Title: Simple LED Control with a Button

Introduction: This experiment demonstrates how to control an LED using a push button on the STM32 Bluepill (STM32F103C8T6) microcontroller. It introduces essential concepts in digital input/output (I/O) operations by using a button press to change the LED state. Such applications are foundational in embedded system design and form the basis for creating user interfaces, menus, and event-based systems.

Circuit Diagram and description :

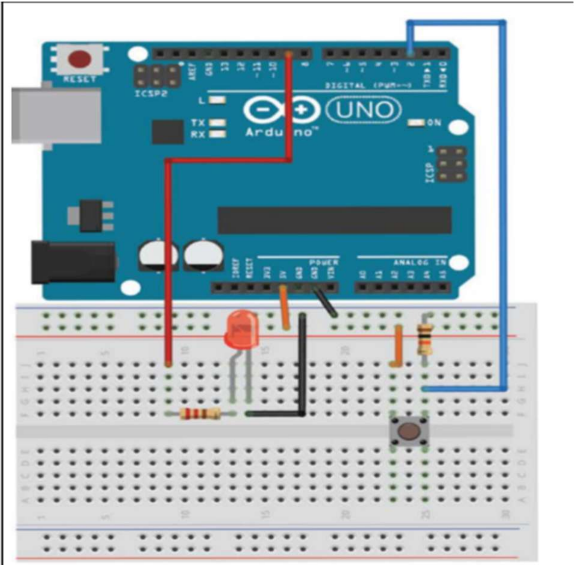

Figure 2-6: Wiring an Arduino to a button and an LED

Circuit Diagram

- **Microcontroller:** STM32F103C8T6 (Bluepill)
- **LED:** Connected to GPIO pin PB0 (through a 220Ω current-limiting resistor)
- **Push Button:** Connected to GPIO pin PC15
- **Connections:**
    - LED anode → PB0 through 220Ω resistor
    - LED cathode → GND
    - Button one terminal → PC15
    - Button the other terminal → GND
    - PC15 uses an internal pull-up to remain HIGH when the button is not pressed

Procedure:

1. Setting up the hardware:
    - We connected the LED's anode to PB0 via a 220Ω resistor.

    - Then, the LED's cathode is connected to GND.
    - Connected one terminal of the push button to PC15 and the other to GND.
2. Used internal pull-up on PC15 to ensure the input reads HIGH when not pressed.
3. Using the Arduino framework, we wrote and uploaded the code to the Bluepill board.
4. Observe the LED turning ON when the button is **not pressed** and OFF when the button is **pressed**.

Code:

```
const int LED = PB0;

const int BUTTON = PC15;

void setup() {

  pinMode(LED, OUTPUT);

  pinMode(BUTTON, INPUT_PULLUP); // Enable internal pull-up resistor

}

void loop() {

  if (digitalRead(BUTTON) == LOW) {

    digitalWrite(LED, LOW); // Turn OFF LED when button is pressed

  } else {

    digitalWrite(LED, HIGH); // Turn ON LED when button is not pressed

  }

}
```

Discussion: By interfacing an LED with a push button, I learned how to use GPIO pins effectively and how internal pull-up resistors simplify input configurations. One key takeaway was understanding the logic level behavior of digital inputs—specifically how a button press can bring a pin from HIGH to LOW when connected to ground, and how internal pull-up resistors can help avoid floating states.

I also became more familiar with the STM32 programming environment using the Arduino framework, which provided an accessible way to write and upload code to the microcontroller. Although there were some initial challenges with setting up the ST-Link uploader, I eventually overcame them and successfully uploaded the code.