Programmed Control with Arduino

Activity 4: The Latching Pushbutton (Bistable Operation)

In the previous activity, we learned how to control an LED momentarily (the LED turns off immediately when the button is released). Today, we will solve a real-world problem: how do we create a light switch like the one in our bedroom, where a single click turns the light on and another click turns it off? To achieve this, we will teach Arduino to "remember" states using variables and to detect with mathematical precision the exact instant the button is pressed.

Learning Objectives

By the end of this session, you will be able to:

- Understand the operational difference between astable (momentary) and bistable (latching) systems.
- Implement state variables as internal memory for the microcontroller.
- Design an algorithm to detect a **rising edge** (the transition from LOW to HIGH on an input pin).
- Understand the physical concept of mechanical contact bounces (*bouncing*) in pushbuttons and how to mitigate them via software.

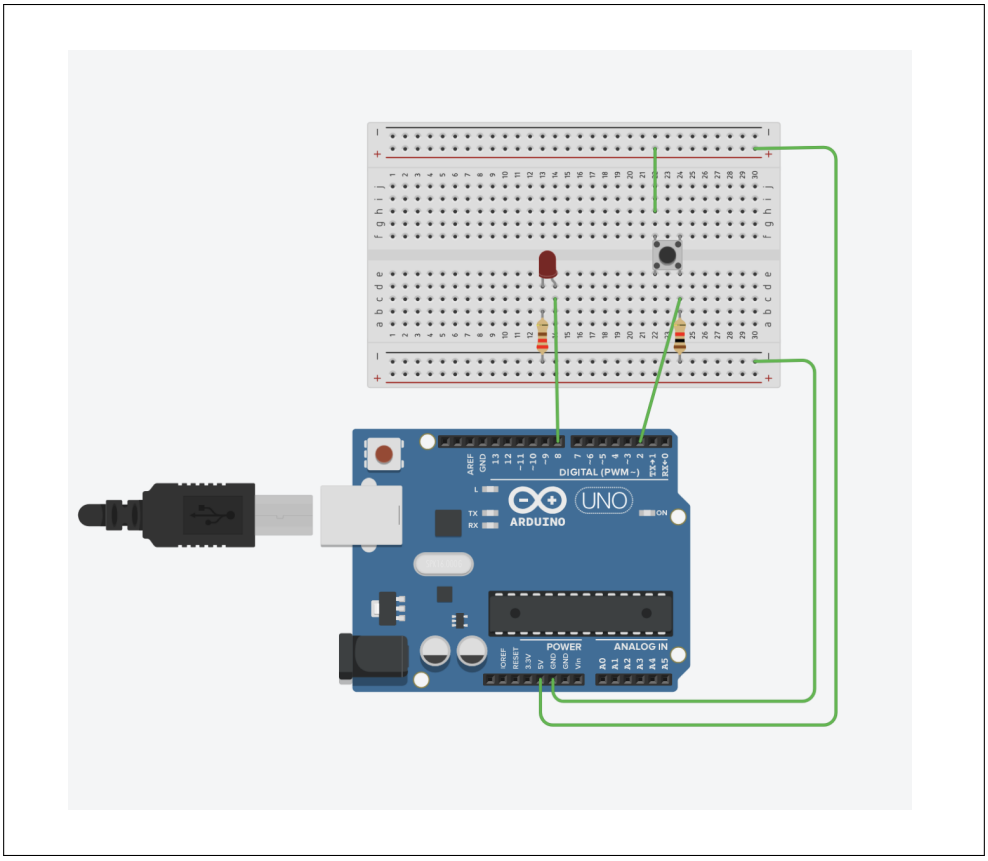
Required Components

Do not touch your previous circuit! This activity uses the exact same connections as Activity 3. We will focus all our effort and time on mastering the programming code.

- 1 Arduino Uno board.
- 1 Small breadboard.
- 1 LED and 1 $220\,\Omega$ resistor.
- 1 Four-pin pushbutton and 1 $10\,\text{k}\Omega$ resistor (for the Pull-Down circuit).
- Jumper wires.

Breadboard Assembly Diagram

Review your assembly in TinkerCad to ensure it retains the connections from the previous activity so the program responds correctly.



Base Code: Rising Edge Detection

Write the following program carefully in your TinkerCad editor. Pay close attention to how we combine the current reading of the pushbutton with the reading obtained from the previous program iteration.

```
// Physical pin declaration
int led = 8;
int pushbutton = 2;

// State control variables
int ledState = LOW;             // Stores the current state of the LED (LOW or HIGH)
int currentReading;              // Stores the instantaneous reading of the pushbutton
int previousReading = LOW;       // Stores the reading value from the previous iteration

void setup() {
  pinMode(led, OUTPUT);
  pinMode(pushbutton, INPUT);
}

void loop() {
  currentReading = digitalRead(pushbutton); // Read the pushbutton pin

  % If the pushbutton is pressed (HIGH) AND it was released in the previous iteration (LOW)
  if (currentReading == HIGH && previousReading == LOW) {

    // Toggle the state (if it was LOW it becomes HIGH, if it was HIGH it becomes LOW)
    if (ledState == HIGH) {
      ledState = LOW;
    } else {
      ledState = HIGH;
    }

    // Physically apply the new state to the LED pin
    digitalWrite(led, ledState);

    // Brief pause (debouncing) to ignore initial electrical noise
    delay(50);
  }

  % Store the current reading to be used as previous in the next iteration
  previousReading = currentReading;
}
```

Listing 1: Code to toggle the state of an LED with each press.

How does the code work?

The Arduino `loop` runs thousands of times per second. If we only checked whether the button is `HIGH`, the LED would toggle between on and off thousands of times in the fraction of a second it takes our finger to release the pushbutton. This would make the final state when lifting our finger completely unpredictable.

- **The logical AND operator ( )**: This operator requires both specified conditions to be met simultaneously. In our code: the button must be pressed *now* (`HIGH`) and it must have been released *just an instant before* (`LOW`). This defines the transition or "rising edge", which occurs only once per button press.
- **The state variable ( )**: Acts as the memory of the circuit. Instead of turning on the LED directly, we first toggle the value of this internal variable and then use `digitalWrite` to reflect it physically.
- **Mechanical Bounce and Debouncing Delay**: When the metal pins of a physical pushbutton clash, they bounce and generate rapid, false electrical connections (electrical noise). The line `delay` briefly pauses the microcontroller to let these initial bounces pass, ensuring a stable reading.

Activity 4 Challenge

Demonstrate your mastery of bistable logic and edge management by addressing the following proposals:

1. **Simplified Logic (NOT Operator)**: In programming, there is a logical operator that inverts any boolean value: the exclamation mark ( ). Modify the base code by replacing the 6 lines of the `if else` block that handles the toggling of `ledState` with the following single-line expression:

   ```
   ledState = !ledState;
   ```

   Verify in TinkerCad that the program still works exactly as intended and analyze why this happens.

2. **Intensity Selector (Power Traffic Light)**: Modify the code so that a single button sequentially cycles through three different states with each press instead of two:
   - First press: A Red LED turns on (Pin 8).
   - Second press: The Red LED turns off and a Green LED turns on (Pin 9).
   - Third press: Both LEDs turn off (restarting the cycle).

3. *Question for reflection*: If we remove the line `previousReading = currentReading;`, what do you think will happen to our program when the button is pressed? Explain your reasoning.