Ministério da Educação
Universidade Tecnológica Federal do Paraná
Coordenação de Engenharia de Computação
Materiais e Equipamentos Elétricos
Campus Apucarana

Laboratório de Materiais e Equipamentos Elétricos

Roteiro 7: Arduino - Entrada Digital

1. Objetivos

- Analisar e compreender as portas (pinos) de um Arduino.
- Definir função de entrada ou saída ao pino desejado.
- Ler dados de um botão pelo Arduino.

2. Material utilizado

- 2 Resistores de 150 Ohms
- 2 LEDs
- Protoboard
- Arduino UNO
- Resistor de 10 kOhms
- Push-button

2.1. Entradas Digitais (Arduino)

Um microcontrolador, como o Arduino, possui portas de entrada digitais (que irão receber como entrada tensões 0 V ou 5 V). A Figura 1 mostra como as portas de um Arduino são distribuídas. As portas de entrada digital são as mesmas de saída digital. O arduino irá converter tensões próximas de **0 V** como nível baixo (**LOW**) e tensões próximas de **5 V** como nível alto (**HIGH**).

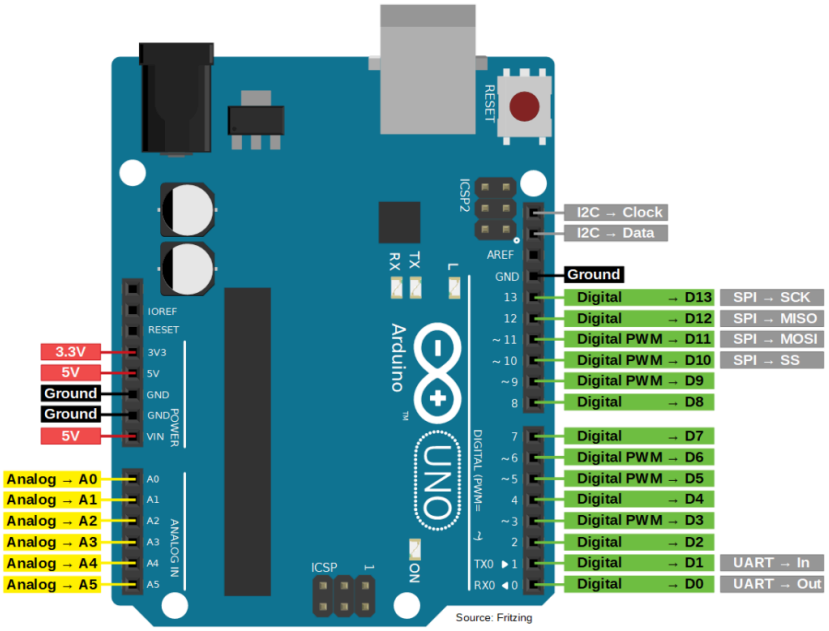


Figura 1: pinos disponíveis no Arduino UNO

2.2. Push-Button

Um push-button, também conhecido como botão de pressão ou chave tátil, é um componente eletrônico simples e amplamente utilizado em diversos dispositivos e circuitos. Um push-button é um interruptor momentâneo que é acionado quando pressionado e retorna ao estado original quando o dedo é retirado. Ele é composto por um botão e um conjunto de contatos elétricos internos. Quando pressionamos o botão, os contatos se conectam temporariamente, permitindo a passagem da corrente elétrica.

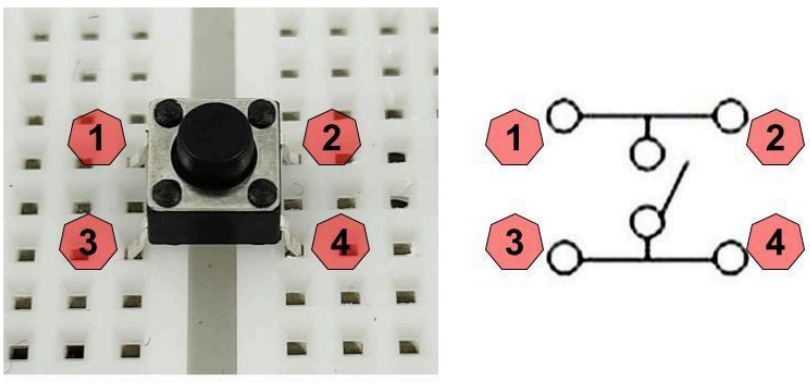


Figura 2: Ligação interna do push-button. Os pinos 1 e 2 são conectados aos pinos 3 e 4 quando o botão é pressionado.

Em sistemas microcontrolados, os push-buttons são frequentemente utilizados como interfaces de entrada, permitindo aos usuários interagirem com o sistema através de comandos manuais. Com a plataforma Arduino, podemos utilizar o push-button para acionar diferentes funcionalidades e controlar o comportamento do sistema. Por exemplo, imagine um projeto em que você queira desenvolver um semáforo simples controlado por um microcontrolador Arduino. Nesse caso, você poderia utilizar um push-button para simular o botão de pedestres, permitindo que os usuários solicitem a mudança do sinal para o pedestre atravessar a rua.

Existem duas formas de ligar um botão no microcontrolador: **pull-up** e **pull-down**. A diferença entre os dois está relacionada à forma como os resistores são conectados ao push-button e ao microcontrolador. A Figura mostra a diferença das ligações nas duas configurações.

- **Resistores de Pull-Up:**

Ao utilizar resistores de pull-up, o resistor é conectado entre o pino do microcontrolador e a alimentação positiva (VCC) do sistema. Nesse arranjo, o valor do resistor puxa o estado do pino para o nível lógico alto (VCC) quando o push-button não está pressionado. Quando o botão é pressionado, o contato é fechado e o pino é conectado ao pino terra (GND), o que resulta em um estado lógico baixo (0V).

Na prática, quando o push-button não está pressionado, o pino do microcontrolador é lido como HIGH (1), indicando que o botão não foi acionado. Quando o botão é pressionado, o pino é conectado ao pino terra através do contato fechado, e o microcontrolador lê o estado como LOW (0), indicando que o botão foi acionado.

- **Resistores de Pull-Down:**

No caso dos resistores de pull-down, o resistor é conectado entre o pino do microcontrolador e o pino terra (GND) do sistema. Nessa configuração, o resistor puxa o estado do pino para o nível lógico baixo (0V) quando o push-button não está pressionado. Quando o botão é pressionado, o contato é fechado e o pino é conectado à alimentação positiva (VCC), resultando em um estado lógico alto (VCC).

Assim como na configuração pull-up, quando o push-button não está pressionado, o pino do microcontrolador é lido como LOW (0), indicando que o botão não foi acionado. Quando o botão é pressionado, o pino é conectado ao VCC através do contato fechado, e o microcontrolador lê o estado como HIGH (1), indicando que o botão foi acionado.

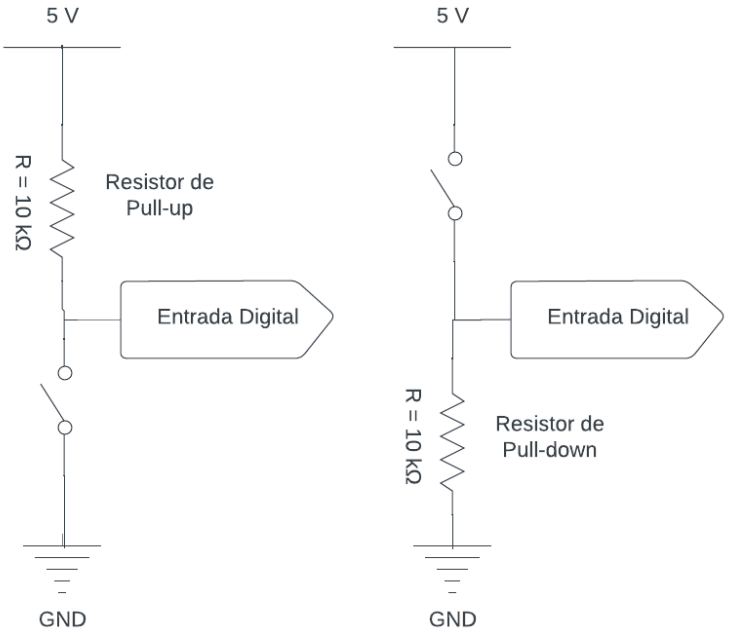


Figura 3: Configurações de ligação do push-button em Pull-up e Pull-down.

Dica 1: em geral os resistores de pull-up e pull-down são da ordem de 10 kOhms.
Dica 2: além dos modos de pino (pinMode) de entrada e saída (INPUT e OUTPUT), existe um modo **INPUT_PULLUP**, que usa um resistor interno do microcontrolador para pull-up, não precisando do resistor externo. Apesar desse modo ser conveniente, não utilizaremos ele por razões didáticas.

2.3. Exemplo de Configuração da porta de Entrada

Considere o caso em que você deseje ligar e desligar um LED que esteja conectado na **porta 13** utilizando um botão em pull-down conectado na **porta 2**, como mostrado na Figura 4 e Figura 5. No projeto queremos que o LED fique aceso enquanto o botão estiver pressionado e apagado quando o botão não estiver pressionado.

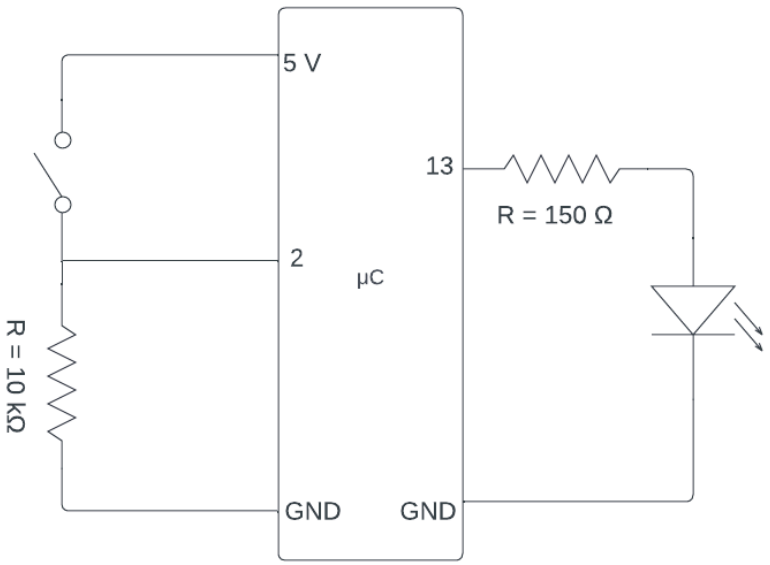


Figura 4: Montagem de um LED em série com resistor na porta 13 do Arduino.

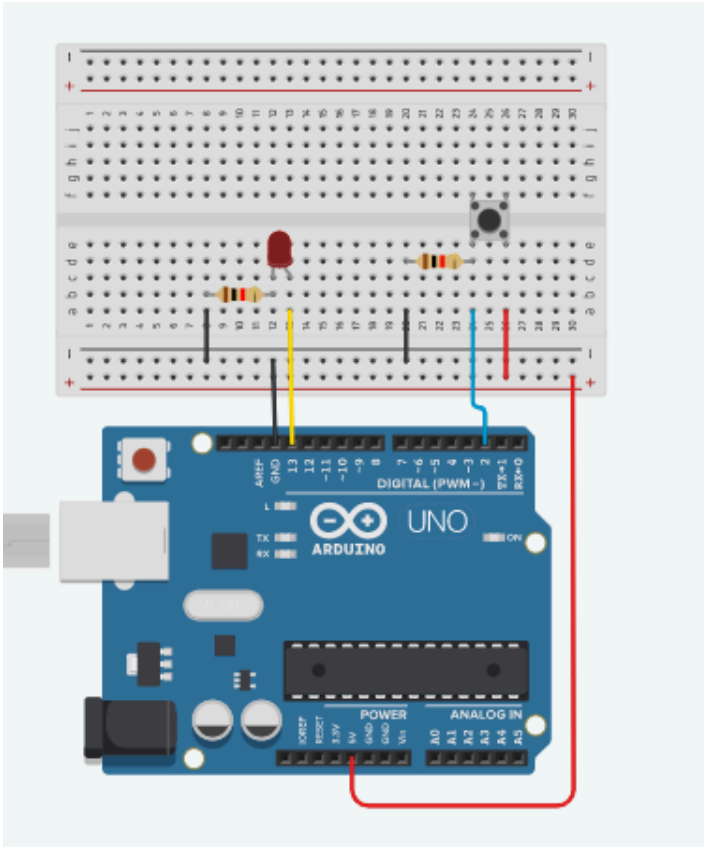


Figura 5: Montagem do circuito em protoboard.

Primeiramente, precisamos configurar que a porta 13 será de **SAÍDA** e a porta 2 será de **ENTRADA.** Antes, podemos definir os pinos por meio de uma variável:

```
const int LED_PIN = 13;
const int BUTTON_PIN = 2;
```

Alternativamente, podemos usar o comando *#define*.

```
#define LED_PIN 13
#define BUTTON_PIN 2
```

Configuramos os pinos dentro do loop setup pela função *pinMode*, onde ajustamos que o pino 13 será de saída (OUTPUT) e pino 2 como entrada (INPUT). Além disso iremos declarar uma **variável de estado** para armazenar o estado do botão.

```
// Variável de estado
int buttonState = 0;

void setup() {
        pinMode(LED_PIN, OUTPUT);
        pinMode(BUTTON_PIN, INPUT);
}
```

No laço principal loop(), iremos ler o estado do botão na variável buttonState. Caso o botão tenha sido pressionado, vamos ligar o LED. Caso contrário, vamos desligar. Um delay de 10 milissegundos é introduzido no final do loop para melhorar a performance, fazendo com que a checagem ocorra 100 vezes por segundo (caso contrário, o loop irá rodar muito mais rápido).

```
void loop() {
        // Lê o estado do push-button
         buttonState = digitalRead(BUTTON_PIN);

        // Verifica se o push-button foi pressionado
         if (buttonState == HIGH) {
                digitalWrite(LED_PIN, HIGH);   // Liga o LED
         }
         else {
                digitalWrite(LED_PIN, LOW);    // Desliga o LED
         }
        delay(10);
}
```

3. Parte prática

Considerando o circuito da Figura 6 com 2 LEDs nas portas 12 e 13 do Arduino e um botão na porta 2, realize os códigos com requisitos abaixo:

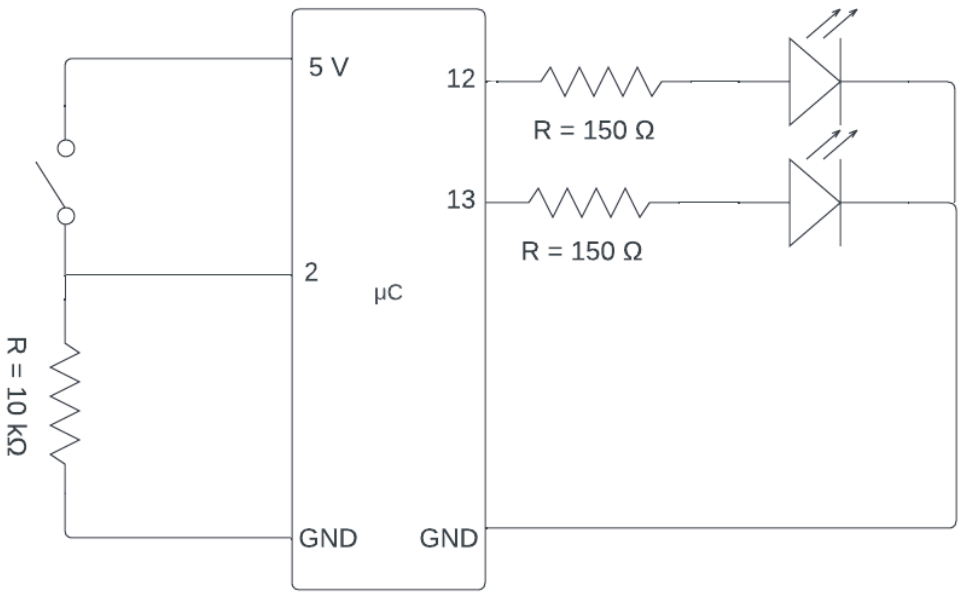
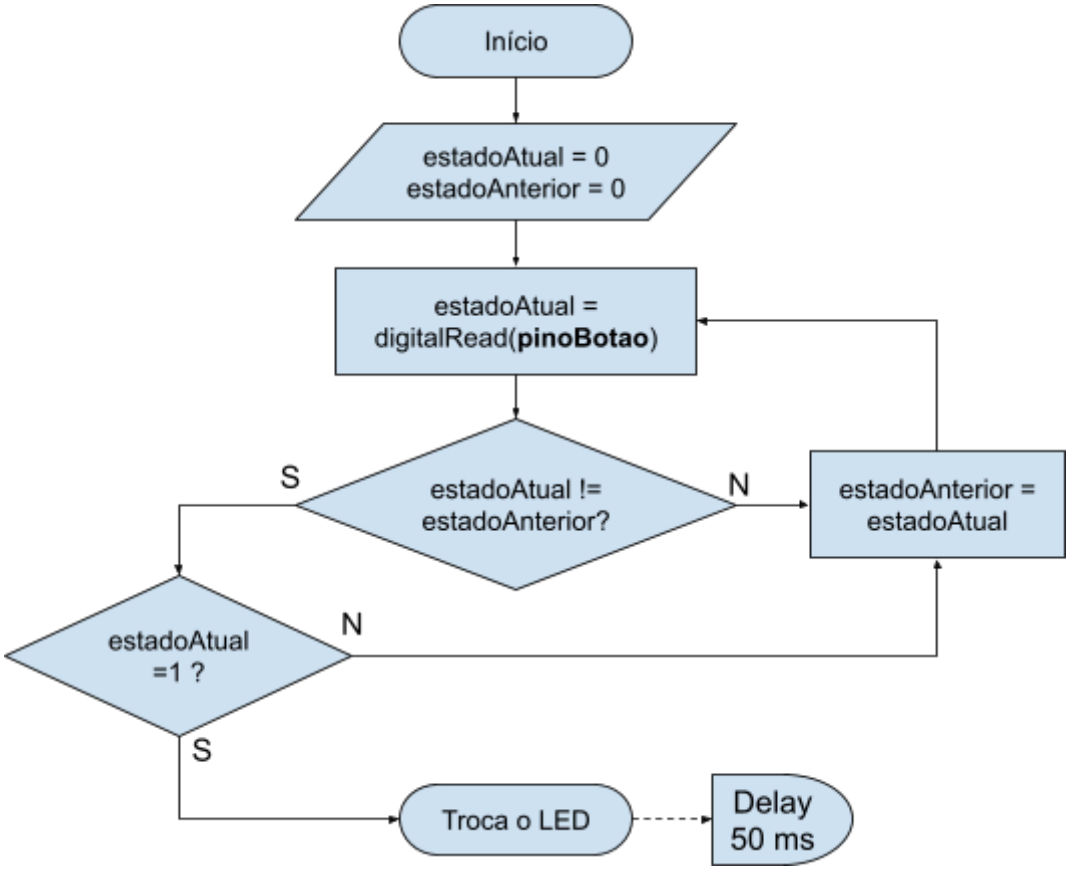


Figura 6: Circuito para a montagem prática

A. Elabore um código para fazer com que o LED vermelho e o verde troquem de estado **enquanto** botão estiver pressionado (quando o botão estiver apertado, o vermelho estará ligado e o verde estará desligado, quando o botão é liberado, os estados se alternarão, ficando vermelho desligado e verde ligado).
B. **Estourando bolhas**: use a função `random(min, max)` que permite gerar um número aleatório dentro do intervalo entre min e max para gerar um valor aleatório. Esse valor será usado para acender o LED verde após um tempo aleatório entre 100 ms e 3 segundos, sendo que o LED sempre inicia apagado. O LED ficará aceso até que o botão seja pressionado. Quando o botão for pressionado o LED verde apaga. Enquanto o botão estiver pressionado, o LED vermelho ficará aceso. Assim que o botão for liberado (solto), o código reinicia.
C. **Desafio**: Crie um código em que os LEDs alternam entre si em um período de 1 segundo. A cada vez que o botão é pressionado, o tempo de alternância reduz em 100 ms. Ao zerar o tempo, o período de alternância retorna para 1 segundo. Dica: o tempo deve ser reduzido sempre que o botão é pressionado, porém não deve reduzir continuamente caso o botão fique pressionado. Ou seja, para reduzir duas vezes, o botão deve ser pressionado e liberado duas vezes.
D. **Desafio**: Crie um código fará os LEDs aternarem toda vez que o botão for pressionado. Exemplo: o LED aceso é o vermelho, você pressionou e liberou o botão, o LED vermelho apaga e o verde acende e permanece nesse estado até que o botão seja pressionado novamente. Além disso, faça uma dupla comparação (conforme fluxograma abaixo). Caso o botão tenha mudado de estado e o novo estado seja pressionado (neste caso =1) ele irá trocar o LED. Um delay de 50 ms pode ser adicionado para evitar o efeito de “bounce” (acionamento falso causado por oscilações eletromecânicas no botão).


Início
estadoAtual = 0
estadoAnterior = 0
estadoAtual = digitalRead(**pinoBotao**)
estadoAtual != estadoAnterior?
S
N
estadoAnterior = estadoAtual
estadoAtual =1 ?
N
S
Troca o LED
Delay 50 ms

E.