Pakistan Institute of Engineering and Applied Sciences

Lab Report

Lab-01: Digital Input and Output

Group Members:

- M. Hammad Tahir
- Ehmaan Shafqat
- Khadija Arif
- Ali Raza

Course Name: Practical IoT

Instructor: Dr. Naveed Akhtar

Submission Date: 28/02/2025

Objectives:

- Understand the fundamentals of digital signals in IoT.
- Learn how to interface a push button (digital input) with a WeMos D1 Mini (ESP8266).
- Control an LED (digital output) using the push button.
- Implement both basic and debounced button control.
- Gain practical experience in reading digital states and outputting digital signals.

Required Components:

- WeMos D1 Mini (ESP8266)
- LED (any color), You can also use built-in LED
- 220Ω resistor (for LED current limiting, in case using External LED)
- Push Button (momentary type)
- Breadboard & Jumper Wires
- USB cable for programming
- (Optional) 10kΩ resistor if using an external pull-down resistor

Task 1: Basic Digital I/O – Controlling an LED with a Push Button:

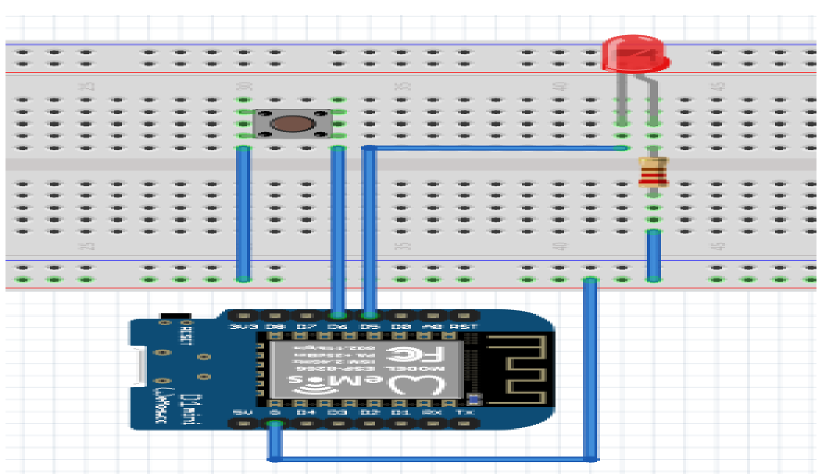
• LED Wiring:

- Anode (+) → WeMos D1 Mini digital pin (e.g., D5)
- Cathode (-) → Ground through a 220Ω resistor

• Push Button Wiring:

- One terminal → Ground
- Other terminal → WeMos D1 Mini digital pin (e.g., D6) with internal pull-up resistor enabled via code (Optionally you can add 10k external resistor)

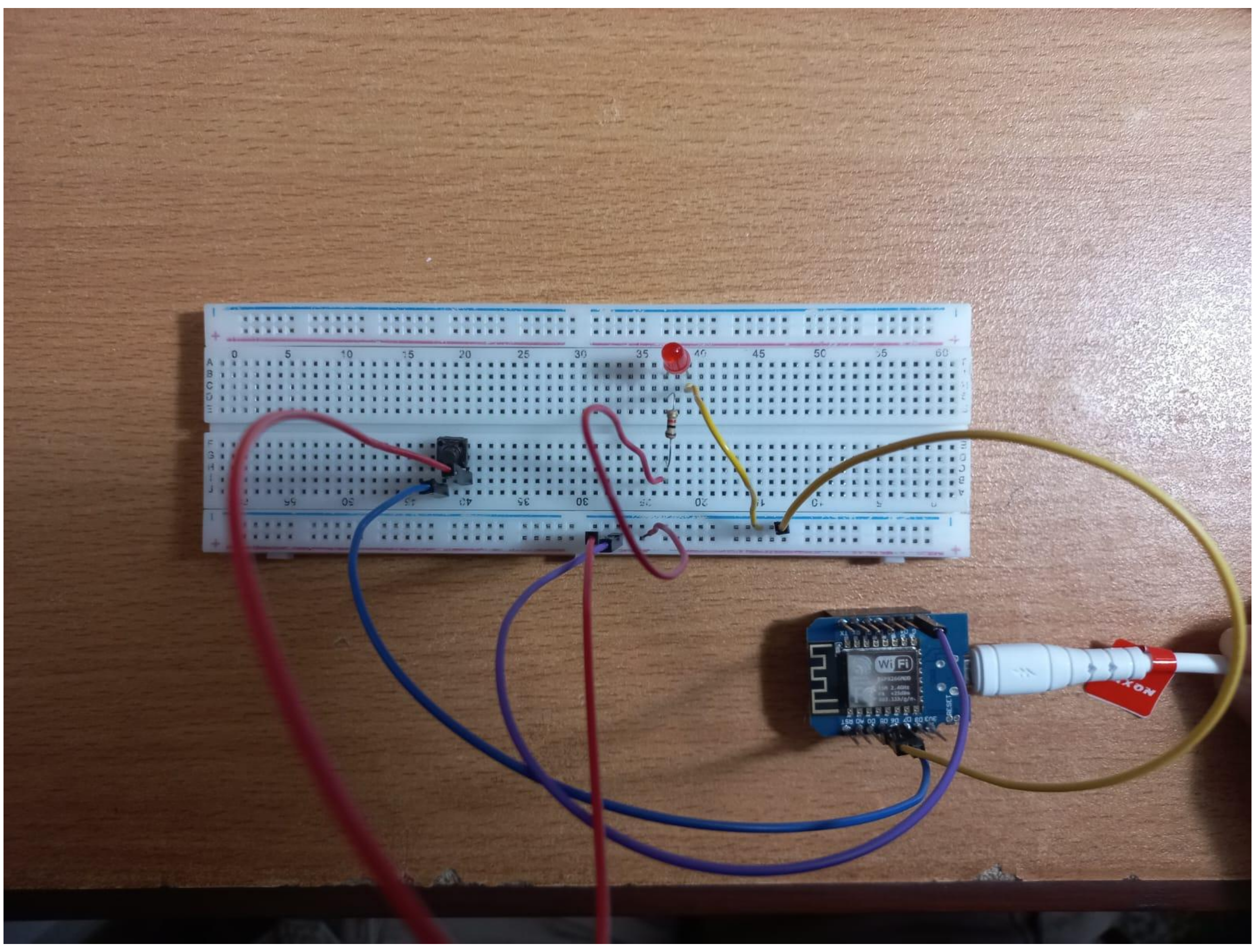
Circuit Diagram with internal Pullup resistor:

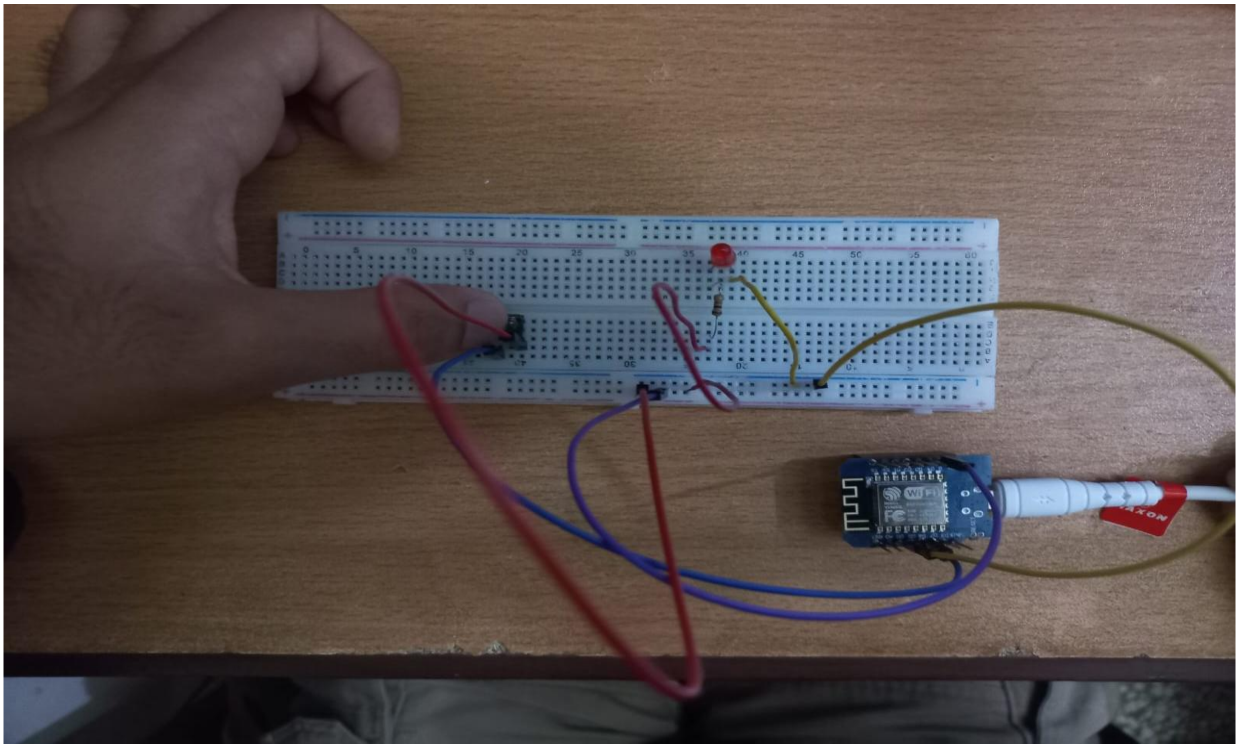


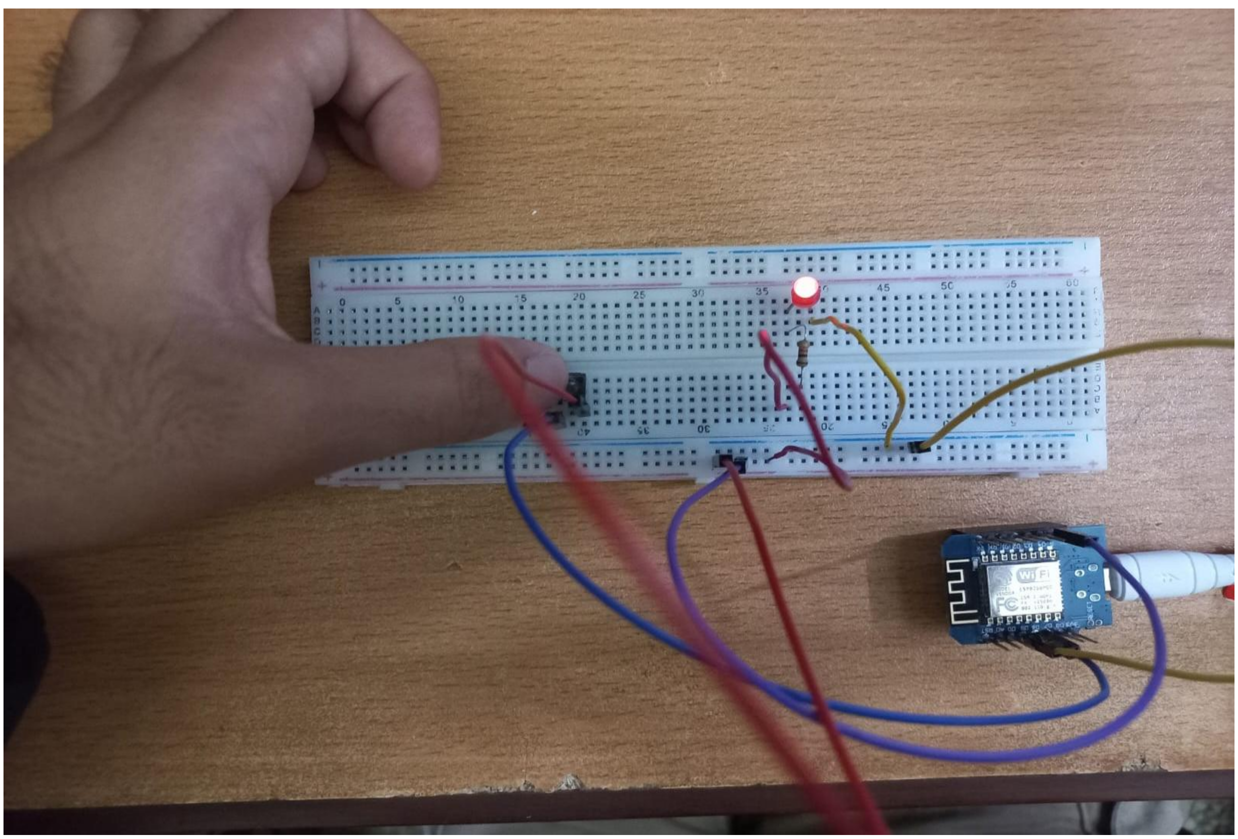
Code:

```
#define LED_PIN D5 //digital pin for LED
#define BUTTON_PIN D6 //digital pin for input button

void setup(){
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP); //using internal pull-up resistance
}

void loop(){
  //read digital state of input button
  if(digitalRead(BUTTON_PIN) == LOW){   //Input button pressed due to pull-up resistance
    digitalWrite(LED_PIN, HIGH);  //LED turn ON
  }
  else{
    digitalWrite(LED_PIN, LOW);   //LED turn OFF
  }
}
```

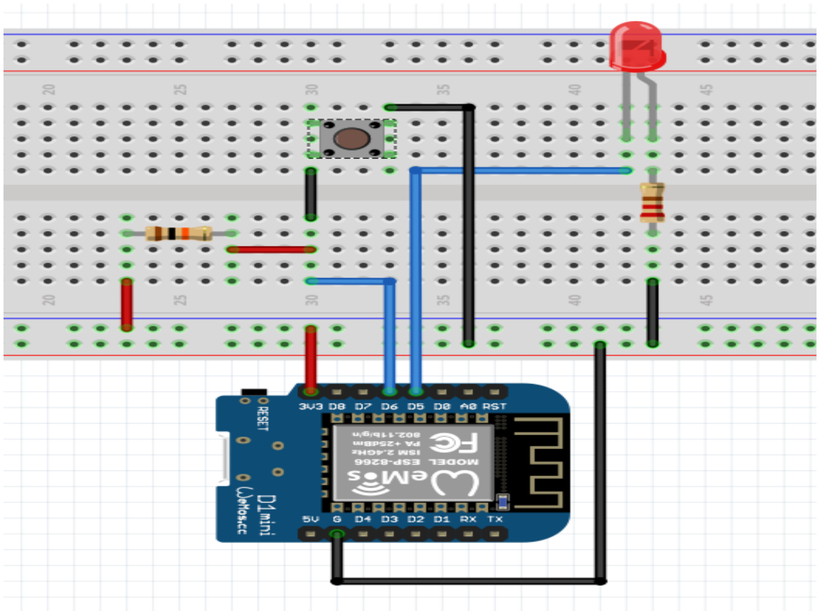
Output:







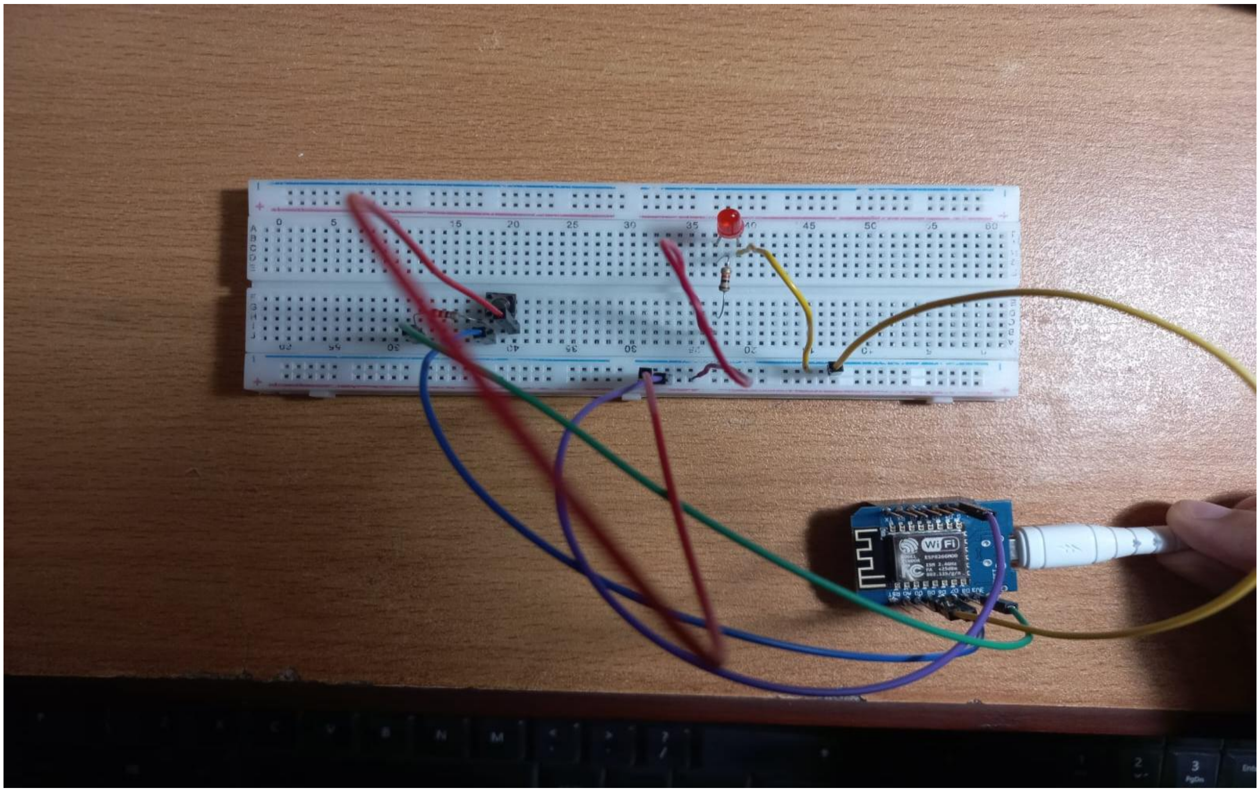
Circuit Diagram with external Pullup resistor:

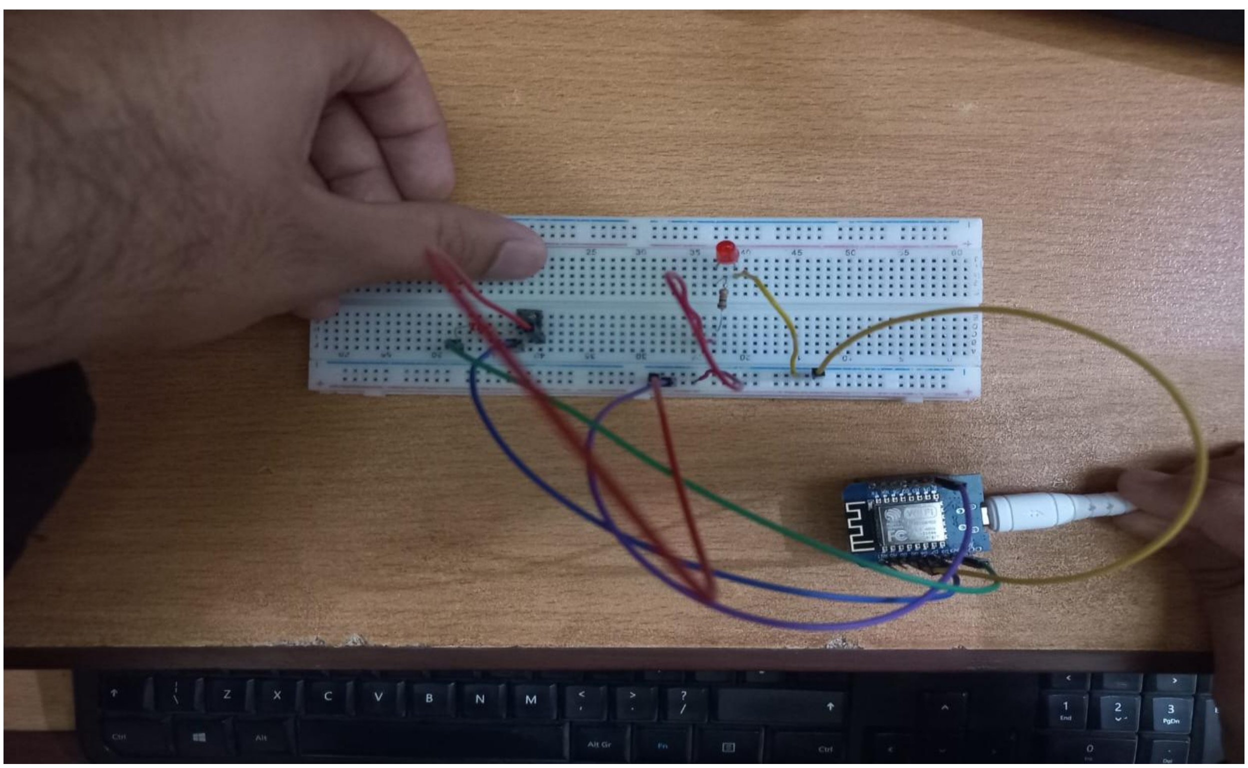


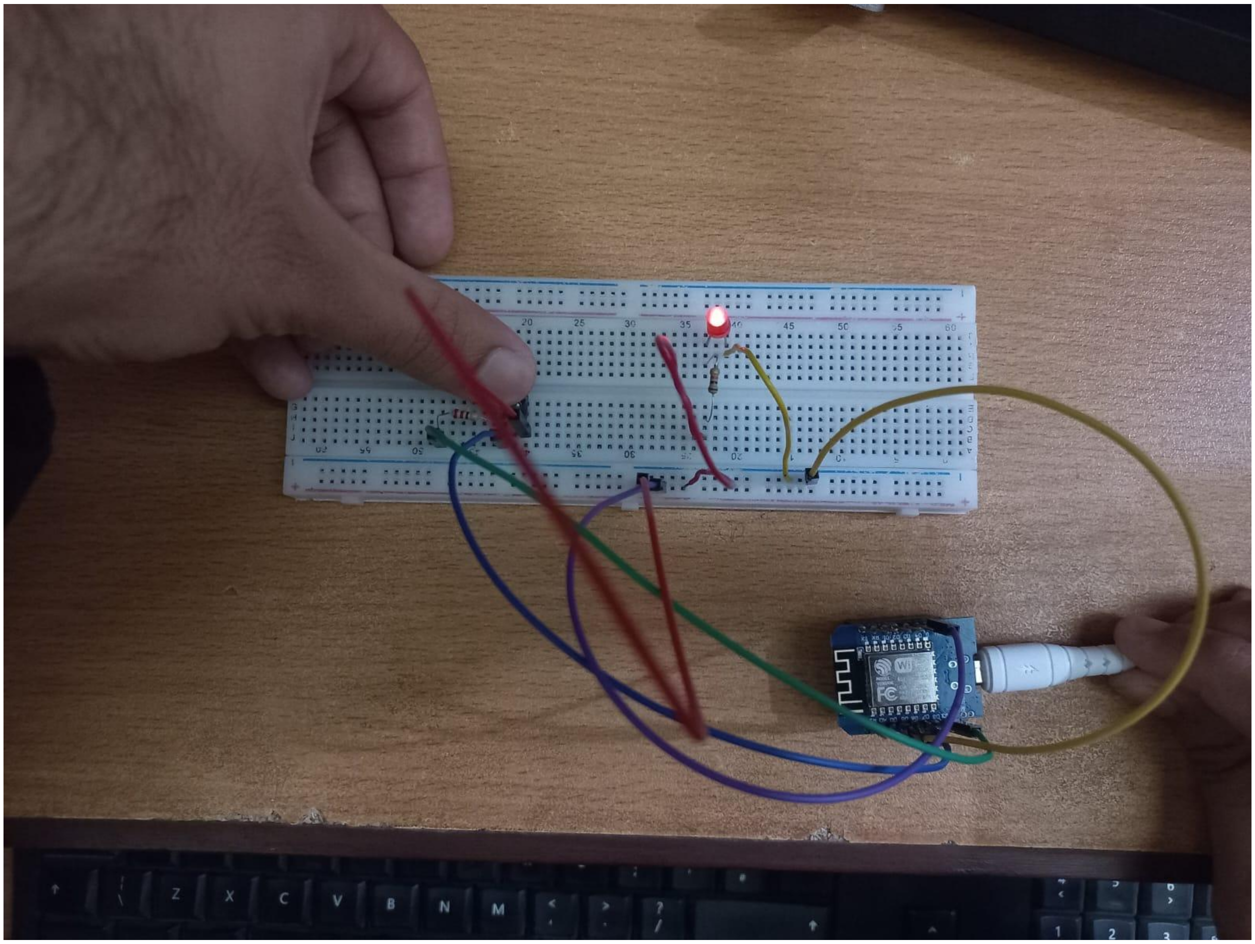
Code:

```
#define LED_PIN D5 //digital pin for LED
#define BUTTON_PIN D6 //digital pin for input button

void setup(){
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT); //using external pull-up resistance
}

void loop(){
  //read digital state of input button
  if(digitalRead(BUTTON_PIN) == LOW){   //Input button pressed due to pull-up resistance
    digitalWrite(LED_PIN, HIGH);   //LED turn ON
  }
  else{
    digitalWrite(LED_PIN, LOW);    //LED turn OFF
  }
}
```

Output:







Explanation:

• **Digital Input (BUTTON_PIN):** The push button is set up with an internal pull-up resistor, so its state is HIGH when not pressed and LOW when pressed.

• **Digital Output (LED_PIN):** The LED is controlled by setting the corresponding digital pin HIGH (on) or LOW (off).

• **Basic Operation:** When the button is pressed, the digital input reads LOW, and the LED is turned on; when released, it reads HIGH, and the LED is turned off.

Task 2: Implementing Debounce in Digital Input

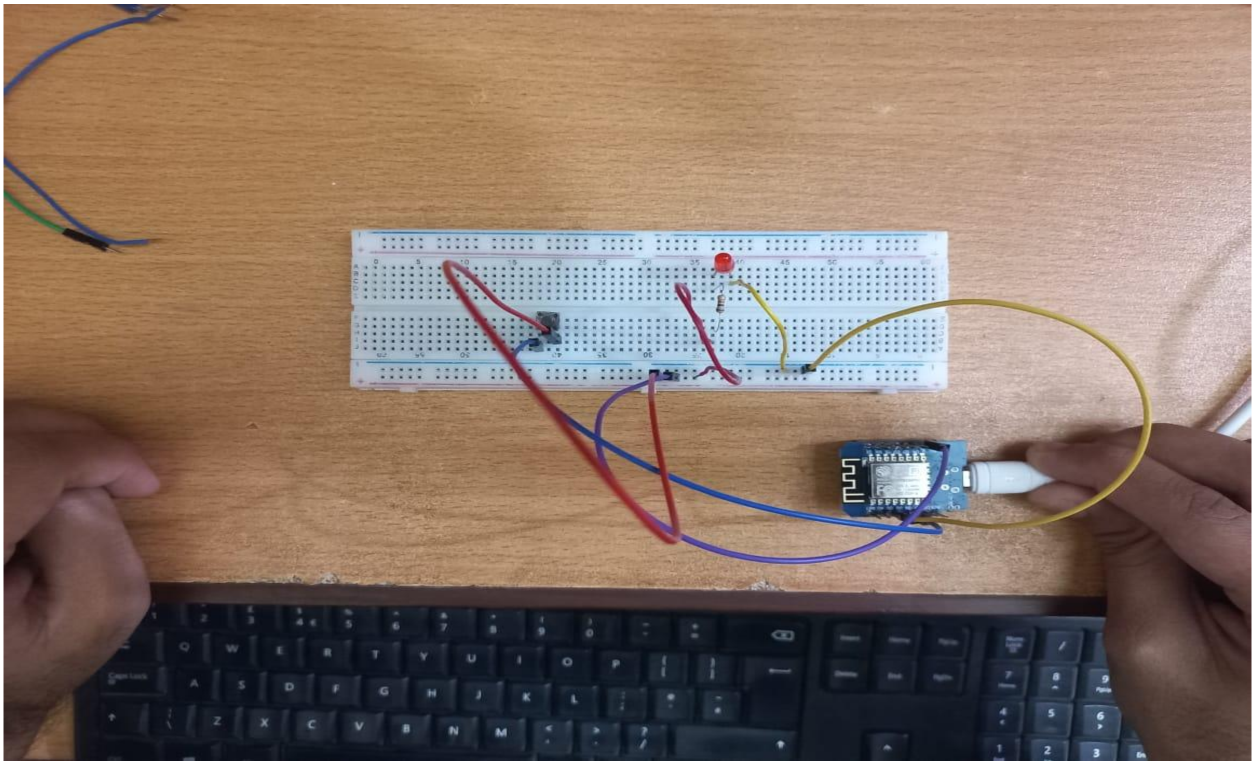
The Problem:

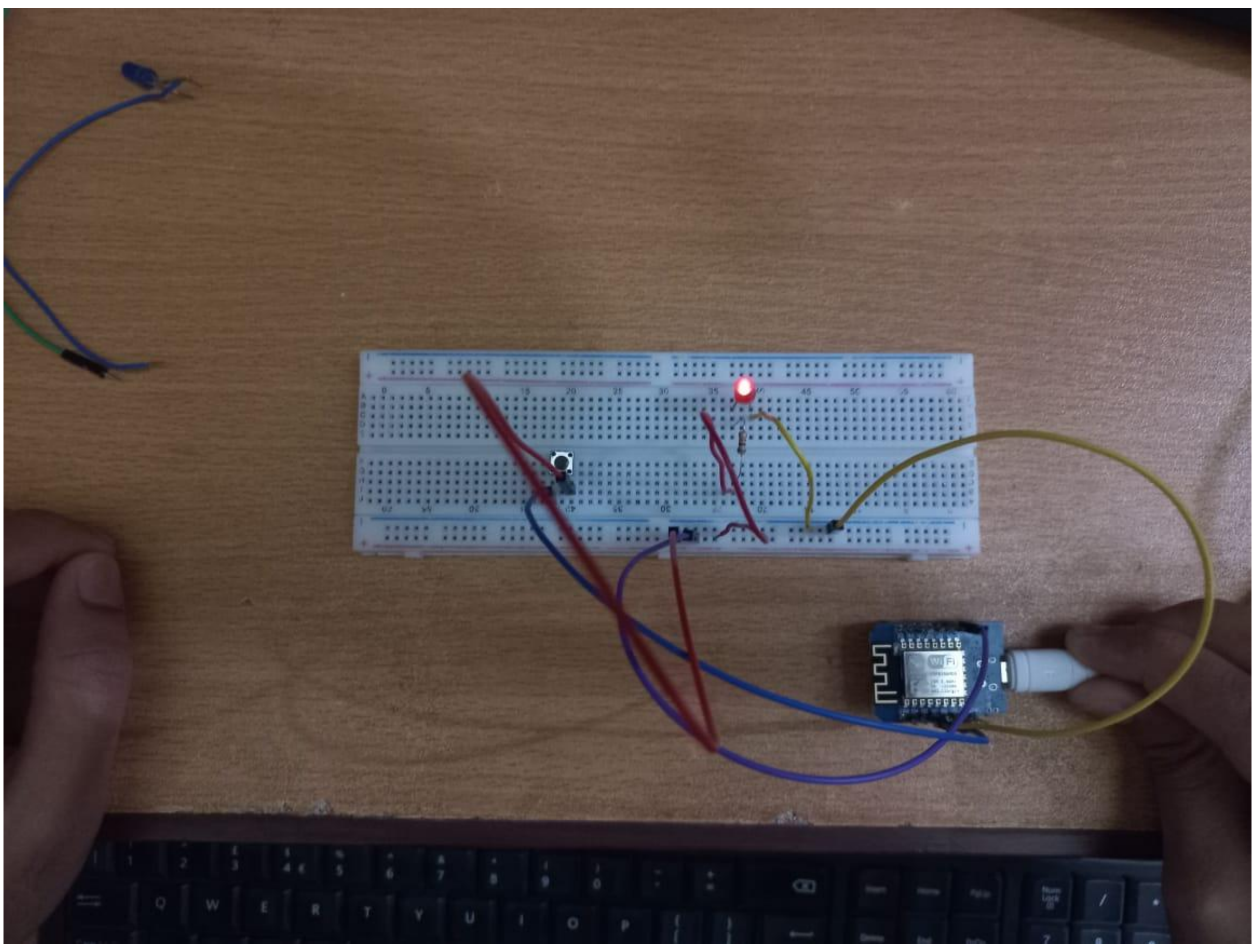
• Mechanical push buttons can produce spurious signals (bounces) which may result in multiple unwanted triggers.

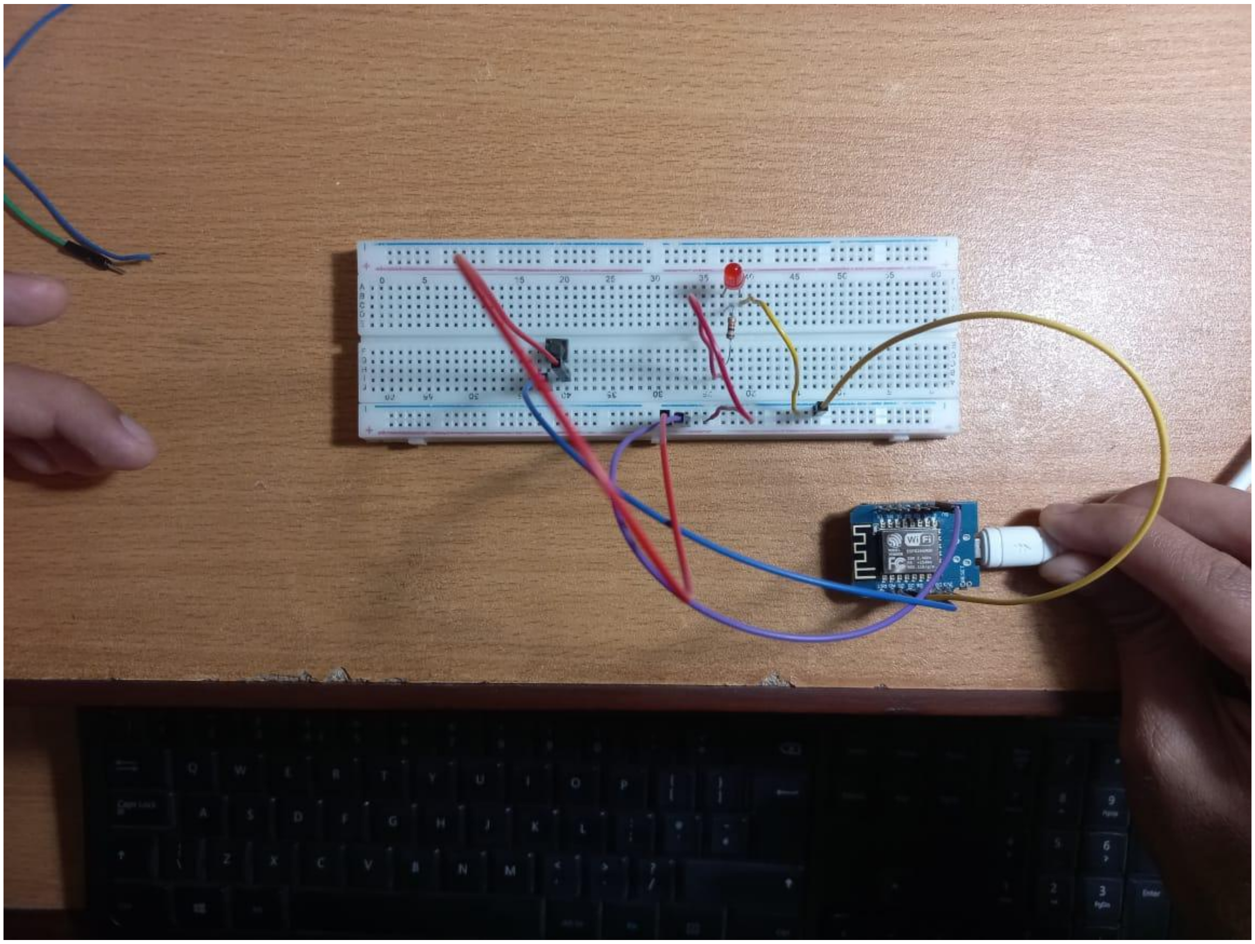
Code:

```
#define LED_PIN D5 //digital pin for LED
#define BUTTON_PIN D6 //digital pin for input button

bool ledState = false;
bool lastButtonState = HIGH;
unsigned long lastDebounceTime = 0;
const unsigned long debouncedDelay = 50;  //50ms debounce interval

void setup(){
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP); //using internal pull-up resistance
}

void loop(){
  //read digital state of input button
  bool reading = digitalRead(BUTTON_PIN);

  //check if the button state has changed
  if(reading != lastButtonState){
    lastDebounceTime = millis();  //  Reset the debounce timer
  }

  //  if the button state has remainedstable for the debounce interval
  if((millis() - lastDebounceTime) > debouncedDelay){
    if (reading == LOW){
      ledState = !ledState;   //  Toggle LED state on button press
      digitalWrite(LED_PIN, ledState);
    }
  }

  lastButtonState = reading;  //  Update the last known state
}
```

Output:







Explanation:

• **Debounce Logic:** The code uses millis() to measure the time since a change in button state was detected. Only if the state remains stable for more than 50 milliseconds does it register a valid press.

• **Toggling LED:** Instead of being on only while the button is pressed, the LED toggles its state with each valid press.

• This approach prevents false triggers caused by button bounce.

Additional Tasks (Optional):

3. Analog Dimming: Use PWM (analogWrite) to dim the LED when the button is held.

Code:

```
#define LED_PIN D5
#define BUTTON_PIN D6

int brightness = 0;  // LED brightness level (0-255)
int fadeAmount = 5;  // Step size for increasing/decreasing brightness

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT_PULLUP);  // Use internal pull-up
}

void loop() {
    if (digitalRead(BUTTON_PIN) == LOW) { // Button pressed
        brightness += fadeAmount;  // Increase brightness
        if (brightness > 255) brightness = 255; // Limit max brightness
    } else { // Button released
        brightness -= fadeAmount;  // Decrease brightness
        if (brightness < 0) brightness = 0; // Limit min brightness
    }

    analogWrite(LED_PIN, brightness);  // Apply brightness level
    delay(30); // Small delay to make dimming effect smooth
}

```

For implementing **analogWrite()** I use the same circuit with internal pull-up resistance but in code, I use one variable for the step size in brightness and the other for the initial brightness, and in the loop section if the button is pressed the brightness starts from 0 and increase by the step size of 5 in every 30 milliseconds until full brightness and when we release the button instead of instantly turn off the Led dims 5 steps in every 30 milliseconds until the brightness becomes 0.

Documentation and report

Design decisions:

For this lab, we choose VMOS d1 mini and we carefully choose pins according to the following table.

- For **Task 1**, first, we used the internal pull-up resistance, so we did not need to use an external resistor. For this, I enabled the internal pull-up resistance on D6 by using **pinMode(Button_Pin, INPUT_PULLUP);**. This is internally connected with +VCC, so we connect D6 with one terminal of the push button, and the other terminal is connected to the ground to implement the pull-up correctly. For the LED, one end is connected to D5, and the other is connected to the ground, so the LED will turn ON when it gets a High signal on D5. As we implement input as **PULLUP**, the **D6** will **normally be High** when the **button is not pressed**, and by code logic, **D5** will be **LOW**. When the **button is pressed**, **D6** gets **LOW**, and as a result, **LED_PIN D5** gets **High** and turns **ON**.

- Then, we implemented Task 1 using an external PULLUP resistor. In the code, we use **pinMode(Button_Pin, INPUT);** at D6, and for the external pull-up implementation, we connect one end of the push button to the ground and the other terminal to D6. Between them, I connected a **10k ohm** resistor with one end connected to **3.3V** so that **D6** gets **3.3V (HIGH)** when the **button is not pressed**; otherwise, it will get **0 (LOW).**

- For **Task 2**, we use the same circuit as in Task 1 (with internal PULLUP resistance) to implement the debouncing logic. In the code, we use four variables for LED state, button state tracking, delay, and time counter. The **LED state toggles only when the button is pressed.** It keeps checking the state of the input button, and if the button's current state is different from the previously stored state, then it resets a counter and checks the difference between the timer and the current time **millis**(). If it is greater than the delay, then it checks if the button state is low to toggle the state of the LED. By using **ledState**, we can store the state of the LED, and similarly, by using **lastButtonState**, we can store the state of the input button, and we can use them further for different tasks.

Label	GPIO	Input	Output	Notes
D0	GPIO16	no interrupt	no PWM or I2C support	HIGH at boot used to wake up from deep sleep
D1	GPIO5	OK	OK	often used as SCL (I2C)
D2	GPIO4	OK	OK	often used as SDA (I2C)
D3	GPIO0	pulled up	OK	connected to FLASH button, boot fails if pulled LOW
D4	GPIO2	pulled up	OK	HIGH at boot connected to on-board LED, boot fails if pulled LOW
D5	GPIO14	OK	OK	SPI (SCLK)
D6	GPIO12	OK	OK	SPI (MISO)
D7	GPIO13	OK	OK	SPI (MOSI)
D8	GPIO15	pulled to GND	OK	SPI (CS) Boot fails if pulled HIGH
RX	GPIO3	OK	RX pin	HIGH at boot
TX	GPIO1	TX pin	OK	HIGH at boot debug output at boot, boot fails if pulled LOW
A0	ADC0	Analog Input	X	

Challenges:

- The microcontroller we received in the lab did not have pre-soldered pins, so we had to attach them manually. We performed soldering in the lab to prepare the controller for use.
- Careful arrangement of components on the board was necessary to implement the circuit efficiently.

Learning outcomes:

Controlling an LED with a Push Button:

- Control an LED using a push button.
- Develop logic in code to read input and make respective decisions (such as turning the LED ON and OFF).
- Implement both internal and external pull-up resistances.

Debounce in Digital Input:

- Simulate different behaviors of the push button from Task 1.
- Implement logic for debouncing to prevent unwanted triggers.

Analog Dimming:

- Learn the use of the analogWrite() function, which simulates an analog system. Its value ranges from 0 to 255, where 0 represents OFF and 255 represents ON. By setting a value in the analogWrite() function, we can vary the voltage within this range.
- Understand that the analogWrite() function uses Pulse Width Modulation (PWM). PWM is a technique for achieving analog results using digital signals. Digital control creates a square wave—an on-and-off signal. By varying the proportion of time, the signal stays ON versus OFF, we can simulate different voltage levels between 0V (OFF) and the board's full voltage (e.g., 5V on an Arduino UNO or 3.3V on an MKR board).
- The duration of the "on time" is called the pulse width. To obtain varying analog values, we adjust (or modulate) the pulse width.