Control programado con Arduino

Práctica 4: El Pulsador con Enclavamiento (Funcionamiento Biestable)

En la práctica anterior aprendimos a controlar un LED de manera momentánea (el LED se apaga inmediatamente al soltar el botón). Hoy daremos solución a un problema del mundo real: ¿cómo creamos un interruptor como el de nuestra habitación, que con un solo clic encienda la luz y con otro clic la apague? Para lograrlo, enseñaremos a Arduino a "recordar" estados utilizando variables y a detectar con precisión matemática el instante exacto en el que el botón es presionado.

Objetivos de Aprendizaje

Al finalizar esta sesión, serás capaz de:

- Comprender la diferencia operativa entre sistemas astables (momentáneos) y biestables (con enclavamiento).
- Implementar variables de estado como memoria interna del microcontrolador.
- Diseñar un algoritmo para detectar el **flanco de subida** (el cambio de LOW a HIGH de un pin de entrada).
- Entender el concepto físico de los rebotes mecánicos (*bouncing*) en pulsadores y cómo mitigarlos por software.

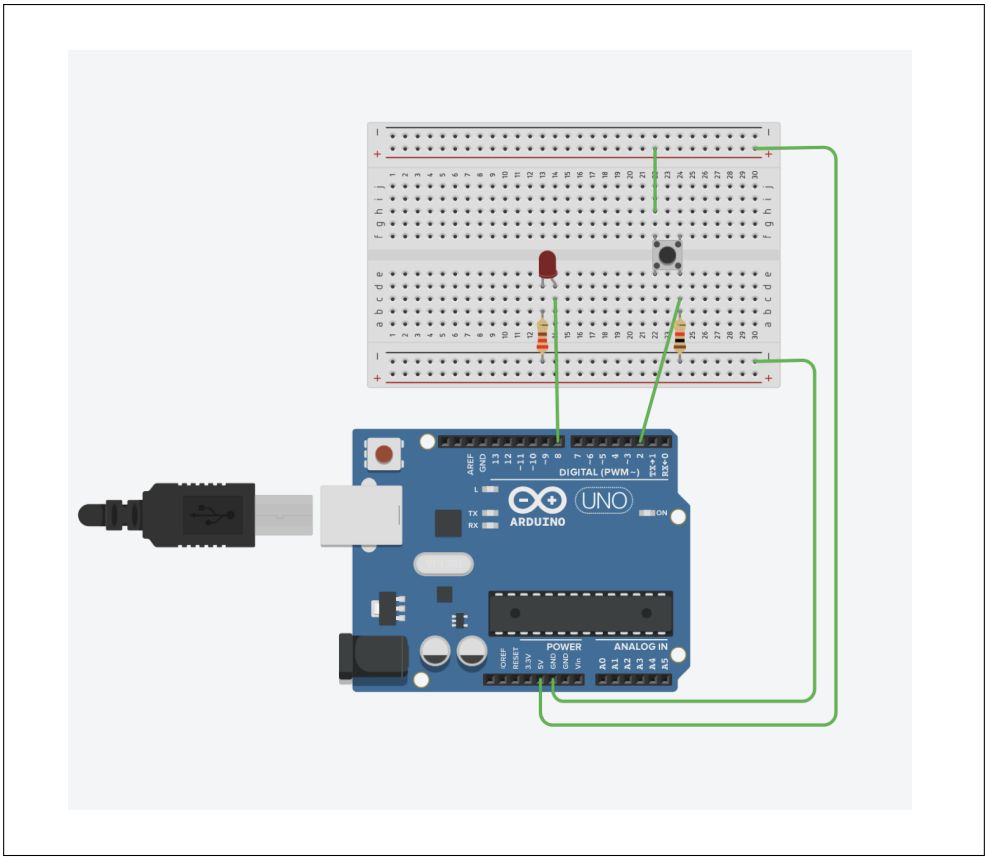
Componentes Necesarios

¡No toques tu circuito anterior! Esta práctica utiliza exactamente las mismas conexiones que la Práctica 3. Centraremos todo nuestro esfuerzo y tiempo en dominar el código de programación.

- 1 Placa Arduino Uno.
- 1 Placa de pruebas pequeña (Protoboard).
- 1 Diodo LED y 1 resistencia de $220\,\Omega$.
- 1 Pulsador de cuatro patillas y 1 resistencia de $10\,\text{k}\Omega$ (para el circuito Pull-Down).
- Cables de conexión.

Esquema de Montaje en la Protoboard

Revisa que tu montaje en TinkerCad mantenga las conexiones que realizamos en la práctica anterior para asegurar que el programa responda correctamente.



Código Base: Detección del Flanco de Subida

Escribe con mucha atención el siguiente programa en tu editor de TinkerCad. Fíjate detalladamente en cómo combinamos la lectura actual del pulsador con la lectura que se obtuvo en la vuelta anterior del programa.

```
// Declaracion de pines fisicos
int led = 8;
int pulsador = 2;

// Variables de control de estado
int estadoLed = LOW;              // Guarda el estado actual del LED (LOW o HIGH)
int lecturaActual;                // Almacena la lectura instantanea del pulsador
int lecturaAnterior = LOW;        // Almacena el valor de la lectura de la vuelta anterior

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

void loop() {
  lecturaActual = digitalRead(pulsador); // Leemos el pin del pulsador

  // Si el pulsador esta presionado (HIGH) Y en la vuelta anterior estaba suelto (LOW)
  if (lecturaActual == HIGH && lecturaAnterior == LOW) {

    // Cambiamos de estado (si estaba LOW pasa a HIGH, si estaba HIGH pasa a LOW)
    if (estadoLed == HIGH) {
      estadoLed = LOW;
    } else {
      estadoLed = HIGH;
    }

    // Aplicamos fisicamente el nuevo estado al pin del LED
    digitalWrite(led, estadoLed);

    // Pequena pausa (anti-rebote) para ignorar los ruidos electricos iniciales
    delay(50);
  }

  // Guardamos la lectura actual para usarla como anterior en la siguiente iteracion
  lecturaAnterior = lecturaActual;
}
```

Listing 1: Código para alternar el estado de un LED con cada pulsación.

¿Cómo funciona el código?

El bucle `loop` de Arduino se ejecuta miles de veces por segundo. Si solo evaluáramos si el botón está en `HIGH`, el LED cambiaría de encendido a apagado miles de veces en la décima de

segundo que nuestro dedo tarda en soltar el pulsador. Esto provocaría que el estado final al levantar el dedo fuera totalmente impredecible.

- **El operador lógico AND ( )**: Este operador exige que se cumplan obligatoriamente las dos condiciones especificadas a la vez. En nuestro código: el botón debe estar presionado *ahora* (`HIGH`) y haber estado suelto *un instante antes* (`LOW`). Esto define la transición o "flanco de subida", que solo ocurre una única vez por cada pulsación.
- **La variable de estado ( )**: Actúa como la memoria del circuito. En lugar de encender el LED directamente, cambiamos primero el valor de esta variable interna y luego usamos `digitalWrite` para reflejarlo físicamente.
- **El rebote mecánico y el Retardo de Estabilización**: Cuando las patillas metálicas de un pulsador físico chocan, rebotan y generan falsas conexiones ultrarrápidas (ruido eléctrico). La línea `delay` detiene brevemente al microcontrolador para dejar pasar estos rebotes iniciales y asegurar una lectura estable.

El Reto de la Práctica 4

Demuestra que dominas la lógica biestable y el manejo de flancos con las siguientes propuestas:

1. **Lógica Simplificada (Operador NOT)**: En programación existe un operador lógico que invierte cualquier valor booleano: el signo de exclamación ( ). Modifica el código base reemplazando las 6 líneas del bloque `if else` que decide el cambio de `estadoLed` por la siguiente expresión de una sola línea:

   ```
   estadoLed = !estadoLed;
   ```

   Verifica en TinkerCad que el programa sigue funcionando exactamente igual de bien y analiza por qué ocurre esto.

2. **Selector de Intensidades (Semáforo de Potencia)**: Modifica el código para que un único botón sirva para alternar de manera secuencial entre tres estados distintos con cada pulsación en lugar de dos:
   - Primera pulsación: Se enciende un LED Rojo (Pin 8).
   - Segunda pulsación: Se apaga el LED Rojo y se enciende un LED Verde (Pin 9).
   - Tercera pulsación: Se apagan ambos LEDs (reiniciando el ciclo).

3. *Pregunta para reflexionar*: Si quitamos la línea `lecturaAnterior = lecturaActual;`, ¿qué crees que le pasará a nuestro programa al pulsar el botón? Razona tu respuesta.