Practical IoT (Internet of Things)

BSCIS – DCIS, PIEAS

Lab 01: Digital Input and Output

Objective

- 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

Key Concepts

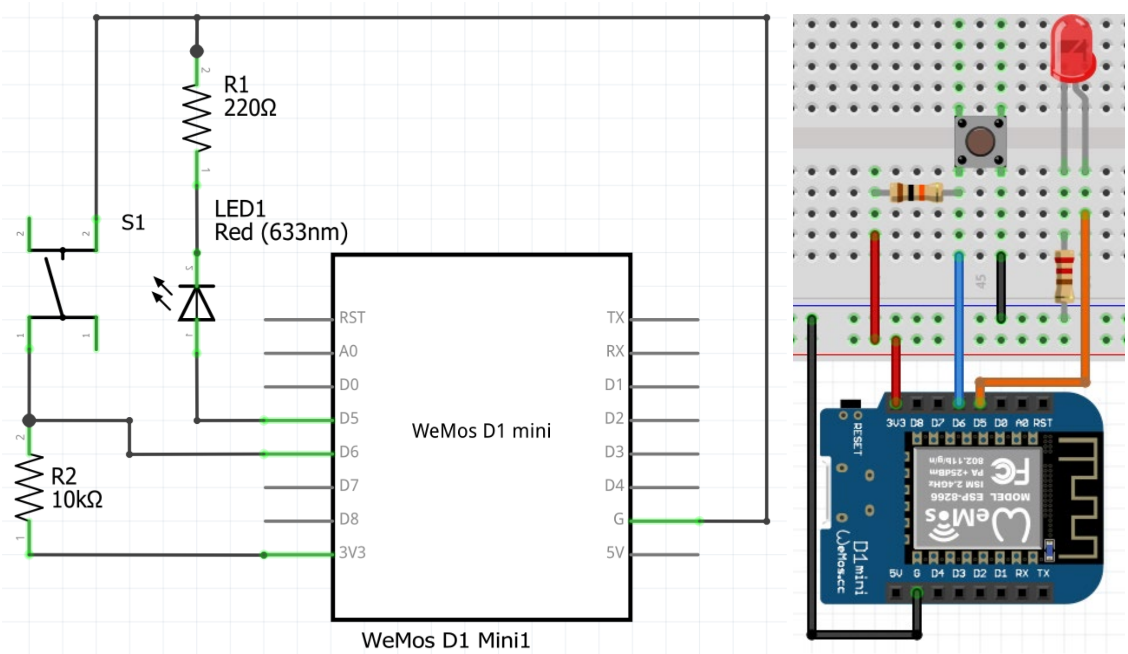
1. **Digital Signals:**
    - Represented as either HIGH (1) or LOW (0). They are used to indicate binary states.
2. **Digital Input:**
    - A signal that has two states: HIGH (3.3V) or LOW (GND).
    - Example: Push button (open = HIGH, pressed = LOW).
3. **Digital Output:**
    - A signal that can be controlled as HIGH (3.3V) or LOW (GND).
    - Example: LED (HIGH = on, LOW = off).
4. **Pull-Up Resistors:**
    - Used to ensure a stable default state (HIGH) for digital inputs.
5. **Debouncing:**
    - A technique used to ensure a single, clean signal is read when a mechanical switch is pressed, mitigating the effects of noise and contact bounce.

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

Circuit Diagram:

- **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)



Code: Basic Button Control

```
#define LED_PIN D5     // Digital pin for LED
#define BUTTON_PIN D6 // Digital pin for push button

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

void loop() {
    // Read the digital state of the push button
    if (digitalRead(BUTTON_PIN) == LOW) {  // Button pressed (LOW due to pull-up
        digitalWrite(LED_PIN, HIGH);        // Turn LED ON
    } else {
        digitalWrite(LED_PIN, LOW);         // Turn LED OFF
    }
}
```

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: Debounced Button Control

```
#define LED_PIN D5
#define BUTTON_PIN D6

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

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
    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 remained stable for the debounce interval
    if ((millis() - lastDebounceTime) > debounceDelay) {
        if (reading == LOW) {
            ledState = !ledState;  // Toggle LED state on button press
            digitalWrite(LED_PIN, ledState);
        }
    }

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

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 the LED being on only while the button is pressed, it toggles its state with each valid press.
- This approach prevents false triggers caused by button bounce.

Additional Tasks (Optional) – if you want to learn more:

1. **Expand to Multiple LEDs:** Modify the code to control multiple LEDs using the same push button.
2. **Multi-Button Control:** Add a second button to control another LED.
3. **Analog Dimming:** Use PWM (analogWrite) to dim the LED when the button is held.

Submission Requirements:

Submit a single PDF containing:

- **Your Arduino Code:** The Arduino sketches for both the basic and debounced button control.
- **Circuit Diagram:** A photo or schematic of your breadboard setup.
- **Short Description:** A 1-2 page explanation of your design choices, challenges faced, and lessons learned regarding digital I/O in IoT.

Conclusion:

- Understanding digital input and output is fundamental to IoT systems.
- Proper debouncing techniques ensure reliable performance in real-world applications.
- This lab lays the groundwork for more complex digital interfacing in future projects.

Grading Rubric:

Criterion	Points	Description
Circuit Setup	30	Correct wiring and clear circuit diagram for push button & LED.
Basic I/O Code	20	Functionality of basic button-controlled LED code.
Debounce Implementation	30	Effective debounce logic in code, preventing multiple triggers.
Documentation & Report	20	Clear explanation of digital signals, design decisions, and outcomes.