**National Textile University, Faisalabad**

A logo with blue and red squares

AI-generated content may be incorrect.

**Department of Computer Science**

|  |  |
| --- | --- |
| **Name:** | Mariha Iftikhar |
| **Class:** | BSCS 5th\_A |
| **Registration No:** | 23-NTU-CS-1045 |
| **Course Name:** | Embeded IoT |
| **Submitted To:** | Sir Nasir |
| **Submission Date:** | 19/10/2025 |

**ASSIGNMENT 1**

**Question no 1: Short Questions**

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

Volatile tell the compiler a variable can change anytime .When a variable is modified inside ISR its value can change anytime , outside the main program control. so it always reads the latest value from memory .

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

Hardware-timer debouncing is accurate and because it uses a hardware timer to ignore switch bounces for a short ,precise interval without blocking the processor. ;delay()-based debouncing stop the whole program temporarily and also it is suitable for basic applications , hardware timer ISR debouncing is preferred for real-time or multitasking system.

**3)What does IRAM\_ATTER do, and why is it needed?**

IRAM\_ATTR stores the ISR in fast internal RAM so it can run quickly even when flash memory is busy. It is mainly used for interrupt service Routines which need to run very quickly and reliably . By placing ISR in IRAM the function can excuete instantly and safely without delays and crashes.

**4)Define LEDC channels , timers and duty cycle?**

LEDC channels are independent PWM outputs that can be assigned to different GPIO pins. Each channel control one signal.

Think of the **LEDC timer** as the **clock** or **heartbeat** that controls how fast the PWM signal turns ON and OFF.

It sets:

* **PWM frequency** (how many ON/OFF cycles per second)
* **PWM resolution** (how finely you can control brightness or speed)

The **duty cycle** tells **how long the signal stays ON (HIGH)** compared to the total time of one complete ON/OFF cycle. This “ON time” controls how much power the device receives **on average**.

**5)Why should you avoid serial prints or long code paths inside ISRs?**

Serial prints or long code inside ISRs slow down interrupts and can cause missed or delayed tasks.Additionally serial communication function can cause crashes or data corruption if used inside an IRS.

**6)What are the advantage of timer-based task scheduling?**

Timer based scheduling gives accurate timing ..It allows tasks to run at precise , regular intervals without blocking the processor , improving timing accuracy and system efficiently. Multiple tasks can schedule independently , enabling real-time control and multitasking.

**7)Describe I^2C signals SDA and SCL?**

In I^2C communication ,SDA carrires the data between devices, while SCL provides the clock signal for synchronization.The master controls the clock on SCL and data is transferred through SDA in sync with it.

**8)What are the difference between polling and interrupt-driven input?**

Polling keeps checking input repeatedly which wastes processing timeinterrupt-driveninput the CPU performs other tasks and responds only when an interrupt occurs. Interrupts are more efficient and suitable for real time multitasking.

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

Contact bounce is rapid ON/OFF flickering when pressing a button ; it must be filtered to avoid false triggers. These false signals can be misread as multiple presses by the microcontroller .It must be handled to ensure accurate and stable input readings.

**10)How does the LEDC peripheral improve PWM precision?**

LEDC uses hardware timers with high resolution , giving smooth and precise PWM signals. It allows fine adjustment of frequency and duty cycle without relying on software delays. This ensures stable, accurate and jitter-free PWM signals for LEDS, motors and other devices.

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

ESP32 has 4 hardware timers per core ,total 8 timers.

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

A prescaler timer is a divider that reduce the input clock frequency before it reaches the timer. It is used to slow down the timer count rate , allowing longer timing intervals or lower PWM frequencies.

**13)Define duty cycle and frequency in PWM?**

Duty cycle is the percentage of time the signal stays high during one complete cycle.The frequency is the number of PWMcycles that occur per second measured in hertz..

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

Duty =(brightness /100)\*max\_duty\_value.

**15)Contrast non-blocking vs blocking-timing?**

Non-blocking use methods like timers or millis() allowing other tasks to run simultaneously. Blocking use methods (like delay()) which stop the program from doing anything else until the delays end.

**16)What resolution does LEDC support?**

LEDC peripherals on ESP32 supports PWM resolutions from 1 bit up to 20 bits. This allows very fine control over duty cycle providing up to 1,048,576(2^20) possible output levels. Higher resolutions gives smoother and more precise brightness or speed control.

**17)Compare general purpose hardware timers and LEDC timers?**

General purpose hardware timers are versatile and can be used for various tasks like delays event counting or time measurement.; LEDC timers are specialized for PWM signal with adjustable frequency and resolution.

**18)What is the difference between Adafruit\_SSD1306 and Adafruit\_GFX?**

Adafruit\_SSD1306 is a library specifically for controlling OLED displays that use the SSD1306 driver.

Adafruit\_GFX is a general graphics library that provide drawing functions like lines, shapes and text.

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

You can optimize text rendering on an OLED by minimizing screen update and redrawing only changed areas instead of the whole display. Use smaller fonts or fewer graphics to reduce data transfer time.

**20)Give short specifications of your selected ESP32 board ?**

NodeMCU-32S:Dual core 240MHz ,520KB SRAM ,Wi-Fi + Bluetooth , 30 GPIOs , 4 timers/core,8 PWM channels , 12-bit ADC.

**Question no 2**

**1)A 10 kHz signal has an ON time of 10 ms .What is the duty cycle?**

Formula : Duty (%) = (T\_on / T\_period)\*100.

Period of 10 kHz = 1 /10,000 = 0.0001 s = 0.1ms

If T\_on = 10ms then Duty = (10ms / 0.1ms)\*100 = 10,000% - impossible (greater than 100%) .So the given numbers comflict.

Likely intended : T\_on = 10 ->Duty =(10/100)\*100 = 10%.

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

Timers:ESP32 has 4 hardware timers per timer-group/core , so 8 hardware timers total.

Interrupts:You can attach interrupts to most GPIOs (roughly 30 +usable GPIOs) and to many peripheral sources; practically dozens of distinct ISR source .Real limit deponds on reserved pins , peripheral occupancy and priorities.

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

LEDC provides 8 independent PWM timers (4 high-speed+4 low-speed) .Each LEDC timer defines frequency/resolution for any channels tied to it. So you can run up to 8 distinct PWM frequencies simultaneously .constraints: total numbers of LEDC channels vs timers, pin-muxing , and the trade-off between frequency and resolution (high freq ->lower max resolution)

**4)Compare 30%duty at 8-bit (1kHz) vs 10-bit?**

8-bit max value = 255.30%of 255 =76.5->77 steps->actual = 77/255 = 30.196% (~ +0.196 percentage points quantization error)

10-bit max =1023.30% of 1023 = 306.9->307 ->307/1023 = 30.029 % (~ +0.029 error)

Conclusion: 10 bit gives much finer duty steps (smaller quantization error = smoother brightness control)

**5)How many characters on a 128\*64 OLED at once (min vs max font)?**

Assumptions stated :use Adafruit\_GFX base font metrices (default 6\*8 pixel per character, including spacing).

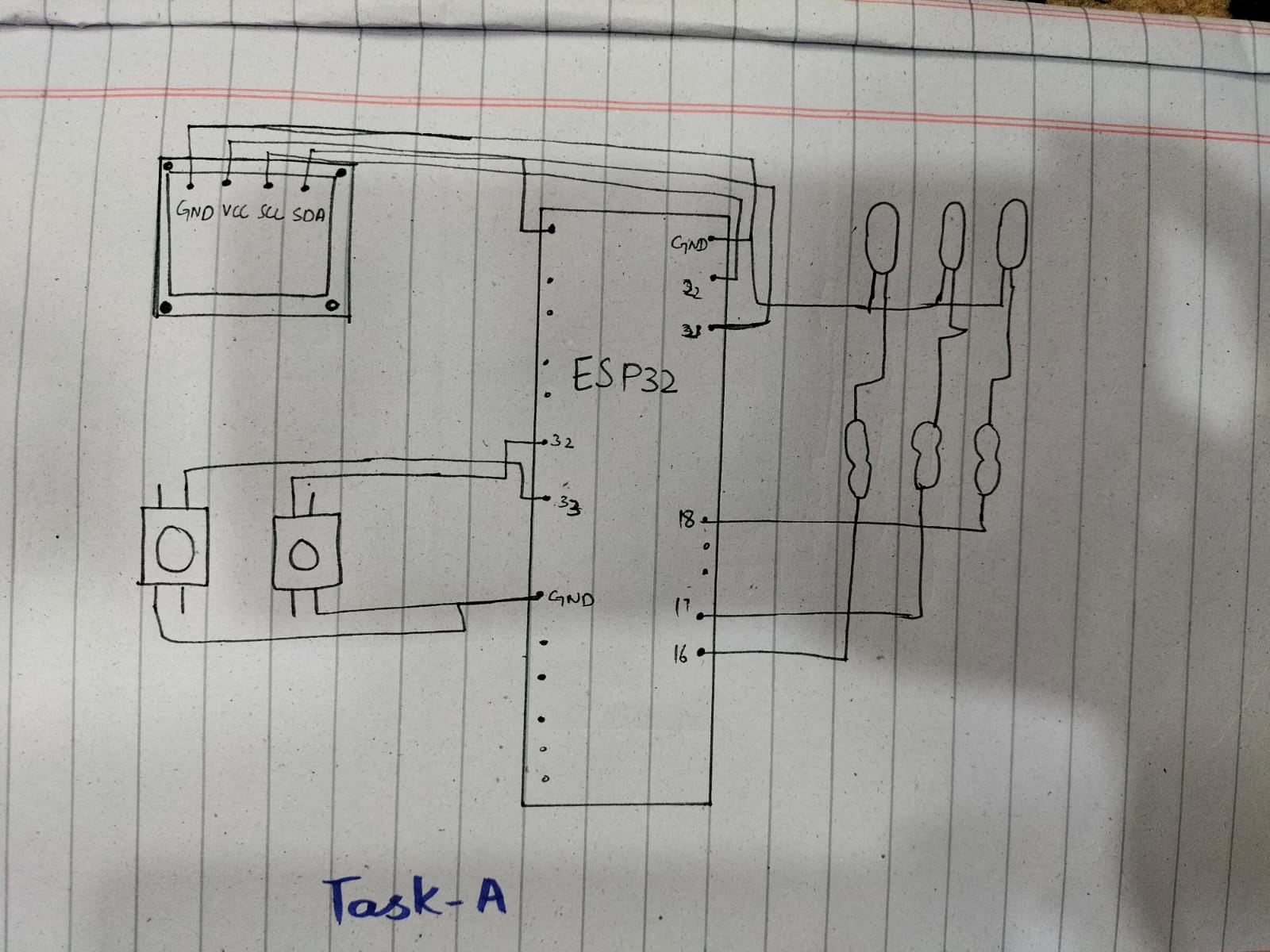
Minimum font (6\*8):columns = 128/6 =21 chars , rows = 64 /8 =8 lines ->21\*8 = 168 characters.

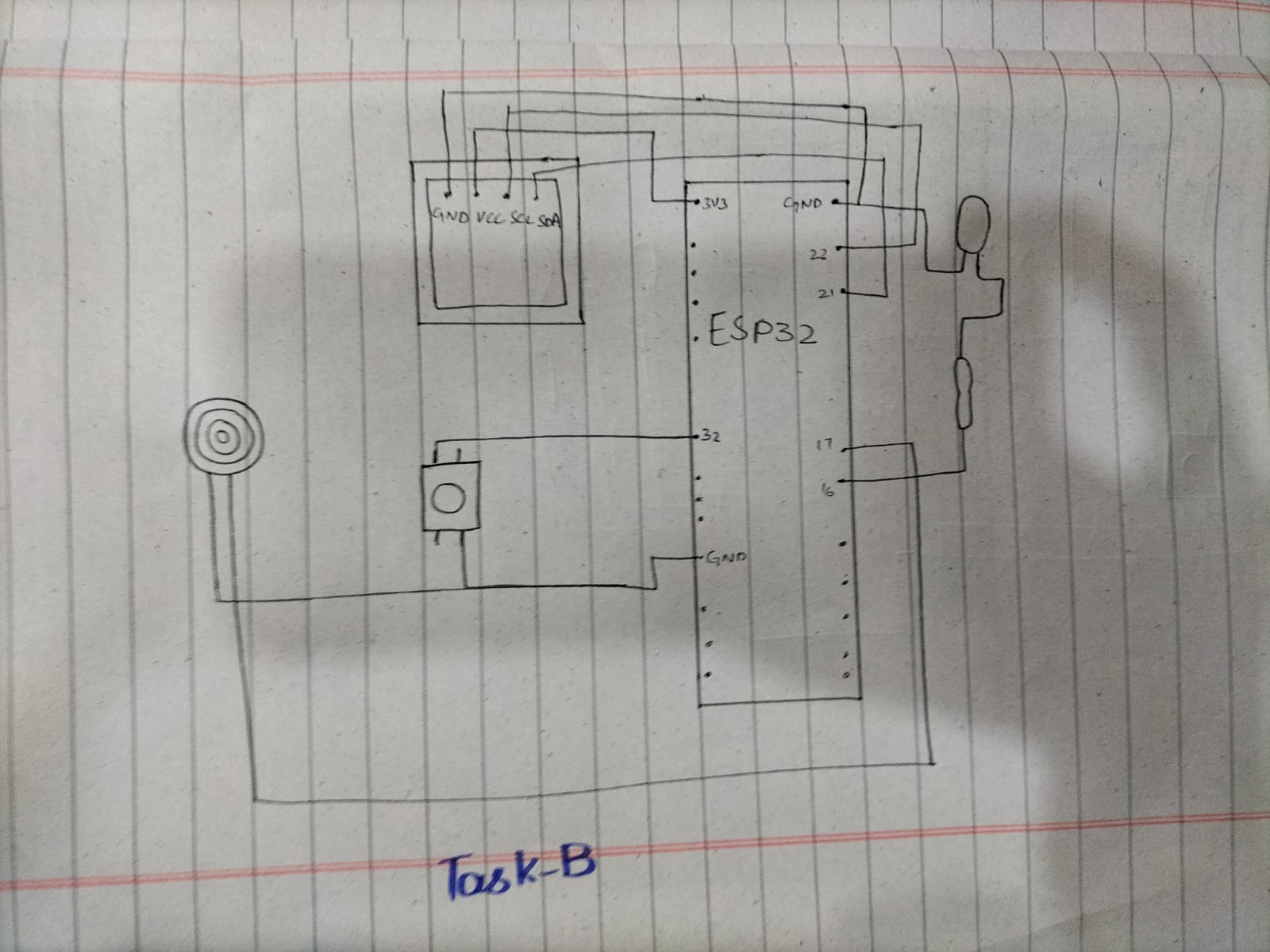
Example :128/24 = 5 ,rows =64 /32 = 2 ->5\*2 = 10 characters.

Edge case :with a single full- screen glyph you could show 1 large character .result depend on chosen font bitmap and spacing.

**Question no 3:**

**Sketch:**

****

****

**Task A:**

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define LED1 16

#define LED2 17

#define LED3 18

#define BTN\_MODE 32

#define BTN\_RESET 33

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

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

int mode = 0;

int brightness = 0;

int fadeAmount = 5;

// Timing variables for non-blocking delays

unsigned long previousBlinkMillis = 0;

unsigned long previousFadeMillis = 0;

unsigned long previousButtonMillis = 0;

unsigned long previousResetMillis = 0;

const long blinkInterval = 200;

const long fadeInterval = 15;

const long buttonInterval = 300;

const long resetDebounce = 200; // debounce for reset button

// Software PWM settings (works in Wokwi)

const unsigned long pwmPeriod = 10; // ms per PWM cycle (~100 Hz)

                                   // lower value -> smoother PWM but more CPU usage

// Blink state variables

int blinkState = 0;

void setup() {

  pinMode(LED1, OUTPUT);

  pinMode(LED2, OUTPUT);

  pinMode(LED3, OUTPUT);

  pinMode(BTN\_MODE, INPUT\_PULLUP);

  pinMode(BTN\_RESET, INPUT\_PULLUP);

  if (!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) while (1);

  display.clearDisplay();

  display.setTextSize(1);

  display.setTextColor(SSD1306\_WHITE);

  showMode("All OFF");

  // ensure LEDs are off

  digitalWrite(LED1, LOW);

  digitalWrite(LED2, LOW);

  digitalWrite(LED3, LOW);

}

void showMode(const char \*text) {

  display.clearDisplay();

  display.setCursor(0, 25);

  display.print("Mode: ");

  display.println(text);

  display.display();

}

void resetToMode0() {

  mode = 0;

  brightness = 0;

  fadeAmount = 5;

  blinkState = 0;

  digitalWrite(LED1, LOW);

  digitalWrite(LED2, LOW);

  digitalWrite(LED3, LOW);

  showMode("Reset to OFF");

}

void loop() {

  unsigned long currentMillis = millis();

  // --- RESET BUTTON (HIGH PRIORITY) with debounce ---

  if (digitalRead(BTN\_RESET) == LOW && (currentMillis - previousResetMillis >= resetDebounce)) {

    previousResetMillis = currentMillis;

    resetToMode0();

    delay(50); // small pause to avoid immediate retrigger

    return; // Exit so reset takes effect immediately

  }

  // --- MODE BUTTON with debounce ---

  if (digitalRead(BTN\_MODE) == LOW && (currentMillis - previousButtonMillis >= buttonInterval)) {

    previousButtonMillis = currentMillis;

    mode++;

    if (mode > 3) mode = 0;

    // Reset LEDs when changing modes

    digitalWrite(LED1, LOW);

    digitalWrite(LED2, LOW);

    digitalWrite(LED3, LOW);

    brightness = 0;

    fadeAmount = 5;

    blinkState = 0;

    previousBlinkMillis = currentMillis;

    previousFadeMillis = currentMillis;

    switch (mode) {

      case 0:

        showMode("All OFF");

        break;

      case 1:

        showMode("Alternate Blink");

        break;

      case 2:

        digitalWrite(LED1, HIGH);

        digitalWrite(LED2, HIGH);

        digitalWrite(LED3, HIGH);

        showMode("All ON");

        break;

      case 3:

        showMode("PWM Fade");

        break;

    }

    delay(50); // tiny extra debounce gap

  }

  // --- MODE BEHAVIORS (NON-BLOCKING) ---

  switch (mode) {

    case 1: // Alternate Blink

      if (currentMillis - previousBlinkMillis >= blinkInterval) {

        previousBlinkMillis = currentMillis;

        // Turn all LEDs off first

        digitalWrite(LED1, LOW);

        digitalWrite(LED2, LOW);

        digitalWrite(LED3, LOW);

        // Turn on the current LED in sequence

        switch (blinkState) {

          case 0:

            digitalWrite(LED1, HIGH);

            break;

          case 1:

            digitalWrite(LED2, HIGH);

            break;

          case 2:

            digitalWrite(LED3, HIGH);

            break;

        }

        blinkState = (blinkState + 1) % 3;

      }

      break;

    case 3: // PWM Fade (software PWM for Wokwi / no analogWrite)

      // update brightness at fadeInterval rate (non-blocking)

      if (currentMillis - previousFadeMillis >= fadeInterval) {

        previousFadeMillis = currentMillis;

        brightness += fadeAmount;

        // clamp/flip on edges for stable behaviour

        if (brightness <= 0) {

          brightness = 0;

          fadeAmount = abs(fadeAmount);

        } else if (brightness >= 255) {

          brightness = 255;

          fadeAmount = -abs(fadeAmount);

        }

      }

      // software PWM cycle: compute duty within pwmPeriod

      {

        unsigned long phase = currentMillis % pwmPeriod; // 0 .. pwmPeriod-1 ms

        unsigned int onTime = (unsigned long)brightness \* pwmPeriod / 255u; // ms LED should be ON this cycle

        bool on = (phase < onTime);

        digitalWrite(LED1, on ? HIGH : LOW);

        digitalWrite(LED2, on ? HIGH : LOW);

        digitalWrite(LED3, on ? HIGH : LOW);

      }

      break;

    // cases 0 and 2 don't need periodic work

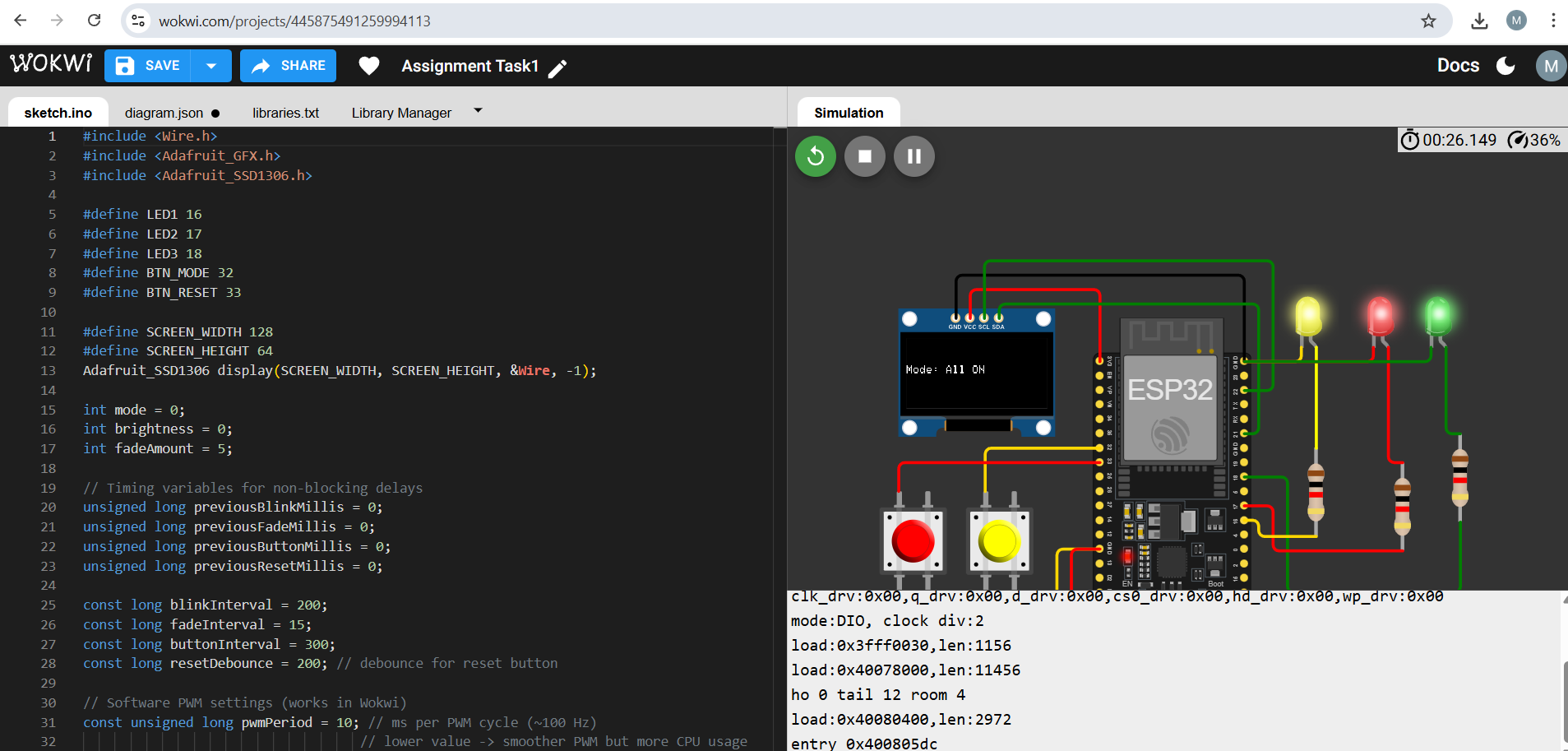
    default:

      break;

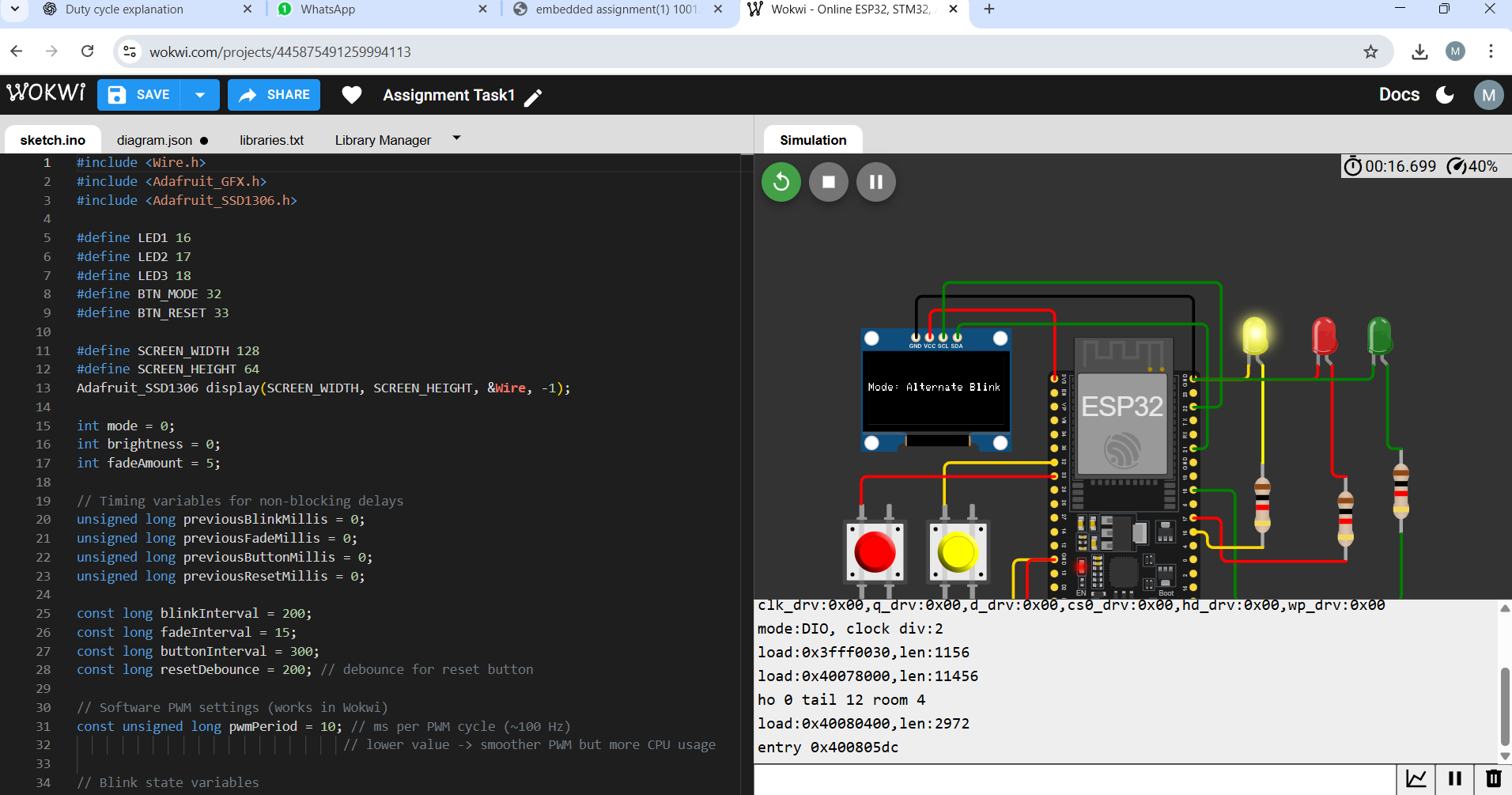
  }

}

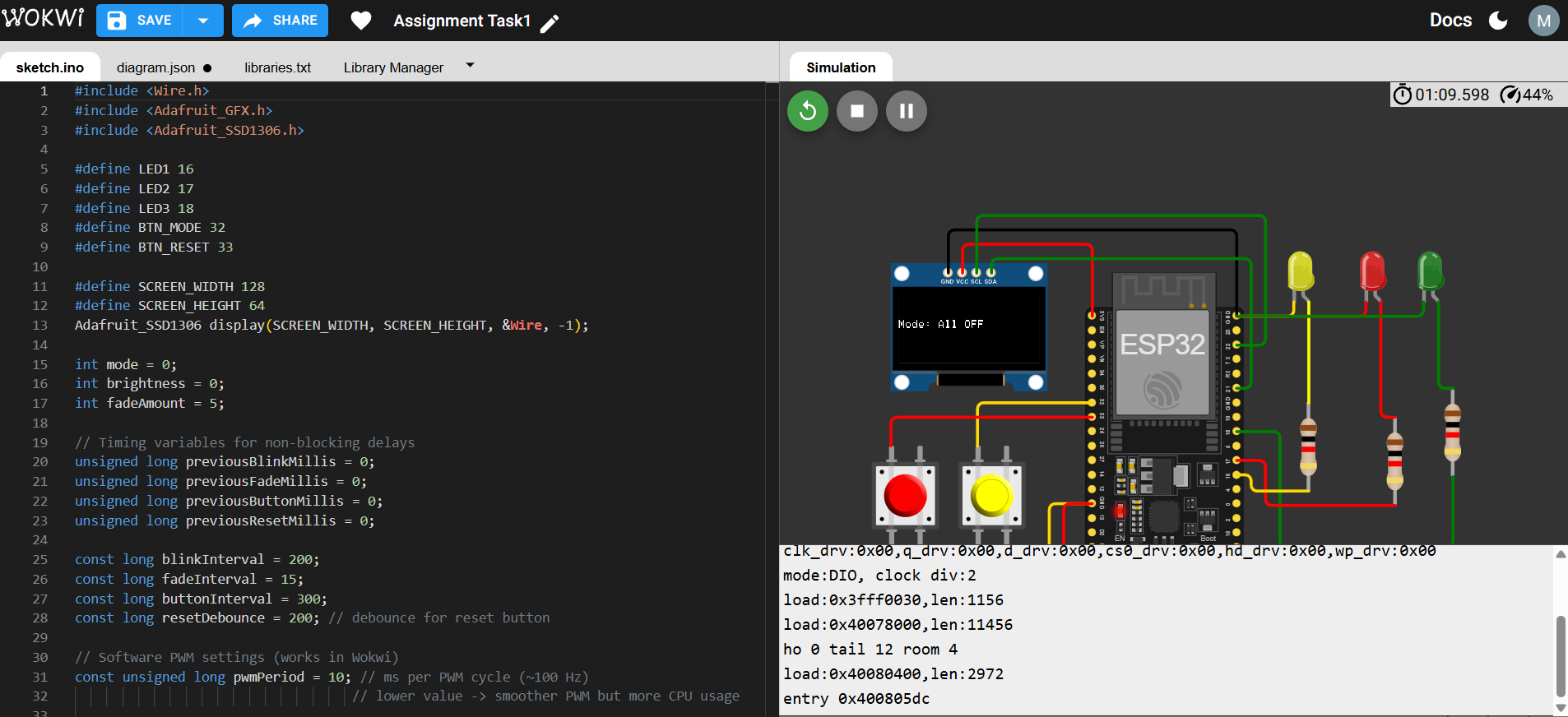
**All LED ON:**



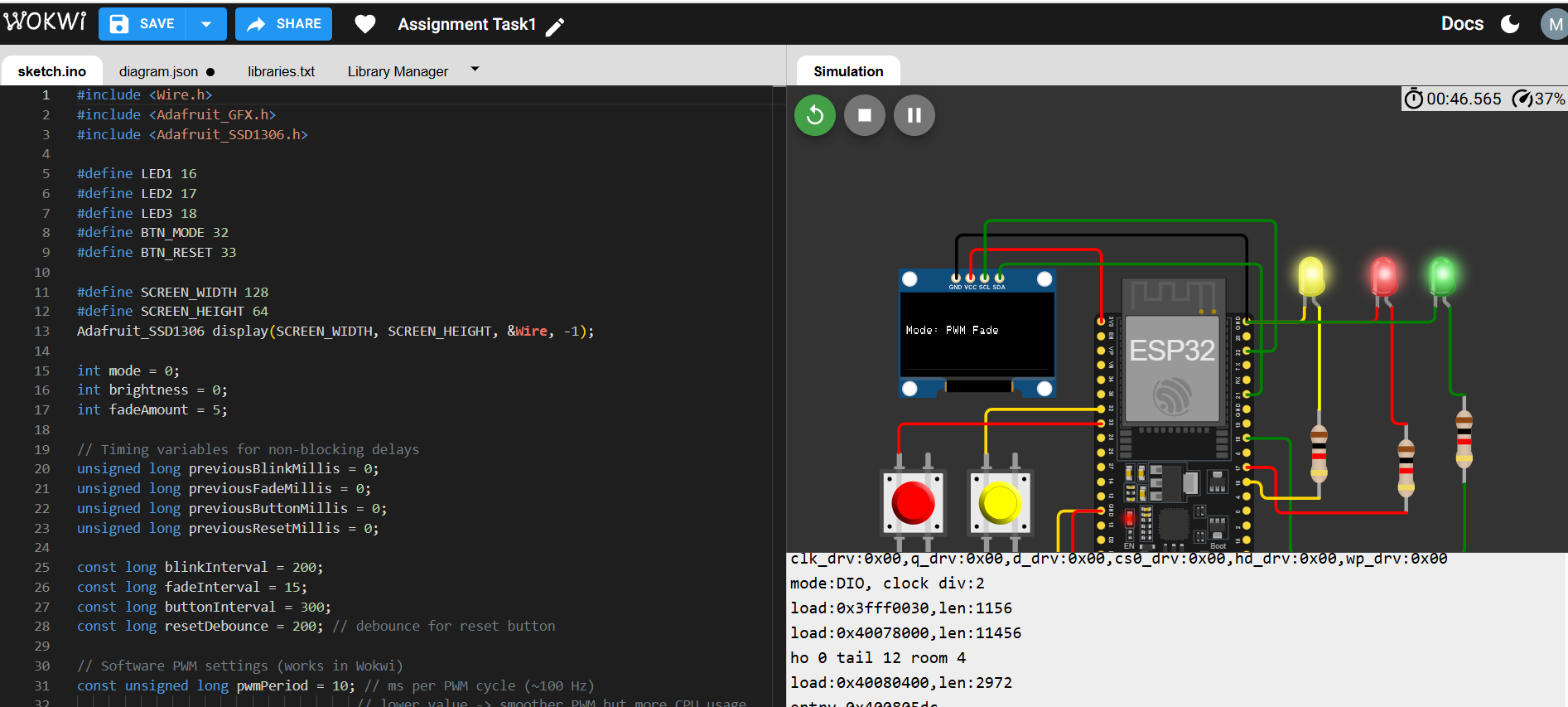
**Alternate LED blink:**



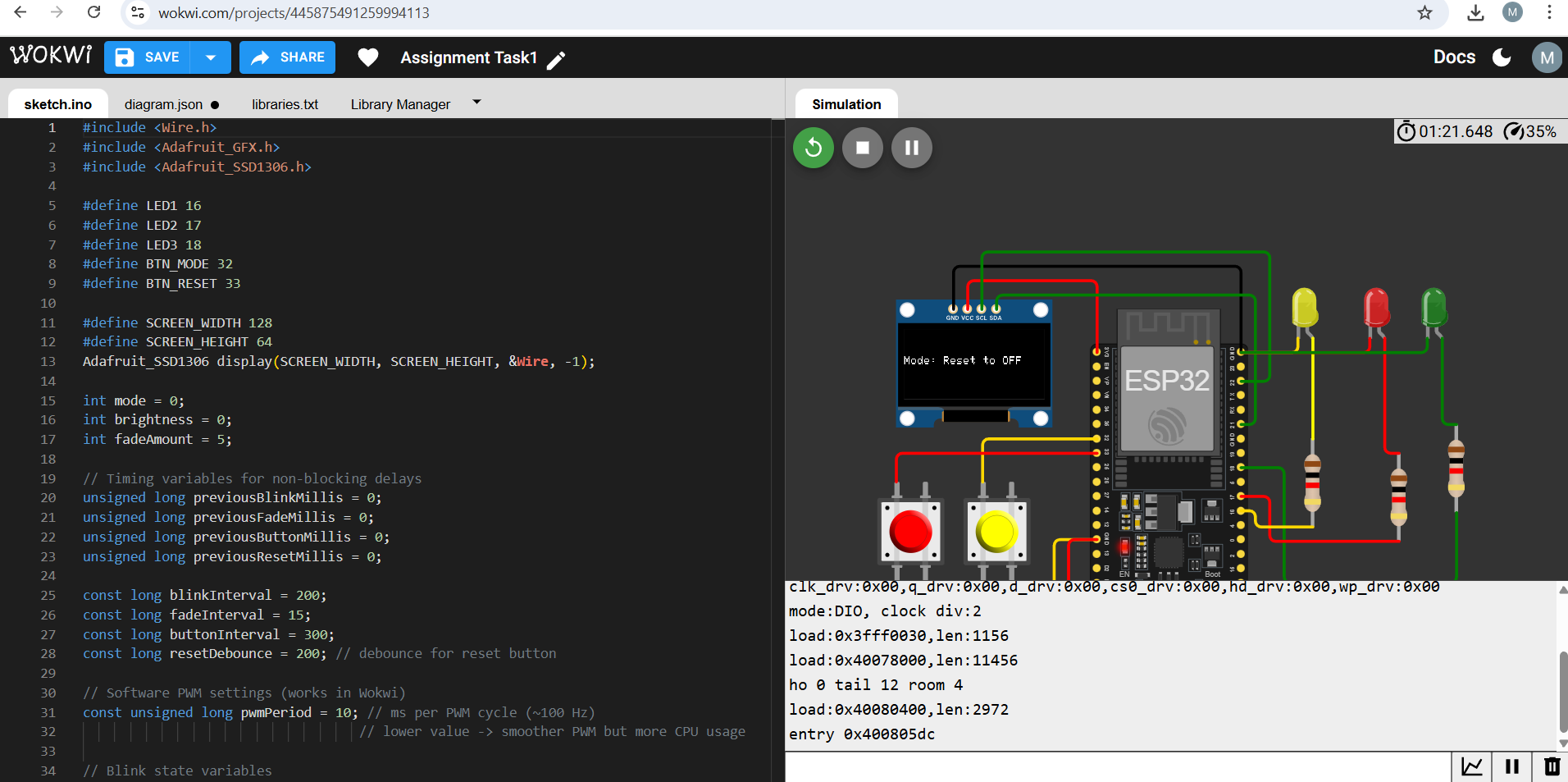
**All LED OFF:**



**LED Fade ON:**



**Reset TO OFF:**



**Link:**

[**https://wokwi.com/projects/445875491259994113**](https://wokwi.com/projects/445875491259994113)

**Task 2:**

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

// Pin configuration

const int buttonPin = 32;  // single button

const int ledPin = 16;     // LED pin

const int buzzerPin = 17;  // buzzer pin

// OLED configuration

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

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

// Variables for button timing

unsigned long buttonPressTime = 0;

bool isButtonPressed = false;

bool ledState = false;

bool longPressActive = false;

const unsigned long longPressDuration = 1500; // 1.5 seconds for long press

const unsigned long debounceDelay = 50;       // 50ms debounce time

void showOLED(const char \*message) {

  display.clearDisplay();

  display.setTextSize(2);

  display.setTextColor(SSD1306\_WHITE);

  display.setCursor(0, 0);

  display.println(message);

  display.display();

}

void setup() {

  pinMode(buttonPin, INPUT\_PULLUP);

  pinMode(ledPin, OUTPUT);

  pinMode(buzzerPin, OUTPUT);

**Wire**.begin(21, 22); // SDA = 21, SCL = 22 for ESP32

**Serial**.begin(115200);

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

**Serial**.println("❌ OLED init failed");

    while (true);

  }

  showOLED("Ready");

  // Initialize LED to OFF state

  digitalWrite(ledPin, LOW);

}

void loop() {

  int buttonState = digitalRead(buttonPin);

  // Button pressed (LOW because of INPUT\_PULLUP)

  if (buttonState == LOW && !isButtonPressed) {

    // Wait for debounce period to confirm the press

    delay(debounceDelay);

    // Check button again after debounce

    if (digitalRead(buttonPin) == LOW) {

      isButtonPressed = true;

      buttonPressTime = millis();

      longPressActive = false;

**Serial**.println("Button pressed - waiting for release");

    }

  }

  // Button is being held

  if (buttonState == LOW && isButtonPressed) {

    if (!longPressActive && (millis() - buttonPressTime > longPressDuration)) {

      // Long press detected – activate buzzer

      longPressActive = true;

      tone(buzzerPin, 1000);

      showOLED("BUZZER");

**Serial**.println("Long press activated - BUZZER ON");

    }

  }

  // Button released (HIGH because of INPUT\_PULLUP)

  if (buttonState == HIGH && isButtonPressed) {

    // Wait for debounce period to confirm the release

    delay(debounceDelay);

    // Check button again after debounce

    if (digitalRead(buttonPin) == HIGH) {

      noTone(buzzerPin); // stop buzzer immediately when released

      if (!longPressActive) {

        // Short press – toggle LED

        ledState = !ledState;

        digitalWrite(ledPin, ledState ? HIGH : LOW);

        showOLED(ledState ? "LED ON" : "LED OFF");

**Serial**.println(ledState ? "LED turned ON" : "LED turned OFF");

      } else {

        // If long press happened

        showOLED("Stopped");

**Serial**.println("Long press stopped");

      }

      // Reset state

      isButtonPressed = false;

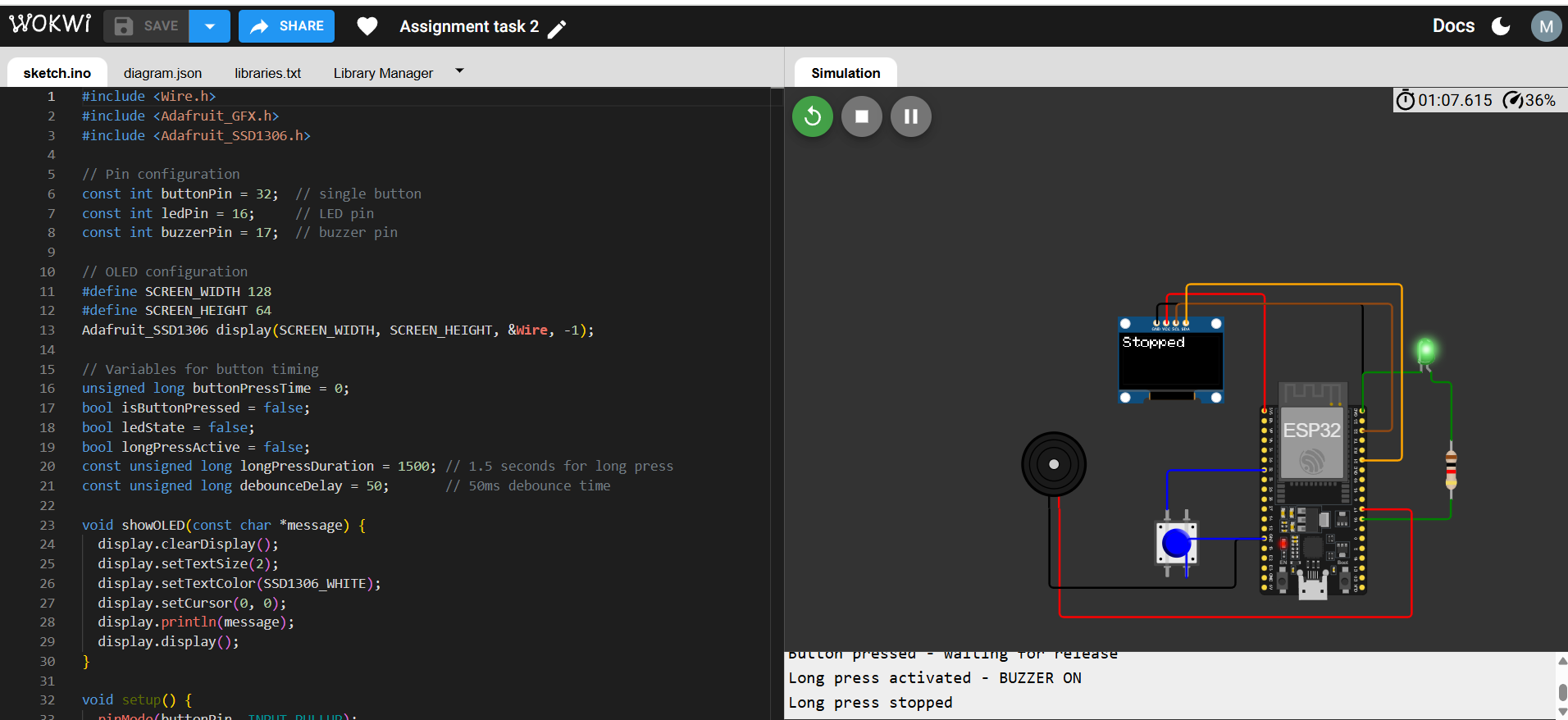
      longPressActive = false;

    }

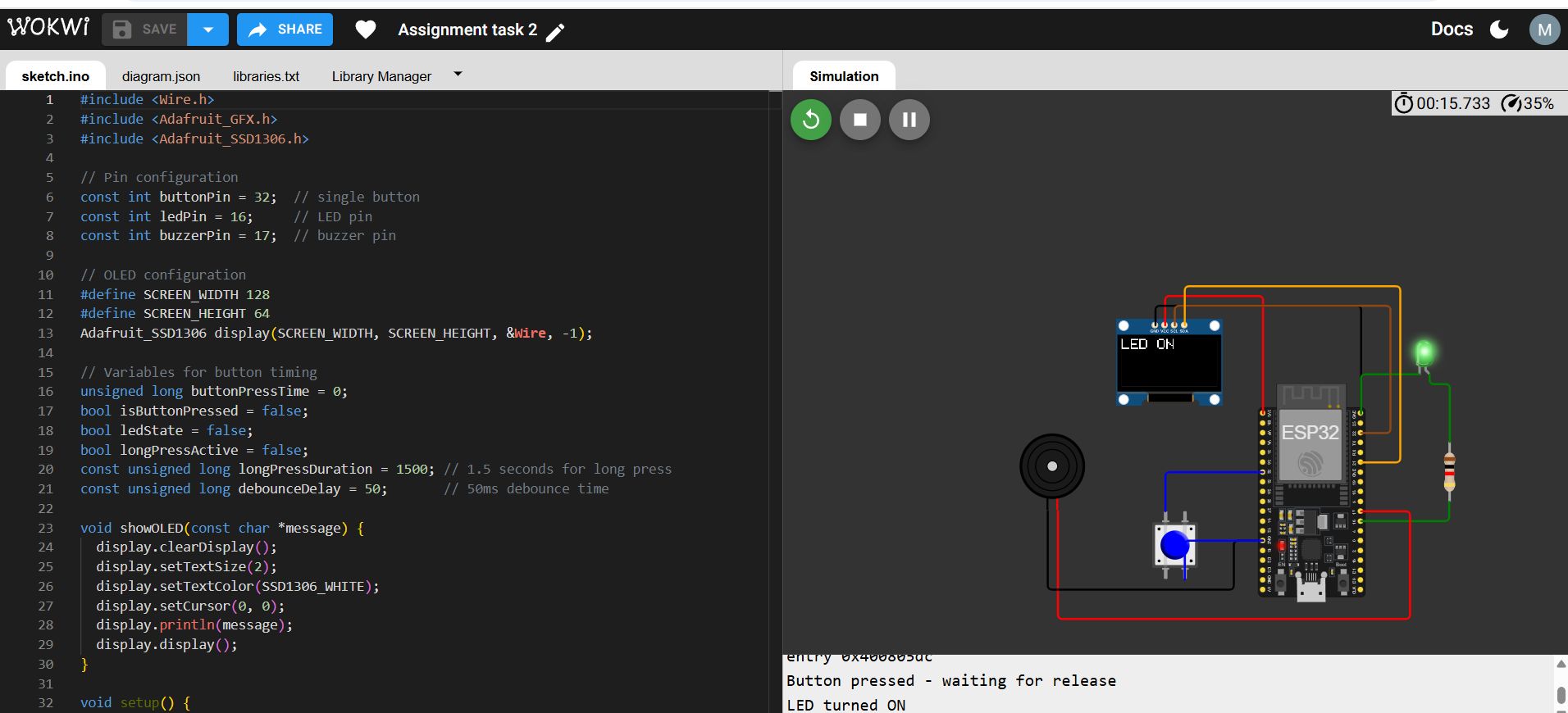
  }

}

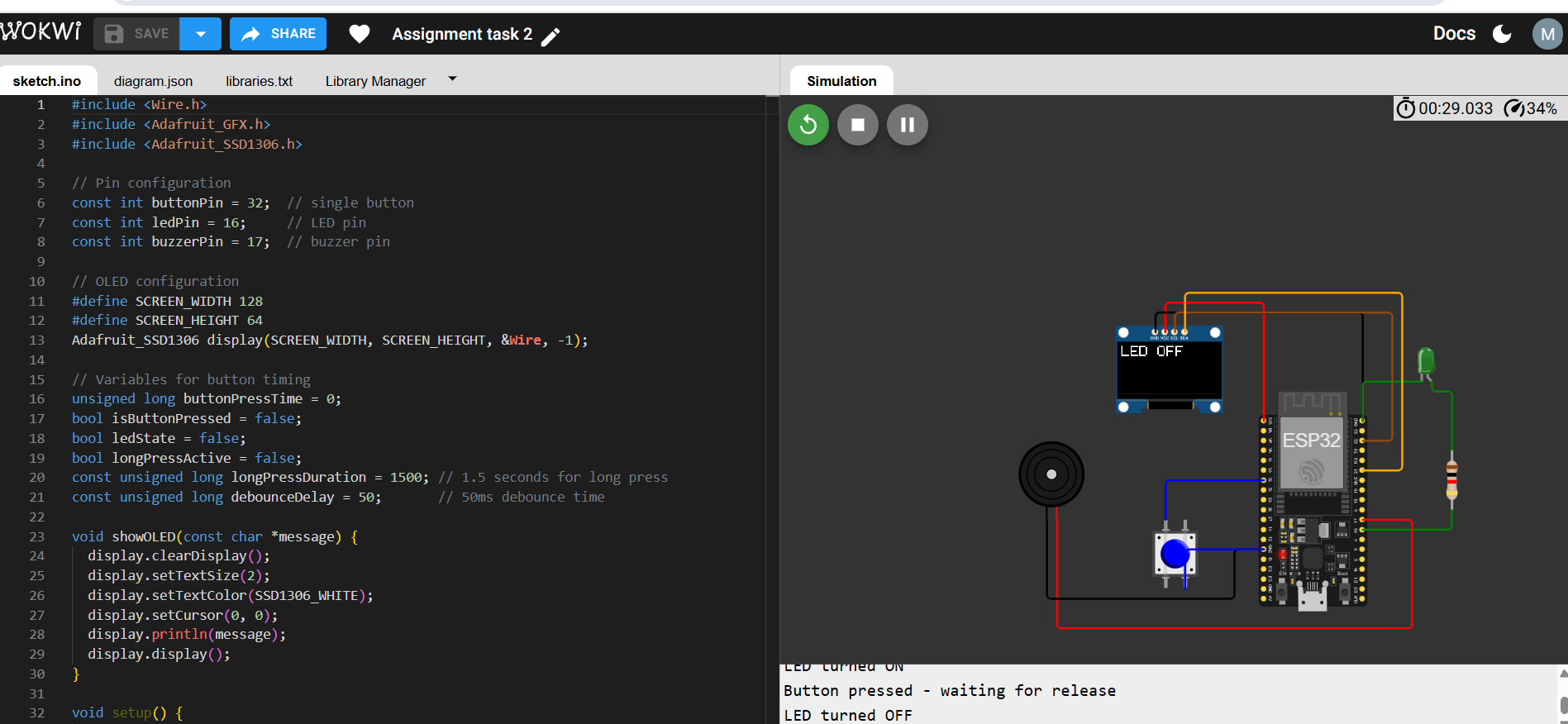
**Buzzer Stop:**



**LED ON:**



**LED OFF:**



**LINK:**

[**https://wokwi.com/projects/445850113932143617**](https://wokwi.com/projects/445850113932143617)