Controlo programado com Arduino

Prática 4: O Botão de Pressão Retentivo (Funcionamento Biestável)

Na prática anterior aprendemos a controlar um LED de maneira momentânea (o LED apaga-se imediatamente ao soltar o botão). Hoje daremos solução a um problema do mundo real: como criamos um interruptor como o do nosso quarto, que com um único clique acende a luz e com outro clique a apaga? Para o conseguir, ensinaremos o Arduino a "lembrar-se" de estados utilizando variáveis e a detetar com precisão matemática o instante exato em que o botão é pressionado.

Objetivos de Aprendizagem

Ao finalizar esta sessão, serás capaz de:

- Compreender a diferença operacional entre sistemas monestáveis (momentâneos) e biestáveis (com retenção / enclavamento).
- Implementar variáveis de estado como memória interna do microcontrolador.
- Desenhar um algoritmo para detetar o **flanco de subida** (a mudança de LOW a HIGH de um pino de entrada).
- Entender o conceito físico dos repiques mecânicos (*bouncing*) em botões de pressão e como mitigá-los por software.

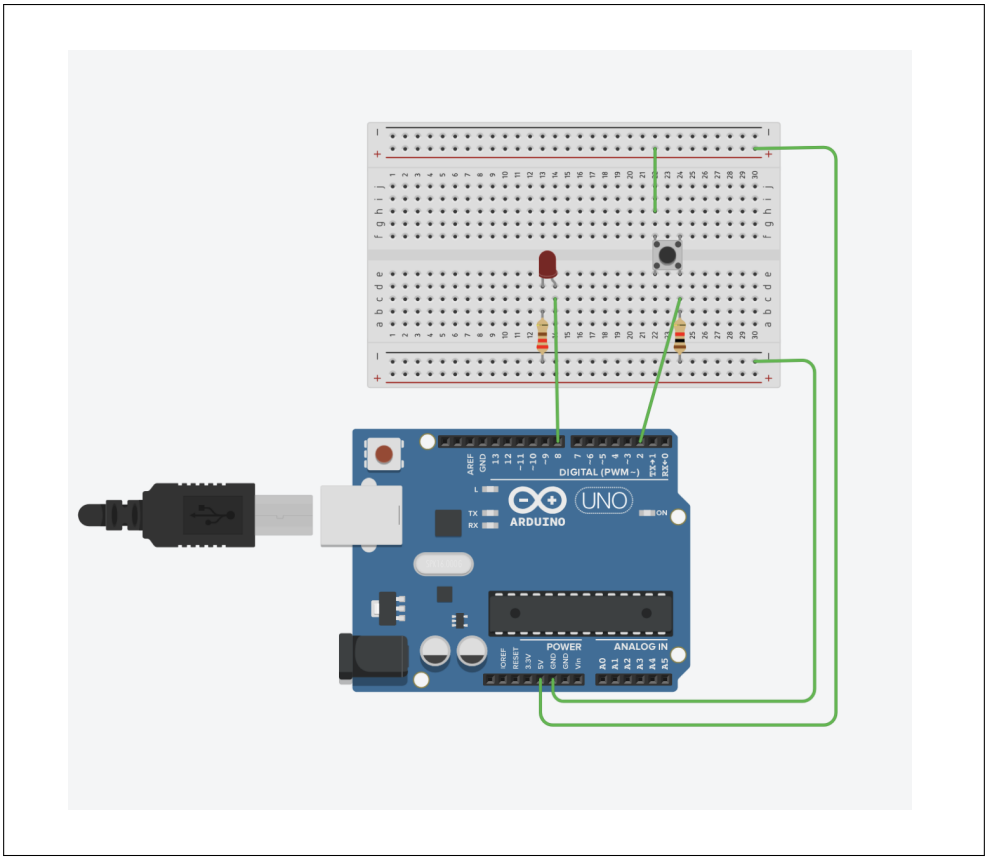
Componentes Necessários

Não mexas no teu circuito anterior! Esta prática utiliza exatamente as mesmas ligações que la Prática 3. Centraremos todo o nosso esforço e tempo em dominar o código de programação.

- 1 Placa Arduino Uno.
- 1 Placa de ensaio pequena (Protoboard).
- 1 Díodo LED e 1 resistência de $220\,\Omega$.
- 1 Botão de pressão de quatro pinos e 1 resistência de $10\,\text{k}\Omega$ (para o circuito Pull-Down).
- Cabos de ligação.

Esquema de Montagem na Protoboard

Verifica se a tua montagem no TinkerCad mantém as ligações que realizámos na prática anterior para garantir que o programa responda corretamente.



Código Base: Deteção do Flanco de Subida

Escreve com muita atenção o seguinte programa no teu editor do TinkerCad. Repara detalhadamente em como combinamos a leitura atual do botão de pressão com a leitura que foi obtida na volta anterior do programa.

```
// Declaracao de pinos fisicos
int led = 8;
int pulsador = 2;

// Variables de controlo de estado
int estadoLed = LOW;              // Guarda o estado atual do LED (LOW ou HIGH)
int lecturaActual;                // Armazena a leitura instantanea do botao
int lecturaAnterior = LOW;        // Armazena o valor da leitura da volta
    anterior

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

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

  // Se o botao estiver pressionado (HIGH) E na volta anterior estava solto (LOW
    )
  if (lecturaActual == HIGH && lecturaAnterior == LOW) {

    // Alternamos de estado (se estava LOW passa a HIGH, se estava HIGH passa a
    LOW)
    if (estadoLed == HIGH) {
      estadoLed = LOW;
    } else {
      estadoLed = HIGH;
    }

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

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

  // Guardamos a leitura atual para usá-la como anterior na seguinte iteracao
  lecturaAnterior = lecturaActual;
}
```

Listing 1: Código para alternar o estado de um LED com cada pulsação.

Como funciona o código?

O ciclo `loop` do Arduino é executado milhares de vezes por segundo. Se apenas avaliássemos se o botão está em `HIGH`, o LED mudaria de aceso para apagado milhares de vezes na décima

de segundo que o nosso dedo demora a soltar o botão. Isto faria com que o estado final ao levantar o dedo fosse totalmente imprevisível.

- **O operador lógico AND ( )**: Este operador exige que se cumpram obrigatoriamente as duas condições especificadas ao mesmo tempo. No nosso código: o botão deve estar pressionado *agora* (`HIGH`) e ter estado solto *um instante antes* (`LOW`). Isto define a transição ou "flanco de subida", que apenas ocorre uma única vez por cada pulsação.
- **A variável de estado ( )**: Atua como a memória do circuito. Em vez de acender o LED diretamente, mudamos primeiro o valor desta variável interna e depois usamos `digitalWrite` para o refletir fisicamente.
- **O repique mecânico e o Atraso de Estabilização**: Quando os contactos metálicos de um botão de pressão físico chocam, eles ressaltam e geram falsas ligações ultrarrápidas (ruído elétrico). A linha `delay` para brevemente o microcontrolador para deixar passar estes repiques iniciais e garantir uma leitura estável.

O Desafio da Prática 4

Demonstra que dominas a lógica biestável e a manipulação de flancos com as seguintes propostas:

1. **Lógica Simplificada (Operador NOT)**: En programação existe um operador lógico que inverte qualquer valor booleano: o ponto de exclamação ( ). Modifica o código base substituindo as 6 linhas do bloco `if else` que decide a mudança de `estadoLed` pela seguinte expressão de uma única linha:

   ```
   estadoLed = !estadoLed;
   ```

   Verifica no TinkerCad que o programa continua a funcionar exatamente igual e analisa por que motivo isto acontece.

2. **Seletor de Intensidades (Semáforo de Potência)**: Modifica o código para que um único botão sirva para alternar de maneira sequencial entre três estados diferentes com cada pulsação, em vez de dois:
   - Primeira pulsação: Acende-se um LED Vermelho (Pino 8).
   - Segunda pulsação: Apaga-se o LED Vermelho e acende-se um LED Verde (Pino 9).
   - Terceira pulsação: Apagam-se ambos os LEDs (reiniciando o ciclo).

3. *Pergunta para refletir*: Se retirarmos a linha `lecturaAnterior = lecturaActual;`, o que achas que acontecerá ao nosso programa quando pressionarmos o botão? Fundamenta a tua resposta.