APOSTILA DE ARDUINO

LIVROS PET
POTÊNCIA

1ª EDIÇÃO

PET
POTÊNCIA

APOSTILA DE ARDUINO

LIVROS PET
POTÊNCIA

1ª EDIÇÃO

PET
POTÊNCIA

Pagina reservada para posterior catalogação.

Programa de Educação Tutorial - Engenharia Elétrica
Universidade Federal do Piauí

APOSTILA DE ARDUINO

LIVROS PET POTÊNCIA

1ª EDIÇÃO

PET POTÊNCIA

Agradecimentos

A escrita da primeira edição deste livro foi um esforço conjunto de todos os PETianos ativos no ano de 2020 durante o período de quarentena e que só foi possível devido às suas contribuições em cada um dos capítulos. Desta forma, o PET faz um agradecimento especial aos PETianos João Paulo Marques de Melo, Miguel Leocádio de Sousa Neto, Giovana Delmondes Andrade, Matheus Ferreira Santos de Vasconcelos, Laís Brito Urano Portela, Antonio Evaldo Vieira de Goes Junior, Lucas Moura Rufini, Artemio Andrade Barros, Maria Clara Castro Higino de Sousa, Izadora da Silva Lima, Francisco Cleber da Conceição Feitosa, Phillip Gustavo Pinheiro Gallas, João Victor Soares de Assunção Santos e Mirla Borges Costa por contribuírem por toda a escrita e revisão desta apostila e ao tutor Marcos Antônio Tavares Lira que acompanhou e orientou os PETianos durante o desenvolvimento. Agradecemos também aos PETianos Mirla Borges Costa, João Victor Soares de Assunção Santos e Wesley Brito Bezerra por realizarem a edição e padronização da apostila quanto a sua formatação e estilização. Faz-se, por último, um agradecimento aos gestores de extensão João Victor Soares de Assunção Santos e Mirla Borges Costa por liderarem o processo de escrita da apostila.

Prefácio

O PET Potência apresenta a primeira edição do que se planeja ser uma coleção de livros abordando temas diversos para dar suporte a todos aqueles que querem aprender algo novo, mas de forma fácil, objetiva e prática. Ao passo que o Arduino se apresenta como uma ferramenta fácil de usar, com a ideia de que qualquer pessoa pode programar e despertar sua criatividade desenvolvendo circuitos, a falta de um matérial de apoio pode aumentar o tempo que se levaria para aprender perfeitamente suas funcionalidades ou ainda pior, desenvolver maus hábitos de programação e de montagem de circuitos.

Tomando como ideia e filosofia principal acompanhar o desenvolvimento dos conhecimentos em eletrônica básica e utilização dos principais módulos para Arduino, espera-se que aqueles que utilizem este material tenham conhecimentos básicos em algoritmos e linguagem de programação (preferencialmente C ou C++) para que seja dada uma atenção maior na utilização das funções nativas do Arduino enquanto a semântica da linguagem fica em segundo plano, bem como compreender o funcionamento dos dos módulos e dar maior atenção aos circuitos.

Ao longo da apostila foram utilizadas imagens de circuitos utilizadas no software Fritzing e todos os códigos desenvolvidos na IDE disponibilizada no site oficial do arduino https://www.arduino.cc, onde também é possível encontrar a documentação da maioria dos módulos utilizados aqui, de todas as funções e bibliotecas nativas disponibilizadas na IDE, além de materiais que orientam quanto a própria linguagem de programação utilizada.

Caso você queira contribuir com este projeto dando sugestões, apontando erros, escrevendo críticas ou elogios, sinta-se livre para entrar em contato pelo endereço de email petpotencia@ufpi.edu.br, pelo nosso instagram @petpotencia ou visitando o nosso site https://petpotencia.eng.br.

Esperamos que goste. Equipe PET Potência.

Sumário

Capítulo 1: Introdução ao Arduino 13

1.1. Placas e Componentes Internos 13

1.2. Trabalhando com Níveis Lógicos 15

1.3. Explorando a IDE 16

1.4. A Linguagem de Programação 18

1.5. Pontos de Tensão na Placa Arduino 20

1.6. Alimentação da Placa Arduino 22

1.7. Monitor Serial Arduíno 23

Capítulo 2: Conceitos Fundamentais e Primeiros Projetos 25

2.1. Blink: Primeiro Projeto 25

2.2. Conceitos Fundamentais em Circuitos Elétricos: Tensão Corrente e Resistência 26

2.2.1. Tensão Elétrica 26

2.2.2. Corrente elétrica. 27

2.2.3. Resistência 27

2.2.4. Potência Elétrica 28

2.3. Componentes eletrônicos básicos aplicados a Arduino: leds e resistores. 30

2.3.1. Leds 30

2.3.2. Resistores 31

2.4. Funções: DigitalRead(), DigitalWrite(); AnalogRead(), AnalogWrite() 36

2.4.1. Digital I/O – Dados de entrada/saída digitais 36

2.4.2. Analog I/O – Dados de entrada/saída analógicos 37

2.5. Operadores de comparação e lógicos 40

2.6. Praticando o Blink e algumas variações 40

2.6.1. O Blink mais Fácil 40

2.6.2. Utilizando a protoboard 41

2.6.3. Pisca na sequência 42

2.6.4. Pisca na Sequência com Aviso 44

Capítulo 3: Botões e Dispositivos Sinalizadores 49

3.1. Botão 49

3.1.1. Funcionamento 49

3.1.2. Aplicações 50

3.2. Display de 7 Segmentos de 1 Dígito 54

3.2.1. Funcionamento 54

3.2.2. Aplicações 55

3.3. Display LCD 72

3.3.1. Funcionamento 72

3.3.2. Aplicações 74

3.4. Buzzer 83

Capítulo 4: Diodos e Transistores 87

4.1. Diodos 87

4.1.1. Características dos Diodos 88

4.1.2. Exemplos de Diodos 89

4.1.3. Aplicações 91

4.2. Transistores 93

4.2.1. Transistor bipolar de junção (TBJ) 93

4.2.2. Aplicações 94

Capítulo 5: Atuadores 97

5.1. Motor DC 97

5.1.1. Aplicação 98

5.2. Servo Motor 101

5.2.1. Aplicação 102

5.3. Motor de Passo 105

5.3.1. Aplicação do Motor de Passo 106

5.4. Pulse Width Modulation (Modulação por Largura de Pulso ou PWM) 108

5.4.1. Aplicação: Controle de Velocidade de Motor DC 109

5.5. Importância de Fontes Externas, Relé e o Transistor como chave 112

5.5.1. Fontes Externas 112

5.5.2. Relé 112

5.5.3. Transistor como Chave 114

Capítulo 6: Sensores 117

6.1. Sensor óptico-reflexivo 118

6.1.1. Noções de funcionamento 119

6.1.2. Atenção ao usar o sensor óptico-reflexivo 122

6.2. Sensor de distância 124

6.2.1. Aplicação do Sensor de distância 126

6.2.2. Atenção ao usar o sensor ultrassônico 130

6.3. Sensor de Umidade e Temperatura 132

6.3.1. Aplicação do Sensor de Umidade e Temperatura 134

6.4. Sensor de Nível 137

6.4.1. Aplicação de Sensor de Nível 138

6.5. Sensor de batimentos cardíacos 140

6.5.1. Aplicação simples 141

6.5.2. Integração com DHT11 (simples detector de mentira) 141

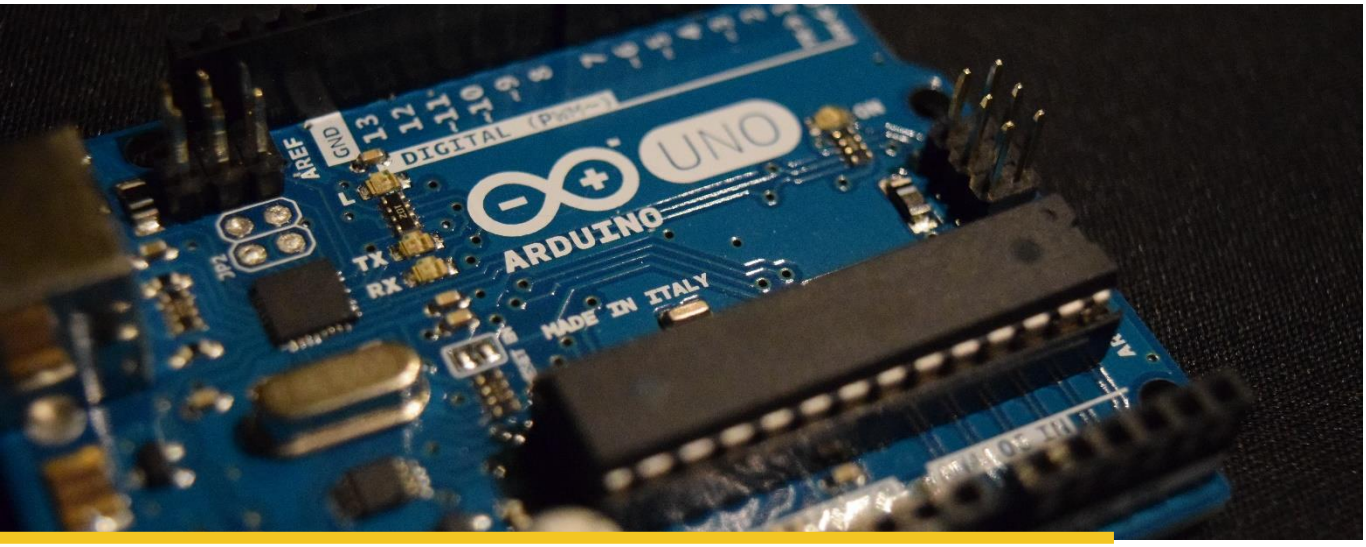
6.6. Outros sensores 143

6.6.1. Leitor de digitais 143

6.6.2. Sensor de temperatura infravermelho 144

6.6.3. Sensor de luz 145

6.7. Projeto sugerido 147



Capítulo 1: Introdução ao Arduino

A plataforma Arduino consiste na união de hardware e software fáceis de usar: são placas eletrônicas que, conforme um código de programação, são capazes de executar instruções fazendo um sistema eletrônico funcionar. O Arduino é um exemplo de microcontrolador, ou seja, é um computador que executa operações lógicas e algébricas, ativando e regulando o funcionamento de periféricos como sensores, displays e motores. Com a devida programação, esses dispositivos podem ser utilizados para numerosas aplicações, desde sistemas domésticos de irrigação de plantas até aplicações na área médica e na indústria.

1.1. Placas e Componentes Internos

A Figura 1.1 mostra a placa de Arduino UNO. Essa é a mais popular placa de Arduino em virtude de sua versatilidade. Essa placa apresenta seis portas analógicas, 14 portas digitais, tem preço acessível e alta disponibilidade no mercado. Esses fatores fazem com que essa placa seja adotada em muitas aplicações às quais se pode ter acesso na internet, e a torna a mais indicada para começar a estudar Arduino. Outros exemplos de placas bastante conhecidas são os Arduinos MEGA, Leonardo, Duemilanove e Nano. Esses modelos variam em quantidade de pinos, memória, terminais de alimentação, conectores para transferência de dados, quantidade de interrupções, entre outras características.

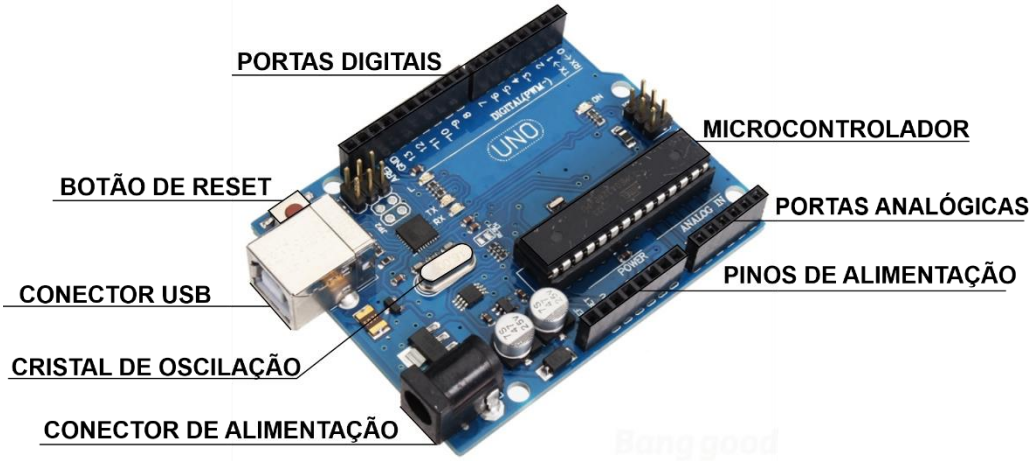


Figura 1.1 - Placa de Arduino UNO com descrição de componentes.

O **botão de Reset** do Arduino garante que toda a rotina do microcontrolador seja reiniciada, executando novamente todo o código gravado no mesmo. Note que esse botão não reseta a memória do microcontrolador e não remove o código gravado, apenas garante que, a qualquer momento, se reinicie toda a rotina, sem que seja necessário desligar o Arduino

Os periféricos usados em sistemas com Arduino necessitam de alimentação, que geralmente vem do próprio Arduino. Para isso, são usados os **pinos de alimentação**. Em protótipos de sistemas microcontrolados, comumente utiliza-se protoboards - mostrada na Figura 1.2 - para as montagens dos sistemas. As protoboards são placas que facilitam a montagem de protótipos de sistemas eletrônicos, dispensando a soldagem de vários elementos dos circuitos. Nesses casos, alimenta-se a protoboard a partir do Arduino. O uso adequado da protoboard será detalhado nos capítulos seguintes.

Figura 1.2 - Protoboard.

O circuito integrado na placa é o **microcontrolador** propriamente dito. Para desenvolver uma aplicação com Arduino, não é necessário ter a placa montada, mas sim esse circuito integrado e mais alguns componentes. Essas montagens são chamadas de *standalone* e têm maior aplicação comercial do que em prototipagem. Ao longo desta apostila, desenvolveremos códigos para aplicações nas placas de Arduino.

O **Conector USB** é o terminal onde será ligado o cabo USB que pode servir para alimentar o Arduino (quando estiver ligado a uma fonte de tensão de 5 V) ou para gravar um código a ser executado pelo microcontrolador (quando estiver conectado ao computador e com o uso da IDE Arduino). O **Conector de alimentação** pode receber energia de uma fonte de 5 V a 12 V de tensão contínua, cuja função exclusiva é de alimentação. O formato desses terminais pode variar de acordo com a placa utilizada.

O **Cristal de Oscilação** é um componente de alta precisão, que utiliza a ressonância de um cristal de material piezoelétrico (geralmente de Quartzo) em vibração. Os materiais piezoelétricos geram tensão elétrica quando submetidos a vibração. No caso dos cristais de oscilação, essa vibração cria um sinal elétrico com uma frequência altamente precisa, normalmente usada para medições de tempo em microcontroladores.

1.2. Trabalhando com Níveis Lógicos

As placas de Arduino são exemplos de sistemas digitais. Isso implica que elas trabalham com níveis lógicos para realizar suas operações. De maneira simplificada, pode-se entender os níveis lógicos como indicadores de passagem de corrente elétrica. Um **nível lógico baixo** (geralmente associado ao número 0) indica que não há passagem de corrente elétrica, o que corresponde a uma tensão elétrica de 0 V no pino observado. Um **nível lógico alto** (geralmente associado ao número 1) indica que há passagem de corrente elétrica no pino. Nas aplicações com Arduino, um nível lógico alto é normalmente resultado de uma tensão elétrica de 5 V.

Para fazer operações nos periféricos do Arduino com os níveis lógicos, utiliza-se linhas de código simples, que escrevem níveis lógicos no pino desejado. Se você deseja fazer corrente elétrica fluir por um Diodo Emissor de Luz (ou LED, do inglês Light Emitting Diode) e o fazer brilhar, você deve escrever um nível lógico alto no pino ao qual esse LED está conectado. Analogamente, para apagar o LED, escreve-se um nível baixo no pino.

O mesmo tipo de operação é feita para operar transistores, relés, válvulas e vários outros componentes de um sistema robótico. A lógica digital é utilizada nas mais variadas formas de circuitos elétricos ou eletrônicos, e funciona bem pela própria natureza binária que muitos dispositivos têm, como lâmpadas que ficam acesas ou apagadas, motores que se movimentam ou ficam parados e transistores, que permitem ou não passagem de corrente elétrica. Assim, é muito comum utilizar a lógica digital na programação de sistemas microcontrolados.

1.3. Explorando a IDE

Na programação do Arduino, utiliza-se o Ambiente de Desenvolvimento Integrado (ou IDE, do inglês Integrated Development Environment), disponível para download no site oficial www.arduino.cc. Após fazer a instalação corretamente da IDE em seu computador e ter o arduino em mãos, não vai faltar mais nada para iniciar a programar e carregar seus códigos em sua placa. Ao abrir a IDE, você deve ter uma janela semelhante a esta:

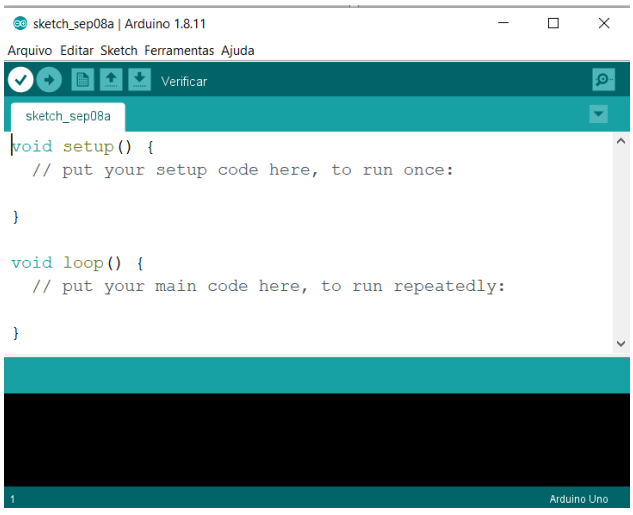


Figura 1.3: Janela da IDE do Arduino.

Inicialmente é importante conhecer as abas **Arquivo**, pois é nela que você irá criar novos *sketchs* (sketchs são programas que você irá escrever no arduino), salvar seu código atual ou abrir outros códigos feitos anteriormente.

A outra aba que você deve ter conhecimento é a aba de **Ferramentas**, nela você irá escolher a **placa** que irá utilizar nos seus projetos, aqui utilizaremos apenas o arduino UNO e será avisado caso não. Também verifique a opção **Porta**, nela você escolherá em qual porta USB do seu computador o arduino está conectado. Caso seu arduino esteja funcionando corretamente, ele irá aparecer na lista de portas disponíveis, basta selecionar ele e começar a escrever os códigos.

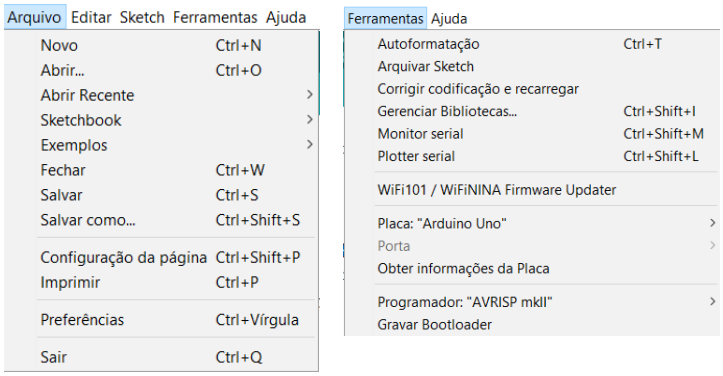


Figura 1.4: Observe as opções disponíveis nos menus.

Abaixo dos menus de abas, você pode ver, na figura 1.3, uma linha verde escura, essa é a linha que mais será utilizada ao escrever um código. O primeiro ícone com o "✓" irá **Verificar** o código encontrando erros de escrita e, como sabemos que C é uma linguagem compilada, irá traduzir o código gerando um arquivo em linguagem de máquina, mas não precisamos mexer com este arquivo, a IDE cuidará dele da melhor forma.

O segundo ícone com o simbolo "→" irá compilar o código e também carregá-lo para a placa conectada na porta devidamente selecionada anteriormente. Os três ícones ao lado são apenas as mesmas opções que você pode encontrar no **Arquivo.** O próximo item dessa linha que nos interessa, é o último no canto direito com o ícone de uma lupa, esse botão serve para abrir uma janela da IDE diferente que é o chamado **Monitor Serial**, essa janela será explicada com detalhes mais a frente.

1.4. A Linguagem de Programação

A programação do microcontrolador é feita em linguagem C++ com algumas modificações, mas é suficiente conhecer a linguagem C para acompanhar o desenvolvimento deste material. O código deve ser escrito e compilado na IDE e então carregado para a placa mediante conexão USB.

Além de pinos relacionados à alimentação e comunicação, as placas de Arduino têm pinos chamados de portas. Essas portas podem ser digitais ou analógicas. As portas digitais são binárias, ou seja, operam apenas com níveis lógicos altos ou baixos. As portas analógicas são muito comumente utilizadas para receber sinais analógicos ou fazer comunicações não binárias com outros dispositivos, como receber dados de temperatura ou pressão vindos de sensores, como veremos nos capítulos seguintes.

Quando se deseja trabalhar em situações como um monitoramento de umidade ou o nível de água em um tanque, não se pode adquirir os valores dessas variáveis (temperatura e altura, respectivamente) com as portas digitais. Essas variáveis podem assumir uma infinidade de valores diferentes (são variáveis flutuantes, que recebem números reais), ou seja, são de natureza analógica.

Um arquivo de sketch tem formato similar a um programa em C, onde se pode declarar variáveis, estabelecer as diretivas **#define**, declarar funções auxiliares, declarar estruturas (struct), porém não necessita de uma função **main()**. As funções necessárias para um programa de Arduino são **setup()** e **loop()**, geralmente declaradas como funções do tipo **void**.

A função **setup()** é executada assim que o Arduino é ligado, uma única vez; a função **loop()**, declarada por último, se repete até o desligamento do sistema. Evidentemente, pode-se declarar quantas funções auxiliares forem necessárias para o funcionamento do programa. A função **setup()** é usada para configurar as

```
pinMode(4, INPUT);
pinMode(2, OUTPUT);
```

portas e inicializar o monitor serial e alguns periféricos, como módulos e sensores. Utiliza-se a função **pinMode()** para declarar se uma porta do Arduino é de entrada ou de saída. Uma porta será de entrada se ela faz com que o Arduino receba alguma informação (transmitida como um sinal elétrico); será de saída se ela transfere níveis lógicos para os periféricos. Assim, uma porta que o microcontrolador irá utilizar para receber a leitura de um sensor de nível de água é uma porta de entrada; uma porta que é usada para acionar um LED é uma porta de saída. Essa função se aplica para portas digitais e analógicas. Sempre se deve declarar como as portas serão ativadas (como entrada ou saída), para que elas possam receber ou emitir sinais de maneira correta. A sintaxe dessa função é simples: se desejarmos configurar, por exemplo, a porta digital 4 de um Arduino (seja qual for o modelo), como porta de Entrada e a porta digital 2 como porta de Saída, escreveremos dentro função **setup()** as seguintes linhas:

Se a intenção for operar portas analógicas, utiliza-se a letra A junto ao número da porta ao passar os parâmetros da função **pinMode()**. Assim, a linha a seguir configura a porta Analógica 0 como porta de entrada:

```
pinMode(A0, INPUT);
```

O uso adequado de todas essas funções será detalhado em códigos ao longo deste material. Assim que uma porta digital é configurada como porta de saída, pode-se escrever níveis lógicos nessa porta, de acordo com a programação desejada. A função usada para esse fim é a função **digitalWrite()**. Para emitir um nível lógico baixo ou alto, por exemplo, na porta digital 7 do Arduino, utiliza-se as seguintes linhas, respectivamente:

```
digitalWrite(7, LOW);
//escreve nível lógico baixo na porta digital 7.

digitalWrite(7, HIGH);
//escreve nível lógico alto na porta digital 7.
```

As funções listadas ao longo desta seção fundamentais para a programação em Arduino e serão aplicadas no primeiro projeto do próximo capítulo.

1.5. Pontos de Tensão na Placa Arduino

Assim como as já apresentadas entradas e saídas de sinais de correspondência analógica e digital, a placa Arduíno também dispõe de portas de entrada e saída de tensão. Esses pontos de tensão têm por finalidade tanto a alimentação da placa Arduíno, quanto a alimentação de componentes, em aplicações específicas como por exemplo o acionamento de um Servomotor.

A organização e o agrupamento das portas do Arduíno são configurados de modo intuitivo quanto a finalidade de utilização, conforme observado na Figura 1.3. O grupo de pontos de tensão será descrito conforme a sequência disposta na imagem, sendo utilizado o modelo de placa Arduíno Uno como referência.

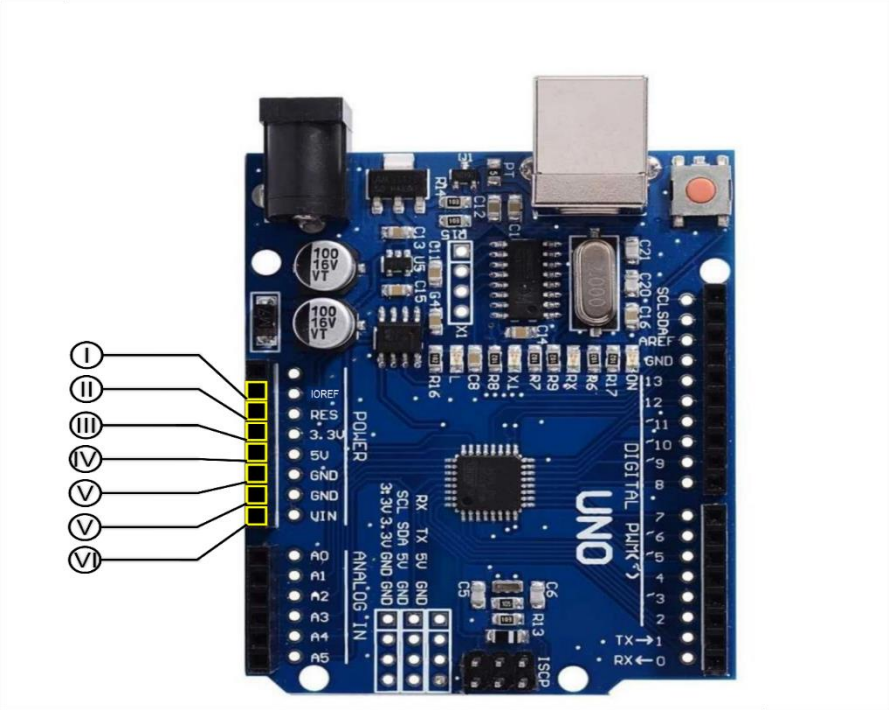


Figura 1.5 - Placa de Arduino UNO com indicação de pontos de tensão.

I. A porta IOREF é capaz de fornecer ao usuário da placa Arduíno o valor de referência de tensão com o qual a placa está trabalhando naquele momento. Este valor é de extrema importância, uma vez que sua medição

e análise permitem que dispositivos periféricos conectados à sua placa possam funcionar de forma correta.

II. A porta RESET possui a mesma função do botão *'reset'* presente na placa do Arduíno. Sua configuração irá se basear em valores lógicos de tensão aplicados a ela. Inicialmente configurada em estado de nível lógico alto 5 V o Arduíno continua a executar normalmente o Sketch, mas, uma vez que esse valor de referência é mudado para o nível lógico baixo 0 V, o reset é executado. Dessa forma o sketch em execução é interrompido e reiniciado. Isso significa que essa configuração para reset pode ser empregada durante a fase de seu código, de forma a funcionar automaticamente sem necessitar de operação manual como é o caso do botão.

III. A porta de 3,3 V é uma porta de saída de tensão, e oferece esse valor de tensão ao usuário. Através dessa porta, é permitido ao usuário que alimente dispositivos periféricos que ele venha a adotar em seus projetos. O valor máximo de corrente elétrica que esta porta consegue oferecer é de 50 mA. É importante atentar-se aos valores de referência para evitar mal funcionamento e possíveis defeitos, tanto à placa Arduíno quanto ao dispositivo conectado.

IV. A porta de 5 V também é uma porta de saída de tensão e oferece esse valor de tensão ao usuário. O valor de corrente máximo que essa porta consegue oferecer é de 40 mA.

V. As duas portas de nomenclatura GND (Ground) correspondem, ao que popularmente chamamos de terra. Essas portas possuem valor de 0 V, e são interligadas internamente. O ground é fundamental para o funcionamento de dispositivos que utilizam corrente contínua, isso porque garantem o potencial 0 V como referencial.

VI. A porta V_{in} é uma porta de entrada de tensão. Ela é destinada para a alimentação da placa Arduíno através de uma fonte de tensão externa como, por exemplo, uma bateria.

É possível observar que a placa Arduíno confere ao usuário a praticidade de possuir saídas de alimentação com dois valores de tensão. A primeira com tensão de saída igual a 3,3 V e a segunda com tensão igual a 5 V. Isso significa na prática, que durante seus projetos ela permitirá a alimentação de componentes diversos, que de acordo com suas especificações trabalham com tensões nessa faixa.

1.6. Alimentação da Placa Arduino

Para que o Arduíno funcione, é necessário que sua placa esteja adequadamente alimentada por uma fonte de tensão. Essa alimentação pode ser obtida de três maneiras diferentes: a primeira configuração é através da conexão de um Cabo USB do tipo B, entre a porta USB presente na placa à porta USB de um computador. Neste arranjo, a conexão via porta USB servirá tanto como fonte de alimentação à placa, quanto à programação e transmissão de dados. O valor de tensão que a porta USB do computador oferece nessa conexão é de 5 V. É importante salientar que, dependendo das configurações do projeto e da placa Arduíno utilizado, é possível utilizar um carregador de celular para alimentá-la.

A segunda configuração possível para alimentar a placa do Arduíno é através da utilização de uma fonte externa de alimentação. A placa já dispõe de um plug especial para essa conexão chamado *conector jack*, e o valor de tensão da fonte deve geralmente estar compreendido entre 9 V e 12 V. Esse arranjo é mais indicado quando o Arduino está sendo empregado em um projeto de configuração fixa, ou seja, configuração e aplicação definitivas com a conveniência de estarem instalados perto de uma tomada.

A terceira e última forma de alimentar a placa Arduíno é através da porta V_{in}. Esta configuração permite ao usuário adequar uma fonte já presente em seu projeto para a alimentação da placa. Sendo possível a conexão direta à porta V_{in} da placa, com a vantagem da regulação de tensão feita por um regulador interno. Vale ressaltar que, para a grande maioria das montagens que se sucedem neste livro, o tipo de alimentação de placa utilizado será através da conexão USB.

1.7. Monitor Serial Arduíno

Uma vez que o Arduíno está conectado ao computador através do USB, é possível utilizar um recurso interessante e de muita utilidade. Trata-se do Monitor Serial. Esta funcionalidade é obtida através do IDE, e permite a comunicação entre o computador e o Arduíno e vice-versa.

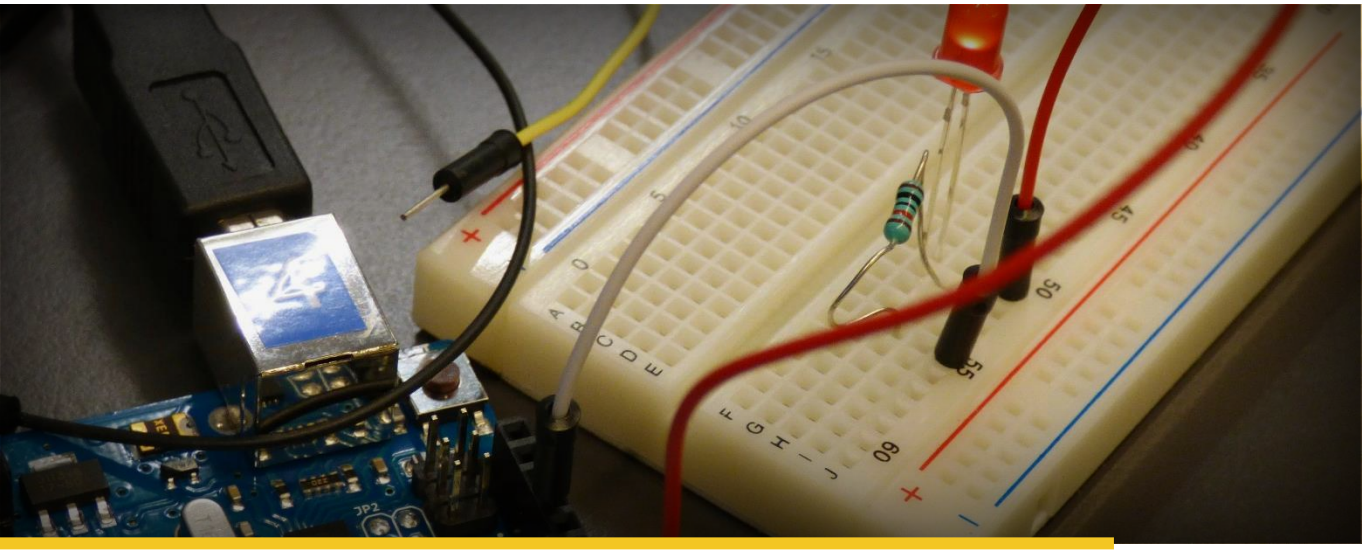
Para efeito de entendimento, podemos considerar o Monitor serial como sendo uma janela interativa, à disposição do usuário. Sendo capaz de processar informações e dados enviados pelo Arduíno e exibi-los. Assim como tornando possível o envio de comandos ao Arduíno pelo computador.

Mas para que essa comunicação aconteça é necessário que o usuário inicie a transmissão serial através de comandos específicos em seu código. Para exemplificar essa descrição utilizaremos o código a seguir:

```
void setup() {
  // Comando que inicializa a comunicação serial
  Serial.begin(9600);
}

void loop() {
  // Comando de envio da mensagem ao computador
  Serial.println(" Olá mundo");
  delay(1000);
}
```

Observe que a comunicação serial é iniciada dentro da função *void setup()*, com a utilização do comando **Serial.begin()**. O parâmetro entre parênteses representa a velocidade em que acontecerá a comunicação. E sua unidade de medida é dada em bits/s. O valor de 9600 adotado no código é o valor padrão, utilizado na comunicação entre Arduino e computador. Já dentro da função *void loop()* é colocada a mensagem que o Arduino deverá enviar, utilizando-se o comando **Serial.println()**. O comando **delay()** representa uma pausa, um tempo de espera até que o código continue sendo executado. E o parâmetro entre parênteses é dado em milissegundos.

Após a fabricação e execução desse sketch, o usuário deve abrir a janela de interação com um simples click no ícone de lupa no canto superior direito do IDE; através do menu ferramentas na barra superior ou através do atalho *ctrl + shift + M*. Assim o usuário irá perceber a mensagem surgindo em sua tela. As aplicações desta ferramenta são vastas. Você pode remodelar o código quantas vezes quiser, sinta-se à vontade para criar.



Capítulo 2: Conceitos Fundamentais e Primeiros Projetos

Para o contato inicial do Arduino associado a componentes externos, precisamos abordar alguns conceitos fundamentais de circuitos elétricos. Neste capítulo, discutiremos sobre o LED, resistores, tensão, corrente elétrica, algumas funções básicas do Arduino, além de explorarmos as possibilidades de reinventar nosso primeiro programa, incrementando-o com estruturas básicas de programação.

2.1. Blink: Primeiro Projeto

A coisa mais simples que pode ser feita utilizando-se um Arduino é o programa blink, que consiste apenas em fazer piscar um LED, acendendo-o e apagando-o com intervalos de tempo predeterminados. Este é considerado o projeto mais simples, principalmente por dois motivos: primeiro, utilizamos apenas duas das funções mais básicas do Arduino; e segundo, porque na própria placa vem um LED conectado à porta digital 13. Desse modo, é um projeto curto em que se utiliza apenas a sua plaquinha de Arduino, mas que também pode ser ampliado, dependendo apenas da sua criatividade.

2.2. Conceitos Fundamentais em Circuitos Elétricos: Tensão Corrente e Resistência

Neste tópico, abordaremos alguns fundamentos de eletricidade para compreender adequadamente o funcionamento e a razão de ser dos circuitos que você utilizará nos projetos com Arduino. Caso você já esteja familiarizado com este assunto, pode, sem prejuízo algum, saltar para o **tópico 2.1.2**.

2.2.1. Tensão Elétrica

O primeiro conceito a ser tratado é o de tensão elétrica, também conhecido como diferença de potencial (ddp) ou voltagem. Para compreendermos esta ideia, é importante entender primeiro a ideia de potencial elétrico em si. De modo simplificado, vamos utilizar uma analogia com a física mecânica: imagine que você está em um ponto A e deseja rolar, por meio de uma rampa de tamanho fixo, uma bola para alguém em um ponto B.

Imagine que ambos os pontos estão na mesma altura, para rolar a bola até seu destino você precisará fazer certo esforço, ou seja, gastar certa energia. Imagine agora que o ponto A esteja em uma altura inferior ao ponto B, você precisará gastar mais energia agora, mesmo que o caminho tenha a mesma distância física! E isto se intensifica conforme a diferença de altura aumenta, pois neste caso você estará enfrentando o efeito da gravidade que se opõe aos seus planos. Por outro lado, se você estiver em uma altura maior no ponto A do que a altura do ponto B, será mais fácil lançar a bola pela rampa pois a gravidade naturalmente atrairá a bola até o destino desejado.

Em eletricidade, há um conceito chamado campo elétrico que desempenha um papel análogo ao da gravidade em nosso exemplo, atraindo ou também repelindo as cargas elétricas, assim, dois pontos que possuem diferentes potenciais elétricos passam a ter uma tensão, ou diferença de potencial (ddp), entre eles e esta tensão originará um campo elétrico entre estes dois pontos que tenderá a impulsionar o deslocamento das cargas caso haja um circuito elétrico fechado assim

como a gravidade empurra uma bola para baixo em uma ladeira. Esta ddp é medida na unidade chamada Volts (V).

2.2.2. Corrente elétrica.

Tratemos agora da ideia de corrente elétrica. Este conceito é simplesmente o deslocamento das cargas elétricas em um circuito. Analogamente a um conjunto de bolas em uma rampa, que naturalmente tendem a descer do ponto mais elevado para o mais baixo em função da gravidade, em um circuito elétrico, as cargas se deslocarão do ponto de tensão positiva para o de tensão negativa, gerando a corrente elétrica, medida com a unidade Ampere (A).

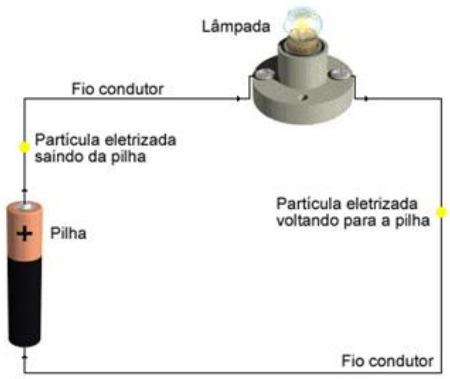


Figura 2.1: Tensão e Corrente Elétrica

2.2.3. Resistência

Um terceiro conceito importante é o de resistência. Continuando as analogias, imagine que no caso anterior existissem dois tipos de rampas que pudessem ser usadas para enviar a bola do ponto A ao ponto B. Ambas as rampas possuem o mesmo comprimento e diferença entre as alturas, mas uma delas é completamente plana e lisa, enquanto a outra é um caminho irregular. Em qual delas será mais fácil as bolas percorrerem? Naturalmente, percebemos que a primeira rampa oferece menos oposição ao deslocamento das bolas de nosso exemplo. Em um circuito elétrico, esta oposição ao deslocamento da corrente elétrica é denominada resistência, que é medida em Ohms (Ω).

Entre os três conceitos comentados há uma intrínseca relação. Isto se torna até mesmo intuitivo pela forma como uma mesma analogia pode ser utilizada para apresentar os três conceitos. Esta relação pode é descrita por meio de uma simples

equação matemática que relaciona tensão, corrente e resistência em um circuito elétrico.

$$V = R \times i$$

V → Tensão
R → Resistência
i → Corrente elétrica

2.2.4. Potência Elétrica

O último conceito que abordaremos dentro de circuitos elétricos, é o de potência e pode ser entendida como a energia que é gasta quando uma corrente atravessa um circuito. Vamos voltar para o exemplo da rampa da seção anterior, no caso de uma rampa muito inclinada e que você precise rolar a bola para cima da rampa, você vai precisar fazer muito mais força e gastar mais energia quanto mais inclinada for a rampa. Mas se você não quiser fazer mais força e se manter fazendo a mesma força que faria numa rampa menos inclinada, ainda iria conseguir subir, mas demorando muito mais.

A relação de potência elétrica é semelhante: para fazer uma carga elétrica (bola) passar por uma resistência (rampa) você vai precisar gastar uma quantidade de energia dependendo tanto da tensão (força) que você quer fazer e da corrente (velocidade) que irá passar pelo resistor. Assim define-se a potência elétrica como o produto da corrente e da tensão e é medida em Watts (W).

$$P = i \times V$$

Com a Lei de Ohm é possível ainda definir a potência a partir de valores de resistência em Ohms (Ω).

$$P = R \times i^2$$

$$P = V^2/R$$

Exercícios de Fixação

1) Conceitue, com suas palavras e levando em conta o seu entendimento pós leitura de diferença de potencial, corrente elétrica e resistência.
2) Como a diferença de potencial influencia para que lado as cargas se locomovem, isto é, qual o sentido da corrente?
3) Qual a equação matemática que relaciona as grandezas que você conceituou no exercício 1?
4) Imagine que você é um projetista de circuitos e ao analisar a corrente elétrica que está passando por um resistor de 220 Ohms descobre que ela tem um valor de 15 miliamperes (0,015 amperes). Como você já sabe utilizar a Lei de Ohm, qual é a diferença de potencial sobre esse resistor?
5) Analisando novamente o resistor de 220 Ohms da questão anterior, você mediu agora a tensão sobre ele, descobrindo um valor de 3,5 Volts. Qual a corrente que passa sobre ele?
6) Em uma montagem, você precisa descobrir qual a diferença de potencial nos terminais de um resistor de 100 Ω (Ohms) onde passa uma corrente elétrica de 20 miliamperes.
7) Sobre um resistor de 100 Ω passa uma corrente de 3 A. Determine qual o valor de potência que está sendo consumida por esse resistor.

2.3. Componentes eletrônicos básicos aplicados a Arduino: leds e resistores.

2.3.1. Leds

Agora que entendemos os fundamentos de circuitos, aprenderemos sobre dois componentes eletrônicos bastante utilizados em montagens com Arduino: leds e resistores. Caso você já conheça suficientemente estes componentes e seu funcionamento, poderá pular para o **tópico 2.1.3**.

Iniciando pelos leds, cujo nome vem da sigla em Inglês light-emitting diode, ou, em Português, diodo emissor de luz. Basicamente, trata-se de um tipo especial de diodo que é capaz de emitir luz de uma específica cor quando diretamente polarizado.

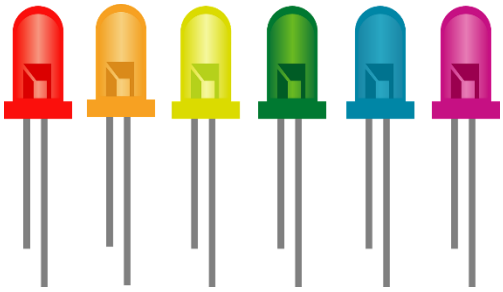


Figura 2.2: Exemplos de leds utilizados em Eletrônica.

Diodos são, basicamente, componentes eletrônicos de dois terminais (ânodo e cátodo) capazes de permitir a passagem de corrente elétrica em um sentido e bloquear a passagem no sentido oposto.

De modo idealizado, podemos compreender o papel do diodo em um circuito como o seguinte: quando uma tensão maior é aplicada ao ânodo (terminal positivo) do que ao cátodo (terminal negativo) ele atua como um condutor, não possuindo resistência; já quando uma tensão maior é aplicada ao cátodo do que ao ânodo, ele atua como uma resistência infinita que impede a passagem de corrente. Assim, podemos compreender o diodo como um componente que limita a passagem de corrente no circuito a um único sentido. Diodos serão mais discutidos nos próximos capítulos.

Sendo um tipo de diodo, o funcionamento de um led é bastante semelhante, porém, eles emitem luz quando ocorre a passagem de corrente no sentido positivo (do ânodo para o cátodo). Lembre-se que quando um diodo permite isso, ele possui uma resistência praticamente nula, logo, a corrente que passaria pelo componente seria muito alta, conforme a relação matemática supracitada. Por este motivo, é sempre necessário que se utilize um resistor em série com o led a fim de limitar a corrente a níveis seguros tanto para o Arduino quanto para o led em si, caso contrário poderia haver danos a um ou ambos os componentes. Além disso, ao contrário dos diodos comuns, os leds não são adequados para resistir a polarização reversa, por isso é importante conectá-los no sentido correto em seu circuito: ânodo recebendo maior tensão que o

Figura 2.3: Conectores em um led comum para eletrônica.

cátodo, que geralmente fica conectado ao GND.

2.3.2. Resistores

Na parte sobre conceitos de circuitos, você foi apresentado sobre a propriedade da resistência em circuitos elétricos, agora você conhecerá o dispositivo utilizado especificamente para introduzir um valor conhecido e controlado desta propriedade em seu circuito: o resistor.

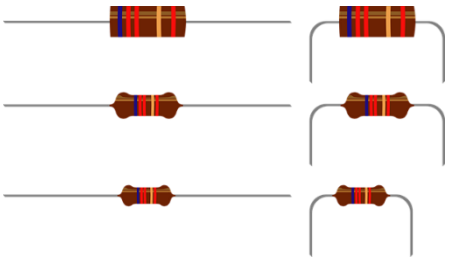


Figura 2.4: Exemplo de resistores utilizados em eletrônica.

Conforme a equação apresentada nos conceitos fundamentais, tensão, corrente e resistência estão relacionados em um circuito elétrico. Em variadas situações você pode necessitar utilizar um resistor para manipular corrente ou tensão em um circuito de acordo com suas necessidades.

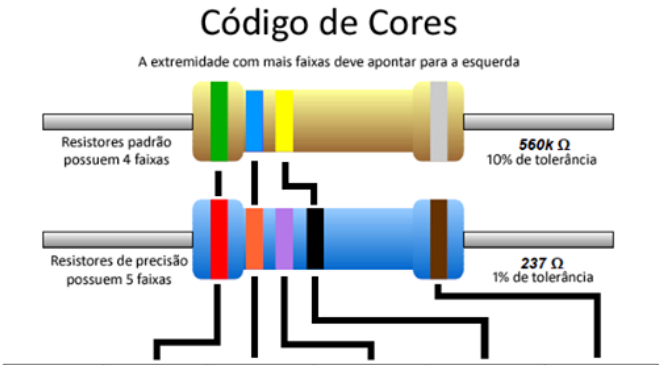
Primeiramente, tenha em mente que é comercialmente inviável ter opções para todos os valores possíveis de resistência, logo, você deverá fazer uso de valores comerciais.

Tabela 2.1 - Exemplos de valores comerciais disponíveis para resistores.

1,0 Ω	10 Ω	100 Ω	1 kΩ	10 kΩ	100 kΩ	1,0 MΩ	10 MΩ
1,1 Ω	11 Ω	110 Ω	1,1 kΩ	11 kΩ	110 kΩ	1,1 MΩ	15 MΩ
1,2 Ω	12 Ω	120 Ω	1,2 kΩ	12 kΩ	120 kΩ	1,2 MΩ	22 MΩ
1,3 Ω	13 Ω	130 Ω	1,3 kΩ	13 kΩ	130 kΩ	1,3 MΩ	
1,5 Ω	15 Ω	150 Ω	1,5 kΩ	15 kΩ	150 kΩ	1,5 MΩ	
1,6 Ω	16 Ω	160 Ω	1,6 kΩ	16 kΩ	160 kΩ	1,6 MΩ	
1,8 Ω	18 Ω	180 Ω	1,8 kΩ	18 kΩ	180 kΩ	1,8 MΩ	
2,0 Ω	20 Ω	200 Ω	2,0 kΩ	20 kΩ	200 kΩ	2,0 MΩ	
2,2 Ω	22 Ω	220 Ω	2,2 kΩ	22 kΩ	220 kΩ	2,2 MΩ	
2,4 Ω	24 Ω	240 Ω	2,4 kΩ	24 kΩ	240 kΩ	2,4 MΩ	
2,7 Ω	27 Ω	270 Ω	2,7 kΩ	27 kΩ	270 kΩ	2,7 MΩ	
3,0 Ω	30 Ω	300 Ω	3,0 kΩ	30 kΩ	300 kΩ	3,0 MΩ	
3,3 Ω	33 Ω	330 Ω	3,3 kΩ	33 kΩ	330 kΩ	3,3 MΩ	
3,6 Ω	36 Ω	360 Ω	3,6 kΩ	36 kΩ	360 kΩ	3,6 MΩ	
3,9 Ω	39 Ω	390 Ω	3,9 kΩ	39 kΩ	390 kΩ	3,9 MΩ	
4,3 Ω	43 Ω	430 Ω	4,3 kΩ	43 kΩ	430 kΩ	4,3 MΩ	
4,7 Ω	47 Ω	470 Ω	4,7 kΩ	47 kΩ	470 kΩ	4,7 MΩ	
5,1 Ω	51 Ω	510 Ω	5,1 kΩ	51 kΩ	510 kΩ	5,1 MΩ	
5,6 Ω	56 Ω	560 Ω	5,6 kΩ	56 kΩ	560 kΩ	5,6 MΩ	
6,2 Ω	62 Ω	620 Ω	6,2 kΩ	62 kΩ	620 kΩ	6,2 MΩ	
6,8 Ω	68 Ω	680 Ω	6,8 kΩ	68 kΩ	680 kΩ	6,8 MΩ	
7,5 Ω	75 Ω	750 Ω	7,5 kΩ	75 kΩ	750 kΩ	7,5 MΩ	
8,2 Ω	82 Ω	820 Ω	8,2 kΩ	82 kΩ	820 kΩ	8,2 MΩ	
9,1 Ω	91 Ω	910 Ω	9,1 kΩ	91 kΩ	910 kΩ	9,1 MΩ	

Um segundo aspecto a ter em mente é que sempre há uma margem de erro percentual em torno do valor alvo. Logo, em alguns casos, pode ser importante conhecer qual o valor da margem para garantir que seu projeto possa lidar com estas variações.

Um último aspecto prático útil é o de saber ler o código de cores utilizados nos componentes para exprimir facilmente os valores de resistência, bem como a margem de erro tolerável, por meio de um conjunto de faixas coloridas.Um último aspecto prático útil é o de saber ler o código de cores utilizados nos componentes para exprimir facilmente os valores de resistência, bem como a margem de erro tolerável, por meio de um conjunto de faixas coloridas.



Cor	1ª Faixa	2ª Faixa	3ª Faixa	Multiplicador	Tolerância
Preto	0	0	0	x 1 Ω	
Marrom	1	1	1	x 10 Ω	+/- 1%
Vermelho	2	2	2	x 100 Ω	+/- 2%
Laranja	3	3	3	x 1K Ω	
Amarelo	4	4	4	x 10K Ω	
Verde	5	5	5	x 100K Ω	+/- .5%
Azul	6	6	6	x 1M Ω	+/- .25%
Violeta	7	7	7	x 10M Ω	+/- .1%
Cinza	8	8	8		+/- .05%
Branco	9	9	9		
Dourado				x .1 Ω	+/- 5%
Prateado				x .01 Ω	+/- 10%

Figura 2.5: Código de cores para resistores.

Para os projetos neste capítulo, utilizaremos resistores de 330Ω para acender os leds. Vale ressaltar também que estes componentes não têm ordem correta de conexão, isto é, não faz diferençaqual terminal do resistor está conectado ao componente e qual está recebendo tensão.

Exercícios de Fixação

1) Indique se a afirmação é verdadeira (V) ou falsa (F). Caso falsa, tente corrigi-la.
   a. ( ) LEDs são diodos que emitem luz quando uma corrente passa do cátodo para o ânodo.
   b. ( ) Quanto maior a resistência de um resistor, maior é a queda da corrente que atravessa por ele.
   c. ( ) A resistência de um led é praticamente nula, portanto, para que ele não receba uma corrente maior do que a que pode receber, é necessário conectar um resistor.
   d. ( ) Para que funcione, um resistor precisa ser conectado em um sentido específico, assim como um led.

2) Você acabou de encontrar uma caixa de resistores desorganizadas e precisa descobrir suas resistências apenas utilizando o código de cores. Indique os valores de resistência dos resistores de 4 faixas com as cores abaixo:
   a. Laranja, laranja, marrom e dourado;
   b. Amarelo, violeta, vermelho e dourado;
   c. Marrom, preto, laranja e prateado;
   d. Marrom, verde, marrom e dourado;
3) Colocar as respectivas cores dos resistores abaixo:
   a. 15KΩ/10%;
   b. 100Ω/20%;
   c. 0,22MΩ/5%;
   d. 820Ω/1%;
4) Embora alguns resistores tragam impressos o valor da resistência, o código de cores é muito utilizado para a devida identificação do resistor. Sendo um resistor de quatro cores, assinale a alternativa correta.
   a. A primeira cor indica o valor do resistor; a segunda cor é o multiplicador; e, a terceira e quarta cor a tolerância.
   b. As duas primeiras cores indicam o valor do resistor; a terceira cor é o multiplicador; e, a quarta cor a tolerância.
   c. As duas primeiras cores indicam o valor do resistor; a terceira cor é a tolerância; e, a quarta cor a margem de erro.
   d. A primeira cor indica o valor do resistor; a segunda e a terceira cor é o multiplicador; e, a quarta cor a margem de erro.
   e. A primeira cor indica o valor do resistor; a segunda cor é o multiplicador; a terceira cor é a tolerância; e, a quarta cor a margem de erro.
5) Leds geralmente precisam de uma corrente de 15 miliamperes para acender. Você tem uma fonte de tensão de 6 volts, calcule qual resistor deve ser usado para acender o led, selecione um modelo comercial da tabela 2.1 para utilizar e encontre o seu código de cores para um resistor de 4 faixas.
6) Você procura por um resistor de 100 Ohms para um projeto no meio de vários itens eletrônicos antigos, você encontra um item similar a um

resistor, no entanto, devido ao tempo, não é possível identificar as cores dele. Após fazer vários testes de tensão e corrente você descobre que com 1 volt passam 0,1 ampere de corrente, 2 volts passam 0,2 amperes de corrente e com 3 volts passam 0,3 amperes. Desse modo, você encontrou o resistor que queria? Se não, informe o valor desse resistor.

7) Os resistores comuns conseguem gastar no máximo uma potência de 0,25 watts, se você forçar uma corrente ou uma tensão muito alta nele, ele irá pegar fogo. Sabendo disso, qual é o menor resistor possível que você pode colocar sob uma tensão de 5 volts?

2.4. Funções: DigitalRead(), DigitalWrite(); AnalogRead(), AnalogWrite()

As funções mais simples do Arduino são as que realizam leituras e escrituras de informações (desde estados de tensão até luminosidade, umidade, etc). Em outras palavras, uma função leitura é associada a um pino (uma porta) e envia dados para o microprocessador do Arduino realizar os comandos programados com tais. Estes são dados de entrada, a placa vai aguardar receber estes dados dos componentes externos. Uma função de escritura, por sua vez, atribui uma informação (por exemplo, um estado HIGH ou LOW) para o pino determinado. Estes são os dados de saída, na programação definimos qual saída será escrita e a qual pino será atribuído.

Como vimos, o Arduino é capaz de realizar computações tanto de dados analógicos, como de dados digitais. Desse modo, vamos entender melhor cada uma destas funções.

2.4.1. Digital I/O – Dados de entrada/saída digitais

A **digitalRead( )** recebe um único parâmetro pino, que é o número da porta digital, que deverá estar previamente declarada como de entrada (utilizando-se a pinMode(PIN,INPUT), conforme já visto). Como o nome diz, essa função somente pode ler dados digitais, ou seja, os valores que correspondem a HIGH ou LOW. Em um projeto de Arduino, a **digitalRead( )** pode ser utilizada, por exemplo, para ler um pino conectado a um botão, ou uma chave, que mudam o estado de voltagem conforme acionados: pressionados, o pino estará passando HIGH; solto, passa LOW.

```
digitalRead(pino);
digitalWrite(pino, HIGH);
```

A **digitalWrite( )** recebe dois parâmetros: o segundo é o estado digital que deseja-se atribuí-lo: HIGH ou LOW. Em um projeto, essa função é utilizada para atribuir voltagem ou não para um pino especificado. No caso do programa desta

seção, o *Blink*, utilizaremos justamente esta função para fazer um LED piscar, isto é, ora estar recebendo voltagem (aceso - HIGH.), hora não (apagado - LOW.)

2.4.2. Analog I/O – Dados de entrada/saída analógicos

Assim como a **digitalRead( )**, a função **analogRead( )** recebe um único parâmetro pino, que é o número da porta analógica, que deverá estar previamente declarada como de entrada (utilizando-se a pinMode(Apin,INPUT), conforme já visto). Essa função permite que o Arduino possa processar esses dados analógicos, através de um conversor analógico-digital de 10 bits, que todas as placas de Arduino possuem. Isso significa que o esse conversor associa tensões de 0V até a tensão operacional (5V ou 3.3V) a valores inteiros, de 0 a 1023.

```
analogRead(pino);
analogWrite(pino, valor);
```

Desse modo, a **analogRead( )** retorna um valor do tipo inteiro que corresponde à leitura analógica do pino especificado, respeitando os limites do intervalo de leitura do seu Arduino.

Em um projeto, a **analogRead( )** pode ser utilizada, por exemplo, conectada a um potenciômetro, que é uma resistência variável, ou sensores como de luminosidade, umidade, temperatura, *etc*. Este tipo de componente será abordado nos próximos capítulos.

Outra função para manipulação de dados analógicos é a **analogWrite( )**, que recebe dois parâmetros: primeiro, é o número do pino analógico previamente configurado como saída; o segundo, é o valor da razão cíclica, isto é, qual o número contido no intervalo de variação de intensidade de tensão que será gravado.

Basicamente, a **analogWrite( )** atribui um valor analógico a um pino, que na verdade é uma onda PWM (sinal digital modificado que se aproxima a uma onda analógica – vide figura abaixo: Em cima, o sinal analógico; embaixo, o sinal PWM digital que gera efeito praticamente igual ao do sinal analógico puro).

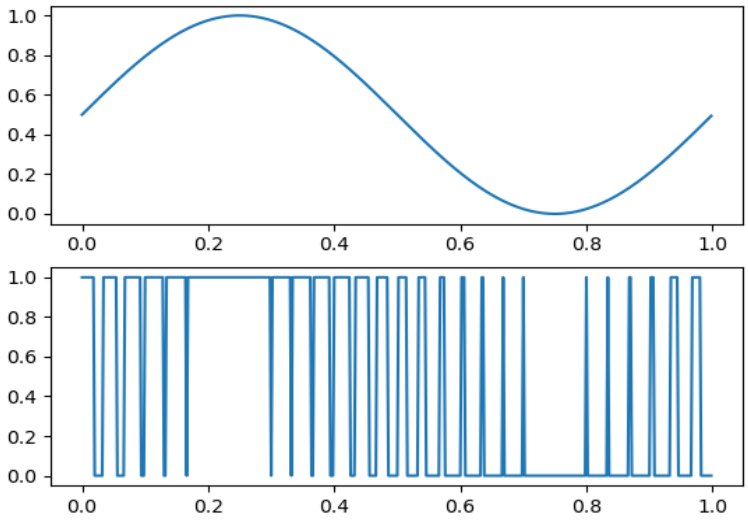


Figura 2.7: – Sinal analógico e PWM equivalentes.

Nessa perspectiva, os pinos que esta função pode operar são os que podem gerar sinais PWM,que, na placa, são sinalizados com um "~" na frente do número da porta. No caso do Arduino UNO, estes pinos são o 3, 5, 6, 9, 10 e 11.

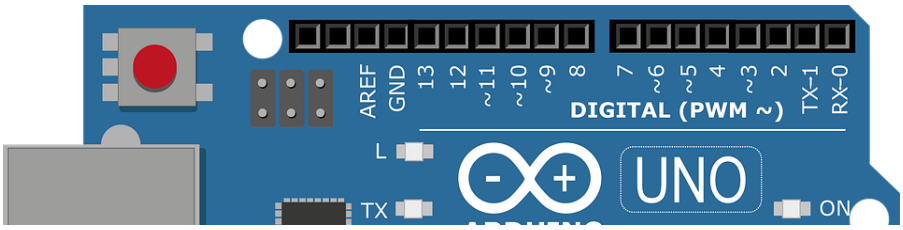


Figura 2.6: Pinos PWM do Arduino Uno.

Para os nossos projetos, vamos utilizar algumas estruturas condicionais, laços de repetição e operadores lógicos. A seguir, vamos abordar algumas estruturas de programação. Caso já esteja familiarizado com elas, pode pular para o **tópico 2.1.6**, no qual, finalmente, partiremos para os exemplos de projetos.

Exercícios de Fixação

1) Quais os parâmetros que a digitalRead e a AnalogRead recebem? O que essas funções fazem?
2) Quais os parâmetros que a digitalWrite e a AnalogWrite recebem? O que essas funções fazem?
3) Explique, com suas palavras, o que o conversor analógico-digital faz.

4) Qual o comando utilizado para escrever um valor num pino digital configurado como INPUT na plataforma Arduino?
   a. digitalRead;
   b. DigitalWrite;
   c. pinMode;
   d. Serial.begin
5) A função analogRead() retorna valores entre 0 e 1024, sendo o zero quando na porta liga-se uma tensão de 0V e 1024 quando se liga a uma tensão de 5V. Normalmente você não trabalha com esses valores pois não representam o valor realmente lido, para isso é preciso fazer uma normalização. Uma normalização padrão é para converter a escala entre 0 e 1024 em uma escala de 0V a 5V. Escreva a formulação matemática que realiza essa mudança de escala para o Arduino.
6) Diga, para os valores de tensão abaixo, qual o valor lido pelo Arduino e retornado para a função analogRead() (exemplo: 5V irá retornar o valor de 1024):
   a. 3,3V
   b. 4,1V
   c. 0,6V
   d. 4,9V
   e. 2,0V

2.5. Operadores de comparação e lógicos

Em linguagens de programação, podemos utilizar certos operadores a fim de fazer comparações entre variáveis e também para verificar ou estabelecer relações lógicas em um algoritmo. A seguir você pode consultar uma tabela resumo sobre os principais.

Tabela 2.2 - Operadores lógicos utilizados pelo arduino.

Tipo	Símbolo	Nome	Função
Comparação	==	Igualdade	Verifica se dois termos são iguais.
Comparação	!=	Diferença	Verifica se dois termos são diferentes.
Comparação	>	Maior que	Verifica se o primeiro termo é maior que o segundo.
Comparação	<	Menor que	Verifica se o primeiro termo é menor que o segundo.
Comparação	>=	Maior ou igual que	Verifica se o primeiro termo é maior ou igual ao segundo.
Comparação	<=	Menor ou igual que	Verifica se o primeiro termo é menor ou igual ao segundo.
Lógico	!	Não	Nega um resultado lógico, o verdadeiro torna falso e o falso, verdadeiro.
Lógico	||	Ou	Estabelece uma relação lógica de "ou" entre dois termos. Um ou outro deverá ser verdadeiro a fim da afirmação ser verdadeira.
Lógico	&&	E	Estabelece uma relação lógica de "e" entre dois termos. Ambos os termos precisam ser verdadeiros para que a afirmação também o seja.

2.6. Praticando o Blink e algumas variações

2.6.1. O Blink mais Fácil

Para este projeto, precisaremos apenas do próprio Arduino. Vamos utilizar o Uno R3. Como dito no início deste capítulo, o programa mais simples utiliza o

LED embutido na própria placa. Este LED está já está devidamente conectado ao pino 13.

```
// A função de setup roda sempre que o código é
// iniciado ou resetado
void setup() {
  // Configurar o pino que possui um LED embutido
  pinMode(LED_BUILTIN, OUTPUT);
  // A variável LED_BUILTIN já corresponde ao pino
  // do LED embutido na placa, que é o pino 13; você
  // pode colocar apenas o número 13 que funciona igual
}

void loop() {
  // Acende o LED (voltagem alta)
  digitalWrite(LED_BUILTIN, HIGH);
  // espera um segundo para executar a próxima instrução
  delay(1000);
  // apaga o led (voltagem baixa)
  digitalWrite(LED_BUILTIN, LOW);
  // espera um segundo para executar a próxima instrução
  delay(1000);
}
```

2.6.2. Utilizando a protoboard

Agora vamos testar uma variação deste código com uma montagem externa, ou seja, utilizando a protoboard. Vamos fazer um LED piscar muito rápido e depois mais devagar. Você precisará de um LED, um resistor de 330Ω e jumpers (fios conectores).

O terminal positivo do LED deve ser conectado ao pino especificado no programa, que nesse é o pino 12, com um resistor de 330Ω; e o terminal negativo, deve estar no GND.

Tenha em mente que a estrutura 'for' é um laço de repetição de 3 parâmetros: a variável de contagem

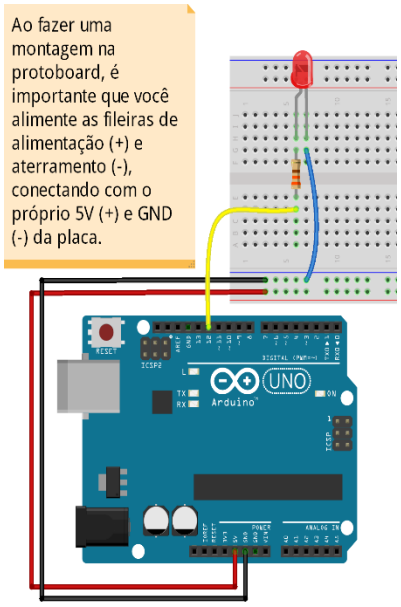


Figura 2.8: Montagem do exemplo.

sendo atribuído o seu valor inicial; a condição de validade; e,por fim, a definição de como será feito o incremento da variável de contagem.

```
// variável auxiliar para contagem
int cont;
void setup() {
  pinMode(12, OUTPUT);
}
void loop() {
  // neste primeiro laço, o LED piscará mais devagar,
  // com uma frequência mais baixa, piscando 10 vezes
  // até sair deste laço
  for (cont = 0; cont<10; cont++ ){
    // para cont inciando em 0, enquanto cont for
    // menor que 10, incremente uma unidade a cont

    digitalWrite(12, HIGH); // acende o LED (voltagem alta)
    delay(500);       // permanece aceso por meio segundo
    digitalWrite(12, LOW); // apaga o led (voltagem baixa)
    delay(500);       // permanece apagado por meio segundo
  }

  // neste segundo laço, o LED piscará mais rápido,
  // com uma frequência maior, piscando 20 vezes até
  // sair deste laço e voltar para laço anterior,
  // seguindo o loop
  for (cont = 20; cont>0; cont--){
    // para cont inciando em 20, enquanto cont for
    // maior que 0, decremente uma unidade a cont

    digitalWrite(12, HIGH); // acende o LED (voltagem alta)
    delay(100);  // permanece aceso por um décimo de segundo
    digitalWrite(12, LOW);  // apaga o led (voltagem baixa)
    delay(100);  // agora apagado por um décimo de segundo
  }
}
```

2.6.3. Pisca na sequência

Vamos trabalhar a criatividade! A ideia deste projeto é fazer quatro LEDs piscarem, um após o outro, da direta para a esquerda, e depois, piscarem na ordem

contrária, da esquerda para a direta. Vamos utilizar 4 LEDs, conectados aos pinos 0 a 3, cada um com seu ânodo ligado a um resistor de 330Ω e seus cátodos no GND.

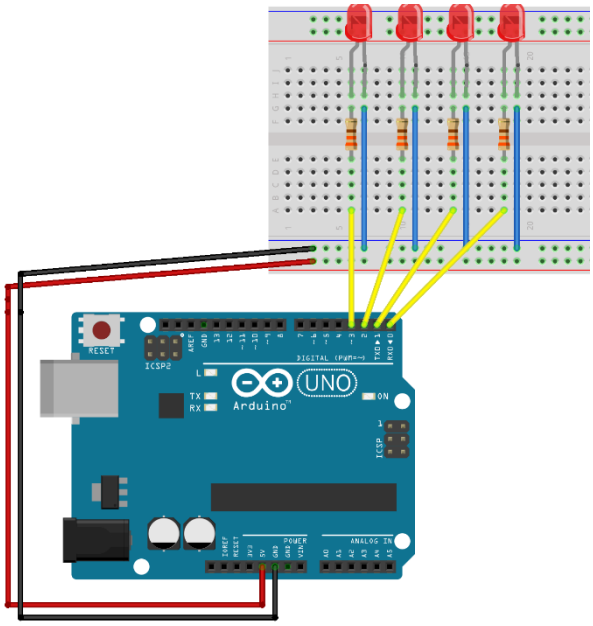


Figura 2.9: Montagem do exemplo

```
int aux;  //variável auxiliar para a sequência de piscagem
void setup() {
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
}
void loop() {
  // laço para piscar da direita para a esquerda
  // (da porta 0 à porta 3)
  for (aux = 0; aux<=3 ; aux++ ){
    // o pino que o led acenderá muda a cada iteração
    digitalWrite(aux, HIGH); // acende o led (voltagem alta)
    delay(200);              // mantém aceso por 200ms
    digitalWrite(aux, LOW);  // apaga o led (voltagem baixa)
  }
  // laço para piscar da esquerda para a direita
  // (da porta 3 à porta 0)
  for (aux = 3; aux>=0 ; aux-- ){
    //o pino que o led acenderá muda a cada interação
    digitalWrite(aux, HIGH); // acende o led (voltagem alta)
    delay(200);              // mantém aceso por 200ms
    digitalWrite(aux, LOW);  // apaga o led (voltagem baixa)
  }
}
```

2.6.4. Pisca na Sequência com Aviso

Agora vamos adaptar o projeto anterior: utilizaremos o monitor serial para avisar em que direção a sequência de LEDs está acendendo. Como utilizaremos a comunicação serial, os pinos 0 e 1 não devem ser utilizados. Aproveitando a montagem anterior, vamos mudar as conexões de 0 a 3 para os 2 a 5.

```
int aux;  //variável auxiliar para a sequência de piscagem
void setup() {
  // Como vamos utilizar comunicação serial,
  // manteremos os pinos 0 e 1 desocupados
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  // laço para piscar da direita para a esquerda
  // (da porta 2 à porta 5)
  for (aux = 2; aux<=5 ; aux++ ){
    // o pino que o led acenderá muda a cada iteração
    Serial.println("Indo para a ESQUERDA");
    digitalWrite(aux, HIGH); // acende o led (voltagem alta)
    delay(200);              // mantém aceso por 200ms
    digitalWrite(aux, LOW);  // apaga o led (voltagem baixa)
  }

  // laço para piscar da esquerda para a direita
  // (da porta 5 à porta 2)
  for (aux = 5; aux>=2 ; aux-- ){
    //o pino que o led acenderá muda a cada iteração
    Serial.println("Indo para a DIREITA");
    digitalWrite(aux, HIGH); // acende o led (voltagem alta)
    delay(200);              // mantém aceso por 200ms
    digitalWrite(aux, LOW);  // apaga o led (voltagem baixa)
  }
}
```

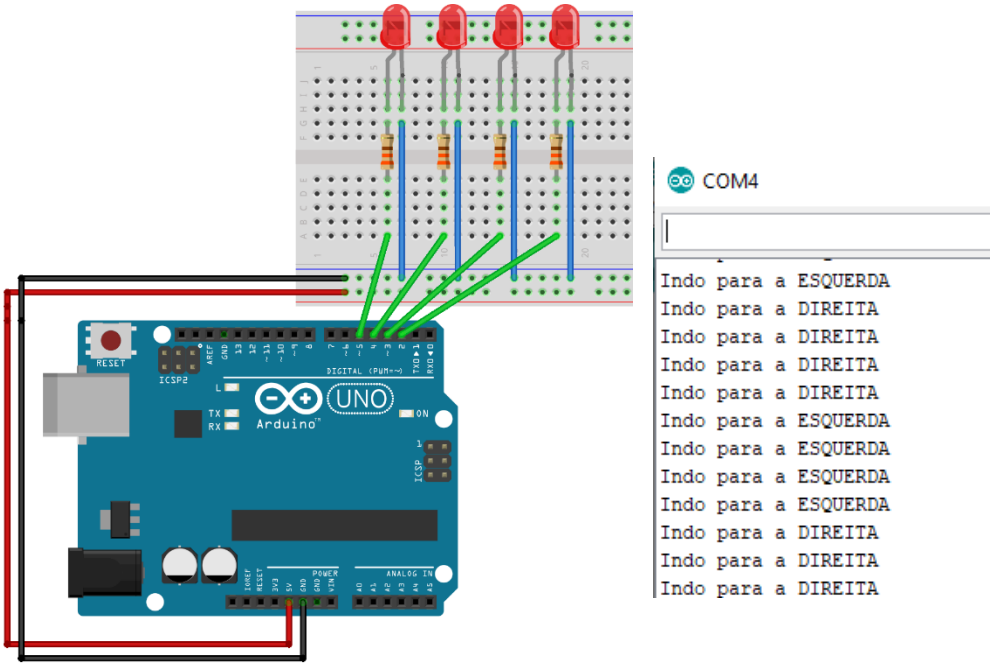


Figura 2.10: Montagem do exemplo e a respectiva saída no Monitor Serial.

Exercício de Fixação

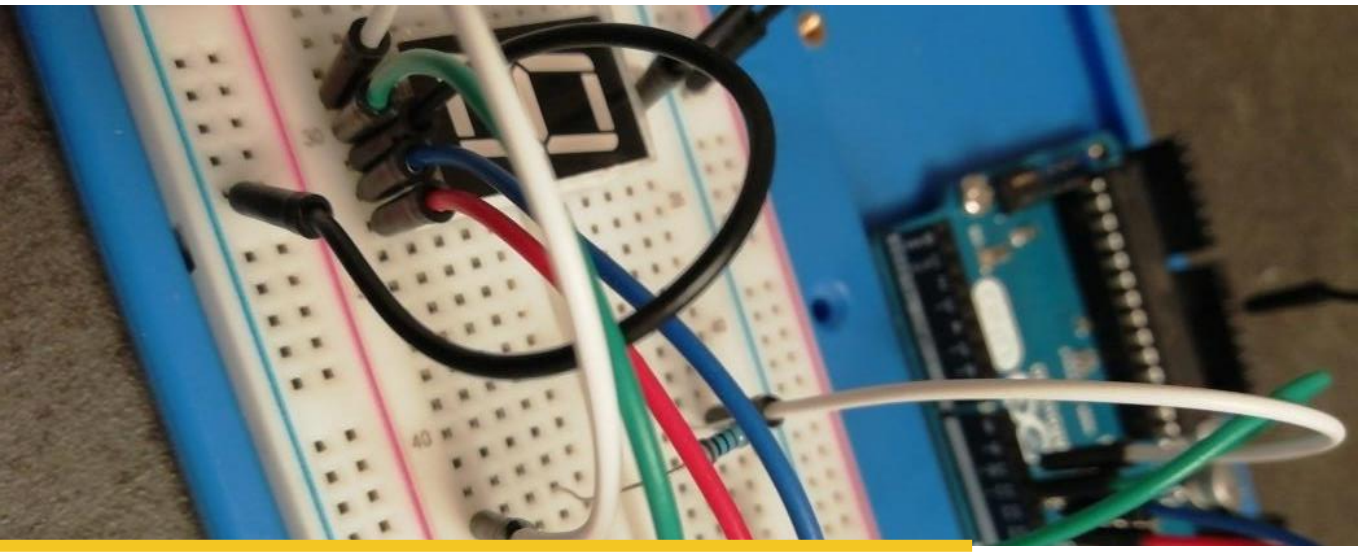
Para estes exercícios, tente utilizar as estruturas de programação que você conhece para melhorar o seu código. Tenha a liberdade de ampliar seus projetos utilizando outras funções consultando a documentação de referência em https://www.arduino.cc/reference/pt/.

1) Com 4 ou mais leds, faça um projeto com que os leds nos pinos pares acendam, enquanto os leds dos pinos ímpares apaguem. Em seguida, os dos pinos ímpares acendem e o dos pinos pares apagam.
2) Para treinar uma implementação mais realista, tente simular um semáforo com três leds vermelho, amarelo e verde atermando ordenadamente assim como um semáforo de verdade.
3) A função random(num_max) gera um número aleatório entre zero e o número entre parêntesis. Assim random(1000) gera um número aleatório entre 0 e 999. No simulador de semáforo substitua delay (num) pela instrução delay (random(1000)).

4) Com 4 ou mais leds, faça um projeto de modo que todos os leds acendam, um após o outro, da esquerda para a direita, de modo que fiquem todos acesos. Após isso, os leds deverão apagar nesta mesma sequência, deixando agora todos apagados. Agora o mesmo deve acontecer da direita para a esquerda.
5) Com um número par de leds, faça um projeto de modo que acendam um após o outro do centro para as extremidades, ficando todos acesos; e apaguem das extremidades para o centro, ficando todos apagados ao final.
6) Utilize a função analogWrite() para dimerizar um LED. A dimerização consiste em controlar o brilho de uma fonte de luz. Você pode testar com vários valores entre 0 e 1024, tente acender o led com vários níveis de tensão diferentes.
7) Agora que você conseguiu dimerizar um LED, faça novamente aquele projeto blink, mas agora o led deve ligar e desligar suavemente, e não apenas acender e apagar repentinamente.
8) Para situações de emergência no mar, barcos salva vidas recebem uma lâmpada de alerta que pisca de forma intermitente um sinal de SOS em código Morse. O código Morse foi criado por Samuel Morse, inventor do telégrafo, em 1835 para servir de base para comunicação à distância. Através do telégrafo é possível mandar um conjunto de sinais elétricos curtos e longos que podem ser ouvidos. Morse estabeleceu um código que faz corresponder a cada letra, algarismos e símbolos de pontuação textual, um conjunto de pontos (sinal curto) e traços (sinal longo) e espaços. O sinal SOS é usado para comunicar uma emergência. No código Morse a letra S é representada por três sinais curtos (pontos) e o O por três sinais longos (traços) e assim o sinal de SOS é "...---...". Essa sequência pode ser comunicada com luz e é exatamente isso que faz a lâmpada do bote salva vidas. Monte um código que crie uma LED com sinal SOS.
9) Dado o código abaixo, explique o que ele faz e quais abordagens esse código poderia ser utilizado:

```
void setup() {
  pinMode(3, OUTPUT);

  pinMode(7, OUTPUT);
  pinMode(10, OUTPUT);
}

void loop() {
  digitalWrite(3,HIGH);
  delay(3000);

  digitalWrite(3,LOW);
  delay(1000);

  digitalWrite(7,HIGH);
  delay(5000);

  digitalWrite(7,LOW);
  delay(1000);

  digitalWrite(10,HIGH);
  delay(7000);

  digitalWrite(10,LOW);
  delay(1000);
}
```

10) Imagine-se em dezembro, perto do natal, e sua família esqueceu de comprar as luzes pisca-pisca, para a árvore. Faça um projeto com várias sequencias de pisca-pisca tendo 10 LEDs para controlar. Fique à vontade para usar sua criatividade para fazer as sequências que desejar para sua árvore de natal!



Capítulo 3: Botões e Dispositivos Sinalizadores

Para a criação de projetos utilizando o Arduino, é necessário saber como funcionam elementos externos pertencentes ao circuito. Neste capítulo, iremos estudar três componentes que possuem inúmeras aplicações: botão, display de 7 segmentos e display LCD. Adiante, veremos como manipulá-los através do código e como configurar suas montagens em projetos.

3.1. Botão

O botão é um elemento que já estamos familiarizados no dia a dia. Ao apertar determinado botão de seu celular, por exemplo, é possível ligar ou desligar o aparelho, reiniciá-lo, aumentar ou diminuir o volume, ou até mesmo só iniciar a tela principal dependendo de qual botão foi pressionado. Isso implica que o botão é um elemento que, ao ser pressionado, executa o comando a que ele está associado.

3.1.1. Funcionamento

Em primeiro momento, devemos observar como é o funcionamento de um botão. Para esse caso, utilizaremos um elemento simples com 4 pinos de contato, como mostrado na Figura 3.1. É notável que dois pinos (A) são a mesma ligação, implicando que eles irão possuir o mesmo nível lógico, enquanto os dois outros pinos (B), também ligados, deverão possuir o nível lógico oposto. A chave aberta

representa o botão como si. No momento que o botão é pressionado, a chave fecha, permitindo a passagem de corrente de um lado para outro (A para B ou B para A, dependendo da polarização aplicada).

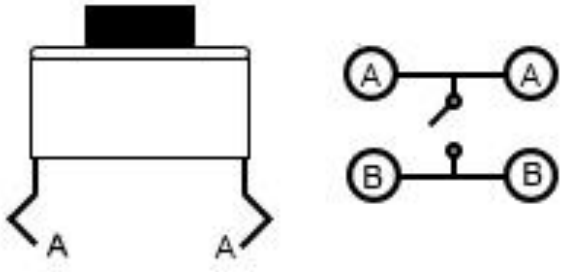


Figura 3.1: Visão Lateral e Interior do Botão.

3.1.2. Aplicações

O botão pode ser conectado de dois modos: pressionando para obter uma entrada de nível lógico alto (em 5V), mais conhecido como configuração no modo pull-down; pressionando para obter uma entrada de nível lógico baixo (em 0V), conhecido também como conexão no modo pull-up. O modo de conexão depende do tipo de objetivo desejado e da preferência do projetista.

a) Configuração no modo Pull-Down

Como falado anteriormente, o modo pull-down é a configuração em que ao pressionar o botão, obtêm-se uma entrada de valor digital alto. Para isso, precisamos configurar que o circuito só receba valor digital alto quando o botão for apertado.

Relembrando a análise do botão, a chave vai controlar a passagem de corrente que estiver indo para a entrada do arduino. Desse modo, considerando o lado 'A' como estando conectado à entrada, ele deve estar conectado ao GND (0V) do circuito, assim, em todos os momentos, o circuito vai estar recebendo nível lógico baixo. No momento que pressionarmos o botão, o circuito de entrada recebe um nível lógico alto, o que implica que a parte 'B' está ligada no VCC (5V). Contudo, para que a passagem do valor digital alto seja feita prioritariamente, ela deve

apresentar uma resistência menor que a de 0V. Desse modo, é necessário que coloquemos uma resistência de 1kΩ entre a conexão A-GND. Essa resistência servirá também para o botão não entrar em curto. Adiante, para proteção da entrada, coloca-se uma resistência de 330Ω, como mostrado no esquemático abaixo. Deixa-se claro que o projetista pode escolher outros valores de resistência, contanto que a do ground seja maior que a do VCC. O código foi feito com base na análise do circuito.

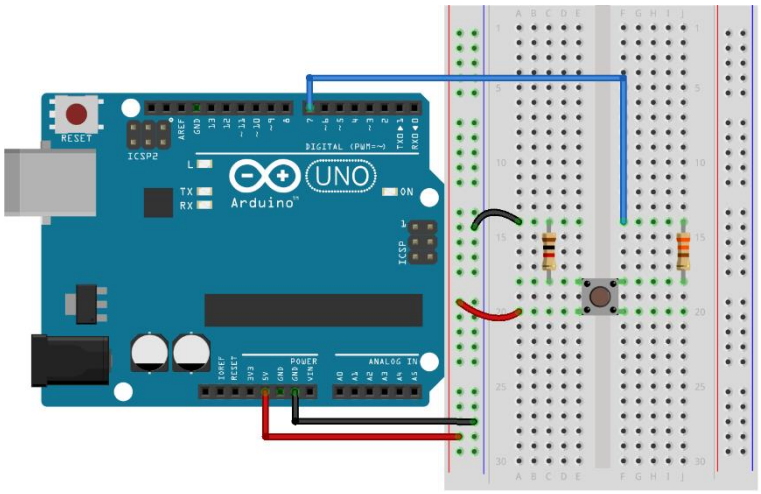


Figura 3.2: Diagrama Elétrico

```
int botao = 7;   // número do pino do botão
int i = 0;       // variavel auxiliar de contagem
int valor = 0;
void setup() {
  // Configura-se o botão como entrada
  pinMode(botao, INPUT);
  Serial.begin(9600);
}
void loop() {
  // Faz-se a leitura do nível lógico do botão
  valor = digitalRead(botao);
  delay(100);
  if(valor != LOW){
    Serial.print(i++);
    Serial.print("-Valor:");
    // Imprime na tela o valor do nível lógico
    Serial.println(valor);
  }
}
```

b) Configuração no modo Pull-Up

O modo pull-up é a configuração em que ao pressionar o botão, obtêm-se uma entrada de 0V. Aproveitando a montagem e código anterior, deve-se inverter os valores de nível lógico. Portanto, o que antes era conectado no VCC agora deve ser conectado no GND e vice-versa.

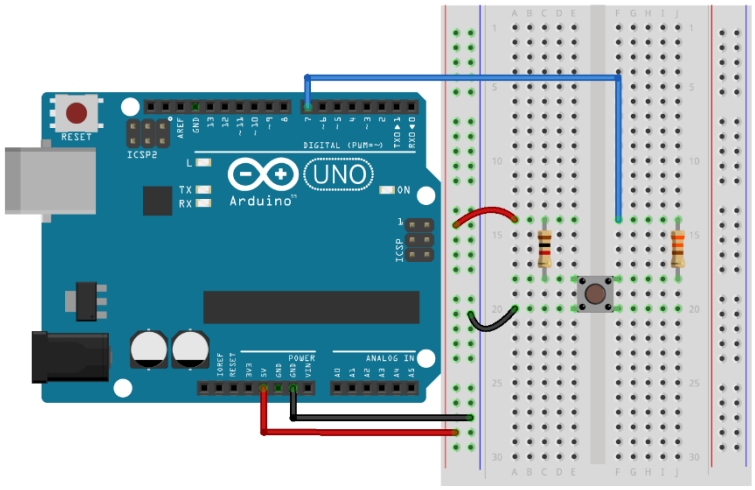


Figura 3.3: Diagrama Elétrico

```
int botao = 7;  // número do pino do botão
int i = 0;      // variavel auxiliar de contagem
int valor = 0;
void setup() {
  pinMode(botao, INPUT);  // Configurando o botão como
entrada
  Serial.begin(9600);
}
void loop(){
  // Faz-se a leitura do nível lógico do botão
  valor = digitalRead(botao);
  delay(100);
  if(valor != HIGH){
    Serial.print(i++);
    Serial.print("-Valor:");
    // Imprime na tela o valor do nível lógico
    Serial.println(valor);
  }
}
```

Exercício de Fixação

1) Faça um programa para acionar um LED controlado por um botão no modo Pull-Up. DICA.: O LED deve ficar aceso se o botão estiver pressionado, isto é, no momento em que o valor recebido pelo pino do botão for nível lógico alto o led acende.
2) Faça um programa de um Semáforo (3 LEDs) controlado por um botão no modo Pull-Down, em que inicialmente o LED verde está aceso. Quando o botão for pressionado, o led amarelo deve acender, informando aos carros que um pedestre está prestes a passar e em seguida o led vermelho deve ficar acesso por um tempo antes do semáforo voltar a ficar verde.
3) Faça um programa utilizando três botões para controlar apenas um led. O primeiro botão serve apenas para ligar o led, o segundo botão serve apenas para desligar o led e o terceiro irá alterar o estado do led (se estiver ligado, apaga; se estiver desligado, liga).
4) Agora tendo 3 LEDs e 3 push buttons, monte um código que cada botão irá acionar um único LED quando for pressionado, e irá desliga-lo caso contrário.
5) Una o projeto do blink com um botão, alternando o estado do led entre piscando e desligado sempre que o botão for pressionado.
6) Agora utilize o botão para alternar várias frequências do blink. Defina no mínimo 3 frequências e cada vez que o botão for pressionado, a frequência de blink é alterada.
7) Utilize um botão para controlar o nível de brilho de um led da seguinte forma: segurar o botão faz com que brilho o vá se alterando lentamente; quando o botão for solto, o brilho irá continuar naquele nível.

3.2. Display de 7 Segmentos de 1 Dígito

Um típico display de 7 segmentos de 1 dígito é um dispositivo que contém 7 segmentos de LEDs embutidos, além de um LED adicional para representar um ponto decimal. Com a liberdade de escolher quais segmentos são acesos, é possível mostrar desde os números de 0 a 9 até algumas letras ou outras formas à escolha.

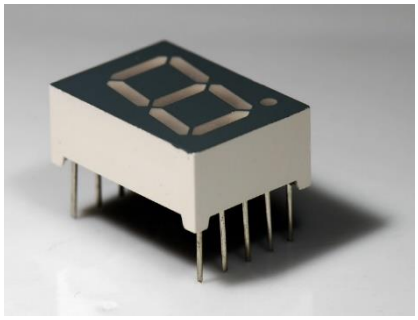


Figura 3.4: Display de 7 segmentos de 1 dígito.

Nessa seção você aprenderá sobre o seu funcionamento e alguns exemplos de aplicações, utilizando vetores (arrays) para armazenar valores e botões para realizar contagens. Caso você já tenha conhecimento sobre como esse display funciona, sinta-se livre para pular para a subseção de aplicações 3.2.2. Contudo, ao realizar as montagens, ainda é recomendável usar como referência o diagrama de pinos do display, que se encontra na próxima subseção.

3.2.1. Funcionamento

O display de 7 segmentos de 1 dígito possui dez pinos, dos quais oito são para os segmentos nomeados de *a* a *g* e o ponto decimal *dp*, e os dois restantes são os pinos comuns. Há dois tipos de displays de 7 segmentos: **ânodo comum** e **cátodo comum**. A estrutura interna dos dois é bastante semelhante, diferindo, em suas montagens, apenas onde os pinos comuns são conectados e, no código, apenas nos níveis lógicos enviados para os pinos dos segmentos.

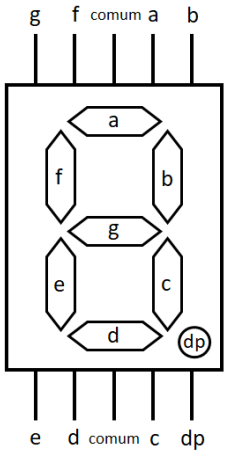


Figura 3.5: Diagrama de pinos do display de 7 segmentos.

No display cátodo comum, os pinos comuns devem ser conectados ao GND, e para acender um segmento

envia-se nível lógico alto para o pino correspondente. Isso se deve pelos LEDs terem seus terminais positivos (ânodos) conectados aos pinos dos segmentos e os terminais negativos (cátodos) conectados aos pinos comuns.

O display ânodo comum funciona de forma oposta: conectam-se os pinos comuns ao 5V e envia-se nível lógico baixo para os segmentos que se deseja acender.

3.2.2. Aplicações

Para os exemplos dessa subseção, será utilizado um display cátodo comum.

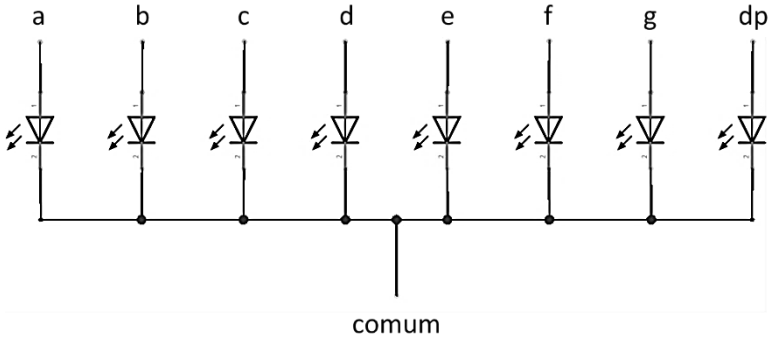


Figura 3.7: Circuito interno do display de 7 segmentos cátodo comum.

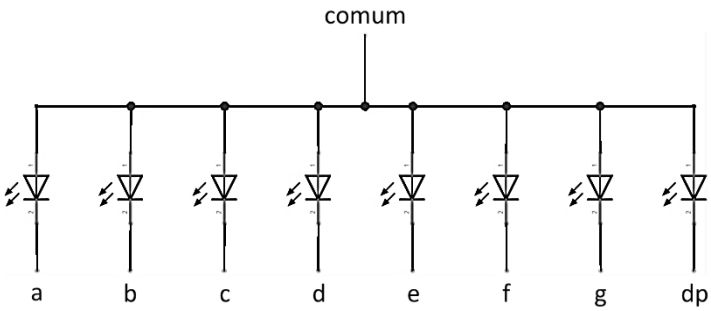


Figura 3.6: Circuito interno do display de 7 segmentos ânodo comum.

Isso significa que para acender um segmento desse display, precisamos enviar nível lógico alto (`HIGH` ou 1) para seu pino, e nível lógico baixo (`LOW` ou 0) para apagá-lo. Dessa forma, todos os códigos começarão com as seguintes diretivas:

```
// diretivas para o display cátodo comum
#define ON HIGH
#define OFF LOW
```

Isso quer dizer que sempre que usarmos ON em nosso código, esse será substituído por HIGH; da mesma forma, qualquer OFF será substituído por LOW.

Entretanto, se você estiver usando um display ânodo comum, os níveis lógicos que você envia para os segmentos se invertem. Então basta trocar as definições por estas:

```
// diretivas para o display cátodo comum
#define ON LOW
#define OFF HIGH
```

Ou seja, em todos os códigos, usaremos ON sempre que quisermos acender um segmento e OFF sempre que quisermos apagar um segmento, independentemente do tipo do seu display. Sempre que você precisar trocar o tipo de display, basta mudar essas definições.

Ao carregar qualquer um dos códigos, certifique-se de que as diretivas `#define` estão adaptadas ao seu display. Se não estiverem de acordo, o display trocará os segmentos acesos pelos apagados e vice-versa.

Nos exemplos de a) a c), será feita a mesma montagem (figura 3.8). Utilizaremos as portas 2 a 9 do Arduino, conectadas aos segmentos de *a* a *dp*, nessa ordem, juntamente com resistores de 330Ω. Lembre-se que, para o display cátodo comum, os pinos comuns são conectados ao GND, e os do ânodo comum são conectados ao 5V. Caso seu display seja ânodo comum, utilize resistores para conectar os pinos comuns ao 5V.

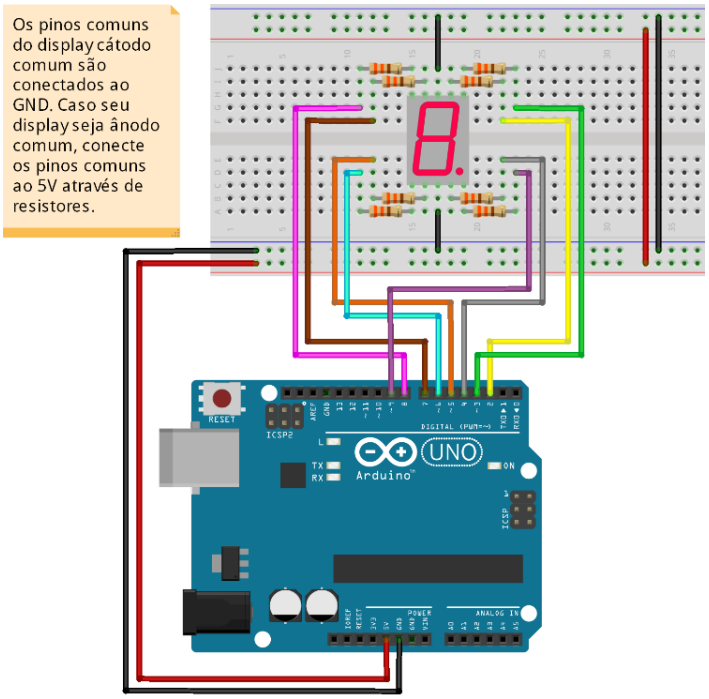


Figura 3.8: Montagem para o display de 7 segmentos cátodo comum.

a) Acendendo e Apagando os Segmentos

Para nos familiarizarmos com o funcionamento do display, acenderemos todos os seus segmentos, com um pequeno intervalo de tempo entre cada um, e em seguida os apagaremos da mesma forma.

Iremos configurar, dentro da função `setup()`, as portas 2 a 9 como saída, e nelas enviaremos OFF, garantindo que o display inicie apagado. Na função `loop()`, utilizaremos uma estrutura *for* para acender os segmentos do display e, em seguida, de forma análoga, apagaremos os segmentos.

```
// diretivas para o display cátodo comum
#define ON HIGH
#define OFF LOW

void setup() {
  // definimos pinos de 2 a 9 como saída e
  // apagamos o display
  for (int pino = 2; pino <= 9; pino++) {
    pinMode(pino, OUTPUT);
    digitalWrite(pino, OFF);
  }
}

void loop() {
  // loop para acender os segmentos
  for (int pino = 2; pino <= 9; pino++) {
    // a cada iteração, acende o segmento
    // do pino correspondente
    digitalWrite(pino, ON);
    // espera meio segundo
    delay(500);
  }

  // loop para apagar os segmentos
  for (int pino = 2; pino <= 9; pino++) {
    // a cada iteração, apaga o segmento
    // do pino correspondente
    digitalWrite(pino, OFF);
    // espera meio segundo
    delay(500);
  }
}
```

b) Piscando um número no display

Neste exemplo, exibiremos e apagaremos o número 0 repetidamente, fazendo-o piscar no display. O código da função `setup()` é quase o mesmo do exemplo anterior, com a diferença de que não é mais necessário apagar os segmentos.

Para exibir o número 0 no display, acendemos os segmentos de *a* a *f* e apagamos *g* e *dp*. Você poderia usar um código como o seguinte para tal objetivo:

```
digitalWrite(2, ON);
digitalWrite(3, ON);
digitalWrite(4, ON);
digitalWrite(5, ON);
digitalWrite(6, ON);
digitalWrite(7, ON);
digitalWrite(8, OFF);
digitalWrite(9, OFF);
```

Entretanto, é interessante fazermos um código mais dinâmico, que sirva para quando queiramos mostrar vários números diferentes no display mais tarde. Assim, guardaremos os níveis lógicos dos segmentos do número em um vetor:

```
bool zero[] = {ON, ON, ON, ON, ON, ON, OFF, OFF};
```

Declararemos esse vetor como uma variável global, pois assim ele será declarado uma única vez e será acessível pelo escopo da função `loop()`. Note que, apesar do vetor ser do tipo `bool`, os valores que estamos armazenando nele são `HIGH` e `LOW`. Isso é possível porque `HIGH` e `LOW` na verdade são convertidos pelo pré-processador do Arduino para 1 e 0, respectivamente.

No Arduino, valores como `HIGH`, 1 e `true` são equivalentes, assim como `LOW`, 0 e `false` são equivalentes. Todos esses valores podem ser atribuídos ao tipo `bool` (e também ao tipo `int`) e podem ser usados como segundo parâmetro da função `digitalWrite()`. Ou seja, a escolha de qual desses valores usamos baseia-se apenas na semântica.

Então, para exibir o número, iremos percorrer o vetor usando uma estrutura for:

```
// índices do vetor vão de 0 a 7
for (int i = 0; i <= 7; i++) {
  // pinos vão de 2 a 9
  int pino = i + 2;
  // a cada iteração, escreve OFF ou ON
  // no segmento do pino correspondente
  digitalWrite(pino, zero[i]);
}
// mantém o número aceso por 200ms
delay(200);
```

Nesse loop, a cada iteração, um valor do vetor será acessado por vez e será escrito no pino correspondente. Por exemplo, na primeira iteração, i = 0, pino = 2 e zero[i] = zero[0], que corresponde ao valor da primeira posição do vetor, que nesse caso é ON; logo, acenderemos o segmento conectado ao pino 2 (segmento *a*). Na segunda iteração, i = 1, pino = 3 e zero[i] = zero[1], cujo valor corresponde a ON; acenderemos o segmento *b*. E assim por diante, até a última iteração onde i = 7, pino = 9 e zero[i] = zero[7]; apagaremos o segmento *dp*.

Por fim, uma vez que o número 0 seja exibido, aplicamos um `delay(200)`, o mantendo aceso por 200ms. Em seguida, apagamos o display com outra estrutura *for* e aplicamos mais um `delay(200)`, o mantendo apagado por 200ms.

```
// diretivas para o display cátodo comum
#define ON HIGH
#define OFF LOW

// declaramos o vetor como uma variável global
bool zero[] = {ON, ON, ON, ON, ON, ON, OFF, OFF};

void setup() {
  // definimos pinos de 2 a 9 como saída
  for (int pino = 2; pino <= 9; pino++) {
   pinMode(pino, OUTPUT);
  }
}

void loop() {
```

```
  // exibindo o número
  for (int i = 0; i <= 7; i++){
    // índices do vetor vão de 0 a 7
    // pinos vão de 2 a 9
    int pino = i + 2;
    // a cada iteração, escreve OFF ou ON
    // no segmento do pino correspondente
    digitalWrite(pino, zero[i]);
  }
  // mantém o número aceso por 200ms
  delay(200);

  // apagando o display
  for (int pino = 2; pino <= 9; pino++) {
    // a cada iteração, escreve OFF no pino correspondente
    digitalWrite(pino, OFF);
  }
  // mantém o número apagado por 200ms
  delay(200);
}
```

Exercício de Fixação

1) Brinque um pouco com o display e escolha outros números para piscarem.
2) Tente antecipar o próximo exemplo: faça um contador com pelo menos 3 números, exibindo cada um durante 1 segundo antes de exibir o próximo. Não há problema caso o código fique extenso.
3) Faça um contador aleatório de 0 a 9, ou seja, ele deve fazer a contagem de modo aleatório. Regra: número anterior não pode ser igual ao novo número. Para isso, use a instrução random(10) que escolhe aleatoriamente um número de 0 a 9.
4) O display de 7 segmentos também pode ser usado para exibir códigos hexadecimais por meio das letras A, b, C, d, E, F. Tente exibir apenas essas seis letras no display de 7 segmentos.
5) Pesquisa! Você deve ter percebido que para acender o display de 7 segmentos utiliza-se uma grande quantidade de pinos do Arduino, deixando-o bem ocupado. Existe uma forma de acender vários displays de 7 segmentos através

de circuitos integrados (CI), pequenos chips que facilitam o uso dos displays de 7 segmentos. Pesquise sobre o CI CD4511 e como ele pode ser usado para controlar um ou mais displays de 7 segmentos utilizando apenas 4 pinos do Arduino ao invés de 7.

6) Desafio! Na questão 2, se você usou os mesmos métodos que utilizamos até agora, seu código provavelmente está começando a ficar extenso e repetitivo. Tente otimizá-lo e deixá-lo mais dinâmico tirando proveito de outras ferramentas da linguagem C, como funções e a estrutura switch. Dica: no próximo exemplo, utilizamos uma variável global de contagem que define o número que será exibido no display. Criamos uma função auxiliar que recebe como parâmetro o valor da variável de contagem. Por fim, dentro da função utilizamos a estrutura switch para escolher o vetor correspondente ao número.

c) Contador Automatizado de 0 a 9

Vamos mostrar todos os números no display! Faremos um contador que inicia em 0, vai até 9 e reinicia a contagem automaticamente. O código da função `setup()` é o mesmo do exemplo anterior.

Primeiramente precisamos guardar os valores dos segmentos de todos os números.

```
// declaramos os vetores globalmente
bool zero[]   = {ON, ON, ON, ON, ON, ON, OFF, OFF};
bool um[]     = {OFF, ON, ON, OFF, OFF, OFF, OFF, OFF};
bool dois[]   = {ON, ON, OFF, ON, ON, OFF, ON, OFF};
bool tres[]   = {ON, ON, ON, ON, OFF, OFF, ON, OFF};
bool quatro[] = {OFF, ON, ON, OFF, OFF, ON, ON, OFF};
bool cinco[]  = {ON, OFF, ON, ON, OFF, ON, ON, OFF};
bool seis[]   = {OFF, OFF, ON, ON, ON, ON, ON, OFF};
bool sete[]   = {ON, ON, ON, OFF, OFF, OFF, OFF, OFF};
bool oito[]   = {ON, ON, ON, ON, ON, ON, ON, OFF};
bool nove[]   = {ON, ON, ON, OFF, OFF, ON, ON, OFF};
```

Definiremos agora uma variável global de contagem, que definirá qual número será exibido no display. Inicializaremos seu valor em 0.

```
int cont = 0;
```

Mas agora, como usar o valor da variável para exibir o número no display? Independentemente do número, queremos ler o valor de cont, acessar o vetor correspondente e escrever os níveis lógicos nos segmentos no display. Podemos criar uma função auxiliar para isso. Vamos criar a função `mostrar_numero()`, que recebe um número como parâmetro.

```
void mostrar_numero(int num) {
  // índices do vetor vão de 0 a 7
  for (int i = 0; i <= 7; i++){
    // pinos vão de 2 a 9
    int pino = i + 2;
```

No código da nossa função, teríamos a mesma estrutura *for* do exemplo anterior, que lia os valores de um vetor e exibia o número no display. Entretanto, agora precisamos escolher qual vetor queremos ler, dependendo do número que a

função recebe como parâmetro. A estrutura switch é capaz de resolver nosso problema.

```
    switch(num) {
      case 0: digitalWrite(pino, zero[i]); break;
      case 1: digitalWrite(pino, um[i]); break;
      case 2: digitalWrite(pino, dois[i]); break;
      case 3: digitalWrite(pino, tres[i]); break;
      case 4: digitalWrite(pino, quatro[i]); break;
      case 5: digitalWrite(pino, cinco[i]); break;
      case 6: digitalWrite(pino, seis[i]); break;
      case 7: digitalWrite(pino, sete[i]); break;
      case 8: digitalWrite(pino, oito[i]); break;
      case 9: digitalWrite(pino, nove[i]); break;
    }
  }
}
```

Pronto! Quando chamarmos a função passando como parâmetro o número 0, por exemplo, a única instrução do switch que será executada durante todo o loop é `digitalWrite(pino, zero[i])`. Da mesma forma, se passarmos o número 5 como parâmetro, apenas a instrução `digitalWrite(pino, cinco[i])` do switch será executada, e assim por diante.

```
// Diretivas para o display cátodo comum
#define ON HIGH
#define OFF LOW

// declaramos os vetores globalmente
bool zero[]   = {ON,  ON,  ON,  ON,  ON,  ON,  OFF, OFF};
bool um[]     = {OFF, ON,  ON,  OFF, OFF, OFF, OFF, OFF};
bool dois[]   = {ON,  ON,  OFF, ON,  ON,  OFF, ON,  OFF};
bool tres[]   = {ON,  ON,  ON,  ON,  OFF, OFF, ON,  OFF};
bool quatro[] = {OFF, ON,  ON,  OFF, OFF, ON,  ON,  OFF};
bool cinco[]  = {ON,  OFF, ON,  ON,  OFF, ON,  ON,  OFF};
bool seis[]   = {OFF, OFF, ON,  ON,  ON,  ON,  ON,  OFF};
bool sete[]   = {ON,  ON,  ON,  OFF, OFF, OFF, OFF, OFF};
bool oito[]   = {ON,  ON,  ON,  ON,  ON,  ON,  ON,  OFF};
bool nove[]   = {ON,  ON,  ON,  OFF, OFF, ON,  ON,  OFF};
```

Agora, na função `loop`, precisamos apenas chamar `mostrar_numero()`, passando a variável `cont` como parâmetro. Uma vez exibido o número no display, incrementamos o valor de `cont` (somamos 1), assim nossa função mostrará o

próximo número na próxima iteração de `loop()`. E finalmente, não podemos esquecer de reiniciar a contagem caso `cont` chegue a 10.

```
// variável global de contagem
int cont = 0;
// função que recebe um número como parâmetro
// e o exibe no display
void mostrar_numero(int num) {
  // índices do vetor vão de 0 a 7
  for (int i = 0; i <= 7; i++){
    // pinos vão de 2 a 9
    int pino = i + 2;

    switch(num) {
      case 0: digitalWrite(pino, zero[i]); break;
      case 1: digitalWrite(pino, um[i]); break;
      case 2: digitalWrite(pino, dois[i]); break;
      case 3: digitalWrite(pino, tres[i]); break;
      case 4: digitalWrite(pino, quatro[i]); break;
      case 5: digitalWrite(pino, cinco[i]); break;
      case 6: digitalWrite(pino, seis[i]); break;
      case 7: digitalWrite(pino, sete[i]); break;
      case 8: digitalWrite(pino, oito[i]); break;
      case 9: digitalWrite(pino, nove[i]); break;
    }
  }
}

void setup() {
  // definimos os pinos de 2 a 9 como saída
  for (int pino = 2; pino <= 9; pino++) {
    pinMode(pino, OUTPUT);
  }
}

void loop() {
  // exibe o número no display baseado no valor de cont
  mostrar_numero(cont);
  // incrementa o valor de cont
  cont++;
  // se o novo valor de cont for 10, reiniciamos a contagem
  if (cont == 10){
    cont = 0;
  }
  // número é exibido por 1 segundo antes de mostrar o próximo
  delay(1000);
}
```

OBS: Há, ainda, outra forma de acessar os diferentes vetores, os guardando em uma matriz (vetor bidimensional). Troque as declarações dos vetores por esta:

```
bool numeros[10][8] = {
  {ON,  ON,  ON,  ON,  ON,  ON,  OFF, OFF},  // 0
  {OFF, ON,  ON,  OFF, OFF, OFF, OFF, OFF},  // 1
  {ON,  ON,  OFF, ON,  ON,  OFF, ON,  OFF},  // 2
  {ON,  ON,  ON,  ON,  OFF, OFF, ON,  OFF},  // 3
  {OFF, ON,  ON,  OFF, OFF, ON,  ON,  OFF},  // 4
  {ON,  OFF, ON,  ON,  OFF, ON,  ON,  OFF},  // 5
  {OFF, OFF, ON,  ON,  ON,  ON,  ON,  OFF},  // 6
  {ON,  ON,  ON,  OFF, OFF, OFF, OFF, OFF},  // 7
  {ON,  ON,  ON,  ON,  ON,  ON,  ON,  OFF},  // 8
  {ON,  ON,  ON,  OFF, OFF, ON,  ON,  OFF},  // 9
};
```

E troque toda a estrutura switch da função `mostrar_numero()` por esta única linha, utilizaremos ela no próximo exemplo:

```
digitalWrite(pino, numeros[num][i]);
```

Exercícios de Fixação

1) Adicione mais exibições ao seu display: depois do 9, mostre as letras de A a F (as letras b e d ficam minúsculas) no contador. O que você irá exibir agora no display é o sistema hexadecimal, onde as letras A a F representam os números de 10 a 15.
2) Faça um contador aleatório pelo sistema hexadecimal, de forma que, ele deve fazer a contagem de modo aleatório. Regra: número anterior não pode ser igual ao novo número. Para isso, use a instrução random(16) que escolhe aleatoriamente um número de 0 a 15.
3) Sempre que o número for alterado, faça o monitor serial imprimir a mensagem "Exibindo o valor *X*", onde *X* é o valor de cont. Utilize as funções **Serial**.print() e **Serial**.println() para exibir a mensagem.

d) Contador Manual de 0 a 9

Nesse exemplo, evoluiremos nossa montagem ao utilizar um botão para incrementar o valor exibido no display. Utilizaremos o botão no modo pull-down, conectado ao pino 12. Aproveitaremos para utilizar o monitor serial para exibir quando o botão for pressionado e quando for solto.

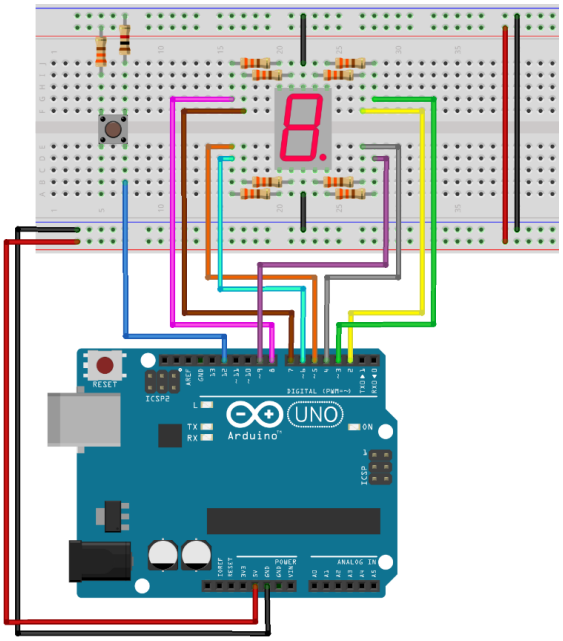


Figura 3.9: Montagem do exemplo d)

Primeiro, vamos reutilizar o código do exemplo anterior, com exceção da função `loop()`. Definiremos uma variável global para o pino do botão. Adicionalmente na função `setup()`, devemos inicializar o monitor serial, definir o pino do botão como entrada e inicializar o display com o valor inicial de `cont`.

O código do botão será mais complexo do que aquele que utilizamos anteriormente, onde apenas precisávamos ler o estado do botão diretamente e informá-lo no monitor serial. Desta vez, precisamos detectar quando o botão é pressionado e guardar o número de vezes que isso acontece na variável `cont`.

Criaremos uma variável global `ultimoEstadoBotao` que iniciará com o valor `LOW`. Na função `loop()`, leremos o estado do botão e o guardaremos em `estadoAtualBotao`, cujo valor será atualizado constantemente. Em seguida, na condição de um bloco *if*, compararemos essas duas variáveis para detectar se o

estado do botão foi alterado. Se a condição for realizada, atualizamos a variável `ultimoEstadoBotao`.

```
bool ultimoEstadoBotao = LOW;

void loop() {
  // lê o estado do botão constantemente
  bool estadoAtualBotao = digitalRead(pinoBotao);

  // verifica se o estado do botão mudou
  if (estadoAtualBotao != ultimoEstadoBotao) {
    // se mudou, atualiza ultimoEstadoBotao
    ultimoEstadoBotao = estadoAtualBotao;

    Serial.print("Estado alterado! Nível lógico atual: ");
    Serial.println(estadoAtualBotao);

    // delay para evitar bouncing do botão
    delay(50);
  }
}
```

Por exemplo, digamos que o programa seja iniciado. Se o botão for pressionado, `estadoAtualBotao` será atualizado para `HIGH` e a condição do bloco *if* será realizada. `ultimoEstadoBotao` será atualizado para `HIGH` e a mensagem será mostrada no monitor serial. Enquanto o botão não for solto, a condição de *if* não será realizada novamente, pois `estadoAtualBotao` e `ultimoEstadoBotao` agora são ambos `HIGH`. Analogamente, quando o botão for solto, `estadoAtualBotao` será atualizado para `LOW`, a condição do *if* será realizada novamente e o bloco será executado, atualizando `ultimoEstadoBotao` para `LOW` também.

Antes do final da execução do bloco *if*, também é necessário aplicarmos um delay de pelo menos 50ms a fim de evitarmos o *bouncing* do botão (esse efeito será melhor explicado posteriormente).

Com essa estrutura já somos capazes de atualizar o display. Dentro do bloco *if*, agora devemos apenas verificar se o estado do botão foi alterado para `HIGH`,

ou seja, se foi pressionado. Em caso afirmativo, incrementamos `cont` e mostramos o número no display.

```
// diretivas para o display ânodo comum
#define ON LOW
#define OFF HIGH

bool numeros[10][8] = {
  {ON, ON, ON, ON, ON, ON, OFF, OFF},        // 0
  {OFF, ON, ON, OFF, OFF, OFF, OFF, OFF},    // 1
  {ON, ON, OFF, ON, ON, OFF, ON, OFF},       // 2
  {ON, ON, ON, ON, OFF, OFF, ON, OFF},       // 3
  {OFF, ON, ON, OFF, OFF, ON, ON, OFF},      // 4
  {ON, OFF, ON, ON, OFF, ON, ON, OFF},       // 5
  {OFF, OFF, ON, ON, ON, ON, ON, OFF},       // 6
  {ON, ON, ON, OFF, OFF, OFF, OFF, OFF},     // 7
  {ON, ON, ON, ON, ON, ON, ON, OFF},         // 8
  {ON, ON, ON, OFF, OFF, ON, ON, OFF},       // 9
};

int pinoBotao = 12;
int cont = 0;
bool ultimoEstadoBotao = LOW;

// função que recebe um número como parâmetro
// e o exibe no display
void mostrar_numero(int num) {
  // índices do vetor vão de 0 a 7
  for (int i = 0; i <= 7; i++) {
    // pinos vão de 2 a 9
    int pino = i + 2;
    // acessa o segmento (i) do vetor do número (num)
    // e escreve no pino correspondente
    digitalWrite(pino, numeros[num][i]);
  }
}

void setup() {
  Serial.begin(9600);

  // definimos os pinos de 2 a 9 como saída
  for (int pino = 2; pino <= 9; pino++) {
    pinMode(pino, OUTPUT);
  }
```

```
  // definimos o pino do botão como entrada
  pinMode(pinoBotao, INPUT);
  // iniciamos o display no número 0
  mostrar_numero(0);
}

void loop() {
  // lê o estado do botão constantemente
  bool estadoAtualBotao = digitalRead(pinoBotao);

  // verifica se o estado do botão mudou
  if (estadoAtualBotao != ultimoEstadoBotao) {
    // se mudou, atualiza ultimoEstadoBotao
    ultimoEstadoBotao = estadoAtualBotao;

    // verifica se o novo estado é nível alto
    if (estadoAtualBotao == HIGH) {
      cont++;
      if (cont == 10) {
        cont = 0;
      }

      Serial.print("Botão pressionado! Valor de cont: ");
      Serial.println(cont);
      mostrar_numero(cont);
    } else {
      Serial.println("Botão solto!");
    }
    // delay para evitar bouncing do botão
    delay(50);
  }
}
```

OBS: o *boucing* é um efeito que pode ocorrer quando o botão é pressionado ou solto e que ocorre devido às próprias limitações da estrutura física do botão. Esse efeito faz com que o circuito do botão seja fechado e aberto muito rapidamente (durante alguns milissegundos), fazendo com que o código interprete que o botão foi pressionado mais de uma vez. Nesse último exemplo realizamos o *debounce* (anulação do *boucing*) usando a função `delay()` depois que o botão fosse pressionado ou solto, assim ignorando o *boucing*, caso viesse a ocorrer. Se você retirar o delay do seu código, notará que alguns números podem ser pulados quando você pressionar o botão. Há, contudo, outras formas de realizar o *debounce* do botão, utilizando outras técnicas no código ou mesmo componentes adicionais

para a montagem. Neste exemplo da documentação do Arduino (em inglês) você pode conferir um código que utiliza a função `millis()`, que possui algumas vantagens em relação ao uso da função `delay()`:

https://www.arduino.cc/en/Tutorial/BuiltInExamples/StateChangeDetection

Exercício de Fixação

1) Altere a montagem de forma que o botão fique no modo pull-up. Realize as alterações necessárias no código para que o funcionamento inicial não seja alterado.
2) Integre o uso de botões ao display de modo que, o display deve contar os números de 0 a 9 apenas enquanto o botão estiver pressionado, quando ele não estiver pressionado, o display deve ficar fixo no número do contador.
3) Agora crie um código que quando o botão for pressionado o display deve mudar seu número para um número aleatório entre 0 e 9. Para isso, use a instrução random(10) que escolhe aleatoriamente um número de 0 a 9.
4) Inverta o contador: agora ele deve iniciar em 9 e o botão deve decrementar seu valor (subtrair 1) até chegar a 0, e então reiniciar em 9.
5) Adicione a exibição de números hexadecimais ao display e integre o uso de botões para incrementar os números de 0 a 15 (sistema hexadecimal) quando o botão for pressionado.
6) Crie um código que quando o botão for pressionado o display deve mudar seu número para um número aleatório entre 0 e 15 (sistema hexadecimal). Para isso, use a instrução random(16) que escolhe aleatoriamente um número de 0 a 15.
7) Agora tente deixar na sua montagem tanto um botão para incremento quanto outro para decremento. O código deve ser responsável por identificar qual dos dois botões foi pressionado para somar ou subtrair ao valor atual.
8) Em vez de um botão, utilize um potenciômetro para variar o valor do display de 7 segmentos. Varia o display de 0 a 9 à medida que o potenciômetro vai sendo girado da sua posição inicial até o máximo.

9) Com o contador automatizado novamente, vamos dar outra função para os botões. Adicione dois botões sendo um deles para incrementar a velocidade de contagem e outro para diminuir a velocidade de contagem. Eles devem poder ser pressionado várias vezes para aumentar e diminuir gradativamente essa velocidade.

3.3. Display LCD

O display LCD (Liquid Crystal Display ou Monitor de Cristal Líquido) é um tipo de monitor fino frequentemente utilizado nos dias de hoje, podendo ser encontrados em aparelhos eletrodomésticos, televisores, painéis de carros, relógios, além de suas inúmeras aplicações em sistemas microprocessados. Eles consomem menos energia e são mais resistentes, além de cansarem menos a vista e apresentarem um preço acessível.

3.3.1. Funcionamento

Os displays LCD apresentam várias interfaces diferentes, possuindo tamanhos diferentes de acordo com a quantidade de linhas e colunas. Para esse estudo, iremos utilizar um display LCD 16x2, que apresenta 16 colunas e 2 linhas como mostrada na Figura 3.10. Contudo, é possível, através das informações adiante, utilizar um display de modulagem ao seu gosto ou necessidade, como por exemplo um de 16x4, com 16 colunas e 4 linhas.

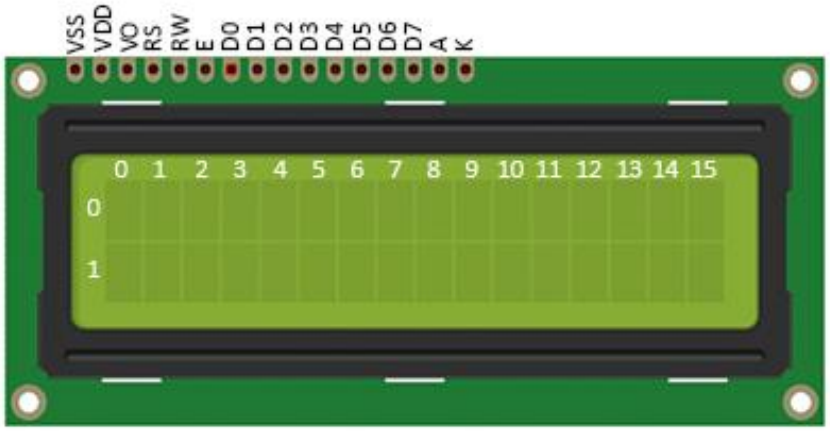


Figura 3.10: Display LCD.

É possível notar que a numeração dos valores de coluna e linha iniciam do zero. Por exemplo, quando quisermos aplicar um valor na segunda linha e quarta coluna, deve-se utilizar o valor de “um” para a linha e “três” para a coluna, ou seja, considera-se um valor a menos que o desejado. Além disso, percebemos a presença de 16 pinos, na qual as atribuições de cada um deles podem ser vistas na tabela a seguir. Como estaremos programando com o arduino, só é necessário que

utilizemos a configuração paralela de 4 bits, sendo necessário apenas os pinos D4, D5, D6 e D7 para impressão dos dados. A configuração em série do display LCD pode ser vista futuramente no Capítulo 7.

Tabela 3.1 – Tabela de conceituação da pinagem do Display LCD

Pino	Representação	Detalhes
1	VSS	Pino de alimentação GND (0V)
2	VDD	Pino de alimentação VCC (+5V)
3	VO	Pino de ajuste do contraste do LCD
4	RS	Instrução – nível 0 e Dados – nível 1
5	R/W	Write/Escrita – nível 0 e Read/Leitura – nível 1
6	E	Desativa o display – nível 0 e Ativa o display – nível 1
7	D0	Representa o bit 0 (interface de 8 bits)
8	D1	Representa o bit 1 (interface de 8 bits)
9	D2	Representa o bit 2 (interface de 8 bits)
10	D3	Representa o bit 3 (interface de 8 bits)
11	D4	Representa o bit 4 (interface de 4 ou 8 bits)
12	D5	Representa o bit 5 (interface de 4 ou 8 bits)
13	D6	Representa o bit 6 (interface de 4 ou 8 bits)
14	D7	Representa o bit 7 (interface de 4 ou 8 bits)
15	A	Ânodo de alimentação do LED backlight (5V)
16	K	Cátodo de alimentação do LED backlight (0V)

Para entender o funcionamento da impressão de um valor no display, é necessário entender como cada caractere funciona. Eles são compostos por uma

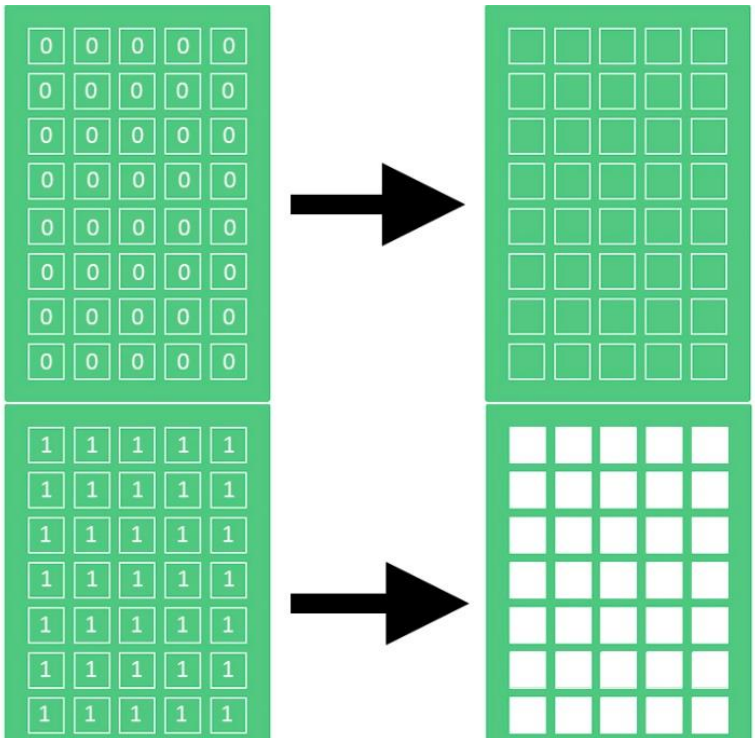


Figura 3.11: Configuração dos pixels ligados e desligados.

matriz de 5 colunas e 8 linhas de pixels, e, através do uso de números binários, é possível ativar cada pixel e formar a imagem a ser mostrada. A Figura 3.11 mostra como são apresentados todos os caracteres ligados e desligados, na qual o nível lógico "0" desliga e o nível lógico "1" liga o pixel. Na Figura 3.12 é mostrado a manipulação dos valores para a formação do caractere que representa a letra "A".

3.3.2. Aplicações

Existem inúmeras aplicações e usos para o Display LCD, a escolha vai de programador para programador. Nessa seção, iremos ver como utilizar as principais funções de exibição para o display e criar caracteres para ser impresso nele. A partir disso, poderá ser feito qualquer programa utilizando esse display, fazendo apenas uma manipulação das funções dele.

a) Funções do Display LCD

As funções do display podem ser vistas na Tabela 3.2. Para utilizá-las, lembre-se de incluir a biblioteca LiquidCrystal no seu código. Cada função apresenta sua finalidade que pode ser aplicada no desenvolvimento de um projeto dependendo da intenção do programador ao utilizá-la.

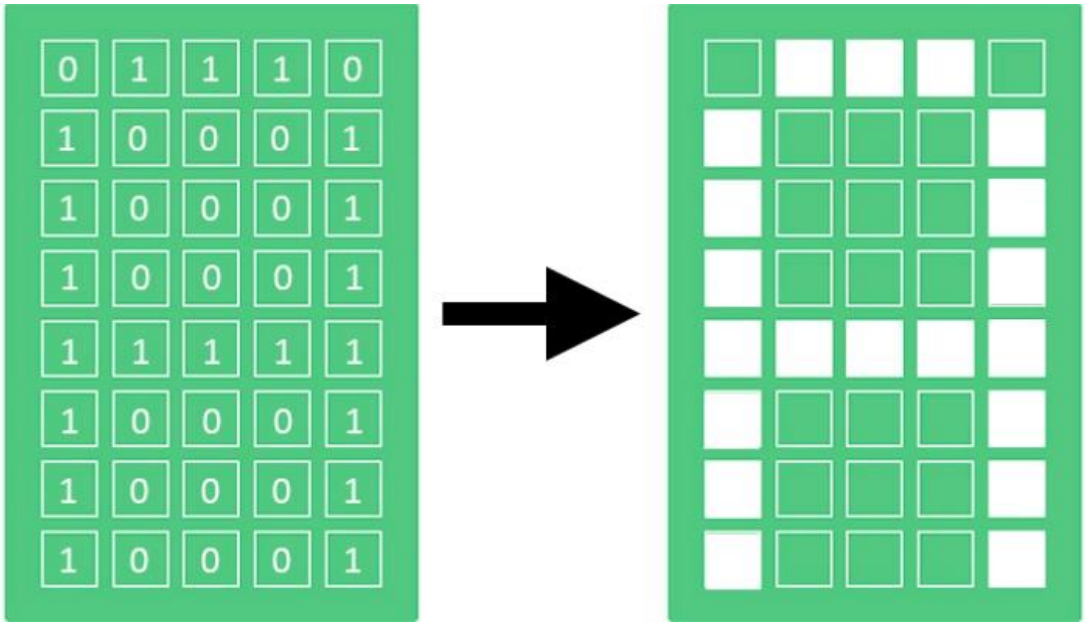


Figura 3.12: Configuração dos pixels formando a letra A.

Tabela 3.2 – Funções da biblioteca LiquidCrystal.

Função	Modo de Uso	Finalidades
LiquidCrystal	• LiquidCrystal(RS, Enable, DB4, DB5, DB6, DB7); • LiquidCrystal(RS, R/W, Enable, DB4, DB5, DB6, DB7); • LiquidCrystal(RS, Enable, DB0, DB1, DB2, DB3, DB4, DB5, DB6, DB7); • LiquidCrystal(RS, R/W, Enable, DB0, DB1, DB2, DB3, DB4, DB5, DB6, DB7);	Utilizada para configurar em qual pino do arduino irá cada pino do display, apresentando as 4 possíveis configurações, deixando para o programador escolher qual usar.
lcd.begin();	lcd.begin(colunas,linhas);	Determina o valor da matriz, informando o número de colunas e linhas que o display utilizado tem.
lcd.cursor();	lcd.cursor();	Inicia o cursor.
lcd.noCursor();	lcd.noCursor();	Desliga o cursor.
lcd.setCursor();	lcd.setCursor(coluna,linha);	Informa a posição que o cursor deve iniciar.
lcd.home();	lcd.home(); = lcd.setCursor(0,0);	Informa que o cursor inicia na posição (0,0).
lcd.write();	lcd.write(variável);	Utilizada para imprimir uma variável de diferentes tipos ou caractere criado.
lcd.createChar();	lcd.createChar(Número, Nome);	Cria um caractere no display, determinado por uma matriz de valores binários com o nome “Nome”, e valor do caractere “Número”, sendo ambos escolhidos pelo programador.
lcd.clear();	lcd.clear();	Limpa qualquer texto ou dado que esteja na tela do display.
lcd.print();	• lcd.print("texto”); • lcd.print(número); • lcd.print(número, tipo);	Imprime na tela do display um texto, número ou número com o formato desejado (bases

		decimal, binária, hexadecimal e octal).
lcd.blink();	lcd.blink();	Inicia o cursor que fica piscando.
lcd.noBlink();	lcd.noBlink();	Desliga o cursor que fica piscando.
lcd.autoscroll();	lcd.autoscroll();	Configura a rolagem automática da mensagem.
lcd.noAutoscroll();	lcd.noAutoscroll();	Desativa a rolagem automática da mensagem.
lcd.rightToLeft();	lcd.rightToLeft();	Imprime a mensagem da direita para a esquerda a partir da posição estabelecida anteriormente do cursor.
lcd.leftToRight();	lcd.leftToRight();	Imprime a mensagem da esquerda para a direita a partir da posição estabelecida anteriormente do cursor.
lcd.scrollDisplayLeft();	lcd.scrollDisplayLeft();	Imprime a mensagem em loop para a esquerda.
lcd.scrollDisplayRight()	lcd.scrollDisplayRight();	Imprime a mensagem em loop para a direita.

Para a construção do projeto, deve-se fazer a montagem do display ao arduino através da conexão de pinos analisando a Tabela 3.1. É possível observar o projeto montado na Figura 3.13.

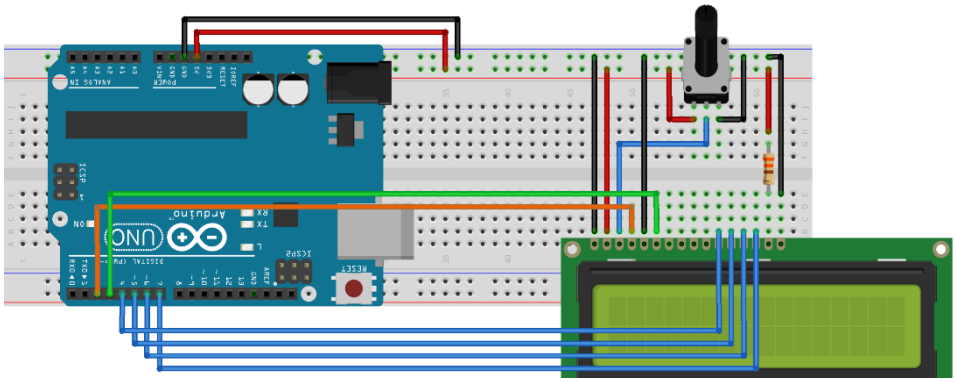


Figura 3.13: Montagem do display LCD.

O valor de Vss deve ser ligado ao ground, assim como o K e o valor do Vdd à 5V, do mesmo modo que o A. Por questão de proteção, coloca-se uma resistência de 330Ω ou inferior no ânodo, escolha feita pelo projetista. Como queremos controlar o valor de entrada do Vo para ajustar o contraste do display, coloca-se um potenciômetro com suas extremidades conectadas a 0V e 5V respectivamente, e seu meio ligado ao pino Vo do LCD. Além disso, como queremos receber os Dados e escreve-los no display, precisamos que ele esteja no modo escrita, portanto, ligaremos o R/W diretamente no GND. Os próximos valores de pinagem cabem ao programador escolher, nessa apostilha iremos atribuir o RS ao pino 2 do Arduino e o E ao pino 3. Os valores de D4 aD7 irão ser ligados aos seus valores de pinos semelhantes (4 a 7).

O código abaixo exemplifica a utilização de algumas funções estabelecidas na Tabela 3.2. Para melhor entendimento, sugerimos que pratique que não foram colocadas no código abaixo.

```
#include <LiquidCrystal.h> // Incluindo a Biblioteca

// definindo as variáveis
int RS = 2, EN = 3, D4 = 4, D5 = 5, D6 = 6, D7 = 7;

/*
 Pode ser configurado de 4 modos diferentes:
 LiquidCrystal(RS, Enable, DB4, DB5, DB6, DB7)
 LiquidCrystal(RS, R/W, Enable, DB4, DB5, DB6, DB7)
 LiquidCrystal(RS, Enable, DB0, DB1, DB2, DB3, DB4, DB5,
DB6, DB7)
 LiquidCrystal(RS, R/W, Enable, DB0, DB1, DB2, DB3, DB4,
DB5, DB6, DB7)
*/

// informando quais serão as portas no display,
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);

void setup() {
  // configura que o display utilizado possui uma
  // matriz de 16 colunas e 2 linhas
  lcd.begin(16, 2);
}
```

```
void imprimeTexto() {
  // o cursor vai iniciar na posição coluna 0 e linha 0
  lcd.home();
  // escreve o texto selecionado a partir da
  // configuração dada anterior
  lcd.print("O PET POTENCIA,");
  // o cursor vai iniciar na posição coluna 4 e linha 1
  lcd.setCursor(4,1);
  // escreve o texto selecionado a partir da
  // configuração dada anterior
  lcd.print("DIZ OI :)");
  delay(1000);
}

void Reiniciar(){
  lcd.noDisplay();  // função que desliga display
  delay(1000);      // atraso de 1 segundo
  lcd.display();    // função que liga display
  delay(1000);      // atraso de 1 segundo
  lcd.clear();      // função que limpa a tela
  delay(1000);      // atraso de 1 segundo
}

int numero;
//variável global para ser utilizad as próximas funções

void RolagemAutomatica() {
  //posicionando o cursor na coluna 15 e linha 1
  lcd.setCursor(15, 1);
  // função de rolagem automática
  lcd.autoscroll();
  // vai imprimir os números de 0 a 5
  for (numero = 0; numero < 6; numero++) {
    lcd.print(numero); // imprime o número
    delay(500);        // atraso de 500 ms
  }
  lcd.noAutoscroll();  // desliga rolagem automática
  lcd.clear();         // limpa a tela do lcd
  delay(1000);         // atraso de 1 segundo
}

void DireitaParaEsquerda() {
  lcd.clear();   // limpa a tela do lcd
  lcd.cursor();  // função que inicia o cursor
  lcd.setCursor(6, 0);  // selecionando coluna 6 e linha 0
```

```
  // vai imprimir os números de 0 a 5
  for (numero = 1; numero < 6; numero++) {
    // imprime da direita para a esquerda
    lcd.rightToLeft();
    // imprime o número com a configuração
    // direita -> esquerda;
    lcd.print(numero);
    delay(500);
  }
  lcd.noCursor();  // desliga o cursor
}

void EsquerdaParaDireita() {
  lcd.clear();   // limpa a tela
  lcd.cursor();  // função que inicia o cursor
  lcd.setCursor(6, 1);  // selecionando coluna 6 e linha 1
  // vai imprimir os números de 0 a 5
  for (numero = 1; numero < 6; numero++) {
    lcd.leftToRight(); // imprime da esquerda para a direita
    lcd.print(numero);
    // imprime o número com a configuração
    // esquerda -> direita;
    delay(500);
  }
  lcd.noCursor();  //desliga o cursor
}

void loop() {
  imprimeTexto();
  Reiniciar();
  RolagemAutomatica();
  DireitaParaEsquerda();
  EsquerdaParaDireita();
}
```

b) Criação de Caractere

Utilizando as funções definidas na seção anterior e usando a mesma montagem, é possível fazer a criação de um caractere. Iremos criar um boneco chamado petiano, como mostrado na Figura a seguir. Dessa maneira, a matriz de bits deve ser controlada de modo a formar um boneco de palito. Usaremos o bit 1 para acender e o 0 para manter apagado os pixels, como citado em momentos anteriores.

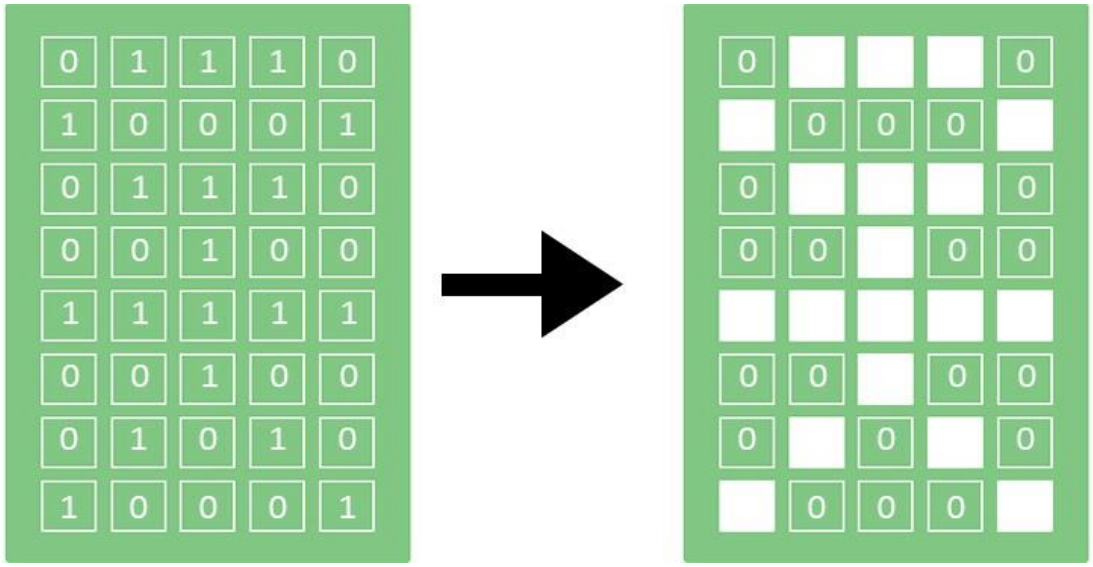


Figura 3.14: Configuração dos pixels para formação do boneco "petiano".

Do mesmo modo que foi “desenhado” na Figura anterior deve ser colocado na matriz byte. Após isso, cria-se um caractere utilizando essa matriz e associando um número para posteriormente referenciar o caractere através dele. Portanto, se você criar mais de um caractere, cada um poderá ser referenciado posteriormente com o número que foi colocado para ele. É possível ver esse procedimento no código a seguir.

```
#include <LiquidCrystal.h> // Incluindo a Biblioteca

// definindo as variáveis
int RS = 2, EN = 3, D4 = 4, D5 = 5, D6 = 6, D7 = 7;

// informando quais serão as portas no display,
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);

byte petiano[8] = {
  B01110,
  B10001,
  B01110,
  B00100,
  B11111,
  B00100,
  B01010,
  B10001,
};

void setup() {
```

```
  lcd.begin(16,2);
  // cria o caractere "petiano" e associa ao 0
  lcd.createChar(0, petiano);
  lcd.setCursor(7,0);
  // posiciona o caractere 0 na posição atual do cursor
  lcd.write((byte)0);
}

void loop() {
}
```

Exercício de fixação

1) Faça um programa que imprima a mensagem “Apostila do” na primeira linha e que ela mova em loop para a direita, e outra mensagem “PET Potência” na segunda linha com o movimento em loop para a esquerda.
2) Escreva também seu nome no display e o faça ficar piscando.
3) Faça um caractere de uma carinha sorrindo e outro de uma carinha triste e imprima na tela do display.
4) Crie caracteres referentes a letra P, E e T, de modo que cada letra ocupe dois espaços, um na primeira linha e outro na segunda. DICA: Serão necessários seis caracteres.
5) Utilize o display para mostrar a leitura feita por um potenciômetro, exibindo na primeira linha do display o valor lido entre 0 e 1023, e na segunda linha o valor de tensão lido, entre 0V e 5V.
6) Faça um código que o potenciômetro define para que lado deve ser feito a rolagem do texto. Desse modo, ao rodar o potenciômetro para a direita o texto deve rolar para direita, e ao rolar o potenciômetro para a esquerda, o texto deve rolar para a esquerda.

7) Salve 5 pequenas frases em variáveis no seu Arduino e utilize um botão para ficar alternando entre essas frases sempre que apertado. Após isso, coloque um segundo botão para ligar e desligar o display.
8) Faça um código que execute a rolagem de texto no LCD em apenas uma linha a sua escolha. Dessa forma uma linha deve estar se movendo para esquerda e a outra deve estar fixa. Sugestão: faça seu nome completo ser mostrado na linha 0 pela rolagem e na linha 1 fica fixo o seu ano de nascimento.
9) Pesquisa! Assim como o display de 7 segmentos ocupa muitas portas do Arduino, o display de 7 segmentos também ocupa. Existe uma forma de ligar o LCD ocupando apenas dois pinos do nosso Arduino chamado de comunicação I2C, dê uma pesquisada e aprenda também como ligar o display LCD utilizando módulos I2C.

3.4. Buzzer

O buzzer é um dispositivo sinalizador que emite som quando ativado. Quem tem micro-ondas em casa já deve ter ouvido o sinal de alerta informando que o processo foi finalizado, o responsável por aquela zoadinha é o buzzer. Assim como, o barulho que o ar-condicionado emite quando é enviado um comando a ele por meio do controle é emitido por um buzzer. Resumindo, o buzzer é muito utilizado em nosso dia a dia mesmo que, muitas vezes, nem se quer percebemos.

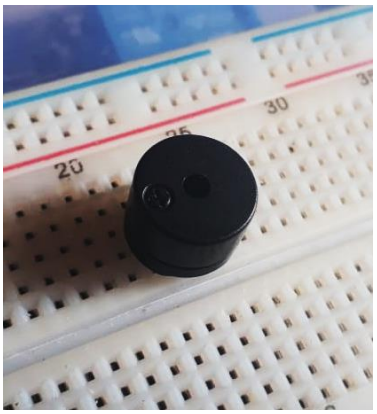


Figura 3.15: Buzzer.

O que faz o buzzer fazer sua famosa zoadinha é seu funcionamento baseado no efeito piezoelétrico reverso. O princípio do efeito piezoelétrico é que se surge uma tensão elétrica a partir de um esforço mecânico e pode ocorrer em alguns tipos de materiais cristalinos. O buzzer usa o efeito reverso, pois, a partir de uma tensão elétrica que é gerado um esforço mecânico, que no caso, é a onda sonora emitida por ele.

Existem dois tipos de buzzer, o passivo e o ativo. O buzzer ativo atua em uma determinada frequência sonora pré-determinada de fábrica, por isso, o Arduino não pode alterar essa frequência e assim, as aplicações com esse buzzer são basicamente ligar/desligar. Por isso, buzzer passivo é o que tem mais fácil utilização, no entanto, não pode ser usado para formar melodias ou alarmes, pois nesses casos, é preciso alterar a frequência do som. Esse tipo é o mais usado em ar-condicionado e micro-ondas pois, no funcionamento deles só é necessário fazer ligar e desligar o buzzer quando necessário. Ele pode ser identificado olhando sua traseira que deve estar lacrada e seus pinos tem tanhos diferentes, sendo o VCC o pino maior.

Já o buzzer passivo, espera receber uma frequência do Arduino para emiti-la em ondas sonoras. Por isso, ele é o mais utilizado para projetos que precisam soar alarmes e/ou melodias. Por ter um funcionamento diferente do buzzer ativo,

muitas pessoas pensam que o buzzer passivo é um buzzer ativo com defeito, o que também vêm de uma má utilização do componente. Para identificar um buzzer passivo, basta olhar sua traseira que não é tampada e fica mostrando sua PCB, além disso, os pinos tem tamanhos iguais, a Fig. 3.15 mostra um exemplo de buzzer passivo visto de cima.

Como é necessário apenas energizar e desenergizar um buzzer ativo para ouvi-lo funcionando, vamos fazer um exemplo com o buzzer passivo. Para podermos definir essa frequência e produzir o som no buzzer utilizamos da função **tone** para utiliza-la basta definir o pino em que o buzzer está conectado, qual frequência sonora é a melodia e a duração que essa frequência deve soar como pode ser visto a seguir: `tone(pino_buzzer, frequencia, duracao)`.

Há também a função **notone** que desabilita o som, para utilizá-lo basta apenas definir o pino, como mostrado a seguir: `notone(pino_buzzer)`.

Vamos agora fazer um código para que um buzzer passivo emita o som de um alarme de sirene. Para isso, utilizaremos um buzzer passivo e um resistor de 330Ω para garantir a proteção do buzzer.

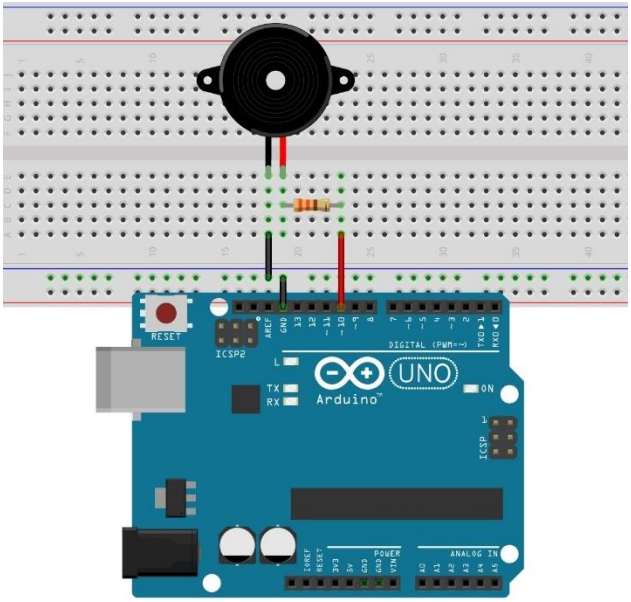


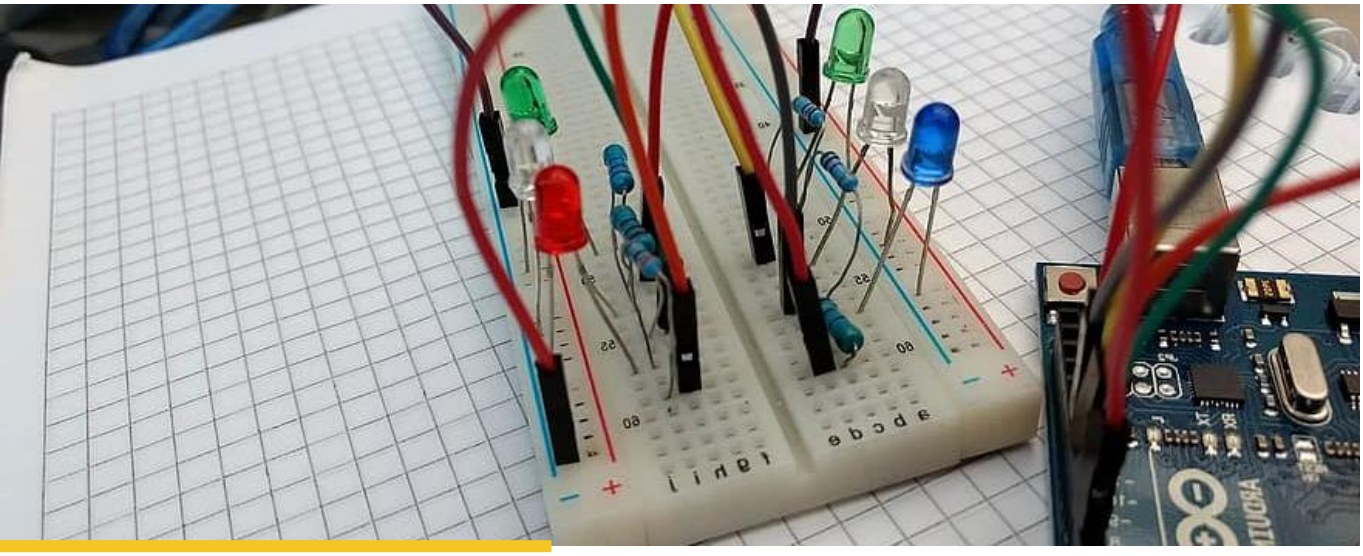
Figura 3.16: Diagrama Elétrico.

```
int pino_buzzer = 10;
int frequencia;
int duracao = 10;
void setup() {
  pinMode(pino_buzzer,OUTPUT); //Pino do buzzer é de saída
}
void loop() {
  for (frequencia = 200; frequencia < 1500; frequencia ++)
  {//altera a frequencia rapidamente
    tone(pino_buzzer, frequencia, duracao);
    delay(5);
  }
  for (frequencia = 1500; frequencia > 200; frequencia -= 1)
  {
    tone(pino_buzzer, frequencia, duracao);
    delay(5);
  }
}
```

Exercício de fixação

1) Busque pesquisar quais frequências referentes às notas DÓ, RÉ, MI, FÁ, SOL, LÁ e SÍ e monte um código que o buzzer toque todas essas notas musicais.
2) Tendo as notas musicais, projete o buzzer para tocar a música "DÓ RÉ MI FÁ".
3) Com um buzzer ativo, monte um sistema que lembre um ar-condicionado, que ao pressionar um botão, o buzzer emite um som.
4) Monte um sistema de alarme, no qual, tendo 3 botões disponíveis, defina uma sequência certa para pressioná-los, e que caso os botões sejam pressionados em ordem errada, soe um alarme com o buzzer alertando erro na sequência.
5) Desafio! Monte o jogo Genius com o Arduino, botões, LEDs e buzzer. O Genius é um jogo de memória que consiste em mostrar uma sequência de cores para os jogadores e após isso, eles devem pressionar os botões correspondentes às cores dadas repetindo a sequência. A sequência começa apenas com uma cor e de forma aleatória o jogo adiciona novas cores formando a sequência de cores para memorizar. Cada partida gera uma sequência diferente, por isso, busque fazer

surgir cores de modo aleatório. O jogo acaba quando o jogador erra a sequência dada. Faça o jogo iniciar quando for pressionado um botão. Dicas: Use a função random(num) para gerar um número aleatório para o jogo, se você usar random(4) ele irá gerar um número aleatório entre 0 e 3 e esse número pode corresponder a uma das 4 cores. Caso tenha ficado alguma dúvida sobre o funcionamento do Genius pesquise no Google ou Youtube sobre o funcionamento desse jogo para uma montagem mais eficaz.



Capítulo 4: Diodos e Transistores

Neste capítulo abordaremos a respeito de dois componentes de grande importância para a eletrônica: os diodos e os transistores. Trataremos de maneira simplificada o funcionamento destes dispositivos, além de alguns exemplos e aplicações simples, a fim de elucidar sobre eles para sua utilização em capítulos futuros.

4.1. Diodos

A princípio, podemos definir o diodo como um componente eletrônico capaz de conduzir corrente elétrica somente em um sentido. Esta característica só é possível devido ao material semicondutor de que ele é feito, onde pode adquirir capacidades tanto de permitir quanto isolar a passagem de corrente. Os materiais normalmente usados na confecção de diodos são o silício e o germânio, dois semimetais da mesma família da tabela periódica.

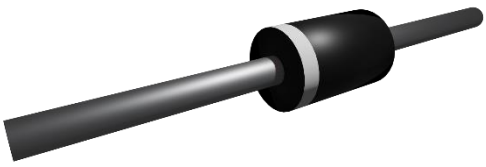


Figura 4.1: Exemplo de diodo

Os diodos podem ser utilizados em duas configurações: diretamente polarizado (mesmo sentido da corrente) ou inversamente polarizado (sentido contrário ao da corrente).

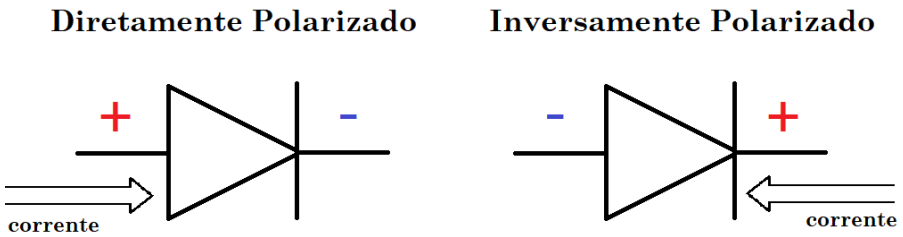


Figura 4.2: Representaçãoes do Diodo.

4.1.1. Características dos Diodos

Diferentemente dos resistores, os diodos não são dispositivos lineares, ou seja, o aumento de tensão não ocasiona um aumento diretamente proporcional no valor da corrente. O comportamento do diodo é mais próximo de uma exponencial como mostrado na Figura Y.

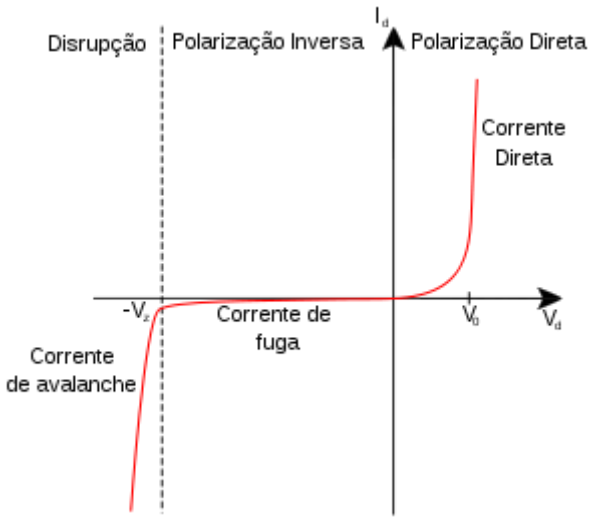


Figura 4.3: Gráfico da curva característica do diodo.

Observa-se no lado direito do gráfico que, para um diodo diretamente polarizado, o valor da corrente não aumenta muito até chegar a um certo valor de tensão V_0, onde ela explode e o diodo passa a conduzir, este valor de tensão é chamado de Tensão de Queda do diodo.

Portanto, o diodo só conduz quando essa barreira da tensão de queda é vencida. Na maioria dos diodos de silício utilizados em projetos, como o 1N4007 ou o 1N4004, esse valor é de aproximadamente 0,7 V.

Já do lado esquerdo do gráfico, verifica-se que para um diodo inversamente polarizado, a corrente é praticamente nula à medida que ocorre o aumento de tensão sob os terminais do diodo. Dessa maneira, como dito anteriormente, o diodo a princípio não deveria conduzir nessa configuração, entretanto, devido às peculiaridades do material semicondutor de que ele é formado, existe um ponto chamado Tensão de Ruptura ($-V_Z$) onde o diodo passa a se comportar como um curto-circuito mesmo estando inversamente polarizado, levando a um aumento repentino no valor da corrente. Assim como a Tensão de Queda, a Tensão de Ruptura tem um valor específico que varia entre os tipos de diodo: no 1N4004, o valor é de 280 V_{RMS}, enquanto que no 1N4007 é de 700 V_{RMS}.

4.1.2. Exemplos de Diodos

Mostrado na figura abaixo, temos um exemplo comum de um diodo retificador. Esse tipo normalmente é feito de silício e, como diz o nome, é utilizado no processo de retificação em eletrônica, mas também pode ser utilizado para proteção.

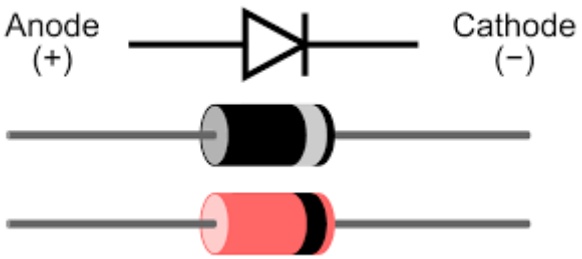


Figura 4.4: Diodo retificador e sua representação gráfica.

Ao longo desta apostila também já foi aplicado diversas vezes o Diodo Emissor de Luz, popularmente conhecido como LED. Como o próprio nome diz, esse componente é capaz de emitir luminosidade quando polarizado diretamente e sua queda de tensão varia de acordo com a cor do LED utilizado. Além disso, vale lembrar que quando utilizado em projetos de eletrônica, a tensão aplicada

normalmente é de 5 V, logo deve vir acompanhado de um resistor (330Ω, 470Ω), a fim de restringir a corrente que flui e evitar a queima do LED.

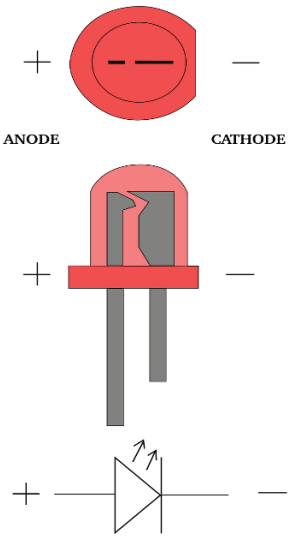


Figura 4.5: LED e sua representação gráfica.

Outro tipo que vale ressaltar, por curiosidade, é o diodo Zener. Ele se comporta como um diodo comum quando polarizado diretamente, porém a sua principal aplicação é para ser utilizado em polarização reversa. Nesse último caso, ele é implementado em reguladores de tensão, ou seja, circuitos que permitem segurar um valor de tensão na saída, por exemplo, manter uma tensão de 5 V ou 11 V.

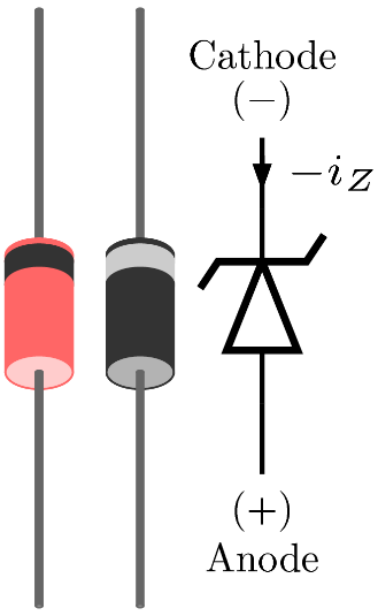


Figura 4.6: Diodo Zener e sua representação gráfica.

4.1.3. Aplicações

a) Diodos em série

Primeiramente, apresentaremos um exemplo só para demonstrarmos a condução entre dois diodos, um retificador representado pelo 1N4007 e um LED vermelho. O primeiro apresenta uma queda de tensão de 0,7 V enquanto que o outro tem uma queda de 2,2 V. O circuito de verificação está representado na figura abaixo:

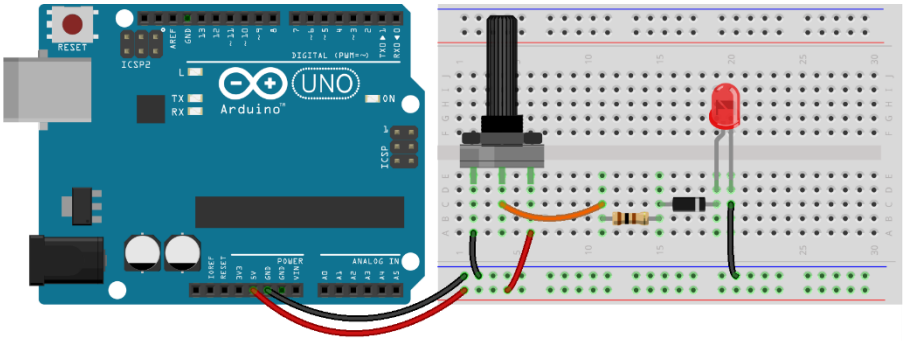


Figura 4.7: Esquemático das ligações do exemplo 1.

Além disso, temos também um potenciômetro de 1kΩ responsável por variar a tensão entre 0 e 5 V e, aliado ao resistor de 100Ω, restringir a corrente que passa no circuito.

Observa-se que a condição para o LED acender é que a queda de tensão tanto dele quanto do diodo 1N4007 devem ser superadas, isto é, 0,7V + 2,2V = 2,9V. Logo, precisa-se no mínimo 2,9V para ligarmos o LED e, caso seja fornecido uma tensão superior a esse valor, o restante ficará distribuído nas resistências.

b) Polarizações de diodos

Agora, faremos um circuito para demonstrar a polarização de dois diodos representados por LEDs e controlados por meio de um único pino, como mostrado na Figura Y. Nesta aplicação, o LED vermelho começará apagado e o LED azul começará aceso, pois o estado lógico do pino 2 que controla a polarização de ambos estará inicialmente em nível lógico baixo. A cada vez que for apertado, um botão será o responsável por inverter o estado lógico do pino 2, por exemplo, ao ser pressionado pela primeira vez, o pino 2 passará a ter nível lógico alto, polarizando diretamente o LED vermelho para acendê-lo, ao passo que despolariza o LED azul,

apagando-o. Dessa forma, cada vez que o botão for acionado os LEDs vão alternar a luz entre si.

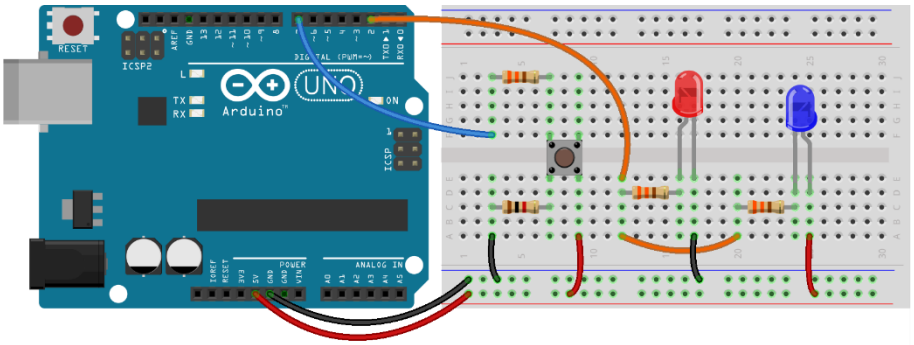


Figura 4.8: Esquemático das ligações do exemplo 2.

Após preparar a montagem, devemos também realizar a programação no nosso arduino. O programa responsável pelo controle descrito está demonstrado abaixo:

```
int botao = 7;  // número do pino do botão
int valor = 0;
void setup() {  // Configura-se o botão como entrada
  pinMode(botao, INPUT);
  // Define-se o pino 2 como saída para controle
  // dos LEDs e o inicia em nível baixo
  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);
}
void loop() {
  valor = digitalRead(botao);
  delay(200);
  if (valor != LOW) {
    // Após apertar o botão, o programa lê o valor atual do
    // pino 2 e sobrescreve com o valor oposto, invertendo
    // o estado lógico do pino e a iluminação dos LEDs
    digitalWrite(2, !digitalRead(2));
  }
}
```

No próximo capítulo serão trabalhadas aplicações com o diodo retificador com relação a atuadores, pois ele pode ser utilizado como proteção para prevenir uma possível ligação invertida e, consequentemente, uma corrente fluindo em um sentido indesejado.

4.2. Transistores

Nesta seção, vamos estudar o funcionamento do Transistor Bipolar de Junção (TBJ), que é um dos principais transistores descritos na literatura. Outro grupo que possui grande destaque no meio são os Transistores de Efeito de Campo (FET), porém, como sua aplicabilidade em sistemas discretos não é tão expressiva quanto aos TBJ's, ele não será abordado nessa seção. Para um aprofundamento neste assunto, consulte o livro "Dispositivos Eletrônicos e Teorias de Circuitos".

4.2.1. Transistor bipolar de junção (TBJ)

O TBJ é um dispositivo eletrônico construído a partir de materiais semicondutores, eles possuem três terminais de conexão, conhecidos como B – base, C – coletor e E - emissor. Eles podem ser classificados em dois grupos: NPN e PNP, que dizem respeito as características do material semicondutor aplicado a cada terminal. Como exemplo, o TBJ do tipo NPN, apresenta excesso de cargas negativa no terminal coletor, excesso de cargas positiva no terminal de base e excesso de cargas negativa no terminal de emissor.

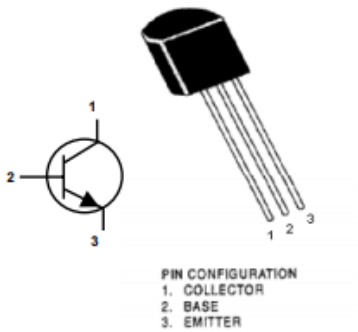


Figura 4.9: Diagrama de pinos do TBJ BC548.

O princípio de funcionamento desses dispositivos é o uso de tensão entre dois terminais para controlar o terceiro terminal. Dentro da eletrônica analógica, eles possuem diversas aplicações, sendo as principais: construção de circuitos

amplificadores, chaves controladas e *transistor-transistor logic* (TTL). No que se refere a sua aplicabilidade na robótica, ela está mais concentrada no controle de chaves para o acionamento de atuadores, como exemplo, os motores elétricos.

4.2.2. Aplicações

Nesta aplicação, vamos entender o funcionamento do TBJ atuando como uma chave controlada. Dessa forma, você conseguirá acionar um atuador através de um sinal de controle emanado do seu arduino. Esse sinal deve ser enviado ao pino de base do transistor, sendo nível lógico alto (+5 V) para acionar seu atuador (Ex. Acionar motor dc, diodo emissor de luz) ou nível lógico baixo (0 V) para desativar seu atuador.

Você pode estar se perguntando: "Por que devo utilizar um transistor para acionar um atuador, sendo que posso fazer isso, simplesmente utilizando uma saída digital do arduino conectada diretamente ao meu atuador?". Então, você até pode fazer isso, porém, em cargas mais potentes, a demanda por corrente elétrica pode ser alta e sua porta digital do arduino não será capaz de fornecer esse nível de potência exigida. Dessa maneira, utilizamos o TBJ como chaves, quando precisamos fazer um isolamento entre o circuito de potência e o circuito lógico.

Assim, esses arranjos de chaves controladas são bastante utilizados em circuitos de *drive* para atuadores que, inclusive, é uma das principais maneiras de aplicarmos as teorias de controle avançado. Mas calma, que você não vai precisar disso tudo para agora.

Voltando para nossa aplicação, um detalhe importante na hora da montagem, é a utilização de um resistor de alto valor na base do transistor, de modo a preservar sua integridade, visto que ele não suporta altos valores de corrente em seu terminal de base. Em seguida, entenda os pinos de coletor e emissor como os pinos de um botão, visto no capítulo anterior. Assim, na ausência de tensão + 5 V na base do transistor implica no circuito aberto entre os pinos de coletor-emissor, já quando houver sinal de tensão + 5 V no pino de base, os pinos

coletor-emissor se comportarão como uma chave fechada. A figura abaixo ilustra o esquemático de ligação.

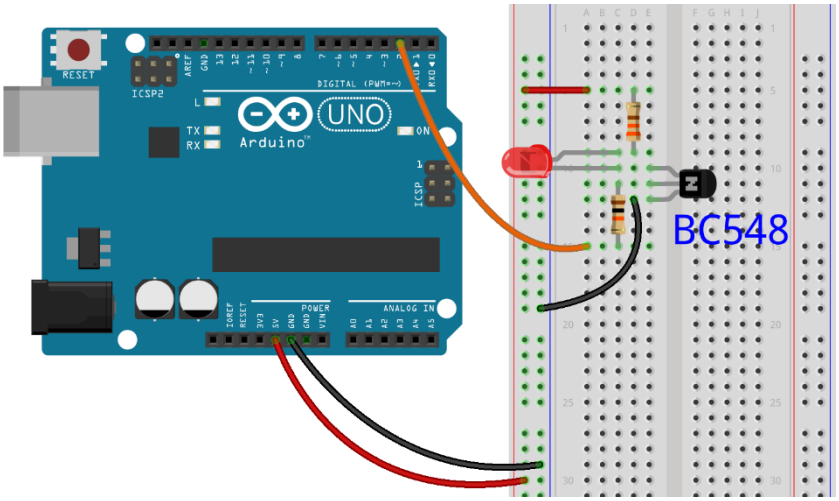


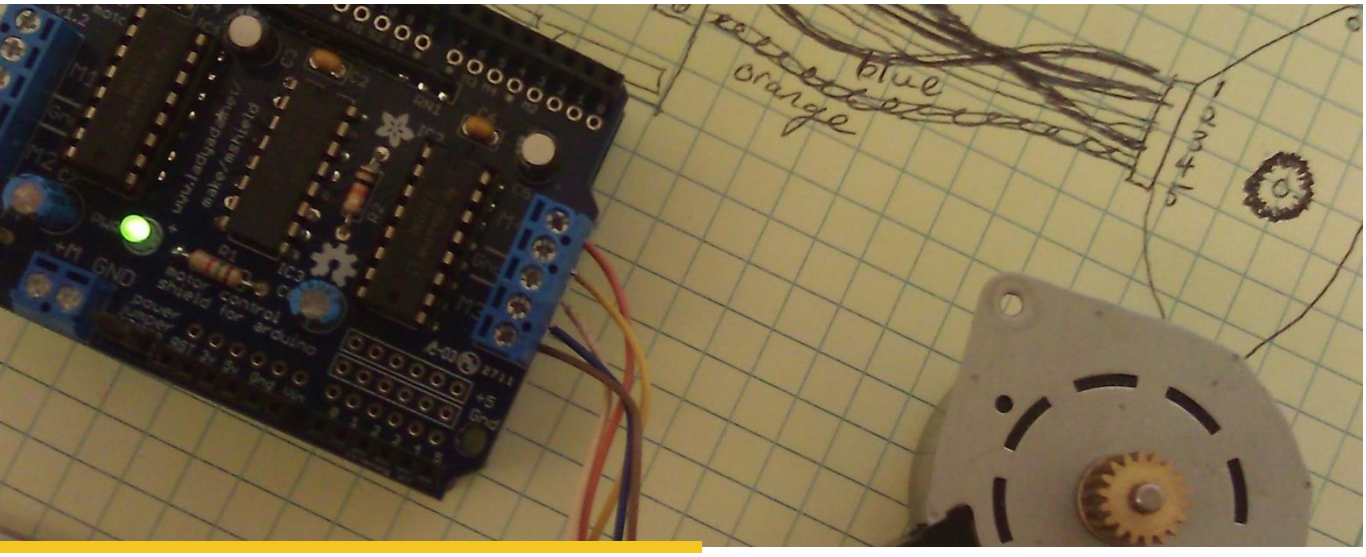
Figura 4.10: Diagrama esquemático de ligação.

Depois de montar todas as conexões, chegou a hora de programar nosso arduino. Assim, utilizamos o pino digital 12 como saída de dados, para que através dele seja feito o controle da chave (TBJ), ou seja, o *ON/OFF* do nosso atuador, que nessa montagem é o diodo emissor de luz (LED). O *sketch* (programa) desenvolvido para essa montagem, está ilustrado no quadro abaixo.

```
//definição de pinos
#define pinoBase 2
void setup() {
  //pino 2 confi. como saida digital
  pinMode(pinoBase, OUTPUT);
  //inicializando o TBJ como chave aberta
  digitalWrite(pinoBase,LOW);
}
void loop() {
  //aciona chave controlada
  digitalWrite(pinoBase,HIGH);
  delay(1000);
  //desativa chave controladora
  digitalWrite(pinoBase,LOW);
  delay(1000);
}
```

Exercício de Fixação

1) Que tal a partir de agora utilizar uma pilha para acender seus leds que você esteve utilizando? Geralmente as pilhas possuem uma tensão de 1,5 volts, calcule qual resistor é o recomendado para acender um LED nesta tensão. Em seguida teste primeiro se você consegue acender o led com o resistor e a pilha. Em seguida no meio do circuito, entre o cátodo do LED e o negativo da pilha, posicione o TBJ 548 com a base dele conectada a um pino digital do Arduino através de um resistor de 10k para controlar se o transistor irá abrir ou fechar. Faça um código blink para piscar esse LED através do TBJ 548 usando a energia da pilha. Cuidado, não esqueça de conectar o GND do Arduino ao negativo da pilha.
2) Por meio do TBJ548, desenvolva um controle ON/OFF para um display de sete segmentos do tipo cátodo comum. (Sugestão.: Coloque um resistor de 10 kΩ no terminal de base do TBJ; conecte o pino cátodo comum ao terminal de coletor do TBJ, o terminal emissor conectado ao GND e o terminal de base sendo controlador por um botão, de forma, que quando o botão estiver pressionado o display deva ser desligado).
3) Desafio! Faça um contador manual, similar ao do capítulo anterior, mas dessa vez, adicione um botão que faça o controle ON/OFF do display de sete segmentos do tipo cátodo comum. (Sugestão.: Faça o arranjo do circuito do botão ativar/desativar a base do TBJ548, que por sua vez, fará a conexão do cátodo ao terminal negativo GND; utilize o resistor de 10 kΩ no terminal de base do TBJ).



Capítulo 5: Atuadores

Neste capítulo, abordaremos alguns atuadores de grande importância para a eletrônica: os motores DC, servos motores e motores de passo. Analisaremos o funcionamento de cada um desses componentes, além da técnica de controle de velocidade através de PWM. Por fim, estudaremos a importância de fontes externas, relés e transistores operando como chave, a fim de entender as aplicações.

5.1. Motor DC

O Motor DC (Direct Current) é um atuador alimentado por corrente continua, que funciona por meio da força eletromotriz. A velocidade, força e torque desse motor é controlada de acordo com a variação da tensão fornecida pela sua alimentação.

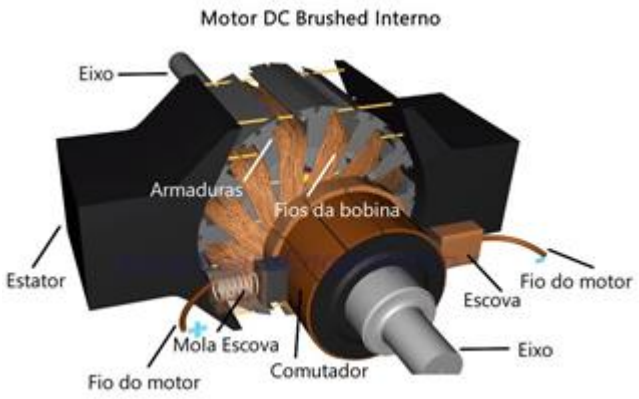


Figura 5.1: Circuito interno de um Motor DC.

De acordo com o que podemos ver na figura 5.1, o estator gera um campo magnético unidirecional contínuo, que pela Lei de Lenz, irá fazer a bobina se movimentar quando esta receber corrente a partir dos anéis do comutador. Dessa forma, uma força eletromotriz é gerada, condicionando o movimento giratório do eixo fixado ao sistema.

Como podemos observar, o motor é composto por várias bobinas, envoltas por armaduras, que são estruturas de aço que facilitam a interação do fluxo magnético, tendo assim uma melhor movimentação do atuador.

É importante considerar que não se deve ligar motores DC diretamente no Arduino, já que, apesar das portas digitais fornecerem uma tensão adequada, esse dispositivo precisa de bem mais corrente do que a enviada pelo microcontrolador, sendo então necessária o uso de uma fonte externa (mais especificado no final deste capitulo).

5.1.1. Aplicação

Agora faremos um circuito para demonstrar a ativação periódica de um Motor DC. O motor será alimentado com uma fonte externa, já que as portas lógicas do Arduino não disponibilizam corrente suficiente para o bom funcionamento do motor, podendo até mesmo queimar o microcontrolador.

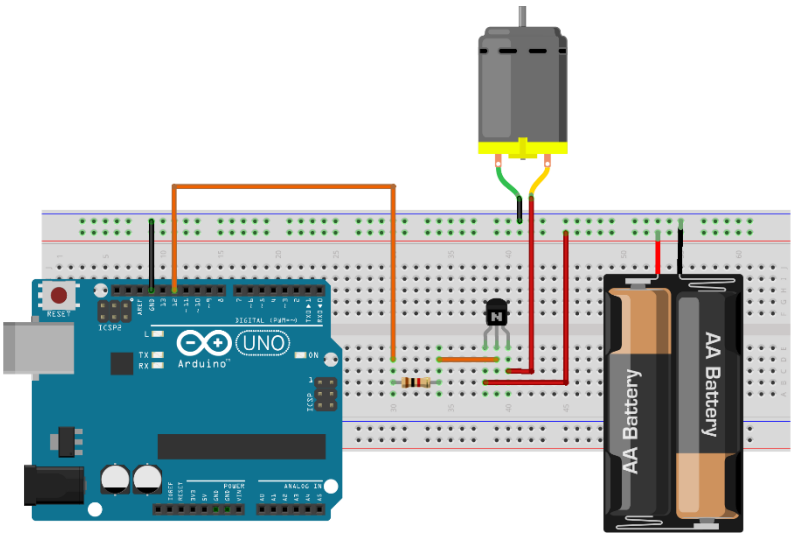


Figura 5.2: Esquemático do sistema.

Junto ao sistema motor-bateria-Arduino, ligaremos um transistor, que servirá como chave de ativação. Nele, uma das extremidades recebe o lado positivo da fonte (coletor), e a outra extremidade recebe o lado positivo do motor (emissor), enquanto a entrada do meio recebe o sinal do Arduino (base), que irá definir a ativação ou desligamento do motor. O funcionamento completo desse dispositivo pode ser relembrado no capítulo 4 desta apostila.

Depois de terminar a montagem, iremos fazer a programação do Arduino. Nele, utilizaremos a porta digital 12 como saída de dados, para que através dela o transistor seja controlado para a ativação do motor. O código abaixo consiste na ativação e desativação do atuador a cada um segundo.

```
#define ATIVAR 12
//define a porta digital 12 com o nome 'ativar'

void setup (){
  pinMode(ATIVAR, OUTPUT); // define a porta 12 como saída
}

void loop(){
  digitalWrite(ATIVAR, HIGH); //ativa o motor
  delay(1000);
  digitalWrite(ATIVAR, LOW); //desativa o motor
  delay(1000);
}
```

Exercícios de Fixação

1) Utilizando um transistor simule o funcionamento de um pisca LED com o motor DC em que ele deve girar e parar, simulando quando a LED ascende e apaga.
2) Relembre que o Arduino utiliza um sinal PWM para simular sinais analógicos, com um sinal que na verdade consegue apenas atingir o valor de 0 e 5V. Isso faz com que este sinal possa ser utilizado para saturação do transistor e seu

comportamento como circuito fechado ocorra. Pensando nisso, utilize agora a função analogWrite() para controlar a velocidade de rotação do motor com a mesma montagem da seção anterior.

3) Uma vez que você conseguiu definir a velocidade, execute a simulação de um pisca LED em velocidade distintas.
4) Controlando a velocidade do motor, faça isso agora controlando com a ajuda de um potenciômetro, fazendo o motor girar mais rápido ou mais devagar.
5) Na montagem da questão anterior, adicione um botão que irá fechar totalmente o circuito, desligando o motor. Quando o motor estiver desligado, o potenciômetro não deve funcionar, sendo preciso ligar o motor e só então controlar sua velocidade.
6) Construa o controlador do motor junto com um medidor de velocidade usando um display LCD ou de 7 segmentos que irá mostrar em uma escala de 0 a 5 a velocidade de um motor DC de acordo com o nível do potenciômetro.

5.2. Servo Motor

O Servo Motor é um controle de posição angular. Em outras palavras, ele não funciona a exemplo do motor DC, girando continuamente, mas apresenta um movimento que segue comandos previamente definidos, de acordo com ângulos específicos. Esse atuador funciona com três dispositivos: um motor, um circuito de controle e um potenciômetro, que servirá como sensor para detectar qual a posição atual do eixo do motor. É válido considerar que o potenciômetro pode ser substituído por outros dispositivos de detecção da posição nas diversas variações de servo motores. Podemos ver, na imagem a seguir, o sistema descrito:

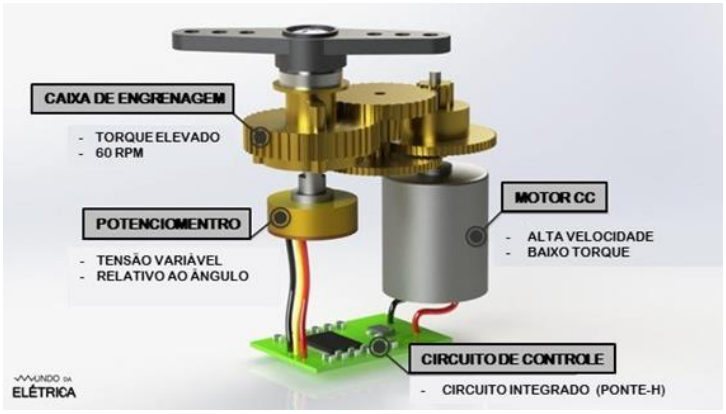


Figura 5.3: Circuito interno de um servo motor.

Esse dispositivo é considerado um sistema de controle em malha fechada, pois possui um sensor que envia os atuais estados do motor para um sistema de controle. Dessa forma, enviamos ao Servo Motor qual a posição exata em que queremos que ele se direcione, e esse sensor, que pode ser considerado um detector de erros, observa se ele já chegou ou não a esse estado, determinando a parada ou continuidade desse movimento.

Tal atuador possui também uma série de engrenagens acopladas ao motor, que fazem uma redução da velocidade angular e o aumento do torque, aumentando assim a força motriz desse dispositivo. Isso tem altos benefícios, considerando a possibilidade de movimentos reduzidos e controlados, e também a capacidade do dispositivo de suportar a sustentação de pesos bem mais elevados que o motor DC, para um mesmo valor de potência.

5.2.1. Aplicação

O Servo Motor apresenta três entradas, sendo duas delas de alimentação e a outra para recepção do sinal do microcontrolador. Nessa montagem iremos utilizar o sinal enviado por um potenciômetro para controlar o movimento do atuador, como podemos observar na imagem 5.4.

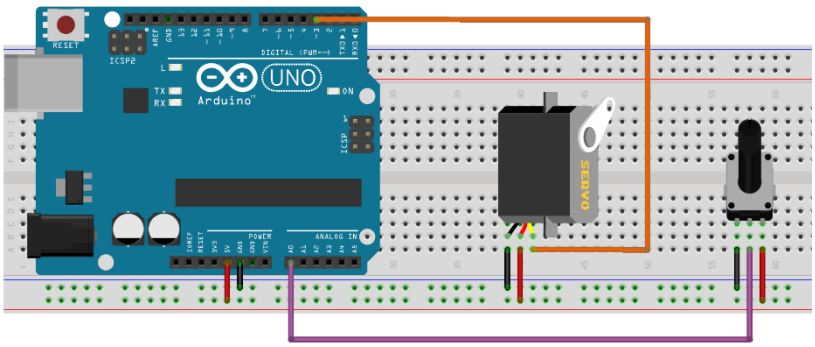


Figura 5.4: Esquemático da montagem com o servo motor.

A seguir, temos o código da IDE do Arduino onde podemos ver a chamada da biblioteca que coordena o servo motor e a definição das portas e objetos que serão utilizados. No void loop, observa-se uma organização em que primeiro lê-se o valor recebido pela porta analógica A0, que está ligada ao potenciômetro, convertendo tal valor para ângulos, e em seguida enviando-o para o atuador iniciar seu movimento.convertendo tal valor para ângulos, e em seguida enviando-o para o atuador iniciar seu movimento.

```
#include <Servo.h> // biblioteca do servo motor
#define servo 3    // pino onde liga o servo motor
#define potenciometro A0
Servo servomotor;  // definição do objeto

void setup(){
  // pino em que o servo esta ligado
  servomotor.attach(servo);
}

void loop(){
  int leitura = analogRead(potenciometro);
  // converte a leitura do potenciometro 0 a 1023
  // para ângulos de 0 a 180
```

```
  byte angulo=map(leitura,0,1023,0,179);

  // definir qual angulo o servo deve ir, a partir
  // da variação do potenciômetro
  servomotor.write(angulo);
}
```

Exercícios de Fixação

1) Você pode alimentar o servo motor com as fontes disponibilizadas pelo Arduino, mas dependendo da aplicação ele pode consumir bastante corrente, sendo necessário que você procure outra forma de alimentar o servo. Tente fazer isso através de três pilhas ligadas em série conectadas aos pinos positivo e negativo do servo e um pino PWM do Arduino conectado ao controle do servo para escolher sua posição. Mais uma vez não esqueça de conectar o GND do Arduino ao ponto negativo da primeira pilha da série.
2) Faça um código que simule um cofre de segurança, onde, o servo motor é o responsável por trancar/destrancar o seu cofre e a sequência de segurança é definida por uma sequência de 4 botões pressionados. Assim, ao pressionar uma sequência definida por você nos botões, o servo sai da angulação em 0° para 90°, "abrindo a porta".
3) Vamos juntar uma montagem com o servo motor e um motor DC agora imaginando que o motor DC controla a velocidade das rodas de um carro e o servo controla a direção para onde a roda está apontando. Assim faça um código que irá controlar tanto a velocidade do motor DC como a direção para onde o servo aponta, fazendo um carro hipotético fazer uma curva.
4) Com o servo motor, simule um cronômetro analógico de 1 minuto, nele, a cada 1 segundo o servo gira em uma angulação correspondente de modo que, chegando em 1 minuto, ele deve estar em 360° de giro.
5) Desafio! Vamos criar um pequeno jogo usando um servo motor e 4 ou mais botões. Posicione fixamente um servo motor em frente a 4 botões na protoboard

e teste posições de forma que ele fique apontando para cada um dos botões. Para iniciar o jogo, basta que você aperte em qualquer um dos botões. Em seguida o servo deve escolher aleatoriamente um botão para o qual apontar e você deve apertar esse botão em menos de um segundo, piscando um led verde caso você consiga e um vermelho caso não consiga. Se você conseguir, ele deve escolher outro botão para apontar e você deve apertar corretamente de novo e assim continuar até errar.

5.3. Motor de Passo

Esse é um motor de alta precisão, que rotaciona de acordo com a quantidade de pulsos e ângulos especificados. Esse atuador se constitui por estatores em conjunto que alteram a energização das bobinas de acordo com o sinal especificado pelo microcontrolador. Ele é um circuito de malha aberta, ou seja, não apresenta sensores para dar feedback da posição atual do eixo de rotação.

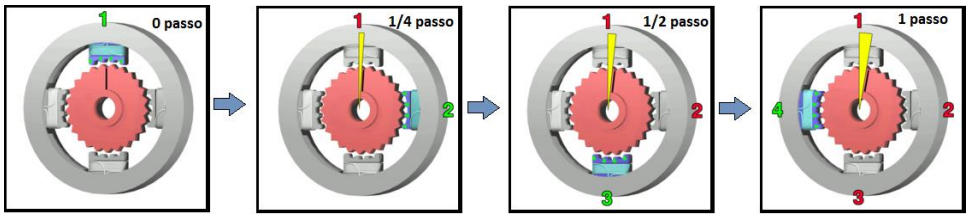


Figura 5.5: Representação da estrutura interna do motor de passos.

Segundo o que se pode ver na figura anterior, de acordo com a polarização de cada solenoide (representada em azul) o conjunto de dentes do rotor central se movimenta para alinhar-se com o centro magnético ativado no momento. Assim, a ativação dos estatores em sequências específicas faz com que se tenha um alto controle sobre esse motor.

Cada movimento acima é chamado de passo, fator que determina o nome do motor. Os diferentes modelos desse atuador possuem diferentes números de passos, sendo que o Motor de Passos 5V, que será utilizado na aplicação dessa apostila, consegue ter 32 passos diferentes, e que, considerando que o sistema completo possui uma caixa de redução de 1/64, totalizam-se 2048 passos necessários para rodar o eixo total do atuador uma única vez, definindo assim uma precisão muito grande em seus movimentos.

A melhor forma de controlar esse dispositivo através do Arduino é por meio da utilização do driver ULN2003, que é um dispositivo que converte os sinais de comando do microcontrolador para sinais de potência enviados ao Motor de Passos 5V.

5.3.1. Aplicação do Motor de Passo

Inicialmente, é importante frisar que o Motor de Passos 5V também necessita de uma fonte externa, ligada ao driver, por demandar uma corrente superior à fornecida pelas portas digitais do Arduino. As ligações são previamente definidas pelo datasheet do driver ULN2003.

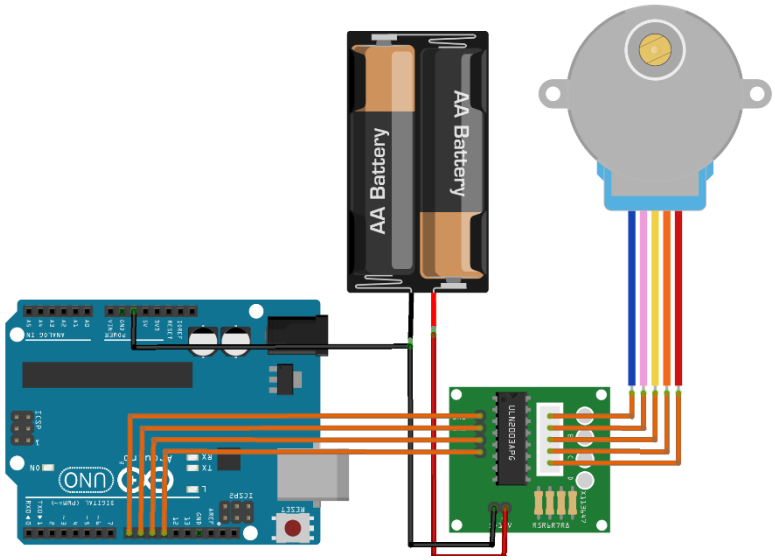


Figura 5.6: Esquemático da montagem no arduino.

No código a seguir iremos incluir a biblioteca do Motor de Passo, e a partir daí configurar o dispositivo para inicialmente dar uma volta completa e depois retornar, no sentido anti-horário, para o estado inicial.

```
#include <Stepper.h> //biblioteca do servo motor

//quanto passos o motor da em uma volta.
int numPassos=32;
// define o número de passos dados e
// as portas digitais usadas
Stepper mp(numPassos,8,9,10,11);

void setup(){
  mp.setSpeed(500); //velocidade do motor
}
void loop(){
  mp.step(2048); //passos que o motor vai dar
  delay(500);
  mp.step(-2048);
}
```

Exercícios de Fixação

1) Motores de passo são muito utilizados por terem bastante torque e uma versatilidade grande para utilização com caixas de engrenagens. Continue exercitando o que aprendeu com motores de passo e crie diferentes padrões de rotação e várias velocidades verificando como ele é realmente forte principalmente em velocidades mais baixas.
2) Simule uma furadeira elétrica com seu motor de passo, ao pressionar o botão o motor deve girar rapidamente como um furadeira e ao soltar o botão ele deve parar.
3) Para verificar a precisão do motor de passo, crie um programa que faça o seu motor dar apenas um passo quando um botão for pressionado.
4) Compare a força do motor de passo com a do motor DC tentando parar o eixo de rotação de ambos. Não force muito pois pode provocar uma sobrecorrente nos motores. Verifique também a velocidade máxima que eles conseguem chegar. Ter essa noção vai te ajudar muito a escolher qual dos motores é mais adequado para cada projeto.

5.4. Pulse Width Modulation (Modulação por Largura de Pulso ou PWM)

Para entender seu funcionamento, vamos imaginar uma lâmpada e um interruptor ON/OFF. Caso o interruptor esteja ligado, a lâmpada estará ligada em sua máxima potência. No caso contrário, a lâmpada estará desligada, sem nenhuma potência. Se controlarmos o tempo em que o interruptor fica ligado e desligado dentro de um período, conseguimos controlar a tensão e potência média enviada para a carga. Por exemplo, caso a chave fique 50% do tempo ligada e 50% desligada, a potência enviada à carga alimentada por esse PWM será de 50%, sendo que quanto maior o tempo ON, maior será a potência, pois há uma maior aproximação com uma tensão contínua ao longo de todo o período. Essa proporção do tempo ON em relação ao período é chamada de Duty Cycle (largura do pulso ON) e é calculada, em porcentagem, da seguinte forma:

$$Duty\ Cycle\ (\%) = \frac{t_{on}}{t_{on} + t_{off}} \cdot 100$$

Sendo assim, dois parâmetros são analisados, a largura do pulso ON (PW) e o período, conforme ilustrado na figura 5.7:

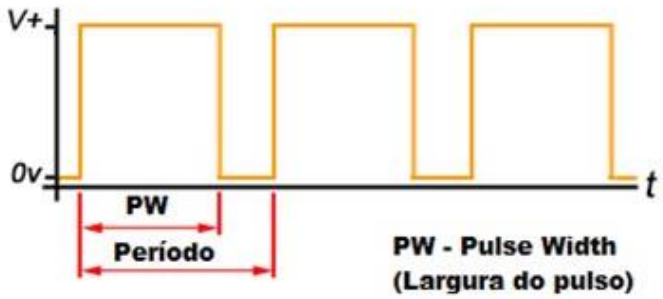


Figura 5.7: PWM de onda quadrada.

Quanto à tensão e potência de alimentação, estas são calculadas a partir da multiplicação do Duty Cycle pela tensão máxima e potência máxima, respectivamente, conforme ilustrado na tabela 5.1:

Tabela 5.1: Relação entre tensão média, potência média e o Duty Cycle

DUTY CYCLE	TENSÃO MÉDIA	POTÊNCIA MÉDIA
0%	0 V	0 W
50%	0,5 $V_{máx}$	0,5 $P_{máx}$
100%	$V_{máx}$	$P_{máx}$

5.4.1. Aplicação: Controle de Velocidade de Motor DC

Após compreender essa relação, vamos entender como o PWM pode ser usado para o controle de velocidade de motores DC.

O motor gira com uma velocidade, em RPM, que é controlada pela potência enviada ao mesmo, ou seja, quanto maior a potência fornecida, maior a velocidade de rotação do motor. Sendo assim, ao alterar o PWM, alteramos a tensão média de alimentação e a potência fornecida que, consequentemente, altera a velocidade de giro. Dessa forma, a relação é explicitada na Tabela 5.2:

Tabela 5.2 - Relação entre potência média e velocidade de rotação do motor.

DUTY CYCLE	POTÊNCIA MÉDIA	VELOCIDADE DO MOTOR (RPM)
0%	0 W	Motor Parado
50%	0,5 $P_{máx}$	Motor girando com metade da sua velocidade de rotação máxima
100%	$P_{máx}$	Motor girando com sua velocidade de rotação máxima

Como os valores de programação do Duty Cycle são analógicos e codificados em 8 bits, significam que variam de 0 a 255. Sendo assim, para um Duty de 100% temos um valor de 255 e para um duty de 40%, temos 102 e assim por diante.

Para aprofundar nessa aplicação, vamos utilizar o Arduino para acionar o motor em velocidade crescente (0% a 100% da velocidade de rotação máxima) e depois decrescente (100% a 0% da velocidade de rotação máxima), sempre no mesmo sentido.

Quanto à montagem do circuito, não podemos apenas conectar o motor DC direto no Arduino, pois o mesmo só possui os terminais de alimentação e não de controle. Para contornar essa situação, vamos utilizar um driver de motor com ponte H (L298N), que permite o controle da velocidade e do sentido de rotação do atuador. Após alimentar o driver e conectar a ele a alimentação do motor, conectaremos o pino de controle em uma das entradas de PWM do Arduino (indicadas por um ~), nesse caso, iremos utilizar a porta 3, conforme a Fig. X3.

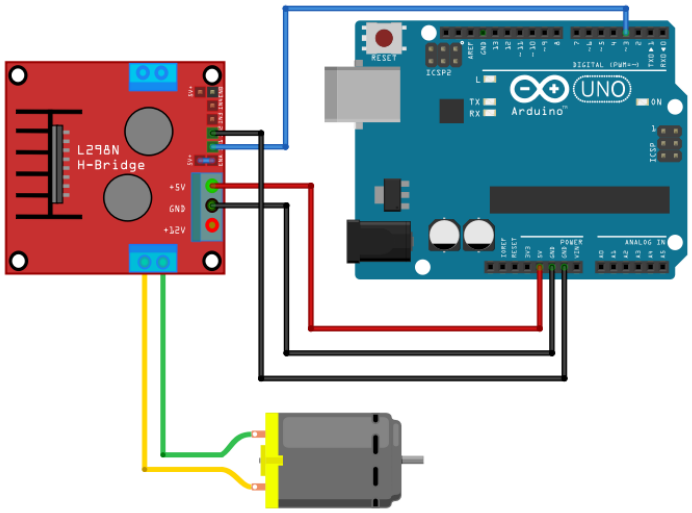


Figura 5.8: Esquemático de ligação.

```
#define pwm_motor 3
// define o pino 3 como sendo o pino (de pwm) do arduino
// que estará conectado no driver L298N, controlando o motor

void setup()
{
  //configura como saída pino terá a ligação para
  // o driver de motor (ponte H) L298N)
  pinMode(pwm_motor, OUTPUT);
}

void loop()
{
  // define a variavel que armazena o valor do duty cycle
  // (0 a 255 que é equivalente a 0% a 100%)da rotação
  // do motor
```

```
  int duty_cycle = 0;
  // 1ª Parte: Velocidade Crescente (0% a 100%)

  // laço contador para o valor do duty cycle
  // indo de 0 até 255, incrementando de um por um
  for (duty_cycle = 0; duty_cycle < 256; duty_cycle++){
    // função que leva o valor do duty cycle ao pino
    // onde está conectado o motor, controlando, assim,
    // a sua velocidade de rotação
    analogWrite(pwm_motor, duty_cycle);

    // tempo em ms de permanência em uma velocidade
    // até passar para a próxima do laço
    delay(300);
  }

  // 2ª Parte: Velocidade Decrescente (100% a 0%)
  // laço contador para o valor do duty cycle,
  // indo de 255 até 0, decrementando de um por um
  for (duty_cycle = 255; duty_cycle >= 0; duty_cycle--) {
    analogWrite(pwm_motor, duty_cycle);
    delay(300);
  }
}
```

Exercícios de Fixação

1) Altere o programa acrescentando outro motor DC. Mantenha um dos motores girando na sua velocidade máxima, enquanto o outro terá sua velocidade reduzida através do PWM.
2) Supondo que cada motor controla uma roda de um carro, o que esse controle provocaria no carro? Nesse caso, qual a vantagem de usar o PWM, ao invés de uma mudança brusca de velocidade?
3) Com o driver motor com ponte H exercite a inversão de rolagem do motor DC.

5.5. Importância de Fontes Externas, Relé e o Transistor como chave

5.5.1. Fontes Externas

Apesar das fontes internas de alimentação dos microcontroladores, como o Arduino, serem adequadas para o funcionamento do mesmo, em certas aplicações, em que utilizamos componentes eletrônicos mais robustos e/ou quase todos os pinos, haverá um consumo de corrente tão elevado que a porta USB poderá não suportar. Isso pode provocar um mal funcionamento do circuito ou até mesmo danificação de alguns componentes.

Diante desse contexto e visando solucionar esse problema, surgem as alimentações externas, que podem ser baterias (normalmente de 9V ou 12V), reguladores de tensão (por exemplo, a família 78XX) ou transformadores.

5.5.2. Relé

Ao se trabalhar com sistemas de automação, é comum a necessidade de se controlar cargas das mais diversas potências, como lâmpadas, motores, resistências e, para isso, são utilizados relés. Paralelo a isso, temos o Arduino que tem a funcionalidade de ativar e desativar suas portas para realizar, também, um possível controle de carga. Então... **"por que não utilizar o próprio Arduino em todas as situações?"**

A resposta está relacionada à robustez elétrica do microcontrolador, que não é capaz de controlar diretamente altas tensões como a de uma lâmpada de 220V, por exemplo. Microcontroladores trabalham com baixas tensões e correntes. Desse modo, os relés se apresentam como uma melhor alternativa.

Esse dispositivo serve para produzir modificações súbitas de estado lógico (1 para 0, ou o contrário), mas de maneira predeterminada. O relé possui um circuito de comando, no qual a alimentação por corrente elétrica aciona um eletroímã, ocasionando a mudança de posição de outro par de contatores que estão ligados a um circuito secundário. A grosso modo, o relé pode ser considerado como

uma chave programável entre os estados lógicos 0 e 1, com a vantagem de poder trabalhar com maiores valores de potência.

Para compreender melhor o funcionamento do relé, veja a figura abaixo:

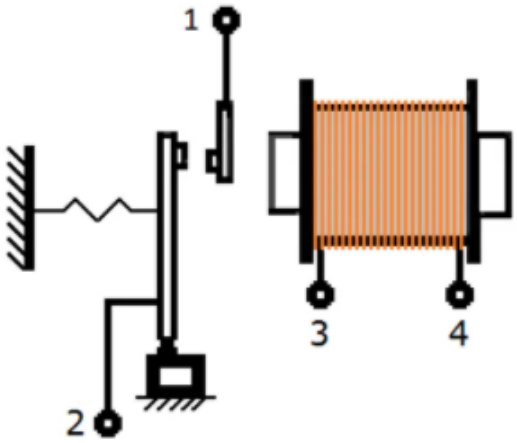


Figura 5.9: Esquemático interno de um relé.

Alimentando os terminais 3 e 4 da bobina, ela gera um campo eletromagnético que atrai a haste conectada ao terminal 2, que vence a força exercida pela mola conectada a ela. Isso faz com que essa haste se mova até encostar na haste conectada ao terminal 1. Com isso, teremos o contato fechado.

Quando paramos de alimentar a bobina através dos terminais 3 e 4, a força magnética exercida sobre a haste conectada ao terminal 2 deixa de existir e a mola puxa a haste de volta ao seu lugar inicial, abrindo o contato. Portanto, agora temos um interruptor acionado por uma bobina.

Usando um relé, podemos controlar um dispositivo, tal como a lâmpada de 220V. A corrente e a tensão que a bobina necessita varia conforme a especificação do relé.

Para trabalhar com Arduino, temos o **módulo relé**, que consiste na junção de um relé com o circuito auxiliar de alimentação (geralmente, a partir de 5V). Na figura a seguir, podemos conferir esse módulo:

Figura 5.10: Módulo Relé Arduino.

5.5.3. Transistor como Chave

Uma das grandes aplicações de transistores é no chaveamento eletrônico de circuitos. Isto é, dependendo do controle de polarização desse componente, podemos controlar o momento em que a chave abre e fecha. Em eletrônica digital, essas duas situações do dispositivo equivalem respectivamente a valores lógicos do tipo 0 e 1 (falso ou verdadeiro). Para atuarem dessa forma, devem estar polarizados ou no modo de corte ou de saturação, como ilustrado na Fig. 5.11.

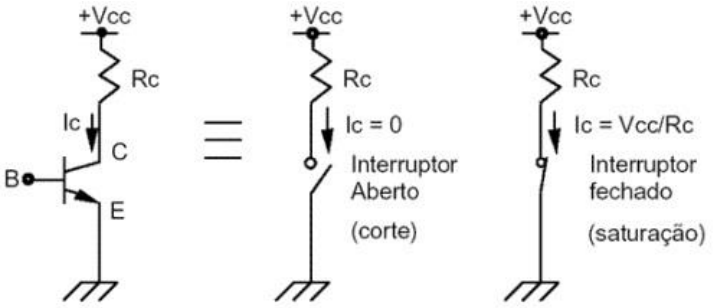


Figura 5.11: Representação do transistor operando como chave aberta e fechada, respectivamente.

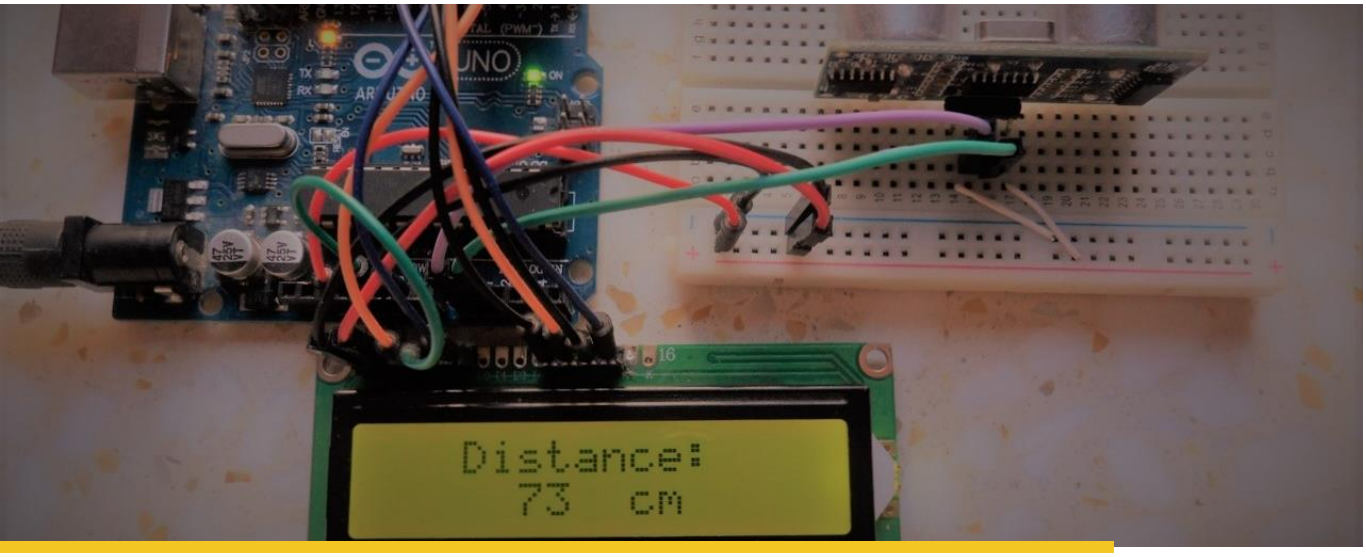
Na zona de corte, no caso do transistor NPN, a tensão na base é menos positiva do que a tensão no coletor e no emissor, sendo assim, o transistor funciona

como um interruptor aberto, uma vez que no coletor a corrente será nula. Logo a tensão entre coletor e emissor, equivale a tensão contínua aplicada sobre ele ($V_{ce} = V_{cc}$). Nesse caso $I_b \cong 0$.

Por outro lado, na zona de saturação a tensão aplicada na base, no caso do transistor NPN, é mais positiva do que a tensão no coletor (cerca de 0.5V maior) e emissor (aproximadamente 0.7V maior), portanto, o transistor funciona como um interruptor fechado. Dessa forma a tensão entre coletor e emissor será em torno de 0.2V e a corrente no coletor atinge seu valor máximo, sendo limitada apenas pelo resistor ligado em série com este.

Exercícios De Fixação

1) Anteriormente utilizamos o transistor como chave para acionar um motor DC. Isso é possível apenas para motores de baixa potência pois transistores possuem uma corrente máxima muito limitada. Tente realizar o acionamento de um motor DC utilizando um Relé, que possui uma corrente máxima muito maior que um transistor TBJ, podendo ser utilizado para várias aplicações de potência maior. Para isto, substitua o transistor do circuito de acionamento de um motor por um relé, conectando sua chave liga/desliga ao Arduino. Neste caso você não precisa conectar o negativo da pilha que estiver utilizando ao GND do Arduino, pois no relé os dois circuitos são totalmente independentes.
2) Desafio! Você aprendeu a utilizar o transistor como chave e como controlar a velocidade de um motor de passo. Mas e se você quiser fazer o motor girar no sentido inverso, como imagina que seria possível? É possível montar um arranjo com transistores capaz de alternar o sentido da corrente elétrica no motor, fazendo-o girar no sentido contrário ou direto, esse arranjo é chamado de ponte H. Pesquise um pouco sobre as pontes H e implemente uma com os transistores BC548, controlando o sentido do motor com o Arduino.



Capítulo 6: Sensores

Entende-se como sensor aquele dispositivo que traduz e informa ao meio digital eventos ou mudança de estados do meio físico, tais como calor, luz, distância, umidade, entre outros. Nós, seres humanos, também somos corpos cheios desses tipos de dispositivos - orgânicos, claro - já instalados e calibrados, sem os quais não conseguiríamos nos manter de pé, nos orientar, perceber que está claro demais, enxergar cores, sentir odores, até perceber materiais quentes que venham a nos causar danos ao toque, etc. Sensores em máquinas são usados de maneira análoga: de braços robóticos em indústrias parando movimentos ao detectar presença de humanos na proximidade, sensores de fumaça na cozinha ou lugares propensos a incêndio para alertar em caso necessário, sensores de proximidade em carros para evitar acidentes na hora de manobrar e até mesmo para o estudo do comportamento do clima em certa região por um período de tempo.

A seguir, algumas aplicações construídas sobre o visto em capítulos anteriores, sendo os exemplos clássicos o acionamento de cargas quando certa condição sensoriada for satisfeita, ou mesmo o controle reajustável de alguma variável em um processo (controle em malha fechada).

6.1. Sensor óptico-reflexivo

O sensor óptico-reflexivo é composto por um emissor e um detector de luz infravermelha, isolados um do outro. O funcionamento do sensor é baseado na detecção da luz infravermelha emitida pelo dispositivo, quando essa é refletida por um objeto dentro do campo de visão efetiva do componente.

Dois dos componentes óptico-reflexivo TCRT5000 sugeridos são mostrados na Figura 6.1.

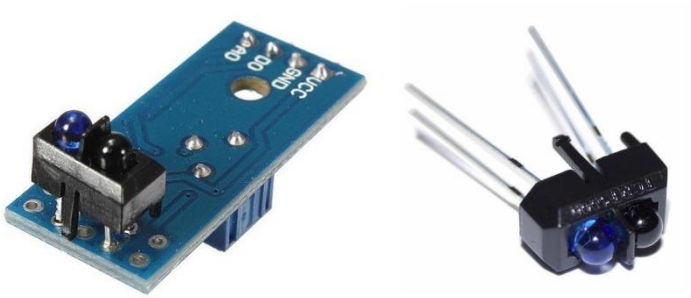


Figura 6.1: Módulo de sensor óptico-reflexivo com shield na esquerda e TCRT5000 sem shield, versão usada na apostila, a direita.

O detector (um fotodiodo) mede a intensidade da luz recebida. Quanto mais forte for o sinal, mais próximo está o objeto, inversamente, quanto mais fraco for o sinal, mais longe. Vamos a um exemplo prático de funcionamento. Para ligar o sensor ao Arduino, vamos precisar de uma protoboard e dois resistores (tipicamente um de 330 Ω para o LED e 5 kΩ para o fotodiodo. Alguns *shields* já vêm com os resistores embutidos e não precisam de adicionais).

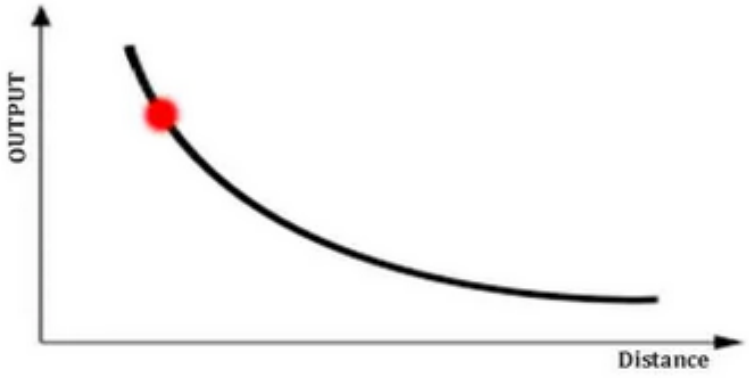


Figura 6.2: Relação esperada do output do sensor pela distância real.

6.1.1. Noções de funcionamento

Abordaremos a seguir um exemplo simples da implementação do IR ("infravermelho", do inglês, "*infrared*"), em que faremos o sistema devolver uma leitura de comportamento semelhante à da Fig. 6.2. O infravermelho deverá estar instalado na protoboard como mostrado na Fig. 6.3 de forma a apontar para campo aberto (sua frente deve estar limpa e ausente de obstáculos). Enquanto o sistema estiver em operação (com o código instalado no Arduino), aproxime um objeto, como uma folha de papel, ao sensor. O *output* poderá ser observado na tela do computador através do *serial plotter*.

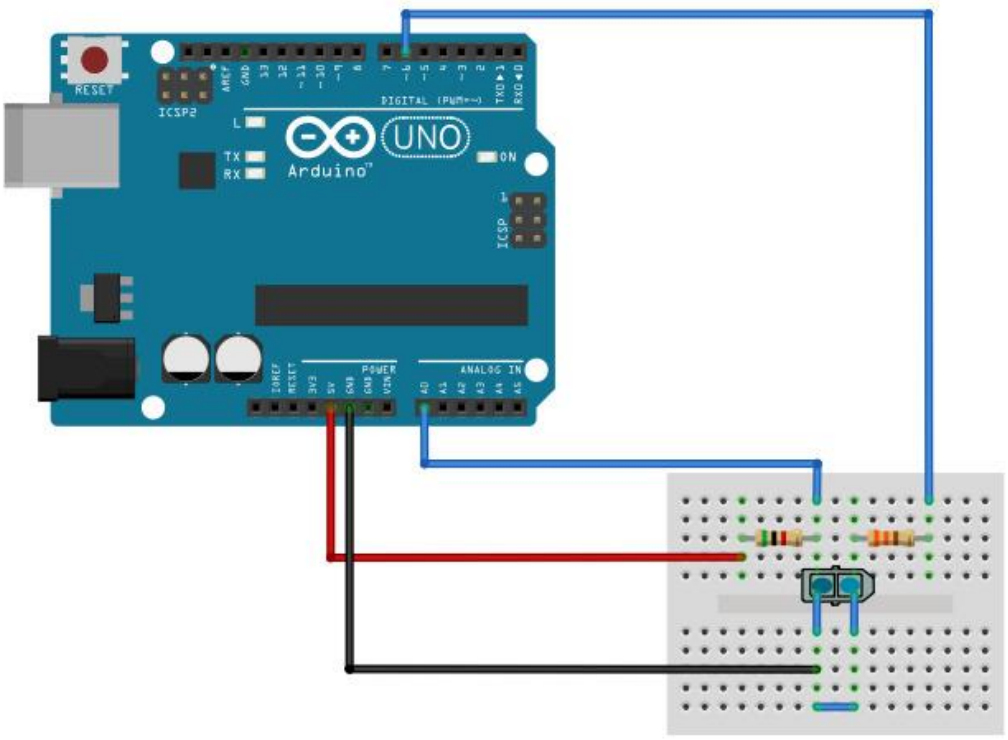


Figura 6.3: Conexões do Arduino UNO e o sensor IR TCRT5000.

O *serial plotter* é uma das ferramentas da IDE do Arduino que pode ler dados contínuos (como a humidade, temperatura ou distância no tempo, por exemplo) e nos permitir visualizá-los como formas de onda. O *serial plotter* pode visualizar não somente uma, mas múltiplas variáveis ao mesmo tempo no mesmo gráfico. A conexão também é feita através do cabo USB. Pode ser ligado no botão Ferramentas – Serial Plotter ou usar o atalho **Crtl + shift +**L.

O plotter possui uma caixa para display os dados com duas dimensões. O *X-axis* (eixo-X) representa a passagem do tempo, temos nesse eixo 500 pontos, e cada ponto é preenchido a cada chamada da função **Serial**.println() e a variação de tempo entre cada ponto geralmente é igual ao tempo do loop em que está inserida. O *Y-axis* (eixo-Y) representa a amplitude dos dados fornecidos e é dinamicamente ajustado conforme os dados crescem ou decrescem em valor.

Agora, podemos movimentar um objeto na área de ação do IR. Assim, o computador imprime a percepção de distância do objeto pelo dispositivo e o gráfico deve se alterar conforme a posição do objeto mudar.

```
void setup() {
  Serial.begin(9600);
  pinMode(6,OUTPUT);
  pinMode(A0,INPUT);
}
void loop() {
  digitalWrite(6, HIGH);
  // LED emissor pronto
  delay(500);
  value = analogRead(A0);
  Serial.println(value);
  // Lendo o valor obtido no pino A3
}
```

Vamos observar como o sistema funciona. Primeiro, claro, estabelecemos a comunicação serial e os pinos a serem usados. Em seguida, no loop, fazemos o pino 6 enviar sinal (acendendo LED) e a cada meio segundo, aproximadamente, o Arduino lê o sinal recebido pelo fotodiodo (mais especificamente, a tensão sobre seus terminais) e envia para o monitor serial por meio da função **Serial**.println**()**.

Se na sua sala bem iluminada, de janelas fechadas, você fizer o teste, os resultados serão consistentes. Ótimo, então nosso sistema funciona como deveria, certo? Não. O sistema está incompleto. Se apagar a luz ou abrir a janela e receber a luz solar, por exemplo, perceberá a leitura apresentados resultados diferentes. O mesmo fenômeno acontece ao se mover o sistema para áreas externas.

É importante manter em mente que o detector percebe não somente a reflexão do IR proveniente do dispositivo, como também faixas da luz visível do ambiente. Outras fontes de luz infravermelha também impactarão na recepção dos sinais sob a forma de ruído. Fontes tais como a luz solar, luz proveniente de lâmpadas acesas, materiais aquecidos ou emissores de calor, entre outras fontes que emitam IR. Surge, a partir desta constatação, a necessidade de filtrar o ruído, caso contrário seria necessário recalibrar o sistema toda vez que as condições do ambiente mudam.

A maneira mais simples de fazer a filtragem do ruído é realizar duas observações rapidamente a cada rodada do loop: leitura com LED do sensor *on* e leitura com LED *off* e subtrair da primeira a segunda. Essa abordagem pode não entregar os resultados mais precisos, porém é uma maneira simples e eficaz de retirar o ruído da observação.

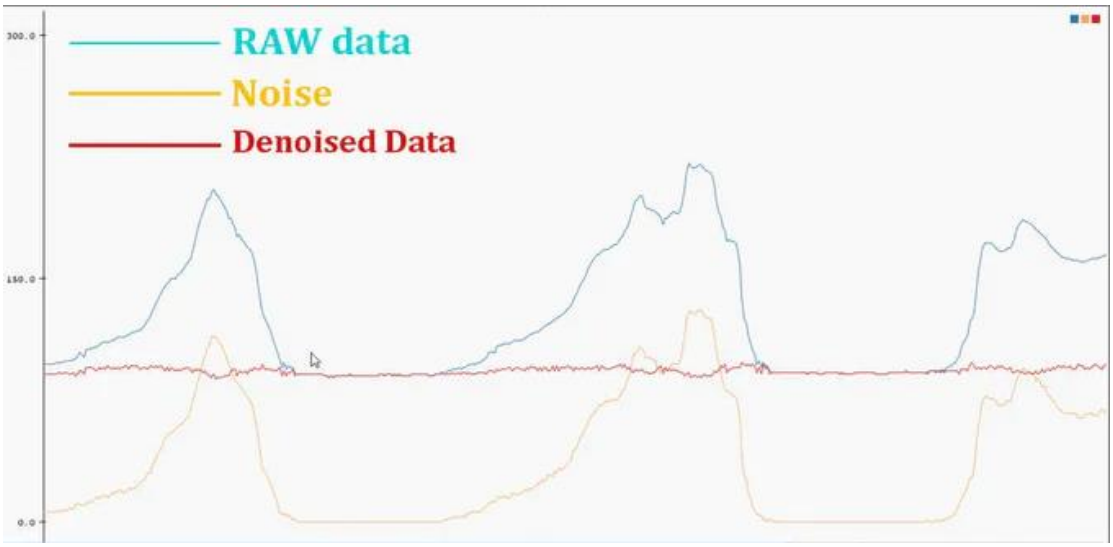


Figura 6.4: Leitura real, ruído e leitura com ruído subtraído.

Abaixo, a implementação do código de filtragem simples que dá origem a um gráfico de funcionamento semelhante ao da Fig. 6.4.

```
void setup() {  int sinal,ruido,filtrado;
  // sinal: IR emitido pelo dispositivo
  Serial.begin(9600);
  pinMode(6,OUTPUT);    // LED
  pinMode(A0,INPUT);    // Sensor
```

```
}
void loop() {
  digitalWrite(6, HIGH);
  delayMicroseconds(100);
  sinal = analogRead(A0);

  digitalWrite(6,LOW);
  delayMicroseconds(100);
  ruido = analogRead(A0);

  filtrado = sinal - ruido;
  Serial.println(filtrado);  // Imprimir sinal filtrado
}
```

Algumas ideias de aplicação para este sensor incluem: a verificação de velocidade de rotação do rotor de um Motor DC (vide capítulo 5); o destravamento de uma tranca ao se verificar proximidade de alguém, acionando um relé através de um transistor (vistos nos capítulos 4 e 5); alertar um robô, como seguidor de linha, sobre a iminência de colisão ou da distinção entre branco e preto.

6.1.2. Atenção ao usar o sensor óptico-reflexivo

Como vimos, é necessário estarmos atentos ao ambiente em questão e desenvolvermos um algoritmo de filtragem que se adapte bem à situação envolvida.

Devido à natureza de seu funcionamento, não é eficaz contra objetos transparentes ao IR, então isso também deve ser levado em consideração.

O Datasheet deste dispositivo específico (TCRT5000) informa que a distância eficaz de operação é 15 mm. Sendo assim, o dispositivo não é empregável em aplicações cuja medição de distância requeira maiores medidas.

Exercícios De Fixação

1) O sensor IR trabalha diretamente com o conceito da reflexão da luz, e esta por sua vez é bastante afetada pelo objeto que está refletindo essa luz. Posicione objetos com diferentes tonalidades de cor (dos mais claros aos mais escuros) e pretos em frente ao sensor de presença e verifique como ele se comporta e se existe alguma diferença notável.

2) Utilize junto com o sensor IR, um display LCD para imprimir se o sensor está obstruído, com algum objeto na sua frente.
3) Utilize um sensor IR para simular um sistema de segurança em uma residência, que ao passar algum objeto (ou pessoa) próximo ao sensor deve aparecer um alarme através de LEDs que devem piscar.
4) Utilize o sensor IR para parar um motor quando um objeto estiver a sua frente. Você pode controlar o motor através de um potenciômetro, mas quando o sensor detectar um objeto a sua frente, deve parar imediatamente o motor.
5) Você pode utilizar um sensor de presença para contar quantas vezes um objeto passou pela frente, assim instalando em uma linha de montagem para contar peças, em uma porta para contar quantas vezes ela foi aberta ao longo do dia... Implemente um contador que irá contar sempre que um objeto passar pelo sensor, pisque um led sempre que detectar uma passagem. Se o objeto ficar parado na frente do sensor, seve ser contado apenas uma vez.
6) Desafio! Construa um sistema rapaz de medir a velocidade de um Motor DC, anexando ortogonalmente ao seu rotor um papel leve. O sensor irá contar quantas vezes o papel passa por ele a cada segundo, imprimindo a velocidade (em rotações por segundo) no monitor serial ou em algum tipo de display. Compare a medição da velocidade de rotação das hélices de um ventilador com o sensor ultrassônico e o IR.
7) Desafio 2! Com o sistema do desafio anterior montado, ligue um display LCD mostrando na primeira linha a quantidade de rotações por segundo do motor com a leitura do sensor. Adicione dois botões à montagem para aumentar ou diminuir a velocidade do motor como o usuário desejar, e essa velocidade desejada deve ser impressa na segunda linha do display LCD. O seu código deve ser capaz de ajustar a velocidade do motor (mostrada na primeira linha) até que a velocidade lida pelo sensor seja igual àquela configurada pelo usuário a partir dos botões (impressa na segunda linha).

6.2. Sensor de distância

O sensor de distância, como o próprio nome sugere, se destina a estabelecer uma métrica entre o elemento sensor e um determinado obstáculo. Nessa categoria, se destacam os sensores ultrassônicos, sobretudo para pequenas distâncias, com aplicação bastante comum em carrinhos e semelhantes. Para distâncias um pouco maiores, apesar de existirem ultrassônicos mais robustos que podem ser utilizados, costuma-se preferir a aplicação de sensores com a tecnologia de infravermelho (vide seção 6.1).

Um exemplo bem conceituado é o Sensor Ultrassônico HC-SR04, capaz de medir distâncias entre 2 cm e 4 m com precisão de 3mm. O sensor funciona através do trabalho conjunto de um emissor e receptor de ondas ultrassônicas que atingem o alvo, são refletidas e captadas novamente pelo sensor, calculando pelo tempo de retorno a distância até o objeto. A Fig. 6.5 Apresenta um exemplo desse modelo de sensor.

Figura 6.5: Módulo de sensor ultrassônico.

Analisando a fig. 6.5, percebemos os elementos que constituem o modulo sensor como: Emissor de sinal ultrassônico à esquerda, identificado pela letra T; Receptor de sinal ultrassônico à direita, identificado pela letra R; Cristal Oscilador de Quartzo para geração de clock e, consequentemente, contagem de tempo, com frequência de 4000 MHz, identificado por Y1 na região superior central; Quatro pinos para uso do sensor: Alimentação positiva e negativa nos pinos VCC e GND respectivamente, e pinos de sinal trigger e echo, responsáveis respectivamente pelo

controle do envio e recebimento dos sinais ultrassônicos (vide Fig. 6.6), vale ressaltar que o range de envio das ondas é de 15º para esse modelo.

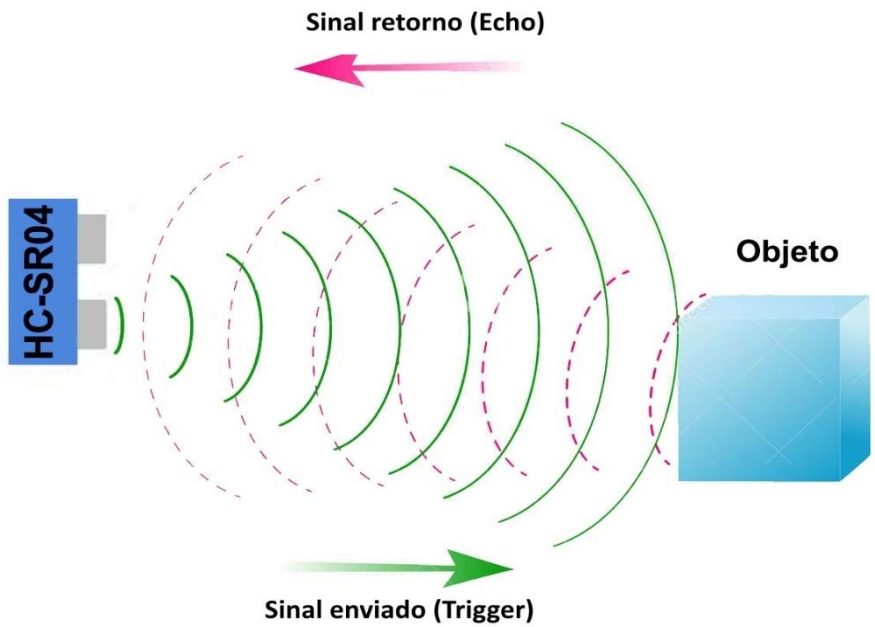


Figura 6.6: Módulo de sensor ultrassônico.

Este sensor possui uma biblioteca amplamente utilizada para facilitar o seu uso no Arduino. As especificações de trigger e echo são, portanto, detalhes que podem ser reservados aos usuários mais curiosos ou para aplicações mais complexas. A saber, o sensor calcula a distância através de um pulso de 10 µs ativado pelo trigger, em seguida são emitidos 8 pulsos de 40 KHz através do Emissor e, finalmente, o Receptor aguada em nível alto o retorno das ondas refletidas. O cálculo da então se faz Distância = (tempo echo em nível alto * velocidade do som)/2. Vide fig. 6.7 para ilustração.

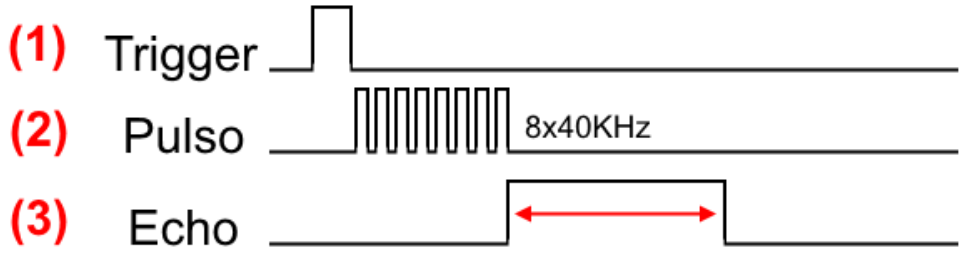


Figura 6.7: Sinais do sensor ultrassônico HC-SR04.

6.2.1. Aplicação do Sensor de distância

Imagine que você está construindo um carrinho seguidor de linha para uma competição de robótica na escola, o trajeto inclui obstáculos que precisam ser desviados. Para tal, é necessário identificar a distância até esses obstáculos, identificação essa que pode ser feita através do sensor HC-SR04.

Façamos inicialmente as ligações físicas no Arduino. Basta seguir o esquemático da Fig. 6.8. Serão usadas duas portas digitais do Arduino para o trigger e o echo, nas portas digitais 11 e 12 do Arduino, respectivamente, e a alimentação de 5V.

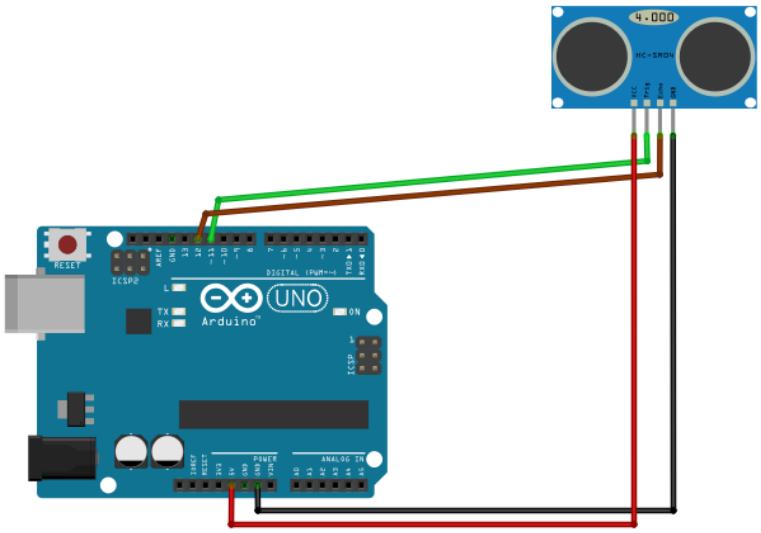


Figura 6.8: Ligação física do Sensor Ultrassônico HC-SR04.

Para o código usaremos a biblioteca Ultrasonic.h conforme código abaixo para fazer a leitura da distância captada pelo sensor em centímetros a cada meio segundo.

```
//Carrega a biblioteca do sensor ultrassonico
#include <Ultrasonic.h>
//Define os pinos para o trigger e echo
#define pino_trigger 11
#define pino_echo 12
//Inicializa o sensor nos pinos definidos acima
Ultrasonic ultrasonic(pino_trigger, pino_echo);
void setup(){
  Serial.begin(9600);
```

```
  Serial.println("Lendo dados do sensor...");
}

void loop(){
  //Le as informacoes do sensor em cm
  float cmMsec;
  long microsec = ultrasonic.timing();
  // Método pertencente à biblioteca usada para a
  // leitura de dados do sensor. Esse método acessa a
  // informação de tempo medido entre a emissão do
  // sinal e seu retorno

  cmMsec = ultrasonic.convert(microsec, Ultrasonic::CM);
  // Este método converte automaticamente a informação de
  // diferença de tempo alocada na variável longa microsec
  // em uma medida de distância em centímetros

  //Exibe informacoes no serial monitor
  Serial.print("Distancia em cm: ");
  Serial.print(cmMsec);
  delay(500);
}
```

Podemos usar desse sensor como uma ferramenta para manter registro de um sistema de contagem em uma esteira, por exemplo. Imagine que em determinado processo de produção se pretende encaminhar a outro setor caixas com objetos para serem lacradas. Cada caixa deve conter exatos 9 objetos que estão sendo transportados através de uma esteira. Para evitar de manter contagem visual, o que seria, além de extremamente entediante, um sistema altamente suscetível a falhas, podemos usar o sensor ultrassônico para construir um sistema de contagem.

Iremos aproveitar o circuito que fizemos para leitura dos dados do sensor no monitor serial, inclusive porque podemos usar essa leitura para acompanhar se o funcionamento do sensor está de acordo. Faremos então a conexão de um display de 7 segmentos para acompanhar em tempo real o processo de leitura do sensor. As conexões desse sensor devem acompanhar o esquemático da Fig. 6.9 quando usando um display catodo comum.

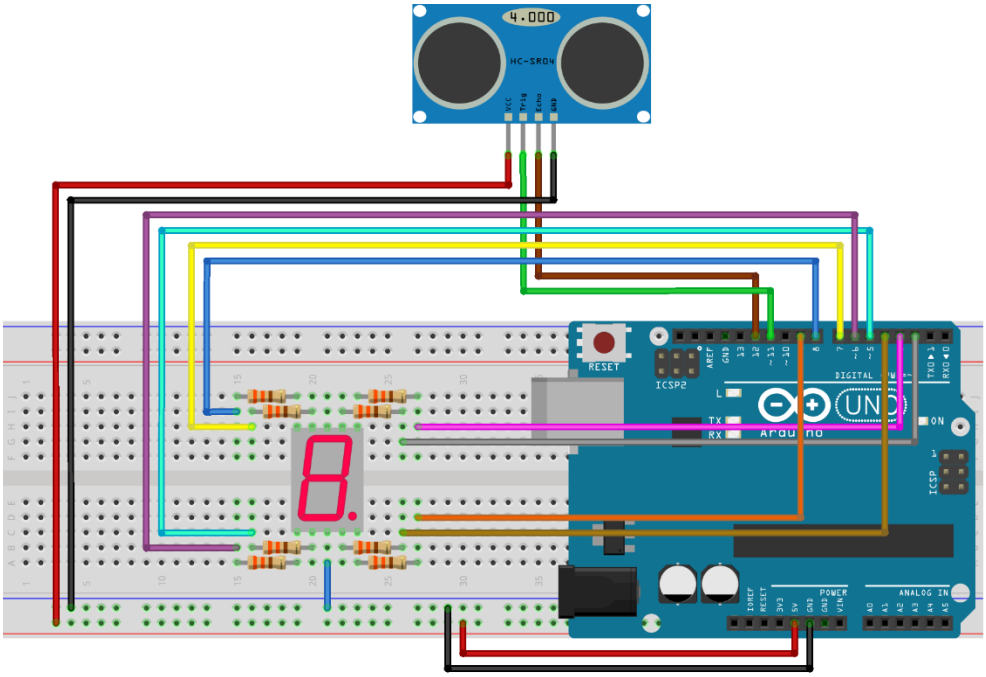


Figura 6.9: Ligação física com Sensor Ultrassônico HC-SR04 e display de 7 segmentos

Importante atenção ao tipo de display utilizado. Um teste simples e rápido pode ser realizado aplicando uma tensão de 5V entre o terminal do resistor e o último pino no canto inferior direito, caso o ponto do display acenda, trata-se de um display do tipo anodo comum, em caso contrário se trata de um display do tipo catodo comum. Finalmente, basta acompanhar os seguintes acréscimos ao código:

```
//Carrega a biblioteca do sensor ultrassonico
#include <Ultrasonic.h>

//Define os pinos para o trigger e echo
#define pino_trigger 11
#define pino_echo 12

//Inicializa o sensor nos pinos definidos acima
Ultrasonic ultrasonic(pino_trigger, pino_echo);

//Configurando display de 7 segmentos
const int l=1, d=0; //Display com catodo comum

byte disp7seg[10][7] = {
  //Identificação de cada dígito do display
  { l,l,l,l,l,l,d },  //DIGITO 0
  { d,l,l,d,d,d,d },  //DIGITO 1
  { l,l,d,l,l,d,l },  //DIGITO 2
```

```
  { l,l,l,l,d,d,l },  //DIGITO 3
  { d,l,l,d,d,l,l },  //DIGITO 4
  { l,d,l,l,d,l,l },  //DIGITO 5
  { l,d,l,l,l,l,l },  //DIGITO 6
  { l,l,l,d,d,d,d },  //DIGITO 7
  { l,l,l,l,l,l,l },  //DIGITO 8
  { l,l,l,d,d,l,l }   //DIGITO 9
};

int num1 = 0;
// Referência para identificar qual número está em exibição

void setup(){
  Serial.begin(9600);
  Serial.println("Lendo dados do sensor...");
  for (int i = 2; i < 10; i++){
    pinMode(i, OUTPUT);
    // configurando pinos destinados ao display como saídas
  }
}

void loop(){  //Lê as informacoes do sensor
  float cmMsec;

  long microsec = ultrasonic.timing();
  // Método pertencente à biblioteca usada para a leitura
  // de dados do sensor. Esse método acessa a informação
  // de tempo medido entre a emissão do sinal e seu retorno

  Distancia = ultrasonic.convert(microsec, Ultrasonic::CM);
  // Este método converte automaticamente a informação
  // de diferença de tempo alocada na variável longa
  // microsec em uma medida de distância em centímetros

  //Exibe informacoes no serial monitor
  Serial.print("Distancia em cm: ");
  Serial.print(distancia);
  if (distancia <= 30){
    //Verifica se existe algum objeto próximo ao sensor
    num1 += 1;
    while(distancia>30){}
     //Caso não exista objeto, aguarda que surja e
     // evita contagem descontrolada
  }
```

```
  //Evitando estouro de contagem
  if (num1 < 0) {num1 = 9;}
  if (num1 > 9) {num1 = 0;}

  //Exibindo numeração no display
  for (int j = 2; j <9; j++) {
    digitalWrite(j, disp7seg[num1][j-2]);
  }
  delay(500);
}
```

6.2.2. Atenção ao usar o sensor ultrassônico

O sensor ultrassônico pode ser muito eficaz, se for empregado dentro do que foi projetado para fazer. Devido à natureza de seu funcionamento, devemos estar atentos ao material dos objetos que poderão passar dentro deu seu campo de visão, o tamanho dos corpos e que seus posicionamentos podem interferir na leitura dos dados.

Materiais como tecidos, carpete, lã de vidro, borracha, i,e, materiais isolantes, acústicos irão comprometer o retorno dos dados, já que absorvem as ondas sonoras.

O tamanho dos objetivos também pode apresentar problemas: objetos pequenos podem não refletir o suficiente para ser detectado, ou mesmo podem acabar refletindo em direções diferentes. O mesmo problema é apresentado para radares profissionais, como os militares. É dito que uma nave é invisível quando seu rastro sonoro, térmico ou outro seja tão pequeno que é comparável ao de um pássaro — efetivamente invisível.

Exercícios De Fixação

1) Na montagem padrão do sensor ultrassônico, verifique fazendo testes quais as maiores e menores distâncias que esse sensor consegue medir. Faça comparações com uma trena ou fita métrica para testar a precisão do sensor.

2) Utilize junto com o sensor ultrassônico, um display LCD para imprimir a distância que o sensor está captando de algum objeto na sua frente.
3) Monte um sistema que simula o aviso de estacionamento de carro com sensor ultrassônico, que conforme o sensor vai se aproximando, uma LED pisca de maneira mais frequente, alertando o motorista.
4) Utilize o sensor de distância para controlar a velocidade de um motor da seguinte forma: se não há nenhum obstáculo a frente, o motor pode girar em velocidade máxima, à medida que se aproxima um obstáculo, o motor deve ir reduzindo até que o obstáculo esteja muito próximo, parando o motor.
5) Incremente a prática da seção 6.2.1 colocando um botão para resetar o contador a zero em qualquer momento que pressionado durante a contagem feita pelo sensor ultrassônico. Acrescente também um buzzer para irritar pessoas próximas sempre que um objeto for contado.

6.3. Sensor de Umidade e Temperatura

Por mais que se tratem de duas informações distintas, comumente se encontram sensores disponíveis que interrelacionam essa as variáveis Umidade e Temperatura em um mesmo componente. O DHT11 é o modelo de sensor mais comum em aplicações básicas e aprendizado. Para aqueles mais necessitados de precisão, o DHT22 é uma variação ainda de sensor para aplicações básicas, porém dada sua disponibilidade financeira e necessidade de acurácia, sondas para esses fins usadas em pesquisas e condições extremas podem chegar ao custo de centenas de milhares de dólares. Mas de fato, trataremos aqui de leituras simples com o DHT11, presente na Fig. 6.10, e seus princípios de funcionamento.

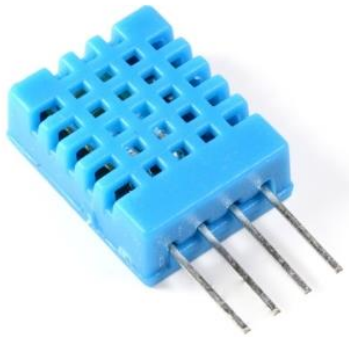


Figura 6.10: Sensor de umidade e temperatura DHT11.

O DHT11 é capaz de medir temperatura entre 0 e 50 ºC e umidade entre 20 e 90% UR. Sua precisão é de respectivamente 2ºC e 5%, o que não configura precisão suficiente para aplicações extremamente sensíveis, mas pode ser usado em ambientes onde uma aproximação da temperatura local já é informação suficiente e onde os limites de temperatura atingíveis são conhecidos. A alimentação desse sensor varia de 3 a 5V, podendo ser usado desde placas como o NodeMCU de saída 3,3V até as convencionais de 5V sem necessidade de regulação externa de tensão. O tempo mínimo necessário para atualização dos dados desse sensor é de 2s, portanto, deve-se prever um delay de pelo menos 2000ms para que as leituras sejam atualizadas.

Para entendermos o funcionamento desse sensor, precisamos recapitular alguns conceitos, afinal sabemos em relação ao sensor tanto quanto sabemos em relação à propriedade física medida.

A umidade medida é a umidade relativa do ar, que seria a relação entre a quantidade de vapor de água presente no ar, considerada umidade absoluta, e a quantidade de vapor de água máxima possível de existir para uma mesma dada temperatura, denominada ponto de saturação. O ponto de saturação depende diretamente da temperatura, uma vez que o ar frio pode conter menos vapor de água antes da saturação e o ar quente justamente o contrário. Na saturação a água passa a se condensar e se acumular na forma de orvalho. Para calcular essa umidade relativa, podemos utilizar a equação:

$$RH = (Pw / Ps) * 100\%$$

RH → umidade relativa;
Pw → densidade de vapor no ar;
Ps → densidade de vapor no ponto de saturação.

A forma que o sensor age é justamente detectando o vapor de água através da medição de resistência entre dois eletrodos, dessa forma, uma mudança na resistência elétrica entre esses eletrodos é proporcional à umidade relativa. Quando a umidade relativa é alta, a resistência diminui e quando a umidade diminui, a resistência aumenta.

Quanto à temperatura, a análise pode ser bastante simplificada: trata-se também de sensoriamento resistivo, nesse caso a relação entre a resistência e a temperatura é bem mais direta. No DHT11 é utilizado um sensoriamento do tipo coeficiente de temperatura negativa ou NTC, dessa forma, a resistência diminui com o aumento da temperatura.

Essas informações, tanto de umidade quanto de temperatura, são então codificadas em um único pino através de um protocolo em 8bits, onde uma

biblioteca específica já direcionada a esse fim, facilmente decodifica e acessa essas informações.

6.3.1. Aplicação do Sensor de Umidade e Temperatura

As aplicações mais comuns desse tipo de sensor são estufas de pequeno porte, sistemas de irrigação, automação residencial, mas quando se trata de sensores, o limite é sempre apenas o da imaginação. Trataremos aqui de uma leitura novamente de informações, porém sem a necessidade do monitor serial, usando um display LCD. Essa aplicação é deveras útil ao se tratar de aplicações de campo para monitoramento local. Vale ressaltar que ainda é perfeitamente possível, e fica como treino, o acionamento de cargas quando certas condições de umidade e temperatura forem atingidas. Dica: use "E"s ou "OU"s em "IF"s para ligar ou desligar alguma coisa.

Para essa aplicação, usaremos o Arduino Uno, coletaremos informações com o Sensor DHT11, exibiremos essas informações através de um Display LCD de 16x2, cuja visibilidade será ajustada através de um Potenciômetro de 10k e conexões facilitadas com Jumpers. Faça as conexões conforme fig. 6.11.

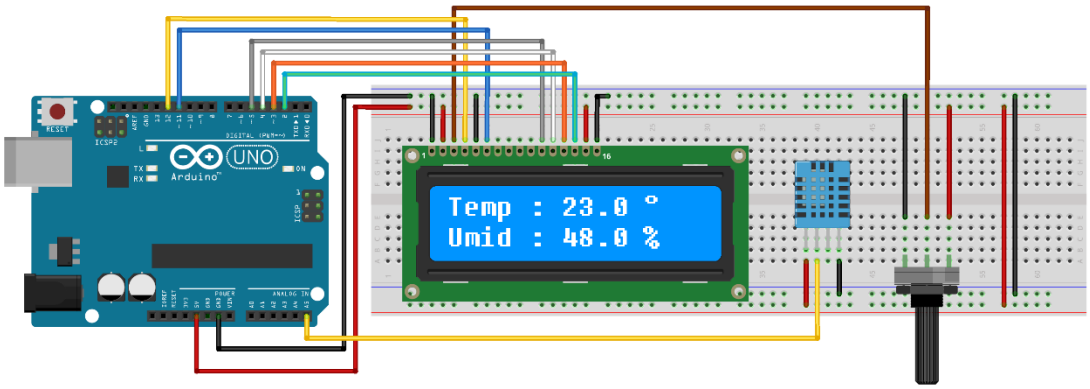


Figura 6.11: Ligação física do Sensor DHT11 com Display LCD.

O código para essa aplicação já não deve ser estranho aos leitores nesse ponto da apostila, siga atentamente o código para sanar quaisquer dúvidas ou ajustes que deseja fazer na montagem. Se preferir, retorne um pouco para revisar o Display LCD.

```
#include <LiquidCrystal.h>
//Biblioteca dedicada ao display LCD
#include <DHT.h>
//Biblioteca dedicada ao sensor

//Definicao do pino dedicado ao sensor
#define DHTPIN A5

//Definicao do modelo do sensor
#define DHTTYPE DHT11 DHT dht(DHTPIN, DHTTYPE);

//Pinos usados pelo Display
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

//Para o simbolo de grau, usaremos um caractere personalizado
byte grau[8] = {
  B00001100,
  B00010010,
  B00010010,
  B00001100,
  B00000000,
  B00000000,
  B00000000,
  B00000000
};

void setup(){
  Serial.begin(9600); //Inicializa a serial
  lcd.begin(16,2); //Inicializa LCD
  lcd.clear(); //Limpa o LCD

  //Cria o caractere personalizado com o simbolo do grau
  lcd.createChar(0, grau);
  dht.begin();
}

void loop(){
  float h = dht.readHumidity(); //Le o valor da umidade
  float t = dht.readTemperature();

  //Le o valor da temperatura
  lcd.setCursor(0,0); //Reposiciona o cursor para escrita
  lcd.print("Temp : ");
  lcd.print(" ");
  lcd.setCursor(7,0); //Reposiciona o cursor para escrita
  lcd.print(t,1);
  lcd.setCursor(12,0); //Reposiciona o cursor para escrita
```

```
  //Mostra o símbolo do grau formado pelo array
  lcd.write((byte)0);
  lcd.setCursor(0,1); //Reposiciona o cursor para escrita
  lcd.print("Umid : ");
  lcd.print(" ");
  lcd.setCursor(7,1); //Reposiciona o cursor para escrita
  lcd.print(h,1);
  lcd.setCursor(12,1); //Reposiciona o cursor para escrita
  lcd.print("%");

  // Intervalo recomendado para leitura do sensor visto o
  // tempo necessário para atualização
  delay(2000);
}
```

Exercícios De Fixação

1) Os ar-condicionado funcionam medindo a temperatura da sala onde está instalado e quando ela alcança a temperatura desejada, ele desliga o aparelho. Com o sensor de temperatura, faça com que o Arduino desligue o ar-condicionado (você pode simular esse acionamento com um led ou com um relé) quando o quarto chegar nessa temperatura e legando quando esquentar dois graus a mais do que o desejado.
2) Em cidades frias é muito comum o uso de chuveiros elétricos para que a água do banho fique quente mesmo em ambiente frio. Com o sensor de temperatura é possível automatizar a escolha da temperatura do chuveiro elétrico. Desse modo, faça um programa que o sensor de umidade e temperatura ajuste o valor de temperatura do chuveiro elétrico, de modo que, em dias frios, o chuveiro deve ser ativado para esquentar a água.
3) Imagine agora que você quer controlar o seu mais novo umidificador ultrassônico para o seu quarto, fazendo ele acionar sempre que o clima estiver muito seco. Faça um programa que acione um relé (que você pode conectar ao seu umidificador de ar) sempre que a umidade relativa estiver menor que 55% e desligar quando chegar em 60%.

6.4. Sensor de Nível

Em diversos projetos pode vir a ser conveniente ou indispensável saber o nível de algum líquido. Na indústria química, a exemplo, diversos processos exigem que uma precisa quantidade de um líquido seja adicionado ou armazenado para uma dada reação ocorrer da forma esperada. Aplicações mais simples podem ser observadas em cenários de automação residencial. Vale ressaltar ainda a validade desse tipo de informação no estudo de técnicas de controle por estudantes de engenharia.

A Fig. 6.12 apresenta um exemplo de sensor de nível facilmente encontrado quando ao se deparar com esse problema à luz de Arduino. Trata-se de módulo capaz de perceber a presença de água ao longo de uma certa área de contato. Por conta desse formato, pode também ser considerado um sensor de chuva. De qualquer forma, o sensor percebe a presença de água e retorna ao microcontrolador um sinal analógico correspondente e proporcional à quantidade da superfície que está em contato como líquido.

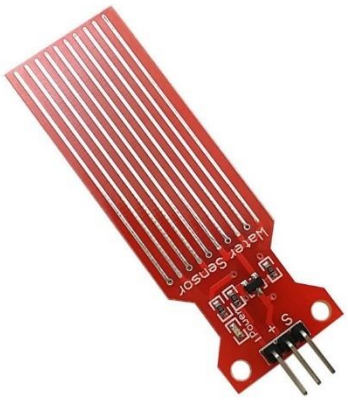


Figura 6.12: Sensor de Nível/Chuva.

A superfície do sensor é bastante condutiva, além de possuir proteção contra oxidação, portanto, vale ressaltar que os pinos para conexão, assim como os componentes eletrônicos externos à superfície condutiva do sensor, devem ser protegidos da água para evitar curtos.

6.4.1. Aplicação de Sensor de Nível

Mesmo já tendo comentado possíveis aplicações da utilização desse tipo de sensor, como automação residencial, controle de nível e técnicas de aprendizado, é importante apontar a limitação da versão comercial mais comumente encontrada contar com uma pequena área sensível. Caso não seja possível adequar seu projeto a essa limitação, é sempre possível usar um conjunto de boias como sensores para espaços maiores.

Aqui faremos uma aplicação simples de leitura e acionamento de cargas baseado nas condições lidas pelo sensor. Ligaremos um LED quando a leitura do sensor atingir um determinado valor. Esse LED pode simbolizar um relé que ativa uma bomba para encher ou esvaziar um recipiente, por exemplo. O esquemático da figura 6.13 apresenta as ligações a serem feitas. O código para essa implementação segue a mesma simplicidade do circuito.

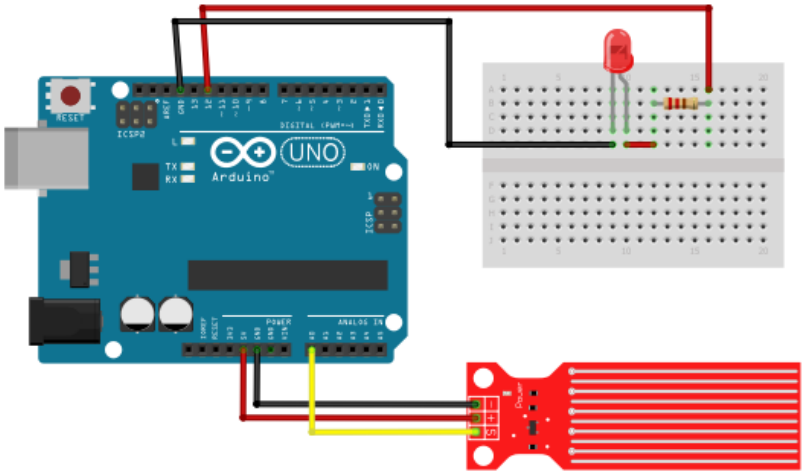


Figura 6.13: Ligação física do Sensor de Nível com carga qualquer representada por LED.

```
const int pinoSensor = A0; //Sinal para recepção do sensor
const int pinoLED = 12; //Pino usado para o LED
void setup(){
  //Serial.begin(9600); //Inicializando Serial
  pinMode(pinoSensor, INPUT); //
  pinMode(pinoLED, OUTPUT); //
}
```

```
void loop(){
  if(analogRead(pinoSensor) > 690){
    //Condicao para ativacao
    digitalWrite(pinoLED, HIGH); //Ativação
  }
  else{ //Condicao de desativação
    digitalWrite(pinoLED, LOW); //Desativação
  }
}
```

Exercícios De Fixação

1) Caixas d'água são controladas por meio de uma boia que fecha o encanamento ao chegar em um certo nível. Defeitos na boia causam um extravasamento da caixa d'água fazendo a água sair pelo ladrão. Imagine que você quer criar um alarme para avisar os moradores quando houver transbordamento ao colocar um sensor na beirada da caixa então crie um código que irá apitar um buzzer continuamente e piscar leds, que podem ser posicionados por toda a casa, quando o sensor detectar água chegando nos limites da caixa.
2) Sistemas de plantações automatizados precisam saber qual o nível de água e umidade está o solo que as raízes da planta se encontram, para isso, um sensor de nível consegue dizer se tem água ou não naquele solo. Portanto, faça um projeto que o sensor de nível indique se tem água nele, e caso não tenha, faça ele ativar uma LED que avisa que o solo precisa de água.
3) Em um grande instituto meteorológico existem vários medidores de nível de água da chuva espalhados pela área externa do instituto, para verificar o nível, o responsável técnico deve sair de seu escritório anotar os dados dos medidores e retirar a água deles. Sendo assim, faça um projeto que, ao pressionar um botão, o sensor de nível capta o nível da água e transforma para o volume correto, mande o valor no serial. Depois disso, ative um servo motor que abrirá a tampa que segura a água em 90° para a saída e aula.

6.5. Sensor de batimentos cardíacos

Poder extrair informações do corpo humano abre portas para diversas possibilidades de projetos interessantes, desde aplicações médicas à dispositivos de detecção de mentira.

Pequenos ajustes físicos ao sensor deverão ser feitos a fim de deixá-lo preso em fitas de velcro para ser firmemente posicionado nos dedos do corpo humano. Um esquemático simples, sem ajustes, é fornecido na Fig. 6.14.

Como sugestão, indicamos o sensor XD-58C. O jumper roxo, representa a conexão com o pino S do sensor que deve ser conectado à entrada analógica do Arduino UNO.

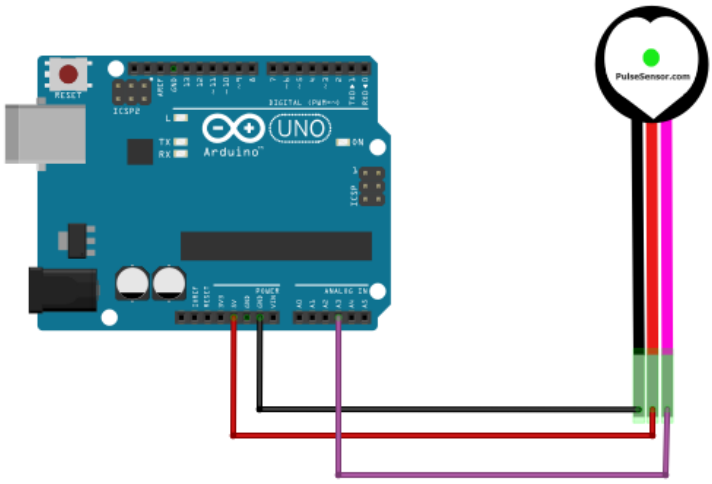


Figura 6.14: Esquemático simples para o sensor XD-58C conectado ao Arduino UNO.

Importante! É importantíssimo isolar as partes condutoras do sensor, que normalmente vêm não isolados, antes de conectar ao corpo humano. Além disso, é importante que seja também isolado eletricamente o indivíduo ao qual o sensor for conectado.

6.5.1. Aplicação simples

O código simplificado de implementação do sensor é bastante direto. Pode-se ver o gráfico formado com o tempo abrindo o serial plotter.

```
void setup() {
  Serial.begin(9600);
  pinMode(A3,INPUT);
}

void loop() {
  // Usando serial plotter
  Serial.println(analogRead(A3));
  delayMicroseconds(20);
}
```

6.5.2. Integração com DHT11 (simples detector de mentira)

Antes de tudo, é importante manter em mente que o dispositivo detector de mentira não detecta exatamente mentira, isso é determinado pela expertise de um profissional tomando como base o padrão dos batimentos cardíacos, da transpiração, temperatura do corpo em certas regiões, pressão arterial, dilatação da pupila, entre outros aspectos corpóreos e comportamentais, em situação de não estresse (e.g., quando se pergunta o nome por confirmação, se o investigado quer beber água) comparados com os mesmos parâmetros em situações de estresse (e.g., quando o investigado é exposto a perguntas relacionadas ao acontecimento em si)

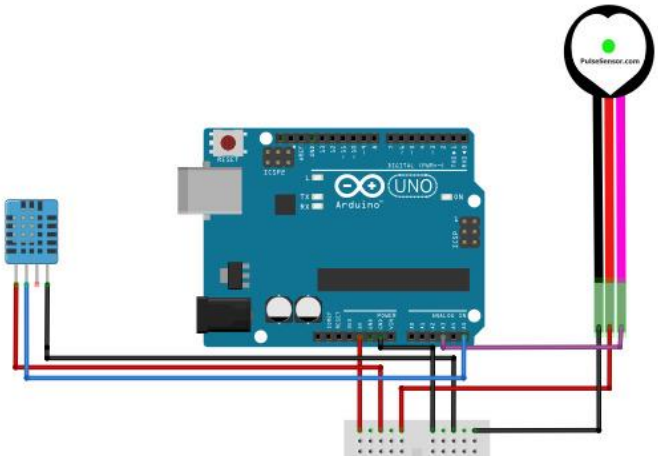


Figura 6.15: Detector de mentira (simples).

O leitor já deve estar acostumado com as configurações dos dispositivos ao Arduino. Novamente, DHT11, no pino A5, sensor de batimentos cardíacos no pino A3. Usar o serial plotter.

```
#include <DHT.h>
#define DHTPIN A5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  pinMode(A3,INPUT);
  pinMode(A5,INPUT);

  dht.begin();
}

void loop() {
  float hum = dht.readHumidity();
  Serial.print(hum);

  Serial.println(analogRead(A3));
}
```

Exercícios De Fixação

1) Simule um sistema de acionamento de alarme pelo sensor cardíaco, no qual, quando o sensor cardíaco não sentir os batimentos, deve ser soado um alarme com o buzzer.

6.6. Outros sensores

A seguir alguns outros sensores que podem vir a ser úteis para o leitor em aplicações futuras em projetos de diferentes naturezas, porém seu funcionamento fica a cargo do interesse do estudante.

6.6.1. Leitor de digitais

Às vezes, algumas situações requerem uma camada a mais de segurança. Felizmente a implementação de tecnologia de leitura de digitais hoje é bem mais acessível devido ao avanço da indústria de hardware, inclusive de código aberto, e facilidade de acesso à informação. Existem inúmeros projetos *open source* que implementam esses dispositivos pela internet.

O sensor FPM10A é um dos mais populares sensores digitais. Outro bastante consolidado é o Adafruit Optical Fingerprint Sensor.

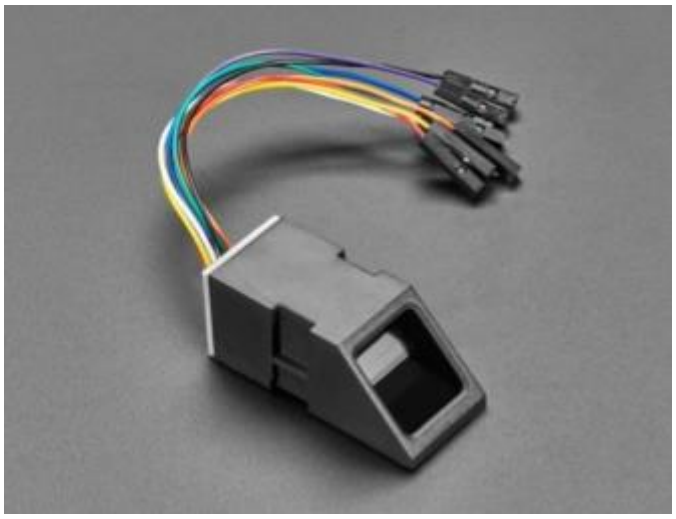


Figura 6.16: Sensor leitor de digitais da Adafruit, por ilustração.

A biblioteca da Adafruit, *fingerprint*, será muito importante no processo de desenvolvimento do projeto, por ser bastante completa e simples de usar (encontrado no Github da mesma).

6.6.2. Sensor de temperatura infravermelho

A maioria das técnicas de medição de temperatura envolvem contato físico entre o sensor e o objeto a ser medido. Porém, para algumas aplicações isso não seria adequado. Seja por higiene, pela capacidade do material de resistir à temperatura a ser medida, ou qualquer outra razão que leve a esse impedimento, o desenvolvimento da tecnologia levou ao surgimento dos sensores de temperatura infravermelho.

Recomendamos o sensor MLX90614 para checagem de temperatura sem contato físico e com alta precisão. Esse dispositivo é bastante empregado em aplicações industriais, caseiras e médicas devido a sua confiabilidade. Existe uma biblioteca que facilita a comunicação com o sensor, da Adafruit, **Adafruit_MLX90614.h.**

Figura 6.17: Sensor de temperatura infravermelho MLX90614.

Naturalmente, seu uso deve ser restrito a aplicações em que o infravermelho possa ser utilizado apropriadamente. Deve-se considerar o material dos objetos a serem medidos, a refração, absorção e reflexão de raios infravermelhos, a distância do objeto, entre outros aspectos físicos. A leitura do Datasheet revela outros detalhes importantes da aplicação do dispositivo.

Algumas ideias de projetos que podem ser implementados com tal dispositivo incluem medidor de temperatura do corpo humano com precisão de

aproximadamente 0.02°C, temperatura de panelas em aquecimento, checagem de temperatura de motores em alta rotação.

6.6.3. Sensor de luz

Um simples, barato e eficiente sensor para a maioria dos projetos cuja detecção de intensidade de luz é necessária é o LDR ("resistor dependente de luz", do inglês "Light Dependent Resistor"). Seu funcionamento consiste na diminuição da resistência do elemento quando exposto à luz, e seu subsequente aumento na ausência de luminosidade.

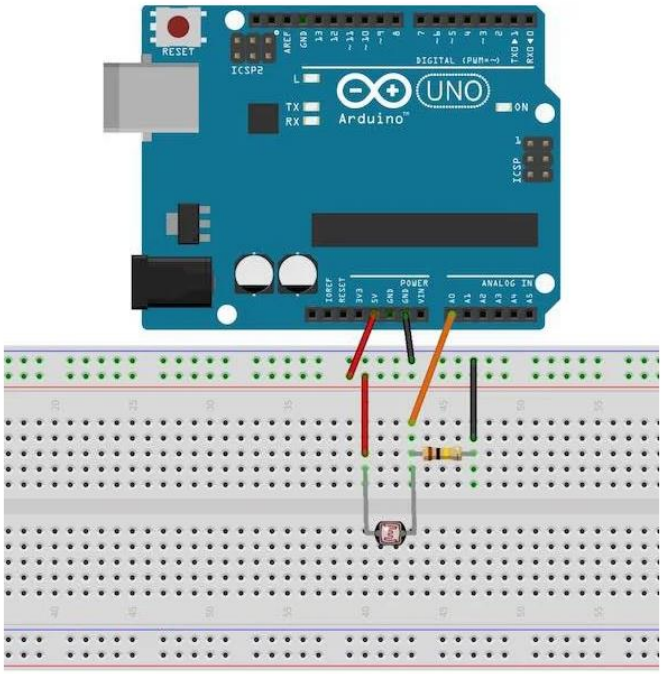


Figura 6.18: Circuito do sensor de luz.

Instalando um resistor (de 100 kΩ, por exemplo) com o LDR como divisores de tensão, fazemos a tensão dos terminais do LDR retornarem o equivalente à iluminação no ambiente.

Exercícios De Fixação

1) Desenvolva um sistema de irrigação capaz de ligar duas válvulas (pode simular as válvulas como LEDs) que liberem água para uma plantação, use um sensor de chuva fincado no solo e um sensor de temperatura e umidade para monitorar

os valores. As válvulas devem liberar água quando o sensor de chuva ler abaixo de 50% ou quando a umidade estiver abaixo de 20%.

2) Imagine um cofre totalmente lacrado que não pode ser aberto sem um código de ativação, a sala que esse cofre se encontra é totalmente iluminada e o foco das luzes são todas nesse cofre, mas como ele é lacrado, não há luminosidade dentro dele. Assim, faça um projeto com o sensor LDR que deve ficar dentro desse cofre escuro e caso ele seja aberto ative um alarme com um buzzer e um LED alertando o rompimento e possível roubo desse cofre.
3) Faça um termômetro digital com um aviso de sol ou chuva em tempo real com um sensor de temperatura e umidade, um resistor LDR e um display LCD. Pelas medições feitas, informe no LCD a temperatura do sensor de umidade e o nível de luz pelo LDR.
4) Desafio! Pela questão anterior altere os dados de temperatura para Celsius e o nível de luminosidade no LDR para identificar se o tempo está com sol, com nuvens ou completamente nublado.

6.7. Projeto sugerido

O controle de temperatura e umidade é importante para manter o ambiente confortável. A OMS determina que ambientes com umidade abaixo dos 20% podem ser danosos ao ser humano, provocando ressecamento dos olhos, das mucosas e da pele, além aumentarem os riscos de incidência de incêndio. Por outro lado, a umidade muito alta (acima de 70%) também contribui para aumento da sensação de calor no ambiente.

Construa um esquema de umidificador automatizado que constantemente informe a temperatura e umidade pelo display LCD e acenda apenas um LED verde quando o nível de água de o reservatório estiver acima de 50% da capacidade. Apenas um LED amarelo quando o nível estiver entre 10% e 50% e apenas um LED vermelho quando estiver abaixo de 10%. Um transistor deve ser usado, na saída do Arduino, para acionar ou desligar um aparelho umidificador.

Durante o dia ou quando o ambiente estiver sendo usado por alguém com luz acesa, se a umidade estiver abaixo de 25%, o sistema deve acionar o transistor. Se a temperatura estiver acima de 35°C e a umidade estiver acima de 40%, cessar o umidificador. O umidificador também deve ser desligado quando a umidade estiver acima de 50%, independente da temperatura.

Considere no projeto (na ausência de um umidificador que se possa manipular) a saída do transistor ligada a um quarto LED de cor arbitrária.