

National Textile University, Faisalabad



Name:	Fiza Mahboob
Class:	BSCS(A)
Semester:	5 th -CS-A
Registration No:	23-NTU-CS-1027
Course Name:	Embedded IoT System
Submitted To:	Sir Nasir Mahmood
Submission Date:	19 Oct,2025

Assignment 1

Question 1 — Short Questions

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

The volatile keyword tells the compiler that a variable's value can change at any time, from outside the main program flow (like from an Interrupt Service Routine). Without it, the compiler might optimize the code by using a cached, outdated value, not realizing it was updated by an ISR.

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

Timer ISR Debouncing: Uses a hardware timer to check the button's state after a set time has passed. It's non-blocking, accurate, and allows the main code to run smoothly while waiting.

delay()-based Debouncing: Uses delay() to pause the entire program for a few milliseconds. It's simple but blocks all other code execution, making the system unresponsive.

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

IRAM_ATTR forces a function (like an ISR) to be loaded into the ESP32's fast Internal RAM. This is needed because if the ISR code is in slow external flash memory and an interrupt occurs, the processor might not be able to fetch the code quickly enough, leading to crashes or missed interrupts.

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

Channels: These are the pathways that carry the PWM signal to a specific GPIO pin. The ESP32 has 16 independent channels.

Timers: These are the engines that generate the base frequency and resolution for the PWM signal. Multiple channels can share one timer.

Duty Cycle: This is the fraction of time the signal is "ON" (high) compared to the total period of one cycle, expressed as a percentage. It controls the average power delivered.

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

ISRs should be as fast as possible because they interrupt the main code. Serial.print() is very slow and can cause the ISR to run for too long, which can make the system unresponsive, cause other interrupts to be missed, or lead to watchdog timer resets.

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

It allows you to run tasks at very precise and regular intervals without blocking the main loop. This makes the system more responsive and reliable, as you can guarantee that critical tasks (like reading a sensor) happen exactly when they are supposed to.

7. Describe I²C signals SDA and SCL.

SDA (Serial Data): This is the line that carries the actual data between devices.

SCL (Serial Clock): This is the line that carries the clock signal, which synchronizes the data transfer. The master device generates the clock.

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

Polling: The microcontroller continuously and repeatedly checks the state of a pin in the main loop. It's simple but inefficient and can miss short events.

Interrupt-driven: The microcontroller does nothing until a specific change (like a button press) occurs on the pin. When it does, the main loop is paused, and an ISR is run immediately. This is efficient and responsive.

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

When a mechanical button or switch is pressed, its metal contacts don't make a clean connection instantly. They physically "bounce" against each other for a few milliseconds, causing the microcontroller to see multiple rapid presses instead of one. It must be handled so that a single press is registered as a single, clean event.

10. How does the LEDC peripheral improve PWM precision?

The LEDC is a dedicated hardware module on the ESP32 specifically designed for PWM. It uses hardware timers with high resolution, giving smoother and more accurate PWM signals.

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

The ESP32 has 4 hardware timers in total. These are separate from the 8 timers inside the LEDC (PWM) module.

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

A prescaler is a divider that reduces the frequency of the main clock signal before it reaches the timer. It's used to slow down the timer's counting speed, allowing us to work with lower, more practical frequencies and longer time intervals.

13. Define duty cycle and frequency in PWM.

Duty cycle: How long the signal stays HIGH per cycle (in %).

Frequency: How fast the signal completes one full ON/OFF cycle.

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

You calculate the duty value based on the PWM resolution:

$$\text{duty} = (\text{brightness} / \text{max_brightness}) \times \text{max_duty_value}$$

15. Contrast non-blocking vs. blocking timing.

Blocking Timing (using delay()): The entire program stops and waits. Nothing else can happen during the delay.

Non-Blocking Timing (using millis() or timers): The code checks if enough time has passed without stopping. The microcontroller can execute other tasks while waiting.

16. What resolution (bits) does LEDC support?

LEDC supports **1 to 20 bits** resolution (commonly 8, 10, 12, or 13 bits).

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

General-Purpose Timers: Used for precise timing of events, generating interrupts, or measuring time. Their main output is an "interrupt signal."

LEDC Timers: A specialized part of the LEDC peripheral whose main job is to generate the base frequency and resolution for PWM waveforms. Their output is a "PWM signal."

18. What is the difference between Adafruit_SSD1306 and Adafruit_GFX?

Adafruit_SSD1306: This is the "driver" library. It knows how to talk specifically to the SSD1306 OLED chip—sending commands and data to control the hardware.

Adafruit_GFX: This is the "graphics" library. It contains all the standard functions for drawing shapes, text, and bitmaps, but it doesn't know how to talk to any specific screen. The SSD1306 library uses the GFX library to actually draw on the screen.

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

By updating only changed areas, using smaller fonts, or avoiding full-screen clears every frame.

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

Dual-core 32-bit Xtensa LX6, up to 240 MHz

Wi-Fi + Bluetooth (BLE + Classic)

520 KB SRAM, 4 MB Flash

30 GPIO pins

3.3 V operating voltage

Supports UART, SPI, I²C, ADC, DAC, PWM

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.

First, we need to find the period of one cycle (T).

- Frequency (f) = 10 kHz = 10,000 Hz
- Period (T) = $1 / f = 1 / 10,000 = 0.0001$ seconds = 0.1 ms

The ON time is given as 10 ms.

Justification using the duty cycle formula:

- Duty Cycle (D) = (ON Time / Total Period) * 100%
- $D = (10 \text{ ms} / 0.1 \text{ ms}) * 100\%$
- $D = 100 * 100\% = 10,000\%$

This result of 10,000% is impossible, as duty cycle cannot exceed 100%. The error lies in the mismatch of units. The ON time (10 ms) is **longer** than the total period of the signal (0.1 ms). This is not physically possible for a single pulse.

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

Hardware Interrupts: All GPIO pins (GPIO 0-39) can be used concurrently as hardware interrupt pins. As we learned every GPIO pin supports external interrupts. The constraint is not the number of pins, but how you manage the code in the Interrupt Service Routines (ISRs) to avoid conflicts and performance issues.

General-Purpose Hardware Timers: The ESP32 has **4 general-purpose hardware timers** that can be used concurrently. These are separate from the PWM timers and are used for precise timing and generating timer interrupts

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

On LEDC, the ESP32 has **8 PWM timers (Timer 0 to Timer 7)**.

Maximum Distinct Frequencies: Therefore, you can have up to **8 PWM-driven devices** running at 8 completely different frequencies at the same time.

Constraint: The 16 PWM channels must share these 8 timers. If two channels are attached to the same timer, they are forced to share the same frequency and resolution, but can have different

duty cycles. To get a unique frequency, a device must be on a channel that uses its own dedicated timer.

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).

The key difference is in the **precision and smoothness** of the control.

8-bit Resolution: The duty cycle can be set in 256 steps (0 to 255). A 30% duty cycle corresponds to a value of $0.30 * 255 = 76.5$. Since we can only use whole numbers, we have to choose either 76 or 77. This gives us an **actual duty cycle** of either 29.8% or 30.2%. There is a small error.

10-bit Resolution: The duty cycle can be set in 1024 steps (0 to 1023). A 30% duty cycle corresponds to a value of $0.30 * 1023 = 306.9$. We can choose 307, which gives an **actual duty cycle** of 30.01%. This is much closer to the desired 30%.

Conclusion: The 10-bit resolution allows for a more accurate and finer control of the 30% duty cycle compared to 8-bit.

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.

Assumptions:

- We are using the Adafruit_GFX library's built-in default font (which is 5x7 pixels per character, as used in `setTextSize(1)`).
- We assume one line of text has a height of 8 pixels (7 for the character + 1 for spacing).
- We are only considering the visible area and simple text rendering.

Calculations:

- **Minimum Font Size (`setTextSize(1)`):**
 - Character Width: ~6 pixels (5 pixels + 1 pixel spacing)
 - Characters per line: $128 / 6 \approx \mathbf{21 \text{ characters}}$
 - Number of lines: $64 / 8 = \mathbf{8 \text{ lines}}$
 - **Total Characters:** 21 chars/line * 8 lines = **~168 characters**
- **Maximum Font Size (Let's assume `setTextSize(4)`):**
 - The `setTextSize(n)` multiplies the default 5x7 font by 'n'.

- At size 4, a character is 20x28 pixels.
- Character Width: ~24 pixels (20 + spacing)
- Characters per line: $128 / 24 \approx \mathbf{5 \text{ characters}}$
- Number of lines: $64 / 28 \approx \mathbf{2 \text{ lines}}$
- **Total Characters:** 5 chars/line * 2 lines = **~10 characters**

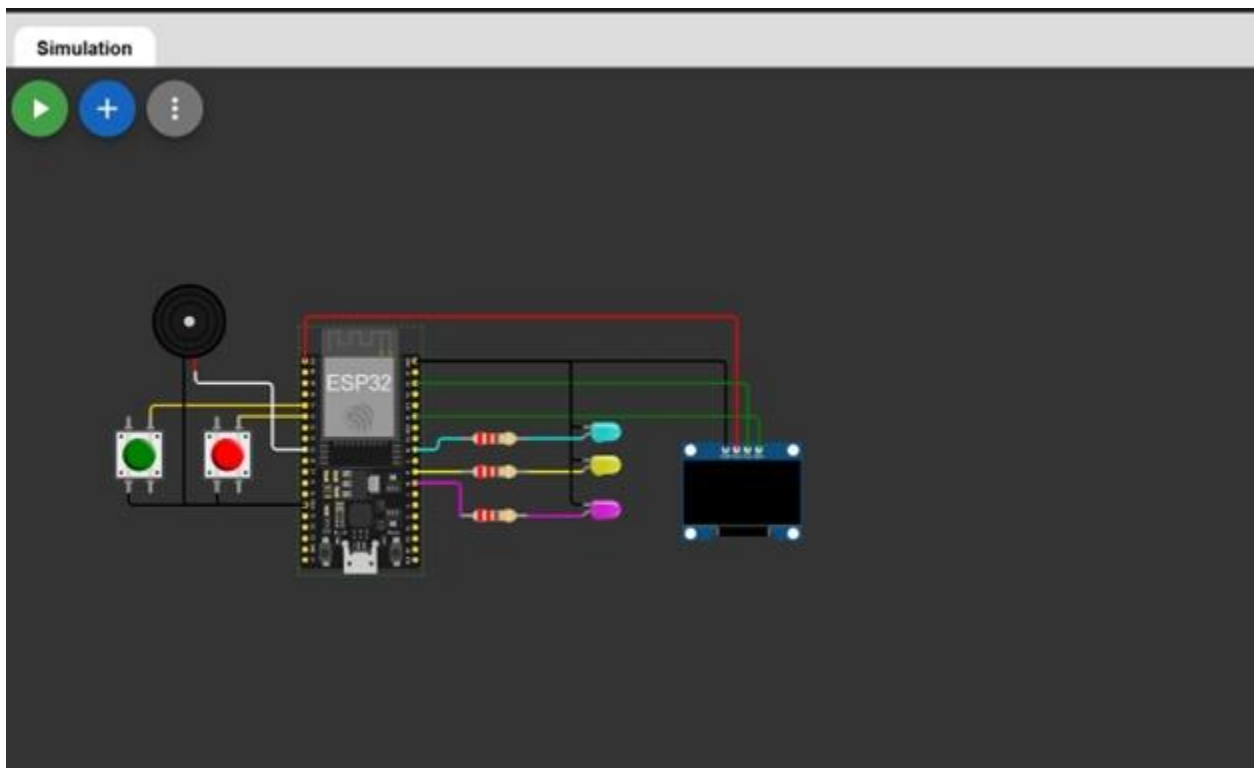
Conclusion: With the smallest font, you can fit approximately 168 characters. With a very large font, you might only fit around 10 characters.

Question 3 — Implementation

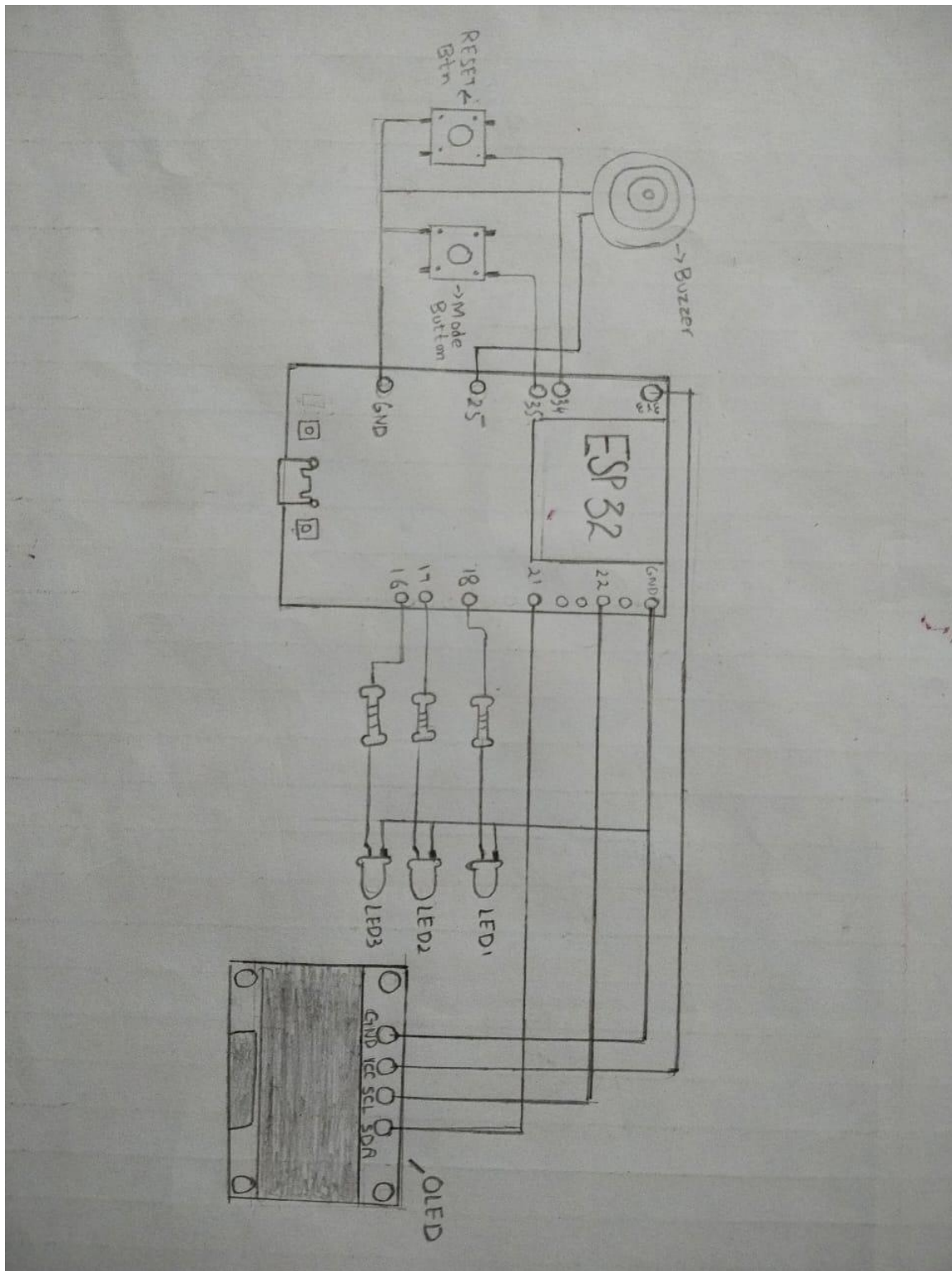
Circuit Diagram: Design a Wokwi circuit and draw a neat hand-sketch including:

- 2 push buttons
- 3 LEDs
- 1 buzzer
- 1 OLED

Circuit Diagram:



Hand Sketch:



Task A — Coding: Use one button to cycle through LED modes (display the current state on the OLED):

1. Both OFF
2. Alternate blink
3. Both ON
4. PWM fade

Use the second button to reset to OFF.

Source Code:

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// ----- OLED Configuration -----
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_ADDR 0x3C
// ----- Pin Definitions -----
const int LED1_PIN = 18;
const int LED2_PIN = 17;
const int LED3_PIN = 16;
const int BTN_MODE = 34;
const int BTN_RESET = 35;
const int BUZZER_PIN = 25;
// ----- Debounce Configuration -----
const unsigned long DEBOUNCE_DELAY = 25; // Debounce time in ms
// OLED display object
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
// ----- Operating Modes -----
enum Mode { BOTH_OFF, ALT_BLINK, BOTH_ON, PWM_FADE };
Mode currentMode = BOTH_OFF;
// ----- Timing Variables -----
unsigned long lastToggle = 0; // Tracks LED blink timing
const unsigned long blinkInterval = 400; // Time interval for alternating blink
// ----- Button State Tracking -----
unsigned long lastModePress = 0;
unsigned long lastResetPress = 0;
int lastModeState = HIGH;
int lastResetState = HIGH;
bool modePressed = false;
bool resetPressed = false;
// ----- Fade Control Variables -----
int fadeVal = 0;
int fadeDir = 1;
// ----- Function Prototypes -----
void setLedsToOff();
void setLedsToHigh();
void updateOLED(const char *msg);
```

```

void handleModeChange();
void setup() {
  Serial.begin(115200);
  // Configure pin modes
  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  pinMode(LED3_PIN, OUTPUT);
  pinMode(BTN_MODE, INPUT_PULLUP);
  pinMode(BTN_RESET, INPUT_PULLUP);
  pinMode(BUZZER_PIN, OUTPUT);
  // Initialize I2C and OLED
  Wire.begin(21, 22);
  if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR)) {
    Serial.println("OLED failed!");
    for (;;); // Stop execution if OLED setup fails
  }
  // Display initial mode
  updateOLED("BOTH OFF");
  setLedsToOff();
  Serial.println("Setup complete");
}
void loop() {
  unsigned long now = millis();
  // ----- MODE Button Handler -----
  int readingMode = digitalRead(BTN_MODE);
  // Detect change in state
  if (readingMode != lastModeState) {
    lastModePress = now;
  }
  // Debounce check
  if ((now - lastModePress) > DEBOUNCE_DELAY) {
    if (readingMode == LOW && !modePressed) {
      modePressed = true;
      digitalWrite(LED3_PIN, HIGH);
      handleModeChange();
      Serial.println("Mode button pressed");
    }
    else if (readingMode == HIGH) {
      modePressed = false;
      digitalWrite(LED3_PIN, LOW);
    }
  }
  lastModeState = readingMode;
  // ----- RESET Button Handler -----
  int readingReset = digitalRead(BTN_RESET);
  if (readingReset != lastResetState) {
    lastResetPress = now;
  }
  if ((now - lastResetPress) > DEBOUNCE_DELAY) {
    if (readingReset == LOW && !resetPressed) {
      resetPressed = true;
      digitalWrite(LED3_PIN, HIGH);
      currentMode = BOTH_OFF;
      updateOLED("BOTH OFF");
      setLedsToOff();
      Serial.println("Reset button pressed");
    }
  }
}

```

```

    }
    else if (readingReset == HIGH) {
        resetPressed = false;
        digitalWrite(LED3_PIN, LOW);
    }
}
lastResetState = readingReset;
// ----- LED behavior based on Mode -----
switch (currentMode) {
    case BOTH_OFF:
        break;
    case ALT_BLINK:
        // Alternate LED blinking
        if (now - lastToggle >= blinkInterval) {
            lastToggle = now;
            static bool toggle = false;
            toggle = !toggle;
            analogWrite(LED1_PIN, toggle ? 255 : 0);
            analogWrite(LED2_PIN, toggle ? 0 : 255);
            digitalWrite(LED3_PIN, LOW);
        }
        break;
    case BOTH_ON:
        // Both LEDs stay continuously on
        break;
    case PWM_FADE:
        // Smooth fading of both LEDs using PWM
        fadeVal += fadeDir * 1;
        if (fadeVal >= 180) { fadeVal = 180; fadeDir = -1; }
        if (fadeVal <= 0) { fadeVal = 0; fadeDir = 1; }
        analogWrite(LED1_PIN, fadeVal);
        analogWrite(LED2_PIN, fadeVal);
        digitalWrite(LED3_PIN, LOW);

        delay(20);
        break;
}
}
// ----- FUNCTION DEFINITIONS -----
// Turns off all LEDs
void setLedsToOff() {
    analogWrite(LED1_PIN, 0);
    analogWrite(LED2_PIN, 0);
    digitalWrite(LED3_PIN, LOW);
}
// Turns on all LEDs at full brightness
void setLedsToHigh() {
    analogWrite(LED1_PIN, 255);
    analogWrite(LED2_PIN, 255);
    digitalWrite(LED3_PIN, HIGH);
}
// Handles switching between LED operation modes
void handleModeChange() {
    currentMode = (Mode)((currentMode + 1) % 4);

    switch (currentMode) {

```

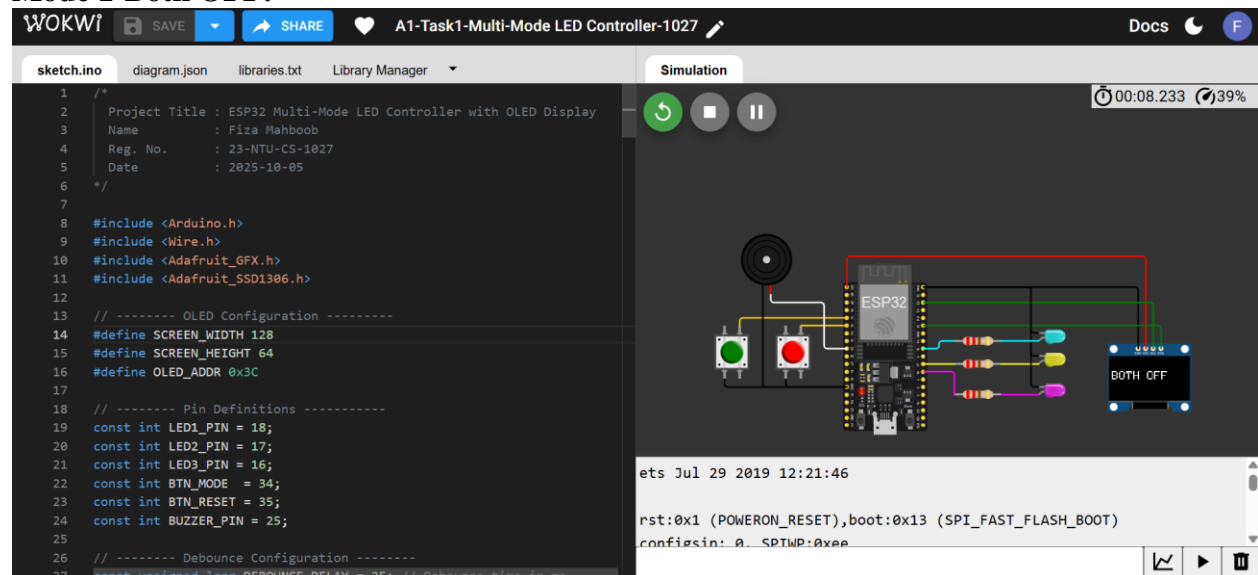
```

case BOTH_OFF:
    updateOLED("BOTH OFF");
    setLedsToOff();
    break;
case ALT_BLINK:
    updateOLED("ALT BLINK");
    setLedsToOff();
    lastToggle = millis();
    break;
case BOTH_ON:
    updateOLED("BOTH ON");
    setLedsToHigh();
    break;
case PWM_FADE:
    updateOLED("PWM FADE");
    fadeVal = 0;
    fadeDir = 1;
    break;
}
}
// Updates the OLED display with current mode information
void updateOLED(const char *msg) {
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 20);
    display.println(msg);
    display.display();
}

```

Output Screenshot

Mode 1-Both OFF:



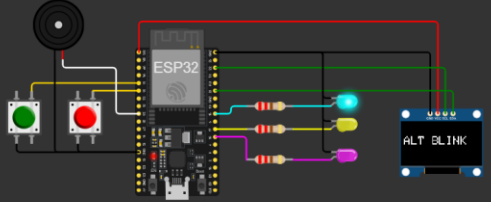
Mode 2- Alternate blink: LED1-ON

WOKWI SAVE SHARE A1-Task1-Multi-Mode LED Controller-1027 Docs F

sketch.ino diagram.json libraries.txt Library Manager

```
1 /*
2  Project Title : ESP32 Multi-Mode LED Controller with OLED Display
3  Name : Fiza Mahboob
4  Reg. No. : 23-NTU-CS-1027
5  Date : 2025-10-05
6 */
7
8 #include <Arduino.h>
9 #include <Wire.h>
10 #include <Adafruit_GFX.h>
11 #include <Adafruit_SSD1306.h>
12
13 // ----- OLED Configuration -----
14 #define SCREEN_WIDTH 128
15 #define SCREEN_HEIGHT 64
16 #define OLED_ADDR 0x3C
17
18 // ----- Pin Definitions -----
19 const int LED1_PIN = 18;
20 const int LED2_PIN = 17;
21 const int LED3_PIN = 16;
22 const int BTN_MODE = 34;
23 const int BTN_RESET = 35;
24 const int BUZZER_PIN = 25;
25
26 // ----- Debounce Configuration -----
```

Simulation 00:28.166 39%



ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

config:pin:0_SPTWP:0xee

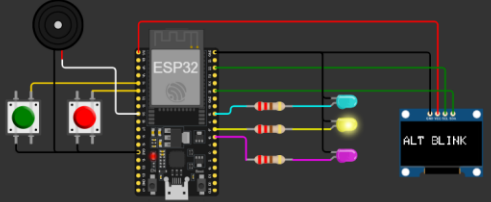
LED2-ON

WOKWI SAVE SHARE A1-Task1-Multi-Mode LED Controller-1027 Docs F

sketch.ino diagram.json libraries.txt Library Manager

```
1 /*
2  Project Title : ESP32 Multi-Mode LED Controller with OLED Display
3  Name : Fiza Mahboob
4  Reg. No. : 23-NTU-CS-1027
5  Date : 2025-10-05
6 */
7
8 #include <Arduino.h>
9 #include <Wire.h>
10 #include <Adafruit_GFX.h>
11 #include <Adafruit_SSD1306.h>
12
13 // ----- OLED Configuration -----
14 #define SCREEN_WIDTH 128
15 #define SCREEN_HEIGHT 64
16 #define OLED_ADDR 0x3C
17
18 // ----- Pin Definitions -----
19 const int LED1_PIN = 18;
20 const int LED2_PIN = 17;
21 const int LED3_PIN = 16;
22 const int BTN_MODE = 34;
23 const int BTN_RESET = 35;
24 const int BUZZER_PIN = 25;
25
26 // ----- Debounce Configuration -----
```

Simulation 00:22.233 40%



ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

config:pin:0_SPTWP:0xee

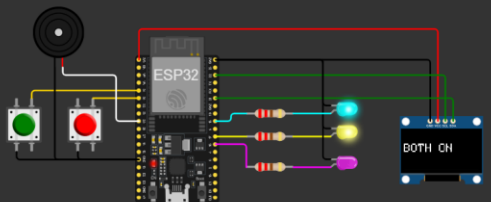
Mode3-Both ON:

WOKWI SAVE SHARE A1-Task1-Multi-Mode LED Controller-1027 Docs F

sketch.ino diagram.json libraries.txt Library Manager

```
1 /*
2  Project Title : ESP32 Multi-Mode LED Controller with OLED Display
3  Name : Fiza Mahboob
4  Reg. No. : 23-NTU-CS-1027
5  Date : 2025-10-05
6 */
7
8 #include <Arduino.h>
9 #include <Wire.h>
10 #include <Adafruit_GFX.h>
11 #include <Adafruit_SSD1306.h>
12
13 // ----- OLED Configuration -----
14 #define SCREEN_WIDTH 128
15 #define SCREEN_HEIGHT 64
16 #define OLED_ADDR 0x3C
17
18 // ----- Pin Definitions -----
19 const int LED1_PIN = 18;
20 const int LED2_PIN = 17;
21 const int LED3_PIN = 16;
22 const int BTN_MODE = 34;
23 const int BTN_RESET = 35;
24 const int BUZZER_PIN = 25;
25
26 // ----- Debounce Configuration -----
```

Simulation 00:35.366 41%



ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

config:pin:0_SPTWP:0xee

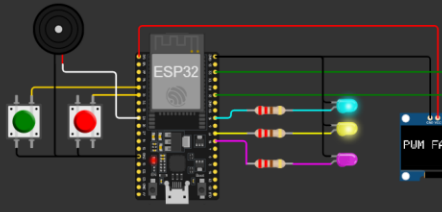
Mode4-PWM Fade

WOKWI SAVE SHARE A1-Task1-Multi-Mode LED Controller-1027 Docs F

sketch.ino diagram.json libraries.txt Library Manager

```
1 /*
2  Project Title : ESP32 Multi-Mode LED Controller with OLED Display
3  Name : Fiza Mahboob
4  Reg. No. : 23-NTU-CS-1027
5  Date : 2025-10-05
6 */
7
8 #include <Arduino.h>
9 #include <Wire.h>
10 #include <Adafruit_GFX.h>
11 #include <Adafruit_SSD1306.h>
12
13 // ----- OLED Configuration -----
14 #define SCREEN_WIDTH 128
15 #define SCREEN_HEIGHT 64
16 #define OLED_ADDR 0x3C
17
18 // ----- Pin Definitions -----
19 const int LED1_PIN = 18;
20 const int LED2_PIN = 17;
21 const int LED3_PIN = 16;
22 const int BTN_MODE = 34;
23 const int BTN_RESET = 35;
24 const int BUZZER_PIN = 25;
25
26 // ----- Debounce Configuration -----
27 const unsigned long DEBOUNCE_DELAY = 25; // Debounce time in ms
```

Simulation 00:44.333 31%



ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0 SPIWP:0xee

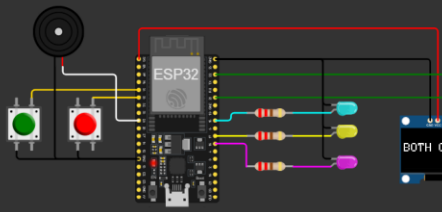
RESET-To OFF:

WOKWI SAVE SHARE A1-Task1-Multi-Mode LED Controller-1027 Docs F

sketch.ino diagram.json libraries.txt Library Manager

```
1 /*
2  Project Title : ESP32 Multi-Mode LED Controller with OLED Display
3  Name : Fiza Mahboob
4  Reg. No. : 23-NTU-CS-1027
5  Date : 2025-10-05
6 */
7
8 #include <Arduino.h>
9 #include <Wire.h>
10 #include <Adafruit_GFX.h>
11 #include <Adafruit_SSD1306.h>
12
13 // ----- OLED Configuration -----
14 #define SCREEN_WIDTH 128
15 #define SCREEN_HEIGHT 64
16 #define OLED_ADDR 0x3C
17
18 // ----- Pin Definitions -----
19 const int LED1_PIN = 18;
20 const int LED2_PIN = 17;
21 const int LED3_PIN = 16;
22 const int BTN_MODE = 34;
23 const int BTN_RESET = 35;
24 const int BUZZER_PIN = 25;
25
26 // ----- Debounce Configuration -----
27 const unsigned long DEBOUNCE_DELAY = 25; // Debounce time in ms
```

Simulation 00:48.100 37%



ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0 SPIWP:0xee

Wowki Link: <https://wokwi.com/projects/445812600209838081>

Task B — Coding: Use a single button with press-type detection (display the event on the OLED):

- Short press → toggle LED
- Long press (> 1.5 s) → play a buzzer tone

Source Code:

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// --- OLED Display Configuration ---
#define SCREEN_WIDTH 128
```

```

#define SCREEN_HEIGHT 64
#define OLED_ADDR 0x3C
// --- Pin Assignments ---
const int LED_PIN = 18;    // LED output pin
const int BTN_PIN = 34;    // Button input pin (connected to GND)
const int BUZZER_PIN = 25; // Buzzer output pin
// --- Timing Parameters ---
const unsigned long DEBOUNCE_DELAY = 50;    // Debounce time for stable button input
const unsigned long LONG_PRESS_TIME = 1500;  // Long press threshold (in milliseconds)
// --- OLED Display Object ---
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
// --- State Variables ---
bool buttonPressed = false;    // Tracks current button state
bool ledState = false;        // Stores LED ON/OFF state
unsigned long pressStartTime = 0; // Time when button was first pressed
bool longPressTriggered = false; // Prevents repeated long-press action
// --- Function Prototypes ---
void updateOLED(const char *msg);
void playBuzzerTone();
void setup() {
    Serial.begin(115200);
    // Configure I/O pins
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(BTN_PIN, INPUT_PULLUP); // Using internal pull-up (LOW when pressed)
    // Initialize OLED with I2C
    Wire.begin(21, 22);
    if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR)) {
        Serial.println("OLED initialization failed!");
        while (true); // Stop execution if OLED fails
    }
    digitalWrite(LED_PIN, LOW); // Start with LED turned OFF
    updateOLED("LED OFF");
    Serial.println("System initialized - Task B");
}
void loop() {
    unsigned long now = millis();
    int reading = digitalRead(BTN_PIN);
    // Detect when button is pressed (FALLING edge)
    if (reading == LOW && !buttonPressed) {
        buttonPressed = true;
        pressStartTime = now;
        longPressTriggered = false;
        Serial.println("Button pressed");
    }
    // Handle long press detection
    if (buttonPressed && reading == LOW && !longPressTriggered) {
        if ((now - pressStartTime) >= LONG_PRESS_TIME) {
            longPressTriggered = true;
            playBuzzerTone();
            updateOLED("BUZZER ON");
            Serial.println("Long press detected");
        }
    }
    // Detect when button is released (RISING edge)
    if (reading == HIGH && buttonPressed) {

```

```

unsigned long pressDuration = now - pressStartTime;
// Execute short press action if not a long press
if (pressDuration < LONG_PRESS_TIME && !longPressTriggered) {
  ledState = !ledState;
  digitalWrite(LED_PIN, ledState);
  if (ledState) {
    updateOLED("LED ON");
    Serial.println("Short press → LED ON");
  } else {
    updateOLED("LED OFF");
    Serial.println("Short press → LED OFF");
  }
}
// Restore OLED after long press ends
else if (longPressTriggered) {
  if (ledState) updateOLED("LED ON");
  else updateOLED("LED OFF");
}
buttonPressed = false;
delay(DEBOUNCE_DELAY);
}
}
// --- OLED Text Update Function ---
// Clears the screen and displays the provided message
void updateOLED(const char *msg) {
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 15);
  display.println(msg);
  display.display();
}
// --- Buzzer Activation Function ---
// Produces a 1000Hz tone for 0.5 seconds
void playBuzzerTone() {
  tone(BUZZER_PIN, 1000, 500);
}

```

Output ScreenShot:

Short Pressed-LED OFF(Toggle)

WOKWI SAVE SHARE A1-TaskB-Press Type Detection Docs

sketch.ino diagram.json libraries.txt Library Manager

```

6 #include <Arduino.h>
7 #include <Wire.h>
8 #include <Adafruit_GFX.h>
9 #include <Adafruit_SSD1306.h>
10
11 // --- OLED Display Configuration ---
12 #define SCREEN_WIDTH 128
13 #define SCREEN_HEIGHT 64
14 #define OLED_ADDR 0x3C
15
16 // --- Pin Assignments ---
17 const int LED_PIN = 18; // LED output pin
18 const int BTN_PIN = 34; // Button input pin (connected to GND)
19 const int BUZZER_PIN = 25; // Buzzer output pin
20
21 // --- Timing Parameters ---
22 const unsigned long DEBOUNCE_DELAY = 50; // Debounce time for stable
23 const unsigned long LONG_PRESS_TIME = 1500; // Long press threshold (in
24
25 // --- OLED Display Object ---
26 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
27
28 // --- State Variables ---
29 bool buttonPressed = false; // Tracks current button state
30 bool ledState = false; // Stores LED ON/OFF state
31 unsigned long pressStartTime = 0; // Time when button was first pressed
32 bool longPressTriggered = false; // Prevents repeated long-press action

```

Simulation

00:07.532 41%

ESP32

LED OFF

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

Again short Pressed-LED ON:

WOKWI SAVE SHARE A1-TaskB-Press Type Detection Docs

sketch.ino diagram.json libraries.txt Library Manager

```

6 #include <Arduino.h>
7 #include <Wire.h>
8 #include <Adafruit_GFX.h>
9 #include <Adafruit_SSD1306.h>
10
11 // --- OLED Display Configuration ---
12 #define SCREEN_WIDTH 128
13 #define SCREEN_HEIGHT 64
14 #define OLED_ADDR 0x3C
15
16 // --- Pin Assignments ---
17 const int LED_PIN = 18; // LED output pin
18 const int BTN_PIN = 34; // Button input pin (connected to GND)
19 const int BUZZER_PIN = 25; // Buzzer output pin
20
21 // --- Timing Parameters ---
22 const unsigned long DEBOUNCE_DELAY = 50; // Debounce time for stable
23 const unsigned long LONG_PRESS_TIME = 1500; // Long press threshold (in
24
25 // --- OLED Display Object ---
26 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
27
28 // --- State Variables ---
29 bool buttonPressed = false; // Tracks current button state
30 bool ledState = false; // Stores LED ON/OFF state
31 unsigned long pressStartTime = 0; // Time when button was first pressed
32 bool longPressTriggered = false; // Prevents repeated long-press action

```

Simulation

00:10.832 23%

ESP32

LED ON

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

Long Pressed(>1.5) -Buzzer Tone:

WOKWI SAVE SHARE A1-TaskB-Press Type Detection Docs

sketch.ino diagram.json libraries.txt Library Manager

```

6 #include <Arduino.h>
7 #include <Wire.h>
8 #include <Adafruit_GFX.h>
9 #include <Adafruit_SSD1306.h>
10
11 // --- OLED Display Configuration ---
12 #define SCREEN_WIDTH 128
13 #define SCREEN_HEIGHT 64
14 #define OLED_ADDR 0x3C
15
16 // --- Pin Assignments ---
17 const int LED_PIN = 18; // LED output pin
18 const int BTN_PIN = 34; // Button input pin (connected to GND)
19 const int BUZZER_PIN = 25; // Buzzer output pin
20
21 // --- Timing Parameters ---
22 const unsigned long DEBOUNCE_DELAY = 50; // Debounce time for stable
23 const unsigned long LONG_PRESS_TIME = 1500; // Long press threshold (in
24
25 // --- OLED Display Object ---
26 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
27
28 // --- State Variables ---
29 bool buttonPressed = false; // Tracks current button state
30 bool ledState = false; // Stores LED ON/OFF state
31 unsigned long pressStartTime = 0; // Time when button was first pressed

```

Simulation

00:20.699 31%

ESP32

BUZZER ON

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

Wowki Link: <https://wokwi.com/projects/445820132518696961>