary:

- Disables compiler optimizations
- Ensures memory visibility
- Prevents race conditions

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

Hardware debouncing is superior in performance because it allows the main program to continue running, whereas **delay()** halts all execution, causing the system to become unresponsive and potentially missing other events.

Feature	Hardware-timer ISR Debouncing	Delay()-based Debouncing
Execution	**Non-blocking:** Allows the main program to continue running while	**Blocking:** Halts all program execution, including the main loop,

	the interrupt service routine (ISR) manages the debouncing logic.	for the duration of the delay.
Responsiveness	**High:** The system remains responsive to other interrupts and events.	**Low:** The system is unresponsive during the delay, and may miss other button presses or events.
Application	**Ideal for embedded systems:** Crucial for microcontrollers where ISRs must be fast and non-blocking to maintain system stability.	**Unsuitable for ISRs:** delay() functions are explicitly warned against in ISRs because they are blocking and can cause issues with timing and responsiveness.
Performance	**Efficient:** Minimal overhead, as it only uses a timer to schedule a brief check, making it very power-efficient.	**Inefficient:** Wastes processor cycles by blocking the system, which is especially inefficient in a real-time or embedded environment.

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

IRAM_ATTR is a macro that tells the linker to place code, such as interrupt handlers and time-critical functions, into the microcontroller's Internal RAM (IRAM) instead of flash memory.

- This is **needed** to improve performance by allowing faster access to the code, which is essential for real-time applications like interrupted service routines (ISRs), as executing directly from RAM avoids delays caused by flash cache misses.

IRAM_ATTR Task

- Places code in IRAM
- Enables faster execution
- Helps avoid cache misses

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

LEDC Channels:

LEDC **channels** are individual outputs on a hardware peripheral that generate PWM signals.

- **Function:** Each channel can be configured to operate independently, allowing for control of multiple devices (e.g., a separate channel for red, green, and blue LEDs).
- **Grouping:** Channels are often grouped together, and multiple channels can share a single timer.

Timers:

Each channel is associated with a **timer**, which sets the fundamental frequency and resolution for the signal.

- **Function:** The timer defines the total period of the pulse. By changing the timer's configuration, you change the frequency of the PWM signal for all channels using it.
- **Configuration:** You configure the timer first, and then associate it with one or more channels.

Duty Cycle:

The **duty cycle** is the ratio of the 'on' time to the total period of the pulse, which determines the signal's output level, like the brightness of an LED.

- **Function:** It determines the effective output level. For example, a higher duty cycle for an LED corresponds to it being on for a longer duration within each cycle, making it appear brighter.
- **Calculation:** Duty Cycle = (Pulse Width / Pulse Period) * 100%.

- **Control:** The duty cycle can be changed to gradually increase or decrease the output, a process known as fading

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

You should avoid **Serial.print()** or long code paths inside an Interrupt Service Routine (ISR) because they take a significant amount of time to execute, which can disrupt the precise timing of your program, cause other interrupts to be delayed, and lead to unexpected behavior.

- ISRs need to be short and fast to quickly manage their primary task and return control to the main program without causing timing issues or leaving the system unresponsive.

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

Advantages of timer-based task scheduling:

- Precise timing for task execution
- Allows multitasking
- Reduces CPU idling
- Energy-efficient
- Suitable for real-time systems

7. Describe I^2C signals SDA and SCL.

SDA and SCL are the two-wire serial signals used in the I^2C communication protocol. **SDA (Serial Data)** is the bi-directional line for transmitting and receiving data, while **SCL (Serial Clock)** is the line that carries the clock signal for synchronization between devices.

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

The main difference is that **polling** is when the CPU continuously checks a device to see if it needs attention, while an **interrupt-driven** system is when the device signals the CPU that it needs attention, allowing the CPU to continue other tasks until then.

Feature	Polling	Interrupt-Driven
How it works	The CPU periodically and actively checks a device's status to see if an event has occurred.	A device sends a hardware signal to the CPU when an event occurs, causing the CPU to pause its current task to manage the event.
Responsiveness	It can have higher latency, especially if the polling interval is long, and the CPU might miss events if it is busy with a long operation.	Provides near-immediate response, as the CPU is alerted as soon as an event happens.
CPU usage	Resource intensive and inefficient, as the CPU wastes cycles checking for events that may not have occurred.	Efficient, as the CPU can focus on other tasks until a device signals for attention, saving processing time and energy.
Implementation	Simple to implement and is a software-based approach.	More complex to implement, requiring an interrupt handler and managing interrupt priorities.
Best for	Situations where the data is expected frequently or where responsiveness is not critical.	Situations requiring fast and timely responses to external events, such as handling button presses or sensor data.

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

- **Mechanical phenomenon**: When a switch is closed, the physical force of the contacts colliding, combined with the elasticity of the materials, causes the contacts to bounce for a few milliseconds before settling into a solid connection.

- **Electrical effect**: This physical bouncing creates rapid, unstable electrical pulses as the contacts briefly open and close, resulting in a noisy, erratic signal instead of a clean, steady one.
- **Causes**: Bounce is a natural part of the switch's design and is caused by the mass of the moving contact, the elasticity of the mechanism, and the impact forces involved.

Why handled

- Digital circuit errors:
- Impact on performance
- Equipment malfunction
- Debouncing

10. How does the LEDC peripheral improve PWM precision?

The LEDC peripheral improves PWM precision through its **hardware-based, dedicated timers**, which generate high-resolution PWM signals independently of the processor, allowing for smooth and accurate fades and stable output.

This is achieved by configurable parameters like frequency and **high-resolution bit depth**, which can be set independently for each channel to control multiple signals with distinct settings, unlike less precise software-based methods.

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

- **Standard ESP32:** 4 hardware timers
- **ESP32-C3:** 2 hardware timers
- **ESP32-S2:** 4 hardware timers
- **ESP32-S3:** 4 hardware timers

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

A **timer prescaler** is a hardware module that divides the input clock frequency by a programmable factor before it reaches the timer, which slows down the timer's counting speed.

It is **used** to increase the range of measurable time intervals by allowing the timer to count for longer periods before overflowing, especially when the maximum count of the timer register (like 8-bit or 16-bit) is too short for the desired duration.

- Effective Timer Clock = Base Clock / Prescaler

For example, without a prescaler, a timer might overflow in milliseconds, but with a prescaler, it could be extended to seconds.

13. Define duty cycle and frequency in PWM.

in PWM, **duty cycle** is the percentage of a single cycle where the signal is "on" (high) to the total period of one complete cycle. while **frequency** is the number of complete cycles that occur per second, measured in Hertz

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

Duty (%) = (Brightness Level / Max Level) × 100

or for code:

duty = (brightness Level / 255.0) * maxDutyValue;

15. Contrast non-blocking vs. blocking timing.

Blocking timing

A call or operation is blocked until the requested action is completed. The program flow pauses and waits for the operation to finish before moving to the next line of code.

- **Execution:** Sequential. Each statement executes one after the other, and each must finish completely before the next one starts.
- **Characteristics:**
  - Simpler and easier to understand and write.

    - Can lead to performance bottlenecks and unresponsiveness if a long-running operation is encountered.
    - Less efficient for tasks that involve waiting for external events, such as network or I/O operations.
- **Example:** A function that sends a message over a network will wait until the message is fully sent before it returns control to the caller.

Non-blocking timing

A call or operation returns immediately, without waiting for the action to be completed. The program flow continues while the operation is handled in the background.

- **Execution:** Concurrent. Operations can run in parallel. The system handles the result of the operation later.
- **Characteristics:**
    - More complex to implement.
    - Offers better performance and responsiveness, especially in applications that handle multiple operations simultaneously.
    - Requires more careful management of asynchronous results and potential race conditions.
- **Example:** An I/O operation that returns immediately and lets the program continue while data is being transferred in the background.

16. What resolution (bits) does LEDC support?

- General ESP32: 1 to 16 bits is supported.
- ESP32 (max): 1 to 20 bits are supported.

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

Feature	General-Purpose Timers	LEDC (PWM) Timers

Primary Function	To keep track of time, schedule tasks, and trigger events like interrupts.	To generate PWM signals for controlling device intensity, speed, or other analog-like functions.
Signal Output	It can be configured for various functions like generating waveforms but are not inherently specialized for analog-style control.	Outputs a square wave where the **duty cycle** (the proportion of time the signal is 'on') is modulated to control an output's average power.
Flexibility	Highly flexible; can be used for a wide range of tasks including input capture, output compare, and, in some cases, PWM.	Less flexible for general timing; optimized for high-resolution PWM generation and features like fading.
Specialized Features	Often include features like auto-reload counters, prescalers, and up/down counting capabilities.	Optimized for PWM tasks and often include specific features like multiple independent channels, high and low-speed modes, and hardware-based fading.
Example Use	**System Tasks:** Keeping track of system time, creating delays, and triggering timed events.	**Application Examples:** Dimming LEDs, controlling the speed of a DC motor, or generating a control signal for an electronic speed controller.

18. What is the difference between Adafruit_SSD1306 and Adafruit_GFX?

Adafruit_GFX is a **core graphics library** that provides a common set of functions for drawing shapes and text on various displays, while Adafruit_SSD1306 is a **hardware-specific library** that handles low-level communication with an SSD1306-based OLED display

Feature	Adafruit_GFX	Adafruit_SSD1306
Purpose	Core graphics primitives (drawing lines, circles, text)	Handles communication and control of the specific SSD1306 hardware
Functionality	Provides a common interface for different display types	Initializes the display, sets resolution, and controls its settings
Dependencies	Needs a hardware-specific library (like Adafruit_SSD1306) to function with a display	Relies on Adafruit_GFX for drawing and rendering commands
How they work together	Your code calls GFX functions to draw, and the SSD1306 library takes those commands and sends them to the hardware	

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

- Use smaller fonts
- Minimize screen updates
- Use display.display() only after bulk changes
- Use partial updates (if library supports)
- Avoid drawing unchanged text repeatedly

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

Microcontroller & Memory

- Microcontroller: ESP32 (dual-core LX6 microprocessor)
- Clock Speed: Up to 240 MHz
- SRAM: 520 KB
- ROM: 448 KB
- Flash: 32 Mbit (4 MB)

Wireless Connectivity

- Wi-Fi: 802.11 b/g/n (2.4 GHz)
- Bluetooth: v4.2 (Classic and BLE)
- Antenna: Onboard PCB antenna

Peripherals & I/O

- GPIOs: 34 programmable pins, some supporting touch functionality
- ADC: Up to 18 channels of 12-bit SAR ADC
- DAC: 2 x 8-bit DAC outputs
- PWM: 16 channels of LED PWM
- Serial Interfaces:
    - UART: 3 channels (including pins for USB programming)
    - I2C: 2 channels
    - SPI: 2 (VSPI and HSPI)
    - I2S: 2 channels
- Ethernet MAC: Available
- Security: Secure Boot and Flash Encryption

Power & Physical

- Power Supply: 5V (via micro-USB)
- Logic Level: 3.3V
- Dimensions: Approximately 51.4mm x 25.4mm

Question 2 — Logical Questions

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

- **10 kHz** signal means:

$$T = \frac{1}{f} = \frac{1}{10{,}000} = 0.0001 \text{ seconds} = 0.1 \text{ ms (period)}$$

- You were told the **ON time is 10 ms**, which is:

$$10 \text{ ms} > 0.1 \text{ ms (period)}$$

This is **not possible** because the ON time cannot exceed the period.

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

- The ESP32 has 4 general-purpose hardware timers (2 per core).
- It supports up to 32 interrupt sources per core.

This means you can use:

- Up to 4 hardware timers concurrently.
- Multiple interrupts at once, depending on peripheral usage and priority configuration.

Answer:

- Timers: 4 can be used concurrently.
- Interrupts: Up to 32 per core can be used, subject to design constraints.

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

The ESP32 uses the LEDC (LED Control) peripheral for PWM. It has:

- 16 PWM channels
- 8 PWM timers

Each timer can provide a unique frequency. Channels that share a timer must share the same frequency.

Answer:

- ➢ You can run up to 8 PWM devices at distinct frequencies since there are 8 timers.
- ➢ You can drive up to 16 PWM devices total, but only 8 can have unique frequencies.

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).

Common values:

- Frequency: 1 kHz → Period = 1 ms
- Duty cycle: 30%

8-bit resolution:

- $2^8 = 256$ levels
- 30% of 256 = 0.3 × 256 ≈ **77 steps**

10-bit resolution:

- $2^{10} = 1024$ levels
- 30% of 1024 = 0.3 × 1024 ≈ **307 steps**

Comparison:

Feature	**8-bit**	**10-bit**
Step size	~0.39% per step	~0.098% per step
Smoothness	Lower	Higher
Precision	Less	More

Answer:
Both give 30% duty cycle, but **10-bit PWM is smoother and more precise**, due to finer steps

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

OLED Resolution: 128×64 pixels

Assumptions:

- **Minimum font size:** 6×8 pixels
- **Maximum font size:** 16×32 pixels

Minimum font (6×8):

- Columns: 128 / 6 = 21
- Rows: 64 / 8 = 8
- Total: 21 × 8 = **168 characters**

Maximum font (16×32):

- Columns: 128 / 16 = 8
- Rows: 64 / 32 = 2
- Total: 8 × 2 = **16 characters**

Answer:

Font Size	Characters per Line	Rows	Total Characters
6×8 (min)	21	8	168
16×32 (max)	8	2	16