Name : Rabia Naseer **Reg#: 23-NTU-CS-1275**

Embedded IoT Systems (CSE-3080)

Fall 2025

Assignment 1

Question 1 — Short Questions

1. **Why is volatile used for variables shared with ISRs?**
   The volatile keyword prevents the compiler from performing optimizations that might cache the variable's value in a register. It ensures that every access to the variable is a fresh read from or write to main memory, which is necessary when the variable is modified by an Interrupt Service Routine (ISR) and read by the main program (or vice-versa).

2. **Compare hardware-timer ISR debouncing vs. delay()-based debouncing.**
   Hardware-timer ISR debouncing is non-blocking and precise, allowing the main program loop to execute continuously without interruption. delay()-based debouncing is blocking, causing the CPU to halt all other code execution, which is inefficient and unacceptable for multi-tasking or time-sensitive embedded systems.

Feature	Hardware-Timer ISR	delay()-Based Debouncing
Timing Accuracy	High	Low
CPU Blocking	No	Yes
Complexity	Moderate to High	Low
Responsiveness	High	Low
Scalability	Good	Poor
Best Use Case	Real time systems	Simple Projects

3. **What does IRAM_ATTR do, and why is it needed?**
   IRAM_ATTR is a macro (on ESP32) that forces a function or data to be placed in the **Instruction RAM (IRAM)**. It is needed because ISRs must reside in IRAM to execute quickly and consistently, especially during flash/cache operations (like Wi-Fi access) that might otherwise stall code running from flash.

4. **Define LEDC channels, timers, and duty cycle.**
   LEDC Channels are 16 independent output generators for PWM signals. **LEDC Timers** are 4 shared clock generators that determine the **frequency** and **resolution** for one or more channels.

Duty Cycle is the ratio of the time a PWM signal is ON (high) to the total period, usually expressed as a percentage.

5. **Why should you avoid Serial prints or long code paths inside ISRs?**
Serial communication is **slow** and often involves **blocking** waits or complex I/O operations. Long code paths or Serial.print will cause the ISR to take too long, potentially leading to the loss of subsequent interrupts, timing inaccuracies in the main program, or system crashes (like a Watchdog Timer timeout).

6. **What are the advantages of timer-based task scheduling?**
It provides **non-blocking** and **highly accurate periodic timing** for tasks. This allows the CPU to manage multiple time-critical events or background tasks concurrently without relying on software delays or complex state machine logic in the main loop.

7. **Describe I²C signals SDA and SCL.**
SDA (Serial Data) is the bidirectional line used for transmitting and receiving data. **SCL (Serial Clock)** is the signal line driven by the Master device to synchronize the data transfer between the Master and Slave devices.

Signal	Role	Direction	Driven By	Electrical Type	Notes
SDA	Data transfer	Bi-directional	Master/Slave	Open- drain	Carries addresses and data
SCL	Clock signal	Master only	Master	Open- drain	Synchronizes data transmission

8. **What is the difference between polling and interrupt-driven input?**
Polling involves the CPU constantly checking the state of an input in the main loop, which is inefficient. **Interrupt-driven** input configures the hardware to automatically pause the main program and execute a dedicated Interrupt Service Routine (ISR) only when an input event (e.g., a button press) occurs, making it highly efficient.

Feature	Polling	Interrupt-Driven Input
CPU Usage	High (constant checking)	Low (event-triggered)
Responsiveness	Lower	Higher
Complexity	Simple	Moderate to high
Power Efficiency	Poor	Good
Best Use Case	Simple, infrequent events	Real-time, frequent events

9. **What is contact bounce, and why must it be handled?**
   Contact bounce is the physical phenomenon where the metal contacts of a mechanical switch vibrate and rapidly make and break connection for a brief period (typically 5-15 ms) upon closing or opening. It must be handled because the microcontroller will register each bounce as a **separate, false input event**.

Problem Caused by Bounce	Solution
Multiple false triggers	Debounce with delay or timer
Unstable input readings	Use filtering or logic gates
Software misbehavior	Implement input validation

10. **How does the LEDC peripheral improve PWM precision?**
    The ESP32's LEDC peripheral uses dedicated hardware and supports an exceptionally high duty resolution (up to **20 bits**) driven by a high-speed clock (80 MHz). This high resolution offers a significantly greater number of distinct brightness steps compared to standard 8bit PWM, resulting in much smoother and more precise control, especially for dimming LEDs.

Feature	Benefit
High resolution	Fine control over duty cycle
Hardware timers	Stable, jitter-free PWM
Multiple channels	Flexible multi-output control
Fade support	Smooth transitions, no CPU load
Dual-speed modes	Precision vs. power trade-off

11. **How many hardware timers are available on the ESP32?**
    The ESP32 has **four** general-purpose 64-bit hardware timers (two in each of the two timer groups). Additionally, it has **four dedicated LEDC (PWM) timers** and a separate Real-Time Clock (RTC) timer.

12. **What is a timer prescaler , and why is it used?**
    A **prescaler** is a hardware register that divides the high-frequency clock source before it reaches the main timer/counter logic. It is used to **reduce the effective clock speed** of the timer, which increases the time between counts and allows the timer to generate much longer time intervals (slower frequencies) than would otherwise be possible.

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

Frequency is the rate at which the PWM signal completes one full cycle (period,), measured in Hertz (Hz). **Duty Cycle** is the proportion of one period during which the signal is active (ON or HIGH), typically expressed as a percentage.

$$D = T\_(ON\ )/(T * 100\%)$$

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

Since human eyes perceive brightness non-linearly (logarithmically), a linear increase in PWM duty cycle will not result in a linear perceived brightness increase. To achieve a smooth, linear perceived brightness (gamma correction), the duty cycle must be calculated using a power law on the desired brightness value, such as:

$$〚Duty〛\ \_value \propto\ 〚Brightness〛\ \_(Perceived\hat{}')\hat{}\gamma\ \ Where\ \gamma = 2$$

15. Contrast non-blocking vs. blocking timing.

Blocking timing (using delay()) stops all code execution for a set duration. **Non-blocking timing** (checking millis() differences) allows the code to continue executing sequentially, managing time-based events without pauses, which is crucial for handling multiple concurrent tasks efficiently.

Feature	Blocking Timing (`delay()`)	Non-Blocking Timing (`millis()`)
CPU Usage	Idle during delay	Active and multitasking
Responsiveness	Poor	High
Complexity	Low	Moderate
Best Use Case	Simple, single-task logic	Real-time, multitasking systems

16. What resolution (bits) does LEDC support?

The ESP32's LEDC peripheral officially supports up to **20 bits** of duty resolution, although common practice and the Arduino framework often use 10 to 16 bits, with the practical maximum resolution being interdependent with the configured PWM frequency.

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

General-Purpose Hardware Timers are used for general timekeeping, periodic task scheduling, and generating predictable interrupts. **LEDC (PWM) Timers** are specifically designed to set the frequency and resolution for the 16 dedicated PWM output channels, driving devices like LEDs or motors.

Feature	General-Purpose Hardware Timers	LEDC (PWM) Timers
Primary Use	Scheduling tasks, delays, timeouts	Generating PWM signals
Timer Groups	2 groups (TIMER_GROUP_0 & 1), 2 timers each	4 timers shared across 8 PWM channels
Resolution	Up to 64-bit counters	Up to 20-bit resolution for PWM duty
Prescaler Support	Yes (fine-grained control over tick rate)	Yes (sets PWM frequency)
Interrupt Capability	Yes (can trigger ISRs)	No direct ISR; PWM output only
Auto-Reload	Supported	Not applicable (uses duty cycle updates)
Duty Cycle Control	Manual (via ISR or software logic)	Built-in hardware support
Fade Functionality	Not available	Supported (smooth duty transitions)
Clock Source	APB or RTC	APB (high-speed) or RTC (low-speed)
Typical Use Cases	Task scheduling, input capture, alarms	LED dimming, motor control, audio PWM

18. What is the difference between Adafruit_SSD1306 and Adafruit_GFX?

Adafruit_GFX is the **base graphics library** that provides core drawing functions (lines, circles, text). Adafruit_SSD1306 is a **hardware-specific library** that inherits from Adafruit_GFX and implements the low-level functions required to communicate with and control the specific SSD1306 OLED display controller chip.

Library	Role	Functionality	Dependency Relationship
Adafruit_SSD1306	Device-specific driver	Controls SSD1306 OLED displays (I^2C/SPI	Depends on Adafruit_GFX
Adafruit_GFX	Graphics rendering engine	Provides drawing primitives (text, shapes)	Used by many display drivers

19. **How can you optimize text rendering performance on an OLED?**
    Optimization focuses on minimizing the slow I²C/SPI data transfer. Methods include: 1) **Updating only changed areas** of the screen instead of the whole framebuffer (if supported). 2) Using the **smallest font size** necessary to reduce the data footprint. 3) Utilizing the fastest possible **I²C/SPI clock speed**.

20. **Give short specifications of your selected ESP32 board (NodeMCU-32S).**
    The NodeMCU-32S is a development board based on the **ESP32-WROOM-32** module. It features a **dual-core 32-bit Xtensa LX6 CPU** running up to 240 MHz, **4 MB SPI Flash** memory, integrated **Wi-Fi** (802.11 b/g/n), and **Bluetooth 4.2** (BLE/BR/EDR). It exposes numerous peripherals, including **16 PWM channels** and multiple ADC/I²C/SPI interfaces.

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.**

   This scenario describes an impossible physical signal, as the ON time is longer than the entire period of the signal.

   a. **Calculate the Period (T):**

   $$T = 1/Frequency(f) = 1/10000Hz = 0.0001s = 0.1ms$$

   b. **Calculate the Duty Cycle (D) using the formula:**

   $$Duty\ Cycle\ (D) = 〚ON\ Time\ \ (T〛\ \_(ON\ )\ )/Period(T) * 100\%$$
   $$= (10ms/0.1ms) * 100\% = 100 * 100\% = 10{,}000\%$$

   Justification:

   A valid duty cycle must be between 0% and 100%. A 10 kHz signal has a period of 0.1ms. If the ON time is 10ms, the signal is "ON" for 10ms / 0.1ms = 100 complete cycles. For a continuous PWM signal, the ON time must be less than or equal to the period (T_ON ≤ T). Therefore, the calculated 10,000% is mathematically correct for the numbers provided, but the physical signal is invalid.

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

Answer and Justification:

Hardware Timers:

Four 64-bit general-purpose hardware timers can be used concurrently.

Justification:

The ESP32 has two separate Timer Groups (T0 and T1), and each group contains 2 independent 64-bit timers, resulting in four timers that can each run a different periodic task or measure time. (Note: Four additional LEDC timers and an RTC timer are also available, depending on the definition of "hardware timer").

Hardware Interrupts:

Almost all 34 programmable GPIOs can be configured as external interrupt sources, and many dozens of internal peripheral interrupts (for UART, SPI, etc.) are available and can be used concurrently.

Justification:

The ESP32's External Interrupt Controller (EIC) allows nearly every GPIO pin to be mapped to one of the 32 available hardware interrupt lines. In practice, the limiting factor is generally the overhead of running and exiting the numerous Interrupt Service Routines (ISRs) in software, not the hardware capacity itself.

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

Answer and Constraints:

The ESP32 can run a maximum of four PWM-driven devices at distinct frequencies concurrently.

Constraints Explanation:

- The ESP32's LEDC peripheral provides 16 independent output channels for PWM signals.
- However, these 16 channels share only four independent LEDC timers.
- All channels that are assigned to the *same* timer are constrained to share the same clock source, frequency, and resolution. Therefore, to achieve four *distinct* frequencies, one channel must be attached to each of the four available LEDC timers.

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).**

Feature	**30% Duty Cycle at 8-bit Resolution**	**30% Duty Cycle at 10bit Resolution**
Duty Value (Steps)	0.30 * (2^8 - 1) = 0.30 * 255 ~ 77	0.30 * (2^10 - 1) = 0.30 * 1023 ~ 307
Total Steps	2^8 = 256 discrete steps	2^10 = 1024 discrete steps
Precision	$1/256 \sim 0.39\%\ per\ step$	$1/1024{\sim}0.098\%\ per\ step$

Conclusion:

Both signals achieve the same 30% average power/brightness because the duty cycle percentage is identical. The key difference is the precision (or granularity). The 10-bit resolution signal has four times as many steps (1024 vs. 256), offering a much finer control over the duty cycle. This results in smoother transitions, which is important for fading and dimming effects.

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.**

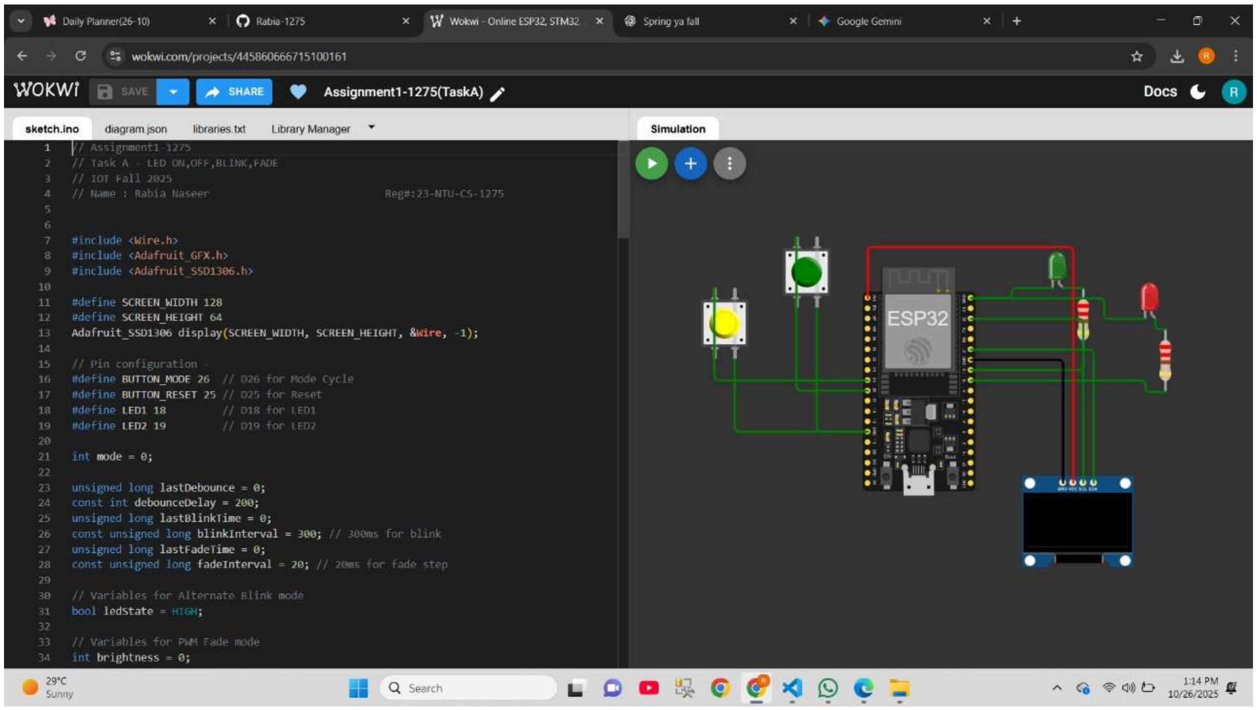
This calculation is based on the following standard assumptions for the Adafruit_GFX library using a 128 * 64 pixel SSD1306 OLED:

- Standard font character size (Size 1) is 5 * 7 pixels, often approximated as 5 * 8 for simplicity, with 1-pixel spacing.
- "Minimum font size" is **Size 1** (5 * 8 px per character with spacing).
- "Maximum font size" is **Size 2 (**10 * 16 px per character with spacing).

Font Size	**Character Dimensions (W x H)**	**Total Footprint (with 1px Spacing)**	**Characters per Row (128 / W_Total)**	**Total Rows (64 / H_Total)**	**Total Characters**
Minimum (size 1)	5px * 8px	6px * 9px	[128/6] = 21	[64/9] = 7	147
Maximum (size 2)	10px * 16px	11px * 17px	[128/11] = 11	[64/17] = 3	33

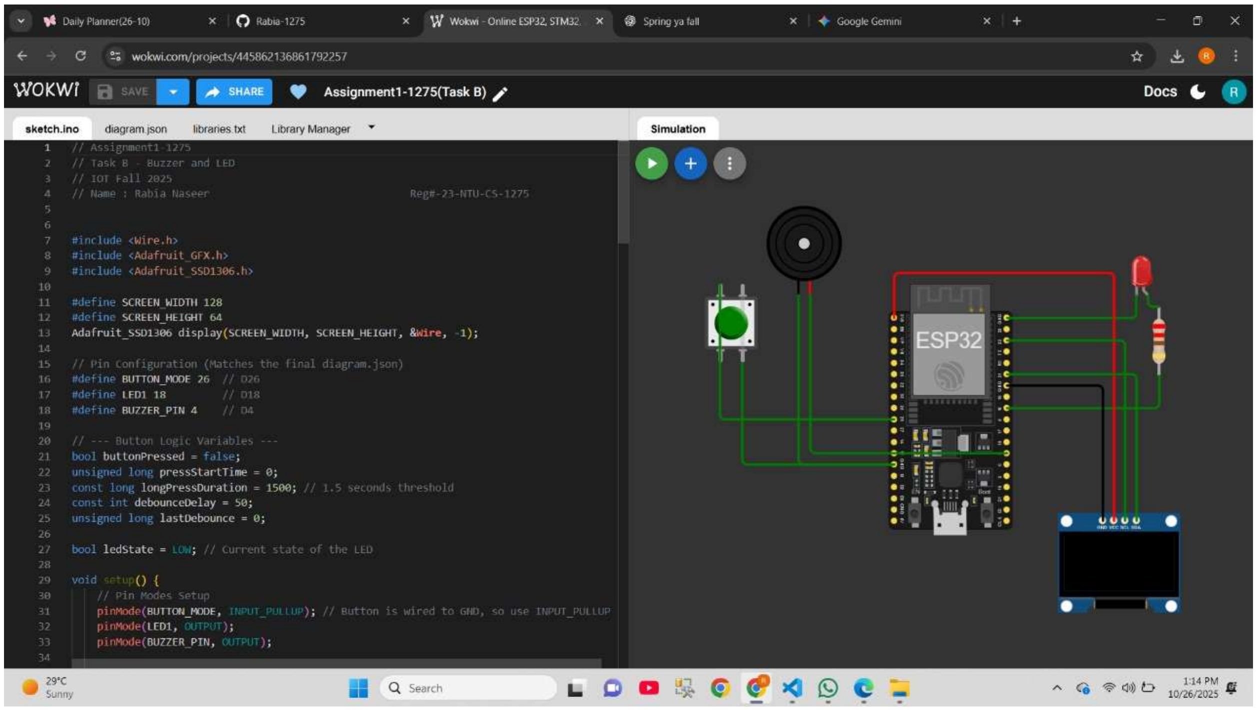
Question 3:

Task A:



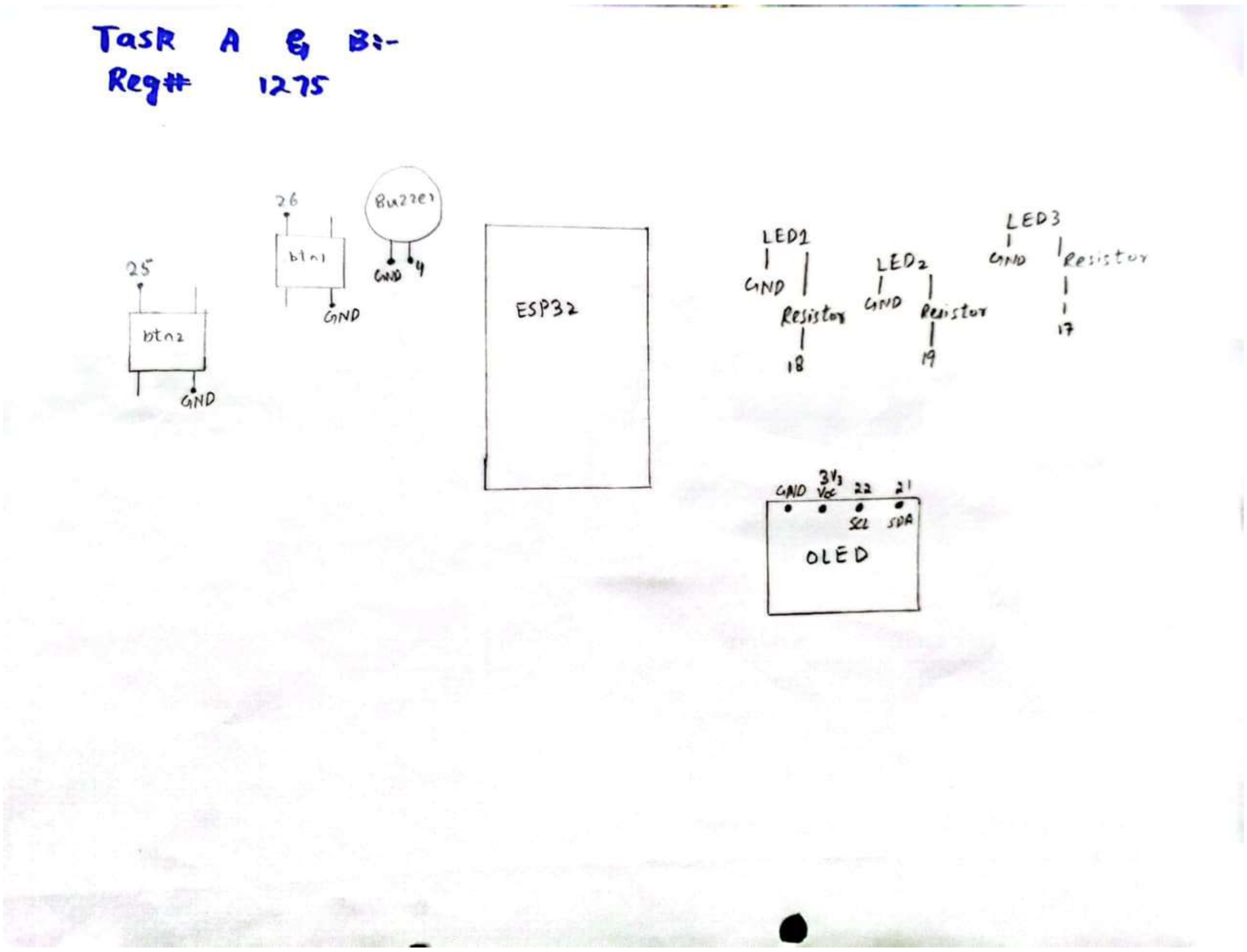
```
// Assignment1-1275
// Task A - LED ON,OFF,BLINK,FADE
// IOT Fall 2025
// Name : Rabia Naseer                           Reg#:23-NTU-CS-1275


#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Pin configuration -
#define BUTTON_MODE 26  // D26 for Mode Cycle
#define BUTTON_RESET 25 // D25 for Reset
#define LED1 18         // D18 for LED1
#define LED2 19         // D19 for LED2

int mode = 0;

unsigned long lastDebounce = 0;
const int debounceDelay = 200;
unsigned long lastBlinkTime = 0;
const unsigned long blinkInterval = 300; // 300ms for blink
unsigned long lastFadeTime = 0;
const unsigned long fadeInterval = 20; // 20ms for fade step

// Variables for Alternate Blink mode
bool ledState = HIGH;

// Variables for PWM Fade mode
int brightness = 0;
```

Task B:



```
// Assignment1-1275
// Task B - Buzzer and LED
// IOT Fall 2025
// Name : Rabia Naseer                               Reg#-23-NTU-CS-1275


#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Pin Configuration (Matches the final diagram.json)
#define BUTTON_MODE 26  // D26
#define LED1 18         // D18
#define BUZZER_PIN 4    // D4

// --- Button Logic Variables ---
bool buttonPressed = false;
unsigned long pressStartTime = 0;
const long longPressDuration = 1500; // 1.5 seconds threshold
const int debounceDelay = 50;
unsigned long lastDebounce = 0;

bool ledState = LOW; // Current state of the LED

void setup() {
    // Pin Modes Setup
    pinMode(BUTTON_MODE, INPUT_PULLUP); // Button is wired to GND, so use INPUT_PULLUP
    pinMode(LED1, OUTPUT);
    pinMode(BUZZER_PIN, OUTPUT);

```

Hand-Sketch:


Task A & B:-
Reg# 1275
25
btn2
GND
26
btn1
GND
Buzzer
GND
4
ESP32
LED1
GND
Resistor
18
LED2
GND
Resistor
19
LED3
GND
Resistor
17
GND
3V3
Vcc
22
SCL
21
SDA
OLED