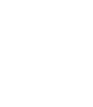
**National Textile University, Faisalabad**



**Department of Computer Science**



|  |  |
| --- | --- |
| **Name** | Malaika Batool |
| **Class** | BSCS |
| **Section** | 5th-A |
| **Registration No** | 23-NTU-CS-1044 |
| **Course Name** | EMBEDED IOT |
| **Submitted To** | SIR NASIR |
| **Submission Date** | 26/10/2025 |

**1. Purpose**

The purpose of this project is to implement an LED mode controller using an ESP32 (or compatible board) where:

* There are **three LEDs** connected to three GPIO pins.
* Two push-buttons (Button A and Button B) control the operational mode.
* An OLED display shows the current mode name.
* The system supports four distinct modes:
  1. Both LEDs OFF
  2. Alternate Blink (LED2 & LED3)
  3. Both ON
  4. Fade (LED1 only)
* Button A cycles through the modes.
* Button B resets the mode back to 0 (both OFF).  
  In short, it demonstrates GPIO control, digital input (buttons), analog output (PWM fade), and I²C OLED display usage.

**2. Hardware Design / Pin Assignment**

**Components**

* Microcontroller board: typically an ESP32 (since GPIO numbers match)
* Three LEDs (with appropriate current-limiting resistors)
* Two push-buttons (configured with internal pull-ups)
* An OLED display (SSD1306) connected via I²C (SDA & SCL)

**Pin assignment**

* LED1 → GPIO 16 (used for the Fade mode)
* LED2 → GPIO 17 (used for blinking)
* LED3 → GPIO 18 (used for blinking or ON mode)
* Button A → GPIO 14 (cycles modes)
* Button B → GPIO 27 (resets mode)
* OLED display:
  + SDA → GPIO 21
  + SCL → GPIO 22
  + I²C address: 0x3C

**Wiring notes**

* Buttons use INPUT\_PULLUP mode: so wiring one side of button to the pin, the other side to ground. When pressed, pin reads LOW.
* LEDs should have resistors (e.g., ~220 Ω) in series to limit current.
* OLED requires proper I²C connections and power (typically 3.3 V or 5 V depending on board/OLED module).
* Ensure that the pin assignments are compatible with the board and that no conflicts exist with other peripherals.

**3. Software Design / Code Explanation**

Below is an annotated walkthrough of the code.

/\*

=====================================================

Title : Task A — LED Mode Controller (Final 3 LEDs)

Author : Malaika batool

Reg No. : 23-NTU-CS-1044

Date : 26/10/2025

Description:

Button A (GPIO14): Cycles LED modes

1. Both OFF

2. Alternate Blink

3. Both ON

4. Fade (LED1 only)

Button B (GPIO27): Resets mode

OLED (GPIO21 SDA, 22 SCL): Displays mode info

=====================================================

\*/

**Libraries and OLED setup**

#include <Arduino.h>

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

#define OLED\_ADDR 0x3C

#define SDA\_PIN 21

#define SCL\_PIN 22

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, -1);

* We include the Wire library for I²C communication.
* Adafruit\_GFX and Adafruit\_SSD1306 libraries are used for the OLED display.
* Screen size defined for 128×64.
* I²C address set to 0x3C (common for SSD1306).
* We initialize display object with the dimensions and I²C interface.

**Pin definitions and variables**

#define LED1 16 // fade

#define LED2 17 // blink

#define LED3 18 // on/off

#define BUTTON\_A 14

#define BUTTON\_B 27

int mode = 0;

unsigned long lastPress = 0;

* LED1 is assigned to GPIO 16, LED2 to 17, LED3 to 18.
* Buttons assigned to 14 (A) and 27 (B).
* mode keeps track of the current mode (0,1,2,3).
* lastPress tracks the time of the last button press to implement debouncing (simple 300 ms window).

**Displaying the mode**

void showMode() {

display.clearDisplay();

display.setTextSize(1);

display.setTextColor(SSD1306\_WHITE);

display.setCursor(0, 10);

display.print("Mode: ");

if (mode == 0) display.println("BOTH OFF");

if (mode == 1) display.println("ALT BLINK");

if (mode == 2) display.println("BOTH ON");

if (mode == 3) display.println("FADE");

display.display();

}

* Clears the OLED screen, sets text size and color.
* Writes “Mode: ” then prints the current mode label.
* Finally calls display.display() to push buffer to screen.
* Makes it user-friendly by showing which mode is active.

**Setup routine**

void setup() {

Wire.begin(SDA\_PIN, SCL\_PIN);

display.begin(SSD1306\_SWITCHCAPVCC, OLED\_ADDR);

pinMode(LED1, OUTPUT);

pinMode(LED2, OUTPUT);

pinMode(LED3, OUTPUT);

pinMode(BUTTON\_A, INPUT\_PULLUP);

pinMode(BUTTON\_B, INPUT\_PULLUP);

showMode();

}

* Begins I²C on GPIO 21 (SDA) and GPIO 22 (SCL).
* Initializes the OLED display with display.begin().
* Sets LEDs as OUTPUT and buttons as INPUT\_PULLUP.
* Calls showMode() once to display initial mode (mode = 0).

**Main loop**

void loop() {

if (digitalRead(BUTTON\_A) == LOW && millis() - lastPress > 300) {

mode = (mode + 1) % 4;

showMode();

lastPress = millis();

}

if (digitalRead(BUTTON\_B) == LOW && millis() - lastPress > 300) {

mode = 0;

showMode();

lastPress = millis();

}

static int fade = 0;

static int dir = 1;

switch (mode) {

case 0: // both off

digitalWrite(LED1, LOW);

digitalWrite(LED2, LOW);

digitalWrite(LED3, LOW);

break;

case 1: // alternate blink

digitalWrite(LED2, millis() / 500 % 2);

digitalWrite(LED3, !digitalRead(LED2));

digitalWrite(LED1, LOW);

break;

case 2: // both on

digitalWrite(LED1, HIGH);

digitalWrite(LED2, HIGH);

digitalWrite(LED3, HIGH);

break;

case 3: // fade on LED1

fade += dir \* 10;

if (fade <= 0 || fade >= 255) dir = -dir;

analogWrite(LED1, fade);

digitalWrite(LED2, LOW);

digitalWrite(LED3, LOW);

delay(10);

break;

}

}

**Button handling**

* If Button A is pressed (pin reads LOW) and more than 300 ms since last press, mode increments (with wrap via %4).
* If Button B is pressed (and time condition) then mode resets to 0.
* After change, showMode() runs and lastPress updated.

**Mode operations**

* **Mode 0 (BOTH OFF):** All LEDs set LOW → off.
* **Mode 1 (ALT BLINK):**
  + LED2 toggles every 500 ms: millis()/500 %2.
  + LED3 is the inverse of LED2.
  + LED1 remains off.
* **Mode 2 (BOTH ON):** All three LEDs set HIGH → constant on.
* **Mode 3 (FADE):**
  + A static fade value moves up or down (by dir \* 10) each loop iteration.
  + When fade hits 0 or 255, dir flips to reverse ramp.
  + analogWrite(LED1, fade): LED1 fades in/out.
  + LED2 and LED3 remain off.
  + A small delay(10) adds pace to fading.

**4. Testing & Validation**

**Initial check**

* On startup, the OLED should show **“Mode: BOTH OFF”**.
* None of the LEDs should light.

**Button A functionality**

* Press Button A: OLED should update to **“Mode: ALT BLINK”** and LED2/LED3 alternate blinking every ~500 ms. LED1 remains off.
* Press again: **“Mode: BOTH ON”**, all LEDs on.
* Press again: **“Mode: FADE”**, only LED1 pulses in brightness, LED2/LED3 off.
* Press again: wraps back to **“Mode: BOTH OFF”**.

**Button B functionality**

* At any mode, pressing Button B resets to **Mode 0 (BOTH OFF)** immediately, with OLED update and LEDs off.

**Edge cases & robustness**

* Buttons are debounced via simple 300 ms time check. Verify that rapid presses don’t cause mode skipping.
* Ensure the fade loop executes smoothly, without blocking the button detection for too long (the small delay helps).
* Check LED brightness and wiring—if too dim or too bright, adjust resistor values.
* Ensure OLED I²C works reliably: display initial text, then updates on mode changes.

**Simulation on Wokwi**

* Run the simulation and verify that the correct GPIO pins correspond to the LEDs/buttons as assigned.
* Watch console and visuals: On the simulator you should see LED states change, and OLED screen updating.

**5. Usage Guide**

1. Power up the board and connect the hardware as per pin assignments.
2. On startup you’ll see the display show “Mode: BOTH OFF”.
3. To cycle through modes, press Button A each time.
4. To reset to mode 0 at any time, press Button B.
5. Modes and behavior explained:
   * **Mode 0:** All LEDs off → energy-saving/off state.
   * **Mode 1:** LED2 & LED3 blink alternately → visual indicator mode.
   * **Mode 2:** All LEDs ON → constant on status.
   * **Mode 3:** LED1 fades in/out → eye-catching or decorative mode.
6. The OLED always displays the current mode label for clarity.

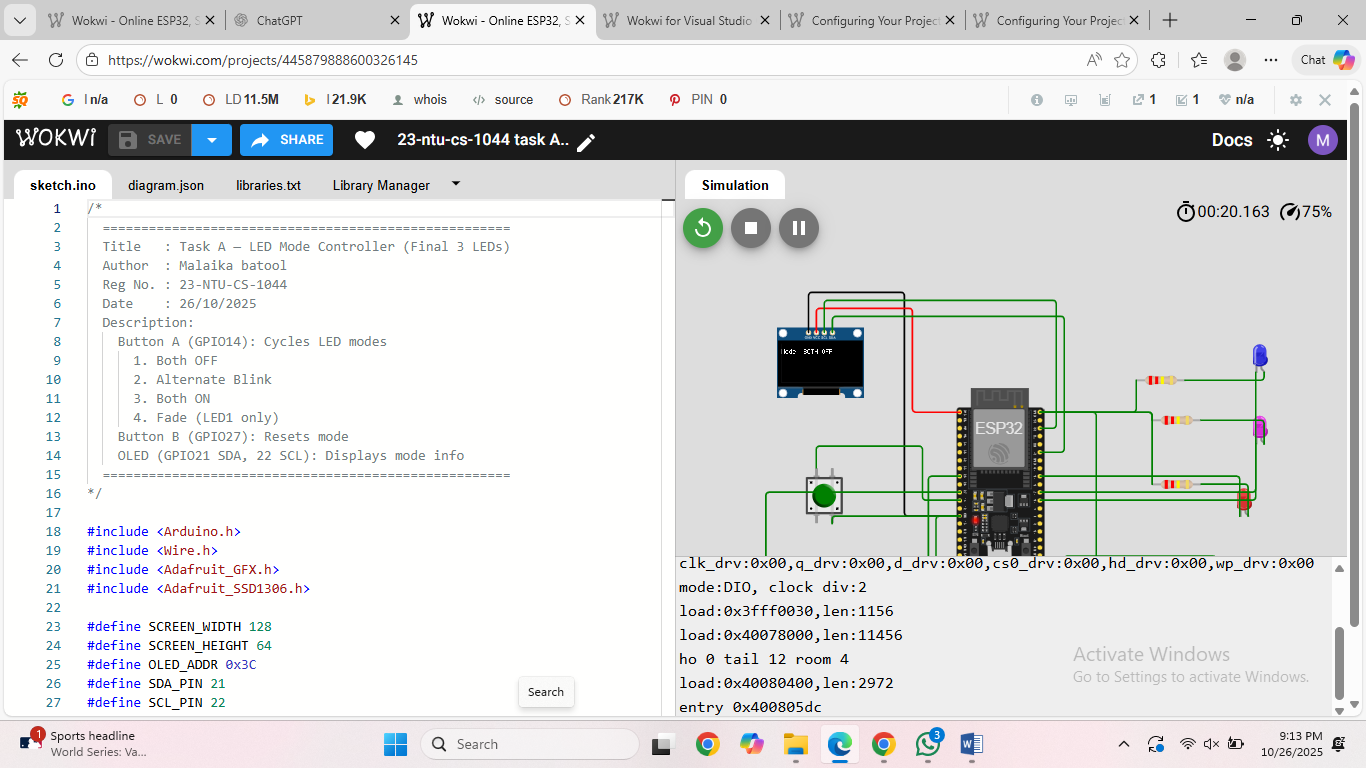
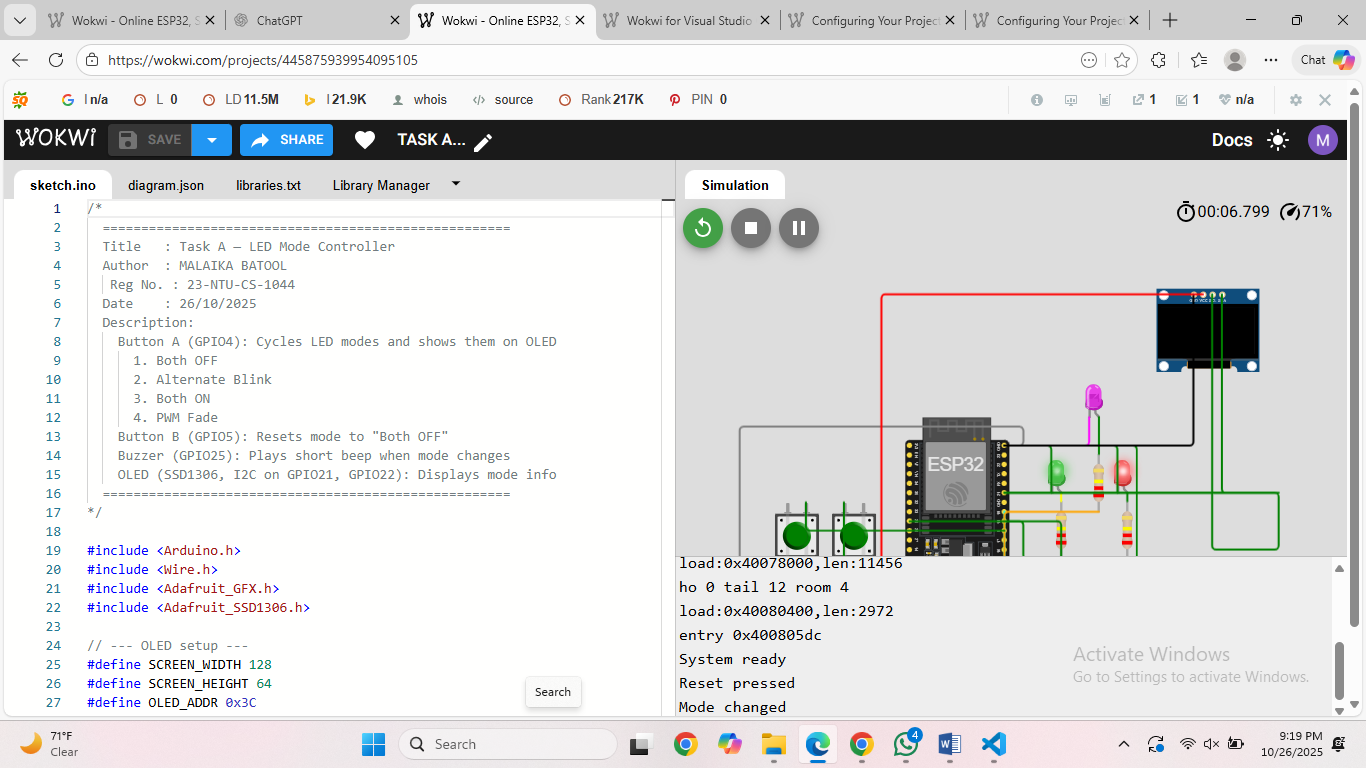
**6. Extensions & Improvements**

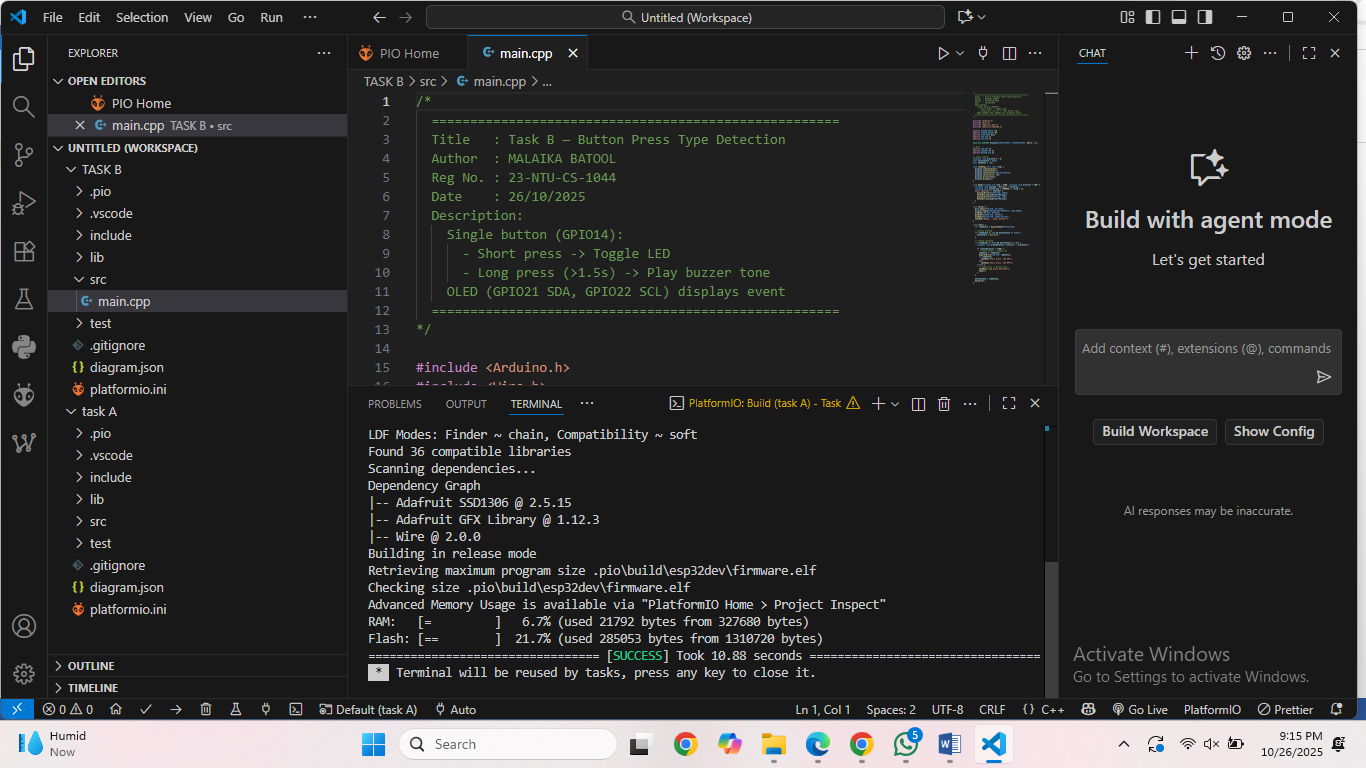
Here are some ideas for further enhancement:

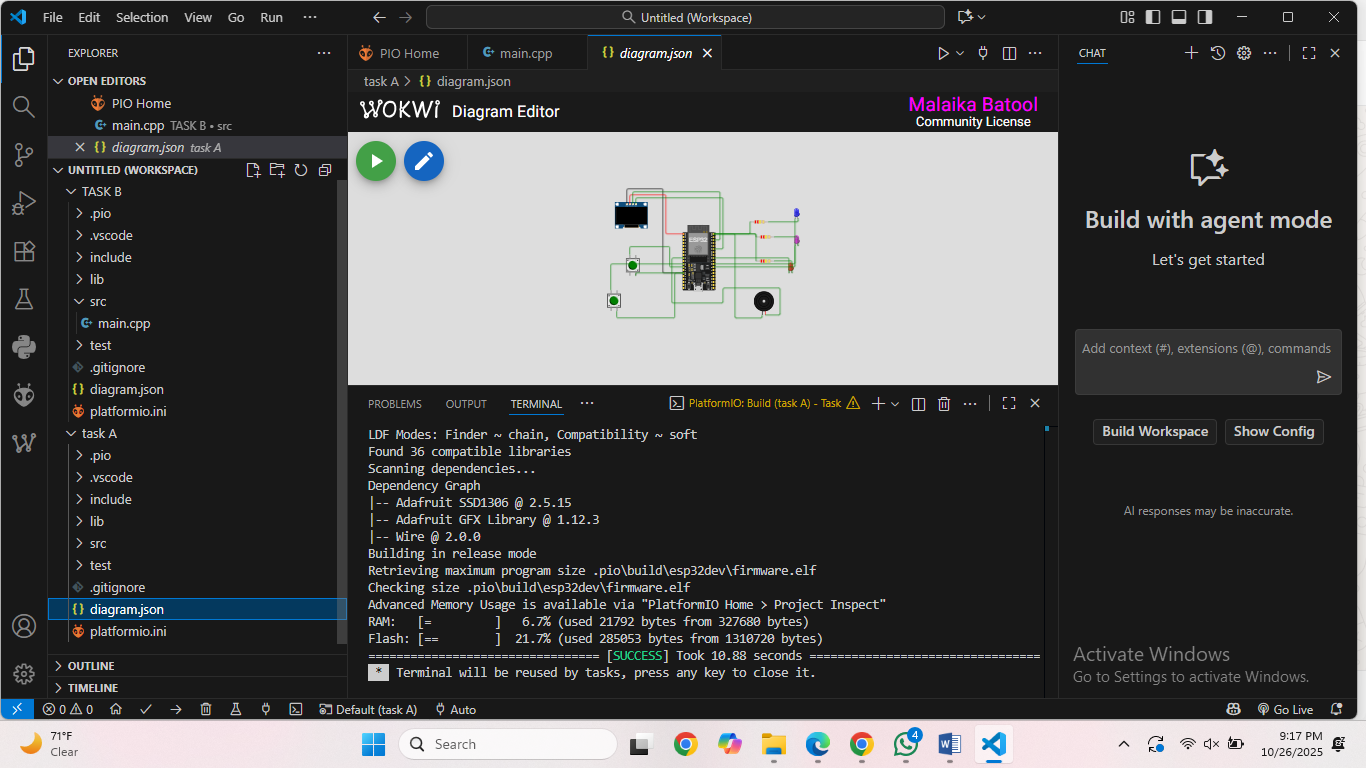
* Add more modes (e.g., chasing LEDs, random blink, PWM on multiple LEDs).
* Use interrupts for button presses instead of polling for more responsive behavior.
* Add long-press detection for additional actions (e.g., toggle speed, brightness).
* Store the last mode in non-volatile memory (EEPROM) so upon restart it resumes previous mode.
* Add speed/pattern adjustable via potentiometer or second set of buttons.
* Expand the OLED display to show more status (e.g., mode number, brightness value, timer).
* Integrate a real-time clock or sensor to trigger mode changes automatically based on time or environment.

**7. Summary**

This project successfully demonstrates a compact controller for LEDs with multiple modes, user interaction via buttons, real-time feedback via an OLED display, and usage of both digital and analog (PWM) pins. The structure is clear: initialisation, input handling, mode logic, output control. It is well-suited as a teaching example of embedded systems design on an ESP32-type board, combining input, output, display and control logic.

**SCREENSHOT:** 





## TASK B

### 1. Objective

To use one button on an ESP-board such that:

* A **short press** (< 1.5 s) toggles an LED.
* A **long press** (> 1.5 s) triggers a buzzer tone.
* An OLED display shows the event (“Short press: LED ON”, “Long press detected”, etc.).

### 2. Hardware Setup

| **Component** | **Pin** | **Function** |
| --- | --- | --- |
| Button | GPIO 14 | Detect press and release |
| LED | GPIO 16 | Toggle ON/OFF on short press |
| Buzzer | GPIO 25 | Sound on long press |
| OLED Display | SDA → GPIO 21, SCL → GPIO 22 | Displays current event |

Button is wired using internal pull-up (INPUT\_PULLUP).

### 3. Software Overview

* Uses Adafruit\_GFX + Adafruit\_SSD1306 for the OLED.
* In loop(), reads the button state.
  + On press (state goes LOW): capture pressStart = millis().
  + On release (state goes HIGH): compute pressDuration = millis() - pressStart.
    - If pressDuration < 1500 ms: it’s a **short press** → toggle LED state, update OLED message.
    - Else: it’s a **long press** → update OLED message + call beep() to drive buzzer.
* The beep() routine toggles the buzzer pin at a frequency (~1000 Hz) for ~400 ms.
* Debouncing via a small delay(10) and tracking previous button state.

### 4. Testing & Validation

* On startup the OLED should display: **“Ready… press button”**.
* Short press the button (quick tap): LED should toggle ON or OFF, and OLED updates accordingly.
* Long press (hold for about 2 s): buzzer sounds, OLED shows **“Long press detected”**.
* Ensure no unintended toggles due to bounce or very short presses.

### 5. Summary

This project demonstrates how to differentiate between **short and long button presses**, taking different actions accordingly (LED toggle vs buzzer), and providing user feedback via an OLED display. It covers digital input reading, timing for press duration, output control (LED + buzzer), and display interaction in an embedded setup.

If you like, I can **create a schematic diagram** of your wiring and a **formatted PDF documentation** ready for submission. Would you like that?

##### You said:

bit long

###### ChatGPT said:

Here’s a **short version** of the documentation for your Wokwi project 👇

## ****Task B — Button Press Type Detection****

**Author:** Malaika Batool  
**Reg No:** 23-NTU-CS-1044 **Date:** 26-10-2025

### ****1. Objective****

Detect **short** and **long** button presses using an ESP32:

* **Short press (<1.5 s):** Toggle LED
* **Long press (>1.5 s):** Activate buzzer
* OLED shows press type and current state

### ****2. Hardware Setup****

| **Component** | **Pin** | **Function** |
| --- | --- | --- |
| Button | GPIO14 | Input (press detection) |
| LED | GPIO16 | Toggle on short press |
| Buzzer | GPIO25 | Beep on long press |
| OLED | SDA → GPIO21, SCL → GPIO22 | Display messages |

All buttons use **INPUT\_PULLUP** mode.

### ****3. Software Overview****

* Uses Adafruit\_SSD1306 and Adafruit\_GFX for OLED.
* Detects press duration using millis().
* **If <1500 ms:** toggles LED and shows “Short Press”.
* **If ≥1500 ms:** sounds buzzer and shows “Long Press”.
* Includes debounce to prevent false readings.

### ****4. Testing****

✅ OLED shows “Ready…” at start  
✅ Short press → LED toggles  
✅ Long press → Buzzer sounds  
✅ Messages update correctly

### ****5. Summary****

A simple project demonstrating **button timing**, **digital I/O**, **buzzer control**, and **OLED feedback** on ESP32.

