**Embedded IoT Systems — Assignment 1**

Name: Shamaim khan  
Registration No: 23-NTU-CS-1284

**Question 1 — Short Answers**

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

`volatile` prevents the compiler from optimizing reads/writes to a variable that may change outside normal program flow (for example inside an ISR or by hardware). This ensures the main code always reads the actual memory value and that writes are not reordered or removed.

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

Timer/ISR debouncing is non-blocking, precise, and allows the main loop to continue; it reads input and schedules a check after the debounce interval. `delay()`-based debouncing blocks execution, reduces responsiveness, and is less suitable when multiple tasks or interrupts must run.

**3. What does IRAM\_ATTR do, and why is it needed?**

`IRAM\_ATTR` places a function or data in internal RAM (IRAM) instead of flash. This allows the function to execute with lower latency and when flash is inaccessible (important for time-critical ISRs on the ESP32).

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

LEDC timer: configures PWM frequency and resolution. LEDC channel: attaches a timer to a GPIO and outputs PWM. Duty cycle: fraction of each PWM period the signal is high (expressed as a count relative to resolution or as a %).

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

Serial.print is slow and may block; long ISRs increase interrupt latency, can cause missed events, and hurt real-time behavior. ISRs should be short and defer heavy work to tasks.

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

Deterministic periodic execution, efficient CPU use (no busy-wait), precise timing, and easier concurrency with non-blocking code.

**7. Describe I²C signals SDA and SCL.**

SDA: bidirectional data line (open-drain). SCL: clock line driven by master (open-drain). Both need pull-up resistors; transfers are synchronized by SCL edges.

**8. Difference between polling and interrupt-driven input.**

Polling repeatedly checks input in software (CPU overhead and latency depends on polling rate). Interrupt-driven uses hardware to signal events immediately (lower CPU usage and faster response).

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

Mechanical switches often produce multiple rapid transitions when toggled (bounce). Unhandled bounce can cause multiple false triggers; debouncing ensures a single clean transition.

**10. How does LEDC improve PWM precision?**

LEDC uses hardware timers with configurable resolution and prescalers, allowing fine-grained duty steps and stable PWM without CPU overhead.

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

ESP32 provides 4 general-purpose hardware timers (2 timer groups × 2 timers). Additional peripherals (LEDC, RTC) have their own timers.

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

A prescaler divides the base clock to create a lower timer clock, enabling lower frequencies or longer timer counts and trading off resolution vs range.

**13. Define duty cycle and frequency in PWM.**

Frequency: how often the PWM period repeats (Hz). Duty cycle: portion of each period the output is high (percentage).

**14. How to compute duty for a given brightness level?**

duty\_count = round((brightness\_level / max\_level) \* (2^resolution - 1)). Example: 50% brightness at 8-bit => 0.5 \* 255 ≈ 128.

**15. Contrast non-blocking vs. blocking timing.**

Non-blocking uses timers/state-machines (`millis()`, hardware timers) and keeps main loop responsive. Blocking uses `delay()` or busy-wait and halts other tasks.

**16. What resolution (bits) does LEDC support?**

LEDC supports configurable resolution up to 20 bits (practical use trades off with achievable frequency).

**17. Compare general-purpose hardware timers and LEDC timers.**

General timers generate interrupts and measure time. LEDC timers generate PWM with channels attached and are specialized for waveform output and resolution.

**18. Difference between Adafruit\_SSD1306 and Adafruit\_GFX.**

Adafruit\_SSD1306: hardware driver for SSD1306 displays (buffering, communication). Adafruit\_GFX: graphics primitives (text, shapes) used by display drivers.

**19. How to optimize text rendering on an OLED?**

Only update changed regions, buffer drawings and call `display()` once per frame, cache glyphs/bitmaps, use smaller fonts, and minimize I2C/SPI transactions.

**20. Short specs of NodeMCU-32S (NodeMCU-32S typical dev board).**

ESP32-WROOM module, dual-core Tensilica LX6 up to 240 MHz, ~520 KB SRAM, commonly 4 MB flash, Wi-Fi & Bluetooth, multiple GPIOs/ADC/DAC/SPI/I2C/UART, 3.3V I/O, USB-Serial interface.

**Question 2 — Logical Questions**

**Q2.1 A 10 kHz signal has an ON time of 10 ms. What is the duty cycle? Justify with the formula.**

Period T = 1 / f. For f = 10 kHz, T = 1/10000 = 0.0001 s = 0.1 ms. Given T\_on = 10 ms, T\_on > T so duty = (T\_on / T) × 100% = 10000% which is impossible. Therefore the statement is inconsistent. If the intended frequency were 10 Hz (T = 100 ms) and T\_on = 10 ms, duty = 10/100 = 10%.

**Q2.2 How many hardware interrupts and timers can be used concurrently? Justify.**

ESP32 supports many interrupt sources: most GPIOs can generate interrupts (edge or level) via the GPIO matrix, so the practical limit is large and determined by software design and IRQ priorities rather than a small fixed number. For hardware timers: there are 4 general-purpose hardware timers (2 groups × 2 timers). Additional peripherals provide timers (LEDC timers up to 8, RTC timers, watchdogs). The number you can use concurrently depends on which timer/peripheral you allocate.

**Q2.3 How many PWM-driven devices can run at distinct frequencies at the same time on ESP32? Explain constraints.**

LEDC provides up to 8 independent timers; each timer defines frequency and resolution and can serve multiple channels (up to 16 channels total). Therefore you can have up to 8 distinct PWM frequencies concurrently. Constraints: limited timers, trade-off between frequency and resolution (higher frequency reduces max resolution), and limited channels/pin availability.

**Q2.4 Compare 30% duty at 8-bit vs 10-bit resolution (1 kHz, all else equal).**

8-bit: max = 255 → 30% ≈ 0.30 × 255 = 76.5 ≈ 77 counts. LSB ≈ 1/255 ≈ 0.39%.  
10-bit: max = 1023 → 30% ≈ 0.30 × 1023 = 306.9 ≈ 307 counts. LSB ≈ 1/1023 ≈ 0.098%.   
Result: nominal duty same, but 10-bit gives finer granularity and lower quantization error (smoother brightness steps).

**Q2.5 How many characters on a 128×64 OLED at once (min vs max font)? State assumptions.**

Assumption : 6×8 pixels per character (5×7 glyph + 1 horizontal spacing). Columns = floor(128 / 6) = 21. Rows = floor(64 / 8) = 8. Total = 21 × 8 = 168 characters.  
If using 8×16 font (common 'large' font): Columns = floor(128 / 8) = 16. Rows = floor(64 / 16) = 4. Total = 16 × 4 = 64 characters.