

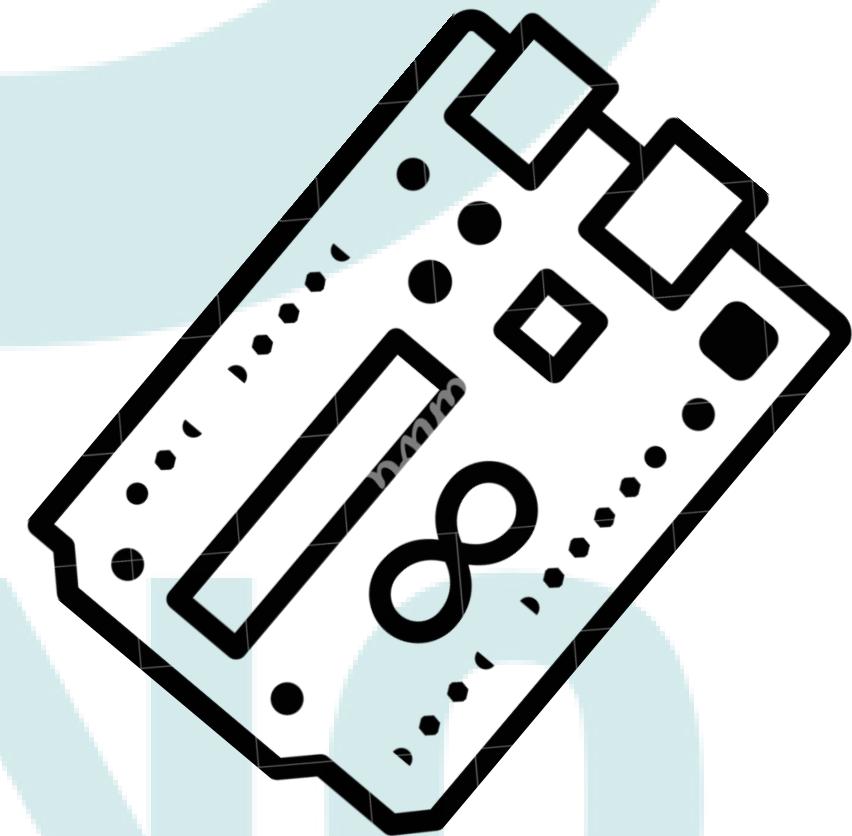
Arduino: Conceitos, Programação e Aplicação Prática



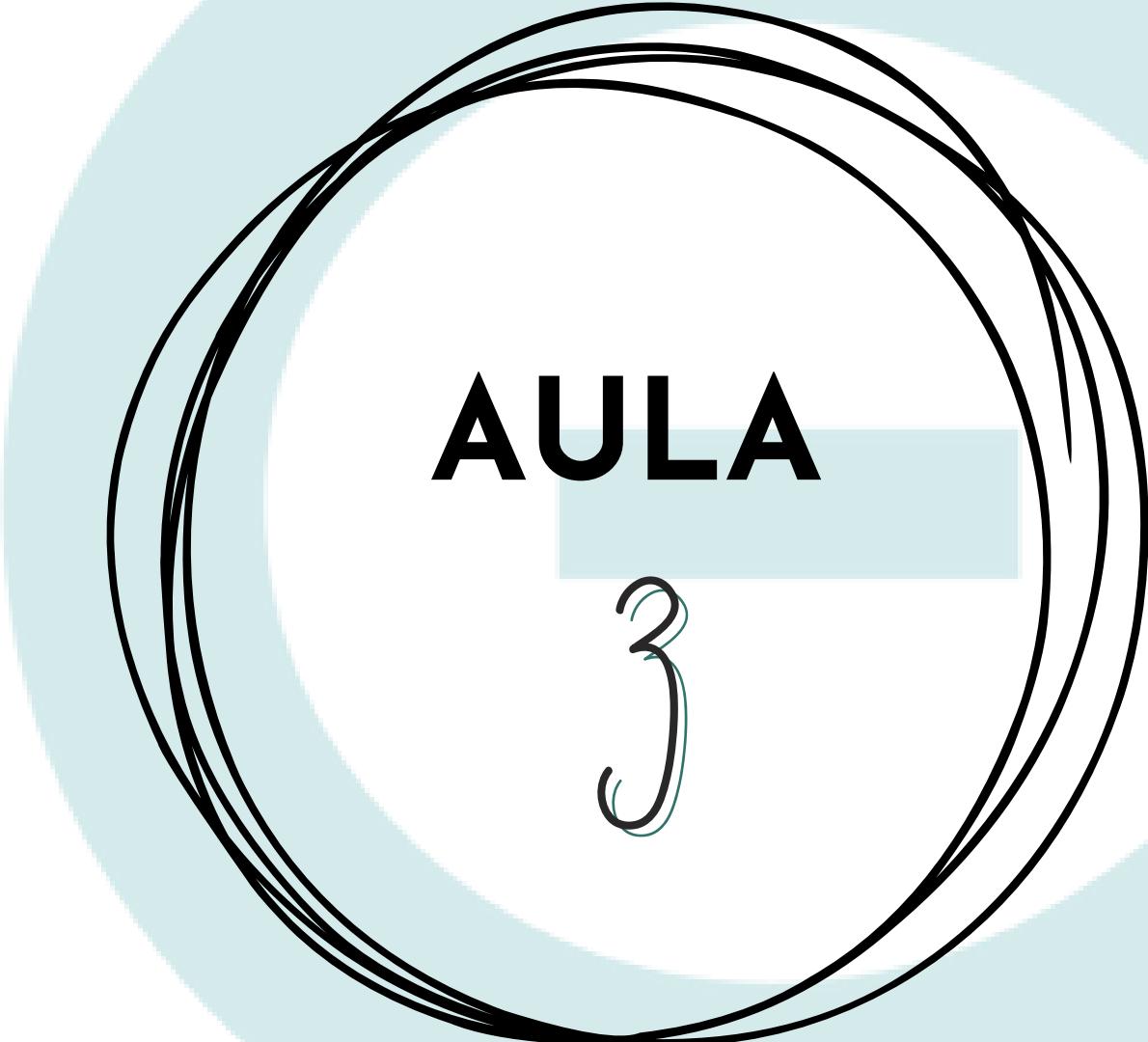
ARDUINO

UTP
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CÂMPUS PATO BRANCO



®



AULA

3



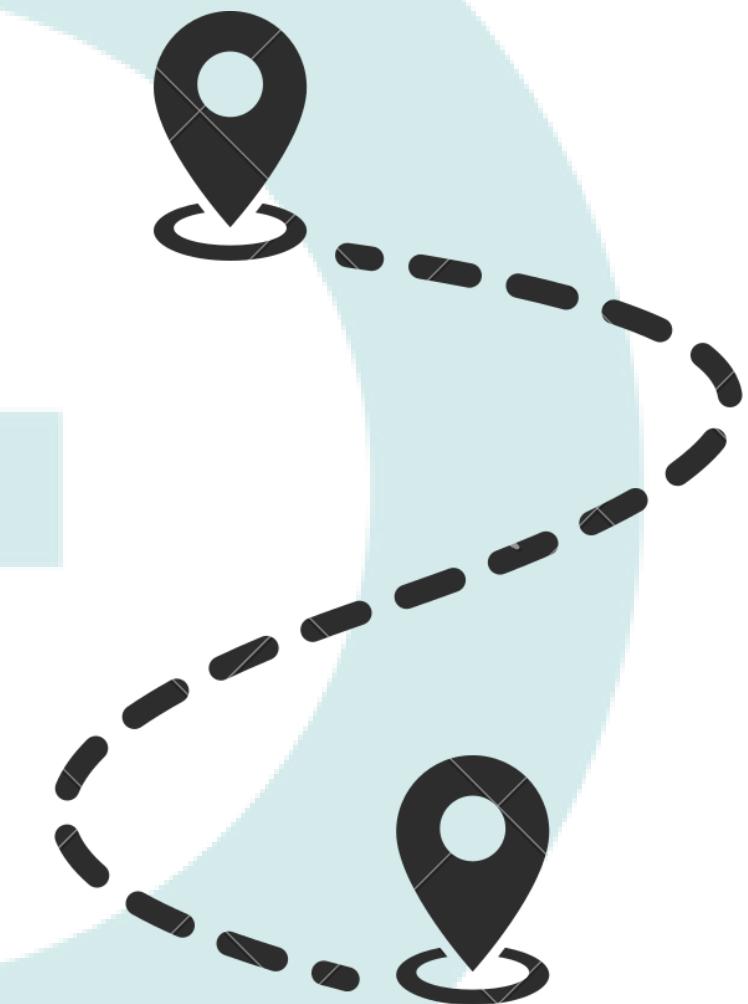
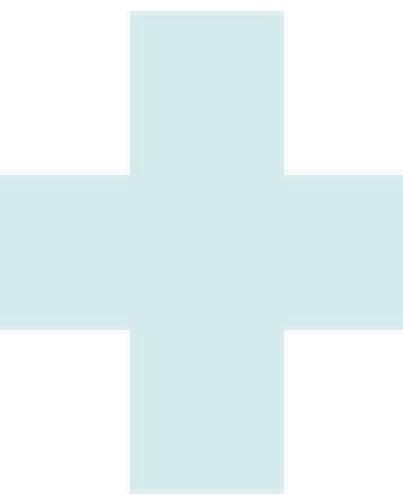
Operações
Aritméticas,
Estrutura de Fluxo e
Botões

Coordenador
Fábio Favarim

Instrutor
Gabriel S. Folly

Roteiro

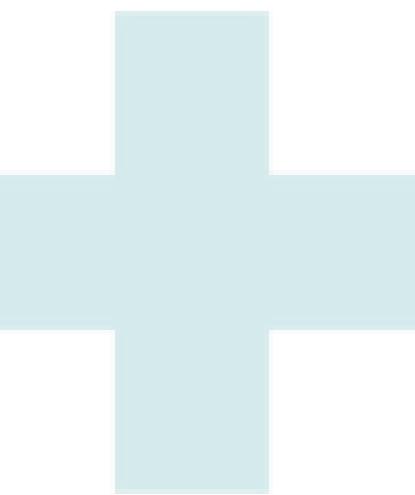
1. Operadores
2. Operações Aritméticas
3. Estruturas de Controle de Decisão
4. Estruturas de Repetição
5. Laboratório
6. Push-Button
7. Modos de Ligar um Push-Button
 - Pull-Up
 - Pull-Down
 - Função INPUT_PULLUP
8. Laboratórios



ARDUTENO

Operadores Aritméticos

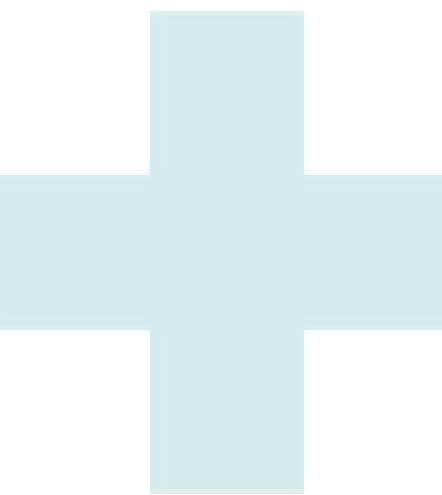
- **'+'** (adição):
 - **Exemplo:** int resultado = a + b;
- **'-'** (subtração):
 - **Exemplo:** int resultado = a - b;
- **'*' (multiplicação):**
 - **Exemplo:** int resultado = a * b;
- **'/' (divisão):**
 - **Exemplo:** int resultado = a / b;
- **'%' (módulo):**
 - **Exemplo:** int resultado = a % b;



ARDUTENO

Operadores Atribuição e Comparação

- '=' (atribuição):
 - **Exemplo:** int a = 5;
- '==' (igual a):
 - **Exemplo:** if (a == b) { /* código */ }
- '!=’ (diferente de):
 - **Exemplo:** if (a != b) { /* código */ }
- '<', '>', '<=', '>=':
 - **Exemplo:** if (a > b) { /* código */ }



ARDUINO

Operadores Lógicos e Incremento/Decremento

- '**&&**' (E lógico):
 - **Exemplo:** if (condicao1 && condicao2) { /* código */ }
- '**||**' (OU lógico):
 - **Exemplo:** if (condicao1 || condicao2) { /* código */ }
- '**!**' (NÃO lógico):
 - **Exemplo:** if (!condicao) { /* código */ }
- '**++**' (incremento) e '**--**' (decremento):
 - **Exemplo:** a++; (incremento de 'a')

ARDUINO

Operações Aritméticas no Arduino

- Amplamente utilizadas em projetos Arduino para controlar sensores, atuadores e tomar decisões com base em cálculos numéricos.
- **Operadores Aritméticos:** Utilização de '+', '-', '*', '/', '%'.
- **Operador de Atribuição:** Uso de '=' para atribuir valores a variáveis.
 - **Exemplo:** int a = 5;
- **Considerações Importantes:** Tipo de dado (int, float, etc.) é crucial para evitar perda de precisão.
- **Aplicações Práticas:** Controle preciso de dispositivos, cálculos para tomada de decisões e adaptação a variáveis do ambiente.

```
1 int soma = 10 + 1; //a variavel soma armazenara o valor 11
2 int subtracao = 10 - 5; //a variavel subtracao armazenara o valor 5
3 int multiplicacao = 2 * 1; //a variavel multiplicacao armazenara o valor 2
4 float divisao = 8.0 / 2; //a variavel divisao armazenara o valor 4.0
5 int resto = 10 % 3; //a variavel resto armazenara o valor de 1
6
7
```

Estruturas de Controle de Decisão

São fundamentais na programação para execução condicional de blocos de código.

- **Estruturas Comuns no Arduino:**

- "if":

- Verificações condicionais simples ou encadeadas.
 - Executa bloco de código se a condição for verdadeira.

- "switch":

- Adequado para múltiplas opções.
 - Abordagem organizada e eficiente.

- **Flexibilidade e Adaptabilidade:** Proporcionam flexibilidade para lidar com diferentes cenários. Melhoram a eficácia dos programas desenvolvidos para plataformas Arduino.

```
1 int sorte = 94;
2
3 if(sorte == 94){ //verifico de o valor contido na variavel sorte eh igual a 94
4     Serial.println("Parabéns você foi o ganhador!"); //caso a resposta seja sim
5 } else {
6     Serial.println("Não foi desta vez!"); //caso a resposta seja nao
7 }
8
```

```
1 float temperatura = 25.5;
2
3 if(temperatura > 30) { //verificando se a temperatura eh maior que 30 graus
4     Serial.println("Esta muito quente!"); //se a resposta for sim
5 } else if(temperatura >= 21 && temperatura <= 30) { //verificando se a temperatura esta entre 21 e 30 graus
6     Serial.println("Temperatura agradavel"); //se a resposta for sim
7 } else {
8     Serial.println("Esta um pouco frio."); //caso nao se encaixe em nenhuma das anteriores a resposta so pode ser esta
9 }
10
```

Exemplo Verificando Maior Idade

```
1 int sorte = 94;  
2  
3 if(sorte == 94){ //verifico de o valor contido na variavel sorte eh igual a 94  
4     Serial.println("Parabéns você foi o ganhador!"); //caso a resposta seja sim  
5 } else {  
6     Serial.println("Não foi desta vez!"); //caso a resposta seja nao  
7 }  
8
```

```
1 float temperatura = 25.5;  
2  
3 if(temperatura > 30) { //verificando se a temperatura eh maior que 30 graus  
4     Serial.println("Esta muito quente!"); //se a resposta for sim  
5 } else if(temperatura >= 21 && temperatura <= 30) { //verificando se a temperatura esta entre 21 e 30 graus  
6     Serial.println("Temperatura agradavel"); //se a resposta for sim  
7 } else {  
8     Serial.println("Esta um pouco frio."); //caso nao se encaixe em nenhuma das anteriores a resposta so pode ser esta  
9 }  
10
```

Estruturas de Repetição no Arduino

São fundamentais para controlar o fluxo de execução em programas Arduino.

- Automatizam tarefas repetitivas, contribuindo para eficiência e versatilidade.
 - **Função 'loop()':** Executa infinitamente o bloco de código contido, fundamental no Arduino.

ARDUTINO

Estrutura 'for' no Arduino

Utilizada para loops com número conhecido de iterações.

- **Três partes:** inicialização, condição de continuação e expressão de iteração.
- **Exemplo:**

```
1  for(int i = 0; i < 10; i++)  
2  {  
3      // as instruções que estiverem aqui dentro serão executadas 10 vezes.  
4      // do 0 até o 9  
5  }  
6  for(int i = 1; i <= 10; i++)  
7  {  
8      // as instruções que estiverem aqui dentro serão executadas 10 vezes.  
9      // do 1 até o 10  
10 }
```

Estrutura ‘while’ no Arduino

Ideal para loops com condição verdadeira por período indefinido.

- Execução continua enquanto a condição definida permanecer verdadeira.
- Exemplo:

```
1 int contador = 0;  
2  
3 while (contador < 10)  
4 {  
5     // as instruções que estiverem aqui dentro serão executadas 10 vezes.  
6     // do 0 até o 9  
7     contador++; // contador deve ser incrementado a cada execução do bloco.  
8 }
```

```
1 int contador = 1;  
2  
3 while (contador <= 10)  
4 {  
5     // as instruções que estiverem aqui dentro serão executadas 10 vezes.  
6     // do 1 até o 10  
7     contador++; // contador deve ser incrementado a cada execução do bloco.  
8 }
```

A

O

Laboratório 01

Piscando um LED em números pares.

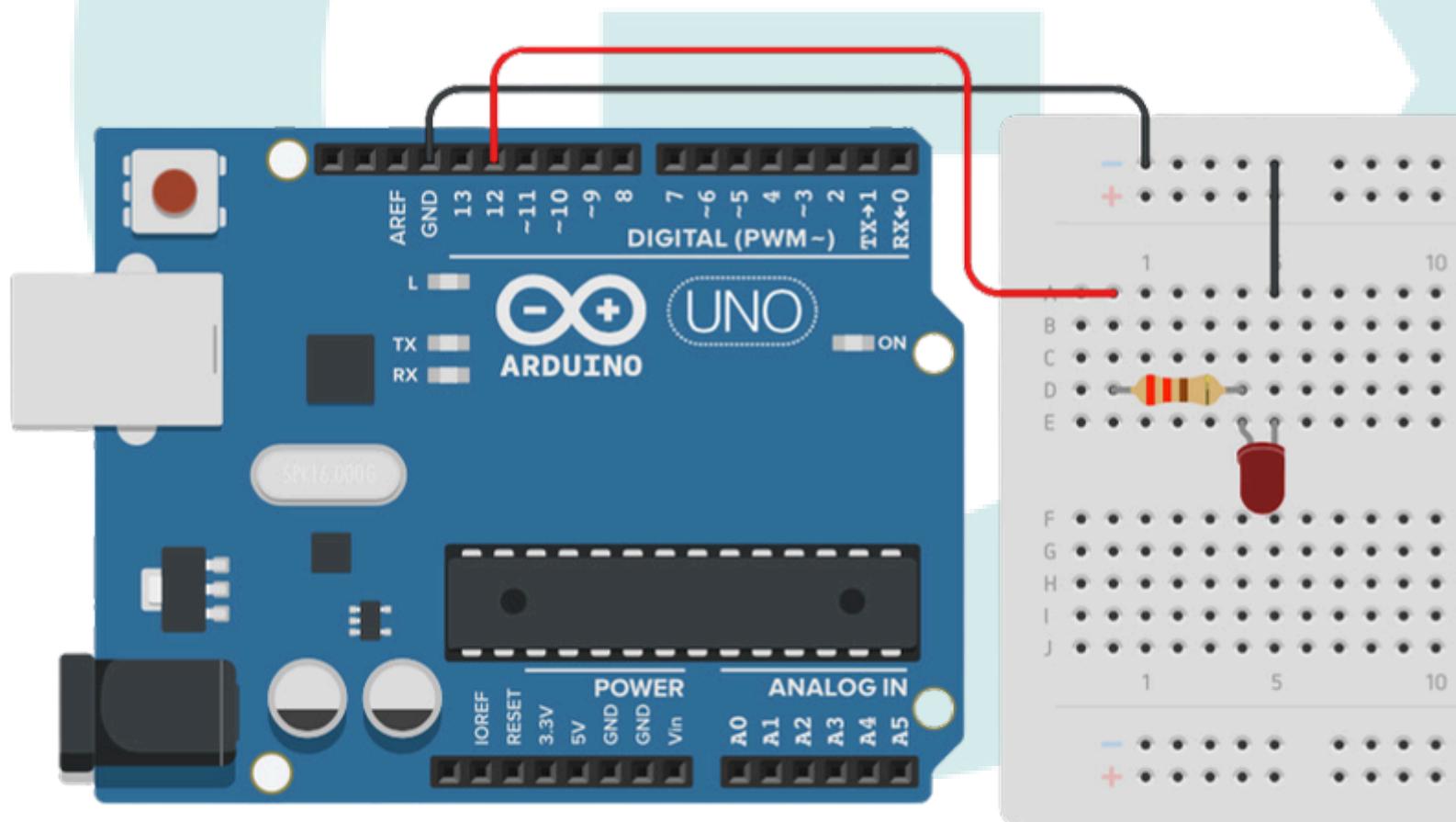
Neste guia, você aprenderá a montar um circuito usando um Arduino para que toda vez que o contador da estrutura de repetição tenha um numero par o LED pisque. O contador deve ser executado 100 vezes.

Materiais Necessários:

- Arduino Uno, Protoboard e Jumpers.
- 1 LED.
- 1 resistor de 220 ohms.

ARDUTINO

Solução Laboratório 01



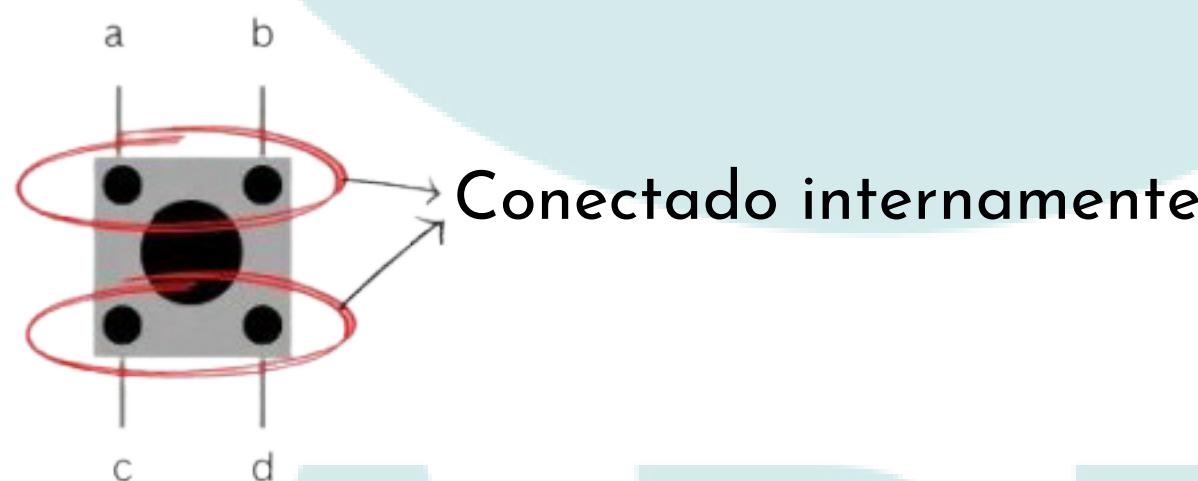
```
1 int LED_VERMELHO = 12;

2
3
4 void setup()
5 {
6     Serial.begin(9600);
7     pinMode(LED_VERMELHO, OUTPUT);
8 }
9
10 void loop()
11 {
12     for(int contador = 0; contador < 100; contador++) {
13         Serial.println(contador);
14
15         if(contador % 2 ==0) {
16             digitalWrite(LED_VERMELHO, HIGH);
17             Serial.println("Ligou o LED");
18             delay(1000);
19         } else{
20             digitalWrite(LED_VERMELHO, LOW);
21             Serial.println("Desligou o LED");
22             delay(1000);
23         }
24     }
25 }
```

ARDUINO

Push-Button no Arduino

- Dispositivo momentâneo que fecha ou abre um circuito quando pressionado.
- Amplamente utilizado para entrada de usuário e controle de eventos.
- **Conexão ao Arduino:**
- Pode ser conectado a um pino digital para detectar estados pressionado/não pressionado.



ARDUTINO

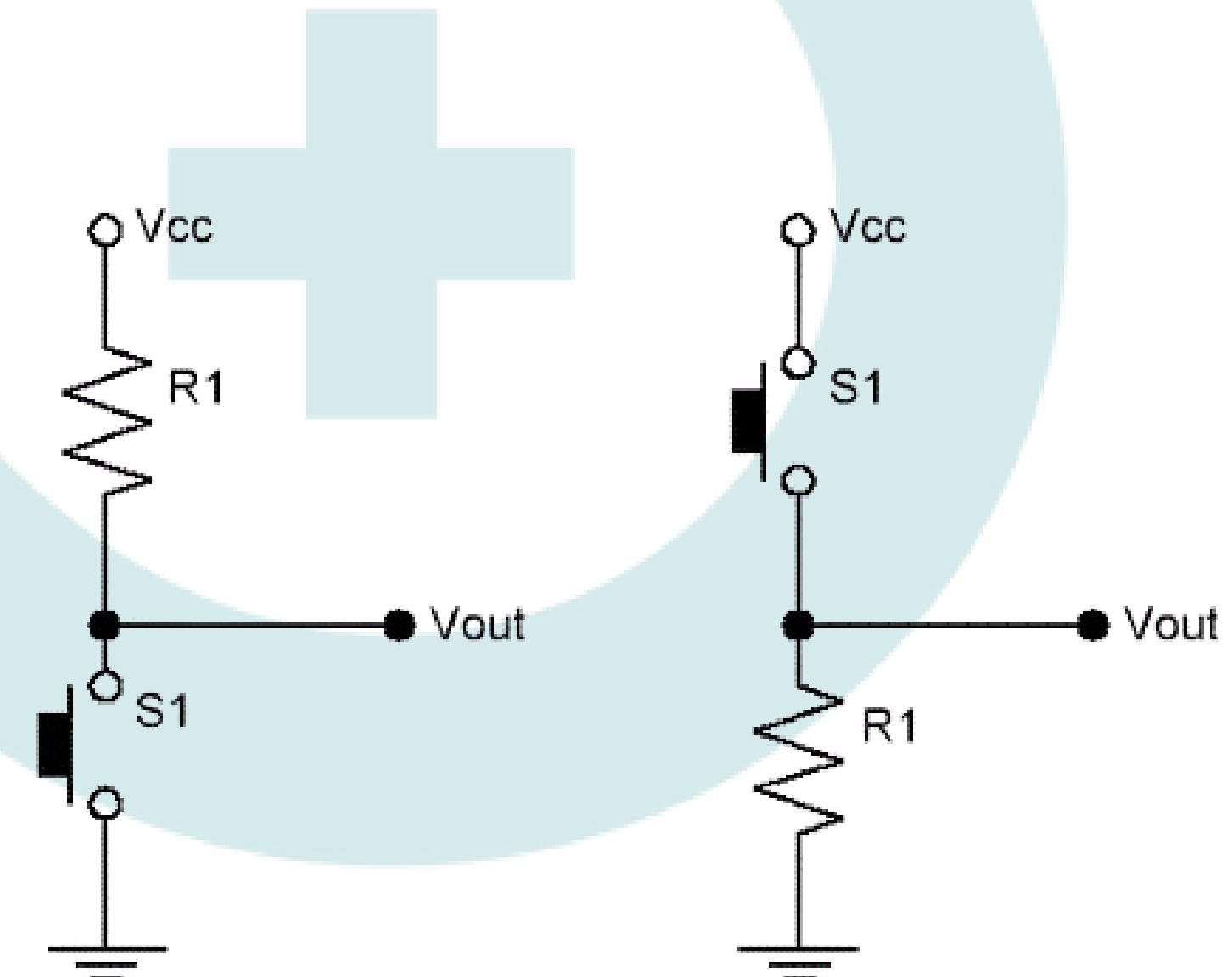
Configuração Pull-Up e Pull-Down

- **Pull-Up:**

- Pino conectado a uma resistência de pull-up para VCC.
- Pino em HIGH quando não pressionado, muda para LOW quando pressionado.
- Evita flutuações no estado do pino quando não pressionado.

- **Pull-Down:**

- Resistência conectada a GND.
- Pino em LOW quando não pressionado, muda para HIGH quando pressionado.
- Evita flutuações quando o botão não está em uso.



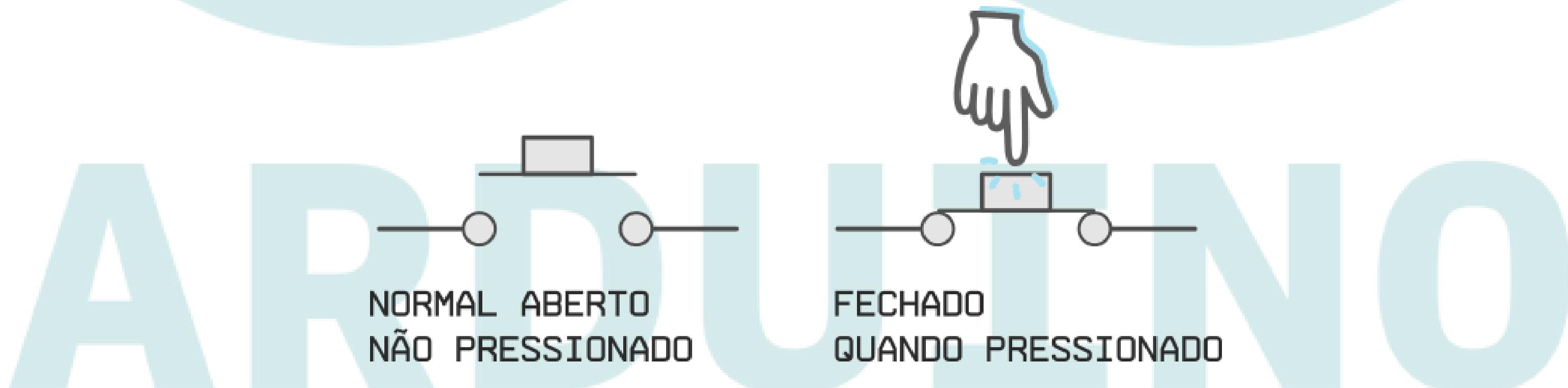
Importância da Configuração Adequada

- **Evitar Instabilidades:**

- Pull-up e pull-down evitam flutuações no estado do pino.
- Garantem leitura confiável do estado do botão no Arduino.

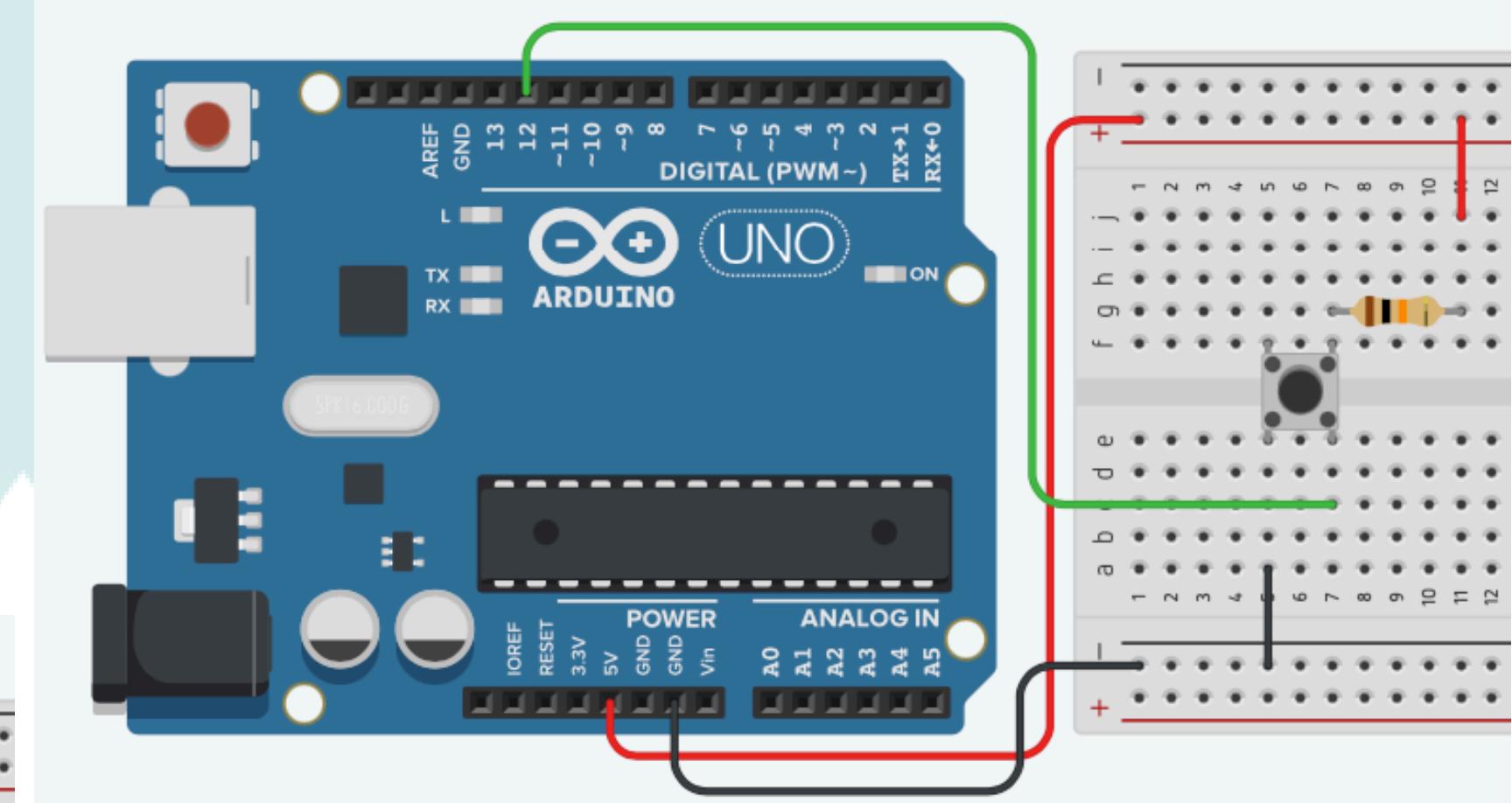
- **Leitura Precisa:**

- A configuração adequada é crucial para uma interação estável e confiável com o sistema.
- Essencial para projetos que dependem de entrada física do usuário.

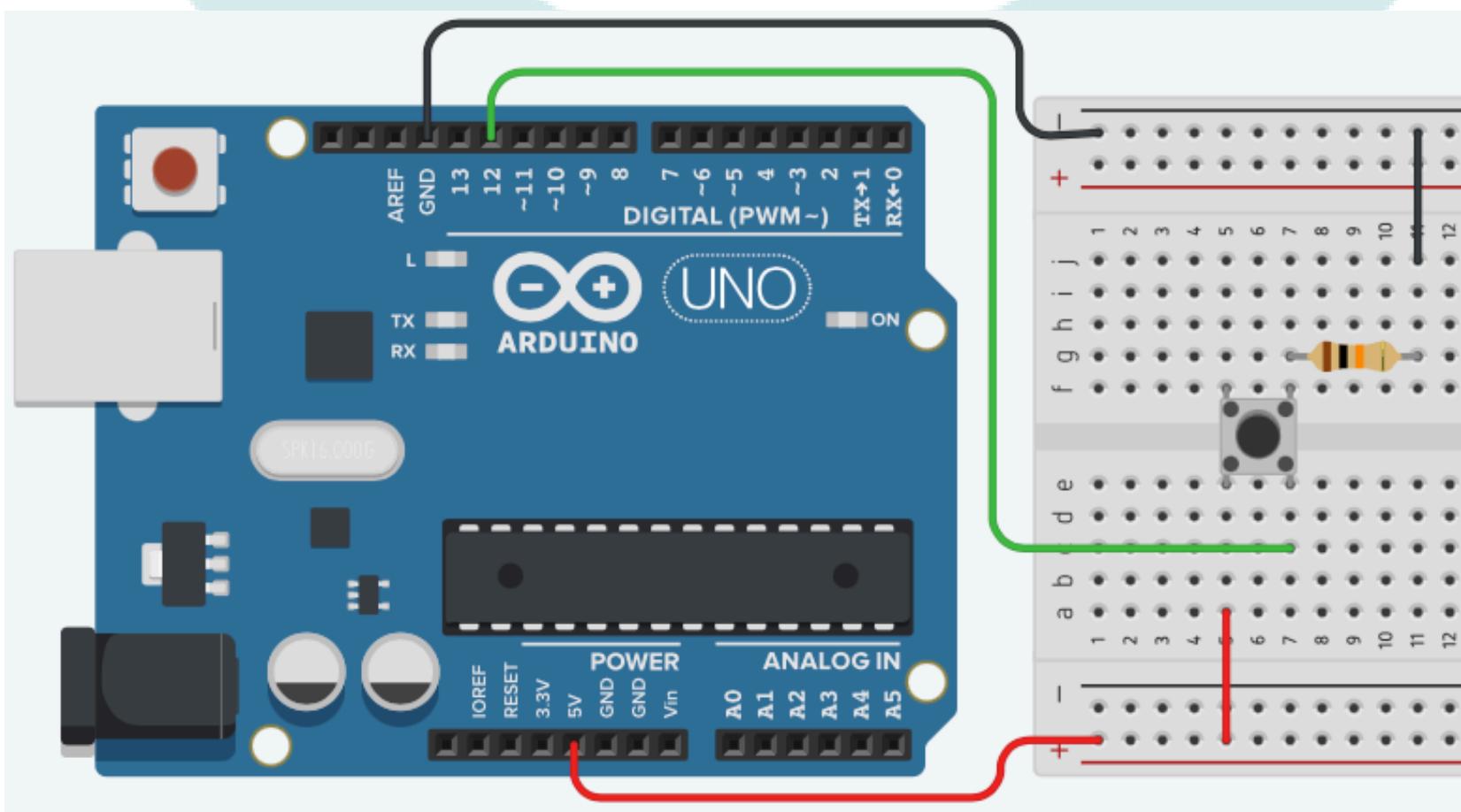


Modos de Ligar Push-Button

- Pull-Up



- Pull-Down

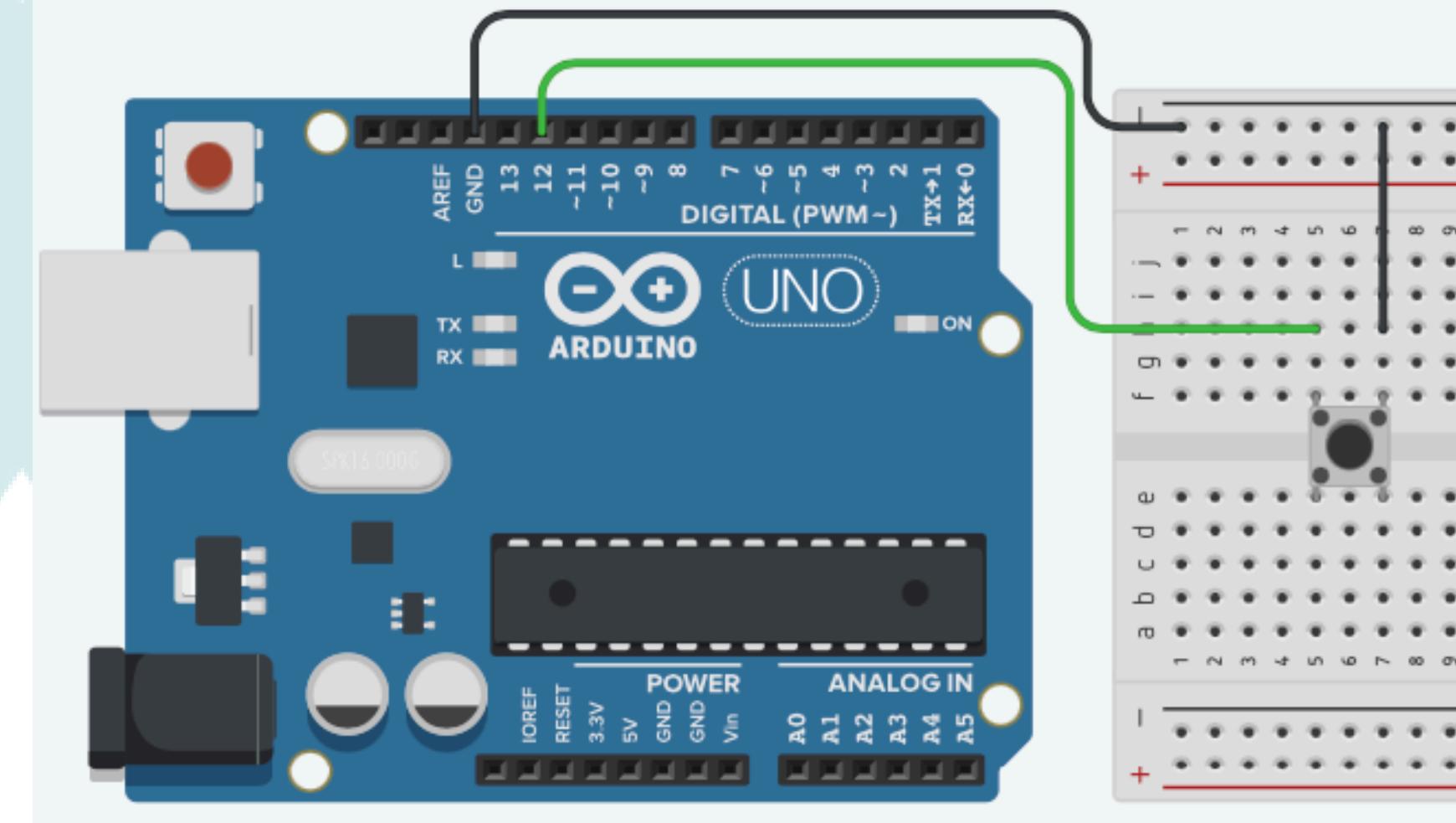


```

1 int botao = 12; // declaração do botão
2 void setup()
3 {
4     pinMode(botao, INPUT);
5     Serial.begin(9600); // inicialização da comunicação serial
6 }
7
8 void loop()
9 {
10    int estado = digitalRead(botao); // leitura e armazenamento na variável para armazenar o estado do botão
11    Serial.println(estado); // imprime no monitor serial o estado do botão
12    // Pull-up imprime 1 e quando pressionado imprime 0
13    // Pull-down imprime 0 e quando pressionado 1
14    delay(1);
15 }
```

Função INPUT_PULLUP

- **Entrada (INPUT):**
Configuração de pinos para receber informações do ambiente externo.
 - **Exemplos:** Sensores, botões, potenciômetros.
- **Leitura de Dados:** Capaz de ler informações, como estado de botões ou leitura de sensores.



```

1 int botao = 12; // declaração do botão
2 void setup()
3 {
4   pinMode(botao, INPUT_PULLUP); // utilizando a função INPUT_PULLUP estaremos utilizando o resistor interno do Arduino
5   Serial.begin(9600); // inicialização da comunicação serial
6 }
7
8 void loop()
9 {
10   int estado = digitalRead(botao); // leitura e armazenamento na variável para armazenar o estado do botão
11   Serial.println(estado); // imprime no monitor serial o estado do botão
12   // com a configuração PULLUP imprime 1 e quando pressionado 0
13   delay(1);
14 }
```

Laboratório 02

Acionamento de um LED por meio de um botão.

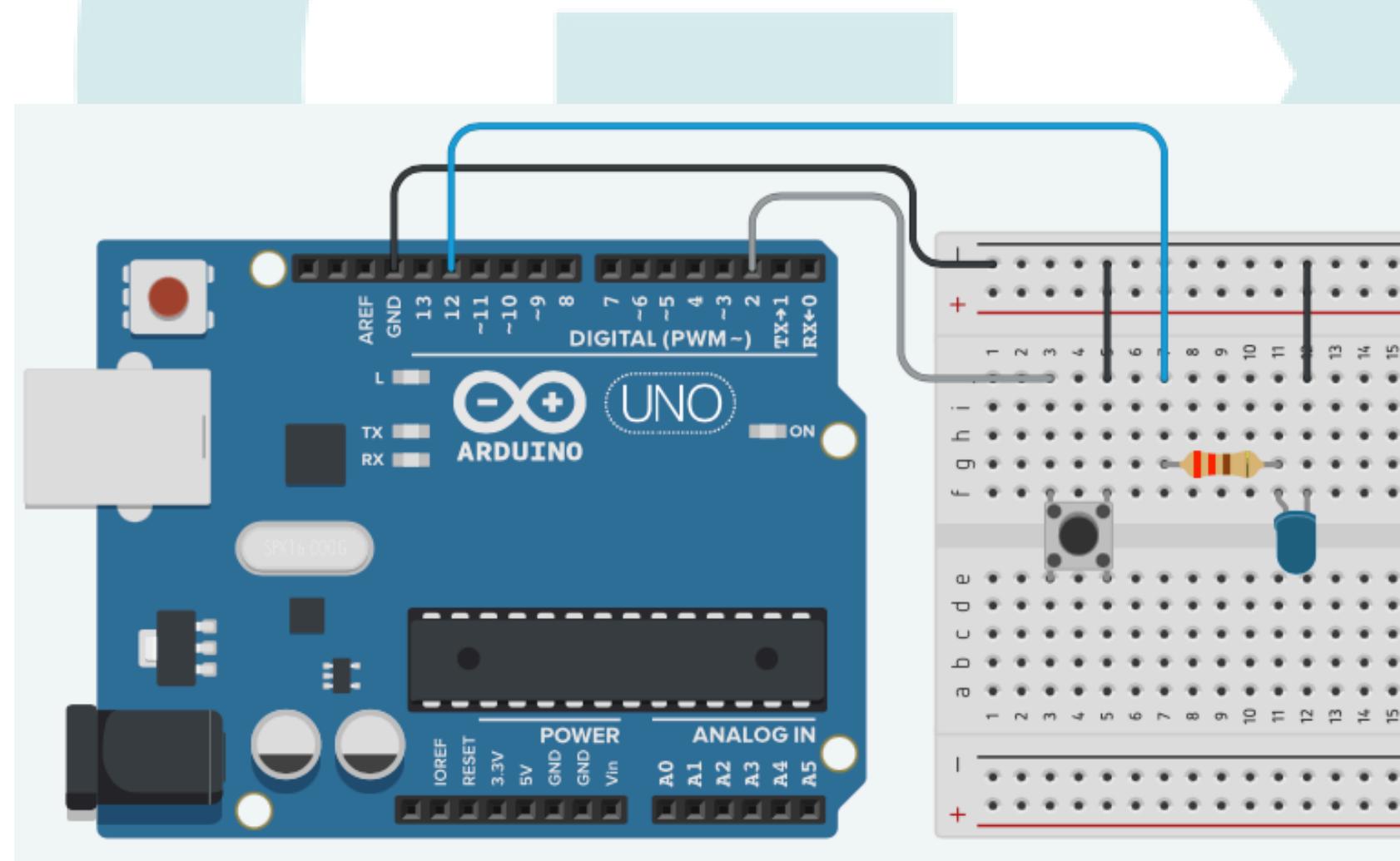
Neste guia, você aprenderá a montar um circuito usando um Arduino para fazer o acionamento de um LED via botão.

Materiais Necessários:

- Arduino Uno, Protoboard e Jumpers.
- 1 LED.
- Botão de pressão.
- Resistor de 220Ω (para o LED).

ARDUTENO

Solução Laboratório 02



```
1 const int button = 2; // Pino digital ao qual o botao esta conectado
2 const int LED = 12; // Pino digital ao qual o LED esta conectado
3
4 void setup() {
5     pinMode(button, INPUT_PULLUP); // Ativa a resistencia pull-up interna no botao
6     pinMode(LED, OUTPUT);
7 }
8
9 void loop() {
10    if (digitalRead(button) == LOW) { //Verifica se o botao esta pressionado
11        digitalWrite(LED, HIGH); //Liga o LED
12    } else {
13        digitalWrite(LED, LOW); //Desliga o LED
14    }
15 }
```

ARDUTINO

Laboratório 03

Semáforo de veículos em Arduino

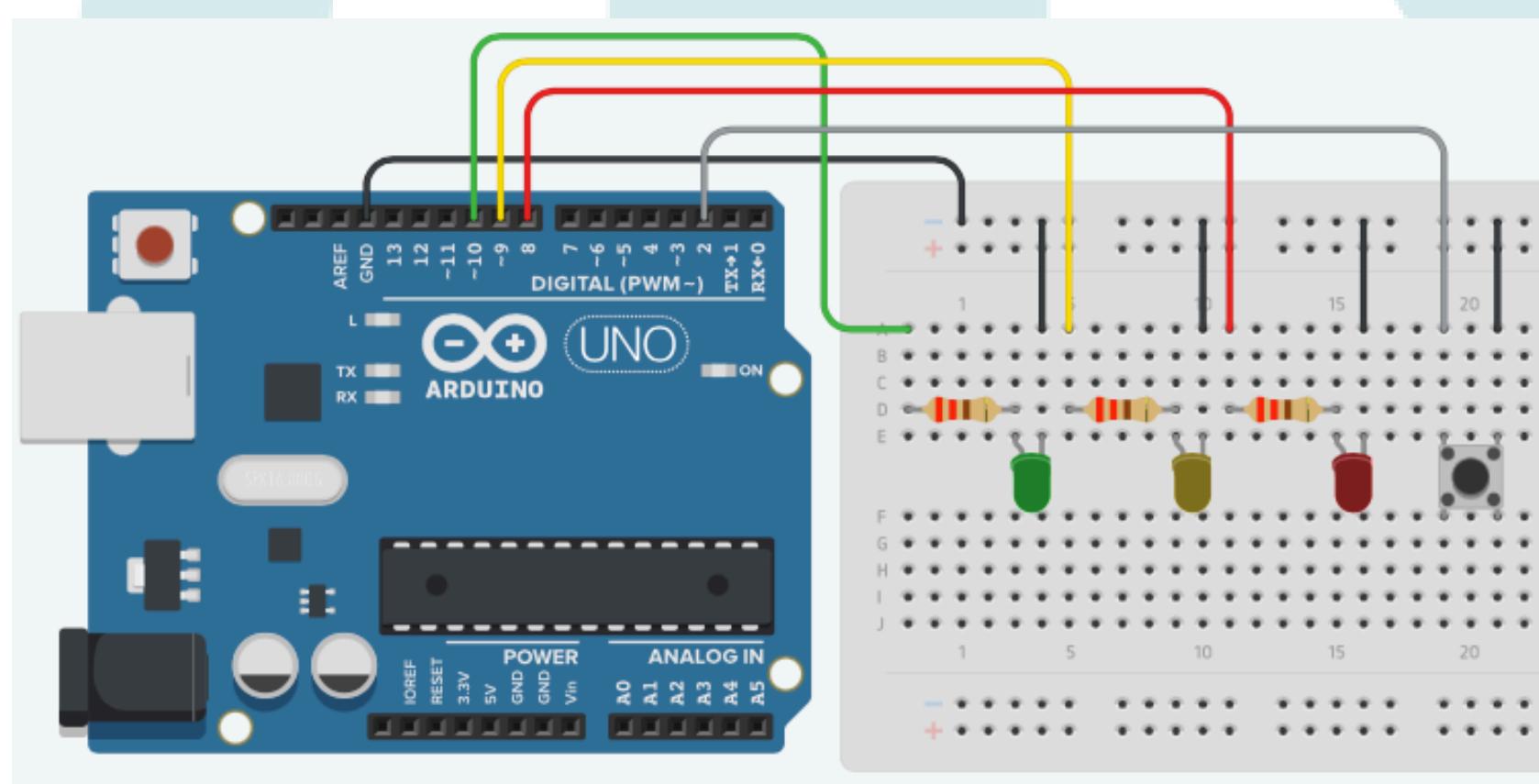
Neste guia, você aprenderá a montar um circuito utilizando 3 LEDs que simulam as fases de um semáforo (vermelho, amarelo e verde), acionado por um botão.

Materiais Necessários:

- Arduino Uno, Protoboard e Jumpers.
- 3 LEDs (vermelho, amarelo, verde).
- Botão de pressão.
- 3 Resistores (220Ω para LEDs).

ARDUTINO

Solução Laboratório 03



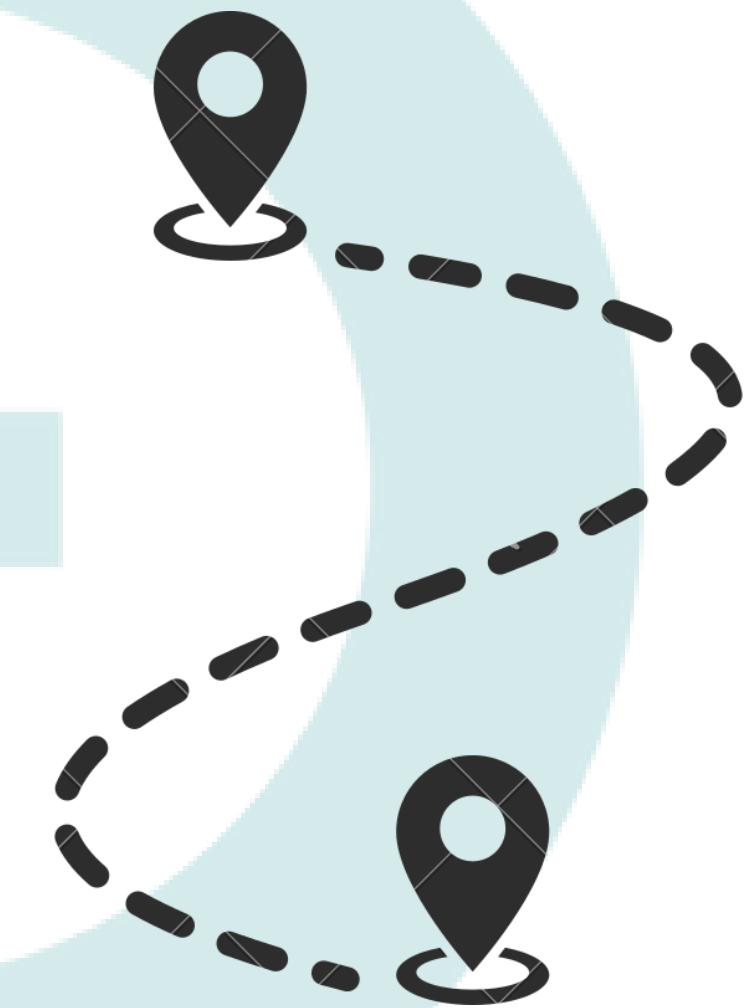
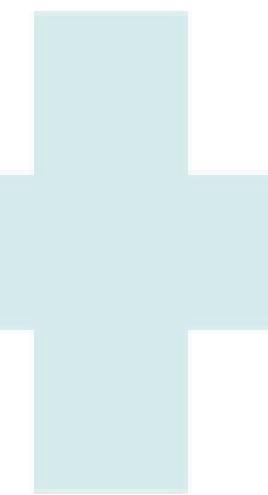
```

1 int botao = 2; // Pino digital ao qual o botao esta conectado
2 int LED_vermelho = 8; // Pino digital ao qual o LED vermelho esta conectado
3 int LED_amarelo = 9; // Pino digital ao qual o LED amarelo esta conectado
4 int LED_verde = 10; // Pino digital ao qual o LED verde esta conectado
5
6 void setup() {
7   pinMode(botao, INPUT_PULLUP); // Ativa a resistencia pull-up interna no botao
8   pinMode(LED_vermelho, OUTPUT);
9   pinMode(LED_amarelo, OUTPUT);
10  pinMode(LED_verde, OUTPUT);
11 }
12
13 void loop() {
14   if (digitalRead(botao) == LOW) { // Botao pressionado
15     digitalWrite(LED_vermelho, HIGH); // LED vermelho aceso
16     delay(3000); // Aguarda 3 segundos
17     digitalWrite(LED_vermelho, LOW); // Desliga o LED vermelho
18     digitalWrite(LED_amarelo, HIGH); // Liga o LED amarelo
19     delay(1000); // Aguarda 1 segundo
20     digitalWrite(LED_amarelo, LOW); // Desliga o LED amarelo
21     digitalWrite(LED_verde, HIGH); // Liga o LED verde
22     delay(3000); // Aguarda 3 segundos
23     digitalWrite(LED_verde, LOW); // Desliga o LED verde
24   }
25 }
```

ARDUINO

Roteiro

1. Componentes Analógicos vs Digitais
2. Sinais Analógicos
3. Leitura de Sinais Analógicos e Conexão de Sensores
4. Conversão Analógico-Digital (A/D) no Arduino
5. Exemplos Práticos
6. Sinais Digitais
7. Lógica Digital com LEDs e Botões
8. Modulação por Largura de Pulso (PWM)
9. Controle de Intensidade em LEDs
10. Laboratórios



ARDUTENO

Analógicos vs Digitais

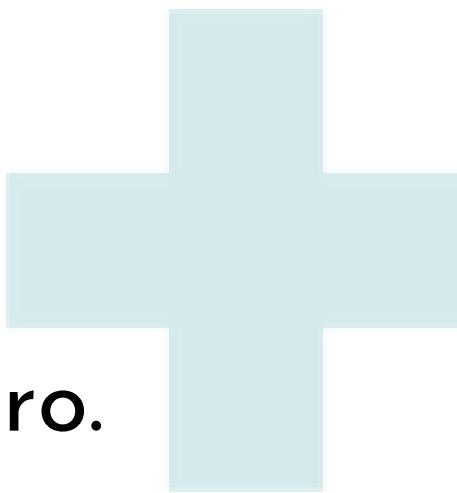
- **Natureza dos Sinais:**

- **Analógicos:**

- Contínuos, assumem infinitos valores.
 - Exemplo: Tensão variável em potenciômetro.

- **Digitais:**

- Discretos, limitados a 0 ou 1.
 - Exemplo: Sinal binário em um interruptor.



ARDUTENO

Precisão, Variações e Representação de Grandezas



- **Precisão e Variações:**

- **Analógicos:** Sensíveis a variações suaves. Precisos em transições suaves entre valores.
- **Digitais:** Menos sensíveis a variações suaves. Operam com valores discretos.

- **Representação de Grandezas:**

- **Analógicos:** Medição precisa de grandezas físicas.
 - **Exemplo:** Sensor de temperatura analógico.
- **Digitais:** Amplamente usados em lógica, automação e controle.
 - **Exemplo:** Botões, LEDs.

Processamento de Sinais e Conversão

- **Processamento de Sinais:**
 - **Analógicos:** Processamento contínuo, adequado para situações suaves.
 - **Digitais:** Processados por lógica booleana discreta, ideal para sistemas digitais.
- **Conversão:**
 - **Analógicos para Digitais (ADC):** Necessária para integrar dados analógicos em sistemas digitais.
 - **Digitais para Analógicos (DAC):** Necessária para gerar sinais analógicos a partir de dados digitais.

ARDUINO

Aplicações Típicas e Flexibilidade

- **Aplicações Típicas:**

- **Analógicos:** Sensores de luz, temperatura, pressão, amplificadores de áudio.
- **Digitais:** Lógica de computadores, controle de dispositivos digitais, comunicação digital.

- **Flexibilidade e Manipulação:**

- **Analógicos:** Mais flexíveis em termos de variações contínuas.
- **Digitais:** Facilidade de manipulação e armazenamento preciso de informações.

ARDUINO

Sinais Analógicos

Representações contínuas de fenômenos físicos. Variam suavemente ao longo do tempo.

- **Características Essenciais:**

- **Amplitude:** Intensidade do sinal.
- **Frequência:** Taxa de variação ao longo do tempo.

- **Aplicações Cruciais:**

- **Áudio:** Amplitude determina o volume.
- **Telecomunicações:** Frequência relacionada à transmissão de informações.

- **Sensores Analógicos:**

- **Exemplos:** termômetros, microfones.
- Geram sinais analógicos.

- **Conversão Analógico-Digital (ADC):**

- Necessária para integrar sinais analógicos em sistemas digitais.
- Fundamental em instrumentação e comunicações digitais avançadas.

Leitura de Sinais Analógicos

- **Conexão Física:**
 - Sensores analógicos conectados aos pinos analógicos do Arduino (por exemplo, A0, A1).
 - Permite a leitura da variação contínua desses sensores.
- **Resolução ADC:**
 - Arduino utiliza Conversor Analógico-Digital (ADC).
 - Determina precisão da conversão e quantidade de valores distintos representáveis.
- **Mapeamento de Valores:**
 - Uso da função map() para reescalonar valores analógicos.
 - Adaptado para atender às necessidades do projeto (controle de LEDs, servos motores, etc.).

Uso de Bibliotecas e Calibração

- **Uso de Bibliotecas:**

- Algumas bibliotecas podem ser necessárias para facilitar leitura e interpretação de sinais.
- Consultar a documentação do sensor para orientações.

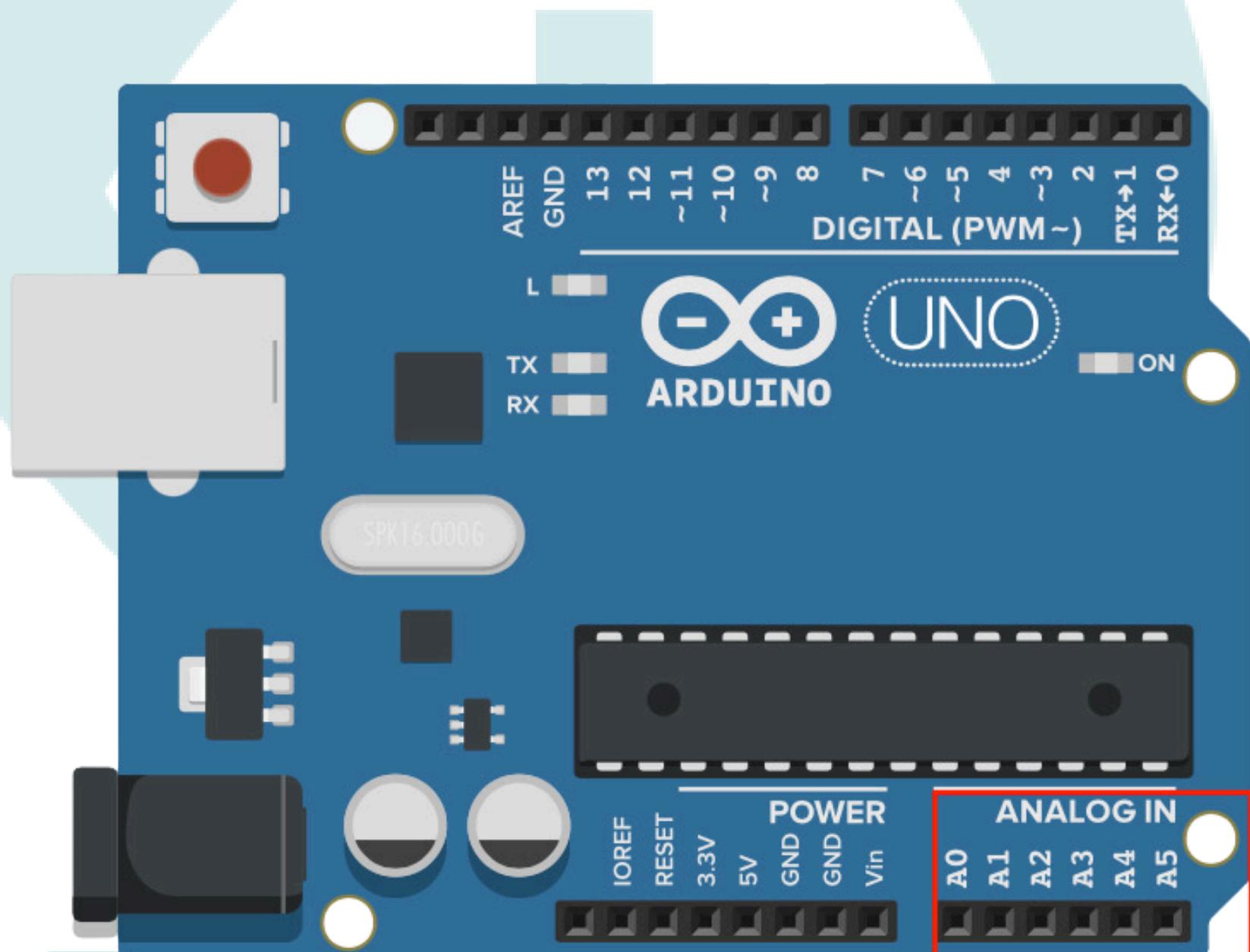
- **Calibração:**

- Alguns sensores podem exigir calibração para leituras precisas.
- Ajuste de código ou configurações para corresponder às condições específicas.
- Fatores como ruído elétrico podem afetar a precisão.
- Possibilidade de calibração para melhorar a precisão em casos críticos.

ARDUINO

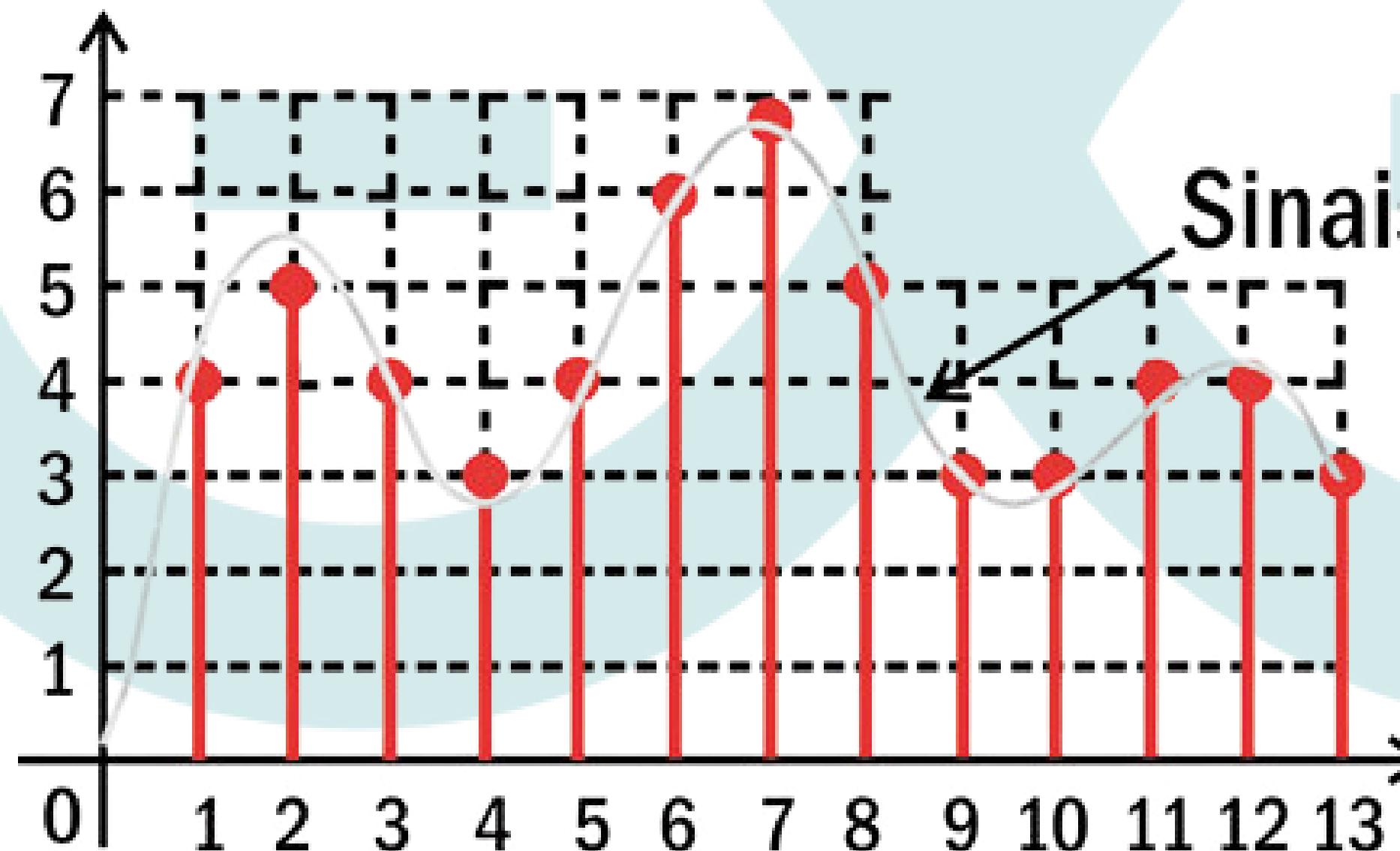
Pinos Analógicos e Aplicações

- Pinos Analógicos no Arduino:
 - A0, A1, A2, A3, A4, A5.
 - Essenciais para conexão de sensores analógicos.
- Aplicações:
 - Captura de informações contínuas do ambiente.
 - Variedade de usos, desde monitoramento ambiental até controle de dispositivos sensíveis



Entradas
Analógicas

Leitura de Sinais Analógicos



ARDUINO

Conversão Analógico-Digital

- **Entradas Analógicas:**
 - Pinos A0, A1, ..., A5 no Arduino medem sinais de tensão contínua de 0 a 5 volts.
- **Resolução A/D:**
 - **Resolução padrão no Arduino Uno:** 10 bits (1024 valores distintos).
- **Função analogRead():**
 - Utilizada para converter sinais analógicos em valores digitais.
- **Sintaxe:** analogRead(pin).

ARDUINO

Mapeamento de Tensão e Uso em Sensores

- **Mapeamento de Tensão:**

- Resultado da `analogRead()` mapeado para intervalo específico (por exemplo, 0 a 1023).
- Correspondência de valores a tensões, como 0 volts a 1023 a 5 volts.

- **Uso em Sensores Analógicos:**

- Potenciômetros, sensores de luz geram sinais variáveis.
- Conversão A/D permite interpretação desses sinais no Arduino.
- Essencial em projetos com sensores analógicos (temperatura, luminosidade, pressão).
- Tomada de decisões ou controle de dispositivos de saída.

ARDUTENO

Exemplo Prático 01

Um sensor envia uma tensão de 4,6V para o Arduino através de uma porta analógica. Convertendo a informação analógica para digital, e sabendo que o valor máximo possível é de 5V analógico equivale a 1023 em digital, temos a regra de três da seguinte maneira:

$$\begin{array}{rcl} 5 & \cdots\cdots & 1023 \\ 4,6 & \cdots\cdots & X \\ X = (4,6 * 1023) / 5 \\ X = 941 \end{array}$$

Portanto, ao receber o valor de 4,6V em uma entrada analógica, o Arduino interpretara o valor como 941 na escala digital.

Exemplo Prático 02

Um sensor envia uma tensão de 2,3V para o Arduino através de uma porta analógica. Convertendo essa informação analógica para digital, e considerando que o valor máximo possível é de 5V (análogo a 1023 em digital), podemos usar a regra de três:

$$\begin{array}{l} 5 \text{ ----- } 1023 \\ 2,3 \text{ ----- } X \\ X = (2,3 * 1023) / 5 \\ X = 472 \end{array}$$

Portanto, ao receber o valor de 2,3V em uma entrada analógica, o Arduino interpretara o valor como 472 na escala digital.

Exemplo Prático 03

Se um sensor gera uma tensão de 3,8V, e queremos saber a leitura digital correspondente em um Arduino com uma resolução de 10 bits (valores de 0 a 1023), a regra de três fica assim:

$$5 \text{ ----- 1023}$$

$$3,8 \text{ ----- } X$$

$$X = (3,8 * 1023) / 5$$

$$X = 779$$

Portanto, ao receber o valor de 3,8V em uma entrada analógica, o Arduino interpretara o valor como 779 na escala digital.

Sinais Digitais

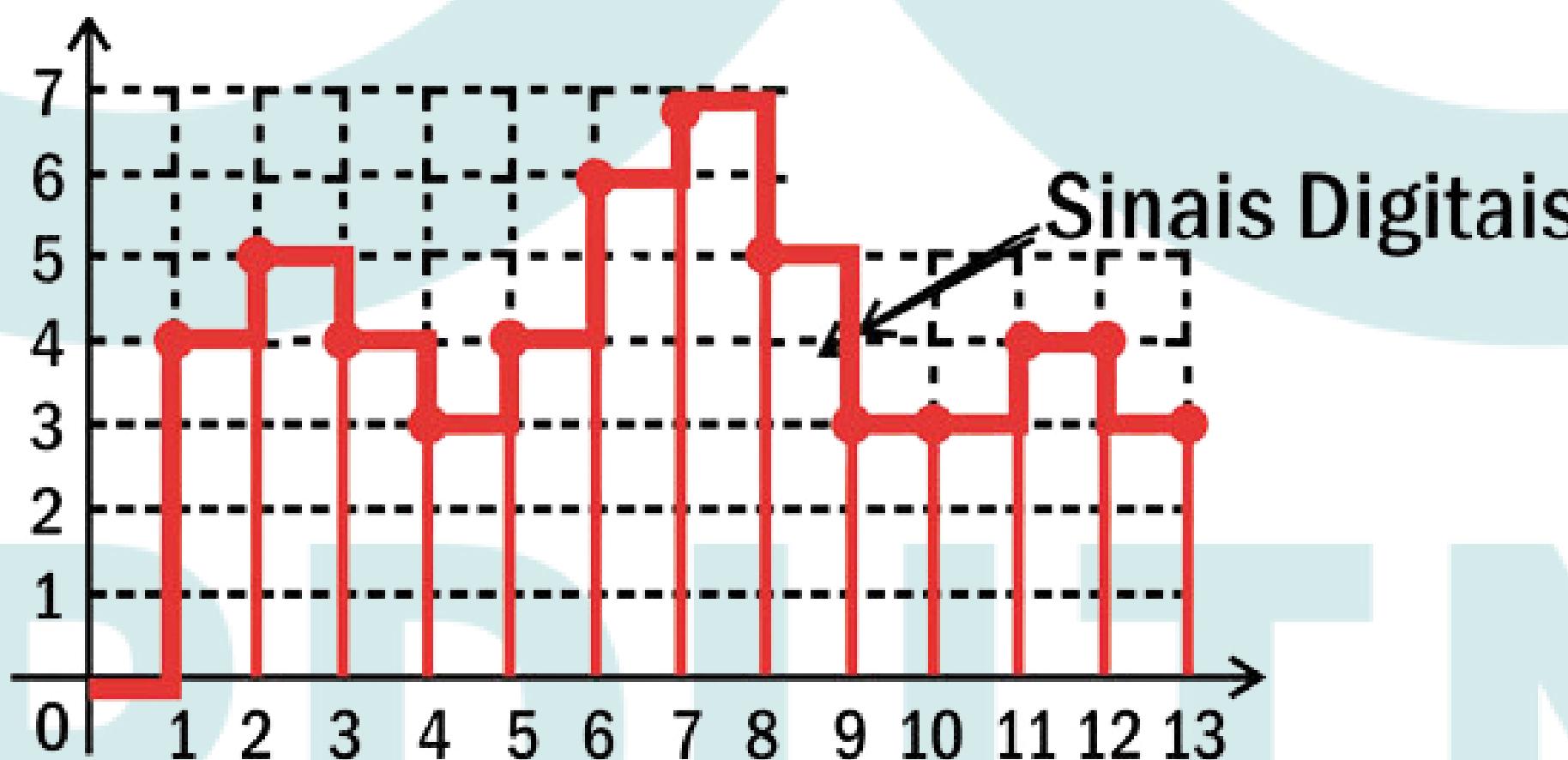
- **Representação binária:** 0 (OFF) e 1 (ON).
- Distinção da natureza discreta em relação aos sinais analógicos.
- Robustez e resistência a interferências.
- Vantagens na transmissão confiável de dados.
- Processo de digitalização de informações analógicas.
- Função das portas lógicas na eletrônica digital.
- Importância na construção de circuitos digitais.
- Sinais digitais como base para execução de algoritmos.
- Utilização de sinais digitais na comunicação serial.
- **Protocolos comuns:** UART, SPI, I2C.
- Codificação digital em sistemas de armazenamento.

ARDUINO

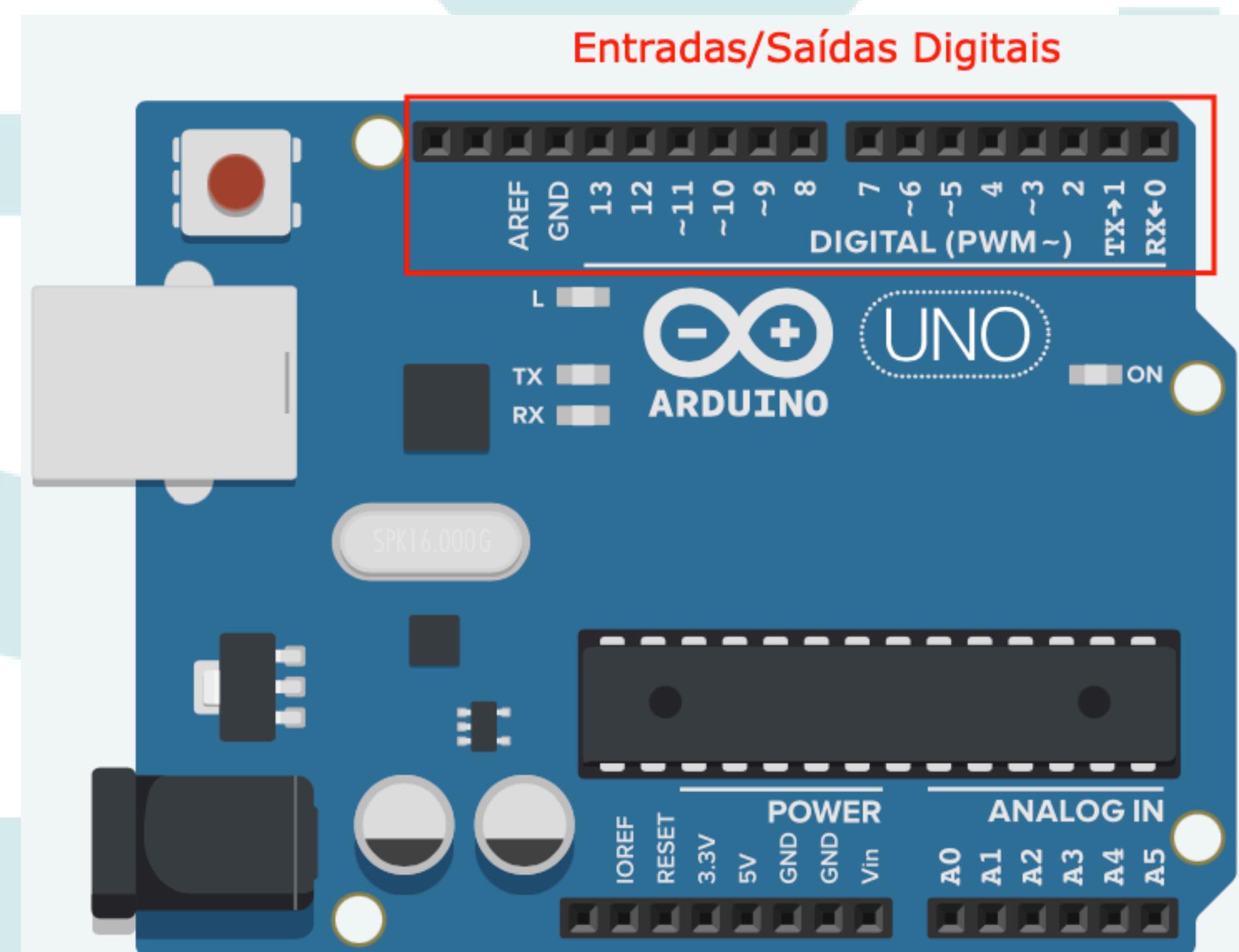
Exatidão, Processamento e Aplicações

Avançadas

- Exatidão e precisão na representação digital.
- Facilidade de processamento por circuitos digitais.
- **Aplicações avançadas:** multiplexação, compressão de dados.



Pinos Digitais no Arduino



Interação LED e Botão com Lógica Digital

- **Controle Bidirecional:** Lógica digital permite controle de LEDs e leitura de botões, estabelecendo interação bidirecional.
- **Sinais Binários:** LEDs representam estados digitais; botões geram sinais binários (0 ou 1) conforme pressionados.
- **Operações Booleanas:** Uso de operações como AND, OR e NOT para configurações complexas de controle.
- **Debouncing:** Implementação para garantir leituras precisas e filtrar ruídos em transições de estado nos botões.

ARDUINO

Dinâmicas e Aplicações da Lógica Digital

- **Contagem e Sequenciamento:** Lógica digital possibilita contagem de pressionamentos e criação de sequências de LEDs.
- **Simulação de Circuitos:** Utilização para simular circuitos complexos antes da implementação física.
- **Feedback Visual:** LEDs como feedback indicativo de estados, confirmações ou alertas.
- **Combinação com Temporizadores:** Integração com temporizadores para comportamentos temporais nos LEDs.
- **Aplicações Educacionais:** Uso comum em projetos educacionais para ensinar conceitos de eletrônica e programação.

ARDUINO

Modulação por Largura de Pulso (PWM)

- **Definição:** Técnica para controlar a potência entregue a dispositivos, variando a largura dos pulsos elétricos.
- **Conceito Básico:** Criação de pulsos de largura variável com frequência constante.
- **Propósito:** Simular sinais analógicos para controle preciso de dispositivos digitais.
- **Frequência e Ciclo de Trabalho:** Características do PWM: frequência (pulsos por segundo) e ciclo de trabalho (proporção do tempo em nível alto).
- **Controle de Velocidade e Intensidade:** Aplicações em motores para controle de velocidade e em LEDs para ajuste da intensidade luminosa.

ARDUINO

Implementação e Aplicações do PWM

- **Implementação no Arduino:** Uso da função `analogWrite()` para gerar sinais PWM e controlar dispositivos.
- **Exemplo Prático:** Controle de luminosidade em LEDs por variação do ciclo de trabalho.
- **Aplicações Avançadas:** Fundamental em sistemas de controle, fontes chaveadas, áudio digital, etc.
- **Eficiência Energética:** Minimiza desperdícios de energia, otimizando consumo em dispositivos eletrônicos.
- **Flexibilidade em Projetos:** Versatilidade do PWM em automação residencial, robótica e sistemas embarcados.

ARDUINO

Controle de Intensidade em LEDs com PWM

- **Princípio de Funcionamento:** PWM varia a largura dos pulsos elétricos para controlar a média de energia entregue ao LED.
- **Ciclo de Trabalho:** Controle da intensidade por meio da porcentagem de tempo que o LED fica ligado durante um ciclo completo.
- **Implementação no Arduino:** Função `analogWrite(pin, value)` no Arduino ajusta o ciclo de trabalho, proporcionando controle suave da intensidade.
- **Aplicações Práticas:** Amplamente utilizado em iluminação residencial, cenográfica e sinalização. Oferece flexibilidade em projetos de automação.

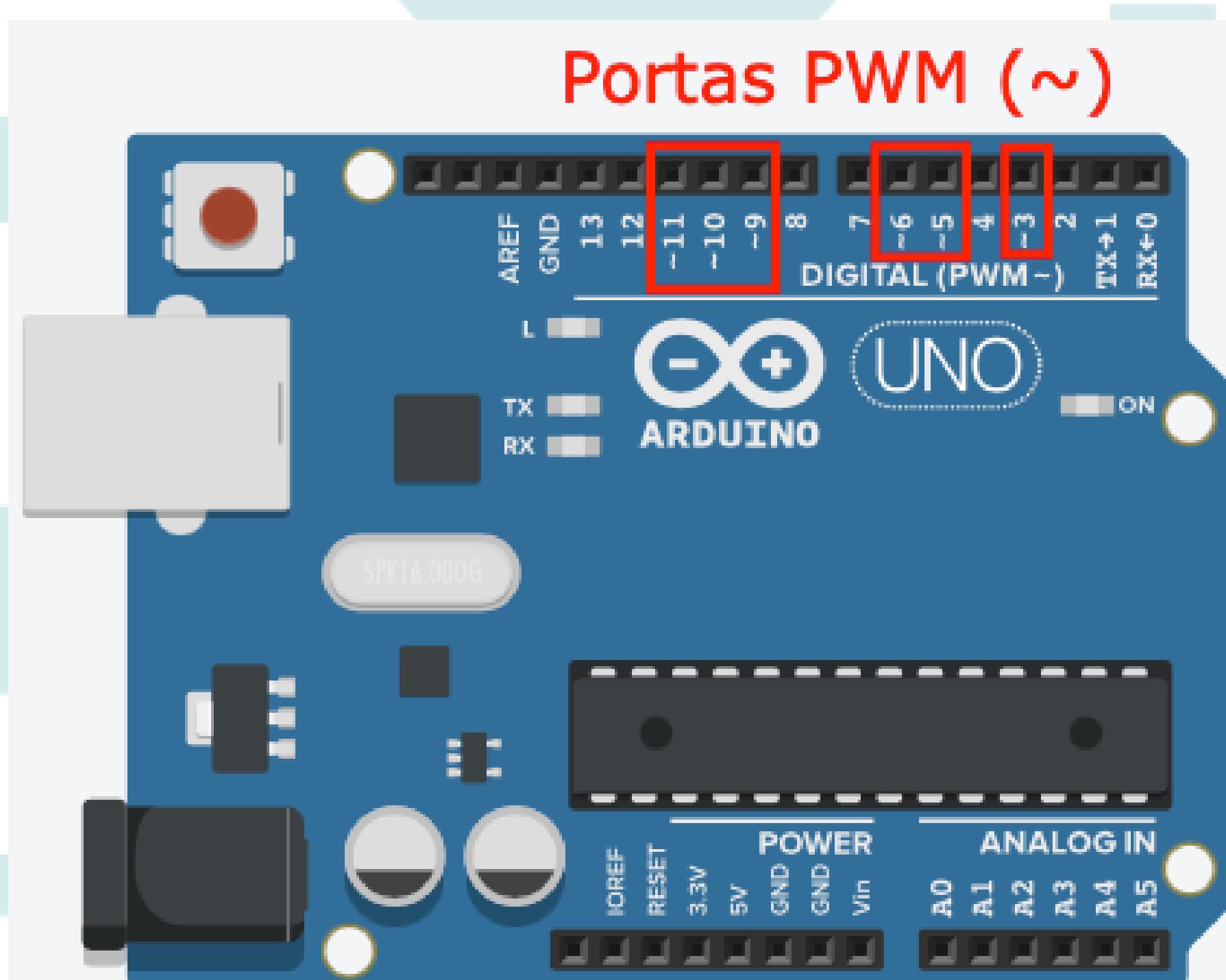
ARDUTINO

Benefícios e Aplicações Avançadas de PWM

- **Economia de Energia:** Destaca-se pela eficiência energética, reduzindo desperdício no ajuste da luminosidade.
- **Variedade de Cores:** Aplicado em LEDs RGB para controlar não apenas a intensidade, mas também criar diversas cores.
- **Experimentação e Aprendizado:** Facilita a compreensão prática de conceitos fundamentais de eletrônica e programação.
- **Exemplo Prático:** Código demonstrando o controle da intensidade luminosa de um LED com PWM.

```
1 void loop() {  
2     //Intensidade mais baixa.  
3     analogWrite(LED, 0);  
4     delay(500); // Esperar meio segundo  
5  
6     //Ligado com 25% da intensidade.  
7     analogWrite(LED, 64);  
8     delay(500); // Esperar meio segundo  
9  
10    //Ligado com 50% da intensidade.  
11    analogWrite(LED, 128);  
12    delay(500); // Esperar meio segundo  
13  
14    //Ligado com 75% da intensidade.  
15    analogWrite(LED, 192);  
16    delay(500); // Esperar meio segundo  
17  
18    //Ligado com 100% da intensidade.  
19    analogWrite(LED, 255);  
20    delay(500); // Esperar meio segundo  
21 }
```

Pinos PWM no Arduino



ARDUINO

Laboratório 01

Aumentando a intensidade luminosa com botão.

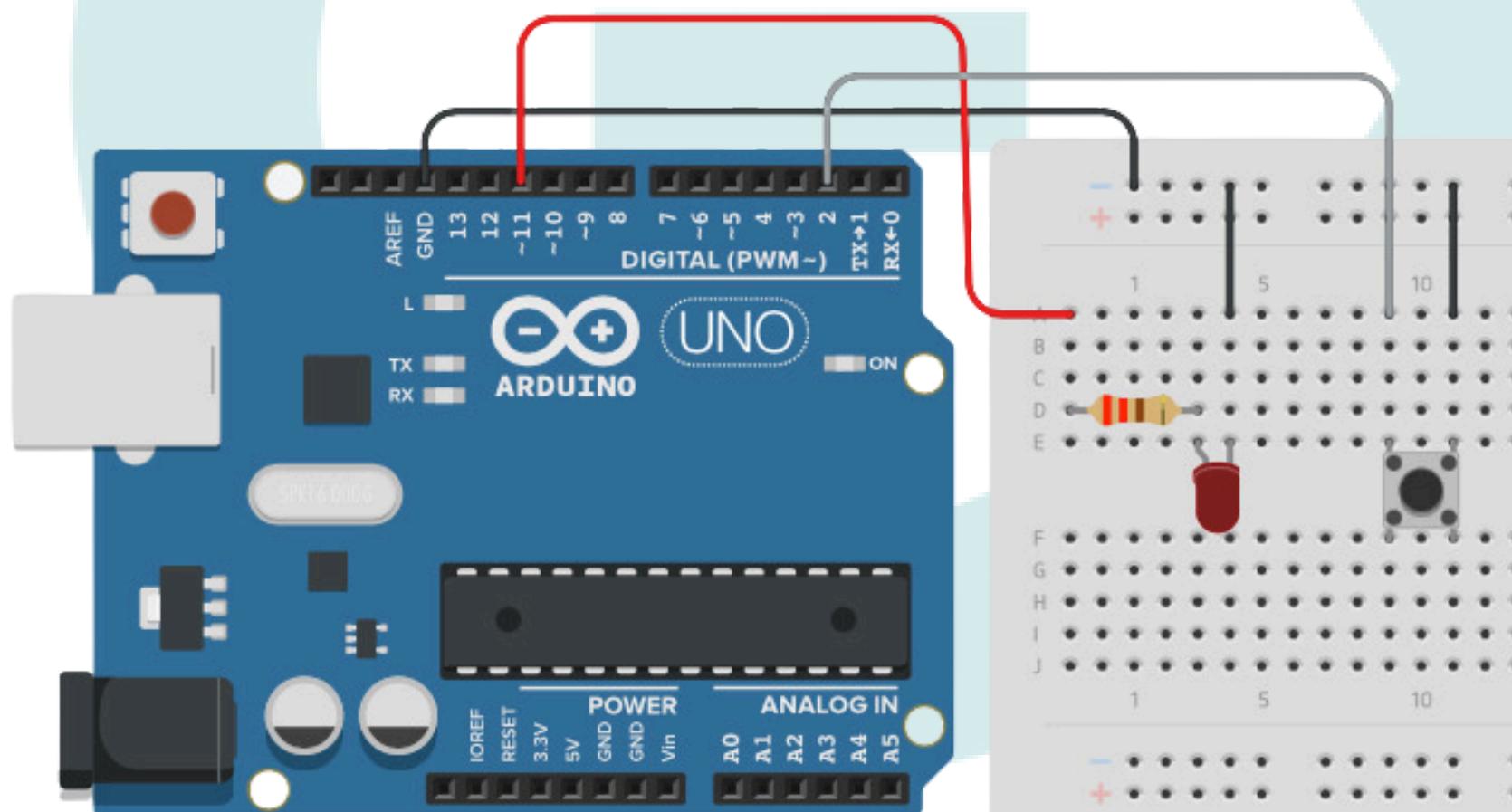
Neste guia, você aprenderá a mudar a intensidade luminosa de um LED através de PWM.

Materiais Necessários:

- Arduino Uno, Protoboard e Jumpers.
- 1 LED.
- Botão de pressão.
- Resistor de 220Ω (para o LED).

ARDUINO

Solução Laboratório 01



```

1 int luminosidade = 0; //Vai armazenar o valor do PWM
2 int botao = 2;
3 int LED_VERMELHO = 11;
4
5 void setup() {
6     pinMode(botao, INPUT_PULLUP);
7     pinMode(LED_VERMELHO, OUTPUT);
8 }
9
10
11 void loop() {
12     //Cada vez que o botao for apertado, aumenta em 1 a intensidade do PWM ligado.
13     //Quando chegar em 255, a intensidade sera maxima, e nao podera passar deste valor.
14     if(digitalRead(botao) == LOW) {
15         luminosidade = luminosidade + 1;
16         if(luminosidade > 255) {
17             luminosidade = 255;
18         }
19     }
20     analogWrite(LED_VERMELHO, luminosidade);
21     delay(50);
22     //Funcao 'analogWrite' utilizada para definir a intensidade do PWM ligado
23     //intensidade armazenada na palavra 'luminosidade'
24 }
25

```

ARDUINO

Laboratório 02

Aumentando a intensidade luminosa com um botão e diminuindo com outro.

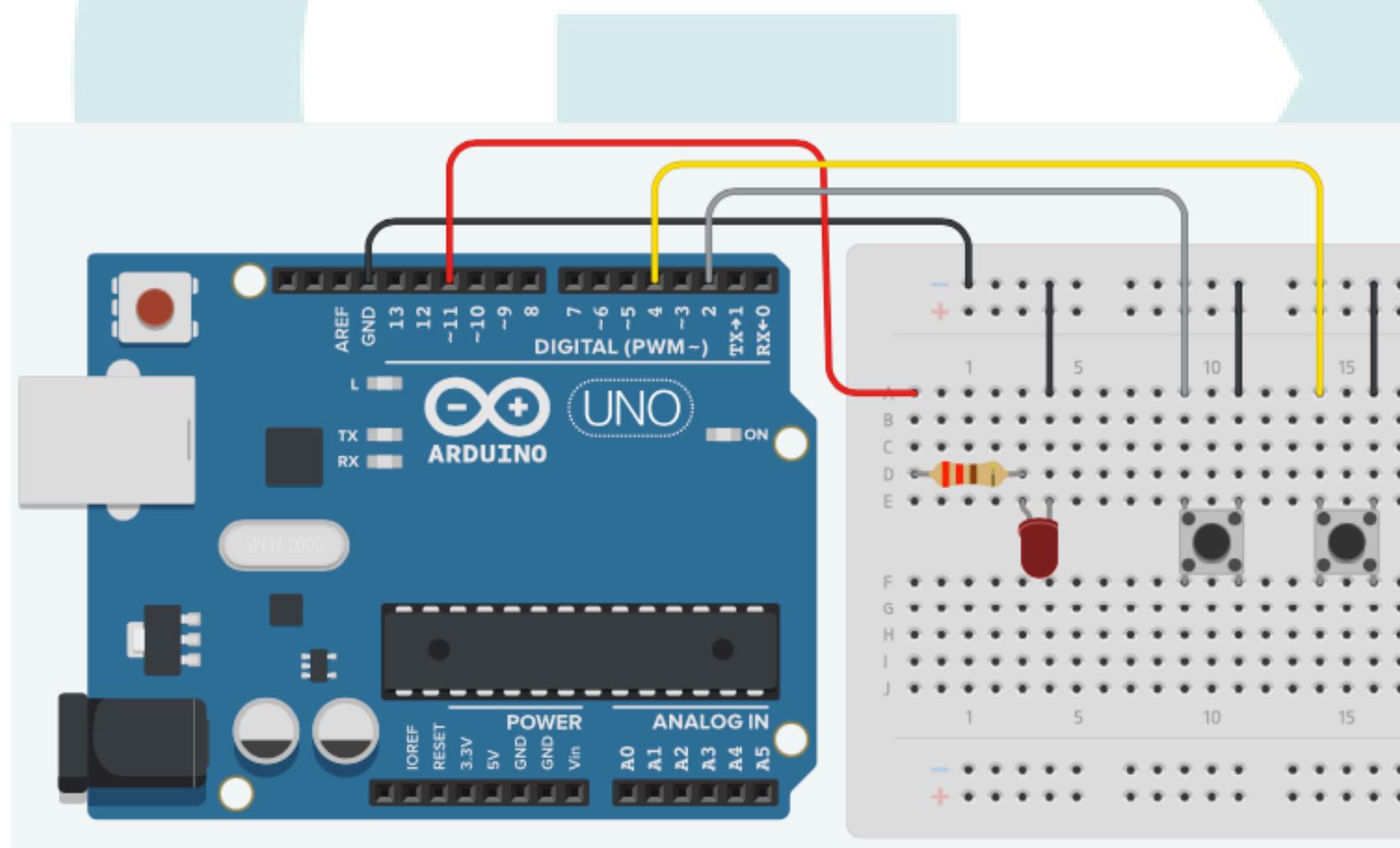
Neste guia, você aprenderá a aumentar e diminuir a intensidade luminosa de um LED através de PWM.

Materiais Necessários:

- Arduino Uno, Protoboard e Jumpers.
- 1 LED.
- 2 Botões de pressão.
- Resistor de 220Ω (para o LED).

ARDUTINO

Solução Laboratório 02



```

1 int luminosidade = 0; //Vai armazenar o valor do PWM
2 int botao_aumenta = 2;
3 int botao_diminui = 4;
4 int LED_VERMELHO = 11;
5
6 void setup() {
7     pinMode(botao_aumenta, INPUT_PULLUP);
8     pinMode(botao_diminui, INPUT_PULLUP);
9     pinMode(LED_VERMELHO, OUTPUT);
10 }
11
12
13 void loop() {
14     //Cada vez que o botao for apertado, aumenta em 1 a intensidade do PWM ligado.
15     //Quando chegar em 255, a intensidade sera maxima, e nao podera passar deste valor.
16     if(digitalRead(botao_aumenta) == LOW) {
17         luminosidade = luminosidade + 1;
18         if(luminosidade > 255) {
19             luminosidade = 255;
20         }
21     }
22     //Quando chegar em 0, a intensidade sera minima, e nao podera passar deste valor.
23     if(digitalRead(botao_diminui) == LOW) {
24         luminosidade = luminosidade - 1;
25         if(luminosidade < 0) {
26             luminosidade = 0;
27         }
28     }
29     analogWrite(LED_VERMELHO, luminosidade);
30     delay(50);
31     //Funcao 'analogWrite' utilizada para definir a intensidade do PWM ligado
32     //intensidade armazenada na palavra 'luminosidade'
33 }
34

```

ARDUINO