Eletrogate

Componentes Eletrônicos

# Apostila Arduino Básico V1.0

www.eletrogate.com

# Sumário

## Prefácio

Esta apostila é gratuita, e é destinada a qualquer pessoa que deseja aprender mais sobre o Arduino e Eletrônica Básica. Esperamos que goste do nosso trabalho!



## Download

Esta postila está pode sofrer atualizações e correções, para obter a ultima versão gratuitamente, acesse: http://apostilas.eletrogate.com

Caso tenha alguma sugestão, crítica ou apenas gostaria de dizer se gostou ou não deste material, entre em contato conosco! Será um imenso prazer receber a sua opinião sobre o nosso trabalho. E caso tenha alguma dúvida, conte com a gente!

## Sobre o Autor:

Nome: Anwar S Hachouche

Contato: anwar.hachouche@gmail.com

Facebook: /anwarhachouche

# 1. Introdução

## 1.1. O que é Arduino

O Arduino consiste-se em uma plataforma de prototipagem em eletrônica, elaborado por Massimo Banzi e David Cuartielles em 2005 na Itália, e tem como objetivo facilitar o desenvolvimento de projetos, desde os mais simples aos mais complexos. Com esta plataforma é possível controlar diversos sensores, motores, *leds*, dentre vários outros componentes eletrônicos.

Um ponto forte sobre o Arduino, é que todo material disponibilizado pelo fabricante, como a IDE de desenvolvimento, bibliotecas e até mesmo o projeto eletrônico das placas são *open-source*, ou seja, é permitida a utilização e reprodução sem restrição sobre os direitos autorais dos idealizadores do projeto. Porém o nome Arduino, logotipo e o design gráfico de suas placas são registrados e protegidos por direitos autorais. Saiba mais acessando página oficial do fabricante[1].

O Projeto Arduino une Hardware e Software, e resulta em uma plataforma de fácil desenvolvimento utilizando um microcontrolador.

[1] Página de Direitos Autorais: http://arduino.cc/en/Trademark/HomePage

## 1.2. Por que usar o Arduino

O Arduino foi criado com o objetivo de facilitar o aprendizado e possibilitar a prototipação e desenvolvimento de projetos com um custo relativamente baixo, além de não exigir um vasto conhecimento em eletrônica. Estes foram sem dúvida os fatores primordiais para a popularização do Arduino em âmbito mundial, não somente entre os desenvolvedores mais experientes, mas também entre os entusiastas e iniciantes.

Outro ponto forte do Arduino, como dito anteriormente, é seguir a filosofia *open-*

*source*. Com isso várias pessoas em todo o mundo contribuem com a plataforma, seja com a construção de um novo hardware ou com novas bibliotecas, materiais de apoio e tutoriais.

A comunidade de pessoas que utilizam esta plataforma vem crescendo cada vez mais, e sempre surgem novos projetos contendo novas ideias, fazendo com que nunca falte ajuda caso haja alguma dúvida sobre a utilização do hardware e/ou software.

### 1.3. Afinal, pra que serve o Arduino?

O Arduino é sem dúvidas uma placa muito eficiente e poderosa. Pode ser utilizado para fazer qualquer coisa, a imaginação é o limite (e claro, as leis da física também!). É possível utilizá-lo para controlar, monitorar, automatizar, etc. Por exemplo, existem projetos de monitoramento da qualidade do ar, medição da temperatura de um líquido, sistemas de irrigação, robôs, impressoras 3D, dentre vários outros.

## 2. Hardware

O hardware (as placas) do projeto possui diferentes modelos, alguns deles são menores que um cartão de crédito. Sim, tudo isso falado anteriormente cabe na palma de sua mão! Para o desenvolvimento deste material utilizaremos o modelo Uno, que é mais comumente utilizado em projetos básicos. Existe uma placa voltada para cada projeto, algumas são menores e mais compactas e outras são maiores, permitindo controlar um maior número de dispositivos eletrônicos. É importante lembrar que o preço também varia, por exemplo, o Arduino Uno custa em torno de R$ 60,00 a R$ 80,00 reais no Brasil.


Figura 1.1 – Arduino Uno R3

O Arduino é um computador como qualquer outro, possuindo:

- Microprocessador (responsável pelos cálculos e tomada de decisão)
- Memória ram (utilizada para guardar dados e instruções, volátil)
- Memória flash (utilizada para guardar o *sotware*, não volátil)
- Temporizadores (*timers*)
- Contadores
- *Clock*, e etc.

Ou seja, é um computador, porém em menor escala. Possui inclusive menos memória e menor poder de processamento. O Arduino Uno, por exemplo, possui as seguintes especificações:

- Microcontrolador: ATmega328
- Portas Digitais: 14
- Portas Analógicas: 6
- Memória Flash: 32KB (0,5KB usado no bootloader[2])
- SRAM: 2KB
- EEPROM: 1KB
- Velocidade do *Clock*: 16MHz

[2]Bootloader: Para dispensar o uso de um gravador externo, a gravação da Flash é feita por um software pré-gravado, o *Bootloader*. O *Bootloader* é o primeiro software executado pelo microcontrolador após um Reset (*Boot*) e carrega na Flash um software que recebe pela serial (*loader*).

### 2.1. Alimentação do Arduino

O circuito interno do Arduino é alimentado com uma tensão contínua de 5V, isto quando é conectado a uma porta Usb do computador. Esta conexão fornece a alimentação e também a comunicação de dados. Caso seja necessário é possível utilizar uma fonte de alimentação externa, que forneça uma saída dentre 7.5V e 12V contínua com um plug P4, ou pode ser ligada diretamente na placa utilizando os pinos Vin e Gnd.

## 3. Software

O Software é utilizado basicamente para escrever o código do programa, salvá-lo, compilá-lo, e realizar a gravação do código compilado no Arduino (memória flash) através da porta Usb do computador. A IDE do Arduino será utilizada para realizar estes passos. Este ambiente de desenvolvimento é baseado no Framework Wiring e na linguagem de programação C/C++.

Uma vez gravado o programa no Arduino, o computador não é mais necessário. A partir do momento em que se utiliza uma fonte de alimentação externa, o Arduino se torna uma placa totalmente independente. Mas antes de tudo é preciso obter os arquivos de instalação e drivers, que vêm juntos no mesmo pacote, que pode ser obtido no site oficial do Arduino[3]. O download deve ser selecionado de acordo com o sistema operacional utilizado, sendo ele compatível com Windows, Linux e Mac OS.

[3] Página de Download: http://arduino.cc/en/Main/Software

Após baixar e extrair os arquivos no local desejado execute o programa Arduino.exe, localizado na raiz da pasta principal. Em seguida conecte o seu Arduino ao computador através de uma porta Usb. Ao conectá-lo, um Led de *power* (pwr) acenderá, isto significa que a placa está energizada. Agora já é possível instalar os Drivers, para isso, será necessário seguir os seguintes passos de acordo com o Sistema Operacional utilizado.

## 3.1. Windows 7, 8 e 8.1

Será solicitado que um novo driver seja instalado, então deverá selecionar a “escolha manual de drivers”, então localize a pasta Drivers dentro do pacote extraído anteriormente.
Obs.:
1 - Caso não ocorra a detecção automática de Drivers, será necessário abrir:

- Painel de Controle > Gerenciador de Dispositivos

Em seguida selecione os Drivers que estão desatualizados (com uma exclamação) e selecione a opção "Atualizar Driver", logo após selecione a pasta Drivers extraída junto com o pacote anteriormente.

O Windows 8 e 8.1 possui uma particularidade com relação ao Windows 7. Por padrão a instalação de Drivers não assinados é bloqueada no Windows 8 e 8.1, caso não tenha êxito seguindo os passos acima, será necessário desbloquear esta opção.

1. Pressionar a tecla ‘windows’ + ‘R’
2. Digite shutdown.exe /r /o /f /t 00
3. Clique em ‘OK’
4. O sistema irá reiniciar, e abrir uma tela azul.
5. Selecione "Solução de Problemas"
6. Selecione "Opções Avançadas"
7. Selecione "Configurações de Inicialização"
8. Clique em "Reiniciar"
9. O sistema irá reiniciar, então selecione “Desabilitar Imposição de Assinatura de Driver”
10. Pronto! Agora é só seguir o procedimento de instalação novamente

Obs.:

1 - Se você possui um computador que veio com o Windows 8/8.1 pré-instalado, esta tela opções avançadas (modo de segurança) provavelmente será habilitada na *BIOS*

2 - Os passos acima provavelmente não funcionarão se o Modo de Segurança estiver desabilitado na BIOS

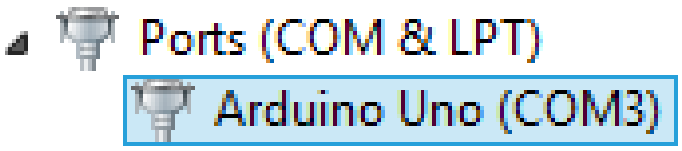



Figura 3.1 – Driver do Arduino Instalado corretamente (o n° da porta COM pode variar)

## 3.2. Linux

No Linux, abra o terminal e execute o seguinte comando:

- sudo aptitude install arduino

Ou procure pelo pacote "arduino" no Synaptic:

- menu Sistema > Administração > Gerenciador de pacotes Synaptic).

## 3.3. Mac OS

Saiba mais em: http://arduino.cc/en/Guide/MacOSX

## 3.4. Visão geral da IDE

Pronto! Após a instalação será possível abrir a IDE do Arduino, que tem a seguinte aparência:

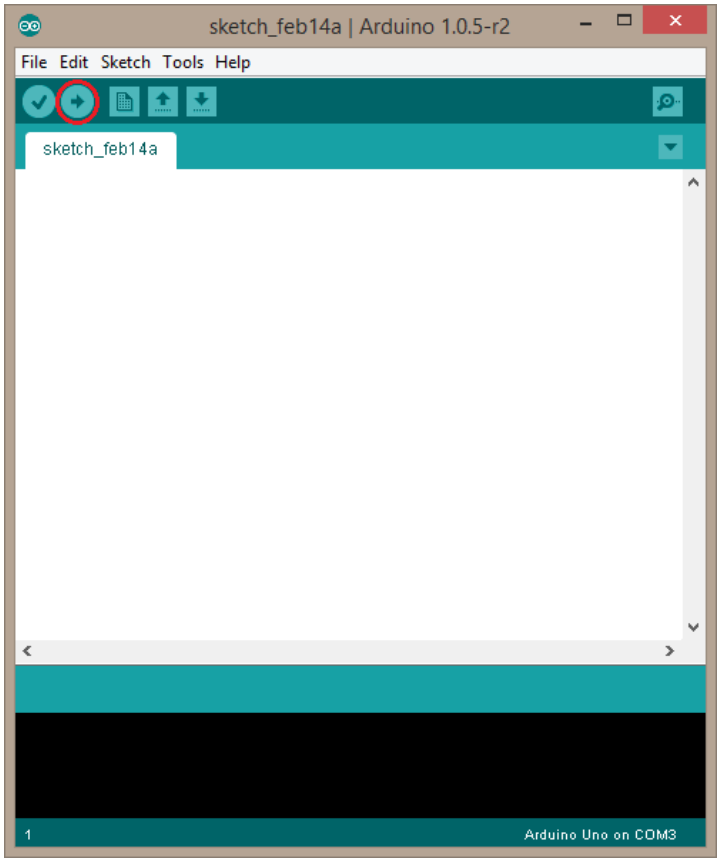



Figura 3.2 – IDE do Arduino versão 1.0.5-R2 sendo executada no Windows 8.1

A IDE do Arduino é muito simples e objetiva, tornando todo o processo de desenvolvimento e gravação bastante intuitivo. Além do espaço em branco destinado ao desenvolvimento do programa, existem 6 botões na parte superior: Verify, Upload (destacado de vermelho), New, Open, Save e Serial Monitor. Eles são utilizados, respectivamente para Verificar se existem erros no código, enviar (gravar) o programa no Arduino, criar um novo código, abrir um código existente e um monitor de dados da porta serial.

Um código desenvolvido para Arduino é chamado de Sketch, traduzindo do inglês ao pé da letra seria algo como “esboço” ou “rascunho”. Isso nos dá uma ideia de que nunca terminamos um código, sempre haverá melhorias e novas funcionalidades. O Sketch possui a extensão ‘.pde’.

A partir de agora sempre que for necessário gravar um novo programa no Arduino, basta conectá-lo na porta Usb, Selecionar a placa utilizada em Tools > Board e em seguida selecionar a Porta Serial (COM) associada a ele, neste caso Tools > Serial Port > COM3. Em

seguida, após abrir o programa desejado, basta clicar em “Upload”. Após executar estes passos, a IDE deverá exibir uma mensagem no final “Done Uploading”.

# 4. Fundamentos da Programação no Arduino

## 4.1. Algoritmo

Pode se dizer que Algoritmo é uma sequência de passos que devem ser seguidos para atingir um objetivo bem definido. Exemplo, uma receita de bolo.

## 4.2. Variáveis e Constantes

Um dado é constante quando não sofre nenhuma alteração ao decorrer do programa. Ou seja, do início ao fim da execução do programa o seu valor permanece o mesmo, inalterado.

A declaração de constantes pode ser feita de duas maneiras:

- Usando a palavra reservada “const”. Exemplo:

```
const int x = 10;
```

- Usando a palavra reservada “define”. Exemplo:

```
#define X = 10
```

Existem algumas constantes pré-definidas, cujos nomes não podem ser utilizados para a declaração de variáveis. Estas são chamadas palavras reservadas. Por exemplo:

- true – indica um valor lógico verdadeiro

- false – indica um valor lógico falso
- HIGH – indica que uma porta está ativada, ou seja, está em 5V.
- LOW – indica que uma porta está desativada, ou seja, está em 0V.
- INPUT – indica que uma porta será utilizada como entrada de dados.
- OUTPUT – indica que uma porta será utilizada como saída de dados.

Variáveis são posições da memória (lugares) cujo principal objetivo é armazenar dados.

- As variáveis são acessadas através de um identificador único.
- O conteúdo de uma variável pode variar ao longo do tempo durante a execução de um programa.
- Uma variável só pode armazenar um valor a cada vez.
- Um identificador para uma variável é formado por um ou mais caracteres, obedecendo a seguinte regra: o primeiro caractere deve ser uma letra.

Importante: Um identificador de uma variável ou constante não pode conter caracteres especiais ou palavras reservadas da linguagem.

### 4.2.1. Tipos de Variáveis

- void - Indica um tipo indefinido. Usado geralmente para informar que uma determinada função não retorna nenhum valor.
- boolean - Os valores possíveis são true (1) e false (0). Ocupa um byte de memória.
- char - Pode ser uma letra ou um número. A faixa de valores válidos é de -128 a 127. Ocupa um byte de memória.
- unsigned char - O mesmo que o char, porém a faixa de valores válidos é de 0 a 255.
- byte - A faixa de valores é de 0 a 255. Ocupa 8 bits de memória.
- Int - Armazena números inteiros e ocupa 16 bits de memória (2bytes). A faixa de valores é de -32.768 a 32.767.
- unsigned int - O mesmo que o int, porém a faixa de valores válidos é de 0 a 65.535.

- word - O mesmo que um unsigned int.
- long - Armazena números de até 32 bits (4 bytes). A faixa de valores é de -2.147.483.648 até 2.147.483.647.
- unsigned long - O mesmo que o long, porém a faixa de valores é de 0 até 4.294.967.295.
- short - Armazena número de até 16 bits (2 bytes). A faixa de valores é de -32.768 até 32.767.
- float - Armazena valores de ponto flutuante (com vírgula) e ocupa 32 bits (4 bytes) de memória. A faixa de valores é de -3.4028235E+38 até 3.4028235E+38
- double - O mesmo que o float.

### 4.2.2. Declaração de Variáveis

A atribuição de valores a variáveis e constantes é feito com o uso do operador de atribuição "=". Exemplos:

- int valor = 10;
- const float pi = 3.14;

Importante:

- O operador de atribuição não vale para o comando #define.

- A linguagem de programação do Arduino, como dito anteriormente, é baseada no C/C++ portanto é *Case Sensitive*, diferenciando letras maiúsculas de minúsculas. Portanto a declaração "int valSensor;" é diferente de "int ValSensor;".

## 4.3. Vetores e Matrizes

Uma variável escalar pode armazenar muitos valores ao longo da execução do programa, porém não ao mesmo tempo. Existem variáveis que podem armazenar mais de um valor ao mesmo tempo, essas variáveis são conhecidas como "variáveis compostas

homogêneas". A programação do Arduino permite trabalhar com dois tipos de variáveis compostas homogêneas: Vetores e Matrizes.

### 4.3.1. Vetor

A declaração de um vetor é feita da mesma maneira que uma variável escalar, entretanto é necessário definir o seu tamanho (quantidade de itens). Exemplo:

```
1       int vetor[5];
```

- Cria um vetor de tamanho 5 (cinco) do tipo inteiro.

Para atribuir um valor a uma determinada posição do vetor, basta usar o índice, ou seja, a posição onde o valor será armazenado no vetor. Exemplo:

```
1       vetor[0] = 3;
```

- Atribui o valor 3 a posição 0 do vetor.

Para acessar um valor em uma determinada posição do vetor, basta usar o índice, ou seja, a posição onde o valor está armazenado no vetor. Exemplo:

```
1       digitalWrite(vetor[0], HIGH);
```

-Ativa a porta cujo número está definido na posição 0 do vetor.

### 4.3.2. Matriz

Uma matriz é similar a um vetor, entretanto pode ser formada por duas ou mais dimensões. Este elemento do tipo bidimensional possui um determinado número de linhas e de colunas. Exemplo:

```
1       int matriz[3][6];
```

-Matriz com 3 (três) linhas e 6 (seis) colunas de elementos do tipo inteiro.

Para atribuir um valor a uma determinada posição da matriz, basta usar o índice da linha e o índice da coluna, ou seja, a posição onde o valor será armazenado na matriz.

Exemplo:

```
1       matriz[1][2] = 6;
```

-Atribui o valor 6 a posição 1 (linha), 2 (coluna) da matriz.

Para acessar um determinado valor em uma posição da matriz, basta usar o índice da linha e o da coluna, ou seja, a posição onde o valor está armazenado na matriz. Exemplo:

```
1       digitalWrite(matriz[0][0], HIGH);
```

-Ativa a porta cujo número está definido na posição 0 (linha), 0 (coluna) da matriz.

## 4.4. Operadores

Em uma linguagem de programação existem vários operadores que permitem operações do tipo: Aritmética, Relacional, Lógica e Composta.

### 4.4.1. Aritméticos

- + Adição
- - Subtração
- * Multiplicação
- / Divisão
- % Módulo (resto da divisão inteira)

### 4.4.2. Relacionais

- > Maior
- < Menor
- >= Maior ou igual
- <= Menor ou igual

- == Igual
- != Diferente

### 4.4.3. Lógicos

- && E (AND)
- || OU (OR)
- ! NÃO (NOT)

### 4.4.4. Compostos

- ++ Incremento
- -- Decremento
- += Adição com atribuição
- -= Subtração com atribuição
- *= Multiplicação com atribuição
- /= Divisão com atribuição
- != Não Igual

## 4.5. Comentários

Muitas vezes é necessário comentar alguma parte do código do programa. Existem duas maneiras de adicionar comentários a um programa em Arduino.

- Comentário de Linha: //

```
// Este é um comentário de linha
```

- Comentário de Bloco /* */

/* Permite acrescentar comentários com mais de uma linha */

Nota: Quando o programa é compilado os comentários são automaticamente eliminados do arquivo executável (que será gravado no Arduino).

## 4.6. Comandos de Seleção

Em vários momentos em um programa precisamos verificar uma determinada condição, possibilitando a seleção de uma ou mais ações que serão executadas. Um comando de seleção também é conhecido por desvio condicional, ou seja, dada um condição um parte do programa é executada. Os comandos de seleção podem ser do tipo:

- If - Seleção simples
- If/Else - Seleção composta
- Switch/Case/Break - Seleção de múltipla escolha

### 4.6.1. Simples

Um comando de seleção simples avalia uma determinada condição, ou expressão, para executar uma ação ou conjunto de ações. O comando de seleção simples é:

```
1    if (expressão) {
2    comando1;
3    comando2;
4    }
```

onde:
expressão – representa uma expressão a ser avaliada que pode ser do tipo lógica, relacional ou aritmética. O resultado da avaliação de uma expressão é sempre um valor lógico.
Comando 1 e 2 – comandos a serem executados.

### 4.6.2. Composta

Um comando de seleção composta é complementar ao comando de seleção simples. O objetivo é executar um comando mesmo que a expressão avaliada pelo comando if (expr) retorne um valor falso. O comando de seleção composta é:

```
1    if (expressão) {
2    comando1;
3    }
4    else {
5    comando2;
6    }
```

onde:

expressão – representa uma expressão a ser avaliada que pode ser do tipo lógica, relacional ou aritmética. O resultado da avaliação de uma expressão é sempre um valor lógico.

Comando 1 e 2 – comandos a serem executados.

### 4.6.3. Múltipla Escolha

Na seleção de múltipla escolha é possível avaliar mais de um valor. O comando de seleção de múltipla escolha é:

```
1    switch (valor) {
2    case x: comando1;
3    break;
4    case y: comando2;
5    break;
6    default: comando3;
7    }
```

onde:

valor – é um dado a ser avaliado. É representado por uma variável de memória.

Comando 1, 2 e 3 – comandos a serem executados.

case– indica a opção a ser executada.

default – opção padrão, no caso nenhuma outra alternativa tenha sido selecionada.

## 4.7. Comandos de Repetição

Muitas vezes é necessário repetir um ou mais trechos do código mais de uma vez, nestes casos devem ser utilizados os comandos de repetição para manter um "laço" em uma instrução ou conjunto de instruções. Os comandos de repetição podem ser:

- For - Baseado em um contador
- While - Baseado em uma expressão com teste no início
- Do-While: Baseado em uma expressão com teste no final

### 4.7.1. For

Este tipo de comando de repetição deve ser utilizado quando se sabe a **quantidade de vezes** que um determinado trecho do código deve ser executado. Exemplo:

```
1    for (contador início; expressão; incremento do contador) {
2    comando1;
3    }
```

onde:

contador - é uma variável do tipo int

expressão - é uma expressão relacional

incremento do contador - passo de incremento do contador

### 4.7.2. While

Este tipo de comando de repetição avalia uma expressão, **enquanto** for verdadeira, um determinado trecho do código permanece sendo executado. Exemplo:

```
1	while (expressão) {
2	comando1;
3	}
```

onde:

expressão – é uma expressão que pode ser lógica, relacional ou aritmética. A permanência de execução do "laço" é garantida enquanto a expressão for verdadeira.

Nota: Neste tipo de comando de repetição a avaliação da expressão é realizada no início do laço, ou seja, pode ser que o comando1 não execute nenhuma vez.

### 4.7.3. Do-While

Este tipo de comando de repetição executa um determinado trecho do código e em seguida avalia uma expressão. **Enquanto** essa expressão for verdadeira, o trecho permanece sendo executado. Exemplo:

```
1	do {
2	comando1;
3	} while (expressão);
```

onde:

expressão – é uma expressão que pode ser lógica, relacional ou aritmética. A permanência de execução do "laço" é garantida enquanto a expressão for verdadeira.

comando1 – comando a ser executado.

Nota: Neste tipo de comando de repetição a avaliação da expressão é realizada no final do laço, ou seja, é garantido que pelo menos uma vez o comando1 será executado.

### 4.8. Bibliotecas

Biblioteca é uma coleção de subprogramas utilizados no desenvolvimento de programas. Contém código e dados auxiliares, que provém serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular.

A IDE do Arduino já possui algumas bibliotecas padrões para as funções mais básicas, mas caso queira utilizar alguma outra, utilize o comando "#include <Biblioteca.h>"

Exemplo:

#include <SPI.h>

Nota: Para importar uma biblioteca externa, utilize a opção "Sketch > Import Library".

## 5. Tipos de Portas e Comunicação Serial

O Arduino possui dois tipos de portas: Analógicas e Digitais, sendo este último tipo dividido entre binárias e PWM. Existe esta distinção de portas, que devem ser designadas de acordo com o resultado esperado com os componentes ligados a elas.

### 5.1. Portas Digitais

As portas digitais trabalham com apenas dois valores de tensão: 0V e 5V. É importante frisar que os componentes ligados a estas portas só podem trabalhar com estas duas tensões, seja enviando ou recebendo dados. As principais funções para manipular as portas digitais são:

- digitalRead(pino)

Verifica a porta "pino" e retorna HIGH caso esteja em 5V e LOW caso esteja em 0V.

- digitalWrite(pino, valor)

Atribui os valores HIGH (5V) ou LOW (0V) para a porta "pino".

Nota: Apesar de possível não é recomendável utilizadas as portas digitais 0 e 1, pois elas estão conectadas diretamente ao sistema de comunicação do Arduino (pinos TX e RX, transmissão e recepção de dados, respectivamente).

Exemplo:

Geralmente quando aprendemos uma nova linguagem de programação, o primeiro programa mostrado é o famoso “Hello World”, mas como ainda não aprendemos a trabalhar com o Display LCD, o nosso primeiro programa será piscar um Led. Este e outros exemplos básicos de utilização de vários dispositivos são encontrados na própria IDE do Arduino, acessando: menu File > Examples

O nosso primeiro projeto com o Arduino será o Blink, que pode ser acessado no caminho: Menu File > Examples > Basics > Blink. O código apresentado terá a seguinte estrutura:

```
1   int led = 13;
2   void setup() {
3   pinMode(led, OUTPUT);
4   }
5   void loop() {
6   digitalWrite(led, HIGH);
7   delay(1000);
8   digitalWrite(led, LOW);
9   delay(1000);
10  }
```

Após abrir este código, e configurar corretamente a placa e a porta COM utilizada,

clique em Upload. A mensagem esperada é “Done Uploading”. Note que agora existe um led na placa Arduino piscando com intervalos de 1 segundo. Gostou? :) Agora vamos explicar como tudo funciona.

O Arduino Uno, conforme comentado anteriormente possui 14 portas digitais, que podem ser usadas como Entrada ou Saída. Quando utilizamos uma porta como Entrada (INPUT) significa que será feita uma leitura desta porta, já como Saída (OUTPUT) significa que vamos “escrever” alguma coisa nela, ou seja, o pino poderá ser alimentado.

Na linha 1 criamos uma variável do tipo “int” (vamos falar dos tipos de variáveis mais pra frente) e atribuímos o valor 13 a ela. Nada de mais até aí, certo? Mas quando dizemos pinMode(**led**, OUTPUT) que dizer “ModoDoPino (**13**, Saída)” significa que a porta 13 será de saída, sendo assim será possível atribuir um valor (HIGH = 5V ou LOW = 0V). Resumindo, será possível que tenha uma saída de 0V (led apagado) e 5V (led aceso).

No caso do Arduino Uno, a porta 13 possui um Led ligado internamente à placa, sendo assim quando esta porta fica em nível lógico alto (HIGH) a tensão nela será de 5V fazendo com que o Led acenda.

Em seguida chama-se a função Delay com o parâmetro 1000 que quer dizer algo como “Espere 1000 milissegundos (ou 1 segundo)”, e em seguida coloca-se a porta 13 novamente em nível lógico baixo (LOW) e executa novamente a função Delay.

Você pode observar que mesmo para um exemplo simples, existem duas funções:

- void setup() - serve para definir as configurações iniciais do programa, e é executada apenas uma vez.
- void loop() - é a função principal do programa e como o próprio nome já diz, é executada infinitamente até que a placa seja reiniciada ou desligada.

Estas são funções básicas, e devem ser utilizadas em todos os códigos, pois são necessárias para que a compilação do programa seja realizada corretamente. Caso esqueça uma delas, o código não compila.

Resumindo o funcionamento, o programa liga o Led, espera 1 segundo, desliga o led, espera 1 segundo. Ele faz isso sucessivamente até que a placa seja desligada ou reiniciada.

### 5.1.1. Resistor de Pull-Up/Pull-Down

Os resistores de Pull-Ups/Pull-Down são utilizados para evitar flutuação em pinos de entrada (INPUT). Na maioria das vezes é necessário implementar externamente, mas muitas vezes há resistores Pull-Up implementados internamente em alguns pinos do microcontrolador. No caso do Arduino, já existem Pull-Ups internos em todos os pinos digitais e analógicos (OBS: Só use Pull-Up nos pinos analógicos caso utilizar estes como digitais), portanto não há necessidade de implementar Pull-Up externamente.

Caso seja necessário de utilizar Pull-Ups externamente, segue abaixo algumas recomendações:

Para escolher o resistor de Pull-Up é necessário satisfazer duas condições:

- Quando o botão é pressionado, o pino de entrada vai para LOW. O resistor R1 limita a corrente do VCC que passa pelo botão e vai pro GND.
- Quando o botão não é pressionado, o pino de entrada vai para HIGH. O resistor R1 limita a tensão no pino de entrada.

Para essas condições o resistor não pode ter o valor muito baixo, pois passará uma corrente elevada pelo pino de entrada. E o resistor não pode ser muito alto senão não passará a tensão necessária para o pino de entrada.

Em geral, o resistor R1 deve ser um décimo menor que a impedância do pino de entrada, mas geralmente a impedância de entrada varia entre 100KΩ e 1MΩ. Mas suponha que seja necessário limitar a corrente do pino de entrada para 1mA(0.001A). Fazendo o cálculo pela Lei de Ohm:

$$V = R \times I$$

Sendo: V = 5V(tensão de alimentação)

I = 1mA(corrente através do resistor e chegando no pino de entrada)

R = (resistor de Pull-Up)

Resolvendo o cálculo:

5 = Rx0.001

Portanto: R = 5000Ω para o resistor de Pull-Up

## 5.2. Portas PWM

As portas PWM (Pulse Width Modulation, do inglês Modulação por Largura d Pulso) se diferenciam das portas digitais binárias pois podem trabalhar não apenas com as tensões 0V e 5V, mas com uma escala que vai de 0 a 255 entre essas tensões, onde o '0' quer dizer 0V e '255' quer dizer 5V. Ou seja, as portas PWM permitem obter resultados analógicos com meios digitais e são capazes de controlar a potência de saída de um sinal. Pode se controlar, por exemplo, a potência em um Led, permitindo aumentar ou diminuir sua intensidade luminosa.

## 5.3. Portas Analógicas

As portas Analógicas são utilizadas para entrada de dados. Diferentes das portas digitais, permitem não apenas ler os valores 0V e 5V, mas qualquer valor entre eles dentro de uma escala de 0 a 1023, onde o 0 representa 0V e o 1023 representa 5V, ou seja. Isto porque os conversores ADC (do Inglês Analog Digital Converter) são de 10 bit (10 bit = 2^10 = 1024 valores) e fornecem uma precisão de 0.005V ou 5mV. Essas portas são utilizadas, por exemplo, para ler os valores de um sensor.

## 5.4. Comunicação Serial

A Comunicação Serial permite a comunicação entre o Arduino e o computador, possibilitando o envio de mensagens entre ambos. As mensagens podem ser enviadas através do teclado, ou até mesmo de algum programa instalado no computador, permitindo monitorar e controlar uma aplicação do Arduino.

As principais funções para manipular a comunicação serial são:

- Serial.begin(velocidade)

Inicia a interface serial. O parâmetro velocidade é a taxa de transferência, por padrão utiliza-se 9600.

- Serial.print("Mensagem")

Exibe uma mensagem no monitor serial. Esta mensagem pode ser, por exemplo, a leitura de um sensor.

- Serial.available()

Retorna a quantidade de bytes disponíveis para leitura na porta serial.

- Serial.read()

Lê os dados na porta serial. Por exemplo, uma mensagem digitada a partir do teclado.

# 6. Fundamentos de Eletrônica

Antes de começar a desenvolver um circuito eletrônico, mesmo que seja muito simples, é necessário conhecer um pouco sobre os componentes básicos da eletrônica e o seu funcionamento. A eletricidade é sem dúvida o elemento mais importante, pois é onde tudo começa. Por isso se faz necessário conhecer os seus conceitos mais básicos, evitando assim acidentes desnecessários e componentes queimados devido ao mau uso. A

eletricidade se surge a partir do nível atômico. O átomo é constituído basicamente de um núcleo, onde são encontrados os prótons e os nêutrons. Existe uma camada chamada eletrosfera que envolve o átomo, é nesta camada onde são encontrados os elétrons, que ficam em órbita. Os elétrons possuem carga negativa, os prótons possuem carga positiva e os neutros possuem carga neutra (nula).

O átomo pode perder ou ganhar elétrons, e isto faz com que sua carga resultante seja positiva ou negativa. Se ele ganha um elétron ela fica sua carga resultante fica negativa, por sua vez se ele perde um elétron ela fica positiva. Os elétrons se movimentam de forma aleatória, mas quando são submetidos a um campo magnético ou uma DDP (Diferença de Potencial) eles passam a se mover de forma ordenada, neste momento é gerada uma Corrente Elétrica. A DDP também pode ser chamada de Tensão, e é gerada quando dois pontos possuem potenciais elétricos diferentes, o que faz com que os elétrons se movam do ponto de maior potencial para o de menor potencial elétrico. Este fenômeno ocorre devido ao equilíbrio, pois tudo na natureza tende a buscar seu estado de equilíbrio natural. Imagine um elástico, quando é esticado recebe energia potencial elástica. Quando para de ser esticado é rapidamente contraído e tende a voltar ao seu estado natural. Ou seja, o elástico tende a sair do estado de maior potencial para o de menor potencial. A unidade de tensão elétrica padrão é o Volt (V). Um exemplo é a tomada de sua casa, onde basicamente existe uma Fase e um Neutro. A fase possui o potencial elétrico maior, e consequentemente o neutro possui o menor. Quando um aparelho elétrico é ligado a esta tomada, o circuito é fechado e a partir deste momento existe uma corrente elétrica, que passa do ponto de maior potencial elétrico (Fase) para o menor (Neutro).

# 7. Componentes Eletrônicos Básicos

## 7.1. Cabos (Jumpers)

Figura 7.1 – Jumper (fio) para Protoboard

Os Jumpers são utilizados para conectar componentes sem a necessidade de soldá-los. Geralmente são utilizados em protótipos, nas protoboards, e são construídos de material condutor envolto de um material isolante.

## 7.2. Resistores

Figura 7.2 – Resistor de 1KΩ

Os resistores oferecem uma oposição à passagem de corrente elétrica. Esta oposição é chamada de resistência elétrica ou impedância, e possui a unidade de medida ohm, representado pela letra grega Ω(ômega maiúsculo). Estes componentes causam uma queda de tensão na região do circuito em questão, e nunca uma queda de corrente, apesar de limitá-la. Ou seja, a corrente elétrica que entra em um terminal do resistor é a mesmo corrente que sai pelo outro terminal, porém existe uma queda de tensão.

É importante frisar que todo material condutor possui certa resistência, mesmo que isso não seja ideal, pois não existe condutor perfeito.

A relação entre tensão, corrente e resistência, através de um objeto é dada por uma simples equação, Lei de Ohm:

$$R = \frac{V}{I}$$

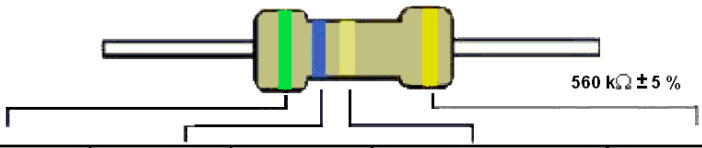
Onde V é a tensão em Volts, I é a corrente que circula através do objeto em Amperes e R é a resistência em ohms.

### 7.2.1. Código de cores

Como os resistores mais utilizados em eletrônica possuem o tamanho muito reduzido, a impressão do valor de sua resistência em ohms numericamente na estrutura é inviável. Por este motivo é utilizado um código de cores, que consiste em quatro faixas coloridas ao redor do resistor, indicadas como A, B, C e Tolerância.

As primeiras três faixas indicam o valor nominal de sua resistência e a tolerância indica a porcentagem na qual a resistência pode variar seu valor nominal, conforme a seguinte equação:

$$R = (10a + b) \cdot 10^c$$




| COR | 1ª BANDA | 2ª BANDA | 3ª BANDA | MULTIPLICADOR | TOLERANCIA | |
|---|---|---|---|---|---|---|
| PRETO | 0 | 0 | 0 | 1Ω | | |
| MARROM | 1 | 1 | 1 | 10Ω | ±1% | (F) |
| VERMELHO | 2 | 2 | 2 | 100Ω | ±2% | (G) |
| LARANJA | 3 | 3 | 3 | 1KΩ | | |
| AMARELO | 4 | 4 | 4 | 10KΩ | | |
| VERDE | 5 | 5 | 5 | 100KΩ | ±0,5% | (D) |
| AZUL | 6 | 6 | 6 | 1MΩ | ±0,25% | (C) |
| VIOLETA | 7 | 7 | 7 | 10MΩ | ±0,1% | (B) |
| CINZA | 8 | 8 | 8 | | ±0,05% | |
| BRANCO | 9 | 9 | 9 | | | |
| DOURADO | | | | 0,1 | ±5% | (J) |
| PRATEADO | | | | 0,01 | ±10% | (K) |

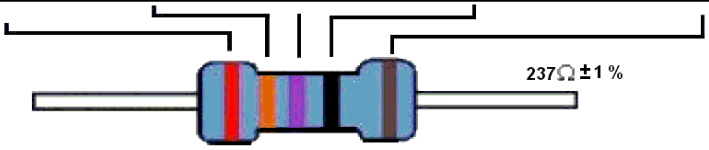



Figura 7.3 – Tabela de Cores dos Resistores

### 7.2.2. Associação de Resistores em Série

Os resistores podem ser associados em série, e sua resistência equivalente (neste caso, é soma de todas as resistências) é obtida através da seguinte fórmula:

$$R_{eq} = R_1 + R_2 + ... + R_n$$

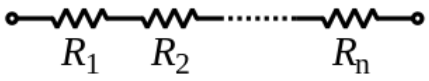



Figura 7.4 – Exemplo de associação de resistores em Série

### 7.2.3. Associação de Resistores em Paralelo

Os resistores também podem ser associados em paralelo, e sua resistência equivalente será o inverso da doma das resistências e será obtida através da seguinte fórmula:

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_n}$$

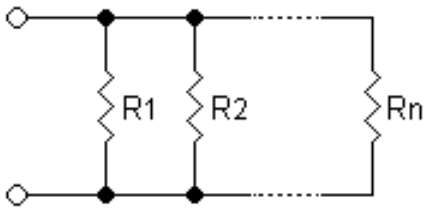



Figura 7.5 – Exemplo de associação de resistores em Paralelo

### 7.2.4. Divisor de Tensão

Como falado anteriormente, os resistores causam uma queda de tensão no circuito em questão. E através deste efeito é possível criar os divisores de tensão. Os divisores de tensão são circuitos com resistores que quando aplicada uma tensão sua saída é uma fração desta tensão de entrada.

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

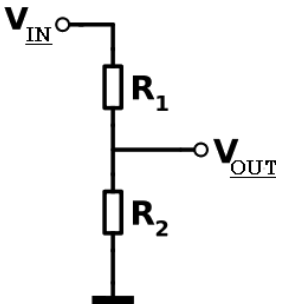



Figura 7.6 – Circuito divisor de tensão

## 7.3. Potenciômetro

Figura 7.7 - Potenciômetro

Potenciômetros (ou reostatos) são resistores que possui a sua resistência elétrica ajustável. Geralmente possui três conectores, dois laterais e um central. A resistência varia entre os pinos laterais e o central.

## 7.4. Capacitores

Figura 7.8 – Capacitor Eletrolítico

Capacitores são componentes capazes de armazenar energia elétrica. Pode-se dizer que funcionam como pequenas pilhas/bateria, porém com capacidade muito menor. Geralmente são formados por dois eletrodos ou placas que armazenam cargas opostas. Estas duas placas são condutoras separadas por um isolante (ou dielétrico). Quando acumula carga suficiente, a rigidez dielétrica do isolante é rompida, passando a conduzir corrente e toda a energia acumulada é liberada, de forma quase instantânea. Devido ao fato de cada placa armazenar cargas iguais, porém opostas, a carga total no dispositivo é sempre zero. A unidade de medida é Farad (F).

## 7.5. Indutores

Figura 7.9 – Indutor com núcleo de ferrite

O indutor é um componente elétrico passivo, que armazena energia em forma de campo magnético. Geralmente um indutor é construído como uma bobina de material condutor, como por exemplo, fio de cobre. Quando possui um núcleo de material ferroso, sua indutância é aumentada, concentrando as linhas de força do campo magnético que fluem no interior de suas espiras (quantidade de voltas do material condutor). A unidade de medida é Henry (H).

## 7.6. Leds

Figura 7.10 – Led Difuso Vermelho

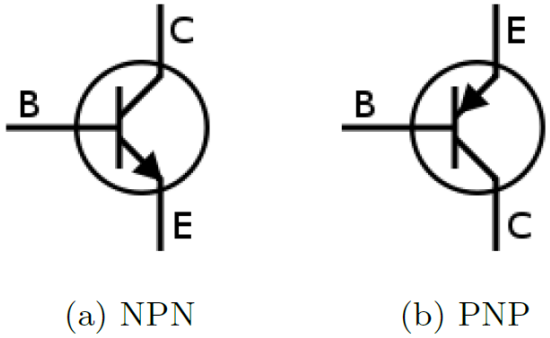
O Led (Light Emitting Diode) por exemplo são é um diodo emissor de luz. O diodo é um componente semicondutor, ou seja, permite a circulação de corrente em um sentido com muito mais facilidade do que no outro. Seu funcionamento baseia-se em um material contendo cargas negativas extras, chamado de material tipo-N, que fica separado por uma certa distância (chamada de zona vazia) de um outro material contendo cargas positivas extras, chamado de material tipo-P Ao conectar o catodo (material tipo-P) ao polo positivo do circuito, e o anodo (material tipo-N) do polo negativo, essas cargas irão se repelir, fazendo com que haja circulação de corrente elétrica no circuito, e no caso do Led, haverá emissão de luz.

## 7.7. Transistores


Figura 7.11 – Transistor com encapsulamento TO-92

Transistores são dispositivos semicondutores usados como amplificadores ou chaveadores. Seu funcionamento básico é uma tensão/corrente de entrada que é alterada

na saída. Os transistores são a base de todos os Circuitos Integrados (CI's), por exemplo, o processador do seu computador possui milhões deles. Existem vários tipos, mas focaremos nos transistores de estrutura de junção bipolar ou do inglês, BJT (Bipolar Junction Transistor), com polaridade PNP e NPN.




(a) NPN (b) PNP

Figura 7.12 – Representação dos Transistores no Circuito Elétrico

Os transistores possuem três terminais: Base, coletor e emissor. Cujas equações são:

IC + IB = IE (Corrente do Coletor + Corrente da Base = Corrente do Emissor)

IC = β IB (Corrente do Coletor = Beta x Corrente da Base)

Onde β é uma constante, também conhecida como hFE. A segunda equação mostra o poder de amplificação (ou ganho) do transistor onde IC pode ter um valor maior ou menor, dependendo do Beta e da corrente da base.

## 7.8. Chaves

Figura 7.14 – Chave Push Button

As chaves, ou interruptores são utilizados para abrir ou fechar um circuito, permitindo ou não a passagem de corrente elétrica. Possuem vários tipos de chaves, como por exemplo, o push buttom que ao ser pressionado permite a passagem de corrente elétrica e ao soltar ele impede novamente. Outros tipos de chave ao serem pressionadas permitem a passagem e somente ao pressioná-las novamente tornam a impedir.

## 7.9. Protoboard

Protoboard é uma placa utilizada para a prototipação, ou seja, no ensaio de montagem de circuitos eletrônicos experimentais. Sua vantagem é devido à facilidade de inserção e remoção de componentes, uma vez que não é necessário soldá-los. As placas geralmente variam de 170 a 1800 furos (ou pontos), e contém conexões verticais e horizontais.

Na superfície de uma matriz de contato há uma base de plástico contendo os furos onde são encaixados os componentes. Em sua parte inferior existem contatos metálicos que interligam eletricamente os componentes inseridos na placa. Geralmente a corrente suportada na protoboard é de 500mA. Os contatos metálicos estão em diferentes sentidos na matriz, que podem ser melhor visualizados na figura abaixo.

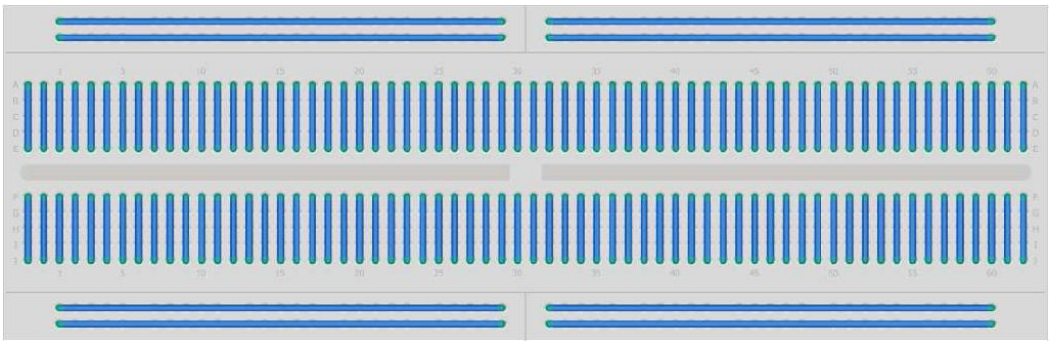


Figura 7.15 – Ligação interna da protoboard em azul

## 7.10. Sensores

O sensor é um dispositivo eletrônico que responde a um estímulo físico e/ou químico de maneira mensurável analogicamente. São utilizados para ler e interpretar variáveis ambientes, como luz, temperatura, som, distância, etc. Geralmente possuem uma resistência interna que varia de acordo com o estímulo recebido.

### 7.10.1. Sensor de luz LDR

Figura 7.16 – Sensor de Luz LDR

Os sensores de luz detectam a intensidade de luz no ambiente. Possuem uma resistência interna que varia de acordo com a quantidade de luz recebida, ou seja, quanto maior esta quantidade menor será a resistência interna.

### 7.10.2. Sensor de temperatura LM35DZ


Figura 7.18 – Sensor de Temperatura LM35DZ

Os sensores de temperatura funcionam como termômetros, detectando a temperatura no ambiente. Possuem uma resistência interna que varia de acordo com a temperatura, ou seja, quanto maior esta quantidade menor será a resistência interna.

## 7.11. Displays

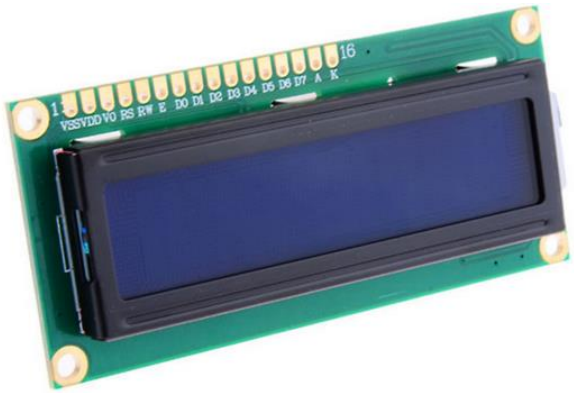



Figura 7.19 – Display 16x2 com Backlight Azul

Os displays são interfaces gráficas utilizadas para representar informações de forma visual. É possível exibir informações do programa, como por exemplo, uma mensagem ou a leitura de um sensor. Existem vários modelos de displays, entre eles o de 7 segmentos, 16x2 (16 linhas e 2 colunas) e 20x4. Alguns displays possuem um backlight, uma luz de fundo que facilita na leitura das informações.

## 7.12. Buzzer

Figura 7.20 – Buzzer

Buzzer é um dispositivo de áudio que pode ser mecânico, eletrônico ou piezoelétrico.

Entre diversas aplicações, buzzers são principalmente utilizados como beeps (buzina).

# 8. Eletrônica Digital

Um circuito digital é aquele na qual suas entradas e saídas possuem sinais digitais, ou seja, valores bem definidos (ou discretos). Geralmente estes circuitos trabalham apenas com dois valores, 0 e 1. Por isso são chamados de sistemas digitais binários. Por este motivo, toda informação será codificada em binário (bit). Por exemplo, se um circuito opera com as tensões 0V e 5V, a tensão 0V será representada pelo bit 0 e 5V será representada pelo bit 1.

## 8.1. Portas lógicas

As portas lógicas permitem que sejam realizadas operações aritméticas com os Bits, como adição, subtração, multiplicação e divisão. Para simplificar, será adotado que as operações lógicas contenham uma ou duas entradas e apenas uma saída. A saída 1 será tratada como **verdadeira** e 0 como **falsa**. Para um número finito de entradas, é possível utilizar a Tabela Verdade para obter todos os possíveis resultados da operação lógica.

### 8.1.1. AND

A porta lógica AND (E) (também conhecida como conjunção lógica) é uma operação lógica de dois operandos que resulta em um valor lógico verdadeiro somente se todos os operandos tiverem um valor verdadeiro (1). Equivale a uma multiplicação. Supondo que essa porta lógica tem duas entradas, A e B, e que A possui um bit em nível lógico alto e B um bit em nível lógico baixo:

A = 1 e B = 0, a saída “S” será um bit em nível lógico baixo pois 1 x 0 = 0.

### 8.1.2. OR

A porta lógica OR (OU) (também é chamada de disjunção lógica) é uma operação lógica entre dois ou mais operandos que resulta em um valor lógico falso somente se todos os operandos tiverem um valor falso (0), caso contrário será verdadeira. Equivale a uma multiplicação. Supondo que essa porta lógica tem duas entradas, A e B, e que A possui um bit em nível lógico alto e B um bit em nível lógico baixo:
A = 1 e B = 0, a saída "S" será um bit em nível lógico alto pois 1+0 = 1

### 8.1.3. NOT

A porta lógica NOT (NÃO) (também conhecida como inversora) é uma porta lógica digital que implementa a negação lógica. Uma entrada verdadeira (1) resulta em uma saída falsa (0), bem como uma entrada falsa (0) resulta em uma saída verdadeira (1). Ou seja, a porta NOT sempre produzirá como saída o inverso de sua entrada.

### 8.1.4. NAND

A porta lógica NAND possui o mesmo princípio da porta AND, porém com a saída barrada (negada). Ou seja, representa a junção da porta lógica AND com a porta inversora (NOT). Supondo que essa porta lógica tem duas entradas, A e B, e que A possui um bit em nível lógico alto e B um bit em nível lógico baixo:
A = 1 e B = 0, a saída "S" será um bit em nível lógico alto (1).

### 8.1.5. NOR

A porta lógica NOR possui o mesmo princípio da porta OR, porém com a saída

barrada (negada). Ou seja, representa a junção da porta lógica OR com a porta inversora (NOT). Supondo que essa porta lógica tem duas entradas, A e B, e que A possui um bit em nível lógico alto e B um bit em nível lógico baixo:
A = 1 e B = 0, a saída “S” será um bit em nível lógico baixo (0).

### 8.1.6. XOR

A porta lógica XOR (também conhecida como disjunção exclusiva ou exclusive OR) é uma operação lógica entre dois operandos que resulta em um valor lógico verdadeiro (1) somente se um dos operandos possui valor verdadeiro (1). Pode ser sintetizado como um detector de diferenças entre dois operandos lógicos. Supondo que essa porta lógica tem duas entradas, A e B, e que A possui um bit em nível lógico alto e B um bit em nível lógico baixo:
A = 1 e B = 0, a saída “S” será um bit em nível lógico alto (1).

### 8.1.7. XNOR

A porta lógica XNOR (também conhecida pelo termo função coincidência ou exclusive NOR) é a operação inversa da porta XOR. Possui o mesmo principio da porta lógica XOR, porém com a saída barrada (negada). Ou seja, a saída somente será verdadeira quando as entradas tiverem o mesmo valor (mesmo se valores baixos).

## 8.2. Representação das Portas e Operações Lógicas

As portas lógicas e respectivas operações são representadas conforme a figura abaixo:

**BLOCOS LÓGICOS BÁSICOS**

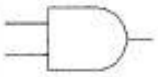
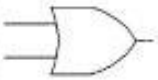
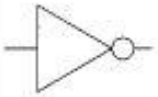
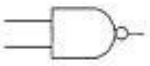
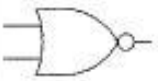
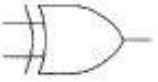
| PORTA | Simbologia | Tabela da Verdade | Função Lógica | Expressão |
|---|---|---|---|---|
| **AND** |  | A B S<br>0 0 0<br>0 1 0<br>1 0 0<br>1 1 1 | **Função E:** Assume 1 quando todas as variáveis forem 1 e 0 nos outros casos. | **S=A.B** |
| **OR** |  | A B S<br>0 0 0<br>0 1 1<br>1 0 1<br>1 1 1 | **Função OU:** Assume 0 quando todas as variáveis forem 0 e 1 nos outros casos. | **S=A+B** |
| **NOT** |  | A S<br>0 1<br>1 0 | **Função NÃO:** Inverte a variável aplicada à sua entrada. | **$S=\overline{A}$** |
| **NAND** |  | A B S<br>0 0 1<br>0 1 1<br>1 0 1<br>1 1 0 | **Função NE:** Inverso da função E. | **$S=\overline{(A.B)}$** |
| **NOR** |  | A B S<br>0 0 1<br>0 1 0<br>1 0 0<br>1 1 0 | **Função NOU:** Inverso da função OU. | **$S=\overline{(A+B)}$** |
| **XOR** |  | A B S<br>0 0 0<br>0 1 1<br>1 0 1<br>1 1 0 | **Função OU Exclusivo:** Assume 1 quando as variáveis assumirem valorem diferentes entre si. | **$S=A\oplus B$**<br>**$S=\overline{A}.B+A.\overline{B}$** |
| **XNOR** |  | A B S<br>0 0 1<br>0 1 0<br>1 0 0<br>1 1 1 | **Função Coincidência:** Assume 1 quando houver coincidência entre os valores das variáveis. | **$S=A\odot B$**<br>**$S=\overline{A}.\overline{B}+A.B$** |

Figura 8.1 – Representação de Portas e Operações Lógicas