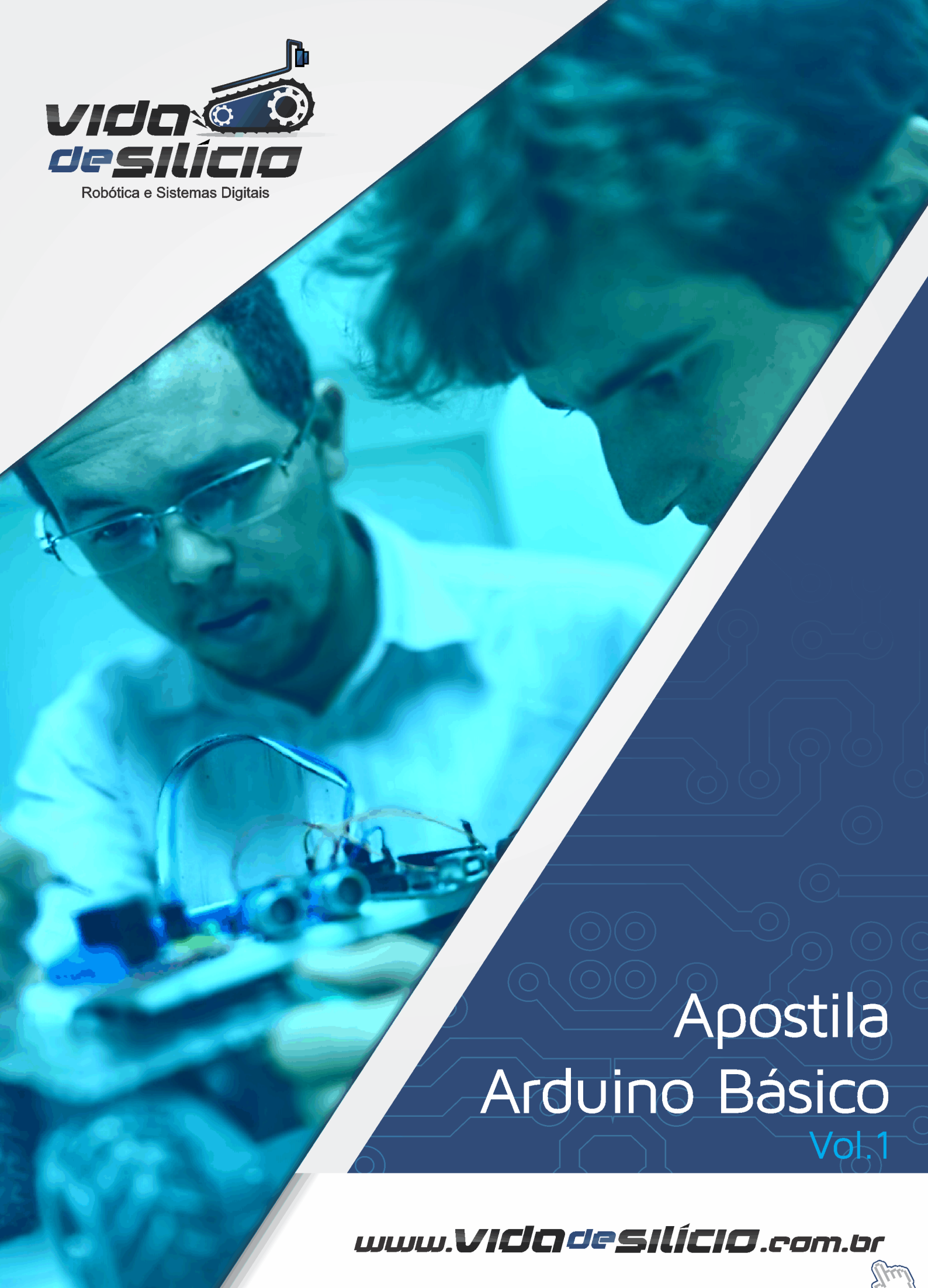
Vida de Silício

Robótica e Sistemas Digitais



Apostila Arduino Básico

Vol.1

www.vidadesilicio.com.br

Sumário

1. Introdução

1.1. Arduino

Há não muito tempo, para se confeccionar um circuito interativo, era necessário fazer projetos do zero para uma aplicação específica. Para se fazer pequenas alterações nas funcionalidades do circuito era necessário um estudo crítico e bastante trabalho.

Com o advento dos microcontroladores, foi possível que problemas que eram tratados com hardware fossem tratados usando software de computadores. Dessa forma, um mesmo circuito poderia tomar funções totalmente diferentes, reprogramando e alterando alguns parâmetros do programa.

Mas mesmo assim, trabalhar com microcontroladores não é tão trivial. Desta forma, um grupo de pesquisadores italianos teve a ideia de fazer um dispositivo que tornasse o seu uso simples e acessível a qualquer um. O resultado foi o Arduino.

A filosofia é fazer com que qualquer pessoa possa criar um projeto interativo, sem a necessidade de ter que aprender sobre matérias complexas de engenharia. Dessa forma, qualquer um pode ser um criador de tecnologia, não importando idade ou especialidade, apenas algo presente em sobra no ser humano: a criatividade.

1.2. O que realmente é o Arduino?

No site oficial da Arduino, encontramos a seguinte definição (traduzida):

"*Arduino é uma plataforma open-source de prototipagem eletrônica com hardware e software flexíveis e fáceis de usar, destinado a artistas, designers, hobbistas e qualquer pessoa interessada em criar objetos ou ambientes interativos*."

Ou seja, o Arduino é uma plataforma formada por dois componentes: A placa, que é o Hardware que usaremos para construir nossos projetos e a IDE Arduino, que é o Software onde escrevemos o que queremos que a placa faça.

A maior vantagem dessa plataforma de desenvolvimento sobre as demais é a sua facilidade de sua utilização: pessoas que não são da área técnica podem, rapidamente, aprender o básico e criar seus próprios projetos em um intervalo de tempo relativamente curto.

1.2.1. A placa Arduino

O hardware do Arduino é simples, porém muito eficiente. Vamos analisar a partir desse momento o hardware do Arduino UNO. Ele é composto pelos seguintes blocos:

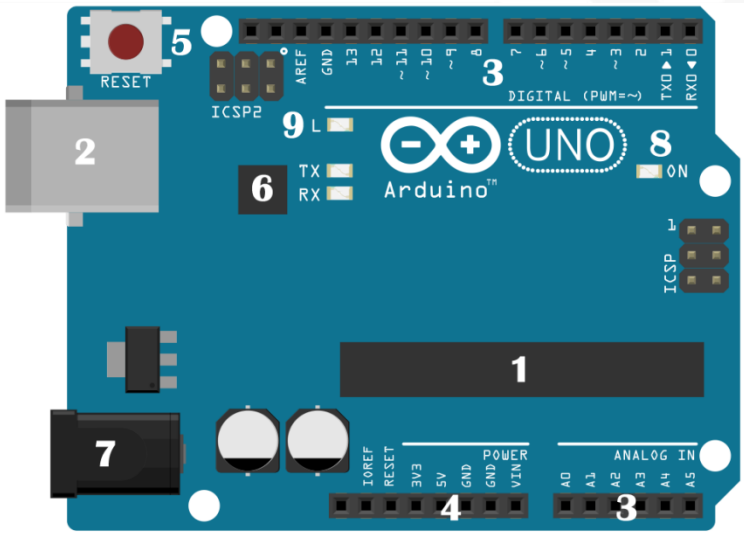


Figura 1 - Arduino Uno

1. **Microcontrolador:** O cérebro do Arduino. Um computador inteiro dentro de um pequeno chip. Este é o dispositivo programável que roda o código que enviamos à placa. No mercado existem várias opções de marcas e modelos de microcontroladores. A Arduino optou pelo uso dos chips da ATmel, a linha ATmega. O Arduino UNO usa o microcontrolador ATmega32.
2. **Conector USB:** Conecta o Arduino ao computador. É por onde o computador e o Arduino se comunicam com o auxílio de um cabo USB, além de ser uma opção de alimentação da placa.
3. **Pinos de Entrada e Saída:** Pinos que podem ser programados para agirem como entradas ou saídas fazendo com que o Arduino interaja com o meio externo. O Arduino UNO possui 14 portas digitais (I/O), 6 pinos de entrada analógica e 6 saídas analógicas (PWM).
4. **Pinos de Alimentação:** Fornecem diversos valores de tensão que podem ser utilizados para energizar os componentes do seu projeto. Devem ser usados com cuidado, para que não sejam forçados a fornecer valores de corrente superiores ao suportado pela placa.
5. **Botão de Reset:** Botão que reinicia a placa Arduino.
6. **Conversor Serial-USB e LEDs TX/RX:** Para que o computador e o microcontrolador conversem, é necessário que exista um chip que traduza as

informações vindas de um para o outro. Os LEDs TX e RX acendem quando o Arduino está transmitindo e recebendo dados pela porta serial respectivamente.

7. **Conector de Alimentação:** Responsável por receber a energia de alimentação externa, que pode ter uma tensão de no mínimo 7 Volts e no máximo 20 Volts e uma corrente mínima de 300mA. Recomendamos 9V, com um pino redondo de 2,1mm e centro positivo. Caso a placa também esteja sendo alimentada pelo cabo USB, ele dará preferência à fonte externa automaticamente.
8. **LED de Alimentação:** Indica se a placa está energizada.
9. **LED Interno:** LED conectado ao pino digital 13.

1.2.2. IDE Arduino

Quando tratamos de software na plataforma Arduino, podemos referir-nos ao ambiente de desenvolvimento integrado do Arduino e o programa desenvolvido por nós para enviar para a nossa placa.

Uma das grandes vantagens dessa plataforma está no seu ambiente de desenvolvimento, que usa uma linguagem baseada no C/C++, linguagem bem difundida, usando uma estrutura simples. Mesmo pessoas sem conhecimento algum em programação conseguem, com pouco estudo, elaborar programas rapidamente.

1.2.2.1. Baixando e instalando a IDE Arduino

Acesse o site oficial da Arduino (www.arduino.cc). No site, clique na aba Download.

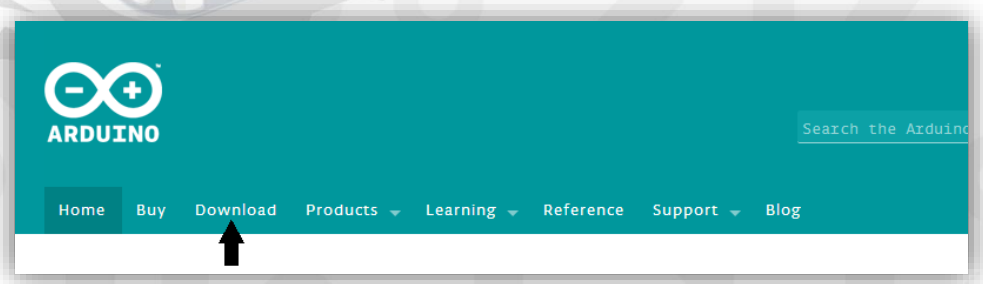


Figura 2 - Site oficial Arduino

Na página Download, procure pela última versão do Arduino IDE. No dia em que escrevo essa apostila, a última versão, ainda BETA, é a 1.5.8.

Windows

Primeira Opção: baixar o instalador (Installer) que funciona como qualquer outro instalador de programa.

Segunda Opção: Baixar todos os arquivos da IDE Arduino compactados para Windows (ZIP file), nessa versão basta baixar e descompactar na pasta que você desejar, inclusive no seu pendriver ou HD virtual. Eu costumo descompactar na Área de Trabalho.

Linux

Baixar todos os arquivos da IDE Arduino compactados para Linux (32bit ou 64bit), nessa versão basta baixar e descompactar na pasta que você desejar, inclusive no seu pendriver ou HD virtual. Eu costumo descompactar na Área de Trabalho.

1.2.2.2. Entendendo a IDE Arduino

Em resumo, é um program simples de se utilizar e de entender com bibliotecas que podem ser facilmente encontradas na internet. As funções da IDE do Arduino são basicamente três: permitir o desenvolvimento do software, de enviá-lo à placa para que possa ser executado e de se interagir com a placa Arduino.

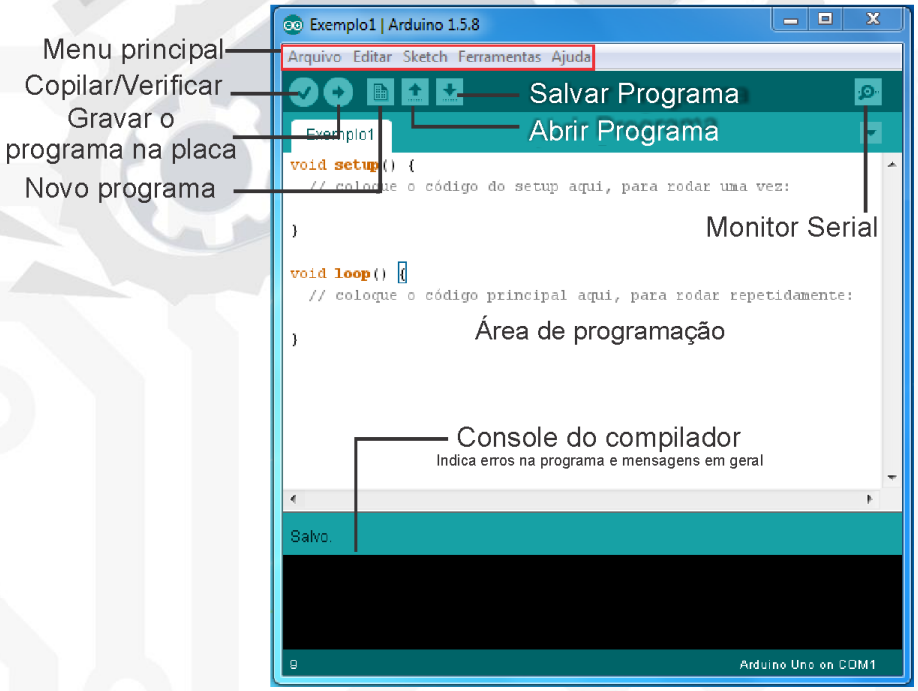


Figura 3 - IDE Arduino

2. Saídas Digitais

2.1. Experiência 1 – Blink

Vamos à nossa primeira experiência. Nessa experiência iremos fazer o Arduino piscar um LED de um em um segundo.

O Arduino UNO REV3 possui um LED na placa ligado à porta digital 13. Com isso, podemos testar o funcionamento de forma rápida e simples.

2.1.1. Programando

Com seu Arduino conectado ao computador e devidamente configurado, abra o menu File > Examples > 01. Basics > Blink. Uma nova janela deve ser aberta com um texto escrito, esse é o programa.

```
       /* Blink - Pisca um led, de um em um segundo

  Este código exemplo é de domínio publico. */

// Existe um LED conectado no pino 13 da maioria dos Arduinos
// Daremos um nome a este pino:
int led = 13;

// Esta função "setup" roda uma vez quando a placa e ligada ou resetada

void setup() {
  // Configura o pino do led (digital) como saída
  pinMode(led, OUTPUT);
}

// Função que se repete infinitamente quando a placa é ligada
void loop() {
  digitalWrite(led, HIGH); // Liga o LED (HIGH = nível lógico alto)
  delay(1000);             // Espera um segundo
  digitalWrite(led, LOW);  // Desliga o LED (LOW = nível lógico baixo)
  delay(1000);             // Espera um segundo
}
```

Agora clique em Upload para que o programa seja transferido para seu Arduino.

Se tudo foi feito corretamente, o led do pino 13, presente na placa do Arduino UNO, irá piscar intermitentemente.

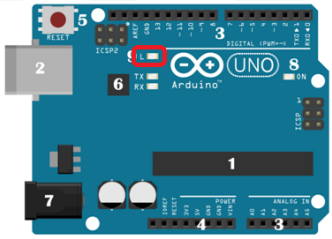


Figura 4 - Led conectado ao pino 13

2.1.2. Grandezas Digitais e Analógicas

Para entendermos o Arduino, é necessário que saibamos diferenciar uma grandeza analógica de uma grandeza digital.

Digital x Analógico

Grandezas digitais são aquelas que não variam continuamente no tempo, mas sim em saltos entre valores bem definidos. Um exemplo são os relógios digitais: apesar do tempo em si variar continuamente, o visor do relógio mostra o tempo em saltos de um em um segundo. Um relógio desse tipo nunca mostrará 12,5 segundos, pois, para ele, só existem 12 e 13 segundos. Qualquer valor intermediário não está definido.

Grandezas analógicas são aquelas que, ao contrário das grandezas digitais, variam continuamente dentro de uma faixa de valores. O velocímetro de um carro, por exemplo, pode ser considerado analógico, pois o ponteiro gira continuamente conforme o automóvel acelera ou freia. Se o ponteiro girasse em saltos, o velocímetro seria considerado digital.

Outra analogia interessante pode ser feita comparando uma escada com uma rampa: enquanto uma rampa sobe de forma contínua, assumindo todos os valores de altura entre a base e o topo, a escada sobe em saltos, com apenas alguns valores de altura definidos entre a base e o topo. A escada representa, portanto, uma grandeza digital, enquanto a rampa representa uma grandeza analógica. A quantidade de degraus em uma escada define quais posições podemos escolher. Por exemplo se uma escada tem um degrau em 1,00 m de altura do solo e o próximo está a 1,50 m nós não podemos ocultar a posição 1,38 m do solo porque não existe um degrau lá. Quanto mais degraus adicionamos em um intervalo de altura menor mais perto da rampa nos aproximamos.

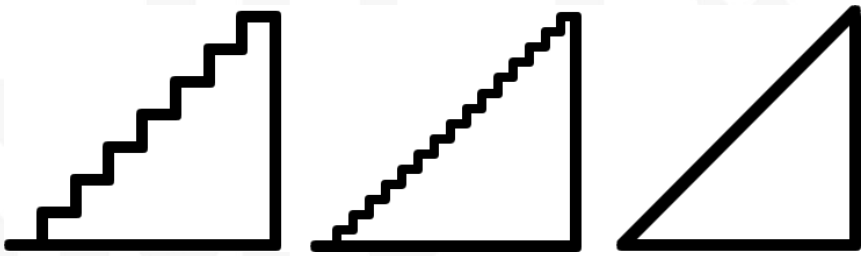


Figura 5 - Sinal analógico

Os circuitos e equipamentos elétricos ditos digitais trabalham com apenas dois valores de tensão definidos:

- Um nível lógico **alto**, que no caso do Arduino é 5V;
- Um nível lógico **baixo**, que no caso do Arduino é 0V.

Na prática existem faixas de valores próximos a esses números em que o circuito digital entende como nível alto ou baixo. Também existe uma faixa intermediária não definida que pode gerar resultados inesperados e que, portanto, deve ser evitada.

Os circuitos e equipamentos elétricos ditos analógicos trabalham com uma faixa de valores chamada de range:

- No Arduino essa faixa de valores (range) vai de 0V a 5V.

As placas Arduino possuem uma clara divisão entre os pinos de entrada e saída digitais/analógicos, porém em algumas placas como o modelo Arduino Uno qualquer pino pode ser utilizado como entrada ou saída digital.

2.1.3. Entendendo o programa

- *Pensando como um computador*

Computadores e microcontroladores não possuem uma inteligência tal como um ser humano ou um animal. Eles são projetados para resolver problemas a partir de uma lista de tarefas, semelhante a uma receita de bolo.

Dessa forma, para resolver um problema em um computador, tal como fazemos para resolver problemas cotidianos, fazer um bolo, por exemplo, devemos descrever a solução de uma forma clara e precisa, por meio de passos a serem seguidos até que se atinja um resultado esperado.

O nome dessa lista de passos é o algoritmo. Um algoritmo é um conjunto finito de regras que fornece uma sequência de operações a fim de solucionar um problema.

Veja o exemplo da seguinte receita de bolo:

```
// RECEITA DE BOLO COMUM DE OVOS
INÍCIO
Passo 1: Separar os ingredientes
Ingredientes:
2 ovos;
3 xícaras de farinha de trigo;
1 e ½ colher de fermento;
¾ xícara de leite.
1/2 xícaras de açúcar;
250g de manteiga;


Modo de preparo:
Passo 2: Aqueça o forno a 180 graus;
Passo 3: Quebre os ovos e separe as claras da gema;
Passo 4: Bata as claras em neve e as deixe separadas;
Passo 5: Em uma vasilha, bata o açúcar, a manteiga e as gemas;
Passo 6: Misture a farinha e o leite;
Passo 7: Bata bem, até ficar bem homogêneo;
Passo 8: Acrescente o fermento;
Passo 9: Adicione as claras em neve e mexa cuidadosamente;
Passo 10: Unte uma forma com manteiga e farinha de trigo.
Passo 11: Coloque a massa na forma untada
Passo 12: Leve ao forno médio para assar por aproximadamente 35 minutos ou até que, ao
espetar um palito, esse saia seco;
Passo 13: Após assado, desligue o forno e deixe o bolo esfriar;
Passo 11: Desenforme e saboreie.
FIM
```

Essa receita se assemelha aos algoritmos que iremos fazer ao longo dessa apostila. Ele define o que iremos usar, e depois diz passo a passo o que devemos fazer.

Veja que não podemos fazer o passo 4 sem antes fazer o passo 3, nem o passo 11 sem os passos anteriores. Existe uma ordem a ser seguida para que o bolo fique conforme esperado.

Não é diferente para o computador, ele precisa de ordens organizadas e coerentes, que ao final da receita resulte na solução de seu problema.

Além da ordem de passos, você deve falar em uma língua que o computador entenda, não adianta uma receita em grego para um brasileiro. Para o nosso caso usamos uma linguagem baseada em C++. Tal como qualquer língua, ela possui suas regras de como usá-la adequadamente.

Despois de traduzido o algoritmo para a linguagem do Arduino, teremos o código-fonte, código de programa ou simplesmente programa. Muitos gostam de chamar também de sketch.

- *Sempre faça comentários*

Sempre quando escrevemos um programa é importante que deixemos notas explicando o seu uso, tanto para que outras pessoas entendam quanto para nós mesmo.

Muitas vezes fazemos um programa para uma determinada função achando que só iremos usá-lo uma única vez. Quando, depois de muito tempo, nos damos conta de que precisaremos dele novamente, já é tarde. Por mais que você seja bom de memória, é muito difícil lembrar de tudo que você pensou durante a confecção de seu programa.

Dessa forma, quando não deixamos notas explicando como o programa funciona, perdemos tempo relendo todo o programa para entendê-lo. Além disso, caso alguém peça seu programa emprestado, se o mesmo não estiver comentado, será de difícil entendimento.

Por isso, temos códigos especiais na IDE Arduino para comentários. Lembre-se de sempre usá-las.

Em resumo, um comentário é um texto que será desconsiderado na hora de rodar o programa, ele não tem valor nenhum para o Arduino. Na IDE Arduino, os comentários ficam em tonalidade cinza.

Existem duas formas de fazer comentários, podemos comentar um bloco de texto ou podemos apenas comentar uma linha.

Para comentar um bloco de texto devemos definir o comentário indicando o início usando /* (barra e asterisco) e o final usando */ (asterisco e barra).

```
/* Blink - Pisca um led, de um em um segundo

  Este código exemplo é de domínio publico. */
```

Para comentar uma linha, definimos apenas o início do comentário, usando // (duas barras). Não é necessário que se comece um comentário de linha no início da linha.

```
// Existe um LED conectado no pino 13 da maioria dos Arduinos

digitalWrite(led, HIGH); // Liga o LED (HIGH = nível lógico alto)
```

Às vezes, quando queremos testar o programa ignorando uma linha do código, simplesmente comentamos ela. Dessa forma ela não será lida na hora de carregar o programa para o Arduino.

```
 //digitalWrite(led, HIGH); Essa linha não é lida pelo Arduino
```

- *Variáveis*

Quando programamos, precisamos continuamente guardar uma informação para usarmos posteriormente. Por exemplo, caso você queira somar dois números, você terá que armazenar o resultado em algum lugar. Esse lugar reservado se chama variável.

Neste exemplo o lugar seria x, que recebe o resultado da soma 1+1.

```
x = 1 + 1; //x recebe 2, que é o resultado da soma 1+1
```

Imagine que a memória do seu Arduino possui muitas gavetas, quando queremos usar uma, devemos dar um nome a ela e falar qual tipo de conteúdo ela guardará. Cada gaveta é uma variável.

Ao declarar uma variável devemos definir qual tipo de conteúdo ela receberá. Para isso você deve saber com qual tipo de dado mexerá. Em nosso exemplo, x deve ser uma variável do tipo int, isso porque ela receberá um valor inteiro.

```
int x;  //x é uma variável do tipo inteiro
x = 1 + 1; //x recebe 2, que é o resultado da soma 1+1
```

No Apêndice 1 você pode conferir uma tabela com os principais tipos de variáveis e suas particularidades.

Ao criar uma variável, é interessante que ela tenha um nome associado à informação que ela armazena. No exemplo anterior, no lugar x poderíamos ter usado o nome *soma* ou *resultado*.

Em nosso programa declaramos a variável do tipo **int** chamada led que armazena o número 13, que é o número da porta onde está localizado o LED que pisca com a execução do programa. Dessa forma, quando escrevemos led em nosso programa, ele corresponderá a 13, que está escrito dentro da variável.

```
int led = 13;
```

- *Função*

Muitas vezes desempenhamos um conjunto de instruções. Por exemplo: pense em uma situação cotidiana, constantemente precisamos saber qual é a hora. Suponhamos que para isso você desempenhe o seguinte conjunto de instruções.

```
Pegue seu celular;
Ligue a tela;
Leia qual é a hora;
Desligue a tela;
Guarde o celular;
```

Para facilitar nossa vida, podemos chamar esse conjunto de instruções por um nome para que sempre que quisermos saber a hora, não seja preciso descrever todas as instruções necessárias para isso.

```
void lerHora(){
  Pegue seu celular;
  Ligue a tela;
  Leia qual é a hora;
  Desligue a tela;
  Guarde o celular;
}
```

Em resumo, função é um bloco de tarefas a serem executadas pelo programa quando solicitada. Agora, quando escrevemos Lerhora(), estamos chamando o conjunto de instruções correspondentes.

```
lerHora(); // Excuta o conjunto de instruções da função lerHora
```

Toda função deve ter um nome e um tipo. No caso das duas, o tipo é void e o nome é setup e loop, respectivamente. Para sabermos onde começa e termina o bloco de tarefas de uma função usamos "{" para indicar o início e "}" para indicar o final.

- *Função setup() e função loop()*

No Arduino existem duas funções que são fundamentais ao programa, a setup e a loop. Elas são chamadas automaticamente pelo microcontrolador quando ligado:

A função setup é executada apenas uma vez ao ligar ou reiniciar sua placa Arduino. Nela colocamos todas as instruções que queremos que o Arduino execute apenas uma vez. Geralmente, é nela que definimos parâmetros iniciais de nosso programa.

```
// Esta rotina "setup" roda uma vez quando a placa e ligada ou reiniciada
void setup() {
  // Configura o pino do led (digital) como saída
  pinMode(led, OUTPUT); // led corresponde a 13
}
```

Nesse exemplo, o único comando contido na função setup é o *pinMode*, esse comando é muito importante no uso das entradas e saídas digitais do Arduino. Com esse comando definimos se o pino é uma entrada (input) ou uma saída (output). No nosso caso queremos que o pino 13 seja uma saída.

```
  pinMode(led, OUTPUT); // led corresponde à 13
```

A função loop é executada ciclicamente pelo Arduino. Dessa forma, ao chegar na última linha, ele volta para a primeira. Sendo assim, nessa função escrevemos as funções que queremos que ocorram continuamente.

```
// Função que repete infinitamente quando a placa é ligada
void loop() {
  digitalWrite(led, HIGH); // Liga o LED (HIGH = nivel lógico alto)
  delay(1000);             // Espera um segundo
  digitalWrite(led, LOW);  // Desliga o LED (LOW = nivel lógico baixo)
  delay(1000);             // Espera um segundo
}
```

Na função loop temos dois comandos. O primeiro comando, o *digitalWrite*, define qual o estado de saída, **LOW** para nível lógico baixo (0V) e **HIGH** para nível lógico alto (5V) de uma determinada porta, no nosso caso a porta 13 representada pela variável led. Veja que devemos definir o pino que queremos mexer para depois escolher o estado para o qual ele deve ir.

```
digitalWrite(led, HIGH); // Liga o LED (HIGH = nivel lógico alto)
```

O segundo é o *delay.* O comando delay cria uma pausa no algoritmo pelo tempo determinado em milissegundos (1 segundo é igual a 1000 milissegundos).

```
delay(1000);             // Espera um segundo
```

- *Ponto e virgula e chave*

Como já explicado, o microcontrolador do Arduino lê o código linha por linha, executando instrução por instrução. Dessa forma, quando escrevemos uma instrução, devemos informar onde ela começa e acaba. Para isso serve o ponto e vírgula (;).

```
digitalWrite(led, HIGH); // Liga o LED (HIGH = nivel lógico alto)
delay(1000);             // Espera um segundo
```

Veja o exemplo do delay(1000), essa instrução pede para que o Arduino pare por 1 segundo. Para que o Arduino leia a instrução, entenda e execute, é necessário que ele saiba onde ela começa e onde ela termina. Nesse caso, ela começa logo após o ponto e vírgula da linha anterior e acaba no ponto e vírgula da própria linha.

Para blocos de instruções usamos chaves, aberta "{" para começar e fechada "}" para terminar. Um exemplo de bloco de instruções são as funções.

```
void setup() {
    pinMode(led, OUTPUT);
}
```

Nesse exemplo o bloco de função setup começa no colchete aberto e termina no colchete fechado. Já a instrução pinMode começa no colchete aberto e termina no ponto e vírgula.

2.2. Experiência 2 – Usando a Protoboard

Essa experiência será semelhante à anterior com a diferença de que aprenderemos a usar um equipamento fundamental para quem quer realizar experiências com circuitos eletrônicos, a Protoboard.

2.2.1. A Protoboard (Matriz de Contatos)

Quando trabalhamos com circuitos elétricos em laboratório, muitas vezes usamos a protoboard. Ele tem o intuito de simplificar o processo de estudo e pesquisa de circuitos elétricos e eletrônicos. Sendo uma matriz de contatos reutilizável, evitamos a necessidade de confeccionar uma placa de circuito impresso e possibilita a fácil alteração do circuito, deixando ele flexível.

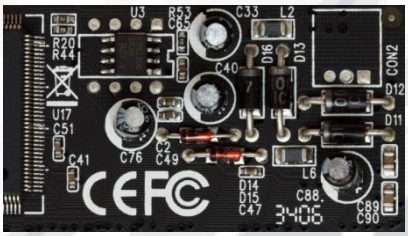


Figura 6 - Circuito impresso

Nas extremidades podemos notar dois barramentos de contatos paralelos ao formato da protoboard. No centro temos barramentos perpendiculares com um pequeno espaçamento no meio. A imagem a seguir mostra como está ordenado os barramentos.

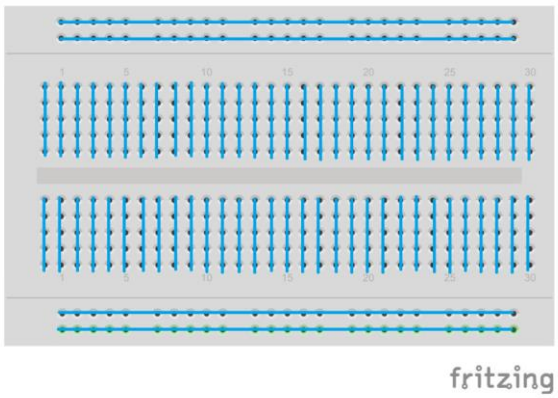


Figura 7 - Protoboard 400 furos

Usando o Arduino, continuamente teremos que montar circuitos com LED's, sensores, entre outros. Dessa forma, o uso de protoboards será comum no seu dia-a-dia como um desenvolvedor.

2.2.2. Separando os Ingredientes

Para essa experiência precisaremos dos seguintes componentes:

- 1 LED 5mm
- 1 Resistor 470 Ω
- Fios Jumper's
- 1 Protoboard

2.2.3. Misturando os Ingredientes

Agora vamos conectar os componentes do projeto. Para isso monte seu circuito conforme a figura a seguir.

Garanta que seu Arduino esteja desligado durante a montagem e que o seu LED esteja conectado corretamente, com a perna mais longa (Anodo) conectado ao resistor e a perna menor (catodo) ao GND.

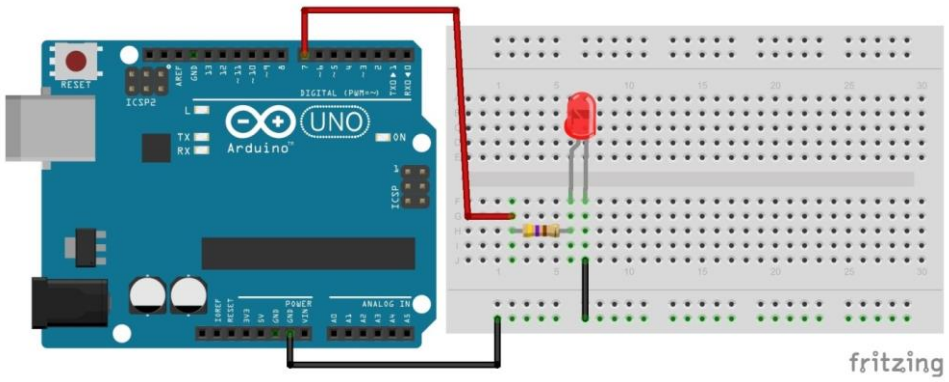


Figura 8 - Circuito da experiência 2

2.2.4. Levando ao forno

Agora temos que carregar o programa no Arduino. Caso você tenha carregado outro programa ou ainda não fez o exemplo 1 e seu Arduino ainda não esteja com o programa Blink, volte ao item 2.1.1 e faça o mesmo procedimento da experiência 1, já que usaremos o mesmo programa.

Depois que o Arduino estiver ligado com o programa carregado, o LED deve começar a piscar intermitentemente.

2.2.5. Entendendo o Hardware

Quando configuramos um pino como saída digital, tal como a porta 13 nessa experiência, ele pode fornecer 0 ou 5 V fazendo com que ele drene ou forneça corrente do circuito controlado. O valor máximo dessa corrente varia de placa para placa, mas, em geral, é de 30mA. Essa corrente é mais do que suficiente para ligar um LED de alto-brilho e alguns sensores, porém não é suficiente para ligar a maioria dos relés e motores. Caso

uma corrente maior que o limite, passe por um pino, este poderá ser danificado.

> Ao usar um pino como saída tenha cuidado para não drenar mais que o máximo suportado. Caso uma corrente maior que o limite passe por um pino, este poderá ser danificado. No Arduino esse valor costuma ser de 30mA.

Quando precisamos de correntes maiores usamos dispositivos que, a partir da tensão de saída do pino digital do Arduino, não chaveados para conduzir ou não conduzir a corrente vinda de uma fonte externa. Exemplo:

- ✓ Transistor (EXTRA);
- ✓ Relé;
- ✓ Ponte H (EXTRA)

- *Resistor*

Componente eletrônico que dificulta a passagem de corrente elétrica. Esta dificuldade de passagem de corrente é denominada resistência e é medida em ohms.

Cada componente e equipamento eletrônico possui uma corrente máxima de funcionamento, por isso, o uso de resistores é essencial. Ele terá o papel de limitar a corrente máxima do circuito.

Figura 9 - Resistor

- *LED's*

O LED (Diodo Emissor de Luz) é um componente que, quando uma corrente elétrica passa por ele em um determinado sentido, passa a emitir luz. Por isso o LED tem que estar no sentido correto no circuito para não ter riscos de queimá-lo.

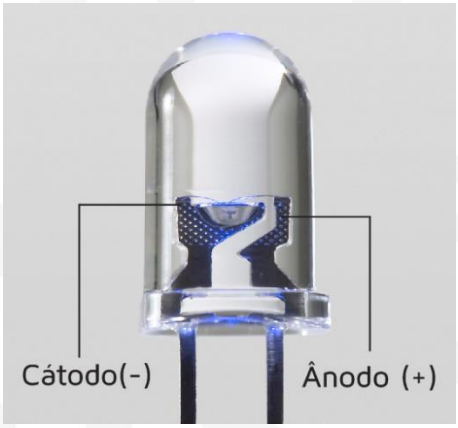
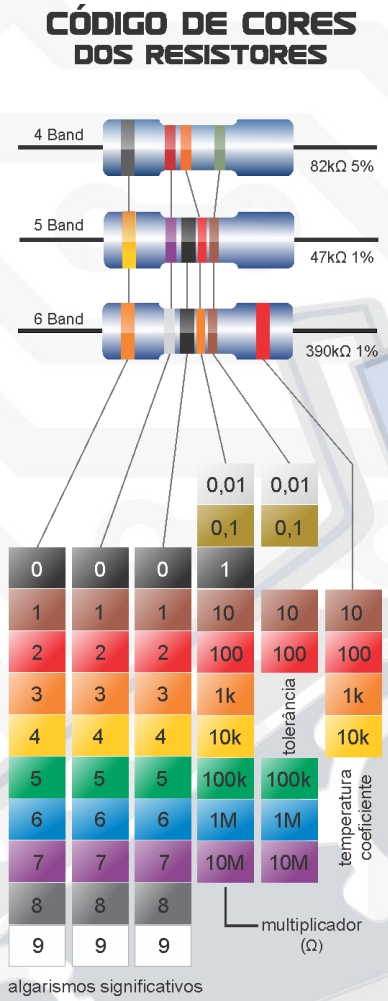


Figura 10 - LED

2.2.6. Código de cores dos resistores



3. Entradas Digitais

No capítulo anterior vimos como controlar uma saída digital e fizemos um LED piscar intermitentemente. No entanto, na vida real, um sistema interativo trabalha processando entradas e atuando através de saídas.

Nesse capitulo, iremos aprender como acender um LED (saída) a partir de um botão (entrada).

3.1. Experiência 3 – Leitura de botões

3.1.1. Ingredientes

- Botão de pressão
- Resistor 1kΩ
- 1 LED 5mm
- 1 Resistor 470 Ω
- Fios Jumper's
- Protoboard

3.1.2. Misturando os ingredientes

Agora vamos conectar os componentes do projeto. Para isso monte seu circuito conforme a figura a seguir.

Garanta que seu Arduino esteja desligado durante a montagem, e que o seu LED esteja conectado corretamente, com a perna mais longa (Anodo) conectado ao resistor e a perna menor (catodo) ao GND.

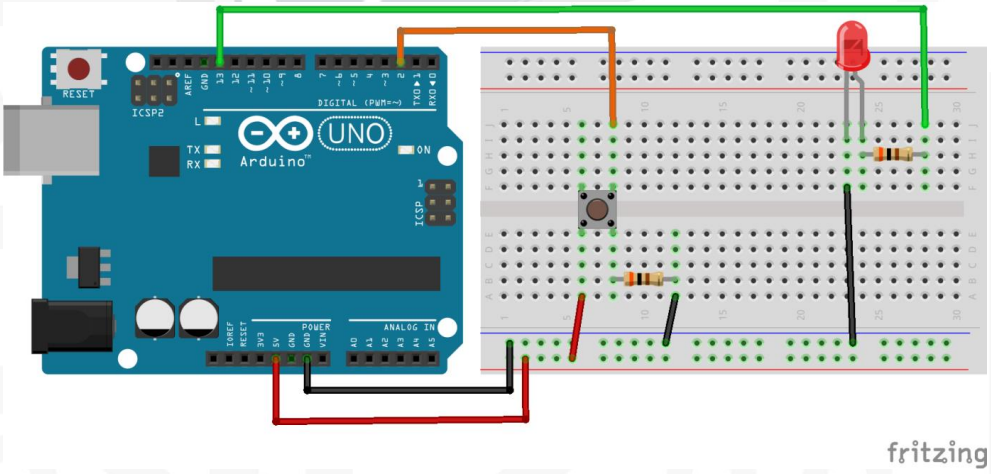


Figura 11 - Circuito da Experiência 3

3.1.3. Levando ao forno

Conecte seu Arduino ao computador e abra a IDE Arduino. No menu Tools, certifique que a porta serial (serial port) está selecionada e se a placa configurada é a que você está usando (board).

3.1.4. Preparando a cobertura

Agora temos que dar vida ao Arduino. Para isso, dentro da IDE Arduino: abra o menu File > Examples > 01. Basics > Button.

Uma nova janela deve ser aberta com o programa conforme é apresentado a seguir. Clique em Upload para que o programa seja transferido para seu Arduino.

```
/*
  Button

 Utiliza um botão de pressão, conectado ao pino 2, para ligar e
 desligar um LED conectado ao pino digital 13.

 */

// Determinamos constantes para os números dos pinos utilizados
const int buttonPin = 2;     // Numero do pino do botão de pressão
const int ledPin =  13;      // Numero do pino do led

// Variaveis
int buttonState = 0;         // Variável para leitura do estado do botão

// Executa uma vez ao ligar ou reiniciar a placa
void setup() {
    pinMode(ledPin, OUTPUT); //Inicializa o pino do LED como saída (OUTPUT)
    pinMode(buttonPin, INPUT); // Inicializa o pin do botão como entrada
(INPUT)
}

// Executa infinitamente quando liga a placa
void loop() {
  // Lê o estado do botao (HIGH -> +5V -> botão press.) (LOW -> 0V)
  buttonState = digitalRead(buttonPin);

  // Testa se o botão está pressionado
  // Se sim, o estado do botão e alto (HIGH)
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);    // Liga o LED
  }
  // Senão (Botao nao pressionado)
  else {
    digitalWrite(ledPin, LOW);    // Desliga o LED
  }
}
```

3.1.5. Experimentando o prato

Caso tenha ocorrido tudo como esperado, quando você apertar o botão acenderá o LED embutido na placa, ou o LED da protoboard caso tenha optado por manter o LED da experiência 2. Quando você soltar o botão, o LED apagará.

3.2. Entendendo o Hardware

Quando configuramos um pino como entrada digital ele apresentará uma característica chamada alta impedância. Isso significa que uma pequena corrente consegue fazer com que seu estado mude. Podemos usar essa configuração, por exemplo, para ler botões, fotodiodos entre outros. E a partir do estado lido ou das mudanças desses estados, concluir o que está acontecendo no mundo externo e então tomar ações baseadas nessas medidas.

> Caso o pino seja configurado como entrada mas não estiver conectado a nada, ele poderá alterar seu estado aleatoriamente por ser afetado pelo ruído elétrico do ambiente.

Para evitar esse problema, podemos utilizar um resistor de pull up ou pull down. Esses resistores farão com que a tensão em nossa entrada esteja bem definida quando o mesmo não estiver conectado a nada. As figuras mostram três ligações de um botão a um pino do microcontrolador configurado como entrada digital. Para cada caso temos:

- Entrada sem pull up ou pull down: O estado do pino não estará bem definido fazendo com que este esteja suscetível a ruído.

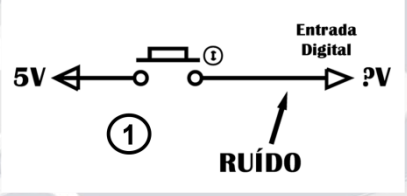


Figura 12 - Entrada sem pull-down ou pull-up

- Pull-down: Neste caso, devido à presença do resistor de pull-down, seu estado está bem definido e ele enxergará GND, consequentemente definindo seu estado como baixo.

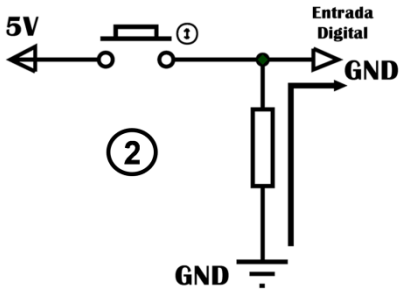


Figura 13 - Entrada com pull-down

- Pull-up: Neste caso, devido à presença do resistor de pull-up, seu estado está bem definido e ele enxergará +5 V, consequentemente definindo seu estado como alto.

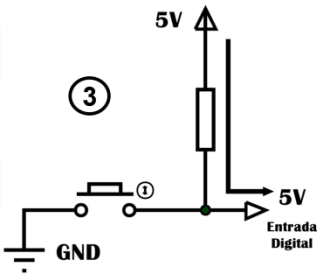


Figura 14 - Entrada com pull-up

Na experiência usamos um pul-down. Dessa forma, o pino 2 tem 0V normalmente. Só alterando quando alguém pressionar o botão, passando para 5V enquanto pressionado.

3.3. Entendendo o Programa

Boa parte desse programa é semelhante ao usado na experiência 1 e 2. Focaremos agora na parte não apresentada anteriormente.

- *Constantes*

Quando definimos uma variável que não irá mudar ao longo da execução do programa, definimos ela como uma constante.

Um exemplo de variável constante é o número dos pinos do Arduino que usaremos, já que eles não mudarão ao longo do programa.

```
// Determinamos constantes para os números dos pinos utilizados
const int buttonPin = 2;     // Numero do pino do botão de pressão
const int ledPin =  13;      // Numero do pino do led
```

- *Comando pinMode*

Na função setup podemos encontrar o comando pinMode. Como já dito na experiência 1, esse comando é muito importante no uso das entradas e saídas digitais do Arduino. Ele é responsável por definir se um pino é uma entrada (INPUT) ou uma saída (OUTPUT).

```
void setup() {
    pinMode(ledPin, OUTPUT); //Inicializa o pino do LED como saida (OUTPUT)
    pinMode(buttonPin, INPUT); // Inicializa o pin do botao como entrada
(INPUT)
```

Na primeira linha da função setup de nosso exemplo, temos o pino 13, definido pela constante ledpin, sendo configurado como saída digital.

```
pinMode(ledPin, OUTPUT); //Inicializa o pino do LED como saída (OUTPUT)
```

Na segunda linha temos o pino 2, definido pela constante buttonPin, sendo configurado como uma entrada digital.

```
pinMode(buttonPin, INPUT); // Inicializa o pin do botao como entrada (INPUT)
```

- *Comando digitalRead*

Na função Loop temos o programa que rodará ciclicamente em nosso Arduino. Como desejamos acender um LED a partir de um botão, primeiramente devemos ler o seu estado. Para isso, usamos o comando digitalRead, que retorna o valor de estado de uma entrada digital passada como parâmetro.

Nesse comando, caso a tensão na entrada digital seja 5V, ele retornará 1, caso seja 0V, retornará 0.

Para usar esse resultado devemos armazená-lo em algum lugar, nada melhor que uma variável para isso. No exemplo, armazenaremos o valor do botão na variável buttonState

No nosso caso, quando o botão estiver pressionado, teremos 5V na porta 2, ou seja, 1 na variável buttonState.

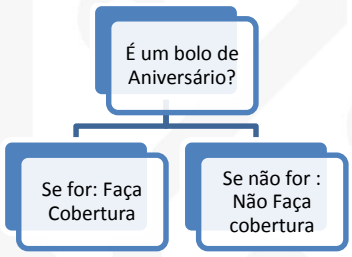
```
// Comando que lê o estado do botao (HIGH -> +5V -> botao press.) (LOW -> 0V)
```

- *Bloco IF*

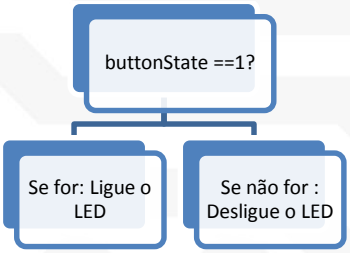
Ao resolver um problema, constantemente teremos que tomar decisões lógicas a partir dos resultados coletados. Vamos voltar à receita do bolo. Caso esse bolo seja para uma festa de aniversário, é interessante que se faça uma cobertura, caso contrário, podemos deixar o bolo sem cobertura. Veja como ficaria o algoritmo para essa situação:

```
Função bolo(){
  Faça o bolo;
  Se o bolo é de aniversário {
       Faça uma cobertura;
  }Senão
       Não faça uma cobertura
  }
fim
```

Veja que usamos um parâmetro para decidir se iremos ou não executar uma atividade. Veja o fluxograma a seguir.



Dessa forma, em nosso exemplo, caso buttonState seja igual a 1 (botão pressionado), devemos acender o LED, caso contrário, devemos apaga-lo.



No programa fica assim:

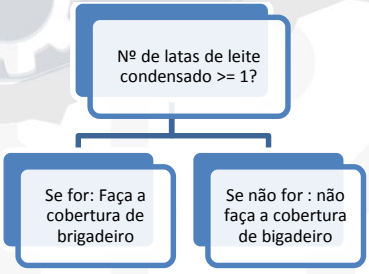
```
// Testa se o botão está pressionado
// Se sim, o estado do botão e alto (HIGH)
if (buttonState == HIGH) {
  digitalWrite(ledPin, HIGH);    // Liga o LED
}
// Senão (Botao nao pressionado)
else {
  digitalWrite(ledPin, LOW);    // Desliga o LED
}
```

- *Operadores Lógicos*

Para usar blocos lógicos, tal como o bloco IF, usaremos operadores lógicos. Os operadores lógicos, junto aos parâmetros, são responsáveis por criar as condições de execução de um bloco lógico.

Voltando ao bolo, caso você não tenha leite condensado, será inviável fazer uma cobertura de brigadeiro. Dessa forma, o número de latas de leite condensado disponíveis deve ser maior ou igual a 1 latas para que seja viável fazer essa cobertura.

Vejamos o fluxograma resultante:



A condição maior ou igual é um operador lógico, veja os possíveis operadores lógicos na tabela a seguir.

Operadores de Comparação	
==	Igual
!=	Diferente
<	Menor
>	Maior
<=	Menor e igual
>=	Maior e igual

4. Entrada Analógica

O uso de entradas digitais limita nossas possibilidades, visto que o mundo é analógico. Sendo assim, a leitura de valores analógicos muitas vezes se faz necessário.

Nesse capítulo, aprenderemos como podemos usar as entradas analógicas das placas Arduino.

4.1. Experiência 4 – Lendo uma entrada analógica

4.1.1. Ingredientes

- Potenciômetro
- Fios Jumper's
- Protoboard

4.1.2. Misturando os ingredientes

Agora vamos conectar os componentes do projeto. Para isso, monte seu circuito conforme a figura a seguir e garanta que seu Arduino esteja desligado durante a montagem.

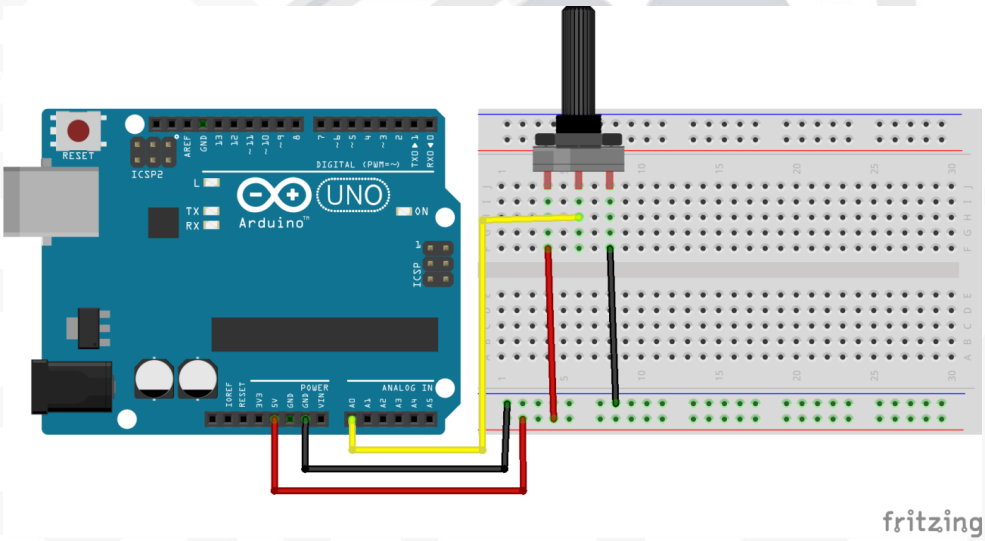


Figura 15 - Circuito da experiência 4

4.1.3. Levando ao Forno

Conecte seu Arduino ao computador e abra a IDE Arduino. No menu Tools, certifique de que a porta serial (serial port) está selecionada e que a placa configurada é a que você está usando (board).

4.1.4. Preparando a cobertura

Agora vamos à implementação do programa. Nesse algoritmo iremos fazer diferente. Você terá que escrever o programa tal como escrito abaixo, isso é fundamental

para que você acostume com a linguagem e a estrutura da programação do Arduino.

Para isso, dentro da IDE Arduino: abra o menu File > New ou aperte “ctrl+N”. Uma nova janela, em branco deve ser aberta.

Agora devemos dar um nome ao nosso sketch (esse é o nome dado aos programas no Arduino), mas podemos chamar de arquivo de programa ou código. Dessa forma, dentro da IDE Arduino: Abra o menu File e clique em Save. Uma nova janela será aberta onde você escolherá onde salvar seu programa e qual será seu nome. Para facilitar a identificação, dê o nome de “programa_entradaanologica”.

Com o seu programa salvo, escreva nele o código conforme escrito abaixo.

```
// Esta função "setup" roda uma vez quando a placa e ligada ou resetada
unsigned int valorLido;

void setup() {
  Serial.begin(9600); //Inicia porta serial e define a velocidade de transmissão
}

// Função que se repete infinitamente quando a placa é ligada
void loop() {
 valorLido = analogRead(A0);
 Serial.println(valorLido);
 delay(1000); // Espera um segundo
}
```

Depois de escrever o código, Clique em Upload para que o programa seja transferido para seu Arduino.

4.1.5. Experimentando o prato

Para essa experiência, você terá que abrir o serial Monitor que será melhor explicado mais à frente. Para isso, o serial monitor pode ser aberto pelo ícone mostrado na figura a seguir.

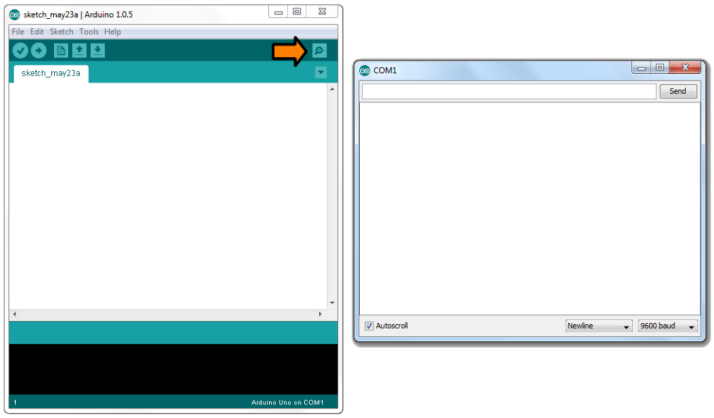


Figura 16 - Monitor serial

Caso tenha ocorrido tudo como esperado, conforme você varie a resistência do

potenciômetro, girando seu eixo, aparecerá no serial monitor um valor de 0 à 1023.

4.2. Entendendo o Hardware

- *Potenciômetro e trimpot*

Potenciômetros e trimpot são resistores variáveis e ajustáveis, e por isso são usados para controle analógico de funcionalidades de alguns aparelhos eletrônicos, tal como o volume de um aparelho de som.

Eles costumam possuir três pernas. Dois são ligados às extremidades da resistência (A e B) e a terceira a um cursor que anda de ponta a ponta da resistência (C).Dessa forma, podemos usar o resistor entre A e C ou B e C.

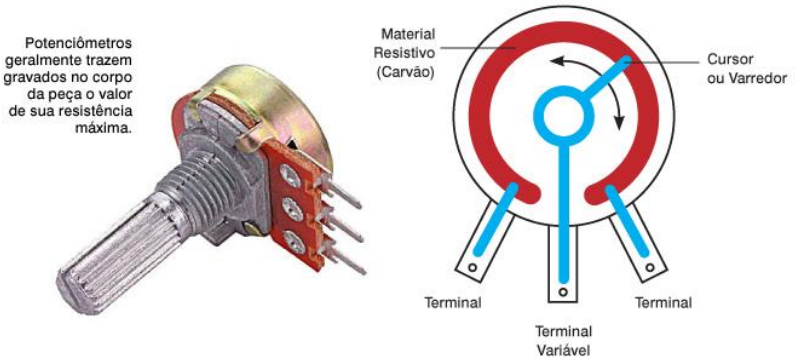


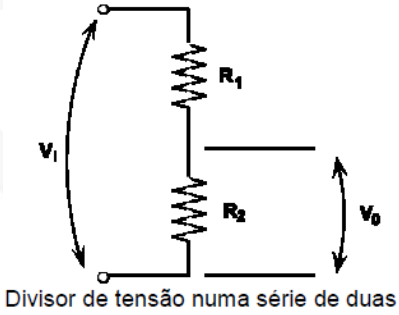
Figura 17 - Potenciômetro

- *Divisor de tensão*

Quando temos n resistência associadas em série temos o nome de divisor de tensão. Em um circuito divisor de tensão, temos uma queda de tensão em cada resistência igual ao produto da resistência com a corrente do circuito.

Como a corrente do circuito é calculada pela divisão da tensão em cima de todos os resistores dividido pela soma dos resistores, teremos a tensão em cima de um resistor igual a resistência desse resistor vezes a tensão total dividida pela soma dos resistores.

O exemplo a seguir mostra como funciona o cálculo o para dois resistores.



$$V_0 = \frac{R_2}{R_1 + R_2} \times V_i$$

Figura 18 - Divisor de tensão

Quando usamos um potenciômetro, podemos usar a propriedade do divisor de tensão. Sabemos que a tensão total e a resistência total são fixas. Dessa forma, o divisor de tensão vai variar com a resistência A0 e GND.

4.3. Entendendo o programa

4.3.1. Lendo da Entrada Analógica

A leitura da entrada analógica é feita com a função analogRead, que recebe como parâmetro o pino analógico a ser lido e retorna o valor digital que representa a tensão no pino. Como o conversor analógico-digital do Arduino possui uma resolução de 10 bits, o intervalo de tensão de referência, que no nosso caso é 5 V, será dividido em 1024 pedaços (2^10) e o valor retornado pela função será o valor discreto mais próximo da tensão no pino.

```
unsigned int valorLido = analogRead(A0);
```

O código acima lê o valor analógico de tensão no pino A0 e guarda o valor digital na variável valorLido. Supondo que o pino está com uma tensão de 2V, o valor retornado pela conversão será:

$$2 \times 1024 / 5 = 409,6$$

O resultado deve ser inteiro para que nosso conversor consiga representá-lo, logo o valor 410 será escolhido por ser o degrau mais próximo. Esse valor representa a tensão 2,001953125, inserindo um erro de 0,001953125 em nossa medida devido a limitação de nossa resolução.

> O uso do comando unsigned na declaração da variável informa que usaremos apenas números positivos para essa variável.

4.3.2. Comunicação Serial

A comunicação serial é amplamente utilizada para comunicar o Arduino com outros dispositivos como módulos ZigBee, Bluetooth, entre outros. A comunicação com o computador é possível através do conversor serial USB presente nas placas. A biblioteca padrão do Arduino possui algumas funcionalidades para a comunicação serial de modo a facilitar a utilização desta função.

- *Serial Monitor*

A possibilidade de visualizar na tela do computador os dados coletados e processados pelo Arduino é fundamental, visto que é um forma rápida e prática de visualizar esses dados. Para isso, usamos o Serial Monitor.

O Serial Monitor disponibiliza uma interface gráfica que facilita a comunicação entre o Arduino e um computador. Após selecionarmos a porta serial adequada, o serial monitor pode ser aberto pelo ícone no canto superior direito da janela, onde é representado por uma lupa.

```
Serial.begin(9600); //Inicia porta serial e define a velocidade de transmissão
```

- *Enviando Dados*

Na maioria das vezes, o envio de dados pela porta serial pode ser feito através das funções print e println. A função print imprime os dados na serial utilizando o código ASCII. Essa função recebe dois parâmetros e retorna a quantidade de bytes que foram escritos.

```
Serial.print(var); // Imprime a variável var
Serial.print("olá");// Imprime o texto olá
```

O parâmetro "var" recebe o valor a ser impresso. A função é sobrecarregada de modo a ser capaz de receber qualquer tipo padrão (char, int, long, float etc).

Quando o texto estiver entre aspas, isso simboliza que você deseja imprimir um texto. Caso contrário, você deseja imprimir uma variável.

A função println imprime o valor desejado semelhantemente a função print, porém imprime na sequência os caracteres '\r' e '\n' de modo que crie uma nova linha.

5. PWM

PWM (Pulse Width Modulation – Modulação por Largura de Pulso) é uma técnica para obter resultados analógicos por meios digitais. Essa técnica consiste na geração de uma onda quadrada em uma frequência muita alta em que pode ser controlada a porcentagem do tempo em que a onda permanece em nível lógico alto. Esse tempo é chamado de Duty Cycle e sua alteração provoca mudança no valor médio da onda, indo desde 0V (0% de Duty Cycle) a 5V (100% de Duty Cycle) no caso do Arduino.

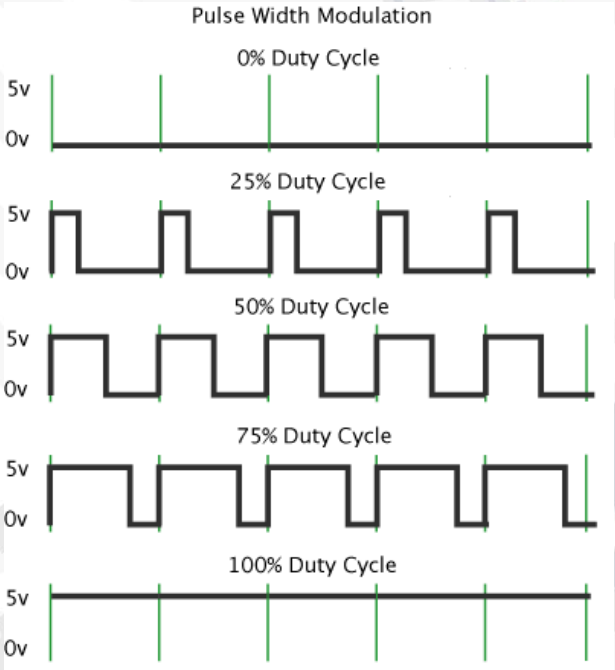


Figura 19 - Sinal PWM

O duty cycle é a razão do tempo em que o sinal permanece na tensão máxima (5V no Arduino) sobre o tempo total de oscilação, como está ilustrado na figura abaixo:

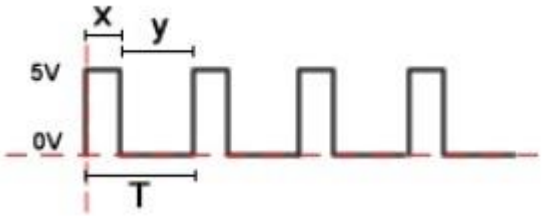


Figura 20 - Duty cycle

$$\text{Duty Cycle (\%)} = (x/x+y)*100\% = (x/T)*100\%$$

$$V_{médio} = V_{max} * \text{Duty Cycle(\%)}$$

O valor do Duty Cycle usado pelo Arduino é um inteiro armazenado em 8 bits, de forma que seu valor vai de 0 (0%) a 255 (100%).

5.1. Usando a saída PWM

Para escrever um sinal na saída PWM utiliza-se a função analogWrite, que recebe como parâmetros o pino PWM e o valor do duty cycle, respectivamente. Esse último parâmetro é guardado em 8 bits, de modo que esse valor deve estar entre 0 (0% de duty cycle) e 255 (100% de duty cycle).

```
analogWrite(9,127);//Escreve no pino 9 um sinal PWM com 50% de duty cycle
//(50% de 255=127)

analogWrite(10,64);//Escreve no pino 10 um sinal PWM com 25% de duty cycle
//(25% de 255=64)
```

Variando o duty cycle altera-se também o valor médio da onda, de modo que o efeito prático obtido com o PWM em algumas aplicações é um sinal com amplitude constante e de valor igual ao valor médio da onda. Isso permite que se possa, por exemplo, controlar a intensidade do brilho de um LED, ou a velocidade de um motor de corrente contínua.

5.2. Experiência 5 – Led com controle de intensidade

5.2.1. Ingredientes

- Potenciômetro
- LED 5mm
- Resistor 470Ω
- Fios Jumper's
- Protoboard

5.2.2. Misturando os ingredientes

Agora vamos conectar os componentes do projeto. Para isso, monte seu circuito conforme a figura a seguir.

Garanta que seu Arduino esteja desligado durante a montagem e que o seu LED esteja conectado corretamente, com a perna mais longa (Anodo) conectado ao resistor e a perna menor (catodo) ao GND.

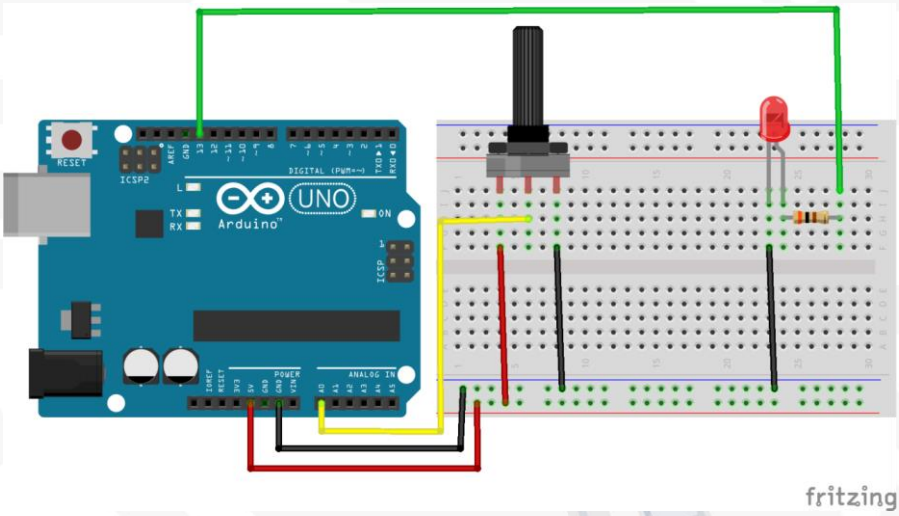


Figura 21 - Circuito da experiência 5

5.2.3. Levando ao forno

Conecte seu Arduino ao computador e abra a IDE Arduino. No menu Tools, certifique que a porta serial (serial port) está selecionada e se a placa configurada é a que você está usando (board).

5.2.4. Preparando a cobertura

Crie um programa (sketch) e salve com o nome de "programa_pwm". Com o seu programa salvo, escreva nele o código conforme escrito abaixo.

```
// Daremos um nome ao pino que está conectado o LED
int led = 13;

unsigned int valorLido;
unsigned int pwm;

// Esta função "setup" roda uma vez quando a placa e ligada ou resetada
void setup() {
  pinMode(led, OUTPUT); // Configura o pino do led (digital) como saída
}

// Função que se repete infinitamente quando a placa é ligada
void loop() {
 valorLido = analogRead(A0); // valor entre 0 e 1024
 pwm = map(valorLido, 0, 1023, 0, 255); // Mudança de escala
 analogWrite(led,pwm);//Escreve no led um sinal PWM proporcional ao valorLido
}
```

Depois de escrever o código, Clique em Upload para que o programa seja transferido para seu Arduino.

5.2.5. Experimentando o prato

Se tudo der certo, conforme girarmos o potenciômetro, a intensidade de luz emitida pelo LED diminuirá ou aumentará.

5.3. Entendendo o programa

Como já explicado no início do capítulo, o PWM pode ser usado para simular uma saída analógica. Variando um sinal de saída de PWM de 0 a 255, estamos variando o Duty Cycle, que por sua vez, resulta numa saída de 0V a 5V.

Como a intensidade de luz no LED está diretamente ligada à quantidade de corrente que passa por ele e essa corrente é proporcional a tensão do resistor em série com o LED, conforme variamos a tensão do pino 13 através do PWM, alteramos a intensidade de luz emitida pelo LED.

```
valorLido = analogRead(A0); // valor entre 0 e 1024
```

Para variarmos o PWM, usamos o valor analógico lido no potenciômetro e armazenamos na variável valorLido, que armazena valores entre 0 e 1023. Porém, para que seja possível usar essa variável para controlar o PWM, devemos mudar sua escala para 0 a 255. Para isso usaremos a função. Tal função recebe uma variável e muda sua escala.

```
pwm = map(valorLido, 0, 1023, 0, 255); // Mudança de escala
```

Depois de mudarmos de escala, basta escrevermos o valor de PWM na saída do LED.

```
analogWrite(led,pwm);//Escreve no led um sinal PWM proporcional ao valorLido
```

6. [EXTRA] Interrupção

Imagine que você esteja fazendo seu bolo e no meio da receita seu telefone toque. Possivelmente você irá parar o que está fazendo e irá atender o telefone, assim que encerrar a chamada você irá retornar ao ponto que parou em sua receita.

Quando estamos executando um tarefa muitas vezes temos que a interromper para resolver outra tarefa importante para só depois retornar do ponto que se parou. Isso se chama interrupção e é usada com freqüência na programação de microcontroladores.

Uma interrupção tem dois pontos chaves, são eles:

- *Condição de interrupção*

É a condição que indica uma interrupção. Ela avisa ao programa que é a hora de executar uma tarefa extraordinária. No nosso exemplo, essa condição é o toque do telefone.

- *Função a ser executada*

Quando algo indica a interrupção, temos que executar uma lista de instruções referentes a essa interrupção. No exemplo dado, temos que parar de fazer o bolo e ir atender ao telefone. A função atender telefone é uma função extraordinária que só é executada pelo fato de ter ocorrido a condição de interrupção, o toque do telefone.

Para aprender como implementar uma interrupção, vamos fazer a experiência 4. Nela você poderá entender melhor esse conceito de interrupção em um microcontrolador.

6.1. Experiência 6 – Implementando uma interrupção

6.1.1. Ingredientes

- Botão de pressão
- LED 5mm
- Resistor 1kΩ
- Resistor 470Ω
- Fios Jumper’s
- Protoboard

6.1.2. Misturando os ingredientes

Agora vamos conectar os componentes do projeto. Para isso, monte seu circuito conforme a figura a seguir.

Garanta que seu Arduino esteja desligado durante a montagem e que o seu LED esteja conectado corretamente, com a perna mais longa (Anodo) conectado ao resistor e

a perna menor (catodo) ao GND.

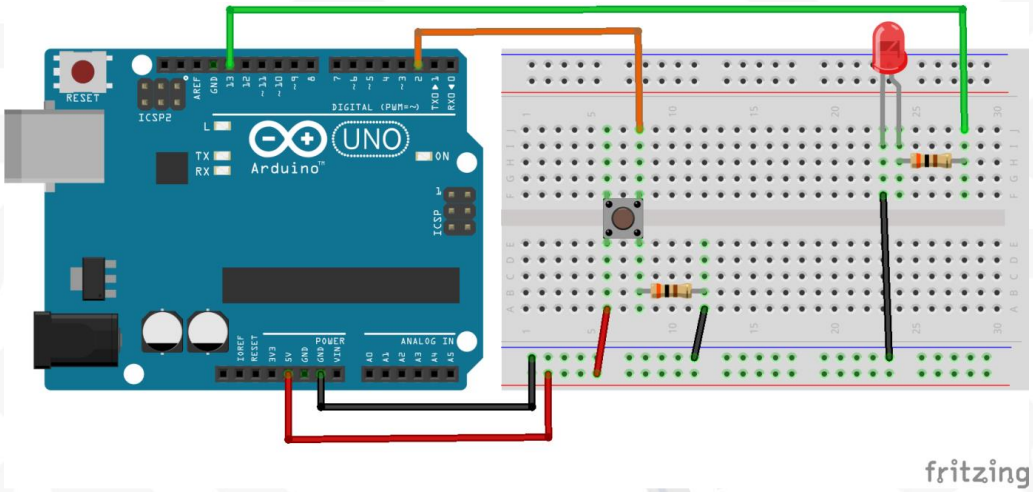


Figura 22 - Circuito da experiência 6

6.1.3. Levando ao forno

Conecte seu Arduino ao computador e abra a IDE Arduino. No menu Tools, certifique-se que a porta serial (serial port) está selecionada e que a placa configurada é a que você está usando (board).

6.1.4. Preparando a cobertura

Crie um programa (sketch) e salve com o nome de “programa_interrupcao”.

Com o seu programa salvo, escreva nele o código conforme escrito abaixo.

```
// Existe um LED conectado no pino 13 da maioria dos Arduinos
// Daremos um nome a este pino:
int led = 13;

void interrupção(){
  digitalWrite(led, HIGH); // Liga o LED (HIGH = nível lógico alto)
  delay(5000);
}

// Esta função "setup" roda uma vez quando a placa e ligada ou resetada
void setup() {
  pinMode(led, OUTPUT); // Configura o pino do led (digital) como saída
  attachInterrupt(0,interrupcao,RISING); //Configurando a interrupção
}

// Função que se repete infinitamente quando a placa é ligada
void loop() {
  digitalWrite(led, HIGH); // Liga o LED (HIGH = nível lógico alto)
  delay(1000);             // Espera um segundo
  digitalWrite(led, LOW);  // Desliga o LED (LOW = nível lógico baixo)
  delay(1000);             // Espera um segundo
}
```

Depois de escrever o código, clique em Upload para que o programa seja

transferido para seu Arduino.

6.1.5. Experimentando o prato

Caso tenha ocorrido tudo como esperado, o LED deve piscar intermitentemente. Quando você apertar o botão, o LED da protoboard permanecerá aceso por 5 segundos. Caso você não pressione o botão novamente, ele voltará a piscar.

6.2. Entendendo o Hardware

As placas Arduino possuem pinos que podem desempenhar a função de entrada de sinal para interrupção externa. No Arduino UNO são as portas digitais 2 e 3, que para tal função são nomeadas de INT0 e INT1, respectivamente. Veja a tabela a seguir com os pinos de cada placa Arduino que possuem essa qualidade.

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

Dessa forma, para que seja possível o uso da interrupção externa, escolhemos o pino digital 2 (INT0), no qual conectamos o botão.

6.3. Entendendo o programa

Com o conhecimento adquirido até agora, você já pode entender a maioria dos programas. Dessa forma, iremos nos ater as novidades.

- *Configurando a interrupção*

Para que o Arduino leia uma interrupção, devemos configurá-lo. Para tal usaremos o comando attachInterrupt().

```
attachInterrupt(INT,FUNÇÃO,MODO); //Configurando a interrupção
```

Como explicado anteriormente, numa interrupção temos dois pontos chaves: a condição da interrupção e a função que será executada. Dessa forma, o comando attachInterrupt é usado para informar ao programa esses dados. São eles:

INT: Número da porta usada para a interrupção. No Arduino UNO INT 0 corresponde à porta digital 2 e INT 1 corresponde à porta digital 3;

FUNÇÃO: Nome da função que será chamada quando ocorre a interrupção;

MODO: Define em qual tipo de variação do sinal a interrupção será disparada. As opções são:

- LOW: Dispara a interrupção quando a tensão no pino está em 0V
- CHANGE: Dispara sempre que o sinal no pino muda de estado, borda 0V (0) para 5V(1) ou vice-versa;
- RISING: Dispara somente borda de subida, 0v (0) para 5V (1);
- FALLING: Dispara somente borda de descida, 5V (1) para 0V (0)

Em nosso programa, usaremos esse comando da seguinte forma:

```
attachInterrupt(0,interrupcao,RISING); //Configurando a interrupção
```

Portanto, temos como condição de interrupção a mudança de estado de 0V (0) para 5V(1) no pino digital 2 (INT 0) e a função a ser executa se chama interrupção.

- *Função interrupcao()*

Como já explicado no capítulo 2, função é um bloco de tarefas a serem executadas pelo programa quando solicitada.

```
void interrupção(){ //Função executada quando ocorre a interrupção externa
  digitalWrite(led, HIGH); // Liga o LED (HIGH = nível lógico alto)
  delay(5000);
}
```

No nosso caso, a função será solicitada quando ocorrer a interrupção. As tarefas a serem executadas serão: acender o LED e esperar 5 segundos.

7. Apêndice - Tabela de consulta

Funções Matemáticas e de tempo	
delay(t)	O programa tem uma pausa de t milissegundos
delayMicroseconds(t)	O programa tem uma pausa de t microssegundos
millis()	Retorna o tempo, em milissegundos, desde que o programa começou a rodar
randomSeed(referência)	Gera uma referência para o primeiro número aleatório (Função setup)
random(min,max)	Gera um valor pseudo aleatório int entre min e max (a função acima é necessária)
abs(x)	Retorna o módulo (valor absoluto) do número real passado como parâmetro
map(valor,min1,max1,min1 ,max2)	Converte um valor inserido em uma faixa de valores para um proporcional em uma nova faixa de valores. Mudança de range.
sin(x)	Retorna o seno de x(rad)

Entradas Analógicas	
anologRead(Pino)	Lê entrada analógica 0-5V transformando em 10 bit's (resolução 4,9mV)
Pinos analógicos podem ser usados como porta digitais usando a função pinMode(), quando usado como porta analógica não necessitam de configuração.	

Saídas/entradas Digitais e PWM	
pinMode(porta,Tipo)	Define se a porta será uma entrada (TIPO=INPUT) ou uma saída (TIPO= OUTPUT).
digitalWriter (pino, VL)	Coloca 0V (VL =LOW) ou 5V(VL = HIGH) na saída.
digitalRead(pino)	Lê o sinal digital no pino citado.
analogWrite(pino, x)	Saída PWM 500Hz (0 <= x <=255).
analogWrite(pino, x)	Saída PWM 500Hz (0 <= x <=255).
tone(pino,frequência,duração)	Gera uma frequência no pino durante um determinado tempo.
tone(pino,frequência)	Gera uma frequência no pino até que ocorra um comando de mudança de Freq.
noTone(pino)	Cessa a geração do tom no pino.
pulseIn(pino,valor,espera)	Mede a largura em microssegundo de um pulso no pino digital, "valor" é o tipo de pulso a ser medido (LOW ou HIGH), espera (opcional) faz com que a medida do pulso só comece após o tempo em microssegundos especificado.
attachInterrupt(pino,função, modo)	É uma interrupção, ou seja, caso a condição "modo" ocorra no pino especificado a função é executada imediatamente.

LOW	Dispara a interrupção quando o pino está em 0
CHANGE	Dispara sempre q o pino muda de estado (borda 0-> 1 ou vice-versa)
RISING	Somente borda de subida (0 para 1)
FALLING	Somente borda de descida (1 para 0)

Variáveis		
Tipo de dado	**RAM**	**Intervalo numérico**
boolean	1 byte	0 a 1 (false ou true)
int	2 bytes	-32.768 a 32.767
unsigned int	2 bytes	0 a 65.535
Word	2 bytes	0 a 65.535
Char	1 byte	-128 a 127
usigned char	1 byte	0 a 255
Byte	1 byte	0 a 255
void keyword	N/A	N/A
Long	4 bytes	-2.147.483.648 a 2.147.483.647
Usigned long	4 byte	0 a 4.294.967.295
float	4 byte	-3,4028235e+38 a 3,4028235e+38
Double	4 byte	-3,4028235e+38 a 3,4028235e+38
string	1 byte + x	Sequência de caracteres
array (vetor)	1byte + x	Sequência de variáveis
tipovar nomeMatriz[nº de posições]		

Comunicação Serial	
Serial.begin(TAXA)	Habilita a porta serial e fixa a taxa de transmissão (função setup)
Serial.end()	Desabilita a porta serial para permitir o uso dos pinos digitais
Serial.flush()	Libera caracteres que estão na linha serial, deixando-a vazia e pronta para entradas e saídas.
Serial.available()	Retorna o número de bytes disponíveis para leitura no buffer da porta serial.
Serial.read()	Lê o primeiro byte que está no buffer da porta serial
Serial.print('valor',formato)	Envia para a porta serial um caractere ASCII
Serial.println('valor',formato)	O mesmo que o anterior, porem pula uma linha

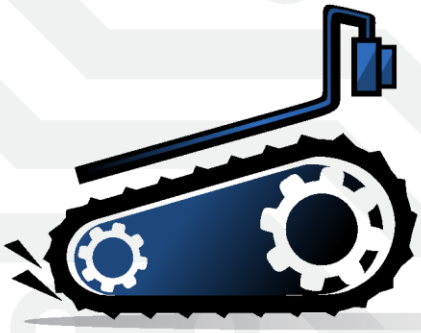
Operadores de Comparação	
==	Igual
!=	Diferente
<	Menor
>	Maior
>=	Maior ou igual
<=	Menor ou igual

Operadores Lógicos	
&&	AND
||	OR
!	NOT

Símbolos Compostos	
x++	Incrementa x
x--	Decrementa x
x+=y	x = x+y
x-=y	x = x-y

x*=y	x = x*y
x/=y	x = x/y
Símbolos	
{}	Entre as chaves fica o conteúdo da função
;	Final de um comando/linha
//	Linha de comentário
/* ... */	Comentário de varias linhas

Funções	
If(condição) **{}** **else** **{}**	Função Se e Se não
if(condição) **{}** **else if(condição 2)** **{}**	Função Se em cascata
switch(expressão){ **case expressão = x: Bloco1;** **break;** **case expressão = y: Bloco2;** **break;** **default: bloco3** **}**	Função Caso
while(condição){bloco funções}	Função Enquanto
do{ **bloco de instruções** **}** **while(condição);**	Função Enquanto, ela é executada pelo menos uma vez.
for(var;condição;incremento) **{}**	Função Para
(condição) ? bloco1:bloco2;	Operador ternário '?' caso condição seja verdadeira ele executa o bloco 1, caso contrario, executa o bloco 2. Ex.: y = (x >10)? 15:20; // caso x>10 y=15, caso contrario, y = 20



Vida de Silício

Robótica e Sistemas Digitais

Já parou pensar o que aconteceria se todos soubessem como criar robôs? Todos compartilhando seus conhecimentos e ajudando outras pessoas em seus projetos, projetos inovadores sendo criados a todo instante. Tudo isso sendo feito por um único propósito, mudar o mundo.

Essa revolução já vem acontecendo e a internet tem unido as pessoas de forma nunca vista antes. O conhecimento nunca foi tão acessível como tem sido agora e a pessoas têm cada vez mais aderido à filosofia do Open Source.

O Vida de Silício quer fazer parte de tudo isso ajudando as outras pessoas a construírem um mundo melhor. Tem muita coisa para ser criada ou melhorada, como por exemplo, a robótica de reabilitação, área que ainda tem muito para evoluir.

Para isso, montamos uma loja onde oferecemos a maior variedade possível de itens para confecção de projetos de robótica, eletrônica e automação, com preços amigáveis.

Além disso, temos um blog com tutoriais bacanas e didáticos para que todos possam aprender a usar um Arduino e outras placas de desenvolvimento, de forma simples e prática.

Conheça nosso site, ficaremos felizes em lhe receber!

Atenciosamente,
Equipe Vida de Silício

ARDUINO
DISTRIBUIDOR
OFICIAL

vidadesilicio.com.br

blog.vidadesilicio.com.br

contato@vidadesilicio.com.br

#vidadesilicio

fb.com/vidadesilicio