Control programado con Arduino

Práctica 4: O pulsador con enclavamento (Funcionamento biestable)

Na práctica anterior aprendemos a controlar un LED de xeito momentáneo (o LED apágase inmediatamente ao soltar o botón). Hoxe daremos solución a un problema do mundo real: como creamos un interruptor coma o da nosa habitación, que cun só clic acenda a luz e con outro clic a apague? Para logralo, ensinarémoslle a Arduino a "lembrar" estados utilizando variables e a detectar con precisión matemática o instante exacto no que o botón é premido.

Obxectivos de Aprendizaxe

[1pt]

Ao rematar esta sesión serás capaz de:

- Comprender a diferenza operativa entre sistemas astables (momentáneos) e biestables (con enclavamento).
- Implementar variables de estado como memoria interna do microcontrolador.
- Deseñar un algoritmo para detectar o **flanco de subida** (o cambio de LOW a HIGH dun pin de entrada).
- Entender o concepto físico dos rebotes mecánicos (*bouncing*) en pulsadores e como mitigalos por software.

Compoñentes Necesarios

[1pt]

Non toques o teu circuíto anterior! Esta práctica utiliza exactamente as mesmas conexións que a Práctica 3. Centraremos todo o noso esforzo e tempo en dominar o código de programación.

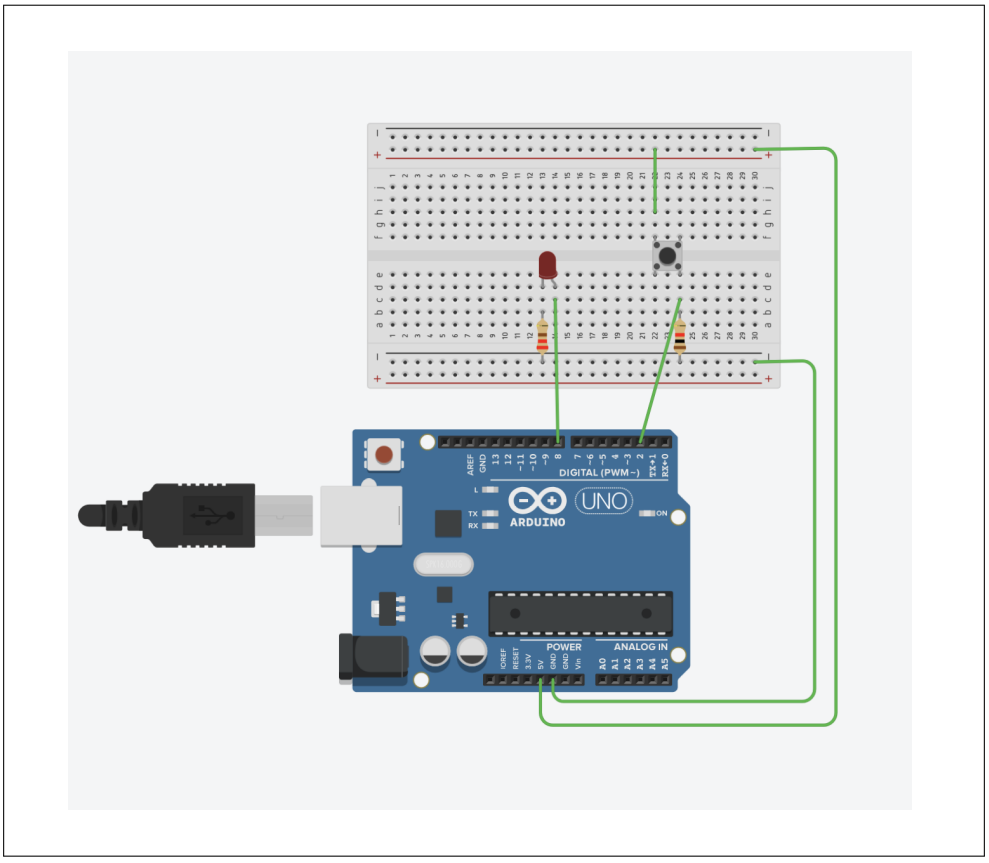
- 1 Placa Arduino Uno.
- 1 Placa de prototipado pequena (Protoboard).
- 1 Diodo LED e 1 resistencia de $220\,\Omega$.

- 1 Pulsador de catro patillas e 1 resistencia de $10\,\mathrm{k}\Omega$ (para o circuíto Pull-Down).
- Cables de Amazon de conexión.

Esquema de montaxe na Protoboard

[1pt]

Revisa que a túa montaxe en TinkerCad manteña as conexións que realizamos na práctica anterior para asegurar que o programa responda correctamente.



Código Base: Detección do Flanco de Subida [1pt]

Escribe con moita atención o seguinte programa no teu editor de TinkerCad. Fíxate detalladamente en como combinamos a lectura actual do pulsador coa lectura que se obtivo na volta anterior del programa.

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

// Variables de control de estado
int estadoLed = LOW;              // Garda o estado actual do LED (LOW o HIGH)
int lecturaActual;                // Almacena a lectura instantanea do pulsador
int lecturaAnterior = LOW;        // Almacena o valor da lectura da volta
    anterior

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

void loop() {
  lecturaActual = digitalRead(pulsador); // Lemos o pin do pulsador

  // Se o pulsador esta premido (HIGH) E na volta anterior estaba solto (LOW)
  if (lecturaActual == HIGH && lecturaAnterior == LOW) {

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

    // Aplicamos fisicamente o novo estado ao pin do LED
    digitalWrite(led, estadoLed);

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

  // Gardamos a lectura actual para usala como anterior na seguinte iteracion
  lecturaAnterior = lecturaActual;
}
```

Listing 1: Código para alternar o estado dun LED con cada pulsación.

Como funciona o código?

[1pt]

O bucle `loop` de Arduino execútase miles de veces por segundo. Se só avaliaramos se o botón está en `HIGH`, o LED cambiaría de acendido a apagado miles de veces na décima de segundo que o noso dedo tarda en soltar o pulsador. Isto provocaría que o estado final ao levantar o dedo fose totalmente impredecible.

- **O operador lóxico AND ( )**: Este operador esixe que se cumpran obrigatoriamente as dúas condicións especificadas á vez. No noso código: o botón debe estar premido *agora* (`HIGH`) e ter estado solto *un instante antes* (`LOW`). Isto define a transición ou "flanco de subida", que só ocorre unha única vez por cada pulsación.
- **A variable de estado ( )**: Actúa como a memoria do circuíto. En lugar de acender o LED directamente, cambiamos primeiro o valor desta variable interna e logo usamos `digitalWrite` para reflectilo fisicamente.
- **O rebote mecánico e o Retardo de Estabilización**: Cando as patillas metálicas dun pulsador físico chocan, rebotan e xeran falsas conexións ultrarrápidas (ruído eléctrico). A liña `delay` detén brevemente ao microcontrolador para deixar pasar estes rebotes iniciais e asegurar unha lectura estable.

O Reto da Práctica 4

Demostra que dominas a liña lóxica biestable e o manexo de flancos coas seguintes propostas:

1. **Lóxica Simplificada (Operador NOT)**: En programación existe un operador lóxico que invirte calquera valor booleano: o signo de exclamación ( ). Modifica o código base substituíndo as 6 liñas do bloque `if else` que decide o cambio de `estadoLed` pola seguinte expresión dunha soa liña:

   ```
   estadoLed = !estadoLed;
   ```

   Verifica en TinkerCad que o programa segue funcionando exactamente igual de ben e analiza por que ocorre isto.

2. **Selector de Intensidades (Semáforo de Potencia)**: Modifica o código para que un único botón sirva para alternar de xeito secuencial entre tres estados distintos con cada pulsación en lugar de dous:
   - Primeira pulsación: Acéndese un LED Vermello (Pin 8).
   - Segunda pulsación: Apágase o LED Vermello e acéndese un LED Verde (Pin 9).
   - Terceira pulsación: Apáganse ambos LEDs (reiniciando o ciclo).

3. *Pregunta para reflexionar*: Se quitamos a liña `lecturaAnterior = lecturaActual;`, que cres que lle pasará ao noso programa ao premer o botón? Razona a túa resposta.