**National Textile University, Faisalabad**



**Department of Computer Science**

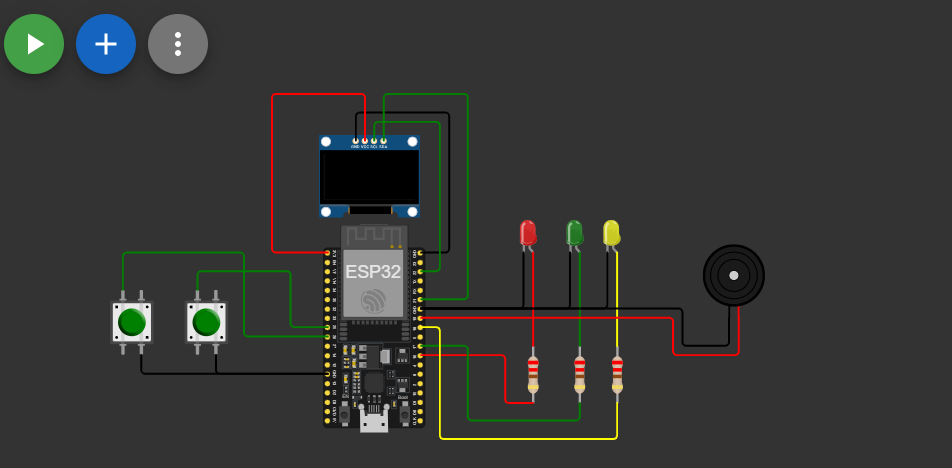
|  |  |
| --- | --- |
| **Name:** | Arslan Ali |
| **Class:** | BSCS 5A |
| **Registration No:** | 23-NTU-CS-1019 |
| **Assignment:** | 1 |
| **Course Name:** | IOT |
| **Submitted To:** | Dr. Nasir Sb |
| **Submission Date:** | 26 October 2025 |

**Question 3----Implementation**

**Circuit-Diagram**

Draw a Wokwi circuit and draw a neat hand-sketch including

* 2 push buttons
* 3 LEDs
* 1 Buzzer
* 1 OLED

****

**Task A—Coding**

Use On Buttonto cycle through LED Models(Display the current state on OLED):

* Both OFF
* Alternate Blink
* Power ON
* PWM Fade

**Code:**

#include <Arduino.h>

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

// tell the compiler that OLED does not have a physical reset pin

#define OLED\_RESET    -1

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &**Wire**, OLED\_RESET);

// ---------------- LED PINS ----------------

// define LED pins connected to the ESP32 board

#define LED\_YELLOW 18

#define LED\_GREEN  17

#define LED\_RED    16

// ---------------- BUTTON PINS ----------------

// define pins for buttons used to cycle and reset LED modes

#define BTN\_CYCLE 25

#define BTN\_RESET 26

// ---------------- BUTTON DEBOUNCE VARIABLES ----------------

// stores the \*\*current stable state\*\* of the cycle button — after it has stopped bouncing

int stableButtonState = HIGH;

// used to compare and detect button changes — stores the last read value

int lastReading = HIGH;

// stores the \*\*previous stable state\*\* of the button

int lastStableState = HIGH;

// store the time when the button last changed state

unsigned long lastDebounceTime = 0;

// debounce delay — how long to wait before confirming a button press

const unsigned long DEBOUNCE\_MS = 30;

// similar variables for the reset button

int stableResetState = HIGH;

int lastResetReading = HIGH;

int lastStableResetState = HIGH;

unsigned long lastResetDebounceTime = 0;

// ---------------- LED MODE VARIABLES ----------------

// LED state: 0 = ALL OFF, 1 = ALL ON, 2 = BLINK, 3 = PWM FADE

int ledMode = 0;

// ---------------- BLINK VARIABLES ----------------

// store the last time the LED blinked

unsigned long lastBlinkTime = 0;

// interval between LED blinks (in milliseconds)

const unsigned long BLINK\_INTERVAL = 500;

// keeps track of current ON/OFF blink state

bool blinkState = false;

// ---------------- PWM FADE VARIABLES ----------------

// stores current LED brightness level (0 = OFF, 255 = full brightness)

int fadeValue = 0;

// direction of fade: 1 = fading up, -1 = fading down

int fadeDirection = 1;

// stores the last time the brightness was updated

unsigned long lastFadeTime = 0;

// time between each brightness update (smaller = smoother fade)

const unsigned long FADE\_INTERVAL = 10;

void setup() {

  // set button pins as input with internal pull-up resistors

  pinMode(BTN\_CYCLE, INPUT\_PULLUP);

  pinMode(BTN\_RESET, INPUT\_PULLUP);

  // set LED pins as outputs

  pinMode(LED\_YELLOW, OUTPUT);

  pinMode(LED\_GREEN, OUTPUT);

  pinMode(LED\_RED, OUTPUT);

  // ------------- INITIALIZE OLED DISPLAY -------------

  // initialize display and check if it connected properly

  if (!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) {

    for (;;); // stop if display is not found

  }

  // clearDisplay() → clears the OLED screen to black

  display.clearDisplay();

  // setTextSize(2) → doubles the font size

  display.setTextSize(2);

  // setTextColor(SSD1306\_WHITE) → sets text color to white

  display.setTextColor(SSD1306\_WHITE);

  // initialize button states at startup

  lastReading = digitalRead(BTN\_CYCLE);

  stableButtonState = lastReading;

  lastStableState = stableButtonState;

  lastResetReading = digitalRead(BTN\_RESET);

  stableResetState = lastResetReading;

  lastStableResetState = stableResetState;

  // make sure all LEDs start OFF

  analogWrite(LED\_YELLOW, 0);

  analogWrite(LED\_GREEN, 0);

  analogWrite(LED\_RED, 0);

}

void loop() {

  unsigned long currentMillis = millis(); // current time (used for timing events)

  // -------- HANDLE BTN\_CYCLE (MODE SWITCH) --------

  int reading = digitalRead(BTN\_CYCLE);

  // check if button state has changed

  if (reading != lastReading) {

    lastDebounceTime = currentMillis;  // reset debounce timer

    lastReading = reading;

  }

  // if stable for enough time, confirm the change

  if (currentMillis - lastDebounceTime >= DEBOUNCE\_MS) {

    if (stableButtonState != reading) {

      stableButtonState = reading;

      // detect falling edge (HIGH → LOW) = button press

      if (lastStableState == HIGH && stableButtonState == LOW) {

        ledMode++;  // move to next LED mode

        if (ledMode > 3) ledMode = 0;  // wrap back to first mode

        // reset variables for blink/fade

        fadeValue = 0;

        fadeDirection = 1;

        blinkState = false;

        lastBlinkTime = currentMillis;

        lastFadeTime = currentMillis;

      }

      lastStableState = stableButtonState;

    }

  }

  // -------- HANDLE BTN\_RESET (RESET TO OFF) --------

  int resetReading = digitalRead(BTN\_RESET);

  if (resetReading != lastResetReading) {

    lastResetDebounceTime = currentMillis;

    lastResetReading = resetReading;

  }

  if (currentMillis - lastResetDebounceTime >= DEBOUNCE\_MS) {

    if (stableResetState != resetReading) {

      stableResetState = resetReading;

      if (lastStableResetState == HIGH && stableResetState == LOW) {

        // reset everything to OFF mode

        ledMode = 0;

        analogWrite(LED\_YELLOW, 0);

        analogWrite(LED\_GREEN, 0);

        analogWrite(LED\_RED, 0);

      }

      lastStableResetState = stableResetState;

    }

  }

  // -------- HANDLE LED MODES --------

  switch (ledMode) {

    case 0: // ALL OFF

      analogWrite(LED\_YELLOW, 0);

      analogWrite(LED\_GREEN, 0);

      analogWrite(LED\_RED, 0);

      break;

    case 1: // ALL ON

      analogWrite(LED\_YELLOW, 255);

      analogWrite(LED\_GREEN, 255);

      analogWrite(LED\_RED, 255);

      break;

    case 2: // ALTERNATE BLINK

      // toggle blink state every BLINK\_INTERVAL ms

      if (currentMillis - lastBlinkTime >= BLINK\_INTERVAL) {

        blinkState = !blinkState;

        lastBlinkTime = currentMillis;

      }

      // alternate ON/OFF pattern

      analogWrite(LED\_YELLOW, blinkState ? 255 : 0);

      analogWrite(LED\_GREEN, blinkState ? 0 : 255);

      analogWrite(LED\_RED, blinkState ? 255 : 0);

      break;

    case 3: // PWM FADE

      // update brightness every FADE\_INTERVAL ms

      if (currentMillis - lastFadeTime >= FADE\_INTERVAL) {

        fadeValue += fadeDirection \* 5;  // increase or decrease brightness

        // reverse direction at limits

        if (fadeValue >= 255) { fadeValue = 255; fadeDirection = -1; }

        if (fadeValue <= 0)   { fadeValue = 0; fadeDirection = 1; }

        // apply brightness to all LEDs

        analogWrite(LED\_YELLOW, fadeValue);

        analogWrite(LED\_GREEN, fadeValue);

        analogWrite(LED\_RED, fadeValue);

        lastFadeTime = currentMillis;

      }

      break;

  }

  // -------- UPDATE OLED DISPLAY --------

  display.clearDisplay();    // clear old text

  display.setCursor(0, 20);  // set text position

  // print current LED mode to OLED

  switch (ledMode) {

    case 0: display.println("ALL OFF"); break;

    case 1: display.println("ALL ON"); break;

    case 2: display.println("BLINKING"); break;

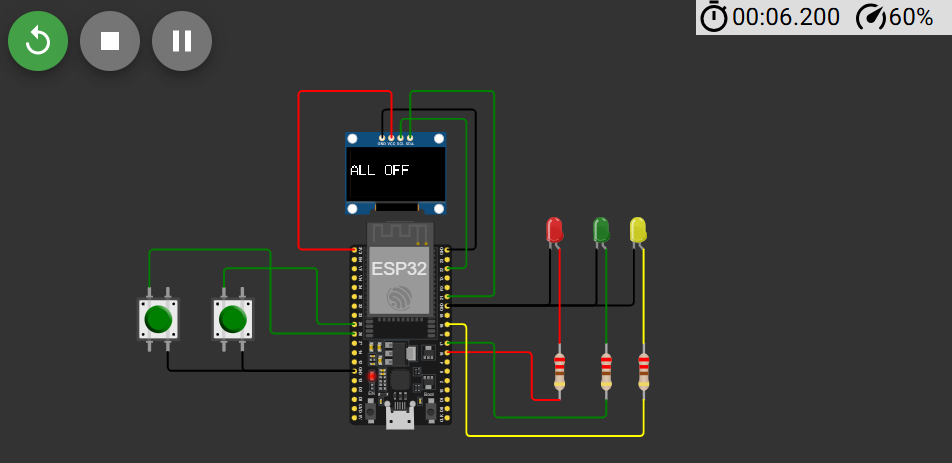
    case 3: display.println("PWM FADE"); break;

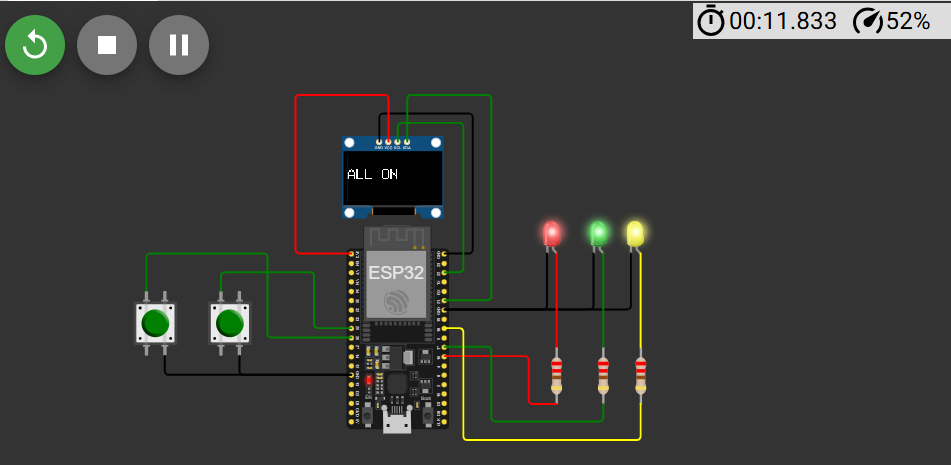
  }

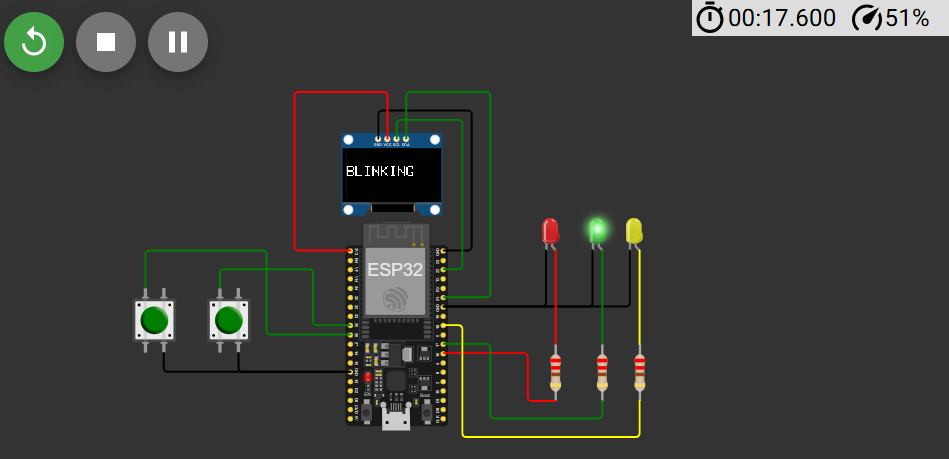
  display.display();         // send buffer to the screen

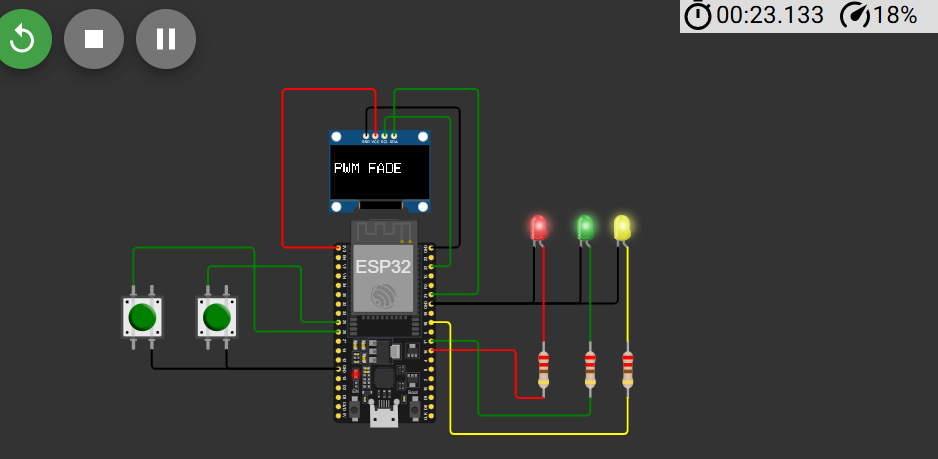
}

**Output:**

****

****

****

****

**Task B**

**Code:**

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

#define OLED\_RESET -1

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);

// --- Pin Definitions ---

#define BTN\_PIN 25

#define LED\_PIN 18

#define BUZZER 19

// --- LED PWM Setup (ESP32) ---

#define CH\_LED 0

#define PWM\_FREQ 5000

#define PWM\_RES 8 // 8-bit (0–255)

// --- Debounce and Press Detection ---

bool lastButtonState = HIGH;

bool buttonPressed = false;

unsigned long pressStartTime = 0;

unsigned long lastDebounceTime = 0;

const unsigned long DEBOUNCE\_MS = 30;

const unsigned long LONG\_PRESS\_MS = 1500;

// --- LED & Buzzer states ---

bool ledState = false;

bool buzzerOn = false;

unsigned long buzzerStartTime = 0;

const unsigned long BUZZER\_DURATION = 200;

void setup() {

pinMode(BTN\_PIN, INPUT\_PULLUP);

pinMode(BUZZER, OUTPUT);

digitalWrite(BUZZER, LOW);

// Setup LED PWM

ledcSetup(CH\_LED, PWM\_FREQ, PWM\_RES);

ledcAttachPin(LED\_PIN, CH\_LED);

ledcWrite(CH\_LED, 0);

// Setup OLED

if (!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) {

for (;;);

}

display.clearDisplay();

display.setTextSize(2);

display.setTextColor(SSD1306\_WHITE);

display.setCursor(0, 20);

display.println("Ready");

display.display();

}

void loop() {

unsigned long currentMillis = millis();

bool reading = digitalRead(BTN\_PIN);

// Debounce button

if (reading != lastButtonState) {

lastDebounceTime = currentMillis;

lastButtonState = reading;

}

if ((currentMillis - lastDebounceTime) > DEBOUNCE\_MS) {

// Button pressed

if (!buttonPressed && reading == LOW) {

buttonPressed = true;

pressStartTime = currentMillis;

}

// Button released

if (buttonPressed && reading == HIGH) {

unsigned long pressDuration = currentMillis - pressStartTime;

buttonPressed = false;

if (pressDuration >= LONG\_PRESS\_MS) {

// --- Long Press ---

digitalWrite(BUZZER, HIGH);

buzzerOn = true;

buzzerStartTime = currentMillis;

display.clearDisplay();

display.setCursor(0, 20);

display.println("Long Press");

display.display();

} else {

// --- Short Press ---

ledState = !ledState;

ledcWrite(CH\_LED, ledState ? 255 : 0);

display.clearDisplay();

display.setCursor(0, 20);

display.println(ledState ? "LED ON" : "LED OFF");

display.display();

}

}

}

// Turn off buzzer after duration

if (buzzerOn && (currentMillis - buzzerStartTime >= BUZZER\_DURATION)) {

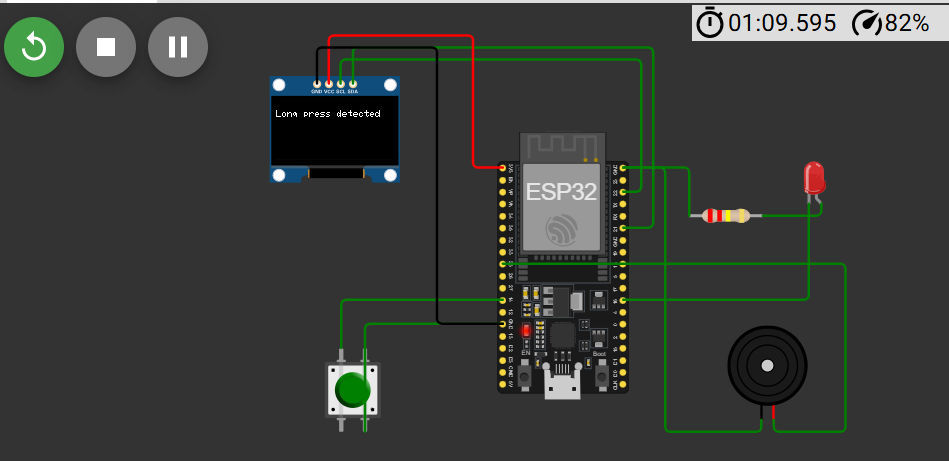
digitalWrite(BUZZER, LOW);

buzzerOn = false;

}

}

**Output:**

****

